CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Implementation of Lumped Parameter Network Boundary Conditions for the Patient-Specific CFD Simulations of Coronary Arteries in OpenFOAM

Muhammad Ahmad Raza, B.Sc.

IRC PhD Scholar

School of Mechanical and Materials Engineering,

University College Dublin (UCD), Ireland.

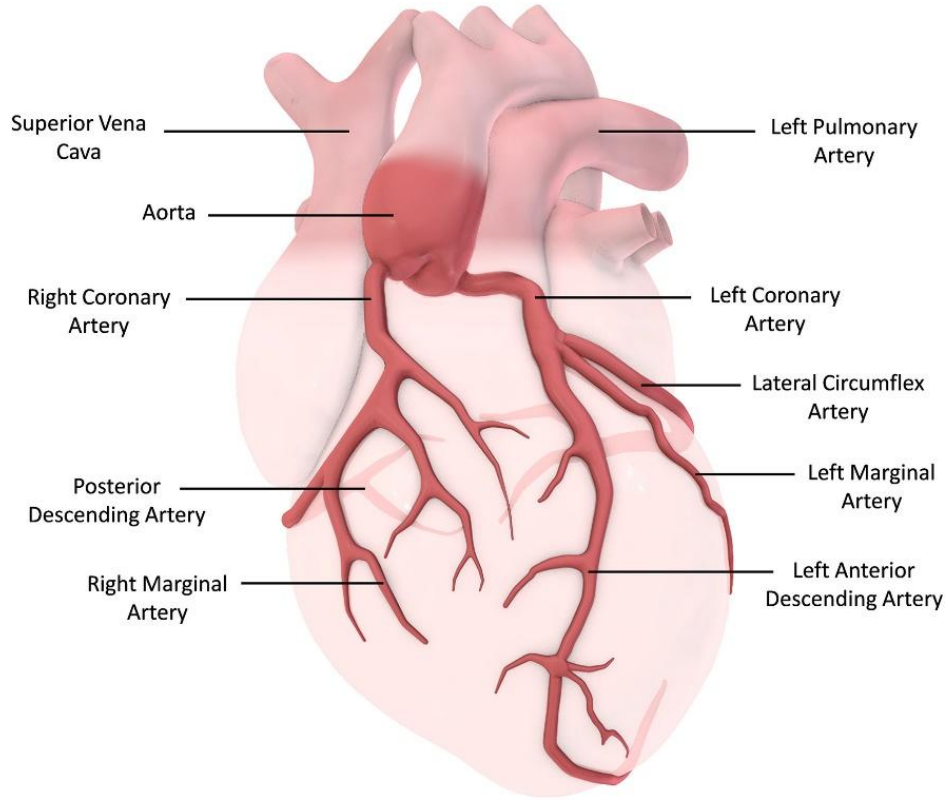
muhammad.a.raza@ucdconnect.ie

IRISH RESEARCH COUNCIL
An Chomhairle um Thaighde in Éirinn

23/01/2025

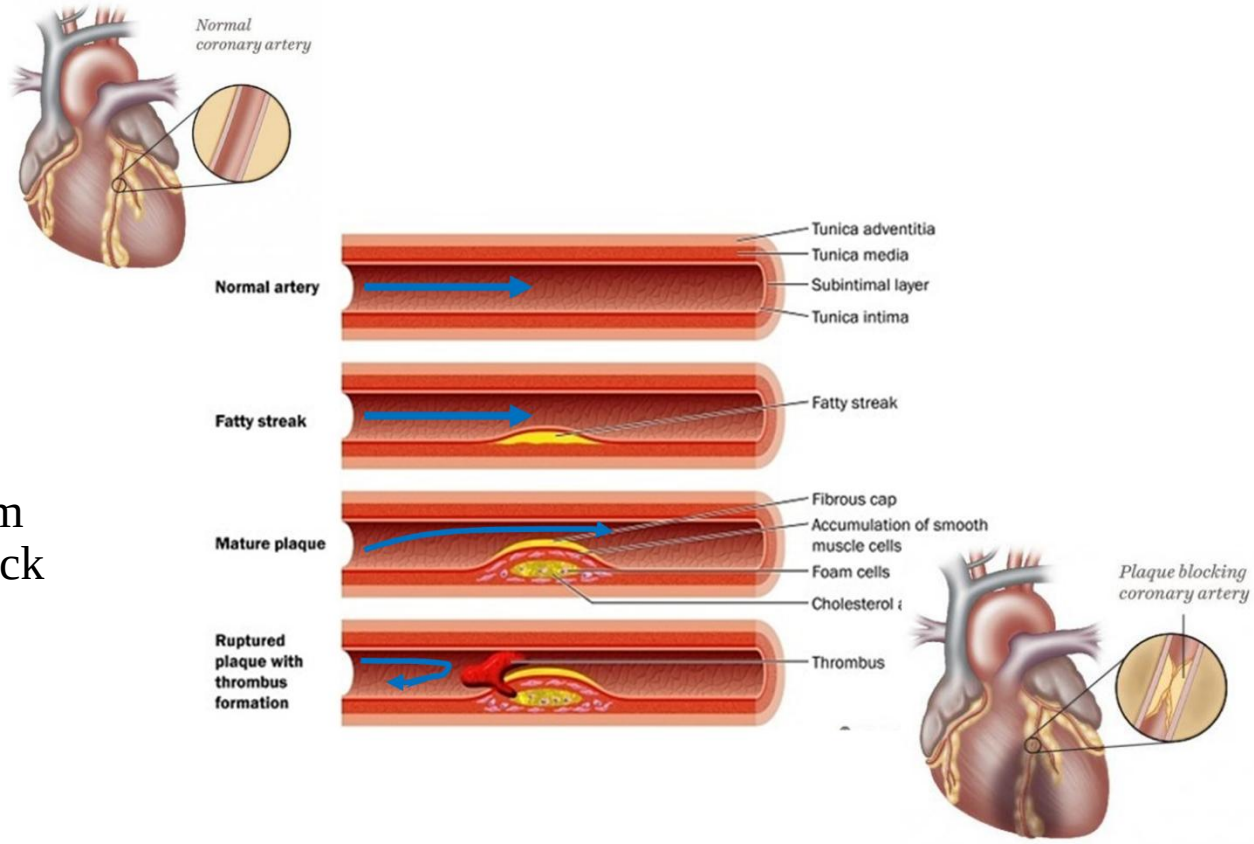
Coronary Arteries

- Supply oxygenated blood (oxygen and nutrients) to the heart muscle i.e., myocardium
- Left coronary artery (LCA)
  - Supplies blood to the left atrium and left ventricle i.e., heart's main pumping chamber
  - Typically larger than RCA
  - Two main branches: left anterior descending (LAD) artery and left circumflex (LCx) artery
  - “Widowmaker" artery
- Right coronary artery (RCA)
  - Supplies blood to the right atrium, the right ventricle, a portion of the septum, and atrioventricular (AV) node
  - Typically smaller than LCA
- Size
  - $\sim 3\ cm$ (Aorta) → $\sim 4\ mm$ (Aortic sinus) → $\sim 100\ \mu m$ (Branches) → $\sim 10\ \mu m$ (Arterioles) → $\sim 5\ \mu m$ (Capillary bed)



Coronary Artery Disease

- Build-up of atherosclerotic plaque in the coronary arteries
  - Fats, cholesterol, calcium and other substances in the blood
- Local diameter reduction i.e., CA stenosis
- Restriction in the flow of oxygen-rich blood to the myocardium
- Increased blood velocity in the stenosis upstream
- Additional loading on the calcified plaque
- Plaque rapture triggers blood clot
- Sudden blockage of oxygen-rich blood supply to the myocardium downstream a CA branch i.e., myocardial infarction or heart attack



Diagnosis and Treatment

- Fractional Flow Reserve (FFR)
  - Ratio of maximum achievable blood flow through a blockage (area of stenosis) to the maximum achievable blood flow in the same vessel in the hypothetical absence of the blockage

$$\text{FFR} = \frac{\text{Pressure distal to the lesion (blockage)}}{\text{Pressure proximal to the lesion (blockage)}} = \frac{P_D}{P_A}$$

  - Correlates with the degree of blockage of the artery (mild, moderate/intermediate or severe)
  - The "normal" ratio is expected to be 1 → no stenosis
- Invasive procedure done through a standard diagnostic catheter at the time of a coronary angiogram or cardiac catheterization
  - Gold Standard
- Useful in the assessment of "intermediate" blockages (CAD) to determine the need for angioplasty, stenting or CABG
  - $\text{FFR} < 0.75$ → Ischemia-producing stenosis → Revascularization
  - $\text{FFR} > 0.80$ → Non–ischemia-producing stenosis → Medical Therapy
  - $0.75 > \text{FFR} > 0.80$ → Gray zone → Revascularization vs. Medical therapy
- Alternative: non-invasive FFR-CT
  - Numerical simulations (CFD/FSI) on patient specific geometries from CT imaging

Diagnosis and Treatment (Cont'd…)

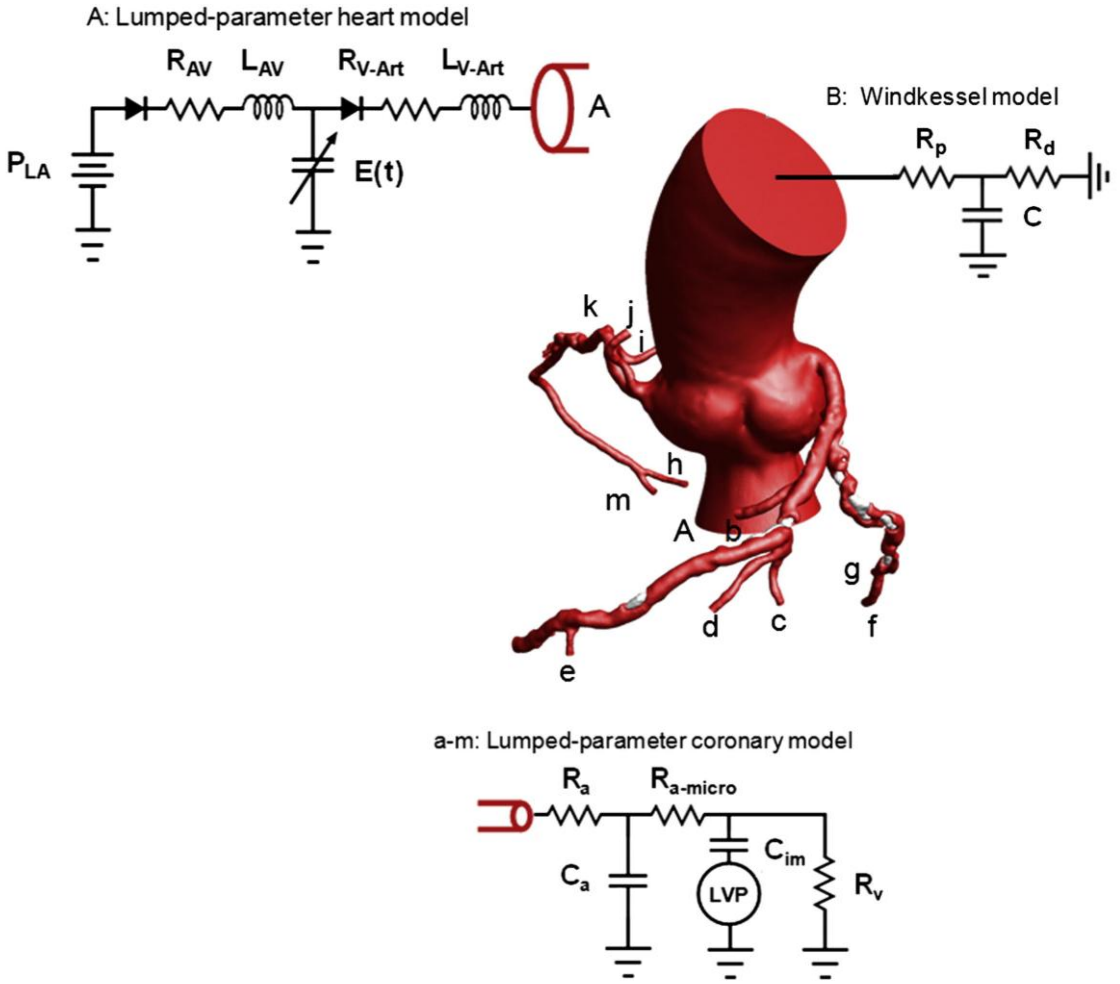
- Coronary artery bypass grafting (CABG)
  - Rerouting blood around arterial blockages using a graft
  - Typically sourced from another part of the patient's body
- Percutaneous coronary intervention (PCI)
  - Minimally invasive
  - Insertion of a balloon catheter into the affected coronary artery
  - Balloon is then inflated to reopen the vessel using coronary stent
- Surgical planning: virtual surgery/stenting
  - Assessment of intervention outcomes in priori using numerical simulations (CFD/FSI)

Numerical Simulations of Coronary Arteries

- Wide applications in the analysis of cardiovascular system
  - Examining the hemodynamics of both healthy and diseased vessels e.g., Fontan hemodynamics
  - Development and evaluation of vascular medical devices e.g., mechanical circulatory support devices (blood pumps)
  - Aiding in surgical planning, and forecasting the outcomes of interventions e.g., stenting
- Infrequent applications to predict the pulsatile flow and pressure fields within 3D coronary vascular networks
  - Intricate relationship between coronary vascular dynamics and the interplay between the heart and the arterial system
  - Coronary blood flow diminishes during ventricular contraction (systolic phase) as intramyocardial pressure rises, exerting a compressive force on coronary vessels
  - In contrast, coronary flow increases during ventricular relaxation (diastolic phase), when intramyocardial pressure decreases, reducing the extravascular compressive force
  - Realistic simulation requires integrated models that account for both the heart and arterial system, as well as their dynamic interactions
- In the past, majority of 3D studies typically prescribed coronary flow instead of predicting
  - Unrealistic pressure modeling
  - Traction-free BCs regardless of whether the wall are rigid, compliant or subject to cardiac motion

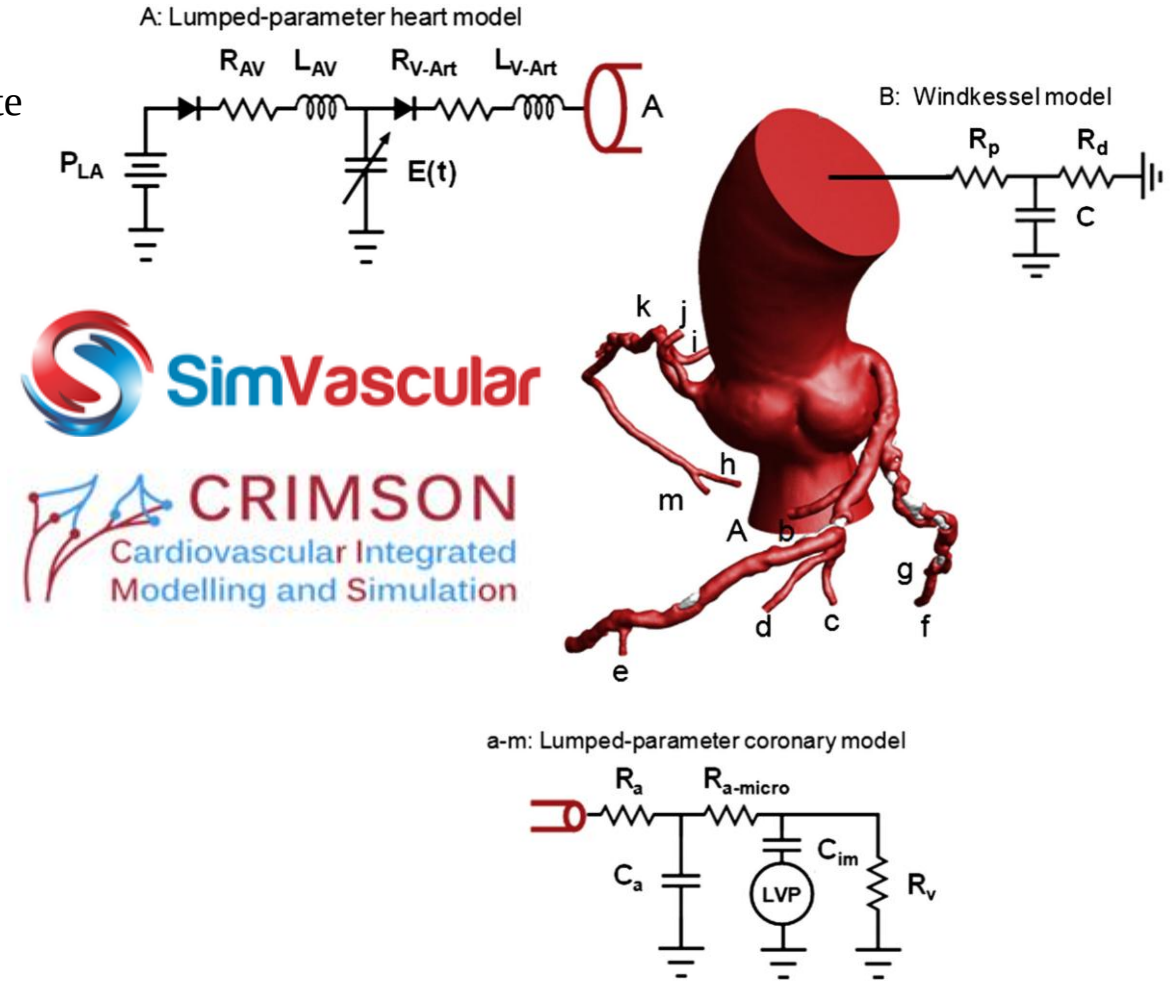
BCs for the Numerical Simulations of Coronary Arteries

- Explicitly coupled analytic boundary condition models require
  - Subiterations within each time step
  - The use of small time steps dictated by the stability of explicit time integration schemes
- Patient-specific computational models must be
  - Both robust and stable enough to handle complex flow characteristics
  - Efficiently integrating different computational scales
- Coupling of 3D models with analytical 0D models
  - Fully implicit coupling
  - Robust and versatile
  - Produce physiologically realistic flow and pressure fields
- Proposed boundary conditions
  - Aortic inlet: lumped-parameter heart model or patient-specific flow rate profile
  - Aortic outlets: three-element windkessel model
  - Coronary outlets: lumped parameter coronary vascular bed model



BCs for the Numerical Simulations of Coronary Arteries (Cont'd…)

- Proposed boundary conditions
  - Aortic inlet: lumped-parameter heart model or patient-specific flow rate profile
  - Aortic outlets: three-element windkessel model
  - Coronary outlets: lumped parameter coronary vascular bed model
- Implemented in the finite element method-based tools
  - SimVascular: An open-source pipeline for cardiovascular simulations
  - CRIMSON: An advanced simulation environment for subject specific hemodynamic analysis
- Implementation in OpenFOAM
  - Some implementation of three-element windkessel models in old versions
  - No implementation of coronary LPN



Aims of the Current Project

- Developement of differential equations describing the relationship between pressure and flow rate for different lumped parameter network (LPN) models including single-element resistive (R) model, two-element windkessel (RC) model, three-element windkessel (RCR) model, and four-element series as well as parallel windkessel (RCRL) models, and coronary outlet LPN model
- Implementation of lumped parameter network (LPN) models including single-element resistive (R) model, two-element windkessel (RC) model, three-element windkessel (RCR) model, and four-element series as well as parallel windkessel (RCRL) models for the evaluation of the outlet pressure boundary condition in larger blood vessels such as aorta in OpenFOAM
- Implementation of lumped parameter network (LPN) model for downstream coronary vascular beds for the evaluation of the coronary outlet pressure boundary condition in OpenFOAM
- Validation of the implementation against the solution obtained by the numerical treatment of LPNs in MATLAB with the given flow rate and network parameters' values for a pulsatile flow in a straight tube

Windkessel Effect

- Describes how large elastic arteries moderate pulsatile blood flow generated by the heart to maintain a continuous and steady flow through the circulatory system
- The term “windkessel,” derived from the German word for “air chamber,” historically referred to the air reservoirs in fire engines, which smoothed out intermittent water flow
- In the human cardiovascular system, the aorta and other large elastic arteries act as a “hydraulic accumulator,” damping pressure fluctuations and ensuring consistent blood supply during both systole and diastole
  - Large arteries, such as the aorta, contain elastin fibers in their walls, which enable them to stretch during systole to store energy as the arteries accommodate the incoming stroke volume
  - During diastole, when the heart is at rest, these arteries recoil, releasing the stored energy and maintaining blood flow despite the lack of cardiac ejection
- The “windkessel effect” arises from the interplay between the compliance of large arteries and the resistance of smaller arteries and arterioles
- Reduces pulse pressure by absorbing the systolic surge and maintaining diastolic flow, ensuring stable organ perfusion throughout the cardiac cycle

Elements of Windkessel Models

- Vessel Resistance (R): Resistance encountered by blood as it flows through the network of blood vessel
- Vessel Compliance (C): Elasticity and ability of blood vessels to expand and contract during the cardiac cycle
- Blood Inertia (L): This accounts for the momentum of blood as it moves through the heart and circulatory system

Element	**Electric Circuit**	**Hydraulic/Physiological Circuit**	**Circuit Symbol**	**Dimensions**
Resistance (R)	$V = IR_e$	$P(t) = Q(t)R$		$[ML^{-4}T^{-1}]$
Compliance (C)	$I = C_e \frac{\mathrm{d}V}{\mathrm{d}t}$	$Q(t) = C \frac{\mathrm{d}P(t)}{\mathrm{d}t}$		$[M^{-1}L^{4}T^{2}]$
Inertance (L)	$V = L_e \frac{\mathrm{d}I}{\mathrm{d}t}$	$P(t) = L \frac{\mathrm{d}Q(t)}{\mathrm{d}t}$		$[ML^{-4}]$

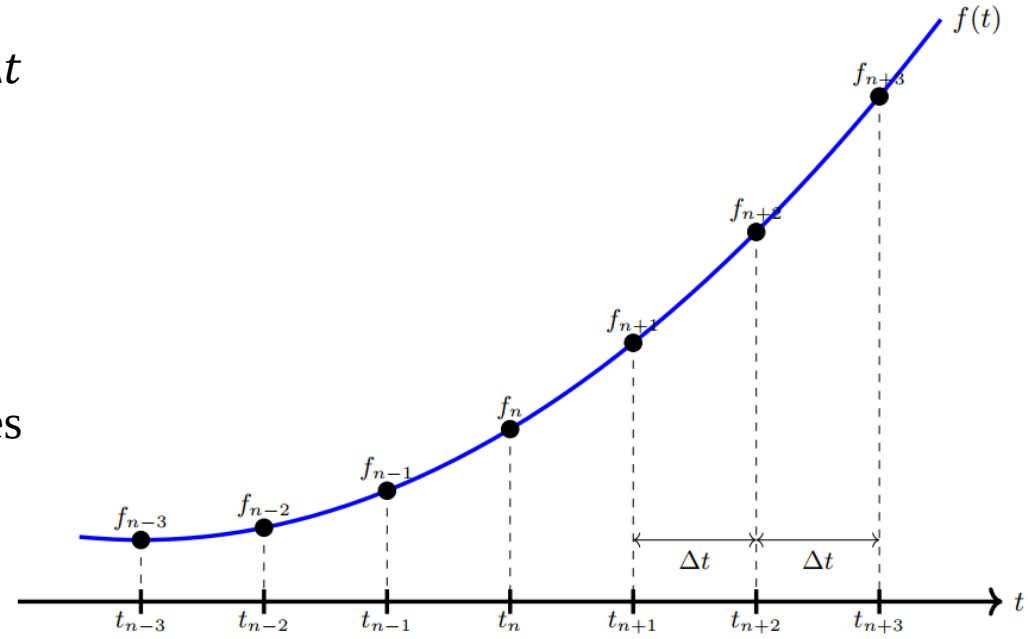
Differencing/Discretization Scheme

- Any function $f(t)$ can be discretized using FDM with a time-step size of Δt
- $\mathcal{O}(\Delta t)$ i.e., first-order backward difference approximation of derivatives at n^{th} time step can be given as:

$$\left.\frac{\mathrm{d}f(t)}{\mathrm{d}t}\right|_n = \frac{f_n - f_{n-1}}{\Delta t} \quad \& \quad \left.\frac{\mathrm{d}^2 f(t)}{\mathrm{d}t^2}\right|_n = \frac{f_n - 2f_{n-1} + f_{n-2}}{\Delta t^2}$$

- $\mathcal{O}(\Delta t^2)$ i.e., second-order backward difference approximation of derivatives at n^{th} time step can be given as:

$$\left.\frac{\mathrm{d}f(t)}{\mathrm{d}t}\right|_n = \frac{3f_n - 4f_{n-1} + f_{n-2}}{2\Delta t} \quad \& \quad \left.\frac{\mathrm{d}^2 f(t)}{\mathrm{d}t^2}\right|_n = \frac{2f_n - 5f_{n-1} + 4f_{n-2} - f_{n-3}}{\Delta t^2}$$



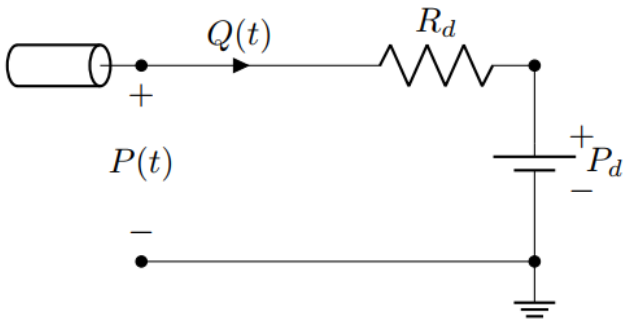
Single-Element Resistive Model

- Contains only a distal resistance R_d and a distal pressure term P_d
- The relationship between flow rate and pressure is given by:

$$P(t) = R_d Q(t) + P_d$$

- The pressure at the 3D domain's outlet at n^{th} time step can be given as:

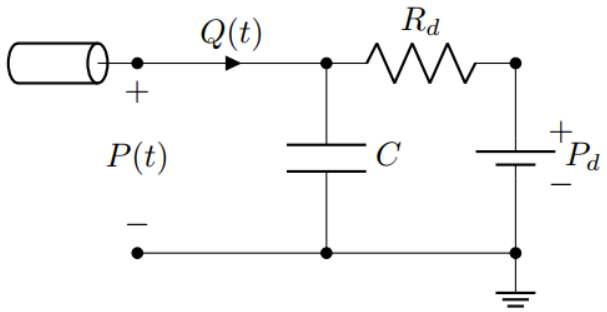
$$P_n = R_d Q_n + P_d$$



2-Element Windkessel Model

- Contains a distal resistance R_d, a capacitance C, and a distal pressure term P_d
- The relationship between flow rate and pressure is given by:

$$P(t) = R_d Q(t) + P_d - R_d C \frac{\mathrm{d}P(t)}{\mathrm{d}t}$$



- First-order backward difference approximation of pressure at n^{th} time step:

$$P_n = \left(\frac{1}{1 + \frac{R_d C}{\Delta t}}\right)\left[R_d Q_n + P_d + R_d C \frac{P_{n-1}}{\Delta t}\right]$$

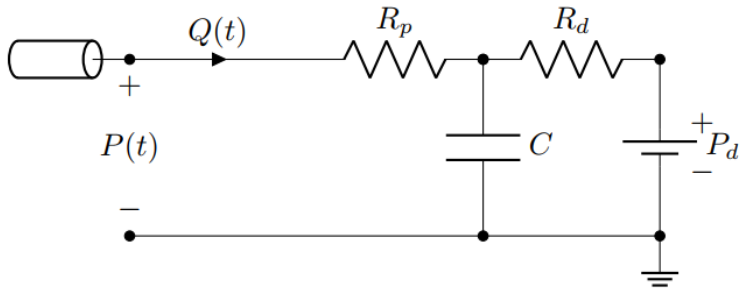
- Second-order backward difference approximation of pressure at n^{th} time step:

$$P_n = \left(\frac{1}{1 + \frac{3R_d C}{2\Delta t}}\right)\left[R_d Q_n + P_d - R_d C \frac{P_{n-2} - 4P_{n-1}}{2\Delta t}\right]$$

3-Element Windkessel Model

- Improves a 2-element windkessel model by considering proximal resistance R_p
- The relationship between flow rate and pressure is given by:

$$P(t) = R_p R_d C \frac{\mathrm{d}Q(t)}{\mathrm{d}t} + (R_p + R_d)Q(t) + P_d - R_d C \frac{\mathrm{d}P(t)}{\mathrm{d}t}$$



- First-order approximation of pressure at n^{th} time step:

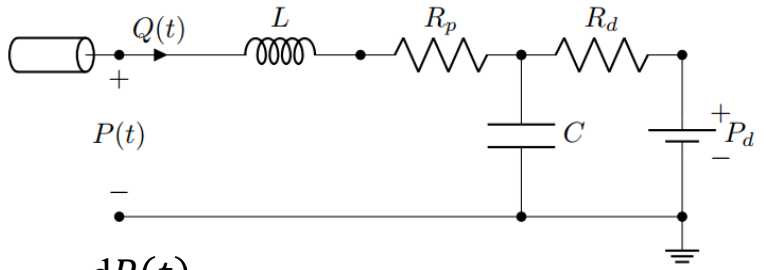
$$P_n = \left(\frac{1}{1 + \frac{R_d C}{\Delta t}}\right)\left[R_p R_d C \frac{Q_n - Q_{n-1}}{\Delta t} + (R_p + R_d)Q_n + P_d + R_d C \frac{P_{n-1}}{\Delta t}\right]$$

- Second-order approximation of pressure at n^{th} time step:

$$P_n = \left(\frac{1}{1 + \frac{3R_d C}{2\Delta t}}\right)\left[R_p R_d C \frac{3Q_n - 4Q_{n-1} + Q_{n-2}}{2\Delta t} + (R_p + R_d)Q_n + P_d - R_d C \frac{P_{n-2} - 4P_{n-1}}{2\Delta t}\right]$$

4-Element Series Windkessel Model

- Includes an inductor in the main branch of the circuit
- Accounts for the inertia to blood flow
- The relationship between flow rate and pressure is given by:



$$P(t) = (R_p + R_d)Q(t) + (L + R_p R_d C)\frac{\mathrm{d}Q(t)}{\mathrm{d}t} + R_d CL\frac{\mathrm{d}^2 Q(t)}{\mathrm{d}t^2} + P_d - R_d C\frac{\mathrm{d}P(t)}{\mathrm{d}t}$$

- First-order approximation of pressure at n^{th} time step:

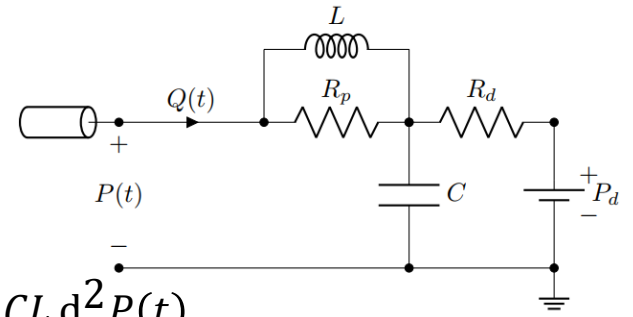
$$P_n = \left(\frac{1}{1 + \frac{R_d C}{\Delta t}}\right)\left[(R_p + R_d)Q_n + (L + R_p R_d C)\frac{Q_n - Q_{n-1}}{\Delta t} + R_d CL\frac{Q_n - 2Q_{n-1} + Q_{n-2}}{\Delta t^2} + P_d + R_d C\frac{P_{n-1}}{\Delta t}\right]$$

- Second-order approximation of pressure at n^{th} time step:

$$P_n = \left(\frac{1}{1 + \frac{3R_d C}{2\Delta t}}\right)\left[(R_p + R_d)Q_n + (L + R_p R_d C)\frac{3Q_n - 4Q_{n-1} + Q_{n-2}}{2\Delta t} + R_d CL\frac{2Q_n - 5Q_{n-1} + 4Q_{n-2} - Q_{n-3}}{\Delta t^2} + P_d - R_d C\frac{P_{n-2} - 4P_{n-1}}{2\Delta t}\right]$$

4-Element Parallel Windkessel Model

- Includes an inductor in parallel to proximal resistance
- Accounts for the inertia to blood flow



- The relationship between flow rate and pressure is given by:

$$P(t) = R_d Q(t) + L\left(1 + \frac{R_d}{R_p}\right)\frac{\mathrm{d}Q(t)}{\mathrm{d}t} + R_d CL\frac{\mathrm{d}^2 Q(t)}{\mathrm{d}t^2} + P_d - \frac{L + R_p R_d C}{R_p}\frac{\mathrm{d}P(t)}{\mathrm{d}t} - \frac{R_d CL}{R_p}\frac{\mathrm{d}^2 P(t)}{\mathrm{d}t^2}$$

- First-order approximation of pressure at n^{th} time step:

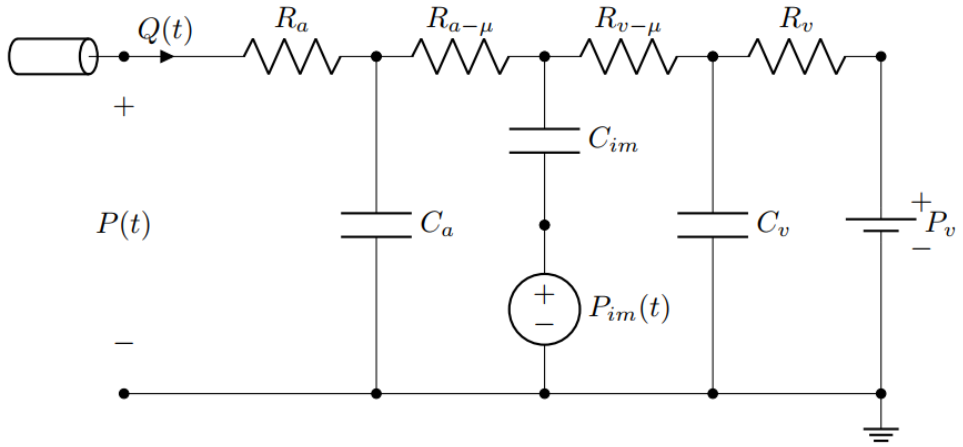
$$P_n = \left(\frac{1}{1 + \frac{L + R_p R_d C}{R_p \Delta t} + \frac{R_d CL}{R_p \Delta t^2}}\right)\left[R_d Q_n + L\left(1 + \frac{R_d}{R_p}\right)\frac{Q_n - Q_{n-1}}{\Delta t} + R_d CL\frac{Q_n - 2Q_{n-1} + Q_{n-2}}{\Delta t^2} + P_d + \frac{L + R_p R_d C}{R_p}\frac{P_{n-1}}{\Delta t} + \frac{R_d CL}{R_p}\frac{2P_{n-1} - P_{n-2}}{\Delta t^2}\right]$$

- Second-order approximation of pressure at n^{th} time step:

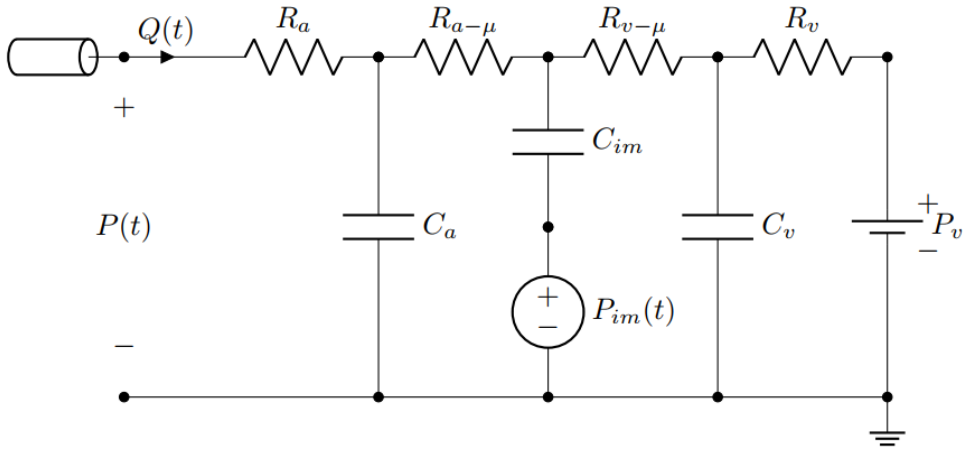
$$P_n = \left(\frac{1}{1 + \frac{3(L + R_p R_d C)}{2R_p \Delta t} + \frac{2R_d CL}{R_p \Delta t^2}}\right)\left[\begin{array}{c} R_d Q_n + L\left(1 + \frac{R_d}{R_p}\right)\frac{3Q_n - 4Q_{n-1} + Q_{n-2}}{2\Delta t} + R_d CL\frac{2Q_n - 5Q_{n-1} + 4Q_{n-2} - Q_{n-3}}{\Delta t^2} \\ + P_d - \frac{L + R_p R_d C}{R_p}\frac{P_{n-2} - 4P_{n-1}}{2\Delta t} - \frac{R_d CL}{R_p}\frac{-5P_{n-1} + 4P_{n-2} - P_{n-3}}{\Delta t^2} \end{array}\right]$$

LPN Model of Downstream Coronary Vascular Beds

- Represents the impedance of the downstream coronary vascular networks absent in the 3D computational domain
- The model has an arterial side, a venous side, and a junction between them modeling the effect of intramyocardial parameters
- The arterial side consists of a coronary arterial resistance R_a, coronary arterial compliance C_a, and coronary arterial microcirculation resistance $R_{a-\mu}$
- Similarly, the venous side consists of a coronary venous resistance R_v, coronary venous compliance C_v, and coronary venous microcirculation resistance $R_{v-\mu}$
- Myocardial compliance C_{im} represents the compliance of the myocardium, which affects the flow and pressure in the coronary arteries
- Whereas, intramyocardial pressure $P_{im}(t)$ is the time-variant pressure exerted by the heart muscle on the coronary vessels
  - Represented by either the left or right ventricular pressure, depending on the location of the coronary arteries
  - The left ventricular pressure is used for the left coronary arteries, and the right ventricular pressure for the right coronary arteries



LPN Model of Downstream Coronary Vascular Beds (Cont'd…)



- Coronary venous microcirculation compliance C_v can be removed from the original model to simplify the numerics
- Intermediate pressure $P_i(t)$ taking into account the veinous side and intramyocardial parameters can be given as:

$$P_i(t) = \left(R_{v-\mu} + R_v\right)Q(t) - \left(R_{v-\mu} + R_v\right)C_a \frac{\mathrm{d}P(t)}{\mathrm{d}t} + \left(R_{v-\mu} + R_v\right)C_{im} \frac{\mathrm{d}P_{im}(t)}{\mathrm{d}t} + P_v - \left(R_{v-\mu} + R_v\right)C_{im} \frac{\mathrm{d}P_i(t)}{\mathrm{d}t}$$

- The overall pressure $P(t)$ at 3D domain's outlet is:

$$P(t) = \left(R_{a-\mu} + R_a\right)Q(t) + R_{a-\mu}R_aC_a \frac{\mathrm{d}Q(t)}{\mathrm{d}t} + P_i(t) - R_{a-\mu}C_a \frac{\mathrm{d}P(t)}{\mathrm{d}t}$$

LPN Model of Downstream Coronary Vascular Beds (Cont'd…)

- First-order approximation of pressure at n^{th} time step:

$$P_{i,n} = \left(\frac{1}{1+\frac{(R_{v-\mu}+R_v)C_{im}}{\Delta t}}\right)\left[(R_{v-\mu}+R_v)Q_n - (R_{v-\mu}+R_v)C_a\frac{P_n-P_{n-1}}{\Delta t} + (R_{v-\mu}+R_v)C_{im}\frac{P_{im,n}-P_{im,n-1}}{\Delta t} + P_v + (R_{v-\mu}+R_v)C_{im}\frac{P_{i,n-1}}{\Delta t}\right]$$

$$P_n = \left(\frac{1}{1+\frac{R_{a-\mu}C_a}{\Delta t}}\right)\left[(R_{a-\mu}+R_a)Q_n + R_{a-\mu}R_aC_a\frac{Q_n-Q_{n-1}}{\Delta t} + P_{i,n} + R_{a-\mu}C_a\frac{P_{n-1}}{\Delta t}\right]$$

- Second-order approximation of pressure at n^{th} time step:

$$P_{i,n} = \left(\frac{1}{1+\frac{3(R_{v-\mu}+R_v)C_{im}}{2\Delta t}}\right)\left[\begin{array}{l}(R_{v-\mu}+R_v)Q_n - (R_{v-\mu}+R_v)C_a\frac{3P_n-4P_{n-1}+P_{n-2}}{2\Delta t} + (R_{v-\mu}+R_v)C_{im}\frac{3P_{im,n}-4P_{im,n-1}+P_{im,n-2}}{2\Delta t} \\ \qquad +P_v - (R_{v-\mu}+R_v)C_{im}\frac{(P_{i,n-2}-4P_{i,n-1})}{\Delta t}\end{array}\right]$$

$$P_n = \left(\frac{1}{1+\frac{3R_{a-\mu}C_a}{2\Delta t}}\right)\left[(R_{a-\mu}+R_a)Q_n + R_{a-\mu}R_aC_a\frac{3Q_n-4Q_{n-1}+Q_{n-2}}{2\Delta t} + P_{i,n} - R_{a-\mu}C_a\frac{(P_{n-2}-4P_{n-1})}{2\Delta t}\right]$$

Implementation of Windkessel BCs: Headers & Includes

- Files: Header (.H) and Source (.C)
  - `windkesselOutletPressureFvPatchScalarField.H`
  - `windkesselOutletPressureFvPatchScalarField.C`
- Definition of a header guard using the preprocessor directive
- Include the base class: `fixedValueFvPatchFields.H`

```
231 #ifndef windkesselOutletPressureFvPatchScalarField_H
232 #define windkesselOutletPressureFvPatchScalarField_H
233
234 #include "fixedValueFvPatchFields.H"
```

- Declare the namespace: `namespace Foam`
- Define enumerations
  - For windkessel models
  - For differencing scheme
- Other necessary includes in the source file

```
241 //- Enumerator for supported windkessel model types
242 enum WindkesselModel
243 {
244     Resistive = 0,      // Simple resistive model (R)
245     WK2 = 1,            // 2-element Windkessel model (RC)
246     WK3 = 2,            // 3-element Windkessel model (RCR)
247     WK4Series = 3,      // 4-element Windkessel with inductance in series (RCRL)
248     WK4Parallel = 4     // 4-element Windkessel with inductance in parallel
249 };
250
251 //- Enumerator for for time differencing schemes
252 enum DifferencingScheme
253 {
254     firstOrder = 0,     // First-order backward differencing scheme
255     secondOrder = 1     // Second-order backward differencing scheme
256 };
```

```
26 #include "windkesselOutletPressureFvPatchScalarField.H"
27 #include "addToRunTimeSelectionTable.H"
28 #include "fvPatchFieldMapper.H"
29 #include "volFields.H"
30 #include "surfaceFields.H"
31 #include "backwardDdtScheme.H"
32 #include "word.H"
```

Windkessel BCs: Class Declaration

- Declare the class as a subclass of the base class
  - Encapsulates the properties and methods required for the windkessel model

```
282 class windkesselOutletPressureFvPatchScalarField
283 :
284     public fixedValueFvPatchScalarField
```

- Declare windkessel parameters, state variable, LPN configurations, and time index as private data
- Runtime type information

```
316 public:
317
318     //- Runtime type information
319     TypeName("windkesselOutletPressure");
```

```
287 private:
288
289     // Private data
290
291         //- Windkessel parameters
292         scalar Rp_;      // Proximal resistance (Rp)
293         scalar Rd_;      // Distal resistance (Rd)
294         scalar C_;       // Compliance or capacitance (C)
295         scalar L_;       // Inductance (L)
296         scalar Pd_;      // Distal pressure (Pd)
297         scalar rho_;     // Fluid density (rho)
298
299         //- Windkessel state variables
300         scalar Pooo_;    // Pressure three time steps ago
301         scalar Poo_;     // Pressure two time steps ago
302         scalar Po_;      // Previous time-step pressure
303         scalar Pn_;      // Current pressure
304         scalar Qooo_;    // Flow rate three time steps ago
305         scalar Qoo_;     // Flow rate two time steps ago
306         scalar Qo_;      // Previous time-step flow rate
307         scalar Qn_;      // Current flow rate
308
309         //- Windkessel LPN configuration
310         WindkesselModel windkesselModel_;   // Selected Windkessel model type
311         DifferencingScheme diffScheme_;     // Selected time differencing scheme
312
313         //- Time index
314         label timeIndex_;   // Tracks the simulation time index
```

Windkessel BCs: Default Constructor

- Accepts a patch and an internal field as input, allowing for the basic setup of the boundary condition

```
321     // Constructors
322
323         //- Construct from patch and internal field
324         windkesselOutletPressureFvPatchScalarField
325         (
326             const fvPatch&,
327             const DimensionedField<scalar, volMesh>&
328         );
```

- Initializes the object with default values is defined to assign predefined values to the class variables

```
40  //- Constructor: Initialize with default values
41  Foam::windkesselOutletPressureFvPatchScalarField::
42  windkesselOutletPressureFvPatchScalarField
43  (
44      const fvPatch& p,
45      const DimensionedField<scalar, volMesh>& iF
46  )
47  :
48      fixedValueFvPatchScalarField(p, iF), // Call base class constructor
49      Rp_(1),      // Proximal resistance
50      Rd_(1),      // Distal resistance
51      C_(1),       // Compliance
52      L_(1),       // Inductance
53      Pd_(0),      // Distal pressure
54      rho_(1),     // Fluid density
55      Pooo_(0), Poo_(0), Po_(0), Pn_(0), // Initialize pressures to zero
56      Qooo_(0), Qoo_(0), Qo_(0), Qn_(0), // Initialize flow rates to zero
57      windkesselModel_(WK3),      // Default windkessel model
58      diffScheme_(secondOrder),  // Default differencing scheme
59      timeIndex_(-1)  // Initialize time index to invalid
60  {}
```

Windkessel BCs: Dictionary Constructor

- Provides additional functionality by enabling the initialization of the class with a dictionary, which can contain additional parameters for configuring the boundary condition

```
330         //- Construct from patch, internal field, and dictionary
331         windkesselOutletPressureFvPatchScalarField
332         (
333             const fvPatch&,
334             const DimensionedField<scalar, volMesh>&,
335             const dictionary&
336         );
```

- Initializes the object from a dictionary allows for more flexible parameter initialization by extracting values directly from a dictionary, typically used for user input.

```
62  //- Constructor: Initialize from dictionary (typically used in user input)
63  Foam::windkesselOutletPressureFvPatchScalarField::
64  windkesselOutletPressureFvPatchScalarField
65  (
66      const fvPatch& p,
67      const DimensionedField<scalar, volMesh>& iF,
68      const dictionary& dict
69  )
70  :
71      fixedValueFvPatchScalarField(p, iF, dict),      // Call base class constructor
72      Rp_(dict.lookupOrDefault<scalar>("Rp", 1)),     // Proximal resistance
73      Rd_(dict.lookupOrDefault<scalar>("Rd", 1)),     // Distal resistance
74      C_(dict.lookupOrDefault<scalar>("C", 1)),       // Compliance
75      L_(dict.lookupOrDefault<scalar>("L", 1)),       // Inductance
76      Pd_(dict.lookupOrDefault<scalar>("Pd", 0)),     // Distal pressure
77      rho_(dict.lookupOrDefault<scalar>("rho", 1)),   // Fluid density
78      Pooo_(dict.lookupOrDefault<scalar>("Pooo", 0)),
79      Poo_(dict.lookupOrDefault<scalar>("Poo", 0)),
80      Po_(dict.lookupOrDefault<scalar>("Po", 0)),
81      Pn_(dict.lookupOrDefault<scalar>("Pn", 0)),
82      Qooo_(dict.lookupOrDefault<scalar>("Qooo", 0)),
83      Qoo_(dict.lookupOrDefault<scalar>("Qoo", 0)),
84      Qo_(dict.lookupOrDefault<scalar>("Qo", 0)),
85      Qn_(dict.lookupOrDefault<scalar>("Qn", 0)),
86      timeIndex_(-1)  // Time index reset
```

Windkessel BCs: Dictionary Constructor (Cont'd…)

- Read user-specified windkessel model, and print model properties

```
{

    Info << "\n\nApplying windkesselOutletPressure BC on patch: " << patch().name() << endl;

    //- Retrieve the model as a string and map to the enumerator
    word WKModelStr = dict.lookupOrDefault<word>("windkesselModel", "WK3");

    if (WKModelStr == "Resistive")
    {
        windkesselModel_ = Resistive;

        //- Print Windkessel model properties for debugging/verification
        Info << "\n\nProperties of Windkessel Model: \n"
             << "Model Type: 1-Element (Resistive) \n"
             << "Distal Resistance: Rd = " << Rd_ << " kgm^-4s^-1 \n"
             << "Distal Pressure: Pd = " << Pd_ << " Pa \n"
             << "Density: rho = " << rho_ << " kgm^-3 \n" << endl;
    }
    else if (WKModelStr == "WK2")
    {
        windkesselModel_ = WK2;

        //- Print Windkessel model properties for debugging/verification
        Info << "\n\nProperties of Windkessel Model: \n"
             << "Model Type: 2-Element (RC) \n"
             << "Distal Resistance: Rd = " << Rd_ << " kgm^-4s^-1 \n"
             << "Compliance: C = " << Rd_ << " m^4s^2kg^-1 \n"
             << "Distal Pressure: Pd = " << C_ << " Pa \n"
             << "Density: rho = " << rho_ << " kgm^-3 \n" << endl;
    }
    else if (WKModelStr == "WK3")
    {
        windkesselModel_ = WK3;

        //- Print Windkessel model properties for debugging/verification
        Info << "\n\nProperties of Windkessel Model: \n"
             << "Model Type: 3-Element (RCR) \n"
             << "Proximal Resistance: Rp = " << Rp_ << " kgm^-4s^-1 \n"
             << "Distal Resistance: Rd = " << Rd_ << " kgm^-4s^-1 \n"
             << "Compliance: C = " << C_ << " m^4s^2kg^-1 \n"
             << "Distal Pressure: Pd = " << Pd_ << " Pa \n"
             << "Density: rho = " << rho_ << " kgm^-3 \n" << endl;
    }
    else if (WKModelStr == "WK4Series")
    {
        windkesselModel_ = WK4Series;

        //- Print Windkessel model properties for debugging/verification
        Info << "\n\nProperties of Windkessel Model: \n"
             << "Model Type: 4-Element (RCRL with L in Series to Rp) \n"
             << "Proximal Resistance: Rp = " << Rp_ << " kgm^-4s^-1 \n"
             << "Distal Resistance: Rd = " << Rd_ << " kgm^-4s^-1 \n"
             << "Compliance: C = " << C_ << " m^4s^2kg^-1 \n"
             << "Inertance: L = " << L_ << " kgm^-4 \n"
             << "Distal Pressure: Pd = " << Pd_ << " Pa \n"
             << "Density: rho = " << rho_ << " kgm^-3 \n" << endl;
    }
    else if (WKModelStr == "WK4Parallel")
    {
        windkesselModel_ = WK4Parallel;

        //- Print Windkessel model properties for debugging/verification
        Info << "\n\nProperties of Windkessel Model: \n"
             << "Model Type: 4-Element (RCRL with L in Parallel to Rp) \n"
             << "Proximal Resistance: Rp = " << Rp_ << " kgm^-4s^-1 \n"
             << "Distal Resistance: Rd = " << Rd_ << " kgm^-4s^-1 \n"
             << "Compliance: C = " << C_ << " m^4s^2kg^-1 \n"
             << "Inertance: L = " << L_ << " kgm^-4 \n"
             << "Distal Pressure: Pd = " << Pd_ << " Pa \n"
             << "Density: rho = " << rho_ << " kgm^-3 \n" << endl;
    }
```

Windkessel BCs: Dictionary Constructor (Cont'd…)

- Read and print user-specified differencing scheme

```
166     //- Retrieve the scheme as a string and map to the enumerator
167     word schemeStr = dict.lookupOrDefault<word>("diffScheme", "secondOrder");
168
169     Info << "Differencing Scheme for Windkessel Model: " << schemeStr << endl;
170
171     if (schemeStr == "firstOrder")
172     {
173         diffScheme_ = firstOrder;
174     }
175     else if (schemeStr == "secondOrder")
176     {
177         diffScheme_ = secondOrder;
178     }
179     else
180     {
181         FatalErrorInFunction << "\n\nUnknown Differencing Scheme: " << schemeStr
182                              << "\nValid Differencing Schemes (diffScheme) are : \n\n"
183                              << " 2 \n ( \n firstOrder \n secondOrder \n ) \n"
184                              << exit(FatalError);
185     }
186
187 }
```

Windkessel BCs: Mapping Constructor

- Supports the mapping of an existing object onto a new patch, which is useful when the simulation mesh is modified

```
338         //- Map existing object onto a new patch
339         windkesselOutletPressureFvPatchScalarField
340         (
341             const windkesselOutletPressureFvPatchScalarField&,
342             const fvPatch&,
343             const DimensionedField<scalar, volMesh>&,
344             const fvPatchFieldMapper&
345         );
```

- Maps an existing field onto a new patch facilitates the copying of the boundary condition from one patch to another, with an optional field mapping operation
- Copies the relevant properties, such as the windkessel parameters, historical pressure and flow rate variables, and the model selection
- Allows the boundary condition to be reassigned to a new patch, ensuring that all the properties are appropriately transferred

```
189 //- Constructor: Map an existing field onto a new patch
190 Foam::windkesselOutletPressureFvPatchScalarField::
191 windkesselOutletPressureFvPatchScalarField
192 (
193     const windkesselOutletPressureFvPatchScalarField& ptf, // Existing field
194     const fvPatch& p, // New patch to map onto
195     const DimensionedField<scalar, volMesh>& iF, // Internal field reference
196     const fvPatchFieldMapper& mapper // Field mapper for mapping operations
197 )
198 :
199     fixedValueFvPatchScalarField(ptf, p, iF, mapper), // Map base class properties
200     Rp_(ptf.Rp_),      // Copy proximal resistance parameter
201     Rd_(ptf.Rd_),      // Copy distal resistance parameter
202     C_(ptf.C_),        // Copy compliance parameter
203     L_(ptf.L_),        // Copy inertance parameter
204     Pd_(ptf.Pd_),      // Copy distal pressure parameter
205     rho_(ptf.rho_),    // Copy fluid density
206     Pooo_(ptf.Pooo_), // Copy previous state pressure variables
207     Poo_(ptf.Poo_),
208     Po_(ptf.Po_),
209     Pn_(ptf.Pn_),
210     Qooo_(ptf.Qooo_), // Copy previous state flow rate variables
211     Qoo_(ptf.Qoo_),
212     Qo_(ptf.Qo_),
213     Qn_(ptf.Qn_),
214     windkesselModel_(ptf.windkesselModel_), // Copy Windkessel model type
215     diffScheme_(ptf.diffScheme_), // Copy numerical differencing scheme
216     timeIndex_(ptf.timeIndex_) // Copy time index
217 {}
```

Windkessel BCs: Copy Constructor

- Allows the creation of a new object based on an existing instance
- Includes a method to construct and return a clone, ensuring that an identical copy of the object can be created when needed

```
347         //- Copy constructor
348         windkesselOutletPressureFvPatchScalarField
349         (
350             const windkesselOutletPressureFvPatchScalarField&
351         );
352
353         //- Construct and return a clone
354         virtual tmp<fvPatchScalarField> clone() const
355         {
356             return tmp<fvPatchScalarField>
357             (
358                 new windkesselOutletPressureFvPatchScalarField(*this)
359             );
360         }
```

- Creates a duplicate field, acceptind an existing object as input
- Copies all its properties, including windkessel parameters, pressure and flow rate history, and model type
- Essentially a direct duplication of the object's state, enabling the creation of identical copies when necessary

```
219 //- Copy constructor: Create a duplicate field with all properties
220 Foam::windkesselOutletPressureFvPatchScalarField::
221 windkesselOutletPressureFvPatchScalarField
222 (
223     const windkesselOutletPressureFvPatchScalarField& wkpsf // Source field
224 )
225 :
226     fixedValueFvPatchScalarField(wkpsf), // Copy base class properties
227     Rp_(wkpsf.Rp_),      // Copy proximal resistance
228     Rd_(wkpsf.Rd_),      // Copy distal resistance
229     C_(wkpsf.C_),        // Copy compliance
230     L_(wkpsf.L_),        // Copy inertance
231     Pd_(wkpsf.Pd_),      // Copy distal pressure
232     rho_(wkpsf.rho_),    // Copy fluid density
233     Pooo_(wkpsf.Pooo_), // Copy pressure variables
234     Poo_(wkpsf.Poo_),
235     Po_(wkpsf.Po_),
236     Pn_(wkpsf.Pn_),
237     Qooo_(wkpsf.Qooo_), // Copy flow rate variables
238     Qoo_(wkpsf.Qoo_),
239     Qo_(wkpsf.Qo_),
240     Qn_(wkpsf.Qn_),
241     windkesselModel_(wkpsf.windkesselModel_), // Copy Windkessel model type
242     diffScheme_(wkpsf.diffScheme_), // Copy differencing scheme
243     timeIndex_(wkpsf.timeIndex_) // Copy time index
244 {}
```

Windkessel BCs: Another Copy Constructor

- Facilitates the creation of an object by setting a reference to an internal field, ensuring proper data linkage

```
362         //- Construct as copy setting internal field reference
363         windkesselOutletPressureFvPatchScalarField
364         (
365             const windkesselOutletPressureFvPatchScalarField&,
366             const DimensionedField<scalar, volMesh>&
367         );
368
369         //- Construct and return a clone setting internal field reference
370         virtual tmp<fvPatchScalarField> clone
371         (
372             const DimensionedField<scalar, volMesh>& iF
373         ) const
374         {
375             return tmp<fvPatchScalarField>
376             (
377                 new windkesselOutletPressureFvPatchScalarField(*this, iF)
378             );
379         }
```

- Copies all the class properties
- Ensures that the internal field reference is updated
- suitable for scenarios where the internal field needs to be modified or reassigned without altering other properties of the object

```
246 //- Copy constructor with new internal field reference
247 Foam::windkesselOutletPressureFvPatchScalarField::
248 windkesselOutletPressureFvPatchScalarField
249 (
250     const windkesselOutletPressureFvPatchScalarField& wkpsf, // Source field
251     const DimensionedField<scalar, volMesh>& iF // New internal field
252 )
253 :
254     fixedValueFvPatchScalarField(wkpsf, iF), // Map base class properties
255     Rp_(wkpsf.Rp_),     // Copy proximal resistance
256     Rd_(wkpsf.Rd_),     // Copy distal resistance
257     C_(wkpsf.C_),       // Copy compliance
258     L_(wkpsf.L_),       // Copy inertance
259     Pd_(wkpsf.Pd_),     // Copy distal pressure
260     rho_(wkpsf.rho_),   // Copy fluid density
261     Pooo_(wkpsf.Pooo_), // Copy pressure variables
262     Poo_(wkpsf.Poo_),
263     Po_(wkpsf.Po_),
264     Pn_(wkpsf.Pn_),
265     Qooo_(wkpsf.Qooo_), // Copy flow rate variables
266     Qoo_(wkpsf.Qoo_),
267     Qo_(wkpsf.Qo_),
268     Qn_(wkpsf.Qn_),
269     windkesselModel_(wkpsf.windkesselModel_), // Copy Windkessel model type
270     diffScheme_(wkpsf.diffScheme_), // Copy differencing scheme
271     timeIndex_(wkpsf.timeIndex_) // Copy time index
272 {}
```

Windkessel BCs: Member Function Declaration

- Two important member functions: evaluation and utility functions

```
381     // Member functions
382
383         //- Evaluation function to update coefficients for the boundary
384         //  condition based on flow rate and model parameters
385         virtual void updateCoeffs();
386
387         //- Write boundary condition data to output stream
388         virtual void write(Ostream&) const;
```

- `updateCoeffs()` function: Responsible for updating the coefficients of the boundary condition based on the flow rate and pressure histories and model parameters
- `write()` function: Responsible for outputting the boundary condition data to an output stream

```
275 // * * * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * //
276
277 //- Update coefficients for boundary condition
278 void Foam::windkesselOutletPressureFvPatchScalarField::updateCoeffs()
```

```
487 //- Write the boundary condition properties to an output stream
488 void Foam::windkesselOutletPressureFvPatchScalarField::write(Ostream& os) const
```

Windkessel BCs: updateCoeffs() Member Function

- Check of if the coefficients are updated
- Retrieve/compute essential data: time step size, flux through the patch, pressure, total patch area

```
280     //- Check if coefficients have already been updated for this time step
281     if (updated())
282     {
283         return;
284     }
285
286     //- Retrieve the time step size from the database
287     const scalar dt = db().time().deltaTValue();
288
289     //- Retrieve the flux field (phi) from the database
290     const surfaceScalarField& phi =
291         db().lookupObject<surfaceScalarField>("phi");
292
293     //- Retrieve the boundary pressure field from the database
294     const fvPatchField<scalar>& p =
295         db().lookupObject<volScalarField>("p").boundaryField()[this->patch().index()];
296
297     //- Calculate the total area of the patch
298     scalar area = gSum(patch().magSf());
```

- Check if the time has advanced
- Update flow rate and pressure histories

```
300     //- Update pressure and flow rate histories if the time index has advanced
301     if (db().time().timeIndex() != timeIndex_)
302     {
303         timeIndex_ = db().time().timeIndex();
304
305         //- Update pressure history variables
306         Pooo_ = Poo_;
307         Poo_ = Po_;
308         Po_ = Pn_;
309         Pn_ = gSum(p * patch().magSf()) / area; // Compute mean pressure over the patch
310
311         //- Update flow rate history variables
312         Qooo_ = Qoo_;
313         Qoo_ = Qo_;
314         Qo_ = Qn_;
315         Qn_ = gSum(phi.boundaryField()[patch().index()]); // Compute total flux
316     }
```

Windkessel BCs: updateCoeffs() Member Function (Cont'd…)

- Check the time index
- Switch to the selected windkessel model
- Switch to the selected differencing scheme
- Calculate the pressure value using discretized LPN equations

```
318     //- Calculate new pressure using backward differencing scheme
319     if
320     (
321         db().time().timeIndex()>1 // Perform calculations only if the time index is greater than 1
322     )
323     {
324         //- Switch to the specified windkessel model
325         switch (windkesselModel_)
326         {
327             case Resistive: // Single-element resistive model (R)
328 
329                 Pn_ = Rd_ * Qn_ + Pd_;
330 
331                 break;
332 
333             case WK2: // 2-element windkessel model (RC)
334 
335                 //- Switch to the specified differencing scheme
336                 switch (diffScheme_)
337                 {
338                     case firstOrder:
339 
340                         Pn_ = Rd_ * Qn_
341                             + Pd_
342                             + Rd_ * C_ * Po_ / dt;
343 
344                         Pn_ /= (1.0 + Rd_ * C_ / dt) + SMALL;
345 
346                         break;
```

```
347 
348                     case secondOrder:
349 
350                         Pn_ = Rd_ * Qn_
351                             + Pd_
352                             - Rd_ * C_ * ((Poo_ - 4 * Po_) / (2 * dt));
353 
354                         Pn_ /= (1.0 + 3 * Rd_ * C_ / (2 * dt)) + SMALL;
355 
356                         break;
357 
358                     default:
359 
360                         FatalErrorInFunction << "Unknown differencing scheme!" << exit(FatalError);
361                 }
```

Windkessel BCs: updateCoeffs() Member Function (Cont'd...)

```
365             case WK3: // 3-element windkessel model (RCR)
366
367                 //- Switch to the specified differencing scheme
368                 switch (diffScheme_)
369                 {
370                     case firstOrder:
371
372                         Pn_ = Rp_ * Rd_ * C_ * ((Qn_ - Qo_) / dt)
373                             + (Rp_ + Rd_) * Qn_
374                             + Pd_
375                             + Rd_ * C_ * (Po_ / dt);
376
377                         Pn_ /= (1.0 + Rd_ * C_ / dt) + SMALL;
378
379                         break;
380
381                     case secondOrder:
382
383                         Pn_ = Rp_ * Rd_ * C_ * ((3 * Qn_ - 4 * Qo_ + Qoo_) / (2 * dt))
384                             + (Rp_ + Rd_) * Qn_
385                             + Pd_
386                             - Rd_ * C_ * ((Poo_ - 4 * Po_) / (2 * dt));
387
388                         Pn_ /= (1.0 + 3 * Rd_ * C_  / (2 * dt)) + SMALL;
389
390                         break;
```

```
399             case WK4Series: // 4-element series windkessel model (RCRL-Series)
400
401                 //- Switch to the specified differencing scheme
402                 switch (diffScheme_)
403                 {
404                     case firstOrder:
405
406                         Pn_ = (Rp_ + Rd_) * Qn_
407                             + (L_ + Rp_ * Rd_ * C_) * ((Qn_ - Qo_) / dt)
408                             + Rd_ * C_ * L_ * ((Qn_ - 2 * Qo_ + Qoo_) / (pow (dt, 2)))
409                             + Pd_
410                             + Rd_ * C_ * (Po_ / dt);
411
412                         Pn_ /= (1.0 + Rd_ * C_ / dt) + SMALL;
413
414                         break;
415
416                     case secondOrder:
417
418                         Pn_ = (Rp_ + Rd_) * Qn_
419                             + (L_ + Rp_ * Rd_ * C_) * ((3 * Qn_ - 4 * Qo_ + Qoo_) / (2 * dt))
420                             + Rd_ * C_ * L_ * ((2 * Qn_ - 5 * Qo_ + 4 * Qoo_ - Qooo_) / (pow (dt, 2)))
421                             + Pd_
422                             - Rd_ * C_ * ((Poo_ - 4 * Po_) / (2 * dt));
423
424                         Pn_ /= (1.0 + 3 * Rd_ * C_  / (2 * dt)) + SMALL;
425
426                         break;
```

Windkessel BCs: updateCoeffs() Member Function (Cont'd…)

```
435             case WK4Parallel: // 4-element parallel windkessel model (RCRL-Parallel)
436
437                 //- Switch to the specified differencing scheme
438                 switch (diffScheme_)
439                 {
440                     case firstOrder:
441
442                         Pn_ = Rd_ * Qn_
443                             + L_ * (1 + Rd_ / Rp_ ) * ((Qn_ - Qo_) / dt)
444                             + Rd_ * C_ * L_ * ((Qn_ - 2 * Qo_ + Qoo_) / (pow (dt, 2)))
445                             + Pd_
446                             + ((L_ + Rp_ * Rd_ * C_) / Rp_) * (Po_ / dt)
447                             + (Rd_ * C_ * L_ / Rp_) * ((2 * Po_ - Poo_) / (pow (dt, 2)));
448
449                         Pn_ /= (1.0 + ((L_ + Rp_ * Rd_ * C_) / (Rp_ * dt)) + (Rd_ * C_ * L_ / (Rp_ *
    pow (dt, 2)))) + SMALL;
450
451                         break;
452
453                     case secondOrder:
454
455                         Pn_ = Rd_ * Qn_
456                             + L_ * (1 + Rd_ / Rp_ ) * ((3 * Qn_ - 4 * Qo_ + Qoo_) / (2 * dt))
457                             + Rd_ * C_ * L_ * ((2 * Qn_ - 5 * Qo_ + 4 * Qoo_ - Qooo_) / (pow (dt, 2)))
458                             + Pd_
459                             - ((L_ + Rp_ * Rd_ * C_) / Rp_) * ((Poo_ - 4 * Po_) / (2 * dt))
460                             - (Rd_ * C_ * L_ / Rp_) * ((-5 * Po_ + 4 * Poo_ - Pooo_) / (pow (dt, 2)));
461
462                         Pn_ /= (1.0 + (3 * (L_ + Rp_ * Rd_ * C_) / (2 * Rp_ * dt)) + (2 * Rd_ * C_ *
    L_ / (Rp_ * pow (dt, 2)))) + SMALL;
463
464                         break;
```

- Calculate the final value of outlet pressure and apply the implicit update
- Call the base class function to finalize the update

```
480     //- Apply implicit pressure update to the boundary field
481     operator==(Pn_ / (rho_ + SMALL));
482
483     //- Call base class function to finalize the update
484     fixedValueFvPatchScalarField::updateCoeffs();
```

Windkessel BCs: write() Member Function

- Call the base class function
- Write the common, model-specific, and scheme-specific properties

```
490     //- Call the base class function to write common boundary field properties
491     fvPatchScalarField::write(os);
492
493     //- Write the common windkessel model details to the output stream
494     os.writeKeyword("Rd") << Rd_ << token::END_STATEMENT << nl; // Resistance parameter
495     os.writeKeyword("Pd") << Pd_ << token::END_STATEMENT << nl; // Prescribed pressure
496     os.writeKeyword("rho") << rho_ << token::END_STATEMENT << nl; // Fluid density
498     //- Write the specific windkessel model details to the output stream
499     switch (windkesselModel_)
500     {
501         case Resistive:
502             os.writeKeyword("windkesselModel") << "Resistive" << token::END_STATEMENT << nl;
503             break;
504
505         case WK2:
506             os.writeKeyword("windkesselModel") << "WK2" << token::END_STATEMENT << nl;
507             os.writeKeyword("C") << C_ << token::END_STATEMENT << nl; // Compliance
508             break;
509
510         case WK3:
511             os.writeKeyword("windkesselModel") << "WK3" << token::END_STATEMENT << nl;
512             os.writeKeyword("Rp") << Rp_ << token::END_STATEMENT << nl; // Proximal resistance
513             os.writeKeyword("C") << C_ << token::END_STATEMENT << nl; // Compliance
514             break;
515
516         case WK4Series:
517             os.writeKeyword("windkesselModel") << "WK4Series" << token::END_STATEMENT << nl;
518             os.writeKeyword("Rp") << Rp_ << token::END_STATEMENT << nl;
519             os.writeKeyword("C") << C_ << token::END_STATEMENT << nl;
520             os.writeKeyword("L") << L_ << token::END_STATEMENT << nl; // Inductance
521             break;
```

```
523         case WK4Parallel:
524             os.writeKeyword("windkesselModel") << "WK4Parallel" << token::END_STATEMENT << nl;
525             os.writeKeyword("Rp") << Rp_ << token::END_STATEMENT << nl;
526             os.writeKeyword("C") << C_ << token::END_STATEMENT << nl;
527             os.writeKeyword("L") << L_ << token::END_STATEMENT << nl;
528             break;
529
530         default:
531             FatalErrorInFunction << "Unknown Windkessel Model: " << windkesselModel_ << exit(
        FatalError);
532     }
534     //- Write the differencing scheme used for calculations
535     switch (diffScheme_)
536     {
537         case firstOrder:
538             os.writeKeyword("diffScheme") << "firstOrder" << token::END_STATEMENT << nl;
539             break;
540
541         case secondOrder:
542             os.writeKeyword("diffScheme") << "secondOrder" << token::END_STATEMENT << nl;
543             break;
544
545         default:
546             FatalErrorInFunction << "Unknown differencing scheme: " << diffScheme_ << exit(FatalError)
        ;
547     }
```

Windkessel BCs: write() Member Function

- Write flow rate and pressure histories necessary for a perfect restart of the simulation
- Write the boundary field value entry

```
549     //- Write flow history parameters
550     os.writeKeyword("Qooo") << Qooo_ << token::END_STATEMENT << nl; // Third past flow rate
551     os.writeKeyword("Qoo") << Qoo_ << token::END_STATEMENT << nl; // Second past flow rate
552     os.writeKeyword("Qo") << Qo_ << token::END_STATEMENT << nl; // Previous flow rate
553     os.writeKeyword("Qn") << Qn_ << token::END_STATEMENT << nl; // Current flow rate
554
555     //- Write pressure history parameters
556     os.writeKeyword("Pooo") << Pooo_ << token::END_STATEMENT << nl; // Third past pressure
557     os.writeKeyword("Poo") << Poo_ << token::END_STATEMENT << nl; // Second past pressure
558     os.writeKeyword("Po") << Po_ << token::END_STATEMENT << nl; // Previous pressure
559     os.writeKeyword("Pn") << Pn_ << token::END_STATEMENT << nl; // Current pressure
560
561     //- Write the boundary field value entry to the output stream
562     writeEntry("value", os);
```

Windkessel BCs: Runtime Type Selection and Usage

- Register the boundary condition in the namespace by defining the boundary field type for the class

```
567 namespace Foam
568 {
569     //- Define the boundary field type for this class
570     makePatchTypeField
571     (
572         fvPatchScalarField,                          // Base class
573         windkesselOutletPressureFvPatchScalarField  // Derived class
574     );
575 }
```

- Note the method to prescribe the windkessel outlet pressure boundary condition in 0/p dictionary of the OpenFOAM case

```
1  <patchName>
2  {
3  type                windkesselOutletPressure;
4  windkesselModel     <Resistive, WK2, WK3, WK4Series or WK4Parallel>; // Type of model, Default: WK3
5  Rp                  <value>; // Proximal resistance in [kgm^-4s^-1], Default: 1
6  Rd                  <value>; // Distal resistance in [kgm^-4s^-1], Default: 1
7  C                   <value>; // Capacitance or compliance in [m^4s^2kg^-1], Default: 1
8  L                   <value>; // Inductance or inertance in [kgm^-4], Default: 1
9  Pd                  <value>; // Distal pressure in [Pa], Default: 0
10 rho                 <value>; // Fluid density in [kgm^-3], Default: 1
11 diffScheme          <firstOrder or secondOrder>; // Differencing scheme, Default: secondOrder
12 value               uniform 0;
13 }
```

Implementation of Coronary BCs: Headers & Includes

- Files: Header (.H) and Source (.C)
  - `coronaryOutletPressureFvPatchScalarField.H`
  - `coronaryOutletPressureFvPatchScalarField.C`
- Definition of a header guard using the preprocessor directive
- Include the base class: `fixedValueFvPatchFields.H`

```
#ifndef coronaryOutletPressureFvPatchScalarField_H
#define coronaryOutletPressureFvPatchScalarField_H

#include "fixedValueFvPatchFields.H"
#include "scalar.H"
#include "word.H"
#include <vector>  // Required for std::vector
#include <utility> // Required for std::pair
```

- Declare the namespace: `namespace Foam`
- Define enumerations
  - For differencing scheme
- Other necessary includes in the source file

```
// Enumerator for numerical differencing schemes used in the LPN model
enum DifferencingScheme
{
    firstOrder = 0, // First-order backward differencing scheme
    secondOrder = 1 // Second-order backward differencing scheme
};
```

```
// Include the header file defining the class and related dependencies
#include "coronaryOutletPressureFvPatchScalarField.H"
#include "addToRunTimeSelectionTable.H" // Macro for runtime selection table
#include "fvPatchFieldMapper.H"         // Field mapper utility
#include "volFields.H"                  // Volume fields
#include "surfaceFields.H"              // Surface fields
#include "scalar.H"                     // Scalar type definition
#include <fstream>                      // File handling for reading/writing
#include <string.h>                     // String manipulation utilities
#include "IOdictionary.H"               // Dictionary I/O operations
#include "fileName.H"                   // FileName handling
#include "OSspecific.H"                 // OS-specific operations (e.g., paths)
#include <cstdlib>                      // Environment variable handling
```

Coronary BCs: Class Declaration

- Declare the class as a subclass of the base class
  - Encapsulates the properties and methods required for the coronary LPN model

```
189 class coronaryOutletPressureFvPatchScalarField
190 :
191     public fixedValueFvPatchScalarField
```

- Declare LPN parameters, state variable, LPN configurations, and time index as private data
- Note the changes to handle intramyocardial pressure time-series data
- Runtime type information

```
236 public:
237
238     //- Runtime type information
239     TypeName("coronaryOutletPressure");
```

```
194 private:
195
196     // Private data
197
198         //- Model parameters
199         scalar Ra_;          // Arterial resistance
200         scalar Ram_;         // Micro-arterial resistance
201         scalar Rv_;          // Veinous resistance
202         scalar Rvm_;         // Micro-veinous resistance
203         scalar Ca_;          // Arterial compliance
204         scalar Cim_;         // Intramyocardial compliance
205         scalar PimScaling_;  // Scaling factor for intramyocardial pressure
206         scalar Pv_;          // Pressure distal to Rv
207         scalar rho_;         // Fluid density
208
209         //- State variables for flow rates and pressures
210         scalar Qoo_, Qo_, Qn_;       // Historical flow rate values
211         scalar Poo_, Po_, Pn_;       // Historical pressure values
212         scalar Pioo_, Pio_, Pin_;    // Historical intermediate pressure values
213         scalar Pimoo_, Pimo_, Pimn_; // Historical intramyocardial pressure values
214
215         //- LPN model configuration
216         DifferencingScheme diffScheme_; // Numerical differencing scheme
217         Foam::fileName PimFile_;      // File containing intramyocardial pressure data
218
219         //- Intramyocardial pressure data
220         std::vector<std::pair<scalar, scalar>> PimData_; // Time-series data for (time, Pim)
221
222         //- Time index
223         label timeIndex_;             // Current time index for updates
```

Coronary BCs: Constructors

- 5 constructors (similar to windkessel BC)
- Default constructor

```
243         //- Construct from patch and internal field
244         coronaryOutletPressureFvPatchScalarField
245         (
246             const fvPatch& patch,
247             const DimensionedField<scalar, volMesh>& field
248         );
```

- Dictionary constructor

```
250         //- Construct from patch, internal field, and dictionary
251         coronaryOutletPressureFvPatchScalarField
252         (
253             const fvPatch& patch,
254             const DimensionedField<scalar, volMesh>& field,
255             const dictionary& dict
256         );
```

- Mapping constructor

```
258         //- Map existing object onto a new patch
259         coronaryOutletPressureFvPatchScalarField
260         (
261             const coronaryOutletPressureFvPatchScalarField& other,
262             const fvPatch& patch,
263             const DimensionedField<scalar, volMesh>& field,
264             const fvPatchFieldMapper& mapper
265         );
```

- 2 copy constructors

```
267         //- Copy constructor
268         coronaryOutletPressureFvPatchScalarField
269         (
270             const coronaryOutletPressureFvPatchScalarField& other
271         );
272
273         //- Construct and return a clone
274         virtual tmp<fvPatchScalarField> clone() const
275         {
276             return tmp<fvPatchScalarField>
277             (
278                 new coronaryOutletPressureFvPatchScalarField(*this)
279             );
280         }
```

```
282         //- Construct as copy setting internal field reference
283         coronaryOutletPressureFvPatchScalarField
284         (
285             const coronaryOutletPressureFvPatchScalarField& other,
286             const DimensionedField<scalar, volMesh>& field
287         );
288
289         //- Construct and return a clone setting internal field reference
290         virtual tmp<fvPatchScalarField> clone
291         (
292             const DimensionedField<scalar, volMesh>& iF
293         ) const
294         {
295             return tmp<fvPatchScalarField>
296             (
297                 new coronaryOutletPressureFvPatchScalarField(*this, iF)
298             );
299         }
```

Coronary BCs: Dictionary Constructor

- Initializes the object from a dictionary
- Allows for more flexible parameter initialization
- Typically used for user input
- Provides default values if the entries are not found
- Read differencing scheme

```
    Info << "\n\nApplying coronaryOutletPressure BC on patch: " << patch().name() << endl;

    //- Extract differencing scheme from dictionary and validate
    word schemeStr = dict.lookupOrDefault<word>("diffScheme", "secondOrder");

    Info << "Differencing Scheme for Coronary LPN Model: " << schemeStr << endl;

    if (schemeStr == "firstOrder")
    {
        diffScheme_ = firstOrder; // First-order scheme
    }
    else if (schemeStr == "secondOrder")
    {
        diffScheme_ = secondOrder; // Second-order scheme
    }
    else
    {
        FatalErrorInFunction << "\n\nUnknown Differencing Scheme: " << schemeStr
                             << "\nValid Differencing Schemes (diffScheme) are : \n\n"
                             << " 2 \n ( \n firstOrder \n secondOrder \n ) \n"
                             << exit(FatalError);
    }
```

```
//- Constructor from a dictionary, typically used for user input configuration.
Foam::coronaryOutletPressureFvPatchScalarField::
coronaryOutletPressureFvPatchScalarField
(
    const fvPatch& p,                                   // Patch reference
    const DimensionedField<Foam::scalar, volMesh>& iF, // Internal field reference
    const dictionary& dict                              // Dictionary containing user-provided values
)
:
    fixedValueFvPatchScalarField(p, iF, dict), // Call the base class constructor
    Ra_(dict.lookupOrDefault<Foam::scalar>("Ra", 1)),       // Arterial resistance
    Ram_(dict.lookupOrDefault<Foam::scalar>("Ram", 1)),     // Micro-arterial resistance
    Rv_(dict.lookupOrDefault<Foam::scalar>("Rv", 1)),       // Venous resistance
    Rvm_(dict.lookupOrDefault<Foam::scalar>("Rvm", 1)),     // Micro-venous resistance
    Ca_(dict.lookupOrDefault<Foam::scalar>("Ca", 1)),       // Arterial compliance
    Cim_(dict.lookupOrDefault<Foam::scalar>("Cim", 1)),     // Intramyocardial compliance
    PimScaling_(dict.lookupOrDefault<Foam::scalar>("PimScaling", 1)), // Pim scaling factor
    Pv_(dict.lookupOrDefault<Foam::scalar>("Pv", 0)),       // Distal pressure
    rho_(dict.lookupOrDefault<Foam::scalar>("rho", 1)),     // Fluid density
    Qoo_(dict.lookupOrDefault<Foam::scalar>("Qoo", 0)),     // Historical flow rate values
    Qo_(dict.lookupOrDefault<Foam::scalar>("Qo", 0)),
    Qn_(dict.lookupOrDefault<Foam::scalar>("Qn", 0)),
    Poo_(dict.lookupOrDefault<Foam::scalar>("Poo", 0)),     // Historical pressure values
    Po_(dict.lookupOrDefault<Foam::scalar>("Po", 0)),
    Pn_(dict.lookupOrDefault<Foam::scalar>("Pn", 0)),
    Pioo_(dict.lookupOrDefault<Foam::scalar>("Pioo", 0)),   // Historical intermediate pressure values
    Pio_(dict.lookupOrDefault<Foam::scalar>("Pio", 0)),
    Pin_(dict.lookupOrDefault<Foam::scalar>("Pin", 0)),
    Pimoo_(dict.lookupOrDefault<Foam::scalar>("Pimoo", 0)), // Historical intramyocardial pressure
    Pimo_(dict.lookupOrDefault<Foam::scalar>("Pimo", 0)),
    Pimn_(dict.lookupOrDefault<Foam::scalar>("Pimn", 0)),
    timeIndex_(-1) // Initialize time index
```

Coronary BCs: Handling $P_{im}(t)$ Data

- Extract path to the file containing intramyocardial pressure data within in dictionary constructor
- Load the $P_{im}(t)$ data from the specified file

```
131     //- Extract file name for intramyocardial pressure (Pim) data
132     word PimFileStr = dict.lookupOrDefault<fileName>("PimFile", "PimData");
133
134     PimFile_ = PimFileStr;
135
136     //- Load the Pim data from the specified file
137     loadPimData(PimFile_);
```

- Helper methods to handle $P_{im}(t)$ data

```
225     // Helper methods
226
227         //- Function to interpolate intramyocardial pressure (Pim) data at a given time
228         scalar interpolatePim(const scalar& time) const;
229
230         //- Function to load intramyocardial pressure (Pim) data from a specified file
231         void loadPimData(const word& fileName);
232
233         //- Function to expands environment variables in a given string
234         static std::string expandEnvironmentVariables(const std::string& input);
```

- Intramyocardial pressure time-series data in `PimData` file
- Time-pressure value pairs

```
(
( 0.00000 2.85419827E+03 )
( 0.00100 2.88803679E+03 )
( 0.00200 2.92568009E+03 )
( 0.00300 2.96726253E+03 )
.
.
.
( 0.99710 2.78917846E+03 )
( 0.99810 2.81575383E+03 )
( 0.99910 2.81578041E+03 )
);
```

Coronary BCs: Reading $P_{im}(t)$ Data

- Loading $P_{im}(t)$ data

```
392 //- Loads intramyocardial pressure data (PimData_) from a file
393 void Foam::coronaryOutletPressureFvPatchScalarField::loadPimData(const word& fileName)
395     //- Clear existing data
396     PimData_.clear();
397
398     //- Expand environment variables in the file name
399     std::string expandedFileName = expandEnvironmentVariables(fileName);
400
401     //- Open the file
402     std::ifstream file(expandedFileName.c_str());
403
404     //- Check if the file was successfully opened
405     if (!file.is_open())
406     {
407         FatalErrorInFunction << "Cannot open intramyocardial pressure file: " << fileName << exit(
     FatalError);
408     }
410     std::string line;
411     while (std::getline(file, line)) // Read the file line by line
412     {
413         //- Remove parentheses from the line
414         line.erase(std::remove(line.begin(), line.end(), '('), line.end());
415         line.erase(std::remove(line.begin(), line.end(), ')'), line.end());
416
417         //- Skip empty or invalid lines
418         if (line.empty()) continue;
419
420         //- Parse the line into time and Pim values
421         std::istringstream iss(line);
422         scalar time, Pim;
423         if (iss >> time >> Pim) // If parsing is successful
424         {
425             PimData_.emplace_back(time, Pim); // Store the time and Pim values
426         }
427     }
428
429     file.close(); // Close the file
```

- Expanding environment variables in the file path

```
432 //- Expands environment variables in the given string
433 std::string Foam::coronaryOutletPressureFvPatchScalarField::expandEnvironmentVariables(const std::
      string& input)
434 {
435     std::string result = input; // Start with the input string
436     size_t pos = 0;
437
438     //- Look for occurrences of "$" followed by an alphabetic character (environment variable pattern)
439     while ((pos = result.find("$", pos)) != std::string::npos)
440     {
441         size_t endPos = result.find_first_not_of("ABCDEFGHIJKLMNOPQRSTUVWXYZ_0123456789", pos + 1);
442         if (endPos == std::string::npos)
443         {
444             //- No valid environment variable found after "$"
445             break;
446         }
447
448         //- Extract the environment variable name
449         std::string varName = result.substr(pos + 1, endPos - pos - 1);
450
451         //- Get the environment variable value from the system
452         const char* envValue = std::getenv(varName.c_str());
453
454         if (envValue) // If the environment variable exists
455         {
456             //- Replace the variable with its value
457             result.replace(pos, endPos - pos, envValue);
458             pos += std::string(envValue).size(); // Move past the replaced value
459         }
460         else
461         {
462             //- If the environment variable is not found, skip to the next "$"
463             pos = endPos;
464         }
465     }
466     return result; // Return the string with expanded variables
467 }
```

Coronary BCs: Interpolating $P_{im}(t)$ Data

- Helper function designed to compute the interpolated value of intramyocardial pressure `Pimn_` at current time
- Achieved using a dataset, `PimData_`, which must be pre-populated with time-pressure pairs
- Get the start and end time from the data set
- Calculate the effective time to handle periodicity
- Perform the linear interpolation
- Data set is repeated after every time period (periodic behavior)
- Final returned value is multiplied by the scaling factor, useful when using the same pressure data as $P_{im}(t)$ for both LCA and RCA (Default value: 1)

```
351 // Helper methods
352
353 //- Interpolates the intramyocardial pressure (Pim) at a given time
354 Foam::scalar Foam::coronaryOutletPressureFvPatchScalarField::interpolatePim(const scalar& time) const
356     //- Ensure PimData_ is populated
357     if (PimData_.empty())
358     {
359         FatalErrorInFunction << "PimData_ is empty. Cannot interpolate!" << exit(FatalError);
360     }
362     //- Get the start and end times of the Pim data
363     const scalar tStart = PimData_.front().first; // Start time of the dataset
364     const scalar tEnd = PimData_.back().first;   // End time of the dataset
365
366     //- Calculate the effective time using modulo to handle periodicity
367     scalar effectiveTime = tStart + std::fmod(time - tStart, tEnd - tStart);
368     if (effectiveTime < tStart)
369     {
370         effectiveTime += (tEnd - tStart); // Adjust for negative modulo results
371     }
373     //- Perform linear interpolation to find the corresponding Pim value
374     for (size_t i = 1; i < PimData_.size(); ++i)
375     {
376         if (PimData_[i - 1].first <= effectiveTime && PimData_[i].first >= effectiveTime)
377         {
378             scalar t1 = PimData_[i - 1].first;      // Previous time point
379             scalar t2 = PimData_[i].first;          // Current time point
380             scalar Pim1 = PimData_[i - 1].second;   // Pim value at previous time
381             scalar Pim2 = PimData_[i].second;       // Pim value at current time
382
383             //- Linear interpolation formula
384             return PimScaling_ * (Pim1 + (Pim2 - Pim1) * (effectiveTime - t1) / (t2 - t1));
385         }
386     }
387
388     //- Fallback case: return the last Pim value (should not typically be reached)
389     return PimData_.back().second;
```

Coronary BCs: Member Functions Declaration

- Two important member functions: evaluation and utility functions

```
301     // Member functions
302
303         //- Evaluation function to update coefficients for the boundary
304         //  condition based on flow rate and model parameters
305         virtual void updateCoeffs();
306
307         //- Write boundary condition data to output stream
308         virtual void write(Ostream& os) const;
```

- `updateCoeffs()` function: Responsible for updating the coefficients of the boundary condition based on the flow rate and pressure histories and model parameters

```
241 //- Update coefficients for boundary condition
242 void Foam::coronaryOutletPressureFvPatchScalarField::updateCoeffs()
```

- `write()` function: Responsible for outputting the boundary condition data to an output stream

```
469 //- Write the boundary condition properties to an output stream
470 void Foam::coronaryOutletPressureFvPatchScalarField::write(Ostream& os) const
```

Coronary BCs: updateCoeffs() Member Function

- Check of if the coefficients are updated
- Retrieve/compute essential data: time step size, flux through the patch, pressure, total patch area

```
244     //- Check if coefficients are already updated
245     if (updated())
246     {
247         return;
248     }
249
250     //- Get time step size and current simulation time
251     const scalar dt = db().time().deltaTValue(); // Time step size
252     const scalar currentTime = db().time().value(); // Current time
253
254     //- Retrieve the flux field (phi) from the database
255     const surfaceScalarField& phi =
256         db().lookupObject<surfaceScalarField>("phi");
257
258     //- Retrieve the boundary pressure field from the database
259     const fvPatchField<scalar>& p =
260         db().lookupObject<volScalarField>("p").boundaryField()[this->patch().index()];
261
262     //- Calculate total patch area
263     scalar area = gSum(patch().magSf());
```

- Check if the time has advanced
- Update flow rate and pressure histories

```
265     //- Update pressure and flow rate histories if time index has advanced
266     if (db().time().timeIndex() != timeIndex_)
267     {
268         timeIndex_ = db().time().timeIndex(); // Update time index
269
270         //- Update flow rate history
271         Qoo_ = Qo_;
272         Qo_ = Qn_;
273         Qn_ = gSum(phi.boundaryField()[patch().index()]); // Compute current flow rate
274
275         //- Update pressure history
276         Poo_ = Po_;
277         Po_ = Pn_;
278         Pn_ = gSum(p * patch().magSf()) / area; // Compute current pressure (area-weighted average)
279
280         //- Update Pim history
281         Pimoo_ = Pimo_;
282         Pimo_ = Pimn_;
283         Pimn_ = interpolatePim(currentTime); // Interpolate to compute current Pim
284
285         //- Update intermediate pressure history
286         Pioo_ = Pio_;
287         Pio_ = Pin_;
288     }
```

Coronary BCs: updateCoeffs() Member Function (Cont'd...)

- Check the time index
- Switch to the selected differencing scheme
- Calculate the pressure value using discretized coronary LPN equations

```
    //- Perform calculations only if time index exceeds 1
    if (db().time().timeIndex() > 1)
    {
        //- Choose differencing scheme
        switch (diffScheme_)
        {
            case firstOrder: // First-order scheme

                //- Compute intermediate pressure at current time
                Pin_ = (Rvm_ + Rv_) * Qn_
                    - (Rvm_ + Rv_) * Ca_ * ((Pn_ - Po_) / dt)
                    + (Rvm_ + Rv_) * Cim_ * ((Pimn_ - Pimo_) / dt)
                    + Pv_
                    + (Rvm_ + Rv_) * Cim_ * Pio_ / dt;

                Pin_ /= (1.0 + (Rvm_ + Rv_) * Cim_ / dt) + SMALL;

                //- Compute boundary pressure at current time
                Pn_ = (Ram_ + Ra_) * Qn_
                    + Ram_ * Ra_ * Ca_ * ((Qn_ - Qo_) / dt)
                    + Pin_
                    + Ram_ * Ca_ * (Po_ / dt);

                Pn_ /= (1.0 + Ram_ * Ca_ / dt) + SMALL;

                break;

            case secondOrder: // Second-order scheme

                //- Compute intermediate pressure at current time
                Pin_ = (Rvm_ + Rv_) * Qn_
                    - (Rvm_ + Rv_) * Ca_ * ((3 * Pn_ - 4 * Po_ + Poo_) / (2 * dt))
                    + (Rvm_ + Rv_) * Cim_ * ((3 * Pimn_ - 4 * Pimo_ + Pimoo_) / (2 * dt))
                    + Pv_
                    - (Rvm_ + Rv_) * Cim_ * (Pioo_ - 4 * Pio_) / (2 * dt);

                Pin_ /= (1.0 + 3 * (Rvm_ + Rv_) * Cim_ /(2 * dt)) + SMALL;

                //- Compute boundary pressure at current time
                Pn_ = (Ram_ + Ra_) * Qn_
                    + Ram_ * Ra_ * Ca_ * ((3 * Qn_ - 4 * Qo_ + Qoo_) / (2 * dt))
                    + Pin_
                    - Ram_ * Ca_ * (Poo_ - 4 * Po_) / (2 * dt);

                Pn_ /= (1.0 + 3 * Ram_ * Ca_ / (2 * dt)) + SMALL;

                break;

            default:
                //- Handle unknown schemes
                FatalErrorInFunction << "Unknown differencing scheme!" << exit(FatalError);
        }
    }
```

Coronary BCs: updateCoeffs() Member Function (Cont'd...)

- Calculate the final value of outlet pressure and apply the implicit update
- Call the base class function to finalize the update

```
244     //- Apply implicit update to the boundary field
245     operator==(Pn_ / (rho_ + SMALL));
246
247     //- Call base class function to finalize the update
248     fixedValueFvPatchScalarField::updateCoeffs();
```

Coronary BCs: write() Member Function

- Call the base class function
- Write the common, and scheme-specific properties

```
472     //- Call base class method to write common properties
473     fvPatchScalarField::write(os);
474
475     //- Write each parameter with its keyword and value
476     os.writeKeyword("Ra") << Ra_ << token::END_STATEMENT << nl;
477     os.writeKeyword("Ram") << Ram_ << token::END_STATEMENT << nl;
478     os.writeKeyword("Rv") << Rv_ << token::END_STATEMENT << nl;
479     os.writeKeyword("Rvm") << Rvm_ << token::END_STATEMENT << nl;
480     os.writeKeyword("Ca") << Ca_ << token::END_STATEMENT << nl;
481     os.writeKeyword("Cim") << Cim_ << token::END_STATEMENT << nl;
482     os.writeKeyword("PimFile") << PimFile_ << token::END_STATEMENT << nl;
483     os.writeKeyword("PimScaling") << PimScaling_ << token::END_STATEMENT << nl;
484    os.writeKeyword("Pv") << Pv_ << token::END_STATEMENT << nl;
485    os.writeKeyword("rho") << rho_ << token::END_STATEMENT << nl;
487    //- Write the differencing scheme
488    switch (diffScheme_)
489    {
490        case firstOrder:
491            os.writeKeyword("diffScheme") << "firstOrder" << token::END_STATEMENT << nl;
492            break;
493        case secondOrder:
494            os.writeKeyword("diffScheme") << "secondOrder" << token::END_STATEMENT << nl;
495            break;
496        default:
497            FatalErrorInFunction << "Unknown differencing scheme: " << diffScheme_ << exit(FatalError)
     ;
498    }
```

- Write flow rate and pressure histories necessary for a perfect restart of the simulation
- Write the boundary field value entry

```
500     //- Write flow history parameters
501     os.writeKeyword("Qoo") << Qoo_ << token::END_STATEMENT << nl;
502     os.writeKeyword("Qo") << Qo_ << token::END_STATEMENT << nl;
503     os.writeKeyword("Qn") << Qn_ << token::END_STATEMENT << nl;
504
505     //- Write pressure history parameters
506     os.writeKeyword("Pimoo") << Pimoo_ << token::END_STATEMENT << nl;
507     os.writeKeyword("Pimo") << Pimo_ << token::END_STATEMENT << nl;
508     os.writeKeyword("Pimn") << Pimn_ << token::END_STATEMENT << nl;
509     os.writeKeyword("Pioo") << Pioo_ << token::END_STATEMENT << nl;
510     os.writeKeyword("Pio") << Pio_ << token::END_STATEMENT << nl;
511     os.writeKeyword("Pin") << Pin_ << token::END_STATEMENT << nl;
512     os.writeKeyword("Poo") << Poo_ << token::END_STATEMENT << nl;
513     os.writeKeyword("Po") << Po_ << token::END_STATEMENT << nl;
514     os.writeKeyword("Pn") << Pn_ << token::END_STATEMENT << nl;
515
516     //- Write the boundary field value entry to the output stream
517     writeEntry("value", os);
```

Windkessel BCs: Runtime Type Selection and Usage

- Register the boundary condition in the namespace by defining the boundary field type for the class

```
522 //- Registration
523 namespace Foam
524 {
525     //- Define the boundary field type for this class
526     makePatchTypeField
527     (
528         fvPatchScalarField,                       // Base class
529         coronaryOutletPressureFvPatchScalarField  // Derived class
530     );
531 }
```

- Note the method to prescribe the coronary LPN outlet pressure boundary condition in 0/p dictionary of the OpenFOAM case

```
1  <patchName>
2  {
3  type                coronaryOutletPressure;
4  Ra                  <value>;        //Arterial resistance in [kgm^-4s^-1], Default: 1
5  Ram                 <value>;        //Micro-arterial resistance in [kgm^-4s^-1], Default: 1
6  Rv                  <value>;        //Veinous resistance in [kgm^-4s^-1], Default: 1
7  Rvm                 <value>;        //Micro-venous resistance in [kgm^-4s^-1], Default: 0
8  Ca                  <value>;        //Arterial compliance in [m^4s^2kg^-1], Default: 1
9  Cim                 <value>;        //Intramyocardial compliance in in [m^4s^2kg^-1], Default: 1
10 PimFile             "<Path to Pim Data File>"; //Path to Pim time-series data file, Default: PimData
11 PimScaling          <value>;        // Intramyocardial pressure scaling factor, Default: 1
12 Pv                  <value>;        //Pressure distal to Rv in [Pa], Default: 0
13 rho                 <value>;        // Fluid density in [kgm^-3], Default: 1
14 diffScheme          <firstOrder or secondOrder>; // Differencing scheme, Default: secondOrder
15 value               uniform 0;
16 }
```

LPN BCs: Directory Structure & Compilation

- `Make/options` file: Contains various includes

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude

LIB_LIBS = \
    -lfiniteVolume \
    -lmeshTools
```

- `Make/files` file: Contains paths to the source files for both BC (windkessel and coronary LPN)

```
windkesselOutletPressure/windkesselOutletPressureFvPatchScalarField.C
coronaryOutletPressure/coronaryOutletPressureFvPatchScalarField.C
LIB = $(FOAM_USER_LIBBIN)/liblumpedParameterNetworkBC
```

- Can be compiled using: `wmake libso`
- BC library created within the user library directory i.e., `$FOAM_USER_LIBBIN`: `liblumpedParameterNetworkBC.so`
- Can be added in the `ControlDict` of the OpenFOAM case while using

```
lumpedParameterNetworkBC/
├── {Allwclean, Allwmake}
├── Make/
│   └── {files, options}
├── coronaryOutletPressure/
│   └── coronaryOutletPressureFvPatchScalarField.{C, H}
└── windkesselOutletPressure/
    └── windkesselOutletPressureFvPatchScalarField.{C, H}
```

Test Case: Master Directory Structure

- Contains:
  - All the BC files (Source and Header)
  - A master OpenFOAM test case for both windkessel and coronary BC (Total cases: 11)
  - MATLAB script for postprocessing
  - Bash scripts to create directory structure from the master directory, and delete the already created structure (if required)
  - Bash scripts to run and clean all the cases one by one

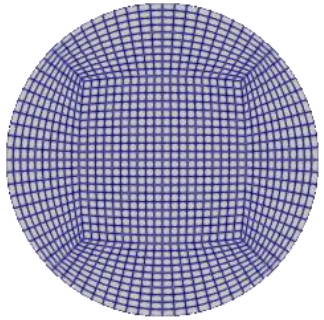
```
LumpedParameterNetworkBCTestCase/
├── Allclean, Allrun, CreateDirStruc, DeleteDirStruc
├── LPNBCTestCase/
│   ├── 0.orig/
│   │   ├── U.{Coronary, WK}
│   │   ├── p.{Resistive}
│   │   └── p.{WK2, WK3, WK4Parallel, WK4Series, Coronary}.{firstOrder,
│   │       secondOrder}
│   ├── Allclean.{Coronary, WK}
│   ├── Allrun.{Coronary, WK}
│   ├── DataFiles/
│   │   └── AorticInletFlowRate, CoronaryInletFlowRate, PimData
│   ├── constant/
│   │   ├── RASProperties, dynamicMeshDict, fluidProperties,
│   │   │   physicsProperties, transportProperties, turbulenceProperties
│   │   └── polyMesh/
│   │       └── blockMeshDict.m4.{Coronary, WK}
│   └── system/
│       └── controlDict, decomposeParDict, fvSchemes, fvSolution
├── MATLABPostprocessing/
│   └── LPNBCTestCasePostprocessing.m
└── lumpedParameterNetworkBC/
    ├── Allwclean, Allwmake
    ├── Make/
    │   └── files, options
    ├── coronaryOutletPressure/
    │   └── coronaryOutletPressureFvPatchScalarField.{C, H}
    └── windkesselOutletPressure/
        └── windkesselOutletPressureFvPatchScalarField.{C, H}
```

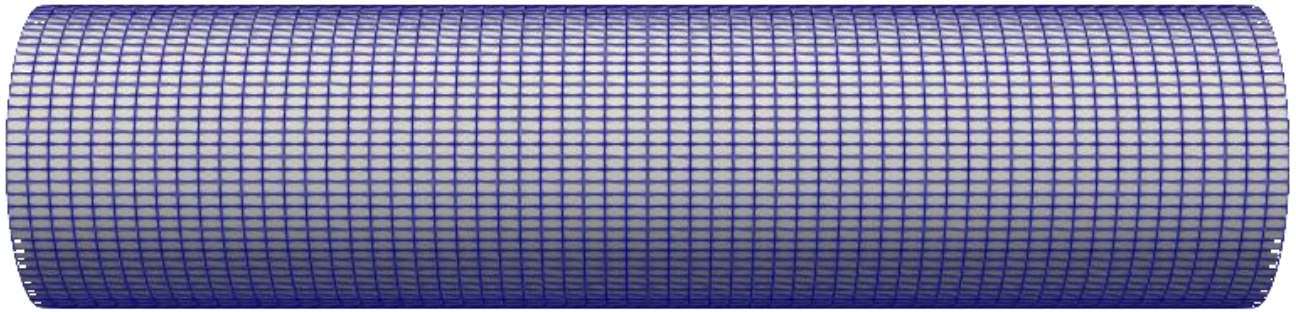
Test Case: solids4foam Case Directory Structure

```
LPNBCTestCase_Coronary_secondOrder/
├── 0/ {U, p}
├── {Allclean, Allrun}
├── DataFiles/ {CoronaryInletFlowRate, PimData}
├── constant/
│   ├── {RASProperties, dynamicMeshDict, fluidProperties, physicsProperties,
│   │   transportProperties, turbulenceProperties}
│   └── polyMesh/ {blockMeshDict.m4}
└── system/
    └── {controlDict, decomposeParDict, fvSchemes, fvSolution}
```

Test Case: Geometry

- Generated using `.m4` script
- For windkessel BCs: A simple cylindrical representation of aorta ($D = 30\ mm$, $L = 4D$)
- For coronary LPN BCs: A simple cylindrical representation of LCA ($D = 3\ mm$, $L = 4D$)
- Mesh:
  - Hexahedra cells: 72000
  - Maximum aspect ratio: 7.05029367
  - Maximum non-orthogonality: 29.075478
  - Maximum skewness: 1.01487831
  - Difference only in the scale or size of the domain for aorta and LCA





```
convertToMeters 0.001;

// Inner square side half
define(s, 7.50)

// Inner square side curvature
define(sc, 8)

// cylinder radius
define(r, 15)

// Height of cylinder
define(z, 120.0)

// Base z
define(Zb, 0)

// Outlet z
define(Zt, calc(Zb + z))

// Number of cells at inner square
define(Ns, 20)

// Number of cells between inner square and circle
define(Ni, 10)

// Number of cells in the cylinder height
define(Nz, 60)
```

Test Case: Simulation Setup

- Physics of the simulation (`constant` Directory)
  - Type: Fluid (`constant/physicsProperties`)
  - Mesh dynamics: static (`constant/physicsProperties`)
  - Material: Blood [Newtonian, $\rho = 1060\ kg/m^3$, $\nu = 3.77 \times 10^{-6}\ \mathrm{m}^2/\mathrm{s}$] (`constant/transportProperties`)
  - Turbulence properties: `RASModel` (`constant/turbulenceProperties`)
  - RAS Properties: Laminar (`constant/RASProperties`)
  - Fluid model: `pimpleFluid` (`constant/fluidProperties`)
- Discretization schemes (`system/fvSchemes` Dictionary)
  - Time: `Euler`
  - Gradient: `leastSquares`
  - Divergence: `Gauss linear`
  - Laplacian: `Gauss linear corrected`
  - Interpolation: `linear`
  - Surface-normal gradient: `skewCorrected 0.5`
- Solution algorithm (`system/fvSolution` Dictionary)
  - U solver: PCG with DIC preconditioner
  - p solver: PBiCG with DILU preconditioner
  - Algorithm: PIMPLE

```
47      nOuterCorrectors            1;
48      nCorrectors                 2;
49      nNonOrthogonalCorrectors    1;
50
51      residualControl
52      {
53          U
54          {
55              relTol      0;
56              tolerance   1e-7;
57          }
58          p
59          {
60              relTol      0;
61              tolerance   1e-7;
62          }
63      }
```

Test Case: Simulation Setup

- Physics of the simulation (`constant` Directory)
  - Type: Fluid (`constant/physicsProperties`)
  - Mesh dynamics: static (`constant/physicsProperties`)
  - Material: Blood [Newtonian, $\rho = 1060\ kg/m^3$, $\nu = 3.77 \times 10^{-6}\ \mathrm{m}^2/\mathrm{s}$] (`constant/transportProperties`)
  - Turbulence properties: `RASModel` (`constant/turbulenceProperties`)
  - RAS Properties: Laminar (`constant/RASProperties`)
  - Fluid model: `pimpleFluid` (`constant/fluidProperties`)
- Discretization schemes (`system/fvSchemes` Dictionary)
  - Time: `Euler`
  - Gradient: `leastSquares`
  - Divergence: `Gauss linear`
  - Laplacian: `Gauss linear corrected`
  - Interpolation: `linear`
  - Surface-normal gradient: `skewCorrected 0.5`
- Decomposition (`system/decomposeParDict` Dictionary)

```
17 numberOfSubdomains       8;
18
19 method                    simple;
20
21 simpleCoeffs
22 {
23     n           (1 1 8);
24     delta       0.001;
25 }
```

- Solution algorithm (`system/fvSolution` Dictionary)
  - U solver: PCG with DIC preconditioner
  - p solver: PBiCG with DILU preconditioner
  - Algorithm: PIMPLE

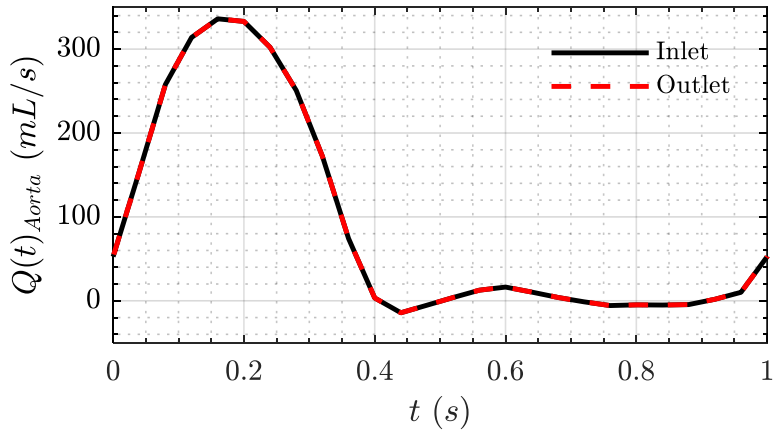
```
47     nOuterCorrectors            1;
48     nCorrectors                 2;
49     nNonOrthogonalCorrectors    1;
50
51     residualControl
52     {
53         U
54         {
55             relTol      0;
56             tolerance   1e-7;
57         }
58         p
59         {
60             relTol      0;
61             tolerance   1e-7;
62         }
63     }
```

Test Case: Inlet Flow Rate BC

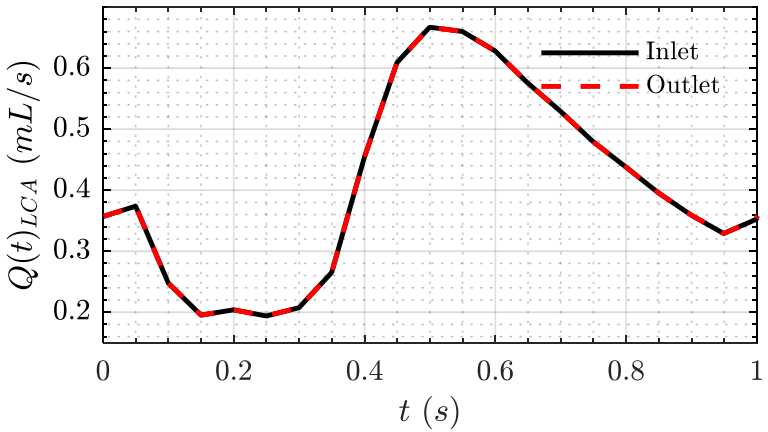
- `0/U` Dictionary:
  - Walls: No-slip
  - Outlet: Zero-gradient
  - Inlet: Time-varying inlet flow rate from file

```
    inlet
    {
        type            timeVaryingFlowRateInletVelocity;
        flowRate        0;              // Volumetric/mass flow rate [m3/s or kg/s]
        value           uniform (0 0 0); // placeholder
        "file|fileName" "$FOAM_CASE/DataFiles/AorticInletFlowRate";
        outOfBounds     repeat;         // (error|warn|clamp|repeat)
    }
```

```
(
( 0.00000 5.25792000E-05 )
( 0.00100 5.51130000E-05 )
( 0.00200 5.76468200E-05 )
( 0.00300 6.01806300E-05 )
.
.
.
( 0.99800 5.04552600E-05 )
( 0.99900 5.15172200E-05 )
( 1.00000 5.25792000E-05 )
);
```




```
    inlet
    {
        type            timeVaryingFlowRateInletVelocity;
        flowRate        0;              // Volumetric/mass flow rate [m3/s or kg/s]
        value           uniform (0 0 0); // placeholder
        "file|fileName" "$FOAM_CASE/DataFiles/CoronaryInletFlowRate";
        outOfBounds     repeat;         // (error|warn|clamp|repeat)
    }
```




```
(
( 0.00000 3.56788000E-07 )
( 0.00100 3.57129822E-07 )
( 0.00200 3.57471644E-07 )
( 0.00300 3.57813465E-07 )
.
.
.
( 0.99710 3.51495783E-07 )
( 0.99810 3.51964391E-07 )
( 0.99910 3.52433000E-07 )
);
```

Test Case: Outlet Pressure BC

- `0/p` Dictionary (Walls: Zero-gradient, Inlet: Zero-gradient, Outlet: windkessel model)

```
    outlet
    {
        type                windkesselOutletPressure;
        windkesselModel     Resistive;
        Rd                  17690000;   //Distal resistance in [kgm^-4s^-1]
        Pd                  9.7363e+03; // Distal pressure in [Pa]
        value               uniform 0;
    }
```

```
    outlet
    {
        type                windkesselOutletPressure;
        windkesselModel     WK2;
        diffScheme          firstOrder;
        Rd                  141520000;  // Distal resistance in [kgm^-4s^-1]
        C                   8.3333e-09; // Compliance in [m^4s^2kg^-1]
        Pd                  0;          // Distal pressure in [Pa]
        value               uniform 0;
    }
```

```
    outlet
    {
        type                windkesselOutletPressure;
        windkesselModel     WK3;
        diffScheme          firstOrder;
        Rp                  13997000;   // Proximal resistance in [kgm^-4s^-1]
        Rd                  141520000;  // Distal resistance in [kgm^-4s^-1]
        C                   1.0000e-08; // Compliance in [m^4s^2kg^-1]
        Pd                  0;          // Distal pressure in [Pa]
        value               uniform 0;
    }
```

```
    outlet
    {
        type                windkesselOutletPressure;
        windkesselModel     WK4Series;
        diffScheme          firstOrder;
        Rp                  13997000;   // Proximal resistance in [kgm^-4s^-1]
        Rd                  141520000;  // Distal resistance in [kgm^-4s^-1]
        C                   2.5000e-08; // Compliance in [m^4s^2kg^-1]
        L                   1.7995e+04; // Blood inertance in [kgm^-4]
        Pd                  0;          // Distal pressure in [Pa]
        value               uniform 0;
    }
```

```
    outlet
    {
        type                windkesselOutletPressure;
        windkesselModel     WK4Parallel;
        diffScheme          firstOrder;
        Rp                  1.3997e+10; // Proximal resistance in [kgm^-4s^-1]
        Rd                  141520000;  // Distal resistance in [kgm^-4s^-1]
        C                   1.0000e-08; // Compliance in [m^4s^2kg^-1]
        L                   1.7995e+03; // Blood inertance in [kgm^-4]
        Pd                  0;          // Distal pressure in [Pa]
        value               uniform 0;
    }
```

Test Case: Outlet Pressure BC (Cont'd…)

- `0/p` Dictionary (Walls: Zero-gradient, Inlet: Zero-gradient, Outlet: Coronary LPN model)

```
28      outlet
29      {
30          type            coronaryOutletPressure;
31          Ra              1.0179e+10;     // Arterial resistance [kg m^-4 s^-1]
32          Ram             1.6541e+10;     // Micro-arterial resistance [kg m^-4 s^-1]
33          Rv              5.0896e+09;     // Veinous resistance [kg m^-4 s^-1]
34          Rvm             0;              // Micro-veinous resistance [kg m^-4 s^-1]
35          Ca              8.6482e-12;     // Arterial compliance [m^4 s^2 kg^-1]
36          Cim             6.9972e-11;     // Intramyocardial compliance [m^4 s^2 kg^-1]
37          PimScaling      1.5;            // Intramyocardial pressure scaling (1.5 for LCA, 0.5 for RCA)
38          Pv              0;              // Distal pressure [Pa]
39          diffScheme      firstOrder;
40          PimFile         "$FOAM_CASE/DataFiles/PimData"; // File containing intramyocardial pressure
        data
41          value           uniform 0; // Initial value for pressure [Pa]
42      }
```

```
(
( 0.00000 2.85419827E+03 )
( 0.00100 2.88803679E+03 )
( 0.00200 2.92568009E+03 )
( 0.00300 2.96726253E+03 )
.
.
.
( 0.99710 2.78917846E+03 )
( 0.99810 2.81575383E+03 )
( 0.99910 2.81578041E+03 )
);
```

Test Case: Simulation Control

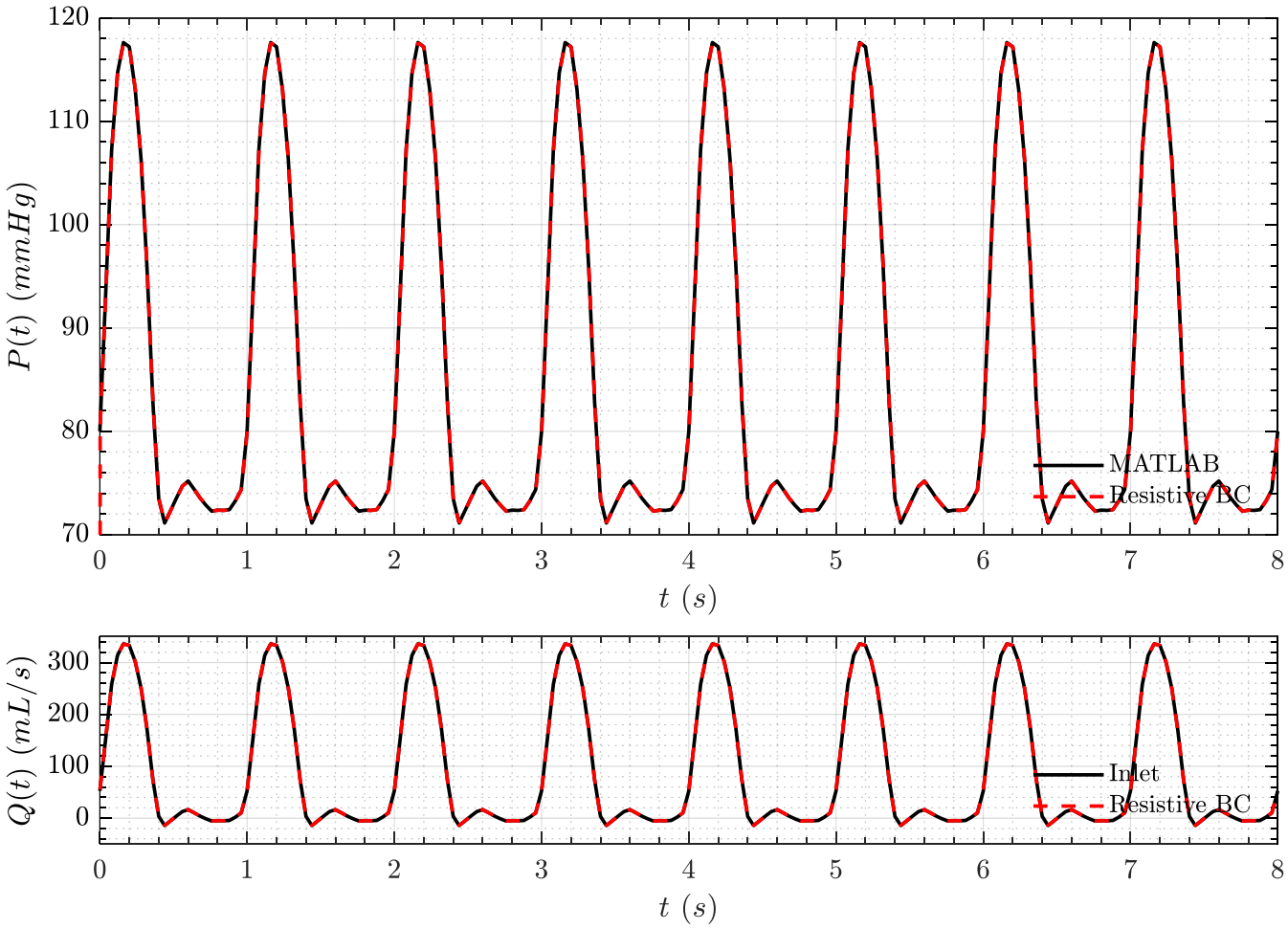
- `system/controlDict` Dictionary

```
libs                ("liblumpedParameterNetworkBC.so");

application         solids4Foam;

startFrom           latestTime;

startTime           0;

stopAt              endTime;

endTime             8;

deltaT              0.001;

writeControl        timeStep;

writeInterval       50;

purgeWrite          0;

writeFormat         ascii;

writePrecision      9;

writeCompression    uncompressed;

timeFormat          general;

timePrecision       6;

runTimeModifiable   yes;

adjustTimeStep      no;

maxCo               0.5;
```

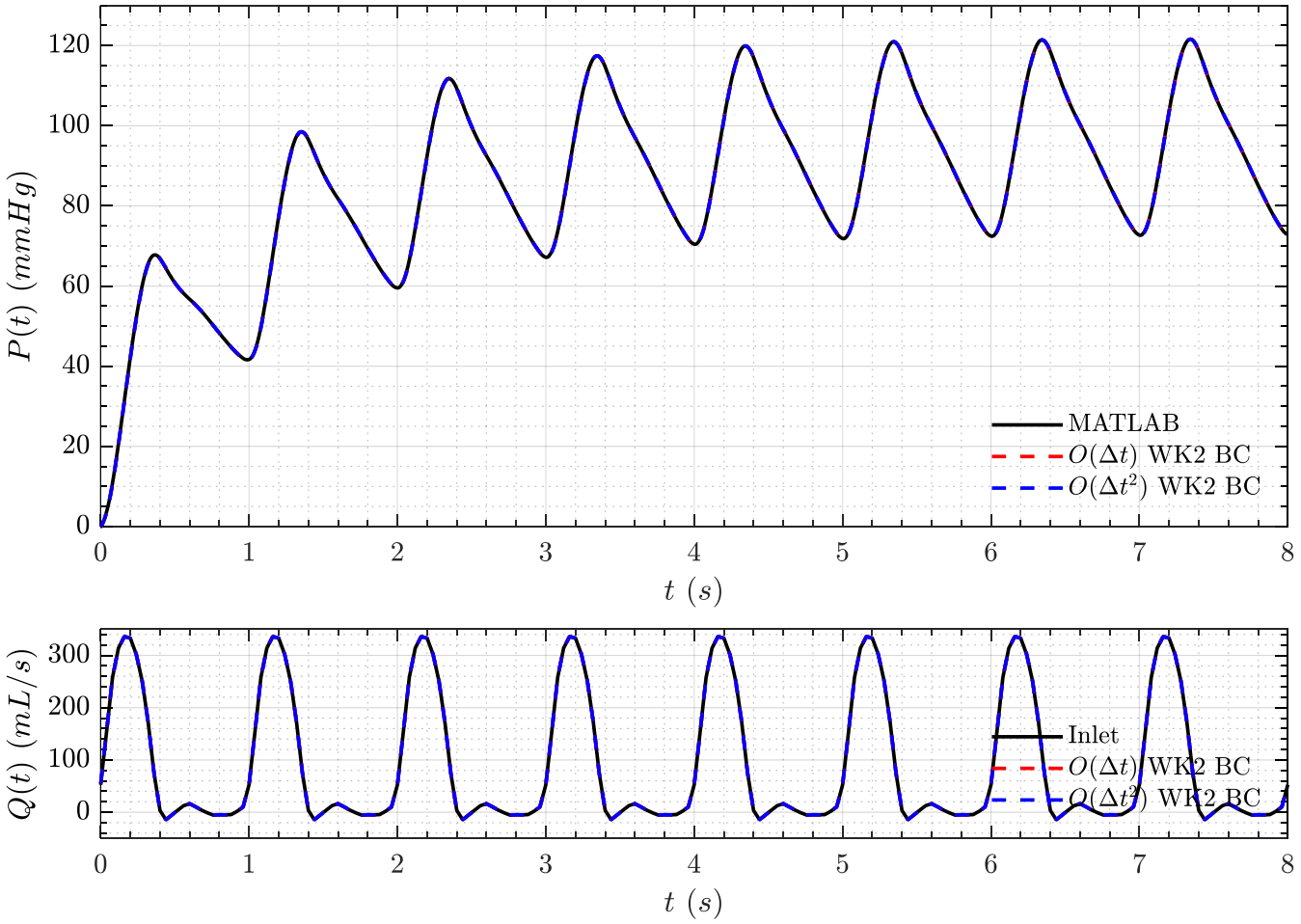
```
    flowRateInlet
    {
        type            faceSource;
        functionObjectLibs ("libfieldFunctionObjects.so");
        enabled         true;
        outputControl   timeStep;
        log             true;
        valueOutput     true;
        source          patch;
        sourceName      inlet;
        operation       sum;
        surfaceFormat   off;

        fields
        (
            phi
        );
    }

    flowRateOutlet
    {
        type            faceSource;
        functionObjectLibs ("libfieldFunctionObjects.so");
        enabled         true;
        outputControl   timeStep;
        log             true;
        valueOutput     true;
        source          patch;
        sourceName      outlet;
        operation       sum;
        surfaceFormat   off;

        fields
        (
            phi
        );
    }
```

```
    pAverageInlet
      {
        type            faceSource;
        functionObjectLibs ("libfieldFunctionObjects.so");
        enabled         true;
        outputControl   timeStep;
        log             true;
        valueOutput     true;
        source          patch;
        sourceName      inlet;
        operation       areaAverage;
       surfaceFormat   off;
       fields
       (
           p
       );
      }

    pAverageOutlet
      {
        type            faceSource;
        functionObjectLibs ("libfieldFunctionObjects.so");
        enabled         true;
        outputControl   timeStep;
        log             true;
        valueOutput     true;
        source          patch;
        sourceName      outlet;
        operation       areaAverage;
       surfaceFormat   off;
       fields
       (
           p
       );
      }
```

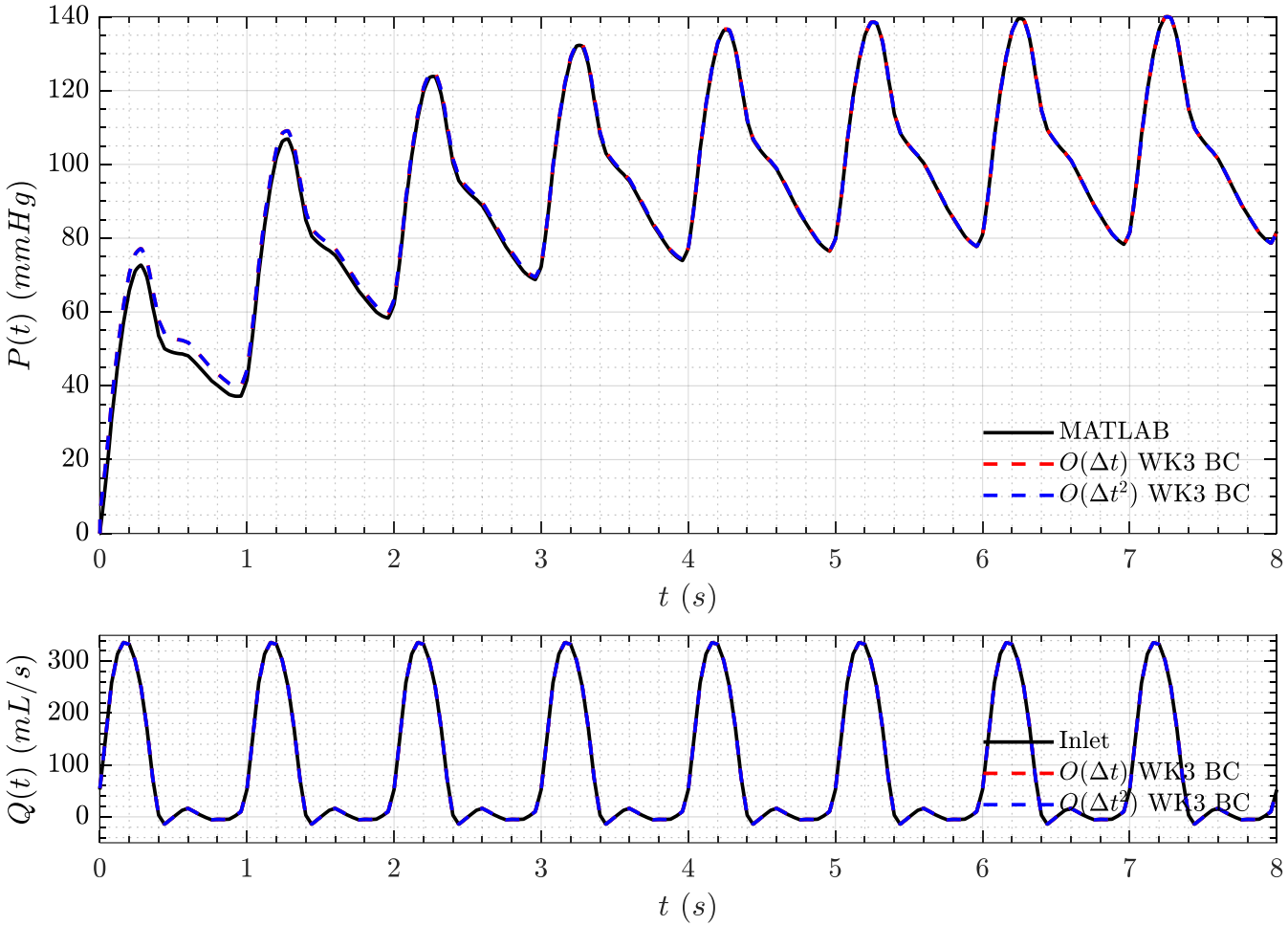
Test Case: Results (Resistive)



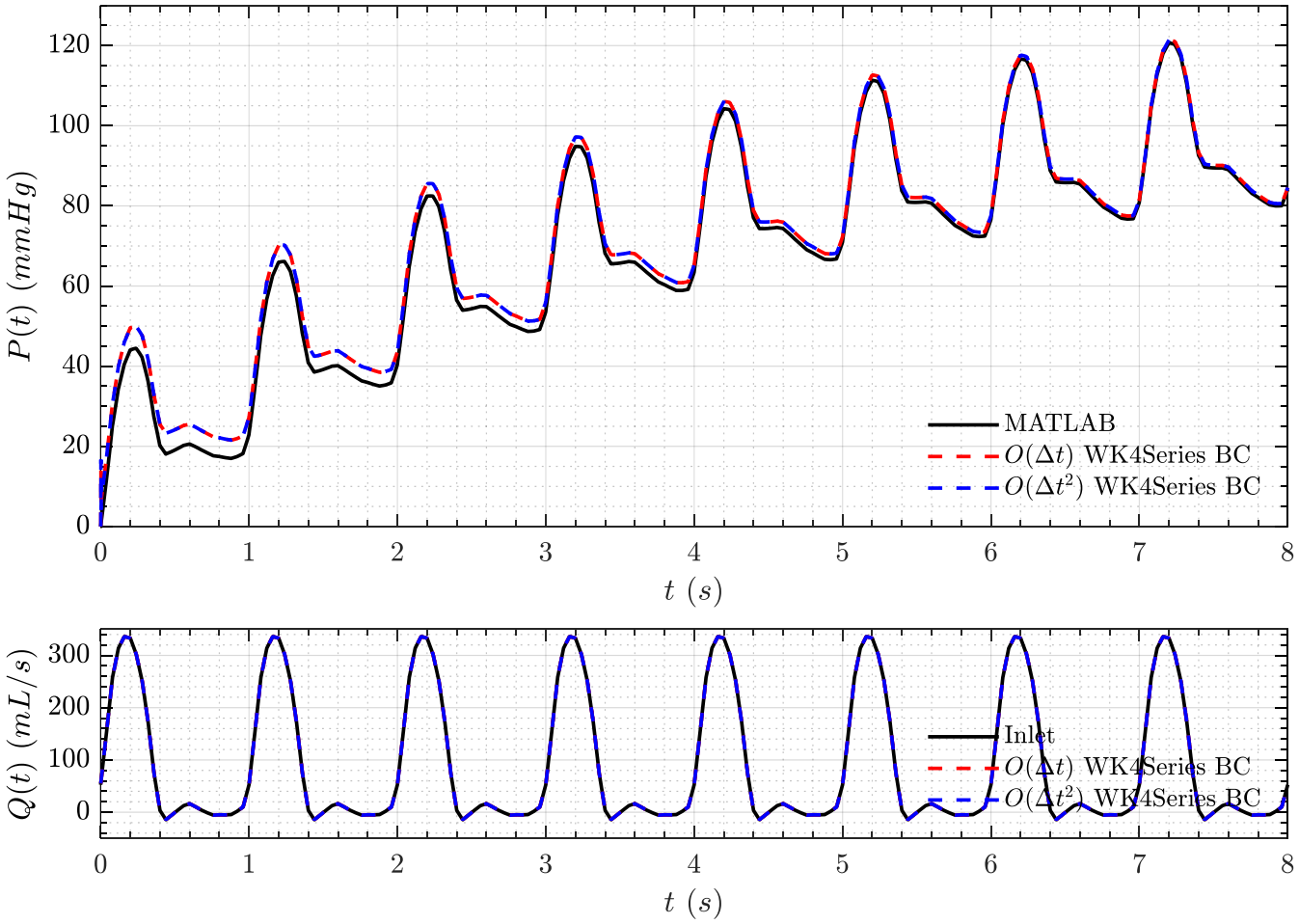
Test Case: Results (WK2)



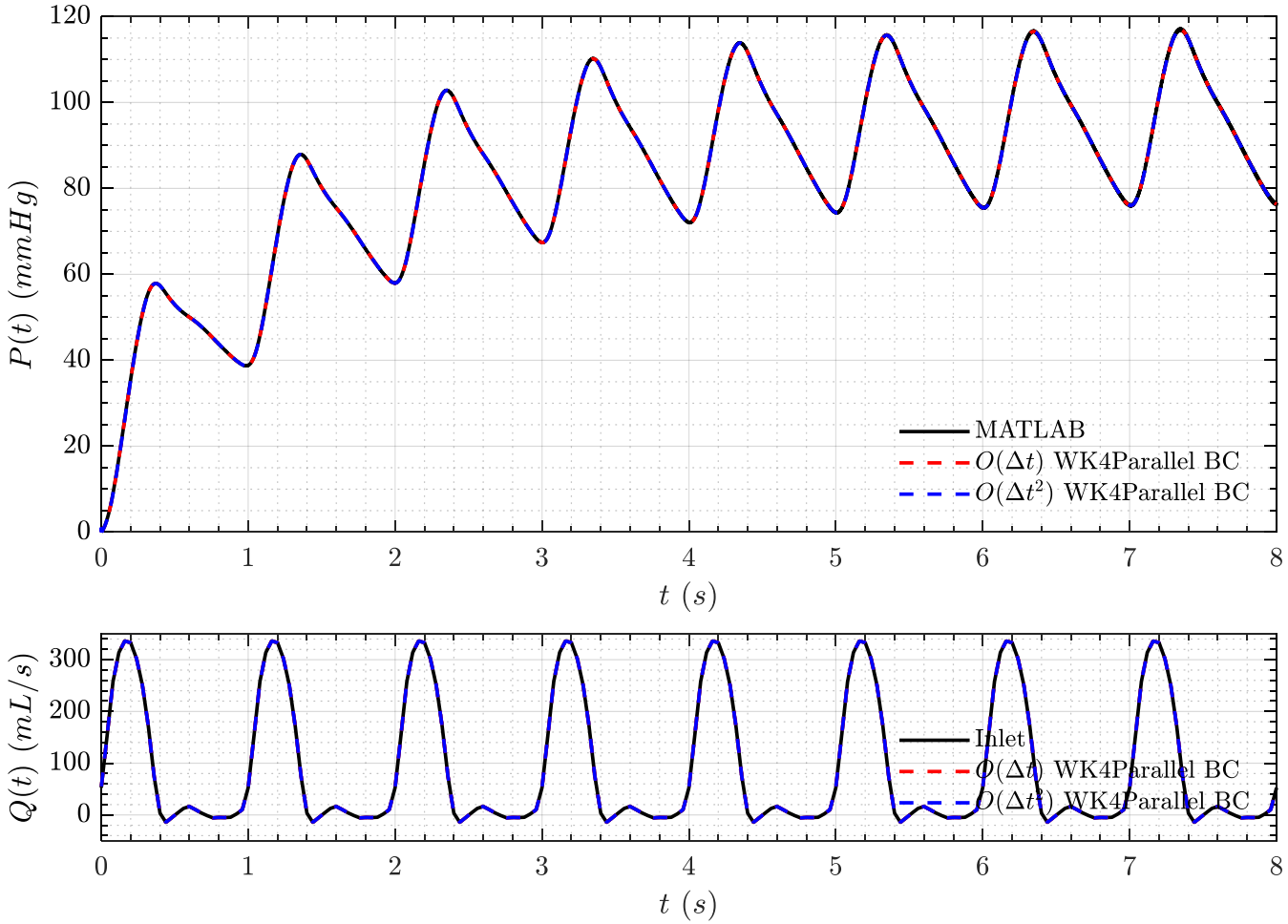
Test Case: Results (WK3)



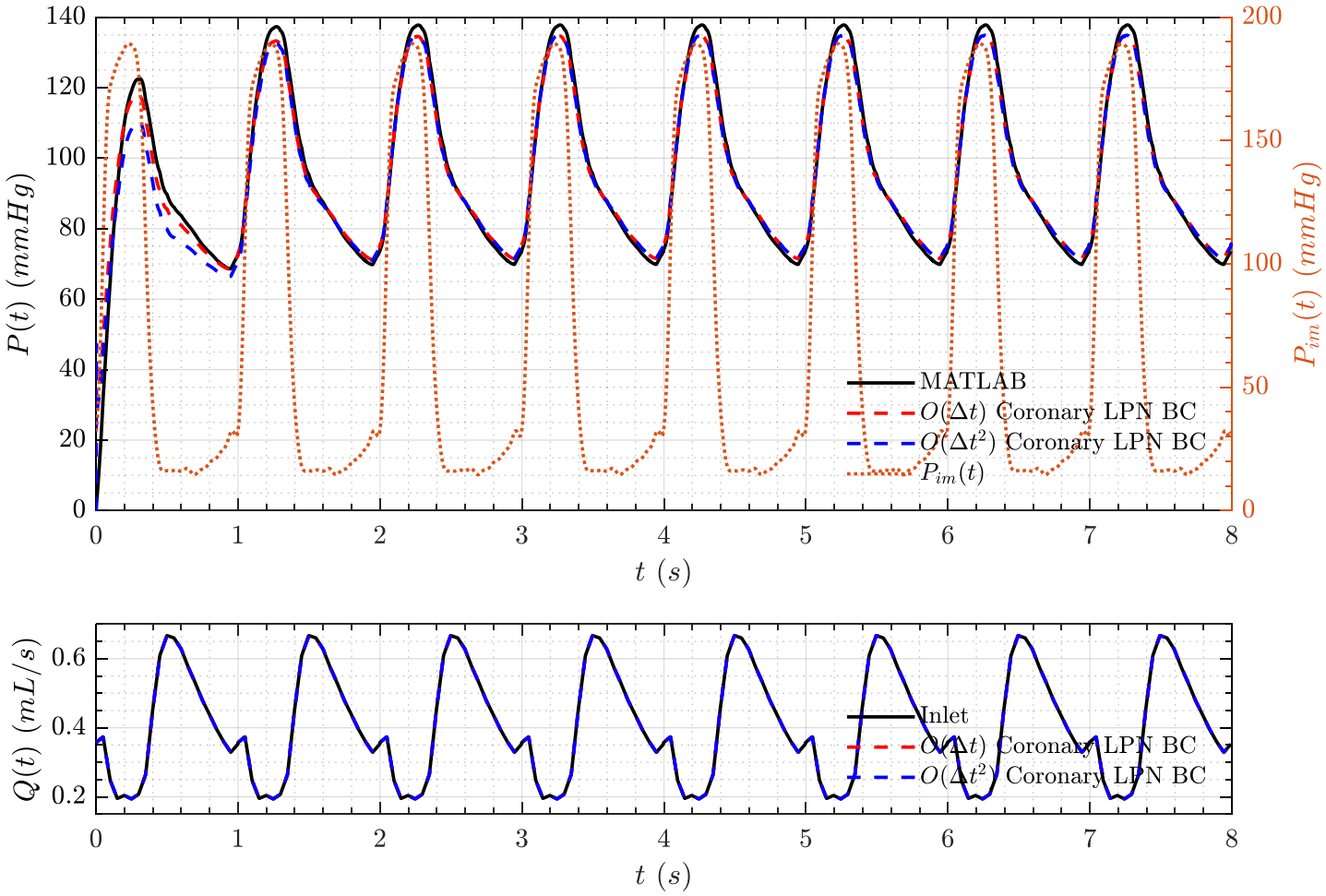
Test Case: Results (WK4Series)



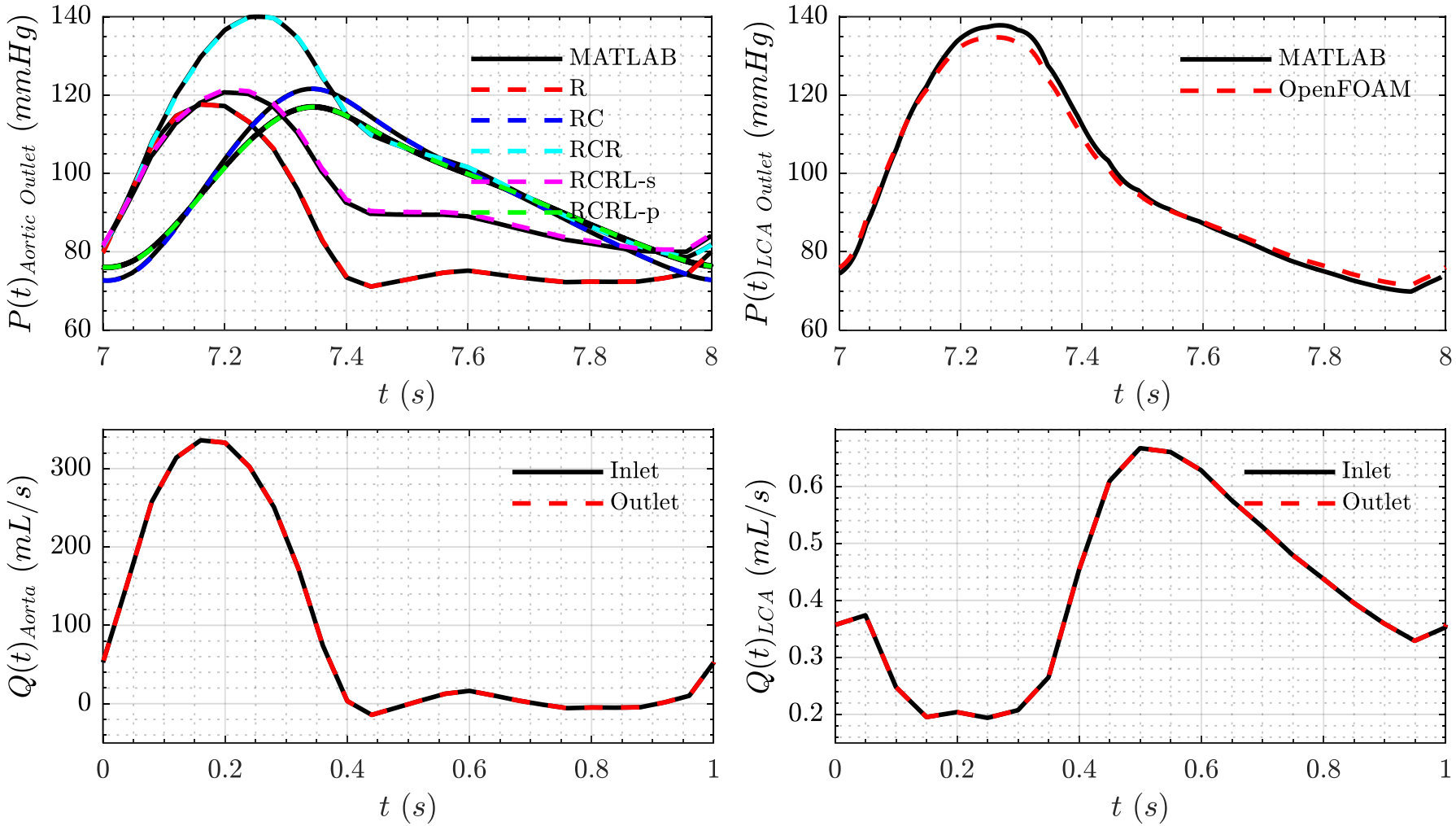
Test Case: Results (WK4Parallel)



Test Case: Results (Coronary LPN)



Test Case: Results (Comparison)



Conclusion

- The complexities of coronary hemodynamics, including the distinct systolic and diastolic flow phases and the influence of intramyocardial pressure, have been effectively addressed through the implementation of specialized LPN models
- Windkessel models, encompassing various configurations of resistive, capacitive, and inductive elements, were derived to simulate the unique physiological characteristics of blood flow in larger blood vessels, known as windkessel vessels such as aorta
- The coronary LPN model has been distinguished from traditional windkessel models by incorporating elements that account for intramyocardial compliance and the time-varying pressure experienced during the cardiac cycle
- This differentiation has enabled the simulation of out-of-phase flow and pressure relationships that are characteristic of coronary circulation
- The numerical implementation has been validated against standard solution of ODEs describing respective LPNs obtained through MATLAB, demonstrating a high degree of consistency and accuracy
- The backward finite differencing schemes employed for temporal discretization have shown robustness in solving the ordinary differential equations governing the LPN models
- The implementation has been structured to accommodate modularity, enabling future extensions and modifications to the boundary condition models

Future Work

- Patient-specific CFD simulations of a complete coronary artery tree with windkessel BCs applied the aortic outlets and coronary LPN BCs applied at coronary outlets
- Investigating the suitability of developed boundary conditions for the FSI simulations in solids4foam toolbox
- Incorporating a more detailed heart model (For inlet BC and to evaluate intramyocardial pressure)
- Estimating and optimizing LPN model parameters
- Improving the computational efficiency of the implemented method

Note: The code would be under process and updated. The latest updates on the ongoing development and implementation of a framework for the patient-specific CFD and FSI simulations of coronary arteries in OpenFOAM and solids4foam toolbox will be available on the author's GitHub profile (https://github.com/MA-Raza)

Thank You!!!

For further information:

Muhammad Ahmad Raza
IRC PhD Scholar
Cardiovascular Biomechanics Research Laboratory,
School of Mechanical and Materials Engineering,
University College Dublin (UCD), Ireland.
Email: muhammad.a.raza@ucdconnect.ie
LI: https://www.linkedin.com/in/mahmad-raza/
GitHub: https://github.com/MA-Raza

OS CFD 2024

My GitHub

Cite as: Raza, M. A.: Implementation of Lumped Parameter Network Boundary Conditions for the Patient-Specific CFD Simulations of Coronary Arteries in OpenFOAM. In Proceedings of CFD with OpenSource Software, 2024, Edited by Nilsson. H., http://dx.doi.org/10.17196/OS_CFD#YEAR_2024

The presented research is funded by the Irish Research Council and AZ Delta VZW under Enterprise Partnership Scheme Postgraduate Scholarship (Project ID: EPSPG/2023/429)