Supporting Information

Cluster Code

Elías Vera-Sigüenza[1]

[1]The University of Auckland, Department of Mathematics,
Level 2, Building 303, 38 Princes Street, Auckland CBD, New Zealand.

E-mail: [a]elias.siguenza@auckland.ac.nz

Cluster Code

Here I will describe a bunch of stuff about the way we connect our models (Nathan's Ca2+ and Elias' FFR model.) I will also describe every bit of the code package that I'll send to you. Even if it may be overkill its never a bad thing to have a full description of the lastest version.

FFR Code

So the idea here is that MATLAB requires two things to run their ode solver routine ode15s:

- Firstly, you need to make a 'function' file, say `model.m`, in which you write down the differential equation or differential equation system you want to solve. In our case I have the file `cluster.m`.

- Secondly, we need an executable line. For simplicity I usually define a handle called `f_secretion ... = @(t,x)cluster(t,x,Ca,parameters)` (I will explain how that works in a second). The important thing here is that then I can use the ode routine by defining two outputs (time and a vector of solutions (or variables that change in time)): `[t,U] = ode15s(f_secretion,[0 ... 5e2],par.IC_new, options);`.

FFR Files: var.m

So `var.m` is a function I created with the sole purpose of arranging the variables of the system in a way that I could work with them and made sense to me. Ofcourse this file is not necessary, and you might find it necessary to change. But here's how it works:

John's cluster mesh data includes a column of cell values which are common to other cells and a second list but backwards, that is, a neighbour cell to the cell we're interested in. As such this creates something like (Note: (i,j) denotes ith cell jth neighbour)...

Cell 1 has an apical membrane divided into four:

- (1,1) This piece of apical membrane leads to water out into a luminal space shared with no other cell.
- (1,2) this piece of apical membrane is shared with cell 2.
- (1,3) this piece of apical membrane is shared with cell 3.
- (1,4) this piece of apical membrane is shared with cell 4.

Cell 2 has an apical membrane divided into six:

- (2,1) This piece of apical membrane is shared with cell 1.
- (2,2) This piece of apical membrane leads to water out into a luminal space shared with no other cell.
- (2,3) This piece of apical membrane is shared with cell 3.
- (2,4) This piece of apical membrane is shared with cell 4.
- (2,5) This piece of apical membrane is shared with cell 5.
- (2,6) This piece of apical membrane is shared with cell 6.

Cell 3 has an apical membrane divided into five:

- (3,1) This piece of apical membrane is shared with cell 1.
- (3,2) This piece of apical membrane is shared with cell 2.
- (3,3) This piece of apical membrane leads to water out into a luminal space shared with no other cell.
- (3,4) This piece of apical membrane is shared with cell 4.
- (3,5) This piece of apical membrane is shared with cell 5.

Cell 4 has an apical membrane divided into six:

- (4,1) This piece of apical membrane is shared with cell 1.
- (4,2) This piece of apical membrane is shared with cell 2.
- (4,3) This piece of apical membrane is shared with cell 3.
- (4,4) This piece of apical membrane leads to water out into a luminal space shared with no other cell.
- (4,5) This piece of apical membrane is shared with cell 5.
- (4,6) This piece of apical membrane is shared with cell 6.

Cell 5 has an apical membrane divided into five:

- (5,2) This piece of apical membrane is shared with cell 1.
- (5,3) This piece of apical membrane is shared with cell 3.
- (5,4) This piece of apical membrane is shared with cell 4.
- (5,6) This piece of apical membrane is shared with cell 6.
- (5,7) This piece of apical membrane is shared with cell 7.

Cell 6 has an apical membrane divided into five:

- (6,2) This piece of apical membrane is shared with cell 2.
- (6,4) This piece of apical membrane is shared with cell 4.
- (6,5) This piece of apical membrane is shared with cell 5.
- (6,6) This piece of apical membrane leads to water out into a luminal space shared with no other cell.
- (6,7) This piece of apical membrane is shared with cell 7.

Cell 7 has an apical membrane divided into two:

- (7,5) This piece of apical membrane is shared with cell 5.
- (7,6) This piece of apical membrane is shared with cell 6.

I created a cell array in the parameters object called `par.neigh` which gives me this information:

```
par.neigh{1}
ans = 1     2     3     4
>> par.neigh{2}
ans = 1     2     3     4     5     6
>> par.neigh{3}
ans = 1     2     3     4     5
>> par.neigh{4}
ans = 1     2     3     4     5     6
>> par.neigh{5}
ans = 2     3     4     6     7
>> par.neigh{6}
ans = 2     4     5     6     7
```

So `var.m` takes as an input an organised vector of variables. I've arranged the initial condition vector, which I call `par.IC_new` in this way:

For the intracellular: (This happens 7×8, so `par.IC_new(1:56)` are all intracellular values)

1. Volume
2. Na^+
3. K^+
4. Cl^-
5. HCO_3^-
6. H^+
7. V_a
8. V_b

The luminal variables are a bit more complicated. As you can see from above, some indices are repeated. That is, cell 7 for instance has an apical bit shared with cell 5. Conversely, cell 5 has an apical bit shared with cell 7. However, in the luminal space shared between these cells I only account for one amount of sodium one amount of potassium etc... This lead me to create a new neighbour cell array. This time I remove all the repeated indices according to neighbour, not to cell in question (I call it `par.neigh_clust`:

```
>> par.neigh_clust{1}
ans = 1
>> par.neigh_clust{2}
ans = 1     2
>> par.neigh_clust{3}
ans = 1     2     3
>> par.neigh_clust{4}
ans = 1     2     3     4
>> par.neigh_clust{5}
ans = 2     3     4
>> par.neigh_clust{6}
ans = 2     4     5     6
>> par.neigh_clust{7}
ans = 5     6
```

According to these I arranged the variables as follows: from `par.IC_new(56+1:57+19)`, I fill it with luminal sodium (`Nal`). From `par.IC_new(76:76+19)`, I fill it with luminal potassium (`Kl`). From `par.IC_new(76:76+19)`, I fill it with luminal potassium (`Kl`). From `par.IC_new(94:113)`, I fill it with luminal chloride (`Cll`).

Now, in the `var.m` I have arranged the intracellular variables (in a matrix I call `int`), that is, the stuff that is only relevant inside the cell as follows:

$$int = \begin{pmatrix} Cell1 & Cell2 & Cell3 & Cell4 & Cell5 & Cell6 & Cell7 \\ \text{Vol.} & \text{Vol.} & \text{Vol.} & \text{Vol.} & \text{Vol.} & \text{Vol.} & \text{Vol.} \\ \text{Na}^+ & \text{Na}^+ & \text{Na}^+ & \text{Na}^+ & \text{Na}^+ & \text{Na}^+ & \text{Na}^+ \\ \text{K}^+ & \text{K}^+ & \text{K}^+ & \text{K}^+ & \text{K}^+ & \text{K}^+ & \text{K}^+ \\ \text{Cl}^- & \text{Cl}^- & \text{Cl}^- & \text{Cl}^- & \text{Cl}^- & \text{Cl}^- & \text{Cl}^- \\ \text{HCO}_3^- & \text{HCO}_3^- & \text{HCO}_3^- & \text{HCO}_3^- & \text{HCO}_3^- & \text{HCO}_3^- & \text{HCO}_3^- \\ \text{H}^+ & \text{H}^+ & \text{H}^+ & \text{H}^+ & \text{H}^+ & \text{H}^+ & \text{H}^+ \\ V_a & V_a & V_a & V_a & V_a & V_a & V_a \\ V_b & V_b & V_b & V_b & V_b & V_b & V_b \end{pmatrix}$$

This way I can call any given variable as follows, say I want chloride of cell 1 , I would call it as: `int(4,cell_number)`.

Now, the luminal variables are arranged as follows, say for luminal sodium `Nal`:

$$Nal = \begin{pmatrix} \text{Na}_l^+ & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{Na}_l^+ & \text{Na}_l^+ & 0 & 0 & 0 & 0 & 0 \\ \text{Na}_l^+ & \text{Na}_l^+ & \text{Na}_l^+ & 0 & 0 & 0 & 0 \\ \text{Na}_l^+ & \text{Na}_l^+ & \text{Na}_l^+ & \text{Na}_l^+ & 0 & 0 & 0 \\ 0 & \text{Na}_l^+ & \text{Na}_l^+ & \text{Na}_l^+ & 0 & 0 & 0 \\ 0 & \text{Na}_l^+ & 0 & \text{Na}_l^+ & \text{Na}_l^+ & \text{Na}_l^+ & 0 \\ 0 & 0 & 0 & 0 & \text{Na}_l^+ & \text{Na}_l^+ & 0 \end{pmatrix}$$

I repeat this for `Kl`, and `Cll`:

$$Kl = \begin{pmatrix} \text{K}_l^+ & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{K}_l^+ & \text{K}_l^+ & 0 & 0 & 0 & 0 & 0 \\ \text{K}_l^+ & \text{K}_l^+ & \text{K}_l^+ & 0 & 0 & 0 & 0 \\ \text{K}_l^+ & \text{K}_l^+ & \text{K}_l^+ & \text{K}_l^+ & 0 & 0 & 0 \\ 0 & \text{K}_l^+ & \text{K}_l^+ & \text{K}_l^+ & 0 & 0 & 0 \\ 0 & \text{K}_l^+ & 0 & \text{K}_l^+ & \text{K}_l^+ & \text{K}_l^+ & 0 \\ 0 & 0 & 0 & 0 & \text{K}_l^+ & \text{K}_l^+ & 0 \end{pmatrix}$$

$$Cll = \begin{pmatrix} \text{Cl}_l^- & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{Cl}_l^- & \text{Cl}_l^- & 0 & 0 & 0 & 0 & 0 \\ \text{Cl}_l^- & \text{Cl}_l^- & \text{Cl}_l^- & 0 & 0 & 0 & 0 \\ \text{Cl}_l^- & \text{Cl}_l^- & \text{Cl}_l^- & \text{Cl}_l^- & 0 & 0 & 0 \\ 0 & \text{Cl}_l^- & \text{Cl}_l^- & \text{Cl}_l^- & 0 & 0 & 0 \\ 0 & \text{Cl}_l^- & 0 & \text{Cl}_l^- & \text{Cl}_l^- & \text{Cl}_l^- & 0 \\ 0 & 0 & 0 & 0 & \text{Cl}_l^- & \text{Cl}_l^- & 0 \end{pmatrix}$$

Now the final thing I do in `var.m` file is to fill in the respective apical bits that need to be according to the `par.neigh` as seen above. This is because each cell has a distinct ratio of apical to basolateral, this is

important in the calculation of plasma membrane potentials. So the output of the `var.m` file would result in:

$$Nal = \begin{pmatrix} Na_l^+ & Na_l^+ & Na_l^+ & Na_l^+ & 0 & 0 & 0 \\ Na_l^+ & Na_l^+ & Na_l^+ & Na_l^+ & Na_l^+ & Na_l^+ & 0 \\ Na_l^+ & Na_l^+ & Na_l^+ & Na_l^+ & Na_l^+ & 0 & 0 \\ Na_l^+ & Na_l^+ & Na_l^+ & Na_l^+ & Na_l^+ & Na_l^+ & 0 \\ 0 & Na_l^+ & Na_l^+ & Na_l^+ & 0 & Na_l^+ & Na_l^+ \\ 0 & Na_l^+ & 0 & Na_l^+ & Na_l^+ & Na_l^+ & Na_l^+ \\ 0 & 0 & 0 & 0 & Na_l^+ & Na_l^+ & 0 \end{pmatrix}$$

`Nal` and `Cll` follow. Finally, the `var.m` file is called inside the `cluster.m` function as: `[int,Nal,Kl,Cll] ... = var(x,par);`

```
function [int,Nal,Kl,Cll] = var(x,par)
% ##########################################################################
% --------------------------------------------------------------------------
% Author: Elias Siguenza
% Location: The University of Auckland, New Zealand
% Date: 22 March 2019
% Version: 2.1
% --------------------------------------------------------------------------
% Purpose:
% This function sorts out the variables to a matrix of seven columns
% (that represent each cell) and 8 rows (representing concentrations and PM
% potentials along with cell volume).
% The order is:
% Volume,
% Na+
% K+
% Cl-
% HCO3-
% H+
% Va
% Vb
% --------------------------------------------------------------------------
% ##########################################################################
% Initialise and preallocate memory for the loop.
int = zeros(8,7);
Nal = zeros(7,7);
Kl = zeros(7,7);
Cll = zeros(7,7);
n = 1;
l = 57;
%%%%%%%%%%%%%%%%%% Begin Loop
for i = 1:7
    int(1:8,i) = x(n:n+7);
    n= n+8;
    for j = 1: par.ind_clust{i}
        ngh = par.neigh_clust{i}(j);
            Nal(i,ngh) = x(l);
            Kl(i,ngh) = x(l+sum(cell2mat(par.ind_clust)));
            Cll(i,ngh) = x(l+(2*sum(cell2mat(par.ind_clust))));
            l = l+1;
    end
end
%%%%%%%%%%%%%%%%%%
% Luminal concentration matrix:
```

```
% |1,0,0,0,0,0,0|
% |1,1,0,0,0,0,0|
% |1,1,1,0,0,0,0|
% |1,1,1,1,0,0,0|
% |0,1,1,1,0,0,0|
% |0,1,0,1,1,1,0|
% |0,0,0,0,1,1,0|
%%%%%%%%%%%%%%%%%
% NOTE: The luminal concentration matrix is
% essentially a lower triangular matrix whose rows represent cell number
% and its columns represent neighbour. However, for  calculation of
% membrane potentials and tight junctional fluxes some of these must be
% repeated. However they are not variables of the system. To get around
% that, I just add the matrix's transpose to the variable matrix and I
% obtain what I want:

Nal = Nal + (tril(Nal)'-diag(diag(Nal)));
Kl = Kl + (tril(Kl)'-diag(diag(Kl)));
Cll = Cll + (tril(Cll)'-diag(diag(Cll)));
%%%%%%%%%%%%%%%%%

end
```

FFR Files: fx_ba.m

This function is straight forward. It contains the equations concerning the flux of channels and transporters of the cell that are located in the basolateral pm.

```
function [Qb, JNaK, JNkcc1, JAe4, JNhe1, JBB, JK, Ii] = fx_ba(int,Ca,par,cell_no)
% ########################################################################
% ------------------------------------------------------------------------
% Author: Elias Siguenza
% Location: The University of Auckland, New Zealand
% Date: 22 March 2019
% Version: 2.1
% ------------------------------------------------------------------------
% Purpose:
% This function takes as an input the intracellular variables of the cell
% model and the Ca in order to calculate the mebrane ionic fluxes,
% and the flow rate into the cell, at the basolateral side of any
% particular cell.
% ------------------------------------------------------------------------
% ########################################################################
% Basolateral Flow Rate
    Qb = par.Lb * ( 2 * ( int(2,cell_no) + int(3,cell_no) + int(6,cell_no) ) + ...
        par.CO20 - par.Ie);
% K+ Nernst Potential
    VK = par.RTF * log(par.Ke/int(3,cell_no));
% Ca2+ Activated K+ Channels open probability
    PK = ...
        sum((1./(1+(par.KCaKC./(Ca{2}{cell_no})).^par.eta2)).*par.Sb_k{cell_no})./par.Sb{cell
% Ca2+ Activated K+ Channels Total Flux
    JK = par.GK * PK * ( int(8,cell_no) - VK ) / par.F;
% 3Na+/2K+ ATPases
    JNaK  = ...
        par.Sb{cell_no}*par.aNaK*(par.r*par.Ke^2*int(2,cell_no)^3/(par.Ke^2+par.alpha1*int(2,
% Na+ K+ 2Cl- Cotransporters
```

```
    JNkcc1= ...
        par.aNkcc1*par.Sb{cell_no}*(par.a1-par.a2*int(2,cell_no)*int(3,cell_no)*int(4,cell_nc
% Na+-2HCO3-/Cl- Anion Exchanger 4
    JAe4  = par.Sb{cell_no}*par.G4 ...
        *((par.Cle/(par.Cle+par.KCl))*(int(2,cell_no)/(int(2,cell_no)+par.KNa))*(int(5,cell_n
% Na+/H+ Antiporter 1
    JNhe1 = ...
        par.Sb{cell_no}*par.G1*((par.Nae/(par.Nae+par.KNa))*(int(6,cell_no)/(par.KH+int(6,cel
% CAIV Intracellular Carbonic Anhydrases
    JBB   = ...
        int(1,cell_no)*par.GB*(par.kp*par.CO20-par.kn*int(5,cell_no)*int(6,cell_no));
% Intracellular Osmolarity (Using electroneutrality principle)
    Ii = 2*(int(2,cell_no) + int(3,cell_no) + int(6,cell_no)) + par.CO20;
% ########################################################################
end
```

A few remarks, the parameter vector: `par.Sb_k{cell_no}` is a cell array that contains the surface area of each triangle in the basolateral membrane of the i^{th} cell. `par.Sb{cell_no}` is a cell array that contains the total surface area of the basal region of each cell.

FFR Files: fx_ap.m

This function is straight forward. It contains the equations concerning the flux of channels and transporters of the cell that are located in the apical and tight junction areas.

```
function [JtNa,JtK,JCl,Qa, Qtot] = fx_ap(int,Nal,Kl,Cll,Jb,Ca,par,c_no,ngh)
% ########################################################################
% ------------------------------------------------------------------------
% Author: Elias Siguenza
% Location: The University of Auckland, New Zealand
% Date: 22 March 2019
% Version: 2.1
% ------------------------------------------------------------------------
% Purpose:
% This function takes as an input the intracellular and luminal variables
% of the cell model and the Ca in order to calculate the apical
% mebrane ionic fluxes, and the flow rate out and into the lumen,
 % of any particular cell.
% ------------------------------------------------------------------------
% ########################################################################
% Tight Junctional Membrane Potential
Vt = int(7,c_no)-int(8,c_no);
% ------------------------------------------------------------------------
% Ca2+ Activated Apical Cl- Channels
PCl=1./(1+(par.KCaCC./Ca{1}{c_no,ngh}).^par.eta1);
PrCl = sum(PCl.*par.com_tri_ap{c_no,ngh}(:,3))/par.Sa{c_no};
VCl = par.Sa_p{c_no,ngh} * par.RTF * log(Cll(c_no,ngh)/int(4,c_no));
JCl= par.GCl * PrCl * (int(7,c_no) + (VCl/par.Sa_p{c_no,ngh}))/par.F;
% ------------------------------------------------------------------------
% Tight Junctional Fluxes
VtNa = par.Sa_p{c_no,ngh} * par.RTF * log(Nal(c_no,ngh)/par.Nae);
JtNa = par.Sa_p{c_no,ngh}*par.GtNa*par.St*(Vt-(VtNa/par.Sa_p{c_no,ngh}))/par.F;
% ------------------------------------------------------------------------
VtK = par.Sa_p{c_no,ngh} * par.RTF * log(Kl(c_no,ngh)/par.Ke);
JtK = par.Sa_p{c_no,ngh}*par.GtK*par.St*(Vt-(VtK/par.Sa_p{c_no,ngh}))/par.F;
% ------------------------------------------------------------------------
% Luminal Osmolarity (Using electroneutrality principle)
Il = 2*Cll(c_no,ngh) + par.Ul;
```

```
% ------------------------------------------------------------------------
% Flow Rates
% Apical Flow Rate
Qa = par.Sa_p{c_no,ngh} * par.La * (Il - Jb);
% Tight Junctional Flow Rate
Qt = par.Sa_p{c_no,ngh} * par.Lt * (Il - par.Ie);
% Total Fluid Flow Rate (into/out of lumen)
Qtot =(Qa + Qt);
% ########################################################################
end
```

Note: the cell array `par.Sa_p{c_no,ngh}` contains the ratio of piece of shared apical surface area to total apical surface area, that is, `sum(par.Sa_ck{c_no,ngh})/ par.Sa{c_no}`.

FFR functions: ieq.m

```
function eq = ieq(JCL,JtNa,JtK,Jb,int,c_no)
% ########################################################################
% ------------------------------------------------------------------------
% Author: Elias Siguenza
% Location: The University of Auckland, New Zealand
% Date: 22 March 2019
% Version: 2.1
% ------------------------------------------------------------------------
% Purpose:
% This function calculates the differential equations of the intracellular
% ionic concentrations and the membrane potentials of any given cell
% using the int matrix of seven columns
% (that represent each cell) and 8 rows (representing concentrations and PM
% potentials along with cell volume).
% The order is:
% Volume,
% Na+
% K+
% Cl-
% HCO3-
% H+
% Va
% Vb
% ------------------------------------------------------------------------
% ########################################################################
% dw/dt
eq(1,1) = Jb(c_no,9);
% d[Na+]i/dt
eq(2,1) = ...
    (Jb(c_no,3)-3*Jb(c_no,2)+Jb(c_no,5)-Jb(c_no,4)-Jb(c_no,9)*int(2,c_no))/int(1,c_no);
% d[K+]i/dt
eq(3,1) = (Jb(c_no,3)+2*Jb(c_no,2)-Jb(c_no,7)-Jb(c_no,9)*int(3,c_no))/int(1,c_no);
% d[Cl-]i/dt
eq(4,1) = (2*Jb(c_no,3)+Jb(c_no,4)+JCL(c_no,1)-Jb(c_no,9)*int(4,c_no))/int(1,c_no);
% d[HCO3-]i/dt
eq(5,1) = (Jb(c_no,6)-2*Jb(c_no,4)-Jb(c_no,9)*int(5,c_no))/int(1,c_no);
% d[H+]i/dt
eq(6,1) = (Jb(c_no,6)-Jb(c_no,5)-Jb(c_no,9)*int(6,c_no))/int(1,c_no);
% dVa/dt
eq(7,1) = - JCL(c_no,1) - (sum(JtNa(c_no,:)) + sum(JtK(c_no,:)));
% dVb/dt
```

```
eq(8,1) = - Jb(c_no,2) - Jb(c_no,7) + (sum(JtNa(c_no,:)) + sum(JtK(c_no,:)));
end
```

Again, I think this function is pretty straight forward. All it does is calculates the differential equations using the fluxes from the functions above and gives me the values to update the variables.

FFR functions: lum_adj.m

```
function [JtNa,JtK,JCl,Qtot,Nal,Kl,Cll,QwNa,QwK,QwCl] = ...
                                lum_adj(Nal,Kl,Cll,JtNa,JtK,JCl,Qtot,par)
 % ########################################################################
% ------------------------------------------------------------------------
% Author: Elias Siguenza
% Location: The University of Auckland, New Zealand
% Date: 22 March 2019
% Version: 1.1
% ------------------------------------------------------------------------
% Purpose:
% This function uses the adjacency matrix to calculate the luminal structure
% equations.
% This is a dirty function and should be re-written.
% ------------------------------------------------------------------------
% ########################################################################
% Add upper to lower triangles:
    Qtot = (Qtot - triu(Qtot)) + triu(Qtot)';
    JtNa  = (JtNa - triu(JtNa)) + triu(JtNa)';
    JtK   = (JtK - triu(JtK)  ) + triu(JtK)';
    JCl  = (JCl - triu(JCl)  ) + triu(JCl)';

    Nal = triu(Nal)';
    Kl = triu(Kl)';
    Cll = triu(Cll)';

    [qa,qb]=find(Qtot');
    qa = [qb,qa];

    JtNad = zeros(19,1);
    JtKd = zeros(19,1);
    JCld = zeros(19,1);
    Qtotd = zeros(19,1);
    Nald = zeros(19,1);
    Kld = zeros(19,1);
    Clld = zeros(19,1);

    for j = 1:19
        Nald(j,1) = Nal(qa(j,1),qa(j,2));
        Kld(j,1)  = Kl(qa(j,1),qa(j,2));
        Clld(j,1) = Cll(qa(j,1),qa(j,2));
        Qtotd(j,1) = Qtot(qa(j,1),qa(j,2));
        JtNad(j,1) = JtNa(qa(j,1),qa(j,2));
        JtKd(j,1)  = JtK(qa(j,1),qa(j,2));
        JCld(j,1)  = JCl(qa(j,1),qa(j,2));
    end

    % Multiply by adjacency matrix
    JtNa = par.Adjs.*JtNad;
    JtK = par.Adjs.*JtKd;
    JCl = par.Adjs.*JCld;
```

```
    Qtot = par.Adjs.*Qtotd;
    Nal = par.Adjs.*Nald;
    Kl = par.Adjs.*Kld;
    Cll = par.Adjs.*Clld;


    % Precalculate the water/ion influx:
    QwNa = Qtot.*Nal;
    QwNa = QwNa - diag(diag(QwNa));
    QwK = Qtot.*Kl;
    QwK = QwK - diag(diag(QwK));
    QwCl = Qtot.*Cll;
    QwCl = QwCl - diag(diag(QwCl));

end
```

This function is where I calculate things according to the connectivity matrix. Firstly, it begins by adding the upper triangle to the lower triangle (remember only the lower triangle is important which is the one for the variables.) Then, it arranges the variables (19 of them in a list to multiply (dot multiplication) by the adjacency matrix (i.e connectivty matrix)) for calculation of the equations of the luminal structures.

cluster.m

Heres the final cluster file: (or main file)

```
function dx = cluster(¬,x,Ca,par)
% ########################################################################
% ------------------------------------------------------------------------
% Author: Elias Siguenza
% Location: The University of Auckland, New Zealand
% Date: 22 March 2019
% Version: 4.1
% ------------------------------------------------------------------------
% Purpose:
% This function solves the differential equation system involved in the
% simulation of flow rate and ionic homeostasis in seven parotid acinar
% cells. The cells are connected according the connectivity matrix
% determined by the mesh created by John Rugis.
% This function requires an initial condition and a Calcium input.
% Such Ca input is given by Nathan Pages' spatial [Ca2+]_i model.
% ------------------------------------------------------------------------
% ########################################################################
%%%% Variables
% Sort out the variables (Arguably this could change, but I like order).
% The order is:
% 1. Volume
% 2. Na+
% 3. K+
% 4. Cl-
% 5. HCO3-
% 6. H+
% 7. Va
% 8. Vb
    [int,Nal,Kl,Cll] = var(x,par);
% ------------------------------------------------------------------------
%%%% Memory Preallocation
    Jb = zeros(7,9);JCl = zeros(7,7);JCL= zeros(7,1);JtNa = JCl;JtK = JCl;
    Qa=JCl; Qtot=JCl;a = 1;dx=zeros(length(x),1);
```

```
% ------------------------------------------------------------------------
%%%% Begin calculation of submodels.
    for c_no = 1:7
%%%% Basolateral fluxes and Membrane Potentials
% Sort out the basolateral fluxes:
% The order is: Qb, JNaK, JNkcc1, JAe4, JNhe1, JBB, JK, Ii, and Jwater.
        [Jb(c_no,1), Jb(c_no,2),...
            Jb(c_no,3), Jb(c_no,4),...
                      Jb(c_no,5), Jb(c_no,6),...
                                  Jb(c_no,7),Jb(c_no,8)] ...
                                                = fx_ba(int,Ca,par,c_no);
% ------------------------------------------------------------------------
%%%% Apical fluxes
% Sort out apical and tight junctiuonal fluxes:
        for j = 1:par.ind{c_no}
            ngh = par.neigh{c_no}(j);
            [JtNa(c_no,ngh), JtK(c_no,ngh), JCl(c_no,ngh),...
                                        Qa(c_no,ngh), Qtot(c_no,ngh)]...
                         = fx_ap(int,Nal,Kl,Cll,Jb(c_no,8),Ca,par,c_no,ngh);
        end
% ------------------------------------------------------------------------
%%%% Sort out the fluxes
% Sum all apical chloride fluxes:
        JCL(c_no,1) = sum(JCl(c_no,:));
        Jb(c_no,9) = Jb(c_no,1) - sum(Qa(c_no,:));
% ------------------------------------------------------------------------
%%%% Set the intracellular concentration differential equations
        dx(a:a+7,1) = ieq(JCL,JtNa,JtK,Jb,int,c_no);
        a=a+8;
    end
% ------------------------------------------------------------------------
%%%% Set the luminal concentration equations:
% Use adjacency matrix for connectivity of the lumen.
 [JtNad,JtKd,JCld,Qtotd,Nald,Kld,Clld,QwNa,QwK,QwCl] = ...
                              lum_adj(Nal,Kl,Cll,JtNa,JtK,JCl,Qtot,par);
% Initialise loop indices:
    ina = length(int)*7;
    ik = ina+19;
    icl = ik+19;

    for i = 1:19
        dx(ina+i,1) = JtNad(i,i) + sum(QwNa(:,i)) ...
                                 -  sum(Qtotd(:,i)) * Nald(i,i); % Sodium
        dx(ik+i,1)  = JtKd(i,i) + sum(QwK(:,i)) ...
                                -  sum(Qtotd(:,i)) * Kld(i,i);  % Potassium
        dx(icl+i,1) = - JCld(i,i) + sum(QwCl(:,i)) ...
                                  -  sum(Qtotd(:,i)) * Clld(i,i);% Chloride
    end
% ------------------------------------------------------------------------
%%%%
% End of calculations.
end
```

Calcium - FFR model interaction

So in order to merge both models, Nathan and I came up with a pretty cool scheme in which we calculate one step in time for the FFR model using the Ca2+ initial conditions. This gives us a value for each of the cells volume and this is fed into the Ca2+ model (as volume affects the concentration of Calcium).

The process is repeeated, this time I use the new Ca value as an input for the FFR which gives us a new volume model for the Ca2+ model and so on...

So nathan's model has an ordered list of Ca values (these are values at each node in the cluster). But, the FFR model requires the average surface triangle calcium for apical and basolateral a-like. So, to do so he created a matrix called: `mtriangles`. This is a sparse matrix that acts as an operator upon the vector/matrix of nodal Ca values and calculates the average Ca point per triangle. `mtriangles` is defined as follows:

```
% Matrix from p to triangles
m_triangles = cell(n_cells,1);
for i = 1:n_cells
    ind_x = repmat((1:n_triangles(i))',1,3);
    ind_y = triangles{i};
    ind_x = ind_x(:);
    ind_y = ind_y(:);
    val = (1/3)*ones(3*n_triangles(i),1);
    m_triangles{i} = sparse(ind_x,ind_y,val,n_triangles(i),np(i));
end
```

We use that matrix(operator) to convert all the calcium points into the average of each triangle. Then I use the index of each triangle (given by John's mesh) in a cell array I called `par.sb_tri`, to order the values I need to take for the basolateral region and call it `ca_b`. Additionally I also separate the apical ones using the common triangles of apical region, denoted as `par.com_tri_ap`. This is arranged just as the matrices up there; that is, a 7x7 matrix where the row denotes cell of interest and column its neighbour:

```
 for j = 1:n_cells
            c=x_tilde{j}(1:np(j));
            cav_tri  = m_triangles{j}(:,1:size(x_h{j},1))*c(1:size(x_h{j},1));
            ca_b{j} = cav_tri(par.sb_tri{j});
            for ii = 1:n_cells
                if ¬isempty(par.sa_tri{j,ii})
                    ca_a{j,ii}=cav_tri(par.com_tri_ap{j,ii}(:,1));
                else
                    ca_a{j,ii}=[];
                end
            end
        end
        Ca = {ca_a,ca_b};
```

Finally, I just merge everything into a cell array called Ca which I use as an input for the secretion file.

How to run the PDE in MATLAB

Firstly you need to go to the file called `fun_param.m`, run it as it is and then you'll get a string. This string will be stored as a variable in MATLAB called str. Now in the editor just type: `solver(str,0)`. This just tells the function which parameter file we are runing and that 0 is the folder, this parameter file is located.