CS 421: Fall 1997

Solutions to Assignment 5

1. Solve one nonlinear equation in one unknown.

```
function x=nonlinsolve(func_name,x0)
% Solve a nonlinear equation F(x)=0 in one unknown.
% func_name is a string containing the name of an m-file
%           that defines the function F and returns the
%           derivative as the second output argument.
% x0 is the starting point for the Newton iteration.

kmax=50;
tol=1e-12;
x=x0;
k=0;
[f,fp]=feval(func_name,x);
fprintf('k = %2d   x = %14.12f   f = %11.4e\n',k,x,f)
while (abs(f)>tol) & (k<kmax)
    k=k+1;
    xnew=x-f/fp;
    fnew=feval(func_name,xnew);
    j=0;
    while (abs(fnew)>abs(f))
        xnew=(x+xnew)/2;
        fnew=feval(func_name,xnew);
        j=j+1;
    end
    x=xnew;
    [f,fp]=feval(func_name,x);
    fprintf('k = %2d   x = %14.12f   f = %11.4e',k,x,f)
    fprintf('   (used %d backtracking steps)\n',j)
end
```

(a) $f(x) = \sin(x) - \cos(2x)$, $x_0 = 1$

```
function [y,yp]=func_a(x)
y = sin(x) - cos(2*x);
if nargout>1  % More than one output argument
    yp = cos(x) + 2*sin(2*x);
end
```

```
>> nonlinsolve('func_a',1);
k =  0   x = 1.000000000000   f =  1.2576e+00
k =  1   x = 0.466861954418   f = -1.4476e-01   (used 0 backtracking steps)
k =  2   x = 0.524749886580   f =  2.9917e-03   (used 0 backtracking steps)
k =  3   x = 0.523599156331   f =  9.8917e-07   (used 0 backtracking steps)
k =  4   x = 0.523598775598   f =  1.0891e-13   (used 0 backtracking steps)
```

Quadratic convergence.

(b) $f(x) = x^3 - 7x^2 + 11x - 5$, $x_0 = 2, 7$

```
function [y,yp]=func_b(x)
y = x.^3 - 7*x.^2 + 11*x - 5;
if nargout>1
    yp = 3*x.^2 - 14*x + 11;
end
```

```
>> nonlinsolve('func_b',2);
k =  0   x = 2.000000000000   f = -3.0000e+00
k =  1   x = 1.400000000000   f = -5.7600e-01   (used 0 backtracking steps)
k =  2   x = 1.188235294118   f = -1.3506e-01   (used 0 backtracking steps)
k =  3   x = 1.091734921817   f = -3.2889e-02   (used 0 backtracking steps)
k =  4   x = 1.045322767125   f = -8.1235e-03   (used 0 backtracking steps)
k =  5   x = 1.022530779230   f = -2.0191e-03   (used 0 backtracking steps)
k =  6   x = 1.011233392015   f = -5.0334e-04   (used 0 backtracking steps)
k =  7   x = 1.005608775825   f = -1.2566e-04   (used 0 backtracking steps)
k =  8   x = 1.002802417621   f = -3.1392e-05   (used 0 backtracking steps)
k =  9   x = 1.001400717447   f = -7.8453e-06   (used 0 backtracking steps)
k = 10   x = 1.000700236034   f = -1.9610e-06   (used 0 backtracking steps)
k = 11   x = 1.000350087363   f = -4.9020e-07   (used 0 backtracking steps)
k = 12   x = 1.000175036021   f = -1.2255e-07   (used 0 backtracking steps)
k = 13   x = 1.000087516095   f = -3.0636e-08   (used 0 backtracking steps)
k = 14   x = 1.000043757568   f = -7.6588e-09   (used 0 backtracking steps)
k = 15   x = 1.000021878663   f = -1.9147e-09   (used 0 backtracking steps)
k = 16   x = 1.000010939296   f = -4.7867e-10   (used 0 backtracking steps)
k = 17   x = 1.000005469632   f = -1.1967e-10   (used 0 backtracking steps)
k = 18   x = 1.000002734802   f = -2.9916e-11   (used 0 backtracking steps)
k = 19   x = 1.000001367443   f = -7.4794e-12   (used 0 backtracking steps)
k = 20   x = 1.000000683744   f = -1.8696e-12   (used 0 backtracking steps)
k = 21   x = 1.000000341946   f = -4.6807e-13   (used 0 backtracking steps)

>> nonlinsolve('func_b',7);
k =  0   x = 7.000000000000   f =  7.2000e+01
k =  1   x = 5.800000000000   f =  1.8432e+01   (used 0 backtracking steps)
k =  2   x = 5.200000000000   f =  3.5280e+00   (used 0 backtracking steps)
k =  3   x = 5.017391304348   f =  2.8069e-01   (used 0 backtracking steps)
k =  4   x = 5.000149281582   f =  2.3887e-03   (used 0 backtracking steps)
k =  5   x = 5.000000011141   f =  1.7826e-07   (used 0 backtracking steps)
k =  6   x = 5.000000000000   f = -7.1054e-15   (used 0 backtracking steps)
```

Linear convergence for $x_0 = 2$ and quadratic for $x_0 = 7$. This is because 1 is a double root of the polynomial, whereas 5 is a simple root.

(c) $f(x) = \sin(x) - \cos(x)$, $x_0 = 1, 2.7$

```
function [y,yp]=func_c(x)
y = sin(x) - cos(x);
if nargout>1
    yp = cos(x) + sin(x);
end

>> nonlinsolve('func_c',1);
k =  0   x = 1.000000000000   f =  3.0117e-01
k =  1   x = 0.782041901539   f = -4.7465e-03   (used 0 backtracking steps)
k =  2   x = 0.785398176000   f =  1.7822e-08   (used 0 backtracking steps)
k =  3   x = 0.785398163397   f = -1.1102e-16   (used 0 backtracking steps)

>> nonlinsolve('func_c',2.7);
k =  0   x = 2.700000000000   f =  1.3315e+00
k =  1   x = 4.096553005990   f = -2.3865e-01   (used 1 backtracking steps)
k =  2   x = 3.925346861543   f =  2.3249e-03   (used 0 backtracking steps)
k =  3   x = 3.926990818468   f = -2.0944e-09   (used 0 backtracking steps)
k =  4   x = 3.926990816987   f =  2.2204e-16   (used 0 backtracking steps)
```

Quadratic convergence in both cases.

(d) $f(x) = x^4 - 12x^3 + 47x^2 - 60x + 24$, $x_0 = 0, 2, 4$

```
function [y,yp]=func_d(x)
y = x.^4 - 12*x.^3 + 47*x.^2 - 60*x + 24;
if nargout>1
    yp = 4*x.^3 - 36*x.^2 + 94*x - 60;
end
```

```
>> nonlinsolve('func_d',0);
k =  0   x = 0.000000000000   f =  2.4000e+01
k =  1   x = 0.400000000000   f =  6.7776e+00   (used 0 backtracking steps)
k =  2   x = 0.642889908257   f =  1.8343e+00   (used 0 backtracking steps)
k =  3   x = 0.779939032158   f =  4.7073e-01   (used 0 backtracking steps)
k =  4   x = 0.850335037872   f =  1.0878e-01   (used 0 backtracking steps)
k =  5   x = 0.880224105639   f =  1.8287e-02   (used 0 backtracking steps)
k =  6   x = 0.887769696423   f =  1.1328e-03   (used 0 backtracking steps)
k =  7   x = 0.888303104579   f =  5.6235e-06   (used 0 backtracking steps)
k =  8   x = 0.888305779005   f =  1.4130e-10   (used 0 backtracking steps)
k =  9   x = 0.888305779072   f =  0.0000e+00   (used 0 backtracking steps)

>> nonlinsolve('func_d',2);
k =  0   x = 2.000000000000   f =  1.2000e+01
k =  1   x = 1.250000000000   f =  1.4414e+00   (used 0 backtracking steps)
k =  2   x = 1.090948275862   f =  3.1656e-01   (used 0 backtracking steps)
k =  3   x = 1.026300330557   f =  6.4215e-02   (used 0 backtracking steps)
k =  4   x = 1.003985632440   f =  8.2408e-03   (used 0 backtracking steps)
k =  5   x = 1.000126005072   f =  2.5228e-04   (used 0 backtracking steps)
k =  6   x = 1.000000134652   f =  2.6931e-07   (used 0 backtracking steps)
k =  7   x = 1.000000000000   f =  3.0553e-13   (used 0 backtracking steps)

>> nonlinsolve('func_d',4);
k =  0   x = 4.000000000000   f =  2.4000e+01
k =  1   x = 4.750000000000   f =  2.2441e+01   (used 3 backtracking steps)
k =  2   x = 0.930186170213   f = -5.4024e-02   (used 1 backtracking steps)
k =  3   x = 0.875284227967   f =  3.0747e-02   (used 1 backtracking steps)
k =  4   x = 0.887013661424   f =  2.7498e-03   (used 0 backtracking steps)
k =  5   x = 0.888290447397   f =  3.2241e-05   (used 0 backtracking steps)
k =  6   x = 0.888305776864   f =  4.6425e-09   (used 0 backtracking steps)
k =  7   x = 0.888305779072   f = -7.1054e-15   (used 0 backtracking steps)
```

The local convergence rate is quadratic for all three cases.

Let x_* be a solution to the equation $f(x) = 0$. If $f'(x_*) \neq 0$, then Newton's method converges quadratically to x_* when sufficiently close. We see this in all but one of the above cases. The value of $|f(x_k)|$ may decrease slowly to start with, but as the iterates get close to x_* quadratic convergence kicks in. However, if $f'(x_*) = 0$, then convergence is only linear and this is the case for function (b) at $x_* = 1$.

2. Since g is twice continuously differentiable in $[x_*-\rho, x_*+\rho]$ and $g'(x_*) = 0$, if $|x_k - x_*| < \rho$, then

$$\begin{aligned}
x_{k+1} - x_* &= g(x_k) - g(x_*) \\
&= g'(x_*)(x_k - x_*) + \frac{1}{2} g''(\xi_k)(x_k - x_*)^2 \\
&= \frac{1}{2} g''(\xi_k)(x_k - x_*)^2,
\end{aligned}$$

for some ξ_k between x_* and x_k. Let

$$C = \sup_{|z - x_*| \leq \rho} \frac{1}{2} |g''(z)|$$

which is finite since g'' is continuous in the interval. Then, if $|x_k - x_*| < \rho$,

$$|x_{k+1} - x_*| \leq \frac{1}{2} |g''(\xi_k)| |x_k - x_*|^2 \leq C |x_k - x_*|^2.$$

Suppose $|x_0 - x_*| < \min\{\rho, \frac{1}{2C}\}$. Then by induction the above implies

$$|x_k - x_*| < \min\{\rho, \frac{1}{2C}\}$$

and

$$|x_{k+1} - x_*| \leq \frac{1}{2} |x_k - x_*|.$$

This shows that the fixed point iteration $x_{k+1} = g(x_k)$ converges to x_* and because $|x_{k+1} - x_*| \leq C|x_k - x_*|^2$ the convergence rate is at least quadratic.

3. Newton's method used to compute eigenvalues:

```
function d=eigNewton(A,x)
% Use Newton's method to compute an eigenvalue of A
% with starting vector x

kmax=50;
tol=1e-12;
n=length(x);
I=eye(n,n);

tic
d=x'*A*x;
k=0;
r=A*x-d*x;
while (norm(r)>tol) & (k<kmax)
   B=[A-d*I, -x; 2*x', 0];
   b=-[r; x'*x-1];
   s=B\b;
   k=k+1;
   x=x+s(1:n);
   d=d+s(n+1);
   r=A*x-d*x;
   %fprintf('k=%2d   d=%15.12f   norm(r)=%11.4e\n',k,d,norm(r))
end
t=toc;
fprintf('Newton: iterations=%d  time=%f  norm(residual)=%e\n',k,t,norm(r))
```

The power method:

```
function d=eigPower(A,x)
% Use the power method to compute an eigenvalue of A
% with starting vector x

kmax=200;
tol=1e-12;

tic
d=x'*A*x;
k=0;
r=A*x-d*x;
while (norm(r)>tol) & (k<kmax)
   k=k+1;
   x=A*x;
   x=x/norm(x);
   d=x'*A*x;
   r=A*x-d*x;
   %fprintf('k=%2d   d=%15.12f   norm(r)=%11.4e\n',k,d,norm(r))
end
t=toc;
fprintf('Power: iterations=%d  time=%f  norm(residual)=%e\n',k,t,norm(r))
```

The results from some test cases are shown on next page. They depend on how the matrix A is chosen. It the entries are chosen from an uniform distribution, the dominant eigenvalue is well separated from the rest and the power method converges reasonably fast. But it fails to converge in 200 steps when a normal distribution is used. This is because the convergence rate of the power method depends on the ratio between the second largest and largest modulus of the eigenvalues. For normal distribution this ratio is close to one.

Newton's method has quadratic local convergence, while the power method converges linearily. So for high accuracy Newton's method might be preferable. However, ecah step of Newton's method is more costly than for the power method, so even if the former usues fewer iterations it can take longer time.

If we start with an real vector (for a real matrix A) both methods will converge to an real eigenvalue, even though the matrix has complex eigenvalues. Newton's method can be used to find complex roots if a complex starting point is used. Similarly, the power method can converge to a complex eigenvalue if we use a complex starting vector. This is done in the last test case.

Given an arbitrary nonzero unit vector x_0, the Rayleigh quotient $\lambda_0 = x_0^T A x_0$ is the best approximation to an eigenvalue of A. It is the solution to $\min_\lambda ||Ax_0 - \lambda x_0||_2$.

Some test runs:

```
>> format compact
>> A=rand(10,10); x0=randn(10,1); x0=x0/norm(x0);
>> d_newt=eigNewton(A,x0)
Newton: iterations=8  time=0.123471  norm(residual)=3.672718e-13
d_newt =
    0.1121
>> d_pow=eigPower(A,x0)
Power: iterations=19  time=0.025947  norm(residual)=5.594374e-13
d_pow =
    4.7284
>> d=eig(A)'
d =
  Columns 1 through 4
   4.7284           -0.0833- 0.8769i  -0.0833+ 0.8769i  -0.7246
  Columns 5 through 8
   0.7392           -0.4248            0.0583- 0.4261i   0.0583+ 0.4261i
  Columns 9 through 10
   0.0739            0.1121
>>
>> A=randn(10,10); x0=randn(10,1); x0=x0/norm(x0);
>> d_newt=eigNewton(A,x0)
Newton: iterations=11  time=0.124436  norm(residual)=2.209850e-16
d_newt =
    0.2809
>> d_pow=eigPower(A,x0)
Power: iterations=200  time=0.593866  norm(residual)=2.459831e+00
d_pow =
    1.5044
>>
>> A=rand(50,50); x0=randn(50,1); x0=x0/norm(x0);
>> d_newt=eigNewton(A,x0)
Newton: iterations=15  time=0.696249  norm(residual)=3.655547e-14
d_newt =
   -0.3576
>> d_pow=eigPower(A,x0)
Power: iterations=13  time=0.043717  norm(residual)=6.655506e-13
d_pow =
   24.9111
>>
>> A=rand(10,10); x0=randn(10,1)+sqrt(-1)*randn(10,1); x0=x0/norm(x0);
>> d_newt=eigNewton(A,x0)
Newton: iterations=7  time=0.042006  norm(residual)=2.504934e-16
d_newt =
  -0.0819+ 0.0896i
>> d_pow=eigPower(A,x0)
Power: iterations=19  time=0.116499  norm(residual)=3.735357e-13
d_pow =
   5.1815+ 0.0000i
```