The computer code: Vascular smooth muscle cell-single cell (VSMC-SC)

1 Introduction

We have used Kapela et al. [1] model as a building block to develop the new mathamatical model for VSMC. We have included new or modified description to NCX exchanger, IP_3R channel, NSC channel, agonist-induced IP_3 production, SR structure, and functioning. The SR is defined as a single luminaly continuous store with homogeneous luminal calcium. IP_3R is tetrameric with four subunits forming a single calcium-conducting channel. The opening of the IP_3R channel is regulated by calcium ions and IP_3 molecules. Calcium-mediated regulation of IP_3R is biphasic; it enhances channel opening at low calcium concentrations and deactivates channels at high calcium concentrations. We have incorporated a simplified sequential-binding IP_3R model in our VSMC model. The equation of the NCX is modified by incorporating the allosteric factor. A different description is given to the ionic current equation of the NSC channel. The Lemon et al. [2] model for agonist-induced IP_3 production is simplified by eliminating receptor activity regulation.

The VSMC model contains only the most relevant channels, pupms and exchangers for the agonist-induced calcium dynamics. For the specific study we are doing, we have eliminated the following components of the original Kapela et al. [1] model; NO/cGMP signalling, sodium-potassium pump (NaK), voltage-dependent potassium channel (K_V), potassium leak channel (K_{leak}) and NaKCl.

Please refer to the Jaijus's Doctoral thesis at https://ir.canterbury.ac.nz/handle/10092/12605 for model equations and the related information.

2 Numerical Method

Mascagni [3] developed a numerical procedure to solve Hodgkin-Huxley equations using a backward Euler method and fixed point iteration. Later on, Rempe and Chopp [4] implemented it in a branched neural network. We have based the solution procedure on numerical methods developed by Rempe and Chopp's algorithm [4]. The model equations can be written in a general form, for a particular state variable x_i.

$$\dot{x}_i = x_i g_i(x_1, ..., x_n) + h_i(x_1, ..., x_n) \tag{1}$$

The discretized equations, using Backward-Euler method, of all state variables can written in the form to give,

$$x_i(t+\Delta t)(1-\Delta t\ g_i(x_1(t+\Delta t), ..., x_n(t+\Delta t))) = x_i(t) + \Delta t\ h_i(x_1(t+\Delta t), ..., x_n(t+\Delta t)) \tag{2}$$

A fixed point iteration method is used to solve the above equation at a given time step [4, 3]. Functions g and h are calculated explicitly at each iteration using x_i from the previous iteration. The resultant equation of x_i for the iterative procedure is obtained as follows,

$$x_i^{(k+1)} = \frac{x_i(t) + \Delta t\ h_i\left(x_1^{(k)}, ..., x_n^{(k)}\right)}{1 - \Delta t\ g_i\left(x_1^{(k)}, ..., x_n^{(k)}\right)} \tag{3}$$

Where superscript k represents the iteration number. For $k = 1$, we choose $x_i^k = x_i(t)$ and a series of approximations x_i^k $(k = 2, 3, \ldots)$ to $x_i(t+\Delta t)$ is obtained using the equation 3. When the fixed point iterations converge, x_i^{k+1} is a solution to the equation 2. In the current model, iteration is continued until all the state variables converge to an error tolerance limit such that $x_i^{k+1} - x_i^k \leq 10^{-4}$ for all i and time step is fixed to 0.1 ms.

3 The structure of the code

The computer code is divided mainly into 5 parts (see Figure 8),

1. Defining model constants and parameters

2. Memory allocation
3. Initialization
4. Solution to single cell dynamics
5. Writing output files

3.1 Model definition

3.1.1 Model_constants.h

All the model constants are defined globally in Model_constants.h. Each model constants or simulation parameters are defined with its name, value, unit, and a short description. For example, the model constant, G_vocc (refer to Figure 1),

name = G_vocc
value = 2.63
unit = nS
short description = Whole cell conductance of VOCC

Model_constants.h

```
const double    G_vocc  =   2.63; ///< Whole cell conductance of VOCC,
nS

const double    G_bkca  =   11.1;///< Whole cell conductance of BKCa
channel, nS
const double    t_pf    =   0.84e-3; ///< Fast activation time constant,
s
const double    t_ps    =   35.9e-3; ///< Slow activation time constant,
s

const double    G_cacc  =   0.112 * C_m; ///< Whole cell conductance of
CACC, nS
const double    n_cacc  =   2; ///< order of the function shows
depenedncy on intracellular calcium
```

Fig. 1 A snapshot of a part of Model_constant.h file

3.1.2 Main.cpp

The simulation control parameters such as final time, time step tolerance limit, file writing frequency etc. are defined locally in the main file (main.cpp). For example, the simulation parameter, tfinal (refer to Figure 2),

name = tfinal
value = 300
unit = s
short description = Final time

```
int main(int argc, char* argv[]){

    /* Sigmoidal function */
        double L_max = 301; // Maimum agonist concentration, nM
        double L_incre = 50; // increment of agonist concentration, nM
        double L_ini = 300.0; // initial agonist concentration, nM
        double L_var = L_ini; // Agonist concentration, nM

    /* Simulation controlling parameters */
        double tnow = 0.0; // Initial time , s
        double tfinal = 300; // Final time, s
        double interval = 1e-4; // Time interval, s
        double tol_state = 1e-4; // Tolerance limit for the convergence
in the iteration
        double delay = 50; // Time allowed for variables to converge
        double stim_time = tfinal; // Total stimulation time, s
        double count_delay = 0; // control variable for the loop
        int tol_count = 0; // Control varialbe for fixed point iteration
```

Fig. 2 A snapshot of a part of main.cpp file

3.1.3 computelib.h

The model parameters, variables, and functions are defined in the computelib.h file. For example, the variable y,

variable name = y
Short description = Model state variables

For example, the function initialize() is defined with a short description, refer to Figure 4, as follows,

```
double *par; ///< par - model parameters
double *y; ///< y - model state variables
```

Fig. 3 A snapshot of a part of computelib.h file

Function name = initialize
Short description = This function initializes the model variables and parameters

```
void initialize(); ///< This function initializes the variables
void allocatememory(int var_tot, int par_tot); ///< This function
allocates memory for all the variables defined above
```

Fig. 4 A snapshot of a part of computelib.h file

3.1.4 Macros.h

The macros.h contains all the macros used to represent each model state variables in ‘y’ and parameters in ‘par’. For example, state variable membrane potential,

macro name = smc_vm
macro value = 0
Short description = Membrane potential

```
#define         smc_vm          0 ///< Membrane potential
#define         smc_ca          1 ///< Intracellular calcium
```

Fig. 5 A snapshot of a part of macros.h file

3.2 Memory allocation

The file Memory_allocation.cpp assigns the required memory space.

3.3 Initialization

All the state variables and model parameters are initialized in the Initializing.cpp file.

3.4 Solution to single cell dynamics

The Singlecell.cpp file contains all the single cell equations which are solved for each iteration. All the model parameters are calculated at the first part of the Singlecell.cpp file. For example, model parameters of CACC and NSC channels are calculated as given in the figure 6.

```
        /* CACC channel */
                par[i * num_par[0] + p_cacc]        =   (1 / (1 + pow
(k_cacc / y_converge[i * num_var[0] + smc_ca], n_cacc)));
                par[i * num_par[0] + I_cacc]        =   G_cacc * par[i *
num_par[0] + p_cacc] * (y_converge[i * num_var[0] + smc_vm] - par[i *
num_par[0] + E_Cl]);

        /* NSC channel */
                par[i * num_par[0] + p_nscvm]       =   1 / (1+exp(-
(y_converge[i * num_var[0] + smc_vm] + 78) / 45));
                par[i * num_par[0] + p_nscdag]      =   1 / (1 + k_nsc /
y_converge[i * num_var[0] + smc_DAG]);
                par[i * num_par[0] + I_nsc]         =   G_nsc * (par[i *
num_par[0] + p_nscdag]) * par[i * num_par[0] + p_nscvm] * (y_converge[i
* num_var[0] + smc_vm] - E_nsc);
```

Fig. 6 A snapshot of a part of Singlecell.cpp file

After that, the state variables are solved. For an example, the equations of the concentrations of IP_3, DAG and intra-SR calcium are given in the figure 7.

```
        /* IP3 and DAG concentration */
                y_temp[i * num_var[0] + smc_ip3]    =   ((y[i * num_var
[0] + smc_ip3] / interval) + (par[i * num_par[0] + r_hg] * PIP_2T /
gamma_G)) / (K_degG + 1 / interval);
                y_temp[i * num_var[0] + smc_DAG]    =   ((y[i * num_var
[0] + smc_DAG] / interval) + (par[i * num_par[0] + r_hg] * PIP_2T /
gamma_G)) / (K_DAG + 1 / interval);

        /* IntraSR calcium */
                y_temp[i * num_var[0] + smc_SR_ca]  =   ((y[i * num_var
[0] + smc_SR_ca] / interval) + par[i * num_par[0] + I_serca] / beta -
par[i * num_par[0] + I_srleak] / beta   + Q_ip3r[i] * par[i * num_par[0]
+ p_ip3r] * y_converge[i * num_var[0] + smc_ca] +  Q_ryr[i] * P2
(y_converge[i * num_var[0] + smc_R10]) * y_converge[i * num_var[0] +
smc_ca]) / ( Q_ip3r[i] * par[i * num_par[0] + p_ip3r] +  Q_ryr[i] * P2
(y_converge[i * num_var[0] + smc_R10]) + 1 / interval);
```

Fig. 7 A snapshot of a part of Singlecell.cpp file

3.5 Writing output files

The last part of the code (Outputdata.cpp) is to save the results in the .txt format. The .txt files are imported into Matlab for plotting and analysing the results.

4 The computer algorithm

The first step of the algorithm is to define all the parameters to satisfy the simulation requirements and problem definition.

4.1 Agonist concentration

A while loop is used to repeat the simulation to run the code over a range of agonist concentrations. The variable 'L_var' is the agonist concentration which is changed along the loop and is assigned to the variable 'L_cell'. The variable 'L_cell' is used to determine the cell dynamics. The variables 'L_max' and 'L_ini' represents the final and initial values of agonist concentration, respectively, refer to Figure 9. The variable L_incre is the increment of agonist concentration for the while loop.

4.2 Simulation controlling parameters

The initial time, final time, time interval, and tolerance limit are assigned to 'tnow', 'tfinal', 'interval', and 'tol_state variables', respectively, refer Figure 10. The simulation is run for first 'delay'seconds to get steady state values with the base agonist concentration (0 nM). The variable 'L_rest' is used to store base agonist concentration.

The frequency of saving output data is based on the file writing frequency defined. As shown in the figure 10, the data is saved for every 'file_write_freq' seconds.

After defining the problem, the storage space for the variables are allocated. Once the memory allocation is successful, the output files are created, refer to Figure 11. The name of the file is defined in the command line starting with 'sprintf'. For example, the intracellular calcium and membrane potential data are stored as 'Ca_timeseries.txt' and 'Vm_timeseries.txt', respectively. Then, the variables and parameters are initialized and saved in the output files, refer to Figure 12.

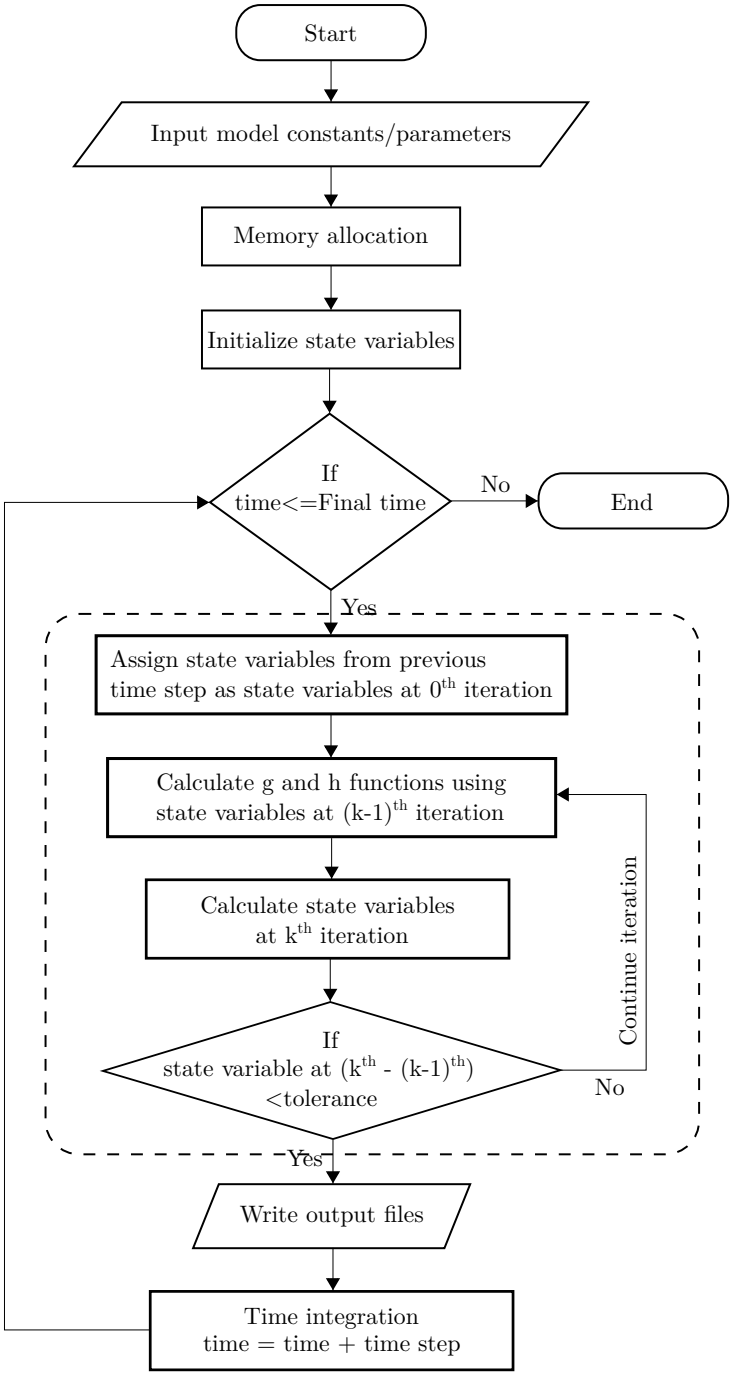

Fig. 8 Computer algorithm: Contains 5 parts 1) define model constants/parameters, 2) memory allocation, 3) initialization, 4) fixed point iteration and 5) writing output data. Part of the algorithm for fixed point iteration is given in the dashed box.

```
/* Agonist concentration */
    double L_max = 301; // Maimum agonist concentration, nM
    double L_incre = 50; // increment of agonist concentration, nM
    double L_ini = 300.0; // initial agonist concentration, nM
    double L_var = L_ini; // Agonist concentration, nM
```

Fig. 9 A snapshot of a part of main.cpp file: Agonist concentration

```
/* Simulation controlling parameters */
    double tnow = 0.0; // Initial time , s
    double tfinal = 300; // Final time, s
    double interval = 1e-4; // Time interval, s
    double tol_state = 1e-4; // Tolerance limit for the convergence in the iteration
    double delay = 50; // Time allowed for variables to converge
    double stim_time = tfinal; // Total stimulation time, s
    double count_delay = 0; // control variable for the loop
    int tol_count = 0; // Control varialbe for fixed point iteration
    double file_write_freq = 0.01; // File writing frequency, s
    int folder_status; // variable to check the successful creartion of folder
```

Fig. 10 A snapshot of a part of main.cpp file: Simulation parameters

```
sprintf(temp_name,"%s/Ca_timeseries.txt",sub_folder);
fca = fopen(temp_name,"w+");
sprintf(temp_name,"%s/Vm_timeseries.txt",sub_folder);
fvm = fopen(temp_name,"w+");
```

Fig. 11 A snapshot of a part of main.cpp file: Creating output files

```
initialize(); // Initialize the variables
dump_data(ft, fvm, fca, fpmca, fsrleak, fip3, fserca, fncx, fvocc, fbkca, fip3r, fcacc, fnsc ,
fsrca, fryr,fdag, tnow); // Creating output files
```

Fig. 12 A snapshot of a part of main.cpp file: Initializing

4.3 Solution algorithm

The solution to the problem is determined for each time intervals. When the time is greater than or equal to ‘delay’ seconds, the required agonist concentration is assigned to the cell, refer to Figure 13.

The solution of the single cell dynamics is computed using a fixed point iteration algorithm, (refer to Figure 8). All the operations inside the dashed line box in Figure 8 represent the iteration algorithm. The corresponding part of the code is given in the figure 14. At the start of the algorithm, g and h functions are calculated using the state variables from the previous iteration, $(k\text{-}1)^{th}$. This is achieved by calling the subprogram ‘singlecell’, refer to the file singlecell.cpp. To calculate the g and h functions for the first iterative step, state variables from the previous time step are used. Using these g and h values, state variables are calculated in the

```
// Assign required agonist concentration to the cell after "delay" seconds.
if (tnow >= (delay) && count_delay < 1)
{
    count_delay = 2;
    for (int i = 0; i < num_cell[0]; i++)
        {
        L_cell[i] = L_var;
        }

}
```

Fig. 13 A snapshot of a part of main.cpp file: Assigning agonist concentration

new iteration, k^{th}. The state variables calculate in the $(k-1)^{th}$ and k^{th} iterations are compared to check for tolerance limit, refer to Figure 14. If the tolerance limit is not satisfied, iteration algorithm is repeated.

```
    singlecell(tnow, interval); // Solving single cell model equations

/* Fixed point iteration checking*/
    for (int i=0;i<num_cell[0];i++)
    for (int j=0;j<num_var[0];j++)
        if (fabs(y_temp[i * num_var[0] + j] - y_converge[i * num_var[0] + j])>=tol_state)
        {
            tol_count = 0;
            /* Not converged */
            for (int k=0;k<num_cell[0];k++)
                for (int m=0;m<num_var[0];m++)
                        y_converge[k * num_var[0] + m] = y_temp[k * num_var[0] + m];
            break;
        }
```

Fig. 14 A snapshot of a part of main.cpp file: Solution algorithm

Once the solution is converged, the output data are saved in the corresponding files by calling the subprogram ‘dump_data’ for the given file frequency rate, refer to Figure 15.

```
                //  Checking file writing condition
                    if (int(tnow/file_write_freq) == count)
                        {
                        dump_data(ft, fvm, fca, fpmca, fsrleak, fip3, fserca, fncx, fvocc, fbkca, fip3r,
fcacc, fnsc , fsrca, fryr, fdag, tnow);
                        count++;
                        }
```

Fig. 15 A snapshot of a part of main.cpp file: Writing files

The last two parts of the code, fixed point iteration and writing output files, are repeated until the final time is reached.

References

[1] Kapela, A., Bezerianos, A., and Tsoukias, N. M. (2008). A mathematical model of Ca2+ dynamics in rat mesenteric smooth muscle cell: agonist and NO stimulation. *Journal of theoretical biology*, 253(2):238–260.

[2] Lemon, G., Gibson, W., and Bennett, M. (2003). Metabotropic receptor activation, desensitization and sequestration-I: modelling calcium and inositol 1,4,5-trisphosphate dynamics following receptor activation. *Journal of Theoretical Biology*, 223(1):93–111.

[3] Mascagni, M. (1990). The Backward Euler Method for Numerical Solution of the Hodgkin-Huxley Equations of Nerve Conduction. *SIAM Journal on Numerical Analysis*, 27(4):941–962.

[4] Rempe, M. J. and Chopp, D. L. (2006). A Predictor-Corrector Algorithm for Reaction-Diffusion Equations Associated with Neural Activity on Branched Structures. *SIAM Journal of Scientific Computing*, 28(6):2139–2161.