Biomath Communications

www.biomathforum.org/biomath/index.php/conference

Dynamic simulation and steady-state computation of 3D physiologically structured population models.

Michiel Van Dyck[1], Xavier Woot De Trixhe[2], Wim Vanroose[1]
[1]Dept. of Mathematics and Computer Science, University of Antwerp, Antwerp, Belgium
Michiel.VanDyck@uantwerpen.be
[2]Dept. Modelling, Janssen Pharmaceutica, Beerse, Belgium
xwootdet@its.jnj.com

Abstract

The PSP modelling approach allows you to model biological/pharmaceutical behaviour by combining micro-scale ordinary differential equation (ODE) models with macro-scale ODE models and their bi-directional interaction. E.g.: based on the model of a single cell, billions of cells can be simulated to get the response of an entire organ (also incorporating the organ to cell reaction). The PSP approach allows to simulate this in a mathematically efficient way by characterising different cells by a set of physiologically relevant quantities [2, 3].

Our framework is capable of taking into account up to 3 physiological parameters resulting in a 3D structure for which a partial differential equation (PDE) should be solved. We achieve considerable speed-up by using a semi-Lagrangian PDE solver that allows big stable time stepping. Combining this

Citation: Michiel Van Dyck, Xavier Woot De Trixhe, Wim Vanroose, Dynamic simulation and steady-state computation of 3D physiologically structured population models, http://dx.doi.org/10.11145/bmc.2017.12.237

with a low level C-language implementation, we achieve exceptional efficient usage of the computing processing unit (CPU). Furthermore we accelerate the computation of the steady-state by using a Newton-Krylov method. For ease of use, a PSP description language is constructed to allow straightforward input of different models. The simulator should empower future computational/mathematical biologists to create and evaluate more detailed models than currently is common practice.

Keywords: PSP model, simulator, multi-scale model, differential equations

1 Introduction

A physiologically structured population model is a multi-scale model which allows you to combine 2 models of different scale : a macroscopic or environment level model and a microscopic or cellular level model. Typically a compartmental model, a set of ordinary differential equations is used to model the environment. The microscopic model and the way it interacts at macroscopic level can be modelled in different ways: One way is to simulate each microscopic entity individually. The response of all the cells is accumulated to get the response at macroscopic scale. Another, less accurate, but computationally less demanding way, is to simulate an averaged microscopic cell and multiply this response with the number of cells. The PSP modelling method is a modelling method ranging between the former 2 extremes. Depending the number of discretisation points, the model will resemble more an averaged compartment. But when more points are used, it will resemble more a brute force approach where every single cell is simulated.

In [3] they modelled and simulated the Hepatitis-C virus - drug dynamics by using a PSP model. In [5] a numerical scheme was build to simulate this model by using a semi-Lagrangian partial differential equation solver to update the distribution of the different types of cells.

In this paper we will extend this framework towards more complex models which can incorporate more physiologically relevant quantities inside the cellular model.

Furthermore, we also created an improved algorithm to calculate the steady-state of de PSP model using a Newton-Raphson-Krylov solver.

At last, we also polished the structure of 1) the PSP language, 2) the PSP solver and 3) the generated PSP model code.

2 The PSP solver

The PSP solver consists of four different types of differential equation solvers. We use 3 ordinary differential equation (ODE) solvers and one partial differential equation (PDE) solver.

Two ODE solvers are used to solve the macroscopic and microscopic model f and g respectively. (eq. 1, 2)

$$\dot{y} = f(y, \Psi) \tag{1}$$

$$\dot{x} = g(x, y) \tag{2}$$

y is a vector of macroscopic state variables. x is a scalar or vector of microscopic state variables. Ψ is a scalar or vector representing the macroscopic effect of all the microscopic entities combined. This is calculated by integrating the distribution of the microscopic entities over the different physiologically structured quantities. (eq. 7)

A partial differential equation is used to link the microscopic simulation with the macroscopic simulation. This equation updates the distribution of the microscopic entities over the different possible states.

$$\begin{cases} \frac{\mathrm{d}n}{\mathrm{d}t} = \frac{\partial n(x,t)}{\partial t} + \nabla_x \cdot \left(g(y,x)\, n(x,t)\right) & (3) \\ = -\lambda(y,x)\, n(x,t) & (4) \\ \text{Boundary cond.:} \quad g(x)\, n(x,t)\, \hat{\mathrm{x}} = b(y,x) & (5) \\ \text{Initial conditions:} \quad n(x,0) = n_0(x) & (6) \end{cases}$$

$$\Psi(t) = \int_{\Omega} \phi(x)\, n(x,t)\, \mathrm{d}x \tag{7}$$

The third ordinary differential equation solver is used to solve the interactions between the previous three equations. The combination of all the equations is put into the operator $\Re$.

$$\dot{p} = \Re(p) \tag{8}$$

$$p = \begin{bmatrix} Y \\ \Psi \\ N \end{bmatrix} \tag{9}$$

Different types of ordinary differential equation solvers can be used. We currently use a Runge-Kutta 4 to simulate the macroscopic and microscopic models.

For the partial differential equation, which mainly consists of an advection term, we use the modified method of characteristics. This is an unconditionally stable method which allows to simulate the partial differential equation with large time-steps.

When more accuracy is desired the step size can be decreased. $\Re$ is solved with an implicit Euler ODE solver

3 Towards the simulation of more complex cellular models

When using the modified method of characteristics, we construct characteristic curves to create a mapping from the distribution of the different types of cells, at one point in time, to the distribution of the cells in next point in time.

For stability reasons, we simulate the characteristic curves in a forward and backward direction(see fig. 1).

The endpoint of the characteristic curve is most of the time not a grid point.

To resolve this we do an interpolation between the neighbouring endpoints, to find the mapping weights of a grid-point towards a few grid points in the next time step and vice versa for the backward direction.

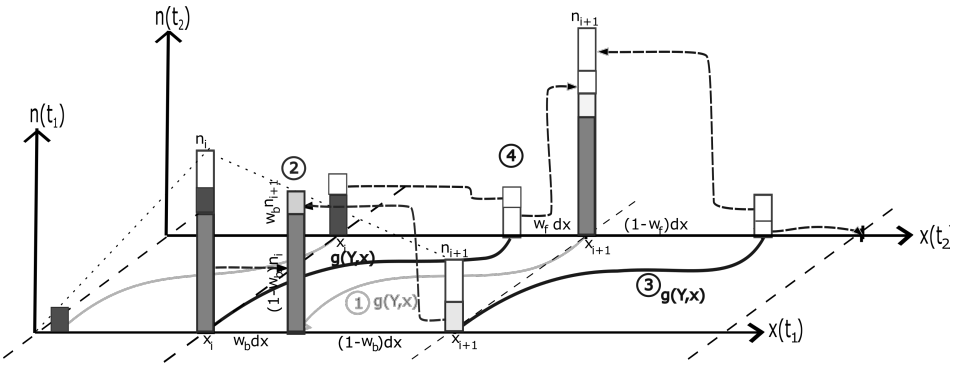

Figure 1: Modified method of characteristics applied on the distribution of the PSP. 1) Construction of the backward characteristic curves from the grid-points in t_2. 2) Interpolation at the characteristic endpoint, and subtraction of these values from the neighbouring grid-points. 3) Construction of the forward characteristic curves to project the remainder of the grid-point values. 4) Distribution of this remainder over the neighbouring grid-points.

When we characterize a cell by more than one characteristic quantity, we have to construct characteristic curves in a more-than one-dimensional grid.

Because of this, we also need to do interpolation in a higher dimensional grid.

Because our grid is uniform, we can use the bi-linear and tri-linear interpolation for the two and three dimensional case respectively.

The tri-linear interpolation interpolates a 3 dimensional point by using the 8 surrounding points. (This in contrast to barycentric coordinates which uses only 4 surrounding points.)

These 8 points are paired to construct orthogonal lines on which the first interpolation points are constructed.

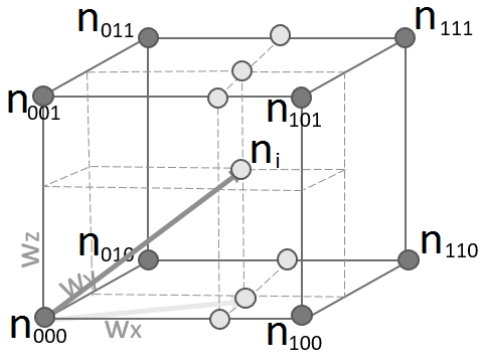

Figure 2: Tri-linear interpolation of a point in a regular 3D grid. .

These four interpolation points are paired again to form two new lines on which two new points of interpolation are constructed.

At last these 2 points are paired to form a line on which the point of interest, the end of the characteristic curve can be found.

With this construction, the interpolation weights of the 3 dimensional point can be found.

$$
\begin{aligned}
n_i = (1-w_z)(& \\
& (1-w_y)((1-w_x)n_{000} + w_x n_{100}) \\
& + w_y(1-w_x)n_{010} + w_x n_{110}) \\
& + w_z(\\
& (1-w_y)((1-w_x)n_{001} + w_x n_{101}) \\
& + w_y((1-w_x)n_{011} + w_x n_{111}))
\end{aligned}
\tag{10}
$$

As done in the 1D case, we drop these interpolation weights in the 2 mapping matrices W_b and W_f.

Then the 2 matrices are combined and corrected for mass conservation into the mapping matrix S.

$$
s_c = \left[1, 1, ...1\right] W_b \tag{11}
$$

$$W_b^* = W_b \times diag(1/\max(1, s_c)) \tag{12}$$

$$W_f^* = W_f \times diag(\max(1 - s_c, 0)) \tag{13}$$

$$S = W_b^* + W_f^* \tag{14}$$

Then the S matrix is used to map the distribution I from one point in time to the next.(eq.15)

$$I_2 = SI_1 \tag{15}$$

4 Steady-state computation

In some cases we are not interested in the dynamics of the distribution, we just want to know to which state the PSP model will converge to eventually. One way to do this is to simulate the system for a very long time. As was suggested in [5].

This is certainly a valid option since the time steps size of the modified method of characteristics is not limited by the Courant, Friedrich Levinson (CFL) constraint.

$$dt < C \cdot \frac{dx}{g(x)} \tag{16}$$

The CFL constraints the stepsize for Eulerian PDE solvers but the modified method of characteristics, is a semi-Langrangian method, which allows to make very big time steps. (see fig. 3)

Another way to calculate the steady state of the system is to solve the steady state equation:

$$U(p) = p \tag{17}$$

With $U(p)$ the updated PSP-state. When we use the implicit Euler update rule to update the overall system $\Re$, then $U(p_1) = p_2$ with p_2 the solution of the system

$$p_2 = p_1 + dt\Re(p_2) \tag{18}$$

To find the steady state, we need to find the roots of $U(p) - p = 0$.

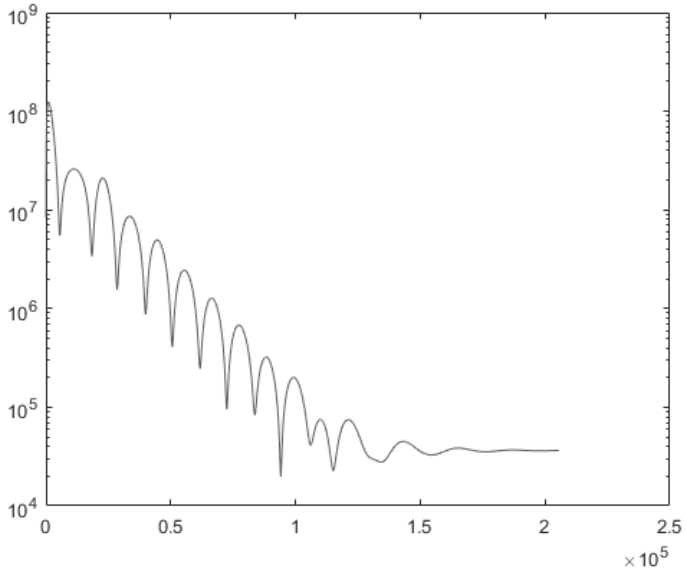

Figure 3: Convergence of the PSP simulator towards the steady state in function of the number of " big " timesteps.

Since the PSP update function is a non-linear function, we need to use a non-linear system solver.

A commonly used one, is the Newton-Raphson solver.

This method also needs the Jacobian J of the system.

If the Jacobian of the system is not available, the Jacobian can be approximated by calculating n, $(n = \text{length}(p))$ finite differences.

For each entry of the PSP state vector, we perturb p with a small value ϵ

$$J \approx \forall i \in \big[1 : \text{length}(p)\big] : J(i,:) = \frac{U(p + \epsilon\vec{e_i}) - U(p)}{\epsilon} \qquad (19)$$

Since the PSP model is a multi scale model, we typically have to deal with a large dynamic range of numbers: very big and very small numbers to model the macroscopic and microscopic model respectively. Because of this we encountered numerical issues due to

the finite numerical accuracy of the processor. To avoid these, the perturbation size should be chosen not too big since then the Jacobian is not calculated very accurately (and the solver might become also unstable) but also not too small since too small perturbations are numerically insignificant for the computer, resulting in a singular Jacobian matrix.

When the Jacobian is known we use the Jacobian to solve the next system:

$$J\Delta p = -r \tag{20}$$

Δp is the change of the PSP state each Newton step (eq. 21) and r is the residual of the steady state equation at the previous Newton iteration. (eq. 22)

$$p_2 = p_1 + \Delta p \tag{21}$$

$$r = U(p) - p \tag{22}$$

It should be noted that also other methods exist to approximate and use the Jacobian: all these are called quasi-Newton methods.

4.1 Solving the steady state equation

When solving the steady-state equation we have to take into account of couple of things. First, since the partial differential equation can have a lot of discretisation points, the system, and so the size of the Jacobian matrix, can become very big, . Secondly, We do note that the matrix is very sparse. (see fig. 4)

When inverting the matrix, this sparsity feature is exploited by the Krylov subspace methods.

Third thing to note is that the Jacobian Matrix, or the approximation of it, can be singular or ill-conditioned. For this, a form of regularisation should be used to solve eq. 20.

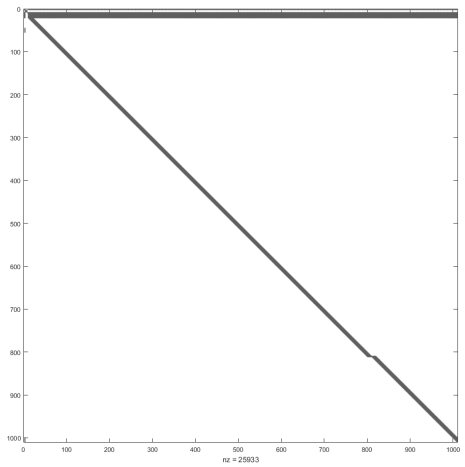

Figure 4: (black) the nonzero elements in the Jacobian Matrix.

4.2 Combining the systems

On the one hand, we have to construct the Jacobian on the other hand, we also have to solve the system. $J\Delta p = -r$ which essentially means inverting the Jacobian.

When using the subspace Krylov method GMRES, we iteratively build a Krylov subspace K_r conceptually by applying the matrix J multiple times on the residual r.

$$K_r = span\{r, Jr, J^2r, J^3r, \cdots J^nr\} \\ = span\{q_0, q_1, q_2, q_3, \cdots q_n\} \quad (23)$$

In practice, each iteration an orthonormal direction v is added on the matrix Q which spans the same Krylov subspace. Each iteration, an upper (almost triangular) Hessenberg matrix H is extended, decomposing the residual r further in the span of Q.

The application of the Jacobian matrix J on this direction v, returns the derivative of the steady state function in that direction v.

But instead of multiplying J with v, we can also find this by doing a finite difference approximation in the direction of v [1]:

$$Jv \simeq \frac{(U(p+hv) - (p+hv)) - (U(p) - p)}{h} \tag{24}$$

Note, that all these search directions are orthogonal, so that after n iterations also the total Jacobian is built.($n = \text{size}(p)$)

When solving the system $J\Delta p = -r$ using GMRES we also know that the total system is solved after maximal n iterations. Furthermore, the Arnoldi iteration inside GMRES, is a matrix-free algorithm: i.e. it only needs a function which returns the result of the matrix J applied to the vector v. This in contrast with most other commonly used linear algebra solvers: e.g. LU-decomposition. By combining the construction, application and inversion of the Jacobian, we can find the PSP state update Δp in only n Krylov iterations per Newton step. So worst case we have to apply the J matrix n times onto the residual, each Newton step. But for this application we do not need to construct the Jacobian explicitly but we can evaluate the finite difference approximation in the search direction of the GMRES algorithm v (eq.24). After n or less evaluations, GMRES will produce the Newton update Δp which allows us to update p (*eq*.21).

$$p = p + \Delta p \tag{25}$$

These Newton steps are repeated until convergence.

$$\Delta p < C_{tol} \tag{26}$$

This algorithm calculates efficiently the steady state solution of the PSP model.

4.3 Regularisation

Depending on the system which is simulated, the convergence of the Newton Method or the GMRES method might be slow or maybe not converging at all.

When this is the case, one needs to regularize the system to make it more stable.

One way of regularisation is early stopping, meaning that we don't iterate the GMRES algorithm n times but stop before total convergence of the GMRES has happened.

For the GMRES algorithm, the condition number of the system relates to the speed of convergence.

Our initial system had a condition number of over $1E20$.

When calculating the eigenvalues of the Jacobian we discovered that 2 eigenvalues were equal to Zero.

To invert this matrix in a stable fashion we did early stopping by allowing GMRES only to do $\max 10$ iterations per Newton step.

This allowed us to stably converge towards the solution.(fig.5)

Further analysis revealed that these zero valued eigenvalues originate from the Ψ variables. Indeed, these variables are calculated by integrating the distribution each iteration, which results into the fact that perturbations on these variables don't have any effect on the resulting simulation. Explaining the 2 zero eigenvalues.

$$\Psi = \int \psi(a) i dr$$

$$\psi(a) = r, R = \int r i dr$$

$$\psi(a) = 1, I = \int i dr$$

By removing these variables out of the steady state equations, the condition number of the Jacobian decreased to 1E13.

With these singular values removed, we could increase the number of iterations the GMRES algorithm could do each Newton step without the steady state solver becoming unstable.

We could increase the number of GMRES iterations each Newton step even up to the size of the PSP Vector. Meaning solving the system exactly. When we did this, the steady state equation was solved in 3-4 Newton steps, with each Newton step consisting of $\approx n$ GMRES steps.

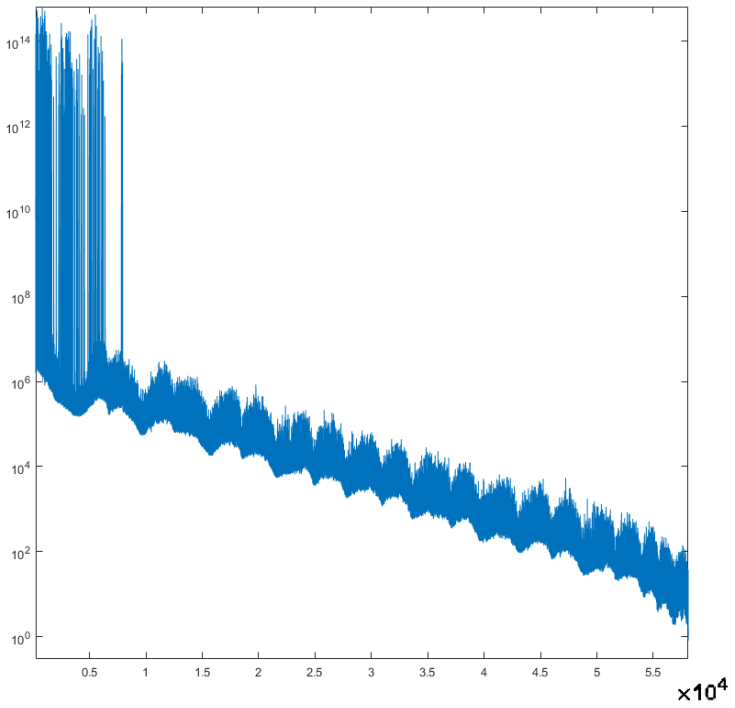

Figure 5: Noisy convergence of the Newton solver with early stopping regularisation.

So each Newton step takes a bit longer than the simulation of a certain time step, but overall the convergence of the Steady State of the PSP is accelerated by some orders of magnitude. (see fig. 6)

5 Framework

The PSP solver was put into a framework to simulate in a familiar way different types of virus-drug models.

For this a Domain specific Language (DSL) was constructed with a parser which converts the *.psp file into different types of source code.

Depending on the pharmacologist needs, he/she can choose to con-

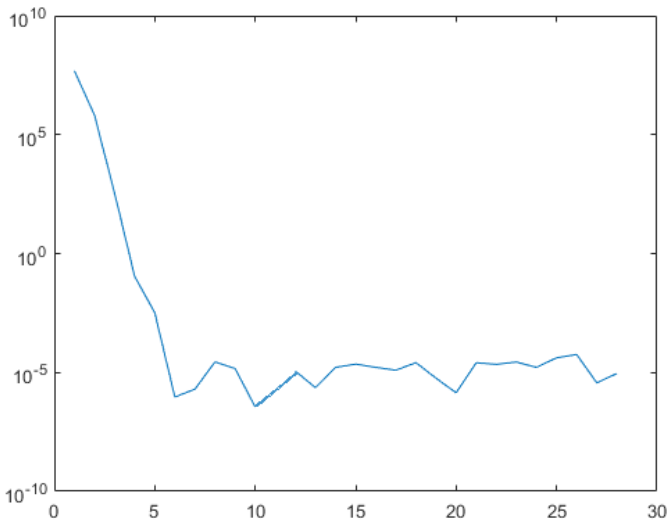

Figure 6: Fast convergence towards the solution of the discretised system by complete GMRES inversion each Newton step.

vert the psp model to C, C++, object oriented C++ or Matlab code. This code can be used to simulate the PSP models and to interface the models and simulators with different packages and libraries.

The *.C files can be compiled to an *.exe file which can be called from different scripting languages alternatively the *.C files can be compiled to a *.mex files which can be used in Matlab.

The advantage of these compiled *.mex files over the generated *.m files is that these mex files can be up to a factor 1000x faster than the native matlab script files.

Other libraries can interface with the object oriented C++ version of the PSP solver. This C++ code is build very modular which allows to easily replace one module with another: E.g. replacing one of the 3 ODE solver with your own ODE solver.

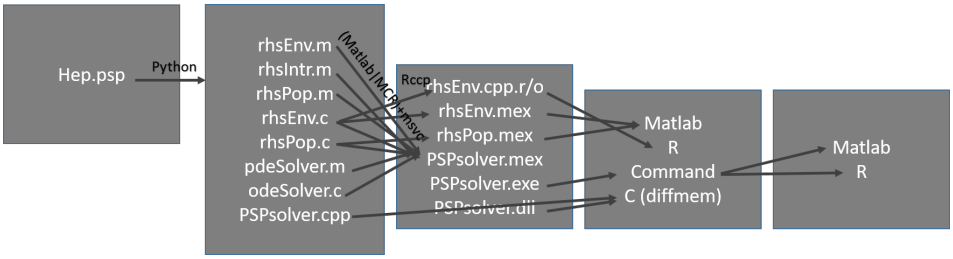

Figure 7: Software processing overview, illustrating the process from *.psp model to *.exe file.

6 Conclusion

We have developed a framework which allows to simulate a wide range of physiologically structured population models.

We increased the number of features of the PSP solver by introducing higher dimensional partial differential equations which originate from an increased number of characteristic quantities to describe a single cell. Further-on we accelerated the computation of the steady state solver by multiple orders of magnitude by using a Newton-Krylov method.

References

[1] Knoll D.A., Keyes D.E. , *Jacobian-Free Newton-Krylov methods: a survey of approaches and applications*, Journal of Computational Physics, August 2003.

[2] Diekmann, O.,et all , *Daphnia revisited: local stability and bifurcation theory for PSP models explained by way of an example.* Journal of Mathematical Biology, 61, (pp. 277-318) (42 p.). DOI:10.1007/s00285-009-0299-y.

[3] Woot de Trixhe, X., et all. *vRNA structured population model for Hepatitis C Virus dynamics.* Journal of theoretical biology, 378, DOI:10.1016/j.jtbi.2015.04.017. 2015

[4] Van Dyck, M. and Peremans, H., *Realtime 3D Sensor Based Air Flow Reconstruction.*, The Eurographics Association, DOI:10.2312/conf/EG2012/posters/041-042, (2012)

[5] Van Dyck, M. and Woot de Trixhe, X., and Vermeulen, A. and Vanroose, W.(2017) *A robust simulator for physiologically structured population models.*, Manuscript submitted for publication