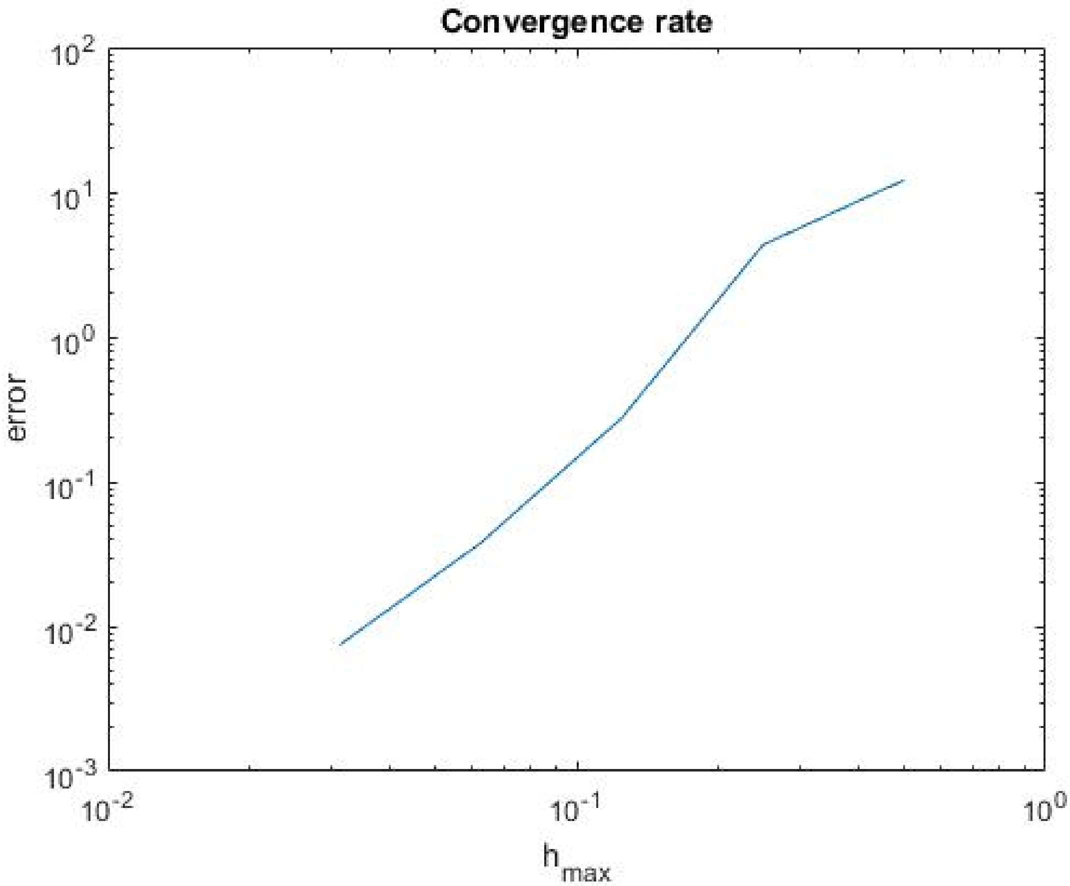
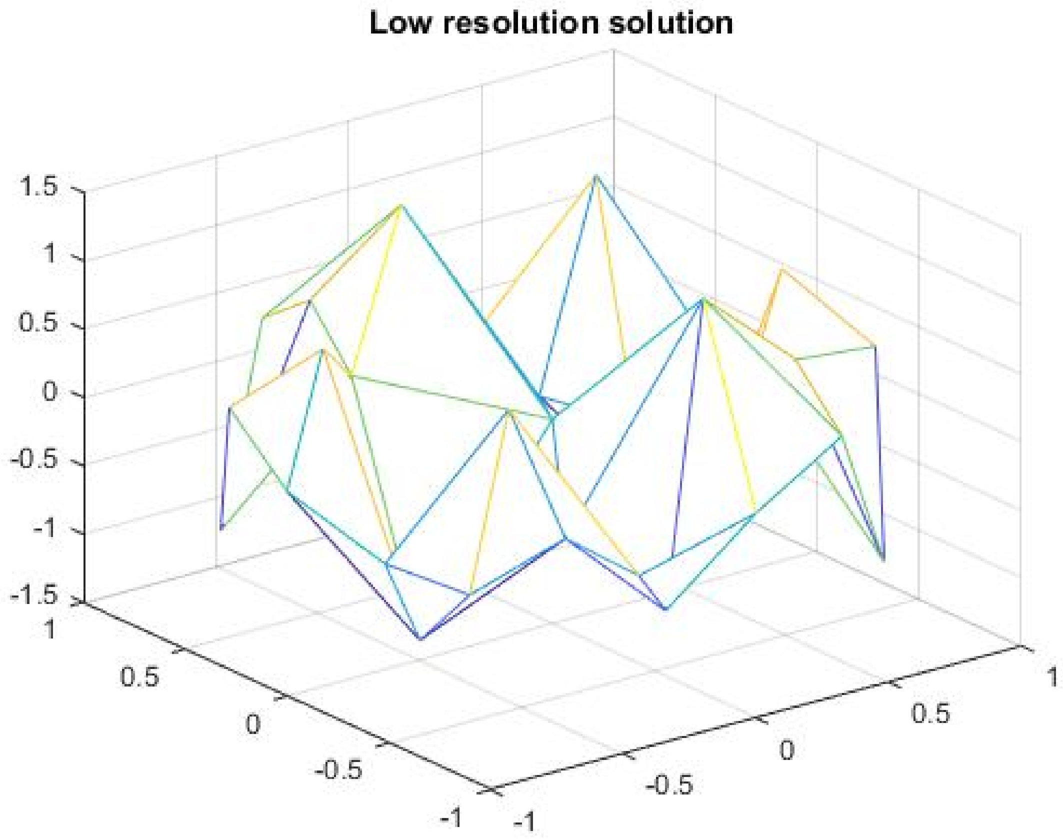
Applied FEM
Assignment
Peter Kardos

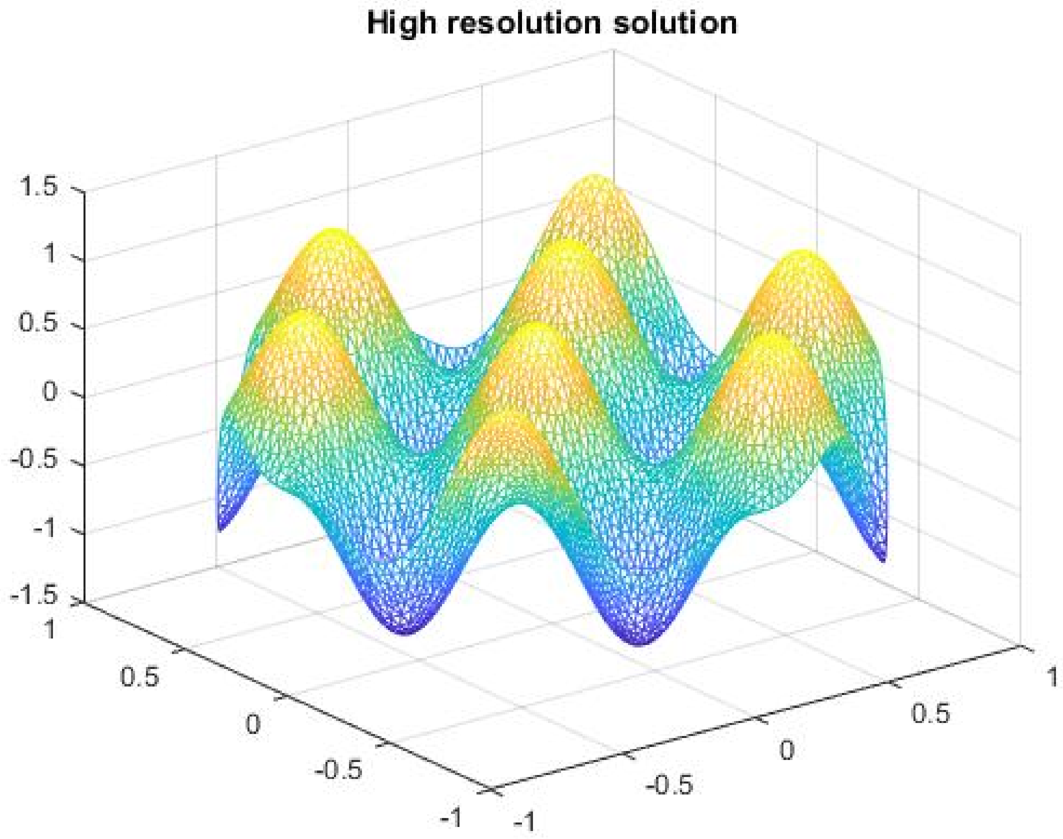
Part B

Part1 graphs

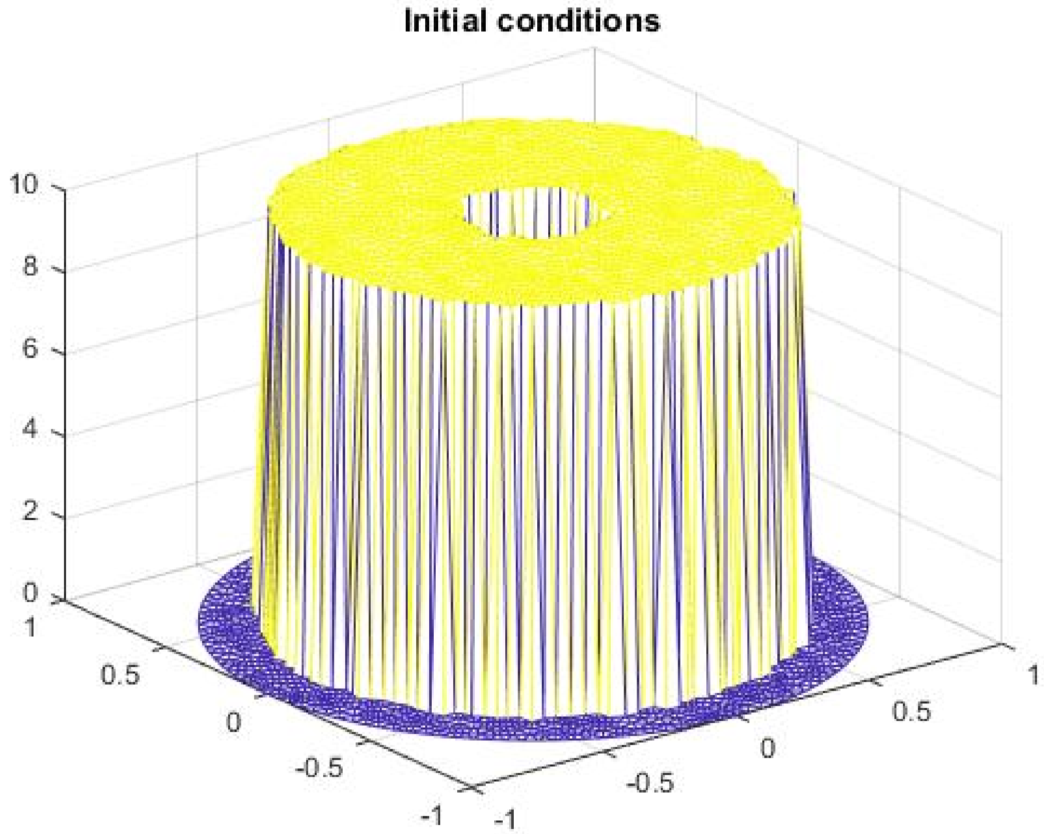


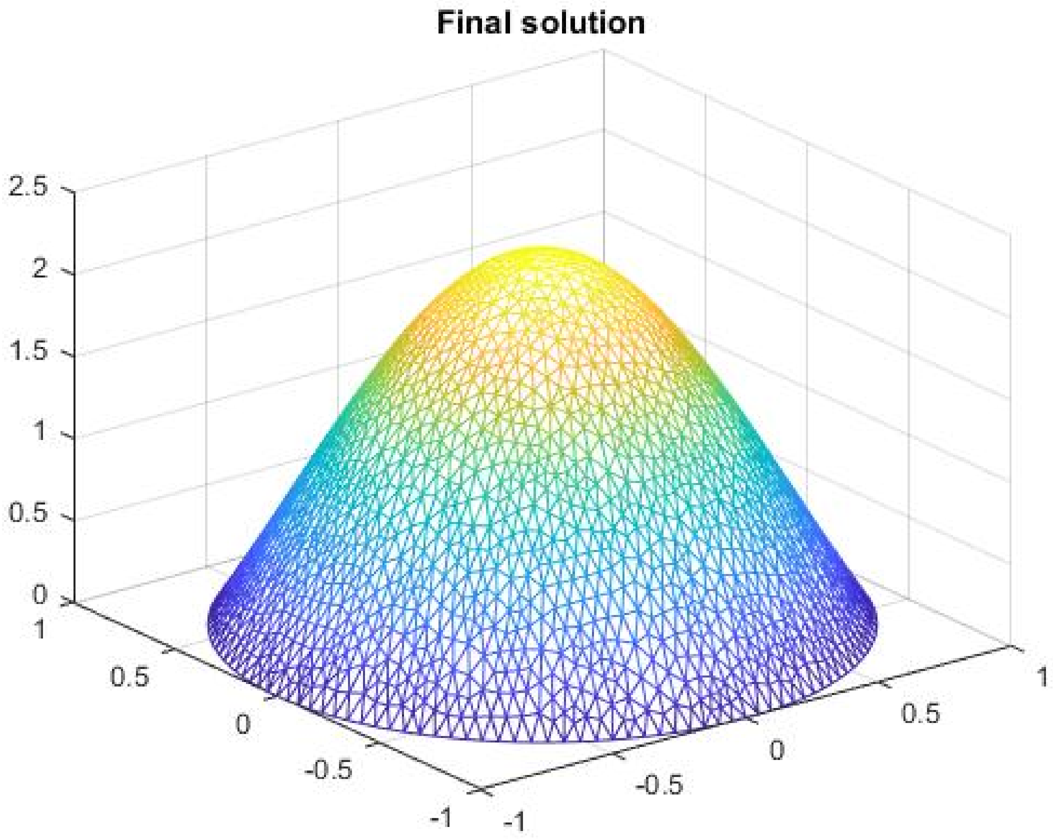


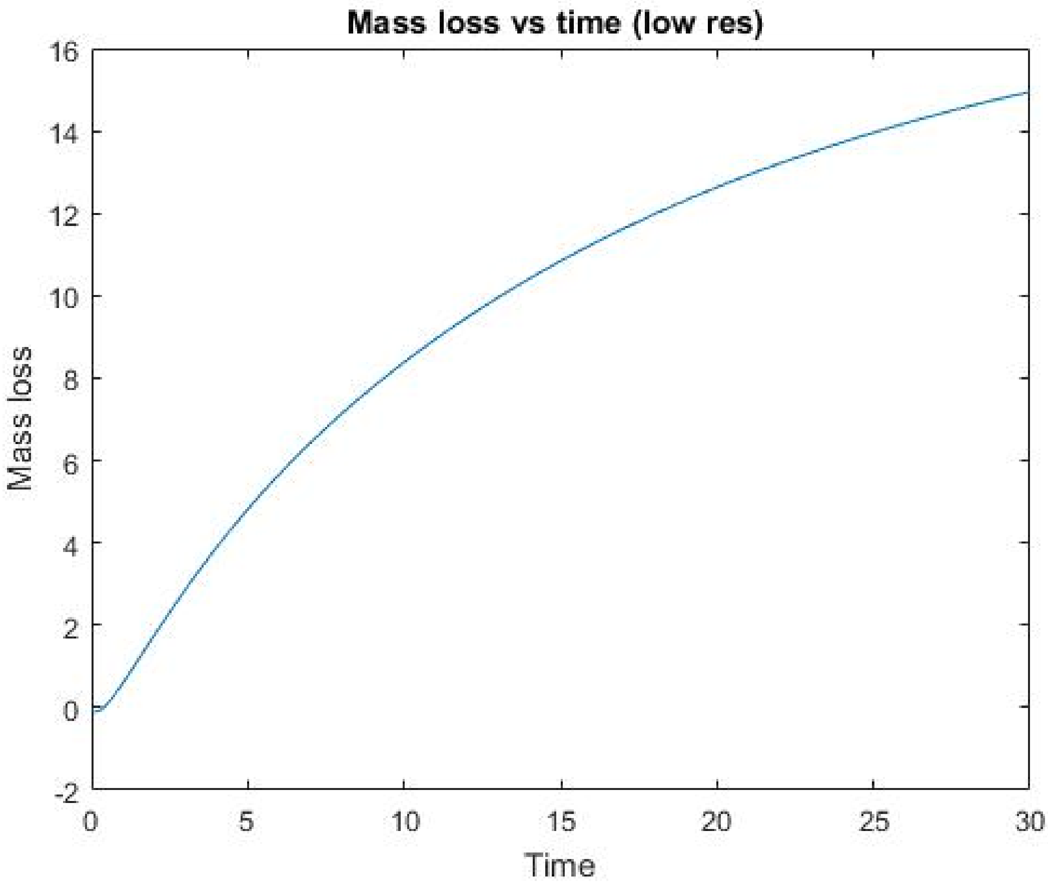
Low resolution solution
1.5
1
0.5
0
-0.5
-1
-1.5
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

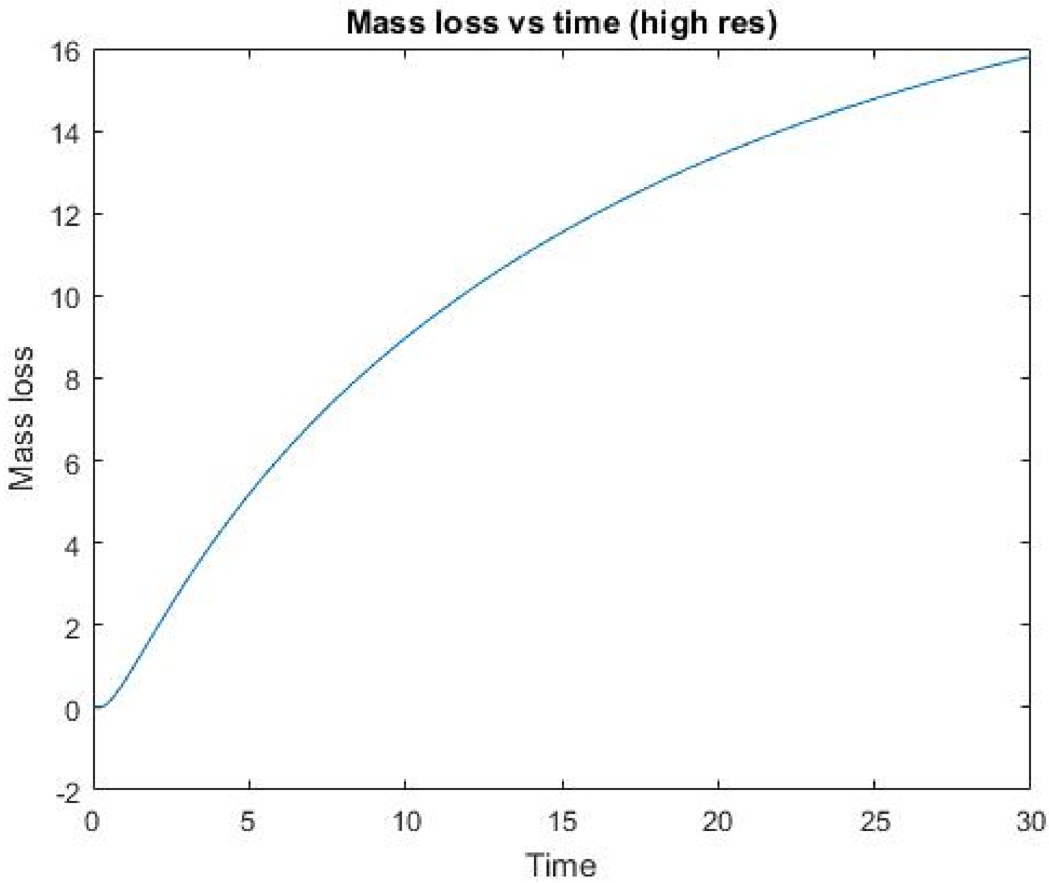

High resolution solution
1.5
1
0.5
0
-0.5
-1
-1.5
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

Part2 graphs




Final solution
2.5
2
1.5
1
0.5
0
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1


Mass loss vs time (low res)
Mass loss
Time
16
14
12
10
8
6
4
2
0
-2
0
5
10
15
20
25
30


Mass loss vs time (high res)
Mass loss
Time
16
14
12
10
8
6
4
2
0
-2
0
5
10
15
20
25
30

task2_p1.m

```
% mesh
geometry = @circleg;

% exact solution
u_exact = @(x) sin(2*pi*x(1,:)).*sin(2*pi*x(2,:));

% perturbation function
f = @(x) 8*pi^2*sin(2*pi*x(1,:)).*sin(2*pi*x(2,:));


hmax = 1./(2.^(1:5))';
err_norm = zeros(length(hmax),1);

for i=1:length(hmax)
    % mesh params and generation
    [p,e,t] = initmesh(geometry, 'hmax', hmax(i));

    % assemble and solve linear system
    A = stiffness_matrix(p, e, t);
    b = load_vector(p, e, t, f);

    I = eye(length(p));
    A(e(1,:),:) = I(e(1,:),:);
    b(e(1,:)) = u_exact(p(:,e(1,:)));

    c = A\b;
    c_exact = u_exact(p)';

    % save lowest resolution mesh
    if (i==1)
        p_worst = p;
        t_worst = t;
        c_worst = c;
    end

    % calculate error
    err = c_exact - c;
    err_norm(i) = err'*A*err;
end

% plot error
figure(1);

loglog(hmax, err_norm);
```

```
polycoeff = polyfit(log(hmax), log(err_norm), 1);
title('Convergence rate');
xlabel('h_{max}');
ylabel('error');
disp(['error ~ hmax^', num2str(polycoeff(1))]);

% plot solution and reference
figure(2);
trimesh(t_worst(1:3,:)', p_worst(1,:), p_worst(2,:), c_worst);
title("Low resolution solution");
figure(3);
trimesh(t(1:3,:)', p(1,:), p(2,:), c);
title("High resolution solution");
figure(4);
trimesh(t(1:3,:)', p(1,:), p(2,:), c_exact);
title("Exact solution");
```

task2_p2.m

```
hmax_v = [1/5, 1/20];
for rep=1:2

% mesh
geometry = @circleg;
hmax = hmax_v(rep);
[p,e,t] = initmesh(geometry, 'hmax', hmax);
N = length(p);

% properties
f = @(x) 0;
alpha = 0.01;
R = 0.5;
r = 0.3;
deltaT = 0.05;
Tend = 30;
rho = 10;

% initial condition
c_initial = zeros(N, 1);
for i=1:N
    pt = p(:, i);
    if (abs(R - norm(pt)) < r)
        c_initial(i) = rho;
    end
end


% assemble and solve linear system
M = mass_matrix(p, e, t);
A = alpha*stiffness_matrix(p, e, t);
b = load_vector(p, e, t, f);

G = (0.5/deltaT)*M + (1/2)*A;
H = (-0.5/deltaT)*M + (1/2)*A;

I = speye(N);
G(e(1,:),:) = I(e(1,:),:);

% plot initial state
figure(1);
trimesh(t(1:3,:)', p(1,:), p(2,:), c_initial);
title('Initial conditions');

% iterate solver
```

```
c_prev = c_initial;
figure(2);
animplot = trimesh(t(1:3,:)', p(1,:), p(2,:), c_initial);
ml = zeros(1);

iter = 1;
for time=0:deltaT:Tend
    % solve system
    d = b - H*c_prev;
    d(e(1,:)) = 0;

    %c = G\d;
    [c, flag] = bicg(G, d);
    c_prev = c;

    % calculate mass loss
    ml(iter) = mass_loss( p, t, c_initial, c);

    % plot animation
        if (mod(iter, 10) == 1 && rep==2)
        trimesh(t(1:3,:)', p(1,:), p(2,:), c);
        axis manual
        axis([-1, 1, -1, 1, min(c_initial)-1, max(c_initial)+1])
            ;
        title(['T = ', num2str(time)]);
        drawnow;
    end

    iter = iter + 1;
end
set(gcf,'visible','off');


if (rep == 2)
figure(3);
trimesh(t(1:3,:)', p(1,:), p(2,:), c_prev);
title('Final solution');
end

figure(3+rep);
plot(0:deltaT:Tend, ml);
restext = [' (low res) '; ' (high res)'];
title(['Mass loss vs time', restext(rep, :)]);
xlabel('Time');
ylabel('Mass loss');
```

end

get_element_transform.m

```
function [ M ] = get_element_transform( p1, p2, p3 )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
    b11 = 0; b12 = 0;
    b21 = 1; b22 = 0;
    b31 = 0; b32 = 1;
    A = [...
        b11, b12, 0, 0, 1, 0;...
        0, 0, b11, b12, 0, 1;...
        b21, b22, 0, 0, 1, 0;...
        0, 0, b21, b22, 0, 1;...
        b31, b32, 0, 0, 1, 0;...
        0, 0, b31, b32, 0, 1;...
    ];
    b = [p1(1), p1(2), p2(1), p2(2), p3(1), p3(2)]';
    Mvec = A\b;
    M = zeros(2,2);
    M(1,1) = Mvec(1);
    M(1,2) = Mvec(2);
    M(2,1) = Mvec(3);
    M(2,2) = Mvec(4);
end
```

load_vector.m

```
function [ b ] = load_vector(p, e, t, func)
    N = size(p, 2);

    b = zeros(N,1);

    for i=1:size(t, 2)
       i1 = t(1, i);
       i2 = t(2, i);
       i3 = t(3, i);
       p1 = p(:, i1);
       p2 = p(:, i2);
       p3 = p(:, i3);

       zi1 = func(p1) / 3;
       zi2 = func(p2) / 3;
       zi3 = func(p3) / 3;

       p1 = p1 - p3;
       p2 = p2 - p3;
       area = abs(p1(1)*p2(2) - p1(2)*p2(1))/2;

       b(i1) = b(i1) + zi1*area;
       b(i2) = b(i2) + zi2*area;
       b(i3) = b(i3) + zi3*area;
    end
end
```

mass_loss.m

```
function [ deltaM ] = mass_loss( p, t, c_initial, c )
%MASS_LOSS Summary of this function goes here
%   Detailed explanation goes here
    diff = c_initial - c;

    sum = 0;

    for i=1:size(t, 2)
       d = diff(t(1:3, i));

       i1 = t(1, i);
       i2 = t(2, i);
       i3 = t(3, i);
       p1 = p(:, i1);
       p2 = p(:, i2);
       p3 = p(:, i3);
       p1 = p1 - p3;
       p2 = p2 - p3;
       area = abs(p1(1)*p2(2) - p1(2)*p2(1))/2;

       sum = sum + mean(d)*area;
    end

    deltaM = sum;
end
```

mass_matrix.m

```
function [ M ] = mass_matrix( p, e, t )
%MASS_MATRIX Summary of this function goes here
%   Detailed explanation goes here
    N = size(p, 2);

    M = sparse(N, N);

    % integrate gradient for elements
    for i=1:size(t, 2)
        i1 = t(1, i);
        i2 = t(2, i);
        i3 = t(3, i);
        p1 = p(:, i1);
        p2 = p(:, i2);
        p3 = p(:, i3);
        v11 = integrate_phii_phij(p1, p2, p3, 1, 1);
        v12 = integrate_phii_phij(p1, p2, p3, 1, 2);
        v13 = integrate_phii_phij(p1, p2, p3, 1, 3);
        v22 = integrate_phii_phij(p1, p2, p3, 2, 2);
        v23 = integrate_phii_phij(p1, p2, p3, 2, 3);
        v33 = integrate_phii_phij(p1, p2, p3, 3, 3);

        M(i1, i2) = M(i1, i2) + v12;
        M(i1, i3) = M(i1, i3) + v13;
        M(i2, i3) = M(i2, i3) + v23;

        M(i1, i1) = M(i1, i1) + v11;
        M(i2, i2) = M(i2, i2) + v22;
        M(i3, i3) = M(i3, i3) + v33;

        M(i2, i1) = M(i2, i1) + v12;
        M(i3, i1) = M(i3, i1) + v13;
        M(i3, i2) = M(i3, i2) + v23;
    end
end


function [v] = integrate_phii_phij(p1, p2, p3, i, j)
    M = get_element_transform(p1, p2, p3);
    scale = det(M);

    Ipre = [...
        1/6     1/12     1/12;...
        1/12     1/6     1/12;...
        1/12     1/12     1/6;...
```

```
    ];

    d = Ipre(i,j);
    v = scale*d;
end
```

stiffness_matrix.m

```
function [ A ] = stiffness_matrix( p, e, t )
%STIFFNESS_MATRIX Summary of this function goes here
%   Detailed explanation goes here
    N = size(p, 2);

    A = sparse(N, N);

    % integrate gradient for elements
    for i=1:size(t, 2)
        i1 = t(1, i);
        i2 = t(2, i);
        i3 = t(3, i);
        p1 = p(:, i1);
        p2 = p(:, i2);
        p3 = p(:, i3);
        v11 = integrate_dphii_dphij(p1, p2, p3, 1, 1);
        v12 = integrate_dphii_dphij(p1, p2, p3, 1, 2);
        v13 = integrate_dphii_dphij(p1, p2, p3, 1, 3);
        v22 = integrate_dphii_dphij(p1, p2, p3, 2, 2);
        v23 = integrate_dphii_dphij(p1, p2, p3, 2, 3);
        v33 = integrate_dphii_dphij(p1, p2, p3, 3, 3);

        A(i1, i2) = A(i1, i2) + v12;
        A(i1, i3) = A(i1, i3) + v13;
        A(i2, i3) = A(i2, i3) + v23;

        A(i1, i1) = A(i1, i1) + v11;
        A(i2, i2) = A(i2, i2) + v22;
        A(i3, i3) = A(i3, i3) + v33;

        A(i2, i1) = A(i2, i1) + v12;
        A(i3, i1) = A(i3, i1) + v13;
        A(i3, i2) = A(i3, i2) + v23;
    end
end


function [v] = integrate_dphii_dphij(p1, p2, p3, i, j)
    M = get_element_transform(p1, p2, p3);
    MiT = inv(M)';
    scale = det(M);

    dphi = [-1, 1, 0;...
            -1, 0, 1];
```

```
    d = (MiT*dphi(:,i))' * (MiT*dphi(:,j)) / 2;
    v = scale*d;
end
```

unit_phi_mc.m

```
% monte-carlo integrates phii*phij

N = 10000000;
points = rand(2, N);

f1 = @(x) [-1, -1]*x + 1;
f2 = @(x) [1, 0]*x + 0;
f3 = @(x) [0, 1]*x + 0;

for i=1:N
    p = points(:, i);
    % reflect points to keep it in unit triangle instead of unit
        square
    if (p(1) + p(2) > 1)
        points(:, i) = [1; 1] - p;
    end
end

M = zeros(3,3);

for i=1:3
    for j=i:3
        if (i==1)
            a = f1;
        elseif (i==2)
            a = f2;
        else
            a = f3;
        end
        if (j==1)
            b = f1;
        elseif (j==2)
            b = f2;
        else
            b = f3;
        end
        f = @(x) a(x).*b(x);
        r = f(points);
        int = mean(r);
        M(i,j) = int;
        M(j,i) = int;
    end
end
%scatter(points(1,:),points(2,:));
disp(M);
```