



**Michigan
Technological
University**

Michigan Technological University
Digital Commons @ Michigan Tech

Dissertations, Master's Theses and Master's Reports

2017

MODELING AND SIMULATION OF THE PERISTALTIC FLOW OF NEWTONIAN AND NON-NEWTONIAN FLUIDS WITH APPLICATION TO THE HUMAN BODY

Samer Alokaily
Michigan Technological University, saalokai@mtu.edu

Copyright 2017 Samer Alokaily

Recommended Citation

Alokaily, Samer, "MODELING AND SIMULATION OF THE PERISTALTIC FLOW OF NEWTONIAN AND NON-NEWTONIAN FLUIDS WITH APPLICATION TO THE HUMAN BODY", Open Access Dissertation, Michigan Technological University, 2017.
<https://doi.org/10.37099/mtu.dc.etdr/312>

Follow this and additional works at: <https://digitalcommons.mtu.edu/etdr>



Part of the [Numerical Analysis and Computation Commons](#)

MODELING AND SIMULATION OF THE PERISTALTIC FLOW OF
NEWTONIAN AND NON-NEWTONIAN FLUIDS WITH APPLICATION TO
THE HUMAN BODY

By

Samer A. Alokaily

A DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

In Mathematical Sciences

MICHIGAN TECHNOLOGICAL UNIVERSITY

2017

© 2017 Samer A. Alokaily

This dissertation has been approved in partial fulfillment of the requirements for the Degree of DOCTOR OF PHILOSOPHY in Mathematical Sciences.

Department of Mathematical Sciences

Dissertation Co-advisor: *Prof. Feigl, Kathleen A*

Dissertation Co-advisor: *Prof. Tanner, Franz X*

Committee Member: *Prof. Labovsky, Alexander*

Committee Member: *Prof. Yang, Song L*

Department Chair: *Prof. Gockenbach, Mark S*

Contents

List of Figures	xi
List of Tables	xxv
Acknowledgments	xxvii
List of Abbreviations	xxix
Abstract	xxxiii
1 Introduction	1
1.1 Background and Literature Review	2
1.1.1 General Peristaltic Flow	2
1.1.2 Investigations Related to the Human Stomach	8
1.2 Goals	14
1.3 Contributions of Thesis	15
2 Mathematical Models and Numerical Methods	19
2.1 An Introduction to Fluid Dynamics	19
2.2 Derivation of Governing Equations	22

2.3	Rheology	24
2.3.1	Power Law Model	32
2.3.2	Carreau-Yasuda Model	33
2.4	Numerical Methods	34
2.4.1	Finite Volume Method on Static Grids	34
2.4.1.1	Spatial Discretization of the Convection Term . . .	38
2.4.1.2	Spatial Discretization of the Diffusion Term	42
2.4.1.3	Spatial Discretization of the Source Term	44
2.4.1.4	Temporal Discretization	46
2.4.2	Finite Volume Method on Non-Static Grids	51
2.4.3	Description of TransientSimpleDyMFoam Solver	57
2.5	Moving Grid Handling	66
2.5.1	Grid Validity and Quality Metrics	67
2.5.1.1	Grid Validity Metrics	68
2.5.1.2	Grid Quality Metrics	69
2.5.2	Diffusivity Models	73
2.5.2.1	Quality-Based models	73
2.5.2.2	Distance-Based Models	74
2.5.3	Grid Motion Solvers	75
2.5.3.1	Laplace Equation	77
2.5.3.2	Solid Body Rotation Stress Equation	78

2.5.3.3	Radial Basis Function Interpolation	82
3	Fluid Transport Via Peristaltic Motion	87
3.1	Computational Models	89
3.1.1	Geometries	89
3.1.2	Governing Equations	91
3.1.3	Numerical Methods and Computational Details	96
3.1.4	Mesh Independence Study	101
3.1.5	Solver Validation	104
3.2	Results and Discussion	107
3.2.1	Variation of the Newtonian Fluid	108
3.2.1.1	Axisymmetric Tubular Simulations	109
3.2.1.2	Axisymmetric Conical Simulations	113
3.2.2	Effect of Shear-Thinning Behavior	120
3.2.2.1	Axisymmetric Tubular Simulations	121
3.2.2.2	Axisymmetric Conical Simulations	125
3.2.3	Variation of Wave Occlusion and Wave Speed	130
3.2.3.1	Axisymmetric Tubular Simulations	130
3.2.3.2	Axisymmetric Conical Simulations	132
3.3	Summary and Conclusion	139
4	Gastric Digestion and Mixing Via Peristaltic Motion	143
4.1	Computational Models	145

4.1.1	Geometry	145
4.1.2	Governing Equations	150
4.1.3	Computational Details and Convergence Considerations . . .	151
4.1.4	Mesh Independence Study	154
4.2	Results and Discussion	157
4.2.1	Variation of the Newtonian Fluid	157
4.2.2	Effect of Shear-Thinning Behavior	171
4.2.3	Variation of Wave Speed	178
4.3	Food Mixing and Particle Tracking Technique	184
4.4	Comparison with Literature	189
4.5	Summary and Conclusions	193
5	Summary and Future Work	197
	References	201
A	Open Outlet Simulations	227
A.1	2-D Planar Tubular Simulations	227
A.1.1	Case Setup	227
A.1.2	dynPerCircleApproxGradually BC.	249
A.2	2-D Axisymmetric Tubular Simulations	274
A.2.1	Case Setup	274
A.2.2	dynPerCircleAxisymm BC.	296

A.3	2-D Axisymmetric Conical Simulations	317
A.3.1	Case Setup	317
A.3.2	dynPerCircleConicalAxisymmMultiwaves BC.	343
B	Closed Outlet Simulations	377
B.1	Circular ACWs Simulations	377
B.1.1	Case Setup	377
B.1.2	dynPerCircleConicalAxisymmMultiwaves BC.	400
B.2	Parabolic ACWs Simulations	400
B.2.1	Case Setup	400
B.2.2	dynPerParabolicConicalAxisymmMultiwaves BC.	413
C	OpenFOAM Codes	447
C.1	shearRate	447
C.2	stressComponentsMag	451
C.3	movingWallNormalVel BC.	461
C.4	transientSimpleDyMFoam	476
D	Transport Efficiency via Peristaltic Motion in 2-D planer Tube .	493

List of Figures

2.1	Viscosity vs strain rate for different fluids.	28
2.2	Shear stress vs strain rate curves for different types of fluids. . . .	29
2.3	The relationship between shear viscosity and strain rate for a shear-thinning fluid.	32
2.4	Discretization of the computational domain using finite arbitrary hexahedron control volumes.	35
2.5	Schematic diagram for face interpolation and non-orthogonality treatment in the over-relaxed approach.	40
2.6	Grid quality metrics	72
2.7	A clockwise rotation of a vector through angle θ	81
3.1	Computational domain for the axisymmetric tubular model equipped by the circular deformation and relative occlusion parameters. . . .	89
3.2	(A) Schematic diagram of a human stomach. (B) Computational domain for the axisymmetric conical model equipped by the circular deformation and relative occlusion parameters.	91
3.3	Convergence study	100

3.4	Mesh dependence study for the x-component of the velocity of the Newtonian fluid N3 in the axisymmetric tubular model at $t = 16$ s. The wave speed and the relative occlusion are 10 mm/s and 80%, respectively.	103
3.5	Mesh dependence study for the x-component of the velocity of the Newtonian fluid N3 in the axisymmetric conical model at $t = 30$ s. The wave speed and the relative occlusion are 4.6 mm/s and 80%, respectively.	103
3.6	The x-component of the velocity of the non-Newtonian fluid along the centerline for the planar model. (The dots correspond to experiments [1, 2], the solid curves denote the simulations.)	106
3.7	Transport efficiency for five Newtonian fluids in the axisymmetric tubular model. The wave speed is 5 mm/s with three relative occlusions of 20%, 60% and 80% are used.	110
3.8	The velocity vectors of the Newtonian fluid N3 at $t = 32$ s in the axisymmetric tubular model. The wave speed and the relative occlusion are 5 mm/s and 60%, respectively.	110
3.9	The x-component of the velocity of five Newtonian fluids in the axisymmetric tubular model at $t = 32$ s. The wave speed and the relative occlusion are 5 mm/s and 60%, respectively.	112

3.10	The x-component of the velocity of the Newtonian fluid N3 near the outlet in the axisymmetric tubular model at two varying wave positions. The wave speed and the relative occlusion are 5 mm/s and 60%, respectively.	112
3.11	The velocity vectors (m/s) of the Newtonian fluid N3 in the axisymmetric conical model. The wave speed is 2.3 mm/s and the maximum relative occlusion is 66%.	114
3.12	The x-component of the velocity of the Newtonian fluid N3 in the axisymmetric conical model at different times and/or wave positions. The wave speed and the maximum relative occlusion are 2.3 mm/s and 66%, respectively.	115
3.13	The x-component of the velocity of five Newtonian fluids in the axisymmetric conical model along the centerline. The wave speed and the maximum relative occlusion are 2.3 mm/s and 66%, respectively.	116
3.14	The x-component of the velocity of five Newtonian fluids in the axisymmetric conical model near the outlet. The wave speed and the maximum relative occlusion are 2.3 mm/s and 66%, respectively.	116
3.15	Transport efficiency for five Newtonian fluids in the axisymmetric conical model. The wave speed is 2.3 mm/s and the maximum relative occlusion is 66%.	117

3.16	Average transport efficiency of five different Newtonian fluids in the axisymmetric conical model. The wave speed is 2.3 mm/s and three maximum relative occlusions of 21%, 66% and 80% are used. . . .	119
3.17	Shear rate dependent viscosity curves.	121
3.18	Transport efficiency for Newtonian and non-Newtonian fluids in the axisymmetric tubular model. The wave speed is 5 mm/s and three relative occlusions of 20%, 60% and 80% are used.	122
3.19	Strain rate of the Newtonian and non-Newtonian fluids in the axisymmetric tubular model along the center line and at $t = 32$ s. The wave speed and the relative occlusions are 5 mm/s and 60%, respectively.	124
3.20	The x-component of the velocity of the Newtonian and non-Newtonian fluids in the axisymmetric tubular model at $t = 32$ s. The wave speed and the relative occlusion are 5 mm/s and 60%, respectively. . . .	124
3.21	Strain rate of the Newtonian and non-Newtonian fluids in the axisymmetric conical model along the center line and at times $t = 45$ s and $t = 60$ s. The wave speed and the maximum relative occlusion are 2.3 mm/s and 66%, respectively.	126
3.22	The x-component of the velocity of the Newtonian and non-Newtonian fluids in the axisymmetric conical model near the outlet and at different times. The wave speed and the maximum relative occlusion are 2.3 mm/s and 66%, respectively.	127

3.23	Transport efficiency for the Newtonian and non-Newtonian fluids in the axisymmetric conical model. The wave speed is 2.3 mm/s and the maximum relative occlusion is 66%.	128
3.24	Average transport efficiency for the Newtonian and non-Newtonian fluids in the axisymmetric conical model. The wave speed is 2.3 mm/s and three maximum relative occlusions of 21%, 66% and 80% are applied.	129
3.25	Transport efficiency for Newtonian fluid N3 in the axisymmetric tubular model.	130
3.26	The x-component of the velocity of the Newtonian fluid N3 in the axisymmetric tubular model at $t = 32$ s for different relative occlusions. The wave speed is 5 mm/s.	131
3.27	The x-component of the velocity of the Newtonian fluid N3 in the axisymmetric tubular model at wave center of $x = 160$ mm for different wave speeds. The relative occlusion is 60%.	132
3.28	Transport efficiency for the Newtonian fluid N3 in the axisymmetric conical model. The wave speed is 2.3 mm/s and three different ranges of relative occlusions are applied.	134

3.29	The x-component of the velocity of the Newtonian fluid N3 in the axisymmetric conical model near the outlet. The wave speed is 2.3 mm/s and three different sets of relative occlusions are used at times of $t = 57$ s, $t = 58$ s and $t = 60$ s.	135
3.30	The x-component of the velocity of the Newtonian fluid N3 in the axisymmetric conical model along the center line at two different wave centers. Three different wave speeds of 1.15 mm/s, 2.3 mm/s and 4.6 mm/s are used.	137
3.31	The x-component of the velocity of the Newtonian fluid N3 in the axisymmetric conical model near the outlet at two different wave centers. Three different wave speeds of 1.15 mm/s, 2.3 mm/s and 4.6 mm/s are used.	137
3.32	Transport efficiency for the Newtonian fluid N3 in the axisymmetric conical model. The maximum relative occlusion is 66% and three different wave speeds are used.	138
3.33	Average transport efficiency for the Newtonian fluid N3 in the axisymmetric conical model with different three wave speeds and maximum relative occlusions.	138
4.1	(A) Schematic diagram of a human stomach. (B) Computational domain for the axisymmetric conical model equipped by the deformation of the ACWs and relative occlusion parameters.	147

4.2	Motility pattern of the distal-most ACW during digestion. The ACWs speed is 2.3 mm/s.	149
4.3	Convergence study	153
4.4	Mesh dependence study of the x-component of velocity for the Newtonian fluid N3 along the center line. The wave speed and relative occlusion are 4.6 mm/s and 80%, respectively.	155
4.5	Mesh dependence study of the strain rate for the Newtonian fluid N3 along the center line. The wave speed and relative occlusion are 4.6 mm/s and 80%, respectively.	156
4.6	Streamlines of the fluid flow within the lower part of stomach at $t = 60$ s, colored by velocity magnitude (m/s). Maximum relative occlusion of 52% for the distal-most circular ACW is applied.	158
4.7	The velocity vectors (m/s) of the Newtonian fluid N3 at $t = 60$ s in the most occluded section of the pylorus canal. The wave speed is 2.3 mm/s and the maximum relative occlusion is 52%.	159
4.8	The x-component of velocity for five Newtonian fluids along the center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively.	160
4.9	The x-component of velocity for five Newtonian fluids along the center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively.	160

4.10	The effect of fluid viscosity on the average value of vorticity field. The wave speed is 2.3 mm/s and two maximum relative occlusions of 52% and 80% were achieved for the distal-most ACW. The shading bars are consistent with numbers labeled on graphs 4.6 and 4.11.	162
4.11	Contour of vorticity within the lower part of stomach at $t = 63$ s, colored by vorticity magnitude (1/s). Maximum relative occlusion of 52% for the distal-most parabolic ACW is applied.	163
4.12	Kinematic pressure (m^2/s^2) of the Newtonian fluid N3. The wave speed is 2.3 mm/s and maximum relative occlusions of 80% is achieved for the distal-most ACW.	164
4.13	Kinematic pressure for five different Newtonian fluids along the center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively.	165
4.14	Kinematic pressure for five different Newtonian fluids along the center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively.	166
4.15	The x-component of kinematic pressure gradient for five different Newtonian fluids along the center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively.	167

4.16	The x-component of kinematic pressure gradient for five different Newtonian fluids along the center line . The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively.	167
4.17	Strain rate of the Newtonian fluid N3. The wave speed is 2.3 mm/s and maximum relative occlusions of 80% is achieved for the distal-most ACW.	168
4.18	Strain rate for five different Newtonian fluids along the center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively.	169
4.19	Strain rate for five different Newtonian fluids along the center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively.	169
4.20	The magnitude of kinematic viscous stress tensor for five different Newtonian fluids along the center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively.	170
4.21	The magnitude of kinematic viscous stress tensor for five different Newtonian fluids along the center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively.	171
4.22	The x-component of the velocity of the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively.	173

4.23	The x-component of the velocity of the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively.	173
4.24	Kinematic pressure of the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively.	174
4.25	Kinematic pressure of the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively.	174
4.26	The x-component of kinematic pressure gradient for the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively. . .	175
4.27	The x-component of kinematic pressure gradient for the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively. . .	175
4.28	Strain rate of the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively.	176
4.29	Strain rate of the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively.	176

4.30	The magnitude of kinematic viscous stress tensor of the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively. . .	177
4.31	The magnitude of kinematic viscous stress tensor of the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively. . .	177
4.32	The x-component of velocity of the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 52% are applied.	179
4.33	The x-component of velocity of the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 80% are applied.	179
4.34	Kinematic pressure of the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 52% are applied.	180
4.35	Kinematic pressure of the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 80% are applied.	180
4.36	The x-component of kinematic pressure gradient for the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 52% are applied.	181

4.37	The x-component of kinematic pressure gradient for the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 80% are applied.	181
4.38	Strain rate of the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 52% are applied.	182
4.39	Strain rate of the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 80% are applied.	182
4.40	The magnitude of kinematic viscous stress tensor of the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 52% are applied.	183
4.41	The magnitude of kinematic viscous stress tensor of the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 80% are applied.	183
4.42	Mixing in distal antrum illustrated by releasing two sets of four particles from initial locations (black spheres) to locations when the maximum relative occlusion is achieved (white spheres). The initial location for the first set is under the distal-most ACW, while the initial location for the second one is between the last two consecutive ACWs. . . .	184

4.43	Effect of viscosity, relative occlusion and ACW shape on an antal mixing. The wave speed is 2.3 mm/s and the comparison is performed at a time when the strongest retropulsive jet occurred, that is at $t = 60$ s for circular ACWs (left) and at $t = 63$ s for parabolic ACWs (right).	186
4.44	Effect of viscosity and ACW shape on the strain rate trajectory (black path line in Fig. 4.42) of particle number one. The ACWs speed is 2.3 mm/s and the maximum relative occlusion is 80%.	188
4.45	Effect of viscosity and ACW shape on the viscous stress trajectory (black path line in Fig. 4.42) of particle number one. The ACWs speed is 2.3 mm/s and the maximum relative occlusion is 80%. . .	188
D.1	Transport efficiency for five Newtonian fluids in the planar tubular model. The wave speed is 5 mm/s and three relative occlusions of 20%, 60% and 80% are applied.	495
D.2	The x-component of the velocity near the outlet of five Newtonian fluids in the planar tubular model at $t = 32$ s. The wave speed and the relative occlusion are 5 mm/s and 60%, respectively.	495
D.3	The x-component of the velocity near the outlet for the lowest two viscous fluids at $t = 50$ s in the planar tubular model with length of $1.5 \times L$. The wave speed and the relative occlusion are 5 mm/s and 60%, respectively.	496

D.4	Transport efficiency for the lowest two viscous fluids along the domain	
	in the planar tubular model with length of $1.5 \times L$. The wave speed	
	and the relative occlusion are 5 mm/s and 60%, respectively.	496

List of Tables

3.1	Boundary conditions in OpenFOAM for open outlet.	96
3.2	Newtonian fluids parameters.	98
3.3	Relative occlusion and wave speed values used in the axisymmetric tubular and conical models.	99
3.4	Computational mesh details for the axisymmetric tubular simulations.	101
3.5	Computational mesh details for the axisymmetric conical simulations.	102
3.6	Reynolds number at $t = 32$ s for five Newtonian fluids in the axisymmetric tubular model. The wave speed is 5 mm/s and three relative occlusions of 20%, 60% and 80% are used.	109
3.7	Reynolds number at $t = 60$ s for five Newtonian fluids in the axisymmetric conical model. The wave speed is 2.3 mm/s and three relative occlusions of 21%, 66% and 80% are used.	109
3.8	Newtonian and non-Newtonian fluid parameters.	120
4.1	Boundary conditions in OpenFOAM for closed outlet.	151

4.2	Computational mesh details for the axisymmetric conical simulations.	154
4.3	Effect of viscosity and ACW shape on the total strain computed along the (black) path line of particle number one (Fig. 4.42). The ACWs speed and maximum relative occlusion are 2.3 mm/s and 80%, respectively.	189
4.4	Solver validation confirmed by comparing the velocity and vorticity fields in our simulations with the ones reported in literature by Ferrua and Singh [3]. The ACWs speed and maximum relative occlusion are 2.3 mm/s and 80%, respectively.	192
D.1	Transport efficiency for the Newtonian and non-Newtonian fluids at time $t = 32$ s. The wave speed is 5 mm/s.	497
D.2	Transport efficiency for the Newtonian fluid N3 at $x = 160$ mm. Three different speeds of 2.5 mm/s, 5 mm/s and 10 mm/s are examined.	498

Acknowledgments

I would like to express my gratitude to all those who have made this thesis possible. My deepest gratitude goes first and foremost to my advisors, Prof. Kathleen Feigl and Prof. Franz Tanner, for their untiring encouragements and guidance. They have walked with me through all the stages of this thesis, and without their consistent and enlightening instruction, this thesis could not have reached its present form.

I am also deeply indebted to my thesis examination committee professors, Dr. Song Yang and Dr. Alexander Labovsky, for their encouragement and priceless suggestions. Their academic support and input and personal cheering are greatly appreciated. Thank you.

My gratitude also goes to my colleagues in the CFD group: Dr. Abdallah Al-Hababbeh, Dr. Ahmad Baniabedalruhman, Dr. William Case, Dr. Chao Liang, and Dr. Olabanji Shonibare. Thank you for the advice, support, and willingness that allowed me to pursue research on topics for which I am truly passionate. Also, my special thanks to the department of mathematical sciences at Michigan technological university for their financial support.

Finally, but by no means least, thanks go to my mother, father, sisters and my brother Amer for almost unbelievable support. They are the most important people in my world and I dedicate this thesis to them.

List of Abbreviations

CFD	Computational Fluid Dynamics
TE	Transport Efficiency
RO	Relative Occlusion
MRI	Magnetic Resonance Imaging
ACW	Antral Contraction Wave
UVP	Ultrasonic Doppler Velocity Profile Method
RBF	Radial Basis Function
SBR	Solid Body Rotation Stress
GCL	Geometric Conservation Law
SIMPLE	Semi-Implicit Method for Pressure-Linked Equations
PISO	Pressure-Implicit with Splitting of Operators
PIMPLE	Merged PISO–SIMPLE Algorithm
FVM	Finite Volume Method
CV	Control Volume
CD	Central Differencing
UD	Upwind Differencing
BD	Blended Differencing
CG	Conjugate Gradient Method

BiCG	Bi-conjugate Gradient Method
PCG	Preconditioned Conjugate Gradient Method
GAMG	Generalized Geometric Algebraic Multi-Grid Method
DIC	Simplified Diagonal-Based Incomplete Cholesky Method
FE	First-Order Explicit Euler (Forward Euler) Method
BE	First-Order Implicit Euler (Backward Euler) Method
CN	Second-Order Implicit Crank-Nicholson Method
C_o	Courant Number
Re	Reynolds Number
De	Deborah Number
q_f	Value on the Face
$ f $	The Magnitude of a Generic Variable f
ρ	Density
\mathbf{x}	Position Vector
P	Pressure
\mathbf{u}	Fluid Velocity
μ	Dynamic Viscosity, Molecular Viscosity
η_0	Viscosity at Zero Strain Rate
η_∞	Viscosity at Infinity Strain Rate
ν	Kinematic Viscosity
σ	Cauchy Stress Tensor

δ	Unit Tensor
τ	Viscous Tensor
\mathbf{D}	Rate-Strain Stress
$\dot{\gamma}$	Shear/Strain-Rate
$\eta(\dot{\gamma})$	Shear Viscosity
K	Consistency Index
n	Power-Law Index
Ω	Entire Flow Domain
\mathbf{g}	Body Force Per Unit Mass
t	Time

Abstract

Computational models are developed to investigate peristaltic motion in the human gastro-intestinal tract. The peristaltic motion is simulated by means of traveling waves which deform the boundary of the tubes. An axisymmetric tube of uniform diameter is used to model the small intestines, and an axisymmetric conical geometry is developed to model the lower part of the human stomach. The conical geometry represents a simplification of the more complicated three-dimensional models of the human stomach that have been used in other studies. Also, they seek to reduce computational costs and circumvent difficulties of mesh generation. The computations are performed within the open source CFD environment OpenFOAM. Whenever possible, comparisons are made to the predictions of other geometrical models from the literature to validate our results.

First, the transport of fluids via peristaltic motion in a cylindrical or a conical tube is investigated. The effect of flow, fluid, and geometrical parameters on the flow behavior is determined. Of particular interest is the transport efficiency, flow patterns, and strain rates.

Second, the mixing characteristics of peristalsis is investigated for the human stomach when the pylorus is closed. Using the axisymmetric conical geometry, the effect of parameters such as wave speed, wave shape, relative occlusion, and fluid viscosity of Newtonian and non-Newtonian fluids on the flow behavior are determined.

The focus of these investigations is on the quantification of the retropulsive jet induced at the pylorus, as well as on the induced vorticities between peristaltic waves, both of which contribute to the mixing efficiency. Moreover, particle tracking techniques are used to determine strain rates along particle paths which allows the investigation of stresses experienced by food particles.

Chapter 1

Introduction

Peristalsis is the muscular contraction and relaxation of vessel walls which induces a flow of the material inside through wave-like motion. Peristaltic motion appears in many biological systems, including the human body. On one hand, it is an essential mechanism (1) by which food is transported through the digestive tracts including the small intestine, the esophagus and the stomach; (2) in the flow of blood through the veins, the capillaries and the arteries; (3) in the transport of lymph in lymphatic vessels; (4) and in urine transport from kidney to bladder through the ureter. On the second hand, peristalsis play a significant rule in gastric digestion and mixing within a human body, by reproducing the mechanical forces and fluid motions that promote not only the breakdown and mixing of gastric content, but also its chemical digestion and absorption.

In addition, this phenomenon has been exploited in many industrial applications involving biomechanical and biomedical systems in the so-called roller pump. It is used to move sanitary fluid without contamination, transport noxious fluid in the nuclear industry and pump the blood in the heart-lung machine.

The research in this thesis is motivated by the aforementioned transport of material in the stomach and intestines of humans, and the mixing of material in the human stomach. Specifically, computational models are developed and employed to study the transport and mixing characteristics of peristaltic motion for different Newtonian and non-Newtonian fluids under different conditions. The peristaltic motion is simulated by means of traveling waves which deform the walls of tubes of uniform or varying diameter.

1.1 Background and Literature Review

1.1.1 General Peristaltic Flow

Several theoretical and experimental attempts have been carried out by researchers to study the mechanism of peristalsis. Latham [4] was probably the first to study the mechanism of peristaltic pumping in 1966. Specifically, he investigated analytically and experimentally the behavior of a 2-D channel peristaltic pump. Early theoretical

work on ureteral function involving peristalsis includes that of Shapiro [5] who considered retrograde diffusion in a 2-D peristaltic pump; Fung and Yih [6], who studied inertia-free, Newtonian flow driven by sinusoidal transverse waves of small amplitude, and Shapiro et al. [7] who studied Newtonian flow with a periodic train of sinusoidal peristaltic waves of long wavelength and arbitrary amplitude in a 2-D channel. Shapiro et al. [7] also derived conditions for the presence of closed streamlines and offered an explanation of the reflux phenomena. In addition, Burns and Parkes [8] contributed to the theory of peristaltic pumping without reference to physiological applications, while Barton and Raynor [9] investigated analytically the Newtonian fluid flow driven by the peristaltic motion in a flexible tube. Later, Lykoudis [10] and Weinberg et al. [11] proposed models that represent ureteral waves more realistically. The theoretical study of the characteristics of peristalsis in terms of the fluid dynamics encountered in mixing and propulsion of food in small intestine have been studied extensively for Newtonian fluids by Lew et al. [12]. A review of much of the early literature is presented and summarized by Jaffrin and Shapiro [13] in 1971. Some later examples of peristalsis were given by Liron [14] in 1978 and considerable experimental investigations of peristaltic pumping have also been undertaken between 1966 and 1976 for example, in [4, 11, 15, 16, 17, 18, 19, 20, 21].

A review of much of the theoretical investigations up to year 1983, arranged according to geometry, fluid, Reynolds number, wavelength parameter, wave amplitude parameter and wave shape, as well as an account of the experimental attempts on the subject was presented in an excellent article by Srivastava and Srivastava [22]. The significant contributions to the subject after the year 1984 were well referenced by Medhavi [23] in 2008, who investigated the flow induced by sinusoidal peristaltic motion of the tube wall of a non-Newtonian fluid obeying the Herschel-Bulkley equation under long wavelength and low Reynolds number approximation.

The complex rheology of biological and physiological flows has also motivated a number of studies involving non-Newtonian and viscoelastic fluids. The initial investigation to understand the peristaltic motion of non-Newtonian fluids has been made numerically by Raju and Devanathan [24] in 1972. They used the power-law model to investigate shear-thinning and shear-thickening effects in a rigid tube with a sinusoidal deformation at the boundary in the case of small wave amplitude, and they discussed the influence of the applied pressure gradient along with non-Newtonian parameters on the streamlines and velocity profiles. Later, theoretical studies on the same model has been carried out by Picologlou et al. [25] and Shukla and S.P. Gupta [26] to investigate similar effects. Specifically, Shukla and S.P. Gupta [26] investigated the effects of the consistency variation on the peristaltic transport of a non-Newtonian power-law fluid through a tube by taking into account the existence of a peripheral layer. They found that the flow rate flux, for zero pressure drop, increases as the amplitude

of the peristaltic wave increases but it decreases due to the pseudoplastic nature of the fluid. Also they observed that, for zero pressure drop, the flux does not depend on the consistency of peripheral layer while the friction decreases as this consistency decreases. On the other hand, for nonzero pressure drop, the flux increases and the friction force decreases as the consistency of peripheral layer fluid decreases.

Becker [27] presented a theoretical analysis for fluids with shear-dependent viscosity and computed pumping characteristics for a Prandtl–Eyring fluid. Raju and Devanathan [28] and Böhme and Friedrich [29] probed analytically the effects of viscoelasticity. In particular, Böhme and Friedrich [29] studied the mechanism of peristaltic transport of an incompressible viscoelastic fluid by means of an infinite train of sinusoidal waves traveling along the wall of the duct in the case of a plane flow. They studied fluid motion analytically with a second-order approximation with respect to the wave amplitude ratio for sufficiently small values of the ratio of the wave amplitude and the mean height of the channel. They found that the results are influenced by specific values of the complex viscosity of the fluid and that relatively small wave speeds are the best. Since the fluid changes its state slowly so that the memory, and with it the elasticity, of the fluid do not influence the flow field at all. Also, as the dimensionless memory parameter tends to zero, the analytical results reduce to the well-known case of a Newtonian fluid. Siddiqui et al. [30] used the second order fluid model to study the effects of normal stresses in slow non-Newtonian flows.

Peristaltic pumping of blood in small vessels has been studied by Srivastava and coworkers [22, 31, 32]. Specifically, Srivastava and Srivastava [31] studied theoretically the problem of peristaltic transport of a non-Newtonian (power-law) fluid in a uniform and non-uniform tube under zero Reynolds number and long wavelength approximation. They found that the magnitude of pressure rise in the case of a non-Newtonian fluid, when the power-law index is less than 1, at zero (volumetric) flow rate, is larger than the one of a Newtonian fluid model. Further, the pressure rise decreases as the index decreases from 1, at zero flow rate, is independent of the index at a certain value of flow rate, and increases if flow rate exceeds further. Also, at a given flow rate, an increase in the wavelength leads to a decrease in pressure rise and increase in the influence of non-Newtonian behavior. Pressure rise, in the case of a non-uniform geometry, is found to be much smaller than the corresponding value in the case of uniform geometry. Srivastava and Saxena [32] investigated numerically and analytically the problem of blood flow induced by peristaltic waves in a uniform small diameter tube. Blood was represented by a two-fluid model consisting of a core region of suspension of all the erythrocytes, assumed to be a Casson fluid, and a peripheral layer of plasma modeled as a Newtonian fluid. Alden et al [33] presented a theoretical study of viscous effects in peristaltic pumping. They used a lubrication-type flow through an infinitely long axisymmetric tube subjected to a periodic train of transverse waves. Elsehawe et al. [34] considered the problem of peristaltic transport of a non-Newtonian Carreau fluid in a nonuniform 2-D channel under zero Reynolds

number with long wavelength approximation. The problem was formulated using a perturbation expansion in terms of a variant of the Weissenberg number. They obtained analytic forms for the axial velocity component and the pressure gradient, and they computed numerically the pressure rise and friction force.

Other theoretical studies of non-Newtonian peristaltic flow include those of Mernone et al. [35] who studied peristaltic transport of a Casson fluid in a planar channel; Hayat and Ali [36], who analyzed axisymmetric peristaltic motion of Johnson–Segalman fluid through a circular deformable tube; Nadeem and Akbar [37], who simulated the peristaltic flow of a Herschel–Bulkley yield–value fluid in a nonuniform inclined tube; and Nadeem et al. [38], who simulated the peristaltic flow of a Jeffrey–six constant fluid in a uniform inclined tube. The peristaltic flow behavior of non-Newtonian fluids in elastic tubes had been investigated experimentally by Nahar et al. [1, 39]. In particular, Nahar et al. [1] found that increasing the wave speed of peristalsis resulted in higher magnitude of back flow both in the wave crest and trough regions, the approximated wall shear rates at the wave trough were found to be higher than those in the wave crest. In addition, the pressure difference between crest and trough of a peristaltic wave increased as the wave speed increased, and the crest region showed a higher pressure compared to the trough region.

In coordination with the experiments of Nahar et al. [1, 39], a numerical study was performed by Al-Hababbeh [40] to determine the effect of the shear-thinning behavior, the wave speed and the gap width on the transport efficiency of peristaltic motion.

He used the Bird-Carreau model to simulate the non-Newtonian fluid in a deforming 2-D channel. The present work extends that of Al-Hababbeh in two ways. First, we develop a 2-D axisymmetric numerical model to get a realistic tubular peristaltic flow as encountered in the small intestine, and second, we examine the influence of the fluid viscosity variation on the transport efficiency.

1.1.2 Investigations Related to the Human Stomach

The human stomach is a J-shaped, muscular, hollow and dilated part of the gastrointestinal tract that functions as an important organ in the digestive system. It is located between the esophagus and the first part of small intestine (duodenum) in the region of the left side of the upper abdominal cavity. Anatomically, the stomach is subdivided into the fundus, the corpus, and the antrum [41, 42].

Cannon [43], Kelly [44], Urbain et al. [45], Pal et al. [46] and Kong and Singh [47] described the principle functions of human stomach as follows: The upper part of the stomach (the fundus and the proximal corpus) acts a reservoir of chewed up food (bolus) that enters the stomach through the esophagus via the lower esophageal sphincter, while the lower part of the stomach (the antrum and the distal corpus) is responsible for mechanical forces and fluid motions that promote not only the breakdown and mixing of gastric content, but also its chemical digestion, absorption and transport. The mechanical forces and fluid motion are caused by peristalsis induced

by antral waves, or wave-like muscular contractions of the stomach walls. After that, the pyloric sphincter controls the passage of partially digested food (chyme) from the stomach into the duodenum where peristalsis transport the material through the rest of the small intestines. Imai et al. [48] reported that the curved, twisted shape of the stomach not only supports gastric mixing, but also separates the stomach into reservoir and mixing regions.

A number of experimental studies have been carried out to study the emptying of gastric contents in the duodenum via the pylorus and to clarify the functions of the stomach. Specifically, Kelly [44] proved the hypothesis that the proximal stomach has a major role in gastric emptying of liquids and the distal stomach a major role in gastric emptying of solids. Keinke et al. [49] investigated mechanisms controlling gastric emptying of viscous meals in four conscious dogs. They concluded that gastric emptying is controlled by the depth of the antral waves, the pyloric opening, the receptive relaxation of the duodenum and the type of the duodenal contractions. By contrast the sequence of the terminal antral contraction, the pyloric closure and the coordination between pyloric and duodenal contractions played no important role in regulating gastric emptying. King et al. [50] examined the relationships between peristaltic contractions and the movement of gastric contents through the pylorus, by giving ten healthy volunteers a test meal of dilute orange juice and bran, and events at the gastric outlet monitored by real-time ultrasound. Hausken et al. [51] used duplex sonography method to visualize antroduodenal motility and movements

of luminal contents after ingestion of 500 mL of meat soup. They found that the peristaltic closure of the pylorus is normally preceded by a short gush of duodenogastric reflux. Pallotta et al. [52] evaluated, by means of real-time ultrasonographic (US), the relation between the antral-wall contractions and the pylorus opening and closure in relation to transpyloric flow and the mixing of contents during the postprandial phase of gastric digestion and emptying of a nutrient meal.

The effect of body position and stomach volume of ingested contents on gastric emptying have been studied experimentally in an extensive way by many researchers. In particular, Boulby et al. [53] assessed intragastric flow in the gastric antrum of eight healthy volunteers by using a velocity-sensitive version of the high speed magnetic resonance imaging technique, echo planar imaging (EPI). They found that fat delays gastric emptying but increases forward and backward antral flow. The rate of gastric emptying of saline solution has been studied by Burn et al. [54], under the effects of changes in posture of five stomachs who were either sitting, lying on the left side, or lying on the right side. They found that saline solutions emptied from the stomach more rapidly when the stomachs lay on their right sides than when they lay on their left sides. The effects of volume and posture on gastric emptying and intragastric distribution of a solid meal and appetite were evaluated by Doran et al. [55]. They concluded that meal volume has a major effect on gastric emptying; in contrast posture has only a minor impact on intragastric meal distribution, which is observed only after a large meal, and no effect on gastric emptying. Faas et al. [56] investigated the

effects of test meals of different food consistencies and the amount of liquid ingested with the meal on the intragastric distribution of a contrast marker. Also, they used MRI to clarify the distribution processes in the stomach. They found that the intragastric distribution of a marker will be related to the amount of accessible liquid contained in the meal, and that the consistency of the meal will affect the spatial distribution of the contrast marker in the stomach, resulting in large differences in the timing of its delivery to the small intestine. Steingoetter et al. [57] used also MRI analyses to study the effects of body position on gastric emptying and motor function. Twelve volunteers were investigated in seated position (SP) and upside-down position (UDP) after ingestion of 300 mL water. They concluded that the stomach maintains the rate of gastric emptying despite radical changes in body position and intragastric distribution of gastric contents. In SP, hydrostatic pressure (modulated by gastric tone) dictated the gastric emptying. In UDP, gastric emptying also appeared to be mediated by continuous adaptation of gastric tone.

Experimental studies carried out by Doran et al. [55], Edelbroek et al. [58], Goetze et al. [59], and Schwizer et al. [60] concluded that the mechanism of gastric emptying is incompletely understood, because gastric emptying proceeds over hours. Pullan et al. [61] developed an anatomical model of the stomach based on the data of the visible human project that was done by Spitzer et al. [62]. More recently, Imai et al. [48] numerically investigated flow in the stomach during gastric mixing at a time when the pylorus is closed. They used the above anatomical model for the stomach

geometry and free surface flow modeling to analyze the effects of stomach posture on gastric mixing. They found that antral recirculation transports gastric content from distal stomach to the antrum near the pylorus, and it is then mixed by retropulsive flow. They concluded that gastric content inside the antral recirculation is well mixed independently of the initial location, whereas the content outside recirculation is poorly mixed.

During gastric digestion within the stomach, food structure is broken down and mixed by a complex interaction of chemical and mechanical effects. However, despite the fact that the chemical process is usually done by using an in vitro analysis, the possibility of developing an in vitro system capable of reproducing the fluid mechanical forces that promote digestion is still extremely difficult to achieve, if not impossible. Since the beginning of the 1990's, a series of in vitro systems has been developed to analyze human digestion including those of Aoki et al. [63, 64], Molly et al. [65], Kong and Singh [47] and Wickham et al. [66]. Singh [67] offered a promising technique to characterize the mechanisms promoting digestion. Some initial attempts were taken by Pal et al. [46, 68] to simulate the gastric flow and mixing during digestion and emptying using a 2-D numerical model. Ferrua and Singh [3] developed a 3-D numerical model of the geometry and motility of the human stomach during digestion, and used it to characterize and compare the fluid dynamics of gastric contents of different viscosities. Hausken et al. [69] and Schulze [42] showed that the gastric contents

empty into the duodenum whenever a positive gastroduodenal pressure gradient is established in the presence of an open pylorus. Although different mathematical models have been proposed, for example [70, 71, 72], to partially understand and characterize the kinetics of gastric emptying, none of them actually considered the effect of the physiochemical properties of the meal and the pattern of gastric emptying.

The research in this thesis extends the work of Pal et al. [68] and Ferrua and Singh [3] in three ways. First, we develop a simple 2-D axisymmetric numerical model, that reduces the high level of complexity in the full 3-D model used by Ferrua and Singh [3]. With this model, we illustrate the principles of mechanical digestion and mixing within the lower part of human stomach. Second, a parameter study is performed to investigate the effect of various geometrical and rheological parameters on the gastric digestion and mixing, to get a better understanding of the flow field that develops within the lower part of human stomach. Third, antral contractions have been allowed to live in the vicinity of the pylorus, which is where the largest gradients for velocity and pressure occur.

1.2 Goals

The goals of this thesis are three-fold.

1. The development and validation of simple geometrical models which capture the essential flow dynamics of peristaltic motion in tubes of uniform or linearly varying diameter. The peristaltic motion is simulated by means of traveling waves which deform the boundary of the tubes. An axisymmetric tube of uniform diameter is used to model the small intestines, and an axisymmetric conical geometry is developed to model the lower part of the human stomach. The latter geometry represents a simplification of more complicated three-dimensional models of the human stomach that have been used, and seeks to reduce computational costs and difficulty of mesh generation. Whenever possible, comparisons are made to the predictions of other geometrical models to validate our results.
2. The investigation of the transport of fluids via peristaltic motion in tubes of uniform or linearly varying diameter. The effect of flow, fluid, and geometrical parameters on the flow behavior is determined. Of particular interest is the transport efficiency, flow patterns, and strain rates.

3. The investigation of the mixing characteristics of peristalsis in the human stomach when the pylorus is closed. Using our axisymmetric conical geometry, we determine the effect of parameters such as wave speed, wave shape, relative occlusion, and fluid viscosity on the flow behavior of Newtonian and non-Newtonian fluids. We focus on the predicted retropulsive jet and vorticity which contribute to the mixing efficiency. Moreover, particle tracking techniques are used to determine strain rates along particle paths which allows investigation of stresses experienced by food particles.

1.3 Contributions of Thesis

OpenFOAM is used to simulate peristaltic motion in this study. OpenFOAM is an open source library of C++ programs which serves as a modeling and computational fluid dynamics (CFD) platform. It contains standard solvers for many CFD problems, and allows new solvers or modifications to existing solvers or libraries to be constructed.

To the best of our knowledge, no attempt has been made in OpenFOAM to develop realistic numerical models of the small intestine and the lower part of human stomach (antrum) during the emptying, digestion, and mixing processes. Therefore, this dissertation contributes to CFD field in complementary and several ways:

1. Two-dimensional axisymmetric numerical models have been developed to:
 - (a) reduce the high level of complexity in the full 3-D models and hence decrease the computational time and cost.
 - (b) get a plenty of insight into the flow field that develop within the system, because the possibility of developing an in vitro system capable of reproducing the mechanical forces that promote fluid transport, digestion, absorption, and mixing is still extremely difficult to achieve till now.

The first model corresponds to a realistic tubular peristaltic flows as encountered in the small intestine, while the second one simulates the peristaltic flow in the lower part of an idealized human stomach.

2. Two dynamic mesh motion solvers have been written to deform the boundary and hence the mesh by a periodic sequence of circular and parabolic contraction waves. Within the bounds of these solvers, the user can input geometrical and rheological parameters to simulate gastric motility and physiology of the small intestines and the lower part of a human stomach.
3. The characteristic data of our peristaltic simulations have been chosen to match the experimental and numerical ones that were previously reported in the literatures. In general, there is very good agreement between the flow fields reported in the literature and the ones captured in our simulations, and hence confirming that the numerical models and methods are valid for the computation of single

phase peristaltic flow.

4. A parameter study has been performed to examine the influence of several rheological and geometrical parameters on the emptying process in terms of the (average) transport efficiency, on the digestion phase in terms of the forward (eddies) and backward (retropulsive jet) antral flows, and on the antral mixing level in terms of the root mean square radius of the relative spread of particles. In particular, we examine the effect of the Newtonian fluid viscosity, the shear-thinning non-Newtonian fluid behavior, the traveling wave speed, and the amount of deformation in terms of relative occlusion and wave shape.
5. A particle tracking technique has been used to trace the strain rates and viscous stress forces that the fluid elements experienced along their particle paths due to the contraction waves in antrum.
6. Finally, antral contraction waves have been allowed to live in the vicinity of the pylorus which is where the largest gradients of velocity and pressure exist in our conical geometry, at a time when the pylorus is open or closed.

Chapter 2

Mathematical Models and Numerical Methods

2.1 An Introduction to Fluid Dynamics

Fluid dynamics is a science discipline that deals with the flow behavior of liquids and gases and their interactions with solid bodies. It is fundamental in the study of an extreme range of problems, such as determining the mass flow rate in blood flow in the capillaries whose diameter is around 10^{-6} meters to flow in petroleum pipelines, which take into account kilometers in length and meters in diameter, as well as calculation of forces and moments acting on bodies (aircrafts, ships, cars,

etc.) inside a fluid domain. Fluid dynamics also provides methods for studying the evolution of stars, ocean currents and even weather patterns [73].

It is common in most of interested cases to consider a fluid as a substance that flows continuously under applied stress such that its molecular structure offers no resistance to external forces. These driving forces can be classified as surface forces (e.g. pressure, viscous forces in a moving fluid, etc.) which act on the boundary surface and body forces (e.g. gravity, electromagnetic forces, etc.) which act on the bulk of material.

The continuum approach can be used to derive the set of governing equations, which are invoked by the physical principles and laws of conservation of mass, momentum and energy. In this approach, a fluid is regarded as continuum or a continuous substance, hence its individual molecular effects are ignored and it is required that the mean free path to be very small compared to the smallest geometric length scale. Every point in space has finite values for physical properties such as velocity, pressure, stress, temperature, etc. From a point to the next point, the properties may change value, and there may even be surfaces where some properties jump discontinuously (e.g. the interface between a solid and a fluid is a surface where the density jumps from one value to another) [74]. The continuum assumption does not allow properties to become infinite or to jump discontinuously at a single isolated point. The statistical approach is also used to derive the governing equations in a molecular point of view. In this approach, the fluid is treated as a set of molecules whose motion is governed

by the laws of dynamics. It is common and familiar for microfluidics and light gases, but not for liquids and polyatomic gas molecules [73].

Flows where changes in fluid density are important are called compressible flows and its commonly used in gas dynamic area, while flows of either gas or liquid where changes in the fluid density are not important part of physics are call incompressible flows.

A Stokes (or creeping) flow occurs when the inertia of the fluid can be ignored because of a low speed. This kind of flow has a very small Reynolds number Re . The flow is called Laminar if the inertia becomes important but the trajectories that fluid particles follow are smooth. This kind of flow is stable, contains no velocity fluctuations, and its Reynolds number falls below some critical value that depends on the considered geometry. Turbulent flows are flows which contain self-sustaining velocity fluctuations in addition to the main flow. This kind of flow has a very high Reynolds number.

Of particular interest is the study of fluid motions and patterns induced by peristalsis inside a tube of a fixed diameter, the flow is Stokes if $Re \ll 1$, the flow is laminar if $Re < 2100$, and turbulent if $Re > 4000$ and the range $2100 < Re < 4000$ represents the transition range. In this thesis only incompressible, creeping and Laminar fluids are considered.

2.2 Derivation of Governing Equations

The fluid motion can be described by means of the governing equations that are derived from the conservation for mass, momentum, and energy principles and from equations of state ([74, 75]). The law of conservation of mass states that mass can neither be created nor destroyed.

The continuity equation describes the time rate change of the fluid density at a fixed point in space which yields the convective mass transport (net rate of mass addition per unit volume by convection). The differential form of the continuity equation (or mass conservation equation) written by using the tensor notation yields

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0, \quad (2.1)$$

where ρ is the density, \mathbf{u} is the velocity, and t is time.

The equation of motion (the momentum equation) is derived from the momentum balance over a control volume fixed in space. The equation of motion is the application of Newton's second law to an element of the fluid. It can be stated that the net momentum change of fluid mass is equal to the net external force applied on the fluid mass. The differential form of general momentum equation in tensor notation can be written as

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u} \mathbf{u}) = \nabla \cdot \sigma + \rho \mathbf{g}, \quad (2.2)$$

where σ is the Cauchy stress tensor and \mathbf{g} is the acceleration due to the gravity. The Cauchy stress tensor is defined in Eq. (2.3), in which the first term on the right hand side is defined as the (thermodynamics) pressure term P , and the second term on the right hand side is defined as the deviatoric stress (or viscous stress, or extra-stress) tensor term τ .

$$\sigma = -P\delta + \tau, \quad (2.3)$$

where δ is the unit tensor (or Kronecker delta), which is equal to one if its i^{th} and j^{th} components are the same in σ_{ij} otherwise considered as zero.

Substituting Eq. (2.3) into the momentum conservation Eq. (2.2) yields

$$\frac{\partial(\rho\mathbf{u})}{\partial t} + \nabla \cdot (\rho\mathbf{u}\mathbf{u}) = -\nabla P + \nabla \cdot \tau + \rho\mathbf{g}. \quad (2.4)$$

The first term on the left hand side describes the rate of increase of momentum per unit volume (accumulation term). The second term on the left hand side defines the rate of momentum added by convection (convection term). The first term on the right hand side is the pressure term and the second term describes the viscous term. Together they define the momentum added to the system by molecular transport per unit volume. The last term on the right hand side describes the external forces acting on the fluid (gravitational term). Assuming that the fluid is incompressible leads to the time rate of change of density to equal zero, resulting in a simplification of the

conservation equations into the form of

$$\nabla \cdot \mathbf{u} = 0. \quad (2.5)$$

$$\rho \left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right) = -\nabla P + \nabla \cdot \boldsymbol{\tau} + \rho \mathbf{g}. \quad (2.6)$$

Equation 2.5 is called continuity constraint.

2.3 Rheology

Rheology is the science that describes the relationship between stress and deformation (strains) of materials. The mathematical form of this relationship is called the constitutive equation. Constitutive equations are used in order to solve the conservation equations since there are more unknowns than actual equations. In order to solve the viscous term a rheological equation of state which describes the stress in the fluid as a function of the rate of deformation can be used as a constitutive equation. A Newtonian fluid is a fluid in which the stress is linear in the rate-of-strain tensor, and has the following aspects:

1. The only stress generated in the simple shear flow is the shear stress and a fluid exhibits a zero normal stress differences.

2. The viscosity (which is the quantity that describes a fluid's resistance to flow) does not vary with strain rate.
3. The viscosity is also constant in respect to the time of shearing and after the shearing ceases the stress stops immediately.

A fluid showing a deviation from these characteristics can be classified as a non-Newtonian fluid. For general isotropic (no directional preference) Newtonian fluids, the viscous term is expressed by the viscous stress tensor τ , which is defined as

$$\tau = \mu \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T \right) - \frac{2}{3} \mu (\nabla \cdot \mathbf{u}) \delta, \quad (2.7)$$

where μ is the constant dynamic viscosity. When the fluid is assumed as incompressible then the divergence of the velocity vector is equal to zero, therefore the viscous stress tensor for incompressible Newtonian fluids becomes

$$\tau = \mu \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T \right). \quad (2.8)$$

The rate-of-strain (also called rate of deformation) tensor \mathbf{D} is a symmetric tensor that describes the rate of change of the deformation of a material at a certain time, and it can be expressed as

$$\mathbf{D} = \nabla \mathbf{u} + (\nabla \mathbf{u})^T. \quad (2.9)$$

The strain rate scalar $\dot{\gamma}$ which is the magnitude of the strain-of-rate tensor, is defined in Eq. (2.10), and its usually used to describe the flow behavior in liquids rather than strain.

$$\dot{\gamma} = \left\| \frac{1}{2} \mathbf{D} \right\| = \sqrt{\frac{1}{2} (\mathbf{D} : \mathbf{D})} = \sqrt{\frac{1}{2} \sum_{i,j} D_{ij} D_{ji}} \quad (2.10)$$

Substituting the consecutive Eq. (2.8) into the momentum Eq. (2.6) for incompressible flow yields Navier Stokes Equation (NSE)

$$\rho \left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right) = -\nabla P + \mu \nabla^2 \mathbf{u} + \rho \mathbf{g}. \quad (2.11)$$

Non-Newtonian fluids can be classified in three general categories;

1. Time-independent (or inelastic, or purely-viscous) non-Newtonian fluids; when the viscosity of the fluid depends on the strain rate and does not depend on time. The fluids with a strain-rate dependent viscosity (or shear viscosity, or apparent viscosity) $\eta(\dot{\gamma})$ can be sub-divided into three classes:
 - (a) Shear thinning (or pseudoplastic) fluids, when shear viscosity decreases as the strain rate increases. It is often seen in polymer solutions and melts, and complex fluids and suspensions like ketchup, whipped cream, syrups, and nail polish.
 - (b) Shear thickening (or dilatant) fluids, when shear viscosity increases with increasing strain rate. A simple example is cornstarch and water mixtures.

- (c) Yield stress (or yield value, or viscoplastic) fluids, when a specific yield stress has to be exceeded for the fluid to flow. Common yield stress fluids include toothpaste, chocolate, mayonnaise, mustard, blood, slurries, clays, margarine, and paint.

The difference between the various time-independent fluids where the shear viscosity is a function of strain rate is shown in Fig. 2.1. As stated previously, for Newtonian fluids the viscosity does not change with strain rate, thus it has a constant value even with increasing strain rate.

In this thesis only shear-thinning non-Newtonian fluid is considered. The relationship between the stress and the strain rate for different types of fluid is shown in Fig. 2.2.

2. Time-dependent non-Newtonian fluids; when the viscosity of the fluid as well as the shear stress can either increase or decrease with the time of applied shearing. These fluids can be sub-divided into two classes:

- (a) Thixotropic fluids, when the shear viscosity gradually decreases with time while under constant shearing and afterwards recovers gradually when the stress is removed. Some of these fluids return to a gel (more viscous) state almost instantly such as ketchup, others such as yogurt and gum take much longer and can become nearly solid.

- (b) Anti-thixotropic fluids (or Rheopecty), when the shear viscosity gradually increases with time while under constant shearing and afterwards recovers gradually when the stress is removed. Familiar Rheopecty examples include printer ink and gypsum paste.

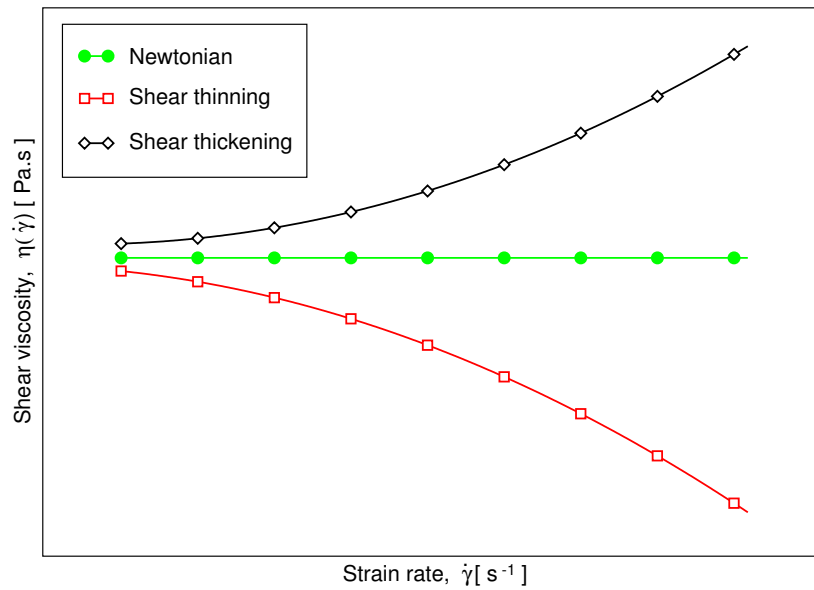


Figure 2.1: Viscosity vs strain rate for different fluids.

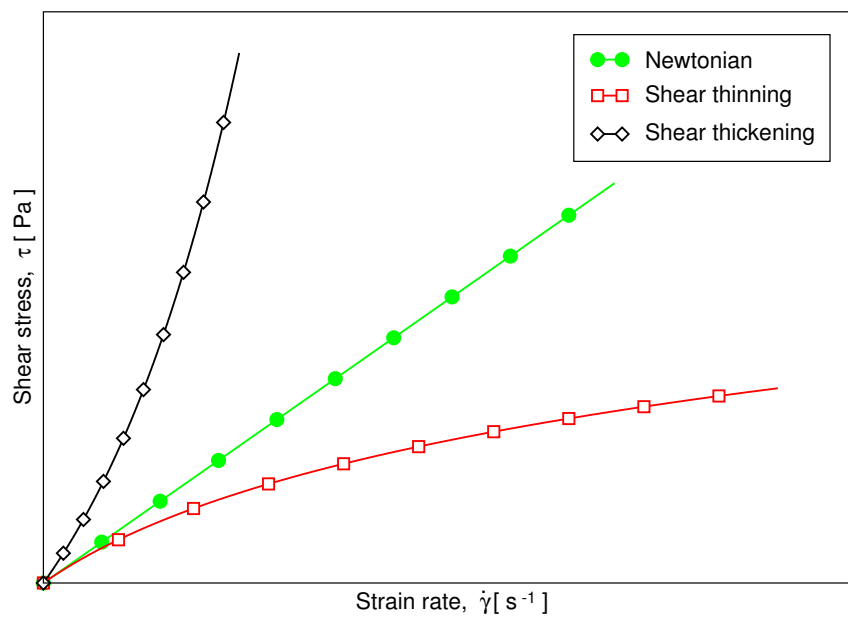


Figure 2.2: Shear stress vs strain rate curves for different types of fluids.

3. Fluids exhibiting viscoelastic behavior; have a simultaneous existence of viscous and elastic properties and, as such, exhibit time-dependent strain. Viscoelastic materials can be used with great success for shock absorbing, protection from vibration issues, relieving stress and pain on the human body as well as for reducing noise transmission.

The dimensionless Deborah number De is used to characterize the fluidity of material under specific flow conditions. In particular, its used to understand the behavior of viscoelastic materials and to distinguish solids from liquids [76]. Formally, it is defined as the ratio of the relaxation time characterizing the time it takes for a material to adjust to applied stresses or deformations, and the characteristic time scale of an experiment (or a computer simulation) probing the response of the material:

$$De = \frac{\text{stress relaxation time}}{\text{time scale of observation}} \quad (2.12)$$

At higher Deborah numbers, the effects of elasticity is increasingly dominated, therefore the material behavior enters the non-Newtonian regime and demonstrating solid-like behavior, also a consecutive equation has to be accounted to describe material elasticity effects. At lower Deborah number, i.e. $De \ll De_{critical}$, the effects of elasticity can be ignored, therefore the material behavior enters the viscous Newtonian regime and behaves in a more fluid like manner. Typically $De_{critical} \approx 1$, however it depends on the flow and the geometry of the problem.

In this thesis only inelastic fluids are studied. Generalized Newtonian (inelastic) fluid assumes a simple constitutive equation like the one for the Newtonian fluid but here the viscosity is a function of the strain rate. The general form of the constitutive equation for the generalized Newtonian fluid models is

$$\boldsymbol{\tau} = \eta(\dot{\gamma}) \mathbf{D}. \quad (2.13)$$

Substituting Eq. 2.13 into Eq.2.6 yields the momentum equation for a generalized Newtonian fluid as follows:

$$\rho \left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right) = -\nabla P + \nabla \cdot (\eta(\dot{\gamma}) \mathbf{D}) + \rho \mathbf{g}. \quad (2.14)$$

Shear thinning (pseudoplastic) fluids are the most common types of time independent non-Newtonian properties exhibiting fluids. At very low and very high strain rates the fluid exhibits a Newtonian behavior, which yields two limiting values of shear viscosity. A zero shear viscosity η_0 at zero strain rate and infinite shear viscosity η_∞ at infinite strain rate, this effect is illustrated in detail in Fig. 2.3. Different empirical models have been proposed to represent the shear thinning behavior, however only two of the more widely used models are examined in this work.

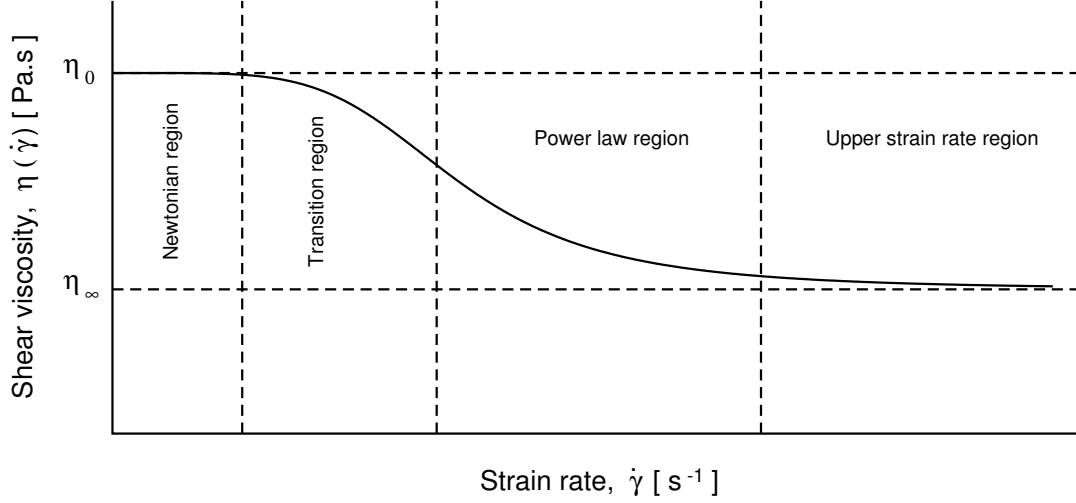


Figure 2.3: The relationship between shear viscosity and strain rate for a shear-thinning fluid.

2.3.1 Power Law Model

A relationship between shear stress and strain rate is described for a power-law fluid in the form of

$$\eta(\dot{\gamma}) = K \dot{\gamma}^{n-1}, \quad (2.15)$$

where K is the consistency index of the fluid with units $[Pa \cdot s^n]$ which reflects the vertical shift in the viscosity curve of power law region, and n is the dimensionless power-law index such that $(n - 1)$ represents the slope of the viscosity curve $\eta(\dot{\gamma})$ in the power law region and reflects how close the fluid is to Newtonian.

If $n < 1$, the fluid exhibits shear-thinning properties, and if $n > 1$, the fluid shows shear-thickening behavior. The fluid behaves like a Newtonian fluid if the power-law and the consistency indexes are equal to one and fluid viscosity, respectively.

Although the power-law model is one of the most widely used models among many engineers because of its simple representation for shear thinning fluids and its capability to get analytical solutions for many problems, this model does not take into account the limiting shear viscosities at zero and infinite strain rates, i.e, this model cannot describe fluid behavior outside the power law region [77].

2.3.2 Carreau-Yasuda Model

In order to improve the power-law model, the behaviour outside of the power-law region needs to be defined. A better fitting is achieved with the four parameter Carreau model that is intended for shear-thinning fluids ($n < 1$) [78] and defined as

$$\eta - \eta_{\infty} = (\eta_0 - \eta_{\infty}) [1 + (k \dot{\gamma})^a]^{\frac{n-1}{a}} , \quad (2.16)$$

in which k is a time constant with units of time describing the transition region in the viscosity curve whose reciprocal gives the critical strain rate at which the fluid changes from the constant viscosity behavior to the power-law behavior, a is a dimensionless constant which affects the shape of the transition region (e.g., increasing a sharpens the transition), and the dimensionless power-law index n describes the slope of viscosity curves $(\eta - \eta_{\infty}) / (\eta_0 - \eta_{\infty})$ in the power-law region. As seen in Fig. 2.3 this model takes the limiting values of the viscosity at extreme strain rates

into account.

The Bird-Carreau model is given by taking a to be 2 in Eq. (2.16), to get

$$\eta - \eta_\infty = (\eta_0 - \eta_\infty) \left[1 + (k \dot{\gamma})^2 \right]^{\frac{n-1}{2}} . \quad (2.17)$$

2.4 Numerical Methods

2.4.1 Finite Volume Method on Static Grids

The finite volume method (FVM) is one of the most versatile discretization techniques used in CFD. The most compelling feature of the FVM is that the resulting solution satisfies the conservation of quantities such as mass, momentum, energy, and species. Its also an ideal method for computing discontinuous solutions arising in compressible flows, and its preferred while solving partial differential equations containing discontinuous coefficients.

The first step in FVM is to partition the computational domain into finite small regions, called cells or control volumes CV, where the variable of interest is located at the centroid of the control volume. These CVs do not overlap and completely cover the computational domain. The second step is to integrate the differential form of the governing equations over each CV. The next step is to discretize the integral

equations by performing numerical integration, applying interpolation schemes, and substituting in the finite difference approximations. The resulting equations are called the discrete (system of algebraic) equations, that express the conservation principles for the variable inside the CV. The description of the basic FVM below follows in part that given in the thesis of Jasak [79].

Figure 2.4 shows a typical CV of hexahedron shape bounded by a set of flat faces, such that face is shared with only one neighboring CV.

V_P stands for the volume of the CV with centroid P , f is a computational point at the center of a face whose has area is S_f , \mathbf{n}_f is the face unit normal vector, N is the computational point of a neighboring CV, \mathbf{d}_f is the vector between the computational points P and N , and \mathbf{r}_P is the vector between the origin and P .

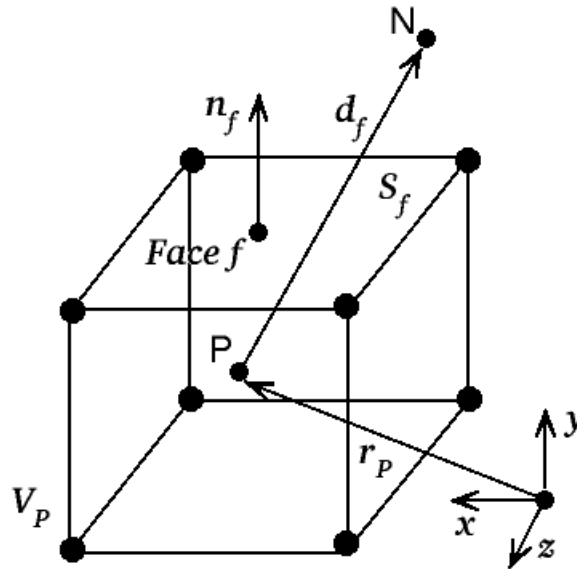


Figure 2.4: Discretization of the computational domain using finite arbitrary hexahedron control volumes.

The locations of the computational points P and f at the centroid of the CV and the face are given by \mathbf{x}_P and \mathbf{x}_f , such that

$$\int_{V_P} (\mathbf{x} - \mathbf{x}_P) dV = \mathbf{0}, \quad (2.18)$$

$$\int_f (\mathbf{x} - \mathbf{x}_f) dS = \mathbf{0}. \quad (2.19)$$

The general unsteady convection-diffusion equation for a tensorial quantity ϕ over a given CV has the following differential form:

$$\underbrace{\frac{\partial(\rho\phi)}{\partial t}}_{\text{temporal derivative}} + \underbrace{\nabla \cdot (\rho \mathbf{u} \phi)}_{\text{convective term}} = \underbrace{\nabla \cdot (\rho \Gamma_\phi \nabla \phi)}_{\text{diffusion term}} + \underbrace{q_\phi(\phi)}_{\text{source term}}, \quad (2.20)$$

where Γ_ϕ is the diffusion coefficient and $q_\phi(\phi)$ is the volume source/sink of ϕ . The integration of Eq. (2.20) over a CV yields

$$\int_{V_P} \frac{\partial(\rho\phi)}{\partial t} dV + \int_{V_P} \nabla \cdot (\rho \mathbf{u} \phi) dV = \int_{V_P} \nabla \cdot (\rho \Gamma_\phi \nabla \phi) dV + \int_{V_P} q_\phi(\phi) dV. \quad (2.21)$$

Using the Gauss divergence theorem, Eq. (2.21) can be written as follows:

$$\int_{V_P} \frac{\partial(\rho\phi)}{\partial t} dV + \oint_{\partial V_P} (\rho \mathbf{u} \phi \cdot \mathbf{n}) dS = \oint_{\partial V_P} (\rho \Gamma_\phi \nabla \phi \cdot \mathbf{n}) dS + \int_{V_P} q_\phi(\phi) dV, \quad (2.22)$$

where \mathbf{n} is the outward-pointing unit normal vector and ∂V_P is the surface bounding the volume V_P .

By integrating Eq. (2.22) in time over a small interval, we get the most general form:

$$\begin{aligned} \int_t^{t+\delta t} \left(\int_{V_P} \frac{\partial(\rho\phi)}{\partial t} dV + \oint_{\partial V_P} (\rho \mathbf{u} \phi \cdot \mathbf{n}) dS - \oint_{\partial V_P} (\rho \Gamma_\phi \nabla \phi \cdot \mathbf{n}) dS \right) dt \\ = \int_t^{t+\delta t} \left(\int_{V_P} q_\phi(\phi) dV \right) dt. \end{aligned} \quad (2.23)$$

A linear variation in FVM is used to approximate ϕ in space and time around the computational point P . This approximation is a second-order accurate and it is represented by:

$$\phi(\mathbf{x}) = \phi_P + (\mathbf{x} - \mathbf{x}_P) \cdot (\nabla \phi)_P, \quad (2.24)$$

$$\phi(t + \delta t) = \phi^t + \delta t \left(\frac{\partial \phi}{\partial t} \right)^t, \quad (2.25)$$

where $\phi_P = \phi(\mathbf{x}_P)$ and $\phi^t = \phi(t)$. Discretization in space and time has been performed in the following sections below.

2.4.1.1 Spatial Discretization of the Convection Term

Since each CV is bounded by a finite number of flat faces, then the surface integral can be written as

$$\oint_{\partial V_P} (\rho \mathbf{u} \phi \cdot \mathbf{n}) dS = \sum_f \left(\int_f (\rho \mathbf{u} \phi \cdot \mathbf{n}) dS \right). \quad (2.26)$$

The term inside the integral on the right-hand side of the above equation, i.e. $\rho \mathbf{u} \phi$, can be approximated by the assumption of linear variation around the point f given in Eq. 2.24 to get

$$\rho \mathbf{u} \phi(\mathbf{x}) = (\rho \mathbf{u} \phi)_f + (\mathbf{x} - \mathbf{x}_f) \cdot (\nabla (\rho \mathbf{u} \phi))_f. \quad (2.27)$$

Hence, the integral inside the sum above is approximated as following:

$$\int_f (\rho \mathbf{u} \phi \cdot \mathbf{n}_f) dS = (\rho \mathbf{u} \phi)_f \cdot \int_f \mathbf{n}_f dS + (\nabla (\rho \mathbf{u} \phi))_f : \int_f (\mathbf{x} - \mathbf{x}_f) \mathbf{n}_f dS, \quad (2.28)$$

where $(\text{tensor})_f$ stands for the value of tensor at the middle of the face f . By assuming that the outward-pointing unit normal vector \mathbf{n}_f is constant on face f , Eq. (2.28) becomes

$$\int_f (\rho \mathbf{u} \phi \cdot \mathbf{n}_f) dS = (\rho \mathbf{u} \phi)_f \cdot \left(\mathbf{n}_f \int_f dS \right) + (\nabla (\rho \mathbf{u} \phi))_f : \left(\mathbf{n}_f \int_f (\mathbf{x} - \mathbf{x}_f) dS \right). \quad (2.29)$$

By substituting Eq. (2.19) into Eq. (2.29), we get

$$\int_f (\rho \mathbf{u} \phi \cdot \mathbf{n}_f) dS = (\rho \mathbf{u} \phi)_f \cdot \mathbf{S}, \quad (2.30)$$

where $\mathbf{S} = \mathbf{n}_f S_f$ is the face outward area vector. Replace the integral in the right-hand side of Eq. 2.26 by Eq. 2.30 to get

$$\begin{aligned} \oint_{\partial V_P} (\rho \mathbf{u} \phi \cdot \mathbf{n}) dS &= \sum_f (\rho \mathbf{u} \phi)_f \cdot \mathbf{S} \\ &= \sum_f \mathbf{S} \cdot (\rho \mathbf{u})_f \phi_f \\ &= \sum_f F \phi_f, \end{aligned} \quad (2.31)$$

where $F = \mathbf{S} \cdot (\rho \mathbf{u})_f$ is the mass flux through the face f . It can be founded by interpolating the values of ρ and \mathbf{u} at faces from the values at the centroids. To estimate the value of ϕ_f that appears in Eq. (2.31), A weighted average approach is used as described below:

$$\phi_f = \lambda_f \phi_P + (1 - \lambda_f) \phi_N \quad (2.32)$$

Depending on the choice of λ_f , three basic methods are reproduced, varying in the stability and accuracy degree as follows:

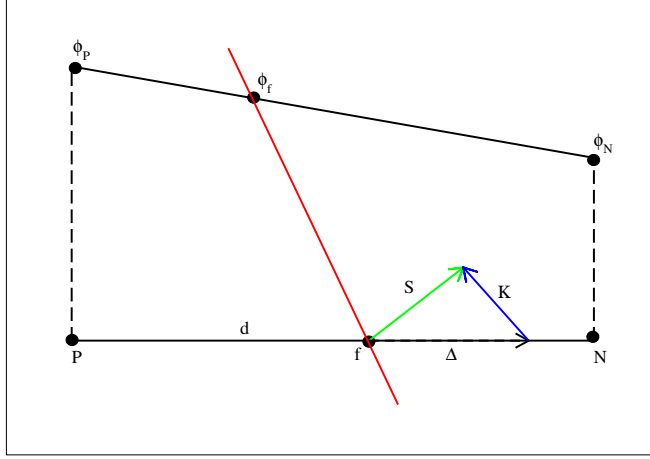


Figure 2.5: Schematic diagram for face interpolation and non-orthogonality treatment in the over-relaxed approach.

1. Central Differencing (CD)

The interpolation factor λ_f in Eq. (2.32) is defined as the ratio of the distance between the face f and the neighboring cell centroid N , and the distance between the centroids P and N as shown in Figs 2.4 and 2.5.

$$\lambda_f = \frac{\overline{fN}}{\overline{PN}}. \quad (2.33)$$

If the mesh is uniform then $\lambda_f = \frac{1}{2}$. The method is second-order but unphysical oscillations appear in the solution for convection-dominated problems, which often makes the solution unbounded. More details are found in Chapter 14 of Hoffman [80] and Chapter 4 of Wesselin [81].

2. Upwind Differencing (UD)

In UD, λ_f in Eq. (2.32) is defined as:

$$\lambda_f = \begin{cases} 1 & \text{if } F \geq 0, \\ 0 & \text{if } F < 0. \end{cases} \quad (2.34)$$

The UD method approximates ϕ_f based on the direction of the flow (or flux), hence the unphysical oscillations are removed and the method becomes bounded and stable. However, it is a first-order accurate because it uses the first-order backward differencing [79], and so it violates the order of accuracy of the discretization. Recall that, $F = \mathbf{S} \cdot (\rho \mathbf{u})_f$ is the flux.

3. Blended Differencing (BD)

The BD method is a combination between CD and UD methods, its developed to preserve the accuracy and boundedness and defined as:

$$\phi_f = (1 - \lambda_f) (\phi_f)_{UD} + \lambda_f (\phi_f)_{CD}, \quad (2.35)$$

where $\lambda_f \in (0, 1)$ is the blending factor that regulate the amount of introduced diffusion [79], $(\phi_f)_{CD}$ and $(\phi_f)_{UD}$ are the face values of ϕ computed by the CD and UD, respectively.

2.4.1.2 Spatial Discretization of the Diffusion Term

By using a similar approach as before, the diffusion term is discretized as

$$\begin{aligned}
 \oint_{\partial V_P} (\rho \Gamma_\phi \nabla \phi \cdot \mathbf{n}) dS &= \sum_f (\rho \Gamma_\phi \nabla \phi)_f \cdot \mathbf{S} \\
 &= \sum_f (\rho \Gamma_\phi)_f \mathbf{S} \cdot (\nabla \phi)_f.
 \end{aligned} \tag{2.36}$$

If the mesh is orthogonal, i.e, vectors \mathbf{d} and \mathbf{S} are parallel, then the term $\mathbf{S} \cdot (\nabla \phi)_f$ can be approximated by:

$$\mathbf{S} \cdot (\nabla \phi)_f = |\mathbf{S}| \frac{\phi_N - \phi_P}{|\mathbf{d}|}. \tag{2.37}$$

Otherwise, as shown in Fig. 2.5, the term $\mathbf{S} \cdot (\nabla \phi)_f$ is broken up into two components as follows:

$$\mathbf{S} \cdot (\nabla \phi)_f = \underbrace{\Delta \cdot (\nabla \phi)_f}_{\text{orthogonal contribution}} + \underbrace{\mathbf{K} \cdot (\nabla \phi)_f}_{\text{non-orthogonal contribution}} \tag{2.38}$$

$$= |\Delta| \frac{\phi_N - \phi_P}{|\mathbf{d}|} + \mathbf{K} \cdot (\nabla \phi)_f, \tag{2.39}$$

where Δ is parallel to the vector \mathbf{d} and $\mathbf{K} = \mathbf{S} - \Delta$. The face value of $\nabla\phi$ that appears in Eq.(2.39) can be approximated by using the weighted average method as:

$$(\nabla\phi)_f = \lambda_f (\nabla\phi)_P + (1 - \lambda_f) (\nabla\phi)_N, \quad (2.40)$$

as an example λ_f is the same as in Eq. (2.33) and $(\nabla\phi)_P$, as well $(\nabla\phi)_N$, can be approximated as follows:

$$\int_{V_P} \nabla\phi dV = \oint_{\partial V_P} (\phi \cdot \mathbf{n}) dS \quad (2.41)$$

$$(\nabla\phi)_P V_P = \sum_f \left(\int_f (\phi \cdot \mathbf{n}_f) dS \right) \quad (2.42)$$

$$(\nabla\phi)_P = \frac{1}{V_P} \sum_f \mathbf{S} \phi_f. \quad (2.43)$$

The right hand-side of Eq. (2.41) is done by using the Gauss divergence theorem. The left hand-side of Eq. (2.42) is done by applying the Gauss one-point centroidal rule, The value of ϕ inside the integral in Eq. (2.42) is approximated at the face f by using the linear variation in Eq. (2.24). Equation (2.43) is done by assuming that \mathbf{n}_f is constant and by using Eq. (2.19). To find Δ vector and hence \mathbf{K} vector that appears in the non-orthogonal contribution part, two approaches are presented when the mesh is non-orthogonal:

1. Minimum Correction Method

In this method Δ is defined to be the orthogonal projection of \mathbf{S} onto \mathbf{d} as:

$$\Delta = \frac{\mathbf{S} \cdot \mathbf{d}}{|\mathbf{d}|^2} \mathbf{d}. \quad (2.44)$$

This approach minimizes the non-orthogonal contribution by choosing \mathbf{K} to be orthogonal to Δ .

2. Over-Relaxed Method

In this method Δ is defined as:

$$\Delta = \frac{|\mathbf{S}|^2}{\mathbf{d} \cdot \mathbf{S}} \mathbf{d}. \quad (2.45)$$

According to Jasak [79], this method is the most robust approach to handle the non-orthogonality contribution from the aspect of boundedness, accuracy, and computational time.

2.4.1.3 Spatial Discretization of the Source Term

Any terms of the transport equation that cannot be written as convection, diffusion or temporal terms are treated as sources. Before the actual discretization, the source

term needs to be linearized as

$$q_\phi(\phi) = q_{\mathbf{u}}(\phi) + q_{\mathbf{p}}(\phi)\phi, \quad (2.46)$$

The importance of the linearization becomes clear in implicit calculations, and it is advisable to treat the source term as implicitly as possible [79]. The volume integral of the source term can be approximated by using Gauss one-point centroidal rule, thus the volume integral form of Eq. (2.46) becomes:

$$\int_{V_P} q_\phi(\phi) dV = (q_\phi(\phi))_P V_P \quad (2.47)$$

$$= (q_{\mathbf{u}}(\phi) + q_{\mathbf{p}}(\phi)\phi)_P V_P \quad (2.48)$$

$$= (q_{\mathbf{u}}(\phi))_P V_P + (q_{\mathbf{p}}(\phi))_P V_P \phi_P. \quad (2.49)$$

For simplicity, the above equation can be written as:

$$\int_{V_P} q_\phi(\phi) dV = q_{\mathbf{u}} V_P + q_{\mathbf{p}} V_P \phi_P. \quad (2.50)$$

2.4.1.4 Temporal Discretization

Assuming that the CVs do not change in time, the temporal derivative after applying the one-point centroid rule simplifies to:

$$\int_{V_P} \frac{\partial(\rho\phi)}{\partial t} dV = \left(\frac{\partial(\rho\phi)}{\partial t} \right)_P V_P. \quad (2.51)$$

Using Eqs (2.31, 2.36, 2.50 and 2.51), Eq. (2.23) becomes:

$$\begin{aligned} \int_t^{t+\delta t} \left(\left(\frac{\partial(\rho\phi)}{\partial t} \right)_P V_P + \underbrace{\sum_f F\phi_f}_{\text{convection term}} - \underbrace{\sum_f (\rho\Gamma_\phi)_f \mathbf{S} \cdot (\nabla\phi)_f}_{\text{diffusion term}} \right) dt \\ = \int_t^{t+\delta t} \underbrace{(q_u V_P + q_p V_P \phi_P)}_{\text{source term}} dt. \end{aligned} \quad (2.52)$$

The above equation is usually called the “semi-discretized” form of the transport equation. Taking in mind the prescribed variation of the function in time (2.25), the

temporal integrals and the time derivative can be calculated directly as:

$$\left(\frac{\partial (\rho\phi)}{\partial t} \right)_P = \frac{\rho_P^n \phi_P^n - \rho_P^o \phi_P^o}{\delta t} \quad (2.53)$$

$$\int_t^{t+\delta t} \phi(t) dt = (b\phi^o + [1-b]\phi^n)\delta t, \quad (2.54)$$

where $\phi^n = \phi(t + \delta t)$, $\phi^o = \phi(t)$ and b is a constant.

Assuming that the density and diffusivity do not change in time, Eqs (2.52, 2.53 and 2.54) give:

$$\begin{aligned} \frac{\rho_P \phi_P^n - \rho_P \phi_P^o}{\delta t} V_P &+ \sum_f [(1-b)F\phi_f^n + bF\phi_f^o] \\ &- \sum_f [(1-b)(\rho\Gamma_\phi)_f \mathbf{S} \cdot (\nabla\phi)_f^n + b(\rho\Gamma_\phi)_f \mathbf{S} \cdot (\nabla\phi)_f^o] \\ &= q_u V_P + (1-b)q_p V_P \phi_P^n + bq_p V_P \phi_P^o. \end{aligned} \quad (2.55)$$

This temporal discretization leads to the first-order explicit Euler method (FE) when $b = 1$, the first-order bounded Euler method (BE) when $b = 0$, and the second-order Crank-Nicholson method (CN) when $b = 0.5$. The CN method of temporal discretization is unconditionally stable, but does not guarantee boundedness of the

solution. As in the case of the convection term, boundedness can be obtained if the equation is discretized to first order temporal accuracy.

Equation (2.55) requires the face values of ϕ and $(\nabla\phi)$ as well as the cell values for both old and new time-level. The face values are calculated from the cell values on each side of the face, using the appropriate differencing scheme in Eq. (2.32) for the convection term, and Eqs (2.39) and (2.40) for diffusion term.

In explicit discretization (FE), the face values of ϕ and $\nabla\phi$ are determined from the old time-field:

$$\phi_f = \lambda_f \phi_P^o + (1 - \lambda_f) \phi_N^o \quad (2.56)$$

$$\mathbf{S} \cdot (\nabla\phi)_f = |\boldsymbol{\Delta}| \frac{\phi_N^o - \phi_P^o}{|\mathbf{d}|} + \mathbf{K} \cdot (\nabla\phi)_f^o. \quad (2.57)$$

Although this approach is very fast in computations, it does not guarantee boundedness especially when the Courant number C_o is greater than one. Courant number is defined as:

$$C_o = \frac{|\mathbf{u}|_f \delta t}{\mathbf{d}} \quad (2.58)$$

In implicit discretization, the face values are determined in terms of the new time-level cell values:

$$\phi_f = \lambda_f \phi_P^n + (1 - \lambda_f) \phi_N^n \quad (2.59)$$

$$\mathbf{S} \cdot (\nabla \phi)_f = |\boldsymbol{\Delta}| \frac{\phi_N^n - \phi_P^n}{|\mathbf{d}|} + \mathbf{K} \cdot (\nabla \phi)_f^n. \quad (2.60)$$

Although this approach is still a first order accurate and takes more computational time than the previous approach, it guarantees boundedness and stability of the system regardless the C_o limits. A second-order Backward Differencing method can be used as well:

$$\begin{aligned} \frac{3\rho_P \phi_P^n - 4\rho_P \phi_P^o + \rho_P \phi_P^{oo}}{2\delta t} V_P &+ \sum_f F \phi_f^n - \sum_f (\rho \Gamma_\phi)_f \mathbf{S} \cdot (\nabla \phi)_f^n \\ &= q_u V_P + q_p V_P \phi_P^n, \end{aligned} \quad (2.61)$$

where $\phi^{oo} = \phi(t - \delta t)$. Once again, the boundedness of the solution using this method cannot be guaranteed, for more details about this temporal scheme refer to Jasak [79]. Since ϕ_f and $(\nabla \phi)_f$ also depend on values of ϕ in the neighboring cells, Eq. (2.55)

and/or Eq. 2.61 produce an algebraic equation for each CV that must be solved for ϕ_P^n :

$$a_P \phi_P^n + \sum_{nb} a_{nb} \phi_{nb}^n = R_P. \quad (2.62)$$

The summation in Eq. (2.62) is over all the neighboring cells of the cell that has centroid P . For all CVs, this kind of equations can be assembled in a system of algebraic equations of the form:

$$\mathbf{C} \cdot \mathbf{y} = \mathbf{rhs}, \quad (2.63)$$

where \mathbf{C} is a sparse matrix with coefficients a_P on the diagonal and a_{nb} off the diagonal. The sparseness pattern of the matrix depends on the order in which the CVs are labeled, with every off-diagonal coefficient above and below the diagonal corresponding to one of the faces in the mesh. \mathbf{y} is the vector with the unknown values of ϕ on all CVs. \mathbf{rhs} is the source vector which includes all terms that can be evaluated without knowing ϕ^n , i.e. it contains the values of the constant part of the source term ($q_u V_P$), and the parts of convection term, diffusion term and temporal derivative at the old time level related to ϕ^o . Numerical approaches to solve the resulting equations are discussed in details in book of Trefethen and Bau [82].

2.4.2 Finite Volume Method on Non-Static Grids

The conservation equations are usually formulated for static boundary but the recent need to describe turbulent flows in complex geometries, especially with moving boundaries, leads to rewrite equations taking into account the motion of the domain. If the CV is not constant within the time due to a moving boundary, the only change in conservation equation will be the appearance of relative velocity in convective terms. The integral form of the governing equation for the a general tensorial property ϕ over an arbitrary moving control volume V , bounded by a closed surface S with an outward pointing unit normal vector \mathbf{n} , is given by (Jasak and Tukovic [83]):

$$\frac{\partial}{\partial t} \int_V (\rho\phi) dV + \oint_S \rho\phi \mathbf{u}_{\text{relative}} \cdot \mathbf{n} dS - \oint_S \rho\Gamma_\phi \nabla\phi \cdot \mathbf{n} dS = \int_V \mathbf{q}_\phi dV. \quad (2.64)$$

This equation is similar to Eq. (2.22), but the velocity of the fluid \mathbf{u} in the convection term is replaced by the relative velocity $\mathbf{u}_{\text{relative}} = \mathbf{u} - \mathbf{u}_b$, where \mathbf{u}_b is the velocity of the boundary (face) surface S , ρ is the fluid density, Γ_ϕ is the diffusion coefficient and \mathbf{q}_ϕ is the volume source/sink of ϕ . As the CV is no longer fixed in space, mass source can be appear in the mass conservation equation. To avoid this, an additional requirement must be satisfied:

$$\frac{\partial}{\partial t} \int_V dV - \oint_S \mathbf{u}_b \cdot \mathbf{n} dS = 0. \quad (2.65)$$

This is known as the Geometric Conservation Law (GCL) [84], which describes the relationship between the rate of change of the volume V and the velocity of the boundary surface \mathbf{u}_b .

Following the discretization approaches used in Sections (2.4.1.1)–(2.4.1.4) and using the definition of the face outward area vector $\mathbf{S} = \mathbf{n}_f S_f$ where S_f is the surface area of a cell face, a second-order FV discretization of Eqs (2.64) and (2.65) transforms the surface integrals into sums of face integrals and approximates them to second order using the mid-point rule. Further, discretization of the above equations depends on the chosen temporal integration scheme and it allows for calculating the mesh motion flux on the basis of the swept volume; in the simplest case of the first-order Euler implicit integration that is used in this thesis, the full discretized forms of the above equations for cell P that is bounded by an arbitrary number N_f of cell faces are:

$$\begin{aligned} \frac{\rho_P^n \phi_P^n V_P^n - \rho_P^o \phi_P^o V_P^o}{\delta t} &+ \sum_f^{N_f} \rho_f^n \phi_f^n (F_f^n - F_{bf}^n) \\ &= \sum_f^{N_f} (\rho \Gamma_\phi)_f^n \nabla \phi_f^n \cdot \mathbf{S}^n + q_\phi^n V_P^n, \end{aligned} \quad (2.66)$$

$$\frac{V_P^n - V_P^o}{\delta t} = \sum_f^{N_f} F_{bf}^n, \quad (2.67)$$

where $F_f = \mathbf{S} \cdot \mathbf{u}_f$ is the face value of fluid flux, $F_{bf} = \mathbf{S} \cdot \mathbf{u}_{bf}$ is the face value of mesh motion flux (or volumetric face flux) which accounts for the grid convection, the subscript P represents the cell values, f the face values, V_P is the cell volume, and superscripts n and o the “new” and “old” time level.

Cell volume V_P^n and V_P^o are calculated directly from geometric considerations and satisfy the discrete form of the GCL. The fluid flux F is usually obtained as a part of the solution algorithm and satisfies the conservation requirements, while the mesh motion flux F_b is calculated as the volume swept by the face in motion during the current time-step rather than from the grid velocity \mathbf{u}_b , making it consistent with the cell volume calculation. For more details about the swept volume calculations in OpenFOAM, refer to [85] and [86].

As a special case of the previous discussion, the integral form of the governing equations for incompressible Navier-Stokes system is given by:

$$\frac{\partial}{\partial t} \int_V dV + \oint_S \mathbf{u}_{\text{relative}} \cdot \mathbf{n} dS = 0, \quad (2.68)$$

$$\frac{\partial}{\partial t} \int_V \mathbf{u} dV + \oint_S \mathbf{u} \mathbf{u}_{\text{relative}} \cdot \mathbf{n} dS - \oint_S \nu \nabla \mathbf{u} \cdot \mathbf{n} dS = - \oint_S P \cdot \mathbf{n} dS. \quad (2.69)$$

Continuity Eq. (2.68) has been produced by setting ϕ to one and \mathbf{q}_ϕ to zero in Eq. (2.64), taking in mind the GCL requirement. Momentum Eq. (2.69) has been produced by setting ϕ to fluid velocity \mathbf{u} , Γ_ϕ to fluid kinematic viscosity ν , and \mathbf{q}_ϕ to kinematic pressure gradient $-\nabla P$ in Eq. (2.64) (we assume in this case, there are no

additional source terms).

The grid can be considered stationary in the treatment because, within a time step, the pressure-correction step operates in an absolute velocity field. In order to provide an informative description of the `transientSimpleDyMFoam` flow solver (discussed later), the discrete forms of the above equations for a fixed CV with a centroid P are:

$$\sum_f^{N_f} \mathbf{u}_f \cdot \mathbf{S} = 0, \quad \text{or simply:} \quad \sum_f^{N_f} F_f = 0, \quad (2.70)$$

$$\frac{\delta \mathbf{u}_P}{\delta t} V_P + \sum_f^{N_f} \mathbf{u}_f F_f - \sum_f^{N_f} \nu_f \nabla \mathbf{u}_f \cdot \mathbf{S} = - \sum_f^{N_f} P_f \mathbf{S}. \quad (2.71)$$

The discrete momentum Eq. (2.71) can be transformed into a linear system of equations that, for each computational cell center P surrounded by N_{nb} neighboring cells, obtains a form:

$$a_P \mathbf{u}_P + \sum_{nb}^{N_{nb}} a_{nb} \mathbf{u}_{nb} = R_P. \quad (2.72)$$

The right-hand-side (R_P) of the equation contains the source contributions arising from the discretizations of the transient, convection and diffusion terms and the pressure gradient. For convenience, the contributions are split into velocity- and pressure-dependent parts as:

$$R_P = r_P(\mathbf{u}) - \nabla P, \quad (2.73)$$

recognizing that $\nabla P = \frac{1}{V_P} \sum_f^{N_f} P_f \mathbf{S}$. Using short-hand notation

$$\mathbf{H}(\mathbf{u}) = - \underbrace{\sum_{nb}^{N_{nb}} a_{nb} \mathbf{u}_{nb}}_{\text{transport part}} + \underbrace{r_P(\mathbf{u})}_{\text{source part}}, \quad (2.74)$$

we can write Eq. (2.72) as

$$a_P \mathbf{u}_P = \mathbf{H}(\mathbf{u}) - \nabla P. \quad (2.75)$$

From this formulation, a new face velocity can be defined that is interpolated onto the cell faces using cell center values:

$$\mathbf{u}_P = \frac{\mathbf{H}(\mathbf{u})}{a_P} - \frac{\nabla P}{a_P}, \quad (2.76)$$

$$\mathbf{u}_f = \left(\frac{\mathbf{H}(\mathbf{u})}{a_P} \right)_f - \left(\frac{\nabla P}{a_P} \right)_f. \quad (2.77)$$

The discrete pressure equation is obtained by substituting Eq. (2.77) into the continuity requirement of Eq. (2.70), yielding

$$\sum_f^{N_f} \left[\left(\frac{1}{a_P} \right)_f (\nabla P)_f \right] \cdot \mathbf{S} = \sum_f^{N_f} \left(\frac{\mathbf{H}(\mathbf{u})}{a_P} \right)_f \cdot \mathbf{S}. \quad (2.78)$$

In OpenFOAM the discrete pressure equation reaches its final form by defining an intermediate velocity field and evaluating the corresponding flux field, which does not

satisfy the continuity requirement, that is:

$$\mathbf{u}^* = \left(\frac{\mathbf{H}(\mathbf{u})}{a_P} \right), \quad (2.79)$$

$$F_f^* = (\mathbf{u}_f^* \cdot \mathbf{S}), \quad (2.80)$$

$$\sum_f^{N_f} \left[\left(\frac{1}{a_P} \right)_f (\nabla P)_f \right] \cdot \mathbf{S} = \sum_f^{N_f} F_f^*. \quad (2.81)$$

Considering the discretized form of the Navier-Stokes system (2.75 and 2.78), the form of the equations shows linear dependence of velocity on pressure and vice-versa. Two approaches to deal with this coupling are presented; The first one operate by solving the complete system of equations simultaneously over the whole domain. Such a procedure might be considered when the number of computational points is small and the number of simultaneous equations is not too large. The cost of a simultaneous solution is so expensive, both in the number of operations and memory requirements. The second approach operate by solving the equations in sequential manner and it is called the segregated approach. A special treatment is required in order to establish the necessary inter-equation coupling.

2.4.3 Description of TransientSimpleDyMFoam Solver

Two flow solvers have been developed by OpenFOAM community for simulating the flow systems described previously, a `transientSimpleDyMFoam` [87] and `pimpleDyMFoam` [88]. These solvers employing an extensive libraries in OpenFOAM, utilizing segregated velocity-pressure coupling algorithms, and featuring both automatic mesh motion and deformation functionality. The `transientSimpleDyMFoam` solver uses the SIMPLE algorithm [89] while `pimpleDyMFoam` uses the PIMPLE algorithm, which is a combination of the SIMPLE and PISO [90] algorithms.

The main difference between these two solvers lies in the pressure correction algorithm. For these solvers, the pressure field is obtained by deriving a pressure correction equation and enforcing mass continuity [91]. The way this is achieved is as follow [92]. To initiate SIMPLE algorithm, the pressure field is predicted. The velocity field is computed by solving the discretized momentum equations with the predicted pressure. This first solution is substituted into the equation of continuity in order to calculate correction factors. In the SIMPLE algorithm, the velocity corrections contributed by cells adjacent to the pole cell are neglected. This approximation does not affect the final solution if convergence is reached. It is acceptable for steady state simulations or if small time steps are used in an unsteady calculation. This step enables to correct the pressure and the velocity. In the last step, all other transport equations are solved. To avoid divergence, some under-relaxation is used to get the

new pressure value. The relaxation needs to be large enough to move the process forward but small enough to avoid divergence.

The PISO algorithm may be seen as an extension of the SIMPLE algorithm with a further corrector step. The first steps of the SIMPLE algorithm are realized. The next step of the PISO algorithm consists in solving a second pressure correction equation without neglecting any term. In the PISO algorithm, no under-relaxation factor is used. It has been shown that despite the increase of computational effort to solve the second pressure equation, the PISO algorithm is efficient and fast.

The PIMPLE algorithm is a combination of these two algorithms. With default parameters, the PISO part of the algorithm is used. To benefit from the SIMPLE part of the algorithm, relaxation factors have to be introduced. The PIMPLE algorithm acts like the PISO one, with under-relaxation correction at the end.

The choice between these two flow solvers depends on user needs as stated by Romain et al. [93]. If robust simulations are needed, the SIMPLE algorithm is simpler to use through the `transientSimpleDyMFoam` solver. However if the need is for an optimized calculation, the PIMPLE algorithm (with `pimpleDyMFoam`) can be set up to benefit from the convergence of the SIMPLE algorithm and the precision and speed of the PISO one. According to Auvinen et al. [87], PIMPLE solver produces an accurate transient solution but suffers from inefficient temporal time marching due to a restricting limitation on the maximum time step length. By contrast, the solver featuring SIMPLE does allow more aggressive time marching-naturally at the expense

of temporal accuracy, which is critical in performing efficient time-accurate analysis of flow systems whose transient behavior evolves over a comparatively long time. In this thesis, the choice of a robust solution is preferred over an optimized calculation and therefore the simulations are carried out with `transientSimpleDyMFoam` solver, whose principal algorithmic description is provided below.

The `transientSimpleDyMFoam` solver is a transient solver for a single-phase, incompressible, laminar flows with a dynamic moving mesh capability from the OpenFOAM package. Further, this solver can be used also for turbulent flows by activating the turbulence models in the `<case>/constant/turbulenceProperties` file.

This solver is located in the folder `OpenFOAM/OpenFOAM-2.1.x/applications/solvers/transientSimpleDyMFoam`. It uses an adaptive time step depending on the Courant number C_o defined in Eq. (2.58). To choose the new time step, a maximum Courant number C_o^m is calculated from the flow conditions, using \mathbf{u} and δt from the previous time step. The new time step δt^n is then calculated using the following expression [94]

$$\delta t^n = \min \left\{ \frac{C_o^{max}}{C_o^m} \delta t^{n-1}; \left(1 + 0.1 \frac{C_o^{max}}{C_o^m} \right) \delta t^o; 1.2 \delta t^{n-1}; \delta t^{max} \right\} \quad (2.82)$$

where δt^{n-1} is the previous time step, C_o^{max} is the pre-set maximum Courant number, and δt^{max} is the pre-set maximum time step. These pre-set values are specified by the user and they are located in the `<case>/system/controlDict` file.

In the solver we are analyzing, the equation system to be solved begins as $\mathbf{C} \cdot \mathbf{u} = \mathbf{c}$

where \mathbf{C} is a matrix, \mathbf{u} is the solution vector, and \mathbf{c} is the right hand side vector. This linear system can be written as $\mathbf{A} \cdot \mathbf{u} = \mathbf{H}$ where the matrix $\mathbf{C} = \mathbf{A} + \mathbf{B}$ and the vector $\mathbf{H} = \mathbf{c} - \mathbf{B} \cdot \mathbf{u}$. The matrix \mathbf{A} has only the diagonal elements of \mathbf{C} while the matrix \mathbf{B} has only the off-diagonal elements of \mathbf{C} . Later we will see in the code, `rUA = 1.0/UEqn.A()` and `U = rUA*UEqn.H()`. The `.A()` operator gives \mathbf{A} formed in the momentum equation `UEqn`, the list of diagonal elements as explained above. This is a scalar field because each element is a scalar and corresponds to one grid cell. The `.H()` operator is a list of the elements of the vector \mathbf{H} described above and formed in `UEqn`. If the variable \mathbf{u} is a vector, then each element of \mathbf{H} will be a vector.

Referring to the developments in Section 2.4.2 the solution procedure implemented in `transientSimpleDyMFoam` can be illustrated by the following procedure:

TIME Loop: while ($t^n < t^{end}$)

t^{end} is the pre-set value of the final (termination) time that is located in the `<case>/system/controlDict` file.

1. Calculate the Courant number by calling the `CourantNo.H` library and adjust the time step by calling the `setDeltaT.H` library to find the new time step δt^n , then increment time as: $t^n = t^{n-1} + \delta t^n$.
2. Convert face fluxes to correspond to an absolute velocity field by calling the function `makeAbsolute(phi, U)`. Since the mesh moves, the flow moves relatively to the mesh. This function makes the flux absolute for the following part

of the code. Its used because at the end of the loop the flux is made relative (discussed later). The above function computes: $F_f = \mathbf{u}_f \cdot \mathbf{S}$.

3. Apply mesh movement (and/or deformation) utilizing a chosen dynamic mesh library. If the mesh is moving we have a different geometry. The solver updates the geometry every time step before going to the SIMPLE loop by using the function `mesh.update()`. To update the geometry, we need to modify point positions (IDs in OpenFOAM) in each time step as is discussed later in Section 2.5.3.

4. Correct the flux field if the mesh has deformed. In case the mesh is moving the mass flux is corrected according to it by applying the following statement: `if (mesh.changing() && correctPhi)` and calling the library `correctPhi.H`. Basically this step computes the face value of mesh motion flux F_{bf} as the volume swept by the face in motion such that GCL is satisfied.

5. Convert face fluxes to correspond to a relative velocity field. This is done by calling the function `makeRelative(phi, U)` to make fluxes relative. This function calculates the following: $F_{\text{relative}} = \mathbf{u}_{\text{relative}} \cdot \mathbf{S} = F_f - F_{bf}$.

6. **SIMPLE Loop:** for (`k=0`; `k < nIter`; `k++`)

6.1. Build the momentum Eq. (2.75) applying implicit relaxation $0 < \alpha_{\mathbf{u}} < 1$

to increase the diagonal dominance of the coefficients matrix:

$$a_P \mathbf{u}_P^k + \frac{(1 - \alpha_{\mathbf{u}})}{\alpha_{\mathbf{u}}} a_P \mathbf{u}_P^k + \sum_{nb}^{N_{nb}} a_{nb} \mathbf{u}_{nb}^k = R_P + \frac{(1 - \alpha_{\mathbf{u}})}{\alpha_{\mathbf{u}}} a_P \mathbf{u}_P^{k-1}. \quad (2.83)$$

The pressure gradient term in R_P is calculated using the pressure distribution from an initial guess or the previous iteration. The coefficients a_P and a_{nb} (if nonlinear) are computed using velocity field from an initial guess or the previous iteration. In this equation we can solve for \mathbf{u}^k , noting that at the matrix level the terms are multiplied by cell volume V_P before the relaxation is applied. The equation is under-relaxed and this stage is called the momentum predictor.

6.2. Define an intermediate velocity field \mathbf{u}^{*k} and compute a corresponding face flux field F_f^{*k} according to Eqs (2.79) and (2.80):

$$\begin{aligned} \mathbf{u}^{*k} &= \left(\frac{\mathbf{H}(\mathbf{u}^k)}{a_P} \right), \\ F_f^{*k} &= (\mathbf{u}_f^{*k} \cdot \mathbf{S}). \end{aligned}$$

The first equation is calculated by calling $\mathbf{U} = \mathbf{rAU} * \mathbf{UEqn.H}()$ function where $\mathbf{rAU} = 1.0 / \mathbf{UEqn.A}()$. The second equation is computed by calling $\mathbf{phi} = (\mathbf{fvc}::\mathbf{interpolate}(\mathbf{U}) \& \mathbf{mesh.Sf}())$ function. Refer to the previous discussion for $\mathbf{.A}()$ and $\mathbf{.H}()$ roles in the momentum equation.

6.3. Store the pressure value of the current iteration by calling the function

`p.storePrevIter()` that computes: $P^{k-1} = P^k$.

6.4. Build the pressure Eq. 2.81 and solve for P^k .

$$\sum_f^{N_f} \left[\left(\frac{1}{a_P} \right)_f (\nabla P^k)_f \right] \cdot \mathbf{S} = \sum_f^{N_f} F_f^{*k}.$$

This done by executing and calling the following statement:

```
fvScalarMatrix pEqn
(
    fvm::laplacian(rAU, p) == fvc::div(phi)
).
```

6.5. Use this calculated pressure P^k to correct the flux field such that it fulfills the continuity requirement in Eq. 2.70:

$$F_f^k = F_f^{*k} - \left(\frac{1}{a_P} \right)_f (\nabla P^k)_f \cdot \mathbf{S}.$$

This is done by calling the function `phi -= pEqn.flux()`.

6.6. Apply an explicit relaxation to the pressure field for momentum corrector:

$$P^k = P^{k-1} + \alpha_P (P^k - P^{k-1}), \quad (2.84)$$

where α_P is the under-relaxation factor for pressure that typically takes on values within range $0.1 \leq \alpha_P \leq 0.3$. This is done by calling the

function `p.relax()`. Note that, P^{k-1} and P^k in the right-hand side of the above equation were calculated from steps 6.3. and 6.4., respectively. The pressure is explicitly relaxed in this step because in SIMPLE, there is an omission of velocity correction of neighbor cells, using this velocity correction to correct velocity is moderate. But for pressure, using this velocity correction to correct pressure is exaggerated. Thus we need to do explicit relaxation in pressure field to make pressure correction to be moderate.

6.7. Convert face fluxes to correspond to a relative velocity field. Once again, this is executed by calling the function `makeRelative(phi, U)` that calculates: $(F_{\text{relative}}^k)_f = F_f^k - F_{bf}$.

6.8. Correct the velocity field utilizing a relaxed pressure field according to Eq. 2.76, by calling the function `U -= rAU*fvc::grad(p)` that calculates:

$$\mathbf{u}_P^k = \mathbf{u}_P^{*k} - \frac{\nabla P^k}{a_P}.$$

6.9. Use the corrected velocity in previous step to update the viscosity (and/or other properties), by calling the function `turbulence->correct()`. In this step, effective viscosity in turbulence models and/or strain-rate dependent viscosity in non-Newtonian modes is computed by using \mathbf{u}_P^k from previous step 6.8..

- 6.10. If the convergence is achieved then continue to step 7. Otherwise, repeat the steps from step 6.1. utilizing the relaxed pressure from step 6.6. and corrected velocity from step 6.8. as the new guessed values for pressure and velocity. This set of conservative fluxes (or velocities) is needed to recalculate the coefficients in $\mathbf{H}(\mathbf{u})$ because the non-linear coupling effects is assumed to be very important. This non-linearity can be seen in the convection term and viscous stress tensor for non-Newtonian fluids.
7. Return to step 1 or exit time loop and terminate simulation.

The discretization in space and time discussed previously generate a linear system of algebraic equations for each variable. This system will be solved by using a suitable linear solver. The initial and final residual are the calculated residuals before and after the linear system is solved. The convergence is checked by the residual values of the velocity and pressure. If the residual of each variable is below a specific tolerance then the solver will stop. Note that, although `transientSimpleDyMFoam` solver featuring SIMPLE algorithm, it is not utilizing the residual controls to test the convergence. This solver is utilizing two PISO algorithm controls that can be specified in the `<case>\system\fvSolution` file: the number of pressure-velocity coupling loops in each time step is controlled by the keyword `nOuterCorrectors`. By increasing this number to a specific value, the convergence is improving and the grid independence becomes robust. If the mesh is non-orthogonal then step 6.4. can be

repeated for a specific number of iterations. This number is controlled by the keyword `nNonOrthogonalCorrectors`. If any of these numbers (or keywords) is set to zero, then the solver performs only one time in each time step. Refer to Appendix C.4 for more information.

2.5 Moving Grid Handling

Moving mesh provides a capability of tackling flow simulations where the spatial domain shape or the position of an internal interface changes during the simulation. This causes a problem of preserving the validity and quality of the mesh during the simulation. Examples include the boundary motion in peristaltic pumps and internal combustion engines, where the calculation of the internal points motion inside the domain is based on the prescribed motion of the boundary, free-surface flow, where the interface between the phases is captured by the mesh, and Fluid-Structure Interaction (FSI), where the deformation of a solid changes the shape of the fluid domain. Several deforming mesh techniques have been presented in the past, with various approaches to define and create a robust mesh motion solvers. Behr et al. [95, 96] use explicit algebraic expressions in the horizontal and vertical direction with a Finite Element (FE) Arbitrary Lagrangian-Eulerian (ALE) solver to simulate free-surface flows with mesh deformation. The most popular method to date is the spring analogy [97, 98], which aims to link each point of the mesh by fictitious

spring and the point motion is obtained as a response to the boundary loading or displacement. However, this approach proved to lack robustness, particularly for arbitrary unstructured polyhedral meshes. Farahat et al. [99, 100] improves the robustness of the method by proposing an additional torsional springs to control all mechanisms of invalidating a tetrahedral cell. However, Jasak and Tukovic [83] show that the cost induced by this improvement can be considered as too expensive. Laplacian smoothing approach [101, 102, 103, 104] and the pseudo-solid approach [105, 106, 107, 108, 109, 110, 111] in the ALE FEM codes are used to create a robust mesh motion solver. In an effort to simultaneously control the position of the free-boundary surface and mesh spacing next to it, Helenbrook [112] proposes the use of a biharmonic equation to govern mesh motion. Bos et al. [113] recognize the fact that the Radial Basis Function (RBF) interpolation may be formulated in purely algebraic terms rather than coded into a form of a partial differential equation. Such formulation would lead to a faster and more robust motion.

2.5.1 Grid Validity and Quality Metrics

If the shape of the domain changes in time, then the solution will be influenced, in fact, the boundary shape itself may in some cases be a part of the solution. Thus one can distinguish between boundary motion and internal point motion. Boundary motion is either prescribed by external factors, e.g. piston and valve motion for

in-cylinder flow simulation in internal combustion engines, or is a part of the solution as in free surface tracking simulations. The role of internal point motion is to accommodate changes in the domain shape (boundary motion) and preserve the validity and quality of the mesh.

2.5.1.1 Grid Validity Metrics

The investigation of mesh validity can be separated into topological and geometrical tests. The first set contains the tests that can be performed without knowing the actual point positions, while the second set deals with the shape of cells and the boundary. The job of mesh generation, `blockMesh` as an example, is to produce a mesh satisfying these requirements.

Topological validity tests consists of the following criteria:

- A point can appear in a given face only once.
- A face cannot belong to more than two cells. A boundary face can belong to only one patch.
- Two cells can share no more than one face.
- Collecting all faces from one cell and decomposing faces into edges, every edge must appear in exactly two cell faces in that given cell.

Geometrical measures (face area, cell volume, face and cell centroid, normal vector, etc. . .) are calculated by decomposing the face into triangles and cell into tetrahedra or pyramids. The tetrahedra are constructed using the cell centroid as apex and the triangles of the face decomposition as a base. Geometrical validity criteria can be summarized as follows:

- All cells and faces must be weakly convex. A face is considered convex if all triangles normals point in the same direction. For a cell, where the metrics are calculated on a tetrahedral decomposition, an equivalent convexness definition is used.
- All cells must be geometrically closed: the sum of outward-pointing face area vectors for cell faces must be zero to machine tolerance.
- For all internal faces, the dot product of the face area vector \mathbf{S} and the vector connecting the two cell centers $\mathbf{d} = \overrightarrow{PN}$, see Fig. 2.6a, must be positive; this is usually termed the orthogonality test.

2.5.1.2 Grid Quality Metrics

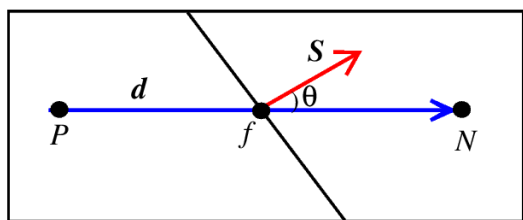
Based on the topology of the mesh changes during simulations, OpenFOAM has two mesh-manipulation approaches. The first approach is called the automatic mesh motion and it is used when the topology of the mesh is not changing during the

simulation and only the point positions change. The objective of automatic mesh motion is to determine the internal point motion to conform with given boundary motion while preserving mesh validity and quality. By contrast, the other approach is used when the topology of the mesh changes during the simulation. Note that, a face in OpenFOAM is stored as a list of point IDs, and not as a list of point coordinates. When topological changes are triggered, points are renumbered and hence there is no correspondence between old and new point IDs, so the correlation between the old and new face and/or volume is no longer valid and needs a special treatment. In this thesis automatic mesh motion approach has been used, for more details about non automatic mesh motion handling refer to Giussani [86].

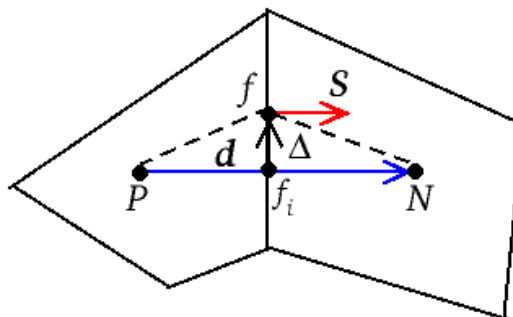
Preserving the mesh quality only relates to the geometrical tests. Once the convexness and orthogonality tests are satisfied, an initially valid mesh remains valid if no faces (triangles or quadrilaterals) and cells (tetrahedrons or hexahedrals) are inverted while the mesh in motion. The most common mesh quality metrics, as illustrated in Fig. 2.6, are:

- Orthogonality (Fig. 2.6a); cell non-orthogonality is defined by an angle $\theta \in [0^\circ, 90^\circ]$ between the face normal vector \mathbf{S} and the vector connecting the two cell centers \mathbf{d} . This angle should be small in order to minimize the truncation error of the diffusion term. Mesh orthogonality affects the gradient of the face center f and it adds diffusion to the solution.

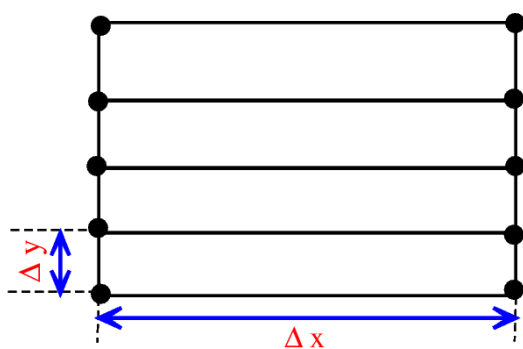
- Skewness (Fig. 2.6b); cell skewness is the deviation of the vector \mathbf{d} that connects the two cells P and N to the face center f . The deviation vector is represented with Δ and f_i is the point where the vector \mathbf{d} intersects the face f . The degree of skewness is expressed by the ratio $\frac{|\Delta|}{|\mathbf{d}|}$. When $\Delta \neq 0$, the cell is skewed, i.e., when $f_i \neq f$. Skewness affects the interpolation of the cell centered quantities to the face center f and it adds diffusion to the solution as well.
- Aspect ratio (Fig. 2.6c); mesh aspect ratio is the ratio between the longest side Δx and the shortest side Δy . Large aspect ratio is fine if gradients in the long direction are small, but usually high aspect ratio leads to smear gradients.
- Smoothness (Fig. 2.6d); also known as expansion rate, growth factor or uniformity, defines the transition in size between contiguous cells as $\frac{\Delta y_1}{\Delta y_2}$. Large transition ratios between cells add diffusion to the solution; ideally the maximum change in mesh spacing should be less than 20%.



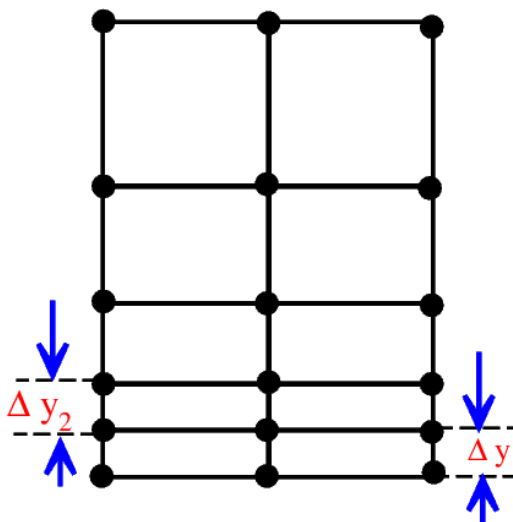
(a) Grid orthogonality



(b) Grid skewness



(c) Grid aspect ratio



(d) Grid smooth transition

Figure 2.6: Grid quality metrics

2.5.2 Diffusivity Models

In this section, we briefly introduce the main diffusivity models in openFOAM that outlined by Mordnia [114]. Diffusivity models determine how the points should be moved after solving the equation of cell motion for each time step. The diffusivity models available in the path:

`OpenFOAM/OpenFOAM-2.1.x/src/fvMotionSolver/motionDiffusivity`, and they should be read from the file `dynamicMeshDict` in the `constant` folder of the case. Two general categories to formulate the variable diffusivity ξ , the distance-based models and the quality-based models.

2.5.2.1 Quality-Based models

In this method, the diffusion field is a function of cell quality measures (i.e, mean cell non-orthogonality and/or cell skewness). The most popular models in this category are:

1. Uniform diffusivity: the mesh manipulation will be done uniformly for all moving boundaries; that is to say, all cells in each region get stretched or squeezed with the same ratio. On the other hand, the different parts of the mesh are handled uniformly, depending on their distance from the moving faces. No specific

data is needed here except the name of the diffusivity model.

2. Directional diffusivity: the mesh stretching or squeezing will be done proportional to the direction of the motion. In this case we need to specify two scalar coefficients for the model to work. One can use a third mixed coefficient as a combination of the two. In this thesis, the diffusivity model has been chosen to be directional.
3. Motion directional diffusivity: the mesh manipulation is done by prioritizing the moving body and adjusting the cells in a way that is more appropriate for the moving body, while the mesh manipulation in previous model is done by considering the slipping boundaries. For the model to work, we need to specify the scalar coefficients in this model as above.
4. Diffusivity with inverse distance: In this case the user specifies one or more boundaries and the diffusivity of the field is based on the inverse of the distance from that boundary.

2.5.2.2 Distance-Based Models

This method is used together with the quality-based method, in which the diffusion field will be a function of cell center distance L to the nearest selected boundary. These models are used with inverse-distance method above. The most popular models in

this category are:

1. Diffusivity with linear inverse distance: in this model the diffusivity field is based linearly on the inverse of the cell center distance to the nearest boundary, that is $\xi(L) = L^{-1}$.
2. Diffusivity with quadratic inverse distance: the same as above, with the only difference being a quadratic relation instead of a linear one, that is $\xi(L) = L^{-2}$.
3. Diffusivity with exponential inverse distance: in this model the field diffusivity is based on the exponential of the inverse of cell-center distance to the selected boundaries, that is $\xi(L) = e^{-L}$.

2.5.3 Grid Motion Solvers

According to the complexity of the prescribed boundary motion, mesh deformation cases can be handled either by simple expressions or by more complex functional forms.

In algebraic expressions approach [115], the effect of moving mesh on the flow field may be reformulated in terms of volumetric body force, calculated as a derivative of motion velocity, either analytically or from user-prescribed motion data. Among dynamic mesh cases, prescribed solid body motion is the easiest to deal with: the complete domain is moving with uniform displacement for each time step. Coupled

with a nonlinear flow model. This approach is efficient and accurate but it is defined for a small subset of geometries.

When the boundary motion is irregular or solution-dependent, the algebraic mesh motion expressions are not flexible. An alternative way of looking at the mesh motion problem is to consider prescribed boundary motion as a boundary condition on the mesh motion equation, and by solving the mesh motion equation, the internal point motion may be determined. Three obvious choices are considered; the Laplace equation, Solid Body Rotation stress (SBR) equation, and Radial Basis Function (RBF) interpolation. The mesh motion solvers available in the path: `OpenFOAM/OpenFOAM-2.1.x/src/fvMotionSolver/fvMotionSolvers`. In OpenFOAM, the mapping between the meshes using the `dynamicFvMesh` library happens behind the scenes, and so the FVM physics solver just has to satisfy the moving mesh terms shown in Eq. (2.64) and it is independent of the mesh. For more details about moving mesh in OpenFOAM refer to Kassiotis [116] and Mordnia [114].

The main aim of this section is to modify point positions (or IDs in OpenFOAM) in each time step. These new points are needed to update the geometry by utilizing `moveMesh` solver, and to compute the mesh motion fluxes by calling the function `sweptVol()` that calculates the volumes swept by the cell faces during the mesh movement.

2.5.3.1 Laplace Equation

The Laplace equation with constant or variable diffusion field ξ is

$$\nabla \cdot (\xi \nabla \mathbf{u}) = \mathbf{0}. \quad (2.85)$$

The solution of Laplace equation is a motion function \mathbf{u} which is continuous, smooth, regular and gives non-overlapping streamlines and hence it passes the mesh validity constraints. When Laplace equation governs the mesh motion, the prescribed boundary deformation is not uniformly distributed through the domain. This potentially leads to local deterioration in mesh quality, because the movement of points close to the moving boundary is greater than for the other points. Fixing this problem can be achieved by prescribing variable diffusivity ξ in the Laplacian. In the above equation, \mathbf{u} may represent either the displacement or velocity of a point. On one hand, by using `displacementLaplacian` solver, the equations of cell motion are solved based on the Laplacian of the diffusivity and the cell displacement; this solver should be read from the file `dynamicMeshDict` in the `constant` folder of the case and an extra file named `pointDisplacement` in the starting time folder should be available. The result of the Eq. (2.85) is transferred to the mesh motion solver to update all mesh points new position as $\mathbf{x}^{new} = \mathbf{u} + \mathbf{x}^{old}$. Thus every mesh point is moved based on its calculated displacement.

On the other hand, by using `velocityLaplacian` solver, the equations of cell motion are solved based on the Laplacian of the diffusivity and the cell motion velocity; this solver should be read again from the file `dynamicMeshDict` and an extra file named `pointMotionU` in the starting time folder should be available, which determines the velocity at which each single boundary is moving. The result of the Eq. (2.85) is transferred to the mesh motion solver to update all mesh points new position as $\mathbf{x}^{new} = \mathbf{u}\delta t + \mathbf{x}^{old}$. Thus every mesh point is moved based on its calculated velocity. In this thesis, the cell velocity solver is used since it is giving better results than the approach using a solver based on cell displacement as shown by Al-Hababbeh [40].

2.5.3.2 Solid Body Rotation Stress Equation

The second method to deform the mesh is based on the linear elasticity equation and is called the solid body rotation stress (SBR Stress) equation. The equation of linear elasticity, valid for small displacements, may be written as

$$\nabla \cdot \sigma = \mathbf{0}, \quad (2.86)$$

where σ is the stress tensor given in terms of the strain tensor ϵ by the constitutive relation

$$\sigma = 2\mu\epsilon + \lambda tr(\epsilon)\mathbf{I}, \quad (2.87)$$

in which tr indicates the trace and λ and μ are the Lamé's constants [117] related to the Young's modulus of elasticity E and Poisson ratio ν as

$$\mu = \frac{E}{2(1 + \nu)}, \quad (2.88)$$

and

$$\lambda = \begin{cases} \frac{\nu E}{(1 + \nu)(1 - \nu)} & \text{for plane stress,} \\ \frac{\nu E}{(1 + \nu)(1 - 2\nu)} & \text{for plane strain and 3-D.} \end{cases} \quad (2.89a)$$

$$(2.89b)$$

$E > 0$ may be thought of as the stiffness of the material, where large E indicates rigidity. Poisson's ratio is a measure of how much the material shrinks in the lateral direction as it extends in the axial direction; for physical materials $-1 < \nu < 0.5$. The following linear strain consecutive equation, also called the linear kinematic law, quantifies the change in length and orientation of a material fiber in the elastic body:

$$\epsilon = \frac{1}{2} \left[\nabla \mathbf{u} + (\nabla \mathbf{u})^T \right], \quad (2.90)$$

where \mathbf{u} is the position of an internal mesh point, which is treated as if it was a linear solid. For convenience, in the plane strain regime consider the matrix

$$R = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}$$

that rotates a given vector \mathbf{x} by a clockwise angle θ in a fixed coordinate system (refer to Fig. 2.7). Then the displacement vector \mathbf{u} and linear strain tensor in Eq. (2.90) are given by:

$$\mathbf{u} = \mathbf{R} \cdot \mathbf{x} - \mathbf{x} \quad \text{or simply:} \quad \mathbf{u} = (\mathbf{R} - \mathbf{I}) \cdot \mathbf{x}, \quad (2.91)$$

$$\epsilon = \mathbf{R} - \mathbf{I}. \quad (2.92)$$

Equation (2.90) does not allow for rotation because it gives a non-zero strain for a rotation (also this can be seen in Eq. (2.92)). To handle this, two approaches have been proposed by Dwight [118] to request that the deformation equations admit rigid body motions of the mesh, that is $\sigma = 0$ is sufficient for rotations. The first approach is done by substituting Eq. (2.92) into Eq. (2.87) to get:

$$\sigma = (\lambda + \mu) [2 (\cos(\theta) - 1)] \mathbf{I}, \quad (2.93)$$

which may be set to zero by choosing $\lambda + \mu = 0$. This is achieved by replacing the expressions in Eqs (2.88) and (2.89b) by $\lambda = -E$ and $\mu = E$. The same effect can be obtained by setting the Poisson ratio ν to a very large value, which emphasizes that the equations can no longer be thought of as a model of elasticity. In this approach, the rigid body is allowed to move and is still linear. Also its computational cost is

similar to the computational cost needed to solve the Laplace equation.

In the second approach, an improvement for Eq. (2.90) is done by adding an extra non-linear term to obtain the strain relation (also called Lagrangian strain tensor):

$$\epsilon = \frac{1}{2} \left[\nabla \mathbf{u} + (\nabla \mathbf{u})^T + \nabla \mathbf{u} \cdot (\nabla \mathbf{u})^T \right]. \quad (2.94)$$

Although this approach raising the computational cost of the method, it has been proven numerically [118] that not only are rigid body rotations admitted, but that the scheme is much more robust to other deformations.

Finally, as with the Laplace equation the solid body rotation stress mesh motion equation uses the diffusivity ξ , acting as a stiffness, to improve the quality of the mesh. Therefore the final form of Eq. (2.86) is achieved by prescribing variable diffusivity ξ in the Laplacian. The result of this final form \mathbf{u} is transferred to the mesh motion solver to update all mesh points new position as: $\mathbf{x}^{new} = \mathbf{u} + \mathbf{x}^{old}$.

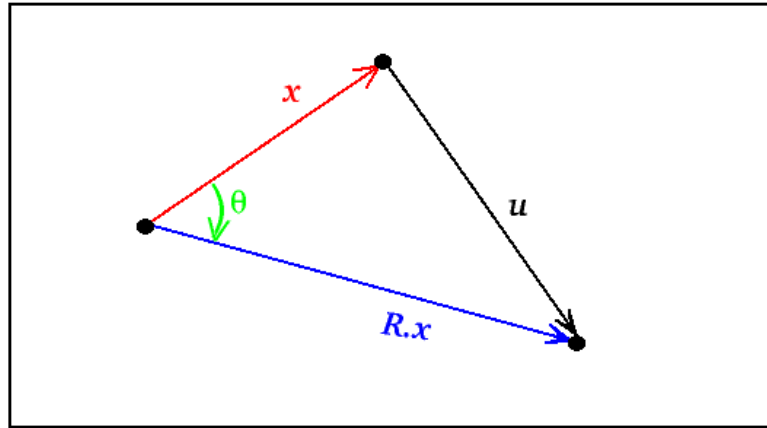


Figure 2.7: A clockwise rotation of a vector through angle θ

2.5.3.3 Radial Basis Function Interpolation

Mesh motion solver based on the Laplace or SBR equation maintains high mesh quality for problem with limited boundary rotations. To handle this, Bos [113] developed a new mesh motion solver based on the RBF interpolation for large rotations. This new mesh motion technique does not need any information about the mesh connectivity and can be applied to arbitrary unstructured meshes containing polyhedral cells, the way OpenFOAM deals with the finite volume implementation.

Suppose a set of pairwise distinct points $S = \{\mathbf{x}_{b_1}, \dots, \mathbf{x}_{b_N}\} \subseteq \mathbb{R}^d$ in the d -Euclidean space is given. These points are normally called 'centers'. Suppose further, we know values f_1, \dots, f_N at the centers and we are searching for a continuous function that interpolates these values at the centers. Then the radial basis function interpolant of the following form can be used:

$$s(\mathbf{x}) = \sum_{j=1}^N \alpha_j \Phi(\|\mathbf{x} - \mathbf{x}_{b_j}\|) + q(\mathbf{x}). \quad (2.95)$$

In Eq. (2.95) $s(\mathbf{x})$ is a interpolation function describing the displacement of all computational mesh points, $\mathbf{x}_{b_j} = (x_{1_{b_j}}, \dots, x_{d_{b_j}})^T$ is known boundary (data) point, N is the number of boundary points, Φ is a given basis function which depends on the Euclidean distance between the target point \mathbf{x} and the data point \mathbf{x}_{b_j} , and q is a polynomial whose minimal degree depends on the choice of Φ . Here f_j contains the

known discrete values of the boundary point displacements.

The coefficients α_j , and the polynomial q are determined by the interpolation conditions:

$$s(\mathbf{x}_{b_j}) = f_j, \quad 1 \leq j \leq N, \quad (2.96)$$

and the additional requirements

$$\sum_{j=1}^N \alpha_j p(\mathbf{x}_{b_j}) = 0, \quad (2.97)$$

for all polynomials p with degree less than or equal than that of q . The interpolation function is unique if Φ is conditionally positive-definite function as is shown in the below theorem whose proof can be found in [119] and [120].

Definition 2.1. *A real-valued continuous function $\Phi : \mathbb{R}^d \longrightarrow \mathbb{R}$ is said to be conditionally positive definite of order m if for any set of pairwise distinct centers $S = \{\mathbf{x}_{b_1}, \dots, \mathbf{x}_{b_N}\} \subseteq \mathbb{R}^d$ and $\alpha = (\alpha_1, \dots, \alpha_N)^T \subseteq \mathbb{R}^N$ satisfying Eq. (2.97) for any real-valued polynomial p of degree at most $m - 1$, the quadratic form*

$$\sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j \Phi(\mathbf{x}_{b_i} - \mathbf{x}_{b_j}) \text{ is non-negative.}$$

Theorem 2.1. *Suppose Φ is conditionally positive definite of order m . Suppose further that the set of centers $S = \{\mathbf{x}_{b_1}, \dots, \mathbf{x}_{b_N}\} \subseteq \mathbb{R}^d$ has the property that the zero*

polynomial is the only polynomial of degree less than m that vanishes on it completely.

Then there exists exactly one function s of the form 2.95 that satisfies both 2.96 and 2.97.

If the basis functions are conditionally positive definite of order $m \leq 2$, a linear polynomial for $q(\mathbf{x})$ can be used [121]. The values for the coefficients α_j and the linear polynomial can be obtained by solving the system:

$$\begin{bmatrix} s(\mathbf{x}_b) \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} f \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \Phi_{bb} & \mathbf{Q}_b \\ \mathbf{Q}_b^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \quad (2.98)$$

where α is containing all coefficients α_j , β is containing all coefficients of the linear polynomial $q(\mathbf{x})$, Φ_{bb} is $(N \times N)$ matrix with general term $[\Phi_{bb}] = [\Phi_{(i,j)}] = [\Phi(\|\mathbf{x}_{b_i} - \mathbf{x}_{b_j}\|)]$ and \mathbf{Q}_b is $(N \times (d+1))$ matrix with row j given by $[1 \ \mathbf{x}_{b_j}]$.

We can solve (2.98) directly (because (2.98) leads to a dense matrix system, which is difficult to solve using standard iterative techniques) using QR-decomposition or LU-decomposition to find the coefficient arrays α and β . These coefficients are used to calculate the values for the displacements of all internal mesh points g using the evaluation function 2.95 as:

$$g_i = s(\mathbf{x}_i) = \sum_{j=1}^N \alpha_j \Phi(\|\mathbf{x}_i - \mathbf{x}_{b_j}\|) + q(\mathbf{x}_i). \quad (2.99)$$

The result of the above equation is transferred to the mesh motion solver to update

all internal mesh points new position. Thus every internal mesh point is moved based on its calculated displacement, such that no connectivity is necessary.

The size of the system 2.98 is $((N + (d + 1)) \times (N + (d + 1)))$ which is considerably smaller than other techniques using mesh connectivity such as Laplace or SBR methods. The mesh connectivity techniques encounter system of size $(N_{int} \times N_{int})$ where N_{int} is the total number of internal mesh points, which is a dimension higher than the total number of boundary points. In contrast to the Laplace and SBR methods, no partial differential equations need to be solved and the evaluation of all internal boundary points is straightforward to implement in parallel, since no mesh connectivity is needed. Concerning the robustness, RBF interpolation method is not using the diffusion coefficient ξ . Instead, the basis function Φ need to be chosen to satisfy the mesh robustness. For more details about RBF, refer to [121] and [113].

Chapter 3

Fluid Transport Via Peristaltic Motion

The main objective of this study is to implement the power of the computational fluid dynamics (CFD) to design two computational models of geometry and motility of the intestines and the lower part of an idealized human stomach during emptying, and to use them to simulate the peristaltic motion for different Newtonian and non-Newtonian fluids. These simulations were performed in the fixed frame of reference with a modified solver from open source software package, OpenFOAM. Moreover, the finite volume method (FVM) is employed to solve the conservation equations of mass and momentum for velocity and pressure, and the Bird-Carreau Yasuda viscosity law is used to model the non-Newtonian fluid.

After investigating the convergence criteria and mesh resolution, a comparison to the experimental and theoretical data has been made to validate the numerical models and methods. In addition, a parameter study is performed to investigate the influence of various geometrical and rheological parameters on the material transport efficiency (TE), i.e. the effect of the traveling wave speed, the amount of deformation in terms of relative occlusion (RO), and the shear-thinning non-Newtonian fluid and the Newtonian fluid viscosity.

A parameter study has been performed by Al-Hababbeh [40] to determine the effect of the shear-thinning behavior, the wave speed and the gap width on the transport efficiency of Newtonian and non-Newtonian fluids in a 2-D channel of uniform width. The present work extends that of Al-Hababbeh in two ways. First, we develop a 2-D axisymmetric numerical model to get a realistic tubular peristaltic flow as encountered in the small intestine, and second, we examine the influence of the fluid viscosity variation on the transport efficiency (refer to Appendix D for more details).

From the fluid mechanics viewpoint, and to the best of our knowledge, no rigorous attempt has been made to develop a realistic model of the lower part of human stomach during the emptying process, that is when the pylorus is open. Based on this, our present work extends that of Pal et al. [68] in two ways. First, we develop a simple 2-D axisymmetric numerical model, which reduces the high level of complexity in full 3-D models, of the geometry and motility of the lower part of human stomach, to get a better understanding of the flow field that develops within the stomach during the

emptying process. Second, a parameter study is performed to investigate the effect of various geometrical and rheological parameters on the gastric emptying in terms of (average) transport efficiency.

3.1 Computational Models

3.1.1 Geometries

Two axisymmetric computational models have been developed to simulate the peristaltic motion for different fluids in the fixed (laboratory) frame of reference where the boundary motion is represented by a traveling wave which deforms the boundary and hence the mesh: a 2-D axisymmetric tubular model and a 2-D axisymmetric conical model.

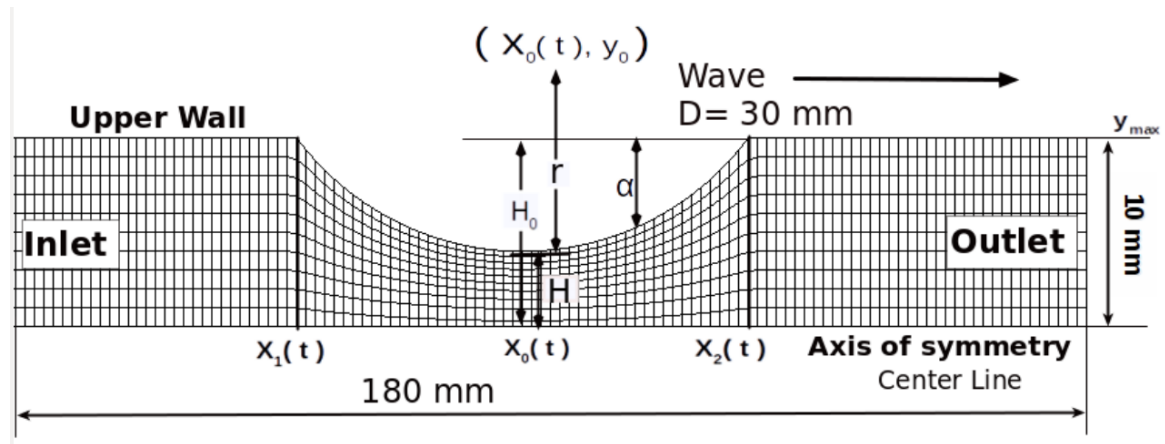


Figure 3.1: Computational domain for the axisymmetric tubular model equipped by the circular deformation and relative occlusion parameters.

The geometry of the first model is specified as a wedge of a small angle of 5° and one cell thick running along the axis of symmetry, straddling one of the coordinate planes. This model reflects a cross sectional of the upper half of a tube whose length and diameter are 180 mm and 20 mm, respectively. The deformation on the upper wall is represented by a circular wave with a diameter of 30 mm as shown in Fig. 3.1. This wave is moving with a uniform speed in the x-direction and it is generated by moving the mesh points of the upper wall up and down along the wedges. The geometry and speed of the wave was chosen to reflect that used in experimental studies of Nahar et al. [1, 39].

On the other hand, the axisymmetric conical model is designed to reflect an axisymmetric cross section of the lower part of an average sized human stomach as is shown in Fig. 3.2. This lower part can be considered as a frustum of a circular cone whose length is 150 mm and with diameters of 100 mm and 10 mm at its widest point and at the pyloric ring, respectively. The upper wall of this geometry inclines from the x-axis by angle of 16.7° and it is deformed by a circular antral wave of a diameter 20 mm. Note that, the center of this wave changes in both horizontal and vertical directions, as the wave propagates toward the pylorus sphincter. Moreover, this antral wave is moving with a uniform speed in the x-direction and it is generated by moving the mesh points of the upper wall along the wedges as discussed in the case of the axisymmetric tubular simulations.

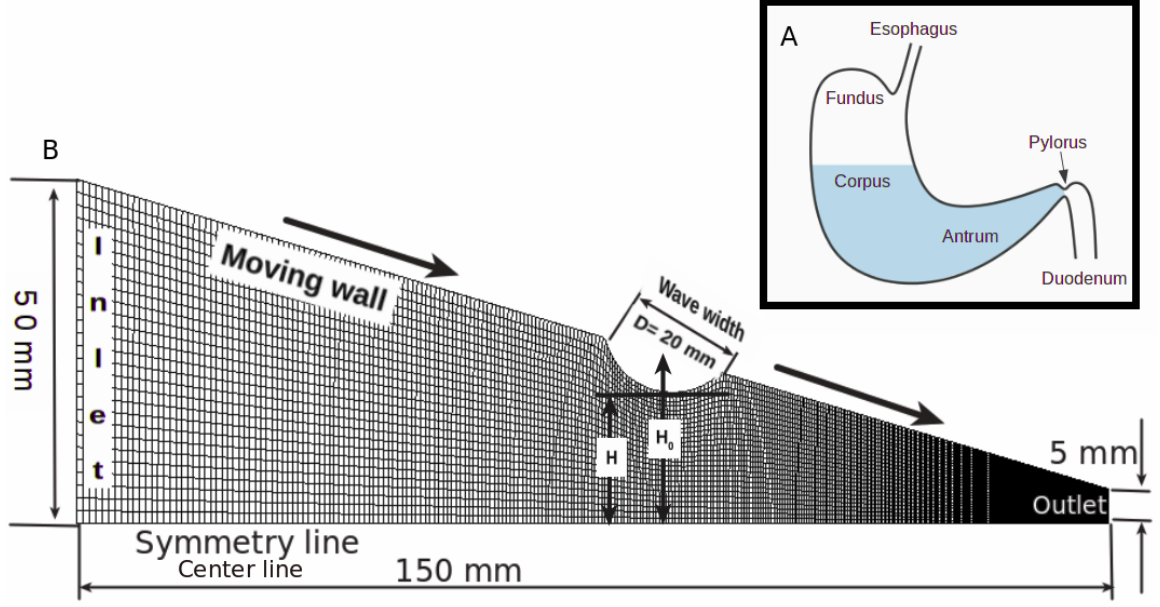


Figure 3.2: (A) Schematic diagram of a human stomach. (B) Computational domain for the axisymmetric conical model equipped by the circular deformation and relative occlusion parameters.

3.1.2 Governing Equations

The fluid is taken to be a single-phase fluid, and the flow is assumed to be incompressible, isothermal and inelastic. Based on these criteria, the motion of the fluid in an arbitrary control volume V bounded by a closed surface S is governed by the conservation laws for mass and momentum:

$$\frac{\partial}{\partial t} \int_V dV + \oint_S (\mathbf{u} - \mathbf{u}_b) \cdot \mathbf{n} dS = 0, \quad (3.1)$$

$$\rho \left(\frac{\partial}{\partial t} \int_V \mathbf{u} dV + \oint_S \mathbf{u} (\mathbf{u} - \mathbf{u}_b) \cdot \mathbf{n} dS \right) = \oint_S \boldsymbol{\tau} \cdot \mathbf{n} dS - \oint_S P \cdot \mathbf{n} dS, \quad (3.2)$$

where \mathbf{u} is the fluid velocity field, P is the pressure field, ρ is the density, \mathbf{n} is the outward pointing unit normal to the surface S , \mathbf{u}_b is the velocity of the surface S , and τ is the viscous stress tensor. Under the scope of this study, neither the wall roughness nor the friction and the gravity forces were considered in the fluid simulations. The constitutive equations for the Newtonian and non-Newtonian fluids are $\tau = \mu \mathbf{D}$ and $\tau = \eta(\dot{\gamma}) \mathbf{D}$, respectively, where μ is the dynamic viscosity, $\mathbf{D} = \nabla \mathbf{u} + \nabla \mathbf{u}^T$ is the rate-of-strain tensor, $\eta(\dot{\gamma})$ is the shear rate dependent viscosity and the scalar $\dot{\gamma}$ is the strain rate defined as the magnitude of the strain rate tensor as is shown in Eq. (2.10). The relationship between the rate of change of the volume V and the velocity \mathbf{u}_b is defined by Eq. (2.65), the so called geometric conservation law (GCL).

The initial and boundary conditions are as follows: The initial conditions for the internal pressure and velocity fields are set to zero. Symmetry boundary conditions for the pressure and the velocity fields have been used along the center line in the both models. Essentially, this kind of boundary conditions guarantee that there is no flow across the center line. On the inlet and outlet, the velocity boundary conditions are set to zero gradient while the pressure boundary conditions are set to the zero total pressure. This reflects the fact that the experimental system used by Nahar et al. [1, 39] was closed, i.e., the inlet and outlet boundaries were connected via a large fluid reservoir which is at constant pressure.

The total pressure P_0 is computed by

$$P_0 = P + \frac{1}{2}\rho|\mathbf{u}|^2, \quad (3.3)$$

where P stands for the static pressure and $\frac{1}{2}\rho|\mathbf{u}|^2$ expresses the dynamic pressure. This means that as long as the velocity field \mathbf{u} changes, the value of the pressure field P is adjusted by the prescribed value of the total pressure P_0 . On the upper wall, the normal gradient of the pressure is set to zero and the velocity is set to the velocity of wall in the normal direction of the wall.

The mesh motion is governed by the Laplace equation

$$\nabla \cdot (\xi \nabla \mathbf{w}) = \mathbf{0}, \quad (3.4)$$

where \mathbf{w} is the cell velocity in a given time step, and ξ is the preset variable of diffusivity that describes how points should be moved when solving the cell motion equation for each time step. For more details refer to González [122]. The movement of the boundary points is propagated into the interior points by diffusion. In the current simulations, the directional diffusivity field has been used in the Laplace equation to fix the local distortion in the mesh quality. The directional diffusivity defines the diffusion coefficients, which are used in the integral form of the conservation equations, for the three directions in space. After Eq. (3.4) is solved for the cell velocity, it is used to determine the velocity of the moving cell boundary \mathbf{u}_b that is used in

the integral form of the conservation equations. For more details refer to Jasak and Tukovic [83].

The boundary conditions for cell velocity are as follows. For both geometrical models, $\mathbf{w} = \mathbf{0}$ on the center line, and the gradient of \mathbf{w} is zero on the inlet boundary. On the outlet boundary, either $\mathbf{w} = \mathbf{0}$ or the gradient of \mathbf{w} is zero, depending on whether we allow the wave to intersect it. The velocity on the upper wall is determined by a prescribed mathematical formula that gives the position of the mesh points as a function of time. In this study, the traveling wave that deforms the upper wall is assumed to be single wave and circular in shape. The circular deformation is described by the following equation (refer to Fig. 3.1)

$$\alpha(x, t) = (y_{max} - x \tan(\theta)) - \left(y_0(t) - \sqrt{r^2 - (x - x_0(t))^2} \right), \quad (3.5)$$

where $x_1(t) \leq x \leq x_2(t)$, y_{max} is the maximum y-component of the points on the undeformed upper wall, $(x_0(t), y_0(t))$ is the center of the circle whose radius is r , $\theta = \arctan\left(\frac{y_{max}-y_{min}}{x_{max}-x_{min}}\right)$ is the inclination angle of the upper wall from the x-axis, y_{min} is the minimum y-component of the points on the undeformed upper wall, x_{max} is the maximum x-component of the points on the center line, and x_{min} is the minimum x-component of the points on the center line. The values of y_{max} , y_{min} , x_{max} and x_{min} are specified by the user in the `<case>/constant/polyMesh/blockMeshDict` file. The x-component of the center and the two ends of the circular arc move with a

uniform speed in the x-direction,

$$x_i(t) = (x_i + ct) \cos(\theta); \quad i = 0, 1, 2,$$

where x_i 's are the initial values at time $t = 0$ and c is the wave speed. The y-component of the center moves in the direction parallel to the undeformed upper wall which is described by the equation: $y = y_{max} - x \tan(\theta)$. Note that, $y_0(t) = y_0$ for all t when $\theta = 0$. The values of x_0 , y_0 and r are specified by the user, while x_1 and x_2 are computed so that $\alpha(x_i(t), t) = 0$; $i = 1, 2$. The above deformation is valid if $y_0(t) \geq y_{max}$ for all t . This circular deformation is generated by moving the mesh points of the upper wall vertically up or down in the y-direction along the boundary wedges, and this movement depends on the wave horizontal motion as described in Eq. (3.5). The parameters for this boundary condition are given in the `<case>/0/pointMotionU` file. These parameters are: `circleRadius` and `speed` to specify the radius and the speed of the wave, respectively, and `yCompFinalCenter` for y_0 . The occlusion diameter H is computed by subtracting `circleRadius` from the y-component of the center $y_0(t)$. In this file additional parameters have been added to control the motion: `numOfWaves` to specify the number of the waves, `period` to initiate a new wave every certain period of time, and l to specify the distance between the front of the wave and the outlet when the wave start to climb up by a specified angle of `beta` and with a speed of `alpha`.

In OpenFOAM, the corresponding initial and boundary conditions are given in the

initial time directory of the case. Table 3.1 shows the boundary conditions discussed above in OpenFOAM language. The condition `movingWallNormalVel` is a projection of an existing boundary condition of OpenFOAM, called `movingWallVelocity`, which corrects the flux due to the mesh motion so that the total flux across the patch is zero. The projection is made in the normal direction to the upper wall, that is, on the wave the relative velocity $\mathbf{u}_{\text{relative}} = \mathbf{u} - \mathbf{u}_b$ is zero.

Table 3.1
Boundary conditions in OpenFOAM for open outlet.

boundary	\mathbf{u}	P	\mathbf{w}
inlet	zeroGradient	zero totalPressure	zeroGradient
outlet	zeroGradient	zero totalPressure	zeroGradient or zero fixedValue
center line	empty	empty	zero fixedValue
upper wall	<code>movingWallNormalVel C.3</code>	zeroGradient	my dynamic mesh solver A.2.2 and A.3.2

3.1.3 Numerical Methods and Computational Details

OpenFOAM was used to simulate peristaltic motion in this study. To solve such transient flow field, a transient solver for an incompressible, laminar flows with a dynamic moving mesh capability from the OpenFOAM package, called `transientSimpleDyMFoam`, is used. Further, this solver can be used also for turbulent flows by activating the turbulence models. This solver is using a segregated SIMPLE-based pressure-velocity coupling algorithm in time-stepping mode, as discussed in Chapter 2.

The `fvSchemes` file in the system directory sets the numerical schemes for terms, such as derivatives in the conservation equations, that are calculated during a simulation. In our simulations, the spatial discretization of the convection-diffusion terms is achieved by a second order finite volume standard Gaussian method with a linear central differencing interpolating scheme. An implicit bounded first order Euler method is used to handle the temporal term.

The `fvSolution` file in the system directory was designed to handle the settings for the linear equation solvers and the algorithms to be used by a solver application. In our simulations, the discrete pressure equations were solved by means of the Generalized Geometric Multi-grid (GAMG) method with the Gauss-Seidel smoother. This smoother is used together with the solver for an asymmetric matrix system, called `smoothSolver`, to solve the discrete momentum velocity equations. The discrete cell motion equations were solved by using the Preconditioned Conjugate Gradient (PCG) method with the Diagonal-based Incomplete Cholesky (DIC) preconditioning for symmetric matrices. Keep in mind that preconditioner is needed for solvers that rely on a preconditioning strategy to speed up their iterative process, and smoother is designed to smooth-out numerical issues that usually arise from ill-formed matrices and strongly uneven intermediate solutions for the matrix equation. For more details on what preconditioning is, see <http://en.wikipedia.org/wiki/Preconditioner>. A good brief source for linear solvers, preconditioners and smoothers in OpenFOAM is given by Behrens [123]. The number of velocity-pressure iterates in each time step

was set to 25 and 50 for the tubular and conical geometries, respectively. This number can be controlled by the keyword `nOuterCorrectors` in `<case>/system/fvSolution`. The absolute tolerances for the linear solvers within each pressure-velocity iterate were set to 10^{-10} and the relative tolerance for all variables was set to zero to force the solutions to the system of equations to absolute tolerances at each time step, which is recommended when using the PISO algorithm, refer to [124] for more details. Initially, a small time step size is assigned and then it is adapted through the motion such that the maximum Courant number (defined by Eq. (2.58)) less than or equal 0.5.

The relative occlusion of the wave (the occlusion diameter to the tube diameter or to the antral diameter, without the wave) is defined by $RO = \left(1 - \frac{H}{H_0}\right) \times 100$ (refer to Figs 3.1 and 3.2). The simulations are carried out for five different Newtonian fluids, where the higher viscous fluid is obtained by increasing the dynamic viscosity of the lower one by a factor of 10. The fluid parameters used in this study are summarized in Table 3.2.

Table 3.2
Newtonian fluids parameters.

Newtonian fluid	Dynamic viscosity (Pa s)	Density (kg/m ³)
N1 (Water)	0.001	1000
N2	0.01	1000
N3	0.1452	1000
N4	1	1360
N5 (Honey)	10	1360

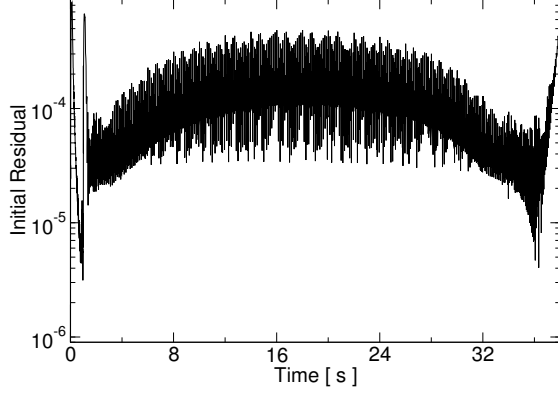
For the axisymmetric tubular simulations, three different relative occlusions of 20%, 60% and 80% together with three different wave speeds of 2.5 mm/s, 5 mm/s and 10 mm/s have been used. For the axisymmetric conical simulations, three wave speeds of 1.15 mm/s, 2.3 mm/s and 4.6 mm/s have been used with three different maximum relative occlusions of 21%, 66% and 80%. These values were chosen to reflect experimental conditions, physical conditions, and to allow parameter study. The above flow cases are summarized in Table 3.3.

Table 3.3
Relative occlusion and wave speed values used in the axisymmetric tubular and conical models.

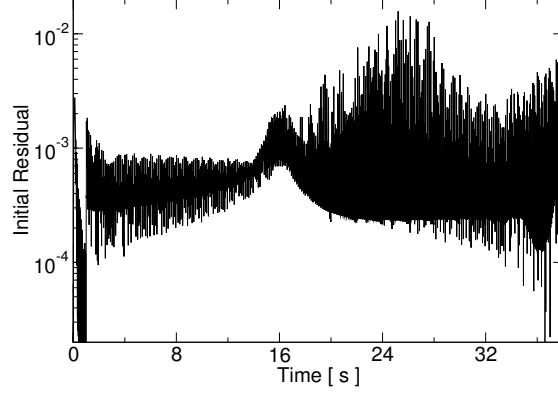
	Axisymmetric tubular model			Axisymmetric conical model		
RO (%)	20	60	80	21	66	80
Wave speed (mm/s)	2.5	5	10	1.15	2.3	4.6

In the standard cases, as is discussed in more details below, a wave speed of 5 mm/s and a relative occlusion of 60% are used in the tubular model simulations whereas a wave speed of 2.3 mm/s and a relative occlusion of 66% are used in the conical model simulations.

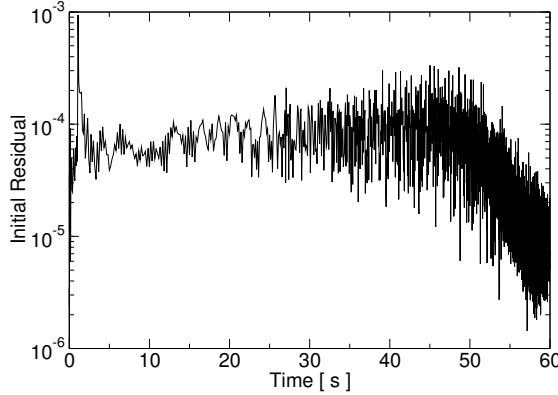
A convergence study is carried out for the Newtonian fluid N3 for the standard simulation cases. Convergence is evaluated by computing the initial residuals, that is, the residuals for the discrete pressure and velocity equations at the beginning of the last PISO iteration in each time step. The initial residuals for the discrete x-momentum and pressure equations are shown in Fig. 3.3 as a function of time.



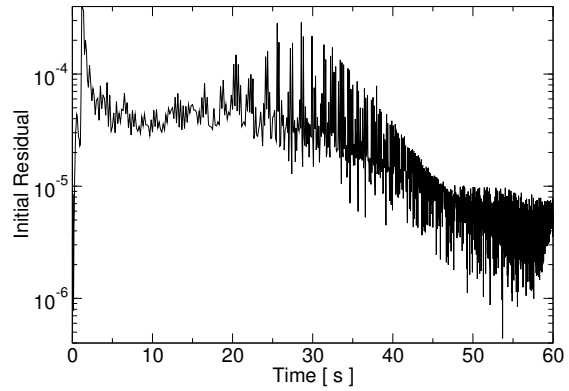
(a) Residual of the discrete x-momentum equation at the beginning of the last (i.e. 25th) PISO iteration in each time step for the axisymmetric tubular model.



(b) Residual of the discrete pressure equation at the beginning of the last (i.e. 25th) PISO iteration in each time step for the axisymmetric tubular model.



(c) Residual of the discrete x-momentum equation at the beginning of the last (i.e. 50th) PISO iteration in each time step for the axisymmetric conical model.



(d) Residual of the discrete pressure equation at the beginning of the last (i.e. 50th) PISO iteration in each time step for the axisymmetric conical model.

Figure 3.3: Convergence study

This figure shows that the residuals oscillating about the mean level which is acceptable for an unsteady transient case like this. The initial residuals were approximately 10^{-4} or less which is sufficient for convergence of the velocity-pressure iterates in each time step.

3.1.4 Mesh Independence Study

To make sure that the computational mesh exhibits a sufficient mesh resolution, a mesh dependence study has been performed for the Newtonian fluid N3 in the case of fastest wave speed and largest relative occlusion. This is presumably the worst case scenario where the largest velocity gradients occurs. Tables 3.4 and 3.5 summarize the undeformed meshes that were used in the tubular and conical simulations, where the finer mesh is obtained from the coarser one by increasing the number of cells in the x and y-directions by a factor of 1.5. As the wave travels across the boundary, and the mesh is deformed, the height of the cells decreases. These computational meshes consist of one structured hexahedral Cartesian block with a uniform cell distribution for the case of the tubular simulations while a non-uniform cell distribution has been used in the case of the conical simulations.

Table 3.4
Computational mesh details for the axisymmetric tubular simulations.

Mesh	Number of cells	Total number of cells	Cell length Δx (mm)	Cell height Δy (mm)
M1	$360 \times 10 \times 1$	3600	0.5	1
M2	$540 \times 15 \times 1$	8100	0.333	0.667
M3	$810 \times 23 \times 1$	18630	0.222	0.435

Table 3.5
Computational mesh details for the axisymmetric conical simulations.

Mesh	Number of cells	Total number of cells	Cell length Δx (mm)	Cell height Δy (mm) near the pylorus
M3	$270 \times 18 \times 1$	4860	0.056	0.278
M4	$405 \times 27 \times 1$	10935	0.037	0.185
M5	$608 \times 41 \times 1$	24928	0.025	0.122

The results of mesh dependence study along the center line and near the outlet are shown in Figs 3.4 and 3.5, in which the left and right vertical dashed lines represent the ends of the wave. Figures 3.4a and 3.5a show that the minimum velocity occurs under the wave and the negative values of x-component of velocity indicate the presence of a back-flow.

Moreover, Fig. 3.4a shows that the values of x-component of velocity obtained from mesh M2 are closer to the ones obtained from mesh M3 than the ones obtained from mesh M1. On the other hand, Figs 3.4b, 3.5a and 3.5b show that different meshes give almost an identical values for the x-component of velocity. Therefore, the meshes M2 and M4 are used as the standard computational meshes for the tubular and conical simulations, respectively.

The undeformed standard tubular mesh had uniform cells of length 0.333 mm and height 0.667 mm. After deformation, the cell height under the wave was reduced to 0.133 mm for the largest relative occlusion of 80%. The undeformed conical standard mesh had the smallest cells located near the pylorus with length and height of 0.037 mm and 0.185 mm, respectively.

After deformation, the cell height under the wave was reduced to 0.074 mm for the largest maximum relative occlusion of 80%.

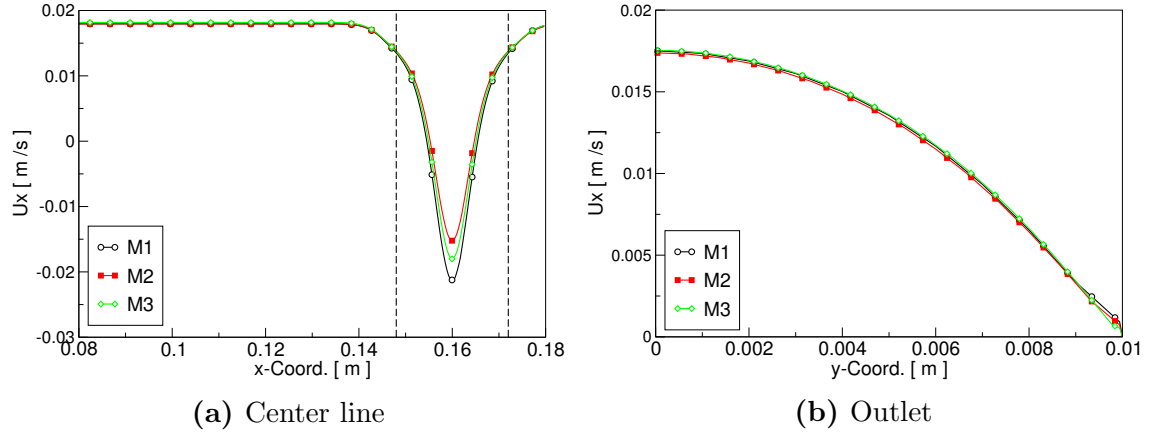


Figure 3.4: Mesh dependence study for the x-component of the velocity of the Newtonian fluid N3 in the axisymmetric tubular model at $t = 16$ s. The wave speed and the relative occlusion are 10 mm/s and 80%, respectively.

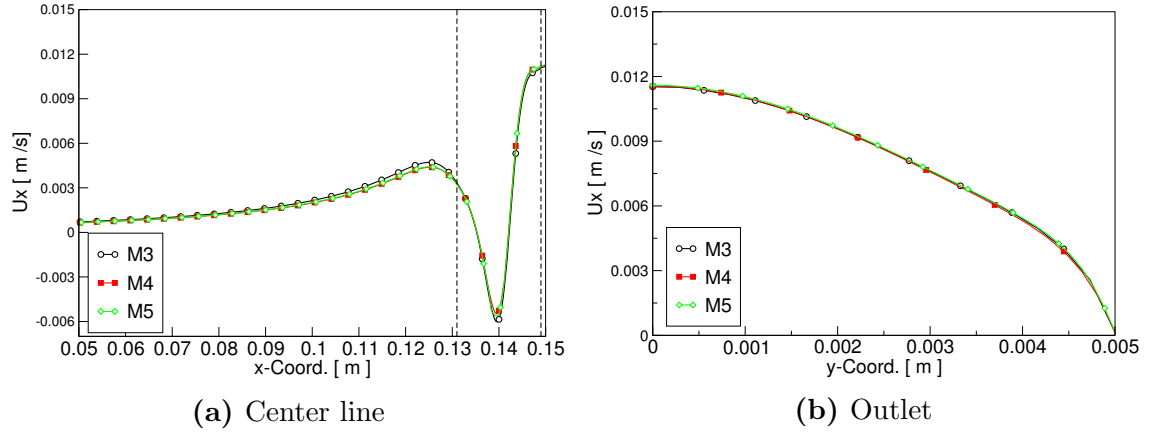


Figure 3.5: Mesh dependence study for the x-component of the velocity of the Newtonian fluid N3 in the axisymmetric conical model at $t = 30$ s. The wave speed and the relative occlusion are 4.6 mm/s and 80%, respectively.

For the case of the axisymmetric tubular simulations, fluid viscosity and relative occlusion influence the flow field values dramatically and subsequently the mesh dependence results will be affected accordingly. It has been shown that the meshes M2 and M3 are plausible computational tubular meshes for all fluids for the case of the relative occlusions of 60% and 20%, respectively, whereas, for the case of the relative occlusion of 80% the mesh M3 is used for the lower viscous fluids, i.e. N1 and N2, while the mesh M2 is used for the higher viscous ones, i.e. N3-N5.

3.1.5 Solver Validation

In this section, the solver is validated with experiments of Nahar et al. [1, 2, 125], in which the peristaltic motion is induced by means of rollers which squeeze a fluid along a flexible closed tube. Where three pairs of rollers are used to induce the peristaltic flow by moving the rollers from left to right.

The length of the tube is 320 mm and the diameter of the undeformed tube, before it is squeezed between the rollers, is 20 mm. After the rollers are applied, the tube expands along the lengths of the rollers and the diameter between consecutive pairs of rollers decreases to 11 mm. The gap width between a pair of rollers is 4 mm.

The fluid used in this experiment is a shear-thinning non-Newtonian carboxymethyl-cellulose aqueous solution at 1.5% w/w with 0.1 M NaCl and $M_w = 2.5 \times 10^5$ g/mol (CMC 1.5%). According to Stranzinger [126], the CMC 1.5% solution is inelastic for

concentrations up to 2%. The rheological measurements of this solution were carried out using a Physica rheometer (MCR 300, CC27), as is documented in Nahar et al. [2, 39, 125]. The fluid density of 1000 kg/m^3 . The measured shear rate dependent viscosity showed a shear-thinning behavior and it is approximated by the Bird-Carreau Eq. (2.17), where $\eta_0 = 0.1452 \text{ Pa s}$, $\eta_\infty = 0 \text{ Pa s}$, $k = 0.02673 \text{ s}$ and $n = 0.7588$. The velocity was measured using the pulsed ultrasound Doppler velocimetry (UVP) technique of Takeda [127].

Since the rollers expand the tube in the third direction, the geometry is no longer axisymmetric. Due to the amount of expansion in the third direction, we take a 2-D planar slice in the center of the tube as the computational domain. Therefore, the 2-D planar model of Al-Hababbeh [40] has been used.

The computational domain that is used to simulate the peristaltic motion reflects the upper half of a deformed channel with length of 180 mm and with height of 5.5 mm. The tubular mesh M3 in Table 3.4 is used as a computational mesh for the case of the planar tubular simulations and consists of one structured hexahedral Cartesian block with a uniform cell distribution. Primarily, the computational planar mesh has 18630 cells with cell length of 0.222 mm and cell height of 0.239 mm. The roller motion is represented by a circular wave, as described in Eq. (3.5), on the upper wall with diameter of 30 mm. The relative occlusion of the wave is approximately of 64% and the contact curve between the roller and the tube is a circular arc with a segment length of 24 mm.

In the deformed mesh, the smallest cells are located under the wave and the cell height is reduced to 0.087 mm. The initial and boundary conditions that reflect experimental setup were similar to those used in the axisymmetric tubular simulations. Moreover, the flow solver, the mesh motion solvers, the linear solvers and the discretization schemes are similar to the ones used in the axisymmetric tubular simulations, with the number of velocity-pressure iterates in each time step set to 20. Confirmation that a planar tubular model equipped with this dynamic physical solver is valid for the peristaltic simulations is presented in Fig. 3.6.

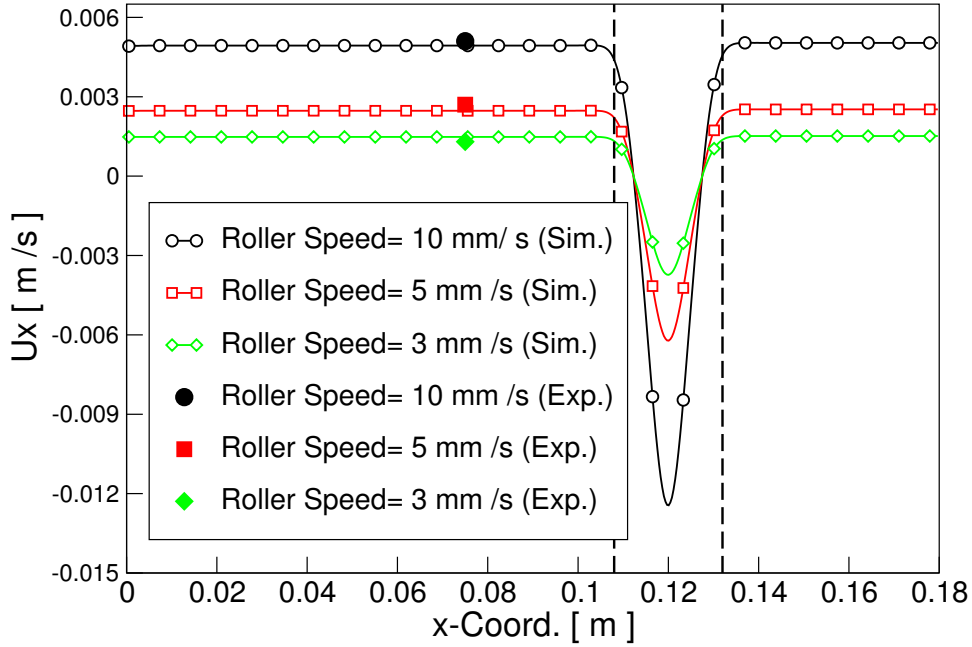


Figure 3.6: The x-component of the velocity of the non-Newtonian fluid along the centerline for the planar model. (The dots correspond to experiments [1, 2], the solid curves denote the simulations.)

It can be seen from this figure that the simulation results show excellent agreement with the experimental data. Note that only one data point for each roller speed can be used in this comparison, because the line of measurements for the UVP intersects our 2-D computational domain at only one point.

3.2 Results and Discussion

Two sets of simulations have been carried out to examine the influence of several rheological and geometrical parameters on the transport efficiency. In particular, we study transport efficiency in terms of a shear-thinning non-Newtonian fluid, Newtonian fluid viscosity, wave speed and relative occlusion. Since the fluid under consideration is incompressible, the mass transport is expressed in terms of the average speed at the outlet. Therefore, the transport efficiency can be computed by

$$\text{TE} = \frac{\text{average outlet speed}}{\text{wave speed}} \quad (3.6)$$

The first set of simulations is the axisymmetric tubular simulations with wave speeds of 2.5 mm/s, 5 mm/s and 10 mm/s and with relative occlusions of 20%, 60% and 80%. The second set represents the axisymmetric conical simulations with wave speeds of 1.15 mm/s, 2.3 mm/s and 4.6 mm/s and with maximum relative occlusions of 21%, 66% and 80%. (refer to Table 3.3).

3.2.1 Variation of the Newtonian Fluid

Due to the laminar behavior of the fluid flow and the proximity of the walls, the flow field that develops within the dynamic system can be significantly affected by the rheological properties of the fluid flow, in particular by its viscosity. To examine the influence of the fluid viscosity on the transport efficiency, several simulations have been performed for five different Newtonian fluids whose parameters are listed in Table 3.2.

The Reynolds number is defined by

$$\text{Re} = \frac{\rho \mathbf{u}_0 L_0}{\eta^0}, \quad (3.7)$$

where L_0 is the characteristic length scale of geometry, \mathbf{u}_0 is the characteristic velocity and η^0 is the characteristic viscosity. For our current simulations, the parameters of the Reynolds number are defined by setting $L_0 = H$, $\mathbf{u}_0 = (\text{wave speed}) \times \left(\frac{H_0}{H}\right)^2$ and $\eta^0 = \text{dynamic viscosity}$. Consult Figs 3.1, 3.2 and Table 3.2 for details.

Under these considerations, the values of the Reynolds number are computed and then listed in Tables 3.6 and 3.7 for the case of the tubular and conical simulations, respectively.

Table 3.6

Reynolds number at $t = 32$ s for five Newtonian fluids in the axisymmetric tubular model. The wave speed is 5 mm/s and three relative occlusions of 20%, 60% and 80% are used.

RO (%)	N1	N2	N3	N4	N5
20	62.5	6.25	0.430	0.085	0.0085
60	125	12.5	0.861	0.17	0.017
80	250	25	1.722	0.34	0.034

Table 3.7

Reynolds number at $t = 60$ s for five Newtonian fluids in the axisymmetric conical model. The wave speed is 2.3 mm/s and three relative occlusions of 21%, 66% and 80% are used.

RO (%)	N1	N2	N3	N4	N5
21	21.4	2.1	0.15	0.03	0.003
66	51.6	5.2	0.36	0.07	0.007
80	88.4	8.8	0.61	0.12	0.012

3.2.1.1 Axisymmetric Tubular Simulations

The transport efficiency results of these simulations are given in Fig. 3.7 for the case of wave speed of 5 mm/s. This figure shows that the transport efficiency is independent of the fluid viscosity and increases with relative occlusion.

The color plot of the velocity vectors for one Newtonian fluid N3 at time $t = 32$ s in the case of the standard tubular simulations is shown in Fig. 3.8 at the time, the wave is near the outlet boundaries. This figure shows that the maximum values of the velocity magnitude are attained in the region near the center line and just under the wave.

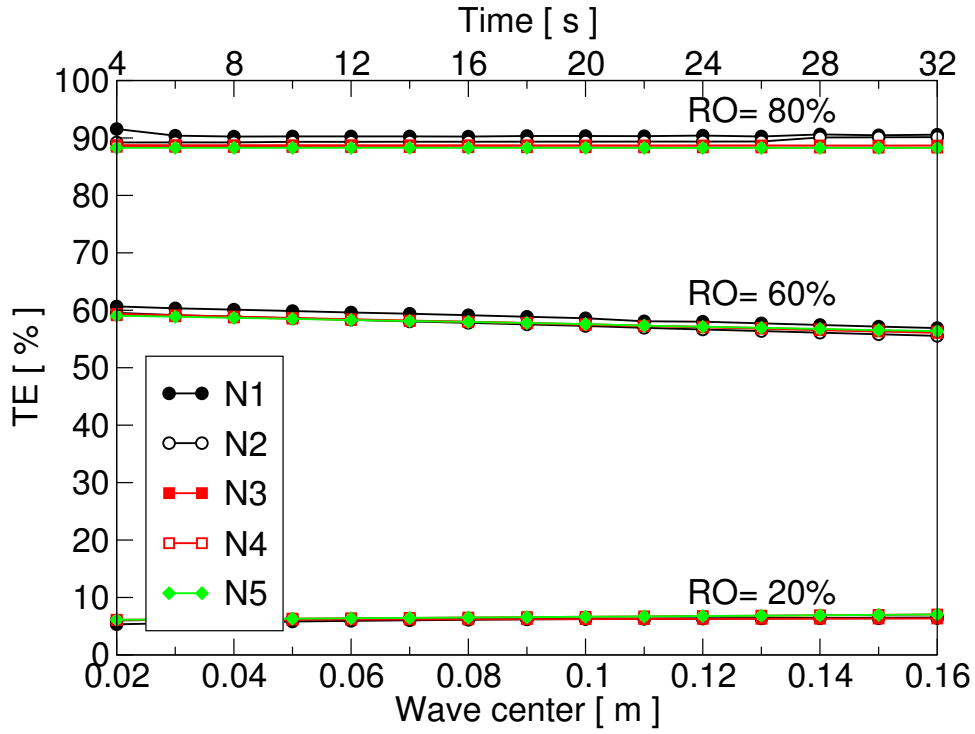


Figure 3.7: Transport efficiency for five Newtonian fluids in the axisymmetric tubular model. The wave speed is 5 mm/s with three relative occlusions of 20%, 60% and 80% are used.

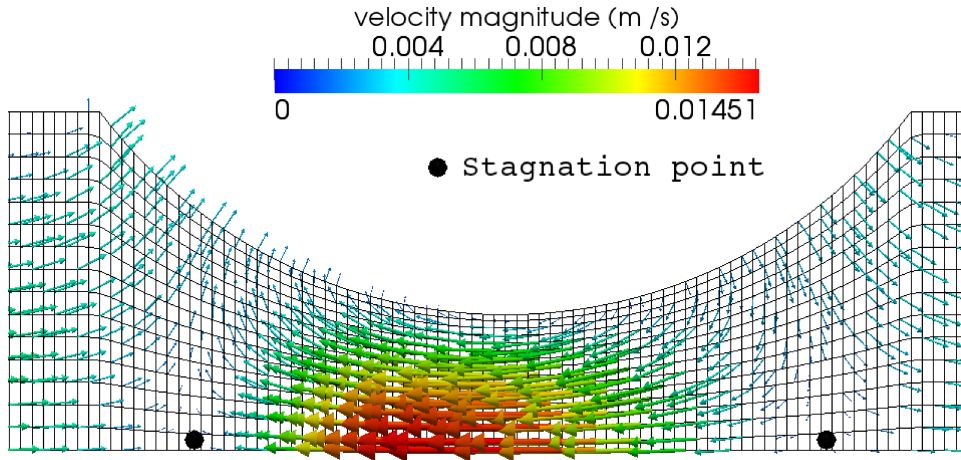


Figure 3.8: The velocity vectors of the Newtonian fluid N3 at $t = 32$ s in the axisymmetric tubular model. The wave speed and the relative occlusion are 5 mm/s and 60%, respectively.

These maximum values are nearly about of three times than that of the wave speed. In addition, the direction of the vectors in this figure indicates the presence of a back-flow. The x-component of velocity for five different Newtonian fluids along the center line and near the outlet at time $t = 32$ s is shown in Fig. 3.9. This figure shows that the values of the velocity are almost identical, except for the lowest viscous fluid N1. This is consistent with the transport efficiency results given in Fig. 3.7. Figure 3.9a shows that all fluids exhibit a back-flow, reaching the minimum of 0.01 m/s in magnitude for the case of the lowest viscous fluids in a region that is under the wave and along the center line, while the higher viscous fluids have almost the same magnitude of back-flow.

Note that the transport efficiency decreases slightly over the domain in the case of relative occlusion equals 60%, as is observed in Fig. 3.7. This decrease is due to small differences in the computed average velocity over the outlet boundary as is shown in Fig. 3.10.

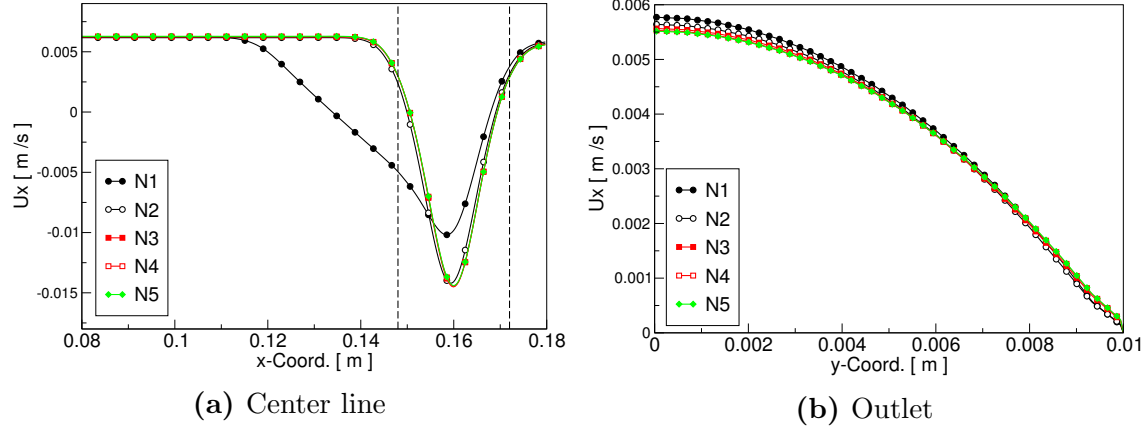


Figure 3.9: The x-component of the velocity of five Newtonian fluids in the axisymmetric tubular model at $t = 32$ s. The wave speed and the relative occlusion are 5 mm/s and 60%, respectively.

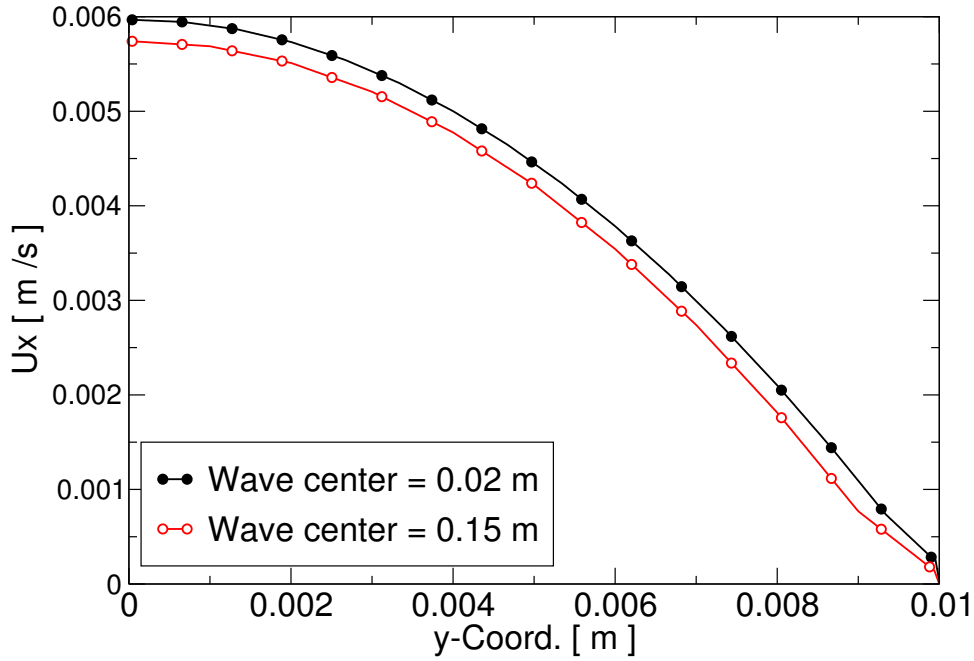


Figure 3.10: The x-component of the velocity of the Newtonian fluid N3 near the outlet in the axisymmetric tubular model at two varying wave positions. The wave speed and the relative occlusion are 5 mm/s and 60%, respectively.

3.2.1.2 Axisymmetric Conical Simulations

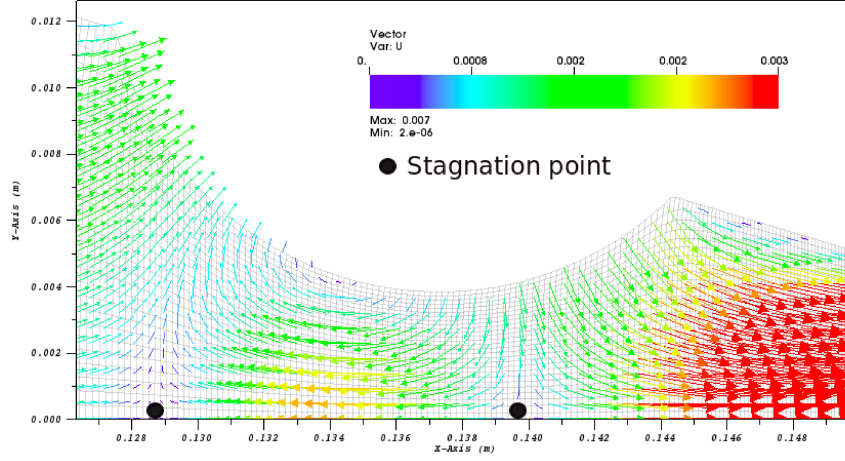
The results of the standard conical simulations for one Newtonian fluid N3 at different times are shown in Figs 3.11 and 3.12. The first figure exhibits color plots of the velocity vectors at the times $t = 58$ s (where the maximum magnitude of velocity is achieved) and $t = 60$ s (where the maximum relative occlusion is achieved), while the second figure shows the x-component of velocity along the center line and near the outlet at different positions.

Unlike in the tubular simulations (Fig. 3.8) in which the maximum velocity magnitudes occurred directly underneath the wave, Fig. 3.11 shows that the maximum velocity magnitudes in the conical simulations are in front of the wave. This is due to the decreasing diameter in the conical geometry.

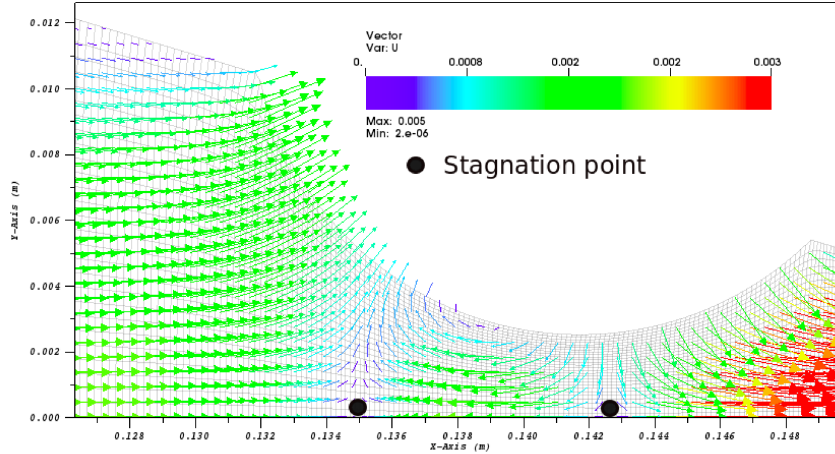
The circulations pattern is characterized by the direction of velocity vectors as is shown in Fig. 3.11. The direction of the vectors in this figure indicates the presence of a back-flow and this is consistent with the negative values of the x-component of velocity shown in Fig. 3.12a.

This back-flow reaches its maximum magnitude at time $t \approx 53$ s, in the region under the wave and along the center line, and then decreases gradually in magnitude as the wave propagates toward the pylorus sphincter. Figure 3.11a shows that as the wave approaches the pylorus sphincter, the increasing occlusion of the wave strengthens the x-component of velocity, reaching the maximum magnitude of three times than

that of the wave speed in the most occluded section of the pylorus canal, that is along the center line and near the outlet at time $t = 58$ s.



(a) $t = 58$ s



(b) $t = 60$ s

Figure 3.11: The velocity vectors (m/s) of the Newtonian fluid N3 in the axisymmetric conical model. The wave speed is 2.3 mm/s and the maximum relative occlusion is 66%.

Due to the disturbance between the back-flow, determined by the two moving stagnation points on the center line (see Fig. 3.11), and the outlet boundary, the effect of

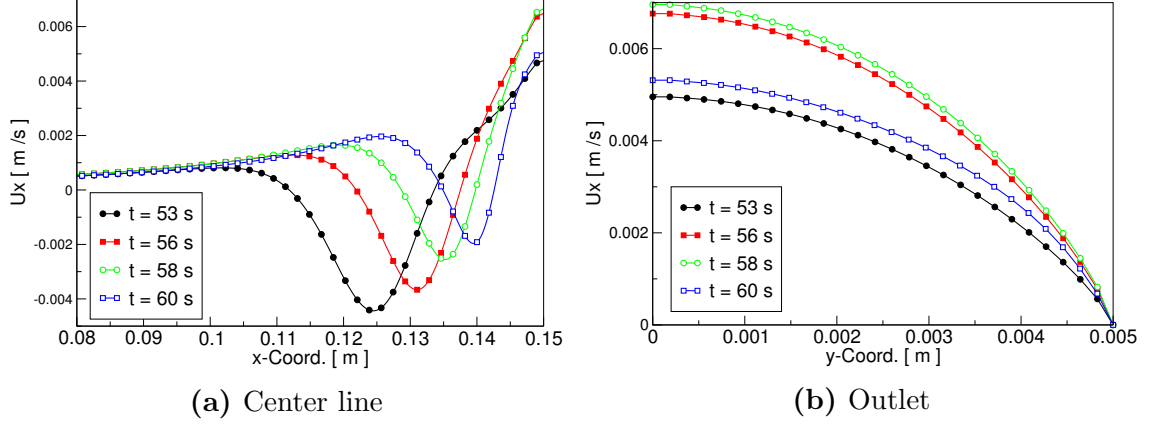


Figure 3.12: The x-component of the velocity of the Newtonian fluid N3 in the axisymmetric conical model at different times and/or wave positions. The wave speed and the maximum relative occlusion are 2.3 mm/s and 66%, respectively.

the outlet boundary conditions is going to be felt. Therefore, a decaying behavior in x-component of velocity is developed after the time $t = 58$ s and till the time $t = 60$ s as was shown in Figs 3.11b and 3.12.

The x-component of velocity at the times $t = 58$ s and $t = 60$ s along the center line and near the outlet for five different viscosities is shown in Figs 3.13 and 3.14, respectively. It is observed from these figures that, at time $t = 60$ s (that is when the maximum relative occlusion of 66% is achieved) all fluids (except the water) behave the same, in which the x-component of velocity becomes smaller in magnitude than the one computed at time $t = 58$ s. Figure 3.13 shows that all fluids exhibit a back-flow, reaching the maximum of 0.0046 m/s in magnitude for the case of the lowest viscous fluids in a region that is under the wave and along the center line, while the higher viscous fluids have almost the same magnitude of back-flow.

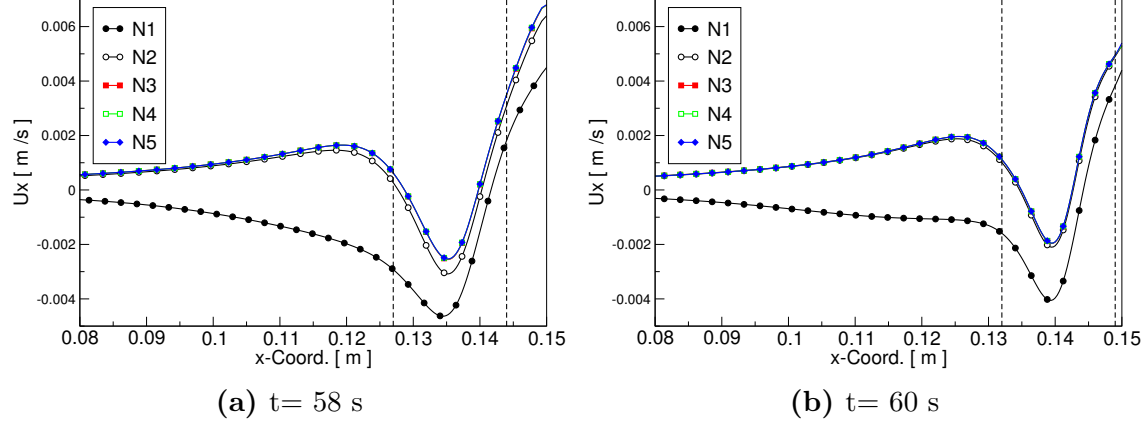


Figure 3.13: The x-component of the velocity of five Newtonian fluids in the axisymmetric conical model along the centerline. The wave speed and the maximum relative occlusion are 2.3 mm/s and 66%, respectively.

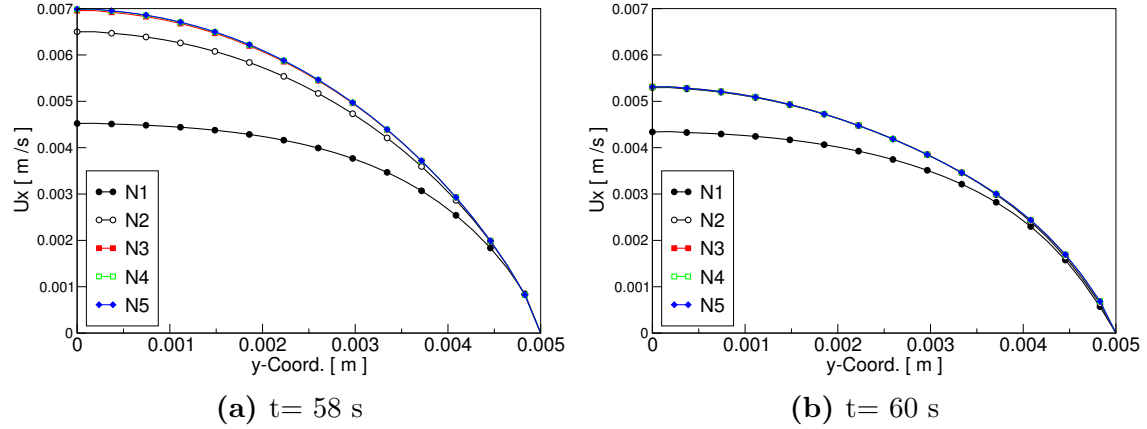


Figure 3.14: The x-component of the velocity of five Newtonian fluids in the axisymmetric conical model near the outlet. The wave speed and the maximum relative occlusion are 2.3 mm/s and 66%, respectively.

These results are in a good agreement with results given in Fig. 3.14. This figure shows that the velocity profiles computed near the outlet are almost identical for the case of the higher viscous fluids, and the maximum magnitudes of x-component of

velocity are achieved along the center line and near the pylorus sphincter.

The above results are consistent with the transport efficiency results for the five different viscosities, as is shown in the Fig. 3.15.

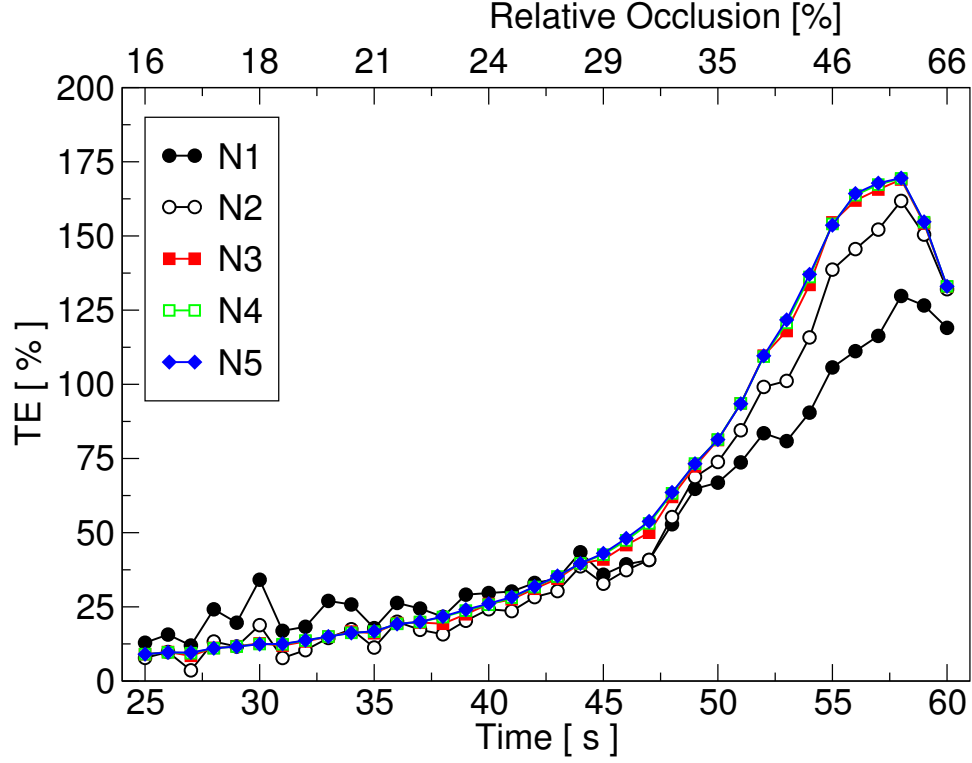


Figure 3.15: Transport efficiency for five Newtonian fluids in the axisymmetric conical model. The wave speed is 2.3 mm/s and the maximum relative occlusion is 66%.

This figure shows that the transport efficiency increases over the domain and attains its maximum value at time $t \approx 58$ s and then decreases until time $t = 60$ s where the maximum relative occlusion of 66% is achieved. The value of the transport efficiency increased with viscosity at slower rates, where the transport efficiency for the largest three viscosities appear nearly identical. The computations of the transport efficiency were initiated at time $t = 25$ s to avoid the transient effects at the beginning of the

motion and ended at time $t = 60$ s, which is also the time when the wave starts gradually climb up and out of the domain.

The fluid transport can be expressed by the average of the transport efficiency as is shown in Fig. 3.16. This figure displays the outcomes of simulating five different Newtonian fluids for the case of the wave speed of 2.3 mm/s and with three distinct maximum relative occlusions of 21%, 66% and 80%. The average transport efficiency is computed by averaging the transport efficiency curves starting at different times or positions. Therefore, the average transport efficiency can be computed by

$$\text{Average TE} = \frac{1}{t_F - t_S} \int_{t_S}^{t_F} \text{TE}(t) dt , \quad (3.8)$$

where t_S is the start time of averaging and t_F is the first time that the wave starts to go up/out of the domain.

Figure 3.16 shows that the average transport efficiency increases with relative occlusion. Specifically, this figure shows that the average transport efficiency increases with viscosity at small rates for the smallest two viscosities, that is for N1 and N2, while it is nearly identical for the largest three viscosities. On the other hand, this kind of the discrepancy between the lower and the higher viscous fluids has not been observed in the tubular simulations as was shown in Fig. 3.7, because there is no acceleration in the flow as the wave propagates toward the outlet. Keep in mind that the conical geometry causes this acceleration. Consequently, the inertial forces will be dominant for the case of the lower viscous fluids while the viscous forces will be

dominant for the case of the higher viscous ones.

Note that, the above figure together with the computed values of Reynolds number Re in Table 3.7, shows: When $Re < 1$, average transport efficiency curves coincide. On the other hand, when $Re > 1$, average transport efficiency decreases with increases Re . This is because more convection leads to larger back-flow, i.e., more fluid being pushed backward as opposed to forward, as was shown in Figs 3.13 and 3.14.

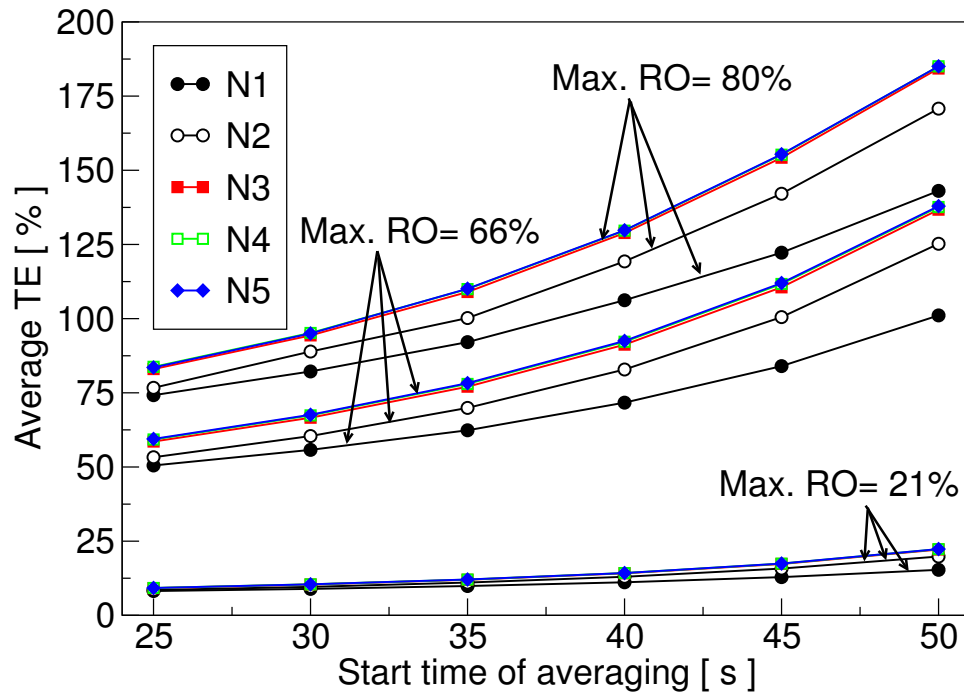


Figure 3.16: Average transport efficiency of five different Newtonian fluids in the axisymmetric conical model. The wave speed is 2.3 mm/s and three maximum relative occlusions of 21%, 66% and 80% are used.

3.2.2 Effect of Shear-Thinning Behavior

A comparison between the Newtonian and the non-Newtonian fluids will be carried out to identify the effect of the shear-thinning non-Newtonian behavior of the fluid. Two non-Newtonian fluids, BCA and BCB, with shear rate dependent viscosity expressed by the Bird-Carreau Eq. (2.17) will be used in this study, with the latter exhibiting considerably more shear-thinning behavior. The fluid parameters are given in Table 3.8 and the viscosity curves are depicted in Fig. 3.17.

Note that the zero shear rate viscosity η_0 for the non-Newtonian fluids is the constant viscosity used for the Newtonian fluid N3. Moreover, the shear-thinning for both non-Newtonian fluids starts at a strain rate of $\dot{\gamma} = \frac{1}{k} = 0.05 \text{ s}^{-1}$.

Table 3.8
Newtonian and non-Newtonian fluid parameters.

Parameters	N3	BCA	BCB
η_0 (Pa s)	0.1452	0.1452	0.1452
η_∞ (Pa s)	0	0	0
k (s)	—	20	20
n	1	0.75	0.5
Density (kg/m ³)	1000	1000	1000

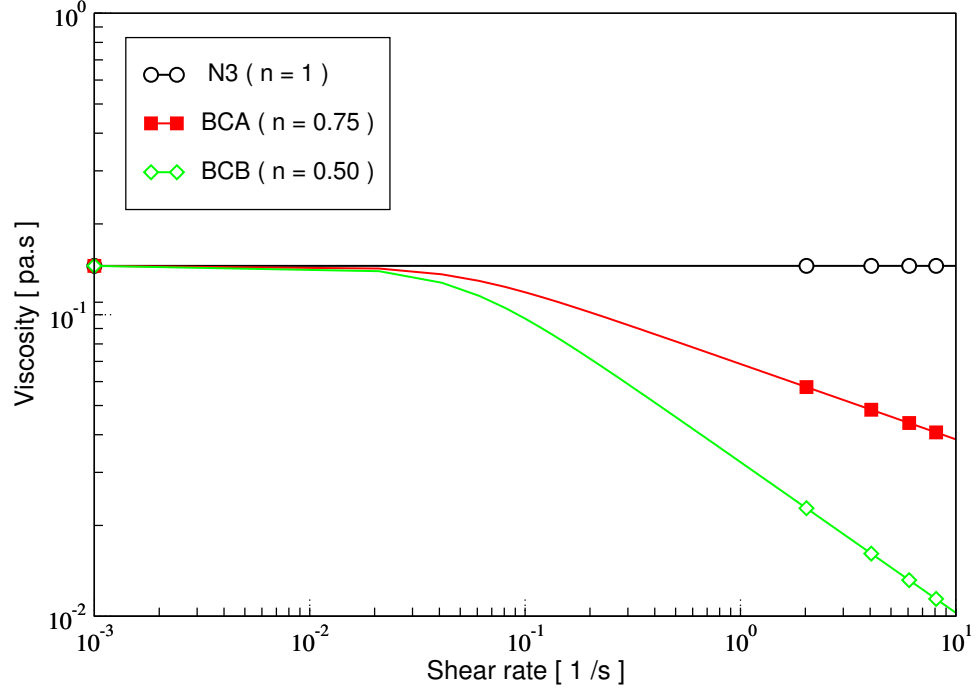


Figure 3.17: Shear rate dependent viscosity curves.

3.2.2.1 Axisymmetric Tubular Simulations

Simulation results of the Newtonian and non-Newtonian fluids for the case of the wave speed of 5 mm/s and with relative occlusions of 20%, 60% and 80% are shown in Fig. 3.18. This figure shows that the transport efficiency increases with relative occlusion and decreases with more shear-thinning fluid behavior for the case of relative occlusion of 60%, while it weakly depends on shear-thinning behavior for the case of the smallest and largest relative occlusions.

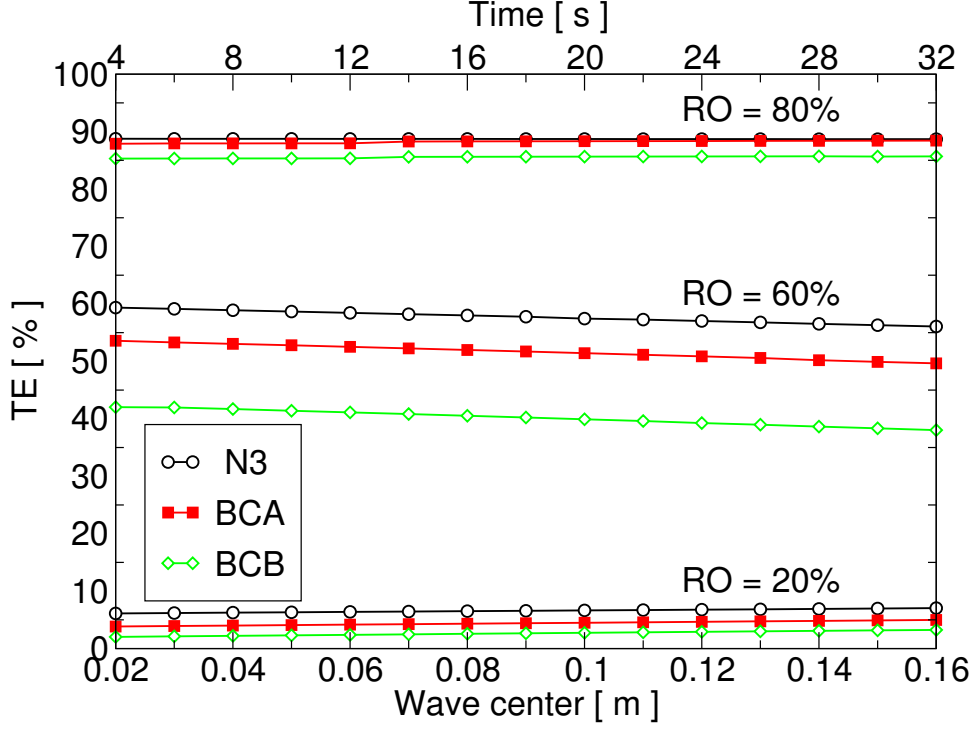


Figure 3.18: Transport efficiency for Newtonian and non-Newtonian fluids in the axisymmetric tubular model. The wave speed is 5 mm/s and three relative occlusions of 20%, 60% and 80% are used.

As we mentioned above, the difference between the transport efficiency curves diminishes for the case of relative occlusions of 20% and 80%, while this difference is clearly obvious for the case of relative occlusion of 60%. This behavior may be explained by considering the two extreme cases for relative occlusion, that is when the relative occlusion either equals 100% or 0%.

For the first extreme case of the relative occlusion of 100%, the peristaltic flow behaves like pressure driven Poiseuille flow, and since the tube was modeled as a closed system with incompressible fluid contents then the mass transport will be conserved to ensure the continuity and therefore the transport efficiency curves will be the same. On the other hand, the peristaltic flow behaves similar to a Couette-type flow for the

case of the relative occlusion of 0% and hence all fluids have the same velocity profiles near the outlet and therefore they have the same transport efficiency curves. In fact, the presence of a back-flow for the above two extreme cases is impossible to occur, which means that the fluid velocity profile near the outlet will not be affected anymore, and consequently the transport efficiency values will be the same. However, this is not the case when relative occlusion varies between the above two extreme cases.

The small differences in the computed average velocity over the outlet boundary as was shown in Fig. 3.10 stand behind the slight decreasing of the transport efficiency values over the (geometry) domain in the case of relative occlusion of 60%, as is shown in Fig. 3.18. This decreasing behavior doesn't evolve clearly in the case of 20% and 80% relative occlusions as is discussed above for the two extreme cases of relative occlusion and as is shown later in Fig. 3.26.

The following study has been performed for the standard tubular case at time $t = 32$ s, and the results are shown in Figs 3.19 and 3.20.

Figure 3.19 presents the strain rates along the center line for the Newtonian and non-Newtonian fluids. This figure shows that the maximum values of the strain rates are achieved at the (left) wave boundary and just under the wave, while the minimum ones are located along the center line. In this figure, the maximum values of the strain rate are observed for the more shear-thinning fluids, and reach the maximum value of 4.7 s^{-1} for the most shear-thinning fluid.

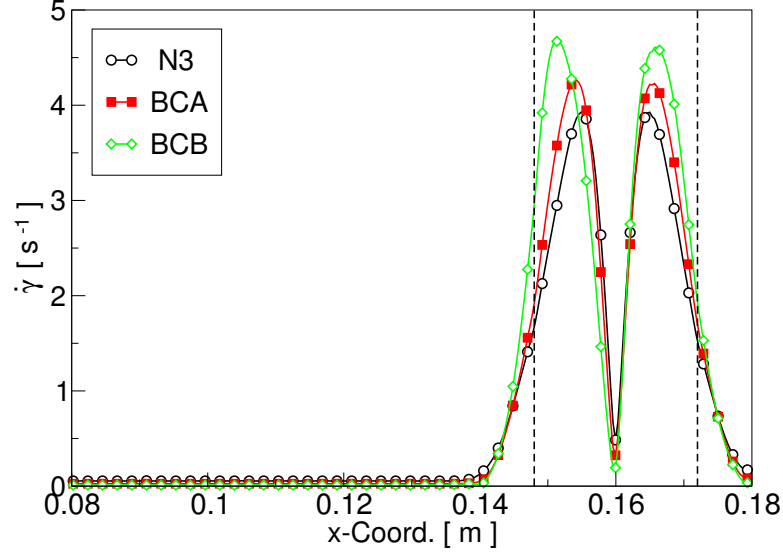


Figure 3.19: Strain rate of the Newtonian and non-Newtonian fluids in the axisymmetric tubular model along the center line and at $t = 32$ s. The wave speed and the relative occlusions are 5 mm/s and 60%, respectively.

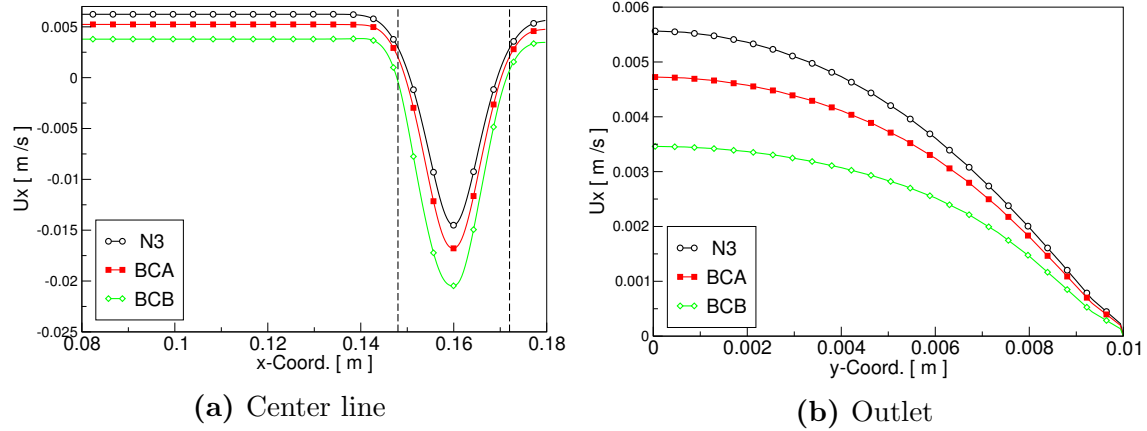


Figure 3.20: The x-component of the velocity of the Newtonian and non-Newtonian fluids in the axisymmetric tubular model at $t = 32$ s. The wave speed and the relative occlusion are 5 mm/s and 60%, respectively.

The x-component of velocity for the Newtonian and non-Newtonian fluids along the center line and near the outlet are shown in Figs 3.20a and 3.20b, respectively. Figure 3.20a shows that all fluids have a back-flow, and the strength of this back-flow

increases in magnitude with the shear-thinning behavior, reaching the maximum magnitude of about four times than that of the wave speed for the most shear-thinning fluids, in a region near the center line and just under the wave. The behavior of this back-flow is consistent with x-component of velocity profile computed near the outlet, as is shown in Fig. 3.20b. In this figure the velocity profile for the Newtonian fluid N3 resembles a parabola, whereas the velocity profiles for the non-Newtonian fluids are more plug-like, with the more shear-thinning fluid being flatter. Note that, the results of x-component of velocity near the outlet are consistent with the transport efficiency results given in Fig. 3.18.

3.2.2.2 Axisymmetric Conical Simulations

The following study has been carried out for the case of the standard conical simulations at different times, and the results are shown in Figs 3.21 - 3.23. Figures 3.21a and 3.21b show the strain rates along the center line for the Newtonian and non-Newtonian fluids at times $t = 45$ s (where the wave is far from the pylorus and a relative occlusion of 29% is achieved) and $t = 60$ s (where the wave is close to the pylorus and the maximum relative occlusion of 66% is achieved), respectively. Figure 3.21 shows that the maximum strain rates along the center line occur under the leading half of the wave.

Moreover, this figure shows that as the wave approaches the pylorus sphincter, the increasing occlusion of the wave strengthens the strain rate values in the narrow part of the pyloric canal.

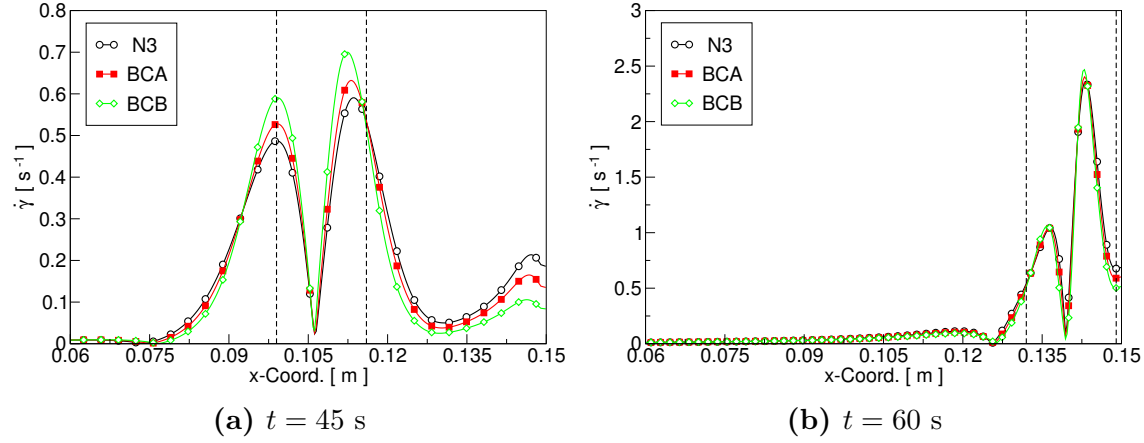


Figure 3.21: Strain rate of the Newtonian and non-Newtonian fluids in the axisymmetric conical model along the center line and at times $t = 45$ s and $t = 60$ s. The wave speed and the maximum relative occlusion are 2.3 mm/s and 66%, respectively.

It can be observed from Fig. 3.21a that the strain rate depends on the shear-thinning behavior when the wave is far from the pylorus, in which the maximum value of the strain rate is observed for the most shear-thinning fluid. On the other hand, this dependence is significantly weakened as the wave propagates gradually toward the pylorus sphincter as is observed in Fig. 3.21b.

Figure 3.22 shows x-component of velocity for the Newtonian and non-Newtonian fluids near the outlet at different times and relative occlusions.

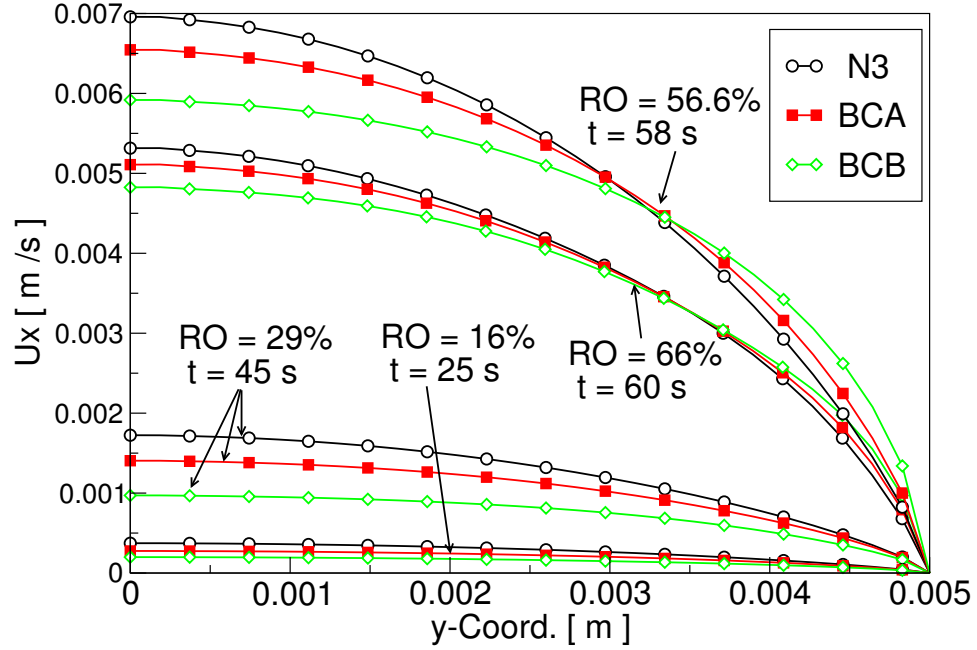


Figure 3.22: The x-component of the velocity of the Newtonian and non-Newtonian fluids in the axisymmetric conical model near the outlet and at different times. The wave speed and the maximum relative occlusion are 2.3 mm/s and 66%, respectively.

It can be seen from this figure that increasing the relative occlusion gives higher velocity along the center line, reaching the maximum value of 7 mm/s at time $t = 58$ s and at relative occlusion of 56.6%, and then decaying until time $t = 60$ s where the maximum relative occlusion of 66% is achieved. As we mentioned before, this decaying behavior is due to the influence of the outlet boundary conditions which becomes less suitable when the wave gets too close to the pylorus. Moreover, the effect of the shear-thinning behavior increases with relative occlusion, as can be seen from the increased deviation between the curves, up until the time when the outlet boundary condition influences the flow behavior. The above observations are consistent with the ones that we observed in the tubular simulations case.

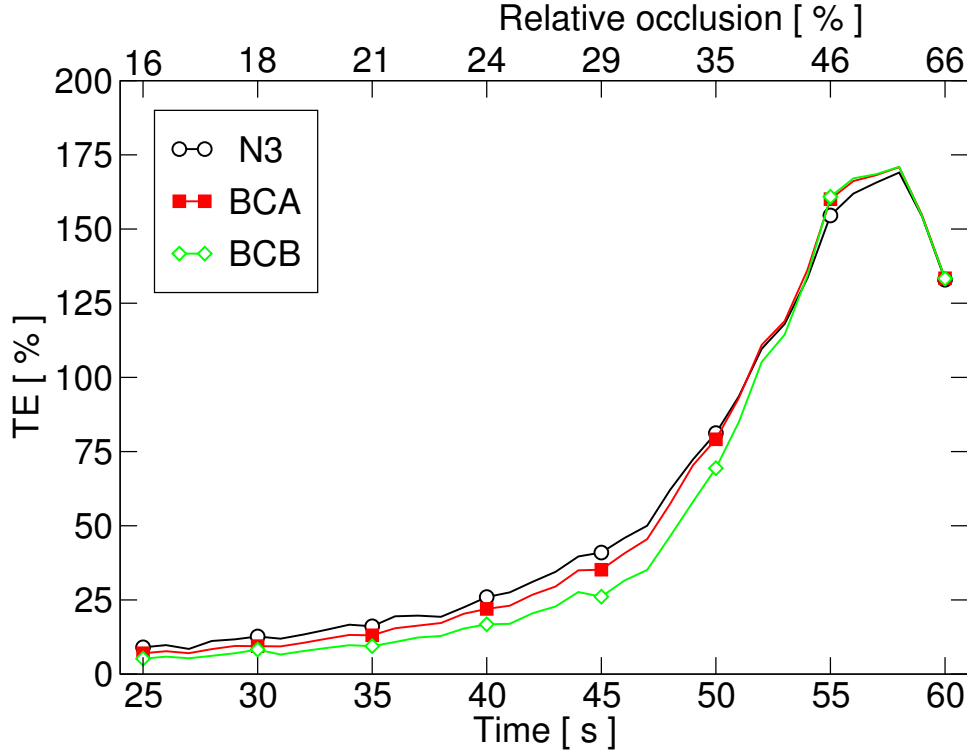


Figure 3.23: Transport efficiency for the Newtonian and non-Newtonian fluids in the axisymmetric conical model. The wave speed is 2.3 mm/s and the maximum relative occlusion is 66%.

Figure 3.23 shows the transport efficiency for the Newtonian and non-Newtonian fluids as a function of time. This figure shows that the transport efficiency is almost independent of the shear-thinning behavior when the wave is either far from or close to the pylorus sphincter, that is for small and/or large values of relative occlusions. On the contrary, the transport efficiency is slightly influenced by the shear-thinning behavior for the case of the intermediate values of occlusions, where it decreases with increasing shear-thinning behavior. These results are in a good agreement with the tubular transport efficiency results shown in Fig. 3.18, where there is little or no effect of power-law index, n , for small and large relative occlusions, while transport

efficiency increases with increased power-law index for intermediate relative occlusions. In addition, this figure shows that the transport efficiency increases over the domain, reaching the maximum value at time $t = 58$ s and then decreases due to the boundary condition effects at the pylorus.

The results of averaging the transport efficiency curves for the Newtonian and non-Newtonian fluids at different times and with three maximum relative occlusions of 21%, 66% and 80% are shown in Fig. 3.24. This figure shows that the average transport efficiency increases with maximum relative occlusion. Also, the average transport efficiency depends weakly on the shear-thinning behavior, reaching the minimum for the most shear-thinning fluid.

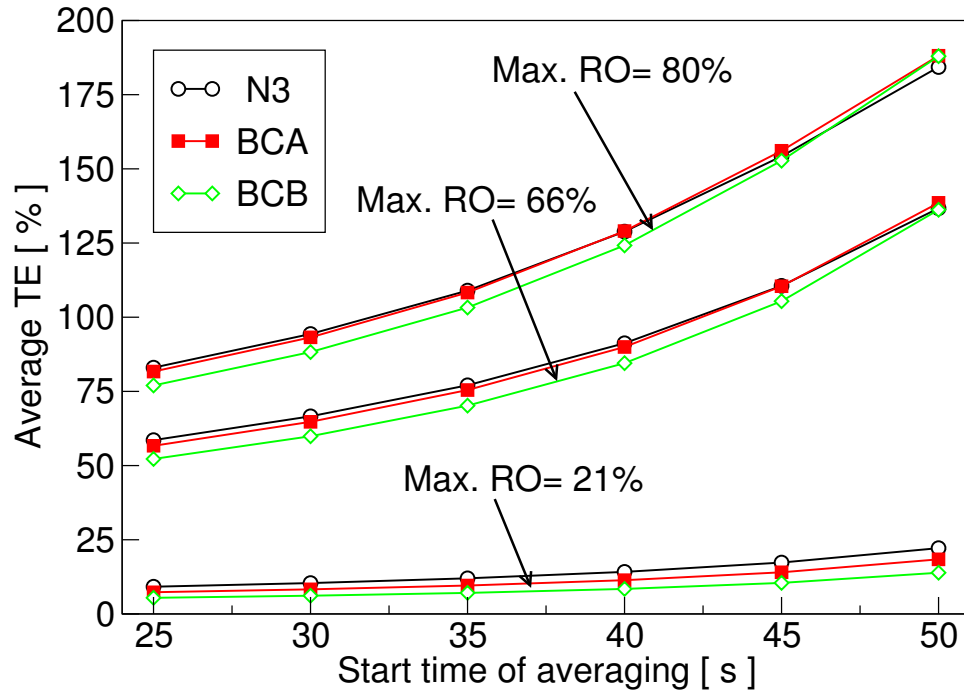


Figure 3.24: Average transport efficiency for the Newtonian and non-Newtonian fluids in the axisymmetric conical model. The wave speed is 2.3 mm/s and three maximum relative occlusions of 21%, 66% and 80% are applied.

3.2.3 Variation of Wave Occlusion and Wave Speed

3.2.3.1 Axisymmetric Tubular Simulations

In order to study the transport efficiency in terms of relative occlusion and wave speed, several simulations for the Newtonian fluid N3 have been carried out and their results are shown in Figs 3.25 - 3.27. Figure 3.25 shows that the transport efficiency is independent of the wave speed and increases with relative occlusion.

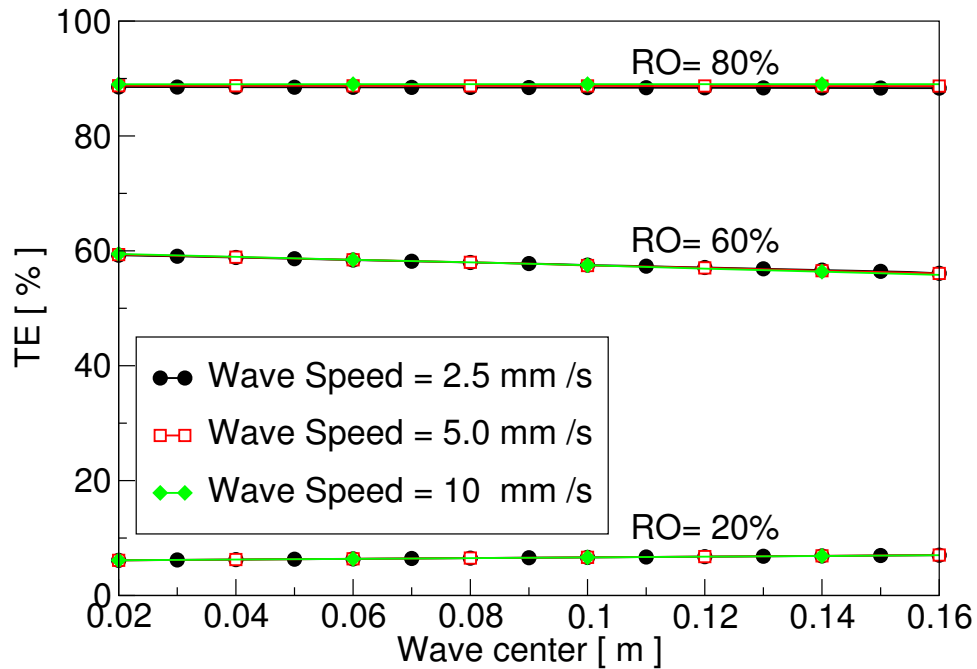


Figure 3.25: Transport efficiency for Newtonian fluid N3 in the axisymmetric tubular model.

Figure 3.26 shows the x-component of velocity along the center line and near the outlet at time $t = 32$ s for three different relative occlusions and with fixed uniform wave speed of 5 mm/s. It can be seen from Fig. 3.26a that there exists a back-flow for all different relative occlusions. It is not surprising to observe from this figure that the maximum magnitude of this back-flow is nearly about of one, two and three times that of the wave speed for the case of relative occlusions of 20%, 80% and 60%, respectively. This can be justified by the fact that there is no back-flow in the case of the extreme values of relative occlusions as we discussed previously (that is when relative occlusion either of 0% or 100%). Figure 3.26b shows that the x-component of velocity near the outlet increases in magnitude with relative occlusion. These results are in a good agreement with the results of the transport efficiency given in Fig. 3.25.

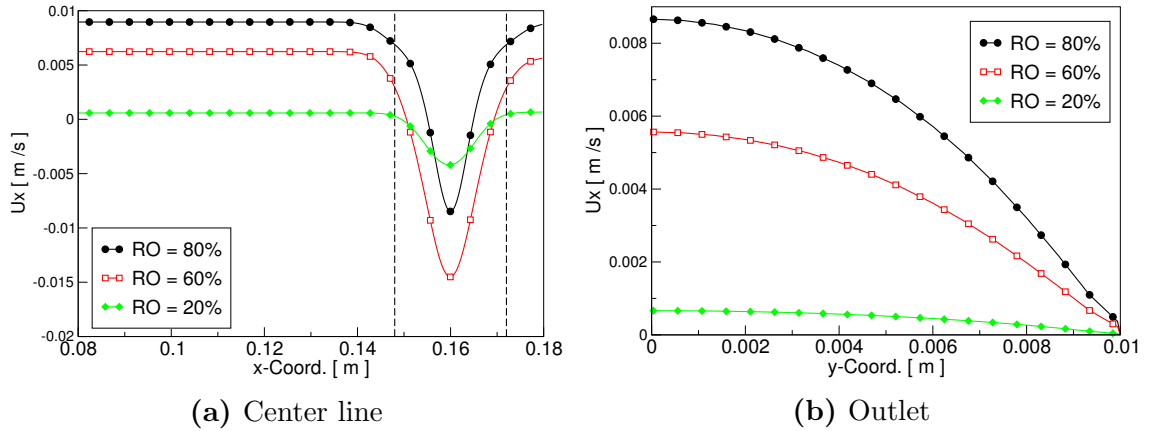


Figure 3.26: The x-component of the velocity of the Newtonian fluid N3 in the axisymmetric tubular model at $t = 32$ s for different relative occlusions. The wave speed is 5 mm/s.

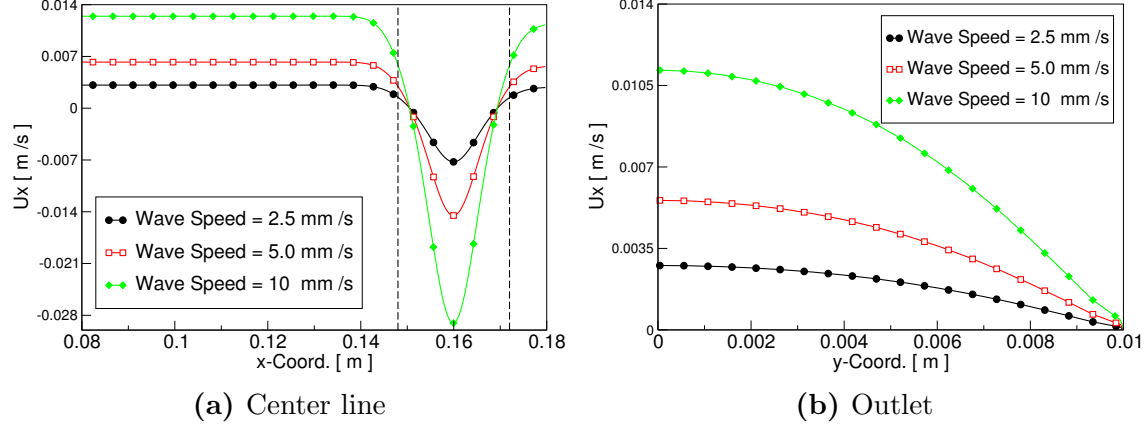


Figure 3.27: The x-component of the velocity of the Newtonian fluid N3 in the axisymmetric tubular model at wave center of $x = 160$ mm for different wave speeds. The relative occlusion is 60%.

Figure 3.27 shows the x-component of velocity along the center line and near the outlet at $x = 160$ mm for three different wave speeds and with a fixed relative occlusion of 60%. Figure 3.27a indicates the presence of a back-flow for all different wave speeds, and this back-flow increases linearly in magnitude with wave speed. The previous results are consistent with x-component of velocity profiles computed near the outlet, as is shown in Fig. 3.27b. It is obvious from this figure that the velocity values increase linearly in magnitude with the wave speed.

3.2.3.2 Axisymmetric Conical Simulations

Two sets of simulations for the Newtonian fluid N3 have been performed at different times and positions to investigate the influence of wave occlusion and wave speed

on the transport efficiency. The results of the first set are shown in Figs 3.28 and 3.29 with a fixed wave speed of 2.3 mm/s and with three different ranges of relative occlusions.

Figure 3.28 shows that as the wave approaches the pylorus sphincter, the increasing occlusion of the wave strengthens the transport efficiency, reaching the maximum of 214%, 168% and 46% at times of $t = 57$ s, $t = 58$ s and $t = 60$ s, respectively. It can be observed from this figure that the effect of the outlet boundary condition when the wave gets to close to the pylorus, which is characterized by the decaying behavior in the transport efficiency, occurs sooner for the highest relative occlusion. These results are consistent with the results of x-component of velocity computed near the outlet at three different times, as is illustrated in Fig. 3.29. It can be observed from this figure that the maximum magnitude of x-component of velocity is about of three to four times that of the wave speed at time $t = 57$ s (i.e. black curve corresponding to Max. RO of 80%) and nearly of three times that of the wave speed at time of $t = 58$ s (i.e. red curve corresponding to Max. RO of 66%), while it is less than the wave speed at time $t = 60$ s (i.e. green curve corresponding to Max. RO of 21%).

Figure 3.29c shows that the x-component of velocity, near the outlet and at time $t = 60$ s, decreases at small rate (on average) as relative occlusion reduced from 80% to 66%, which is consistent with the results of transport efficiency results given in Fig. 3.28.

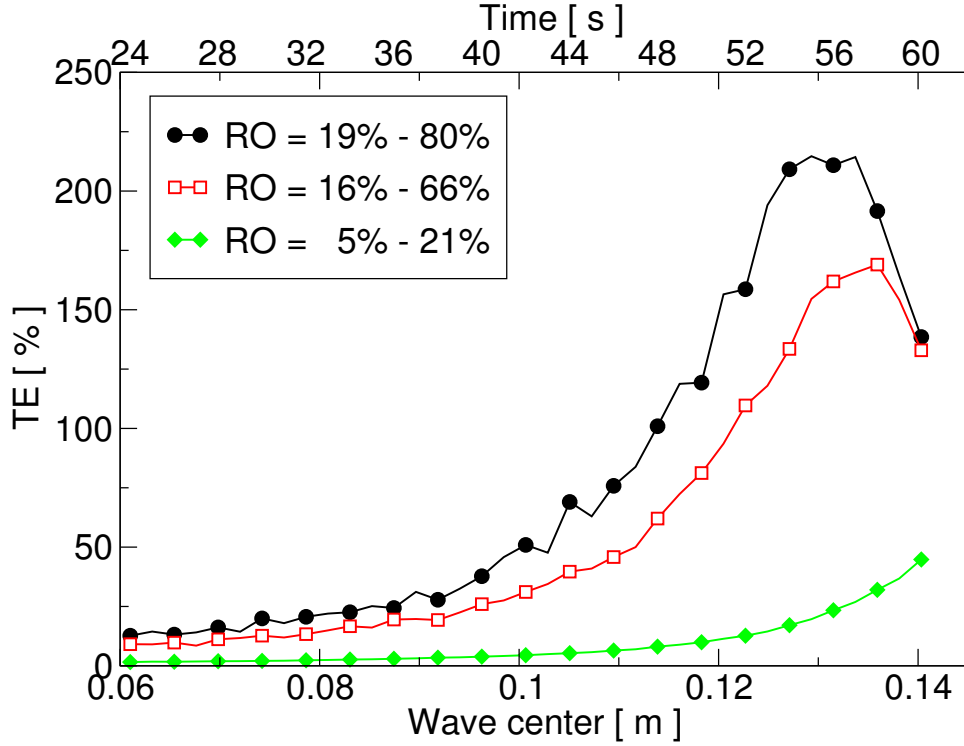


Figure 3.28: Transport efficiency for the Newtonian fluid N3 in the axisymmetric conical model. The wave speed is 2.3 mm/s and three different ranges of relative occlusions are applied.

The results of the second set are given in Figs 3.30 - 3.32 with a fixed maximum relative occlusion of 66% and with three distinct wave speeds of 1.15 mm/s, 2.3 mm/s and 4.6 mm/s. Figs 3.30 and 3.31 show the x-component of velocity along the center line and near the outlet at two wave centers of $x = 135.5$ mm (where the maximum magnitude of velocity is achieved) and of $x = 139.8$ mm (where the maximum relative occlusion of 66% is achieved).

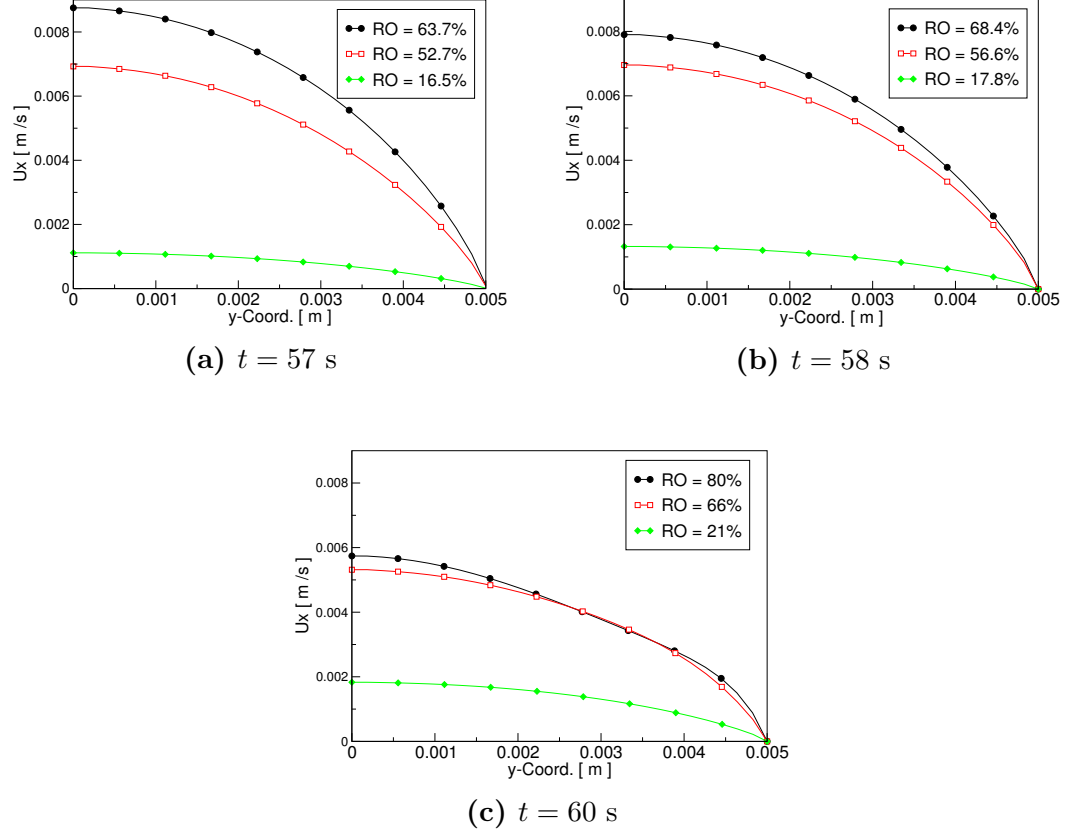
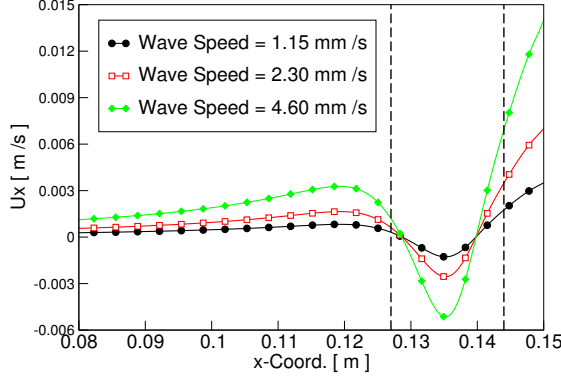


Figure 3.29: The x-component of the velocity of the Newtonian fluid N3 in the axisymmetric conical model near the outlet. The wave speed is 2.3 mm/s and three different sets of relative occlusions are used at times of $t = 57$ s, $t = 58$ s and $t = 60$ s.

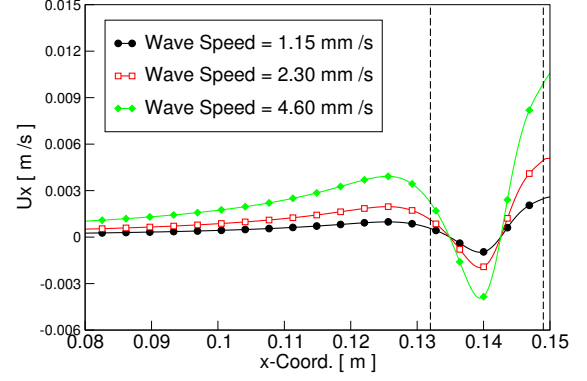
Figure 3.30 shows the presence of a back-flow for all different wave speeds, this back-flow increases linearly in magnitude with wave speed. It can be observed from Fig. 3.30a that the maximum magnitudes of x-component of velocity along the center line are achieved under the wave and they increase linearly in magnitude with wave speed, reaching the maximum of 5.2 mm/s at wave center of 135.5 mm. These observations are consistent with the results of x-component of velocity along the outlet, given in Fig. 3.31. This figure shows that the maximum magnitudes of x-component of

velocity were achieved near the pylorus and they increased linearly with wave speed, reaching the maximum of six times than that of the wave speed at wave center of $x = 135.5$ mm, as is shown in Fig. 3.31a. These results are consistent with the results of the transport efficiency and their average as is shown in Figs 3.32 and 3.33.

Figure 3.32 shows that the transport efficiency is almost independent of the wave speed for a fixed maximum relative occlusion of 66%. Moreover, this figure shows that the transport efficiency increases over the domain, reaching the maximum of 169% at relative occlusion of 56.6% (that is at wave center of $x = 135.5$ mm), and then decreases to 133% at relative occlusion of 66% (that is at wave center of $x = 139.8$ mm). Recall that, this decaying behavior is evolved because the effect of the outlet boundary conditions was being felt. Figure 3.33 shows the average transport efficiency for the three different ranges of relative occlusion and three different wave speeds. This figure shows that the average transport efficiency is essentially independent of the wave speed and increases with maximum relative occlusion.

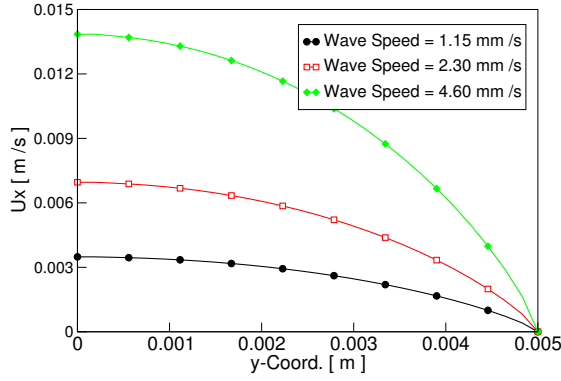


(a) $x = 135.5$ mm, RO = 56.6%

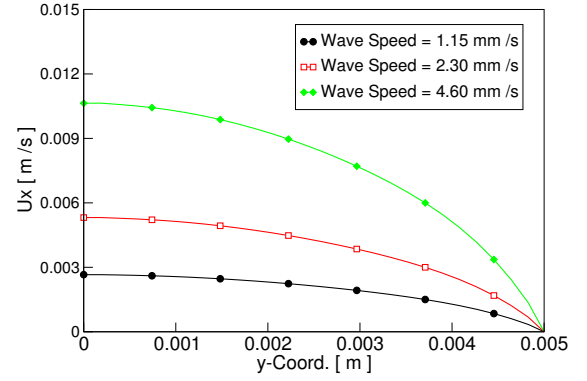


(b) $x = 139.8$ mm, RO = 66%

Figure 3.30: The x-component of the velocity of the Newtonian fluid N3 in the axisymmetric conical model along the center line at two different wave centers. Three different wave speeds of 1.15 mm/s, 2.3 mm/s and 4.6 mm/s are used.



(a) $x = 135.5$ mm, RO = 56.6%



(b) $x = 139.8$ mm, RO = 66%

Figure 3.31: The x-component of the velocity of the Newtonian fluid N3 in the axisymmetric conical model near the outlet at two different wave centers. Three different wave speeds of 1.15 mm/s, 2.3 mm/s and 4.6 mm/s are used.

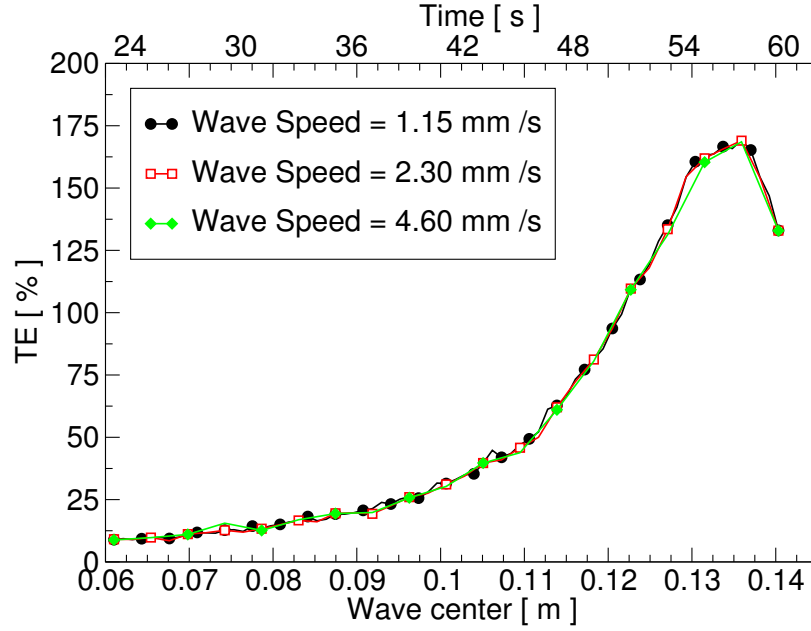


Figure 3.32: Transport efficiency for the Newtonian fluid N3 in the axisymmetric conical model. The maximum relative occlusion is 66% and three different wave speeds are used.

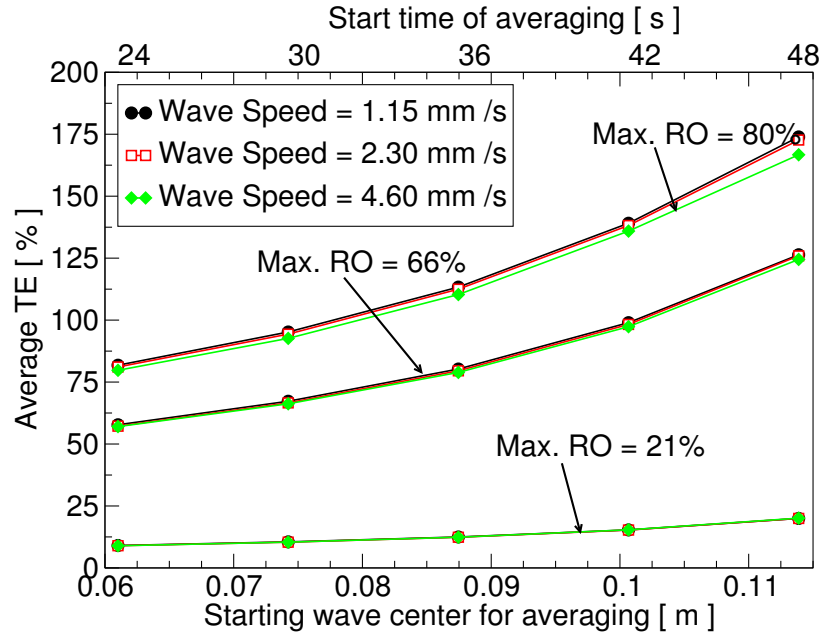


Figure 3.33: Average transport efficiency for the Newtonian fluid N3 in the axisymmetric conical model with different three wave speeds and maximum relative occlusions.

3.3 Summary and Conclusion

Two axisymmetric numerical models were developed to model the emptying process (or the fluid transport) in terms of the (average) transport efficiency, and to get a better understanding of the flow field that develops within the system, a 2-D axisymmetric tubular model and a 2-D axisymmetric conical model. The first model corresponds to a realistic tubular peristaltic flows as encountered in the small intestine, while the second one simulates the peristaltic flow in the lower part of an idealized human stomach. These two models are sufficient to reduce the high level of complexity in full 3-D models.

These models were coupled with the modified CFD finite volume code solver from the open source software package OpenFOAM, and the fixed (laboratory or Eulerian) frame of references is used to simulate the peristaltic motion for different Newtonian and non-Newtonian fluids. The non-Newtonian fluid is modeled by using the Bird-Carreau Yasuda viscosity law. A mesh refinement study showed an adequate mesh independence and the transient computations exhibited plausible convergence in terms of the initial residual.

To reflect an experimental setup in which the peristaltic flow was induced by deforming a tube using rollers that moved along the tube wall, a 2-D planar model has been designed. A good agreement was found with experimental data, hence confirming that the numerical models and methods were valid for the peristaltic simulations.

In general, the presence of back-flow is a ubiquitous feature of all our simulations, and the maximum magnitudes of x-component of velocity increase linearly with the wave speed. Also, the (average) transport efficiency is insensitive to the wave speed and increases with relative occlusion. Moreover, the maximum values of the strain rates are achieved at the wave boundary and under the wave, while the minimum ones are located along the center line and they are insensitive to the power-law index. In addition, (average) transport efficiency is insensitive to the power-law index for small and large relative occlusions, while it decreases at small rate with power-law index for intermediate relative occlusions.

In particular, the simulation results within the tube show that transport efficiency is insensitive to the Newtonian viscosity and increases with relative occlusion. Moreover, the maximum magnitudes of x-component of velocity are attained along the center line and just under the wave. These maximum magnitudes are nearly about of one to two times than that of the wave speed for small and large relative occlusions, while it is increased to three times than that of the wave speed for intermediate relative occlusions. Specifically, the standard tubular simulations results show that the higher viscous Newtonian fluids have almost the same magnitudes of back-flow, reaching the maximum magnitude of about of three times than that of the wave speed, while the minimum magnitude of back-flow, which is about of two times than that of the wave speed, is attained for the lowest viscous Newtonian fluid. This back-flow increases in magnitude by reducing the power-law index for the non-Newtonian fluids, reaching

the maximum magnitude of about of four times than that of the wave speed. By reducing the power-law index, the maximum values of strain rates increase, reaching the maximum for the most shear-thinning fluids.

On the other hand, the simulation results within the lower part of human stomach show that the maximum value of the (average) transport efficiency increased with Newtonian viscosity at smaller rate, where the maximum (average) transport efficiency for the largest three Newtonian viscosities appear nearly identical. Moreover, as the wave approaches the pylorus sphincter, the increasing occlusion of the wave strengthens the magnitudes of x-component of velocity faster, in a region close close to center line and near the pylorus, reaching the maximum magnitude of about of three to four times than that of the wave speed. Specifically, the standard conical simulations results show that the maximum magnitude of x-component of velocity is about of three times than that of the wave speed. In addition, the maximum magnitude of back-flow is achieved for the most lowest viscous Newtonian fluid, which is about of two times than that of the wave speed, while the largest three Newtonian viscosities have almost the same magnitude of back-flow, reaching the maximum magnitude of about of one times than that of the wave speed. The maximum magnitudes of x-component of velocity are insensitive to the power-law index for small and large relative occlusions, while they increase with power-law index for the intermediate values of relative occlusion. When the wave is far from the pylorus, the maximum values of strain rates increase by reducing the power-law index, while they are insensitive to

the power-law index when the wave is close to pylorus.

Chapter 4

Gastric Digestion and Mixing Via Peristaltic Motion

The human stomach is a J-shaped, muscular, hollow and dilated part of the gastrointestinal tract that functions as an important organ in the digestive system. It is located between the esophagus and the first part of small intestine (duodenum) in the region of the left side of the upper abdominal cavity. Anatomically, the stomach is subdivided into the fundus, the corpus, and the antrum (Fig. 4.1-A).

The upper part of the stomach (the fundus and the upper corpus) acts as a reservoir for chewed up food (bolus) that enters the stomach through the esophagus via the lower esophageal sphincter, while the lower part (the antrum and the distal corpus) is responsible for mechanical forces and fluid motions that promote not only the

breakdown and mixing of gastric content, but also its chemical digestion, absorption and transport. After that, the pyloric sphincter controls the passage of partially digested food (chyme) from the stomach into the duodenum where peristalsis takes over to move this through the rest of the small intestines. The curved, twisted shape of the stomach not only supports gastric mixing, but also separates the stomach into reservoir and mixing regions.

The chemical processes are typically investigated by means of in vitro analyses. However, the development of an in vitro system capable of reproducing the fluid mechanical forces that promote digestion and mixing is a real challenge. Pal et al. [68] and Ferrua and Singh [3] have developed realistic computational models for the investigation of the mechanical digestion process when the pylorus valve is closed. A summary of these studies are given in Section 4.4. The present investigations differ from the ones of Pal et al. [68] and Ferrua and Singh [3] in the following three ways: First, we develop a simple 2-D axisymmetric numerical model (Fig. 4.1-B), that reduces the high level of complexity in the full 3-D model, to illustrate the principles of mechanical digestion and mixing within the lower part of a human stomach. Second, a parameter study is performed to investigate the effect of various geometrical and rheological parameters on the gastric digestion and mixing, to get a better understanding of the flow field that developed within the lower part of a human stomach. Third, but most importantly, antral contractions have been allowed to live in the vicinity of the pylorus, which is presumably the worst case scenario

where the largest gradients for velocity and pressure occur.

The simulations were performed in the fixed frame of reference with a modified solver from open source software package, OpenFOAM. Moreover, the finite volume method (FVM) is employed to solve the conservation equations of mass and momentum for velocity and pressure, and the Bird-Carreau Yasuda viscosity law is used to model the non-Newtonian fluids. After investigating the convergence criteria and mesh resolution, a comparison to the experimental and numerical data has been made to validate the numerical models and methods.

4.1 Computational Models

4.1.1 Geometry

As in Chapter 3, a 2-D axisymmetric conical model has been developed to simulate the peristaltic motion for different fluids in the fixed (laboratory) frame of reference, where the boundary motion is represented by traveling waves which deform the boundary and hence the mesh. Within the bounds of this model, the user can input geometrical and rheological parameters to overcome the difficulty of reproducing gastric motility and physiology of the lower part of a human stomach.

The geometry of this model is specified as a wedge of a small angle of 5° and one cell thick running along the axis of symmetry, straddling one of the coordinate planes. The upper wall of this geometry inclines from the x-axis by angle of 16.7° and it is deformed by sequence of the ACWs. The ACW is moving with a uniform speed in the x-direction and it is generated by moving the mesh points of the upper wall along the wedges.

The model is designed to reflect a cross sectional of the lower part of an average sized human stomach (Fig. 4.1-A), where the characteristic dimensions are obtained from the literature of Keet [128], Schulze [42], Pal et al. [46, 68] and Ferrua and Singh [3]. This lower part (antrum) can be considered as a frustum of a circular cone whose length is 150 mm and with diameters of 100 mm and 10 mm at its widest point and at the pyloric ring, respectively, as is shown in Fig. 4.1-B. The relative occlusion of the ACW (the occlusion diameter to the antral diameter without the wave) is defined, as in Chapter 3, by $RO = \left(1 - \frac{H}{H_0}\right) \times 100$, refer to Fig. 4.1 for details, where H_0 is the stomach width and H is the gap width. Two maximum relative occlusions of 52% and 80% and two types of ACW are used in this study, a circular ACW and a parabolic ACW.

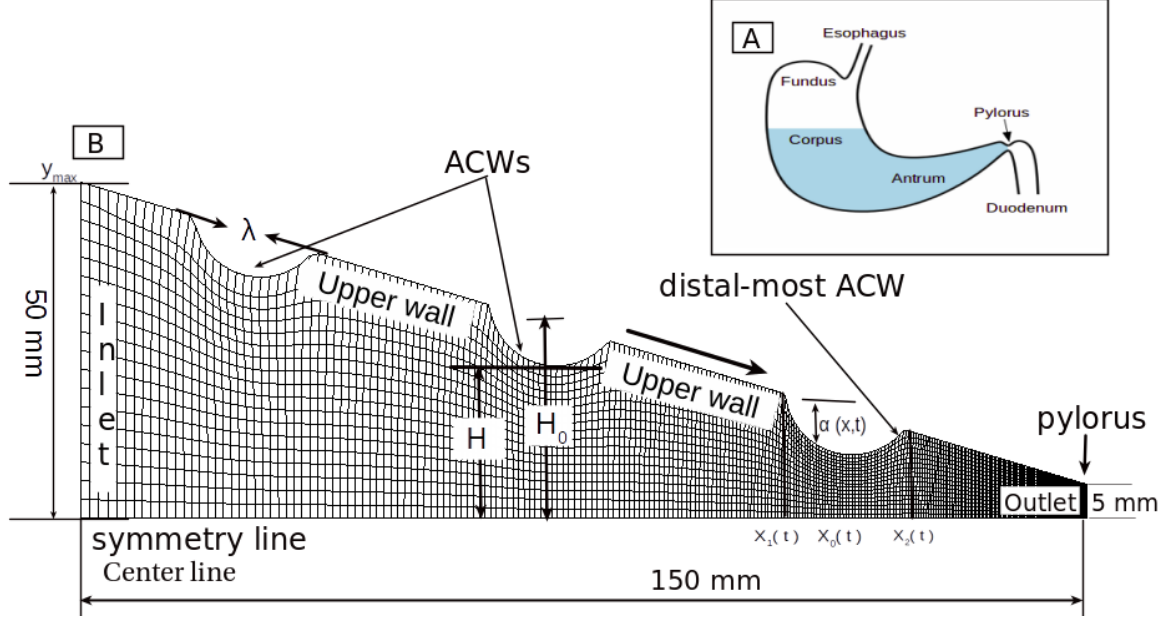


Figure 4.1: (A) Schematic diagram of a human stomach. (B) Computational domain for the axisymmetric conical model equipped by the deformation of the ACWs and relative occlusion parameters.

The circular deformation is given by

$$\alpha(x, t) = [y_{max} - x \tan(\theta)] - \left[y_0(t) - \sqrt{r^2 - (x - x_0(t))^2} \right], \quad (4.1)$$

where $(x_0(t), y_0(t))$ is the center of the circle whose radius is r . The parabolic deformation is described as

$$\alpha(x, t) = [y_{max} - x \tan(\theta)] - \left[y_0(t) + \left(\frac{2(x - x_0(t))}{\lambda} \right)^2 (y_{max} - y_0(t)) \right], \quad (4.2)$$

where $(x_0(t), y_0(t))$ is the vertex of the parabola whose width is λ .

In these deformations: $x_1(t) \leq x \leq x_2(t)$, y_{max} is the maximum y-component

of the points on the undeformed upper wall, $\theta = \arctan\left(\frac{y_{max}-y_{min}}{x_{max}-x_{min}}\right)$ is the inclination angle of the upper wall from the x-axis, y_{min} is the minimum y-component of the points on the undeformed upper wall, x_{max} is the maximum x-component of the points on the center line, and x_{min} is the minimum x-component of the points on the center line. These values are specified by the user in the `<case>/constant/polyMesh/blockMeshDict` file (see Appendix B).

The x-component of the center (vertex) and the two ends of the circular (parabolic) arc move with a uniform speed in the x-direction as

$$x_i(t) = (x_i + ct) \cos(\theta); \quad i = 0, 1, 2,$$

where x_i 's are the initial values at time $t = 0$ and c is the ACW speed. The y-component of the center (vertex) moves in the direction parallel to the undeformed upper wall which is described by the equation: $y = y_{max} - x \tan(\theta)$. Note that, $y_0(t) = y_0$ for all t when $\theta = 0$. The values of x_0 , y_0 , r and λ are specified by the user, while x_1 and x_2 are computed so that $\alpha(x_i(t), t) = 0$; $i = 1, 2$.

In OpenFOAM, these parameters are specified in the `<case>/0/pointMotionU` file (see Appendix B) as follows: `circleRadius` to specify the radius of the circular ACW, `speed` to specify the speed of the ACW, `yCompFinalCenter` (`yCompFinalVertex`) in circular (parabolic) deformation for y_0 , and `chordLength` to specify the width λ of the parabolic ACW. The occlusion diameter H is computed by subtracting `circleRadius` from the y-component of the center $y_0(t)$ in the circular deformation (see Fig. 3.1),

and by setting it to the y-component of the vertex $y_0(t)$ in the parabolic deformation. In this file additional parameters have been added to control the motion: `numOfWaves` to specify the number of the ACWs, `shift` to control the degree of occlusion along the domain in the parabolic deformation, `period` to initiate a new ACW every certain period of time, and l to specify the distance between the front of the distal-most ACW and the outlet when the distal-most ACW start to climb up by a specified angle of `beta` and with a speed of `alpha`. In our current standard simulations (that is when the speed of the ACW is 2.3 mm/s), the circular ACWs were initiated every 20 s at 138.5 mm from the pylorus with a diameter of 20 mm.

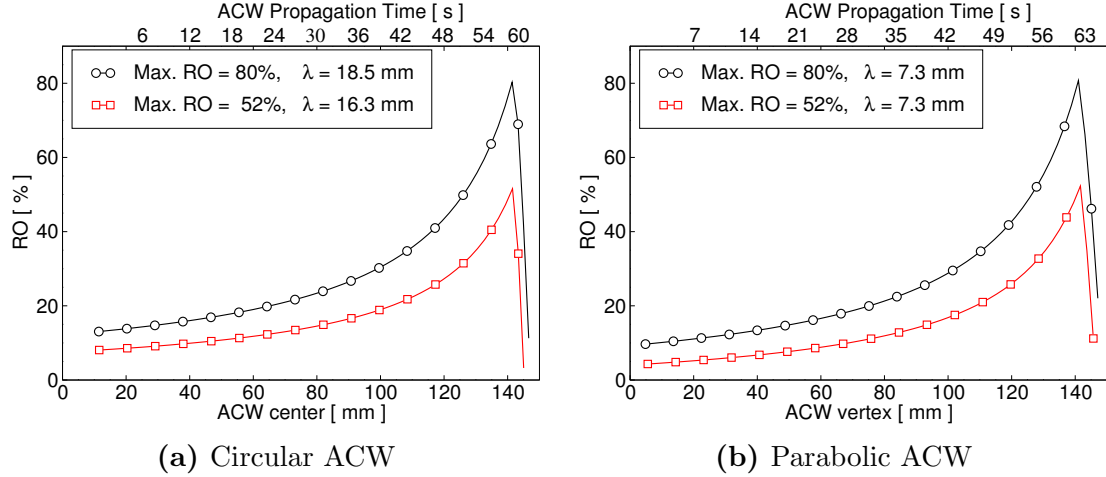


Figure 4.2: Motility pattern of the distal-most ACW during digestion. The ACWs speed is 2.3 mm/s.

The relative occlusion increases over the domain as the circular ACW propagates for 60 s to 8.5 mm from the pylorus, reaching the maximum occlusion of 80% and 52% with corresponding ACW widths of 18.5 mm and 16.3 mm, respectively (Fig. 4.2a). Similarly, the parabolic ACWs exhibit maximum relative occlusions of 80% and 52%,

and were initiated every 20 s at 144.6 mm from the pylorus with a width of 7.3 mm. Also, the relative occlusion increases over the domain as the parabolic ACW propagates for 63 s to 8.5 mm from the pylorus (Fig. 4.2b). After that, the relative occlusion decreases for both ACWs because the distal-most ACW starts to climb up by an elevation angle of 45° . For comparison reasons and as is discussed in more details below, the axial (along x-axis) center and vertex of the distal-most ACW have almost the same distance from the pylorus, regardless degree of occlusion and wave shape.

4.1.2 Governing Equations

The fluid is taken to be a single-phase fluid, and the flow is assumed to be incompressible, isothermal and inelastic. Neither the wall roughness nor the friction and the gravity forces were considered in the fluid simulations. The mesh motion equation and the flow conservation equations are the same as those introduced in the Section 3.1.2. Moreover, the boundary conditions for the pressure and velocity on the inlet and upper wall boundaries and at the center line are the same as those introduced in the emptying process discussed in the Section 3.1.2. However, different boundary conditions for the pressure and velocity are required on the outlet (pylorus) boundary. On the outlet, the velocity boundary conditions are set to zero while the pressure boundary conditions are set to be zero gradient. The boundary conditions

for cell velocity \mathbf{w} are as follows: $\mathbf{w} = \mathbf{0}$ on the center line, inlet and outlet boundaries, i.e. neither the inlet and outlet are allowed to be intersected by the ACWs nor the center line is allowed to move. The velocity on the upper wall is determined by a prescribed mathematical formula either by Eq. (4.1) or Eq. (4.2) that gives the position of the mesh points as a function of time. These boundary conditions are summarized in Table 4.1 in terms of the OpenFOAM key words.

Table 4.1
Boundary conditions in OpenFOAM for closed outlet.

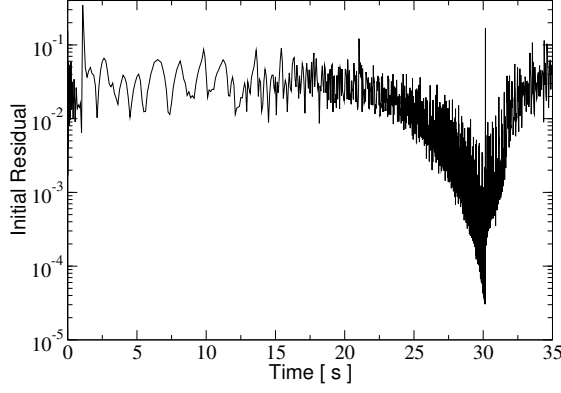
boundary	\mathbf{u}	P	\mathbf{w}
inlet	zeroGradient	zero totalPressure	zero fixedValue
outlet	zero fixedValue	zeroGradient	zero fixedValue
center line	empty	empty	zero fixedValue
upper wall	movingWallNormalVel C.3	zeroGradient	my dynamic mesh solver A.3.2 and B.2.2

4.1.3 Computational Details and Convergence Considerations

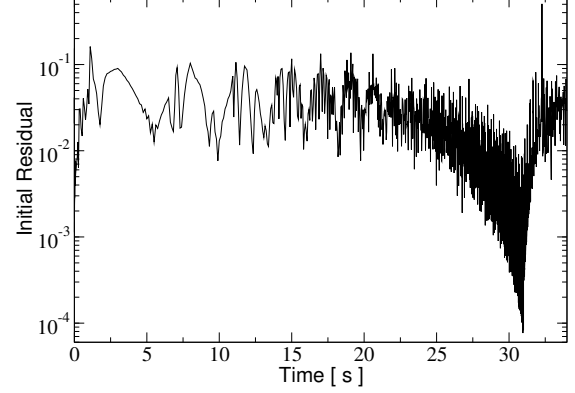
The simulations in this chapter have been performed with `transientSimpleDyMFoam`, the same OpenFOAM code as already described in Chapter 3 for the open pylorus cases. The data used for a representative computation case are listed in Appendix B. The simulations are carried out for five different Newtonian fluids, where the higher viscous fluid is obtained from the lower one by increasing the dynamic viscosity by a factor of 10. The fluid parameters used in this study are summarized in Table 3.2.

Three wave speeds of 1.15 mm/s, 2.3 mm/s and 4.6 mm/s have been used with two different maximum relative occlusions of 52% and 80%. These values were chosen to reflect characteristic data reported in literature, physical conditions, and to allow parameter study. In all simulations, the Courant number (Eq. 2.58) has been set to a value of 0.5 or less, and the absolute tolerances for the linear solvers within each pressure-velocity iteration have been set to 10^{-10} while the relative tolerances were set to zero. For the case of relative occlusion of 52%, the number of velocity-pressure iterates in each time step was set to 50 and the relaxation factor for the velocity was set to 0.7. Whereas for the case of relative occlusion of 80%, the number of velocity-pressure iterates in each time step was reduced to 3 and the relaxation factor for the velocity was decreased to 0.3 to provide stable convergence.

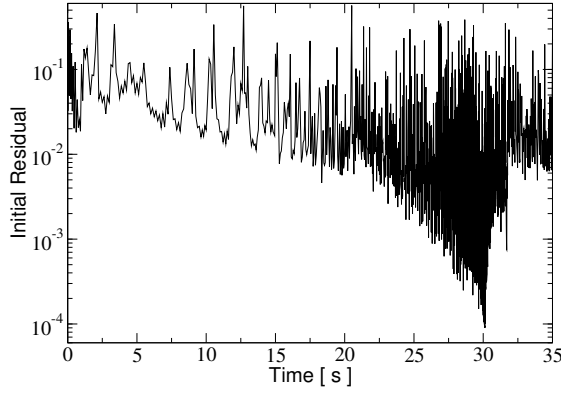
The convergence of the computations is illustrated by means of residual curves in Fig. 4.3 for the Newtonian fluid N3. These curves illustrate that the number of pressure-velocity iterations and the relaxation factors were sufficient to achieve adequate convergence in each time step.



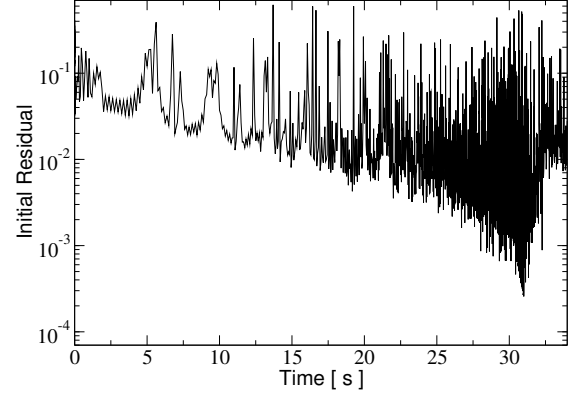
(a) Residual of the discrete x-momentum equation at the beginning of the last (i.e. 3rd) PISO iteration in each time step for the circular ACW.



(b) Residual of the discrete x-momentum equation at the beginning of the last (i.e. 3rd) PISO iteration in each time step for the parabolic ACW.



(c) Residual of the discrete pressure equation at the beginning of the last (i.e. 3rd) PISO iteration in each time step for the circular ACW.



(d) Residual of the discrete pressure equation at the beginning of the last (i.e. 3rd) PISO iteration in each time step for the parabolic ACW.

Figure 4.3: Convergence study

4.1.4 Mesh Independence Study

To make sure that the computational mesh exhibits a sufficient mesh resolution, a mesh dependence study has been performed for the Newtonian fluid N3 in the case of fastest wave speed and largest relative occlusion. This is presumably the worst case scenario where the largest gradients for velocity occur. Table 4.2 summarize the meshes that were used in this study, where the finer mesh is obtained from the coarser one by increasing the number of cells in the x and y-directions by a factor of 1.5.

Table 4.2
Computational mesh details for the axisymmetric conical simulations.

Mesh	Number of cells	Total number of cells	Cell length Δx (mm) for circular deformation	Cell length Δx (mm) for parabolic deformation	Cell height Δy (mm) near the pylorus
M2	$180 \times 12 \times 1$	2160	0.083	0.056	0.417
M3	$270 \times 18 \times 1$	4860	0.056	0.037	0.278
M4	$405 \times 27 \times 1$	10935	0.037	0.025	0.185

The results of the mesh dependence study along the center line for the x-component of velocity and strain rate are shown in Figs 4.4 and 4.5, respectively, in which the left and right vertical dashed lines represent the ends of the distal-most ACW. Figure 4.4 shows that the maximum magnitudes of the x-component of velocity along the center line are achieved near the wave crest. The negative values of the velocity x-component indicate the presence of a back-flow. It can be observed from this figure that the maximum magnitudes of the x-component of velocity for the parabolic ACW are larger than the ones for the circular ACW. It can be seen from this figure that the three different meshes give almost identical values of the velocity x-component away

from the wave crest. Near the wave crest these values differ considerably, however, the values from mesh M3 are closer to the ones obtained from mesh M4 than to the ones obtained from mesh M2, which indicates mesh convergence.

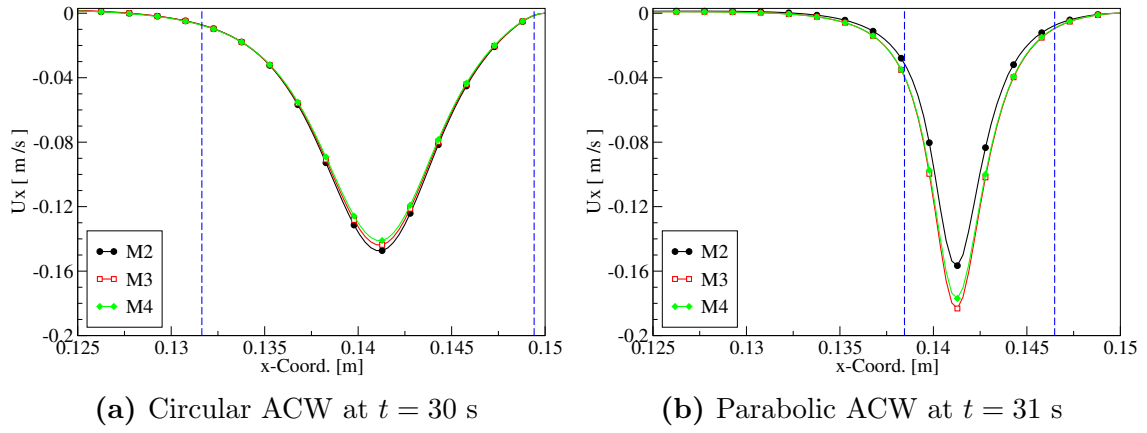


Figure 4.4: Mesh dependence study of the x-component of velocity for the Newtonian fluid N3 along the center line. The wave speed and relative occlusion are 4.6 mm/s and 80%, respectively.

These results are consistent and qualitatively similar to the strain rate results given in Fig. 4.5. From these considerations it can be concluded that mesh M3 exhibits sufficient mesh independence. Therefore, mesh M3 is used as the standard mesh for all the subsequent simulations in this chapter.

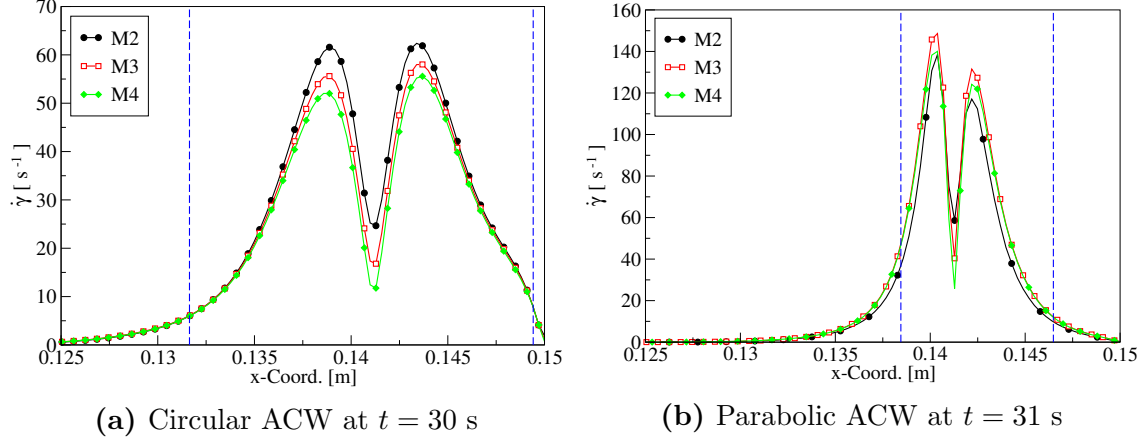


Figure 4.5: Mesh dependence study of the strain rate for the Newtonian fluid N3 along the center line. The wave speed and relative occlusion are 4.6 mm/s and 80%, respectively.

This computational mesh consists of one structured hexahedral Cartesian block with a non-uniform cell distribution in the x-direction, where the smallest cells are at the pylorus. The cell grading is controlled by a geometric distribution whose ratio between the smallest and the largest cells is 0.1 for the circular wave and 0.15 for the parabolic wave. Therefore, the undeformed conical standard mesh had the smallest cells near the pylorus with lengths of 0.056 mm and 0.037 mm for the circular and parabolic deformation, respectively. The cell size distribution in the vertical direction of the undeformed mesh is uniform with the smallest cell height of 0.278 mm at the pylorus. For the deformed mesh, the smallest cells occur under the wave at the maximum occlusion of 80% and have a cell height of 0.11 mm.

4.2 Results and Discussion

In this study the mixing process in a stomach is investigated for various fluid properties and for different geometric parameters. More precisely, the effect of fluid viscosity, the shear-thinning of non-Newtonian fluids, the ACW speeds, the ACW shapes and the relative maximum occlusions are studied.

4.2.1 Variation of the Newtonian Fluid

To explore the sensitivity of flow patterns with respect to variations in fluid viscosity, separate simulations have been performed for five different Newtonian fluids, whose parameters are listed in Table 3.2, for the case of ACWs speed of 2.3 mm/s. The result of these simulations are shown in Figs 4.6 – 4.21.

By propagating ACWs toward the pylorus, the simulations predicted two basic antral flow patterns, at a time when the pylorus is closed. A reverse jet-like pulse (retropulsive jet) developed in the most highly occluded region near the pylorus, and a recirculating flow patterns (eddies) between and under the ACWs crests (Fig. 4.6), both of which contribute to food disintegration and mixing. The same observations have been experimentally reported by Code [129], Keinke et al. [49], Schulze–Delrieu and Brown [130], Brasseur et al. [131], Li et al. [132], Pallotta et al. [52] and Boulby et

al. [53], and numerically by Pal et al. [68], Ferrua and Singh [3], and Imai et al. [48].

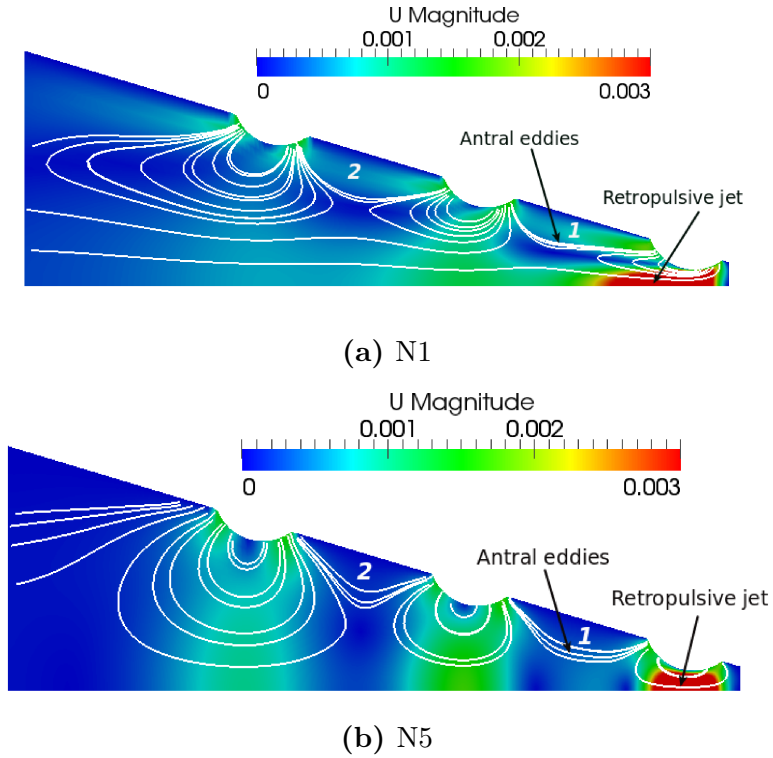


Figure 4.6: Streamlines of the fluid flow within the lower part of stomach at $t = 60$ s, colored by velocity magnitude (m/s). Maximum relative occlusion of 52% for the distal-most circular ACW is applied.

In this study, the strength of the retropulsive jet is quantified by the velocity magnitude $|\mathbf{U}|$ (or by the magnitude of the x-component of velocity for the distal-most ACW along the center line). Figure 4.6 shows that higher retropulsive jet velocities are predicted at the locations of the ACWs crests, reaching the maximum value near the pylorus and along the center line. By contrast, an immediate decay of the retropulsive jet has been identified in a region away from the ACWs. This decay may be explained by the improved diffusion of viscous effects together with the presence of

antrum wall between consecutive ACWs (Ferrua and Singh [3]). Moreover, this figure shows that by increasing the dynamic viscosity of the fluid to 10 Pa.s, the retropulsive jet is confined to a smaller region at the core of the luminal antrum, with a jet length shorter than the one for the case of the lowest viscous fluid N1.

Figure 4.7 gives a color plot of the velocity vector for fluid N3 at a time when the strongest retropulsive jet is achieved for the case of maximum relative occlusion of 52%. These velocity vectors show that, the strongest fluid motions are obtained in the most occluded part of the canal, with directions indicating the presence of a back-flow.

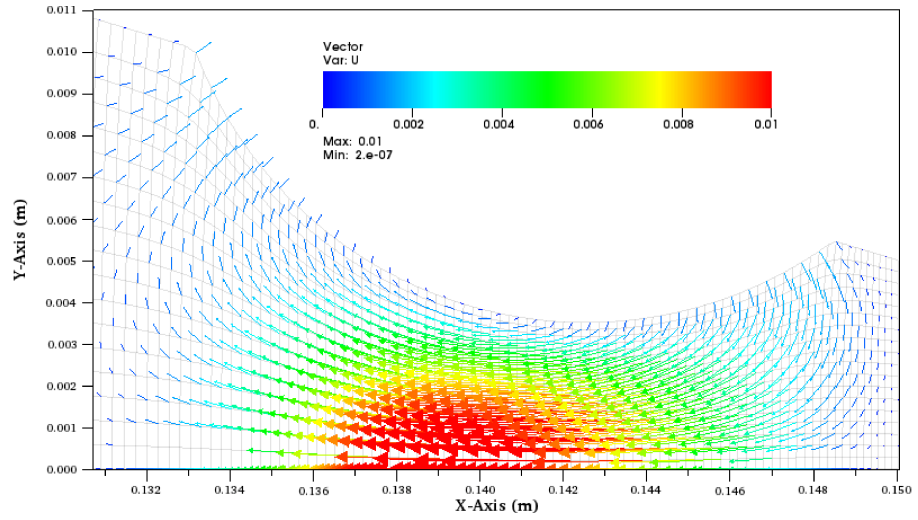


Figure 4.7: The velocity vectors (m/s) of the Newtonian fluid N3 at $t = 60$ s in the most occluded section of the pylorus canal. The wave speed is 2.3 mm/s and the maximum relative occlusion is 52%.

This back-flow is consistent with the negative values of x-component of velocity along the center line as is shown in Figs 4.8 and 4.9. These figures show that all fluids have a back-flow, and this back-flow increases in magnitude as the ACWs approach the pylorus sphincter.

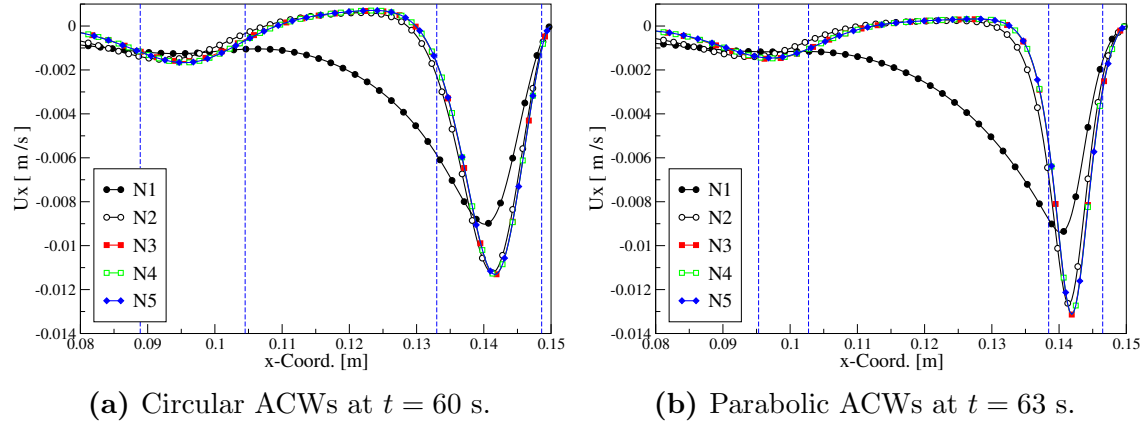


Figure 4.8: The x-component of velocity for five Newtonian fluids along the center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively.

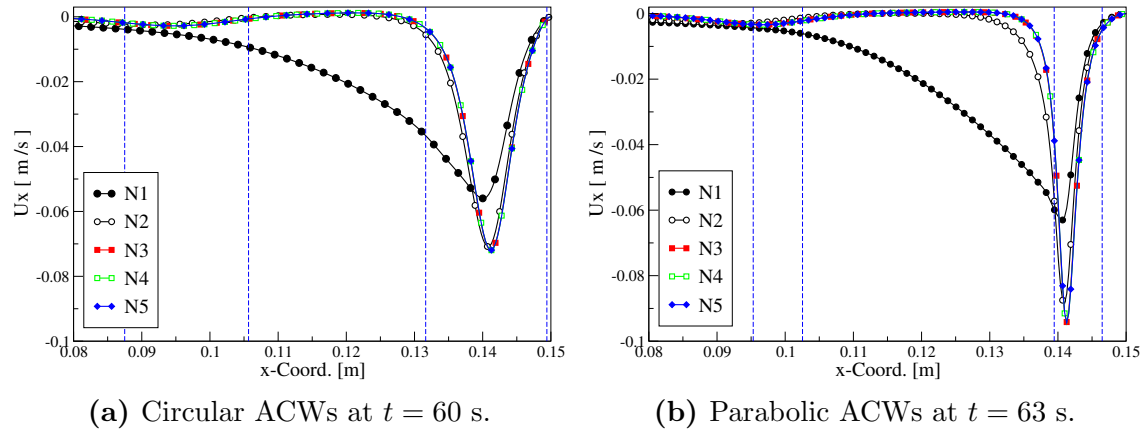


Figure 4.9: The x-component of velocity for five Newtonian fluids along the center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively.

Moreover, the higher viscous fluids behave the same in a location away from the pylorus, while near the pylorus the strength of the retropulsive jet, as well as the magnitude of the back-flow, increases with fluid viscosity. Note that the maximum retropulsive jets for the largest three viscosities appear nearly identical.

Higher viscous fluids exhibit locally more intensive retropulsive jet at the pylorus with strength much larger than the ACWs speed. This jet constitutes the strongest mechanical forces for grinding and breakdown of solid particles and mixing of gastric content.

The strength of the retropulsive jet is relatively insensitive to the ACWs shape and the same qualitative behavior for the retropulsive jet is seen for larger and smaller relative occlusion (Figs 4.8 and 4.9). In particular, the ACWs shape has relatively little influence on low-viscous fluids but larger influence on high-viscous ones, with a slightly stronger retropulsive jet for the parabolic ACWs. Furthermore, the retropulsive jet for 80% relative occlusion is about 6 – 7 times stronger than the one for the case of 52% relative occlusion. Thus, larger relative occlusion leads to much stronger retropulsive jets and hence enhanced mixing and food disintegration.

The strength of the eddy structures was quantified by the volume averaged vorticity magnitude $|\omega_{\text{avg}}|$ (that is, the average rotational motion within the eddy between two consecutive ACWs), while the formation of the eddies was captured by the development the streamlines (lines drawn tangent to velocity vectors) and vorticity contours of the gastric fluid flow at a fixed time.

Similar to the retropulsive jet, the strength and formation of eddies were also sensitive to fluid viscosity as follows: A global recirculation along the center line of the domain for fluid N1 and a more local recirculation between consecutive ACWs for fluid N5 have been observed (Fig. 4.6), both of which account for global and local mixing. A lower and more uniform vorticity field that confined to regions close to the ACWs has been identified for fluid N5 (Figs 4.10) and 4.11. These results are consistent with

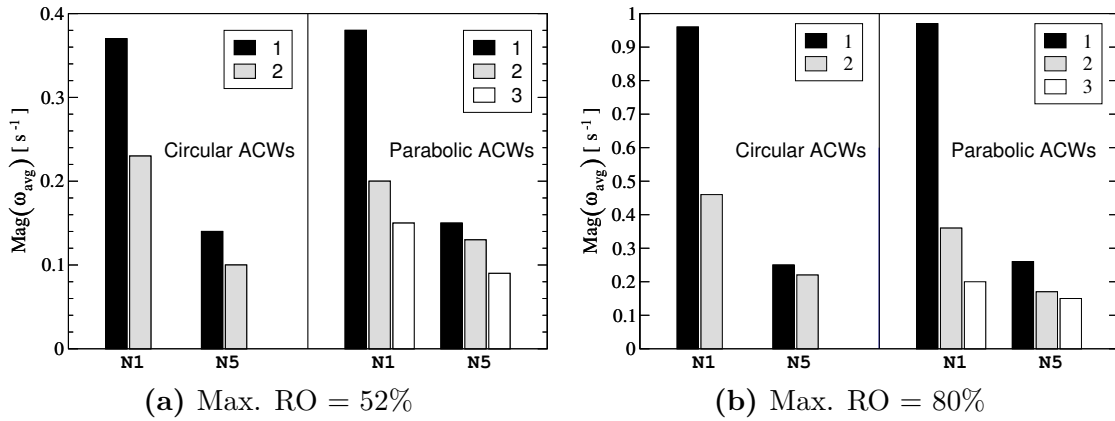
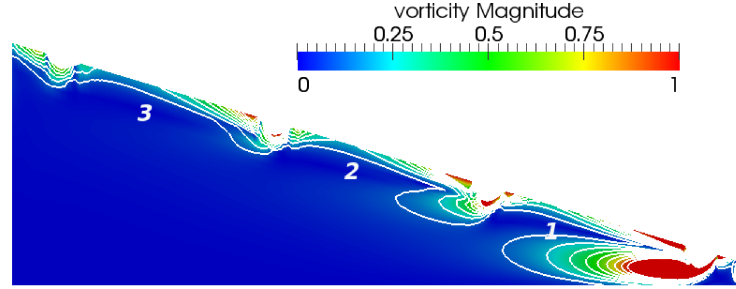


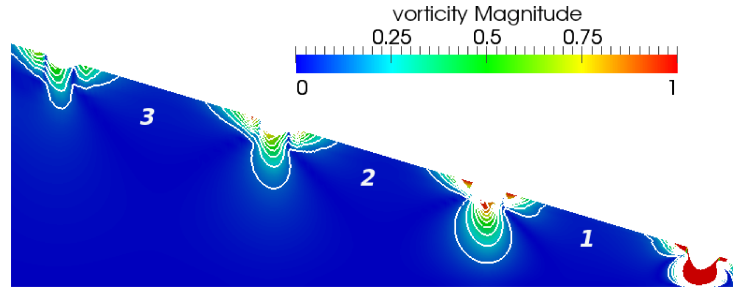
Figure 4.10: The effect of fluid viscosity on the average value of vorticity field. The wave speed is 2.3 mm/s and two maximum relative occlusions of 52% and 80% were achieved for the distal-most ACW. The shading bars are consistent with numbers labeled on graphs 4.6 and 4.11.

numerical and experimental observations previously reported in literature (Marciani et al. [133] and Ferrua and Singh [3]), suggest that a gastric content associated with high viscous meals seems to be poorly mixed. Akin to the observations of Pal et al. [68], Fig. 4.10 shows that as the ACW propagates toward the pylorus, its associated eddies strengthen and reach a maximum magnitude of average vorticity in the most occluded section of the stomach.

As for the retropulsive jet, the strength of the antral eddies are relatively insensitive to the ACW shape and increase with relative occlusion.



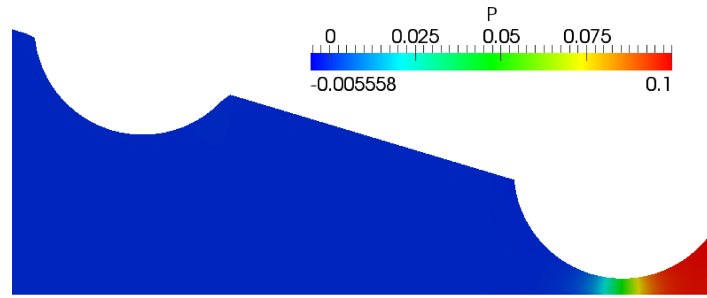
(a) N1



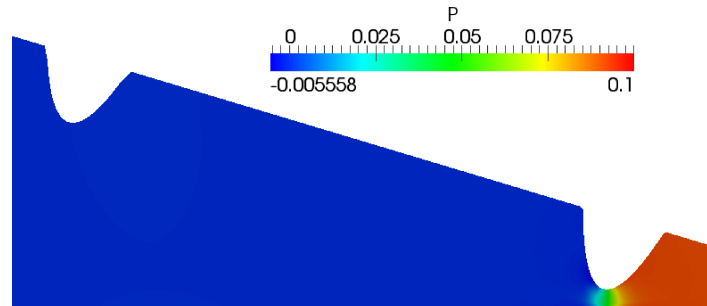
(b) N5

Figure 4.11: Contour of vorticity within the lower part of stomach at $t = 63$ s, colored by vorticity magnitude (1/s). Maximum relative occlusion of 52% for the distal-most parabolic ACW is applied.

Figure 4.12 shows that, as the ACW approaches the pylorus sphincter, the kinetic pressure field significantly increases, reaching a maximum value in a region close to the propagation of the distal-most ACW and pylorus. This is consistent with the fact that the ACW induces large pressure values in front of and under-pressure values behind the ACW. Observations previously reported by Zuckerman and Lior [134, 135] suggest that as antral flow approaches the closed pylorus, it behaves like an impinging jet flow and hence a higher pressure is induced at the pylorus. These pressure build-ups are then converted into kinetic energy when the fluid is reflected at the pylorus. This is the source of the retropulsive jets, as is discussed below in more details.



(a) Circular ACWs at $t = 60$ s



(b) Parabolic ACWs at $t = 63$ s

Figure 4.12: Kinematic pressure (m^2/s^2) of the Newtonian fluid N3. The wave speed is 2.3 mm/s and maximum relative occlusions of 80% is achieved for the distal-most ACW.

These results were consistent with pressure field behavior along the center line for five Newtonian viscosities, as is shown in Figs 4.13 and 4.14. These figures show that pressure field strengthened with viscosity and relative occlusion, reaching a maximum value of about 16 times stronger for the largest viscous fluids in a region under the distal-most ACW and close to the pylorus sphincter. In contrast, these figures show that the ACW shape has only a small influence on the pressure field near the pylorus.

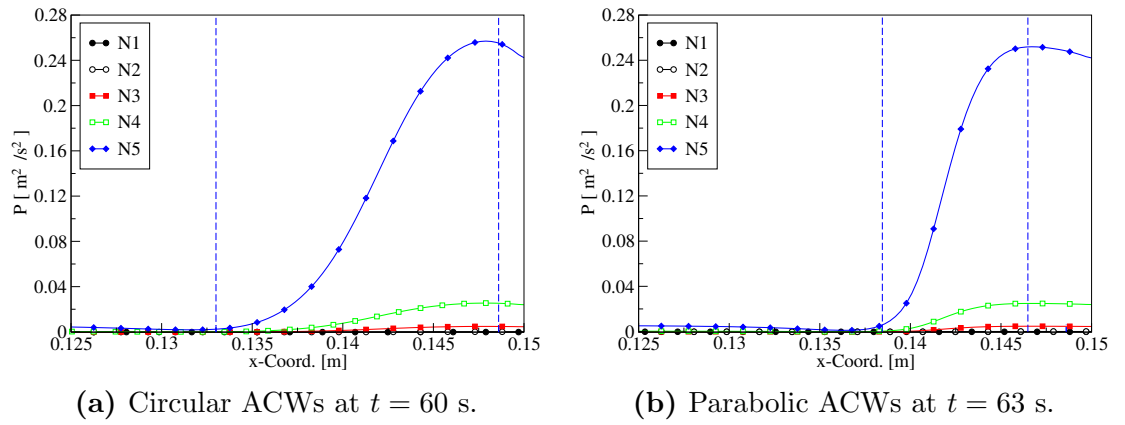


Figure 4.13: Kinematic pressure for five different Newtonian fluids along the center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively.

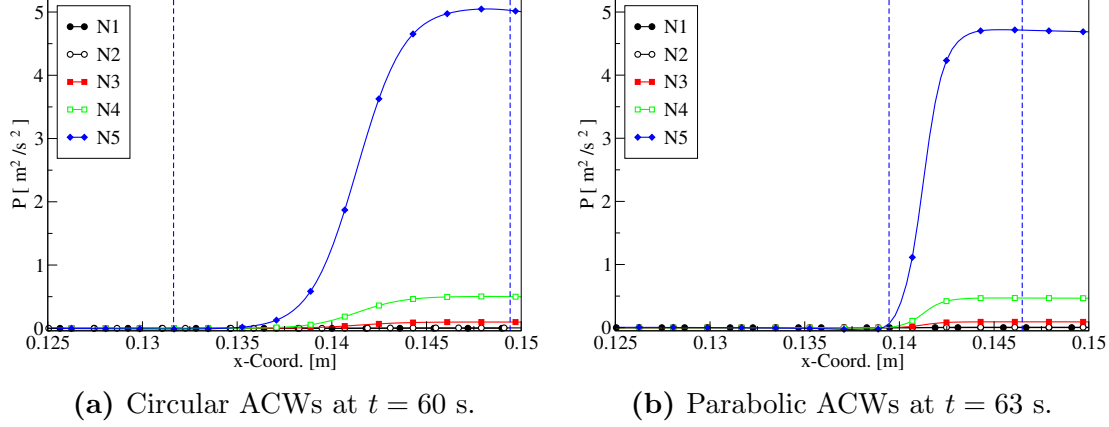


Figure 4.14: Kinematic pressure for five different Newtonian fluids along the center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively.

However, as is illustrated in Figs 4.15 and 4.16, the maximal pressure gradients for the parabolic waves are considerably higher than for the circular waves. The reason for this is that the same pressure change for the parabolic wave occurs over a much shorter distance than for the circular wave. This distance is illustrated in the figures with the vertical dashed lines which signify the beginning and the end of the ACW. The x-component of pressure gradient field along the center line are illustrated in Figs 4.15 and 4.16 for the circular and parabolic wave shapes with maximum relative occlusions of 52% and 80%.

These figures illustrate the mechanism which causes the retropulsive jet: The flow induced by the ACWs builds up a high pressure near the pylorus, which then is converted into kinetic energy when the flow is reflected from the pylorus wall. Since higher viscous fluids build up a larger pressure, the corresponding retropulsive jet is locally stronger. These results are consistent with experimental and simulation data

from the literature (Feinle et al. [136], Choe et al. [137], Simonian et al. [138], Pal et al. [68], and Ferrua and Singh [3]), which suggest a critical role of the pressure gradient field in improving digestion process of gastric contents associated with high viscous meals.

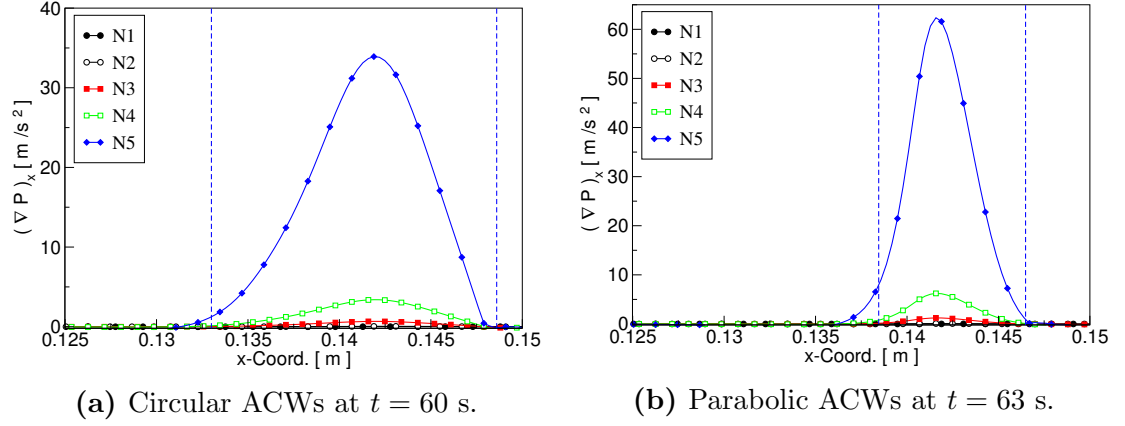


Figure 4.15: The x-component of kinematic pressure gradient for five different Newtonian fluids along the center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively.

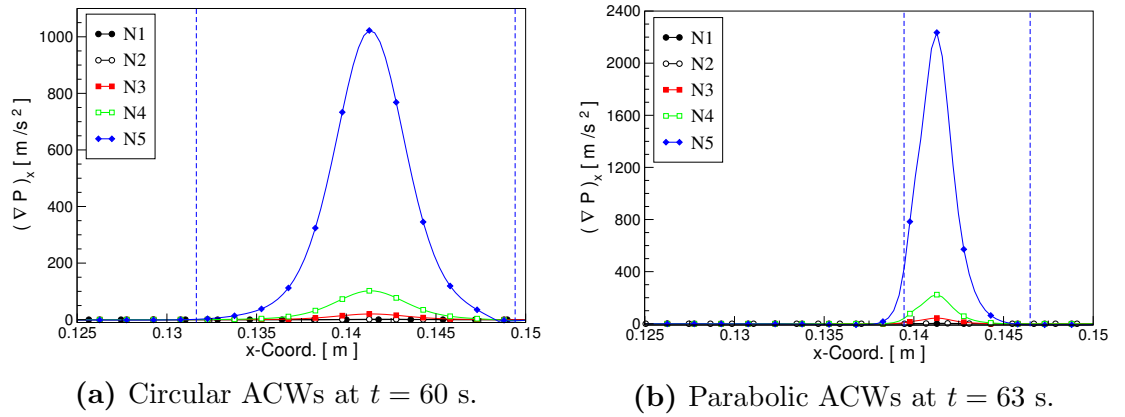
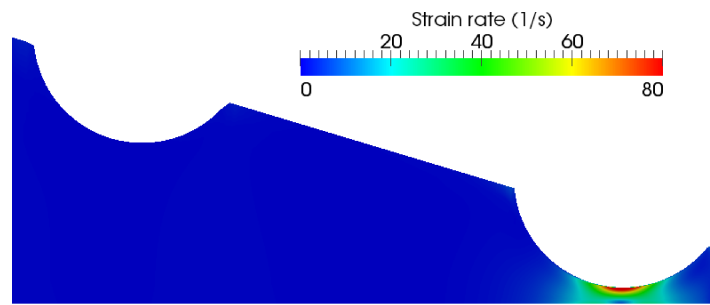
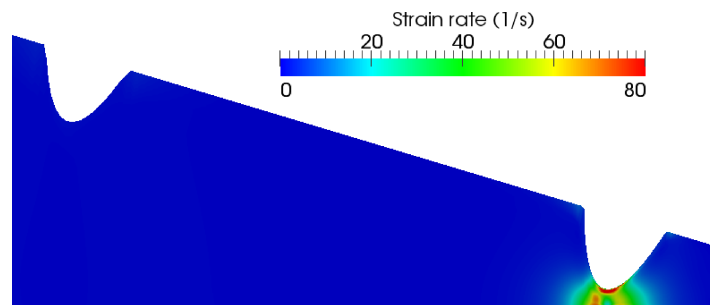


Figure 4.16: The x-component of kinematic pressure gradient for five different Newtonian fluids along the center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively.

Figure 4.17 is a color plot of the strain rates of Newtonian fluid N3 at a time when strongest retropulsive flow occurs. This figure shows that the largest values of strain rates are achieved in the distal antrum in the region of direct contact with the ACW and just under the ACW. Moreover, as the ACW approaches the pylorus sphincter, the increasing occlusion of the ACW strengthened the values of strain rates, reaching maximum values in the most occluded section of the pylorus canal. This result may be explained by the increased action of the retropulsive jet at the core of the luminal region, both of which enhance breakdown and mixing of food near the pylorus.



(a) Circular ACWs at $t = 60$ s



(b) Parabolic ACWs at $t = 63$ s

Figure 4.17: Strain rate of the Newtonian fluid N3. The wave speed is 2.3 mm/s and maximum relative occlusions of 80% is achieved for the distal-most ACW.

The above observations are consistent with the strain rate fields computed along the center line for five Newtonian viscosities, as is shown in Figs 4.18 and 4.19.

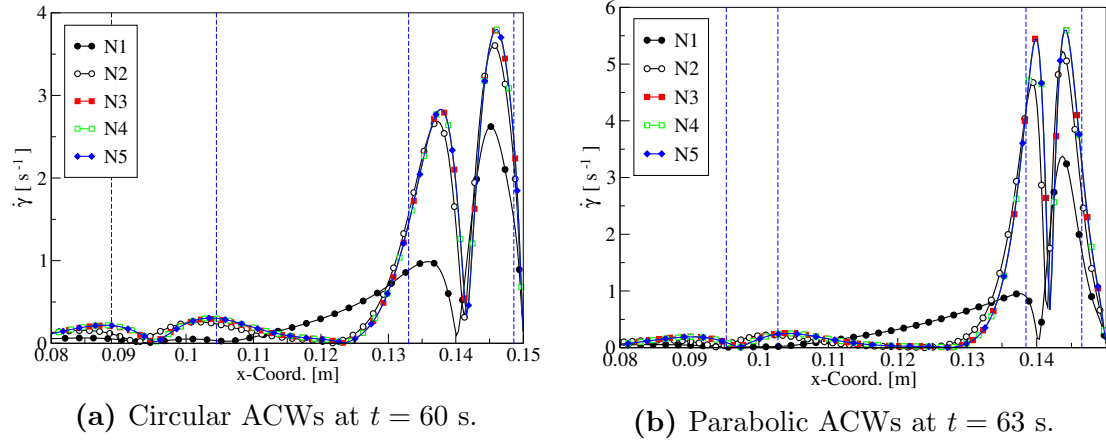


Figure 4.18: Strain rate for five different Newtonian fluids along the center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively.

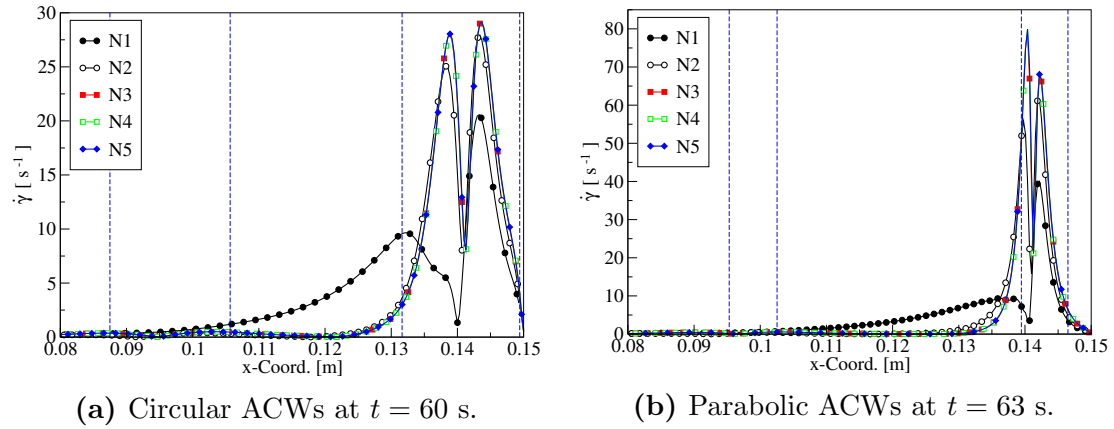


Figure 4.19: Strain rate for five different Newtonian fluids along the center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively.

These figures show that, all fluids behave almost the same in a region away from the

pylorus, while near the pylorus the strain rate field strengthens with viscosity and relative occlusion, where the largest three viscosities seem to produce nearly identical results. Also, note that the strain rate field associated with circular ACW is weaker than that associated with parabolic ACW.

The effect of the strain rate on the digestion process is expressed by the viscous stress acting on the material. Figures 4.20 and 4.21 show the magnitude of the viscous stress tensor for the five Newtonian fluids N1 to N5 along the center line for circular and parabolic ACWs with velocity 2.3 mm/s at the respective relative occlusions of 52% and 80%. In like manner as discussed previously, these forces can be attributed to the mechanical disintegration of high viscous meals near the pylorus, strengthen with viscosity and relative occlusion, and become more intensive for parabolic ACW.

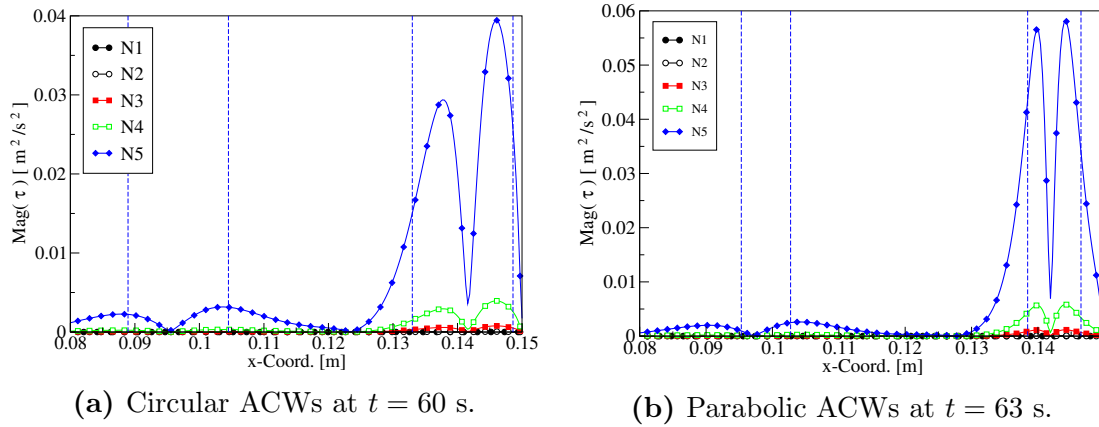


Figure 4.20: The magnitude of kinematic viscous stress tensor for five different Newtonian fluids along the center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively.

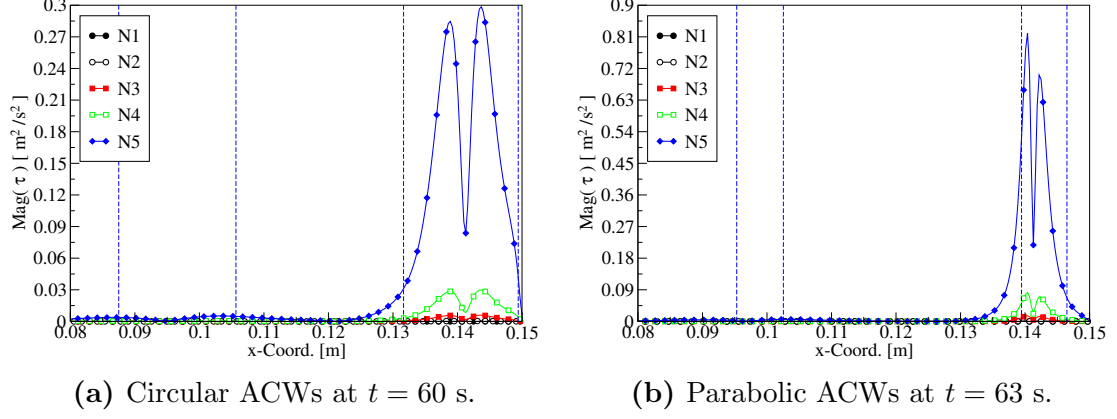


Figure 4.21: The magnitude of kinematic viscous stress tensor for five different Newtonian fluids along the center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively.

4.2.2 Effect of Shear-Thinning Behavior

To identify the effect of the shear-thinning non-Newtonian behavior on fluid motions and mixing during the digestion process within the antrum, several simulations have been carried out for one Newtonian fluid and two non-Newtonian fluids, for the ACW speed of 2.3 mm/s. The results are given in Figs 4.22 – 4.31.

The Bird-Carreau Eq. (2.17) expresses the shear rate dependent viscosity η for these two non-Newtonian fluids, BCA and BCB, with the later is exhibiting considerably more shear-thinning behavior. The fluids parameters are given in Table 3.8 and the viscosity curves are depicted in Fig. 3.17. Note that, the zero shear rate viscosity η_0 for the non-Newtonian fluids is the constant viscosity used for the Newtonian fluid N3. Moreover, the shear-thinning for both non-Newtonian fluids starts at a strain rate of $\dot{\gamma} = \frac{1}{k} = 0.05 \text{ s}^{-1}$.

The simulations show that, as the ACW approaches the pylorus sphincter, the increasing occlusion of the ACW increases the physical quantity faster, reaching the maximum in the most occluded section of the antrum. Also, all fluids behave nearly the same in a location away from the pylorus, while near the pylorus the value of physical quantity weakened with power-law index. Moreover, the maximum magnitude of physical quantity is stronger and more intensive for parabolic ACW. This is an ubiquitous feature of all physical quantities, \mathbf{U}_x , P , $(\nabla P)_x$, $\dot{\gamma}$ and $|\tau|$, at a time when strongest retropulsive jet is achieved on closed pylorus.

Specifically, the negative velocities under the ACWs crests (Figs 4.22 and 4.23) indicate the presence of a back-flow. This back-flow increases in magnitude with the power-law index, reaching maximum value under the distal-most ACW and along the center line for the Newtonian fluid N3. Further, the influence of the ACW shape on the pressure field is almost negligible (Figs 4.24 and 4.25), where the maximum values of pressure are achieved in the direction of the distal-most ACW motion for the least shear-thinning fluids. In other words, N3 reaches a maximum value of about 20 times larger at 80% relative occlusion than that at 52% relative occlusion.

These observations are in good agreement with theoretical results, suggest a smaller viscosities of the shear-thinning fluid result in a smaller pressure drop in a channel or tube (Figs 4.26 and 4.27). In particular, these figures show that axial pressure gradient is stronger for parabolic ACW than circular ACW by nearly a factor of 2. This has been explained in the previous section. It has to do that the same pressure

drop occurs over a much shorter distance. Finally, a little effect of power-law index for 52% relative occlusion on the maximum strain rate values (Fig. 4.28), while higher strain rates are obtained with increasing power-law index (Fig. 4.29).

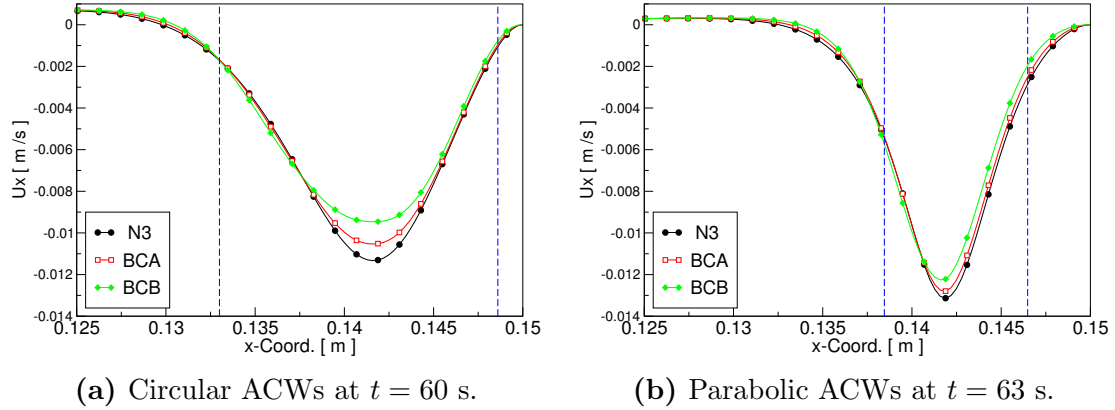


Figure 4.22: The x-component of the velocity of the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively.

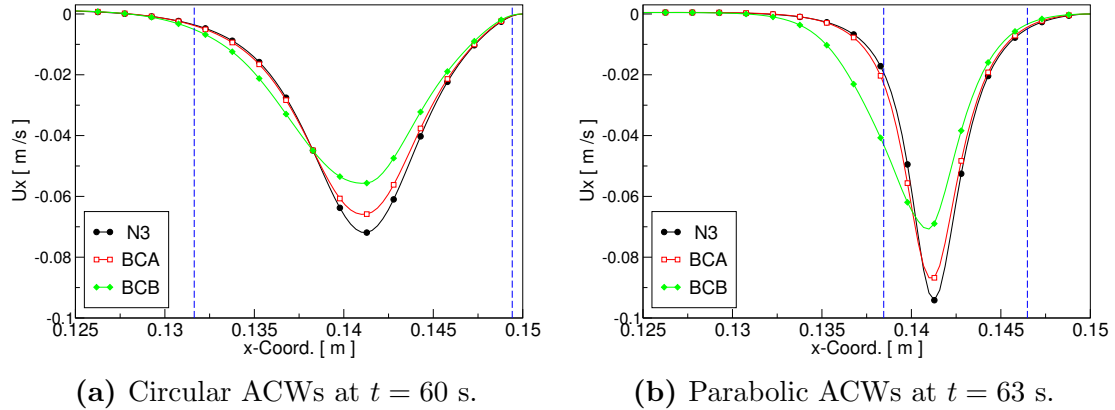
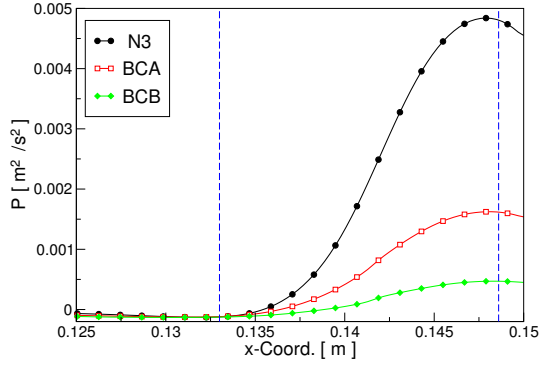
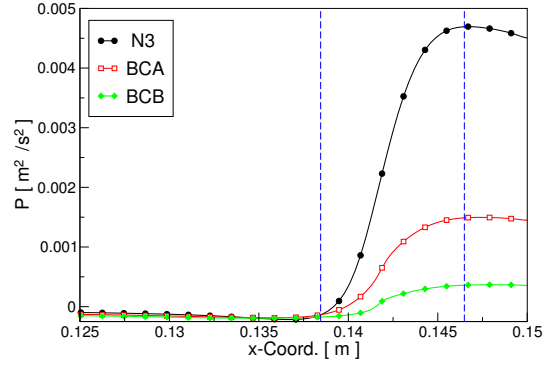


Figure 4.23: The x-component of the velocity of the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively.

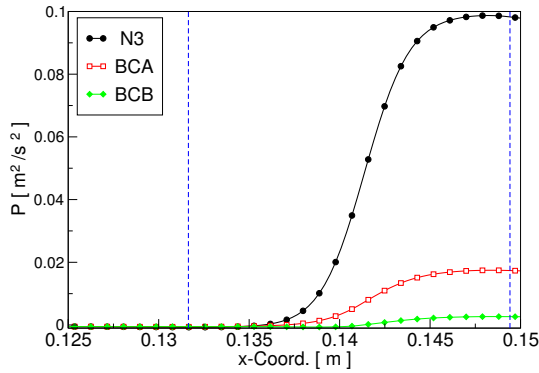


(a) Circular ACWs at $t = 60$ s.

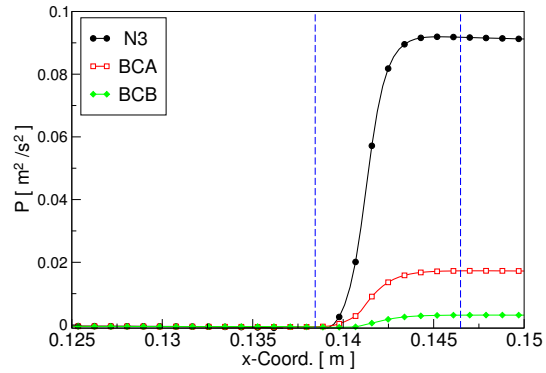


(b) Parabolic ACWs at $t = 63$ s.

Figure 4.24: Kinematic pressure of the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively.

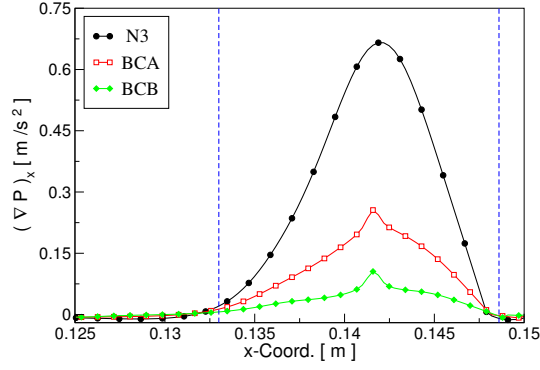


(a) Circular ACWs at $t = 60$ s.

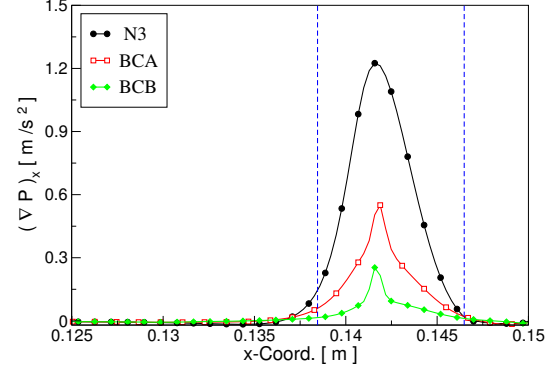


(b) Parabolic ACWs at $t = 63$ s.

Figure 4.25: Kinematic pressure of the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively.

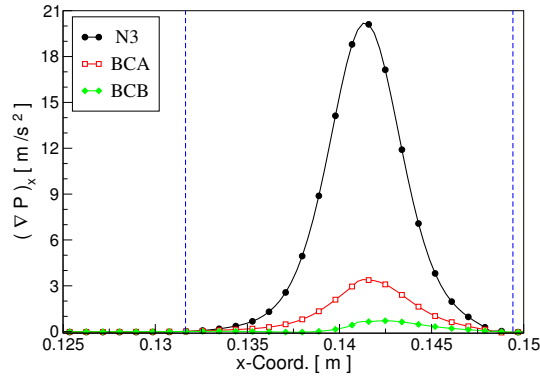


(a) Circular ACWs at $t = 60$ s.

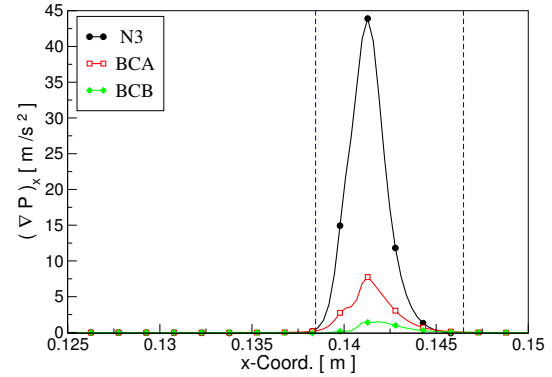


(b) Parabolic ACWs at $t = 63$ s.

Figure 4.26: The x-component of kinematic pressure gradient for the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively.

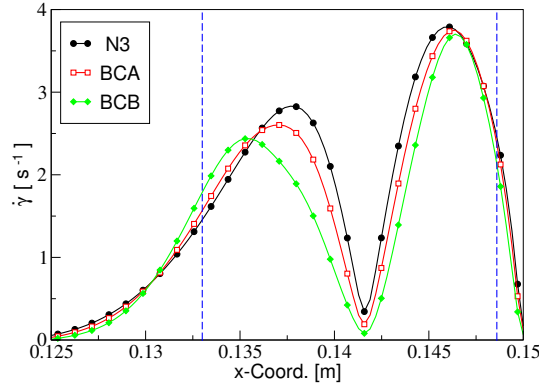


(a) Circular ACWs at $t = 60$ s.

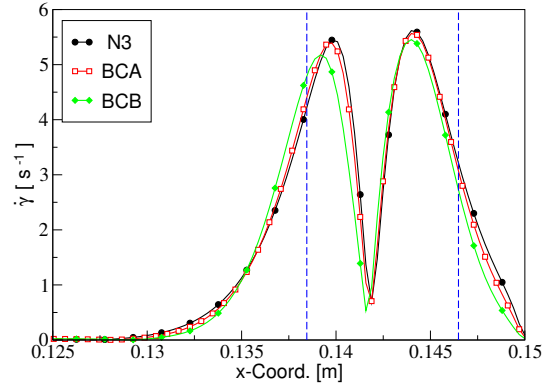


(b) Parabolic ACWs at $t = 63$ s.

Figure 4.27: The x-component of kinematic pressure gradient for the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively.

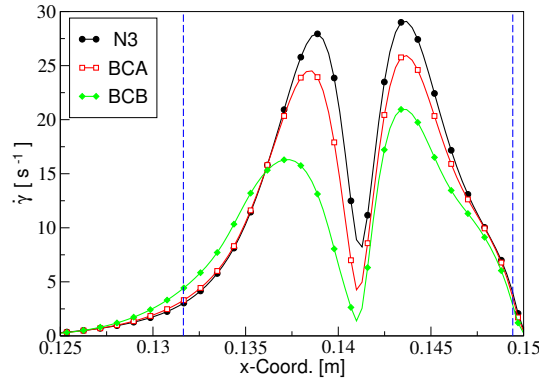


(a) Circular ACWs at $t = 60$ s.

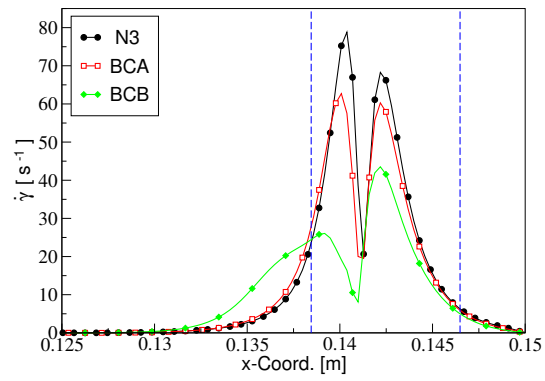


(b) Parabolic ACWs at $t = 63$ s.

Figure 4.28: Strain rate of the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively.

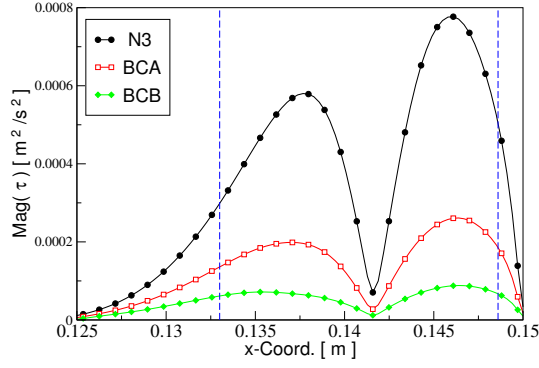


(a) Circular ACWs at $t = 60$ s.

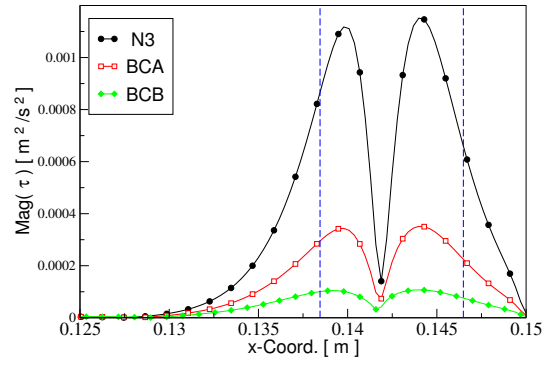


(b) Parabolic ACWs at $t = 63$ s.

Figure 4.29: Strain rate of the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively.

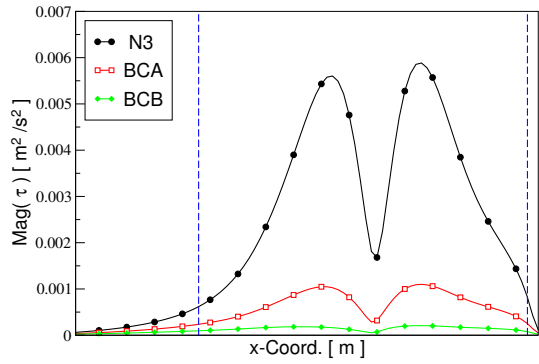


(a) Circular ACWs at $t = 60$ s.

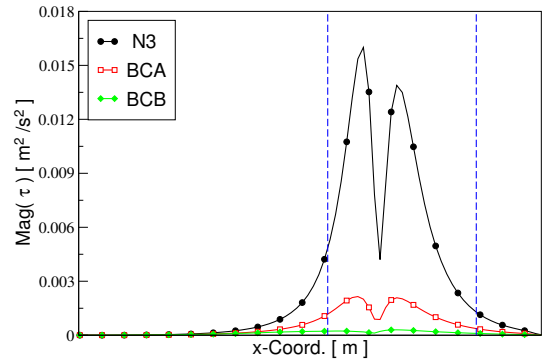


(b) Parabolic ACWs at $t = 63$ s.

Figure 4.30: The magnitude of kinematic viscous stress tensor of the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 52%, respectively.



(a) Circular ACWs at $t = 60$ s.



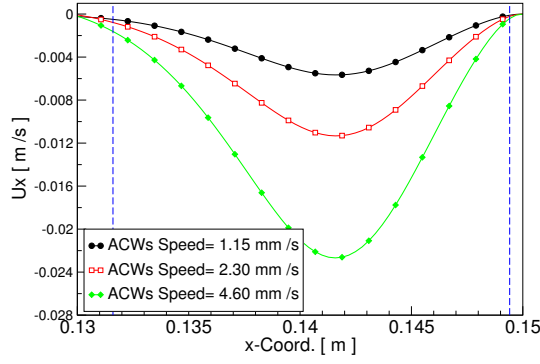
(b) Parabolic ACWs at $t = 63$ s.

Figure 4.31: The magnitude of kinematic viscous stress tensor of the Newtonian and non-Newtonian fluids along center line. The wave speed and the maximum relative occlusion are 2.3 mm/s and 80%, respectively.

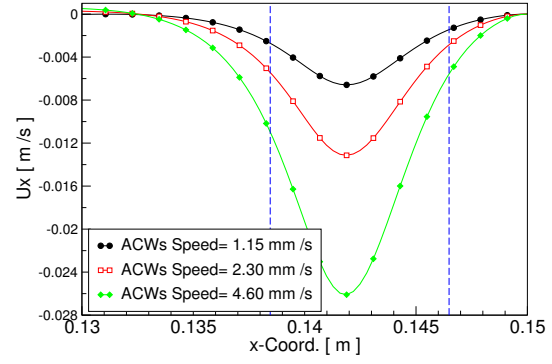
4.2.3 Variation of Wave Speed

To explore sensitivity in flow patterns within the lower part of a human stomach with respect to the ACW speed variations, separate simulations were carried out for one Newtonian fluid N3, with the ACW speed increased and reduced by a factor of two from the standard case. The results are shown in Figs 4.32 – 4.41. These figures show that, all physical quantities increase with relative occlusion of the ACW and increase (almost) linearly with the ACW speed, reaching the maximum magnitude at the core of the luminal region.

In particular, Figs 4.32 and 4.33 show the presence of a back-flow for all different speeds and relative occlusions of the ACW. This back-flow increases linearly in magnitude with the ACW speed and strengthens in magnitude with the relative occlusion, and hence attribute to more food mixing and disintegration. Moreover, retropulsive jet and antral pressure field are relatively insensitive to the ACW shape, with a little advantage for the parabolic ACWs. The parabolic ACW is observed to produce much stronger pressure gradients, strain rates and stress viscous forces near the pylorus. Thus, higher relative occlusion and parabolic shape of the ACWs improve gastric fluid motions and promote mixing.

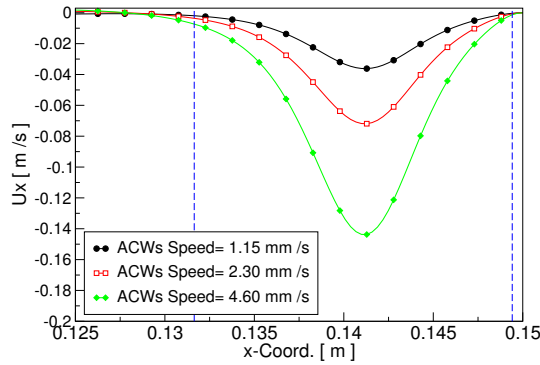


(a) Circular ACWs.

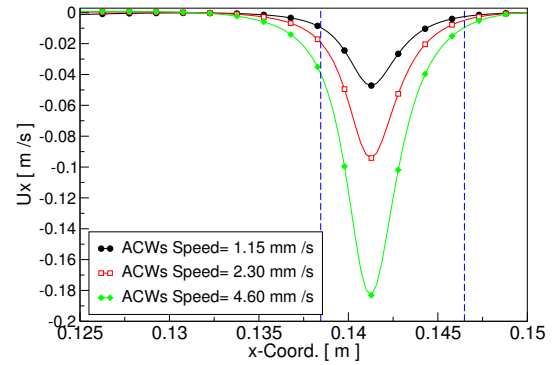


(b) Parabolic ACWs.

Figure 4.32: The x-component of velocity of the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 52% are applied.

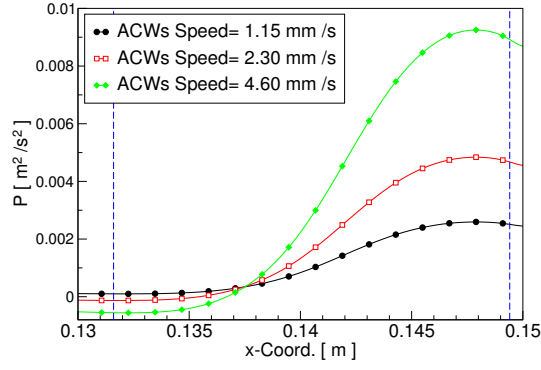


(a) Circular ACWs.

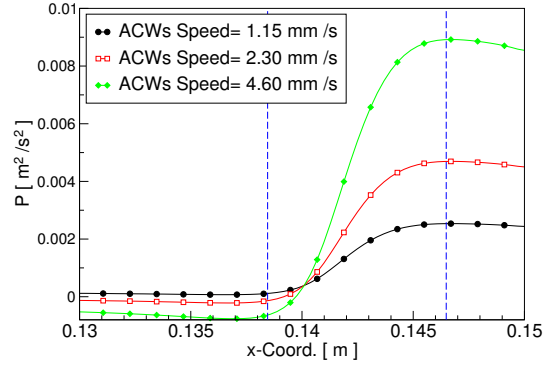


(b) Parabolic ACWs.

Figure 4.33: The x-component of velocity of the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 80% are applied.

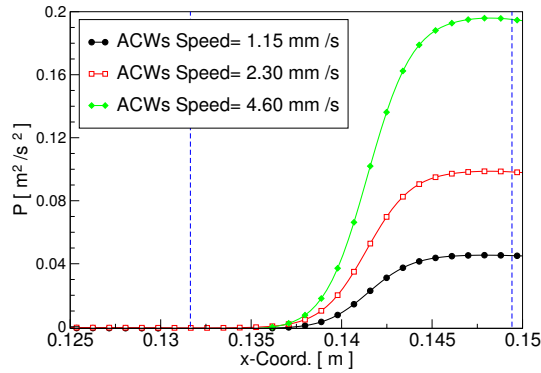


(a) Circular ACWs.

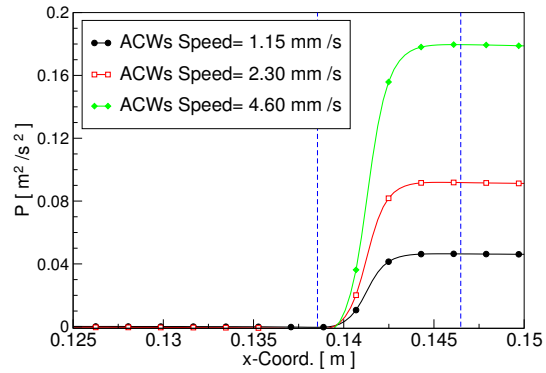


(b) Parabolic ACWs.

Figure 4.34: Kinematic pressure of the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 52% are applied.



(a) Circular ACWs.



(b) Parabolic ACWs.

Figure 4.35: Kinematic pressure of the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 80% are applied.

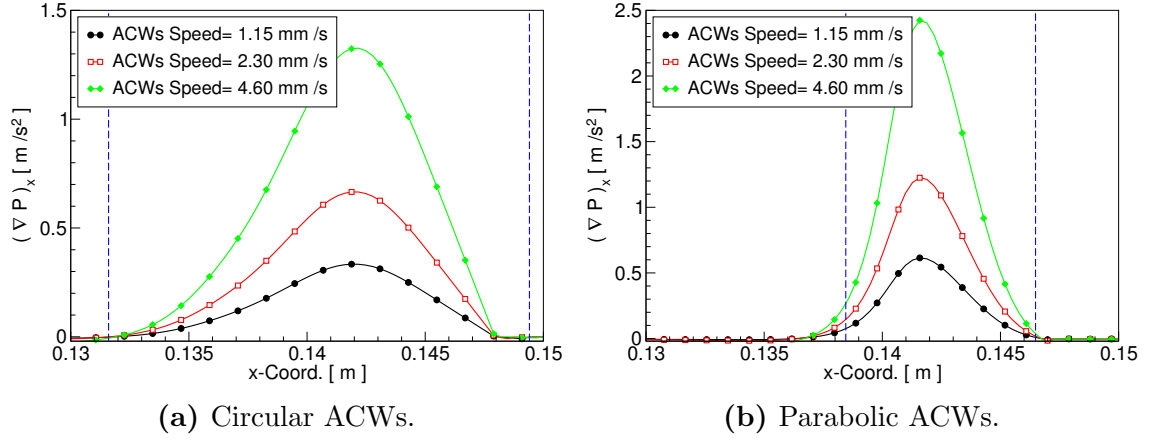


Figure 4.36: The x-component of kinematic pressure gradient for the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 52% are applied.

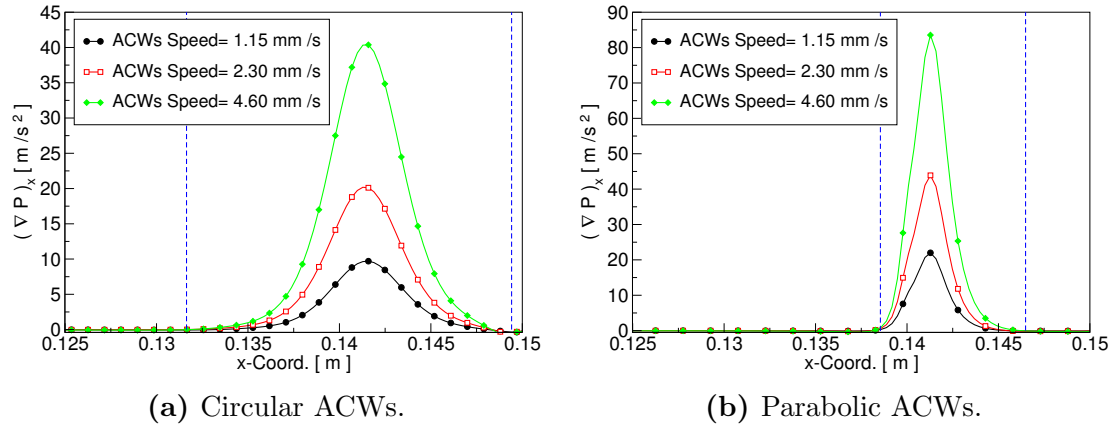
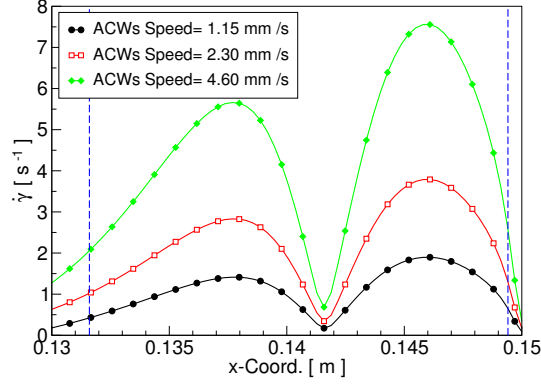
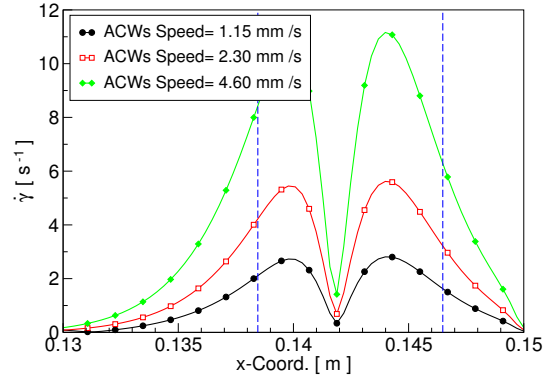


Figure 4.37: The x-component of kinematic pressure gradient for the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 80% are applied.

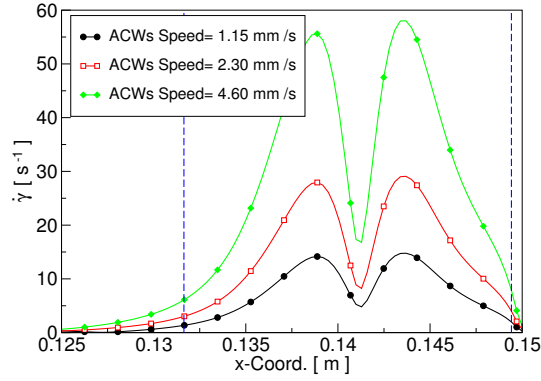


(a) Circular ACWs.

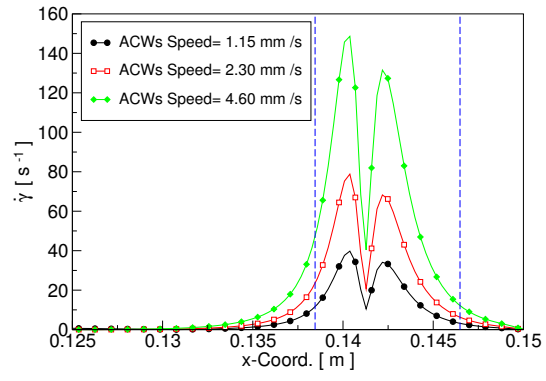


(b) Parabolic ACWs.

Figure 4.38: Strain rate of the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 52% are applied.

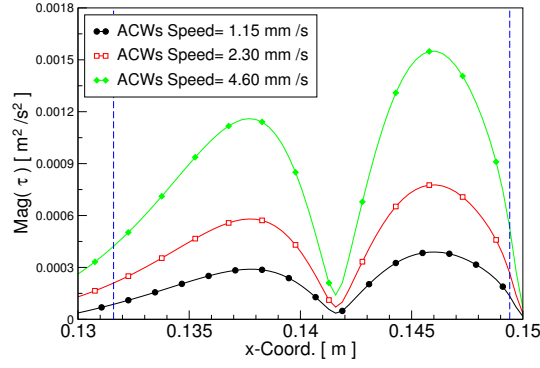


(a) Circular ACWs.

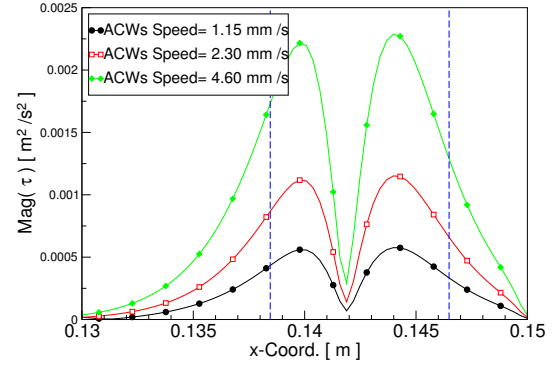


(b) Parabolic ACWs.

Figure 4.39: Strain rate of the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 80% are applied.

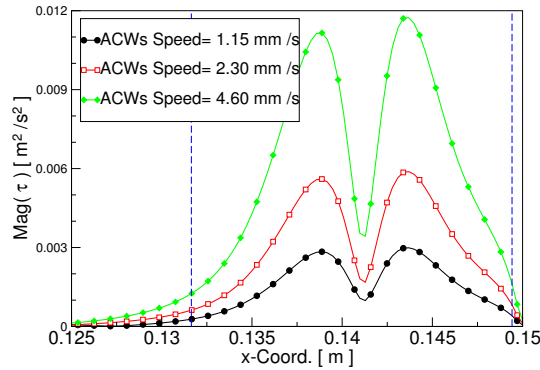


(a) Circular ACWs.

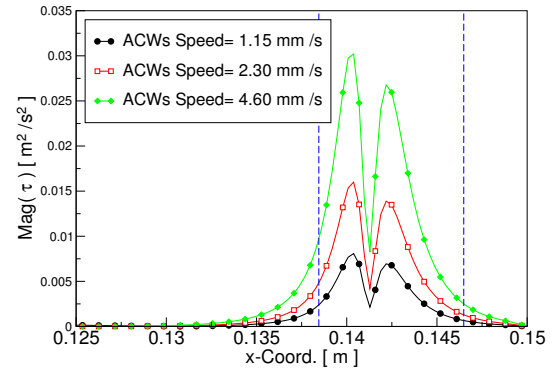


(b) Parabolic ACWs.

Figure 4.40: The magnitude of kinematic viscous stress tensor of the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 52% are applied.



(a) Circular ACWs.



(b) Parabolic ACWs.

Figure 4.41: The magnitude of kinematic viscous stress tensor of the Newtonian fluid N3 along the center line. Three different ACWs speeds and a maximum relative occlusion of 80% are applied.

4.3 Food Mixing and Particle Tracking Technique

In agreement with classical description of the stomach function, Pal et al. [68] suggested that the principle region of gastric mixing is in the antrum near the pylorus, where the occluding ACWs generate the mechanical forces and fluid motions that promote food mixing and disintegration. In our study, antral mixing has been investigated by releasing two sets of four particles from initial locations (black spheres) and tracing them throughout the simulations. This procedure is illustrated in Fig. 4.42. The paths of the particles are indicated by the white curves, and the final position of the particles is marked by white spheres.

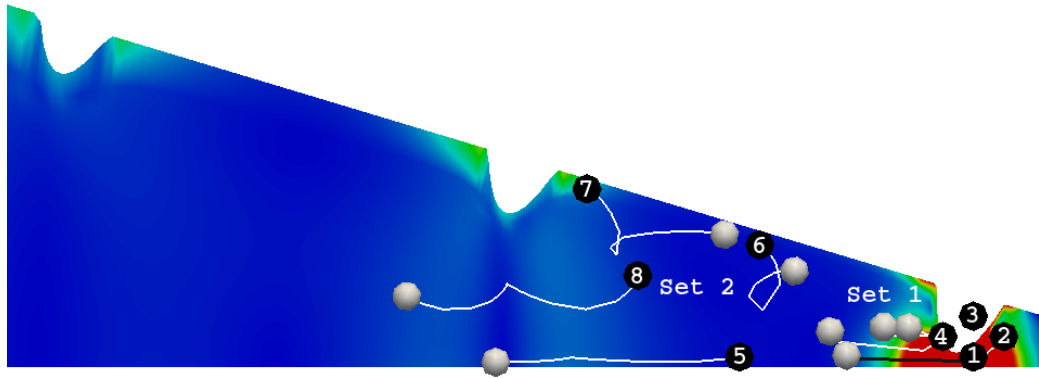


Figure 4.42: Mixing in distal antrum illustrated by releasing two sets of four particles from initial locations (black spheres) to locations when the maximum relative occlusion is achieved (white spheres). The initial location for the first set is under the distal-most ACW, while the initial location for the second one is between the last two consecutive ACWs.

The first set has been planted under the distal-most ACW near the pylorus (where the maximum retropulsive jet occurred), while the particles of the second one (spheres

5-8) have been planted randomly in a region between the last two consecutive ACWs (where the largest recirculation occurred).

Figure 4.42 shows that both flow patterns contribute to mixing in complementary ways. The gastric content near the stomach wall is transported towards the pylorus (spheres 6 and 7) whereas the content away from the wall is transported back into the corpus of the stomach. The content near the pylorus (spheres 1-4) experiences the high velocities and shear rates of the retropulsive jet, which assist in the breakdown of food, and causes its transport back into the corpus.

These results are consistent with the observations of Pal et al. [68], suggest that retropulsive jet near the pylorus causes particles to separate longitudinally, and antral eddies between two ACW contractions transport particles laterally toward the antral wall. To quantify the strength of antral gastric mixing of set k of N_k tracer particles, we use the mixing parameter

$$M_k = \frac{R(t^n, k)}{R(t^0, k)}, \quad (4.3)$$

proposed by Pal et al. [68], that defines the level of mixing between time t^0 and t^n from the relative spread of particles. The relative spread is measured by using the root mean square radius (or mixing radius) and is calculated by the equation

$$R(t^n, k) = \sqrt{\frac{1}{N_k} \sum_{j=1}^{N_k} \left[(x_j^n - x_m^n[k])^2 + (y_j^n - y_m^n[k])^2 + (z_j^n - z_m^n[k])^2 \right]}, \quad (4.4)$$

where (x_j^n, y_j^n, z_j^n) is the position vector of particle j at time t^n , and

$(x_m^n[k], y_m^n[k], z_m^n[k])$ is the center of mass of the N_k particles in the set k at time t^n .

For convenience, it has been assumed that the mass of all simulated fluid particles is the same, and hence the center of mass of set k is computed by averaging the position vector over all N_k particles. Further, particles that have left the antrum are excluded from the above calculations.

Figure 4.43 shows that parabolic (narrower ACWs) and more highly occluding ACWs generate higher values of antral mixing, while gastric contents associated with high viscous meals seem to be poorly mixed. Specifically, antral mixing of low and high viscous contents is observed to be more effective in the region where the maximum retropulsive jet and antral eddy occurred, respectively.

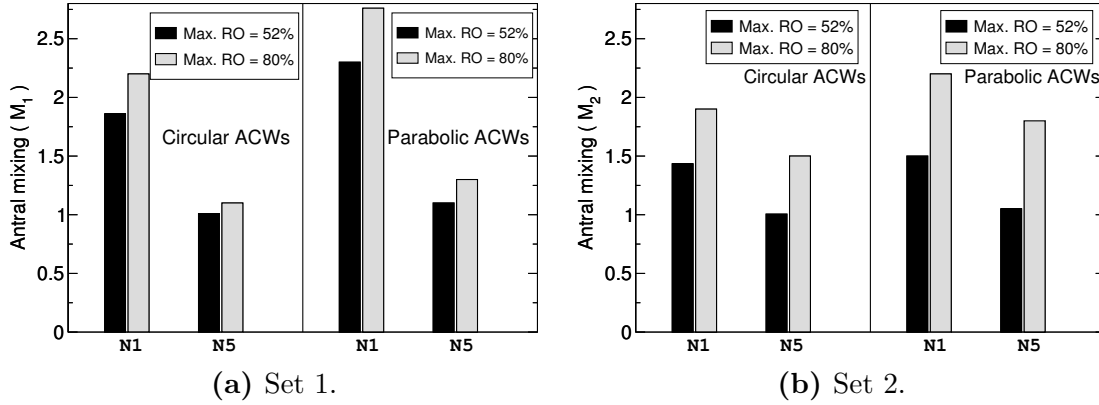


Figure 4.43: Effect of viscosity, relative occlusion and ACW shape on antral mixing. The wave speed is 2.3 mm/s and the comparison is performed at a time when the strongest retropulsive jet occurred, that is at $t = 60$ s for circular ACWs (left) and at $t = 63$ s for parabolic ACWs (right).

The actual paths traveled by individual fluid particles (white path lines) over a period of time (60 s and 63 s for circular and parabolic ACWs, respectively) are given in Fig. 4.42. This figure shows that, as the ACW propagates into the pylorus, the associated axial stretching and recirculation patterns interact and strengthen, spreading particles rapidly and broadly, enhancing mixing in the lower part of stomach.

Since the maximum retropulsive jet is obtained along the center line and under the distal-most ACW near the pylorus, we selected in this study particle number one and its associated path line (colored black) to investigate the strain rates (Fig. 4.44) and viscous stress forces (Fig. 4.45) over time. These figures show that the maximum magnitudes of strain rates and viscous stress tensor associated with high viscous fluids are larger than lower viscous ones and strengthen with circular ACW.

The dimensionless quantity γ , expresses the total strain rate that a fluid particle experiences by its motion, and is calculated by the line integral of the strain rate field along the path.

$$\gamma = \int_{t^0}^{t^n} \dot{\gamma}(s) ds . \quad (4.5)$$

The results of these integrations are summarized in Table 4.3. This table shows that the gastric content associated with high viscous meals and driven by circular contractions exerts higher strains, and this establishes the necessary forces to deform fluid particles and hence break it down eventually.

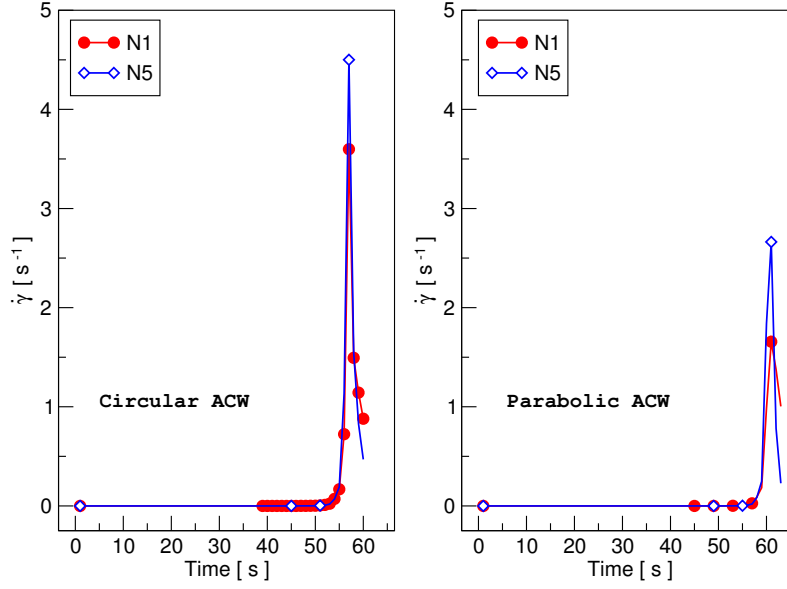


Figure 4.44: Effect of viscosity and ACW shape on the strain rate trajectory (black path line in Fig. 4.42) of particle number one. The ACWs speed is 2.3 mm/s and the maximum relative occlusion is 80%.

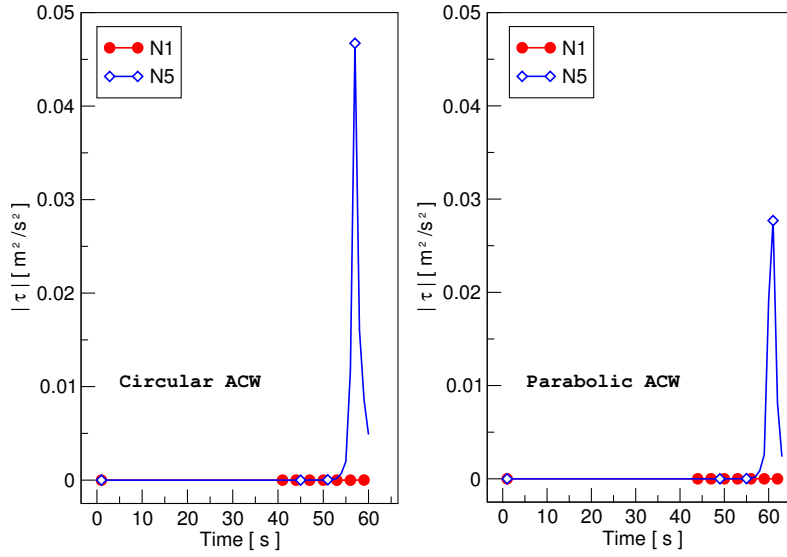


Figure 4.45: Effect of viscosity and ACW shape on the viscous stress trajectory (black path line in Fig. 4.42) of particle number one. The ACWs speed is 2.3 mm/s and the maximum relative occlusion is 80%.

Table 4.3

Effect of viscosity and ACW shape on the total strain computed along the (black) path line of particle number one (Fig. 4.42). The ACWs speed and maximum relative occlusion are 2.3 mm/s and 80%, respectively.

Circular ACWs		Parabolic ACWs	
N1	N5	N1	N5
7.7	8.5	4.8	5.8

4.4 Comparison with Literature

The main aim of this study is to investigate the solver capability for the computation of single phase peristaltic flow in a deformed lower part of an idealized human stomach during the digestion process. In order to do that, the flow fields predicted in our 2-D axisymmetric conical model was compared against the ones predicted by Pal et al. [68] and by Ferrua and Singh [3]. They assumed that the stomach was fully filled with liquid contents and ignored the effect of gravity, they only considered instantaneous flow and mixing, and their stomach geometries were assumed to be plane-symmetrical.

Our model was coupled with the CFD finite volume code solver, that is `transientSimpleDyMFoam` from the OpenFOAM software package, and a sequence of parabolic ACWs deform the upper wall of the geometry, at a time when three ACWs coexist and the pylorus is closed.

Pal et al. [68] used statical data reported in literature by Pallotta et al. [52] and

Indireskumar et al. [139] to develop a sophisticated 2-D numerical planar model of gastric mixing and emptying in the stomach. They also used the magnetic resonance imaging (MRI) movies of a healthy human stomach to capture the total propagation time and overall pattern of contraction, and later, their numerical model was effectively used to reveal “Magenstrasse” for gastric emptying [46]. MRI analysis showed that, after the ingestion of 500 mL of glucose solution (10%, w/w), the ACW activity is developed for 20 min as follows: An ACW originated at the corpus (middle part of stomach) propagates into the antrum (lower part of stomach) while the pylorus remains closed, then the pylorus opens briefly while the ACW is in the mid-antrum to empty gastric contents into the duodenum. When the ACW is within 30 mm of the pylorus, the antral segment between the ACW and pylorus contracts segmentally, closing the pylorus.

The model developed by Pal et al. [68] was characterized as follows: The lattice - Boltzmann numerical method is used together with MRI results to create the geometry. Diameters of 100 mm and 11 mm were specified at the antrum widest point and at the pyloric ring, respectively. A Newtonian fluid with density of 1000 kg/m^3 and a dynamic viscosity of 1 Pa.s is used in simulations. The wave speed (on average) and width (along x-axis) of the ACWs were assumed to be 2.5 mm/s and 18 mm , respectively, and the ACWs were initiated every 20 s at 144 mm from the pylorus. The relative occlusion linearly increased from 0% to 40% as the ACW propagated for 17.5 s to 100 mm from the pylorus, then it remained constant for 16 s. After that, it

started to linearly increase for 24 s, reaching the maximum occlusion of 90% at the pylorus with a life span of 57.5 s.

The characteristic details of the pylorus function during the last 17.5 s of the distal-most ACW life are as follows: An antral segment between the ACW and pylorus retracts gradually, opening the pylorus briefly as the ACW propagates for 2 s to 39 mm from its sphincter, then the pylorus remains fully open for 6 s to allow some gastric content into the duodenum. After that, the antral segment contracts gradually for 2 s, closing the pylorus when the ACW is within 20 mm from the pylorus sphincter. Finally, the pylorus remains closed for the last 5.5 s of the ACW life span to separate, break apart and mix gastric content.

The above observations are consistent with physiological studies of Indireskumar et al. [139] and Hausken et al. [69]. These studies suggest that a gastric emptying tends to occur during the period when the ACW is distant from the pylorus and the terminal antrum has not initiated segmental contraction.

More recently, Ferrua and Singh [3] used a fully 3-D numerical model to investigate effects of content viscosity on gastric flow during the digestion process, and at a time when the pylorus is closed. A detailed characterization of their model is as follows: The CFD software FluentTM [140] is used for numerical computations, the stomach geometry is composed by a 3-D software Gambit [141], and the characteristic dimensions of the stomach geometry are obtained from the literature of Keet [128] and Schulze [42] in conjunction with the MRI analysis of Pal et al. [46, 68]. Diameters of

100 mm and 12 mm were specified at the antrum widest point and at the pyloric ring, respectively. Two Newtonian fluids with density of 1000 kg/m^3 are used, where the first fluid, N1, has a dynamic viscosity of 0.001 Pa.s and the second one, N4*, has a dynamic viscosity of 1 Pa.s . The speed (on average) and width (along the x-axis) of the ACWs were assumed to be 2.3 mm/s and 20 mm , respectively. The ACWs were initiated every 20 s at 150 mm from the pylorus and the relative occlusion increased from 0% to 80% as the ACW propagated for 58 s to 5.4 mm from the pylorus.

The flow fields predicted by Ferrua and Singh [3] are compared against the ones that our model captured and the results, shown in Table 4.4, are generally very good. Our simulations predict that the magnitude of velocity and vorticity fields are relatively smaller than the ones reported in the literature by Ferrua and Singh [3], and larger by nearly one order of magnitude than the ones predicted by Pal et al. [68].

Table 4.4

Solver validation confirmed by comparing the velocity and vorticity fields in our simulations with the ones reported in literature by Ferrua and Singh [3]. The ACWs speed and maximum relative occlusion are 2.3 mm/s and 80% , respectively.

Fluid	Simulation		Literature	
	$ \mathbf{U} _{\max} \text{ cm /s}$	$(\omega_{\text{avg}})_{\max} \text{ s}^{-1}$	$ \mathbf{U} _{\max} \text{ cm /s}$	$(\omega_{\text{avg}})_{\max} \text{ s}^{-1}$
N1	7.74	0.96	7.8	1
N4*	11.23	0.195	11.9	0.21

4.5 Summary and Conclusions

A 2-D axisymmetric conical model has been developed to reflect a cross section of the lower part of an average sized human stomach (antrum), to get a better understanding of the flow fields that develop within the antrum during the digestion and mixing process at a time when the pylorus is closed, and to reduce the high level of complexity in full 3-D models. This model was coupled with the modified CFD finite volume code solver from the open source software package OpenFOAM, and the fixed (laboratory or Eulerian) frame of references is used to simulate the peristaltic motion for different Newtonian and non-Newtonian fluids. The non-Newtonian fluid is modeled by using the Bird-Carreau Yasuda viscosity law. A mesh refinement study showed an adequate mesh independence and the transient computations exhibited plausible convergence in terms of the initial residual.

The flow fields predicted by using a fully 3-D numerical model are compared against the ones that our model captured, and the results are generally very good. Our simulations predict that, the magnitude of velocity and vorticity fields are smaller than the ones reported in the 3-D model, and larger by nearly one order of magnitude than the ones predicted by using a 2-D numerical planar model.

By propagating ACWs toward the pylorus, two basic antral flow patterns are captured, a backward retropulsive jet flow developed in the most highly occluded region near the pylorus and a forward recirculation flow between and under the ACWs

crests. Both of which together with the other physical quantities such as pressure, strain rates and viscous stresses contribute to food disintegration and mixing. Moreover, the presence of back-flow is a ubiquitous feature of all our simulations, and this back-flow increases in magnitude as the ACW approaches the pylorus sphincter, reaching the maximum magnitude near the pylorus and under the distal-most ACW. The increasing occlusion of the ACW strengthens these physical quantities faster, reaching the maximum in the most occluded section of the antrum. Specifically, the maximum magnitude of velocity (quantities the retropulsive jet strength) is achieved under the distal-most ACW near the pylorus and along the center line, with strength much larger than the ACWs speed. This jet constitutes the strongest forces for the mixing and disintegration of food. The maximum magnitude of the averaged volume vorticity field (quantities eddies strength) is achieved in a region between the last two consecutive ACWs near the pylorus. These eddies support a gentle mixing and disintegration of the gastric contents. The maximum values of kinetic pressure are achieved in a region close to the propagation of the distal-most ACW and pylorus, and the highest strain rates are achieved in the region of direct contact with the distal-most ACW near the pylorus.

Along the center line, all physical quantities behave nearly the same in a location away from the pylorus, while near the pylorus they strengthen in magnitude with fluid viscosity, power-law index and relative occlusion of the ACW. The same qualitative behavior is seen for larger and smaller relative occlusions and the strength of

all physical quantities increases linearly with the speed of ACWs. For the high viscous fluids, although the strength of the retropulsive jet is nearly insensitive to fluid viscosity (Figs 4.8 and 4.9) and the general behavior of pressure field is relatively insensitive to fluid viscosity at small rate (Figs 4.13 and 4.14), pressure gradient field is not (Figs 4.15 and 4.16). In fact and as previously reported in literature, the pressure gradients have an essential role in promoting the gastric digestion of high viscous meals. For the low viscous fluids, the induced back-flow near the center line leads to a global recirculation of the gastric contents, hence to a large scale mixing. Shear-thinning of the fluids is relevant in the retropulsive jet region, where the maximum speed decreases with viscosity.

On one hand, retropulsive jet, eddies and antral pressure field are relatively insensitive to the ACW shape, with a little obvious advantage for parabolic ACWs. On the other hand, parabolic ACW is observed to produce much stronger and more intensive pressure gradient field, strain rates and stress viscous forces near the pylorus. Particularly, the shape of the ACWs has relatively little influence on the low-viscous fluids but larger influence on the high-viscous ones, with retropulsive jet is slightly stronger for parabolic ACWs.

In the most occluded region of the antrum, the strength of antral mixing increases with relative occlusion and parabolic shape of the ACW. In particular, low viscous meals are highly mixed under the distal-most ACW near the pylorus, while high viscous ones are highly mixed in a region between the last two consecutive ACWs near

the pylorus (Fig. 4.43).

Chapter 5

Summary and Future Work

In this thesis we have developed computational models for the investigation of fluid transport and mixing in the human gastro-intestinal tract. The peristaltic motion was modeled by means of traveling waves which deform the boundary of the tubular vessels. An axisymmetric tube of uniform diameter was used to describe the small intestines, and an axisymmetric conical vessel was utilized to model the lower part of the human stomach.

The computations were performed with a modified finite volume solver within the open source CFD environment OpenFOAM. The simulations were performed for different Newtonian and non-Newtonian fluids, where the non-Newtonian fluids were modeled by means of the Bird-Carreau Yasuda viscosity law.

The fluid was taken to be a single-phase fluid, and the flow was assumed to be incompressible, isothermal and inelastic. Neither the wall roughness nor the friction and the gravity forces were considered in the simulations. The mesh motion is governed by the Laplace equation with a directional diffusion field, which is used to control the deforming mesh quality. The motion of the fluid is governed by the conservation laws for mass and momentum on a moving mesh.

Mesh refinement studies showed adequate mesh independence for all cases. The transient computations exhibited sufficient convergence in terms of the velocity and pressure residuals. Whenever possible, comparisons are made to simulations from the literature to validate our results. In general, the agreement can be considered very good. These comparisons show that our 2-D axisymmetric models are sufficient to reproduce the high level of complexity of fully 3-D models. Also, our simulations confirm that the numerical methods are valid for such peristaltic motions.

Two fundamentally different flow phenomena were considered, namely fluid transport and fluid mixing due to peristaltic motion. One of the main fluid transport properties was the transport efficiency which was investigated in a cylindrical tube and a conical-shaped vessel. In both cases, the relative occlusion played the most important part in the transport efficiency, whereas (surprisingly) the wave speed had little or no influence. The results of this study is documented in Chapter 3 and is summarized in Section 3.3. The fluid mixing was investigated in the conical vessel for a closed

pylorus valve. The simulations confirmed that the main mechanism for food disintegration is the retropulsive jet near the pylorus which is induced by the peristaltic waves. In addition to the retropulsive jet, the mixing and particle tracking provided additional insights into the mechanical digestion process of the human stomach. The results of the mixing simulations are documented in Chapter 4 and the comparison with other studies are discussed in Section 4.4 and the conclusions are reported in Section 4.5.

Future Work

- It is also of interest to study the effect of gravity and volume of gastric contents on emptying, digestion and mixing processes.
- Further, more mathematical analyses needed to be done to trace the fluid elements along their particle path lines within the lower part of the stomach to study the stress, the fluid deformation, the strain rates and the break up of physical drops along its particle path lines.
- The relationship between the initial locations of tracer particles and the stomach posture on the overall mixing process in stomach is an interesting future work.
- Moreover, modifying the moving-mesh boundary conditions to generate more general and realistic types of ACWs and simulating complex fluids using multiphase flows, could be the subject of future investigations.

- It would be interesting to use the RBF mesh motion solver to obtain the new mesh points and to compare the mesh validity and quality metrics against the ones that obtained from the other techniques, such as Laplace and SBR methods.
- Finally, to quantitatively validate the flow fields that develop within the stomach, an experimental work confirmation need to be done, and use it as a reference for numerical computations. In particular, of using a non-intrusive flow measurement technique, for example, particle imaging velocimetry, to trace the flow field that develops within a closed system.

References

- [1] S. Nahar, S. A. K. Jeelani, and E. J. Windhab, “Peristaltic flow characterization of a shear thinning fluid through an elastic tube by UVP,” *Applied Rheology*, vol. 22, no. 4, p. 43941, 2012.
- [2] S. Nahar, *Steady and unsteady flow characteristics of non-Newtonian fluids in deformed elastic tubes*. PhD thesis, Swiss Federal Institute of Technology (ETH), 2012.
- [3] M. J. Ferrua and R. P. Singh, “Modeling the fluid dynamics in a human stomach to gain insight of food digestion,” *Journal of Food Science*, vol. 75, no. 7, pp. 151–162, 2010.
- [4] T. W. Latham, “Fluid motions in a peristaltic pump,” Master’s thesis, Massachusetts Institute of Technology, 1966.
- [5] A. H. Shapiro, “Pumping and retrograde diffusion in peristaltic waves,” in *Proceeding Workshop Ureteral Reflux in Children*, Nat. Acad. Sci. Washington,

DC, 1976.

- [6] Y. Fung and C. Yih, “Biofluid mechanics in flexible tubes,” *Trans. ASME: Journal of Applied Mechanics*, vol. 35, no. 4, pp. 669–675, 1968.
- [7] A. H. Shapiro, M. Y. Jaffrin, and S. L. Weinberg, “Peristaltic pumping with long wavelengths at low Reynolds number,” *Journal of Fluid Mechanics*, vol. 37, no. 4, pp. 799–825, 1969.
- [8] J. C. Burns and T. Parkes, “Peristaltic motion,” *Journal of Fluid Mechanics*, vol. 29, no. 4, pp. 731–743, 1968.
- [9] C. Barton and S. Raynor, “Peristaltic flow in tubes,” *Bull Math Biophys*, vol. 30, pp. 663–680, 1968.
- [10] P. S. Lykoudis, “Peristaltic pumping: A bioengineering model,” in *Proceeding Workshop Hydrodynamic Upper Urinary Tract, Nat. Acad. Sci., Washington, DC*, 1971.
- [11] S. L. Weinberg, M. Y. Jaffrin, and A. H. Shapiro, “A hydrodynamical model of ureteral function,” in *Proceeding Workshop Hydrodynamic Upper Urinary Tract, Nat. Acad. Sci., Washington, DC*, 1971.
- [12] H. S. Lew, Y. C. Fung, and C. B. Lowenstein, “Peristaltic carrying and mixing of chyme in the small intestine (An analysis of a mathematical model of peristalsis of the small intestine),” *Journal of Biomechanics*, vol. 4, pp. 297–315, 1971.

- [13] M. Y. Jaffrin and A. H. Shapiro, “Peristaltic pumping,” *Annual Review of Fluid Mechanics*, vol. 3, no. 1, pp. 13–37, 1971.
- [14] N. Liron, “A new look at peristalsis and its functions,” *Horizons in Biochemistry and Biophysics*, vol. 5, pp. 161–182, 1978.
- [15] E. C. Eckstein, S. L. Weinberg, and A. H. Shapiro, “An experimental study of peristaltic pumping,” *Journal of Fluid Mechanics*, vol. 49, no. 3, pp. 461–479, 1971.
- [16] M. G. Mank, *Berechnung der peristaltischen Flüssigkeits Forderung mit Method der finiten Element*. PhD thesis, Hannover, 1976.
- [17] A. H. Shapiro and T. W. Latham, “On peristaltic pumping (abstract),” in *Proceeding of Annual Conference on Engineering in Medicine and Biology, Holden Day, San Francisco*, vol. 8, p. 147, 1966.
- [18] E. C. Eckstein, “Experimental and theoretical pressure studies of peristaltic pumping,” Master’s thesis, Massachusetts Institute of Technology, 1970.
- [19] S. L. Weinberg, *Theoretical and experimental treatment of peristaltic pumping and its relation to ureteral function*. PhD thesis, Massachusetts Institute of Technology, 1970.
- [20] Y. C. Fung and F. C. P. Yin, “Comparison of theory and experiment in peristaltic transport,” *Journal of Fluid Mechanics*, vol. 47, no. 1, pp. 93–112, 1971.

- [21] T. K. Hung and T. D. Brown, “Solid-particle motion in two-dimensional peristaltic flows,” *Journal of Fluid Mechanics*, vol. 73, no. 1, pp. 77–96, 1976.
- [22] L. M. Srivastava and V. P. Srivastava, “Peristaltic transport of blood: Casson model II,” *Journal of Biomechanics*, vol. 17, no. 11, pp. 821–829, 1984.
- [23] A. Medhavi, “Peristaltic pumping of a non-newtonian fluid,” *Applications and Applied Mathematics: An International Journal (AAM)*, vol. 3, no. 1, pp. 137–148, 2008.
- [24] K. K. Raju and R. Devanathan, “Peristaltic motion of a non-newtonian fluid,” *Rheologica Acta*, vol. 11, no. 2, pp. 170–178, 1972.
- [25] B. F. Picologlou, P. D. Patel, and P. S. Lykoudis, “Biorheological aspects of colonic activity. Part I. Theoretical considerations,” *Biorheology*, vol. 10, no. 3, pp. 431–440, 1973.
- [26] J. B. Shukla and S. P. Gupta, “Peristaltic transport of a power-law fluid with variable consistency,” *Trans. ASME K: Journal of Biomechanical Engineering*, vol. 104, no. 3, pp. 182–186, 1982.
- [27] E. Becker, “Simple non-newtonian fluid flows,” *Advances in Applied Mechanics*, vol. 20, pp. 177–226, 1980.
- [28] K. K. Raju and R. Devanathan, “Peristaltic motion of a non-newtonian fluid. Part II. Visco-elastic fluid,” *Rheologica Acta*, vol. 13, no. 6, pp. 944–948, 1974.

- [29] G. B. Böhme and R. Friedrich, “Peristaltic flow of viscoelastic liquids,” *Journal of Fluid Mechanics*, vol. 128, no. 1, pp. 109–122, 1983.
- [30] A. M. Siddiqui, A. Provost, and W. H. Schwarz, “Peristaltic pumping of a second-order fluid in a planar channel,” *Rheologica Acta*, vol. 30, no. 3, pp. 249–263, 1991.
- [31] L. M. Srivastava and V. P. Srivastava, “Peristaltic transport of a non-newtonian fluid: applications to the vas deferens and small intestine,” *Annals of biomedical engineering*, vol. 13, no. 2, pp. 137–153, 1985.
- [32] V. P. Srivastava and M. Saxena, “A two-fluid model of non-newtonian blood flow induced by peristaltic waves,” *Annals of Biomedical Engineering*, vol. 34, no. 4, pp. 406–414, 1995.
- [33] A. M. Provost and W. H. Schwarz, “A theoretical study of viscous effects in peristaltic pumping,” *Journal of Fluid Mechanics*, vol. 279, pp. 177–195, 1994.
- [34] E. F. Elshehawey, A. M. Misery, and A. H. A. Naby, “Peristaltic motion of generalized newtonian fluid in a non-uniform channel,” *Journal of the Physical Society of Japan*, vol. 67, no. 2, pp. 434–440, 1998.
- [35] A. V. Mernone, J. N. Mazumdar, and S. K. Lucas, “A mathematical study of peristaltic transport of a casson fluid,” *Mathematical and Computer Modelling*, vol. 35, no. 7-8, pp. 895–912, 2002.

- [36] T. Hayat and N. Ali, “A mathematical description of peristaltic hydromagnetic flow in a tube,” *Applied mathematics and computation*, vol. 188, no. 2, pp. 1491–1502, 2007.
- [37] S. Nadeem and N. S. Akbar, “Influence of heat transfer on a peristaltic transport of herschel–bulkley fluid in a non-uniform inclined tube,” *Communications in Nonlinear Science and Numerical Simulation*, vol. 14, no. 12, pp. 4100–4113, 2009.
- [38] S. Nadeem, N. S. Akbar, M. Malik, and C. Lee, “Peristaltic flow of a jeffrey-six constant fluid in a uniform inclined tube,” *International Journal for Numerical Methods in Fluids*, vol. 69, no. 9, pp. 1550–1565, 2011.
- [39] S. Nahar, S. Jeelani, and E. Windhab, “Influence of elastic tube deformation on flow behavior of a shear thinning fluid,” *Chemical Engineering Science*, vol. 75, no. 4, pp. 445–455, 2012.
- [40] A. A. Al-Hababbeh, *Simulation of Newtonian and non-Newtonian flows in deformable tubes*. PhD thesis, Michigan Technology University, 2013.
- [41] DI. Soybel, “Anatomy and physiology of the stomach,” *Surgical Clinics of North America*, vol. 85, no. 5, pp. 875–894, 2005.
- [42] K. Schulze, “Imaging and modeling of digestion in the stomach and the duodenum,” *Neurogastroenterology and Motility*, vol. 18, no. 3, pp. 172–183, 2006.

- [43] W. B. Cannon, “Movements of the stomach, studied by means of the röntgen rays,” *Journal of the Boston Society of Medical Sciences*, vol. 2, no. 6, pp. 59–66, 1898.
- [44] K. A. Kelly, “Gastric emptying of liquids and solids: roles of proximal and distal stomach,” *American Journal of Physiology–Gastrointestinal and Liver Physiology*, vol. 239, no. 2, pp. G71–G76, 1980.
- [45] J. L. Urbain, E. Van Cutsem, J. A. Siegel, S. Mayeur, A. Vandecruys, J. Janssens, M. De Roo, and G. Vantrappen, “Visualization and characterization of gastric contractions using a radionuclide technique,” *American Journal of Physiology–Gastrointestinal and Liver Physiology*, vol. 259, no. 6, pp. G1062–G1067, 1990.
- [46] A. Pal, J. G. Brasseur, and B. Abrahamsson, “A stomach road or ”magenstrasse” for gastric emptying,” *Journal of Biomechanics*, vol. 40, no. 6, pp. 1202–1210, 2007.
- [47] F. Kong and R. P. Singh, “A model stomach system to investigate disintegration kinetics of solid foods during gastric digestion,” *Journal of Food Science*, vol. 73, no. 5, pp. E202–E210, 2008.
- [48] Y. Imai, I. Kobayashi, S. Ishida, T. Ishikawa, M. Buist, and T. Yamaguchi, “Antral recirculation in the stomach during gastric mixing,” *American Journal*

- of Physiology–Gastrointestinal and Liver Physiology*, vol. 304, no. 5, pp. G536–G542, 2013.
- [49] O. Keinke, M. Schemann, and HJ. Ehrlein, “Mechanical factors regulating gastric emptying of viscous nutrient meals in dogs,” *Quarterly Journal of Experimental Physiology*, vol. 69, no. 4, pp. 781–795, 1984.
- [50] PM. King, RD. Adam, A. Pryde, WN. McDicken, and RC. Heading, “Relationships of human antroduodenal motility and transpyloric fluid movement: non-invasive observations with real-time ultrasound,” *Gut*, vol. 25, no. 4, pp. 1384–1391, 1984.
- [51] T. Hausken, S. Ørregard, K. Matre, and A. Berstad, “Antroduodenal motility and movements of luminal contents studied by duplex sonography,” *Gastroenterology*, vol. 102, no. 5, pp. 1583–1590, 1992.
- [52] N. Pallotta, M. Cicala, C. Frandina, and E. Corazzari, “Antro-pyloric contractile patterns and transpyloric flow after meal ingestion in humans,” *The American Journal of Gastroenterology*, vol. 93, no. 12, pp. 2513–2522, 1998.
- [53] P. Boulby, R. Moore, P. Gowland, and RC. Spiller, “Fat delays emptying but increases forward and backward antral flow as assessed by flow-sensitive magnetic resonance imaging,” *Neurogastroenterology and Motility*, vol. 11, no. 1, pp. 27–36, 1999.

- [54] R. Burn-Murdoch, M. A. Fisher, and J. N. Hunt, “Does lying on the right side increase the rate of gastric emptying?,” *The Journal of Physiology*, vol. 302, pp. 395–398, 1980.
- [55] S. Doran, K.L. Jones, J.M. Andrews, and M. Horowitz, “Effects of meal volume and posture on gastric emptying of solids and appetite,” *American Journal of Physiology*, vol. 275, no. 5, pp. R1712–R1718, 1998.
- [56] H. Faas, A. Steingoetter, C. Feinle, T. Rades, H. Lengsfeld, P. Boesiger, M. Fried, and W. Schwizer, “Effects of meal consistency and ingested fluid volume on the intragastric distribution of a drug model in humans—a magnetic resonance imaging study,” *Alimentary Pharmacology and Therapeutics*, vol. 16, no. 2, pp. 217–224, 2002.
- [57] A. Steingoetter, M. Fox, R. Treier, D. Weishaupt, B. Marincek, P. Boesiger, M. Fried, and W. Schwizer, “Effects of posture on the physiology of gastric emptying: A magnetic resonance imaging study,” *Scandinavian Journal of Gastroenterology*, vol. 41, no. 10, pp. 1155–1164, 2006.
- [58] M. Edelbroek, M. Horowitz, A. Maddox, and J. Bellen, “Gastric emptying and intragastric distribution of oil in the presence of a liquid or a solid meal,” *Journal of Nuclear Medicine*, vol. 33, pp. 1283–1290, 1992.
- [59] O. Goetze, A. Steingoetter, D. Menne, I.R. van der Voort, M.A. Kwiatek, P. Boesiger, D. Weishaupt, M. Thumshirn, M. Fried, and W. Schwizer, “The effect

- of macronutrients on gastric volume responses and gastric emptying in humans: a magnetic resonance imaging study,” *American Journal of Physiology–Gastrointestinal and Liver Physiology*, vol. 292, no. 1, pp. G11–G17, 2007.
- [60] W. Schwizer, A. Steingötter, M. Fox, T. Zur, M. Thumshirn, P. Bösiger, and M. Fried, “Non-invasive measurement of gastric accommodation in humans,” *Gut*, vol. 51, pp. i59–i62, 2002.
- [61] A. Pullan, L. Cheng, R. Yassi, and M. Buist, “Modelling gastrointestinal bioelectric activity,” *Progress in Biophysics and Molecular Biology*, vol. 85, no. 2-3, pp. 523–550, 2004.
- [62] V. Spitzer, M. J. Ackerman, A. L. Scherzinger, and D. Whitlock, “The visible human male: a technical report,” *Journal of the American Medical Informatics Association*, vol. 3, no. 2, pp. 118–130, 1996.
- [63] S. Aoki, K. Uesugi, H. Ozawa, and M. Kayano, “Evaluation of the correlation between in vivo and in vitro release of phenylpropanolamine HCl from controlled-release tablets,” *International Journal of Pharmaceutics*, vol. 85, no. 1-3, pp. 65–73, 1992.
- [64] S. Aoki, H. Ando, K. Tatsuishi, K. Uesugi, and H. Ozawa, “Determination of the mechanical impact force in the in vitro dissolution test and evaluation of the correlation between in vivo and in vitro release,” *International Journal of Pharmaceutics*, vol. 95, no. 1-3, pp. 67–75, 1993.

- [65] K. Molly, M. Vande Woestyjne, and W. Verstraete, “Development of a 5-step multi-chamber reactor as a simulation of the human intestinal microbial ecosystem,” *Applied Microbiology and Biotechnology*, vol. 39, no. 2, pp. 254–258, 1993.
- [66] M. Wickham, R. Faulks, and C. Mills, “In vitro digestion methods for assessing the effect of food structure on allergen breakdown,” *Molecular Nutrition and Food Research*, vol. 53, no. 8, pp. 952–958, 2009.
- [67] SK. Singh, *Fluid flow and disintegration of food in human stomach*. PhD thesis, University of California, Davis, CA: Biological Systems Engineering, 2007.
- [68] A. Pal, K. Indireskumar, W. Schwizer, B. Abrahamsson, M. Fried, and J. G. Brasseur, “Gastric flow and mixing studied using computer simulation,” in *Proceedings of the Royal Society, London B*, vol. 271, pp. 2587–2594, 2004.
- [69] T. Hausken, M. Mundt, and M. Samsom, “Low antroduodenal pressure gradients are responsible for gastric emptying of a low-caloric liquid meal in humans,” *Neurogastroenterology and Motility*, vol. 14, no. 1, pp. 97–105, 2002.
- [70] CH. Versantvoort, E. Van de Kamp, and CJM. Rompelberg, “Applicability of an in vitro digestion model in assessing the bioaccessibility of mycotoxins from food,” *Food and Chemical Toxicology*, vol. 43, no. 1, pp. 31–40, 2004.
- [71] PM. Hellström, P. Grybäck, and H. Jacobsson, “The physiology of gastric emptying,” *Best Practice and Research Clinical Anaesthesiology*, vol. 20, no. 3, pp. 397–407, 2006.

- [72] F. Kong and R. P. Singh, “Disintegration of solid foods in human stomach,” *Journal of Food Science*, vol. 73, no. 5, pp. R67–R80, 2008.
- [73] I. G. Currie, *Fundamental mechanics of fluids*. CRC Press, 2012.
- [74] K. Hutter and K. Jöhnk, *Continuum methods of physical modeling: continuum mechanics, dimensional analysis, turbulence*. Springer Verlag, 2004.
- [75] I. Wlokas, “Aerothermodynamics, lecture notes, University of Duisburg-Essen, Institute for Combustion and Gasdynamics,” 2014.
- [76] M. Reiner, “The Deborah number,” *Physics Today*, vol. 17, p. 62, 1964.
- [77] R. B. Bird, R. C. Armstrong, and O. Hassager, *Dynamics of Polymeric Liquids*. New York: John Wiley and Son Inc, 2nd ed., 1987.
- [78] J. F. Richardson and R. P. Chhabra, *Non-Newtonian flow and applied rheology*. Oxford: Butterworth-Heinemann, 2nd ed., 2008.
- [79] H. Jasak, *Error analysis and estimation in the Finite Volume method with applications to fluid flows*. PhD thesis, Imperial College, University of London, 1996.
- [80] J. D. Hoffman, *Numerical Methods for Engineers and Scientists*. New York: McGrawHill, 1992.
- [81] P. Wesselin, *Principles of Computational Fluid Dynamics*. Heidelberg: Springer, 2001.

- [82] L. N. Trefethen and D. Bau III, *Numerical linear algebra*. No. 50, Society for Industrial and Applied Mathematics, 1997.
- [83] H. Jasak and Ž. Tuković, “Automatic mesh motion for the unstructured finite volume method,” *Transactions of FAMENA*, vol. 30, no. 2, pp. 1–20, 2006.
- [84] P. Thomas and C. Lombard, “Geometric conservation law and its application to flow computations on moving grids,” *The American Institute of Aeronautics and Astronautics Journal*, vol. 17, no. 10, pp. 1030–1037, 1979.
- [85] V. P. Pillalamarri, “A consistent algorithm for implementing the space conservation law,” Master’s thesis, University of Massachusetts–Amherst, pavan.narsimharao@gmail.com, 2014.
- [86] F. Giussani, “Multidimensional simulation of moving geometries with topological changes by an open source cfd code: https://www.politesi.polimi.it/bitstream/10589/109045/3/2015_07_Giussani.pdf,” Master’s thesis, POLITECNICO DI MILANO, 2014/2015.
- [87] M. Auvinen, J. Ala-Juusela, N. Pedersen, and T. Siikonen, “Time-accurate turbomachinery simulations with open-source cfd; flow analysis of a single-channel pump with openfoam,” in *Proceedings of the V European Conference on Computational Fluid Dynamics, ECCOMAS CFD*, (Lisbon, Portugal), June 2010.

- [88] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby, “A tensorial approach to computational continuum mechanics using object-oriented techniques,” *Computers in Physics*, vol. 12, no. 6, 1998.
- [89] S. Patankar, *Numerical heat transfer and fluid flow*. Taylor & Francis, 1980.
- [90] R. I. Issa, “Solution of the implicitly discretised fluid flow equations by operator-splitting,” *Journal of Computational Physics*, vol. 62, pp. 40–65, 1986.
- [91] F. Moukalled, L. Mangani, and M. Darwish, *The Finite Volume Method in Computational Fluid Dynamics*. Springer, 2016.
- [92] H. K. Versteeg and W. Malalasekera, *An Introduction to Computational Fluid Dynamics*. USA: Pearson Education Limited, 2nd ed., 2007.
- [93] C. Romain, F. Guibault, C. Devals, and B. Nennemann, “Numerical study of rotor-stator interactions in a hydraulic turbine with foam-extend,” *In IOP Conference Series: Earth and Environmental Science*, vol. 49, no. 6, p. 062012, 2016.
- [94] S. M. Damián, *An Extended Mixture Model for the Simultaneous Treatment of Short and Long Scale Interfaces*. PhD thesis, Universidad Nacional del Litoral, 2013.
- [95] M. Behr and T. Tezduyar, “Finite element solution strategies for large-scale

- flow simulations,” *Computer Methods in Applied Mechanics and Engineering*, vol. 112, no. 1–4, pp. 3–24, 1994.
- [96] I. Güler, M. Behr, and T. Tezduyar, “Parallel finite element computation of free-surface flows,” *Computational Mechanics*, vol. 23, no. 2, pp. 117–123, 1999.
- [97] J. Batina, “Unsteady euler airfoil solutions using unstructured dynamic meshes,” *The American Institute of Aeronautics and Astronautics*, vol. 28, no. 8, pp. 1381–1388, 1990.
- [98] F. Blom, “Considerations on the spring analogy,” *International Journal for Numerical Methods in Fluids*, vol. 32, no. 6, pp. 647–668, 2000.
- [99] C. Degand and C. Farhat, “A three-dimensional torsional spring analogy method for unstructured dynamic meshes,” *Computers and Structures*, vol. 80, no. 3–4, pp. 305–316, 2002.
- [100] C. Farhat, C. Degand, B. Koobus, and M. Lesoinne, “Torsional springs for two-dimensional dynamic unstructured fluid meshes,” *Computer Methods in Applied Mechanics and Engineering*, vol. 163, no. 1–4, pp. 231–245, 1998.
- [101] R. Löhner and C. Yang, “Improved ale mesh velocities for moving bodies,” *Communications in Numerical Methods in Engineering*, vol. 12, no. 10, pp. 599–608, 1996.

- [102] D. Littlefield, “The use of r-adaptivity with local, intermittent remesh for modeling hypervelocity impact and penetration,” *International Journal of Impact Engineering*, vol. 26, no. 1–10, pp. 433–442, 2001.
- [103] A. Masud and T. Hughes, “A space-time galerkin/least-squares finite element formulation of the navier-stokes equations for moving domain problems,” *Computer Methods in Applied Mechanics and Engineering*, vol. 146, no. 1–2, pp. 91–126, 1997.
- [104] I. Robertson and S. Sherwin, “Free-surface flow simulation using hp/spectral elements,” *Journal of Computational Physics*, vol. 155, no. 1, pp. 26–53, 1999.
- [105] M. Behr and F. Abraham, “Free-surface flow simulations in the presence of inclined walls,” *Computer Methods in Applied Mechanics and Engineering*, vol. 191, no. 47–48, pp. 5467–5483, 2002.
- [106] R. A. Cairncross, P. R. Schunk, T. A. Baer, R. R. Rao, and P. A. Sackinger, “A finite element method for free surface flows of incompressible fluids in three dimensions. part I. boundary fitted mesh motion,” *International Journal for Numerical Methods in Fluids*, vol. 33, no. 3, pp. 375–403, 2000.
- [107] G. Chiandussi, G. Buggeda, and E. Oñate, “A simple method for automatic update of finite element meshes,” *International Journal for Numerical Methods in Biomedical Engineering*, vol. 16, no. 1, pp. 1–19, 2000.

- [108] A. A. Johnson and T. E. Tezduyar, “Mesh update strategies in parallel finite element computations of flow problems with moving boundaries and interfaces,” *Computer Methods in Applied Mechanics and Engineering*, vol. 119, no. 1–2, pp. 73–94, 1994.
- [109] P. A. Sackinger, P. R. Schunk, and R. R. Rao, “A newton-raphson pseudo-solid domain mapping technique for free and moving boundary problems: A finite element implementation,” *Journal of Computational Physics*, vol. 125, no. 1.
- [110] M. Souli and J. P. Zolesio, “Arbitrary lagrangian-eulerian and free surface methods in fluid mechanics,” *Computer Methods in Applied Mechanics and Engineering*, vol. 191, no. 3–5, pp. 451–466, 2001.
- [111] H. Zhou and J. J. Derby, “An assessment of a parallel, finite element method for three-dimensional, moving-boundary flows driven by capillarity for simulation of viscous sintering,” *International Journal for Numerical Methods in Fluids*, vol. 36, no. 7, pp. 841–865, 2001.
- [112] B. T. Helenbrook, “Mesh deformation using the biharmonic operator,” *International Journal for Numerical Methods in Engineering*, vol. 56, no. 7, pp. 1007–1021, 2003.
- [113] F. M. Bos, *Numerical simulation of flapping foil and wind aerodynamics: Mesh deformation using radial basis functions*. PhD thesis, Technical University, Delft, Netherlands, 2009.

- [114] P. Moradnia, “Project work for the phd course in openfoam, a tutorial on how to use dynamic mesh solver icodmfoam,” 2008.
- [115] H. Jasak and Z. Tuković, “Dynamic mesh handling in openfoam applied to fluid-structure interaction simulations,” in *Proceedings of the V European Conference on Computational Fluid Dynamics (ECCOMAS CFD 2010)(Lisbon, Portugal, 14-17 June 2010)*, JCF Pereira AS, Pereira JMC, (Eds.).(27), 2010.
- [116] C. Kassiotis, “Which strategy to move the mesh in the computational fluid dynamic code openfoam,” *Report École Normale Supérieure de Cachan. Available online: <http://perso.crans.org/kassiotis/openfoam/movingmesh.pdf>*, 2008.
- [117] H. Baruh, *Analytical dynamics*. McGraw-Hill Inc., 1999.
- [118] R. P. Dwight, *Robust mesh deformation using the linear elasticity equations In: Deconinck H, Dick E, editors. Computational fluid dynamics 2006*. Springer, 2004.
- [119] W. R. Madych and S. A. Nelson, “Multivariate interpolation and conditionally positive definite functions.II,” *Mathematics of Computation-American Mathematical Society*, vol. 54, pp. 211–230, 1990.
- [120] H. Wendland, *Konstruktion und Untersuchung radialer Basisfunktionen mit kompaktem Träger*. PhD thesis, Universität zu Göttingen, 1996.

- [121] A. Beckert and H. Wendland, “Multivariate interpolation for fluid-structure-interaction problems using radial basis functions,” *Aerospace Science and Technology*, vol. 5, no. 2, pp. 125–134, 2001.
- [122] A. O. González, A. Vallier, and H. Nilsson, “Mesh motion alternatives in OpenFOAM,” 2009.
- [123] T. Behrens, “Openfoam’s basic solvers for linear systems of equations: Solvers, preconditioners, smoothers,” *Available online: http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2008/TimBehrens/tibeh-report-fin.pdf*, 2009.
- [124] T. Lucchini, “Running openfoam tutorials.”
- [125] S. Nahar, S. Jeelani, and E. Windhab, “Steady and unsteady flow characteristics of a shear thinning fluid through a collapsed elastic tube,” in *7th International Symposium on Ultrasonic Doppler Methods for Fluid mechanics and Fluid Engineering, Chalmers University of Technology, Gothenburg, Sweden*, pp. 7–8, 2010.
- [126] M. Stranzinger, *Numerical and experimental investigations of Newtonian and non-Newtonian Flow in annular gaps with scraper blades*. PhD thesis, Swiss Federal Institute of Technology (ETH), 1999.
- [127] Y. Takeda, “Velocity profile measurement by ultrasound Doppler shift method,” *International Journal of Heat and Fluid Flow*, vol. 7, no. 4, pp. 313–318, 1986.

- [128] AD. Keet, *Infantile hypertrophic pyloric stenosis. In: The pyloric sphincteric cylinder in health and disease.* Springer, 1993.
- [129] CF. Code, “New horizons in gastrointestinal physiology,” *Arch. Fisiol.*, vol. 86, no. 1, pp. 1–24, 1970.
- [130] K. Schulze-Delrieu and C. K. Brown, “Emptying of saline meals by the cat stomach as a function of pyloric resistance,” *American Journal of Physiology–Gastrointestinal and Liver Physiology*, vol. 249, no. 9, pp. G725–G732, 1985.
- [131] J. G. Brasseur, S. Corrsin, and N. Q. Lu, “The influence of a peripheral layer of different viscosity on peristaltic pumping with newtonian fluids,” *Journal of Fluid Mechanics*, vol. 174, pp. 495–519, 1987.
- [132] M. Li, J. G. Brasseur, and W. J. Dodds, “Analyses of normal and abnormal esophageal transport using computer,” *American Journal of Physiology*, vol. 266, no. 4, pp. G525–G543, 1994.
- [133] L. Marciani, PA. Gowland, RC. Spiller, P. Manoj, RJ. Moore, P. Young, and AJ. Fillery-Travis, “Development of a 5-step multi-chamber reactor as a simulation of the human intestinal microbial ecosystem,” *American Journal of Physiology–Gastrointestinal and Liver Physiology*, vol. 280, no. 6, pp. G1227–G1233, 2001.

- [134] N. Zuckerman and N. Lior, “Jet impingement heat transfer: Physics, correlations, and numerical modeling,” *Advances in Heat Transfer*, vol. 39, pp. 565–631, 2006.
- [135] N. Zuckerman and N. Lior, “Radial slot jet impingement flow and heat transfer on a cylindrical target,” *Journal of Thermophysics and Heat Transfer*, vol. 21, no. 3, pp. 548–561, 2007.
- [136] C. Feinle, P. Kunz, P. Boesiger, M. Fried, and W. Schwizer, “Scintigraphic validation of a magnetic resonance imaging method to study gastric emptying of a solid meal in humans,” *Gut*, vol. 44, no. 1, pp. 106–111, 1999.
- [137] SY. Choe, BL. Neudeck, LS. Welage, GE. Amidon, JL. Barnett, and GL. Amidon, “Novel method to assess gastric emptying in humans: the pellet gastric emptying test,” *European Journal of Pharmaceutical Sciences*, vol. 14, no. 4, pp. 347–353, 1999.
- [138] H. P. Simonian, A. H. Maurer, L. C. Knight, S. Kantor, D. Kontos, V. Megalooikonomou, R. S. Fisher, and H. P. Parkman, “Simultaneous assessment of gastric accommodation and emptying: studies with liquid and solid meals,” *Journal of Nuclear Medicine*, vol. 45, no. 7, pp. 1155–1160, 2004.
- [139] K. Indireskumar, JG. Brasseur, H. Faas, GS. Hebbard, P. Kunz, J. Dent, C. Feinle, M. Li, P. Boesiger, M. Fried, and W. Schwizer, “Relative contributions of ”pressure pump” and ”peristaltic pump” to gastric emptying,” *American*

- Journal of Physiolog–Gastrointestinal and Liver Physiology*, vol. 278, no. 4, pp. G604–G616, 2000.
- [140] Anonymous, “Fluent 6.3.26 documentation, lebanon, N.H. : ANSYS Inc., Canonsburg Pa., U.S.A.,” 2007.
- [141] R. D. McKelvey, A. M. McLennan, and T. L. Turocy, “Gambit: Software tools for game theory, version 16.0.0. <http://www.gambit-project.org>,” 2016.
- [142] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst, *Templates for the solution of linear systems: building blocks for iterative methods*. No. 43, Society for Industrial and Applied Mathematics, 1987.
- [143] F. J. Blom, “Considerations on the spring analogy,” *International Journal for Numerical Methods in Fluids*, vol. 32, no. 6, pp. 647–668, 2000.
- [144] B. N. Datta, *Numerical linear algebra and applications*, vol. 116. Society for Industrial and Applied Mathematics, 2010.
- [145] I. Demirdžić and M. Perić, “Space conservation law in finite volume calculations of fluid flow,” *International journal for numerical methods in fluids*, vol. 8, no. 9, pp. 1037–1050, 1988.
- [146] J. H. Ferziger and M. Perić, *Peristaltic transport*, vol. 3. Springer Berlin etc, 2001.

- [147] J. B. Grotberg and O. E. Jensen, “Biofluid mechanics in flexible tubes,” *Annu. Rev. Fluid Mech.*, vol. 36, pp. 121–147, 2004.
- [148] M. Hanin, “The flow through a channel due to transversely oscillating walls,” *Isr. J. Technol.*, vol. 6, pp. 67–71, 1968.
- [149] A. L. Hazel and M. Heil, “Steady finite-reynolds-number flows in three-dimensional collapsible tubes,” *Journal of Fluid Mechanics*, vol. 486, pp. 79–103, 2003.
- [150] M. Heil, “Stokes flow in collapsible tubes: computation and experiment,” *Journal of Fluid Mechanics*, vol. 353, no. 1, pp. 285–312, 1997.
- [151] M. Heil, “Stokes flow in an elastic tube—a large-displacement fluid-structure interaction problem,” *International journal for numerical methods in fluids*, vol. 28, no. 2, pp. 243–265, 1998.
- [152] L. Hogben, *Handbook of linear algebra*. Chapman & Hall, 2007.
- [153] H. Jasak, A. Jemcov, and J. Maruszewski, “Preconditioned linear solvers for large eddy simulation,” in *CFD 2007 Conference, CFD Society of Canada*, 2007.
- [154] H. Jasak and H. Rusche, “Dynamic mesh handling in openfoam,” in *Proceeding of the 47th Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition, Orlando, Florida*, 2009.

- [155] A. Krishnamoorthy and D. Menon, “Matrix inversion using cholesky decomposition,” *arXiv preprint arXiv:1111.4144*, 2011.
- [156] R. Kverneland, “Cfd-simulations of wave-windinteraction,” Master’s thesis, University of Stravanger, Norway, 2012.
- [157] E. W. Merrill, “Rheology of blood,” *Physiol Rev*, vol. 49, no. 4, pp. 863–88, 1969.
- [158] *OpenFOAM User Guide*, 2.1.0 ed., December 2011.
- [159] A. M. Provost and W. Schwarz, “A theoretical study of viscous effects in peristaltic pumping,” *Journal of Fluid Mechanics*, vol. 279, no. 1, pp. 177–195, 1994.
- [160] S. V. Patankar and D. B. Spalding, “A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows,” *International Journal of Heat and Mass Transfer*, vol. 15, no. 10, pp. 1787–1806, 1972.
- [161] M. Peric, *A finite volume method for the prediction of three-dimensional fluid flow in complex ducts*. PhD thesis, Imperial College London (University of London), 1985.
- [162] Y. Saad, *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, 2003.

- [163] J. F. Steffe, *Rheological methods in food process engineering*. Freeman Press, 1996.
- [164] F. X. Tanner, A. A. Al-Hababbeh, K. A. Feigl, S. Nahar, S. A. Jeelani, W. R. Case, and E. J. Windhab, “Numerical and experimental investigation of a non-newtonian flow in a collapsed elastic tube,” *Applied Rheology*, vol. 22, no. 6, p. 63910, 2012.

Appendix A

Open Outlet Simulations

A.1 2-D Planar Tubular Simulations

// A. A. Al-Habahbeh-2013. [40]

A.1.1 Case Setup

- Standard case: wave speed = 5 mm/s, Newtonian fluid N3, maximum relative occlusion = 60%, mesh M3.

case_0: File U

- This file contains boundary and initial conditions for the velocity.

```
dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    leftBoundary // inlet
    {
        type      zeroGradient;
    }

    rightBoundary // outlet
    {
        type      zeroGradient;
    }

    centerLine
```

```

    {
        type            symmetryPlane;
    }

upperWall
{
    type            movingWallNormalVel;
    value            uniform (0 0 0);
}

frontAndBack
{
    type            empty;
}

}

// *****

;case_/0: File p

```

- This file contains boundary and initial conditions for the pressure.


```

dimensions      [0 2 -2 0 0 0 0];

internalField    uniform 0;

boundaryField
{
    leftBoundary // inlet
    {
        type      totalPressure;
        p0         uniform 0;
        gamma      1;
        value      uniform 0;
    }

    rightBoundary // outlet
    {
        type      totalPressure;
        p0         uniform 0;
        gamma      1;
        value      uniform 0;
    }
}

```

```

centerLine
{
    type            symmetryPlane;
}

upperWall
{
    type            zeroGradient;
}

frontAndBack
{
    type            empty;
}

}

// *****

```

case_0: File pointMotionU

- This file contains some input values that control the movement of the upper wall.

```
dimensions      [0 1 -1 0 0 0 0];
```

```
internalField    uniform (0 0 0);
```

```
boundaryField
```

```
{
```

```
    leftBoundary // inlet
```

```
    {
```

```
        type          zeroGradient;
```

```
    }
```

```
    rightBoundary // outlet
```

```
    {
```

```
        type          zeroGradient;
```

```
    }
```

```

centerLine
{
    type            symmetryPlane;
}

upperWall
{
    type            dynPerCircleApproxGradually;
    circleRadius    0.01500;
    xCompInitialCenter    0.0;
    speed            0.00500;
    yCompFinalCenter    0.01900;
    value            uniform (0 0 0);
}

frontAndBack
{
    type            empty;
}

}

```

```
// *****//
```

```
;case_/constant: File dynamicMeshDict
```

- This file contains the choice of mesh motion solver and diffusivity field.

```
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    object       motionProperties;
}

// * * * * * //
```

```
dynamicFvMesh      dynamicMotionSolverFvMesh;
```

```
motionSolverLibs   ("libfvMotionSolvers.dylib");
```

```
solver             velocityLaplacian;
```

```
diffusivity        directional (1 1200 0);
```

```
// ***** //
```

```
case/constant: File transportProperties
```

- This file contains the choice of rheology models.

```
FoamFile
```

```
{
```

```
    version      2.0;
```

```
    format       ascii;
```

```
    class        dictionary;
```

```
    location     "constant";
```

```
    object       transportProperties;
```

```
}
```

```
// ***** //
```

```
transportModel  Newtonian;
```

```
nu              nu [ 0 2 -1 0 0 0 0 ] 0.1452e-03;
```

```
// ***** //
```

`};case;/constant: File turbulenceProperties`

- This file contains the choice of RAS (Reynolds-averaged stress) modeling.

```
FoamFile
{
    version      2.0;

    format       ascii;

    class        dictionary;

    location     "constant";

    object       turbulenceProperties;
}

// * * * * *

simulationType  laminar; // uses no turbulence models

// ***** //
```

`};case;/constant/polyMesh: File blockMeshDict`

- This file contains input for the generation of the mesh.

```
convertToMeters 1.0e-03;
```

```
vertices
```

```
(  
    ( 0.00 0.00 -0.10)// vertex#0  
    (180.00 0.00 -0.10)// vertex#1  
    (180.00 10.00 -0.10)// vertex#2  
    ( 0.00 10.00 -0.10)// vertex#3  
    ( 0.00 0.00 0.10)// vertex#4  
    (180.00 0.00 0.10)// vertex#5  
    (180.00 10.00 0.10)// vertex#6  
    ( 0.00 10.00 0.10)// vertex#7  
);
```

```
blocks
```

```
(  
    hex (0 1 2 3 4 5 6 7) (810 23 1)  
    simpleGrading (1 1 1)  
    // block #0  
);
```



```
edges
```

```
(  
);
```

```
boundary
```

```
(  
  
    leftBoundary // inlet  
  
    {  
  
        type patch;  
  
        faces  
  
        (  
  
            (0 3 7 4)  
  
        );  
  
    }  
  
  
    rightBoundary // outlet  
  
    {  
  
        type patch;  
  
        faces  
  
        (  
  
            (5 6 2 1)  
  
        )  
  
    }  
);
```

```

);

}

centerLine
{
    type symmetryPlane;

    faces
    (
        (1 0 4 5)
    );
}

```

```

upperWall
{
    type wall;

    faces
    (
        (2 3 7 6)
    );
}

```

```

    frontAndBack
    {
        type empty;

        faces
        (
            (0 1 2 3)

            (5 4 7 6)

        );
    }
);

```

```

mergePatchPairs
(
);

```

```

// *****

```

```

;case_/system: File controlDict

```

- This dictionary sets input parameters essential for the creation of the database.

```

application      transientSimpleDyMFoam;

```

```
startFrom      startTime;

startTime      0;

stopAt         endTime;

endTime        48;

deltaT         0.00005;

writeControl    adjustableRunTime;

writeInterval  2;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;
```

```

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

adjustTimeStep  yes;

maxCo           0.5;

maxDeltaT 1; // Maximum deltaT in seconds

libs
(
    "dynPerCircleApproxGradually.dylib"
    "movingWallNormalVel.dylib"
);

// ***** //

```

`case/system: File fvSolution`

- This file controls the equation solvers, tolerances and algorithms.

```
solvers
{
    pcorr
    {
        solver          GAMG;

        tolerance       1e-7;

        relTol          0;

        smoother        GaussSeidel;

        nPreSweeps       0;

        nPostSweeps      2;

        cacheAgglomeration off;

        agglomerator     faceAreaPair;

        nCellsInCoarsestLevel 20;

        mergeLevels      1;

        // controls the speed at which coarsening
        // or refinement levels is performed.

        maxIter          100;
```

```

        minIter      1;
    }

    p
    {
        $pcorr;

        tolerance      1e-6;

        relTol         0;
    }

    pFinal
    {
        $p;

        tolerance      1e-7;

        relTol         0;
    }

    "(U|k|epsilon|omega|nuTilda)"
    {
        solver          smoothSolver;

        smoother        GaussSeidel;
    }

```

```

        nSweeps          1;

        tolerance        1e-07;

        relTol           0;

        maxIter           100;

        minIter           1;

};

"(U|k|epsilon|omega|nuTilda)Final"

{

    solver               smoothSolver;

    smoother             GaussSeidel;

    nSweeps               2;

    tolerance             1e-07;

    relTol                0;

    maxIter               100;

    minIter               1;

}

cellMotionU

{

    solver               PCG;

    preconditioner       DIC;

```



```

        tolerance      1e-08;

        relTol         0;
    }
}

PISO
{
    nCorrectors          2;

    nOuterCorrectors     20;

    nNonOrthogonalCorrectors 0;

    correctPhi           true;
}

relaxationFactors
{
    p                    0.3;

    U                    0.7;

    k                    0.6;

    omega                0.6;

    epsilon              0.6;
}

```

```
}
```

```
// ***** //
```

```
;case_/system: File fvSchemes
```

- This file sets the numerical schemes for terms, such as derivatives in equations.

```
ddtSchemes
```

```
{  
  
    default      Euler;  
  
}
```

```
gradSchemes
```

```
{  
  
    default      Gauss linear;  
  
    grad(p)      Gauss linear;  
  
}
```

```
divSchemes
```

```
{
```

```

    default          none;

    div(phi,U)       Gauss linear;

    div((nuEff*dev(T(grad(U)))) Gauss linear;
}

laplacianSchemes
{
    default          none;

    laplacian(nu,U) Gauss linear corrected;

    laplacian(rAU,pcorr) Gauss linear corrected;

    laplacian(rAU,p) Gauss linear corrected;

    laplacian(diffusivity,cellMotionU)

    Gauss linear uncorrected;

    laplacian(nuEff,U) Gauss linear uncorrected;
}

interpolationSchemes
{
    default          linear;

    interpolate(HbyA) linear;
}

```

```

snGradSchemes

{
    default          corrected;
}

fluxRequired

{
    default          no;

    pcorr            ;

    p                 ;
}

// *****

```

A.1.2 dynPerCircleApproxGradually BC.

- Location: OpenFOAM\OpenFOAM-2.1.x\src.
- Execution: run `wmake clean` and then `wmake libso`.

Make/files file

dynPerCircleApproxGraduallyPointPatchVectorField.C

LIB = \$(FOAM_USER_LIBBIN)/dynPerCircleApproxGradually

Make/options file

EXE_INC = \

-I\$FOAM_SRC/triSurface/lnInclude \
-I\$FOAM_SRC/meshTools/lnInclude \
-I\$FOAM_SRC/dynamicMesh/lnInclude \
-I\$FOAM_SRC/finiteVolume/lnInclude \
-I\$FOAM_SRC/fvMotionSolver/lnInclude

LIB_LIBS = \

-ltriSurface \
-lmeshTools \
-ldynamicMesh \
-lfiniteVolume \
-lfvMotionSolvers

dynPerCircleApproxGraduallyPointPatchVectorField.H

Class

Foam::

dynPerCircleApproxGraduallyPointPatchVectorField

Description

Foam::

dynPerCircleApproxGraduallyPointPatchVectorField

SourceFiles

dynPerCircleApproxGraduallyPointPatchVectorField.C

-----/

#ifndef

dynPerCircleApproxGraduallyPointPatchVectorField_H

#define

dynPerCircleApproxGraduallyPointPatchVectorField_H

```

#include "fixedValuePointPatchField.H"

// * * * * *

namespace Foam
{

/*-----*\

Class
dynPerCircleApproxGraduallyPointPatchVectorField
Declaration
\*-----*/

class
dynPerCircleApproxGraduallyPointPatchVectorField
:
    public fixedValuePointPatchField<vector>
{
    // Private data

    scalar circleRadius_;

    scalar xCompInitialCenter_;

    scalar speed_;

```

```

        scalar yCompFinalCenter_;

        pointField p0_;

public:

    //- Runtime type information
    TypeName("dynPerCircleApproxGradually");

    // Constructors

    //- Construct from patch and internal field
    dynPerCircleApproxGraduallyPointPatchVectorField
    (
        const pointPatch&,
        const DimensionedField<vector, pointMesh>&
    );

    //- Construct from patch, internal field and
    // dictionary
    dynPerCircleApproxGraduallyPointPatchVectorField

```



```

(
    const pointPatch&,
    const DimensionedField<vector, pointMesh>&,
    const dictionary&
);

//- Construct by mapping given patchField<vector>
    // onto
    // a new patch
dynPerCircleApproxGraduallyPointPatchVectorField
(
    const
dynPerCircleApproxGraduallyPointPatchVectorField&,
    const pointPatch&,
    const DimensionedField<vector, pointMesh>&,
    const pointPatchFieldMapper&
);

//- Construct and return a clone
virtual
autoPtr<pointPatchField<vector> > clone() const

```

```

    {
        return autoPtr<pointPatchField<vector> >
            (
                new
dynPerCircleApproxGraduallyPointPatchVectorField
                (
                    *this
                )
            );
    }

```

```

    //- Construct as copy setting internal field
    // reference
dynPerCircleApproxGraduallyPointPatchVectorField
    (
        const
dynPerCircleApproxGraduallyPointPatchVectorField&,
        const DimensionedField<vector, pointMesh>&
    );

```

```

    //- Construct and return a clone setting

```

```

// internal field

// reference

virtual autoPtr<pointPatchField<vector> > clone

(

const DimensionedField<vector, pointMesh>& iF

) const

{

return autoPtr<pointPatchField<vector> >

(

new

dynPerCircleApproxGraduallyPointPatchVectorField

(

*this,

iF

)

);

}

// Member functions

// Mapping functions

```

```

//- Map (and resize as needed) from self given
//  a mapping
//  object

    virtual void autoMap
    (
        const pointPatchFieldMapper&
    );

//- Reverse map the given pointPatchField onto
// this

    // pointPatchField

    virtual void rmap
    (
        const pointPatchField<vector>&,
        const labellist&
    );

    // Evaluation functions

//- Update the coefficients associated with the

```

```

// patch

    // field

        virtual void updateCoeffs();


    //- Write

        virtual void write(Ostream&) const;

};


// * * * * *

} // End namespace Foam


// * * * * *

#endif


// *****//

```

dynPerCircleApproxGraduallyPointPatchVectorField.C

```
#include

"dynPerCircleApproxGraduallyPointPatchVectorField.H"

#include "pointPatchFields.H"

#include "addToRunTimeSelectionTable.H"

#include "Time.H"

#include "polyMesh.H"

#include "mathematicalConstants.H"


// * * * * *

namespace Foam
{

// * * * * * Constructors * * * * *

dynPerCircleApproxGraduallyPointPatchVectorField::
dynPerCircleApproxGraduallyPointPatchVectorField
(
    const pointPatch& p,
```

```

        const DimensionedField<vector, pointMesh>& iF
    )
:
    fixedValuePointPatchField<vector>(p, iF),
    circleRadius_(0.0),
    xCompInitialCenter_(0.0),
    speed_(0.0),
    yCompFinalCenter_(0.0),
    p0_(p.localPoints())
{}

```

```

dynPerCircleApproxGraduallyPointPatchVectorField::
dynPerCircleApproxGraduallyPointPatchVectorField
(
    const pointPatch& p,
    const DimensionedField<vector, pointMesh>& iF,
    const dictionary& dict
)
:
    fixedValuePointPatchField<vector>(p, iF, dict),
    circleRadius_(readScalar

```

```

(dict.lookup("circleRadius"))),
xCompInitialCenter_(readScalar
(dict.lookup("xCompInitialCenter"))),
    speed_(readScalar(dict.lookup("speed"))),
yCompFinalCenter_(readScalar
(dict.lookup("yCompFinalCenter")))
{
    if (!dict.found("value"))
    {
        updateCoeffs();
    }

    if (dict.found("p0"))
    {
        p0_ = vectorField("p0", dict , p.size());
    }
    else
    {
        p0_ = p.localPoints();
    }
}

```



```

dynPerCircleApproxGraduallyPointPatchVectorField::
dynPerCircleApproxGraduallyPointPatchVectorField
(
    const
dynPerCircleApproxGraduallyPointPatchVectorField&
ptf,
    const pointPatch& p,
    const DimensionedField<vector, pointMesh>& iF,
    const pointPatchFieldMapper& mapper
)
:
fixedValuePointPatchField<vector>
(ptf, p, iF, mapper),
    circleRadius_(ptf.circleRadius_),
    xCompInitialCenter_(ptf.xCompInitialCenter_),
    speed_(ptf.speed_),
    yCompFinalCenter_(ptf.yCompFinalCenter_),
    p0_(ptf.p0_)
{}

```

```

dynPerCircleApproxGraduallyPointPatchVectorField::
dynPerCircleApproxGraduallyPointPatchVectorField
(
    const
dynPerCircleApproxGraduallyPointPatchVectorField&
ptf,
    const DimensionedField<vector, pointMesh>& iF
)
:
    fixedValuePointPatchField<vector>(ptf, iF),
    circleRadius_(ptf.circleRadius_),
    xCompInitialCenter_(ptf.xCompInitialCenter_),
    speed_(ptf.speed_),
    yCompFinalCenter_(ptf.yCompFinalCenter_),
    p0_(ptf.p0_)

{}

// * * * * * Member Functions * * * * * //

```

```

void
dynPerCircleApproxGraduallyPointPatchVectorField::
autoMap
(
    const pointPatchFieldMapper& m
)
{
    fixedValuePointPatchField<vector>::autoMap(m);

    p0_.autoMap(m);
}

void
dynPerCircleApproxGraduallyPointPatchVectorField::
rmap
(
    const pointPatchField<vector>& ptf,
    const labelList& addr
)
{
    const
    dynPerCircleApproxGraduallyPointPatchVectorField&

```

```

    aOVptf =
refCast
<const
dynPerCircleApproxGraduallyPointPatchVectorField>
(ptf);

fixedValuePointPatchField<vector>::
rmap(aOVptf, addr);

    p0_.rmap(aOVptf.p0_, addr);
}

void
dynPerCircleApproxGraduallyPointPatchVectorField::
updateCoeffs()
{
    if (this->updated())
    {
        return;
    }

    const polyMesh& mesh =

```

```

this->dimensionedInternalField().mesh()();

const Time& t = mesh.time();

const pointPatch& p = this->patch();

scalar yMax
(
    max(p0_.component(vector::Y)())
);

scalar yCompInitialCenter
(
    yMax+circleRadius_
);

scalar varCenterXcomp
(
    xCompInitialCenter_+speed_*t.value()
);

scalar varCenterYcomp = 0.0;

if

```

```

(
    (t.value() > 0)

    &&

    (t.value() < 1)
)

{
    varCenterYcomp =
        yCompInitialCenter-
        t.value()*
        (yCompInitialCenter-yCompFinalCenter_);
}

else

{
    varCenterYcomp = yCompFinalCenter_;
}

scalar yDiff

(
    yMax-varCenterYcomp
);

```

```
scalar yDiffSquared
```

```
(  
    yDiff*yDiff  
);
```

```
scalar yRadicand
```

```
(  
    circleRadius_*circleRadius_-yDiffSquared  
);
```

```
scalar yRadicandSqrt
```

```
(  
    sqrt(yRadicand)  
);
```

```
scalar lowerBound
```

```
(  
    varCenterXcomp-yRadicandSqrt  
);
```

```
scalar upperBound
```

```

        (
            varCenterXcomp+yRadicandSqrt
        );

    scalar b = -1.0;

    pointField
    yCenterShift(p0_.size(),point(0.0,0.0,0.0));

    pointField
    velocity(p0_.size(),point(0.0,0.0,0.0));

    forAll(p0_,pointI)
    {

        scalar xRadicandSqrt = 0.0;

        if
        (
            (p0_.component(vector::X())[pointI] > lowerBound)

```



```

        &&
        (p0_.component(vector::X())[pointI] < upperBound)
    )
    { //major

        scalar xDiff
        (
        p0_.component(vector::X())[pointI]-varCenterXcomp
        );

        scalar xDiffSquared
        (
            xDiff*xDiff
        );

        scalar xRadicand
        (
            circleRadius_*circleRadius_-xDiffSquared
        );

        xRadicandSqrt = sqrt(xRadicand);
    }
}

```

```

        yCenterShift [pointI]=
            point(0.0,varCenterYcomp-
1*p0_.component(vector::Y)()[pointI],0.0);
    }
    else
    {
        xRadicandSqrt = 0.0;
        yCenterShift [pointI]=point(0.0,0.0,0.0);
    }
    velocity[pointI]
=yCenterShift [pointI]+
b*point(0.0,xRadicandSqrt,0.0);

    }

    pointField::operator=
    (
        (p0_
            +velocity
            -p.localPoints()
            )/t.deltaT().value()
    )

```

```

        );

    fixedValuePointPatchField<vector>::
updateCoeffs();
}

void
dynPerCircleApproxGraduallyPointPatchVectorField::
write
(
    Ostream& os
) const
{
    pointPatchField<vector>::write(os);

    os.writeKeyword("circleRadius")

    << circleRadius_ << token::END_STATEMENT << nl;

    os.writeKeyword("xCompInitialCenter")

    << xCompInitialCenter_ << token::END_STATEMENT << nl;

    os.writeKeyword("speed")

    << speed_ << token::END_STATEMENT << nl;

```

```

        os.writeKeyword("yCompFinalCenter")
    << yCompFinalCenter_ << token::END_STATEMENT << nl;

    p0_.writeEntry("p0", os);

    writeEntry("value", os);
}

```

```

// * * * * *

```

```

makePointPatchTypeField

(
    pointPatchVectorField,
    dynPerCircleApproxGraduallyPointPatchVectorField
);

```

```

// * * * * *

```

```

} // End namespace Foam

```

```

// *****

```

A.2 2-D Axisymmetric Tubular Simulations

// S. Alokaily, 3-7-2017.

A.2.1 Case Setup

- Standard Case: wave speed = 5 mm/s, Newtonian fluid N3, maximum relative occlusion = 60%, mesh M2.

`case/0: File U`

- This file contains boundary and initial conditions for the velocity.

FoamFile

{

version 2.0;

format ascii;

class volVectorField;

object U;

}

// ***** //

```
dimensions      [0 1 -1 0 0 0 0];
```

```
internalField   uniform (0 0 0);
```

```
boundaryField
```

```
{
```

```
    leftBoundary // inlet
```

```
    {
```

```
        type          zeroGradient;
```

```
    }
```

```
    rightBoundary // outlet
```

```
    {
```

```
        type          zeroGradient;
```

```
    }
```

```
    centerLine
```

```
    {
```

```
        type          empty;
```

```
    }
```

```

upperWall
{
    type            movingWallNormalVel;
    value           uniform (0 0 0);
}

back
{
    type            wedge;
}

front
{
    type            wedge;
}

}

// ***** //

```

case/0: File p

- This file contains boundary and initial conditions for the pressure.

```
FoamFile
{
    version      2.0;

    format       ascii;

    class        volScalarField;

    object       p;
}

// *****

dimensions      [0 2 -2 0 0 0 0];

internalField    uniform 0;

boundaryField
{
    leftBoundary // inlet
    {
        type      totalPressure;
```



```

        p0            uniform 0;

        gamma         1;

        value         uniform 0;
    }

    rightBoundary // outlet
    {

        type          totalPressure;

        p0            uniform 0;

        gamma         1;

        value         uniform 0;
    }

    centerLine
    {

        type          empty;
    }

    upperWall
    {

        type          zeroGradient;
    }

```

```

    }

    back
    {
        type      wedge;
    }

    front
    {
        type      wedge;
    }

}

// ***** //

;case;0: File pointMotionU

```

- This file contains some input values that control the movement of the upper wall.

```

FoamFile
{

```

```

    version      2.0;

    format       ascii;

    class        pointVectorField;

    object       pointMotionU;
}

// *****

dimensions      [0 1 -1 0 0 0 0];

internalField    uniform (0 0 0);

boundaryField
{
    leftBoundary // inlet
    {
        type      zeroGradient;
    }

    rightBoundary // outlet
    {

```

```

        type            zeroGradient;
    }

    centerLine
    {
        type            fixedValue;
        value            uniform (0 0 0);
    }

    upperWall
    {
        type            dynPerCircleAxisymm;
        circleRadius      0.01500;
        xCompInitialCenter 0.0;
        speed              0.0050;
        yCompFinalCenter  0.01900;
        value              uniform (0 0 0);
    }

```

```

    back
    {
        type            wedge;
    }

    front
    {
        type            wedge;
    }

}

// ***** //

;case_/constant: File dynamicMeshDict

```

- This file contains the choice of mesh motion solver and diffusivity field.

```

FoamFile
{
    version    2.0;

    format     ascii;

    class      dictionary;
}

```

```

        object      motionProperties;

    }

// *****

dynamicFvMesh      dynamicMotionSolverFvMesh;

motionSolverLibs    ( "libfvMotionSolvers.dylib" );

solver              velocityLaplacian;

diffusivity         directional (1 1200 2000);

// *****

```

;*case*:/constant: File transportProperties

- refer to Section A.1.1.

;*case*:/constant: File turbulenceProperties

- refer to Section A.1.1.

`case/constant/polyMesh: File blockMeshDict`

- This file contains input for the generation of the mesh.

```
FoamFile
```

```
{
```

```
    version      2.0;
```

```
    format       ascii;
```

```
    class        dictionary;
```

```
    object       blockMeshDict;
```

```
}
```

```
// ***** //
```

```
convertToMeters 1.0e-03;
```

```
vertices
```

```
(
```

```
    ( 0.00 0.00 0.00)// vertex#0
```

```
    (180.00 0.00 0.00)// vertex#1
```

```
    (180.00 10.00 -0.40)// vertex#2
```

```
    ( 0.00 10.00 -0.40)// vertex#3
```

```
    ( 0.00 0.00 0.00)// vertex#4
```

```

        (180.00  0.00  0.00)// vertex#5

        (180.00 10.00  0.40)// vertex#6

        (  0.00 10.00  0.40)// vertex#7

    );

```

blocks

```

(

    hex (0 1 2 3 0 1 6 7) (540 15 1)

    simpleGrading (1 1 1)// block #0

);

```

edges

```

(

);

```

boundary

```

(

    leftBoundary // inlet

    {

```



```

type patch;

faces

(
    (0 3 7 0)
);
}


rightBoundary // outlet
{

type patch;

faces

(
    (1 2 6 1)
);
}


centerLine

{

type empty;

faces

(

```

```
(0 1 1 0)
```

```
);
```

```
}
```

```
upperWall
```

```
{
```

```
type wall;
```

```
faces
```

```
(
```

```
(2 3 7 6)
```

```
);
```

```
}
```

```
back
```

```
{
```

```
type wedge;
```

```
faces
```

```
(
```

```
(3 2 1 0)
```

```
);
```

```

    }

    front
    {
        type wedge;
        faces
        (

            (0 1 6 7)

        );
    }
);

mergePatchPairs
(
);

// ***** //

;case_/system: File controlDict

```

- This dictionary sets input parameters essential for the creation of the database.

FoamFile

```

{
    version      2.0;

    format       ascii;

    class        dictionary;

    location     "system";

    object       controlDict;
}

// *****

application     transientSimpleDyMFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         38;

deltaT          0.00005;

```

```
writeControl    adjustableRunTime;
```

```
writeInterval   2;
```

```
purgeWrite      0;
```

```
writeFormat     ascii;
```

```
writePrecision  6;
```

```
writeCompression off;
```

```
timeFormat      general;
```

```
timePrecision   6;
```

```
runTimeModifiable true;
```

```
adjustTimeStep  yes;
```

```
maxCo           0.5;
```

```
maxDeltaT 1; // Maximum deltaT in seconds
```

```
libs
```

```
(
```

```
    "dynPerCircleAxisymm.dylib"
```

```
    "movingWallNormalVel.dylib"
```

```
);
```

```
// ***** //
```

```
case:/system: File fvSolution
```

- This file controls the equation solvers, tolerances and algorithms.

```
FoamFile
```

```
{
```

```
    version    2.0;
```

```
    format     ascii;
```

```
    class      dictionary;
```

```
    object     fvSolution;
```

```
}
```

```
// ***** //
```

```

solvers
{
    pcorr
    {
        solver          GAMG;

        tolerance       1e-10;

        relTol          0;

        smoother        GaussSeidel;

        nPreSweeps       0;

        nPostSweeps     2;

        cacheAgglomeration off;

        agglomerator     faceAreaPair;

        nCellsInCoarsestLevel 20;

        mergeLevels      1;

        // maxIter       100;

        minIter          1;
    }

    p
    {
        $pcorr;
    }
}

```

```

        tolerance      1e-10;

        relTol         0;
    }

    pFinal
    {
        $p;

        tolerance      1e-10;

        relTol         0;
    }

    "(U|k|epsilon|omega|nuTilda)"
    {
        solver          smoothSolver;

        smoother        GaussSeidel;

        nSweeps         1;

        tolerance       1e-10;

        relTol          0;

        // maxIter      100;

        minIter         1;
    };

```



```

"(U|k|epsilon|omega|nuTilda)Final"

{

    solver          smoothSolver;

    smoother        GaussSeidel;

    nSweeps          2;

    tolerance        1e-10;

    relTol           0;

    //  maxIter       100;

    minIter          1;

}

cellMotionU

{

    solver          PCG;

    preconditioner   DIC;

    tolerance        1e-10;

    relTol           0;

}

}

```

PISO

```

{
    nCorrectors                2;

    nOuterCorrectors           25;

    nNonOrthogonalCorrectors    0;

    correctPhi                  true;
}

relaxationFactors
{
    p                0.3;
    U                0.7;
    k                0.6;
    omega            0.6;
    epsilon          0.6;

}

// *****

```

`;case_/system: File fvSchemes`

- refer to Section A.1.1.

A.2.2 `dynPerCircleAxisymm BC`.

- Location: `OpenFOAM/OpenFOAM-2.1.x/src`.
- Execution: run `wmake clean` and then `wmake libso`.

`Make/files file`

`dynPerCircleAxisymmPointPatchVectorField.C`

`LIB = $(FOAM_USER_LIBBIN)/dynPerCircleAxisymm`

`Make/options file`

- refer to Section A.1.2

dynPerCircleAxisymmPointPatchVectorField.H

Class

Foam::dynPerCircleAxisymmPointPatchVectorField

Description

Foam::dynPerCircleAxisymmPointPatchVectorField

SourceFiles

dynPerCircleAxisymmPointPatchVectorField.C

```
/*-----*\
```

```
#ifndef dynPerCircleAxisymmPointPatchVectorField_H
```

```
#define dynPerCircleAxisymmPointPatchVectorField_H
```

```
#include "fixedValuePointPatchField.H"
```

```
// *****//
```

```
namespace Foam
```

```

{

% /*-----*\

Class dynPerCircleAxisymmPointPatchVectorField

Declaration

/*-----*\

class dynPerCircleAxisymmPointPatchVectorField

:

public fixedValuePointPatchField<vector>

{

// Private data

scalar circleRadius_;

scalar xCompInitialCenter_;

scalar speed_;

scalar yCompFinalCenter_;

pointField p0_;

public:

//- Runtime type information

TypeName("dynPerCircleAxisymm");

```

```

// Constructors

//- Construct from patch and internal field
dynPerCircleAxisymmPointPatchVectorField
(
    const pointPatch&,
    const DimensionedField<vector, pointMesh>&
);

//- Construct from patch, internal field and dictionary
dynPerCircleAxisymmPointPatchVectorField
(
    const pointPatch&,
    const DimensionedField<vector, pointMesh>&,
    const dictionary&
);

//- Construct by mapping given patchField<vector>
//onto a new patch
dynPerCircleAxisymmPointPatchVectorField
(
    const dynPerCircleAxisymmPointPatchVectorField&,
    const pointPatch&,
    const DimensionedField<vector, pointMesh>&,

```

```

const pointPatchFieldMapper&
);

//- Construct and return a clone
virtual autoPtr<pointPatchField<vector> > clone() const
{
return autoPtr<pointPatchField<vector> >
(
new dynPerCircleAxisymmPointPatchVectorField
(
*this
)
);
}

//- Construct as copy setting internal field
//reference
dynPerCircleAxisymmPointPatchVectorField
(
const dynPerCircleAxisymmPointPatchVectorField&,
const DimensionedField<vector, pointMesh>&
);

//- Construct and return a clone setting internal

```

```

//field reference

virtual autoPtr<pointPatchField<vector> > clone

(

const DimensionedField<vector, pointMesh>& iF

) const

{

return autoPtr<pointPatchField<vector> >

(

new dynPerCircleAxisymmPointPatchVectorField

(

*this,

iF

)

);

}

// Member functions

// Mapping functions

//- Map (and resize as needed) from self

//given a mapping object

virtual void autoMap

(

```



```

const pointPatchFieldMapper&

);

//- Reverse map the given pointPatchField
//onto this pointPatchField

virtual void rmap

(

const pointPatchField<vector>&,

const labelList&

);

// Evaluation functions

//- Update the coefficients associated with the patch field

virtual void updateCoeffs();

//- Write

virtual void write(Ostream&) const;

};

// *****//

} // End namespace Foam

// *****//

#endif

// *****//

```

dynPerCircleAxisymmPointPatchVectorField.C

```
#include "dynPerCircleAxisymmPointPatchVectorField.H"

#include "pointPatchFields.H"

#include "addToRunTimeSelectionTable.H"

#include "Time.H"

#include "polyMesh.H"

#include "mathematicalConstants.H"

// ***** //

namespace Foam

{

// *****Constructors***** //

dynPerCircleAxisymmPointPatchVectorField::

dynPerCircleAxisymmPointPatchVectorField

(

    const pointPatch& p,

    const DimensionedField<vector, pointMesh>& iF

)

:

    fixedValuePointPatchField<vector>(p, iF),
```

```

        circleRadius_(0.0),

        xCompInitialCenter_(0.0),

        speed_(0.0),

        yCompFinalCenter_(0.0),

        p0_(p.localPoints())
    {}

dynPerCircleAxisymmPointPatchVectorField::
dynPerCircleAxisymmPointPatchVectorField
(
    const pointPatch& p,

    const DimensionedField<vector, pointMesh>& iF,

    const dictionary& dict
)
:
    fixedValuePointPatchField<vector>(p, iF, dict),
    circleRadius_(readScalar(dict.lookup("circleRadius"))),
    xCompInitialCenter_(readScalar(dict.lookup
        ("xCompInitialCenter"))),
    speed_(readScalar(dict.lookup("speed"))),
    yCompFinalCenter_(readScalar(dict.lookup

```

```

        ("yCompFinalCenter")))
{
    if (!dict.found("value"))
    {
        updateCoeffs();
    }

    if (dict.found("p0"))
    {
        p0_ = vectorField("p0", dict , p.size());
    }
    else
    {
        p0_ = p.localPoints();
    }
}

```

```

dynPerCircleAxisymmPointPatchVectorField::

```

```

dynPerCircleAxisymmPointPatchVectorField

```

```

(

```

```

    const dynPerCircleAxisymmPointPatchVectorField& ptf,

    const pointPatch& p,

    const DimensionedField<vector, pointMesh>& iF,

    const pointPatchFieldMapper& mapper

)

:

    fixedValuePointPatchField<vector>(ptf, p, iF, mapper),

    circleRadius_(ptf.circleRadius_),

    xCompInitialCenter_(ptf.xCompInitialCenter_),

    speed_(ptf.speed_),

    yCompFinalCenter_(ptf.yCompFinalCenter_),

    p0_(ptf.p0_)

{}

```

```

dynPerCircleAxisymmPointPatchVectorField::

dynPerCircleAxisymmPointPatchVectorField

(

    const dynPerCircleAxisymmPointPatchVectorField& ptf,

    const DimensionedField<vector, pointMesh>& iF

)

```

```

:

    fixedValuePointPatchField<vector>(ptf, iF),

    circleRadius_(ptf.circleRadius_),

    xCompInitialCenter_(ptf.xCompInitialCenter_),

    speed_(ptf.speed_),

    yCompFinalCenter_(ptf.yCompFinalCenter_),

    p0_(ptf.p0_)

{}

// *****Member Functions***** //

void dynPerCircleAxisymmPointPatchVectorField::autoMap
(
    const pointPatchFieldMapper& m
)
{
    fixedValuePointPatchField<vector>::autoMap(m);

    p0_.autoMap(m);
}

```

```

void dynPerCircleAxisymmPointPatchVectorField::rmap
(
    const pointPatchField<vector>& ptf,
    const labellist& addr
)
{
    const dynPerCircleAxisymmPointPatchVectorField& a0Vptf =
refCast<const dynPerCircleAxisymmPointPatchVectorField>(ptf);

    fixedValuePointPatchField<vector>::rmap(a0Vptf, addr);

    p0_.rmap(a0Vptf.p0_, addr);
}

void dynPerCircleAxisymmPointPatchVectorField::updateCoeffs()
{
    if (this->updated())
    {
        return;
    }

    const polyMesh& mesh = this->dimensionedInternalField().mesh();

```

```

const Time& t = mesh.time();

const pointPatch& p = this->patch();

scalar yMax
(
    max(p0_.component(vector::Y)())
);

scalar yCompInitialCenter
(
    yMax+circleRadius_
);

scalar varCenterXcomp
(
    xCompInitialCenter_+speed_*t.value()
);

scalar varCenterYcomp = 0.0;

if
(

```



```

        (t.value() > 0)

        &&

        (t.value() < 1)

    )

    {

        varCenterYcomp =
yCompInitialCenter-t.value()*
(yCompInitialCenter-yCompFinalCenter_);

    }

    else

    {

        varCenterYcomp = yCompFinalCenter_;

    }


    scalar yDiff

    (

        yMax-varCenterYcomp

    );


    scalar yDiffSquared

    (

```

```

        yDiff*yDiff
    );

    scalar yRadicand
    (
        circleRadius_*circleRadius_-yDiffSquared
    );

    scalar yRadicandSqrt
    (
        sqrt(yRadicand)
    );

    scalar lowerBound
    (
        varCenterXcomp-yRadicandSqrt
    );

    scalar upperBound
    (
        varCenterXcomp+yRadicandSqrt

```

```

    );

    scalar b = -1.0;

    pointField yCenterShift(p0_.size(),point(0.0,0.0,0.0));

    pointField velocity(p0_.size(),point(0.0,0.0,0.0));
    pointField vel(p0_.size(),point(0.0,0.0,0.0));
    forAll(p0_,pointI)
    {

    scalar xRadicandSqrt = 0.0;

    if
    (
        (p0_.component(vector::X)()[pointI] > lowerBound)
        &&
        (p0_.component(vector::X)()[pointI] < upperBound)
    )
    {
        //major

```

```

        scalar xDiff
        (
p0_.component(vector::X)()[pointI]-varCenterXcomp
        );

        scalar xDiffSquared
        (
            xDiff*xDiff
        );

        scalar xRadicand
        (
            circleRadius_*circleRadius_-xDiffSquared
        );

        xRadicandSqrt = sqrt(xRadicand);

        yCenterShift[pointI]
        =point(0.0,varCenterYcomp-1*
p0_.component(vector::Y)()[pointI],0.0);

```

```

        }

    else

    {

        xRadicandSqrt = 0.0;

        yCenterShift [pointI]=point(0.0,0.0,0.0);

    }


    velocity [pointI]

    =yCenterShift [pointI]+b*point(0.0,xRadicandSqrt,0.0);


    if

    (p0_.component(vector::Z)() [pointI] > 0.0)

    {

    vel [pointI]=velocity [pointI]+point(0.0,0.0,0.04*

    velocity.component(vector::Y)() [pointI]);

    }

    else

    {

    vel [pointI]=velocity [pointI]+point(0.0,0.0,-0.04*

    velocity.component(vector::Y)() [pointI]);

```

```

    }

}

// p0_ and p.localPoints() will return back the points the
// the deformation

    pointField::operator=
    (
        (
p0_+
            vel
            -p.localPoints()
            )/t.deltaT().value()

        );

    fixedValuePointPatchField<vector>::updateCoeffs();
}

void dynPerCircleAxisymmPointPatchVectorField::write

```

```

(
    Ostream& os
) const
{
    pointPatchField<vector>::write(os);

    os.writeKeyword("circleRadius")

        << circleRadius_ << token::END_STATEMENT << nl;

    os.writeKeyword("xCompInitialCenter")

        << xCompInitialCenter_ << token::END_STATEMENT << nl;

    os.writeKeyword("speed")

        << speed_ << token::END_STATEMENT << nl;

    os.writeKeyword("yCompFinalCenter")

        << yCompFinalCenter_ << token::END_STATEMENT << nl;

    p0_.writeEntry("p0", os);

    writeEntry("value", os);
}

```

```

// *****

```

```

makePointPatchTypeField

```

```

(

```

```

    pointPatchVectorField,

    dynPerCircleAxisymmPointPatchVectorField

);

// ***** //

} // End namespace Foam

// ***** //

```

A.3 2-D Axisymmetric Conical Simulations

// S. Alokaily, 3-7-2017.

A.3.1 Case Setup

- Standard case: wave speed = 2.3 mm/s, Newtonian fluid N3, maximum relative occlusion = 66%, mesh M4.

case/0: File U

- This file contains boundary and initial conditions for the velocity.

```
FoamFile
{
    version      2.0;

    format        ascii;

    class         volVectorField;

    object        U;
}

// ***** //
```



```
dimensions      [0 1 -1 0 0 0 0];
```



```
internalField    uniform (0 0 0);
```



```
boundaryField
{
    leftBoundary // inlet
    {
        type      zeroGradient;
```

```

}

rightBoundary // outlet
{
    type            zeroGradient;

}

centerLine
{
    type            empty;

}

upperWall
{
    type            movingWallNormalVel;
    value           uniform (0 0 0);
}

back
{

```

```

        type            wedge;
    }

    front
    {
        type            wedge;
    }

}

// *****

;case_/0: File p

```

- This file contains boundary and initial conditions for the pressure.

```

FoamFile
{
    version      2.0;

    format       ascii;

    class        volScalarField;

    object       p;
}

```

```

}

// *****

dimensions      [0 2 -2 0 0 0 0];

internalField    uniform 0;

boundaryField
{
    leftBoundary // inlet
    {
        type            totalPressure;
        p0              uniform 0;
        gamma           1;
        value           uniform 0;
    }

    rightBoundary // outlet
    {
        type            totalPressure;
        p0              uniform 0;
    }
}

```

```

        gamma            1;

        value            uniform 0;

    }

    centerLine
    {
        type            empty;
    }

    upperWall
    {
        type            zeroGradient;
    }

    back
    {
        type            wedge;
    }

    front

```

```

        {

            type            wedge;

        }

    }

// *****

;case_/0: File pointMotionU

```

- This file contains some input values that control the movement of the upper wall.

```

FoamFile
{
    version      2.0;

    format       ascii;

    class        pointVectorField;

    object       pointMotionU;

}

// *****

```

```

dimensions      [0 1 -1 0 0 0 0];

internalField    uniform (0 0 0);

boundaryField
{
    leftBoundary // inlet
    {
        type      zeroGradient;
    }

    rightBoundary // outlet
    {

        type      fixedValue;
        value      uniform (0 0 0);
    }

    centerLine
    {

```

```

        type          fixedValue;

        value          uniform (0 0 0);

    }

    upperWall

    {
type          dynPerCircleConicalAxisymmMultiwaves;
circleRadius      0.010;
xCompInitialCenter  0.0085;
speed              0.0023;
yCompFinalCenter   0.0548;
period              20;
numOfWaves          1;
alpha                1.0;
beta                 60;//inclination angle in radian
l                    0.001;
value                uniform (0 0 0);
    }

    back

```



```

        {
            type            wedge;
        }

    front
    {
        type            wedge;
    }

}

// *****

;case_/constant: File dynamicMeshDict

```

- This file contains the choice of mesh motion solver and diffusivity field.

```

FoamFile
{
    version    2.0;

    format     ascii;

    class      dictionary;

    object     motionProperties;
}

```

```

}

// *****

dynamicFvMesh      dynamicMotionSolverFvMesh;

motionSolverLibs   ( "libfvMotionSolvers.so" );

solver             velocityLaplacian;

//velocityLaplacianCoeffs

diffusivity        directional (1 4 7);

// *****

;case_/constant: File transportProperties

```

- refer to Section A.1.1.

```

;case_/constant: File turbulenceProperties

```

- refer to Section A.1.1.

`case/constant/polyMesh: File blockMeshDict`

- This file contains input for the generation of the mesh.

```
FoamFile
```

```
{
```

```
    version      2.0;
```

```
    format       ascii;
```

```
    class        dictionary;
```

```
    object       blockMeshDict;
```

```
}
```

```
// ***** //
```

```
convertToMeters 1.0e-02;
```

```
vertices
```

```
(
```

```
    (0.00  0.00  0.00)// vertex#0
```

```
    (15.00  0.00  0.00)// vertex#1
```

```
    (15.00  0.50 -0.020)// vertex#2
```

```
    (0.00  5.00 -0.20)// vertex#3
```

```
    (0.00  0.00  0.00)// vertex#4
```

```

        (15.00  0.00  0.00)// vertex#5

        (15.00 0.50  0.020)// vertex#6

        (0.00 5.00  0.20)// vertex#7

    );

    blocks

    (

        hex (0 1 2 3 0 1 6 7) (405 27 1)

        simpleGrading (0.1 1 1)// block #0 fffmsh

    );

    edges

    (

    );

    boundary

    (

        leftBoundary \\ inlet

```

```

{
  type patch;

  faces

  (
    (0 3 7 0)
  );
}

rightBoundary // outlet
{
  type patch;

  faces

  (
    (1 2 6 1)
  );
}

centerLine
{
  type empty;

  faces

```

```
(  
    (0 1 1 0)
```

```
);  
}
```

```
upperWall
```

```
{  
  
type wall;  
  
faces
```

```
(  
    (2 3 7 6)
```

```
);  
}
```

```
back
```

```
{  
  
type wedge;  
  
faces
```

```
(  
    (0 1 2 3)
```

```

    );

}

front
{
    type wedge;

    faces
    (

        (0 7 6 1)

    );
}

);

mergePatchPairs
(

);

// ***** //

```

`case_/system: File controlDict`

- This dictionary sets input parameters essential for the creation of the database.

`FoamFile`

```
{  
    version      2.0;  
    format       ascii;  
    class        dictionary;  
    location     "system";  
    object       controlDict;  
}  
  
// *****  
  
application     transientSimpleDyMFoam;  
  
startFrom       startTime;  
  
startTime       0;  
  
stopAt          endTime;
```



```
endTime      65;

deltaT        0.00005;

writeControl  adjustableRunTime;

writeInterval 1;

purgeWrite    0;

writeFormat   ascii;

writePrecision 8;

writeCompression off;

timeFormat    general;

timePrecision 6;

runTimeModifiable true;
```

```

adjustTimeStep  yes;

maxCo           0.5;

maxDeltaT 1; //Maximum deltaT in seconds

libs
(
    "dynPerCircleConicalAxisymmMultiwaves.so"
    "movingWallNormalVel.so"
);

// *****

;case_/system: File fvSolution

```

- This file controls the equation solvers, tolerances and algorithms.

```

FoamFile
{
    version      2.0;

    format       ascii;

    class        dictionary;

```

```

        object      fvSolution;

}

// *****

solvers
{
    pcorr
    {
        solver      GAMG;

        tolerance   1e-10;

        relTol      0;

        smoother     GaussSeidel;

        nPreSweeps   0;

        nPostSweeps  2;

        cacheAgglomeration off;

        agglomerator  faceAreaPair;

        nCellsInCoarsestLevel 20;

        mergeLevels  1;

        // maxIter    100;

        minIter      1;
    }
}

```

```

p
{
    $pcorr;

    tolerance      1e-10;

    relTol         0;
}

pFinal
{
    $p;

    tolerance      1e-10;

    relTol         0;
}

"(U|k|epsilon|omega|nuTilda)"
{
    solver          smoothSolver;

    smoother        GaussSeidel;

    nSweeps         1;

    tolerance       1e-10;

    relTol          0;
}

```

```

        // maxIter      100;

        minIter      1;

};

"(U|k|epsilon|omega|nuTilda)Final"

{

    solver      smoothSolver;

    smoother      GaussSeidel;

    nSweeps      2;

    tolerance     1e-10;

    relTol        0;

    // maxIter      100;

    minIter      1;

}

cellMotionU

{

    solver      PCG;

    preconditioner  DIC;

    tolerance     1e-10;

    relTol        0;

}

```

```
}
```

```
PISO
```

```
{
```

```
    nCorrectors                2;
```

```
    nOuterCorrectors           50;
```

```
    nNonOrthogonalCorrectors   0;
```

```
    correctPhi                  true;
```

```
}
```

```
relaxationFactors
```

```
{
```

```
    p                          0.3;
```

```
    U                          0.7;
```

```
    k                          0.6;
```

```
    omega                      0.6;
```

```
    epsilon                    0.6;
```

```
}
```

```
// ***** //
```

```
;case_/system: File fvSchemes
```

- This file sets the numerical schemes for terms, such as derivatives in equations.

```
FoamFile
```

```
{
```

```
    version      2.0;
```

```
    format       ascii;
```

```
    class        dictionary;
```

```
    location     "system";
```

```
    object       fvSchemes;
```

```
}
```

```
// ***** //
```

```
ddtSchemes
```

```
{
```

```
    default      Euler;
```

```
}
```

```
gradSchemes
```

```

{
    default          Gauss linear;

    grad(p)          Gauss linear;
}

```

divSchemes

```

{
    default          none;

    div(phi,U)       Gauss linearUpwind cellLimited Gauss linear 1;

    div((nuEff*dev(T(grad(U)))) Gauss linear;
}

```

laplacianSchemes

```

{
    default          none;

    laplacian(nu,U)  Gauss linear corrected;

    laplacian(rAU,pcorr) Gauss linear corrected;

    laplacian(rAU,p) Gauss linear corrected;

    laplacian(diffusivity,cellMotionU) Gauss linear uncorrected;

    laplacian(nuEff,U) Gauss linear uncorrected;
}

```



```
interpolationSchemes
```

```
{
```

```
    default      linear;
```

```
    interpolate(HbyA) linear;
```

```
}
```

```
snGradSchemes
```

```
{
```

```
    default      corrected;
```

```
}
```

```
fluxRequired
```

```
{
```

```
    default      no;
```

```
    pcorr        ;
```

```
    p            ;
```

```
}
```

```
// ***** //
```

A.3.2 dynPerCircleConicalAxisymmMultiwaves BC.

- Location: OpenFOAM/OpenFOAM-2.1.x/src.
- Execution: run `wmake clean` and then `wmake libso`.

Make/files file

```
dynPerCircleConicalAxisymmMultiwavesPointPatchVectorField.C
```

```
LIB = $(FOAM_USER_LIBBIN)/dynPerCircleConicalAxisymmMultiwaves
```

Make/options file

- refer to Section A.1.2

dynPerCircleConicalAxisymmMultiwavesPointPatchVectorField.H

Class

```
Foam::dynPerCircleConicalAxisymmMultiwaves
```

```
PointPatchVectorField
```

Description

Foam::dynPerCircleConicalAxisymmMultiwaves

PointPatchVectorField

SourceFiles

dynPerCircleConicalAxisymmMultiwaves

PointPatchVectorField.C

```
/*-----*\
#ifndef dynPerCircleConicalAxisymmMultiwaves
    PointPatchVectorField_H
#define dynPerCircleConicalAxisymmMultiwaves
    PointPatchVectorField_H
#include "fixedValuePointPatchField.H"
// *****//
namespace Foam
{
/*-----*\
Class dynPerCircleConicalAxisymmMultiwaves
    PointPatchVectorField Declaration
\*-----*/
class dynPerCircleConicalAxisymmMultiwaves
```

```

        PointPatchVectorField

:

        public fixedValuePointPatchField<vector>

{

        // Private data

                scalar circleRadius_;

scalar xCompInitialCenter_;

scalar speed_;

                scalar yCompFinalCenter_;

                scalar period_;

                scalar alpha_;

                scalar beta_;

                scalar l_;

                scalar numOfWaves_;

                scalar k;

                scalar sp;

                scalar spp;

                scalar s;

                scalar varCenterYcompPre;

                scalar varCenterXcompPre;

                pointField p0_;

```

```

public:

    //- Runtime type information
    TypeName("dynPerCircleConicalAxisymmMultiwaves");


    // Constructors


    //- Construct from patch and internal field
    dynPerCircleConicalAxisymmMultiwaves
    PointPatchVectorField
    (
        const pointPatch&,
        const DimensionedField<vector, pointMesh>&
    );


    //- Construct from patch, internal field and dictionary
    dynPerCircleConicalAxisymmMultiwaves
    PointPatchVectorField
    (
        const pointPatch&,

```

```

        const DimensionedField<vector, pointMesh>&,
        const dictionary&

    );

//- Construct by mapping given
// patchField<vector> onto a new patch

    dynPerCircleConicalAxisymmMultiwaves
    PointPatchVectorField
    (
        const dynPerCircleConicalAxisymmMultiwaves
        PointPatchVectorField&,
        const pointPatch&,
        const DimensionedField<vector, pointMesh>&,
        const pointPatchFieldMapper&

    );

//- Construct and return a clone

virtual autoPtr<pointPatchField<vector> > clone() const
{
    return autoPtr<pointPatchField<vector> >
    (

```

```

new dynPerCircleConicalAxisymmMultiwaves

    PointPatchVectorField

        (

            *this

        )

    );

}

//- Construct as copy setting internal field reference
dynPerCircleConicalAxisymmMultiwaves

    PointPatchVectorField

    (

const dynPerCircleConicalAxisymmMultiwaves

PointPatchVectorField&,

const DimensionedField<vector, pointMesh>&

    );

//- Construct and return a clone setting

// internal field reference

virtual autoPtr<pointPatchField<vector> > clone

    (

```

```

const DimensionedField<vector, pointMesh>& iF

    ) const

    {

return autoPtr<pointPatchField<vector> >

    (

new dynPerCircleConicalAxisymmMultiwaves

    PointPatchVectorField

    (

        *this,

        iF

    )

    );

    }

// Member functions


// Mapping functions


//- Map (and resize as needed) from self

// given a mapping object

virtual void autoMap

(

```



```

        const pointPatchFieldMapper&
    );

    //- Reverse map the given
    // pointPatchField onto this
    // pointPatchField
    virtual void rmap
    (
        const pointPatchField<vector>&,
        const labelList&
    );

    // Evaluation functions

    //- Update the coefficients
    // associated with the patch field
    virtual void updateCoeffs();

    //- Write

```

```

        virtual void write(Ostream&) const;

};

```

```

// *****

```

```

} // End namespace Foam

```

```

// *****

```

```

#endif

```

```

// *****

```

dynPerCircleConicalAxisymmMultiwavesPointPatchVectorField.C

```

/*-----*\

```

```

#include "dynPerCircleConicalAxisymmMultiwaves

```

```

        PointPatchVectorField.H"

```

```

#include "pointPatchFields.H"

```

```

#include "addToRunTimeSelectionTable.H"

```

```

#include "Time.H"

```

```

#include "polyMesh.H"

```

```

#include "mathematicalConstants.H"

```

```

#include <stdio.h>

```

```

#include <iostream>      // std::cout

#include <cmath>          // std::abs

#define _USE_MATH_DEFINES

#include <math.h>


// *****

namespace Foam

{

// *****Constructors*****

dynPerCircleConicalAxisymmMultiwavesPointPatchVectorField::
dynPerCircleConicalAxisymmMultiwavesPointPatchVectorField
(
    const pointPatch& p,
    const DimensionedField<vector, pointMesh>& iF
)
:
    fixedValuePointPatchField<vector>(p, iF),
    circleRadius_(0.0),
    xCompInitialCenter_(0.0),
    speed_(0.0),

```

```

    yCompFinalCenter_(0.0),

    period_(0.0),

    numOfWaves_(0.0),

    alpha_(0.0),

    beta_(0.0),

    l_(0.0),

    p0_(p.localPoints())

    {}

```

```

dynPerCircleConicalAxisymmMultiwavesPointPatchVectorField::
dynPerCircleConicalAxisymmMultiwavesPointPatchVectorField
(
    const pointPatch& p,
    const DimensionedField<vector, pointMesh>& iF,
    const dictionary& dict
)
:
    fixedValuePointPatchField<vector>(p, iF, dict),
    circleRadius_(readScalar(dict.lookup("circleRadius"))),
    xCompInitialCenter_(readScalar
        (dict.lookup("xCompInitialCenter"))),

```

```

speed_(readScalar(dict.lookup("speed"))),
yCompFinalCenter_(readScalar
(dict.lookup("yCompFinalCenter"))),
numOfWaves_(readScalar(dict.lookup("numOfWaves"))),
alpha_(readScalar(dict.lookup("alpha"))),
beta_(readScalar(dict.lookup("beta"))),
l_(readScalar(dict.lookup("l"))),
period_(readScalar(dict.lookup("period")))
{
    if (!dict.found("value"))
    {
        updateCoeffs();
    }

    if (dict.found("p0"))
    {
        p0_ = vectorField("p0", dict , p.size());
    }
    else
    {
        p0_ = p.localPoints();
    }
}

```

```

    }
}

dynPerCircleConicalAxisymmMultiwavesPointPatchVectorField::
dynPerCircleConicalAxisymmMultiwavesPointPatchVectorField
(
    const dynPerCircleConicalAxisymmMultiwaves
        PointPatchVectorField& ptf,
    const pointPatch& p,
    const DimensionedField<vector, pointMesh>& iF,
    const pointPatchFieldMapper& mapper
)
:
    fixedValuePointPatchField<vector>(ptf, p, iF, mapper),
    circleRadius_(ptf.circleRadius_),
    xCompInitialCenter_(ptf.xCompInitialCenter_),
    speed_(ptf.speed_),
    yCompFinalCenter_(ptf.yCompFinalCenter_),
    period_(ptf.period_),
    alpha_(ptf.alpha_),

```

```

    beta_(ptf.beta_),
    l_(ptf.l_),
    numOfWaves_(ptf.numOfWaves_),
    p0_(ptf.p0_)
{}

```

```

dynPerCircleConicalAxisymmMultiwavesPointPatchVectorField::
dynPerCircleConicalAxisymmMultiwavesPointPatchVectorField
(
    const dynPerCircleConicalAxisymmMultiwaves
        PointPatchVectorField& ptf,
    const DimensionedField<vector, pointMesh>& iF
)
:
    fixedValuePointPatchField<vector>(ptf, iF),
    circleRadius_(ptf.circleRadius_),
    xCompInitialCenter_(ptf.xCompInitialCenter_),
    speed_(ptf.speed_),
    yCompFinalCenter_(ptf.yCompFinalCenter_),
    period_(ptf.period_),

```

```

        alpha_(ptf.alpha_),
        beta_(ptf.beta_),
        l_(ptf.l_),
        numOfWaves_(ptf.numOfWaves_),
        p0_(ptf.p0_)

    {}

// *****Member Functions***** //

void dynPerCircleConicalAxisymmMultiwaves
    PointPatchVectorField::autoMap
(
    const pointPatchFieldMapper& m
)
{
    fixedValuePointPatchField<vector>::autoMap(m);

    p0_.autoMap(m);
}

void dynPerCircleConicalAxisymmMultiwaves

```



```

        PointPatchVectorField::rmap
(
    const pointPatchField<vector>& ptf,
    const labelList& addr
)
{
    const dynPerCircleConicalAxisymmMultiwaves

        PointPatchVectorField& aOVptf =
refCast<const dynPerCircleConicalAxisymmMultiwaves

        PointPatchVectorField>(ptf);

    fixedValuePointPatchField<vector>::rmap(aOVptf, addr);

    p0_.rmap(aOVptf.p0_, addr);

}

void dynPerCircleConicalAxisymmMultiwaves

    PointPatchVectorField::updateCoeffs()
{

```

```

    if (this->updated())
    {

        return;
    }

    const polyMesh& mesh = this->
        dimensionedInternalField().mesh();

    const Time& t = mesh.time();

    const pointPatch& p = this->patch();

    double gamma=(beta_ *  M_PI/180.0);

    scalar yMax
    (
        max(p0_.component(vector::Y)())
    );

    scalar yMin
    (
        min(p0_.component(vector::Y)())
    );

```

```

scalar xMax
(
    max(p0_.component(vector::X)())
);

scalar xMin
(
    min(p0_.component(vector::X)())
);

double param1
(
    (yMax-yMin)/xMax
);

double theta=atan(param1);

tensor R(cos(theta),-1*sin(theta),0.0,1*sin(theta),
    cos(theta),0.0,0.0,0.0,1.0);

tensor RT(cos(theta),1*sin(theta),0.0,-1*sin(theta),
    cos(theta),0.0,0.0,0.0,1.0);

pointField q1(p0_.size(),point(0.0,0.0,0.0));
pointField q2(p0_.size(),point(0.0,0.0,0.0));
pointField q0(p0_.size(),point(0.0,0.0,0.0));

forAll(p0_,pointJ)

```

```

{

q1[pointJ]=p0_[pointJ]-
    point(xMin,yMax,p0_.component(vector::Z())[pointJ]);
q2[pointJ]=q1[pointJ] & RT;
q0[pointJ]=q2[pointJ]+
    point(xMin,yMax,p0_.component(vector::Z())[pointJ]);
}

```

```

scalar yCompInitialCenter

```

```

(
    yMax+circleRadius_
);

```

```

scalar xMaxq0

```

```

(
    max(q0.component(vector::X)())
);

```

```

pointField M(q0.size(),point(0.0,0.0,0.0));

```

```

double tFinalCenterXcomp=(xMaxq0/speed_);

    if
    (
        t.value() <=tFinalCenterXcomp
    )
    {
        k = 0.0;
    }


    if
    (
        t.value() ==t.deltaT().value()
    )
    {
        sp=xMaxq0;
    }


for( int i = numOfWaves_; i >= 1; i=i-1 )

```

```

{

    for( int j=i; j<=i; j=j+1 )

    {

        if(t.value() > (numOfWaves_-i)*period_)

        {

            double time=t.value()-(numOfWaves_-j)*period_;

            scalar varCenterXcomp

                (

                    xCompInitialCenter_+speed_*time

                );

            scalar varCenterYcomp = 0.0;

            if(time+k*period_ == t.value()) {s=sp;} else {s=l_;}

```

```

if
(
    (time > 0)

    &&

    (time < 1)
)
{

varCenterYcomp = yCompInitialCenter-time*

    (yCompInitialCenter-yCompFinalCenter_);

}
else
{

    if
    (
        (mag(s) < 1_)
    )

    {

varCenterYcomp =varCenterYcompPre+alpha_*

```

```

        speed_*t.deltaT().value()*sin(gamma);
varCenterXcomp=varCenterXcompPre+alpha_*
        speed_*t.deltaT().value()*cos(gamma);
        s=sp;
    }
else
    {
        varCenterYcomp = yCompFinalCenter_;
    }

}

if
(
    varCenterYcomp >= yCompInitialCenter
)

{

    varCenterYcomp = yCompInitialCenter;
}

```



```
scalar yDiff  
  
    (  
  
        yMax-varCenterYcomp  
  
    );
```

```
scalar yDiffSquared  
  
    (  
  
        yDiff*yDiff  
  
    );
```

```
scalar yRadicand  
  
    (  
  
        mag(circleRadius_*circleRadius_-yDiffSquared)  
  
    );
```

```
scalar yRadicandSqrt  
  
    (  
  
        sqrt(yRadicand)  
  
    );
```

```

    scalar lowerBound
        (
            varCenterXcomp-yRadicandSqrt
        );

    scalar upperBound
        (
            varCenterXcomp+yRadicandSqrt
        );

    scalar Xcenterw=(varCenterXcomp*cos(theta)+
        (varCenterYcomp-yMax)*sin(theta));
    scalar Ycenterw=(-1.0*varCenterXcomp*
        sin(theta)+(varCenterYcomp-yMax)*cos(theta)+yMax);

    scalar chordLength=(upperBound-lowerBound);

    scalar Xlowerw=lowerBound*cos(theta);
    scalar Xupperw=upperBound*cos(theta);
    scalar Ylowerw=((yMin-yMax)/xMax)*Xlowerw+yMax;
    scalar Yupperw=((yMin-yMax)/xMax)*Xupperw+yMax;
    scalar Xbar=(lowerBound+upperBound)/2.0;
    scalar Xchordcenterw=Xcenterw;

```

```

scalar Ychordcenterw=((yMin-yMax)/xMax)*Xchordcenterw+yMax;;

scalar gap=varCenterYcomp-circleRadius_;

scalar Xgapw=Xchordcenterw;

scalar Ygapw=Ycenterw-circleRadius_;

scalar Amplitudew=circleRadius_-(varCenterYcomp-yMax);

scalar R0w=(1.0-(Ygapw/Ychordcenterw))*100.0;

scalar DistanceOfAcwCenterFromPylorus=(xMax-Xcenterw);

scalar DistanceOfAcwXupperFromPylorus=(xMax-Xupperw);

```

```

scalar b = -1.0;

```

```

pointField yCenterShift(q0.size(),point(0.0,0.0,0.0));

pointField velocity(q0.size(),point(0.0,0.0,0.0));

pointField q3(q0.size(),point(0.0,0.0,0.0));

pointField q4(q0.size(),point(0.0,0.0,0.0));

pointField q5(q0.size(),point(0.0,0.0,0.0));

```

```

forAll(q0,pointI)

```

```

{

```

```

scalar xRadicandSqrt = 0.0;

    if
    (
        (q0.component(vector::X())[pointI] > lowerBound)
        &&
        (q0.component(vector::X())[pointI] < upperBound)
    )
    {

        scalar xDiff
        (
            q0.component(vector::X())[pointI]-varCenterXcomp
        );

        scalar xDiffSquared
        (
            xDiff*xDiff
        );

        scalar xRadicand
        (

```

```

        circleRadius_*circleRadius_-xDiffSquared
    );

xRadicandSqrt = sqrt(xRadicand);

yCenterShift[pointI] = point(0.0,varCenterYcomp-1*
        q0.component(vector::Y())[pointI],0.0);
    }

else

    {
        xRadicandSqrt = 0.0;
        yCenterShift[pointI]=point(0.0,0.0,0.0);
    }

    velocity[pointI]=
        yCenterShift[pointI]+b*point(0.0,xRadicandSqrt,0.0);

    q3[pointI]=velocity[pointI] & R;

if
    (p0_.component(vector::Z())[pointI] > 0.0)

```

```

{
q4[pointI]=q3[pointI]+
    point(0.0,0.0,0.04*q3.component(vector::Y())[pointI]);
}

else

{
q4[pointI]=q3[pointI]+
    point(0.0,0.0,-0.04*q3.component(vector::Y())[pointI]);
}

}

//  p0_ and p.localPoints() will return back the points the
//  the deformation

if( mag(s) >= l_ ) {s=xMaxq0-upperBound;;}

if( mag(s) <= l_ ) {k=(numOfWaves_ - j);}

    M=M+q4;

    pointField::operator=

```

```

        (
            (
                p0_+
                M
                -p.localPoints()
                )/t.deltaT().value()

        );

if (time+k*period_ == t.value())
{
    varCenterYcompPre = varCenterYcomp;
    varCenterXcompPre = varCenterXcomp; sp=s;
}

}

}

}

```

```

fixedValuePointPatchField<vector>::updateCoeffs();

}

void dynPerCircleConicalAxisymmMultiwaves
    PointPatchVectorField::write
(
    Ostream& os
) const
{
    pointPatchField<vector>::write(os);
    os.writeKeyword("circleRadius")
        << circleRadius_ << token::END_STATEMENT << nl;
    os.writeKeyword("xCompInitialCenter")
        << xCompInitialCenter_ << token::END_STATEMENT << nl;
    os.writeKeyword("speed")
        << speed_ << token::END_STATEMENT << nl;
    os.writeKeyword("yCompFinalCenter")
        << yCompFinalCenter_ << token::END_STATEMENT << nl;
    os.writeKeyword("period")

```



```

        << period_ << token::END_STATEMENT << nl;

os.writeKeyword("alpha")

        << alpha_ << token::END_STATEMENT << nl;

os.writeKeyword("beta")

        << beta_ << token::END_STATEMENT << nl;

os.writeKeyword("l")

        << l_ << token::END_STATEMENT << nl;

os.writeKeyword("numOfWaves")

        << numOfWaves_ << token::END_STATEMENT << nl;

p0_.writeEntry("p0", os);

writeEntry("value", os);

}

// ***** //

makePointPatchTypeField

(

    pointPatchVectorField,

    dynPerCircleConicalAxisymmMultiwaves

    PointPatchVectorField

);

```

```
// ***** //
```

```
} // End namespace Foam
```

```
// ***** //
```


Appendix B

Closed Outlet Simulations

// S. Alokaily, 3-7-2017.

B.1 Circular ACWs Simulations

B.1.1 Case Setup

- wave speed = 2.3 mm/s, Newtonian fluid N3, maximum relative occlusion = 80%, mesh M3.

case/0: File U

- This file contains boundary and initial conditions for the velocity.

```
FoamFile
{
    version      2.0;

    format       ascii;

    class        volVectorField;

    object       U;
}

// ***** //
```



```
dimensions      [0 1 -1 0 0 0 0];
```



```
internalField    uniform (0 0 0);
```



```
boundaryField
```



```
{
```



```
    leftBoundary  \ \ inlet
```



```
    {
```



```
        type              zeroGradient;
```

```
}
```

```
rightBoundary \\ outlet
```

```
{
```

```
    type          fixedValue;
```

```
    value         uniform (0 0 0);
```

```
}
```

```
centerLine
```

```
{
```

```
    type          empty;
```

```
}
```

```
upperWall
```

```
{
```

```
    type          movingWallNormalVel;
```

```
    value         uniform (0 0 0);
```

```
}
```

```
back
```

```
{
```

```

        type            wedge;
    }

    front
    {
        type            wedge;
    }

}

// *****

;case_/0: File p

```

- This file contains boundary and initial conditions for the pressure.

```

FoamFile
{
    version      2.0;

    format       ascii;

    class        volScalarField;

    object       p;
}

```

```

}

// *****

dimensions      [0 2 -2 0 0 0 0];

internalField    uniform 0;

boundaryField
{
    leftBoundary // inlet
    {
        type          totalPressure;

        p0             uniform 0;

        gamma          1;

        value          uniform 0;
    }

    rightBoundary // outlet
    {

        type          zeroGradient;
    }
}

```



```
}
```

```
centerLine
```

```
{
```

```
    type          empty;
```

```
}
```

```
upperWall
```

```
{
```

```
    type          zeroGradient;
```

```
}
```

```
back
```

```
{
```

```
    type          wedge;
```

```
}
```

```
front
```

```
{
```

```
    type          wedge;
```

```
}
```

```
}
```

```
// ***** //
```

```
;case 0: File pointMotionU
```

- This file contains some input values that control the movement of the upper wall.

```
FoamFile
```

```
{
```

```
    version      2.0;
```

```
    format       ascii;
```

```
    class        pointVectorField;
```

```
    object       pointMotionU;
```

```
}
```

```
// ***** //
```

```
dimensions      [0 1 -1 0 0 0 0];
```

```
internalField    uniform (0 0 0);
```

```

boundaryField
{
    leftBoundary \\ inlet
    {
        type            fixedValue;
        value            uniform (0 0 0);
    }

    rightBoundary // outlet
    {
        type            fixedValue;
        value            uniform (0 0 0);
    }

    centerLine
    {
        type            fixedValue;
        value            uniform (0 0 0);
    }

    upperWall

```

```

{
    type                dynPerCircleConicalAxisymmMultiwaves;

    circleRadius        0.01;

    xCompInitialCenter  0.0085;

    speed               0.0023;

    yCompFinalCenter    0.05375;

    period              20;

    numOfWaves          100;

    alpha               1.0;

    beta               45;

    l                   0.0001;

    value               uniform (0 0 0);
}

```

back

```

{
    type                wedge;
}

```

front

```

{

```

```

        type            wedge;

    }

}

// *****

;case;/constant: File dynamicMeshDict

```

- This file contains the choice of mesh motion solver and diffusivity field.

```

FoamFile
{
    version      2.0;

    format       ascii;

    class        dictionary;

    object       motionProperties;
}

// *****

dynamicFvMesh      dynamicMotionSolverFvMesh;

motionSolverLibs    ( "libfvMotionSolvers.so" );

```

```
solver          velocityLaplacian;
```

```
//velocityLaplacianCoeffs
```

```
diffusivity      directional (1 4 7);
```

```
// ***** //
```

```
;case_/constant: File transportProperties
```

- refer to Section A.1.1.

```
;case_/constant: File turbulenceProperties
```

- refer to Section A.1.1.

```
;case_/constant/polyMesh: File blockMeshDict
```

- This file contains input for the generation of the mesh.

```
FoamFile
```

```
{
```

```
    version      2.0;
```

```

    format      ascii;

    class       dictionary;

    object      blockMeshDict;
}

// *****

convertToMeters 1.0e-02;

vertices
(
    (0.00 0.00 0.00) // vertex#0
    (15.00 0.00 0.00) // vertex#1
    (15.00 0.50 -0.020) // vertex#2
    (0.00 5.00 -0.20) // vertex#3
    (0.00 0.00 0.00) // vertex#4
    (15.00 0.00 0.00) // vertex#5
    (15.00 0.50 0.020) // vertex#6
    (0.00 5.00 0.20) // vertex#7
);

```

```

blocks

(

    hex (0 1 2 3 0 1 6 7) (270 18 1)

    simpleGrading (0.1 1 1)// block #0 ffmsh

);

edges

(

);

boundary

(

    leftBoundary // inlet

    {

        type patch;

        faces

        (

            (0 3 7 0)

        );

    };

```



```

}

rightBoundary // outlet
{
  type patch;
  faces
  (
    (1 2 6 1)
  );
}

centerLine
{
  type empty;
  faces
  (
    (0 1 1 0)

  );
}

```

```
upperWall  
  
{  
  
type wall;  
  
faces  
  
(  
  
    (2 3 7 6)  
  
);  
}
```

```
back  
  
{  
  
type wedge;  
  
faces  
  
(  
  
    (0 1 2 3)  
  
);  
}
```

```
front  
  
{  
  
type wedge;
```

```

    faces
    (
        (0 7 6 1)
    );
}
);

```

```

mergePatchPairs
(
);

```

```

// *****

```

```

;case_/system: File controlDict

```

- This dictionary sets input parameters essential for the creation of the database.

```

FoamFile
{
    version      2.0;

    format       ascii;

    class        dictionary;

```

```

        location    "system";

        object      controlDict;
    }

    // *****

    application      transientSimpleDyMFoam;

    startFrom        startTime;

    startTime         0;

    stopAt            endTime;

    endTime           65;

    deltaT            0.00005;

    writeControl       adjustableRunTime;

    writeInterval     1;

```

```
purgeWrite      0;

writeFormat      ascii;

writePrecision   8;

writeCompression off;

timeFormat       general;

timePrecision    6;

runTimeModifiable true;

adjustTimeStep   yes;

maxCo            0.5;

maxDeltaT 1; //Maximum deltaT in seconds

libs
```

```
(
    "dynPerCircleConicalAxisymmMultiwaves.so"
    "movingWallNormalVel.so"
);

// ***** //
```

```
;case_/system: File fvSolution
```

- This file controls the equation solvers, tolerances and algorithms.

```
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    object       fvSolution;
}

// ***** //
```

```
solvers
{
    pcorr
```

```

{
    solver          GAMG;

    tolerance       1e-10;

    relTol          0;

    smoother        GaussSeidel;

    nPreSweeps      0;

    nPostSweeps     2;

    cacheAgglomeration off;

    agglomerator     faceAreaPair;

    nCellsInCoarsestLevel 20;

    mergeLevels      1;

    // maxIter       100;

    minIter          1;
}

```

p

```

{
    $pcorr;

    tolerance       1e-10;

    relTol          0;
}

```

```
pFinal
```

```
{
```

```
    $p;
```

```
    tolerance      1e-10;
```

```
    relTol         0;
```

```
}
```

```
"(U|k|epsilon|omega|nuTilda)"
```

```
{
```

```
    solver          smoothSolver;
```

```
    smoother        GaussSeidel;
```

```
    nSweeps         1;
```

```
    tolerance      1e-10;
```

```
    relTol         0;
```

```
    // maxIter     100;
```

```
    minIter        1;
```

```
};
```

```
"(U|k|epsilon|omega|nuTilda)Final"
```

```
{
```

```
    solver          smoothSolver;
```

```
    smoother        GaussSeidel;
```



```

        nSweeps          2;

        tolerance        1e-10;

        relTol           0;

        //  maxIter       100;

        minIter          1;
    }

    cellMotionU
    {
        solver            PCG;

        preconditioner     DIC;

        tolerance          1e-10;

        relTol             0;
    }
}

```

PISO

```

{
    nCorrectors           2;

    nOuterCorrectors      3;

    nNonOrthogonalCorrectors 0;
}

```

```

        correctPhi                true;
    }

    relaxationFactors
    {
        p                0.3;
        U                0.3;
        k                0.6;
        omega            0.6;
        epsilon          0.6;
    }

    // *****
;case_/system: File fvSchemes

```

- refer to Section A.1.1.

B.1.2 dynPerCircleConicalAxisymmMultiwaves BC.

- refer to Section A.3.2.

B.2 Parabolic ACWs Simulations

B.2.1 Case Setup

- wave speed = 2.3 mm/s, Newtonian fluid N3, maximum relative occlusion = 80%, mesh M3.

`case_i/0`: File U

- refer to Section B.1.1.

`case_i/0`: File p

- refer to Section B.1.1.

case_0: File pointMotionU

- This file contains some input values that control the movement of the upper wall.

FoamFile

{

version 2.0;

format ascii;

class pointVectorField;

object pointMotionU;

}

// ***** //

dimensions [0 1 -1 0 0 0 0];

internalField uniform (0 0 0);

boundaryField

{

leftBoundary // inlet

```

{
    type          fixedValue;
    value          uniform (0 0 0);
}

rightBoundary // outlet
{
    type          fixedValue;
    value          uniform (0 0 0);
}

centerLine
{
    type          fixedValue;
    value          uniform (0 0 0);
}

upperWall
{
    type          dynPerParabolicConicalAxisymmMultiwaves;

```

```

        chordLength      0.007;

        xCompInitialVertex  0.0041;

        speed              0.0023;

        yCompFinalVertex    0.0455;

        period              20;

        numOfWaves          100;

        alpha               1.0;

        beta                45;

        l                   0.0039;

        shift                0.044;

        T                   65;

        value                uniform (0 0 0);
    }

```

```

back

{

    type                wedge;

}

```

```

front

{

```

```

        type            wedge;

    }

}

// *****

;case;/constant: File dynamicMeshDict

```

- This file contains the choice of mesh motion solver and diffusivity field.

```

FoamFile
{
    version      2.0;

    format       ascii;

    class        dictionary;

    object       motionProperties;
}

// *****

dynamicFvMesh      dynamicMotionSolverFvMesh;

```

```
motionSolverLibs    ( "libfvMotionSolvers.so" );
```

```
solver              velocityLaplacian;
```

```
//velocityLaplacianCoeffs
```

```
diffusivity         directional (1 3 6);
```

```
// ***** //
```

```
;case_/constant: File transportProperties
```

- refer to Section A.1.1.

```
;case_/constant: File turbulenceProperties
```

- refer to Section A.1.1.

```
;case_/constant/polyMesh: File blockMeshDict
```

- This file contains input for the generation of the mesh.

```
FoamFile
```

```
{
```



```

    version      2.0;

    format       ascii;

    class        dictionary;

    object       blockMeshDict;
}

// *****

convertToMeters 1.0e-02;

vertices
(
    (0.00  0.00  0.00)    // vertex#0
    (15.00  0.00  0.00)   // vertex#1
    (15.00  0.50 -0.020)  // vertex#2
    (0.00  5.00 -0.20)    // vertex#3
    (0.00  0.00  0.00)    // vertex#4
    (15.00  0.00  0.00)   // vertex#5
    (15.00  0.50  0.020)  // vertex#6
    (0.00  5.00  0.20)    // vertex#7
);

```

```

blocks

(

    hex (0 1 2 3 0 1 6 7) (270 18 1)

    simpleGrading (0.15 1 1)// block #0 ffmsh

);

edges

(

);

boundary

(

    leftBoundary // inlet

    {

        type patch;

        faces

        (

            (0 3 7 0)

        );

    };

```

```

}

rightBoundary // outlet
{
  type patch;
  faces
  (
    (1 2 6 1)
  );
}

centerLine
{
  type empty;
  faces
  (
    (0 1 1 0)

  );
}

```

```
upperWall  
  
{  
  
type wall;  
  
faces  
  
(  
  
    (2 3 7 6)  
  
);  
}
```

```
back  
  
{  
  
type wedge;  
  
faces  
  
(  
  
    (0 1 2 3)  
  
);  
}
```

```
front  
  
{  
  
type wedge;
```

```

    faces
    (
        (0 7 6 1)
    );
}
);

```

```

mergePatchPairs
(
);

```

```

// *****

```

```

;case_/system: File controlDict

```

- This dictionary sets input parameters essential for the creation of the database.

```

FoamFile
{
    version      2.0;

    format       ascii;

    class        dictionary;

```

```

        location      "system";

        object        controlDict;
    }

    // *****

    application        transientSimpleDyMFoam;

    startFrom          startTime;

    startTime           0;

    stopAt              endTime;

    endTime             66;

    deltaT              0.00005;

    writeControl         adjustableRunTime;

    writeInterval        1;

```

```
purgeWrite      0;

writeFormat      ascii;

writePrecision   8;

writeCompression off;

timeFormat       general;

timePrecision    6;

runTimeModifiable true;

adjustTimeStep   yes;

maxCo            0.5;

maxDeltaT 1; // Maximum deltaT in seconds

libs
```

```
(
    "dynPerParabolicConicalAxisymmMultiwaves.so"
    "movingWallNormalVel.so"
);

// ***** //
```

```
;case_/system: File fvSolution
```

- refer to Section B.1.1.

```
;case_/system: File fvSchemes
```

- refer to Section A.1.1.

B.2.2 dynPerParabolicConicalAxisymmMultiwaves BC.

- Location: OpenFOAM/OpenFOAM-2.1.x/src.
- Execution: run `wmake clean` and then `wmake libso`.

Make/files file

```
dynPerParabolicConicalAxisymmMultiwavesPointPatchVectorField.C
```



```
LIB = $(FOAM_USER_LIBBIN)/dynPerParabolicConicalAxisymmMultiwaves
```

Make/options file

- refer to Section A.1.2

dynPerParabolicConicalAxisymmMultiwavesPointPatchVectorField.H

Class

```
Foam::dynPerParabolicConicalAxisymmMultiwaves  
    PointPatchVectorField
```

Description

```
Foam::dynPerParabolicConicalAxisymmMultiwaves  
    PointPatchVectorField
```

SourceFiles

```
dynPerParabolicConicalAxisymmMultiwaves  
PointPatchVectorField.C
```

```

/*-----*\

#ifndef dynPerParabolicConicalAxisymmMultiwaves

    PointPatchVectorField_H

#define dynPerParabolicConicalAxisymmMultiwaves

    PointPatchVectorField_H

#include "fixedValuePointPatchField.H"

// *****//

namespace Foam
{

/*-----*\

    Class dynPerParabolicConicalAxisymmMultiwaves

        PointPatchVectorField Declaration

/*-----*\

class dynPerParabolicConicalAxisymmMultiwaves

    PointPatchVectorField

:

```

```

public fixedValuePointPatchField<vector>

{
    // Private data

    scalar xCompInitialVertex_;

    scalar chordLength_;

    scalar speed_;

    scalar yCompFinalVertex_;

    scalar period_;

    scalar numOfWaves_;

    scalar alpha_;

    scalar beta_;

    scalar l_;

    scalar shift_;

    scalar T_;

    scalar k;

    scalar sp;

    scalar s;

    scalar m;

    scalar varVertexYcompPre;

    scalar varVertexXcompPre;

    pointField p0_;

```

```

public:

    //- Runtime type information
    TypeName("dynPerParabolicConicalAxisymmMultiwaves");


    // Constructors


    //- Construct from patch and internal field
    dynPerParabolicConicalAxisymmMultiwaves
    PointPatchVectorField
    (
        const pointPatch&,
        const DimensionedField<vector, pointMesh>&
    );


    //- Construct from patch, internal field and dictionary
    dynPerParabolicConicalAxisymmMultiwaves
    PointPatchVectorField
    (
        const pointPatch&,

```

```

const DimensionedField<vector, pointMesh>&,

const dictionary&

);

//- Construct by mapping given
// patchField<vector> onto a new patch
dynPerParabolicConicalAxisymmMultiwaves
PointPatchVectorField
(
    const dynPerParabolicConicalAxisymmMultiwaves
        PointPatchVectorField&,
    const pointPatch&,
    const DimensionedField<vector, pointMesh>&,
    const pointPatchFieldMapper&
);

//- Construct and return a clone
virtual autoPtr<pointPatchField<vector> > clone() const
{
    return autoPtr<pointPatchField<vector> >
        (

```

```

        new dynPerParabolicConicalAxisymmMultiwaves
            PointPatchVectorField
                (
                    *this
                )
    );
}

//- Construct as copy setting internal field reference
dynPerParabolicConicalAxisymmMultiwaves
PointPatchVectorField
(
    const dynPerParabolicConicalAxisymmMultiwaves
        PointPatchVectorField&,
    const DimensionedField<vector, pointMesh>&
);

//- Construct and return a clone
//  setting internal field reference
virtual autoPtr<pointPatchField<vector> > clone
(

```

```

        const DimensionedField<vector, pointMesh>& iF

    ) const

    {

        return autoPtr<pointPatchField<vector> >

        (

            new dynPerParabolicConicalAxisymmMultiwaves

                PointPatchVectorField

                (

                    *this,

                    iF

                )

            );

    }

// Member functions


// Mapping functions


//- Map (and resize as needed) from self given a mapping object

virtual void autoMap

(

    const pointPatchFieldMapper&

```

```

    );

    //- Reverse map the given pointPatchField
    // onto this pointPatchField
    virtual void rmap
    (
        const pointPatchField<vector>&,
        const labelList&
    );

    // Evaluation functions

    //- Update the coefficients associated with the patch field
    virtual void updateCoeffs();

    //- Write
    virtual void write(Ostream&) const;
};

```



```
// *****//
```

```
} // End namespace Foam
```

```
// *****//
```

```
#endif
```

```
// *****//
```

dynPerParabolicConicalAxisymmMultiwavesPointPatchVectorField.C

```
/*-----*\
```

```
#include "dynPerParabolicConicalAxisymmMultiwaves
```

```
PointPatchVectorField.H"
```

```
#include "pointPatchFields.H"
```

```
#include "addToRunTimeSelectionTable.H"
```

```
#include "Time.H"
```

```
#include "polyMesh.H"
```

```
#include "mathematicalConstants.H"
```

```

#include <stdio.h>


// *****

namespace Foam
{
// *****Constructors*****

dynPerParabolicConicalAxisymmMultiwaves
PointPatchVectorField::
dynPerParabolicConicalAxisymmMultiwaves
PointPatchVectorField
(
    const pointPatch& p,
    const DimensionedField<vector, pointMesh>& iF
)
:
    fixedValuePointPatchField<vector>(p, iF),
    xCompInitialVertex_(0.0),
    chordLength_(0.0),
    speed_(0.0),

```

```

    yCompFinalVertex_(0.0),

    period_(0.0),

    numOfWaves_(0.0),

    alpha_(0.0),

    beta_(0.0),

    l_(0.0),

    shift_(0.0),

    T_(0.0),

    p0_(p.localPoints())

    {}

```

```

dynPerParabolicConicalAxisymmMultiwaves

```

```

PointPatchVectorField::

```

```

dynPerParabolicConicalAxisymmMultiwaves

```

```

PointPatchVectorField

```

```

(
    const pointPatch& p,

    const DimensionedField<vector, pointMesh>& iF,

    const dictionary& dict

)

:

```

```

fixedValuePointPatchField<vector>(p, iF, dict),
xCompInitialVertex_(readScalar
(dict.lookup("xCompInitialVertex"))),
chordLength_(readScalar
(dict.lookup("chordLength"))),
speed_(readScalar
(dict.lookup("speed"))),
yCompFinalVertex_(readScalar
(dict.lookup("yCompFinalVertex"))),
numOfWaves_(readScalar
(dict.lookup("numOfWaves"))),
alpha_(readScalar(dict.lookup("alpha"))),
beta_(readScalar(dict.lookup("beta"))),
l_(readScalar(dict.lookup("l"))),
shift_(readScalar(dict.lookup("shift"))),
T_(readScalar(dict.lookup("T"))),
period_(readScalar(dict.lookup("period")))
{
    if (!dict.found("value"))
    {
        updateCoeffs();
    }
}

```

```

    }

    if (dict.find("p0"))
    {
        p0_ = vectorField("p0", dict , p.size());
    }
    else
    {
        p0_ = p.localPoints();
    }
}

```

```

dynPerParabolicConicalAxisymmMultiwaves
PointPatchVectorField::
dynPerParabolicConicalAxisymmMultiwaves
PointPatchVectorField
(

```

```

    const dynPerParabolicConicalAxisymmMultiwaves
        PointPatchVectorField& ptf,

```

```

    const pointPatch& p,

    const DimensionedField<vector, pointMesh>& iF,

    const pointPatchFieldMapper& mapper
)
:
    fixedValuePointPatchField<vector>(ptf, p, iF, mapper),
    xCompInitialVertex_(ptf.xCompInitialVertex_),
    chordLength_(ptf.chordLength_),
    speed_(ptf.speed_),
    yCompFinalVertex_(ptf.yCompFinalVertex_),
    period_(ptf.period_),
    alpha_(ptf.alpha_),
    beta_(ptf.beta_),
    l_(ptf.l_),
    shift_(ptf.shift_),
    T_(ptf.T_),
    numOfWaves_(ptf.numOfWaves_),
    p0_(ptf.p0_)
{}

```

```

dynPerParabolicConicalAxisymmMultiwaves

PointPatchVectorField::

dynPerParabolicConicalAxisymmMultiwaves

PointPatchVectorField

(

const dynPerParabolicConicalAxisymmMultiwaves

    PointPatchVectorField& ptf,

const DimensionedField<vector, pointMesh>& iF

)

:

    fixedValuePointPatchField<vector>(ptf, iF),

    xCompInitialVertex_(ptf.xCompInitialVertex_),

    chordLength_(ptf.chordLength_),

    speed_(ptf.speed_),

    yCompFinalVertex_(ptf.yCompFinalVertex_),

    period_(ptf.period_),

    alpha_(ptf.alpha_),

    beta_(ptf.beta_),

    l_(ptf.l_),

    shift_(ptf.shift_),

    T_(ptf.T_),

```

```

        numOfWaves_(ptf.numOfWaves_),
        p0_(ptf.p0_)

    {}

// *****Member Functions***** //

void dynPerParabolicConicalAxisymmMultiwaves
    PointPatchVectorField::autoMap
(
    const pointPatchFieldMapper& m
)
{
    fixedValuePointPatchField<vector>::autoMap(m);

    p0_.autoMap(m);
}

void dynPerParabolicConicalAxisymmMultiwaves
    PointPatchVectorField::rmap
(
    const pointPatchField<vector>& ptf,

```



```

        const labellist& addr
    )
{
    const dynPerParabolicConicalAxisymmMultiwaves

        PointPatchVectorField& a0Vptf =

    refCast<const dynPerParabolicConicalAxisymmMultiwaves

        PointPatchVectorField>(ptf);

    fixedValuePointPatchField<vector>::rmap(a0Vptf, addr);

    p0_.rmap(a0Vptf.p0_, addr);

}

void dynPerParabolicConicalAxisymmMultiwaves

    PointPatchVectorField::updateCoeffs()
{

    if (this->updated())
    {

```

```

        return;
    }

    const polyMesh& mesh = this->
        dimensionedInternalField().mesh();

    const Time& t = mesh.time();

    const pointPatch& p = this->patch();

double gamma=beta_ * (M_PI/180);

    scalar yMax
    (
        max(p0_.component(vector::Y)())
    );

    scalar yMin
    (
        min(p0_.component(vector::Y)())
    );

    scalar xMax
    (

```

```

        max(p0_.component(vector::X)())

    );

    scalar xMin

    (

        min(p0_.component(vector::X)())

    );

    double param1

    (

        (yMax-yMin)/xMax

    );

    double theta=atan(param1);

    tensor R(cos(theta),-1*sin(theta),0.0,1*
        sin(theta),cos(theta),0.0,0.0,0.0,1.0);

    tensor RT(cos(theta),1*sin(theta),0.0,-1*
        sin(theta),cos(theta),0.0,0.0,0.0,1.0);

    pointField q1(p0_.size(),point(0.0,0.0,0.0));
    pointField q2(p0_.size(),point(0.0,0.0,0.0));
    pointField q0(p0_.size(),point(0.0,0.0,0.0));

    forAll(p0_,pointJ)

    {

```

```

q1[pointJ]=p0_[pointJ]-
point(xMin,yMax,p0_.component(vector::Z())[pointJ]);
q2[pointJ]=q1[pointJ] & RT;
q0[pointJ]=q2[pointJ]+
point(xMin,yMax,p0_.component(vector::Z())[pointJ]);
    }

```

```

    scalar yCompInitialVertex= yMax;

```

```

    scalar xMaxq0
    (
        max(q0.component(vector::X)())
    );

```

```

    pointField M(q0.size(),point(0.0,0.0,0.0));

```

```

    double tFinalVertexXcomp=(xMaxq0/speed_);
    if
    (

```

```

t.value() <=tFinalVertexXcomp
    )
    {
        k = 0.0;
    }

    if
    (
        t.value() ==t.deltaT().value()
    )
    {
        sp=xMaxq0;
    }

for( int i = numOfWaves_; i >= 1; i=i-1 )
{

    for( int j=i; j<=i; j=j+1 )

    {

```

```

if(t.value() > (numOfWaves_-i)*period_)
{

double time=t.value()-(numOfWaves_-j)*period_;

scalar varVertexXcomp
(
    xCompInitialVertex_+speed_*time
);

scalar varVertexYcomp = 0.0;

if(time+k*period_ == t.value()) {s=sp;} else {s=l_;}

if
(
    (time > 0)
    &&
    (time < 1)

```

```

    )
    {

varVertexYcomp = yCompInitialVertex-time*

                (yCompInitialVertex-yCompFinalVertex_);

    }
else
    {

        if

            (

                (mag(s) < 1_)

            )

        {

            varVertexYcomp = varVertexYcompPre+alpha_*
            speed_*t.deltaT().value()*sin(gamma);

            varVertexXcomp = varVertexXcompPre+alpha_*
            speed_*t.deltaT().value()*cos(gamma);

            s=sp;

```

```

    }

    else

    {

varVertexYcomp = yCompFinalVertex_+(time-1)*
                ((shift_-yCompFinalVertex_)/(T_-1));

    }

}

if

(

varVertexYcomp > yCompInitialVertex or varVertexYcomp <= 0.0

)

{

varVertexYcomp = yMax;

}

```



```

scalar yDiff
(
    mag(yMax-varVertexYcomp)
);

scalar ratio
(
    (chordLength_/2.0)*(chordLength_/2.0)
);

scalar coeffA
(
    yDiff/ratio
);

scalar lowerBound
(
    varVertexXcomp-(chordLength_/2.0)
);

scalar upperBound
(
    varVertexXcomp+(chordLength_/2.0)
);

```

```

    );

    scalar Xvertexw=(varVertexXcomp*cos(theta)+
        (varVertexYcomp-yMax)*sin(theta));

    scalar Yvertexw=(-1.0*varVertexXcomp*sin(theta)+
        (varVertexYcomp-yMax)*cos(theta)+yMax);

    scalar Xlowerw=(lowerBound*cos(theta));

    scalar Xupperw=(upperBound*cos(theta));

    scalar Ylowerw=(-1.0*lowerBound*sin(theta)+yMax);

    scalar Yupperw=(-1.0*upperBound*sin(theta)+yMax);

    scalar Xbar=(lowerBound+upperBound)/2.0;

    scalar chordLength=chordLength_;

    scalar Xchordcenterw=Xvertexw;

    scalar Ychordcenterw=((yMin-yMax)/xMax)*
        Xchordcenterw+yMax;

    scalar gap=varVertexYcomp;

    scalar Xgapw=(varVertexXcomp*cos(theta)+
        (gap-yMax)*sin(theta));

    scalar Ygapw=(-1.0*varVertexXcomp*
        sin(theta)+(gap-yMax)*cos(theta)+yMax);

```

```

scalar Amplitudew=yMax-varVertexYcomp;

scalar R0w=(1.0-(Ygapw/Ychordcenterw))*100.0;

scalar DistanceOfAcwVertexFromPylorus=(xMax-Xvertexw);

scalar DistanceOfAcwXupperFromPylorus=(xMax-Xupperw);


pointField velocity(q0.size(),point(0.0,0.0,0.0));

pointField q3(q0.size(),point(0.0,0.0,0.0));

pointField q4(q0.size(),point(0.0,0.0,0.0));


    forAll(q0,pointI)
    {

        scalar yParabola=yMax;

        if
        (

(q0.component(vector::X())[pointI]>lowerBound)

            &&

(q0.component(vector::X())[pointI]<upperBound)

        )

        { //major

```

```

        scalar xDiff
        (
        q0.component(vector::X)()[pointI]-varVertexXcomp
        );
        scalar xDiffSqu
        (
        xDiff*xDiff
        );
        yParabola= coeffA*xDiffSqu+varVertexYcomp;

    }
    else
    {
        yParabola= yMax;
    }

    velocity[pointI]=point(0.0,yParabola,0.0)-
        point(0.0,yMax,0.0);

    q3[pointI]=velocity[pointI] & R;

```

```

if
    (p0_.component(vector::Z())[pointI] > 0.0)
        {
            q4[pointI]=q3[pointI]+point(0.0,0.0,0.04*
                q3.component(vector::Y())[pointI]);
        }
else
    {

q4[pointI]=q3[pointI]+point(0.0,0.0,-0.04*
    q3.component(vector::Y())[pointI]);
    }

    }

//  p0_ and p.localPoints() will return back the points the
//  the deformation

if( mag(s) >= l_ ) {s=xMaxq0-upperBound;;}
if( mag(s) <= l_ ) {k=(numOfWaves_ - j);}

```

```

    if(upperBound < xMaxq0 && s > 0.0)
    {
        M=M+q4;

    }

    pointField::operator=
        (
            (
                p0_+
                M
                -p.localPoints()
                )/t.deltaT().value()

        );

    if (time+k*period_ == t.value())
    {
        varVertexYcompPre = varVertexYcomp;
        varVertexXcompPre = varVertexXcomp; sp=s;
    }

```

```

    }

}

}

fixedValuePointPatchField<vector>::updateCoeffs();
}

void dynPerParabolicConicalAxisymmMultiwaves
    PointPatchVectorField::write
(
    Ostream& os
) const
{
    pointPatchField<vector>::write(os);
    os.writeKeyword("chordLength")
        << chordLength_ << token::END_STATEMENT << nl;
    os.writeKeyword("xCompInitialVertex")
        << xCompInitialVertex_ << token::END_STATEMENT << nl;
    os.writeKeyword("speed")

```

```

        << speed_ << token::END_STATEMENT << nl;

os.writeKeyword("yCompFinalVertex")

        << yCompFinalVertex_ << token::END_STATEMENT << nl;

os.writeKeyword("period")

        << period_ << token::END_STATEMENT << nl;

os.writeKeyword("alpha")

        << alpha_ << token::END_STATEMENT << nl;

os.writeKeyword("beta")

        << beta_ << token::END_STATEMENT << nl;

os.writeKeyword("l")

        << l_ << token::END_STATEMENT << nl;

os.writeKeyword("shift")

        << shift_ << token::END_STATEMENT << nl;

os.writeKeyword("T")

        << T_ << token::END_STATEMENT << nl;

os.writeKeyword("numOfWaves")

        << numOfWaves_ << token::END_STATEMENT << nl;

p0_.writeEntry("p0", os);

writeEntry("value", os);

}

```



```

// ***** //

makePointPatchTypeField

(
    pointPatchVectorField,
    dynPerParabolicConicalAxisymmMultiwaves
    PointPatchVectorField
);

// ***** //

} // End namespace Foam

// ***** //

```

Appendix C

OpenFOAM Codes

C.1 shearRate

Application

shearRate

Description

For each time: calculate the shear rate.

-----/

```

#include "fvCFD.H"

// *****

int main(int argc, char *argv[])
{
    timeSelector::addOptions();

    #   include "setRootCase.H"

    #   include "createTime.H"

    instantList timeDirs = timeSelector::
select0(runTime, args);

    #   include "createMesh.H"

    forAll(timeDirs, timeI)
    {
        runTime.setTime(timeDirs[timeI], timeI);

        Info<< "Time = " <<runTime.timeName()<< endl;

```

```

IOobject Uheader
(
    "U",
    runTime.timeName(),
    mesh,
    IOobject::MUST_READ
);

// Check U exists
if (Uheader.headerOk())
{
    mesh.readUpdate();

    Info<< "    Reading U" << endl;

    volVectorField U(Uheader, mesh);

    Info<< "Calculating shearRate" << endl;

    if
    (U.dimensions() == dimensionSet(0, 1, -1, 0, 0))
    {

```

```

        volScalarField shearRate
        (
            IOobject
            (
                "shearRate",
                runTime.timeName(),
                mesh,
                IOobject::NO_READ
            ),
            sqrt(
                0.5*
                (2*symm(fvc::grad(U))&&(2*symm(fvc::grad(U))
                ))
            )
        );

        shearRate.write();
    }

    else
    {
        Info<< "      No U" << endl;
    }

```

```

        Info<< endl;

    }

}

return 0;

}

// ***** //

```

C.2 stressComponentsMag

Application

stressComponents

Description

Calculates and writes the scalar fields
of the six components of the stress
tensor sigma for each time.

-----/

```

#include "fvCFD.H"

#include "incompressible/singlePhaseTransportModel
    /singlePhaseTransportModel.H"

#include "zeroGradientFvPatchFields.H"


// *****

int main(int argc, char *argv[])
{
    timeSelector::addOptions();

    #   include "setRootCase.H"

    #   include "createTime.H"

    instantList timeDirs = timeSelector::select0(runTime, args);

    #   include "createMesh.H"

    forAll(timeDirs, timeI)
    {
        runTime.setTime(timeDirs[timeI], timeI);
    }
}

```

```
Info<< "Time = " << runTime.timeName() << endl;
```

```
IOobject Uheader
```

```
(
```

```
    "U",
```

```
    runTime.timeName(),
```

```
    mesh,
```

```
    IOobject::MUST_READ
```

```
);
```

```
// Check U exists
```

```
if (Uheader.headerOk())
```

```
{
```

```
    mesh.readUpdate();
```

```
Info<< "    Reading U" << endl;
```

```
volVectorField U(Uheader, mesh);
```

```
# include "createPhi.H"
```

```
singlePhaseTransportModel laminarTransport(U, phi);
```



```

        volSymmTensorField sigma
        (
            IOobject
            (
                "sigma",
                runTime.timeName(),
                mesh,
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            laminarTransport.nu()*2*dev(symm(fvc::grad(U)))
        );

//-----
// K.A. Feigl
// Compute magnitude of viscous stress tensor

        volScalarField sigmaMag
        (
            IOobject
            (
                "sigmaMag",

```

```

runTime.timeName(),

mesh,

IOobject::NO_READ

        ),

        mag(sigma)

    );

    sigmaMag.write();

// K.A. Feigl

//-----

    volScalarField sigmaxx

    (

        IOobject

        (

            "sigmaxx",

            runTime.timeName(),

            mesh,

            IOobject::NO_READ

        ),

        sigma.component(symmTensor::XX)

    );

```

```

sigmaxx.write();

volScalarField sigmayy
(
    IOobject
    (
        "sigmayy",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ
    ),
    sigma.component(symmTensor::YY)
);
sigmayy.write();

```

```

volScalarField sigmazz
(
    IOobject
    (
        "sigmazz",
        runTime.timeName(),

```

```

        mesh,

        IOobject::NO_READ

    ),

    sigma.component(symmTensor::ZZ)

);

sigmazz.write();

```

```

volScalarField sigmaxy

(
    IOobject
    (
        "sigmaxy",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ
    ),
    sigma.component(symmTensor::XY)

);

sigmaxy.write();

```

```

volScalarField sigmaxz

```

```
(
    IOobject
    (
        "sigmaxz",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ
    ),
    sigma.component(symmTensor::XZ)
);
sigmaxz.write();
```

```
volScalarField sigmayz
```

```
(
    IOobject
    (
        "sigmayz",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ
    ),
```

```

        sigma.component(symmTensor::YZ)
    );

    sigmayz.write();

volVectorField Ub
(
    IOobject
    (
        "Ub",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ
    ),
    U,
    zeroGradientFvPatchVectorField::typeName
);

Ub.correctBoundaryConditions();

Ub.write();

volScalarField sigmaUn
(

```

```

IObject
(
    "sigmaUn",
    runTime.timeName(),
    mesh,
    IObject::NO_READ
),
0.0*sigma.component(symmTensor::YZ)
);

forAll(sigmaUn.boundaryField(), patchI)
{
    sigmaUn.boundaryField()[patchI] =
    (
        mesh.boundary()[patchI].nf()
        & sigma.boundaryField()[patchI]
    ).component(vector::X);
}

sigmaUn.write();
}

```

```

        else

        {

            Info<< "      No U" << endl;

        }

        Info<< endl;

    }

    Info<< "End" << endl;

    return 0;

}

// ***** //

```

C.3 movingWallNormalVel BC.

movingWallNormalVelFvPatchVectorField.H

SourceFiles

movingWallNormalVelFvPatchVectorField.C

```
\*-----*/
```

```
#ifndef movingWallNormalVelFvPatchVectorField_H
```

```
#define movingWallNormalVelFvPatchVectorField_H
```

```
#include "fvPatchFields.H"
```

```
#include "fixedValueFvPatchFields.H"
```

```
// * * * * * * * * * * * * * * * * * * * * //
```

```
namespace Foam
```

```
{
```

```
/*-----*\
```

```
    Class movingWallNormalVelFvPatch Declaration
```

```
\*-----*/
```

```
class movingWallNormalVelFvPatchVectorField
```

```
:
```

```

    public fixedValueFvPatchVectorField
{
    // Private data

    //- Name of velocity field
    word UName_;

public:

    //- Runtime type information
    // aaalhaba110 08-15-12 (changed the type name
    // from movingWallVelocity to:
    TypeName("movingWallNormalVel");
    // aaalhaba111

    // Constructors

    //- Construct from patch and internal field
    movingWallNormalVelFvPatchVectorField

```

```

(
    const fvPatch&,
    const DimensionedField<vector, volMesh>&
);

//- Construct from patch, internal field and
// dictionary
movingWallNormalVelFvPatchVectorField
(
    const fvPatch&,
    const DimensionedField<vector, volMesh>&,
    const dictionary&
);

//- Construct by mapping given
// movingWallNormalVelFvPatchVectorField
// onto a new patch
movingWallNormalVelFvPatchVectorField
(
    const movingWallNormalVelFvPatchVectorField&,
    const fvPatch&,

```

```

        const DimensionedField<vector, volMesh>&,

        const fvPatchFieldMapper&

    );

    //- Construct as copy

    movingWallNormalVelFvPatchVectorField

    (

    const movingWallNormalVelFvPatchVectorField&

    );

    //- Construct and return a clone

    virtual tmp<fvPatchVectorField> clone() const

    {

        return tmp<fvPatchVectorField>

        (

        new movingWallNormalVelFvPatchVectorField(*this)

        );

    }

    //- Construct as copy setting internal field

    // reference

```

```

        movingWallNormalVelFvPatchVectorField

    (
        const movingWallNormalVelFvPatchVectorField&,
        const DimensionedField<vector, volMesh>&

    );

//- Construct and return a clone setting internal
// field

    // reference

    virtual tmp<fvPatchVectorField> clone

    (
        const DimensionedField<vector, volMesh>& iF

    ) const

    {

        return tmp<fvPatchVectorField>

        (

new

movingWallNormalVelFvPatchVectorField(*this, iF)

        );

    }

```

```

        // Member functions

    //- Update the coefficients associated with the
    // patch field

        virtual void updateCoeffs();

        //- Write

        virtual void write(Ostream&) const;

};

// * * * * *

} // End namespace Foam

// * * * * *

#endif

// *****

```

movingWallNormalVelFvPatchVectorField.C

```
// aalhaba110 08-15-2012

#include

"movingWallNormalVelFvPatchVectorField.H"

// aalhaba111

// Note: replacing all movingWallVelocity to the

// new class movingWallNormalVel

#include "addToRunTimeSelectionTable.H"

#include "volFields.H"

#include "surfaceFields.H"

#include "fvcMeshPhi.H"


// ***** Constructors ***** //

Foam::movingWallNormalVelFvPatchVectorField::

movingWallNormalVelFvPatchVectorField

(

    const fvPatch& p,

    const DimensionedField<vector, volMesh>& iF
```

```

)

:

    fixedValueFvPatchVectorField(p, iF),

    UName_("U")

{}

Foam::movingWallNormalVelFvPatchVectorField::
movingWallNormalVelFvPatchVectorField
(
    const movingWallNormalVelFvPatchVectorField& ptf,

    const fvPatch& p,

    const DimensionedField<vector, volMesh>& iF,

    const fvPatchFieldMapper& mapper

)

:

    fixedValueFvPatchVectorField(ptf, p, iF, mapper),

    UName_(ptf.UName_)

{}

```



```

Foam::movingWallNormalVelFvPatchVectorField::
movingWallNormalVelFvPatchVectorField
(
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF,
    const dictionary& dict
)
:
    fixedValueFvPatchVectorField(p, iF),
    UName_(dict.lookupOrDefault<word>("U", "U"))
{
    fvPatchVectorField::operator=
        (vectorField("value", dict, p.size()));
}

```

```

Foam::movingWallNormalVelFvPatchVectorField::
movingWallNormalVelFvPatchVectorField
(
    const
movingWallNormalVelFvPatchVectorField& mwvpvf

```

```

)

:

    fixedValueFvPatchVectorField(mwvpvf),

    UName_(mwvpvf.UName_)

{}

```

```

Foam::movingWallNormalVelFvPatchVectorField::
movingWallNormalVelFvPatchVectorField
(
    const
movingWallNormalVelFvPatchVectorField& mwvpvf,
    const DimensionedField<vector, volMesh>& iF
)

:

    fixedValueFvPatchVectorField(mwvpvf, iF),

    UName_(mwvpvf.UName_)

{}

```

```

// ***** Member Functions ***** //

```

```

void Foam::
movingWallNormalVelFvPatchVectorField::
updateCoeffs()
{
    if (updated())
    {
        return;
    }

    const fvPatch& p = patch();

    const polyPatch& pp = p.patch();

    const
fvMesh& mesh = dimensionedInternalField().mesh();

    const
pointField& oldPoints = mesh.oldPoints();

    vectorField oldFc(pp.size());

    forAll(oldFc, i)
    {
        oldFc[i] = pp[i].centre(oldPoints);
    }
}

```

```

    }

    const vectorField
Up((pp.faceCentres() - oldFc)/
    mesh.time().deltaTValue());

    const volVectorField&
U = db().lookupObject<volVectorField>(UName_);

    scalarField phip
(
    p.patchField
<surfaceScalarField, scalar>(fvc::meshPhi(U))
    );

    const vectorField n(p.nf());

    const scalarField& magSf = p.magSf();

    tmp<scalarField> Un = phip/(magSf + VSMALL);

/* aaalhaba010 08-15-12
(commented out the old operator)

    vectorField::

```

```

        operator=(Up + n*(Un - (n & Up)));

        aaalhaba011 */

// aaalhaba120 08-15-12 (project the
// movingWallVelocity
// onto normal direction)
// Note: (a & b) is for the dot product between
// vectors
// a and b

        vectorField::
operator=((n & (Up + n*(Un - (n & Up))))*n);

// aaalhaba121

        fixedValueFvPatchVectorField::updateCoeffs();
}

void Foam::
movingWallNormalVelFvPatchVectorField::
write(Ostream& os) const
{
        fvPatchVectorField::write(os);

```

```

writeEntryIfDifferent<word>(os,"U", "U", UName_);

    writeEntry("value", os);

}

```

```

// *****

```

```

namespace Foam
{
    makePatchTypeField
    (
        fvPatchVectorField,
        movingWallNormalVelFvPatchVectorField
    );
}

```

```

// *****

```

C.4 transientSimpleDyMFoam

checkTotalVolume.H

```
scalar newTotalVolume =  
    sum(mesh.cellVolumes());  
  
Info<< "Volume: new = "  
  
<< newTotalVolume << " old = " << totalVolume  
  
    << " change = "  
  
<< Foam::mag(newTotalVolume - totalVolume)<<endl;  
  
    totalVolume = newTotalVolume;
```

correctPhi.H

```
{  
  
wordList pcorrTypes(p.boundaryField().types());  
  
for (label i=0; i<p.boundaryField().size(); i++)  
  
    {
```

```

        if (p.boundaryField()[i].fixesValue())
        {
            pcorrTypes[i] =
fixedValueFvPatchScalarField::typeName;
        }
    }

    volScalarField pcorr
    (
        IOobject
        (
            "pcorr",
            runtime.timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::NO_WRITE
        ),
        mesh,
        dimensionedScalar("pcorr", p.dimensions(), 0.0),
        pcorrTypes
    );

```



```

#   include "continuityErrs.H"

// Flux predictor

phi = (fvc::interpolate(U) & mesh.Sf());

rAU == runTime.deltaT();

for(int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
{
    fvScalarMatrix pcorrEqn
    (
        fvm::laplacian(rAU, pcorr) == fvc::div(phi)
    );

    pcorrEqn.setReference(pRefCell, pRefValue);

    pcorrEqn.solve();

    if (nonOrth == nNonOrthCorr)
    {
        phi -= pcorrEqn.flux();
    }

// Fluxes are corrected to absolute velocity and

```

```

// further corrected

// later. HJ, 6/Feb/2009

    }

}

```

createFields.H

```

    Info<< "Reading field p\n" << endl;

    volScalarField p

    (
        IOobject
        (
            "p",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE

        ),
        mesh
    );

```

```

Info<< "Reading field U\n" << endl;

volVectorField U

(
    IOobject
    (
        "U",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

# include "createPhi.H"

label pRefCell = 0;

scalar pRefValue = 0.0;

setRefCell

(p, mesh.solutionDict().subDict("PISO"), pRefCell, pRefValue);

```

```

    scalar totalVolume = sum(mesh.V()).value();

    volScalarField rAU
    (
        IOobject
        (
            "rAU",
            runtime.timeName(),
            mesh
        ),
        mesh,
        runtime.deltaT(),
        zeroGradientFvPatchScalarField::typeName
    );

    singlePhaseTransportModel
    laminarTransport(U, phi);

    autoPtr<incompressible::RASModel> turbulence
    (
        incompressible::RASModel::

```

```
New(U, phi, laminarTransport)

);
```

readControls.H

```
# include "readTimeControls.H"

# include "readPISOControls.H"


bool correctPhi = false;

if ( piso.found("correctPhi") )
{
correctPhi = Switch(piso.lookup("correctPhi"));
}


bool checkMeshCourantNo = false;

if ( piso.found("checkMeshCourantNo") )
{
    checkMeshCourantNo =
Switch(piso.lookup("checkMeshCourantNo"));
}
```

transientSimpleDyMFoam.C

Application

transientSimpleDyMFoam

Description

Transient solver for incompressible, turbulent
flow of Newtonian
fluids with dynamic mesh. Solver implements a
SIMPLE-based
algorithm in time-stepping mode.

Author

Hrvoje Jasak, Wikki Ltd. All rights reserved.

Modification

Evaluation of turbulence model moved inside
the
SIMPLE loop.

- Mikko Auvinen, Aalto University

```
#include "fvCFD.H"

// The following is a long line, so will break
// into two

#include //<br>

"incompressible/singlePhaseTransportModel/

//<br>

singlePhaseTransportModel.H"

#include "incompressible/RASModel/RASModel.H"

#include "dynamicFvMesh.H"
```

```
int main(int argc, char *argv[])

{

#   include "setRootCase.H"

#   include "createTime.H"

#   include "createDynamicFvMesh.H"

#   include "initContinuityErrs.H"

#   include "createFields.H"
```



```

#         include "volContinuity.H"

        if (correctPhi && meshChanged)
        {

// Fluxes will be corrected to absolute velocity

        // HJ, 6/Feb/2009

#         include "correctPhi.H"

        }


// Make the fluxes relative to the mesh motion

        fvc::makeRelative(phi, U);


        if (checkMeshCourantNo)
        {

#         include "meshCourantNo.H"

        }


// --- SIMPLE loop


for (int ocorr = 0; ocorr < nOuterCorr; ocorr++)
{

```

```

// #          include "CourantNo.H"    -- mikko

#          include "UEqn.H"

rAU = 1.0/UEqn.A();

U = rAU*UEqn.H();

phi = (fvc::interpolate(U) & mesh.Sf());

//+ fvc::ddtPhiCorr(rAU, U, phi);

adjustPhi(phi, U, p);

p.storePrevIter();

for
(int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
{
    fvScalarMatrix pEqn
    (
        fvm::laplacian(rAU, p) == fvc::div(phi)
    );

```

```

        pEqn.setReference(pRefCell, pRefValue);

    if
    (ocorr == nOuterCorr - 1 && nonOrth == nNonOrthCorr)
    {
        pEqn.solve(mesh.solver(p.name() + "Final"));
    }
    else
    {
        pEqn.solve(mesh.solver(p.name()));
    }

    if (nonOrth == nNonOrthCorr)
    {
        phi -= pEqn.flux();
    }
}

#           include "continuityErrs.H"

//Explicitly relax pressure for momentum corrector

```

```

        p.relax();

    // Make the fluxes relative to the mesh motion

        fvc::makeRelative(phi, U);

        U -= rAU*fvc::grad(p);

        U.correctBoundaryConditions();

    // The turbulence model evaluation is necessary
    // within

        // the SIMPLE loop.  -- mikko

        turbulence->correct();

    }

    runTime.write();

    Info<< "ExecutionTime = "

    << runTime.elapsedCpuTime() << " s"

    << "   ClockTime = "

    << runTime.elapsedClockTime() << " s"

```

```

        << nl << endl;

    }

    Info<< "End\n" << endl;

    return(0);

}

// ***** //

```

UEqn.H

```

    fvVectorMatrix UEqn
    (
        fvm::ddt(U)
        + fvm::div(phi, U)
        + turbulence->divDevReff(U)
    );

    UEqn.relax();

```

```
// Solve the momentum equation  
  
solve(UEqn == -fvc::grad(p));
```

Make/files File

```
transientSimpleDyMFoam.C
```

```
EXE = $(FOAM_USER_APPBIN)/transientSimpleDyMFoam
```

Make/options File

```
EXE_INC = \  
  
-I$(LIB_SRC)/dynamicFvMesh/lnInclude \  
  
-I$(LIB_SRC)/dynamicMesh/lnInclude \  
  
-I$(LIB_SRC)/meshTools/lnInclude \  
  
-I$(LIB_SRC)/finiteVolume/lnInclude \  
  
-I$(LIB_SRC)/turbulenceModels/RAS \  
  
-I$(LIB_SRC)/transportModels
```

```
EXE_LIBS = \  
  

```

```
-ldynamicFvMesh \  
-ldynamicMesh \  
-lengine \  
-lmeshTools \  
-lincompressibleRASModels \  
-lincompressibleTransportModels \  
-lfiniteVolume \  
-llduSolvers
```

Appendix D

Transport Efficiency via Peristaltic Motion in 2-D planer Tube

This study extends the work of Al-Hababbeh [40] in two ways. First, we develop a 2-D axisymmetric numerical model to get a realistic tubular peristaltic flow as encountered in the small intestine, see Chapter 3, and second, we examine the influence of the fluid viscosity variation on the transport efficiency (TE). The characteristic data for the 2-D planar tube are the same as those for the 2-D axisymmetric tubular model developed in Section 3.1.1. Recall that the original tube length L is 180 mm. To examine the influence of the fluid viscosity on the transport efficiency, several simulations have been performed for five different Newtonian fluids, whose parameters are listed in Table 3.2, and the results are given in Figs D.1–D.4 for the case of 5 mm/s wave

speed.

Figure D.1 shows that the transport efficiency strengthened with fluid viscosity and relative occlusion, where the transport efficiency for the largest four viscosities appears to be nearly identical. The x-component of velocity for five different Newtonian fluids near the outlet at time $t = 32$ s is shown in Fig. D.2. This figure shows that the values of the velocity are almost identical, except for the lowest viscous fluid N1. This is consistent with the transport efficiency results given in Fig. D.1 for the case of 60% relative occlusion.

The unexpected behavior of the lowest viscous fluids N1 may be explained due to the characteristic geometry length, as is shown in Figs D.3 and D.4 for the case of 60% relative occlusion. These figures show that, by increasing the original tube length by a factor of 1.5, the distance between the viscosity curves of the x-component of velocity near the outlet is almost vanished on overage (Fig. D.3), and hence the transport efficiency is nearly independent of the fluid viscosity (Fig. D.4).

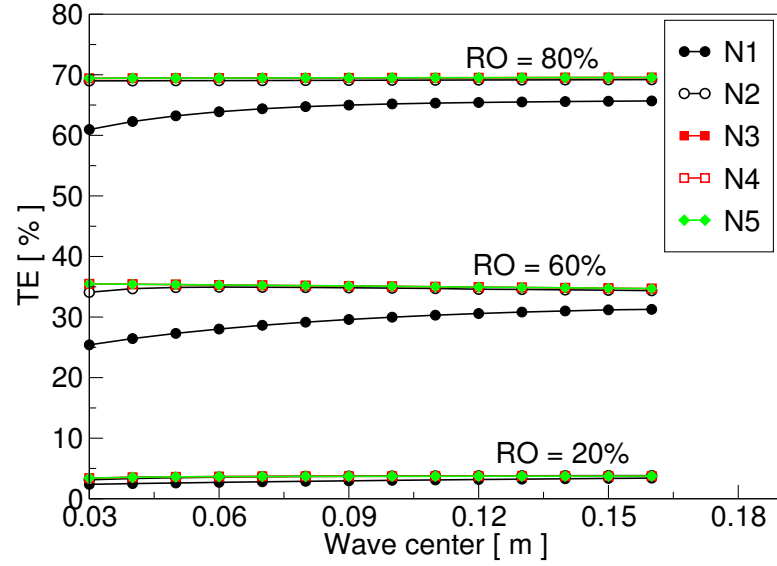


Figure D.1: Transport efficiency for five Newtonian fluids in the planar tubular model. The wave speed is 5 mm/s and three relative occlusions of 20%, 60% and 80% are applied.

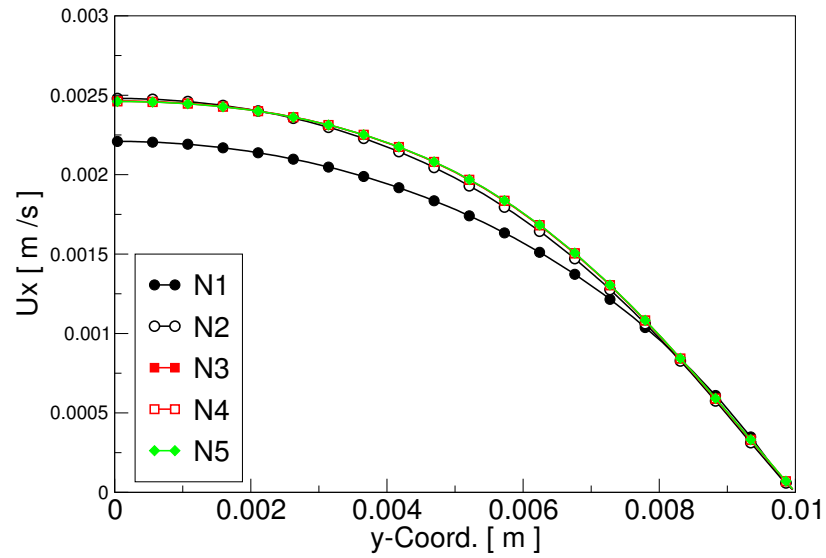


Figure D.2: The x-component of the velocity near the outlet of five Newtonian fluids in the planar tubular model at $t = 32$ s. The wave speed and the relative occlusion are 5 mm/s and 60%, respectively.

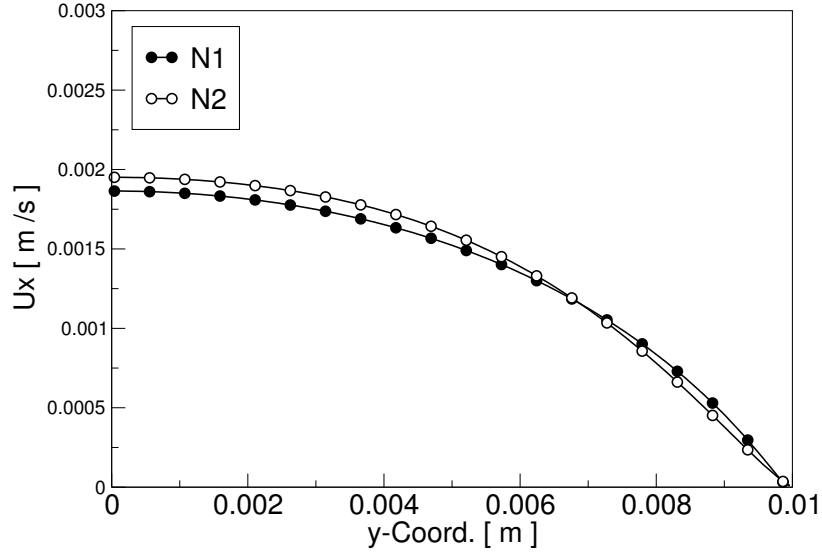


Figure D.3: The x-component of the velocity near the outlet for the lowest two viscous fluids at $t = 50$ s in the planar tubular model with length of $1.5 \times L$. The wave speed and the relative occlusion are 5 mm/s and 60%, respectively.

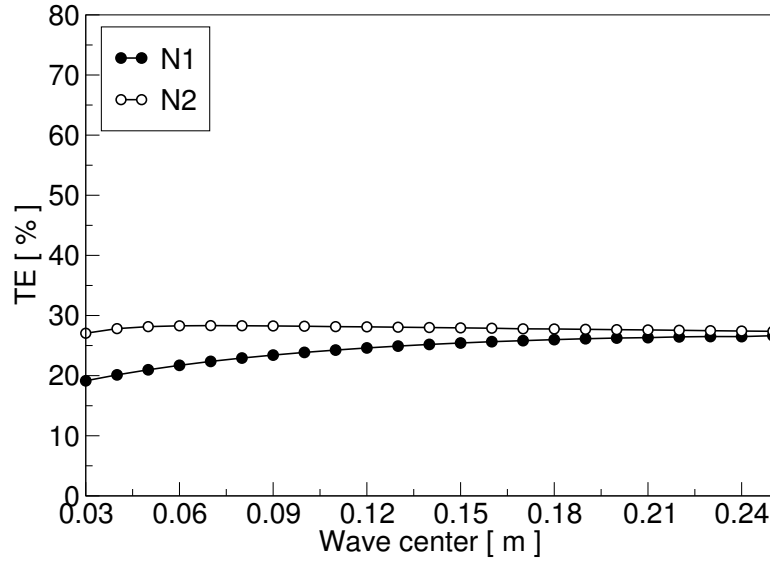


Figure D.4: Transport efficiency for the lowest two viscous fluids along the domain in the planar tubular model with length of $1.5 \times L$. The wave speed and the relative occlusion are 5 mm/s and 60%, respectively.

A comparison between the planar model and the axisymmetric tubular model in terms of the TE at time $t = 32$ s and with a fixed uniform wave speed of 5 mm/s is given in Table D.1. Also, this comparison aims to identify the effect of the shear-thinning non-Newtonian behavior of the fluid on the TE. Two non-Newtonian fluids, BCA and BCB, with shear rate dependent viscosity expressed by the Bird-Carreau Eq. (2.17) will be used in this study, where the fluid parameters are summarized in Table 3.8. The results indicate that although the TE values obtained within the planar model are not qualitatively affected, their magnitude are. In particular, the use of an actual realistic tubular model of the system predicted the TE values to be larger. Specifically, Table D.1 shows that the TE increases with relative occlusion and decreases with the power-law index, n .

Table D.1
Transport efficiency for the Newtonian and non-Newtonian fluids at time
 $t = 32$ s. The wave speed is 5 mm/s.

Relative occlusion (%)	Planar tubular model			Axisymmetric tubular model		
	N3	BCA	BCB	N3	BCA	BCB
20	3.47%	2.4%	1.42%	7.04%	4.98%	3.25%
60	34.68%	30.41%	23.39%	56.07%	49.65%	38.02%
80	69.54%	67.61%	64.21%	88.68%	88.43%	85.68%

Finally, a comparison between the planar model and the axisymmetric tubular model in terms of the TE is given in the Table D.2, in which the simulations are performed for one Newtonian fluid N3 and at $x = 160$ mm for three different wave speeds of 2.5 mm/s, 5 mm/s and 10 mm/s. The results of this Table show that the TE is almost independent of the wave speed and increases with relative occlusion. However, the

effect of the wave speed and relative occlusion on the TE is more significant and considerable for the axisymmetric tubular simulations case.

Table D.2

Transport efficiency for the Newtonian fluid N3 at $x = 160$ mm. Three different speeds of 2.5 mm/s, 5 mm/s and 10 mm/s are examined.

Relative occlusion (%)	Planar tubular model Wave speed (mm/s)			Axisymmetric tubular model Wave speed (mm/s)		
	2.5	5	10	2.5	5	10
20	3.74%	3.74%	3.75%	7.02%	7.04%	7.01%
60	34.67%	34.68%	34.59%	56.08%	56.07%	55.82%
80	69.54%	69.54%	69.53%	88.32%	88.68%	89.02%