Probability Theory 2 Homework 2

Matthew Hudes

February 21, 2024

553.481/681 Numerical Analysis – Homework 2 Solutions

Lowen Peng

Problem 1. In `MATLAB` we rewrite `bisect.m` as `bisect2.m`, `newton.m` as `newton2.m`, and so on. We find the asymptotic error parameters by fitting over straight-line regions in the plot of $ln(e_n + 1)$ vs $\ln(e_n)$.

```
 f=@(x) x.^2-5;
Df=@(x) 2*x;
DDf=@(x) 2+0*x;
DDDf=@(x) 0*x;
xt=sqrt(5);

a=2; b=3;
tol=1e-15;
disp('bisection method')
bisect2
k=k

e=abs(xt-xit(1:k-3));
en=abs(xt-xit(2:k-2));
le=log(e); len=log(en);
PP=polyfit(le,len,1);
figure
plot(le,len,'-b',le,polyval(PP,le),'-r')
xlabel('log(e_n)')
ylabel('log(e_{n+1})')
legend('data','fit','location','northwest')
title('error for bisection')
p=PP(1)
```

```
pt=1
c=exp(PP(2))
ct=1/2
pause

x=2;
clear xit
disp(' ')
disp('newton method')
newton2
x=x
k=k

e=abs(xt-xit(1:k-2));
en=abs(xt-xit(2:k-1));
le=log(e); len=log(en);
PP=polyfit(le,len,1);
figure
plot(le,len,'-b',le,polyval(PP,le),'-r')
xlabel('log(e_n)')
ylabel('log(e_{n+1})')
legend('data','fit','location','northwest')
title('error for newton')
p=PP(1)
pt=2
c=exp(PP(2))
ct=DDf(xt)/Df(xt)/2
pause

a=2;
clear b
disp(' ')
disp('secant method')
secant2
x=b
k=k

e=abs(xt-xit(2:k-2));
en=abs(xt-xit(3:k-1));
le=log(e); len=log(en);
PP=polyfit(le,len,1);
figure
plot(le,len,'-b',le,polyval(PP,le),'-r')
xlabel('log(e_n)')
```

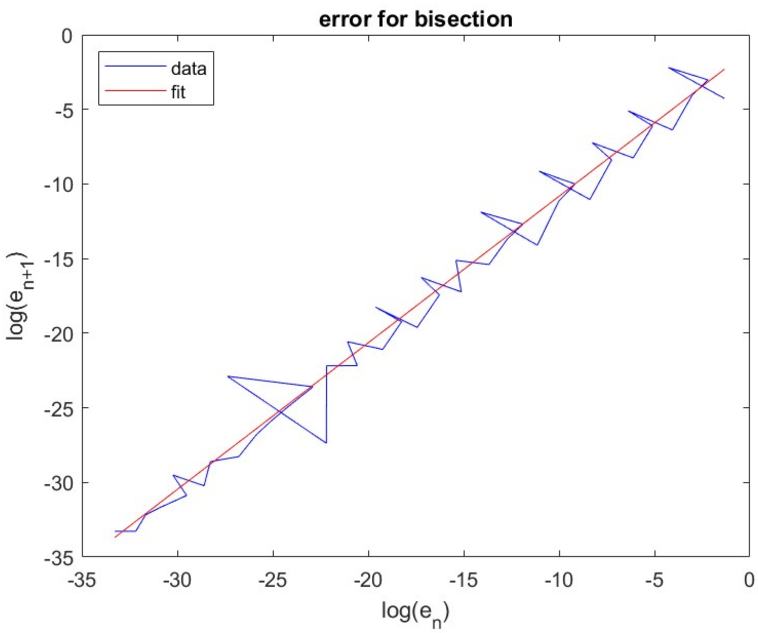
```
ylabel('log(e_{n+1})')
legend('data','fit','location','northwest')
title('error for secant')
p=PP(1)
phi=(1+sqrt(5))/2;
pt=phi
c=exp(PP(2))
ct=(DDf(xt)/Df(xt)/2)^(pt-1)
pause

a=2;
clear b
clear c
disp(' ')
disp('iquadi method')
iquadi2
x=c
k=k

e=abs(xt-xit(1:k-4));
en=abs(xt-xit(2:k-3));
le=log(e); len=log(en);
PP=polyfit(le,len,1);
figure
plot(le,len,'-b',le,polyval(PP,le),'-r')
xlabel('log(e_n)')
ylabel('log(e_{n+1})')
legend('data','fit','location','northwest')
title('error for iquadi')
p=PP(1)
rr=roots([-1 1 1 1]);
pt=rr(1)
c=exp(PP(2))
ct=(DDDf(xt)/Df(xt)/6)^((pt-1)/2)
```

(a) Since the error should halve every step, we expect $p = 1$ and $c = 0.5$. We implement bisect2.m as

```
    tic

    itmax=100;

    if sign(f(a))==sign(f(b))
       'failure to bracket root'
```

```
        return
    end

    k=0
    [a b b-a]
    xit=[];

    while abs(b-a)>tol*max(abs(b),1.0)
         if k+1>itmax
           break
         end
         x=(a+b)/2;
         xit=[xit;x];
         if sign(f(x)) == sign(f(b))
            b=x;
         else
            a=x;
         end
         k=k+1
         [a b b-a]
    end
    x=(a+b)/2
    xit=[xit;x];

    toc

```

From output of the above code, we find $c \approx 0.3671$ and $p \approx 0.9822$ by fitting the plot of $\ln(e_{n+1})$ vs. $\ln(e_n)$ with a straight line of slope p and intercept $\ln(c)$, as shown below.



(b) Newton's method has quadratic convergence, so we expect $p = 2$. Also we expect

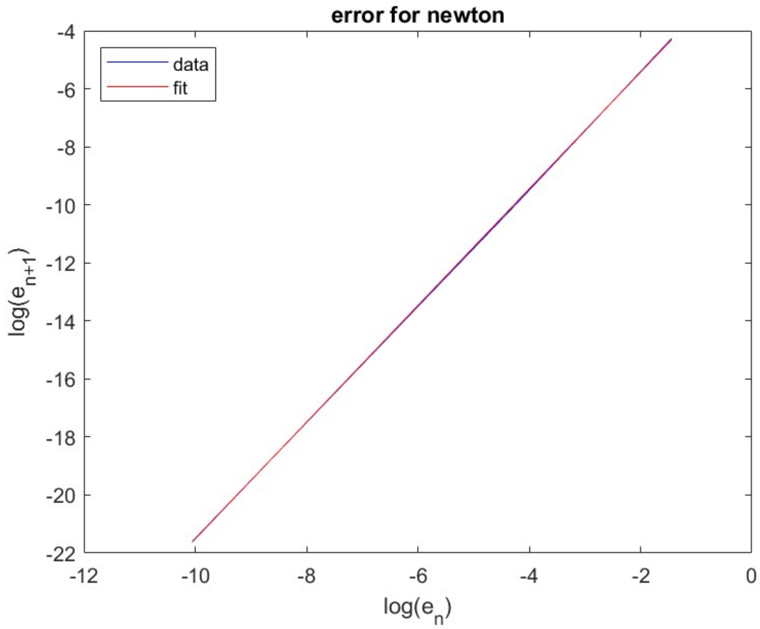
$$c = |\frac{f''(x^*)}{2f'(x^*)}| = \frac{2}{2(2x)_{x^*=\sqrt{5}}} \approx 0.22360679775$$

We implement `newton2.m` as

```
tic

itmax=100;

k=0;
if x ~= 0
  xold=0;
else
  xold=1;
end
[x abs(x-xold)];
xit=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

```

Numerically we find that $p \approx 2.0109$ and $c \approx 0.2452$.



(c) The secant method converges with a power of the golden ratio, so we expect

$$p = \varphi := (1+\sqrt{5})/2 \approx 1.618033988749895$$

Also we expect

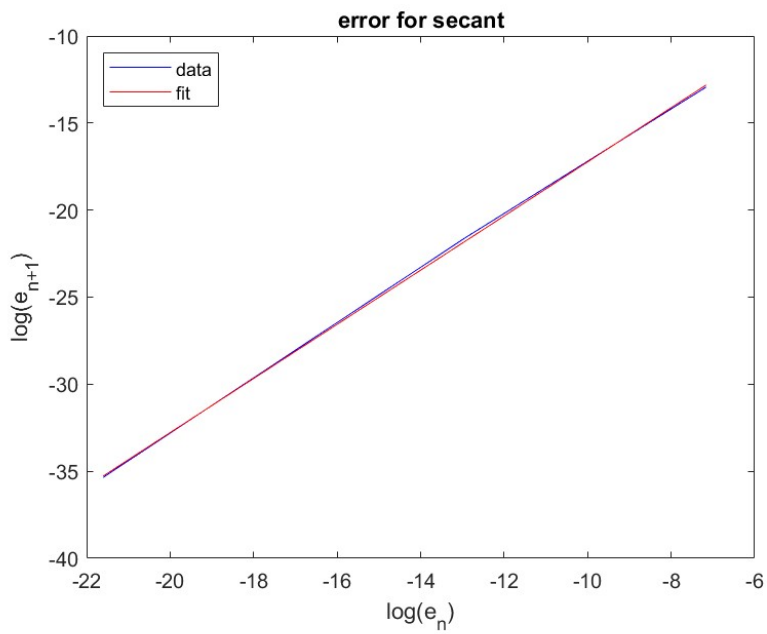
$$c = \left|\frac{f''(x^*)}{2f'(x^*)}\right|^{\varphi-1} = \left|\frac{2}{2(2x)_{x^*=\sqrt{5}}}\right|^{\varphi-1} = \left|\frac{1}{2\sqrt{5}}\right|^{\varphi-1} \approx 0.396241191724$$

. We implement `secant2.m` as

```
    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
```

Numerically we find that $p \approx 1.5552$ and $c \approx 0.1869$.



(d) The inverse quadratic interpolation (IQI) method converges with rate p given by the positive root of $x^3 - x^2 - x - 1$, or $p \approx 1.839286755214161$, with

$$c = |\frac{f'''(x^*)}{2f'(x^*)}|^{(p-1)/2} = |\frac{0}{2\sqrt{5}}|^{(p-1)/2} = 0!$$

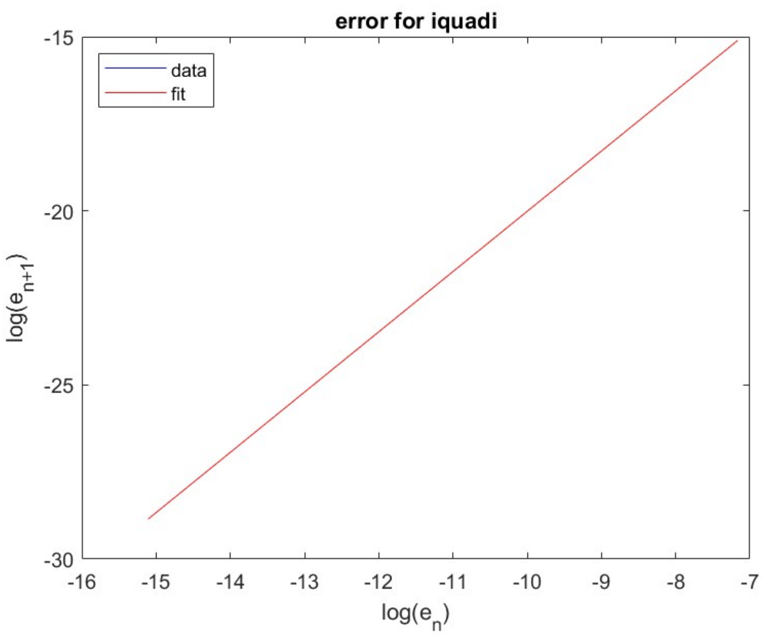
This means that, in this particular example, the order of convergence of IQI is in fact greater than the usual conventional value. This is not surprising, because $f(x) = x^2 - 5$ is a quadratic function and IQI is based on interpolating with quadratic polynomials.

Next `iquadi2.m` can be implemented as follows:

```
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=[xit;c];
        [c abs(c-b)];
    end

    [c abs(c-b)];

    toc

```

Numerically we find that $p \approx 1.7304$ and $c \approx 0.0665$. We see indeed that c is very small. Presumably more iterations in much high-order arithmetic (such as quadruple precision) would verify a faster rate of convergence.



Problem 2. (a) First off,

$$\begin{aligned}
x_{n+1} :&= x_n - \frac{f(x_n)}{f'(x_n)} \\
&= x_n - \Big(\frac{(1+x_n^4)4x_n^3 - (x_n^4)4x_n^3}{(1+x_n^4)^2}\Big)^{-1}\frac{x_n^4}{1+x_n^4} \\
&= x_n - \frac{(1+x_n^4)x_n^4}{(1+x_n^4 - x_n^4)4x_n^3} \\
&= x_n - \frac{(1+x_n^4)x_n}{4} \\
&= \frac{4x_n - x_n - x_n^5}{4} \\
&= \frac{3x_n}{4} - \frac{x_n^5}{4}
\end{aligned}$$

We'll try to look for a point so that $x_{n+1} = -x_n$, so then

$$\begin{aligned}
\frac{3x_n}{4} - \frac{x_n^5}{4} &= -x_n \\
-\frac{x_n^5}{4} &= -\frac{7}{4}x_n \\
x_n^4 &= 7 \\
x_n &= \pm 7^{\frac{1}{4}}
\end{aligned}$$

Here is some code that will verify all of this for $x_0 = 7^{\frac{1}{4}} + eps$, where the Newton iterates diverge to `INF`.

```
f=@(x) x.^4./(1+x.^4);
Df=@(x) 4*x.^3./(1+x.^4).^2;

tol=1e-15;


x=7^(1/4)+eps

newton
pause

x=1
newton

```

It hits `INF` at $k = 22$ - it doesn't take long to diverge!

(b) Note that since $x = 0$ is a root of multiplicity 4, newton has only a linear rate of convergence.

Since

$$x_{n+1} = \frac{3x_n}{4} - \frac{x_n^5}{4} \cong \frac{3}{4}x_n$$

near 0 we expect $\lambda = \frac{3}{4}$. It takes 115 iterations of `newton` to converge with tolerance of 10^{-15} starting at $x_0 := 1$. In the end,

$$\frac{x_{115}}{x_{114}} = 0.7500$$

which is almost exactly our prediction.

Problem 3. In `MATLAB`.

```
xi=single(c(1:8))
%   xi contains the x_{n} for n = 1,...,8
%   xs is the true root

x=xi(3:8);
xp=xi(2:7);
xpp=xi(1:6);

lamest=(x-xp)./(xp-xpp);
errest=lamest.*(x-xp)./(lamest-1);
xx=x-errest;

lamestf=lamest(6)
lam

errestf=errest(6)
error=x(6)-xs

impestf=xx(6)
xss=single(xs)
imperrf=xx(6)-xs

```

(a) We estimate

$$\lambda = \frac{(x_8 - x_7)}{(x_7 - x_6)} \approx -0.3098144$$

The true $\lambda = -0.310074222676809$.

(b) Apply the contraction mapping theorem: $(x_* - x_8) \approx \lambda(x_8 - x_7)/(1 - \lambda) \approx -1.3204626 \times 10^{-04}$, with the latter approximation using the λ estimate from (a). The true $(x_* - x_8)$ is $-1.3218004 \times 10^{-04}$.

(c) Using the error estimate from (b) : $\hat{x}_* := x_8 + (x_* - x_8) \approx 0.6013468$. This agrees to seven digits with the exact answer $x_* = 0.601346767725820$ and greatly improving upon x_8, which is only accurate to three digits.

Problem 4. Take $g(x) := x - f(x)/f'(x) - f(x)^2 f''(x)/2f'(x)^3$ and note that

$$g'(x) = 1 - \frac{f'(x)}{f'(x)} + \frac{f(x)}{(f'(x))^2} f''(x) - \frac{2f(x)f'(x)f''(x)}{2f'(x)^3} - f(x)^2 \frac{\mathrm{d}}{\mathrm{d}x} \frac{f''(x)}{2f'(x)^3} = -f(x)^2 \frac{\mathrm{d}}{\mathrm{d}x} \frac{f''(x)}{2f'(x)^3},$$

$$g''(x) = -2f(x)f'(x) \frac{\mathrm{d}}{\mathrm{d}x} \frac{f''(x)}{2f'(x)^3} - f(x)^2 \frac{\mathrm{d}^2}{\mathrm{d}x^2} \frac{f''(x)}{2f'(x)^3},$$

so that $g'(x_*), g''(x_*) = 0$. Lastly

$$g'''(x) = -2f'(x)^2 \frac{\mathrm{d}}{\mathrm{d}x} \frac{f''(x)}{2f'(x)^3} - 2f(x) \frac{\mathrm{d}}{\mathrm{d}x} \left[f'(x) \frac{\mathrm{d}}{\mathrm{d}x} \frac{f''(x)}{2f'(x)^3} \right] - 2f(x)f'(x) \frac{\mathrm{d}^2}{\mathrm{d}x^2} \frac{f''(x)}{2f'(x)^3} - f(x)^2 \frac{\mathrm{d}^3}{\mathrm{d}x^3} \frac{f''(x)}{2f'(x)^3}.$$

In particular

$$g'''(x_*) = -2f'(x_*)^2 \frac{\mathrm{d}}{\mathrm{d}x} \frac{f''(x_*)}{2f'(x_*)^3} \neq 0 \quad \text{(generically)}.$$

Thus, the general theorem on fixed-point iterations tells us that iteration with this g has order of convergence $p = 3$.

Problem 5. (a) Write

$$\begin{aligned}(A+xy^\top)\left(A^{-1}-\frac{A^{-1}xy^\top A^{-1}}{1+\langle y,A^{-1}x\rangle}\right)&=I+xy^\top A^{-1}-\frac{xy^\top A^{-1}+xy^\top A^{-1}xy^\top A^{-1}}{1+\langle y,A^{-1}x\rangle}\\&=I+xy^\top A^{-1}-\frac{xy^\top A^{-1}+x\langle y,A^{-1}x\rangle y^\top A^{-1}}{1+\langle y,A^{-1}x\rangle}\\&=I+xy^\top A^{-1}-\frac{xy^\top A^{-1}(1+\langle y,A^{-1}x\rangle)}{1+\langle y,A^{-1}x\rangle}\\&=I+xy^\top A^{-1}-xy^\top A^{-1}=I\end{aligned}$$

(b) Recall that $A_n = A_{n-1}+(\Delta f_{n-1}-A_{n-1}\Delta x_{n-1})\Delta x_{n-1}^\top/\|\Delta x_{n-1}\|_2^2$. Use the Sherman–Morrison formula to write

$$\begin{aligned}A_n^{-1}&=A_{n-1}^{-1}-\frac{A_{n-1}^{-1}(\Delta f_{n-1}-A_{n-1}\Delta x_{n-1})\Delta x_{n-1}^\top A_{n-1}^{-1}/\|\Delta x_{n-1}\|_2^2}{1+\left\langle \Delta x_{n-1},A_{n-1}^{-1}(\Delta f_{n-1}-A_{n-1}\Delta x_{n-1})/\|\Delta x_{n-1}\|_2^2\right\rangle}\\&=A_{n-1}^{-1}-\frac{A_{n-1}^{-1}(\Delta f_{n-1}-A_{n-1}\Delta x_{n-1})\Delta x_{n-1}^\top A_{n-1}^{-1}}{\|\Delta x_{n-1}\|_2^2+\left\langle \Delta x_{n-1},A_{n-1}^{-1}\Delta f_{n-1}-\Delta x_{n-1}\right\rangle}\\&=A_{n-1}^{-1}-\frac{A_{n-1}^{-1}(\Delta f_{n-1}-A_{n-1}\Delta x_{n-1})\Delta x_{n-1}^\top A_{n-1}^{-1}}{\left\langle \Delta x_{n-1},A_{n-1}^{-1}\Delta f_{n-1}\right\rangle}\\&=A_{n-1}^{-1}+\frac{(\Delta x_{n-1}-p_{n-1})\Delta x_{n-1}^\top A_{n-1}^{-1}}{\langle \Delta x_{n-1},\Delta p_{n-1}\rangle}\qquad\text{with } \Delta p_{n-1}:=A_{n-1}^{-1}\Delta f_{n-1}\end{aligned}$$

.

Problem 6. Note that

$$\mathbf{Df}(\mathbf{x}) = \begin{bmatrix} 2x & 2y & 4z^3 \\ 2(x-1)+y & x+2y & 0 \\ 1 & 1 & \ln(z)+1 \end{bmatrix}.$$

In MATLAB:

```
f=@(x) [x(1)^2+x(2)^2+x(3)^4-16; (x(1)-1)^2+x(1)*x(2)+x(2)^2-1; x(1)+x(2)+x(3)*log(x(3))-2];
Df=@(x)[2*x(1) 2*x(2) 4*x(3)^3; 2*(x(1)-1)+x(2) x(1)+2*x(2) 0; 1  1  log(x(3))+1];

tol=eps
timnewt=0;
timbroy=0;
for ntry=1:10000

x=[0;0;2];
tic
[xnewt,itnewt]=newtraph(f,Df,x,tol);
timnewt=timnewt+toc;
x=[0;0;2];
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

```

With $N_{\text{repeat}} = 10^4$, Newton–Raphson runs in 6 iterations, Broyden in 11. Relative mean clock-time of Newton–Raphson to Broyden is $1.1 - 1.2$ (depending on the run), so that Broyden is very slightly faster than Newton for this problem.

Problem 7. In MATLAB:

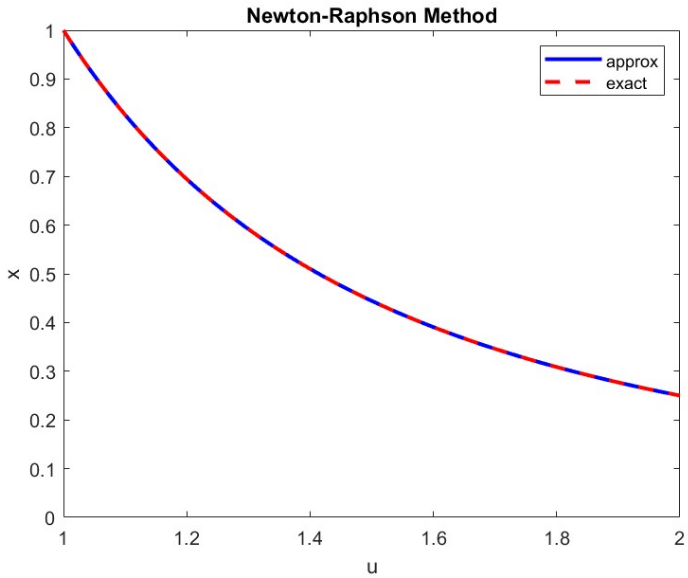
```
n=1000;

u=(1:n).';
u=1+u/(n+1);

x=(7-3*u)/4;
tol=1e-8;
itmax=5000;

newtraph

X=[1;x;.25];
U=[1;u;2];

Y=@(u) u.^(-2);

plot(U,X,'-b',U,Y(U),'--r','LineWidth',2)
xlabel('u')
ylabel('x')
legend('approx','exact')
title('Newton-Raphson Method')
axis([1 2 0 1])
err=norm(X-Y(U))/norm(Y(U))
k=k
pause

x=(7-3*u)/4;
tol=1e-8;

broyden

X=[1;x;.25];
figure
plot(U,X,'-b',U,Y(U),'--r','LineWidth',2)
xlabel('u')
ylabel('x')
legend('approx','exact')
title('Broyden Method')
axis([1 2 0 1])
err=norm(X-Y(U))/norm(Y(U))
k=k
pause
```

```
    x=(7-3*u)/4;
    options=optimoptions('fsolve','Algorithm','Levenberg-Marquardt','Display','iter','FunctionTolerance',1e-8)
    x=fsolve(@(x)f(x),x,options);
    err=norm(X-Y(U))/norm(Y(U))

    X=[1;x;.25];
    figure
    plot(U,X,'-b',U,Y(U),'--r','LineWidth',2)
    xlabel('u')
    ylabel('x')
    legend('approx','exact')
    title('Levenberg-Marquardt Method')
    axis([1 2 0 1])

```

where f.m and Df.m are respectively implemented as

```
function z = f(y)
    n=length(y);
    dx=1/(n+1);

    Y=[1;y;.25];
    z=zeros(size(y));
    for i=1:n
        z(i)=-Y(i+2)-Y(i)+2*Y(i+1)+6*dx^2*Y(i+1)^2;
    end

function Z = Df(y)
    n=length(y);
    dx=1/(n+1);

    Y=[1;y;.25];
    dd=2*(1+6*dx^2*y);
    Z=diag(dd);
    for i=1:n-1
        Z(i,i+1)=-1;
    end
    for i=2:n;
        Z(i,i-1)=-1;
    end

```

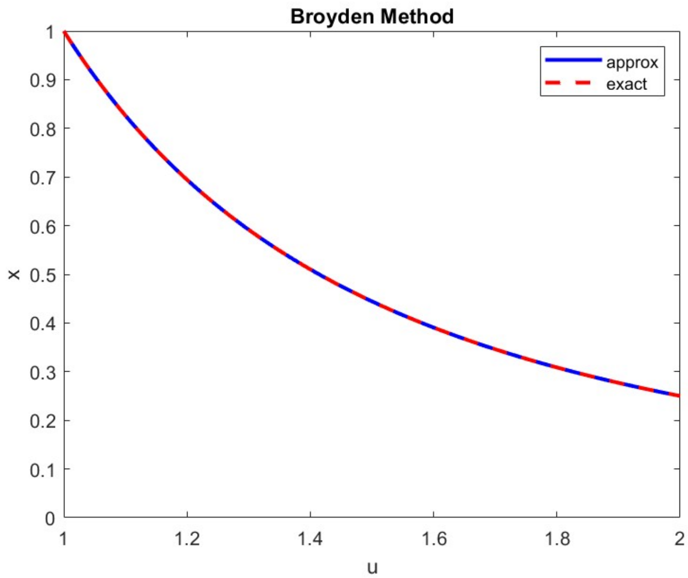
(a) Newton–Raphson has 5 iterations with respective errors 20.9142617286947, 4.11260222463596, 0.236972685599119, 0.000832872072700299, 1.04242895609589$e-08$.



(b) Broyden has 6 iterations with respective errors

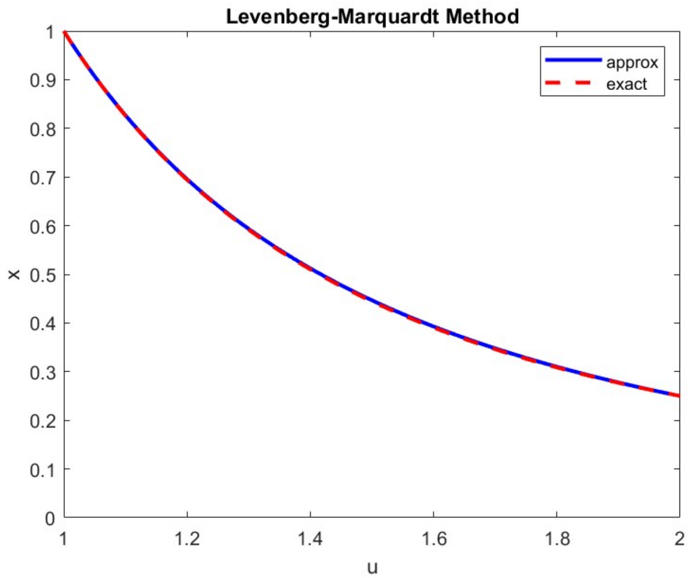
Error Estimate
20.9142617286947
4.11260222463463
0.22352237486458
0.0142528070810406
6.74124501888003e-05
1.10850320098538e-06
2.77989075534263e-08

And here is the plot of the numerical solution together with the exact solution:



(c) Levenberg–Marquadt has 10 iterations. The output of `fsolve` is

Iteration	Func-count	||f(x)||^2	First-order optimality	Lambda	Norm of step
0	1001	9.54392e-09	5.99e-06	0.01	
1	2002	9.46053e-09	1.9e-07	0.001	4.45822e-05
2	3003	9.37631e-09	5.41e-08	0.0001	0.000171579
3	4004	9.23083e-09	1.7e-08	1e-05	0.000719617
4	5005	8.97055e-09	5.38e-09	1e-06	0.00304925
5	6006	8.49645e-09	1.72e-09	1e-07	0.0130528
6	7007	7.6011e-09	5.55e-10	1e-08	0.0570568
7	8008	5.84405e-09	1.8e-10	1e-09	0.253133
8	9009	2.79562e-09	5.3e-11	1e-10	1.02686
9	10010	2.16657e-10	1.11e-11	1e-11	2.28532
10	11011	5.64263e-13	6.4e-13	1e-12	0.89575


Levenberg-Marquardt Method
approx
exact
x
u
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
1.2
1.4
1.6
1.8
2

Problem 8. (a) The steepest descent direction is

$$\mathbf{d}_n = -\nabla \Phi(\mathbf{x}_n)$$

Now taking the partial derivative of $\phi(x)$ in terms of x_i, we see that

$$\frac{\partial}{\partial x_i}\Phi(\mathbf{x}) = \frac{1}{2}\sum_{j=1}^{d} \frac{\partial}{\partial x_i} f_j(\mathbf{x})^2 = \sum_{j=1}^{d} \mathbf{Df}(\mathbf{x})_{ji} f_j(\mathbf{x}),$$

So therefore we get that

$$\mathbf{d}_n = -\mathbf{Df}(\mathbf{x}_n)^\top \mathbf{f}(\mathbf{x}_n)$$

(b) In order to show this we evaluate the following inner product:

$$\begin{aligned}
\langle \Delta \mathbf{x}, \nabla \Phi(\mathbf{x})\rangle &= -\langle \mathbf{Df}(\mathbf{x})^{-1}\mathbf{f}(\mathbf{x}), \mathbf{Df}(\mathbf{x})^\top \mathbf{f}(\mathbf{x})\rangle \\
&= -\langle \mathbf{f}(\mathbf{x}), (\mathbf{Df}(\mathbf{x})^{-1})^\top \mathbf{Df}(\mathbf{x})^\top \mathbf{f}(\mathbf{x})\rangle \\
&= -\|\mathbf{f}(\mathbf{x})\|_2^2 \\
&\leq 0
\end{aligned}$$