Problem 1. *Use Jacobi, Gauss-Seidel, and SOR (with optimal ω) to solve*

$$\Delta u = -\exp\left(-(x-0.25)^2 - (y-0.6)^2\right)$$

on the unit square $(0,1)\times(0,1)$ with homogeneous Dirichlet boundary conditions. Find the solution for mesh spacings of $h = 2^{-5}, 2^{-6}$, and 2^{-7}. What tolerance did you use? What stopping criteria did you use? What value of ω did you use? Report the number of iterations it took to reach convergence for each method for each mesh.

I ran Jacobi, GS-Lex, and SOR (with GS-Lex) iterative methods to approximate the solutions. For the SOR method, I used an optimal ω of

$$\omega = \frac{2}{1+\sin(\pi h)}.$$

I used the difference between successive iterates as a proxy for error since

$$\|e^k\| = \|A^{-1}r^k\| = \|A^{-1}B^{-1}(u^{k+1}-u^k\| \leq \|u^{k+1}-u^k\|.$$

Using relative tolerance with a tolerance of h^2, I computed

$$\|u^{k+1}-u^k\| \leq h^2\|u^k\|.$$

The number of iterations to converge to the above tolerance are given in the following table for Jacobi, GS, and SOR methods.

Grid spacing h	Jacobi	GS	SOR
2^{-5}	388	257	45
2^{-6}	1549	1027	95
2^{-7}	6196	4104	205

Problem 2. *When solving parabolic equations numerically, one frequently needs to solve an equation of the form*

$$u - \delta\Delta u = f,$$

where $\delta > 0$. The analysis and numerical methods we have discussed for the Poisson equation can be applied to the above equation. Suppose we are solving the above equation on the unit square with Dirichlet boundary conditions. Use the standard five point stencil for the discrete Laplacian.

(a) Analytically compute the eigenvalues of the Jacobi iteration matrix, and show that the Jacobi iteration converges.

Using the standard 5-pt stencil for the Laplacian,

$$A = \frac{1}{h^2}\begin{pmatrix} -4 & 1 & & & 1 & & & & \\ 1 & -4 & 1 & & & 1 & & & \\ & \ddots & \ddots & \ddots & & & \ddots & & \\ & 1 & & 1 & -4 & 1 & & 1 & \\ & & \ddots & & \ddots & \ddots & \ddots & & \\ & & & 1 & & 1 & -4 & 1 & \\ & & & & & & 1 & -4 & \end{pmatrix}.$$

We have

$$\begin{aligned} u - \delta \Delta u &= f \\ (I - \delta A)u &= f \end{aligned}$$

Letting

$$I - \delta A = D - L - U,$$

we have that

$$L + U = D - I + \delta A.$$

Then the problem becomes

$$\begin{aligned} (I - \delta A)u &= f \\ (D - L - U)u &= f \\ Du &= (L + U)u + f \\ Du &= (D - I + \delta A)u + f \\ u &= (I - D^{-1}(I - \delta A))u + D^{-1}f \end{aligned}$$

Since the eigenvalues of A are

$$\lambda^{\ell,m} = \frac{2}{h^2}(\cos(\ell \pi h) + \cos(m \pi h) - 2),$$

and since

$$D = \begin{pmatrix} 1 + \dfrac{4\delta}{h^2} & 0 & \dots & 0 \\ 0 & 1 + \dfrac{4\delta}{h^2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 + \dfrac{4\delta}{h^2} \end{pmatrix},$$

the eigenvalues of $I - D^{-1}(I - \delta A)$ are

$$\begin{aligned} \mu^{\ell,m} &= 1 - \frac{1}{1 + \dfrac{4\delta}{h^2}}\big(1 - \delta \lambda^{\ell,j}\big) \\ &= 1 - \frac{1}{1 + \dfrac{4\delta}{h^2}}\left(1 - \frac{2\delta}{h^2}(\cos(\ell \pi h) + \cos(m \pi h) - 2)\right) \\ &= \frac{2\delta}{h^2 + 4\delta}(\cos(\ell \pi h) + \cos(m \pi h)) \end{aligned}$$

Thus

$$\rho\big(I - D^{-1}(I - \delta A)\big) = \frac{4\delta}{h^2 + 4\delta} < 1.$$

So the Jacobi iteration converges.

(b) If $h = 10^{-2}$ and $\delta = 10^{-4}$, how many iterations of SOR would it take to reduce the error by a factor of 10^{-6}? How many iterations would it take for the Poisson equation? Use that the spectral radius of SOR is

$$\rho_{SOR} = \omega_{opt} - 1,$$

where

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \rho_J^2}},$$

and where ρ_J is the spectral radius of Jacobi.

If $h = 10^{-2}$ and $\delta = 10^{-4}$, then $\rho_J = 4/5$. Then

$$\rho_{SOR} = \frac{2}{1 + \sqrt{1 - (4/5)^2}} - 1 = \frac{2}{1 + \sqrt{9/25}} - 1 = \frac{2}{8/5} - 1 = 5/4 - 1 = 1/4.$$

So every iteration of SOR reduces the error by a factor of $1/4$, so to reduce it by 10^{-6} it will take approximately 10 iterations. This is relatively fast, and it is due to the small spectral radius.

For the standard Poisson equation, we showed in lecture that

$$\rho_{SOR} = 1 - 2\pi h$$

Then if $h = 10^{-2}$, $\rho_{SOR} = 1 - \dfrac{\pi}{50} \approx .937168$, which is close to 1 (so we expect the convergence to take a long time). Indeed, to reduce the error by a factor of 10^{-6}, it would take 213 iterations. This is much longer than the previous case because the spectral radius is so close to 1.

Problem 3. *In this problem we compare the speed of SOR to a direct solve using Gaussian elimination. At the end of this assignment is MATLAB code to form the matrix for the 2D discrete Laplacian. The code for the 3D matrix is similar. Note that with 1 GB of memory, you can handle grids up to about* 1000×1000 *in 2D and* $40 \times 40 \times 40$ *in 3D with a direct solve. The range of grids you will explore depends on the amount of memory you have.*

(a) Solve the PDE from problem 1 using a direct solve. Put timing commands in your code and report the time to solve for a range of mesh spacings. Use SOR to solve on the same meshes and report the time and number of iterations. Comment on your results. Note that the timing results depend strongly on your implementation. Comment on the efficiency of your program.

Adapting my code from homework 2 question 1, I created a direct solver with timing and solved the PDE from problem 1.

I solved $\Delta u = f$ in 2D with dirichlet BC's using a direct solver adapted from part a. I used the standard 5-point Laplacian discretization, which has stencil

$$\frac{1}{h^2}[1 \ldots 1 \ -4 \ 1 \ldots 1]$$

where the outer entries are n away from the center.

Then I went back and added timing commands to the code from problem 1 to time the SOR method. The comparison of the solve times are given in the following table, where the times are in seconds.

Grid spacing h	Direct-Solve Time	SOR time	SOR iterations
2^{-5}	0.00255703926086	.620096921921	45
2^{-6}	0.00599694252014	.06566691399	95
2^{-7}	0.0394101142883	3.7182741165	205

Note that the Direct Solve is much faster than SOR for these small grid sizes and few dimensions.

(b) Repeat the previous part in three spatial dimensions for a range of mesh spacings. Change the right side of the equation to be a three dimensional Gaussian. Comment on your results.

Using a RHS of

$$f(x, y, z) = -exp(-(x - 0.25)^2 - (y - 0.6)^2 - z^2),$$

I solved $\Delta u = f$ in 3D with dirichlet BC's using a direct solver adapted from part a. I used the standard 7-point Laplacian discretization, which has stencil

$$\frac{1}{h^2}[1 \dots 1 \dots 1 \ -6 \ 1 \dots 1 \dots 1]$$

where the outer entries are n and n^2 away from the center. The comparison between direct-solver and SOR is as follows, where time is in seconds:

Grid spacing h	Direct-Solve Time	SOR time	SOR iterations
2^{-4}	0.0094039440155	2.02679681778	22
2^{-5}	0.0706140995026	37.7071869373	48
2^{-6}	0.607305049896	704.187182903	104
2^{-7}	4.93989610672	—	—

The direct solver takes significantly more time in 3D compared to the 2D case for the same grid spacings. But it still manages to do much better than my SOR code, but if my SOR code was vectorized instead of having a triple for-loop, then perhaps it could have beat the direct solver. My SOR solver did not finish for $h = 2^{-7}$.

Problem 4. *Periodic boundary conditions for the one dimensional Poisson equation on (0, 1) are $u(0) = u(1)$ and $u_x(0) = u_x(1)$. These boundary conditions are easy to discretize, but lead to a singular system to solve. For example, using the standard discretization, $x_j = jh$ where $h = 1/(N + 1)$, the discrete Laplacian at x_0 is $h^{-2}(u_N - 2u_0 + u_1)$.*

(a) Write the discrete Laplacian for periodic boundary conditions in one dimension as a matrix. Show that this matrix is singular, and find the vectors that span the null space. (Note that this matrix is symmetric, and so you have found the null space of the adjoint).

Since $u(0) = u(1)$ and $u_x(0) = u_x(1)$, the endpoints are essentially equivalent. So

$$u_{xx}(0) = \frac{u_n - 2u_0 + u_1}{h^2}$$

and

$$u_{xx}(1-h) = \frac{u_{n-1} - 2u_n + u_0}{h^2}.$$

Then the Laplacian can be discretized as

$$A = \frac{1}{h^2}\begin{pmatrix} -2 & 1 & & 1 \\ 1 & -2 & 1 & \\ & \ddots & \ddots & \ddots \\ 1 & & 1 & -2 \end{pmatrix},$$

where the vector we apply it to is $u = [u_0, u_1, \ldots, u_n]$ (where $u_n = u(1-h)$). Clearly, this matrix is singular since every row sum is 0. The spanning null vector is given by

$$1^T = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix},$$

(b) What is the discrete solvability condition for the discretized Poisson equation with periodic boundary conditions in one dimension? What is the discrete solvability condition in two dimensions?

In one dimension, the solvability condition for the discretized Poisson equation $Au = f$ is given by

$$f \cdot 1^T = 0 \quad \text{or} \quad \sum_{i=0}^{n} f(x_i) = 0.$$

In two dimensions, the discrete Laplacian with periodic boundary conditions is

$$A = \begin{pmatrix} X & Y \\ Y^T & X \end{pmatrix}$$

with

$$X = \begin{pmatrix} -4 & 1 & 0 & \ldots & 0 \\ 1 & -4 & 1 & \ldots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & 1 & -4 & 1 \\ 1 & \ldots & 0 & 1 & -4 \end{pmatrix}$$

and

$$Y = \begin{pmatrix} 1 & 1 & 0 & \ldots & 1 \\ 0 & 1 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \ldots & 1 & 1 & 0 \\ 1 & \ldots & 0 & 1 & 1 \end{pmatrix}.$$

Then again, this matrix is clearly singular since each row sum is 0, so the null space is spanned by a vector of ones, so again the solvability condition is just

$$f \cdot 1^T = 0 \quad \text{or} \quad \sum_{i=0}^{n^2} f(x_i) = 0.$$

(c) Show that v is in the null space of the matrix A, if and only if v is an eigenvector of the iteration matrix $T = M^{-1}N$ with eigenvalue 1, where $A = M - N$. The iteration will converge if the discrete solvability condition is satisfied provided the other eigenvalues are less than 1 in magnitude (true for Gauss-Seidel and SOR, but not for Jacobi).

$$\begin{aligned} Tv &= 1v \\ M^{-1}Nv &= v \\ Nv &= Mv \\ 0 &= (M-N)v = Av. \end{aligned}$$

All of the above steps are reversible, so $v \in \text{null}(A) \iff Tv = v$.

Python Code used for 1:

```
#MAT228A Homework 3
#Question 1

#Use Jacobi, GS-Lex, and SOR (on top of GS-lex)
#to find soluton to Laplacian u = -e^(-(x-.25)^2 - (y-.6)^2)
#with homogeneous Dirichlet BC's

#use mesh spacings h=2^-5, 2^-6, 2^-7
#track iteration counts for each method

from __future__ import division

import numpy as np
from math import exp, sin, pi

from tqdm import tqdm
import matplotlib.pyplot as plt
import scipy.sparse as sparse
import scipy.sparse.linalg
import tabulate
import argparse
from timeit import default_timer as timer

#use command line input to decide which
parser = argparse.ArgumentParser(description='Pick an iterative method')
parser.add_argument("-j", "--Jac", help = "turn on Jacobi iteration", action="store_true", d
parser.add_argument("-g", "--GS", help = "turn on GS iteration", action="store_true", defaul
```

```
parser.add_argument("-s", "--SOR", help = "turn on SOR iteration",action="store_true", defau
args = parser.parse_args()

do_Jac = args.Jac
do_GS = args.GS
do_SOR = args.SOR

#set empty array to hold iteration counts for different mesh spacings
counts = np.zeros((3,3))
#initialize mesh spacing number, to access proper counts row
mesh_number = 0
#set mesh spacings
mesh_spacings = [2**(-5), 2**(-6), 2**(-7)]

#loop through mesh spacings
for h in tqdm(mesh_spacings):

#start timer
start = timer()
#set relative error tolerance
tol = h**2

#set number of grid points in each row/column
n = int(1/h - 1)


#set x,y grid point vectors (n x 1)
x = [i*h for i in range(n+2)]
y = [j*h for j in range(n+2)]

#set empty Jacobi, GS, and SOR solution matrices
u_Jac = np.zeros((n+2, n+2))
u_GS = np.zeros((n+2, n+2))
u_SOR = np.zeros((n+2, n+2))
old_Jac = np.zeros((n+2, n+2))
old_GS = np.zeros((n+2, n+2))
old_SOR = np.zeros((n+2, n+2))


#set optimum w for SOR
w = 2/(1+sin(pi*h))

#sample f = -e^(-(x-.25)^2 - (y-.6)^2) at grid points
f = [ [-exp(-(x[i]-0.25)**2 - (y[j]-0.6)**2) for i in range(n+2)] for j in range(n+2)]
```

```
#initialize method finished flags and method iteration counts to 0
flag_Jac = do_Jac
itcount_Jac = 0
flag_GS = do_GS
itcount_GS = 0
flag_SOR = do_SOR
itcount_SOR = 0

#begin iterative schemes
while  flag_GS or flag_Jac or flag_SOR:

#update iteration counts
if flag_Jac:
itcount_Jac += 1
if flag_GS:
itcount_GS += 1
if flag_SOR:
itcount_SOR += 1

for j in tqdm(range(1,n+1)):
for i in range(1,n+1):
#do if Jacobi iteration not done
if flag_Jac:
u_Jac[i][j] = (1/4)*(old_Jac[i-1][j]+old_Jac[i][j-1]+old_Jac[i+1][j] + old_Jac[i][j+1] - (h*
#do if GS iteration not done
if flag_GS:
u_GS[i][j] = 0.25*(u_GS[i-1][j]+u_GS[i][j-1]+u_GS[i+1][j] + u_GS[i][j+1]) - 0.25*(h**2)*f[i]
#do if SOR iteration not done
if flag_SOR:
u_SOR[i][j] = (w/4)*(u_SOR[i-1][j]+u_SOR[i][j-1]+u_SOR[i+1][j] + u_SOR[i][j+1] - (h**2)*f[i]

#check whether these methods have relative error within the tolerance THIS IS A BAD ERROR TO

if np.amax(np.abs(u_Jac-old_Jac)) <=  tol*np.amax(np.abs(old_Jac)) and flag_Jac:
flag_Jac = 0
if np.amax(np.abs(u_GS-old_GS)) <=  tol*np.amax(np.abs(old_GS)) and flag_GS:
flag_GS = 0
if np.amax(np.abs(u_SOR-old_SOR)) <= tol*np.amax(np.abs(old_SOR)) and flag_SOR:
flag_SOR = 0

#remember (+0 is so that they dont BECOME THE SAME VARIABLE ?!?!?! WTF WHY)
old_Jac = u_Jac+0
old_GS = u_GS +0
old_SOR = u_SOR + 0

#store iteration counts
```

```
counts[mesh_number, 0] = itcount_Jac
counts[mesh_number, 1] = itcount_GS
counts[mesh_number, 2] = itcount_SOR
# print(counts[mesh_number,0])
# print(counts[mesh_number,1])
# print(counts[mesh_number,2])
mesh_number+=1
end = timer()
print(end-start)

#make table of iteration counts
table = [[1/mesh_spacings[i], counts[i][0], counts[i][1], counts[i][2]] for i in range(3)]
print tabulate.tabulate(table, headers = ["Grid size NxN (N shown)", "Jacobi", "GS", "SOR"],
```

Python Code used for 3a:

```
#MAT228A Homework 3
#Question 3 Part A

#Use standard 3-pt Lapacian discretization
#to find soluton to Au = f, dirichlet bc's

#Perform a Direct Solve and time it!
#compare with SOR from question 1

from __future__ import division

import numpy as np
from math import exp

import matplotlib.pyplot as plt
import scipy.sparse as sparse
import scipy.sparse.linalg
import tabulate
from timeit import default_timer as timer

#mesh spacings
mesh_spacings = [2**(-5), 2**(-6), 2**(-7)]



#Loop for refinement study
for h in mesh_spacings:

#start timer
```

```
start = timer()
#Set number of grid points
N = int(1/h - 1)

#set grid points starting
x = [i*h for i in range(N+2)]
y = [j*h for j in range(N+2)]
#set RHS of PDE, Au(x_i,y_j) = f(x_i,y_j) = b
f = [ [-exp(-(x[i]-0.25)**2 - (y[j]-0.6)**2) for i in range(1,N+1)] for j in range(1,N+1)]
b = np.asarray(f)
b= b.flatten()

#set sparse matrix A, the discrete Laplacian
offdiag = (1/(h**2))*np.ones(N)
diag = np.ones(N)*(-4/(h**2))
data = np.vstack((offdiag, offdiag, diag,offdiag, offdiag))
A = sparse.dia_matrix((data, [-N,-1, 0,1,N]), shape = (N**2,N**2))
A_CSC_form = scipy.sparse.csc_matrix(A)

#solve Au = b
approx_u = scipy.sparse.linalg.spsolve(A_CSC_form, b)
end = timer()
print(end-start)
```

Python Code used for 3b:

```
#MAT228A Homework 3
#Question 3 Part B

#Use standard 7-pt Lapacian discretization
#to find soluton to Au = f, dirichlet bc's

#Perform a Direct Solve and time it!
#compare with SOR from question 1

from __future__ import division

import numpy as np
from math import exp

import matplotlib.pyplot as plt
import scipy.sparse as sparse
import scipy.sparse.linalg
import tabulate
from timeit import default_timer as timer
```

```
#mesh spacings
mesh_spacings = [2**(-5), 2**(-6), 2**(-7)]



#Loop for refinement study
for h in mesh_spacings:

#start timer
start = timer()
#Set number of grid points
N = int(1/h - 1)

#set grid points starting
x = [i*h for i in range(N+2)]
y = [j*h for j in range(N+2)]
z = [k*h for k in range(N+2)]

#set RHS of PDE, Au(x_i,y_j) = f(x_i,y_j) = b
f = [ [ [-exp(-(x[i]-0.25)**2 - (y[j]-0.6)**2 -(z[k])**2) for i in range(1,N+1)] for j in ra
b = np.asarray(f)
b= b.flatten()

#set sparse matrix A, the discrete Laplacian
offdiag = (1/(h**2))*np.ones(N)
diag = np.ones(N)*(-6/(h**2))
data = np.vstack((offdiag, offdiag, offdiag, diag,offdiag, offdiag, offdiag))
A = sparse.dia_matrix((data, [-N**2,-N,-1, 0,1,N, N**2]), shape = (N**3,N**3))
A_CSC_form = scipy.sparse.csc_matrix(A)

#solve Au = b
approx_u = scipy.sparse.linalg.spsolve(A_CSC_form, b)
end = timer()
print(end-start)

#MAT228A Homework 3
#Question 3 part B (2)

#Use Jacobi, GS-Lex, and SOR (on top of GS-lex)
#to find soluton to Laplacian u = -e^(-(x-.25)^2 - (y-.6)^2)
#with homogeneous Dirichlet BC's

#use mesh spacings h=2^-5, 2^-6, 2^-7
#track iteration counts for each method
```

```
from __future__ import division

import numpy as np
from math import exp, sin, pi

from tqdm import tqdm
import matplotlib.pyplot as plt
import scipy.sparse as sparse
import scipy.sparse.linalg
import tabulate
import argparse
from timeit import default_timer as timer

#set empty array to hold iteration counts for different mesh spacings
counts = np.zeros(3)
#initialize mesh spacing number, to access proper counts row
mesh_number = 0
#set mesh spacings
mesh_spacings = [2**(-4), 2**(-5), 2**(-6)]

#loop through mesh spacings
for h in tqdm(mesh_spacings):

#start timer
start = timer()
#set relative error tolerance
tol = h**2

#set number of grid points in each row/column
n = int(1/h - 1)



#set x,y grid point vectors (n x 1)
x = [i*h for i in range(n+2)]
y = [j*h for j in range(n+2)]
z = [k*h for k in range(n+2)]

#set empty SOR solution matrices
u_SOR = np.zeros((n+2, n+2, n+2))
old_SOR = np.zeros((n+2, n+2,n+2))



#set optimum w for SOR
w = 2/(1+sin(pi*h))

#sample f = -e^(-(x-.25)^2 - (y-.6)^2) at grid points
```

```
f = [ [ [-exp(-(x[i]-0.25)**2 - (y[j]-0.6)**2 - z[k]**2) for i in range(n+2)] for j in range


#initialize method finished flags and method iteration counts to 0
flag_SOR = 1
itcount_SOR = 0

#begin iterative schemes
while  flag_SOR:

#update iteration counts
if flag_SOR:
itcount_SOR += 1

for k in tqdm(range(1,n+1)):
for j in range(1,n+1):
for i in range(1,n+1):
u_SOR[i][j][k] = (w/6)*(u_SOR[i-1][j][k] +u_SOR[i][j-1][k] + u_SOR[i][j][k-1]+u_SOR[i+1][j][

#check whether these methods have relative error within the tolerance THIS IS A BAD ERROR TO
if np.amax(np.abs(u_SOR-old_SOR)) <= tol*np.amax(np.abs(old_SOR)) and flag_SOR:
flag_SOR = 0

#remember (+0 is so that they dont BECOME THE SAME VARIABLE ?!?!?! WTF WHY)
old_SOR = u_SOR + 0

#store iteration counts
counts[mesh_number] = itcount_SOR
mesh_number+=1
end = timer()
print(end-start)

#make table of iteration counts
table = [[1/mesh_spacings[i], counts[i]] for i in range(3)]
print tabulate.tabulate(table, headers = ["Grid size NxN (N shown)", "SOR"], tablefmt="latex
```