MATH 151B Homework 6

Zheng Wang (404855295)

June 6, 2019

Question 1

Define

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \text{and} \qquad \mathbf{F}(\mathbf{x}) = \begin{bmatrix} x_1^2 + x_2 - 37 \\ x_1 - x_2^2 - 5 \\ x_1 + x_2 + x_3 - 3 \end{bmatrix}$$

We then compute the Jacobian matrix as follows

$$J(\mathbf{x}) = \begin{bmatrix} 2x_1 & 1 & 0 \\ 1 & -2x_2 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Now, since we have $\mathbf{x}^{(0)} = [1, 1, 1]^t$ the first iteration of the Newton's Method is given as

$$\begin{aligned} \mathbf{x}^{(1)} &= \mathbf{x}^{(0)} - J(\mathbf{x}^{(0)})^{-1}\mathbf{F}(\mathbf{x}^{(0)}) \\ &= \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 2 & 1 & 0 \\ 1 & -2 & 0 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} -35 \\ -5 \\ 0 \end{bmatrix} = \begin{bmatrix} 16 \\ 6 \\ -19 \end{bmatrix} \end{aligned}$$

We observe that the correct solution of the system is $(6, 1, -4)^T$, we see that the error of the first iteration is $||(6, 1, -4)^T - (16, 6, -19)^T|| = 18.7083$, larger than the initial error of 7.0711. This is possible for Newton's method, this is because the assumption of the convergence for Newton's Method requires the initial value to be "close" enough to the actual solution, it is possible that $(1, 1, 1)^T$ does not satisfy this condition.

Using the following code, we obtain that the solution with **4** iterations is $\mathbf{x}^{(4)} = \begin{bmatrix} 6.0006 \\ 1.2091 \\ -4.2097 \end{bmatrix}$ and the solution with **8** iterations is $\begin{bmatrix} 6.0000 \\ 1.0000 \\ -4.0000 \end{bmatrix}$.

The solution obtianed by 8 iteration is exact, with error of 0, and the solution with 4 iteration is very close to the actual solution, with error of 0.2961 ($||(6.0006, 1.2091, -4.2097)^T - (6, 1, -4)^T|| = 0.2961$). In general, 8 iteration give better estimate of the solution than 4 iteration.

```matlab
% MATH 151b, HW 6
% Question 1
fprintf('Solution with 4 iteration is:\n')
disp(Newton(@J,@F,4,[1;1;1]))
fprintf('Solution with 8 iteration is:\n')
disp(Newton(@J,@F,8,[1;1;1]))

% The function to solve
function Y = F(x1,x2,x3)
    y = zeros(3,1);
    y(1) = x1^2 + x2 -37;
    y(2) = x1 - x2^2 - 5;
    y(3) = x1 + x2 + x3 -3;
    Y = y;
end

% The Jacobian of the function
function Jac = J(x1,x2,x3)
    Jacobian = zeros(3);
    Jacobian(1,1) = 2*x1;
    Jacobian(1,2) = 1;
    Jacobian(2,1) = 1;
    Jacobian(2,2) = -2*x2;
    Jacobian(3,1) = 1;
    Jacobian(3,2) = 1;
    Jacobian(3,3) = 1;
    Jac = Jacobian;
end

% Newton's method that solves the function
% INPUTS: J - the Jacobian, F - The function to solve,
%         N - Max iteration, ini - initial guess
function X = Newton(J,F,N,ini)
    x_old = ini;
    for i = 1:N
        Jac = J(x_old(1), x_old(2), x_old(3));
        Y = F(x_old(1), x_old(2), x_old(3));
        x_new = x_old - Jac\Y;
        x_old = x_new;
    end
    X = x_new;
end
```

The output from the consoles are listed below:

```
>> q1
Solution with 4 iteration is:
    6.0006
    1.2091
   -4.2097

Solution with 8 iteration is:
    6.0000
    1.0000
   -4.0000
```

Question 2

We can use the following finite difference formula (The error terms are truncated):

$$y''(x_i) = \frac{y(x_{i+1}) - 2y(x_i) + y(x_{i-1})}{h^2} \quad \text{and}$$

$$y'(x_i) = \frac{y(x_{i+1}) - y(x_{i-1})}{2h}$$

to obtain the following system of equations for $i = 1, 2, ..., 7$:

$$\frac{y(x_{i+1}) - 2y(x_i) + y(x_{i-1})}{h^2} = f\left(x_i, y(x_i), \frac{y(x_{i+1}) - y(x_{i-1})}{2h}\right)$$

In this perticular case, we see that since $f(x, y, y') = 2y^3$, we have the systems of equations has the following form (for $i = 1, 2, ..., 7$):

$$2h^2 w_i^3 - w_{i+1} + 2w_i - w_{i-1} = 0$$

Now as $w_0 = \dfrac{1}{2}$ and $w_8 = \dfrac{1}{3}$, the system of equations are

$$\begin{cases} 2h^2 w_1^3 + 2w_1 - w_2 = \frac{1}{2} \\ 2h^2 w_2^3 - w_3 + 2w_2 - w_1 = 0 \\ \quad\vdots \qquad\qquad \vdots \\ 2h^2 w_6^3 - w_7 + 2w_6 - w_5 = 0 \\ 2h^2 w_7^3 + 2w_7 - w_6 = \frac{1}{3} \end{cases}$$

Therefore, we could define $\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_7 \end{bmatrix}$ and $\mathbf{F}(\mathbf{w}) = \begin{bmatrix} 2h^2 w_1^3 + 2w_1 - w_2 - \frac{1}{2} \\ 2h^2 w_2^3 - w_3 + 2w_2 - w_1 \\ \vdots \\ 2h^2 w_6^3 - w_7 + 2w_6 - w_5 \\ 2h^2 w_7^3 + 2w_7 - w_6 - \frac{1}{3} \end{bmatrix}$. Thus, $J((w)) =$

$$\begin{bmatrix} 6h^2 w_1^2 + 2 & -1 & 0 & \dots & \dots & \dots & 0 \\ -1 & 6h^2 w_2^2 + 2 & -1 & 0 & \dots & \dots & 0 \\ 0 & -1 & 6h^2 w_3^2 + 2 & -1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & \ddots & \ddots & \ddots & -1 \\ 0 & \dots & \dots & \dots & 0 & -1 & 6h^2 w_7^2 + 2 \end{bmatrix}.$$

Thus, we can use the following code to compute that $y\left(-\dfrac{1}{2}\right) = 0.4000$ (Which correspond to w_4, the forth element in the output):

```
% MATH 151b, HW 6
% Question 2
x = [1;1;1;1;1;1;1];
disp(broyden(x,@f,7,10^-5))

function y=f(w)
N = size(w,1);
h = 1/(N+1);
result = zeros(N,1);
result(1) = 2*h^2*w(1)^3 + 2*w(1) - w(2) - 1/2;
result(N) = 2*h^2*w(N)^3 + 2*w(N) - w(N-1) - 1/3;
for i=2:(N-1)
    result(i) = 2*h^2*w(i)^3 + 2*w(i) - w(i-1) - w(i+1);
end
y = result;
end

% Broyden?s Method
function [xv,it]=broyden(x,f,n,tol)
% Broyden's method for solving a system of n non-linear equations
% in n variables.
%
% Example call: [xv,it]=broyden(x,f,n,tol)
% Requires an initial approximation column vector x. tol is required
% accuracy. User must define function f
% xv is the solution vector, parameter it is number of iterations
% taken. WARNING. Method may fail, for example, if initial estimates
% are poor.
%
fr=zeros(n,1); it=0; xv=x;
%Set initial Br
h = 1/(n+1);
Br=zeros(n);
Br(1,1) = 6*h^2*x(1)^2+2;
Br(1,2) = -1;
Br(n,n-1) = -1;
Br(n,n) = 6*h^2*x(n)^2+2;
for i=2:(n-1)
    Br(i,i-1)=-1;
    Br(i,i)= 6*h^2*x(i)^2+2;
    Br(i,i+1) = -1;
end
fr=feval(f, xv);
while norm(fr)>tol
  it=it+1;
  pr=-Br*fr;
  tau=1;
  xv1=xv+tau*pr; xv=xv1;
  oldfr=fr; fr=feval(f,xv);
  %Update approximation to Jacobian using Broydens formula
  y=fr-oldfr; oldBr=Br;
  oyp=oldBr*y-pr; pB=pr'*oldBr;
  for i=1:n
    for j=1:n
      M(i,j)=oyp(i)*pB(j);
    end
```

```
    end
  Br=oldBr-M./(pr'*oldBr*y);
end
end
```

The output from the console is:

```
>> q2
    0.4706
    0.4445
    0.4211
    0.4000
    0.3810
    0.3637
    0.3478
```

Question 3

We first define $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$ and define $\mathbf{F}(\mathbf{x}) = \begin{bmatrix} x_1^3 + x_1^2 x_2 - x_1 x_3 + 6 \\ e^{x_1} + e^{x_2} - x_3 \\ x_2^2 - 2x_1 x_3 - 4 \end{bmatrix}$.
We can then compute the Jacobian, which is the following:

$$J(\mathbf{x}) = \begin{bmatrix} 3x_1^2 + 2x_1x_2 - x_3 & x_1^2 & -x_1 \\ e^{x_1} & e^{x_2} & -1 \\ -2x_3 & 2x_2 & -2x_1 \end{bmatrix}$$

Thus, let $g(\mathbf{x}) = ||\mathbf{F}(\mathbf{x})||_2^2$, and the fact that $\nabla g(\mathbf{x}) = 2J(\mathbf{x})^T \mathbf{F}(\mathbf{x})$, we can implement the following algorithm to obtain the solution of the function.

```
% MATH 151b, HW 6
% Question 3
x = [1;1;1];
fprintf('Use Tolerance = 0.01\n')
solution = Ste_Dec(@F,@J,@g,x,0.01,100000000);
fprintf('Solution find is:\n')
disp(solution)
fprintf('Check the solution is close to actual solution, F(x) is\n')
disp(F(solution))
fprintf('\n\n');
fprintf('Use Tolerance = 10^-5\n')
solution = Ste_Dec(@F,@J,@g,x,10^-5,100000000);
fprintf('Solution find is:\n')
disp(solution)
fprintf('Check the solution is close to actual solution, F(x) is\n')
disp(F(solution))


function Y = F(x)
    y = zeros(3,1);
    y(1) = x(1)^3 + x(1)^2*x(2) - x(1)*x(3) + 6;
    y(2) = exp(x(1)) + exp(x(2)) - x(3);
```

```
    y(3) = x(2)^2 - 2*x(1)*x(3) - 4;
    Y = y;
end

function Jacb = J(x)
    Jac = zeros(3);
    Jac(1,1) = 3*x(1)^2 + 2*x(1)*x(2) - x(3);
    Jac(1,2) = x(1)^2;
    Jac(1,3) = -x(1);
    Jac(2,1) = exp(x(1));
    Jac(2,2) = exp(x(2));
    Jac(2,3) = -1;
    Jac(3,1) = -2*x(3);
    Jac(3,2) = 2*x(2);
    Jac(3,3) = -2*x(1);
    Jacb = Jac;
end

function y = g(x)
    f1 = x(1)^3 + x(1)^2*x(2) - x(1)*x(3) + 6;
    f2 = exp(x(1)) + exp(x(2)) - x(3);
    f3 = x(2)^2 - 2*x(1)*x(3) - 4;
    y = f1^2 + f2^2 + f3^2;
end

function result = Ste_Dec(F,J,g,ini,tol,max_iter)
    x = ini;
    k = 1;
    while k <= max_iter
        g1 = g(x);
        z = 2*J(x).'*F(x);
        z0 = norm(z);
        if z0 == 0
            result = x;
            fprintf('Iteration number:');
            disp(k);
            return
        end
        z = z/z0;
        alpha1 = 0;
        alpha3 = 1;
        g3 = g(x-alpha3*z);
        while g3 >= g1
            alpha3 = alpha3/2;
            g3 = g(x-alpha3*z);
            if alpha3 < tol/2
                fprintf('No likely improvement\n');
                result = x;
                fprintf('Iteration number:');
                disp(k);
                return
            end
        end
        alpha2 = alpha3/2;
        g2 = g(x-alpha2*z);
        % solve for minimum of the interpolation function
```

```
        h1 = (g2-g1)/alpha2;
        h2 = (g3-g2)/(alpha3-alpha2);
        h3 = (h2-h1)/alpha3;
        alpha0 = 0.5*(alpha2-h1/h3);
        g0 = g(x-alpha0*z);
        if g3 <= g0
            alpha = alpha3;
            g_val = g3;
        else
            alpha = alpha0;
            g_val = g0;
        end
        x = x-alpha*z;
        if abs(g_val-g1) < tol
            result = x;
            fprintf('Iteration number:');
            disp(k);
            return
        end
        k = k+1;
    end
    result = x;
    fprintf('Reach max iteration\n');
end
```

The console output the following information:

```
>> q3
Use Tolerance = 0.01
Iteration number:   590

Solution find is:
    0.1565
    2.2333
    9.5493

Confirm that the solution is correct
    4.5639
    0.9505
   -2.0018




Use Tolerance = 10^-5
Iteration number:      144145

Solution find is:
    0.1216
    3.7185
   42.2514
```

```
Confirm that the solution is correct
    0.9178
    0.0805
   -0.4507
```

We can see that the solution from Steepest Desent is not exact, but close to the actual result. This is because the stopping condition of the algorithm is that $g(\mathbf{x})$ no longer changes too much. This could happen when the gradient is small. However, since the gradient is not exactly zero, the $\mathbf{x}$ we obtianed will not be the actual solution. The solution will be closer to the actual solution if we set the tolerance to be smaller, as shown by the result of the code. When the tolerance is set to 10^{-5}, the solution we obtianed is $(0.1216, 3.7185, 42.2514)^T$.