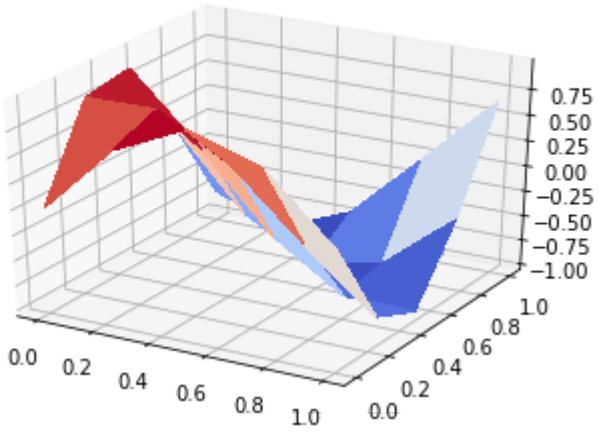
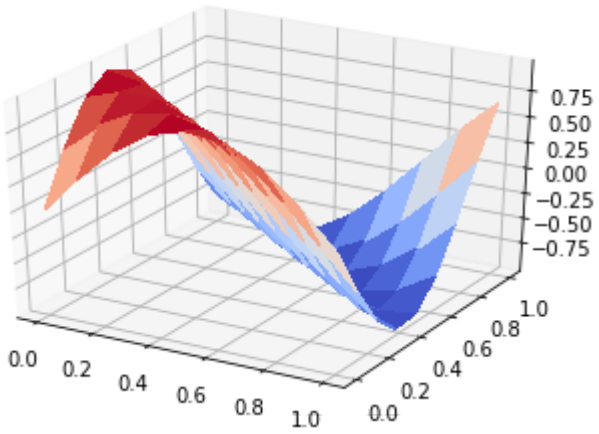
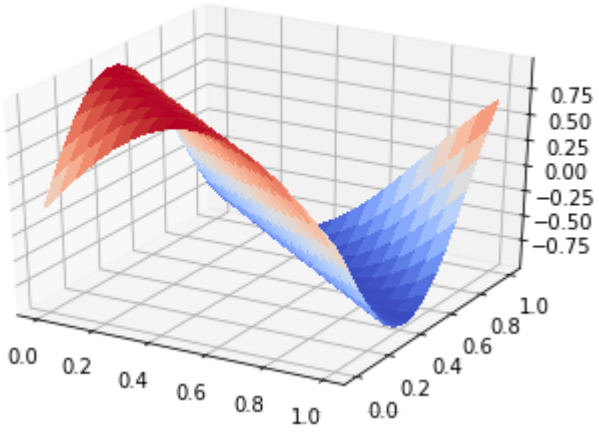
```
import  numpy  as  np
from  scipy.sparse  import  coo_matrix,  csr_matrix

def  g(x,  y):
    if  x==0.:
        g  =  np.sin(4*y)
    elif  y==0.:
        g  =  np.sin(3*x)
    elif  x==1.:
        g  =  np.sin(3+4*y)
    elif  y==1.:
        g  =  np.sin(3*x+4)
    else:
        g  =  0
    return  g


def  wave(N):
    h  =  1/N
    k  =  5
    val1  =  (24-4*(h**2)*(k**2))/9
    val2  =  (-3-(h**2)*(k**2))/9
    val3  =  (-12-(h**2)*(k**2))/36
    A_den  =  np.zeros(((N+1)**2,(N+1)**2))
    b  =  np.zeros((N-1)**2)
    index_P  =  np.array(range(N+1,  (N**2)+N))
    index_delete1  =  (N+1)*np.array(range(N-1))
    index_delete2  =  (N+1)*np.array(range(N-1))+N
    index_delete  =  np.append(index_delete1,index_delete2)
    index  =  np.delete(index_P,  index_delete)
    counter  =  0
    for  i  in  index:
        x,  y  =    (i%(N+1))*h,  (i//(N+1))*h
        A_den[i,  i]  =  val1
        A_den[i,  i+N+2],  A_den[i,  i+N]  =  val3,  val3
        A_den[i,  i-N-2],  A_den[i,  i-N]  =  val3,  val3
        A_den[i,  i+N+1],  A_den[i,  i+1]  =  val2,  val2
        A_den[i,  i-1],  A_den[i,  i-N-1]  =  val2,  val2
        b[counter]  =  -val3*(g(x-h,y+h)+g(x-h,y-h)+g(x+h,y+h)+g(x+h,y-h)
        )-val2*(g(x,y+h)+g(x,y-h)+g(x-h,y)+g(x+h,y))
        counter  +=  1

    A  =  A_den[index][:,  index]

    return  csr_matrix(A),  b
```

```
from  matplotlib  import  pyplot  as  plt
from  mpl_toolkits.mplot3d  import  Axes3D
from  matplotlib  import  cm
from  scipy.sparse.linalg  import  spsolve


def  part2(N):
    h  =  1/N
    fig  =  plt.figure()
    A,  b  =  wave(N)
```

```
    x_estm = spsolve(A, b)
    u_1 = x_estm.reshape((N-1, N-1))
    ran = np.linspace(0, 1, N+1)
    col1, col2 = np.zeros(N+1),np.zeros(N+1)
    row1, row2 = np.zeros(N+1),np.zeros(N+1)
    for i in range(N+1):
        col1[i], col2[i] = g(0,i*h), g(1,i*h)
        row1[i], row2[i] = g(i*h,0), g(i*h,1)
    u_2 = np.c_[np.matrix(col1[1:-1]).T, np.c_[u_1,np.matrix(col2[1:-1]).T]]
    u_h = np.r_[np.matrix(row1), np.r_[u_2,np.matrix(row2)]]
    x, y = np.meshgrid(ran, ran)
    ax = fig.add_subplot(projection='3d')
    surf = ax.plot_surface(x, y, u_h, antialiased=False, cmap=cm.coolwarm)
    plt.show()

part2(4)
part2(8)
part2(16)
```

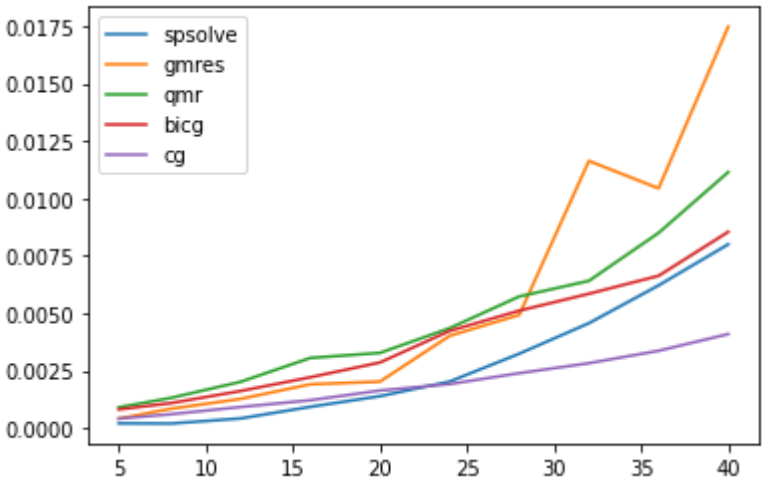






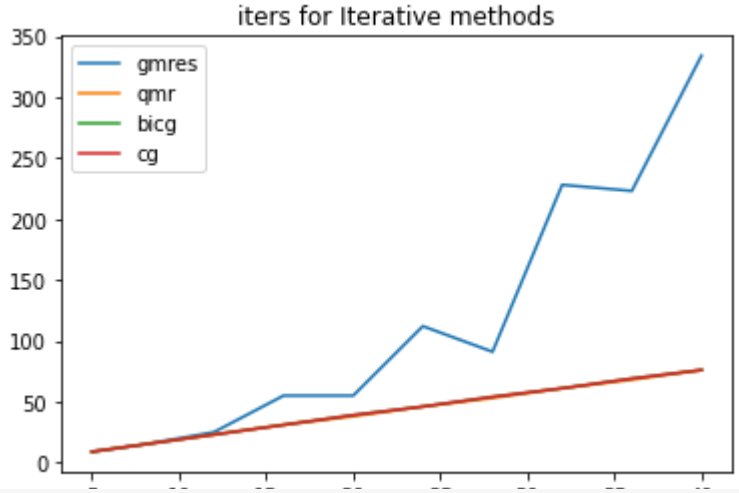

```
from scipy.sparse.linalg import spsolve, gmres, qmr, bicg, cg
from timeit import timeit
```

```
def time_cost(method, N):
    A, b = wave(N)
    t = timeit(lambda: method(A, b),number=10)/10
    return t

N = np.linspace(5,40,10,dtype=int)
method_all = [spsolve, gmres, qmr, bicg, cg]
T_m = np.zeros((5,10))
for i in range(5):
    for j in range(10):
        T_m[i,j] = time_cost(method_all[i], N[j])

plt.figure()
plt.plot(N, T_m[0], label='spsolve')
plt.plot(N, T_m[1], label='gmres')
plt.plot(N, T_m[2], label='qmr')
plt.plot(N, T_m[3], label='bicg')
plt.plot(N, T_m[4], label='cg')
plt.legend()
plt.show()
```




```
def iterate(method, N):
    A, b = wave(N)
    resi = []
    count = lambda res: resi.append(res)
    method(A, b, callback=count)
    return len(resi)

N = np.linspace(5,40,10,dtype=int)
method_iter = [gmres, qmr, bicg, cg]
I_m = np.zeros((4,10))
for i in range(4):
    for j in range(10):
        I_m[i,j] = iterate(method_iter[i], N[j])

plt.figure()
plt.plot(N, I_m[0], label='gmres')
plt.plot(N, I_m[1], label='qmr')
plt.plot(N, I_m[2], label='bicg')
plt.plot(N, I_m[3], label='cg')
plt.legend()
```

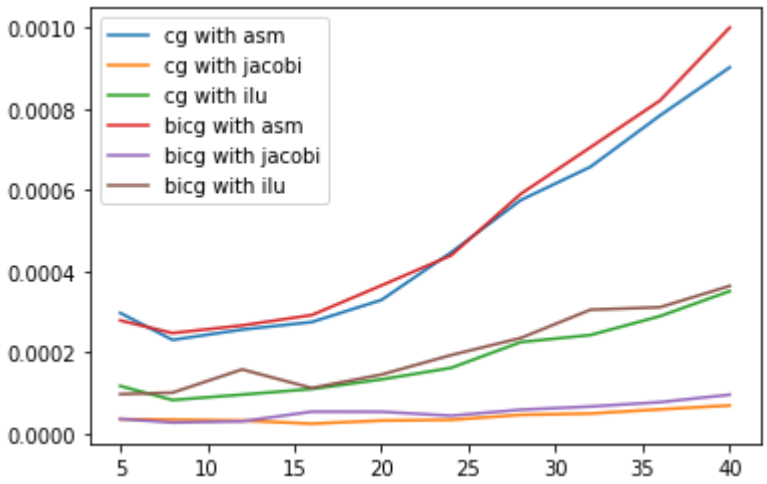
```
plt.title('iters for Iterative methods')
plt.show()
```




```
!pip install petsc4py pyamg
```

```
from scipy.sparse.linalg import spsolve, gmres, qmr, bicg, cg
import time
from petsc4py import PETSc

def time_precon(method, N, precon):
    A1, b = wave(N)
    A = PETSc.Mat()
    A.createAIJ(size=A1.shape,csr=(A1.indptr,A1.indices,A1.data))
    ksp = PETSc.KSP().create()
    ksp.setOperators(A)
    b = A.createVecLeft()
    b.array[:] = b
    x1 = A.createVecRight()
    ksp.setType(method)
    ksp.setConvergenceHistory()
    ksp.getPC().setType(precon)
    b[:] = b
    start = time.time()
    ksp.solve(b,x1)
    end = time.time()
    t = end - start
    return t

N = np.linspace(5,40,10,dtype=int)
precon = ['asm', 'jacobi', 'ilu']

cg_t, bicg_t = np.zeros((3,10)), np.zeros((3,10))

for i in range(len(precon)):
    for j in range(len(N)):
        cg_t[i,j] = time_precon('cg', N[j], precon[i])
        bicg_t[i,j] = time_precon('bicg', N[j], precon[i])

plt.figure()
plt.plot(N, cg_t[0], label='cg with asm')
plt.plot(N, cg_t[1], label='cg with jacobi')
plt.plot(N, cg_t[2], label='cg with ilu')
plt.plot(N, bicg_t[0], label='bicg with asm')
plt.plot(N, bicg_t[1], label='bicg with jacobi')
plt.plot(N, bicg_t[2], label='bicg with ilu')
```

```
plt.legend()
plt.show()
```

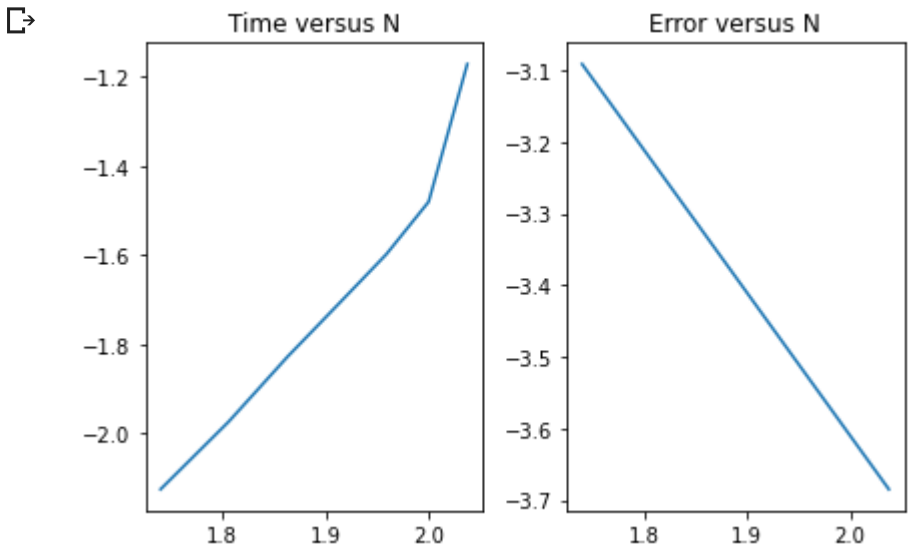


Answer: cg has the best performace with preconditioner jacobi. This is because it cost the least time with small and medium sized value of N. And it also needs the least number of iterations to get the estimated solution.

```
import time
from scipy.sparse.linalg import cg

def time_error(N):
    # measure the time
    h = 1/N
    A, b = wave(N)
    start1 = time.time()
    x_estm, _ = cg(A, b)
    end1 = time.time()
    t = end1 - start1
    # compute the error
    u_1 = x_estm.reshape((N-1, N-1))
    ran = np.linspace(0, 1, N+1)
    col1, col2 = np.zeros(N+1),np.zeros(N+1)
    row1, row2 = np.zeros(N+1),np.zeros(N+1)
    for i in range(N+1):
        col1[i], col2[i] = g(0,i*h), g(1,i*h)
        row1[i], row2[i] = g(i*h,0), g(i*h,1)
    u_2 = np.c_[np.matrix(col1[1:-1]).T, np.c_[u_1,np.matrix(col2[1:-1]).T]]
    u_h = np.r_[np.matrix(row1), np.r_[u_2,np.matrix(row2)]]
    error = 0
    for i in range(N):
        for j in range(N):
            x, y = (h/2)+i*h, (h/2)+j*h
            u_e = np.sin((3*x)+(4*y))
            u_p = (u_h[j,i]+u_h[j+1,i]+u_h[j,i+1]+u_h[j+1,i+1])/4
            error += (h**2)*abs(u_e-u_p)

    return t, error
```

```
N = np.linspace(55,109,7,dtype=int)

T, err = np.zeros(7), np.zeros(7)
for i in range(7):
```

```
    n  =  N[i]
    T[i],  err[i]  =  time_error(n)


fig,  ax  =  plt.subplots(1,2)
ax[0].plot(np.log10(N),  np.log10(T))
ax[1].plot(np.log10(N),  np.log10(err))
ax[0].set_title("Time  versus  N")
ax[1].set_title("Error  versus  N")
fig.tight_layout()
plt.show()
```



Answer: From the plot, we estimate the time complexity of cg is O(N^2) and the order of convergence is 2 where error decreases like O(N^-2). The reason is that in the plot for log10(t) versus log10(N), when N increased from 60 to 100, the slope is nearly 2 where log10(t) = 2*log10(N), t = O(N^2). Thus time complexity is O(N^2). Similarly, when log10(N) increases from 1.8 to 2.0, the log10(error) reduces with slope -2, which means the order of convergence is also 2.

Answer: We could use the multiprocessing module to parallelise our method. For example, because cg is an iterative solver, we normally do the iterations forward, however, we can do the forward and backward iterations at the same time if we can find some midpoints. Therefore, doing two kinds of iterations at the same time will be trivial to parallelise, but finding such a midpoint would be hard. In this situation, this method could be doubled faster.

Colab 付费产品 - 在此处取消合同

✓ 1 秒 完成时间：14:30