Problem 1d

The eigenvalues of A are given by

$\lambda_k = 4/h^2 * sin^2(k * \pi / (2m))$ $for\ k = 1, 2, 3 .... m - 1$

To get an eigenvalue of 0, we need to substitute k = 0 or m. But this corresponds to the boundary condition, which is already known (u0 = um = 0).

Further, if 0 were an eigenvalue for A, then A would not be invertible, which is not the case for A => contradiction! Hence, 0 cannot be an eigenvalue for A.

Problem 1e

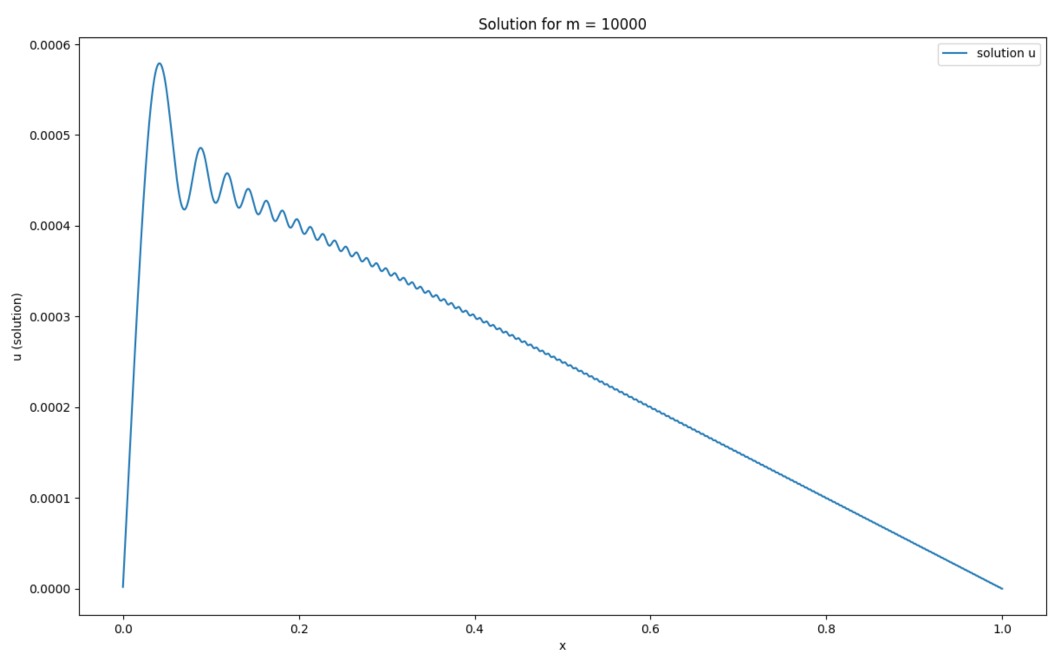
The residual norms for the different values of h are:

h = 0.001, res = 2.5870416919815398e-12
h = 0.0005, res = 1.2190581877291606e-11
h = 0.00025, res = 4.2633674368630636e-11

The convergence rate = 4.0006112759774055 (~= 4)

Problem 1f

f = sin(10000*x*x)
m = 10000
Time taken for solve = 4.859262943267822 s
The plot of the solution is:



Problem 1g

Thomas algorithm

Thomas algorithm is a simplified form of Gaussian elimination that can be used to solve tridiagonal systems of equations. For a tridiagonal system with n equations, the solution can be obtained in O(n) time using the Thomas algorithm as opposed to the O(n^3) time complexity required by Gaussian elimination. In this algorithm, the values below the diagonal are first eliminated, followed by a back substitution using the resulting upper triangular matrix to solve the system.

Problem 1h

f = sin(1000*x*x)

m = 100000

Residual norm for sparse solve = 2.718582026162153e-09

Time taken for sparse solve = 0.04828000068664551 s

The 1D laplace matrix A has eigenvectors where each element is a sin function. And the eigenvalues are multiples of the sin^2 function.

$$\lambda_k = 4/h^2 * sin^2(k * \pi / (2m)) \; for \; k = 1, 2, 3 .... m$$

When m increases, the m eigenvalues and eigenvectors are not estimated with high accuracy for the higher modes.

In this problem, since f is a sin function. So solving Au=f can be interpreted as estimating the eigenvalues of A for the eigenvector f. But since the value of m is very high, the frequency of f is very high. So estimating eigenvalues for higher modes (ie, higher values of k) is not very accurate. So, since the accuracy suffers, the value of R is not very informative with large values in this case.

Code for Problem1:

```
import numpy as np
from scipy.sparse import csc_matrix, csr_matrix, diags
from scipy.sparse.linalg import spsolve
import time
import matplotlib.pyplot as plt

def construct_1D_laplace(m):
   h = 1./m
   diagonal = 2. * np.ones(m-1)
   off_diag = -1. * np.ones(m-2)
   return (1./(h*h)) * (np.diag(diagonal, 0) + np.diag(off_diag, 1) +
```

```
np.diag(off_diag, -1))

def sample_points_1D(m):
   h = 1./m
   x = np.linspace(h, 1-h, num=m-1)
   return x, np.zeros_like(x)

def get_rhs(x, const):
   return np.sin(const*x*x)

def compute_residual_norm(A, u, f):
   return np.linalg.norm(f - np.dot(A, u), np.inf)

def direct_solve(A, f):
   return np.linalg.solve(A, f)

def build_sparse_A(m):
   h = 1./m
   A_sparse = csc_matrix(diags([-1, 2, -1], [-1, 0, 1], shape=(m-1, m-1)))/(h**2)
   return A_sparse

def sparse_solve(A_sparse, f):
   return spsolve(A_sparse,f)

def verify_eigen_values(A):
   n = A.shape[0]
   h = 1./(n+1)
   v_num, _ = np.linalg.eig(A)
   v_num = np.sort(v_num)
   k = np.arange(1, n+1)
   v_an = (4/(h*h)) * np.sin((k*np.pi)/(2*n+2)) * np.sin((k*np.pi)/(2*n+2))
   print("Inf Norm of diff in eigenvalues for m = {} is {}".format(n+1,
np.linalg.norm(v_an-v_num, np.inf)))

m = 8
A = construct_1D_laplace(m)
verify_eigen_values(A)

ms = [1000, 2000, 4000]
rn = []
for m in ms:
```

```
    const = 100
    A = construct_1D_laplace(m)
    x, u = sample_points_1D(m)
    f = get_rhs(x, const)
    u_ds = direct_solve(A, f)
    rn.append(np.linalg.norm(u_ds, np.inf))
    print("\t h = {}, res = {}".format( 1./m, compute_residual_norm(A, u_ds, f)))

print("Convergence rate = {}".format((rn[0]-rn[1])/(rn[1]-rn[2])))

m = 10000
const = 1000
A = construct_1D_laplace(m)
x, u = sample_points_1D(m)
f = get_rhs(x, const)
start = time.time()
u_ds = direct_solve(A, f)
end = time.time() - start
print("Time taken for solve = {}".format(end))

plt.plot(x, u_ds, label='solution u')
plt.title("Solution for m = {}".format(m))
plt.ylabel("u (solution)")
plt.xlabel("x")
plt.legend()
plt.show()

m = 100000
const = 1000

A_sparse = build_sparse_A(m)
x, u = sample_points_1D(m)
f = get_rhs(x, const)

start = time.time()
u_ss = sparse_solve(A_sparse, f)
end = time.time() - start
print("Residual norm for sparse solve = {} for m =
{}".format(np.linalg.norm(f-A_sparse.dot(u_ss), np.inf), m))
print("Time taken for sparse solve = {} for m = {}".format(end, m))
```