CS4220 PS7

Due: April 13, 2015

1: Ascher and Greif, Section 9.4, Problem 6

(a) Suppose Newton's method is applied to a linear system $Ax = b$. How does the iterative formula look and how many iterations does it take to converge.

(b) Suppose the Jacobian matrix is singular at the solution of a nonlinear system of equations. Speculate what can occur in terms of convergence and rate of convergence. Specifically, is it possible to have a situation where the Newton iteration converges but convergence is not quadratic?

Solution:

(a) For a linear system

$$Ax = b, , \quad A \in \mathbb{R}^{n\times n}, \quad x, b \in \mathbb{R}^n, \tag{1}$$

let

$$\mathbf{F}(x) = Ax - b = 0, \tag{2}$$
$$\text{and} \quad \mathbf{F}(x) = \begin{pmatrix} F_1 & F_2 & \dots & F_n \end{pmatrix}^T.$$

Applying Newtion's method to Eq.(2),

$$\text{solve} \quad J(x_{(k)})p_{(k)} = -F(x_{(k)}) \quad \text{for } p_k; \tag{3}$$
$$\text{set} \quad x_{(k+1)} = x_{(k)} + p_{(k)}. \tag{4}$$

where $x_{(k)}$ is the iterate at the k-th step, and the Jacobian matrix

$$J(x) = \begin{pmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \cdots & \frac{\partial F_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial x_1} & \frac{\partial F_n}{\partial x_2} & \cdots & \frac{\partial F_n}{\partial x_n} \end{pmatrix}.$$

Since $F_1(x) = a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n - b_1$, we can easily show for the first row of J, we simply get

$$\frac{\partial F_1}{\partial x_1} = a_{11}, \quad \frac{\partial F_1}{\partial x_2} = a_{12}, \quad \ldots, \quad \frac{\partial F_1}{\partial x_n} = a_{1n}.$$

Similarly for all other rows of $J(x)$. So the Jacobian matrix equals the system matrix in the linear case:

$$J(x) = A.$$

Therefore for this system, we can cast Eq.(3) as follows:

$$Ap_{(k)} = -F(x_{(k)}) = -Ax_{(k)} + b. \tag{5}$$

If we start with $x_{(0)} = \mathbf{0}$, then solution of Eq.(5) gives us the final (converged) solution with exactly one iteration:

$$x_{(1)} = x_{(0)} + A^{-1}b = A^{-1}b.$$

(b) If the Jacobian matrix in Eq.(3) is singular or approaches singularity in the neighborhood of x^*, numerically the step length $\|p_k\|$ approaches 0, so it is a dangerous situation. If the criterion for terminating the iterations is based on the change of the step length without checking whether $\|F\|$ is approaching zero, one can incorrectly conclude that the "root is found". In such situations, the original Newton's method may fail, or the convergence becomes very slow because $J(x)$ is very ill-conditioned. So it is possible for Newton's method to "converge" if the Jacobian is very ill-conditioned (yet not exactly singular), but yet convergence rate is much less than quadratic.

There are a number of remedies to overcome such kind of situations. For instance, instead of solving the singular system in Eq.(3), we find the SVD of the singular Jacobina matrix, then use its the pseudo inverse $J^\dagger$ in sovling for $p_{(k)}$:

$$J = U\Sigma V^T, \tag{6}$$
$$J^\dagger = V\Sigma^{-1}U^T, \tag{7}$$
$$p_{(k)} = -J^\dagger F(x_{(k)}), \tag{8}$$
$$x_{(k+1)} = x_{(k)} + p_{(k)}. \tag{9}$$

2: Naive Newton Consider the nonlinear system of equations

$$x^2 + xy^2 = 9$$
$$3x^2y - y^3 = 4$$

Fill in the following Newton iteration code:

```
for k = 1:20
  F = % Your code here
  if norm(F) < rtol
    break;
  end
  J = % Your code here
  dx = J\F;
  x = x-dx;
end
```

Run your code with an initial guess of $(1, 1)$ and a residual norm tolerance of 10^{-12}. Do you see quadratic convergence?

Solution: Let $F(\mathbf{x}) = (F_1, F_2)^T = 0$, the olution vector $\mathbf{x} = (x_1, x_2)^T = (x, y)^T$, therefore

$$F_1 = x^2 + xy^2 - 9, \quad F_2 = 3x^2y - y^3 - 4, \tag{10}$$

and Jacobian matrix associated with this problem is

$$J = \begin{pmatrix} 2x + y^2 & 2xy \\ \\ 6xy & 3x^2 - 3y^2 \end{pmatrix}. \tag{11}$$

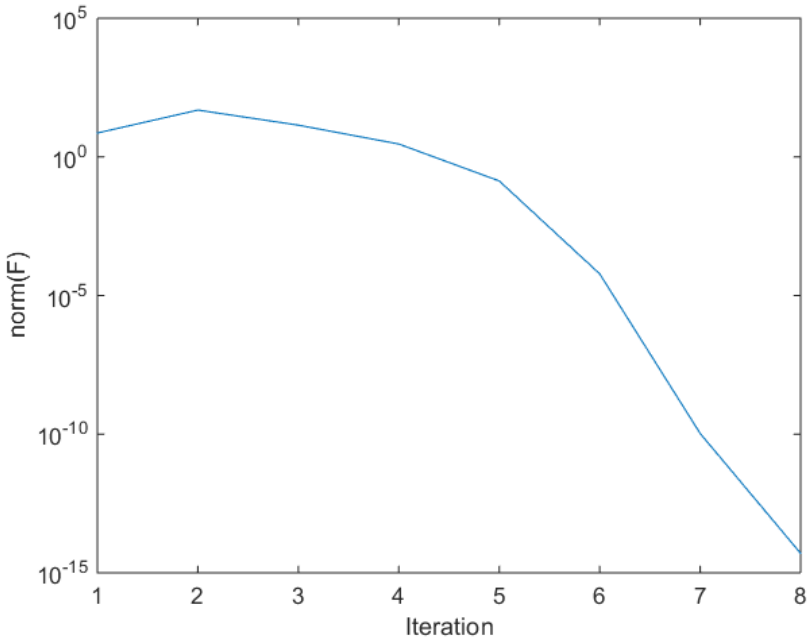
Now we can complete the MATLAB code according to Eqs.(10) and (11) as follows:

```
% PS7 Problem 2
% rtol=1E-12;
  n=2;
  rtol=1E-12;
  kmax=20;
  Fnorm=zeros(kmax,1);
  x=ones(n,1);  % initialization  x0=(1,1)
  for k = 1:kmax
    % x is x(1), y is x(2) in the following expressions:
    f1=x(1)^2+x(1)*x(2)^2-9;
    f2=3*x(1)^2*x(2)-x(2)^3-4;
    F = [f1  f2 ]';
    Fnorm(k)=norm(F);
    if norm(F) < rtol
      break;
    end
    J = [2*x(1)+x(2)^2 2*x(1)*x(2);   6*x(1)*x(2) 3*x(1)^2-3*x(2)^2];
    dx = J\F;
```

```
    x = x−dx;
    disp(sprintf('\n k=%d, x=[%e %e]',k,x(1),x(2)))
    disp(sprintf('\n normF = %e', norm(F)));
  end
% plot out norm(F) versus k:
  hx=[1:k];
  semilogy(hx, Fnorm(1:k));
```

The given problem converges in 7 iterations, the solution $\mathbf{x}$ is given here:

```
k=7,  x=[1.471869e+00  2.154717e+00]
 normF = 1.058961e-10
```



The $\|F\|$ versus iteration number plot for Problem 2. The slope in the plot verifies the convergence rate is quadratic.

3: Continue with Care Consider the boundary value problem

$$v''(x) + \gamma \exp(v(x)) = 0, \quad 0 < x < 1 \tag{12}$$

$$v(0) = v(1) = 0 \tag{13}$$

discretized via finite differences on a mesh with 100 equally spaced points; see example 9.3 in the book.

- Write a code to find v for a range of γ values from 1 to 3.5 (use `gammas = linspace(1,3.5)` to generate the mesh). For the first value of γ, you should use an initial guess of $v = 0$; for subsequent values, use the value of γ at the previous iterate. Plot all your solutions together on a single plot.
- For all γ in the given range, the Jacobian matrix at the solution remains negative definite. Plot $\lambda_{\max}(J(v^*))$ (the eigenvalue closest to zero) as a function of γ. What do you notice?
- Try running your code again, this time going up to a maximum value of 4 rather than 3.5. What happens?

Note: You may start from the following code

```
n = 100;
h = 1/(n+1);
T = diag(ones(n-1,1),-1) + diag(ones(n-1,1),1) - 2*eye(n);
v = zeros(n,1);
```

If n was very large, we might want to use a sparse matrix[1], but it's probably not worth it in this case.

Solution:

- The boundary value problem in Eqs.(12) and (13) is first approximated by the finite-difference method. We use the standard central differencing scheme to approximate $v''(x)$:

$$v''(x) \approx \frac{v_{i-1} - 2v_i + v_{i+1}}{h^2}, \tag{14}$$

so we build a system of n equations

$$F(v) = (F_1, F_2, \ldots, F_n)^T, \tag{15}$$

with the solution vector $\mathbf{v} = (v_1, v_2, \ldots, v_n)^T$, and

$$F_i = \frac{v_{i-1} - 2v_i + v_{i+1}}{h^2} + \gamma\, e^{(v_i)} = 0, \quad i = 1, 2, \ldots, n, \tag{16}$$

The given boundary conditions $v(0) = v(1) = 0$ is applied to the system of equations in Eq.(16) at the ends:

$$v_0 = v_{n+1} = 0.$$

[1] I'd probably switch to a more accurate discretization method, first, but that's a topic for CS 4210.

The Jacobian matrix of this finite-difference formulation given by Eq.(16) is a tridiagonal matrix:

$$J(v) = \frac{1}{h^2}\begin{pmatrix} -2+h^2\gamma e^{v_1} & 1 & & & & \\ 1 & -2+h^2\gamma e^{v_2} & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & 1 & -2+h^2\gamma e^{v_{n-1}} & 1 & \\ & & & 1 & -2+h^2\gamma e^{v_n} & \end{pmatrix}. \tag{17}$$

In writing a numerical code, it is convenient for us set up a matrix T and a column vector $\mathbf{bb}$ as follows:

$$T = \begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{pmatrix}, \quad \mathbf{bb} = \gamma \begin{pmatrix} e^{v_1} \\ e^{v_2} \\ . \\ . \\ e^{v_n} \end{pmatrix}. \tag{18}$$

After applying the boundary conditions to the system, the FD system of equations F can be written concisely as

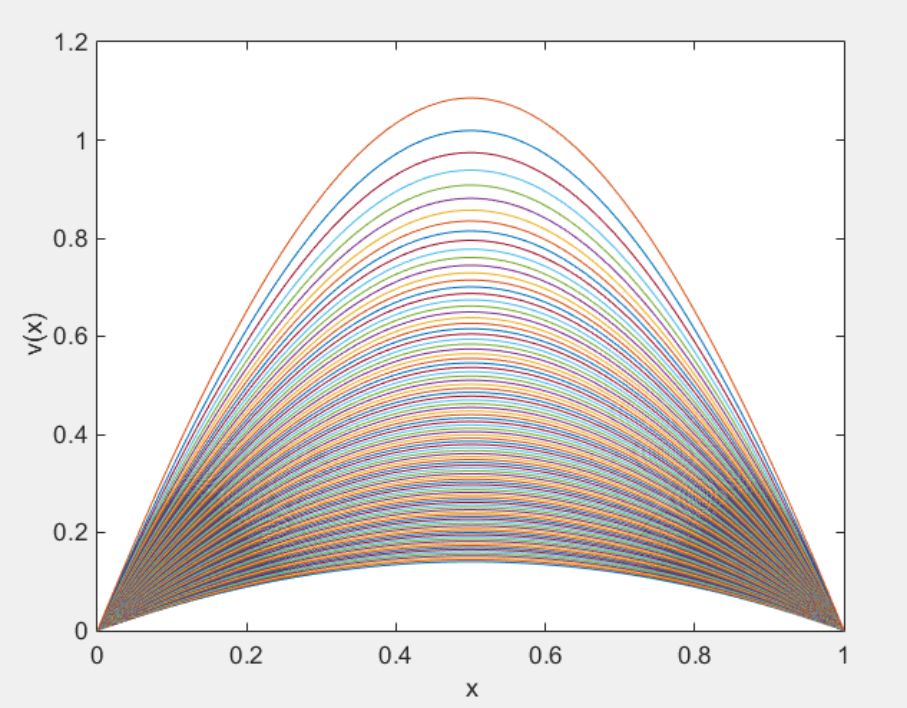
$$\begin{pmatrix} F_1 \\ F_2 \\ . \\ F_n \end{pmatrix} = \frac{1}{h^2} T \begin{pmatrix} v_1 \\ v_2 \\ . \\ v_n \end{pmatrix} + \mathbf{bb} = \frac{1}{h^2} T\mathbf{v} + \mathbf{bb}, \tag{19}$$

and the Jacobian matrix associated with this problem is

$$J(v) = \frac{1}{h^2} T + \mathrm{diag}(\mathbf{bb}). \tag{20}$$

Now we are ready to implement the standard Newton's algorithm in MATLAB. The code also parameterizes γ from 1 to 3.5, then plots all the solutions for different γ values (by storing the solution vectors in a matrix `vv` in the code) on a single plot. Listing of the code is in the appendix.

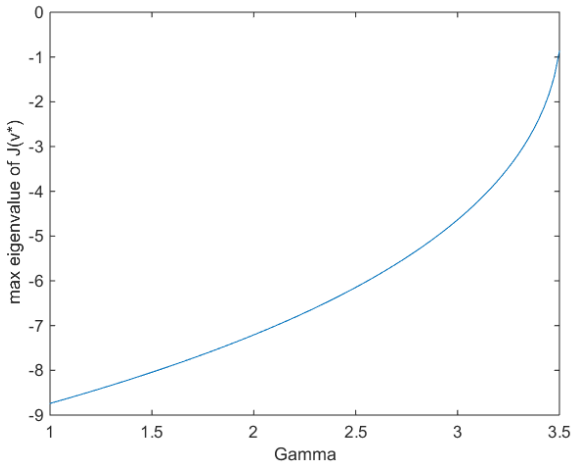
The solutions to the ODE using FDM is shown in the next plot. The solution curve at the top corresponds to $\gamma = 1$, and gradually $v(x)$ becomes flatter as γ becomes larger.



Solution to Eq.(12) by using finite difference method. The system of equations is solved by the Newton's method.

Note that we must use 'sm' in `eigs()` for the lambda closest to zero (the smallest in the absolute magnitude).

We plot $\lambda_{\max}(J(v^*))$ with respect to γ as shown below. Obviously the plot reveals that $\lambda_{\max}(J(v^*))$ are getting close to zero as γ is close to 3.5. In other words, $J(v^*)$ is almost singular (its $\lambda_{\max} \to 0$) as γ approaches 3.5.



- The Jacobian matrix at the solution $J(v^*)$ remains negatively definite. We can calculate $\lambda_{\max}(J(v^*))$ by the built-in MATLAB function `eigs()`:

  eigs(J,1,'sm')

- If we run the code again with γ going up to 4, Newton's method fails (approximately as $\gamma > 3.5$) due to the fact that $J(v^*)$ turns singular when $\gamma > 3.515$.

Appendix: Code Listing for Problem 3

```
% for Problem 3 of pset7:
% Solve:
%   v" + \gamma exp( v ) =0, 0< x <1,
%    v(0)=v(1)=0
% (by the finite-difference method)
% n equally spaced intervals of length h
n=100;
gmax=100;
h=1/(n+1);
tol=1e-10; % tolerance for controling the Newton's iterations
% don't get confused: x is not the solution
% vector, x is the node vector in (0,1),
% v(x) is the solution vector
x=0:h:1;
gammas=linspace(1,3.5,gmax);
e=ones(n-1,1);
% T is the Tridiagonal matrix of n by n from
% which we can build J(v) and F(v):
%
T=diag(e,-1)+diag(e,1)-2*eye(n);
v=zeros(n,1); %v is the solution vector!
% the following vv matrix is for plotting the solution v
% for different gammas:
% n+2 because solution vector plus two nodes: v(0) and v(1)
vv=zeros(n+2, gmax); % vv is a matrix for plotting all solutions
lmax=zeros(gmax,1);
maxiter=30;
k=1;

for k=1:gmax
    for j=1:maxiter
```

```
        bb=gammas(k)*exp(v);
        % The FDM approximation gives the following
        % equation F=0. (F is an n by 1 vector, note that
        %                the mesh has 102 nodes.)
        F=(1/h^2)*T*v + bb ;
        J=(1/h^2)*T + diag(bb);
        %
        % Newton's method is to solve
        % F + J*p = 0, so
        % p= -J\F
        %
        p = -J\F;
        v = v + p;
        norm_p = norm(p);
        norm_v = norm(v);
        if ( norm_p < tol*(1+norm_v) )
            fprintf('\n***j=%d,   norm_v=%e\n', j, norm_v)
            %largest eigenvalue of J(x*): eig which is closest to zero
            %because J is negatively definite , so use 'sm' option
            %in eigs function. J is approaching singular for eig is
            %rapidly approaching zero.
            lmax(k)=eigs(J,1,'sm');
            vv(:,k)=[0 v' 0]';
            break;
        end
    end
end
% Plot the solution from the vv matrix versus x:
figure(1);
plot(x, vv);
%
% Plot the lambda_max of J(v*)
%
figure(2);
plot(gammas,lmax);
```