553.481/681 Numerical Analysis

Homework 2 Solutions
Problem 1. The equation $e^x = x^3$ has a solution given by $x_* = -3W(-1/3)$ in terms of the Lambert W-function, which is calculated by `lambertw` in Matlab.

(a) Use the bisection method to find the root of $f(x) = e^x - x^3$ on $[1, 2]$ with tolerance set to $tol = 10^{-15}$. State the total number of iterations required and also the wall clock time (obtained in Matlab with `tic` and `toc` commands). Rewrite the Matlab script `bisect.m` to save the sequence of iterates and use these to estimate the constants $c$ and $p$ in the asymptotic error relation

$$e_{n+1} \sim c e_n^p$$

where $e_n = |x_* - x_n|$ is the error of the $n$th iterate. You can obtain these constants by plotting $\ln e_{n+1}$ versus $\ln e_n$ and using `polyfit` in Matlab to fit a straight line. Compare your numerical results to the theoretical estimates for both $c$ and $p$.

(b) Repeat part (a) for the Newton method with initial guess $x_0 = 1$.

(c) Repeat part (a) for the secant method with initial guess $a = 1$ and generate $b$ with the Newton method (counting this as the first iteration for secant).

(d) Repeat part (a) for the IQI method with initial guess $a = 1$ and generate $b$ with the Newton method and $c$ with the secant method (counting these as the first and second iterations for IQI).

Bisection: $c = 0.365543515678096$, $p = 0.982899981822632$. Theoretical: $c = 1/2$, $p = 1$.

Newton: $c = 0.508228296963834$, $p = 1.980396209450817$. Theoretical: $c = |f''(x_*)/2f'(x_*)| \approx 0.60093395321401$, $p = 2$.

Secant: $c = 1.017751729851889$, $p = 1.640860588027814$. Theoretical: $c = |f''(x_*)/2f'(x_*)|^{(p-1)/2} \approx 0.72997402982030$, $p = \varphi \approx 1.618033988749895$.

IQI: $c = 0.172561133650878$, $p = 1.728583991325469$. Theoretical: $c = |f'''(x_*)/6f'(x_*)|^{(p-1)/2} \approx 0.181573100247321$, $p = \text{real root of } - x^3 + x^2 + x + 1 \approx 1.83928675521461$. 
\[ f(x) = e^x - x^3; \]
\[ Df(x) = e^x - 3x^2; \]
\[ D^2f(x) = e^x - 6x; \]
\[ D^3f(x) = e^x - 6; \]
\[ xt = -3 \times \text{lambertw}(-1/3); \]
\[ a = 0.1; \quad b = 1; \quad a = 1; \quad b = 2; \quad \text{tol} = 1e-15; \]
\[ \text{disp('bisection method')} \]
\[ \text{bisection2} \]
\[ k = k \]
\[ e = \text{abs}(xt - x_{it}(1:k-3)); \]
\[ en = \text{abs}(xt - x_{it}(2:k-2)); \]
\[ le = \log(e); \quad len = \log(en); \]
\[ \text{figure} \]
\[ \text{nn} = \text{length}(xt); \]
\[ \loglog(\text{abs}(xt - x_{it}(1:nn-1)), \text{abs}(xt - x_{it}(2:nn)),'-b') \]
\[ \text{plot}(le, len, '-b', le, polyval(PP, le), '-r') \]
\[ \text{xlabel('log(e_n)')} \]
\[ \text{ylabel('log(e_{n+1})')} \]
\[ p = PP(1) \]
\[ pt = 1 \]
\[ c = \exp(PP(2)) \]
\[ ct = 1/2 \]
\[ \text{pause} \]
\[ x = 2; \]
\[ \text{clear xit} \]
\[ \text{disp('')} \]
\[ \text{disp('newton method')} \]
\[ \text{newton2} \]
\[ x = x \]
\[ k = k \]
\[ e = \text{abs}(xt - x_{it}(1:k-2)); \]
\[ en = \text{abs}(xt - x_{it}(2:k-1)); \]
\[ le = \log(e); \quad len = \log(en); \]
\[ \text{figure} \]
\[ \text{nn} = \text{length}(xt); \]
\[ \loglog(\text{abs}(xt - x_{it}(1:nn-1)), \text{abs}(xt - x_{it}(2:nn)),'-b') \]
\[ \text{plot}(le, len, '-b', le, polyval(PP, le), '-r') \]
\[ \text{xlabel('log(e_n)')} \]
\[ \text{ylabel('log(e_{n+1})')} \]
\[ p = PP(1) \]
\[ pt = 2 \]
\[ c = \exp(PP(2)) \]
\[ ct = \text{abs}(D^3f(xt)/Df(xt))/2 \]
\[ \text{pause} \]
\[ a = 2; \]
\[ \text{clear xit} \]
\[ \text{clear b} \]
\[ \text{disp('')} \]
\[ \text{disp('secant method')} \]
\[ \text{secant2} \]
\[ x = b \]
\[ k = k \]
\[ e = \text{abs}(xt - x_{it}(1:k-3)); \]
\[ en = \text{abs}(xt - x_{it}(2:k-2)); \]
\[ le = \log(e); \quad len = \log(en); \]
\[ \text{figure} \]
\[ \text{nn} = \text{length}(xt); \]
\[ \loglog(\text{abs}(xt - x_{it}(1:nn-1)), \text{abs}(xt - x_{it}(2:nn)),'-b') \]
\[ \text{plot}(le, len, '-b', le, polyval(PP, le), '-r') \]
\[ \text{xlabel('log(e_n)')} \]
\[ \text{ylabel('log(e_{n+1})')} \]
\[ p = PP(1) \]
\[ \phi = (1+\sqrt{5})/2; \]
\[ pt = \phi \]
\[ c = \exp(PP(2)) \]
\[ c_t = (\abs(DDf(x_t)/Df(x_t)/2))^{(pt-1)} \]
\[ a = 2; \]
\[ \text{clear } b \]
\[ \text{clear } c \]
\[ \text{disp('')} \]
\[ \text{disp('iquadi method')} \]
\[ \text{iquadi2} \]
\[ x = c \]
\[ k = k \]
\[ e = \abs(x_t - x_{it}(1:k-4)); \]
\[ e_n = \abs(x_t - x_{it}(2:k-3)); \]
\[ le = \log(e); \]
\[ len = \log(e_n); \]
\[ PP = \text{polyfit}(le, len, 1); \]
\[ \text{figure} \]
\[ \text{loglog}(\abs(x_t - x_{it}(1:nn-1)), \abs(x_t - x_{it}(2:nn)),'-b') \]
\[ \text{plot}(le, len, '-b', le, \text{polyval}(PP, le), '-r') \]
\[ \text{xlabel('log(e_n)')} \]
\[ \text{ylabel('log(e_{n+1})')} \]
\[ p = PP(1) \]
\[ \text{rr} = \text{roots}([-1 1 1 1]); \]
\[ pt = \text{rr}(1) \]
\[ c = \exp(PP(2)) \]
\[ c_t = (\abs(DDDf(x_t)/Df(x_t)/6))^{(pt-1)} \]

**bisect2.m:**

```
\text{tic} \\
\text{itmax}=100; \\
\text{if} \ \text{sign}(f(a)) == \text{sign}(f(b)) \text{ return} \\
\text{end} \\
k = 0; \\
[a b b-a] \\
xit=[]; \\
\text{while } \abs(b-a) > \text{tol} * \max(\abs(b),1.0) \\
\text{if } k+1 > \text{itmax} \\
\text{break} \\
\text{end} \\
x=(a+b)/2; \\
xit=[xit;x]; \\
\text{if } \text{sign}(f(x)) == \text{sign}(f(b)) \\
b=x; \\
\text{else} \\
\text{a=x;} \\
\text{end} \\
k = k+1 \\
[a b b-a] \\
\text{end} \\
x=(a+b)/2 \\
xit=[xit;x]; \\
\text{toc} \\
```

**newton2.m:**

```
\text{tic} \\
\text{itmax}=100; \\
k=0; \\
\text{if } x ~= 0 \\
xold=0; \\
\text{else} \\
xold=1; \\
\text{end} \\
[x \ \text{abs}(x-xold)];
```
x = x;
while abs(x - xold) > tol * max(abs(x), 1.0)
  if k+1 > itmax
    break
  end
  xold = x;
  k = k+1;
  x = x - f(x)/ Df(x);
  [x abs(x-xold)];
  xit = [xit;x];
end
[x abs(x-xold)];
toc

secant2.m:

  tic
  k = 0;
  if exist('b') == 0
    b = a - f(a)/ Df(a); k = k+1;
  end
  xit = b;
  itmax = 50;
  k;
  [b abs(b-a)];
  fa = f(a);
  while abs(b - a) > tol * max(abs(b), 1.0)
    if k+1 > itmax
      break
    end
    fb = f(b);
    x = b + (b-a)/(fa/fb - 1);
    k = k+1;
    a = b;
    fa = fb;
    b = x;
    [b abs(b-a)];
    xit = [xit;b];
  end
  [b abs(b-a)];
toc

iquadi2.m:

  tic
  itmax = 100;
  k = 0;
  fa = f(a);
  if exist('b') == 0
    b = a - fa/Df(a); k = k+1;
  end
  fb = f(b);
  if exist('c') == 0
    c = b + (b-a)/(fa/fb - 1); k = k+1;
  end
  xit = c;
  k;
  [a b c];
  while abs(c - b) > tol * max(abs(c), 1.0)
if k+1 > itmax
    break
end
fc=f(c);
u=fb/fc;
v=fb/fa;
w=fa/fc;
p=w*(u-w)*(c-b)-(1-u)*(b-a);
p=v*p;
q=(u-1)*(v-1)*(w-1);
x=b+p/q;
k=k+1;
a=b;
fa=fb;
b=c;
fb=fc;
c=x;
xit=[x;xit;c];
[c abs(c-b)];
end
[c abs(c-b)];
toc
Problem 2. Newton’s method for finding a root $x_*$ of $f(x) = 0$ sometimes requires the initial guess $x_0$ to be quite close to $x_*$ in order to obtain convergence. Verify that this is the case for the root $x_* = \pi/2$ of

$$f(x) = \cos(x) + \sin^2(50x)$$

Based on the convergence proof, make a rough estimate how small $|x_0 - x_*|$ must be for iterates to converge to $x_*$. Check this estimate numerically by slowly increasing the distance of $x_0$ from $x_*$ until the Newton iteration fails to converge.

We look for solutions in $I := [\pi/2 - \varepsilon, \pi/2 + \varepsilon]$. To guarantee convergence we require $f'(x) \neq 0$ for $x \in I$ and

$$|x_* - x_0| < \frac{\max_{x \in I} |f''(x)|}{2 \max_{x \in I} |f'(x)|}.$$

We find $f'(x) = -\sin(x) + 50 \sin(100x)$ (using half-angle identities) and $f''(x) = -\cos(x) + 5000 \cos(100x)$. For $\varepsilon \ll 1$ the maxima of $|f'|$, $|f''|$ in $I$ are readily found to be $f'(\pi/2) = -1$ and $f''(\pi/2) = 5000$. Taking $\varepsilon = 1/2500$ works because one can numerically check the nearest zero of $f'$ ($\approx 1.570796326794897$) is outside $I$. Thus $|x_* - x_0| \leq 1/2500 \approx 2.000013 \times 10^{-4}$ is a good requirement. We verify that convergence fails for $|x_* - x_0| = 2.036732 \times 10^{-4}$.

```matlab
f=@(x) cos(x)+(sin(50*x)).^2; Df=@(x) -sin(x)+50* sin(100*x); tol=1e-15; M=2500; figure; fplot(@(x) [Df(x), 0*x], [pi/2-1/2500, pi/2+1/2500]) hold on plot(pi/2 ,0 , 'r* ') delta=1.571-pi/2 pause for ii =1:100 x=pi/2+ii/100/M; newton check=abs(x-pi/2) if check>tol 'failure for', ii dist=ii/100/M delta=delta return end end
```
Problem 3. Given below is a table of iterates from a linearly convergent iteration \( x_{n+1} = g(x_n) \). Estimate from this data (a) the rate of linear convergence, (b) the error \( x_7 - x_* \) and (c) the fixed point \( x_* \).

<table>
<thead>
<tr>
<th>( n )</th>
<th>( x_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.5000000</td>
</tr>
<tr>
<td>1</td>
<td>0.5328847</td>
</tr>
<tr>
<td>2</td>
<td>0.5536649</td>
</tr>
<tr>
<td>3</td>
<td>0.5662537</td>
</tr>
<tr>
<td>4</td>
<td>0.5736789</td>
</tr>
<tr>
<td>5</td>
<td>0.5779878</td>
</tr>
<tr>
<td>6</td>
<td>0.5804645</td>
</tr>
<tr>
<td>7</td>
<td>0.5818802</td>
</tr>
</tbody>
</table>

Given that the true fixed point is \( x_* = 0.583748350883722 \) to 16 decimals, compare your best estimates for \( x_* \) and the error \( x_7 - x_* \) with the true values.

(a) We estimate \( \lambda_7 = (x_7 - x_6)/(x_6 - x_5) \approx 0.5715970 \).

(b) Using Atiken’s error formula \( x_* - x_7 \approx \lambda_7(x_7 - x_6)/(1 - \lambda_7) \approx 0.0018889 \), whereas the true error is \( x_* - x_7 = 0.0018682 \).

(c) From the estimated error we form the improved estimate:

\[
\hat{x}_7 = x_7 + (x_* - x_7) \approx 0.5837690.
\]

The error in the improved estimate is \( x_* - \hat{x}_7 = 2.0672534 \times 10^{-5} \).
Problem 4. Define an iteration formula by

\[ x_{n+1} = \bar{x}_{n+1} - \frac{f(\bar{x}_{n+1})}{f'(x_n)}, \quad \bar{x}_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \]

Show that the order of convergence of \( \{x_n\} \) to \( x^* \) is at least 3. State conditions on the mean times \( \tau_f \) and \( \tau_{f'} \) to evaluate \( f \) and \( f' \), respectively, so that this method converges asymptotically faster than Newton’s method.

Define \( h(x) := x - f(x)/f'(x) \) and \( g(x) := h(x) - f(h(x))/f'(x) \). If \( f(x^*) = 0 \) then \( g(x^*) = h(x^*) = x^* \). Note that

\[ h'(x) = \frac{f(x)f''(x)}{f'(x)^2}, \quad h''(x) = \frac{f''(x)}{f'(x)} + f(x) \frac{d}{dx} \frac{f''(x)}{f'(x)^2}, \]

with \( h'(x^*) = 0 \) and \( h''(x^*) = f''(x^*)/f'(x^*) \). Then

\[ g'(x) = h'(x) - h'(x) \frac{f(h(x))}{f'(x)} + \frac{h(x)f''(x)}{f'(x)^2}, \]

\[ g''(x) = h''(x) - h''(x) \frac{f(h(x))}{f'(x)} - h'(x) \frac{d}{dx} \frac{f'(h(x))}{f'(x)} + h'(x) \frac{d}{dx} \frac{f''(h(x))}{f'(x)} + \frac{f(h(x))h''(x)f''(x)}{f'(x)^3} + \frac{f(h(x))f''(x)}{f'(x)^2}. \]

Thus \( g'(x^*) = g''(x^*) = 0 \) and by a Taylor expansion \( |x^* - g(x)| \propto |g^{(3)}(\xi)(x^* - x)^3| \) for some \( \xi \) between \( x^*, x \). Newton’s method requires 1 evaluation for each of \( f, f' \), whereas \( g \) requires 2 evaluations for \( f \) and 1 of \( f' \).

By asymptotic analysis, the number of iterations required for convergence of 2nd-order Newton method is \( \log_2 K \) and for the 3rd-order method is \( \log_3 K \), with the same constant \( K = \log(M \cdot TOL)/\log(M|x^* - x_0|) > 1 \).

Thus, Newton’s method takes longer when

\[ (\tau_f + \tau_{f'}) \log_2 K > (2\tau_f + \tau_{f'}) \log_3 K \]

or when

\[ \tau_{f'}/\tau_f > \frac{\log(4/3)}{\log(3/2)} \approx 0.7095113. \]
Problem 5. (a) Verify that the Sherman-Morrison formula is correct by checking that

\[
\left( A^{-1} - \frac{A^{-1}xy^\top A^{-1}}{1 + \langle y, A^{-1}x \rangle} \right)(A + xy^\top) = I.
\]

(b) Use the Sherman-Morrison formula to verify that the approximation \( A_n^{-1} \) used in Broyden’s method has an inverse that can be iterated with the formulas

\[
\begin{align*}
\Delta p_{n-1} &= A_{n-1}^{-1}\Delta f_{n-1}, \\
A_n^{-1} &= A_{n-1}^{-1} + \frac{(\Delta x_{n-1} - \Delta p_{n-1})\Delta x_{n-1}^\top A_{n-1}^{-1}}{\langle \Delta x_{n-1}, \Delta p_{n-1} \rangle}
\end{align*}
\]

where \( \Delta x_{n-1} = x_n - x_{n-1} \) and \( \Delta f_{n-1} = f(x_n) - f(x_{n-1}) \).

(a) Write

\[
(A + xy^\top) \left( A^{-1} - \frac{A^{-1}xy^\top A^{-1}}{1 + \langle y, A^{-1}x \rangle} \right) = I + xy^\top A^{-1} - \frac{xy^\top A^{-1} + xy^\top A^{-1}xy^\top A^{-1}}{1 + \langle y, A^{-1}x \rangle}
\]

\[
= I + xy^\top A^{-1} - \frac{xy^\top A^{-1} + x(y, A^{-1}x)y^\top A^{-1}}{1 + \langle y, A^{-1}x \rangle}
\]

\[
= I + xy^\top A^{-1} - \frac{xy^\top A^{-1}(1 + \langle y, A^{-1}x \rangle)}{1 + \langle y, A^{-1}x \rangle}
\]

\[
= I + xy^\top A^{-1} - xy^\top A^{-1}
\]

\[
= I.
\]

(b) The Broyden update is \( A_n = A_{n-1} + (\Delta f_{n-1} - A_{n-1})\Delta x_{n-1}^\top/\|\Delta x_{n-1}\|^2 \). Thus

\[
A_n^{-1} = A_{n-1}^{-1} - \frac{A_{n-1}^{-1}(\Delta f_{n-1} - A_{n-1}\Delta x_{n-1})\Delta x_{n-1}^\top A_{n-1}^{-1}}{1 + \langle \Delta x, A_{n-1}^{-1}(\Delta f_{n-1} - A_{n-1}\Delta x_{n-1})/\|\Delta x_{n-1}\|^2 \rangle}
\]

\[
= A_{n-1}^{-1} - \frac{(A_{n-1}^{-1}\Delta f_{n-1} - \Delta x_{n-1})\Delta x_{n-1}^\top A_{n-1}^{-1}}{\|\Delta x_{n-1}\|^2 + \langle \Delta x, A_{n-1}^{-1}\Delta f_{n-1} - \Delta x_{n-1} \rangle}
\]

\[
= A_{n-1}^{-1} - \frac{(A_{n-1}^{-1}\Delta f_{n-1} - \Delta x_{n-1})\Delta x_{n-1}^\top A_{n-1}^{-1}}{\langle \Delta x, A_{n-1}^{-1}\Delta f_{n-1} \rangle},
\]

where we take \( p_n := A_{n-1}^{-1}\Delta f_n \).
**Problem 6.** Use both Newton-Raphson and Broyden’s methods for a starting vector \(x(0), y(0), z(0) = (1, 1, 1)\) to approximate the solution of the following system of equations to tolerance \(tol = 10^{-15}\):

\[
\begin{align*}
    x^2 + y^2 + z^2 &= 1 \\
    x^2/4 + y^2 + z^2 &= 1/2 \\
    y &= ze^z
\end{align*}
\]

Compare the number of iterations and also the wall clock time required for this accuracy with the two methods. In order to make a more accurate estimate of the relative clock time, repeat the root-finding a large number of times \(N_{repeat}\) for both methods (in the same loop) and average over the repeated trials.

We find both methods converge to \((0.816496580927726, 0.469693072077321, 0.335740601321767)\) in double precision arithmetic.

The difference between the results of the two methods is

\[
(-0.111022302462516, 0.111022302462516, 0.055511151231258) \cdot 10^{-15}.
\]

Newton’s method takes 7 iterations whereas Broyden’s takes 15. After careful timing (see code below), Newton-Raphson requires only about 72.18\% as much time as Broyden.

```matlab
f=@(x) [x(1)^2+x(2)^2+x(3)^2-1; x(1)^2/4+x(2)^2+x(3)^2-1/2; x(2)-x(3)*exp(x(3))];
Df=@(x) [ 2*x(1) 2*x(2) 2*x(3); x(1)/2 2*x(2) 2*x(3); 0 1 -(x(3)+1)*exp(x(3))];
tol=1e-15
timnewt=0;
timbroy=0;
for ntry=1:10000
    x=[1; 1; 1];
    tic
    [xnewt, itnewt]=newtraph(f,Df,x,tol);
    timnewt=timnewt+toc;
    x=[1; 1; 1];
    tic
    [xbroy, itbroy]=broyden(f,Df,x,tol);
    timbroy=timbroy+toc;
end
xnewt=xnewt
itnewt=itnewt
timnewt=timnewt/ntry
xbroy=xbroy
itbroy=itbroy
timbroy=timbroy/ntry
timrat=timnewt/timbroy
dx=xnewt-xbroy
```
Problem 7*. (a) Here we illustrate the relaxation method to solve nonlinear boundary problems by converting them into high-dimensional root-finding problems. As an example, we consider the nonlinear boundary value problem for \( m \in (0, 1) \)

\[
x''(u) + (1 + m)x(u) - 2mx(u)^3 = 0, \quad u \in [0, 2K(m)], \quad x(0) = x(2K(m)) = 0
\]

where \( K(m) \) is the complete elliptic integral of the first kind, given by the Matlab function \texttt{ellipke}. The exact solution of the problem is known to be the Jacobi elliptic function \( x(u) = \text{sn}(u, m) \), given by the Matlab function \texttt{ellipj}.

In the relaxation method, one introduces a numerical grid \( u_i = \frac{2K(m)}{n+1}i, \quad i = 0, 1, ..., n + 1 \) with spacing \( \Delta u = \frac{2K(m)}{n+1} \) and corresponding function values \( x_i = x(u_i) \). With finite-difference approximations to the second-derivatives

\[
x''(u_i) \approx \frac{x_{i+1} + x_{i-1} - 2x_i}{(\Delta u)^2}
\]

the ODE boundary-value problem becomes a nonlinear fixed-point condition \( f(x) = 0 \) for the solution vector \( x = (x_1, x_2, ..., x_n)^\top \). (a) Use the Newton-Raphson method to solve the resulting fixed-point problem for \( n = 100 \) with initial guess \( x_0(u) = \sin(\pi u/2K(m)) \) to a tolerance \( tol = 10^{-8} \), and report the number of iterations and the error estimate at each iteration. For specificity, take \( m = 0.643856219147755 \) so that \( K(m) = 2 \). Finally, plot your numerical solution \( x_i \) versus \( u_i \) for \( i = 0, 1, 2, ..., 101 \) together with the exact solution \( x(u_i) \) for those same \( u_i \) values.

(b) Repeat part (a) using Broyden’s method.

(c) Repeat part (a) using the Levenberg-Marquardt method implemented in Matlab’s function \texttt{fsolve}. Use the function \texttt{optimoption} to choose the algorithm in \texttt{fsolve} to be Levenberg-Marquardt, to set \texttt{display} to \texttt{iter}, and to set \texttt{FunctionTolerance} to \( 10^{-8} \).

(a) Code:

```matlab
m = fzero(@(x) ellipke(x) - 2.0/1.0);  \ m = 0.643856219147755;
for n = 100;  \ n = 100;
```
\begin{verbatim}
u = (1:n); 
K = ellipke(m); 
u = 2*K*u/(n+1); 
x = sin(pi*u/K/2); 
tol = 1e-8; 
itmax = 5000; 
newraph 
X = [0; x; 0]; 
U = [0; u; 2*K]; 
M = m + 0* U; 
Y = ellipj(U, M); 
plot(U, X,'-b', U, Y,'--r', 'LineWidth', 2) 
xlabel('u') 
ylabel('x') 
legend('approx','exact') 
title('Newton-Raphson Method') 
axis([0 2*K 0 1]) 
k=k 
end

Output:
ans =
  0  7.106335201775946

ans =
  1.000000000000000  0.675792031868956

ans =
  2.000000000000000  0.057567871767832

ans =
  3.000000000000000  0.000555362120997

ans =
  4.000000000000000  0.000000056237703

Elapsed time is 0.077108 seconds.
k = 4

(b) Code:
x = sin(pi*u/K/2); 
tol = 1e-8; 
broyden 
X = [0; x; 0]; 
figure 
plot(U, X,'-b', U, Y,'--r', 'LineWidth', 2) 
xlabel('u') 
ylabel('x') 
legend('approx','exact') 
title('Broyden Method') 
axis([0 2*K 0 1]) 
k=k 
end

Output:
ans =
  0  7.106335201775946
\end{verbatim}
ans =
    1.000000000000000  0.675792031868956
ans =
    2.000000000000000  0.063753574988276
ans =
    3.000000000000000  0.005617596757867
ans =
    4.000000000000000  0.000051762419600
ans =
    5.000000000000000  0.000000433304050
ans =
    6.000000000000000  0.000000013458853
Elapsed time is 0.225418 seconds.
k =
    6

(c) Code:
x = sin(pi*u/K/2);
options = optimoptions('fsolve','Algorithm','Levenberg-Marquardt','Display','iter','TolFun',1e-8)
x = fsolve(@(x)f(x),x,options);
X = [0; x; 0];
figure
plot(U,X,'-b',U,Y,'--r','LineWidth',2)
xlabel('u')
ylabel('x')
legend('approx','exact')
title('Levenberg-Marquardt Method')
axis([0 2*K 0 1])

Output:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Func-count</th>
<th>Residual</th>
<th>First-Order</th>
<th>Lambda</th>
<th>Norm of step</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>101</td>
<td>1.33419e-05</td>
<td>5.72e-06</td>
<td>0.01</td>
<td>0.00320648</td>
</tr>
<tr>
<td>1</td>
<td>202</td>
<td>1.3135e-05</td>
<td>5.59e-06</td>
<td>0.001</td>
<td>0.0290099</td>
</tr>
<tr>
<td>2</td>
<td>303</td>
<td>1.13663e-05</td>
<td>4.71e-06</td>
<td>0.0001</td>
<td>0.259412</td>
</tr>
<tr>
<td>3</td>
<td>404</td>
<td>4.47048e-06</td>
<td>2.27e-06</td>
<td>1e-05</td>
<td>0.1555</td>
</tr>
<tr>
<td>4</td>
<td>505</td>
<td>6.24433e-07</td>
<td>3.82e-07</td>
<td>1e-06</td>
<td>0.249749</td>
</tr>
<tr>
<td>5</td>
<td>606</td>
<td>3.43887e-09</td>
<td>4.38e-08</td>
<td>1e-07</td>
<td>0.0191033</td>
</tr>
<tr>
<td>6</td>
<td>707</td>
<td>1.92629e-13</td>
<td>2.77e-10</td>
<td>1e-08</td>
<td>0.000143691</td>
</tr>
<tr>
<td>7</td>
<td>808</td>
<td>2.14192e-19</td>
<td>1.62e-13</td>
<td>1e-09</td>
<td>1.52727e-07</td>
</tr>
<tr>
<td>8</td>
<td>909</td>
<td>2.52945e-27</td>
<td>8.7e-16</td>
<td>1e-10</td>
<td></td>
</tr>
</tbody>
</table>
Problem 8*. The Levenberg-Marquardt algorithm is an example of a method which solves nonlinear equations $f(x) = 0$ by least-squares minimization, or, in other words, by minimizing the scalar function

$$\Phi(x) = \frac{1}{2} ||f(x)||^2$$

by a descent algorithm which combines Steepest Descent and Gauss-Newton methods. These methods make updates of the successive iterates by $x_{n+1} = x_n + \alpha_n d_n$ where the vector $d_n$ is a descent direction for which

$$\langle d_n, \nabla \Phi(x_n) \rangle \leq 0.$$

(a) Find the direction $d_n$ which is the direction of steepest descent for the scalar function $\Phi(x)$ defined in (*) at the point $x = x_n$.

(b) Show that the Newton update vector $\Delta x_n := -(Df(x_n))^{-1}f(x_n)$ is a descent direction for $\Phi(x)$ defined in (*) at $x = x_n$.

(a) By the definition of the gradient, we must have $d_n := -\nabla \Phi(x_n)$. Then

$$\frac{\partial}{\partial x_i} \Phi(x) = \frac{1}{2} \sum_{j=1}^{d} \frac{\partial}{\partial x_i} f_j(x)^2 = \sum_{j=1}^{d} [f(x)]_j[Df(x)]_{ji}.$$

This is identical to $d_n := -Df(x_n)^T f(x_n)$.

(b) We have

$$\langle \Delta x, -\nabla \Phi(x) \rangle = \langle Df(x)^{-1} f(x), Df(x)^T f(x) \rangle = \langle f(x), (Df(x)^{-1})^T Df(x)^T f(x) \rangle = ||f(x)||^2 \geq 0.$$