CS4220 Project 3: MEMS Madness

Due: May 4, 2015

1 Introduction

Micro-Electro-Mechanical Systems (MEMS) are tiny coupled-physics systems that are often built using the same basic processes one uses to make integrated circuits. At characteristic length scales on the order of microns (millionths of a meter), the physics of these devices are still predominantly classical, but the dominant forces might be different than what you would expect. For example, electrostatic attraction can be an incredibly strong force at these scales, and is often used to drive motion.

You are provided with a small finite element code to simulate the displacement of a cantilever beam suspended over an electrode; the geometry is shown in Figure 1. The beam and the electrode are at different voltages, and so attract each other; but elastic forces resist that attraction. The basic force balance equation is

$$Ku = f_e(u, V), \tag{1}$$

where K is a positive definite stiffness matrix and f_e is the electrostatic force for a given displacement and voltage.

The electrostatic forces tend to pull the beam toward the electrode, and scale like with $(g + u)^2$, where g is the initial gap and u is the displacement. The elastic restoring forces, in contrast, scale linearly with u. Hence, at some critical voltage, the electrostatic forces become so strong that the elastic forces cannot balance them, and the system becomes unstable. At this point, the beam is "pulled in" to the electrode underneath.

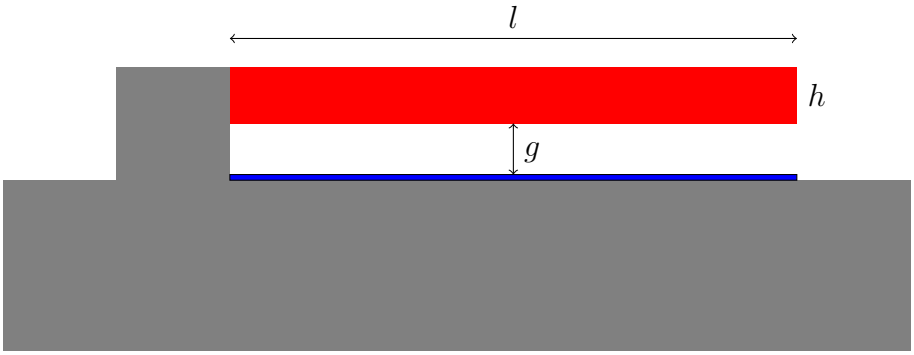

Figure 1: Cantilever beam gap-closing actuator. A potential difference between the beam (red) and an electrode along the substrate (blue) causes the beam to be pulled down.

2 The approaches

2.1 Fixed point iteration

The `gap_solve_fpV` code solves for the deflection in the model problem using the fixed point iteration

$$Ku^{(k+1)} = f_e(u^{(k)}; V)$$

where f_e denotes the electrostatic force vector. The driver `gap_demo` shows how this code is used.

1. Plot the residual error versus iteration count. Does this look like a linearly convergent iteration?

   Solution: The residual error versus iteration count is plotted in Figure(2). It is clear from the semi-log plot that the convergence rate is linear because between two adjacent iteration levels (from k to $k+1$) the ratio of the residual is about the same.

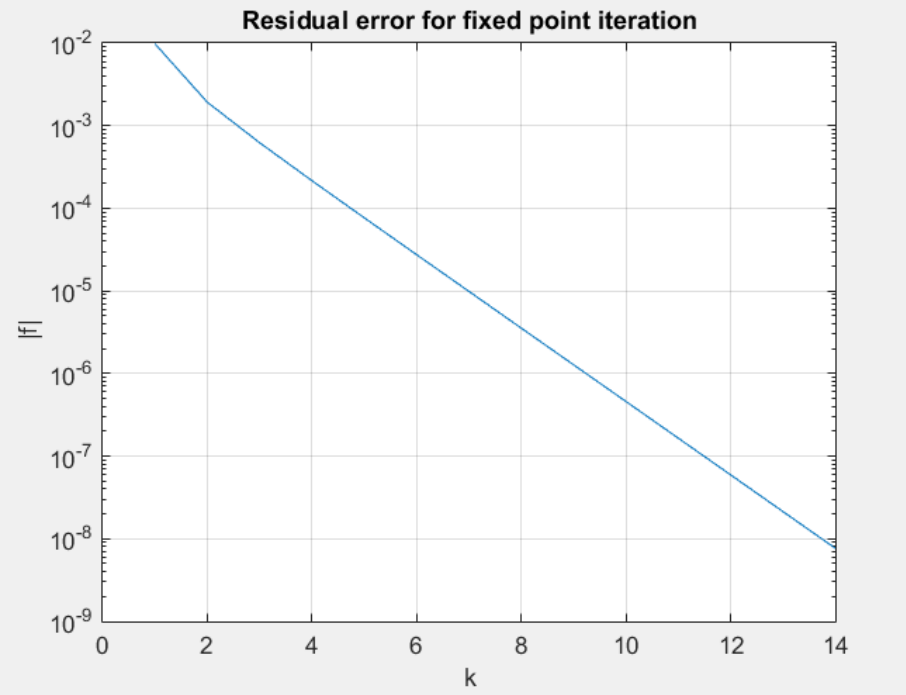

Figure 2: Residuals $|Ku^{(k)} - f^{(k)}|$ versus iteration count is a straightline, so the convergence rate is linear.

2. Write a new script, `gap_continuer0` that uses this fixed point iteration to solve the equation for voltages starting at $V = 0$ and going up to $V = 10$

(past the pull-in voltage). Use a continuation strategy in which the initial guess for u at each voltage is the equilbrium solution for the previous voltage (much like in PS7). Plot the tip displacement versus V for your computations. What is the largest voltage you can manage before the solver diverges?

Solution: As shown in Figure (3), the tip displacement increases from 0 (at $V = 0$, to about -0.32 μm at $V = 6.99$ V. The fixed point iteration would not converge when $V > 6.99$ V.

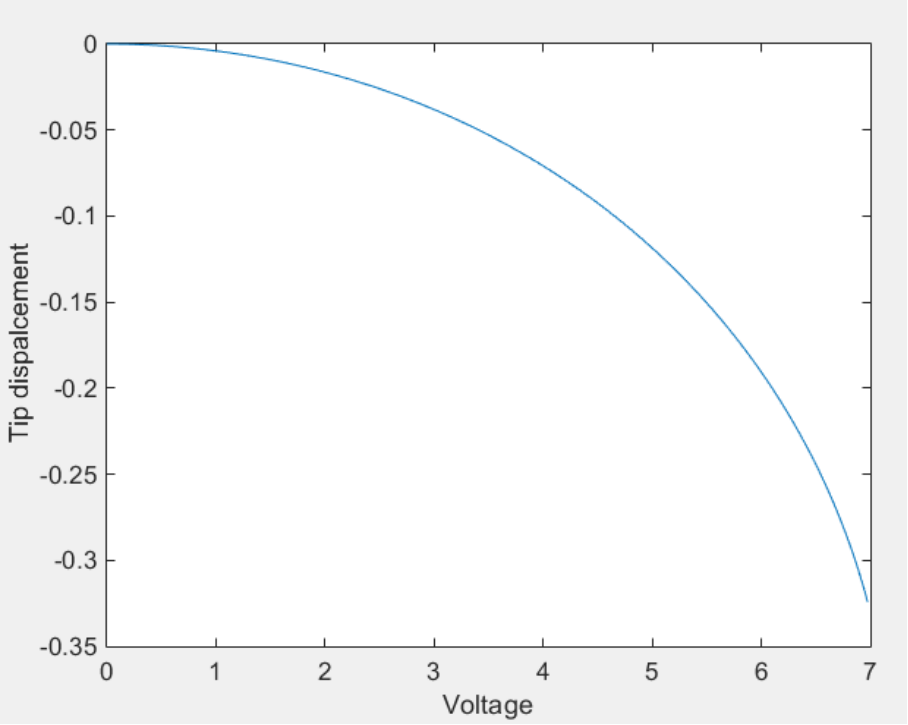

Figure 3: Tip displacement versus V computed by the fixed point iteration.

2.2 Newton iteration: Voltage control

The `gap_solve_fpV` code converges slowly for larger voltages, and eventually diverges as we approach pull-in. However, the `gap_force` routine computes both the electrostatic force $f_e(u; V)$, but also the Jacobian $\partial f_e / \partial u$[1]. Therefore, we can code a Newton iteration.

1. Complete the routine `gap_solve_NV` to compute the displacement at a fixed voltage using a Newton iteration. Check that the rate of convergence is quadratic!

[1]Like K, this Jacobian is a symmetric matrix.

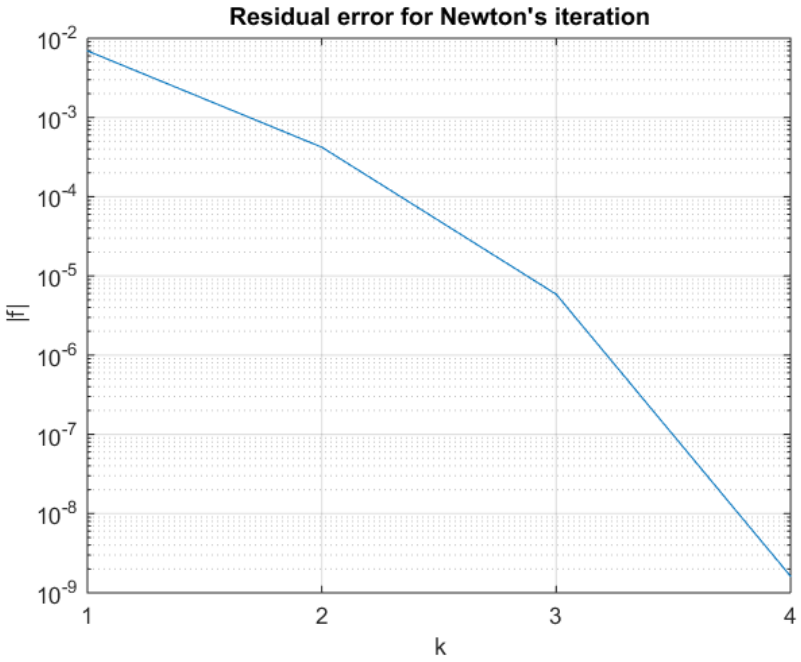

Figure 4: Plot of $\|Ku^k - f_e(u^k, V)\|$ versus iteration count for the same case shown in Figure (2).

Solution: In Eq.(1), we move f_e to the left, and we denote this force balance equation as F (a vector):

$$F = Ku - f_e(u, V) = 0. \tag{2}$$

Newton iteration for solving Eq.(2) is quite straightforward. The Jacobian matrix of F is

$$J_F = \frac{\partial F}{\partial u} = K - J_u,$$

where J_u is the derivative matrix $\partial f_e/\partial u$ (which is the Jacobian matrix of f_e).

The same sample problem $V = 6.3$ V is now solved by Newton's iteration and `gap_solve_NV` quickly converges. The residual plot shows a reduction of about 10^{-9} in 4 iterations. The reduction of the ratio of residual norms doubles between two consecutive iterations, so the convergence rate is quadratic.

2. Write a script, `gap_continuer1` that uses Newton iteration to solve the equation for varying voltages, similar to `gap_continuer0`. What is the largest voltage you can manage before the solver fails to adequately converge?

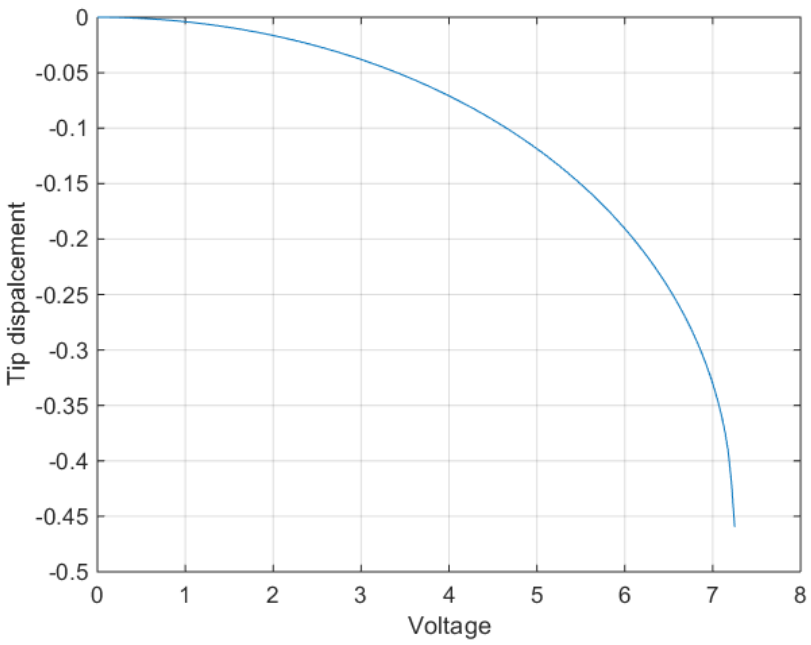

Figure 5: Tip displacement versus V computed by the Newton iteration (output from `gap_continuer1`).

Solution: Using Newtion iteration we can run `gap_continuer1` up to $V \approx 7.25$ V, and the corresponding tip displacement is about -0.42 μm. The plot is shown in Figure (5).

2.3 Newton iteration: Displacement control

So far, we have considered methods in which we control V and compute the corresponding u. We now consider control of one component of u (the tip displacement) and computation of the corresponding V along with the rest of u. That is, we want to solve

$$F(u, V; d) = \begin{pmatrix} Ku - f_e(u, V) \\ e_{tip}^T u - d \end{pmatrix} = 0$$

where e_{tip} is a vector that indicates the controlled degree of freedom (the displacement at the tip).

1. Complete the routine `gap_solve_Nd` to compute the displacement and voltage corresponding to a fixed tip displacement using a Newton iteration. I recommend using the displacement vector u and the *squared* voltage V^2 as your unknowns for the purpose of Newton iteration; otherwise, you will have a singular Jacobian at zero voltage and displacement.

Solution: In solving the equations for the displacement control problem, we add one additional equation to the original system. This additional equation specifies the tip displacement magnitude $e_{tip}^T u = d$, consequently the system computes the voltage V and the displacement vetor u that produces d:

$$F(u, V; d) = \begin{pmatrix} Ku - f_e(u; V) \\ e_{tip}^T u - d \end{pmatrix} = 0, \tag{3}$$

the system of equations F and solution vector z are (`ndof+1`) $\times$ 1 vectors:

$$F = \begin{pmatrix} F_1 & F_2 & \dots & F_n & F_{n+1} \end{pmatrix}^T, \tag{4}$$

$$z = \begin{pmatrix} u_1 & u_2 & \dots & u_n & V^2 \end{pmatrix}^T, \tag{5}$$

where F_{n+1} is simply the last element in Eq.(3):

$$F_{n+1} = e_{tip}^T u - d.$$

The system of Eq.(3) is closed because we solve for n=`ndof+1` unknowns with the same number of equations.

The Jacobian matrix J_F associated with $F(u, V; d)$ can be expressed as

$$J_F(u, V^2; d) = \begin{pmatrix} \frac{\partial F_1}{\partial u_1} & \frac{\partial F_1}{\partial u_2} & \cdots & \frac{\partial F_1}{\partial u_n} & \frac{\partial F_1}{\partial V^2} \\ \frac{\partial F_2}{\partial u_1} & \frac{\partial F_2}{\partial u_2} & \cdots & \frac{\partial F_2}{\partial u_n} & \frac{\partial F_2}{\partial V^2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{\partial F_n}{\partial u_1} & \frac{\partial F_n}{\partial u_2} & \cdots & \frac{\partial F_n}{\partial u_n} & \frac{\partial F_n}{\partial V^2} \\ \frac{\partial F_{n+1}}{\partial u_1} & \frac{\partial F_{n+1}}{\partial u_2} & \cdots & \frac{\partial F_{n+1}}{\partial u_n} & \frac{\partial F_{n+1}}{\partial V^2} \end{pmatrix}. \tag{6}$$

The assembly of J_F is described in more detail below:

- $[K] - [J_u]$ occupies the upper $n \times n$ block of J_F;
- entries in the $(n+1)$-th row (the bottom row) are all zero except for the element at column `Itip` which is 1:

$$J_F(n+1, \texttt{Itip}) = \frac{\partial(e_{tip}^T u - d)}{\partial u_{tip}} = 1.$$

- $\partial f_e/\partial V^2$ occupies the $(n+1)$-th column of J_F except that the last entry in the diagonal in Eq.(6) is zero, because $\partial(e_{tip}^T u - d)/\partial V^2 = 0$. The `gap_force` function computes not only $f_e(u, V)$ but also the derivatives $\partial f_e/\partial u$ (an $n \times n$ matrix) and $\partial f_e/\partial V^2$ (an $n \times 1$ vector).
- The fully assembled Jacobian matrix for the Newton iteration is thus given in Eq.(7).

$$J_F(u, V^2; d) = \left(\begin{array}{ccc|c} & & & -\frac{\partial f_e(u_1, V)}{\partial V^2} \\ & [K] - [J_u] & & \vdots \\ & & & -\frac{\partial f_e(u_n, V)}{\partial V^2} \\ \hline 0 & \dots & 1 \quad 0 & 0 \end{array}\right) \tag{7}$$

In MATLAB notation, J_F is entered as

```
K=pgap.K;
[f,Ju,JV]=gap_force(pgap,u,V);
etip=zeros(ndof,1);
etip(Itip)=1;
JF=[K-Ju-JV; etip' 0];
```

Because f is proportional to V^2, the benefit of using V^2, instead of V, as the variable is to avoid having a singular Jacobian matrix at zero V and zero displacement. This is clearly illustrated by J_F shown in Eq.(7).

The output of `gap_solve_Nd` is the voltage that produces the given tip displacement. The convergence is very fast (typically it converges in about 3 iterations).

2. Write a script, `gap_continuer2` that computes the state for tip displacements between zero and 80% of the gap. Use Newton iteration to solve the equation for varying displacements. Your script should produce a plot of tip displacement vs voltage for the full range of tip displacements considered.

   `gap_continuer2` solves the problem of finding the voltage by varying tip displacements. Figure (6) shows the tip displacement versus voltage plot for beam length equal to 100 μm.

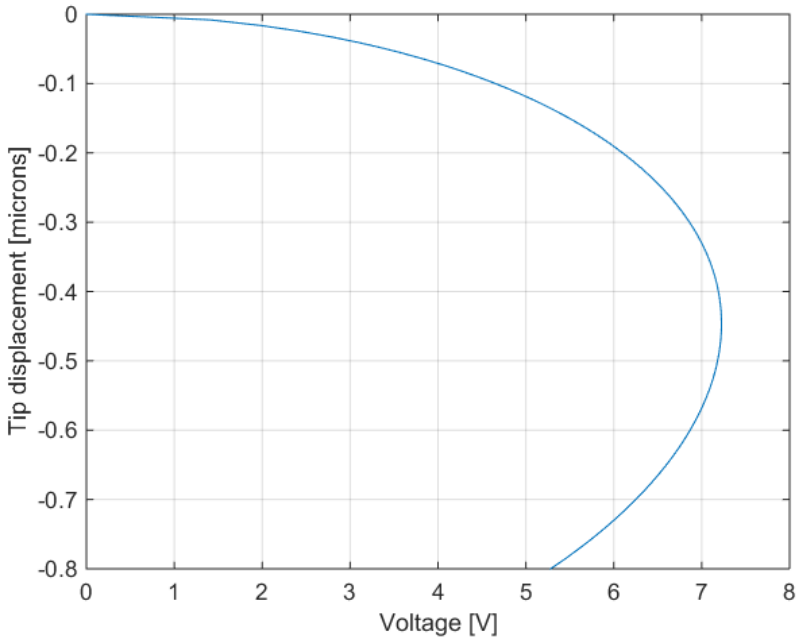

Figure 6: Tip displacement versus voltage plot for the beam of 100 μm in length.

2.4 Finding pull-in

The `gap_continuer2` code should give a pretty clear notion of the critical voltage (pull-in). In particular, the pull-in state is the point along the solution curve at which the Jacobian becomes singular. For the "top" part of the curve of deflection vs voltage, the equilibrium is stable, and the Jacobian matrix is positive definite; for the "bottom" part of the curve, the Jacobian is indefinite.; and the pull-in state is the transition between these two. It is possible to detect whether the Jacobian matrix is positive definite or not by attempting to run Cholesky factorization; that is,

```
[R,p] = chol(J);
if p
  disp('Indefinite');
else
  disp('Positive definite');
end
```

Using this test, it is possible to find the tip displacement associated with pull-in (and the associated voltage) by running bisection on the nonsingularity.

1. Complete the function `gap_pullin` to find the pull-in voltage. You may use the bisection strategy above, or you may do something different, but do make

sure to sanity check your code.

Solution: The function `gap_pullin` is completed by using the the Cholesky factorization for detecting whether the Jacobian matrix has changed from positive definite to indefinite, and a basic bisection strategy is employed in `gap_pullin` to find the pull-in voltage for a given beam. We add one local function `gap_matrix_PDtest` which returns a factor named `fp=-1` if the Jacobian matrix is positive definite, and `fp>0` if the Jacobian matrix is indefinite. (`gap_matrix_PDtest` function is inside file `gap_pullin.m`.)

2. Write a script `gap_pull_sweep` to compute the pull-in voltage for beam lengths between 20 and 100 microns, and show the results on a plot.

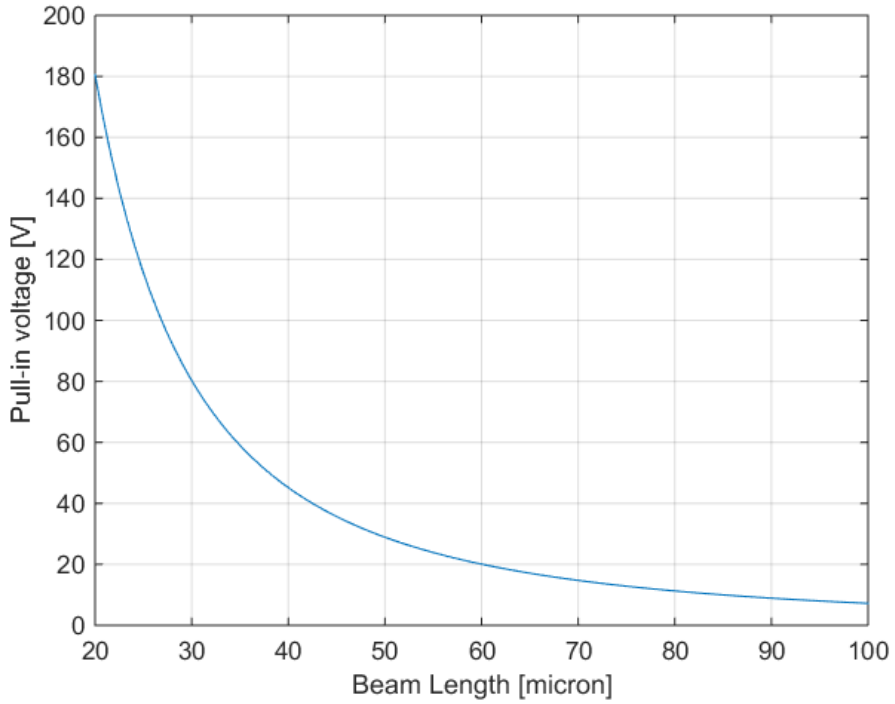

Figure 7: The plot of the pull-in voltage (V) versus beam length (L).

Solution: `gap_pull_sweep` computes the pull-in voltage for beam lengths between 20 and 100 μm. The results are plotted in Figure (7). As the length of the beam gets shorter, the convergence of the displacement control `gap_solve_Nd` becomes more difficult to reach. We have to slightly lower the criterion in order to get convergence over the whol range. The tolerance

of the residual norm of Eq.(3) for Newton's iteration in `gap_solve_Nd` has to be reduced from 10^{-8} to 5×10^{-7}.

3 Notes

1. This project is an example of *numerical bifurcation analysis*, an area often associated with continuation methods.

2. This model should not be taken all *that* seriously. It's a good first cut, but the model we use for the electrostatic field is a "parallel plate" approximation where we assume the potential varies linearly between the substrate and the beam.

3. It is possible to directly write down a system of equations that describes the pull-in state:

$$\begin{aligned} Ku - f_e(u, V) &= 0 \\ \left(K - \frac{\partial f_e}{\partial u}(u, V)\right) w &= 0 \\ c^T w - 1 &= 0 \end{aligned}$$

   where c can be chosen as a random vector. If you want, you can `gap_pullin` based on this augmented system rather than using the bisection technique recommended above.

4 Appendix Listing of Codes

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [u, rnorms] = gap_solve_fpV(pgap, u, V, rtol, maxiter)
%
% [u, rnorms] = gap_solve_fpV(pgap, u, V, rtol, maxiter)
%
% Attempt to solve the gap equation by a fixed point iteration.
%
% Inputs:
%   pgap: Structure returned by gap_setup
%   u: Initial guess of displacement vector
%   V: Voltage
%   rtol: Residual tolerance for convergence (default: 1e-8)
```

```
%   maxiter: Maximum iterations allowed (default: 20)
%
% Outputs:
%   u: Computed displacement vector
%   rnorms: Residual norms at each iteration
%
%function [u, rnorms] = gap_solve_fpV(pgap, u, V, rtol, maxiter)

  % Assign default tolerances if not specified
  if nargin < 4, rtol = 1e-8; end
  if nargin < 5, maxiter = 20; end

  % Main iteration
  K = pgap.K;
  [L,U] = lu(K);
  rnorms = zeros(maxiter,1);
  for iter = 1:maxiter
    f = gap_force(pgap, u, V);
    rnorms(iter) = norm(K*u-f);
    if rnorms(iter) < rtol
      rnorms = rnorms(1:iter);
      return;
    end
    u = U\(L\f);
  end

  % If we got here, we didn't converge in time
  warning('Fixed point iteration did not converge for V=%e', V)
end
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% gap_continuer0.m
%
% Fixed point continuation (controlling voltage)
%

% === Load gap finite element code
% argument in gap_setup: 'c': for cantilever (this problem)
%                         80: number of elements in the beam
pgap = gap_setup('c', 80);
```

```
nelt  = pgap.nelt;
Ce = pgap.Ce;
M = pgap.M;
K = pgap.K;
N = pgap.N;
wg = pgap.wg;
Itip  = pgap.Itip;
ndof = pgap.ndof; %ndof is the "active" degrees of freedom

% === Continuation loop
% V: voltage varying between 0 and 10
%
numV=200;
V=linspace(0,6.975,numV);

utip=zeros(numV,1);
u=zeros(ndof,1); % initial guess for u
for kk=1:numV
    [u, rnorms]=gap_solve_fpV(pgap, u, V(kk));
    utip(kk)=u(pgap.Itip);
end
% === Plot V vs tip displacement
plot(V, utip);
xlabel('Voltage');
ylabel('Tip_dispalcement');
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
function [u, rnorms] = gap_solve_NV(pgap, u, V, rtol, maxiter)
% [u, rnorms] = gap_solve_NV(pgap, u, V, rtol, maxiter)
%
% Attempt to solve the gap equation by Newton's method.
%
% Inputs:
%   pgap: Structure returned by gap_setup
%   u: Initial guess of displacement vector
%   V: Voltage
%   rtol: Residual tolerance for convergence (default: 1e-8)
%   maxiter: Maximum iterations allowed (default: 10)
```

```
%
% Outputs:
%   u: Computed displacement vector
%   rnorms: Residual norms at each iteration
%
%
  % Assign default tolerances if not specified
  if nargin < 4, rtol = 1e-8; end
  if nargin < 5, maxiter = 10; end
  %
  %
  K=pgap.K;
  %
  % Main iteration of Newton's method for F(u,V)=0:
  %   F(u,V)=Ku -f_e =0
  %
  F=zeros(pgap.ndof,1);
  rnorms=zeros(maxiter,1);
  for iter=1:maxiter
    % call gap_force() for f, and Jacobian Ju:
    [f, Ju]=gap_force(pgap, u, V);
    % calculate the residual norm of F=K*u-f
    F=K*u-f;
    rnorms(iter)=norm(F);
    % is it converged yet?
    if rnorms(iter) < rtol
        rnorms=rnorms(1:iter); %resize rnorms for output
        return;
    end
    % solve J*p_k = -F = -(K*u_k - f)
    JF=K-Ju;
    pvec=-JF\F;
    u=u+pvec;
  end

  % If we got here, we didn't converge in time
  warning('Newton iteration did not converge')
  warning('iter=%d, residual norm =%e, V=%e', iter, rnorms(iter), V)

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% gap_continuer1.m
%
%
% Newton continuation (controlling voltage)
%

% === Load gap finite element code

pgap = gap_setup('c', 80);

nelt = pgap.nelt;
Ce = pgap.Ce;
M = pgap.M;
K = pgap.K;
N = pgap.N;
wg = pgap.wg;
Itip = pgap.Itip;
ndof = pgap.ndof;

% === Continuation loop

numV=200;
V=linspace(0,7.26,numV);

utip=zeros(numV,1);
u=zeros(ndof,1); % initial guess for u
for iter=1:numV
    [u, rnorms]=gap_solve_NV(pgap, u, V(iter));
    utip(iter)=u(pgap.Itip);
end
% === Plot V vs tip displacement
plot(V, utip);
xlabel('Voltage');
ylabel('Tip_dispalcement');
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
function [u, V, rnorms] = gap_solve_Nd(pgap,u, V, d, rtol, maxiter)
% [u, V, rnorms] = gap_solve_Nd(pgap, u, V, d, rtol, maxiter)
%
% Attempt to solve the gap equation by Newton's method. Applies
```

```
% displacement control to the tip.
%
% Inputs:
%   pgap: Structure returned by gap_setup
%   u: Initial guess of displacement vector
%   V: Initial guess of voltage
%   d: Tip displacement
%   rtol: Residual tolerance for convergence (default: 1e-8)
%   maxiter: Maximum iterations allowed (default: 10)
%
% Outputs:
%   u: Computed displacement vector
%   V: Computed voltage level
%   rnorms: Residual norms at each iteration
%

% Assign default tolerances if not specified
if nargin < 5, rtol = 5e-7; end
if nargin < 6, maxiter = 10; end

nelt = pgap.nelt;
Ce = pgap.Ce;
M = pgap.M;
K = pgap.K;
N = pgap.N;
wg = pgap.wg;
ndof = pgap.ndof;

K=pgap.K;
Itip=pgap.Itip;

%set etip, the unit base vector for the tip displacement
etip=zeros(ndof,1);
etip(Itip)=1;
%
% Newton's method is applied to the
% vector equations as follows:
%
% F=[ Ku-f     ]
%   [ u(Itip)-d ] =0
%
```

```
%
F=zeros(ndof+1,1);
%
%
u = zeros(ndof,1);
%[u, rnorms1] = gap_solve_NV(pgap, u, V, rtol, 2);
rnorms=zeros(maxiter,1);
%
% the solution vector is called z:
z=[u;V^2];
% Main iteration
%
%fprintf('\n*** initial V=%e, d=%e\n', V, d);
for iter=1:maxiter
    % call gap_force() for f, and Jacobian matrices Ju and JV.
    % obtain V from the solution vector z:
    V=sqrt(z(end));
    %V=z(end);
    u=z(1:ndof);
    [f, Ju, JV]=gap_force(pgap, u, V);
    % calculate the residual norm of F:
    F=[K*u-f; u(Itip)-d];
    rnorms(iter)=norm(F);
    % is it converged yet?
    if rnorms(iter) < rtol
        rnorms=rnorms(1:iter); %resize rnorms for output
        return;
    end
    % solve J_F*p_k = -F
    % Construct the "expanded" Jacobian matrix:
    JF=[K-Ju -JV; etip' 0];
    %pvec=-pinv(JF)*F;
    pvec=-JF\F;
    %
    %pvec(end)=sqrt(pvec(end));
    z=z+pvec;
    %fprintf('\niter=%d, rnorms=%e', iter, rnorms(iter));
    %fprintf('\ntip Disp= %e, V=%g\n', u(Itip), sqrt(z(end)));
end
%
% If we got here, we didn't converge in time
```

```
warning('Newton iteration did not converge')
warning('iter=%d, residual norm =%e, V=%e', iter, rnorms(iter), V)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% gap_continuer2.m
%
% Newton point continuation (controlling tip displacement)
%
pgap = gap_setup('c', 80);
%num: number of tip displacements
num=100;
nelt = pgap.nelt;
Ce = pgap.Ce;
M = pgap.M;
K = pgap.K;
N = pgap.N;
wg = pgap.wg;
Itip = pgap.Itip;
ndof = pgap.ndof;
% as required: tip displacement from 0 to 80% of the gap size
grange=0.8*pgap.g;
tipDisp=linspace(0, grange, num);
tipDisp=-tipDisp;
V=0;
u=zeros(ndof,1);
Volt=zeros(num,1);
PDstate=zeros(num,1);
% === Continuation loop
for iter=1:num
    d=tipDisp(iter);
    [u, V, rnorms] = gap_solve_Nd(pgap, u, V, d);
    fprintf('\n***d=%g,   V=%g\n', d, V);
    Volt(iter)=V;
    %%%%%% adding a section of test code for PD of J:
    [f, Ju, JV]=gap_force(pgap,u, V);
    JF=K-Ju;
    [R,p]=chol(JF);
    PDstate(iter)=p;
end
```

```
% === Plot V vs tip displacement
plot(Volt, tipDisp);
grid on;
ylabel('Tip displacement [microns]');
xlabel('Voltage [V]');

%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [u,V] = gap_pullin(pgap)
% [u,V] = gap_pullin(pgap)
%
% Compute the pull-in displacement and voltage for the given
% structure.
%
% Inputs:
%    pgap: Structure returned by gap_setup
%
% Outputs:
%    u: Displacement vector at pull-in state
%    V: Pull-in voltage
%
% === Load gap finite element code

ndof = pgap.ndof;

% the following interval [disp1, disp2] must
% bracket the transition point of the curve
%
disp1=-0.8*pgap.g;
disp2=0;

%
%
V1=10;
V2=0.01;

% use BiSection method to find the pull-in voltage/displacement
% by using the test that the Jacobian [K -Ju]
% changes from positive definite (p==0)
% to indefinite (p>0). We change p==0 to p<0 for finding this root:
% we start from a=disp1 (lower bound) and b=disp2 (upper bound).
```

```
% | V_pullin − p | < atol
% Bisection code below:
%
a=disp1;
b=disp2;
u=zeros(ndof,1);
u2=zeros(ndof,1);
fa=0; fb=0;
[fa, V1, u]=gap_matrix_PDtest(pgap, u, V1, a);
[fb, V2, u2]=gap_matrix_PDtest(pgap, u2, V2, b);
if (a >= b) | (fa*fb > 0)
    fprintf('problem with the initial input: quitting\n');
    V=NaN;
    %u=zeros(ndof,1);
    return;
end
atol=1e-5;
n = ceil( log2(disp2-disp1) - log2(2*atol));
V=V1;
for k=1:n+1
   p=(a+b)/2;
   % p is the next displacement value for bisection method:
   [fp, V, u]=gap_matrix_PDtest(pgap, u, V, p);
   if (fa*fp < 0)
       b=p;
       fb=fp;
   else
       a=p;
       fa=fp;
   end
end
%
% end of bisection
%
%fprintf('\n*** The pull-in voltage is V=%g', V);
%fprintf('\n*** using atol=%g and bisection method\n', atol);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
function [ fp, V, u ] = gap_matrix_PDtest(pgap, u, V, tipd)
%
```

```
%  [ fp, V, u ] = matrix_PD_test(pgap, u, V, d)
%
%  This function uses the Cholesky factorization of the system Jacobian
%  matrix to detect whether the matrix is positive definite or indefinite.
%
%   inputs:
%     pgap  : structure of gap finite element code
%     u     : displacement vector ( initial guess)
%     V     : voltage ( initial guess)
%     tipd  : desired tip displacement (negative value,
%             [microns]
%
%   output:
%     fp:
%        fp>0 Jacobian of Ku-f_e=0 is indefinite
%        fp<0 Jacobian of Ku-f_e=0 is positive definite
%
%     u: solution vector
%
    K=pgap.K;
    [u, V, rnorms]=gap_solve_Nd(pgap, u, V, tipd);
    [f, Ju, JV]=gap_force(pgap, u, V);
    % We test the Jacobian. Remember to set the
    % positive definite status to -1:
    JF=K -Ju;
    [R,p]=chol(JF);
    % Cholesky decomposition of the symmetric matrix:
    %   p ==0 if JF is PD; p > 0 if JF is indefinite
    %   We change p==0 to p < 0 so that we can bracket the "root".

    if (p==0)
        fp=-1;
    else
        fp=p;
    end

end
%
%%%%%%%%%%%%%%%%%%%%%%%%%
%
% gap_pullin_sweep
```

```
%
%
% Compute pull-in voltage vs beam length
% for lengths from 20 to 100 microns
%
len1=20;
len2=100;
L_beam=linspace(len1, len2);
num=length(L_beam);
VV=zeros(num,1);
for iter=num:-1:1
    %
    % change the length of the beam:
    %
    pgap = gap_setup('c', 80, L_beam(iter));
    [u,V]=gap_pullin(pgap);
    VV(iter)=V;
end

plot(L_beam, VV);
xlabel('Beam Length [micron]');
ylabel('Pull-in voltage [V]');
```