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:

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

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:

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

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:

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

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

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

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

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

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

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

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

4. Prove that 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 $z_2 = y_{2,i+i}$.

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

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

y_{i+1} indexing was used instead of y_i indexing to make the code similar to teaching slides and 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),z));

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

```
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
```