Problem 1)

Laplace's equation in 2D rectangular coordinates

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 0. \tag{1}$$

The Laplace equation applies to problems in steady state heat conduction, ideal fluid flow, electrostatics, etc. Write a code for solving 2D Laplace's equation using finite-difference 2$^{\text{nd}}$-order method and solve 2D Laplace's equation on the square $[0, h] \times [0, h]$ with $h = 10$ and the boundary conditions $f(x, 0) = f(x, h) = f(0, y) = 0$ and $f(h, y) = 10$.

(a) Compute electric potential as a function of x and y.

(b) Create a surface plot of the calculated potential.

(c) Run 10, 100, and 1000 iterations, and note when convergence occurs.

(d) Investigate the effect of varying the step size Δh. Draw conclusions regarding the stability and accuracy of the solution for various Δh.

(e) *Optional*: Investigate the effect of using Gauss-Seidel relaxation versus Jacobi iterations. Which converges faster?

(a) We can discretize our space and convert this continuous partial differential equation into one relating the value of f at neighboring lattice points using the finite difference method. Letting $f(x_i, y_j) = f_{i,j}$, we have

$$\frac{f_{i-1,j} - 2f_{i,j} + f_{i+1,j}}{\Delta x^2} + \frac{f_{i,j-1} - 2f_{i,j} + f_{i,j+1}}{\Delta y^2} = 0. \tag{2}$$

Rearranging, we have

$$f_{i,j} = \frac{1}{2}\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right)^{-1}\left[\frac{f_{i-1,j} + f_{i+1,j}}{\Delta x^2} + \frac{f_{i,j-1} + f_{i,j+1}}{\Delta y^2}\right]. \tag{3}$$

We can devise a method of relaxation based on this equation to construct a sequence for function values at each lattice point such that

$$f_{i,j}^{(n+1)} = \frac{1}{2}\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right)^{-1}\left[\frac{f_{i-1,j}^{(n)} + f_{i+1,j}^{(n)}}{\Delta x^2} + \frac{f_{i,j-1}^{(n)} + f_{i,j+1}^{(n)}}{\Delta y^2}\right], \tag{4}$$

where we choose some initial configuration for $f^{(0)}$. One can terminate the iteration after some chosen number of iterations or after an error tolerance is reached.

(b) In Fig. 1 we plot the numeric solution in the left panel and the difference between the

numeric and analytic solution[1]

$$\frac{V_{\text{analytic}}(x,y)}{V_0} = \frac{4}{\pi}\sum_{n=1}^{\infty}\frac{1}{2n+1}\sin\left(\frac{(2n+1)\pi x}{h}\right)\frac{\sinh[(2n+1)\pi y/h]}{\sinh[(2n+1)\pi]}. \tag{5}$$

We can see that for a sufficient relative tolerance as our termination condition in the iteration, the numeric solution is quite close to the analytic solution. Note that if memory serves correctly, it may be possible to sum this series in closed form, in which case such a treatment would be discussed in *Classical Electrodynamics* by Jackson. For the code implementation here, there is some difficulty for large n given that $\sinh(x)$ diverges when $x \to \infty$. Thus, for to avoid overflow issues and a more efficient evaluation of such terms, it may be desirable to form an asymptotic series for the ratio of the hyperbolic sines in the numerator and denominator or find some other approach.

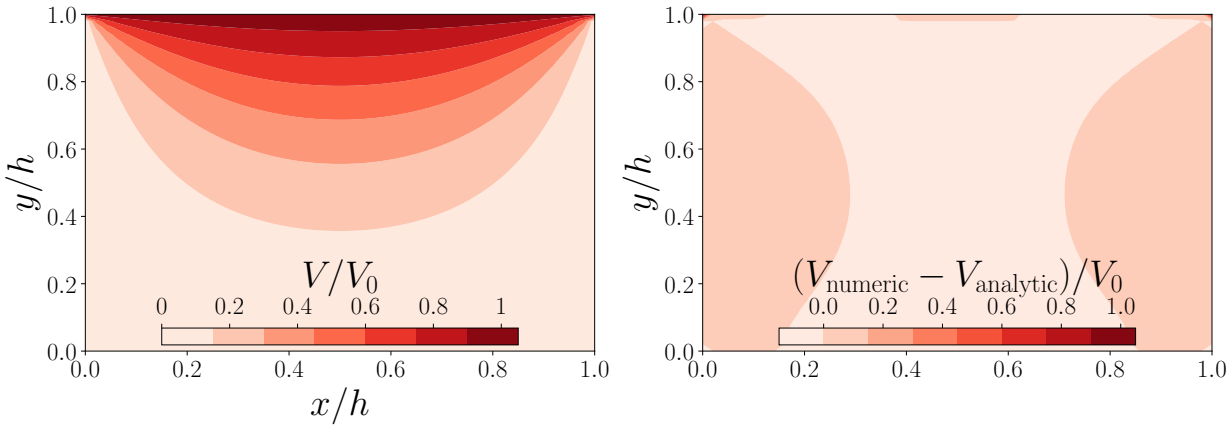

Figure 1: Left: Numeric solution for $N = 50^2$ uniformly distributed points in (x, y) grids with $V_0 = 10$ at $y = h$. **Right**: Absolute difference of the numeric and analytic series solution (evaluted for 100 terms).

(c) In Fig. 2, we show the results of the numeric iteration, running a maximum of $N = 10, 100, 1000$ relaxation iterations for the same configuration as in Fig. 1. Note that none of the panels have converged to satisfy the relative error tolerance: it takes roughly 5000 iterations to do so. However, we can see that our algorithm works as expected, which is that our configuration ideed is relaxing stably to the expected solution.

(d) In Fig. 3, we show the results of the numeric iteration, running on a different grid density with $N = 10^2, 100^2, 500^2$ uniformly spaced points a maximum of 30,000 iterations. Although the algorithm is still stable and the first two configurations converge, the latter fails to in a small running time. This is primarily because of the exponentially larger number of floating point operations. In particular, the propagation of the boundary conditions to the interior of the square region requires a larger number of iterations.

[1]Note that contour plots are shown in this article instead of surface plots. There were some memory related issues that made creating the necessary surface plots problematic on my personal computer.

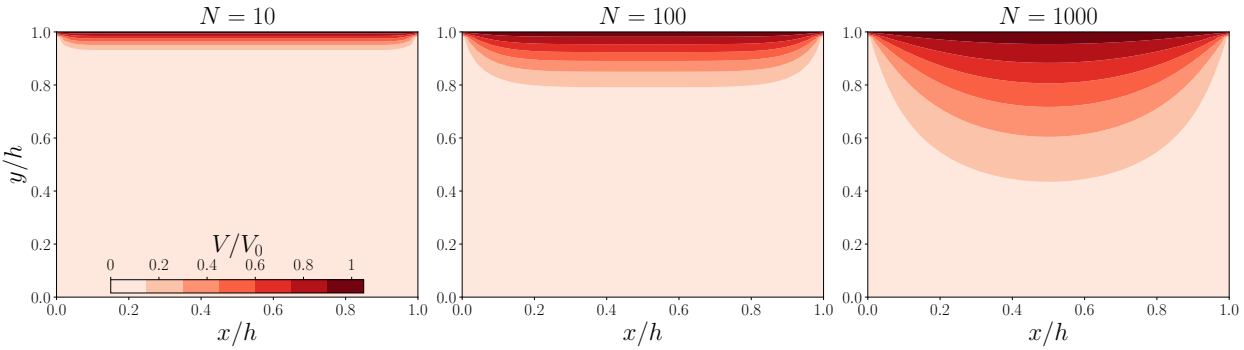

Figure 2: Numeric solutions for $N = 50^2$ uniformly distributed points in (x, y) grids, run with a maximum number of iterations $10, 100, 1000$ from left to right, with $V_0 = 10$ at $y = h$.

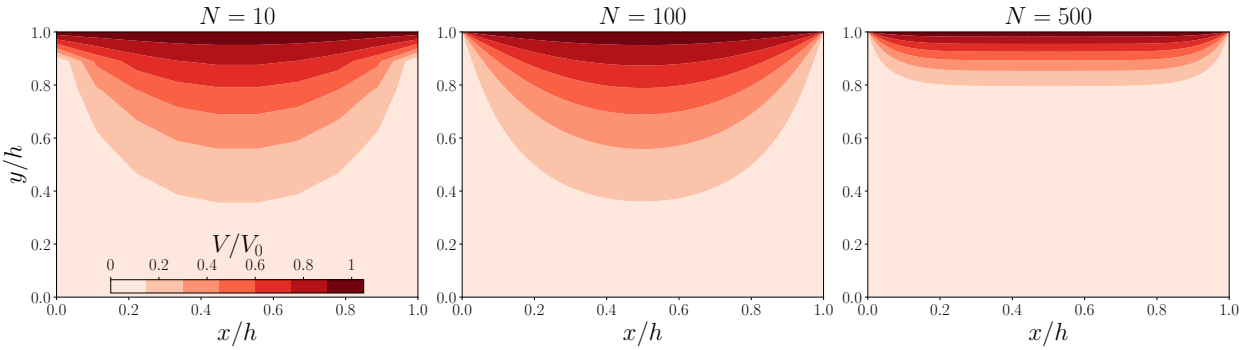

Figure 3: Numeric solutions for $N = 10^2, 100^2, 500^2$ uniformly distributed points in (x, y) grids, from left to right, with $V_0 = 10$ at $y = h$.

```python
import numpy as np

import matplotlib.pyplot as plt
plt.rcParams.update({
    'text.latex.preamble': r'\usepackage{amsmath}',
    'text.usetex': True,
    'font.family': 'sans-serif',
    'font.sans-serif': ['Helvetica']
})
from mpl_toolkits.axes_grid1.inset_locator import inset_axes
import matplotlib.gridspec as gridspec

def relax_f(f,X,Y,max_iter=int(1e5),rho=lambda t1,t2: np.full_like(t1,0),
    aeps=1e-20,reps=1e-6):
    dx = X[0,1] - X[0][0]
    dy = Y[1,0] - Y[0,0]

    rho_ = rho(X,Y)[1:-1,1:-1]
    for i in range(max_iter):
        f1 = f.copy()

        fm0 = f1[:-2,1:-1]
```

```
        fp0 = f1[2:,1:-1]
        f0m = f1[1:-1,:-2]
        f0p = f1[1:-1,2:]

        f1[1:-1,1:-1] = 0.5/(1/dx**2 + 1/dy**2)*((fm0 + fp0)/dx**2 + (f0m +
    f0p)/dy**2 + rho_)

        adiff = f1 - f
        rdiff = adiff/(f + 1e-100)

        f = f1.copy()

        if np.all(np.abs(adiff) < aeps) or np.all(np.abs(rdiff) < reps):
            print(i)
            break

    return f


xa,xb,Nx = 0,1,50
ya,yb,Ny = 0,1,50

x    = np.linspace(xa,xb,Nx)
y    = np.linspace(ya,yb,Ny)
X,Y = np.meshgrid(x,y)

f0 = np.zeros((Nx,Ny))
f0[-1,:] = 1


nrows,ncols = 1,3
fig,ax = plt.subplots(nrows=nrows,ncols=ncols,figsize=(7*ncols,6*nrows))

for i,N in enumerate([10,100,1000]):
    f = relax_f(f0,X,Y,max_iter=N)

    img = ax[i].contourf(X,Y,f,cmap='Reds',vmin=0,vmax=1)
    ax[i].tick_params(axis='both',which='major',direction='out',labelsize
    =20)
    ax[i].set_title(r'$N = %d$'%N,size=30,pad=10)
    ax[i].set_xlabel(r'$x/h$',size=30)

ax[0].set_ylabel(r'$y/h$',size=30)

plt.tight_layout()

ticks = [0,0.2,0.4,0.6,0.8,1]
cbaxes = inset_axes(ax[0], width="70%", height="5%", loc='lower center')
cbar = fig.colorbar(img,cax=cbaxes,orientation='horizontal',ticks=ticks)
cbaxes.xaxis.set_ticks_position('top')
cbar.ax.tick_params(labelsize=20)
cbar.ax.set_xticklabels([r'$%.1g$'%_ for _ in ticks])
cbar.ax.set_title(r'$V/V_0$',size=30)
```

```
plt.show()
fig.savefig(r'part1-iter.pdf',bbox_inches='tight')


nrows,ncols = 1,3
fig,ax = plt.subplots(nrows=nrows,ncols=ncols,figsize=(7*ncols,6*nrows))

for i,N in enumerate([10,100,500]):

    xa,xb,Nx = 0,1,N
    ya,yb,Ny = 0,1,N

    x   = np.linspace(xa,xb,Nx)
    y   = np.linspace(ya,yb,Ny)
    X,Y = np.meshgrid(x,y)

    f0 = np.zeros((Nx,Ny))
    f0[-1,:] = 1
    f = relax_f(f0,X,Y,max_iter=int(1e4))

    img = ax[i].contourf(X,Y,f,cmap='Reds',vmin=0,vmax=1)
    ax[i].tick_params(axis='both',which='major',direction='out',labelsize
   =20)
    ax[i].set_title(r'$N = %d$'%N,size=30,pad=10)
    ax[i].set_xlabel(r'$x/h$',size=30)

ax[0].set_ylabel(r'$y/h$',size=30)

plt.tight_layout()

ticks = [0,0.2,0.4,0.6,0.8,1]
cbaxes = inset_axes(ax[0], width="70%", height="5%", loc='lower center')
cbar = fig.colorbar(img,cax=cbaxes,orientation='horizontal',ticks=ticks)
cbaxes.xaxis.set_ticks_position('top')
cbar.ax.tick_params(labelsize=20)
cbar.ax.set_xticklabels([r'$%.1g$'%_ for _ in ticks])
cbar.ax.set_title(r'$V/V_0$',size=30)

plt.show()
fig.savefig(r'part1-N.pdf',bbox_inches='tight')



def f_exact(x,y,f0=10,N=100):
    n   = 2*np.arange(N) + 1
    A_n = 4*f0/(n*np.pi*np.sinh(n*np.pi))
    f_n = A_n * np.sin(n*np.pi*x) * np.sinh(n*np.pi*y)
    # print(n,A_n,f_n)
    return np.sum(f_n)


xa,xb,Nx = 0,1,50
ya,yb,Ny = 0,1,50
```

```
x    = np.linspace(xa,xb,Nx)
y    = np.linspace(ya,yb,Ny)
X,Y = np.meshgrid(x,y)

f0 = np.zeros((Nx,Ny))
f0[-1,:] = 1
f = relax_f(f0,X,Y)

f_exact_ = np.vectorize(f_exact)(X,Y,f0=1,N=100)


nrows,ncols = 1,2
fig,ax = plt.subplots(nrows=nrows,ncols=ncols,figsize=(7*ncols,5*nrows))

img0 = ax[0].contourf(X,Y,f,cmap='Reds')

ax[0].tick_params(axis='both',which='major',direction='out',labelsize=20)
# ax[0].set_title(r'$\displaystyle V/V_0$',size=30,pad=10)
ax[0].set_xlabel(r'$x/h$',size=30)
ax[0].set_ylabel(r'$y/h$',size=30)

img1 = ax[1].contourf(X,Y,f-f_exact_,cmap='Reds')

ax[1].tick_params(axis='both',which='major',direction='out',labelsize=20)
# ax[1].set_title(r'$\displaystyle V/V_0$',size=30,pad=10)
# ax[1].set_xlabel(r'$x/h$',size=30)
ax[1].set_ylabel(r'$y/h$',size=30)

plt.tight_layout()

ticks = [0,0.2,0.4,0.6,0.8,1]
cbaxes = inset_axes(ax[0], width="70%", height="5%", loc='lower center')
cbar = fig.colorbar(img0,cax=cbaxes,orientation='horizontal',ticks=ticks)
cbaxes.xaxis.set_ticks_position('top')
cbar.ax.tick_params(labelsize=20)
cbar.ax.set_xticklabels([r'$%.1g$'%_ for _ in ticks])
cbar.ax.set_title(r'$V/V_0$',size=30)

# ticks = [0,0.2,0.4,0.6,0.8,1]
# print(np.abs(f - f_exact_) < 1e-2)
cbaxes = inset_axes(ax[1], width="70%", height="5%", loc='lower center')
cbar = fig.colorbar(img1,cax=cbaxes,orientation='horizontal',ticks=ticks)
cbaxes.xaxis.set_ticks_position('top')
cbar.ax.tick_params(labelsize=20)
# cbar.ax.set_xticklabels([r'$%.1g$'%_ for _ in ticks])
cbar.ax.set_title(r'$(V_{\rm numeric} - V_{\rm analytic})/V_0$',size=30)

plt.show()
fig.savefig(r'part1-comp.pdf')



nrows,ncols = 1,1
fig,ax = plt.subplots(nrows=nrows,ncols=ncols,figsize=(7*nrows,5*ncols))
```

```
img = ax.contourf(X,Y,f,cmap='Reds')#,vmin=0,vmax=1)
# cbar = fig.colorbar(img)
# cbar.ax.tick_params(labelsize=20)

ticks = [0,0.2,0.4,0.6,0.8,1]
cbaxes = inset_axes(ax, width="70%", height="5%", loc='lower center')
cbar = fig.colorbar(img,cax=cbaxes,orientation='horizontal',ticks=ticks)
cbaxes.xaxis.set_ticks_position('top')
cbar.ax.tick_params(labelsize=20)
cbar.ax.set_xticklabels([r'$%.1g$'%_ for _ in ticks])
# cbar.ax.set_title(r'$V/V_0$',size=30)
# ticks = [0,0.2,0.4,0.6,0.8,1]
# cbar.ax.set_yticks(ticks,[r'$%.1g$'%_ for _ in ticks])

ax.tick_params(axis='both',which='major',direction='out',labelsize=20)
ax.set_title(r'$\displaystyle V/V_0$',size=30,pad=10)
ax.set_xlabel(r'$x/h$',size=30)
ax.set_ylabel(r'$y/h$',size=30)

plt.show()
```

Problem 2)

Poisson's equation in 2D rectangular coordinates has a nonhomogeneous term

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = -4\pi\rho(x,y). \tag{6}$$

Now consider the following boundary conditions for the square region $[0,h]\times[0,h]$ with $h=\pi$,

$$\begin{aligned} f(x,0) &= \sin x, \quad & f(x,h) &= |\sin 2x| \\ f(0,y) &= \frac{1}{2}\sin y, \quad & f(h,y) &= 1.2\sin y, \end{aligned} \tag{7}$$

and

$$-4\pi\rho(x,y) = \begin{cases} 360 & x=y=h/3 \\ 0 & \text{otherwise} \end{cases}. \tag{8}$$

(a) Compute electric potential as a function of x and y.

(b) Create a surface plot of the calculated potential.

(c) Investigate the effect of varying the step size Δh. Draw conclusions regarding the stability and accuracy of the solution for various Δh.

In this problem, we can utilize the same method of relaxation as in problem 1. We need

only add the source term such that

$$f_{i,j}^{(n+1)} = \frac{1}{2}\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right)^{-1}\left[\frac{f_{i-1,j}^{(n)} + f_{i+1,j}^{(n)}}{\Delta x^2} + \frac{f_{i,j-1}^{(n)} + f_{i,j+1}^{(n)}}{\Delta y^2} + 4\pi\rho_{i,j}\right]. \tag{9}$$

(b,c) In Fig. 4, we plot results for the boundary conditions above. Additionally, we suppose that the goal of the charge distribution was to simulate a point charge. In order to make a better test function, we choose a new charge density modeled as

$$\rho(\boldsymbol{x}; \boldsymbol{x}_0, \sigma) = \frac{1}{\pi\sigma^2} e^{-(\boldsymbol{x}-\boldsymbol{x}_0)^2/(2\sigma^2)}, \tag{10}$$

where $\boldsymbol{x}_0$ and σ are the center of the charge distribution and σ being the spread of the charge. As $\sigma \to 0$, $\rho(\boldsymbol{x}; \boldsymbol{x}_0) \to \delta(\boldsymbol{x} - \boldsymbol{x}_0)$. The primary issue of the original charge distribution is that it is unphysical in that it has a value at a certain point, but has no measure and therefore leads to no total charge when integrated. Additionally, in order to get any contribution from the original proposed charge distribution, one of the grid points must coincide with the "charge" location. By replacing this with a Gaussian, we smear out the effects so that points near the center feel the effects more and those farther from the center feel them less, which replicates our physical intuition for a point charge distribution.

As we observed in problem (1), one can see the effects of increasing the grid size. At the lower end of the range, our solution has converged, but the grid is not capable of capturing the rich variance provided by our boundary conditions and the existence of a charge, and at the higher end of the range, while our grid is capable of capturing the behavior of our potential, it is expensive to reach such a distribution on a personal computer. In the middle of these values, though, we achieve a stable solution within a reasonable number of iterations.

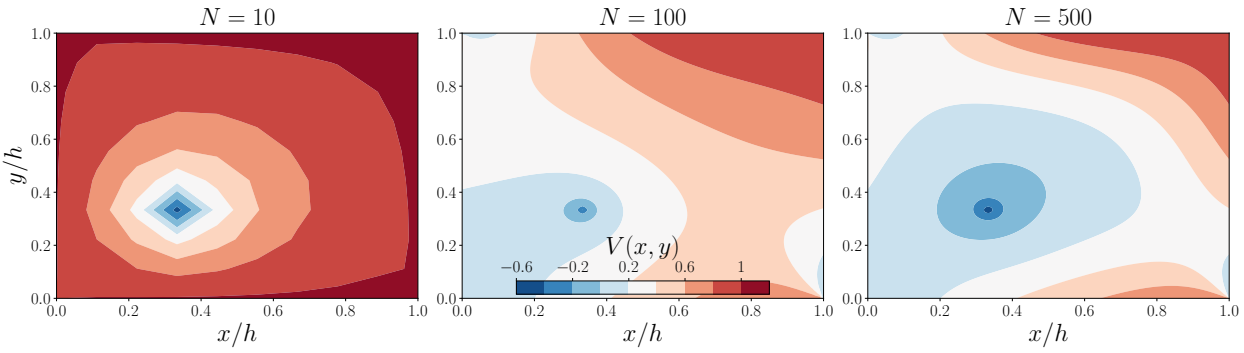

Figure 4: Potential satisfying Poisson's equation on a square region with a point charge located at $(x, y) = (h/3, h/3)$ and represented by a Gaussian of width $\sigma = 0.05$ (in arbitrary units).

```
import numpy as np

import matplotlib.pyplot as plt
```

```
plt.rcParams.update({
    'text.latex.preamble': r'\usepackage{amsmath}',
    'text.usetex': True,
    'font.family': 'sans-serif',
    'font.sans-serif': ['Helvetica']
})
from mpl_toolkits.axes_grid1.inset_locator import inset_axes
import matplotlib.gridspec as gridspec

def relax_f(f,X,Y,max_iter=int(1e5),rho=lambda t1,t2: np.full_like(t1,0),
    aeps=1e-20,reps=1e-6):
    dx = X[0,1] - X[0][0]
    dy = Y[1,0] - Y[0,0]

    rho_ = rho(X,Y)[1:-1,1:-1]
    for i in range(max_iter):
        f1 = f.copy()

        fm0 = f1[:-2,1:-1]
        fp0 = f1[2:,1:-1]
        f0m = f1[1:-1,:-2]
        f0p = f1[1:-1,2:]

        f1[1:-1,1:-1] = 0.5/(1/dx**2 + 1/dy**2)*((fm0 + fp0)/dx**2 + (f0m +
    f0p)/dy**2 + rho_)

        adiff = f1 - f
        rdiff = adiff/(f + 1e-100)

        f = f1.copy()

        if np.all(np.abs(adiff) < aeps) or np.all(np.abs(rdiff) < reps):
            print(i)
            break

    return f


def rho(x,y,x0=1/3,y0=1/3,sig=1,q=10):
    return q*np.exp(-((x-x0)**2 + (y-y0)**2)/2/sig**2)/(2*np.pi*sig**2)#
    360*(np.isclose(x,1/3))*(np.isclose(y,1/3))


xa,xb,Nx = 0,1,100
ya,yb,Ny = 0,1,100

x    = np.linspace(xa,xb,Nx)
y    = np.linspace(ya,yb,Ny)
X,Y = np.meshgrid(x,y)

f0 = np.zeros((Nx,Ny))
f0[0,:]  = np.sin(x)
f0[:,0]  = 0.5*np.sin(y)
f0[-1,:] = np.abs(np.sin(2*x))
```

```
f0[:,-1] = 1.2*np.sin(y)

dx = x[1] - x[0]
dy = y[1] - y[0]


nrows,ncols = 1,1
fig,ax = plt.subplots(nrows=nrows,ncols=ncols,figsize=(7*nrows,5*ncols))

rho_ = rho(X,Y,x0=0.5,y0=0.5,sig=0.001,q=1)
img = ax.contourf(X,Y,rho_,cmap='viridis')#,vmin=0,vmax=1)
cbar = fig.colorbar(img)
cbar.ax.tick_params(labelsize=20)
# ticks = [0,0.2,0.4,0.6,0.8,1]
# cbar.ax.set_yticks(ticks,[r'$%.1g$'%_ for _ in ticks])

ax.tick_params(axis='both',which='major',direction='out',labelsize=20)
ax.set_title(r'$\displaystyle V/V_0$',size=30,pad=10)
ax.set_xlabel(r'$x/h$',size=30)
ax.set_ylabel(r'$y/h$',size=30)

plt.show()


xa,xb,Nx = 0,1,100
ya,yb,Ny = 0,1,100

x  = np.linspace(xa,xb,Nx)
y  = np.linspace(ya,yb,Ny)
X,Y = np.meshgrid(x,y)

f0 = np.zeros((Nx,Ny))
f0[0,:]  = np.sin(x)
f0[:,0]  = 0.5*np.sin(y)
f0[-1,:] = np.abs(np.sin(2*x))
f0[:,-1] = 1.2*np.sin(y)

dx = x[1] - x[0]
dy = y[1] - y[0]


f = relax_f(f0,X,Y,rho=lambda t1,t2: rho(t1,t2,x0=1/3,y0=1/3,sig=0.005,q
    =-1))


nrows,ncols = 1,1
fig,ax = plt.subplots(nrows=nrows,ncols=ncols,figsize=(7*nrows,5*ncols))

img = ax.contourf(X,Y,f,cmap='RdBu_r')#,vmin=0,vmax=1)
cbar = fig.colorbar(img)
cbar.ax.tick_params(labelsize=20)
# ticks = [0,0.2,0.4,0.6,0.8,1]
# cbar.ax.set_yticks(ticks,[r'$%.1g$'%_ for _ in ticks])
```

```
ax.tick_params(axis='both',which='major',direction='out',labelsize=20)
ax.set_title(r'$\displaystyle V/V_0$',size=30,pad=10)
ax.set_xlabel(r'$x/h$',size=30)
ax.set_ylabel(r'$y/h$',size=30)

plt.show()
fig.savefig(r'part2_%d-N.pdf'%Nx,bbox_inches='tight')



nrows,ncols = 1,3
fig,ax = plt.subplots(nrows=nrows,ncols=ncols,figsize=(7*ncols,6*nrows))

for i,N in enumerate([10,100,500]):

    xa,xb,Nx = 0,1,N
    ya,yb,Ny = 0,1,N

    x    = np.linspace(xa,xb,Nx)
    y    = np.linspace(ya,yb,Ny)
    X,Y = np.meshgrid(x,y)

    f0 = np.zeros((Nx,Ny))
    f0[0,:]  = np.sin(x)
    f0[:,0]  = 0.5*np.sin(y)
    f0[-1,:] = np.abs(np.sin(2*x))
    f0[:,-1] = 1.2*np.sin(y)
    f = relax_f(f0,X,Y,rho=lambda t1,t2: rho(t1,t2,x0=1/3,y0=1/3,sig=0.005,
    q=-1),max_iter=30000)

    img = ax[i].contourf(X,Y,f,cmap='RdBu_r')
    ax[i].tick_params(axis='both',which='major',direction='out',labelsize
    =20)
    ax[i].set_title(r'$N = %d$'%N,size=30,pad=10)
    ax[i].set_xlabel(r'$x/h$',size=30)

ax[0].set_ylabel(r'$y/h$',size=30)

plt.tight_layout()

ticks = [-0.6,-0.2,0.2,0.6,1]
cbaxes = inset_axes(ax[1], width="70%", height="5%", loc='lower center')
cbar = fig.colorbar(img,cax=cbaxes,orientation='horizontal',ticks=ticks)
cbaxes.xaxis.set_ticks_position('top')
cbar.ax.tick_params(labelsize=20)
cbar.ax.set_xticklabels([r'$%.1g$'%_ for _ in ticks])
cbar.ax.set_title(r'$V(x,y)$',size=30)

plt.show()
fig.savefig(r'part2-N.pdf',bbox_inches='tight')
```

Problem 3)

The heat equation for 1D case is

$$\frac{\partial T(x,t)}{\partial t} = a\frac{\partial^2 T(x,t)}{\partial x^2}. \tag{11}$$

You are given a metal bar of length $L = 1$ m and aligned along the x-axis. It is insulated along its length though not at its ends. Initially the entire bar is at a uniform temperature of $T_0 = 100°$C and then both ends are placed in contact with ice water at 0°C. Heat flows through the non-insulated ends only.

(a) Write a code for solving $1D$ diffusion equation.

(b) Calculate how the temperature varies along the length of the bar as a function of time.

(c) Create a 3D plot of $T(x,t)$.

(d) Vary the time and space steps in your calculation so that you obtain solutions that are stable in time and vary smoothly in both space and time.

(e) Test what happens when the von Neumann stability condition $a\Delta t/\Delta x^2 \leq 0.5$ is not satisfied.

(f) *Optional*: The diffusion equation above can be solved analytically as series. Compare the analytic and numeric solutions.

(a) Following the same logic as the previous two problems, we discretize our space and use the finite difference method to write

$$T_{i,j+1} = T_{i,j} + \frac{a\Delta t}{\Delta x^2}\Big(T_{i-1,j} - 2T_{i,j} + T_{i+1,j}\Big), \tag{12}$$

where $T_{i,j} \equiv T(x_i, t_j)$. With this, we already have a natural time-sequence from this equation given that $T(x,0)$ is supplied as well as the boundary conditions.

(b,c,d,f) In Fig. 5, we have simulated the dynamical behavior of the temperature of the bar for the given initial configuration on a lattice of 100 points with $a = 1$ (in arbitrary units) and $\beta = 0.25$, where $\beta = a\Delta t/\Delta x^2$. From the plot, we can see the expected behavior, which is that the temperature decreases throughout the bar, converging to the temparature of the ends, and that the temperature in the center of the bar decreases slower than those closer to the ends as expected. We also have plotted the difference between the numeric and analytic solution, given by

$$T(x,t) = \frac{4T_0}{\pi}\sum_{n=1}^{\infty}\frac{1}{2n+1}e^{-a(2n+1)^2\pi^2 t/h^2}\sin\left(\frac{(2n+1)\pi x}{h}\right). \tag{13}$$

As opposed to the Laplace's equation analytic solution's y-dependence, there is no difficulty in implementing the time dependence given that here, the time dependence kills off the higher-"energy" modes.

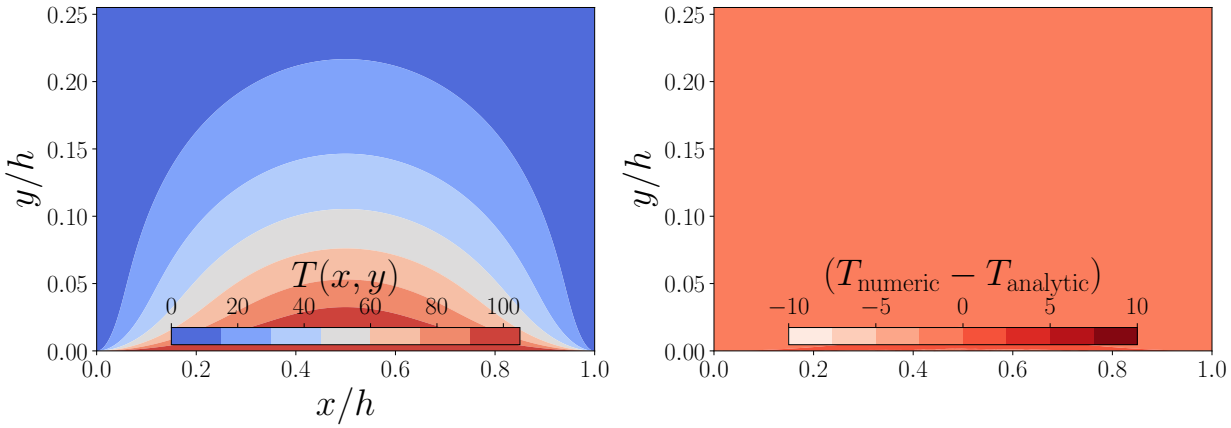

Figure 5: Solution to the heat diffusion equation with $T_0 = 100$ (in arbitrary units), $T = 0$ for all t at the endpoints, and with 100 grid points and $\beta = 0.25$.

(f) In Fig. 6, we plot the result of the finite difference method with $\beta = 0.51$, which violates the von Neumann stability condition, and very clearly, we can see that the numerics are incredibly poorly behaved. Indeed, the instability propagates through each successive step, leaving us at the last step in a place very far from the desired solution and initial configuration.

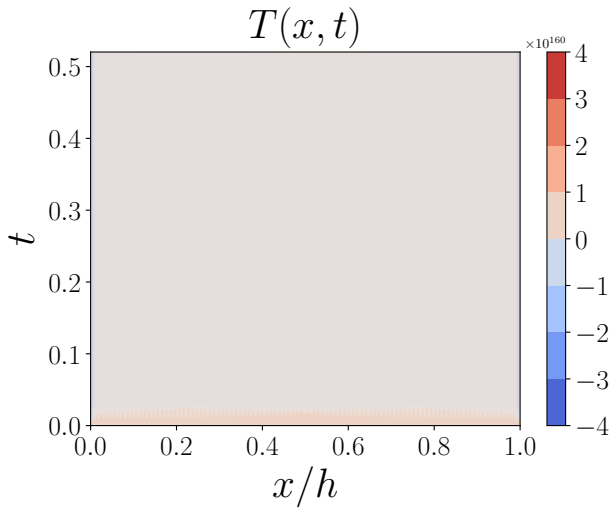

Figure 6: Demonstration of the instability of the finite difference method when the von Neumann condition $\beta = a\Delta t/\Delta x^2 \leq 0.5$ is not satisfied.

```
import numpy as np

import matplotlib.pyplot as plt
plt.rcParams.update({
    'text.latex.preamble': r'\usepackage{amsmath}',
    'text.usetex': True,
    'font.family': 'sans-serif',
    'font.sans-serif': ['Helvetica']
})
```

```
from mpl_toolkits.axes_grid1.inset_locator import inset_axes
import matplotlib.gridspec as gridspec



Nx = 100
x = np.linspace(0,1,Nx)
dx = x[1] - x[0]
a = 1
b = 0.25

dt = b*dx**2/a
Nt = 10000
t = dt*np.arange(Nt)

X,T = np.meshgrid(x,t)



f0 = np.zeros(Nx)
f0[1:-1] = 100

f = [f0]
for i in range(Nt-1):
    temp = f[-1].copy()
    temp[1:-1] += b*(temp[:-2] - 2*temp[1:-1] + temp[2:])
    f.append(temp)


for _ in f[::1000]:
    plt.plot(x,_)

plt.show()



nrows,ncols = 1,1
fig,ax = plt.subplots(nrows=nrows,ncols=ncols,figsize=(7*nrows,5*ncols))

img = ax.contourf(X,T,f,cmap='coolwarm')#,vmin=0,vmax=1)
cbar = fig.colorbar(img)
cbar.ax.tick_params(labelsize=20)
# ticks = [0,0.2,0.4,0.6,0.8,1]
# cbar.ax.set_yticks(ticks,[r'$%.1g$'%_ for _ in ticks])

ax.tick_params(axis='both',which='major',direction='out',labelsize=20)
ax.set_title(r'$\displaystyle T(x,t)$',size=30,pad=10)
ax.set_xlabel(r'$x/h$',size=30)
ax.set_ylabel(r'$t$',size=30)

plt.show()
fig.savefig(r'part3-sol_stable.pdf',bbox_inches='tight')
```

```
def f_exact(x,t,f0=100,a=1,N=100):
    n   = 2*np.arange(N) + 1
    A_n = 4*f0/(n*np.pi)
    f_n = A_n * np.sin(n*np.pi*x) * np.exp(-a*n**2*np.pi**2*t)
    return np.sum(f_n)

f_exact_ = np.vectorize(f_exact)(X,T,f0=100,N=100)



nrows,ncols = 1,2
fig,ax = plt.subplots(nrows=nrows,ncols=ncols,figsize=(7*ncols,5*nrows))

img0 = ax[0].contourf(X,T,f,cmap='coolwarm')

ax[0].tick_params(axis='both',which='major',direction='out',labelsize=20)
# ax[0].set_title(r'$\displaystyle V/V_0$',size=30,pad=10)
ax[0].set_xlabel(r'$x/h$',size=30)
ax[0].set_ylabel(r'$y/h$',size=30)

img1 = ax[1].contourf(X,T,f-f_exact_,cmap='Reds')

ax[1].tick_params(axis='both',which='major',direction='out',labelsize=20)
# ax[1].set_title(r'$\displaystyle V/V_0$',size=30,pad=10)
# ax[1].set_xlabel(r'$x/h$',size=30)
ax[1].set_ylabel(r'$y/h$',size=30)

plt.tight_layout()

ticks = [0,20,40,60,80,100]
cbaxes = inset_axes(ax[0], width="70%", height="5%", loc='lower center')
cbar = fig.colorbar(img0,cax=cbaxes,orientation='horizontal',ticks=ticks)
cbaxes.xaxis.set_ticks_position('top')
cbar.ax.tick_params(labelsize=20)
cbar.ax.set_xticklabels([r'$%.3g$'%_ for _ in ticks])
cbar.ax.set_title(r'$T(x,y)$',size=30)

ticks = [-10,-5,0,5,10]
# print(np.amax)
print(np.amax(f - f_exact_))
cbaxes = inset_axes(ax[1], width="70%", height="5%", loc='lower center')
cbar = fig.colorbar(img1,cax=cbaxes,orientation='horizontal',ticks=ticks)
cbaxes.xaxis.set_ticks_position('top')
cbar.ax.tick_params(labelsize=20)
# cbar.ax.set_xticklabels([r'$%.1g$'%_ for _ in ticks])
cbar.ax.set_title(r'$(T_{\rm numeric} - T_{\rm analytic})$',size=30)

plt.show()
fig.savefig(r'part3-sol.pdf')
```