Mathematical Modelling in the Natural Sciences SS21
Solutions to Exercises on Sheet 7

Exercises und Lecture Notes

Contents

- **Exercise 1: Cerclage Model** **1**
  - Task . . . 1
  - Solution . . . 2
- **Exercise 2: Bungee Cord** **10**
  - Task . . . 10
  - Solution . . . 10
- **Exercise 3: Pastry Sheet** **15**
  - Task . . . 15
  - Solution . . . 15
- **Exercise 4: Meditation Model** **23**
  - Task . . . 23
  - Solution . . . 23
- **Exercise 5: Comparison of Membrane Models** **37**
  - Task . . . 37
  - Solution . . . 37

• Exercise 1: Cerclage Model

◦ Task

Derive the necessary stationarity condition on page 190 of the lecture notes,

$$-\left\{T(u)\frac{uu_\phi}{\sqrt{u^2+u_\phi^2}}\sin\phi\right\}_\phi + T(u)\frac{2u^2+u_\phi^2}{\sqrt{u^2+u_\phi^2}}\sin\phi$$

$$=\left\{f(u\sin\phi,u\cos\phi)\frac{(u\cos\phi)_\phi}{\sqrt{u^2+u_\phi^2}}+(p_1+\lambda)\right\}u^2\sin\phi,\quad 0<\phi<\pi,$$

$$u_\phi=0,\quad \phi=0,\pi,\qquad \int_0^\pi u^3\sin\phi d\phi = 2r_1^3$$

Write a Matlab code to solve this system as indicated on pages 191 - 192, i.e., by iterating a cell centered finite difference approximation of

$$\begin{bmatrix} A(u) & K(u) \\ K^*(u) & 0 \end{bmatrix}\begin{bmatrix} v \\ \lambda \end{bmatrix} = \begin{bmatrix} F(u,p) \\ G(u) \end{bmatrix} \qquad \begin{aligned} u &= u+\alpha v \\ p &= p+\alpha\lambda \end{aligned}$$

where

$$A(u)v = -\left\{ \frac{T(u)u\sin\phi}{\sqrt{u^2+u_\phi^2}} v_\phi \right\}_\phi + \frac{T(u)u\sin\phi}{\sqrt{u^2+u_\phi^2}} v \;\approx -\frac{\delta F}{\delta u}(u;v)$$

$$K(u)\lambda = -\lambda u^2 \sin\phi \;\approx -\frac{\delta F}{\delta p}(p;\lambda), \quad K^*(u)v = -\int_0^\pi v u^2 \sin\phi d\phi \;\approx -\frac{\delta G}{\delta u}(u;v)$$

$$F(u,p) = -A(u)u - T(u)\sin\phi\sqrt{u^2+u_\phi^2} + \left[f(u\sin\phi, u\cos\phi)\frac{(u\cos\phi)_\phi}{\sqrt{u^2+u_\phi^2}} + p \right] u^2\sin\phi$$

$$G(u) = \int_0^\pi u^3 \sin\phi d\phi - 2r_1^3$$

◦ Solution

In the lecture notes the model of the band force $f(u\sin(\phi), u\cos(\phi))$ is given on page 187 and the model of tension $T(u)$ is given on pages 188-189. The Lagrangian functional on page 186,

$$L(u,\lambda) = \tfrac{1}{2\pi}\left[J_{\rm i}(u) + J_{\rm e}(u) - \lambda J_{\rm c}(u)\right]$$

has the variational derivatives seen on page 184,

$$\frac{\delta J_{\rm i}}{\delta u}(u;v) = 2\pi \int_0^\pi T(u)\frac{u_\phi^2 v + 2u^2 v + u u_\phi v_\phi}{\sqrt{u^2+u_\phi^2}}\sin(\phi)d\phi$$

and on page 186

$$\frac{\delta J_{\rm e}}{\delta u}(u;v) = -2\pi \int_0^\pi \left\{ f(u\cos(\phi), u\sin(\phi))\frac{(u\cos(\phi))_\phi}{\sqrt{u^2+u_\phi^2}} + p \right\} v u^2 \sin(\phi)d\phi$$

and the variational derivative of the contraint functional $J_{\rm c}$ on page 186 is given by

$$\frac{\delta J_{\rm c}}{\delta u}(u;v) = 2\pi \int_0^\pi u^2 v \sin(\phi)d\phi.$$

Setting $\delta L(u;v)/\delta u = 0$ gives the condition

$$\int_0^\pi T(u)\frac{u_\phi^2 v + 2u^2 v + u u_\phi v_\phi}{\sqrt{u^2+u_\phi^2}}\sin(\phi)d\phi - \int_0^\pi \left\{ f(u\cos(\phi), u\sin(\phi))\frac{(u\cos(\phi))_\phi}{\sqrt{u^2+u_\phi^2}} + p \right\} v u^2 \sin(\phi)d\phi - \lambda 2\pi \int_0^\pi u^2 v \sin(\phi)d\phi = 0, \quad \forall v \in \mathcal{C}([0,\pi])$$

The only term among these which contains a derivative of a test function v is

$$\int_0^\pi T(u)\frac{u u_\phi v_\phi}{\sqrt{u^2+u_\phi^2}}\sin(\phi)d\phi = -\int_0^\pi \left\{ T(u)\frac{\sin(\phi)u u_\phi}{\sqrt{u^2+u_\phi^2}} \right\}_\phi v d\phi + \left\{ T(u)\frac{\sin(\phi)u u_\phi}{\sqrt{u^2+u_\phi^2}} \right\}_\phi v \Big|_0^\pi$$

Setting the test function v to be concentrated in the iterior of the interval $(0, \pi)$ gives the necessary stationarity condition on u,

$$-\left\{ T(u)\frac{uu_\phi}{\sqrt{u^2+u_\phi^2}}\sin\phi \right\}_\phi + T(u)\frac{2u^2+u_\phi^2}{\sqrt{u^2+u_\phi^2}}\sin\phi$$

$$= \left\{ f(u\sin\phi, u\cos\phi)\frac{(u\cos\phi)_\phi}{\sqrt{u^2+u_\phi^2}} + (p_1+\lambda) \right\} u^2\sin\phi, \quad 0<\phi<\pi,$$

$$, \qquad \int_0^\pi u^3\sin\phi d\phi = 2r_1^3$$

Letting the test function v be concentrated on the boundary of the interval gives the Neumann boundary condition,

$$u_\phi = 0, \quad \phi = 0, \pi$$

Finally setting $\delta L(\lambda;\mu)/\delta\lambda = 0$ gives the condition

$$\int_0^\pi u^{\star 3}\sin(\phi)d\phi = 2r_1^3.$$

The saddle point problem is solved using the following Matlab code, in which various parameters appear as indicated in the lecture notes.

```
    h1 = figure(1);
    close(h1);
    h1 = figure(1);
    set(h1,'Position',[0   0 384 384])
    h2 = figure(2);
    close(h2);
    h2 = figure(2);
    set(h2,'Position',[400 0 384 384])
    h3 = figure(3);
    close(h3);
    h3 = figure(3);
    set(h3,'Position',[800 0 384 384])

        n = 101;                % number of phi subintervals
        h = pi/n;               % width of subinterval
  phib = h*(0:n)';              % subinterval boundaries
  phic = phib(1:n) + 0.5*h;     % subinterval centroids

    r1 = 12.25;                 % normal radius is 12 mm
%   p1 = 23.0;                  % normal pressure is 20 mmHg
    p1 = 20.0;                  % normal pressure is 20 mmHg
    T1 = 0.5*p1*r1;             % Laplace Law
    V1 = 4.0*pi*r1^3/3.0;       % initial volume
  Ed_m = 16232.0;               % elasticity modulus * membrane thickness
```

```
%     Ed_m = 21753.0;            % elasticity modulus * membrane thickness
        rg = 1.0/80.0;           % occular rigidity

        p0 = p1;
        V0 = V1 + log(p0/p1)/rg;
        r0 = (0.75*V0/pi)^(1/3);
        p1 = p0;
        V1 = V0;
        r1 = r0;
        T1 = 0.5*p1*r1;

%     Ed_b = 14852.0*0.75;       % elasticity modulus * band thickness
%     Ed_b = 19578.0*0.75;       % elasticity modulus * band thickness
      Ed_b = 24453.0*0.75;       % elasticity modulus * band thickness
      Rhat = 10.35;              % rest radius of band

        bt = 1.0;                % half the band width
      phi1 = acos(bt/r1);        % first angle of band edge
      phi2 = pi - phi1;          % second angle of band edge
        z1 = r1*cos(phi1);       % plane of lower band edge
        z2 = r1*cos(phi2);       % plane of upper band edge
        R1 = r1*sin(phi1);       % height of lower band edge
        R2 = r1*sin(phi2);       % height of upper band edge
        ax = 15;                 % x-axis radius
        ay = 15;                 % y-axis radius

       mxp = 100000;             % max iterations to reach original pressure
       mxv = 100000;             % max iterations to solve for geometry
      tolp = 1.0e-6;             % pressure iteration tolerance
      tolv = 1.0e-6;             % geometry interation tolerance
        ep = 1.0e-15;            % code zero
        rx = 0.9;                % geometry relaxation parameter 0 < rx =< 1

        r0 = r1;                 % set the first equilibrium state
        V0 = V1;
        p0 = p1;
        T0 = T1;

        Vs = V1;                 % initialize state histories
        ps = p1;
        Ts = T1;
                                 % set min pressure greater than p1
        pm = 2.0*p1;

    for itp=1:mxp
                                 % initialize residual histories
        er = 1.0;
        es = 1.0;
```

```
                        % set equilibrium state without band
 v = r0*ones(n,1);
 V = V0;
 p = p0;
 T = T0*ones(n,1);

rz = rx;

for itv=1:mxv

    if (mod(itv,100) == 0)
        rz = 0.5*rz;
    end
                        % build band force:
    vc = v.*cos(phic);
    vs = v.*sin(phic);
                        % build v' at cell centers
    vp = (v(3:end)-v(1:end-2))/(2.0*h);
    vp = [2.0*vp(1)-vp(2),vp',2.0*vp(end)-vp(end-1)]';
    sq = sqrt(v.^2 + vp.^2);
                        % band force
     f = (vp.*cos(phic)-v.*sin(phic))./sq;
     f = Ed_b*f.*(1.0/Rhat - 1.0./vs).* (vs > Rhat)...
      .* (z2 <= vc) .* (vc <= z1);
                        % coefficient for 0th order term in PDE
    a0 = T.*vs./sq;
                        % coefficient for 1st order term in PDE
    a1 = (a0(2:n)+a0(1:(n-1)))/(2.0*h^2);
                        % build discretized differential operator
     A = diag(a0) + diag([a1;0]) + diag([0;a1])...
                  - diag(a1,-1)  - diag(a1,1);
                        % multiplier coefficient
     k = -3.0*v.*vs;
                        % build discretized SQP operator
     B = [[A,k];[k',0]];
                        % build rhs for SQP equation
     b = (f + p).*v.*vs - A*v - T.*sin(phic).*sq;
     e = sum((v.^3).*sin(phic)) - 0.75*V/(pi*sin(h/2));
     F = [b;e];
                        % solve SQP system
     u = B\F;
lambda = u(n+1);
     u = u(1:n);
                        % update v and p
    vo = v;
     v = vo + rz*u;
    po = p;
     p = po + rz*lambda;
```

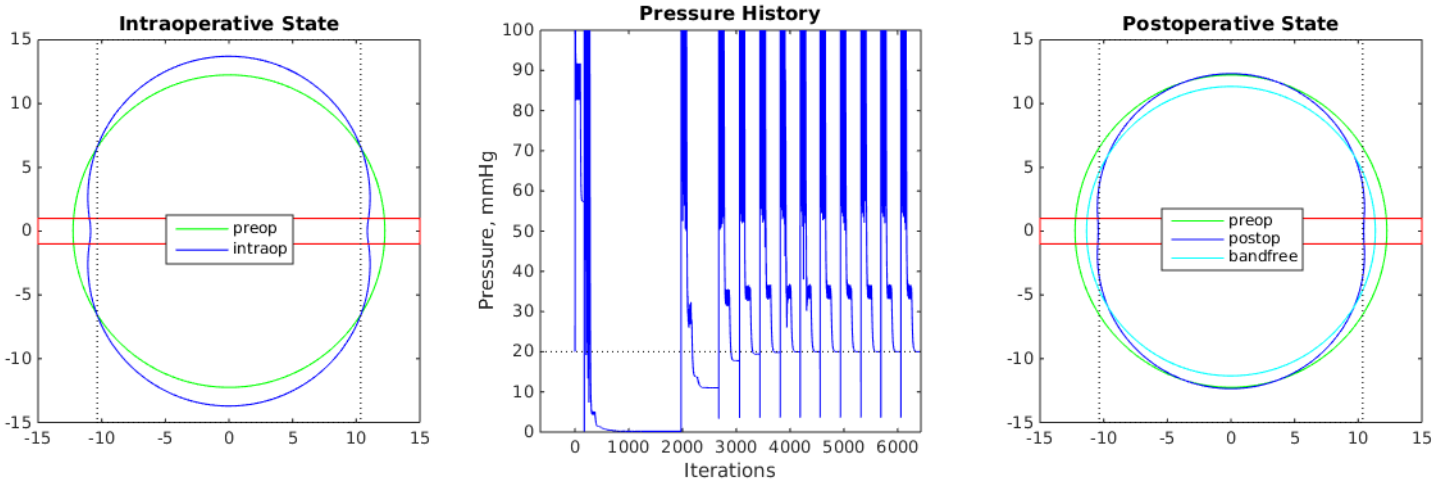
```
     p = max([p,ep]);
     v = 0.5*(v + v(end:-1:1));
     v = min(v,1.5*r0*ones(n,1));
     v = max(v,Rhat*ones(n,1));
     u = v-vo;
lambda = p-po;
                          % update tension
    vp = (v(3:end)-v(1:end-2))/(2.0*h);
    vp = [2.0*vp(1)-vp(2),vp',2.0*vp(end)-vp(end-1)]';
    vs = v.*sin(phic);
     T = T0 + Ed_m*(sum(sqrt(v.^2+vp.^2).*vs)/sum(r0^2*sin(phic))-1);
                          % record histories
    ps = [ps,p];
                          % plot pressure history
    if (mod(itv,mxv) == 0)
        figure(2);
        plot(1:length(ps),ps,'b-');
        axis([(-0.1*length(ps)) length(ps) 0 100]);
        hold on
        plot([-0.1*length(ps),length(ps)],[20,20],'k:');
        hold off
        xlabel('Iterations');
        ylabel('Pressure, mmHg');
        title('Pressure History');
    end
                          % plot new geometry:
                          % points on spherical eye
    x0 = r0*cos(phic);
    x0 = [x0;x0(n:-1:1);x0(1)];
    y0 = r0*sin(phic);
    y0 = [y0;-y0(n:-1:1);y0(1)];
                          % points on eye before operation
    x1 = r1*cos(phic);
    x1 = [x1;x1(n:-1:1);x1(1)];
    y1 = r1*sin(phic);
    y1 = [y1;-y1(n:-1:1);y1(1)];
                          % points on eye after the operation
    x2 = v.*cos(phic);
    x2 = [x2;x2(n:-1:1);x2(1)];
    y2 = v.*sin(phic);
    y2 = [y2;-y2(n:-1:1);y2(1)];
                          % points on eye under the band
    vc = v.*cos(phic);
    fi = find((phi1 <= phic) & (phic <= phi2));
    x3 = v(fi).*cos(phic(fi));
    y3 = v(fi).*sin(phic(fi));
    x4 = x3(end:-1:1);
    y4 = -y3(end:-1:1);
```

```
if (mod(itv,1000000) == 0)
                    % plot 360 degrees
    figure(3);
    plot(y1,x1,'g',y2,x2,'b');
    axis([-ax ax -ay ay]);
    pbaspect([1 1 1]);
    legend('preop','intraop',0,    ...
           'Location',[0.47 0.45 0.1 0.1]);
    hold on;
    plot([-ax  ax  ax -ax -ax], ...
         [z2 z2 z1 z1 z2],'r')
    plot([-Rhat  Rhat  Rhat -Rhat -Rhat], ...
         [-ay -ay ay ay -ay],'k:')
    title('Postoperative State');
    hold off;
    drawnow;
end
                    % break with small residual errors
if ((max(abs(u)) < tolv*max(abs(v)))...
&&  (abs(lambda) < tolv*p))

    figure(3);
    plot(y1,x1,'g', ...
         y2,x2,'b', ...
         y0,x0,'c');
    axis([-ax ax -ay ay]);
    pbaspect([1 1 1]);
    legend('preop','postop','bandfree',0, ...
           'Location',[0.47 0.45 0.1 0.1]);
    hold on;
    plot([-ax  ax  ax -ax -ax], ...
         [z2 z2 z1 z1 z2],'r');
    plot([-Rhat  Rhat  Rhat -Rhat -Rhat], ...
         [-ay -ay ay ay -ay],'k:')
    title('Postoperative State');
    hold off;
    drawnow;

    figure(2);
    plot(1:length(ps),ps,'b-');
    axis([(-0.1*length(ps)) length(ps) 0 100])
    hold on
    plot([-0.1*length(ps),length(ps)],[20,20],'k:');
    hold off
    xlabel('Iterations');
    ylabel('Pressure, mmHg');
    title('Pressure History');
```

```
        break;
    end

end

if (itv == mxv)
    error('max inner iterations reached');
end

if (itp == 1)
    disp('Intraoperative State:');
    disp(strcat('Equation residual=',num2str(er(end))));
    disp(strcat('Pressure residual=',num2str(abs(lambda))));
    disp(strcat('Volume residual=',num2str(es(end))));
    disp(strcat('Pressure comparison: p1=',num2str(p1),' p=',num2str(p)));
    disp(strcat('Tension comparison: T1=',num2str(T1),' T=',num2str(T)));
    disp(strcat('Geometry comparison: r1=',num2str(r1),' r=',num2str(r1)));
    disp(strcat('min/max v=',num2str(min(v)),',',num2str(max(v))));
    disp(strcat('Volume comparison: V1=',num2str(V1),' V=',num2str(V1)));
                        % plot 360 degrees
    figure(1);
    plot(y1,x1,'g',y2,x2,'b');
    axis([-ax ax -ay ay]);
    pbaspect([1 1 1]);
    legend('preop','intraop',0, ...
        'Location',[0.47 0.45 0.1 0.1]);
    hold on;
    plot([-ax  ax  ax -ax -ax], ...
         [z2 z2 z1 z1 z2],'r')
    plot([-Rhat  Rhat  Rhat -Rhat -Rhat], ...
         [-ay -ay ay ay -ay],'k:')
    title('Intraoperative State');
    hold off;
    drawnow;
end

pm = min([pm,p]);        % update min pressure
if (pm >= p1)
    pb = p1;
    rb = r1;
    fb = p;
    ra = Rhat;
    Va = 4.0*pi*ra^3/3.0;
    pa = p1*exp(rg*(Va-V1));
    if (pa < ep)
        pa = ep;
        Va = V1 + log(pa/p1)/rg;
```

```
            ra = (0.75*Va/pi)^(1/3);
        end
        fa = 0;
                            % update equilibrium state without band
        p0 = pa;
        V0 = Va;
        r0 = ra;
        T0 = 0.5*p0*r0;
    else                    % p0=pa :=> p<p1, p0=pb :=> p>p1
        if (p > p1)
            rb = r0;        % if p>p1 move rb inward to r0
            fb = p;
        else
            ra = r0;        % if p<=p1 move ra inward to r0
            fa = p;
        end                 % secant/bisect, r0 = weighted ave(ra,rb)
        r0 = ((fb-p1)*ra + (p1-fa)*rb)/(fb-fa);
        if ((r0 < ra) || (r0 > rb))
            r0 = 0.5*(ra + rb);
        end
                            % update equilibrium state without band
        V0 = 4.0*pi*r0^3/3.0;
        p0 = p1*exp(rg*(V0-V1));

        if (p0 < ep)
            p0 = ep;
            V0 = V1 + log(p0/p1)/rg;
            r0 = (0.75*V0/pi)^(1/3);
        end

        T0 = 0.5*p0*r0;
    end
                            % break with small pressure error
    if (abs(p-p1) < tolp*p1)
        break;
    end
end

disp('Postoperative State:');
disp(strcat('Equation residual=',num2str(er(end))));
disp(strcat('Pressure residual=',num2str(abs(lambda))));
disp(strcat('Volume residual=',num2str(es(end))));
disp(strcat('Pressure comparison: p1=',num2str(p1),' p0=',num2str(p0)));
disp(strcat('Tension comparison: T0=',num2str(T0),' T=',num2str(T)));
disp(strcat('Geometry comparison: r1=',num2str(r1),' r0=',num2str(r0)));
disp(strcat('min/max v=',num2str(min(v)),',',num2str(max(v))));
disp(strcat('Volume comparison: V1=',num2str(V1),' V0=',num2str(V0)));
```

The results from this code are shown graphically as follows.



• Exercise 2: Bungee Cord

◦ Task

Recall the constructions in the lecture notes on pages 204-207 for modelling the dynamic state of a bungee cord. Develop a Matlab code to simulate the cord displacement. Hint: Consider this code, which contains yet some riddles.

◦ Solution

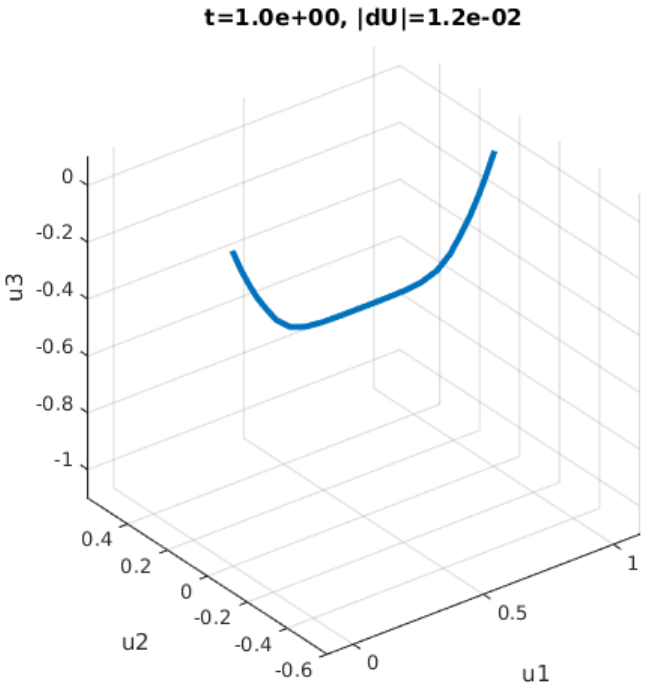
```
% setup figure
    h1 = figure(1);
    close(h1);
    h1 = figure(1);
    set(h1,'Position',[10 10 500 500]);

% model parameters
    r0 = 1.0e0;                     % elasticity
    c0 = 1.0e0;                     % damping
    f0 = 1.0e0;                     % downward force
    clampall = true;                % true => xmin/xmax, otherwise xmin

% grid parameters
    Nx  = 20;                       % Nx cells
    xmin = 0; xmax = 1;
    x  = linspace(xmin,xmax,Nx+1);  % x(i) is a "node"
    hx = x(2)-x(1);
    xc = x(1:Nx) + hx/2;            % xc(i) is a "cut" (cell center)
    dt = 1.0e-2;
    Nt = 10000;
    T  = Nt*dt;                     % final time
```

```
    tol = 1.0e-4;                    % termination criterion
    th  = 1.0;                       % theta time stepping

% gradient, from nodes to cuts directly between them
    dx = spdiags(ones(Nx,1),1,Nx,Nx+1) ...
       - spdiags(ones(Nx,1),0,Nx,Nx+1);
    dx = dx/hx;

% zeros and identities
    Z  = sparse(Nx+1,Nx+1);
    Z3 = kron(speye(3),Z);
    I  = speye(Nx+1);
    I3 = kron(speye(3),I);
    I6 = kron(speye(6),I);

% boundary projection
    if (clampall)
        b   = ones(Nx+1,1); b(1)=0; b(Nx+1)=0;
        Bd  = spdiags(b,0,Nx+1,Nx+1);
    else
        b  = ones(Nx+1,1); b(1)=0;
        Bd  = spdiags(b,0,Nx+1,Nx+1);
    end

% damping acting only away from Dirchlet BCs
    C  = c0*Bd;
    C3 = kron(speye(3),C);

% downward force acting only away from Dirichlet BCs
    f1 =    zeros(Nx+1,1); f1 = Bd*f1;
    f2 =    zeros(Nx+1,1); f2 = Bd*f2;
    f3 = -f0*ones(Nx+1,1); f3 = Bd*f3;
    f  = [f1;f2;f3];
    F  = [zeros(3*(Nx+1),1);f];

% initial state, displacements
    u1 = x';
    u2 = zeros(Nx+1,1);
    u3 = zeros(Nx+1,1);
% and velocities
    u1t = zeros(Nx+1,1);
    u2t = zeros(Nx+1,1);
    u3t = zeros(Nx+1,1);

% the state vector
    U = [u1;u2;u3;u1t;u2t;u3t];

% start time stepping
```

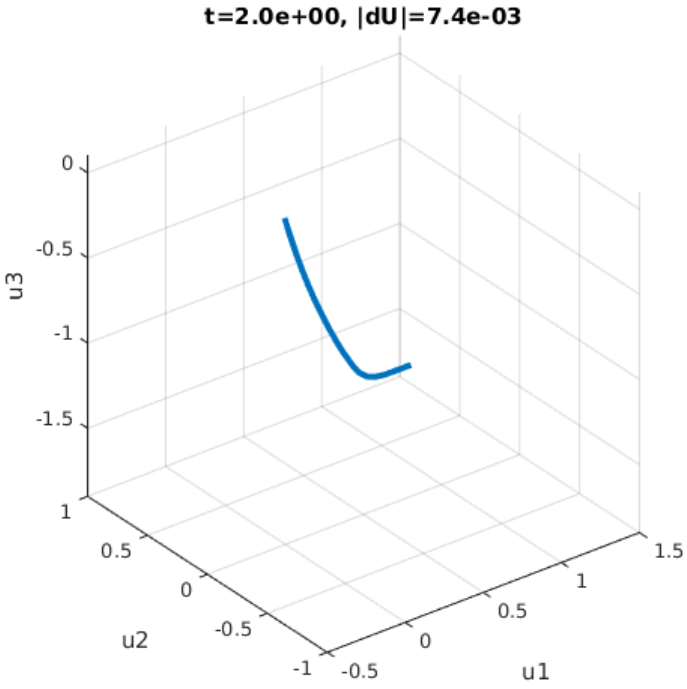
```
    for k=1:Nt
        t  = k*dt;                                  % current time
        Ua = U;                                     % save for stopping criterion

% ensure BCs
        u1 = U(         1:  (Nx+1));
        u2 = U(  (Nx+1)+1:2*(Nx+1));
        u3 = U(2*(Nx+1)+1:3*(Nx+1));
        if (clampall)
            u1(1)=xmin; u1(Nx+1)=xmax;
            u2(1)=0;    u2(Nx+1)=0;
            u3(1)=0;    u3(Nx+1)=0;
        else
            u1(1)=xmin;
            u2(1)=0;
            u3(1)=0;
        end
        U(1:3*(Nx+1)) = [u1;u2;u3];

% gradients at cuts
        u1x = dx*u1;
        u2x = dx*u2;
        u3x = dx*u3;

% length increments on cuts
        dL = sqrt(u1x.^2 + u2x.^2 + u3x.^2);

% coefficients on cuts
        cf = r0*(1 - 1./dL);

% second order differential operator
        L  = dx'*spdiags(cf(:),0,Nx,Nx)*dx;
        L  = Bd*L;
        L3 = kron(speye(3),L);

% U' = A*U + F
        A6 = [Z3,I3;-L3,-C3];

% update with the theta method
        U  = (I6 - th*dt*A6) \ ((I6 + (1-th)*dt*A6)*U + dt*F);

% graph the current state
        plot3(u1,u2,u3,'LineWidth',3);
        title(sprintf('t=%0.1e, |dU|=%0.1e',t,norm(U-Ua)/norm(U)))
        xlabel('u1'); ylabel('u2'); zlabel('u3');
        grid on;
        if (clampall)
            d = 0.6; axis([0.5-d,0.5+d,0.0-d,0.0+d,0.1-2*d,0.1]);
```

```
    else
        d = 1.0; axis([0.5-d,0.5+d,0.0-d,0.0+d,0.1-2*d,0.1]);
    end
    pbaspect([1 1 1]);
    drawnow;

    if (norm(U-Ua) <= tol*norm(U))           % stop when iterates
        break;                               % do not differ
    end
end
```

The result at time $t = 1$ for a cord falling while fastened at both ends is shown graphically as follows.



The result at time $t = 2$ for a cord falling while fastened at only one end is shown graphically as follows.



• Exercise 3: Pastry Sheet

◦ Task

Recall the definitions on page 210 of the lecture notes. Based upon the principle of least action, derive the stationary state for the Lagrangian functional

$$L(u) = \int_0^T \int_0^1 \int_0^1 \left[\tfrac{1}{2}\rho\|u_t\|^2 - \tfrac{1}{2}\kappa(\|u_\xi \times u_\eta\| - 1)^2 + f \cdot u\right] d\xi d\eta dt$$

$$u(\xi, 0, t) = (\xi, 0, 0), \quad u(\xi, \eta, 0) = (\xi, \eta, 0), \quad u_t(\xi, \eta, 0) = (0, 0, 0)$$

with gravity $f = (0, 0, -\phi/\rho)$ and $\rho, \kappa, \phi > 0$. Develop a Matlab code to simulate the membrane displacement. Hint: Consider this code, which contains yet some riddles.

◦ Solution

Let $\Omega = (0, 1)^2$ and $Q = \Omega \times (0, T)$ and suppose a test function v is sufficiently smooth and vanishes at the spatial boundary $\eta = 0$ and at initial and final times $t = 0, T$. For the computation of variational derivatives note that

$$\begin{aligned}
&\frac{d}{d\epsilon}\left[(u+\epsilon v)_\xi \times (u+\epsilon v)_\eta\right] \cdot \left[(u+\epsilon v)_\xi \times (u+\epsilon v)_\eta\right] = \\
&\left\{\frac{d}{d\epsilon}\left[(u+\epsilon v)_\xi \times (u+\epsilon v)_\eta\right]\right\} \cdot \left[(u+\epsilon v)_\xi \times (u+\epsilon v)_\eta\right] + \left[(u+\epsilon v)_\xi \times (u+\epsilon v)_\eta\right] \cdot \left\{\frac{d}{d\epsilon}\left[(u+\epsilon v)_\xi \times (u+\epsilon v)_\eta\right]\right\} \\
&= \left\{\left[\frac{d}{d\epsilon}(u+\epsilon v)_\xi\right] \times (u+\epsilon v)_\eta + (u+\epsilon v)_\xi \times \left[\frac{d}{d\epsilon}(u+\epsilon v)_\eta\right]\right\} \cdot \left[(u+\epsilon v)_\xi \times (u+\epsilon v)_\eta\right] \\
&+ \left[(u+\epsilon v)_\xi \times (u+\epsilon v)_\eta\right] \cdot \left\{\left[\frac{d}{d\epsilon}(u+\epsilon v)_\xi\right] \times (u+\epsilon v)_\eta + (u+\epsilon v)_\xi \times \left[\frac{d}{d\epsilon}(u+\epsilon v)_\eta\right]\right\} \\
&= \left\{v_\xi \times (u+\epsilon v)_\eta + (u+\epsilon v)_\xi \times v_\eta\right\} \cdot \left[(u+\epsilon v)_\xi \times (u+\epsilon v)_\eta\right] \\
&+ \left[(u+\epsilon v)_\xi \times (u+\epsilon v)_\eta\right] \cdot \left\{v_\xi \times (u+\epsilon v)_\eta + (u+\epsilon v)_\xi \times v_\eta\right\} \\
&\xrightarrow{\epsilon \to 0} (v_\xi \times u_\eta + u_\xi \times v_\eta) \cdot (u_\xi \times u_\eta) + (u_\xi \times u_\eta) \cdot (v_\xi \times u_\eta + u_\xi \times v_\eta) \\
&= 2(u_\xi \times u_\eta) \cdot (v_\xi \times u_\eta + u_\xi \times v_\eta) \\
&= 2(u_\eta \times u_\xi) \cdot (u_\eta \times v_\xi) + 2(u_\xi \times u_\eta) \cdot (u_\xi \times v_\eta)
\end{aligned}$$

and using the following notation

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}, \quad [\![a]\!] = \begin{bmatrix} 0 & -a_3 & +a_2 \\ +a_3 & 0 & -a_1 \\ -a_2 & +a_1 & 0 \end{bmatrix}, \quad u \times v = [\![u]\!]v = -[\![u]\!]^\top v$$

the derivative becomes

$$\begin{aligned}
&= 2([\![u_\xi]\!]u_\eta) \cdot ([\![u_\xi]\!]v_\eta) + 2([\![u_\eta]\!]u_\xi) \cdot ([\![u_\eta]\!]v_\xi) \\
&= 2u_\eta^\top [\![u_\xi]\!]^\top [\![u_\xi]\!]v_\eta + 2u_\xi^\top [\![u_\eta]\!]^\top [\![u_\eta]\!]v_\xi = -2u_\eta^\top [\![u_\xi]\!]^2 v_\eta - 2u_\xi^\top [\![u_\eta]\!]^2 v_\xi.
\end{aligned}$$

Then the variational derivative of L is given by

$$\frac{\delta L}{\delta u}(u; v) = \int_Q \left[\rho u_t \cdot v_t + \kappa(\|u_\xi \times u_\eta\| - 1)\frac{u_\eta^\top [\![u_\xi]\!]^2 v_\eta + u_\xi^\top [\![u_\eta]\!]^2 v_\xi}{\|u_\xi \times u_\eta\|} + f \cdot v\right]$$

$$= \int_Q v \cdot \left[-\rho u_{tt} + \left(\kappa\frac{1 - \|u_\xi \times u_\eta\|}{\|u_\xi \times u_\eta\|}[\![u_\eta]\!]^2 u_\xi\right)_\xi + \left(\kappa\frac{1 - \|u_\xi \times u_\eta\|}{\|u_\xi \times u_\eta\|}[\![u_\xi]\!]^2 u_\eta\right)_\eta + f\right]$$

$$+\int_\Omega \rho u_t \cdot v \Big|_{t=0}^{t=T} + \int_0^T \kappa \frac{\|u_\xi \times u_\eta\| - 1}{\|u_\xi \times u_\eta\|} u_\xi^\top [\![u_\eta]\!]^2 v \Big|_{\xi=0}^{\xi=1} + \int_0^T \kappa \frac{\|u_\xi \times u_\eta\| - 1}{\|u_\xi \times u_\eta\|} u_\eta^\top [\![u_\xi]\!]^2 v \Big|_{\eta=0}^{\eta=1}$$

or with $g = f/\rho = (0, 0, -\phi)$

$$\frac{\delta L}{\partial u}(u; v) = \rho \int_Q v \cdot [-u_{tt} + (A(u)u_\xi)_\xi + (B(u)u_\eta)_\eta + g]$$

$$+\rho \int_\Omega u_t \cdot v \Big|_{t=0}^{t=T} - \rho \int_0^T u_\xi^\top A(u) v \Big|_{\xi=0}^{\xi=1} - \rho \int_0^T u_\eta^\top B(u) v \Big|_{\eta=0}^{\eta=1}$$

where the following are defined

$$\delta(u) = \frac{\kappa}{\rho} \frac{1 - \|u_\xi \times u_\eta\|}{\|u_\xi \times u_\eta\|}, \quad A(u) = \delta(u)[\![u_\xi]\!]^2 = A(u)^\top, \quad B(u) = \delta(u)[\![u_\eta]\!]^2 = B(u)^\top.$$

Since $v = 0$ at $t = 0, T$, for the first integral vanishes and implicitly requires initial and final time conditions on u, but instead the initial position and velocity are imposed,

$$u(\xi, \eta, 0) = (\xi, \eta, 0), \quad u_t(\xi, \eta, 0) = (0, 0, 0)$$

and under the condition of well-posedness of the resulting intial- and boundary-value problem, the final time values $u(\xi, \eta, T)$ may be regarded as having been imposed as a condition on the Lagrangian functional L. The condition that $v = 0$ hold at $\eta = 0$ corresponds to the given Dirichlet boundary condition

$$u(\xi, 0, t) = (\xi, 0, 0), \quad \xi \in [0, 1], \quad t \in [0, T].$$

Letting v be concentrated on ∂Q at $\eta = 1$ gives the natural boundary condition

$$B(u)u_\eta(\xi, 1, t) = (0, 0, 0), \quad \xi \in [0, 1], \quad t \in [0, T].$$

Letting v be concentrated on ∂Q at $\xi = 0, 1$ gives the natural boundary conditions

$$\begin{array}{l} A(u)[u_\xi(0, \eta, t) - (1, 0, 0)] = (0, 0, 0) \\ A(u)[u_\xi(1, \eta, t) - (1, 0, 0)] = (0, 0, 0) \end{array} \quad \eta \in [0, 1], \quad t \in [0, T]$$

where the term $-(1, 0, 0)$ is required for compatibility with the Dirichlet boundary condition $u(\xi, 0, t) = (\xi, 0, 0)$ at $\eta = 0$. The necessary optimality condition for a stationary u is

$$\left\{ \begin{array}{rll} \rho u_{tt} = (A(u)u_\xi)_\xi + (B(u)u_\xi)_\eta + f, & & (\xi, \eta, t) \in Q \\ u(\xi, 0, t) = (\xi, 0, 0), \quad B(u)u_\eta(\xi, 1, t) = (0, 0, 0), & \xi \in [0, 1], & t \in [0, T] \\ A(u)[u_\xi(\xi, \eta, t) - (1, 0, 0)] = (0, 0, 0), \quad \xi = 0, 1, & \eta \in [0, 1], & t \in [0, T] \\ u(\xi, \eta, 0) = (\xi, \eta, 0), \quad u_t(\xi, \eta, 0) = (0, 0, 0), & & (\xi, \eta) \in \Omega \end{array} \right.$$

The following Matlab code is used to solve the initial and boundary-value problem.

```
% setup figure
    h1 = figure(1);
    close(h1);
    h1 = figure(1);
    set(h1,'Position',[10 10 500 500]);
```

```
% model parameters
    r0 = 5.0e-1;                                % elasticity
    c0 = 2.0e-1;                                % damping
    f0 = 1.0e-1;                                % downward force
    clampall = false;                           % true => x,y,min/max
                                                %   otherwise y=ymin
% grid parameters
    Nx  = 10;                                   % Nx X Ny cells
    Ny  = 10;                                   % (Nx+1) X (Ny+1) "nodes"
    xmin = 0; xmax = 1;
    ymin = 0; ymax = 1;
    x  = linspace(xmin,xmax,Nx+1);              % (x(i),y(j)) is a "node"
    y  = linspace(ymin,ymax,Ny+1);
    hx = x(2)-x(1);
    hy = y(2)-y(1);
    xc = x(1:Nx) + hx/2;                        % (xc(i),y(j)) is an "x-cut"
    yc = y(1:Ny) + hy/2;                        % (x(i),yc(j)) is a "y-cut"
    dt = 1.0e-2;
    Nt = 100000;
    T  = Nt*dt;                                 % final time
    dt = T/Nt;
    tol = 1.0e-6;                               % termination criterion
    th  = 1;                                    % theta time stepping

% gradient, from nodes to cuts directly between them
    dx = spdiags(ones(Nx,1),1,Nx,Nx+1) ...
       - spdiags(ones(Nx,1),0,Nx,Nx+1);
    dx = dx/hx;
    iy = speye(Ny+1);
    Dx = kron(iy,dx);                           % x-derivative, nodes to x-cuts

    dy = spdiags(ones(Ny,1),1,Ny,Ny+1) ...
       - spdiags(ones(Ny,1),0,Ny,Ny+1);
    dy = dy/hy;
    ix = speye(Nx+1);
    Dy = kron(dy,ix);                           % y-derivative, nodes to y-cuts

% gradient, from nodes to cuts opposite direction
    dx = spdiags(ones(Nx+1,1),+1,Nx+1,Nx+1) ...
       - spdiags(ones(Nx+1,1),-1,Nx+1,Nx+1);    % x-derivative, nodes to nodes
    dx = dx/(2*hx);
    dx(   1, 1) = -1/hx; dx(   1,   2) = 1/hx;  % one-sided at boundary
    dx(Nx+1,Nx) = -1/hx; dx(Nx+1,Nx+1) = 1/hx;
    ay = spdiags(ones(Ny,1), 1,Ny,Ny+1) ...     % y-average, nodes to y-cuts
       + spdiags(ones(Ny,1), 0,Ny,Ny+1);
    ay = ay/2;
    Cx = kron(ay,dx);                           % nodes to y-cuts
```

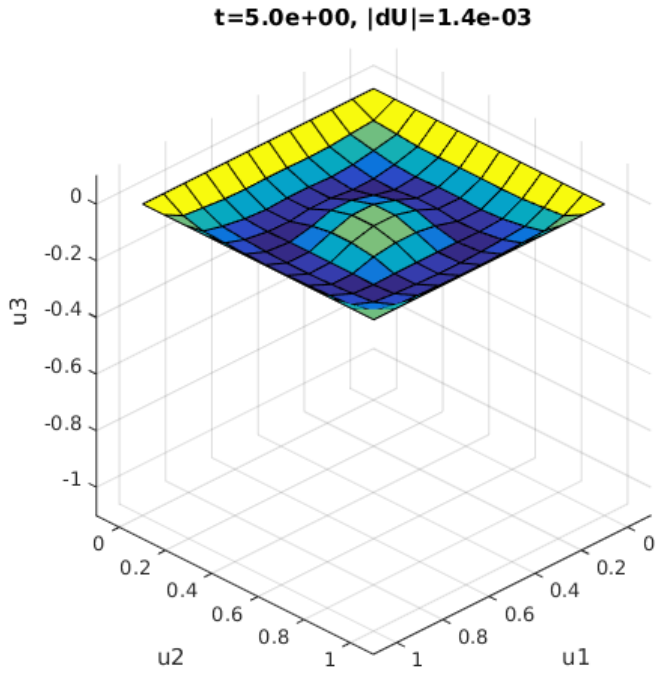
```
    dy = spdiags(ones(Ny+1,1),+1,Ny+1,Ny+1) ...
       - spdiags(ones(Ny+1,1),-1,Ny+1,Ny+1);    % y-derivative, nodes to nodes
    dy = dy/(2*hy);
    dy(   1, 1) = -1/hy; dy(   1,   2) = 1/hy;  % one-sided at boundary
    dy(Ny+1,Ny) = -1/hy; dy(Ny+1,Ny+1) = 1/hy;
    ax = spdiags(ones(Nx,1), 1,Nx,Nx+1) ...     % x-average, nodes to x-cuts
       + spdiags(ones(Nx,1), 0,Nx,Nx+1);
    ax = ax/2;
    Cy = kron(dy,ax);                           % nodes to x-cuts

% zeros and identities
    Z  = sparse((Nx+1)*(Ny+1),(Nx+1)*(Ny+1));
    Z3 = kron(speye(3),Z);
    I  = speye((Nx+1)*(Ny+1));
    I3 = kron(speye(3),I);
    I6 = kron(speye(6),I);

% boundary projection
    if (clampall)
        b   = ones(Nx+1,Ny+1);
        b(1,:)=0; b(Nx+1,:)=0; b(:,1)=0; b(:,Ny+1)=0; b=b(:);
        Bd  = spdiags(b,0,(Nx+1)*(Ny+1),(Nx+1)*(Ny+1));
        Bd3 = kron(speye(3),Bd);                % Dirichlet
        Bx  = I;
        Bx3 = I3;                               % Neumann, x-direction
        By  = I;
        By3 = I3;                               % Neumann, y-direction
    else
        b   = ones(Nx+1,Ny+1);
        b(:,1)=0; b=b(:);
        Bd  = spdiags(b,0,(Nx+1)*(Ny+1),(Nx+1)*(Ny+1));
        Bd3 = kron(speye(3),Bd);                % Dirichlet
        b   = ones(Nx+1,Ny+1);
        b(1,:)=0; b(Nx+1,:)=0; b=b(:);
        Bx  = spdiags(b,0,(Nx+1)*(Ny+1),(Nx+1)*(Ny+1));
        Bx3 = kron(speye(3),Bx);                % Neumann, x-direction
        b   = ones(Nx+1,Ny+1);
        b(:,Ny+1)=0; b=b(:);
        By  = spdiags(b,0,(Nx+1)*(Ny+1),(Nx+1)*(Ny+1));
        By3 = kron(speye(3),By);                % Neumann, y-direction
    end

% damping acting only away from Dirchlet BCs
    C  = c0*Bd;
    C3 = kron(speye(3),C);

% downward force acting only away from Dirichlet BCs
    f1 =    zeros(Nx+1,Ny+1);  f1 = Bd*f1(:);
```

```
    f2 =    zeros(Nx+1,Ny+1);  f2 = Bd*f2(:);
    f3 = -f0*ones(Nx+1,Ny+1);  f3 = Bd*f3(:);
    f  = [f1;f2;f3];

% initial state, displacements (saved in v1,v2,v3)
    u1 = kron(x',ones(1,Ny+1));  u1 = u1(:); v1=u1;
    u2 = kron(ones(Nx+1,1),y);   u2 = u2(:); v2=u2;
    u3 = zeros(Nx+1,Ny+1);       u3 = u3(:); v3=u3;
% and velocities
    u1t = zeros(Nx+1,Ny+1);      u1t = u1t(:);
    u2t = zeros(Nx+1,Ny+1);      u2t = u2t(:);
    u3t = zeros(Nx+1,Ny+1);      u3t = u3t(:);

% the state vector
    U   = [u1;u2;u3;u1t;u2t;u3t];

% start time stepping
    for k=1:Nt
        t  = k*dt;                                % current time
        Ua = U;                                   % save for stopping criterion

% ensure BCs
        u1 = reshape(U(                1:  (Nx+1)*(Ny+1)),Nx+1,Ny+1);
        u2 = reshape(U(  (Nx+1)*(Ny+1)+1:2*(Nx+1)*(Ny+1)),Nx+1,Ny+1);
        u3 = reshape(U(2*(Nx+1)*(Ny+1)+1:3*(Nx+1)*(Ny+1)),Nx+1,Ny+1);
        if (clampall)
            u1(1,:)=xmin;  u1(Nx+1,:)=xmax;
            u1(:,1)=x';    u1(:,Ny+1)=x';
            u2(1,:)=y;    u2(Nx+1,:)=y;
            u2(:,1)=ymin; u2(:,Ny+1)=ymax;
            u3(1,:)=0; u3(Nx+1,:)=0;
            u3(:,1)=0; u3(:,Ny+1)=0;
        else
            u1(:,1)=x';
            u2(:,1)=ymin;
            u3(:,1)=0;
        end
        u1 = u1(:); u2 = u2(:); u3 = u3(:);
        U(1:3*(Nx+1)*(Ny+1)) = [u1;u2;u3];

% gradients at x-cuts
        u1x  = Dx*u1; u2x  = Dx*u2; u3x  = Dx*u3;
        u1y_ = Cy*u1; u2y_ = Cy*u2; u3y_ = Cy*u3;

% gradients at y-cuts
        u1x_ = Cx*u1; u2x_ = Cx*u2; u3x_ = Cx*u3;
        u1y  = Dy*u1; u2y  = Dy*u2; u3y  = Dy*u3;
```

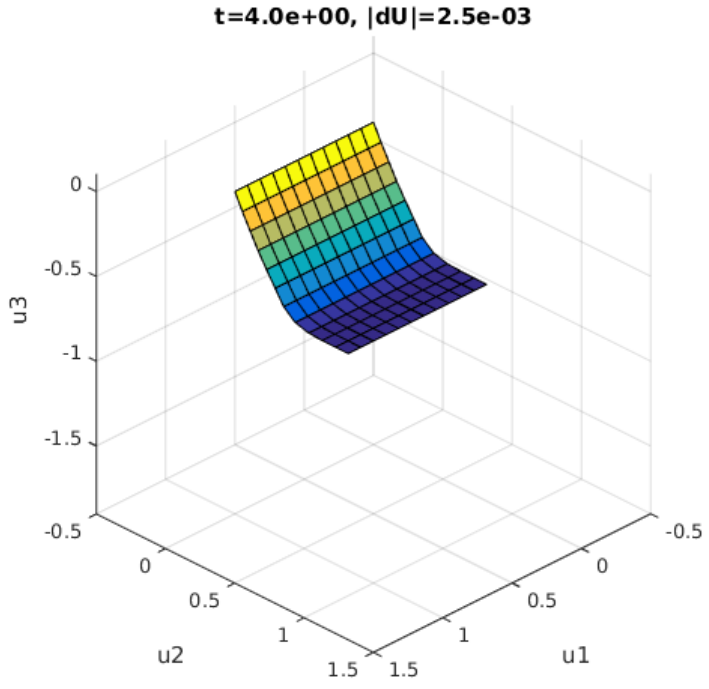
```
% area increments on x-cuts
        dEx = sqrt((u2x.*u3y_ - u3x.*u2y_).^2 + ...
                   (u1x.*u3y_ - u3x.*u1y_).^2 + ...
                   (u1x.*u2y_ - u2x.*u1y_).^2);

% area increments on y-cuts
        dEy = sqrt((u2x_.*u3y - u3x_.*u2y).^2 + ...
                   (u1x_.*u3y - u3x_.*u1y).^2 + ...
                   (u1x_.*u2y - u2x_.*u1y).^2);

% coefficients on x- and y-cuts
        cfx = r0*(1 - 1./dEx);
        cfy = r0*(1 - 1./dEy);

% coefficients in the second order differential operator
%     utt + c ut = -Lu + f
%     Lu = Dx' { r0 (1 - 1/|ux X uy|)[uy]'[uy] } Dx u
%        + Dy' { r0 (1 - 1/|ux X uy|)[ux]'[ux] } Dy u
%
%     [u] = [  0,-u3, u2; u3,  0,-u1;-u2, u1,  0]
%
% [u]'[u] = [u2^2+u3^2,-u1u2,-u1u3;-u1u2,u1^2+u3^2,-u2u3;-u1u3,-u2u3,u1^2+u2^2]

% compute coefficients [uy]'*[uy] on x-cuts
        Uy =  [spdiags(u2y_.^2 + u3y_.^2,0,Nx*(Ny+1),Nx*(Ny+1)),     ...
               spdiags(      -u1y_.*u2y_,0,Nx*(Ny+1),Nx*(Ny+1)),     ...
               spdiags(      -u1y_.*u3y_,0,Nx*(Ny+1),Nx*(Ny+1));     ...
                 spdiags(      -u1y_.*u2y_,0,Nx*(Ny+1),Nx*(Ny+1)),   ...
                 spdiags(u1y_.^2 + u3y_.^2,0,Nx*(Ny+1),Nx*(Ny+1)),   ...
                 spdiags(      -u2y_.*u3y_,0,Nx*(Ny+1),Nx*(Ny+1));   ...
                   spdiags(      -u1y_.*u3y_,0,Nx*(Ny+1),Nx*(Ny+1)), ...
                   spdiags(      -u2y_.*u3y_,0,Nx*(Ny+1),Nx*(Ny+1)), ...
                   spdiags(u1y_.^2 + u2y_.^2,0,Nx*(Ny+1),Nx*(Ny+1))];

% compute coefficients [ux]'*[ux] on y-cuts
        Ux =  [spdiags(u2x_.^2 + u3x_.^2,0,(Nx+1)*Ny,(Nx+1)*Ny),     ...
               spdiags(      -u1x_.*u2x_,0,(Nx+1)*Ny,(Nx+1)*Ny),     ...
               spdiags(      -u1x_.*u3x_,0,(Nx+1)*Ny,(Nx+1)*Ny);     ...
                 spdiags(      -u1x_.*u2x_,0,(Nx+1)*Ny,(Nx+1)*Ny),   ...
                 spdiags(u1x_.^2 + u3x_.^2,0,(Nx+1)*Ny,(Nx+1)*Ny),   ...
                 spdiags(      -u2x_.*u3x_,0,(Nx+1)*Ny,(Nx+1)*Ny);   ...
                   spdiags(      -u1x_.*u3x_,0,(Nx+1)*Ny,(Nx+1)*Ny), ...
                   spdiags(      -u2x_.*u3x_,0,(Nx+1)*Ny,(Nx+1)*Ny), ...
                   spdiags(u1x_.^2 + u2x_.^2,0,(Nx+1)*Ny,(Nx+1)*Ny)];

% second order differential operator
        Lx = kron(speye(3),Dx')* ...
             kron(speye(3),spdiags(cfx,0,Nx*(Ny+1),Nx*(Ny+1)))*Uy ...
```

```
            *kron(speye(3),Dx);
        Ly = kron(speye(3),Dy')* ...
             kron(speye(3),spdiags(cfy,0,(Nx+1)*Ny,(Nx+1)*Ny))*Ux ...
            *kron(speye(3),Dy);
        L3 = Lx + Ly;
        L3 = Bd3*L3;                            % only away from Dirichlet BCs

% U' = A*U + F
        A6 = [Z3,I3;-L3,-C3];
        dv = (I3-Bx3)*L3*[v1;0*v2;0*v3];        % Neumann BC for compatibility
        F  = [zeros(3*(Nx+1)*(Ny+1),1);f + dv]; % total force

% update with the theta method
        U  = (I6 - th*dt*A6) \ ((I6 + (1-th)*dt*A6)*U + dt*F);
        U  = U.*(abs(U) > tol);                 % purely for round-off
                                                % increase damping otherwise
% graph the current state
        surf(reshape(u1,Nx+1,Ny+1), ...
             reshape(u2,Nx+1,Ny+1), ...
             reshape(u3,Nx+1,Ny+1));
        xlabel('u1'); ylabel('u2'); zlabel('u3');
        view([135,30])
%       view([30*t,30])
        grid on;
        if (clampall)
            d=0.6; axis([0.5-d,0.5+d,0.5-d,0.5+d,0.1-2*d,0.1]);
        else
            d=1.0; axis([0.5-d,0.5+d,0.5-d,0.5+d,0.1-2*d,0.1]);
        end
        pbaspect([1 1 1]);
        title(sprintf('t=%0.1e, |dU|=%0.1e',t,norm(U-Ua)/norm(U)))
        drawnow;

        if (norm(U-Ua) <= tol*norm(U))          % stop when iterates
            break;                              % do not differ
        end

    end
```

The result at time $t = 5$ for a membrane falling while fastened on all sides is shown graphically as follows.



The result at time $t = 4$ for a membrane falling while fastened only along one edge is shown graphically as follows.



• Exercise 4: Meditation Model

◦ Task

Develop a wave equation model for the phenomena illustrated in this video of a simulation, and implement your model in Matlab.

◦ Solution

The following model is proposed for the simulation of known neurophysiological phenomena associated with waking and dream or meditative states. First, the spatial domain $\Omega = (0,1)^2$ is divided into halves,

$$\Omega_{\rm l} = \{(x,y) \in \Omega : y \in [0, \tfrac{1}{2}]\}, \quad \Omega_{\rm r} = \{(x,y) \in \Omega : y \in [\tfrac{1}{2}, 1]\}$$

where $\Omega_{\rm l}$ models the left hemisphere of the brain and $\Omega_{\rm r}$ the right. Let $\hat{n}$ denote a unit vector which is outwardly directed with respect to $\partial\Omega_{\rm l}$ and $\partial\Omega_{\rm r}$. Let $A(u) = \mathrm{diag}(\alpha, \omega)$ be the diagonal matrix with wave speeds $\sqrt{\alpha}$ and $\sqrt{\omega}$ in the x- and y-directions respectively. Let f be a forcing function yet to be specified. Then the wave displacement u is modelled according to:

$$\begin{cases} u_{tt} &= \nabla \cdot (A \nabla u) + f(u), & (x,y) \in (0,1)^2, & & t \in (0,T) \\ u(0,y,t) &= u(1,y,t), & & y \in [0,1], & t \in [0,T] \\ u_t(x,y,t) &= \hat{n}^\top A \nabla u(x,y,t), & x \in [0,1], & y \in \{0, 1/2^{\pm}, 1\}, & t \in [0,T] \\ u &= 0, & (x,y) \in (0,1)^2, & & t = 0 \\ u_t &= 0, & (x,y) \in (0,1)^2, & & t = 0. \end{cases}$$

Since the left hemisphere has been found to carry out serial processing, this characteristic is modelled by allowing waves to propagate in only one direction according to

$$\alpha = 0, \quad \omega > 0, \quad \Omega_{\rm l}.$$

On the other hand, the right hemisphere has been found to carry out parallel processing, and this characteristic is modelled by letting waves travel unhindered in any direction according to

$$\alpha, \omega > 0, \quad \Omega_{\rm r}.$$

At the boundary between the hemispheres transmission is accomplished through $f(u)$ discussed below, but otherwise

$$\omega = 0, \quad \partial\Omega_{\rm r} \cap \partial\Omega_{\rm l}.$$

The squared wave speeds α and ω are otherwise spatially varying,

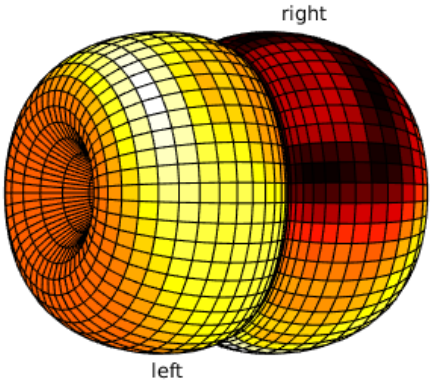
$$\alpha = \alpha(x,y), \quad \omega = \omega(x,y)$$

where more spatially uniform wave speeds are thought to correspond to the result of meditative practice as discussed below.

The spatial geometry is torus-like with periodic boundary conditions at $x = 0, 1$. Also the left and right hemispheres are connected at $y = 0, 1$ and at $y = \pm\frac{1}{2}$. For instance, a given wave function u defined on cells `Nx`×`Ny` in Ω may be mapped onto a torus-like surface according to

```
a = kron(ones(Ny,1),linspace(0,2*pi,Nx));
b = kron(linspace(0,2*pi,Ny)',ones(1,Nx));
y = (2-(1+cos(b)).*cos(b)).*cos(a);
z = (2-(1+cos(b)).*cos(b)).*sin(a);
x = 2*(1+cos(b)).*sin(b);
surf(x,y,z,u);
```

to illustrate the modelling of waves on and connections between the left and right hemispheres as illustrated in the following graphic.



The boundary conditions at $y \in \{0, 1/2^{\pm}, 1\}$ are non-reflecting

$$\begin{aligned} u_t - \sqrt{\omega} u_y &= 0, \quad y = 0 \\ u_t + \sqrt{\omega} u_y &= 0, \quad y = 1/2^- \\ u_t - \sqrt{\omega} u_y &= 0, \quad y = 1/2^+ \\ u_t + \sqrt{\omega} u_y &= 0, \quad y = 1. \end{aligned}$$

To illustrate these, note that

$$\text{a left-travelling } (y = -t\sqrt{\omega}) \text{ wave } u = u(y + t\sqrt{\omega}) \text{ satisfies } (\partial_t - \sqrt{\omega}\partial_y)u(y + t\sqrt{\omega}) = 0$$

and

$$\text{a right-travelling } (y = +t\sqrt{\omega}) \text{ wave } u = u(y - t\sqrt{\omega}) \text{ satisfies } (\partial_t + \sqrt{\omega}\partial_y)u(y - \sqrt{\omega}t) = 0.$$

Thus, at the boundaries of $\Omega_{\rm l}$ or $\Omega_{\rm r}$ at $y \in \{0, 1/2^{\pm}, 1\}$ only out-going waves and no in-coming waves are supported.

One aspect of the forcing function $f(u)$ is to facillitate transmission of waves between hemispheres. For instance, when a wave from the right hemisphere impinges on (a cell near) its boundary at $x = \hat{x}$ and $y = 1/2^-$ with a certain threshold intensity, then a wave is initiated in the left hemisphere from its boundary at $y = 1/2^+$. Since in waking states such impulses are networked, this wave is propagated from several locations at (cells near) $x_j = x_j(\hat{x}) \in [0, 1]$ and $y = 1/2^+$. Since in dream and meditative states this networking is not active, the wave is propagated only from (a cell at) $x = \hat{x}$ and $y = 1/2^+$. Because of the non-reflecting boundary conditions at $y = 1/2^{\pm}$, the impinging wave from the right hemisphere does not reflect back from where it came, and the wave initiated in the left hemisphere marches only forward into the left hemisphere. This process may be regarded as conscious thoughts emerging from the subconscious. In a waking states, a wave in the left brain impinges on (a cell near) its boundary at $x = \check{x}$ and $y = 0$ with a certain intensity, and it then initiates a wave in the right hemisphere from its boundary at (a cell near) $x = \check{x}$ and $y = 1$. Because of the non-reflecting

boundary conditions at $y = 0, 1$, the impinging wave from the left hemisphere does not reflect back from where it came, and the wave initiated in the right hemisphere marches only forward into the right hemisphere. This process may be regarded as a feedback mechanism whereby conscious thoughts affect subconscious memory. In a dream or meditative state, this feedback mechanism is absent. Finally, the forcing function $f(u)$ initiates impulses in the right brain with a probability which gives advantage to a natural rhythm on the one hand but also allows competing random impulses.

The following Matlab code is used to solve the initial and boundary-value problem.

```
% set up figure
    h1 = figure(1);
    close(h1);
    h1 = figure(1);
    set(h1,'Position',[10 30 1000 400]);

% true if meditating
    med = true;

% max wave speeds and max excitation
    al0 = 1;
    om0 = 1;
    u0  = 2;

% there are Nx x Ny cells, Nt time steps and left-right interface at My
    Nx = 51;
    Ny = 51;
    My = round(Ny/2);
    Nt = 1200;

% dimensions of domain
    xmin = 0;
    xmax = 1;
    ymin = 0;
    ymax = 1;
    T    = Nt/500;

% cell sizes
    hx = (xmax-xmin)/Nx;
    hy = (ymax-ymin)/Ny;
    ht = T/Nt;

% cell centers
    x  = linspace(xmin,xmax,Nx)';
    y  = linspace(ymin,ymax,Ny);
    xx = kron(x,ones(1,Ny));
    yy = kron(ones(Nx,1),y);

% first order differential operators
    dx = spdiags(ones(Nx,1), 0,Nx,Nx) ...
```

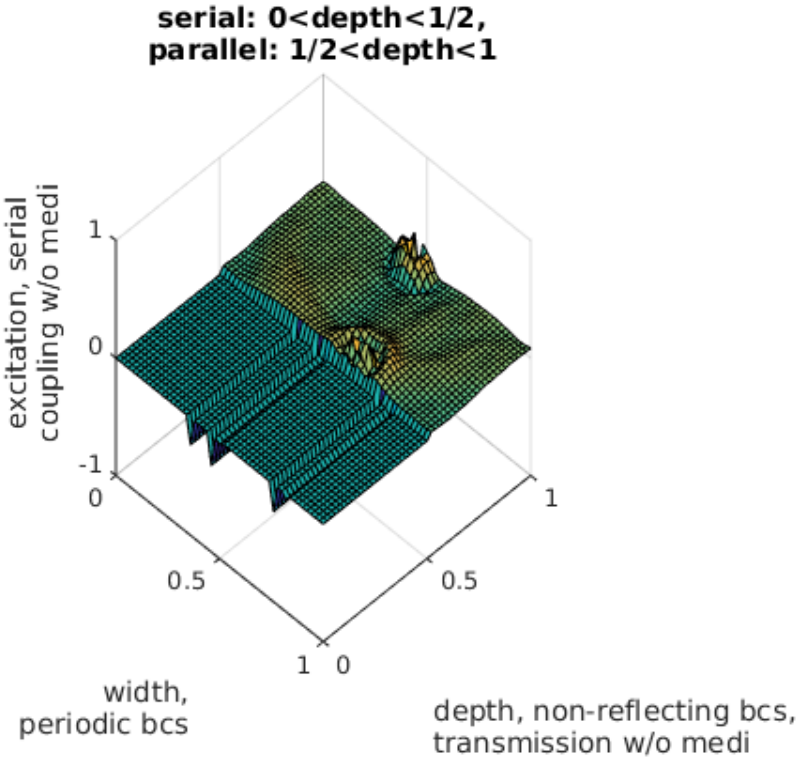
```
       - spdiags(ones(Nx,1),-1,Nx,Nx); dx(1,Nx) = -1;
    dx = dx/hx;
    iy = speye(Ny);
    Dx = kron(iy,dx);
    dy = spdiags(ones(Ny-1,1),1,Ny-1,Ny) ...
       - spdiags(ones(Ny-1,1),0,Ny-1,Ny);
    dy = dy/hy;
    ix = speye(Nx);
    Dy = kron(dy,ix);

% (squared) wave speeds
    if (med)
        al = zeros(Nx,Ny); al(:,My+1:Ny) = al0;
        om = om0*ones(Nx,Ny-1); om(:,My) = 0;
    else
        al = zeros(Nx,Ny); al(:,My+1:Ny) = al0*(1 + rand(Nx,Ny-My))/2;
        om = om0*(1 + rand(Nx,Ny-1))/2; om(:,My) = 0;
    end

% second order differential operators
    Lx = Dx'*spdiags(al(:),0,Nx*Ny,Nx*Ny)*Dx;
    Ly = Dy'*spdiags(om(:),0,Nx*(Ny-1),Nx*(Ny-1))*Dy;
    L  = Lx + Ly;

% identities and zeros
    I  = speye(Nx*Ny);
    I2 = kron(speye(2),I);
    Z  = sparse(Nx*Ny,Nx*Ny);

% non-reflecting boundary conditions through proper damping coefficient
    c  = zeros(Nx,Ny);
    c(:,1)    = sqrt(om(:,1))/hy;
    c(:,My)   = sqrt(om(:,My-1))/hy;
    c(:,My+1) = sqrt(om(:,My+1))/hy;
    c(:,Ny)   = sqrt(om(:,Ny-1))/hy;
    C = spdiags(c(:),0,Nx*Ny,Nx*Ny);

    B  = [Z,I;-L,-C];

% time stepping method
    th = 1.0;

% starting state
    u  = zeros(Nx,Ny);
    ut = zeros(Nx,Ny);
    U  = [u(:);ut(:)];

% association of serial excitations in left brain
```

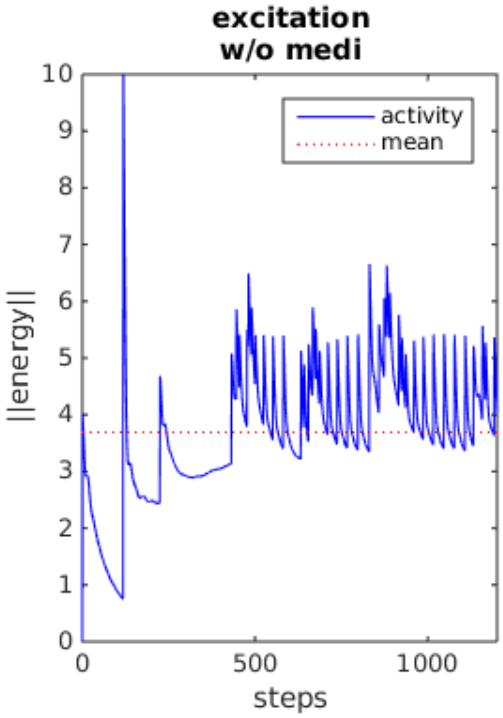
```
    r  = randperm(Nx);
    R  = zeros(Nx,Nx);
    nc = 5;
    l = mod(Nx,nc);
    for i=1:nc:(Nx-l)
        for j=0:(nc-1)
            for k=0:(nc-1)
                R(r(i+j),r(i+k)) = 1;
            end
        end
    end
    if (l ~= 0)
        i = i+nc;
        for j=0:(l-1)
            for k=0:(l-1)
                R(r(i+j),r(i+k)) = 1;
            end
        end
    end

% initialize activation measure
    uact = sqrt((sum((sqrt(al(:))*(Dx*u(:))).^2) ...
               + sum((sqrt(om(:))*(Dy*u(:))).^2) ...
               + sum(ut(:).^2))*hx*hy);

% start time stepping
    for k=1:Nt

% random excitation in right brain
        urmax = max(abs(u(:,My+1:Ny)));
        if (urmax < 0.01*u0)
            u(round(Nx/2),round((My+Ny)/2)) = u0;
        else
            if (rand(1) < 0.005)
                i = randi(Nx); j = randi(Ny-My-2)+My+1;
                u(i,j) = u0;
            end
        end

% excitation from right brain into left brain
        urmax = max(abs(u(:,My+1)));
        ulmax = max(max(abs(u(:,1:My))));
        if ((urmax > 0.005*u0) && (ulmax < 0.01*u0))
            i = find(abs(u(:,My+1)) == urmax,1);
% if med, then only one impulse, otherwise assciated impulses
            if (med)
                u(i,My) = u0; ut(i,My) = -u0/hy;
            else
```

```
                r = find(R(:,i));
                u(r,My) = u0; ut(r,My) = -u0/hy;
            end
        end

% if not med, excitation from left brain into right brain
        if (~med)
            ulmax = max(max(abs(u(:,1))));
            if (ulmax > 0.1*u0)
                i = find(abs(u(:,1)) == ulmax);
                for j=i
                    if (max(abs(u(j,Ny))) < 0.01*u0)
                        u(j,Ny) = u0; ut(j,Ny) = -u0/hy;
                    end
                end
            end
        end

        U  = [u(:);ut(:)];
        U = (I2 - th*ht*B) \ ((I2 + (1-th)*ht*B)*U);

% update activation measure
        uact = [uact,sqrt((sum((sqrt(al(:))*(Dx*u(:))).^2) ...
                         + sum((sqrt(om(:))*(Dy*u(:))).^2) ...
                         + sum(ut(:).^2))*hx*hy)];

% decompose state vector for plot of excitation waves
        u  = reshape(U(1:Nx*Ny),Nx,Ny);
        ut = reshape(U(1+Nx*Ny:2*Nx*Ny),Nx,Ny);

% plot excitation waves
        subplot(1,2,1)
        surf(xx,yy,u)
        xlabel(sprintf('width,\nperiodic bcs'));
        ylabel(sprintf('depth, non-reflecting bcs,\ntransmission w/o medi'));
        zlabel(sprintf('excitation, serial\ncoupling w/o medi'));
        axis([xmin xmax ymin ymax -1 1])
        title(sprintf('serial: 0<depth<1/2,\nparallel: 1/2<depth<1'))
        view([45,45]);
        set(gca,'XTick',[0.0 0.5 1.0], ...
                'YTick',[0.0 0.5 1.0], ...
                'ZTick',[-1.0 0.0 +1.0])
        drawnow;

% plot activation measure
        if (mod(k,50) == 0)
            subplot(1,2,2)
            plot(0:k,uact,'b',[0,k],[mean(uact),mean(uact)],'r:')
```

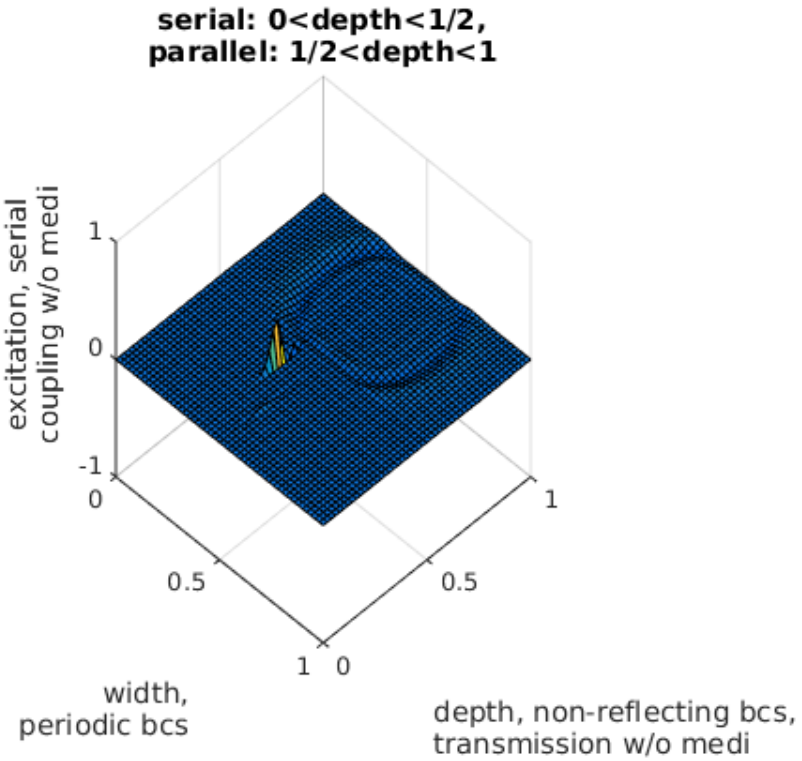
```
        axis([0 k 0 10])
        legend('activity','mean')
        xlabel('steps')
        ylabel('||energy||')
        if (med)
            title(sprintf('excitation\nw/ medi'));
        else
            title(sprintf('excitation\nw/o medi'));
        end
        drawnow;
    end

end
```

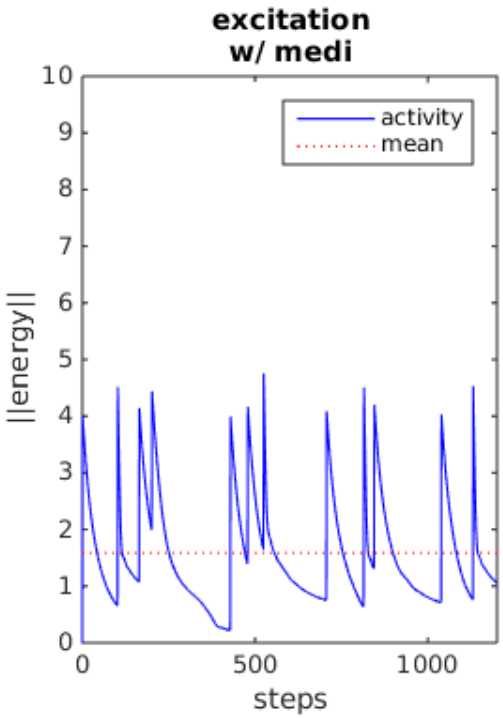
Following are graphical representations of the results with `med = false`,





and with `med = true`,





Note that the mean energy level

$$\frac{1}{|\Omega|T}\int_0^T\int_\Omega [u_t^2 + \nabla u^\top A^2 \nabla u] d\boldsymbol{x}dt$$

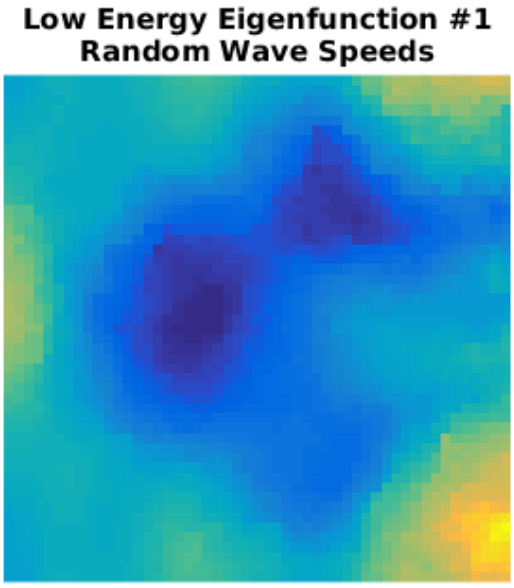
is significantly lower in the model with meditation than in the model without.

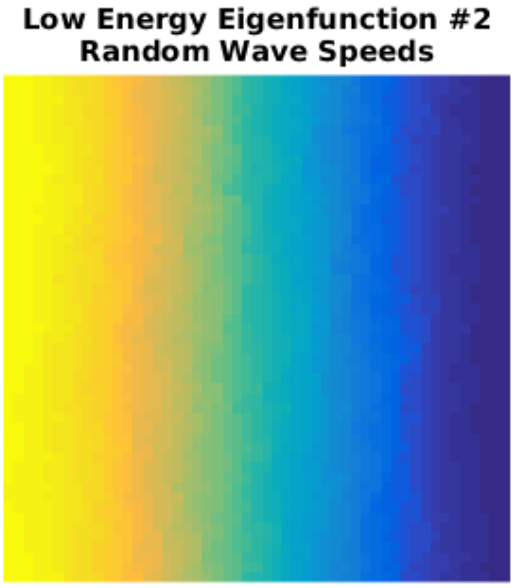
To investigate the effect of random or uniform wave speeds, which presumably can be affected by meditative practice, the eigenfunctions of the second order differential operator are shown as images for the respective cases as follows. The following Matlab code used to produce these images.

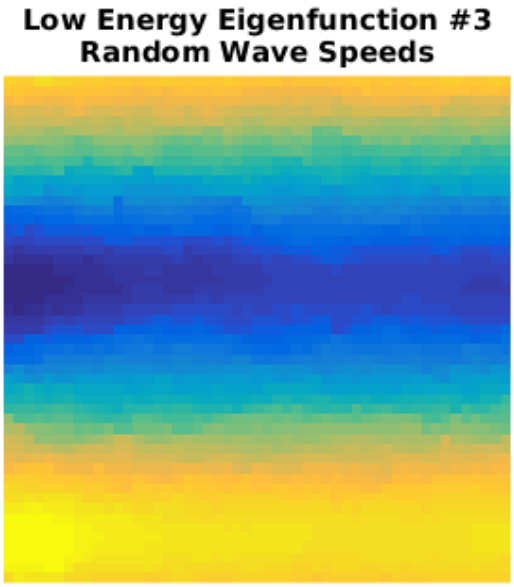
```
% true if meditating
    med = true;

% max wave speeds and max excitation
    al0 = 1;
    om0 = 1;

% there are Nx x Ny cells, Nt time steps and left-right interface at My
    Nx = 51;
    Ny = 51;

% dimensions of domain
    xmin = 0; xmax = 1;
    ymin = 0; ymax = 1;

% cell sizes
    hx = (xmax - xmin)/Nx;
    hy = (ymax - ymin)/Ny;

% first order differential operators
    dx = spdiags(ones(Nx,1), 0,Nx,Nx) ...
       - spdiags(ones(Nx,1),-1,Nx,Nx); dx(1,Nx) = -1;
    dx = dx/hx;
    iy = speye(Ny);
    Dx = kron(iy,dx);
    dy = spdiags(ones(Ny-1,1),1,Ny-1,Ny) ...
       - spdiags(ones(Ny-1,1),0,Ny-1,Ny);
    dy = dy/hy;
    ix = speye(Nx);
    Dy = kron(dy,ix);

% (squared) wave speeds
    if (med)
        al = al0*ones(Nx,Ny);
        om = om0*ones(Nx,Ny-1);
    else
        al = al0*rand(Nx,Ny);
        om = om0*rand(Nx,Ny-1);
    end
```

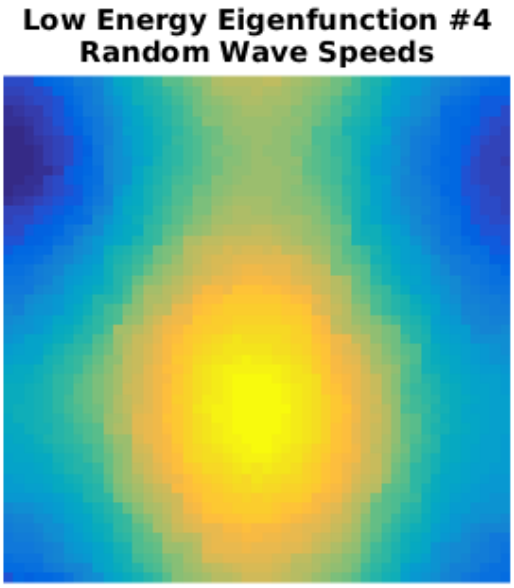
```
% second order differential operators
    Lx = Dx'*spdiags(al(:),0,Nx*Ny,Nx*Ny)*Dx;
    Ly = Dy'*spdiags(om(:),0,Nx*(Ny-1),Nx*(Ny-1))*Dy;
    L  = Lx + Ly;

% eigenspace decomposition
    [V,D] = eig(full(L));

% set up figure
    h1 = figure(1);
    close(h1);
    h1 = figure(1);
    set(h1,'Position',[10 30 800 800]);

% display high energy states
    for i=1:4
        subplot(2,2,i)
        imagesc(reshape(V(:,Nx*Ny-i+1),Nx,Ny))
        axis image; axis off;
        if (med)
            title(sprintf(['High Energy Eigenfunction #%0.0f\n', ...
                           'Uniform Wave Speeds'],Nx*Ny-i+1))
        else
            title(sprintf(['High Energy Eigenfunction #%0.0f\n', ...
                          'Random Wave Speeds'],Nx*Ny-i+1))
        end
    end

% set up figure
    h2 = figure(2);
    close(h2);
    h2 = figure(2);
    set(h2,'Position',[10 30 800 800]);

% display low energy states
    for i=1:4
        subplot(2,2,i)
        imagesc(reshape(V(:,i),Nx,Ny))
        axis image; axis off;
        if (med)
            title(sprintf(['Low Energy Eigenfunction #%0.0f\n', ...
                           'Uniform Wave Speeds'],i))
        else
            title(sprintf(['Low Energy Eigenfunction #%0.0f\n', ...
                           'Random Wave Speeds'],i))
        end
    end
```

For the case that α and ω are random throughout Ω, the lower energy eigenfunctions associated with the 4 smallest eigenvalues are

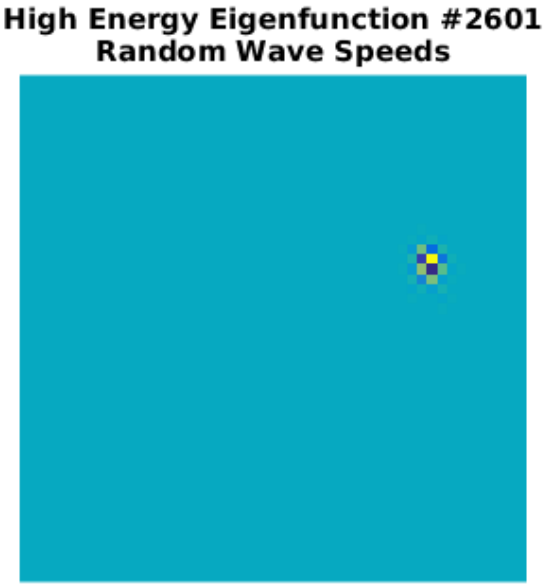


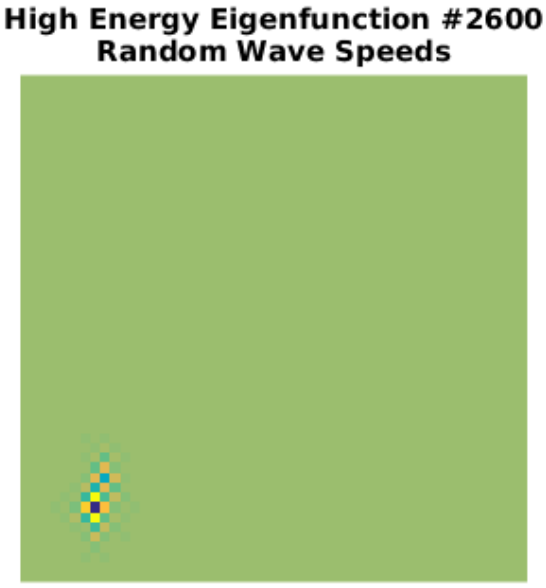


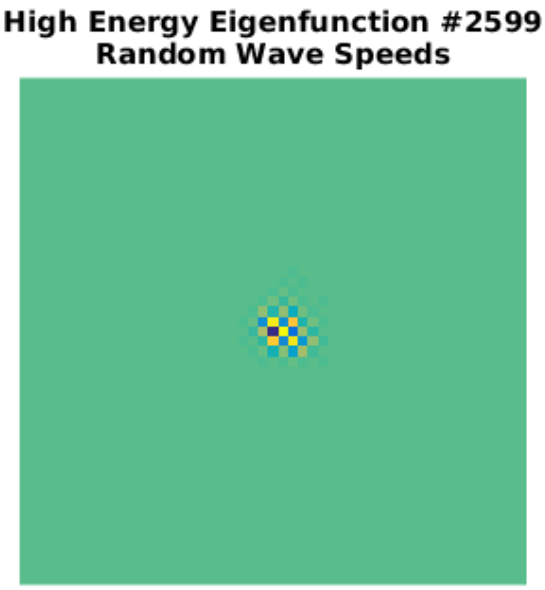


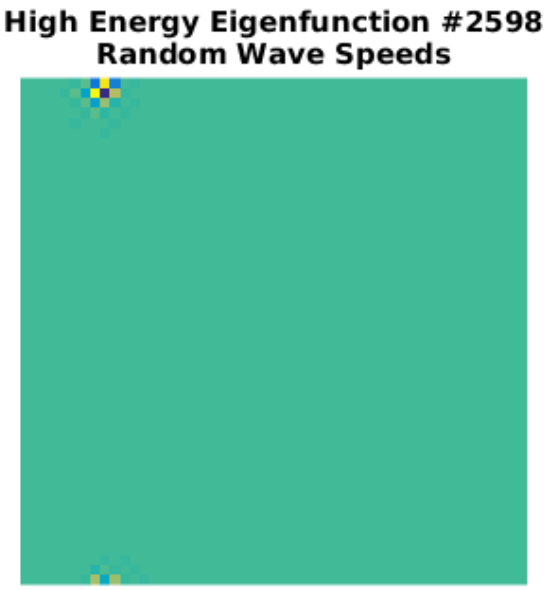


and the higher energy eigenfunctions associated with the 4 largest eigenvalues are

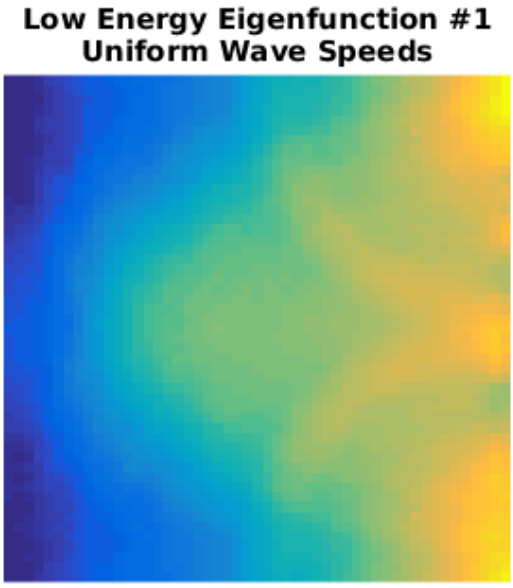


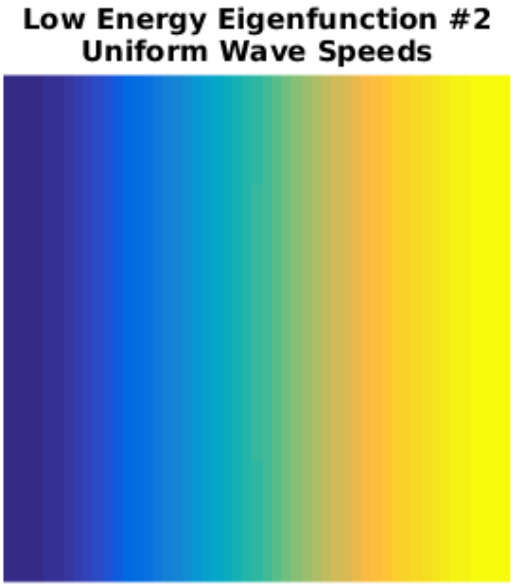


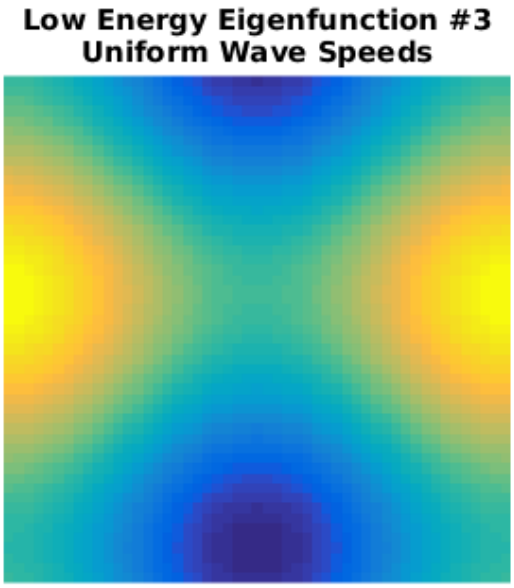


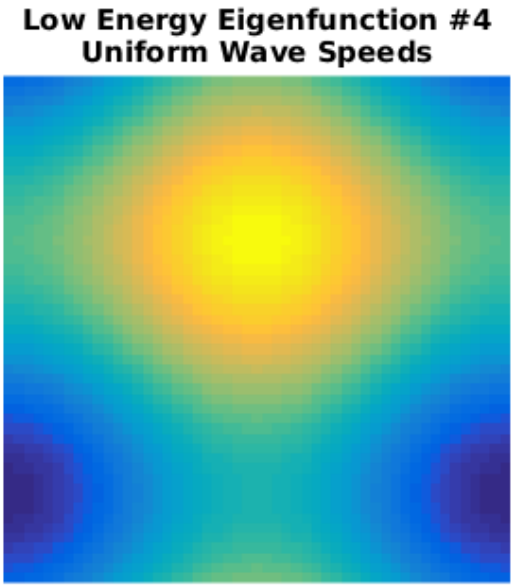


For the case that α and ω are uniform throughout Ω, the lower energy eigenfunctions associated with the 4 smallest eigenvalues are

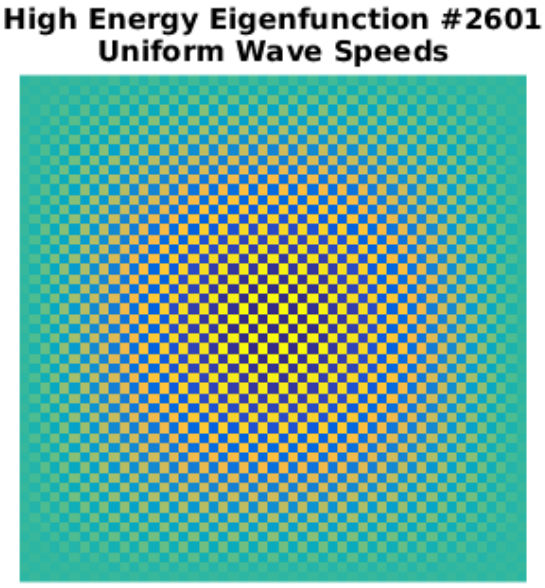


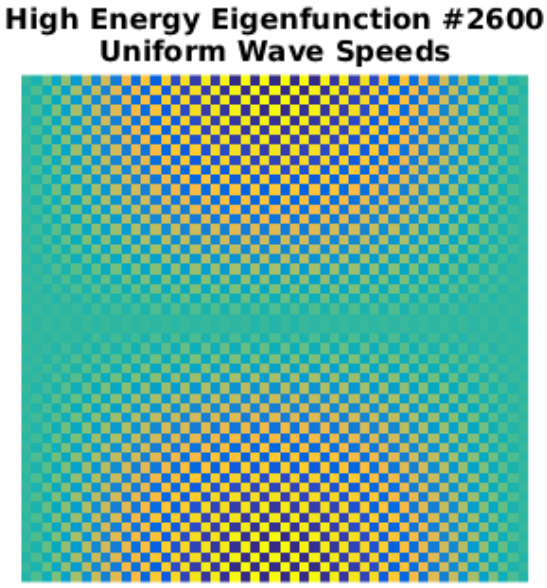


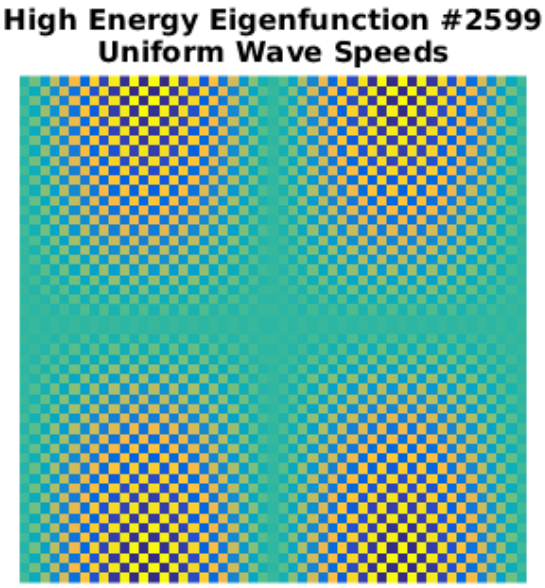


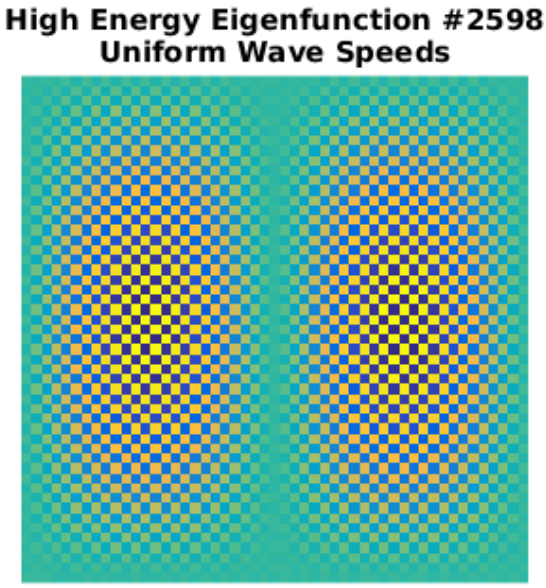


and the higher energy eigenfunctions associated with the 4 largest eigenvalues are









Note that the forms of the low energy states are comparable when the wave speeds are either random or uniform, although those associated with random wave speeds are noisier. Yet the forms of the

high energy states are conspicuously more localized for the random wave speeds in relation to broadly supported one for the uniform wave speeds.

• Exercise 5: Comparison of Membrane Models

◦ Task

Summarize the functionals used in the lecture to model membranes or strings and explain under what circumstances the minimizing or stationary state for one functional agrees with that of another functional.

◦ Solution

The energy functional on page 172 for the equilibrium state of a membrane defined on $\Omega \subset \mathbb{R}^2$ is

$$J_1(u) = \int_\Omega T\sqrt{1+|\nabla u|^2} - \int_\Omega fu$$

where $u : \Omega \to \mathbb{R}^2$ denotes the vertical displacement of a membrane parameterized by $(x, y) \subset \Omega$. The cerclage functional on pages 184-186 is a variant of this functional for a spherical geometry. The small displacement approximation of this functional is discussed on page 174,

$$J_2(u) = \int_\Omega \tfrac{1}{2}T|\nabla u|^2 - \int_\Omega fu$$

The energy functional on page 208 for the equilibrium state of a membrane defined on $\Omega \subset \mathbb{R}^2$ undergoing large deformations is

$$J_3(\boldsymbol{u}) = \int_\Omega \tfrac{1}{2}T(\|\boldsymbol{u}_\xi \times \boldsymbol{u}_\eta\| - 1)^2 - \int_\Omega \boldsymbol{f}\cdot\boldsymbol{u}$$

where $\boldsymbol{u} : \Omega \to \mathbb{R}^3$ denotes the arbitrary three-dimensional displacement of the membrane parameterized by $(\xi, \eta) \in \Omega$. The taut state approximation is discussed on page 206,

$$J_4(\boldsymbol{u}) = \int_\Omega \tfrac{1}{2}T\|\boldsymbol{u}_\xi \times \boldsymbol{u}_\eta\|^2 - \int_\Omega \boldsymbol{f}\cdot\boldsymbol{u}$$

Each of these functionals has its counterpart for a string in case $\Omega \subset \mathbb{R}^2$ and $\nabla u = u_x$. All of these functionals have their dynamic counterpart which is related to the principle of least action or Newton's Law, as discussed on page 204.

To establish a common notation for these functionals, assume for the latter two that the motion is restricted to be vertical so that $\xi = x$, $\eta = y$, $\boldsymbol{f}(x, y) = (0, 0, f(x, y))$, and $\boldsymbol{u}(x, y) = (x, y, u(x, y))$. Then with $\boldsymbol{u}_x = (1, 0, u_x)$ and $\boldsymbol{u}_y = (0, 1, u_y)$,

$$\|\boldsymbol{u}_\xi \times \boldsymbol{u}_\eta\| = \sqrt{1+|\nabla u|^2}$$

Thus, the functional J_4 agrees naturally with functional J_2, since a very taut state is only inclined to undergo correspondingly small displacements. For the relation between J_1 and J_3 consider that the integrands are a factor ϕ of one another

$$\tfrac{1}{2}[(1+|\nabla u|^2) - 2\sqrt{1+|\nabla u|^2} + 1] = \tfrac{1}{2}(\|\boldsymbol{u}_\xi \times \boldsymbol{u}_\eta\| - 1)^2 = \phi\sqrt{1+|\nabla u|^2}$$

when

$$\phi = \frac{1}{2}\left[\sqrt{1+|\nabla u|^2} + \frac{1}{\sqrt{1+|\nabla u|^2}}\right] - 1 \in [0,\infty).$$

With respect to the subsequent argument, note on the other hand that the factor ψ in

$$\tfrac{1}{2}[(1+|\nabla u|^2) - 2\sqrt{1+|\nabla u|^2}+1] = \tfrac{1}{2}(\|\boldsymbol{u}_\xi \times \boldsymbol{u}_\eta\| - 1)^2 = \psi\left[\sqrt{1+|\nabla u|^2}-1\right]$$

satisfies

$$\psi = \sqrt{1+|\nabla u|^2} - 1 \in [0,\infty).$$

The relation between J_1 and J_3 is revealed more directly when it is considered that J_1 actually involves surface area differences,

$$J_1(u) \to \tilde{J}_1(u) = \int_\Omega T\left[\sqrt{1+|\nabla u|^2}-1\right] - \int_\Omega fu$$

where the neglected term $-\int_\Omega T$ does not affect the minimizing state. Also, as hinted on page 200, J_3 has its Hookean counterpart for $\|\boldsymbol{u}_\xi \times \boldsymbol{u}_\eta\|$ sufficiently small

$$J_3(\boldsymbol{u}) \approx \tilde{J}_3(\boldsymbol{u}) = \int_\Omega T\left[\|\boldsymbol{u}_\xi \times \boldsymbol{u}_\eta\| - 1\right] - \int_\Omega \boldsymbol{f}\cdot\boldsymbol{u}$$

In this way the functional forms of $\tilde{J}_1$ and $\tilde{J}_3$ can be seen to agree.