3. 2.5 Computer Problems 1

Use the Jacobi Method to solve the sparse system within six correct decimal places (forward error in the infinity norm) for n = 100 and n = 100000. The correct solution is [1, . . . ,1]. Report the number of steps needed and the backward error.

Run for n =100 case, steps = 36, b_error = 4.5785e-07

```
[a,b]=sparsesetup(100);
[steps,b_error]=Jacobi(a,b,0.00000025)
```

Run for n=100000 case, steps = 48, b_error = 2.7012e-06

```
% first, set up sparse matrix
function [a,b] = sparsesetup(n)
e = ones(n,1);
a = spdiags([-e 3*e -e],-1:1,n,n); % Entries of a
b=zeros(n,1); % Entries of r.h.s. b
b(1)=2;
b(n)=2;
b(2:n-1)=1;
end
function[steps,b_error]=Jacobi(a,b,tol)
n=length(b);
d=diag(a);
r=a-diag(d);
x=zeros(n,1);
p=zeros(n,1);
c=zeros([n,1]);
e=zeros([n,1]);
n1=0; error=0; rel_error=1;
while (1)
    x=(b-r*x)./d;
error=abs(norm(x-p));
rel_error=error/(norm(x)+eps);
p=x; n1=n1+1;
if (error<tol)||(rel_error<tol)
    break
end
end
xc=x;
steps=n1;
% correct solution of the system
for i=1:n
    for j=1:n
        xa(j)=1;
```

```
        c(i)=c(i)+a(i,j)*xa(j);
        e(i)=e(i)+a(i,j)*xc(j);
    end
end
% calculate forward and backward errors
for i=1:n
    f_dif(i)=abs(xa(i)-xc(i));
    b_dif(i)=abs(e(i)-c(i));
end
f_error=max(f_dif,[],2);
b_error=max(b_dif,[],2);
end
```

4. 2.5 Computer Problems 1

Use the Gaussian-Seidel Method to solve the sparse system

By Gaussian-Seidel Method, sparse system solves to x = [1,1,1,...,1], dim =100
Similarly, solves to x = [1,1,1,...,1], dim=100000, when n =100000

```
% Guassian Seidel Method
function x= guassian_seidel(a,b,tol,m)
n=length(b);
p=zeros(n,1);
for k=1:m
    for j=1:n
        if j==1
            x(1)=(b(1)-a(1,2:n)*p(2:n))/a(1,1);
        elseif j==n
            x(n)=(b(n)-a(n,1:n-1)*(x(1:n-1))')/a(n,n);
        else
            x(j)=(b(j)-a(j,1:j-1)*(x(1:j-1))'-a(j,j+1:n)*p(j+1:n,1))/a(j,j);
        end
    end
    error=abs(norm(x'-p));
    rel_error=error/(norm(x)+eps);
    p=x';
    if (error<tol)|(rel_error<tol)
        break
    end
end
x=x';
end
```

6. 2.6 Computer Problems 5

Use the Conjugate Gradient Method to solve (2.45) for n = 100,1000, and 10, 000. Report the size of the final residual, and the number of steps required.

For n =100 case, if we want to be precise to six decimals, it takes 16 steps, and final residual to be 8.3028e-08
For n = 1000 case, it takes 16 steps, and final residual to be 8.1708e-08
For n = 10000 case, it takes 16 steps, and final residual to be 8.1633e-08

```
[a,b]=sparsein(100);
[steps, residual]=conj_grad(a,b)

% conjugate gradient method
function[steps, residual] = conj_grad(a,b)
[n,n]=size(a);
x=zeros(size(b));
r=b;
z=r'*r;
p=r;
tol=0.00000025;
k=0;
while(n)
    q=a*p;
    u=z/(p'*q);
    x=x+u*p;
    z0=z;
    r=r-u*q;
    z=r'*r;
    v=z/z0;
    p=r+v*p;
    res=norm(r,inf);
    k=k+1;
    if (res<tol)
        break;
    end
end
residual = norm((b-a*x),inf);
steps=k;
return
end
```

8. 2.7 Computer Problems 1

Implement Newton's Method with appropriate starting points to find all solutions. Check with Exercise 3 to make sure your answers are correct.

(a)

```
P=[0,1]; % set initial guess
[P,itr]=multi_newton('f','Jacobian_f',P,20)
```

Output:

P=[0.500000,0.8660254], steps=6

```
% define function f(X)
function F=f(X)
u = X(1);
v = X(2);
F = zeros(1,2);
F(1)=u^2+v^2-1;
F(2)=(u-1)^2+v^2-1;
end
% define jacobian of f(X)
function DF=Jacobian_f(X)
u=X(1);
v=X(2);
DF=[2*u,2*v;2*(u-1),2*v];
end
% calculate steps and iterative solution
function [P,steps]=multi_newton(f,Jacobian_f,P,max)
Y=feval(f,P);
tol=0.000001;
for i = i:max
    J=feval(DF,P);
    Q=P-(J\Y')';
    R=feval(f,Q);
    error=norm(Q-P);
    rel_error=error/(norm(Q)+eps);
    P=Q;
    Y=R;
    steps=i;
    if(rel_error<=tol)
        break;
    end
end
end
```

(b)

```
P=[0,1]; % set initial guess
[P,itr]=multi_newton('f','Jacobian_f',P,20)
```

Output:

P=[0.8944272,0.8944272], steps=5

```
% define function f(X)
function F=f(X)
u = X(1);
v = X(2);
F = zeros(1,2);
F(1)=u^2+4*v^2-4;
F(2)=4*u^2+v^2-4;
end
% define jacobian of f(X)
function DF=Jacobian_f(X)
u=X(1);
v=X(2);
DF=[2*u,8*v;8*u,2*v];
end
```

(c)

```
P=[0,1]; % set initial guess
[P,itr]=multi_newton('f','Jacobian_f',P,20)
```

Output:

P=[2.7595918,-0.9507033], steps=8

```
% define function f(X)
function F=f(X)
u = X(1);
v = X(2);
F = zeros(1,2);
F(1)=u^2-4*v^2-4;
F(2)=(u-1)^2+v^2-4;
end
% define jacobian of f(X)
function DF=Jacobian_f(X)
u=X(1);
v=X(2);
DF=[2*u,-8*v;2*(u-1),2*v];
end
```

9. 2.7 Computer Problems 7

Apply Broyden I with starting guesses x0 = (1,1) and A0 = I to the systems in Exercise 3. Report the solutions to as much accuracy as possible and the number of steps required.

(a)

```
func=@(x)[x(1)^2+x(2)^2-1;(x(1)-1)^2+x(2)^2-1];
P=[1;1];
[P,steps]=Broyden1(func,P,20)
```

Output:

P = [0.500000000000000

0.866025403786128]

Steps = 10

```
% Broyden I
function [P,steps]=Broyden1(func,P,stop)
[n,m]=size(P);
a=eye(n,n);
tol=0.000001;
for i=1:stop
    J=a\feval(func,P);
    Q=P-J;
    del=Q-P;
    delta=feval(func,Q)-feval(func,P);
    a=a+((delta-a*del)*(del'))/(del'*del);
    error=norm(Q-P);
    rel_error=error/(norm(Q)+eps);
    P=Q;
    steps=i;
    if(rel_error<=tol)
        break;
    end
end
end
```

(b)

```
func=@(x)[x(1)^2+4*x(2)^2-4;4*x(1)^2+x(2)^2-4];
P=[1;1];
[P,steps]=Broyden1(func,P,20)
```

Output:

P = [0.894427191005772

0.894427190994059]

Steps = 8

(c)

```
func=@(x)[x(1)^2-4*x(2)^2-4;(x(1)-1)^2+x(2)^2-4];
P=[1;1];
[P,steps]=Broyden1(func,P,40)
```

Output:

P = [2.75959179278683

-0.950703266403661]

Steps = 35

10. 2.7 Computer Problems 8

Apply Broyden II with starting guesses (1,1) and B0 = I to the systems in Exercise 3. Report the solutions to as much accuracy as possible and the number of steps required.

(a)

```
func=@(x)[x(1)^2+x(2)^2-1;(x(1)-1)^2+x(2)^2-1];
P=[1;1];
[P,steps]=Broyden1(func,P,20)
```

Output:

P = [0.500000000000000

0.866025403786128]

Steps = 10

```
% Broyden II
function [P,steps]=Broyden2(func,P,stop)
[n,m]=size(P);
b=eye(n,n);
tol=0.000001;
for i=1:stop
    J=b*(feval(func,P));
    Q=P-J;
    del=Q-P;
    delta=feval(func,Q)-feval(func,P);
    b=b+((b*del-delta)*(del'*b))/(del'*b*delta);
    error=norm(Q-P);
    rel_error=error/(norm(Q)+eps);
    P=Q;
    steps=i;
    if(rel_error<=tol)
        break;
```

```
    end
end
end
```

(b)

```
func=@(x)[x(1)^2+4*x(2)^2-4;4*x(1)^2+x(2)^2-4];
P=[1;1];
[P,steps]=Broyden1(func,P,20)
```

Output:

P = [0.894427190999916

0.894427190999916]

Steps = 8

(c)

```
func=@(x)[x(1)^2-4*x(2)^2-4;(x(1)-1)^2+x(2)^2-4];
P=[1;1];
[P,steps]=Broyden1(func,P,40)
```

Output:

P = [2.75959179278683

-0.950703266403665]

Steps = 35