M1 - M-ENG
Advanced Manufacturing

ALEMO Project - Bone Remodelling

Sudarson Nanthacoumarane

Objective - To create a solver for the Komarova's model and use it to simulate random boneremodeling.

1. Complete KomarovaModel.m

The governing equations for Komarova's Model are:

$$\dot{y}_1 = a_1 y_1^{g_{11}} y_2^{g_{21}} - b_1 y_1$$

$$\dot{y}_2 = a_2 y_1^{g_{12}} y_2^{g_{22}} - b_2 y_2$$

This has been coded into *KomarovaModel.m* as:

```
y1 = y(1);      # Stores y1
y2 = y(2);      # Stores y2

ydot1 = a1 * y1.^g11 * y2.^g21 - b1*y1;
ydot2 = a2 * y1.^g12 * y2.^g22 - b2*y2;
```

We take these *ydot1* and *ydot2* values and store them in a functional **f(y)** ,where:

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \quad \text{and} \quad f(y) = \begin{bmatrix} a_1 y_1^{g_{11}} y_2^{g_{21}} - b_1 y_1 \\ a_2 y_1^{g_{12}} y_2^{g_{22}} - b_2 y_2 \end{bmatrix} \quad \Leftrightarrow \quad \dot{y} = f(y)$$

2. Complete KomarovaModel_Jac.m

The governing equations for the Jacobian in Komarova Model are:

$$J(y) = \frac{\partial f}{\partial y} = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} \end{bmatrix}$$

Solving these partial differential equations, we get:

$$J_{11} = a_1 g_{11} y_1^{(g_{11}-1)} y_2^{g_{21}} - b_1 \qquad J_{12} = a_1 g_{21} y_1^{g_{11}} y_2^{(g_{21}-1)}$$

$$J_{21} = a_2 g_{12} y_1^{(g_{12}-1)} y_2^{g_{22}} \qquad J_{22} = a_2 g_{22} y_1^{g_{12}} y_2^{(g_{22}-1)} - b_2$$

These values are stored in *KomarovaModel_Jac.m* as:

```
y1 = y(1);      # Stores y1
y2 = y(2);      # Stores y2

J11 = g11 * a1 * y1.^(g11-1) * y2.^g21 - b1;
J12 = g21 * a1 * y1.^g11     * y2.^(g21-1);
J21 = g12 * a2 * y1.^(g12-1) * y2.^g22;
J22 = g22 * a2 * y1.^g12     * y2.^(g22-1) - b2;
```

3. Write Backward Euler's formula for Komarova's equations

Backward Euler scheme is a method of finite difference approximation that allows us to find the slope of a curve at a point by using a point that is present behind the current point. It is generally written as:

$$y_i' = \frac{y_i - y_{i-1}}{\Delta t_{i-1}}$$

Where y'_i is the slope at point i , y_i is the value of the function y at point i , y_{i-1} is the value of the function y at a point $i-1$, and $\Delta\, t_i$ is the time step. In our equations, since we know that slope is **f(y)** , we can write:

$$f(y_i) = \frac{y_i - y_{i-1}}{\Delta t_{i-1}} \quad \Leftrightarrow \quad f(y_{i+1}) = \frac{y_{i+1} - y_i}{\Delta t_i}$$

If we seperate **y** into y_1 and y_2 values, we can write their scalar forms as:

$$f(y_{1,i+1}) \;=\; a_1 y_{1,i+1}^{g_{11}} y_{2,i+1}^{g_{21}} - b_1 y_{1,i+1} \;=\; \frac{y_{1,i+1} - y_{1,i}}{\Delta t_i}$$

$$\Rightarrow \quad y_{1,i+1} - y_{1,i} - \Delta t_i (a_1 y_{1,i+1}^{g_{11}} y_{2,i+1}^{g_{21}} - b_1 y_{1,i+1}) \;=\; 0$$

$$f(y_{2,i+1}) \;=\; a_2 y_{1,i+1}^{g_{12}} y_{2,i+1}^{g_{22}} - b_2 y_{2,i+1} \;=\; \frac{y_{2,i+1} - y_{2,i}}{\Delta t_i}$$

$$\Rightarrow \quad y_{2,i+1} - y_{2,i} - \Delta t_i (a_2 y_{1,i+1}^{g_{12}} y_{2,i+1}^{g_{22}} - b_2 y_{2,i+1}) \;=\; 0$$

4. Prove that $\mathbf{y_{i+i}}$ is the root of g(z) = 0

The scalar forms of Backward Euler formula for Komarova's equation can be rewritten as a vector values function. Here we introduce a new variable z which is equal to $[z_1 ; z_2]$, with $z_1 = y_{1,i+i}$ and $z_2 = y_{2,i+i}$.

$$g(z) = z - y_i - \Delta t_i\; f(z) \qquad where\;\; z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} y_{1,i+1} \\ y_{2,i+1} \end{bmatrix}$$

The root of an equation is that value which when substituted in the equation, makes the equation true. In this case, we need to find the value of z which is true for the function $g(z) = 0$. In order to prove that y_{i+1} is the root of $g(z)$ we substitute y_{i+1} for the value of z in the function.

Recall that,

$$f(y_{i+1}) = \frac{y_{i+1} - y_i}{\Delta t_i}$$

$$\therefore \quad g(z) = g(y_{i+1}) = y_{i+1} - y_i - \Delta t_i f(y_{i+1}) \quad \Rightarrow \quad g(y_{i+1}) = y_{i+1} - y_i - \Delta t_i \left(\frac{y_{i+1} - y_i}{\Delta t_i} \right)$$

$$\Rightarrow \quad g(y_{i+1}) = y_{i+1} - y_{i+1} + y_i - y_i = 0$$

Since the value of *$g(z) = 0$* when *$z = y_{i+1}$* it has been proved that *y_{i+1}* is the root of *$g(z)$*.

5. Derive a mathematical expression for gradient of g(z)

The gradient of a function is the sum of of its partial derivatives. Since **g(z)** is a vector function, we require two sets of partial derivatives to describe it. Spliting up **g(z)** into its scalar components and differentiating, we obtain:

$$g(y_{1,i+1}) = y_{1,i+1} - y_{1,i} - \Delta t_i f(y_{1,i+1}) \quad \Rightarrow \quad g(y_{1,i+1}) = y_{1,i+1} - y_{1,i} - \Delta t_i (a_1 y_{1,i+1}^{g_{11}} y_{2,i+1}^{g_{21}} - b_1 y_{1,i+1})$$

$$g(y_{2,i+1}) = y_{2,i+1} - y_{2,i} - \Delta t_i f(y_{2,i+1}) \quad \Rightarrow \quad g(y_{2,i+1}) = y_{2,i+1} - y_{2,i} - \Delta t_i (a_2 y_{1,i+1}^{g_{12}} y_{2,i+1}^{g_{22}} - b_2 y_{2,i+1})$$

Taking the partial derivatives of these two terms with respect to $y_{1,i+1}$ and $y_{2,i+1}$ the gradient of **g(z)** can be split up into the following parts:

$$\frac{\partial g(y_{1,i+1})}{\partial y_{1,i+1}} = 1 - \Delta t_i(a_1 g_{11} y_{1,i+1}^{(g_{11}-1)} y_{2,i+1}^{g_{21}} - b_1 \qquad \frac{\partial g(y_{1,i+1})}{\partial y_{2,i+1}} = -\Delta t_i(a_1 g_{21} y_{1,i+1}^{g_{11}} y_{2,i+1}^{(g_{21}-1)})$$

$$\frac{\partial g(y_{2,i+1})}{\partial y_{1,i+1}} = -\Delta t_i(a_1 g_{12} y_{1,i+1}^{(g_{12}-1)} y_{2,i+1}^{g_{21}} \qquad \frac{\partial g(y_{2,i+1})}{\partial y_{2,i+1}} = 1 - \Delta t_i(a_2 g_{22} y_{1,i+1}^{g_{12}} y_{2,i+1}^{(g_{22}-1)} - b_2$$

Therefore, the gradient of **g(z)** can be written as:

$$L(z) = \frac{\partial g}{\partial z} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \Delta t_i \begin{bmatrix} a_1 g_{11} y_{1,i+1}^{(g_{11}-1)} y_{2,i+1}^{g_{21}} - b_1 & a_1 g_{21} y_{1,i+1}^{g_{11}} y_{2,i+1}^{(g_{21}-1)} \\ a_1 g_{12} y_{1,i+1}^{(g_{12}-1)} y_{2,i+1}^{g_{21}} & a_2 g_{22} y_{1,i+1}^{g_{12}} y_{2,i+1}^{(g_{22}-1)} - b_2 \end{bmatrix}$$

We know that Jacobian **J(z)** is:

$$J(z) = J(y_{i+1}) = \begin{bmatrix} \dfrac{\partial f_1}{\partial y_{1,i+1}} & \dfrac{\partial f_2}{\partial y_{1,i+1}} \\ \dfrac{\partial f_1}{\partial y_{2,i+1}} & \dfrac{\partial f_2}{\partial y_{2,i+1}} \end{bmatrix} = \begin{bmatrix} a_1 g_{11} y_{1,i+1}^{(g_{11}-1)} y_{2,i+1}^{g_{21}} - b_1 & a_1 g_{21} y_{1,i+1}^{g_{11}} y_{2,i+1}^{(g_{21}-1)} \\ a_1 g_{12} y_{1,i+1}^{(g_{12}-1)} y_{2,i+1}^{g_{21}} & a_2 g_{22} y_{1,i+1}^{g_{12}} y_{2,i+1}^{(g_{22}-1)} - b_2 \end{bmatrix}$$

$$\therefore\ L(z) = \frac{\partial g}{\partial z} = I - \Delta t_i J(z)$$

6. Complete BwdEuler.m

Backward Euler method is a numerical method for the solution of differential equations. It is an implicit method, i.e. a solution is found by solving an equation involving both the current state of the system and the later one. We find the solution of two functions, **g(z)** and **L(z)** at each time step *dt*. Here, y_{i+1} indexing was used instead of y_i indexing to make the code similar to teaching slides and the equations defined above.

```
for i = 1:numel(t)-1

    % compute Delta t
    dt = t(i+1) - t(i);

    % set g fun
    gfun = @(z) (z - y(:,i) - dt*ffun(t(i+1),z));

    % Set L
    Lfun = @(z) (eye(2) - dt*Jfun(t(i+1),z));

    % solve nonlinear problem using the solution at previous time step as
    % initial guess
    y(:,i+1) = solveNR(gfun,Lfun,y(:,i));

    %Display message
    fprintf('Solved time step %d of %d0,i,numel(time))

 endfor
```

7. Complete solveNR.m

Since we have proved above that the root of **g(z)** = 0 is y_{i+1}, we can find its value using Newton-Rhapson Method.

```
 function znew = solveNR(gfun,Lfun,z1)

 err = 1;
 k = 0;
 znew = z1;

 while (err>1e-10)&&(k<=10000)

     %STORE the old solution
     zold = znew;

     %Increment iteration index k
     k = k+1;

     %COMPUTE THE FUNCTION
     g = gfun(zold);

     %COMPUTE THE GRADIENT
     L = Lfun(zold);

     %UPDATE Z
     znew = zold - inv(L) * g;

     %ERROR ESTIMATION
     err = max(norm(g),norm(zold-znew));
     end

     if err>1e-10
         error('Newton Raphson did not converge! Try increasing the error
tolerance or the number of iterations!')
     end

 end
```

8. Test code using testKomarova.m

Since many calculations had to be repeated over and over again in the course of doing this project, all values of *y* and its corresponding *days, time, and Nt* were saved as a matlab ".mat file". If the values of *y* needed to be calculated using Backward Euler method, a function *dataExist()* was written to check if those values had already been computed. If the values existed, they were loaded into the script to be used. Otherwise, the cell count was computed using the Backward Euler method.

```
 function [y, time] = dataExist(days,Nt,ffun,Jfun,y0)

     fileName = strcat("vectorData/data_D",
                num2str(days),"_N",num2str(Nt),".mat");

     if exist(fileName)
         printf("%s exists. Loading data...0,fileName);
         cmd = sprintf("load %s",fileName);
         eval(cmd);
         y;
         time;
     else
         printf("%s does not exist. Calculating and saving data...0,file-
Name);
         time = linspace(0,days,Nt+1);
         y = BwdEuler(time,ffun,Jfun,y0);
         str = sprintf("save %s y time Nt days",fileName);
         eval(str);
     endif

 endfunction
```

The results for Nt = 1000, 10000, 50000, 100000 are plotted below:

9. Plot solutions on the same graph while increasing the time step

Comparisons were made using time steps for *Nt = 1000, 4000, 9000, 16000, and 25000* with the reference of 0.5 million time steps. The reference plot in both the osteoclast figure and the osteoblast figure are represented by a dash-dotted line. **As the number of time steps increases, the solutions start to get plotted closer and closer to each other.** This means that the solution starts to **converge** as the number of time steps increases, and this can be seen clearly because the solution for the highest number of time steps in these plots *(Nt = 25000)* is very close to the exact/reference solution.

10. Plot error against reference solution

10.1. Interpolating approximate solution and compute the error

The inbuilt function *interp1()* was used to find the interpolation of approximate solution with the exact solution. A user-defined function called *getInterpError()* returned the interpolation error when provided with data for the approximate time step. Error for Osteoclast and Osteoblast count was caclulated as the difference of the norm of the reference values and the interpolated values.

10.2. Increase number of points in time stepping vector and plot the error

In order to plot the evolution of error with increase in number of time steps, a doubling time step was chosen where *Nt = 2000, 4000, 8000, 16000, 32000, 64000, 128000, 256000.* This computation was done in a user-defined function *convergenceStudy()* which called the previously mentioned *getInterpError()* function.

```
 function convergenceStudy(days,ffun,Jfun,y0)
     a = 1000;
     for i=1:20
         N(i) = a;
         if a > 200000
             break;
         endif
         a = a * 2;
     endfor

     for i = 1:length(N)
         Nt = N(i);
         [y, time] = dataExist(days,Nt,ffun,Jfun,y0);
         [e1(1,i),e2(1,i)] = getInterpError(y,days,time,Nt);
     endfor

     loglog(N,e1,'-o','MarkerSize',10,'LineWidth',1.5);
     hold on
     loglog(N,e2,'-o','MarkerSize',10,'LineWidth',1.5);
     xlabel("Time Discretization Nt");
     ylabel("Norm (ref - approx)");
     legend("Error 1", "Error 2");

endfunction
```

```
 function [err1, err2] = getInterpError(apx, d, t, N)
     fileName = "vectorData/data_D500_N500000.mat";
     cmd = sprintf("load %s",fileName);
     eval(cmd);

     if days != d
         error('Number of days is not the same')
     endif

     ref = y;

    r1 = ref(1,:);  # Reference 0.5M value y1 for 'time'
     r2 = ref(2,:);  # Reference 0.5M value y2 for 'time'

    a1 = apx(1,:);  # Approx Nt value y1 for 't'
     a2 = apx(2,:);  # Approx Nt value y2 for 't'

     b1 = interp1(t,a1,time,'spline');   # Interpolation y1
     b2 = interp1(t,a2,time,'spline');   # Interpolation y2

     err1 = norm(r1-b1)  # Find error y1
     err2 = norm(r2-b2)  # Find error y2

 endfunction
```

11. Compute bone mass density and plot M(t) vs Time

For a time period of 500 days, the error between approximate solution and reference solution is suitably small for *Nt = 100000.* This is demonstrated with the following graph, where the approximate solution and reference solution are nearly indistinguishable.

The bone mass density *M(t)* was calculated by integrating the following equation using the trapezoidal rule:

$$M(t) = \int_0^t k_1 x_1(\tau) + k_2 x_2(\tau) \;\; d\tau$$

$$x_i(\tau) = max(y_i - y_{ss}, 0) \quad i = 1, 2$$

```
function M = boneMass(y,yss,t,days,Nt,k1,k2,BMi)

    x1 = max(y(1,:)-yss(1,:),0);
    x2 = max(y(2,:)-yss(2,:),0);

    for i = 2:Nt+1
        dt = t(i)-t(i-1);
        m = k1*(x1(i)+x1(i-1)) + k2*(x2(i)+x2(i-1));
        M(i) = 0.5 * dt*m;
    endfor

    M0 = BMi;
    M = (M0-M)/M0 * 100;

    plot(t,M);
    xlabel("Time [days]");
    ylabel("Bone Mass %");
    grid minor;
    set(gca, 'LineWidth', 1.5, 'FontSize', 14);
    titleName = strcat("Bone Mass Density for",{" "},num2str(Nt)," time
steps");
    title(titleName);

    imageName                        =                        strcat("im-
ages/IMG_BMi_D",num2str(days),"_N",num2str(Nt),".eps");
    str = sprintf("print -depsc %s", imageName);
    eval(str);

endfunction
```

12. Appendix

12.1. dataExist()