PROJECT ASSIGNMENT -2 REPORT

MATH-6601

MUHAMMED EMIN OZTURK

Pseudo code of Jacobi and Gauss-Seidel Algorithm

1 algorithm

1.1 Jacobi

input A, b, $x^{(0)}$, TOL
$m = 0$
while $\frac{\left\|x^{(m+1)} - x^{(m)}\right\|_2}{\left\|x^{(0)}\right\|_2} \geq TOL$
  for $i = 1$ **to** n
    $x_i^{(m+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij} x_j^{(m)}\right)$
  end for
  $m = m + 1$
end while
output $x^{(m)}$, m

1.2 Gauss-Seidel

input A, b, $x^{(0)}$, TOL
$m = 0$
while $\frac{\left\|x^{(m+1)} - x^{(m)}\right\|_2}{\left\|x^{(0)}\right\|_2} \geq TOL$
  for $i = 1$ **to** n
    $x_i^{(m+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j < i} a_{ij} x_j^{(m+1)} - \sum_{j > i} a_{ij} x_j^{(m)}\right)$
  end for
  $m = m + 1$
end while
output $x^{(m)}$, m

In our assignment we just use $||r^m||_F$ / $||r^0||_F$ for checking convergence

1.3 SD

```
function [x,m] = SD(A,b,x0,TOL)
```



```
x=x0;
m=0;
tol=1; % you can set initial tol to be any large number
while tol >= TOL
    if m==0
        r=b-A*x0;
        r0=r;
    end
    alpha=r'*r/(r'*A*r);
    x=x+alpha*r;
    r=b-A*x;
    tol=norm(r)/norm(r0);
    m=m+1;
end
```

1.4 CG

```
function [x,m] = CG(A,b,x0,TOL)
x=x0;
m=0;
tol=1; % you can set initial tol to be any large number
while tol >= TOL
    if m==0
        r=b-A*x0;
        r0=r;
        p=r;
    end
    alpha=r'*r/(p'*A*p);
    x=x+alpha*p;
    r=r-alpha*A*p;
    beta=-r'*A*p/(p'*A*p);
    p=r+beta*p;
    tol=norm(r)/norm(r0);
    m=m+1;
end
```

Question 1-

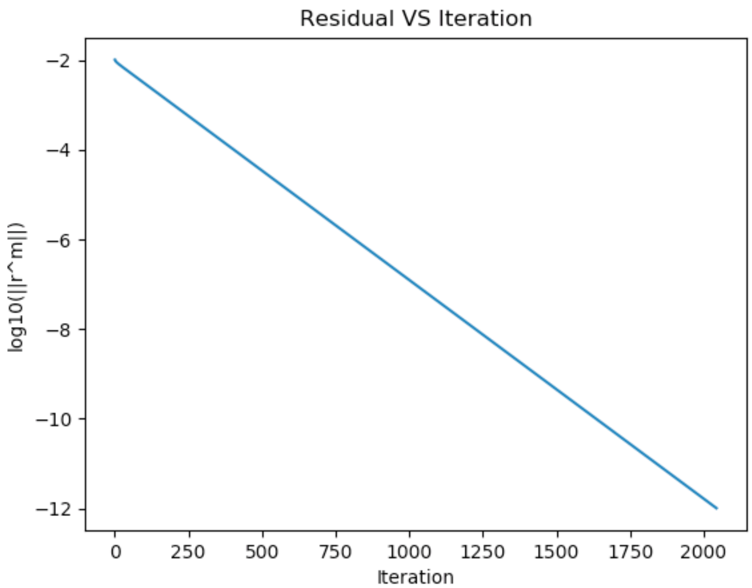
Jacobi

According to question we should not generate A matrix instead we modify algorithm to get rid of explicit A matrix calculation.

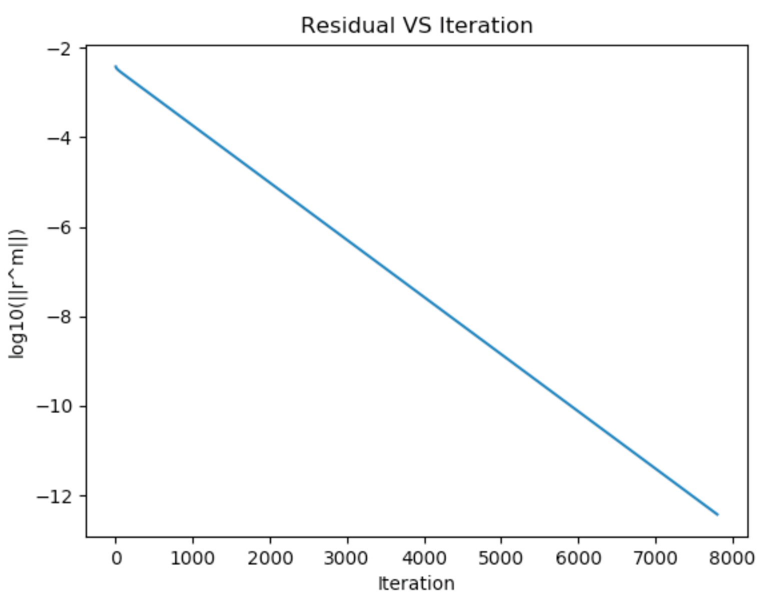
```
def jacobi( b ,R, N , x=None ):

    # Create an initial guess if needed
    if x is None:
      x = zeros(N)

    error =[]
    Iteration=[]

    # Create a vector of the diagonal elements of A
    # and subtract them from A
    #D = diag(A)
    #R = A - diagflat(D)

    x_previous=np.zeros(N)
    k=0 ;

    normfirst= linalg.norm(R)
    norm=0

    for  j in range(100000):

        for m in range(N) :
            x_previous[m]= x[m]

        print(linalg.norm(R) / normfirst)
        norm=linalg.norm(R)

        if (linalg.norm(R) / normfirst) < 10**-10:
            break;

        error.append(np.log10(norm))

        k= k+1
        Iteration.append(k)
        #print(k)
        for i in range(N):
            if i==0:
                x[i] = (b[i] +  x_previous[i+1])/2
            elif  i == N-1 :
                x[i] = (b[i] +  x_previous[i-1])/2
            else :
                x[i] = (b[i]+  x_previous[i-1] + x_previous[i+1])/2

        for i in range(N):
            if i==0:
                R[i] = b[i] + x[i+1] -2*x[i]
            elif  i == N-1:
                R[i] = b[i] + x[i-1] -2*x[i]
            else  :
                R[i] = b[i] + x[i-1] -2*x[i] + x[i+1]



    print (k)
    plt.plot(Iteration,error)
    plt.ylabel('log10(||r^m||)')
    plt.xlabel('Iteration')
    plt.show()

    return x
```

Jacobi Solution For question 1 when N=20 and N=40

Iteration Number for when N is 20 = 2043



Iteration Number for when N is 40 = 7805

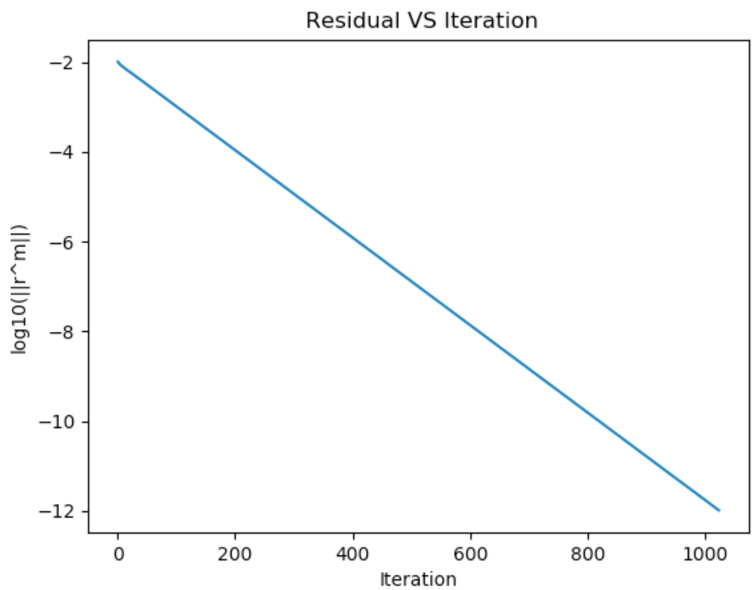


Gauss-Seidel

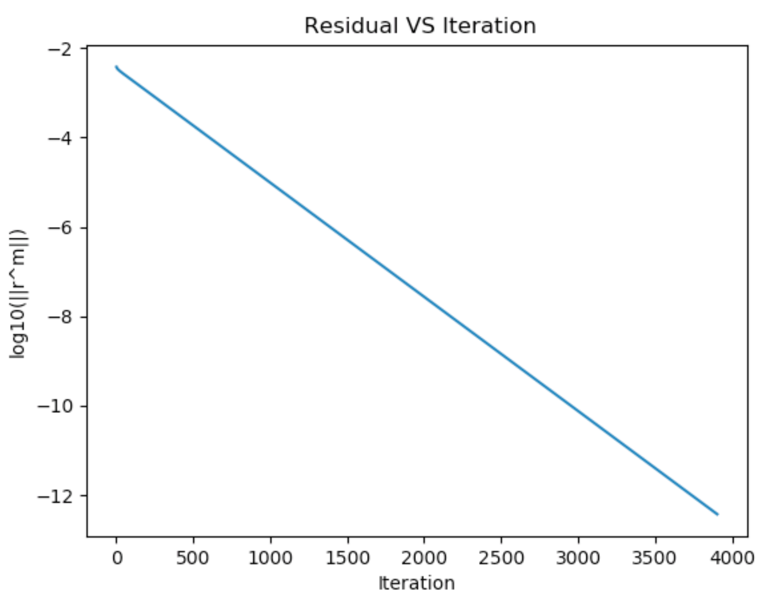
```
def gaussseidel( b ,R, N , x=None ):

   # Create an initial guess if needed
    if x is None:
      x = zeros(N)


    # Create a vector of the diagonal elements of A
    # and subtract them from A
    #D = diag(A)
    #R = A - diagflat(D)

    x_previous=np.zeros(N)
    k=0 ;

    normfirst= linalg.norm(R)

    error =[]
    Iteration=[]

    for  j in range(100000):

        for m in range(N) :
            x_previous[m]= x[m]

        norm=linalg.norm(R)
        print(norm / normfirst)

        if (norm / normfirst) < 10**-10:
            break;

        error.append(np.log10(norm))
        k= k+1
        #print(k)

        Iteration.append(k)

        for i in range(N):
            if i==0:
                x[i] = (b[i] +  x_previous[i+1])/2
            elif  i == N-1 :
                x[i] = (b[i] +  x[i-1])/2
            else :
                x[i] = (b[i]+  x[i-1] + x_previous[i+1])/2

        for i in range(N):
            if i==0:
                R[i] = b[i] + x[i+1] -2*x[i]
            elif  i == N-1:
                R[i] = b[i] + x[i-1] -2*x[i]
            else  :
                R[i] = b[i] + x[i-1] -2*x[i] + x[i+1]




    print (k);
    plt.plot(Iteration,error)
    plt.ylabel('log10(||r^m||)')
    plt.xlabel('Iteration')
    plt.show()
    return x
```

Gauss- Seidel Solution For question 1 when N=20 and N=40

Iteration Number for when N is 20 = 1023



Iteration Number for when N is 40 = 3904

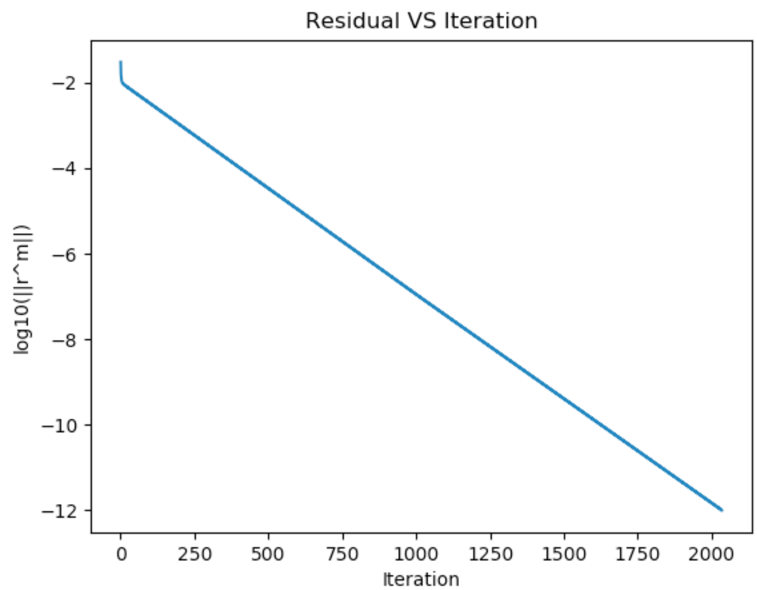


Steepest Descent

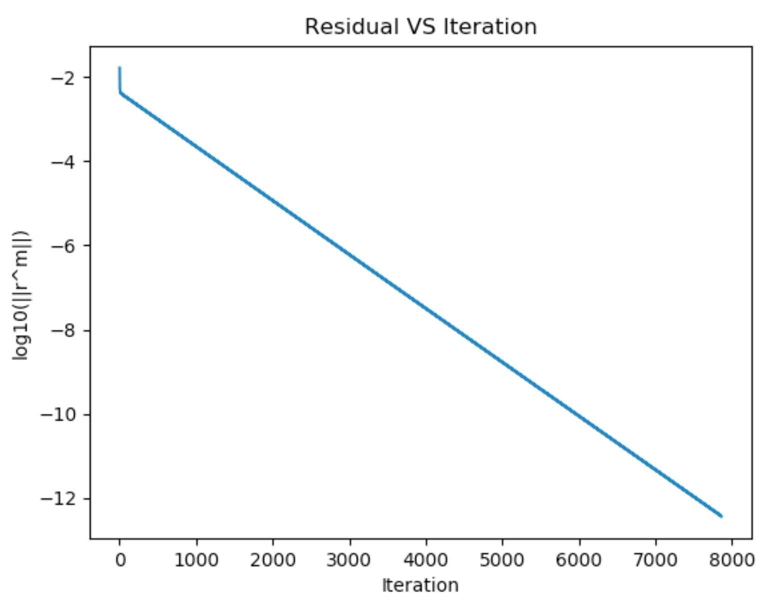
```python
def SD(b, R , N , x_s  )  :

    # Create an initial guess if needed
    if x_s is None:
        x_s = zeros(N)

    error =[]
    Iteration=[]

    x= np.zeros(N)
    m= 0
    tol = 1

    Ar = np.zeros(N)
    normfirst=linalg.norm(R)

    while tol >= 10**-10:

        if m == 0 :
            for i in range(N):
                if i==0:
                    R[i] = b[i] - 2*x[i] + x[i+1]
                elif i==N-1 :
                    R[i] = b[i] - 2*x[i] + x[i-1]
                else:
                    R[i] = b[i] - 2*x[i] + x[i-1] + x[i+1]



        #Calculate Ar
        for i in range(N):
            if i==0:
                Ar[i] =  2*R[i] - R[i+1]
            elif i==N-1:
                Ar[i] =  2*R[i] - R[i-1]
            else:
                Ar[i] =  2*R[i] - R[i-1] - R[i+1]


        alpha = np.dot(np.transpose(R), R) / np.dot(np.transpose(R) , Ar )
        x = x + alpha* R

        # r = b - Ax
        for i in range(N):
            if i==0:
                R[i] = b[i] - 2*x[i] + x[i+1]
            elif i==N-1:
                R[i] = b[i] - 2*x[i] + x[i-1]
            else:
                R[i] = b[i] - 2*x[i] + x[i-1] + x[i+1]

        norm=linalg.norm(R)
        tol = norm / normfirst
        error.append(np.log10(norm))
        print(tol)
        m = m + 1
        Iteration.append(m)
    print (m);
    plt.plot(Iteration,error)
    plt.ylabel('log10(||r^m||)')
    plt.xlabel('Iteration')
    plt.show()
    return x
```

Steepest – Descent Solution For question 1 when N=20 and N=40

Iteration Number for when N is 20 = 2034



Iteration Number for when N is 40 = 7862

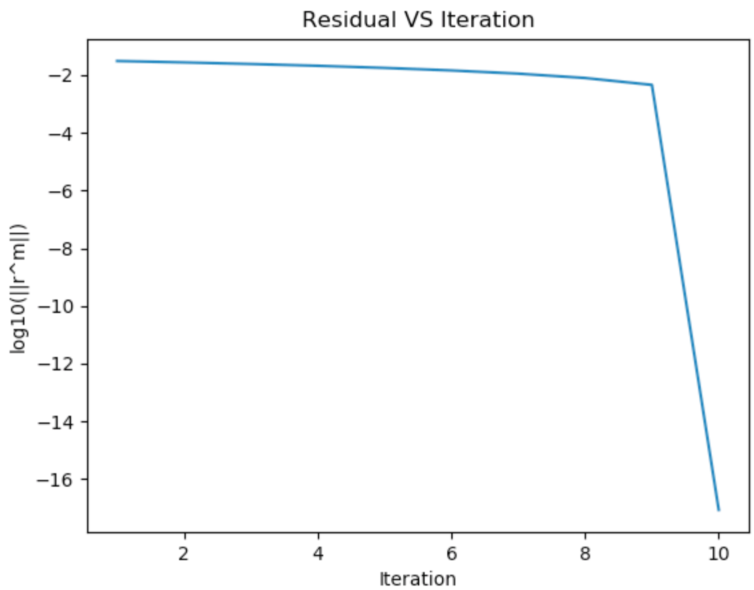


Conjugate Gradient

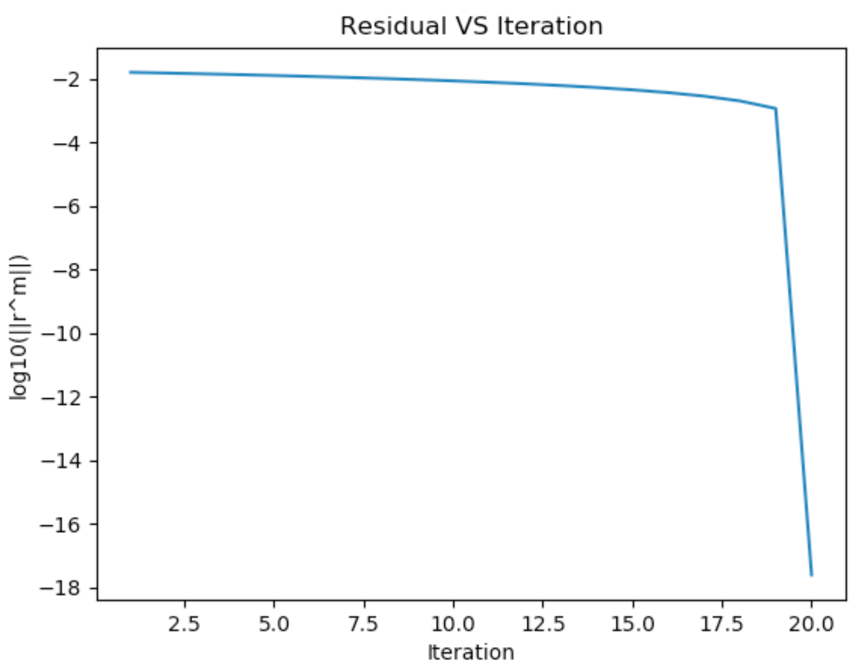
```python
def CG(b, R , N , x  )  :

    x= np.zeros(N)
    m= 0
    tol = 1

    Ap = np.zeros(N)
    p = np.zeros(N)

    error =[]
    Iteration=[]


    while tol >= 10**-10:

        if m == 0 :
            for i in range(N):
                if i==0:
                    R[i] = b[i] - 2*x[i] + x[i+1]
                elif i==N-1 :
                    R[i] = b[i] - 2*x[i] + x[i-1]
                else:
                    R[i] = b[i] - 2*x[i] + x[i-1] + x[i+1]
            for i in range(N):
                p[i] = R[i]
            normfirst=linalg.norm(R)

        #Calculate Ap
        for i in range(N):
            if i==0:
                Ap[i] =  2*p[i] - p[i+1]
            elif i==N-1:
                Ap[i] =  2*p[i] - p[i-1]
            else:
                Ap[i] =  2*p[i] - p[i-1] - p[i+1]


        alpha = np.dot(np.transpose(R), R) / np.dot(np.transpose(p) , Ap )
        x = x + alpha* p

        # r = b - alpha*A*p
        R = R - alpha * Ap
        beta = np.dot(-np.transpose(R),Ap) / np.dot(np.transpose(p), Ap)

        p = R+ beta*p
        norm = linalg.norm(R)
        tol = norm / normfirst
        error.append(np.log10(norm))
        print(tol)
        m = m + 1
        Iteration.append(m)

    print (m);
    plt.plot(Iteration,error)
    plt.title('Residual VS Iteration')
    plt.ylabel('log10(||r^m||)')
    plt.xlabel('Iteration')
    plt.show()
    return x
```

Conjugate Gradient Solution For question 1 when N=20 and N=40

Iteration Number for when N is 20 = 9



Iteration Number for when N is 40 = 19

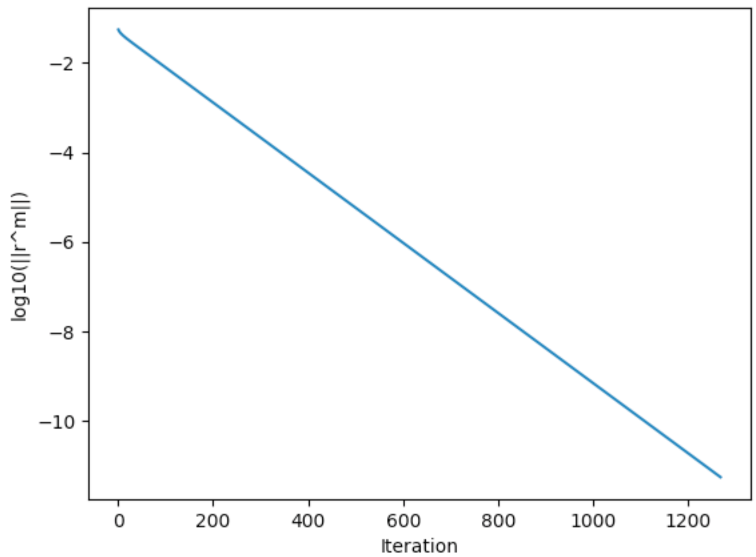


Jacobi For Question -2

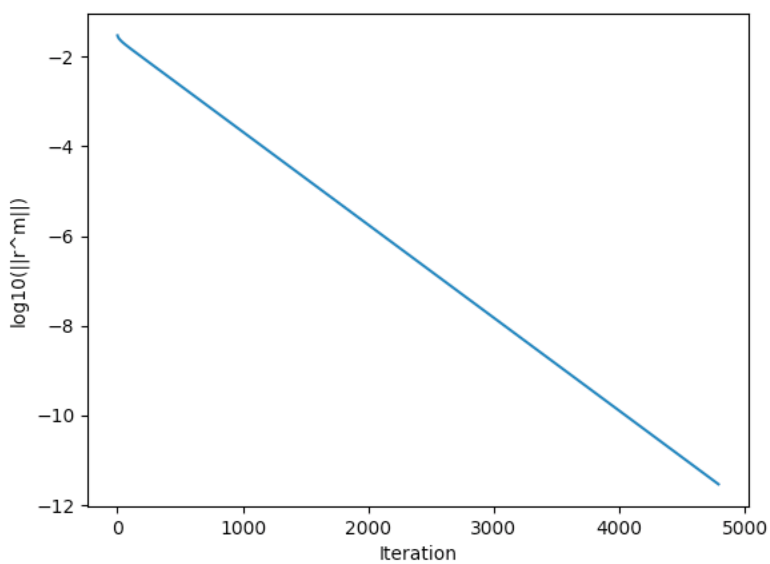
```
def jacobi(b, R , N, x, h) :


    error =[]
    Iteration=[]

    x_previous=np.zeros((N+2,N+2))

    k=0 ;

    normfirst= LA.norm(R, 'fro')
    print(normfirst)


    for  j in range(100000):

        for m in range(N+2) :
            for n in range(N+2):
                x_previous[m][n]= x[m][n]
        norm=linalg.norm(R,'fro')
        print(norm)
        print(LA.norm(R, 'fro') / normfirst)

        if (LA.norm(R, 'fro') / normfirst) < 10**-10:
            break;

        error.append(np.log10(norm))

        k= k+1

        Iteration.append(k)

        print(j)
        for i in range(N+2):
            for j in range(N+2):
                if i==0:
                    x[i][j] = 0
                elif i==N+1:
                    x[i][j] = 0
                elif j==0 :
                    x[i][j] = 0
                elif j==N+1 :
                    x[i][j] = 0
                else :
                    x[i][j]= (x_previous[i-1][j] + x_previous[i][j-1] + x_previous[i][j+1] + x_previous[i+1][j] + h**2)/(4+h**2)

        for i in range(N+2):
            for j in range(N+2):
                if i==0:
                    R[i][j] =b[i][j]
                elif i==N+1:
                    R[i][j] =b[i][j]
                elif j==0:
                    R[i][j] =b[i][j]
                elif j==N+1:
                    R[i][j] =b[i][j]
                else:
                    R[i][j]= h**2 - ((4+h**2)*x[i][j] - x[i-1][j] - x[i][j-1] - x[i+1][j] -x[i][j+1])
```

Jacobi Solution For question 2 when N=16 and N=32

Iteration Number for when N is 16 = 1268



Iteration Number for when N is 32 = 4792

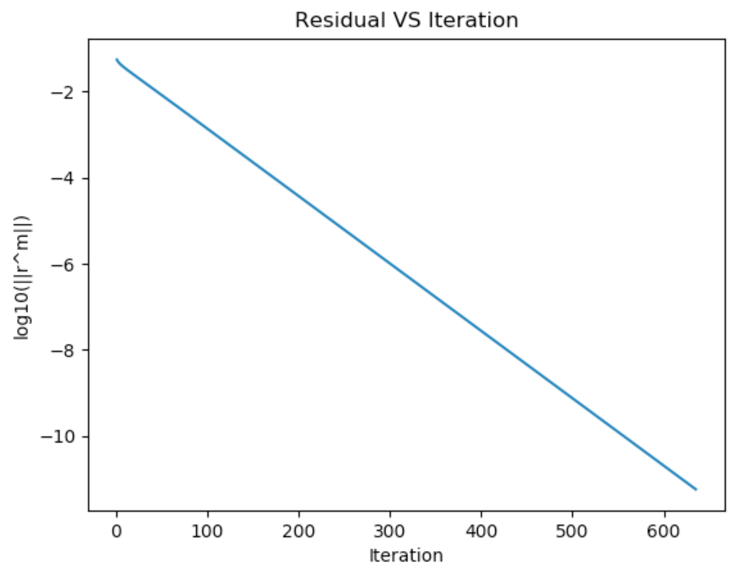


Gauss-Seidel For Question 2

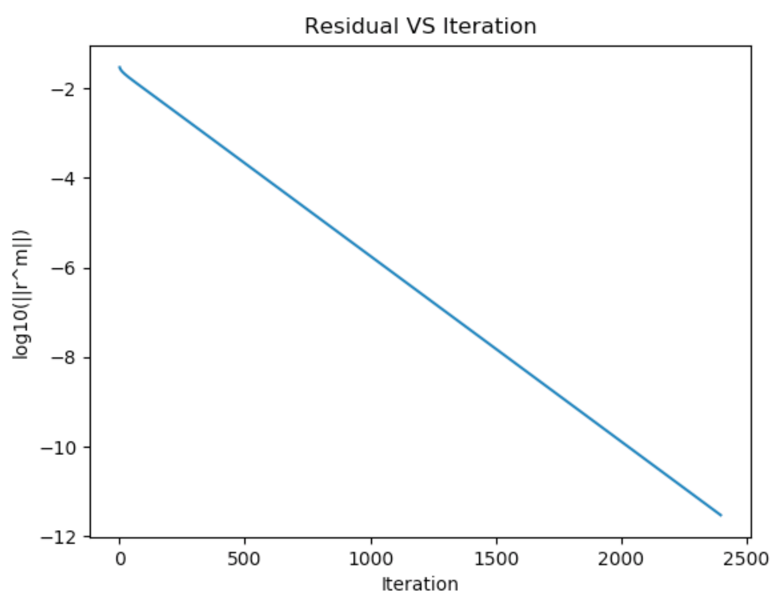
```
def GS(b, R , N, x, h) :


    x_previous=np.zeros((N+2,N+2))

    k=0 ;

    normfirst= LA.norm(R, 'fro')

    error =[]
    Iteration=[]

    for  j in range(100000):

        print(LA.norm(R, 'fro') / LA.norm(b, 'fro'))


        norm=LA.norm(R, 'fro')


        if (LA.norm(R, 'fro') / LA.norm(b, 'fro')) < 10**-10:
            break;

        error.append(np.log10(norm))

        k= k+1

        Iteration.append(k)

        print(j)
        for j in range(N+2):
            for i in range(N+2):
                if i==0:
                    x[i][j] = 0
                elif i==N+1:
                    x[i][j] = 0
                elif j==0 :
                    x[i][j] = 0
                elif j==N+1 :
                    x[i][j] = 0
                else :
                    x[i][j]= (x[i-1][j] + x[i][j-1] + x[i][j+1] + x[i+1][j] + h**2)/(4+h**2)

        for i in range(N+2):
            for j in range(N+2):
                if i==0:
                    R[i][j] =b[i][j]
                elif i==N+1:
                    R[i][j] =b[i][j]
                elif j==0:
                    R[i][j] =b[i][j]
                elif j==N+1:
                    R[i][j] =b[i][j]
                else:
                    R[i][j]= h**2 - ((4+h**2)*x[i][j] - x[i-1][j] - x[i][j-1] - x[i+1][j] -x[i][j+1])
```

Gauss-Seidel Solution For question 2 when N=16 and N=32

Iteration Number for when N is 16 = 635



Iteration Number for when N is 32 = 2397



Steepest Descent for Question 2 (without generating A Matrix explicitly)

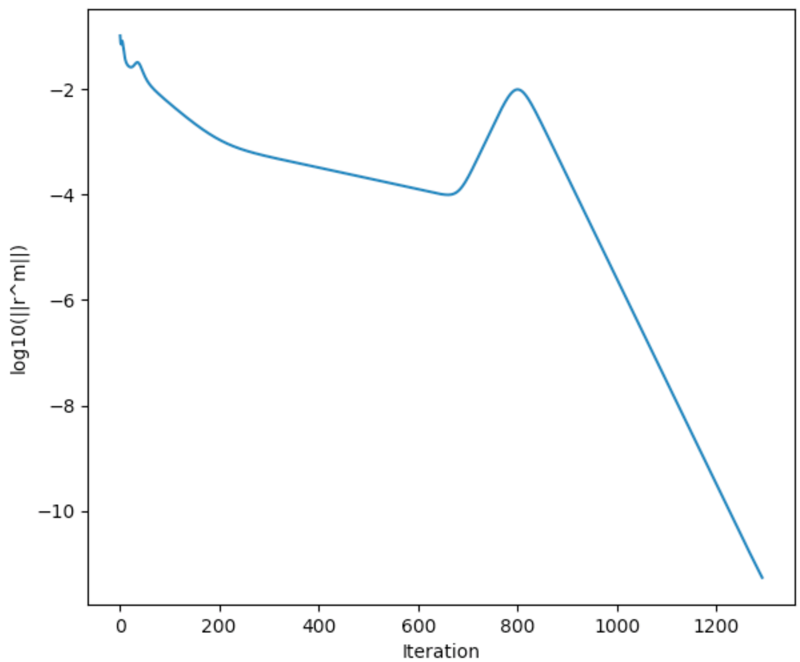
```
def SD(b, R , N , x ,h )  :


    #x= np.zeros((N+2,N+2))
    m= 0
    tol = 1

    Ar = np.zeros((N+2,N+2))

    alpha=0
    part1=0
    part2=0

    error =[]
    Iteration=[]


    while tol >= 10**-10:


        if m == 0:
        #Calculate Ar
        for i in range(N+2):
            for j in range(N+2):
                if i==0:
                    Ar[i][j] = 0
                elif i==N+1:
                    Ar[i][j] = 0
                elif j==0:
                    Ar[i][j] = 0
                elif j==N+1:
                    Ar[i][j] = 0
                else:
                    Ar[i][j] = (4 + h**2)*R[i][j] - R[i-1][j] - R[i][j-1] - R[i+1][j] - R[i][j+1]

        #alpha = np.dot(np.transpose(R), R) / np.dot(np.transpose(R) , Ar )

        for j in range(1,N+1):
            for i in range(1,N+1):
                part1 = part1 + (R[i][j]*R[i][j]) #/(R[i][j]*Ar[i][j])

        for j in range(1,N+1):
            for i in range(1,N+1):
                part2 = part2 + (R[i][j]*Ar[i][j])

        alpha= part1/part2

        #x = x + alpha*R
        for i in range(N+2):
            for j in range(N+2):
                if i==0:
                    x[i][j] = 0
                elif i==N+1:
                    x[i][j] = 0
                elif j==0 :
                    x[i][j] = 0
                elif j==N+1 :
                    x[i][j] = 0
                else:
                    x[i][j] = x[i][j] + alpha* R[i][j]
```

Steepest Descent for Question 2 (without generating A Matrix explicitly)

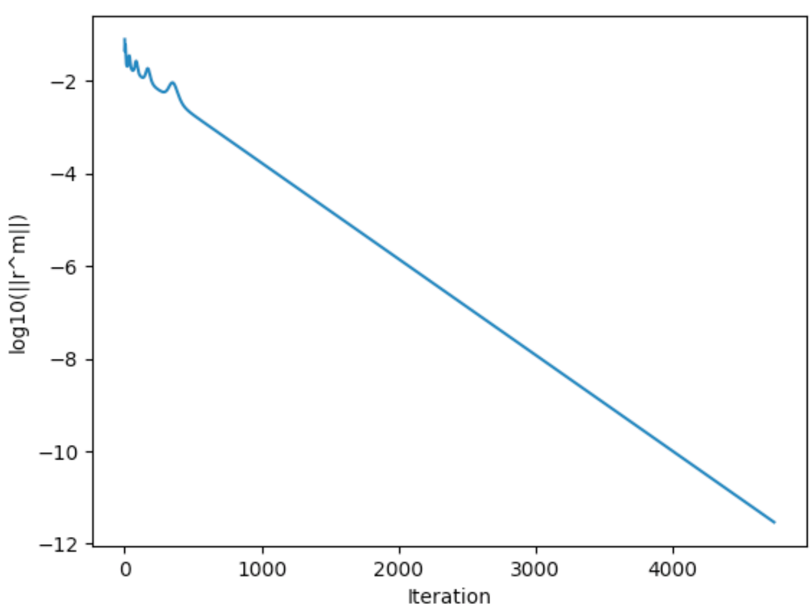
```
while tol >= 10**-10:


    if m == 0:
    #Calculate Ar
    for i in range(N+2):
        for j in range(N+2):
            if i==0:
                Ar[i][j] = 0
            elif i==N+1:
                Ar[i][j] = 0
            elif j==0:
                Ar[i][j] = 0
            elif j==N+1:
                Ar[i][j] = 0
            else:
                Ar[i][j] = (4 + h**2)*R[i][j] - R[i-1][j] - R[i][j-1] - R[i+1][j] - R[i][j+1]

    #alpha = np.dot(np.transpose(R), R) / np.dot(np.transpose(R) , Ar )

    for j in range(1,N+1):
        for i in range(1,N+1):
            part1 = part1 + (R[i][j]*R[i][j]) #/(R[i][j]*Ar[i][j])

    for j in range(1,N+1):
        for i in range(1,N+1):
            part2 = part2 + (R[i][j]*Ar[i][j])

    alpha= part1/part2

    #x = x + alpha*R
    for i in range(N+2):
        for j in range(N+2):
            if i==0:
                x[i][j] = 0
            elif i==N+1:
                x[i][j] = 0
            elif j==0 :
                x[i][j] = 0
            elif j==N+1 :
                x[i][j] = 0
            else:
                x[i][j] = x[i][j] + alpha* R[i][j]

    # r = b - Ax
    for i in range(N+2):
        for j in range(N+2):
            if i==0:
                R[i][j] =b[i][j]
            elif i==N+1:
                R[i][j] =b[i][j]
            elif j==0:
                R[i][j] =b[i][j]
            elif j==N+1:
                R[i][j] =b[i][j]
            else:
                R[i][j]= h**2 - ((4+h**2)*x[i][j] - x[i-1][j] - x[i][j-1] - x[i+1][j] -x[i][j+1])




    print (m)
    norm=linalg.norm(R, 'fro')
    tol = norm / normfirst
```

Steepest Descent Solution For question 2 when N=16 and N=32

Iteration Number for when N is 16 = 1292



Iteration Number for when N is 32 = 4741



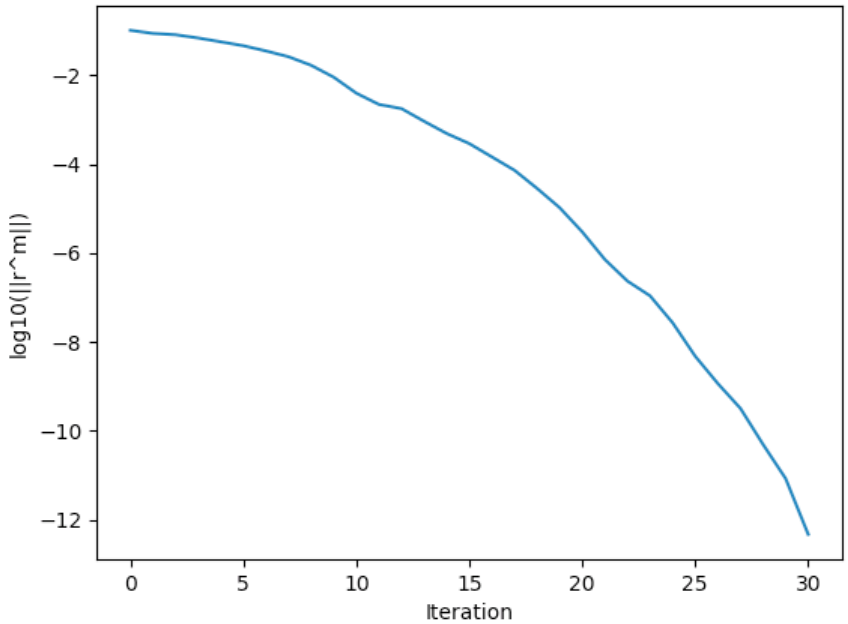
Conjugate Gradient for Question 2 (without generating A Matrix explicitly)

```
while tol >= 10**-10:

    if m == 0 :
        for i in range(N+2):
            for j in range(N+2):
                if i==0:
                    R[i][j]=R[i][j]
                elif i==N+1:
                    R[i][j]=R[i][j]
                elif j==0:
                    R[i][j]=R[i][j]
                elif j==N+1:
                    R[i][j]=R[i][j]
                else:
                    R[i][j]= R[i][j]-((4 + h**2)*x[i][j] - x[i-1][j] - x[i][j-1] - x[i+1][j] - x[i][j+1])

        #for i in range(1,N):
        #    for j in range(1,N):
        #        R_temp[i-1][j-1]=R[i][j]


        normfirst=linalg.norm(R,'fro')

        for i in range(N+2):
            for j in range(N+2):
                    p[i][j] = R[i][j]

        for j in range(0,N+2):
            for i in range(1,N+2):
                partv = partv + (R[i][j]*R[i][j])
        v=partv
        partv=0
    else:

        #Calculate q= Ap
        for i in range(N+2):
            for j in range(N+2):
                if i==0:
                    q[i][j] = 0
                elif i==N+1:
                    q[i][j] = 0
                elif j==0:
                    q[i][j] = 0
                elif j==N+1:
                    q[i][j] = 0
                else:
                    q[i][j] = (4 + h**2)*p[i][j] - p[i-1][j] - p[i][j-1] - p[i+1][j] - p[i][j+1]
```

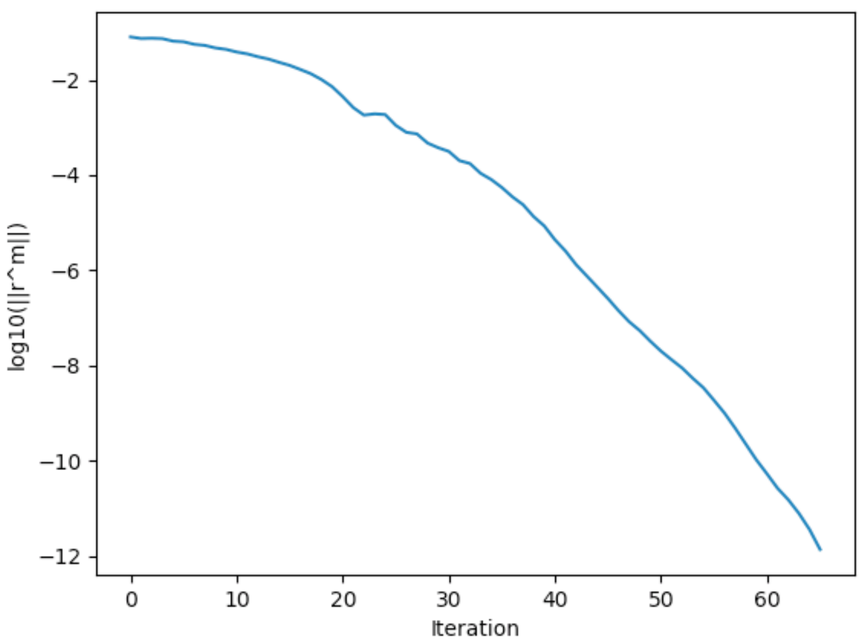
```python
#Calculate Mu= p^T q
for j in range(0,N+2):
    for i in range(0,N+2):
        partMu = partMu + (p[i][j]*q[i][j])

Mu=partMu
partMu=0
alpha= v/Mu



#x = x + alpha* p
for i in range(N+2):
    for j in range(N+2):
            x[i][j] = x[i][j] + alpha* p[i][j]




#R = R - alpha * Ap
for i in range(N+2):
    for j in range(N+2):
            R[i][j] = R[i][j] -alpha*q[i][j]




#v_plus= r^Tr
for j in range(1,N+1):
    for i in range(1,N+1):
        partvp = partvp + (R[i][j]*R[i][j])



v_plus=partvp
partvp=0
beta=v_plus / v

#p = R+ beta*p

for i in range(N+2):
    for j in range(N+2):
        if i==0:
            p[i][j] = 0
        elif i==N+1:
            p[i][j] = 0
        elif j==0:
            p[i][j] = 0
        elif j==0:
            p[i][j] = 0
        else:
            p[i][j] = R[i][j] + beta*p[i][j]

#v=v_plus
v=v_plus

#for i in range(1,N):
#    for j in range(1,N):
#        R_temp[i-1][j-1]=R[i][j]
norm=linalg.norm(R,'fro')
error.append(np.log10(norm))
tol = norm/ normfirst
print(tol)
```

Conjugate Gradient Solution For question 2 when N=16 and N=32

Iteration Number for when N is 16 = 31



Iteration Number for when N is 32 = 66



Conclusion

Results for Question 1

	Jacobi	GS	SD	CG
N=20	2043	1023	2034	9
N=40	7805	3904	7862	19

According to this table above , it can easily be observed that best method is Conjugate Gradient since it take smallest iteration number. Gauss-Seidel is approximately half of Jacobi method. Interestingly steepest descent is almost same with Jacobi .

Result for Question 2

	Jacobi	GS	SD	CG
N=16	1268	635	1292	31
N=32	4792	2397	4741	66

According to second table above, it can also observed that CG is best in terms of iteration rate. When we increase N , in all method iteration number also is increased.