Journal of
open research software

Diem, A K and Bressloff, N W 2017 VaMpy: A Python Package to Solve 1D Blood Flow Problems. *Journal of Open Research Software,* 5: 17, DOI: https://doi.org/10.5334/jors.159

SOFTWARE METAPAPER

VaMpy: A Python Package to Solve 1D Blood Flow Problems

Alexandra K. Diem[1,2] and Neil W. Bressloff[2]

[1] Institute for Complex Systems Simulation, University of Southampton, UK

[2] Computational Engineering and Design, Faculty of Engineering and the Environment, University of Southampton, UK

Corresponding author: Alexandra K. Diem (A.K.Diem@soton.ac.uk)

Finite-differences methods such as the Lax-Wendroff method (LW) are commonly used to solve 1D models of blood flow. These models solve for blood flow and lumen area and are useful in disease research, such as hypertension and atherosclerosis, where flow and pressure are good indicators for the presence of disease. Despite the popularity of the LW method to solve the blood flow equations, no implementation of a LW solver for these equations has been published and made publicly available. This leads to the reimplementation of the same methods within different research groups and makes verification of results more difficult. The Vascular Modelling in Python (VaMpy) toolkit is a Python package that aims to fill this gap. It implements Richtmyer's two-step Lax-Wendroff scheme to solve 1D model equations of blood flow in arterial trees and aims at facilitating the solution of blood flow problems for various medical applications.

Keywords: Python; blood flow; arterial tree; hemodynamics; finite differences; partial differential equations

Funding statement: The development of this software was supported by an EPSRC Doctoral Training Centre grant (EP/G03690X/1).

(1) Overview

Introduction

One-dimensional (1D) modelling of the cardiovascular system is useful in predicting and understanding the dynamics of blood pressure propagation [1, 2, 3, 4, 5, 6]. Here, arteries are regarded as 1D axisymmetric tubes that are described by flux q inside the lumen and cross-sectional area A of the vessel lumen along the vessel length. One popular finite-differences method to numerically solve the equations governing blood flow through arteries is Richtmyer's two-step Lax-Wendroff method [7, 8], which has been used by a number of groups [1, 4, 6, 9, 10, 11]. Alternative methods of solving the blood flow equations include for example variations of the Galerkin finite-element method, which instead solve the blood flow equations for flow velocity u and cross-sectional area A [2, 12].

The computational implementation of the Lax-Wendroff method is straightforward and previously mentioned references have produced results that are validated against experimental results, justifying the popularity of the method. However, no openly available implementation of the Lax-Wendroff method could be found, which results in the same work being carried out numerous times. Whilst one open-source Python package implementing a haemodynamic model exists, pyNS focusses on the implementation of a 0D pulse wave propagation model, representing arteries as electrical circuits [13], and therefore its scope and application are different from VaMpy. Solutions computed using VaMpy are exported to the commonly used CSV file format, thereby allowing for the integration of data with most other software. For example, solutions calculated using VaMpy could be used as a boundary condition for higher order models of larger arteries further upstream.

Arteries are considered to be elastic axisymmetrical tubes of initial radius $r_0(z)$ in a cylindrical coordinate system. The radius at rest is allowed to taper exponentially for an arterial segment if different values are given for the upstream radius R_u and downstream radius R_d. An example geometry for the bifurcation of the common carotid artery, which is used to validate the solution calculated by VaMpy is shown in **Figure 1**. Then the vessel radius for an arterial segment of length L is

$$r_0(z) = R_u \cdot \exp\left(\log\left(\frac{R_d}{R_u}\right)\frac{z}{L}\right). \quad (1)$$

Blood flow through arteries is governed by the Navier-Stokes equations for conservation of mass (continuity equation) and momentum in a 1D cylindrical coordinate system

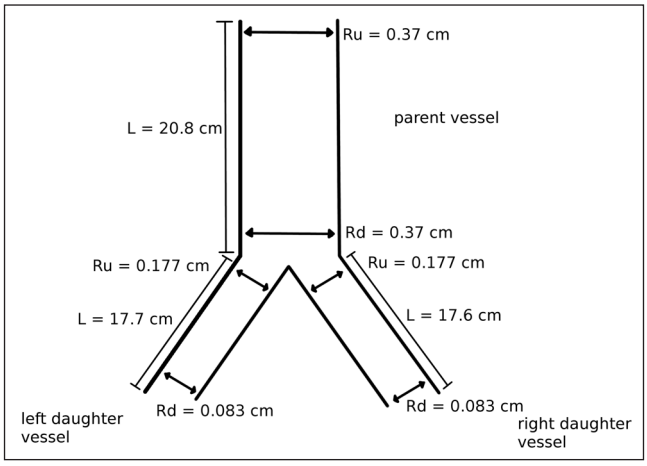

Figure 1: Example geometry of a bifurcation implemented in VaMpy. The example represents the common carotid artery (parent vessel) and its two daughter vessels, which are used for validation purposes of the software. Artery segments have an upstream and downstream radius, where the downstream radius has to be equal to or smaller than the upstream radius. The radius of the vessel then is $r_0(z) = R_u \cdot \exp(\log(R_d/R_u)\, z/L)$.

$$\frac{\partial u_z(r,\,z,\,t)}{\partial z} + \frac{1}{r}\frac{\partial\left(r u_r(r,\,z,\,t)\right)}{\partial t} = 0 \tag{2}$$

$$\begin{aligned}&\frac{\partial u_z(r,\,z,\,t)}{\partial t} + u_z(r,\,z,\,t)\frac{\partial u_z(r,\,z,\,t)}{\partial z}\\ &+ u_r(r,\,z,\,t)\frac{\partial u_z(r,\,z,\,t)}{\partial r} + \frac{1}{\rho}\frac{\partial p(z,\,t)}{\partial z}\\ &= \frac{\nu}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u_z(r,\,z,\,t)}{\partial r}\right),\end{aligned} \tag{3}$$

where $\boldsymbol{u} = (u_z(r, z, t), u_r(r, z, t))$ denotes blood flow velocity, $p(z, t)$ denotes blood pressure, which is assumed to be uniform across r and the parameters ρ and ν denote blood density and viscosity, respectively. By integration of the governing equations over cross-sectional area $A(z, t) = \pi R(z, t)^2$ the 1D conservation law

$$\frac{\partial \boldsymbol{U}}{\partial t} + \frac{\partial \boldsymbol{F}}{\partial z} = \boldsymbol{S} \tag{4}$$

with

$$\boldsymbol{U} = \begin{pmatrix} A(z,\,t) \\ q(z,\,t) \end{pmatrix},\quad \boldsymbol{F} = \begin{pmatrix} q(z,\,t) \\ \dfrac{q(z,\,t)^2}{A(z,\,t)} + f(r_0)\sqrt{A_0(z)A(z,\,t)} \end{pmatrix},$$

$$\boldsymbol{S} = \begin{pmatrix} 0 \\ S_1 \end{pmatrix}$$

$$\begin{aligned}S_1 = &-\frac{2\pi R(z,\,t)}{\delta_b \mathrm{Re}}\frac{q(z,\,t)}{A(z,\,t)} +\\ &\left(2\sqrt{A(z,\,t)}\left(\sqrt{\pi} f(r_0) + \sqrt{A_0(z)}\frac{df(r_0)}{dr_0}\right)\right.\\ &\left. - A(z,\,t)\frac{df(r_0)}{dr_0}\right)\frac{dr_0(z)}{dz}\end{aligned}$$

can be derived. Details on the derivation of (4) can be found elsewhere [1, 6]. Here, the unknowns are the vessel cross-sectional area $A(z, t)$ and flux $q(z, t)$. Elasticity of the vessel is described by the quantity $f(r_0)$ with relaxed vessel radius $r_0(z)$, $A_0(z)$ is the relaxed cross-sectional vessel area, $R(z, t)$ the vessel radius, δ_b is the boundary layer thickness and Re is the Reynold's number.

Although these equations have been commonly used by various groups [1, 4, 5, 14], no publicly accessible implementation of the solution to (4) could be found, meaning that each publication from a separate group resulted in the reimplementation of the same or very similar methods and equations. Therefore, the Vascular Modelling in Python toolkit (VaMpy) was developed and published on GitHub[1] with the documentation available on GitHub Pages.[2] Support for the use of VaMpy is mainly available via the Issue Tracker feature on GitHub, but also via contacting the authors.

Implementation and architecture

The VaMpy implementation and architecture are described in this section. VaMpy is object-oriented to allow for an intuitive understanding of its design and to facilitate the addition of new features. The base of the package is the class ArteryNetwork, which defines the arterial tree. The class contains methods that are applied on the entire network of arteries as well as boundary conditions. Each artery within the tree is defined as an object of the class Artery, which contains its own solver instance. The solver itself is implemented in the independent class LaxWendroff that implements the Lax-Wendroff method as described below. This approach allows for the integration of other solvers within the software.

The code was developed in Python 2.7 and implements Richtmyer's two-step version of the Lax-Wendroff method [7, 8], which is second-order accurate in time and space. For a point in time, n, the solution at the next time step $n + 1$ at grid location m is given by

$$\boldsymbol{U}_m^{n+1} = \boldsymbol{U}_m^n - \frac{\Delta t}{\Delta z}\left(\boldsymbol{F}_{m+1/2}^{n+1/2} - \boldsymbol{F}_{m-1/2}^{n+1/2}\right) + \frac{\Delta t}{2}\left(\boldsymbol{S}_{m+1/2}^{n+1/2} + \boldsymbol{S}_{m-1/2}^{n+1/2}\right), \tag{5}$$

where $\boldsymbol{U}_m^n = \boldsymbol{U}(m\Delta z, n\Delta t)$ is the solution at position $m\Delta z$ and time $n\Delta t$. The half time step values for $\boldsymbol{F}$ and $\boldsymbol{S}$ are determined by

$$\boldsymbol{U}_j^{n+1/2} = \frac{\boldsymbol{U}_{j+1/2}^n + \boldsymbol{U}_{j-1/2}^n}{2} + \frac{\Delta t}{2}\left(-\frac{\boldsymbol{F}_{j+1/2}^n - \boldsymbol{F}_{j-1/2}^n}{\Delta z} + \frac{\boldsymbol{S}_{j+1/2}^n + \boldsymbol{S}_{j-1/2}^n}{2}\right) \tag{6}$$

for $j = m \pm 1/2$. An illustration of the computational procedure to determine $\boldsymbol{U}_m^{n+1}$ is shown in **Figure 2**. It illustrates that both initial conditions at $n = 0$ for all m and left and right boundary conditions are required to determine $\boldsymbol{U}$.

Boundary conditions are applied at both ends of the vessel and are either an inlet, outlet or bifurcation condition. The inlet boundary condition is used at the inlet of the parent vessel only [1]. It requires flux values $q(0, t)$ to be prescribed. The inlet area is then calculated according to (5)

$$A_0^{n+1} = A_0^n - \frac{\Delta t}{\Delta z}\left(q_{1/2}^{n+1/2} - q_{-1/2}^{n+1/2}\right). \tag{7}$$

This requires the evaluation of the term $q_{-1/2}^{n+1/2}$, which is estimated from

$$q_0^{n+1/2} = \frac{1}{2}\left(q_{-1/2}^{n+1/2} + q_{1/2}^{n+1/2}\right), \tag{8}$$

where $q_0^{n+1/2}$ is evaluated from the function prescribing flux values at the inlet and $q_{1/2}^{n+1/2}$ is evaluated from (6), see also [1, 4].

Algorithm 1: Iterative scheme to determine $\boldsymbol{U}_M^{n+1}$ using a 3WK boundary condition [4]. An initial guess is made for p_M^{n+1}, from which q_M^{n+1} is calculated using (9) and A_M^{n+1} is calculated using the discretized mass conservation equation (12). Using A_M^{n+1} the next iteration of p_M^{n+1} is found via the state equation (11). The algorithm stops after $k_{\max}$ iterations or when the difference between pressure estimates is less than the small threshold value ϵ.

$p_M^{n+1} = p_m^n$ # *initial guess for* p_M^{n+1}
$k = 0$
for $k \le k_{\max}$:
  $p_{\text{old}} = p_M^n$
  $q_M^{n+1} = q_M^n + \frac{p_M^{n+1} - p_M^n}{R_1} + \frac{\Delta t p_M^n}{R_1 R_2 C_T} - \frac{\Delta t q_M^n (R_1 + R_2)}{R_1 R_2 C_T}$
  $A_M^{n+1} = A_0^n - \frac{\Delta t}{\Delta z}(q_M^{n+1} - q_{M-1}^{n+1})$
  $p_M^{n+1} = f_M^{n+1}\left(1 - \sqrt{\frac{(A_0)_M}{A_M^{n+1}}}\right)$
  if $| p_{\text{old}} - p_M^{n+1} | \le \epsilon$:
    break
  $k = k + 1$

The outlet boundary condition is a three-element Windkessel (3WK) and its implementation follows [4]. The 3WK equation is

$$\frac{\partial p(z,\, t)}{\partial t} = R_1 \frac{\partial q(z,\, t)}{\partial t} - \frac{p(z,\, t)}{R_2 C_T} + \frac{q(z,\, t)(R_1 + R_2)}{R_2 C_T}, \tag{9}$$

where R_1, R_2 and C_T are resistance and compliance parameters. Discretization of (9) leads to

$$\frac{p_M^{n+1} - p_M^n}{\Delta t} = R_1 \frac{q_M^{n+1} - q_M^n}{\Delta t} - \frac{p_M^n}{R_2 C_T} + \frac{q_M^n (R_1 + R_2)}{R_2 C_T}, \tag{10}$$

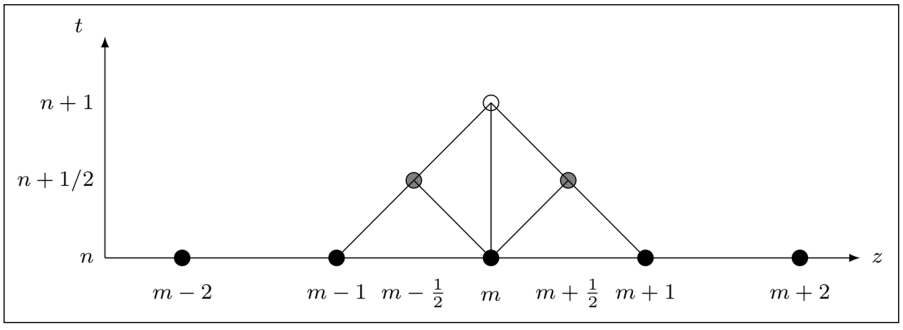

Figure 2: Illustration of the LW method. The solution is fully known at time step n (black circles) and we are looking for the solution at grid point m at time step $n + 1$ (white circle). To determine the unknown solution, two intermediate solutions at half grid points $m \pm 1/2$ and at half time step $n + 1/2$ are determined from grid points $m - 1$, m and $m + 1$ at current time step n. The intermediate solutions are then used in conjunction with the known solution at grid point m and current time step n to calculate the unknown solution at grid point m and next time step $n + 1$ [4].

where M is the spatial position of the outlet. The 3WK boundary condition requires the evaluation of pressure in the vessel, which is related to area via the discretized State Equation [1]

$$p_M^{n+1} = f_M^{n+1}\left(1 - \sqrt{\frac{(A_0)_M}{A_M^{n+1}}}\right). \tag{11}$$

The outlet boundary condition is solved using an iterative scheme with an initial guess for p_M^{n+1} (see Algorithm 1). This requires a discretized version of the mass conservation equation to obtain an estimate for A_M^{n+1}

$$A_M^{n+1} = A_0^n - \frac{\Delta t}{\Delta z}\left(q_M^{n+1} - q_{M-1}^{n+1}\right). \tag{12}$$

Finally, a bifurcation boundary condition applies between any vessel that is not a terminal vessel and its two daughter vessels [1, 4]. Relations between parent and daughter vessels lead to a system of eighteen equations for eighteen unknowns, which are solved using Newton's method according to

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - (\boldsymbol{J}(\boldsymbol{x}_k))^{-1}\boldsymbol{f}_J(\boldsymbol{x}_k) \quad \text{for } k = 0,\ 1,\ 2, \ldots \tag{13}$$

where k indicates the current iteration, = (x_1, x_2, ..., x_{18}), $\boldsymbol{J}$ ($\boldsymbol{x}_k$)) is the Jacobian of the system of equations and $\boldsymbol{f}_J$ ($\boldsymbol{x}_k$) is the vector of residuals. The full system of equations required to solve the boundary conditions at bifurcations can be found elsewhere [6, 15].

Algorithm 2: Setup routine for a network of arteries. The artery network is created as a binary tree and contains $2^{\text{depth}} - 1$ arteries. At each depth level the up- and downstream radii of daughter vessels are calculated using the scaling parameters a and b. Artery objects are then created for each new daughter vessel and stored in a list.

```
pos = 0 # identifier for the next artery
arteries[pos] = Artery (Ru, Rd) # list containing artery
                                # objects
radii_u = [Ru] # list containing upstream radii of arteries
radii_d = [Rd] # list containing downstream radii of
               # arteries

for j in range (depth):
    # lists for storage of upstream/downstream radii
    # of arteries on the next depth level
    new_radii_u = []
    new_radii_d = []

    for i in range (length (radii_u)):
        # set daughter vessel radii using scaling
        # parameters a and b
        ra_u = radii_u[i] * a
        rb_u = radii_u[i] * b
        ra_d = radii_d[i] * a
        rb_d = radii_d[i] * b
        pos + = 1
        # first daughter vessel using scaling
        # parameter a
        arteries[pos] = Artery (ra_u, ra_d)
        pos + = 1
        # second daughter vessel using scaling
        # parameter b
        arteries[pos] = Artery (rb_u, rb_d)

        # store new radii for next iteration over j
        new_radii_u[i] = ra_u
        new_radii_u[i] = rb_u
        new_radii_d[i] = ra_d
        new_radii_d[i] = rb_d
    radii_u = new_radii_u
    radii_d = new_radii_d
```

A simulation model is set up by creating an ArteryNetwork object and solved by executing the following functions

```
from vampy.artery_network import ArteryNetwork
an = ArteryNetwork(Ru, Rd, a, b, lam, k, rho,
nu, p0, depth, ntr, Re)
an.mesh(dx)
an.set_time(dt, T[, tc])
an.initial_conditions(q0)
an.solve(q_in, out_args)
```

A network of arteries is created using the upstream and downstream radii Ru and Rd of the parent vessel, the radius-to-length ratio lam and scaling parameters a and b using Algorithm 2. Two daughter vessels are created for a parent vessel by multiplying their upstream and downstream radii with scaling parameters a and b respectively. This process is repeated until the desired tree depth is reached and the number of arteries in the network is $2^{\text{depth}} - 1$. A second setup routine exists to create an artery network, which is

```
an = ArteryNetwork(Ru, Rd, lam, k, rho, nu, p0,
depth, ntr, Re)
```

Here, Ru, Rd and lam are iterables (for example lists or Numpy arrays) of length depth containing these values for each artery. The remaining parameters required by the ArteryNetwork constructor are the elasticity parameter k, blood density rho, blood viscosity nu, diastolic pressure p0, number of output parameters ntr and Reynold's number Re. The latter method is used by the examples shown in this paper. The spatial discretisation is created by supplying the spatial step size dx, which is used internally to create Numpy arrays for all variables along the vessel.

Timing parameters are the time step size dt, time of one period T and, optionally, the number of periods tc, which defaults to one if left unspecified. Initial conditions are supplied for $q(z, t)$ as a single value q0, while the initial condition for $A(z, t)$ is calculated from the radii at rest. The solve function is supplied with boundary condition parameters q_in and out_args, where q_in contains the values $q(0, t)$ and out_args contains the parameters for a 3WK model.

The solver loops over the simulation time steps and creates a LaxWendroff object for each Artery object

```
lw = LaxWendroff(theta, gamma, artery.nx)
```

with theta = dt/artery.dx, gamma = dt/2 and number of spatial steps artery.nx. The next time step is computed according to (5) and (6) at the inner grid points. Note that the time step size needs to fulfill the Courant-Friedrichs-Lewy (CFL) condition, which in this case is

$$\Delta t \leq \Delta x \cdot \left| \frac{q}{A} \pm c \right|^{-1}, \tag{14}$$

where $c = \sqrt{A/\rho \, \partial p / \partial A}$ is the wave speed. The CFL condition is automatically checked by the ArteryNetwork solver and the simulation stops with an error message if the condition is not met.

The following section demonstrates the use of VaMpy for the simulation of the common carotid artery as done by [4]. This publication was chosen as it provides detailed information on the geometry used and Windkessel parameter for the outlet boundary condition. A detailed walkthrough of how to write configuration and simulation files can be found on the documentation website.[3]

Quality control

The VaMpy Git repository contains unit tests to ensure functions perform as expected. Additionally, the file `bifurcation_example.py` demonstrates VaMpy's performance by validating its results against results in [4] on the common carotid artery bifurcation. Whilst unit tests demonstrate that the functionality of the software meets expecations, validation against experimental results and other researchers' results ensures that the software additionally generates sensible output data and that parameters have been chosen sensibly. Users implementing arteries using other parameters than the ones tested in the example files in VaMpy should therefore always cross-check their results against experimental or other simulation results using the same parameters to ensure that the choice of parameters is realistic.

The solution computed using VaMpy is shown in **Figure 3** and matches the corresponding figures in [4]. Thus this example demonstrates that VaMpy performs as expected.

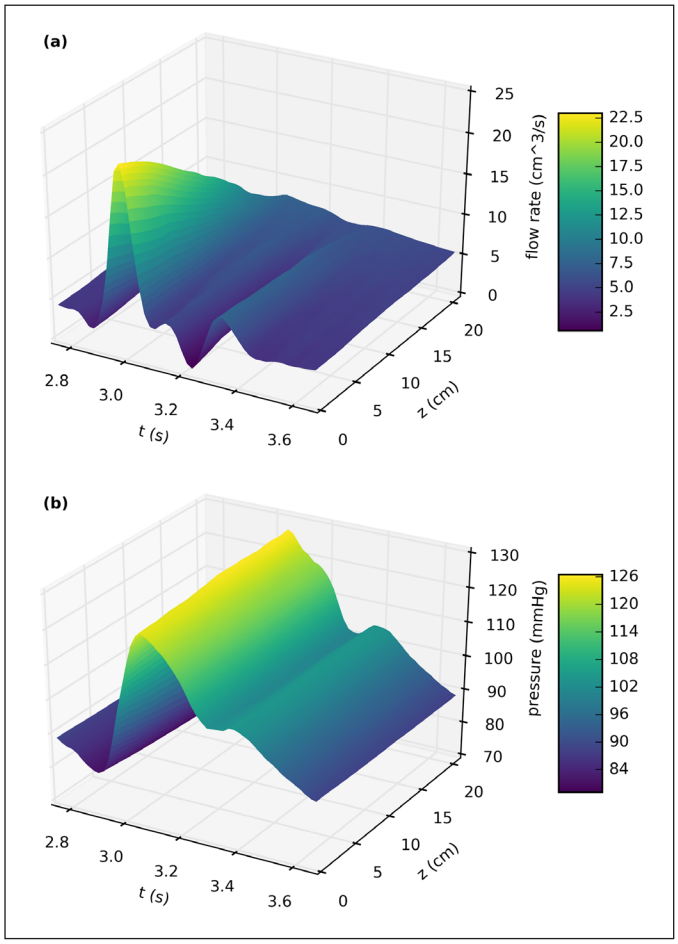

Figure 3: One pulse in the common carotid artery using VaMpy: **a)** flow rate, **b)** pressure. Comparison with the results for the same simulation in [4] validates the implementation of the blood flow equations in VaMpy.

To execute the example run

```
python bifurcation_example.py bifurcation.cfg
```

To plot the data created from the example run

```
python plot_example.py bifurcation.cfg
```

The first version of VaMpy focusses on the simulation of a single bifurcation, i. e. one parent vessel with two daughter vessels. The development of the first version of VaMpy was based on the simulation of flow through the middle cerebral artery in order to evaluate lymphatic drainage through the wall of the artery [6], and for this purpose a single bifurcation was regarded sufficient. Validation on larger networks of arteries with multiple levels of bifurcations has therefore not been carried out yet, but is planned for the next release cycle. Additionally, it is planned to offer a choice of alternative outlet boundary conditions, such as the structured tree [1, 5]. It has been demonstrated that by taking into account bifurcation pressure drops, the accuracy of reduced order models such as the system of equations (4) can improve significantly compared to higher order models [16]. This means that a similar accuracy of blood flow solutions could be achieved for 1D models compared to 2D or 3D models by increasing the depth of the arterial tree to be modelled.

(2) Availability

Operating system

VaMpy is compatible with any operating system that is compatible with Python 2.7 and the dependent packages.

Programming language

VaMpy was written in and for Python 2.7 and above.

Additional system requirements

There are no additional system requirements. However, the requirements for memory and processing power are dependent on the number of the grid points.

Dependencies

NumPy, SciPy, Matplotlib, ConfigParser.

Software location

Archive

Name: GitHub

Persistent identifier: https://github.com/akdiem/vampy/releases/tag/v1.0

Licence: Three-Clause BSD

Publisher: Alexandra K. Diem

Version published: v1.0

Date published: 22/03/2017

Code repository

Name: GitHub

Persistent identifier: https://github.com/akdiem/vampy

Licence: Three-Clause BSD

Date published: 26/04/2016

Language

Python 2.7

(3) Reuse potential

Modelling blood flow dynamics is a useful tool in vascular diseases research and 1D models provide good approximations. The method implemented in VaMpy is used by a variety of research groups [1, 4, 11] and therefore it is expected that the reuse potential for VaMpy is high, especially in multiscale simulations. Because the commonly accepted CSV file format is used for input and output data for VaMpy integration of results from VaMpy simulations with other third-party software packages is expected to be straightforward. For example, VaMpy could be used as a boundary condition for 3D simulations or constitute a part of multi-scale simulations.

The publication of this software additionally provides opportunities for other researchers to add functionality and because VaMpy has been validated on results published in the literature it simplifies and promotes reproducibility of results. The following features are planned for the next releases:

- asymmetric daughter vessel geometries with separate Windkessel parameters,
- validation of the method on bifurcation networks larger than two levels and
- integration of models of the dynamics of the artery wall.

The current release of VaMpy was developed to implement a bifurcation at the middle cerebral artery as part of a multi-scale model of lymphatic flow through the basement membrane embedded in the artery wall, which is relevant for resolving the mechanisms behind the onset and progression of Alzheimer's disease [6, 17].

Notes

[1] https://github.com/akdiem/vampy.

[2] http://akdiem.github.io/vampy/.

[3] http://akdiem.github.io/vampy/walkthrough.html.

Acknowledgements

The authors thank Maximilian Albert for advice and guidance on the use of GitHub repositories and Python code repository conventions.

Competing Interests

The authors have no competing interests to declare.

References

1. **Olufsen, M S,** et al. 2000 "Numerical Simulation and Experimental Validation of Blood Flow in Arteries with Structured-Tree Outflow Condition". In: *Annals of Biomedical Engineering*, 28(11), pp. 1281–1299. URL: http://link.springer.com/article/10.1114/1.1326031. DOI: https://doi.org/10.1114/1.1326031
2. **Sherwin, S J,** et al. 2003 "Computational modelling of 1D blood flow with variable mechanical properties and its application to the simulation of

wave propagation in the human arterial system". In: *International Journal for Numerical Methods In Fluids*, 43(6–7), pp. 673–700. URL: http://onlinelibrary.wiley.com/doi/10.1002/fld.543/abstract. DOI: https://doi.org/10.1002/fld.543

3. **Alastruey, J,** et al. 2007 "Modelling the circle of Willis to assess the effects of anatomical variations and occlusions on cerebral flows". In: *Journal of Biomechanics*, 40(8), pp. 1794–1805. DOI: https://doi.org/10.1016/j.jbiomech.2006.008
4. **Kolachalama, V,** et al. 2007 "Predictive Haemodynamics in a One-Dimensional Carotid Artery Bifurcation. Part I Application to Stent Design". In: *IEEE Transactions on Biomedical Engineering*, 54(5), pp. 802–812. URL: http://ieeexplore.ieee.org/document/4155000/. DOI: https://doi.org/10.1109/TBME.2006.889188
5. **Cousins, W** and **Gremaud, P A** 2014 "Impedance boundary conditions for general transient hemodynamics". In: *International Journal for Numerical Methods in Biomedical Engineering*, 30(11), pp. 1249–1313. URL: http://onlinelibrary.wiley.com/doi/10.1002/cnm.2658/abstract. DOI: https://doi.org/10.1002/cnm.2658
6. **Diem, A K** 2016 "Prediction of Perivascular Drainage of Ab from the Brain Using Computational Modelling: Implications for Alzheimer's Disease". PhD thesis. University of Southampton.
7. **LeVeque, R J** 1992 *Numerical Methods for Conservation Laws*. 2nd. Basel, Switzerland: Birkhäuser Verlag, pp. 122–135. DOI: https://doi.org/10.1007/978-3-0348-8629-1_12
8. **Richtmyer, R D** 1963 "A Survey of Difference Methods for Non-Steady Fluid Dynamics". In: *NCAR Technical Notes*, 63(2). URL: http://opensky.ucar.edu/islandora/object/technotes:49. DOI: https://doi.org/10.5065/D67P8WCQ
9. **Smith, N P, Pullan, A J** and **Hunter, P J** 2002 "An Anatomically Based Model of Transient Coronary Blood Flow in the Heart". In: *SIAM Journal on Applied Mathematics*, 62(3), pp. 990–1018. ISSN: 0036-1399. URL: http://epubs.siam.org/doi/abs/10.1137/S0036139999355199. DOI: https://doi.org/10.1137/S0036139999355199
10. **Azer, K** and **Peskin, C S** 2007 "A One-dimensional Model of Blood Flow in Arteries with Friction and Convection Based on the Womersley Velocity Profile". In: *Cardiovascular Engineering*, 7(2), pp. 51–73. https://link.springer.com/article/10.1007/s10558-007-9031-y. DOI: https://doi.org/10.1007/s10558-007-9031-y
11. **Itu, L M** and **Suciu, C** 2011 "Analysis of outflow boundary condition implementations for 1D blood flow models". In: *Proceedings of the 3rd International Conference on E-Health and Bioengineering*, 4. pp. 24–27. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6150403.
12. **Mynard, J P** and **Nithiarasu, P** 2008 "A 1D arterial blood flow model incorporating ventricular pressure, aortic valve and regional coronary flow using the locally conservative Galerkin (LCG) method". In: *Communications in Numerical Methods in Engineering*, 24. pp. 367–417. ISSN: 20407939. URL: http://onlinelibrary.wiley.com/doi/10.1002/cnm.1117/full. DOI: https://doi.org/10.1002/cnm.1117
13. **Manini, S,** et al. 2015 "pyNS: An Open-Source Framework for 0D Haemodynamic Modelling". In: *Annals of Biomedical Engineering*, 43(6), pp. 1461–1473. ISSN: 15739686. URL: https://link.springer.com/article/10.1007/s10439-014-1234-y. DOI: https://doi.org/10.1007/s10439-014-1234-y
14. **Devault, K,** et al. 2008 "Blood Flow in the Circle of Willis Modeling and Calibration". In: *Multiscale Modeling & Simulation*, 7(2), pp. 888–909. URL: http://epubs.siam.org/doi/abs/10.1137/07070231X. DOI: https://doi.org/10.1137/07070231X
15. **Olufsen, M S** 1998 "Modeling of the Arterial System with Reference to an Anesthesia Simulator". PhD Thesis. University of Roskilde, Denmark. URL: http://rudar.ruc.dk/handle/1800/744.
16. **Chnafa, C,** et al. 2017 "Improved reduced-order modelling of cerebrovascular flow distribution by accounting for arterial bifurcation pressure drops". In: *Journal of Biomechanicsiomechanics*, 51, pp. 83–88. DOI: https://doi.org/10.1016/j.jbiomech.2016.12.004
17. **Bakker, E N T P,** et al. 2016 "Lymphatic Clearance of the Brain Perivascular, Paravascular and Significance for Neurodegenerative Diseases". In: *Cellular and Molecular Neurobiology*, 36(2), pp. 181–194. URL: https://link.springer.com/article/10.1007/s10571-015-0273-8. DOI: https://doi.org/10.1007/s10571-015-0273-8

How to cite this article: Diem, A K and Bressloff, N W 2017 VaMpy: A Python Package to Solve 1D Blood Flow Problems. *Journal of Open Research Software,* 5: 17, DOI: https://doi.org/10.5334/jors.159

Submitted: 13 December 2016 **Accepted:** 18 May 2017 **Published:** 08 June 2017

]u[*Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press

OPEN ACCESS