HW3_Solutions

January 29, 2024

[1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import rcParams
rcParams['font.size'] = 16
rcParams['figure.figsize'] = (10,8)
```

1 Problem 1

1.1 Boundary value problem with a capacitor

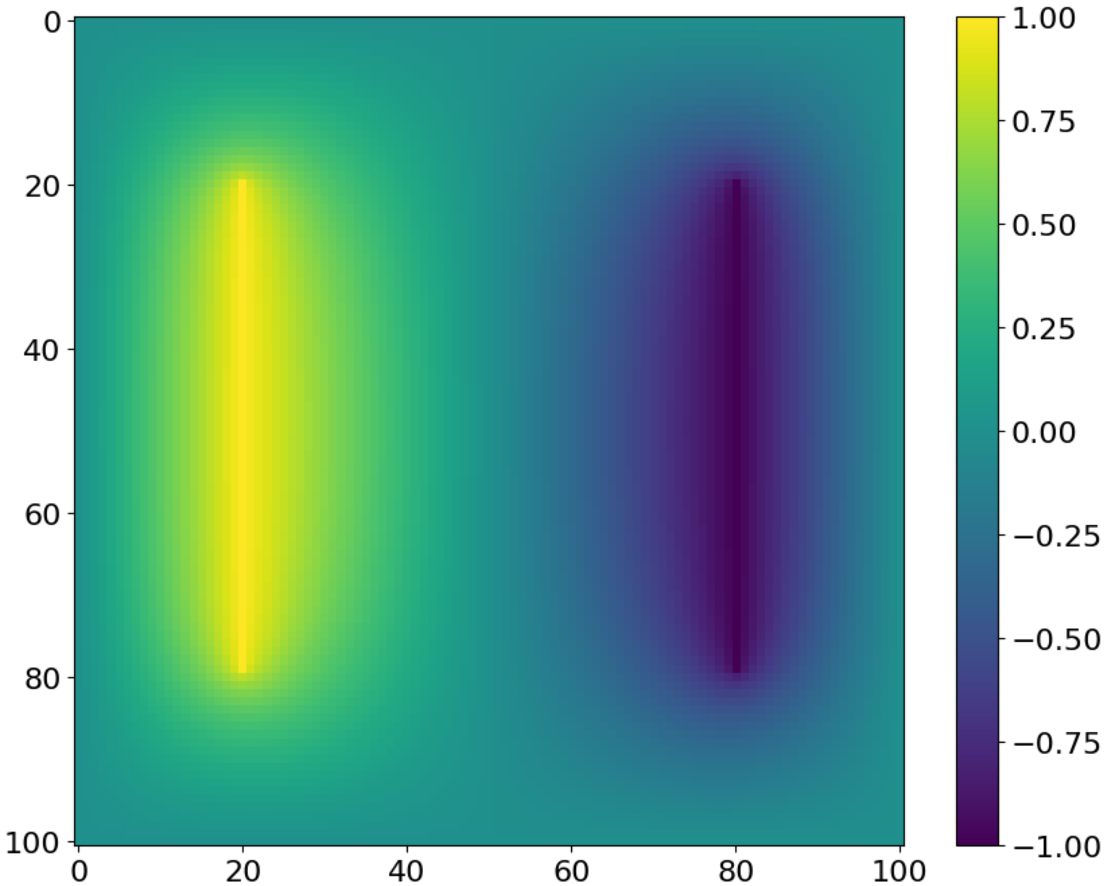
[2]:
```
L = 0.1
N = 100
a = L/N
clo = N//5
chi = 4*N//5
omega = 0.9
```

[3]:
```
# Set up initial values and boundary array
phi = np.zeros([N+1, N+1], float)
phi[clo:chi,clo] = 1.0
phi[clo:chi,chi] = -1.0

plates = np.zeros([N+1,N+1], float)
plates[clo:chi,clo] = 1
plates[clo:chi,chi] = 1
```

[4]:
```
# Solve for phi using Gauss-Seidel and over-relaxation
delta = 1.0
target = 1e-6

while delta > target:
    delta = 0.0
    # Loops are bad and slow, but it's more clear this way
    for i in range(1,N):
        for j in range(1,N):
            if plates[i,j] == 0:
```

```
                oldphi = phi[i,j]
                phi[i,j] = (1+omega)*(phi[i+1,j] + phi[i-1,j] + phi[i,j+1] + \
                                      phi[i,j-1])/4 - omega*phi[i,j]
                epsilon = abs(phi[i,j] - oldphi)
                delta = max(delta, epsilon)
```

[5]:
```
plt.imshow(phi)
plt.colorbar()
plt.show()
```



2 Problem 2

2.1 Thermal diffusion in the Earth's crust

[6]:
```
# Constants
L = 20    # length of the system
N = 100   # Number of spatial points
a = L/N   # Point separation
```

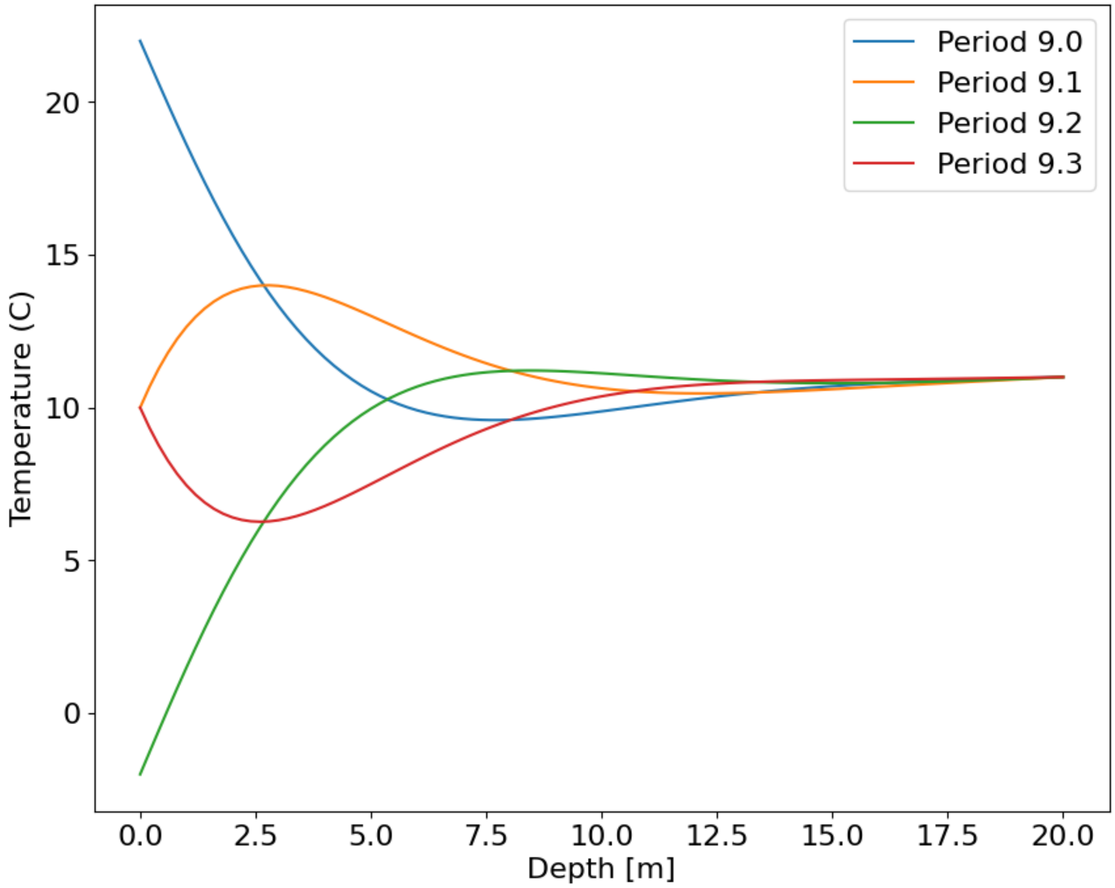
```
h = 0.01 # Time spacing in days
D = 0.1  # Thermal diffusivity of the crust

A = 10.0 # Constants for sinusoidal variation
B = 12.0
tau = 365
Tfixed = 11.0  # Temperature at the bottom
```

[7]:
```
tmax = 10.01*365
steps = int(tmax/h)

s1 = int(9.25*365/h)
s2 = int(9.50*365/h)
s3 = int(9.75*365/h)
s4 = int(10.0*365/h)
```

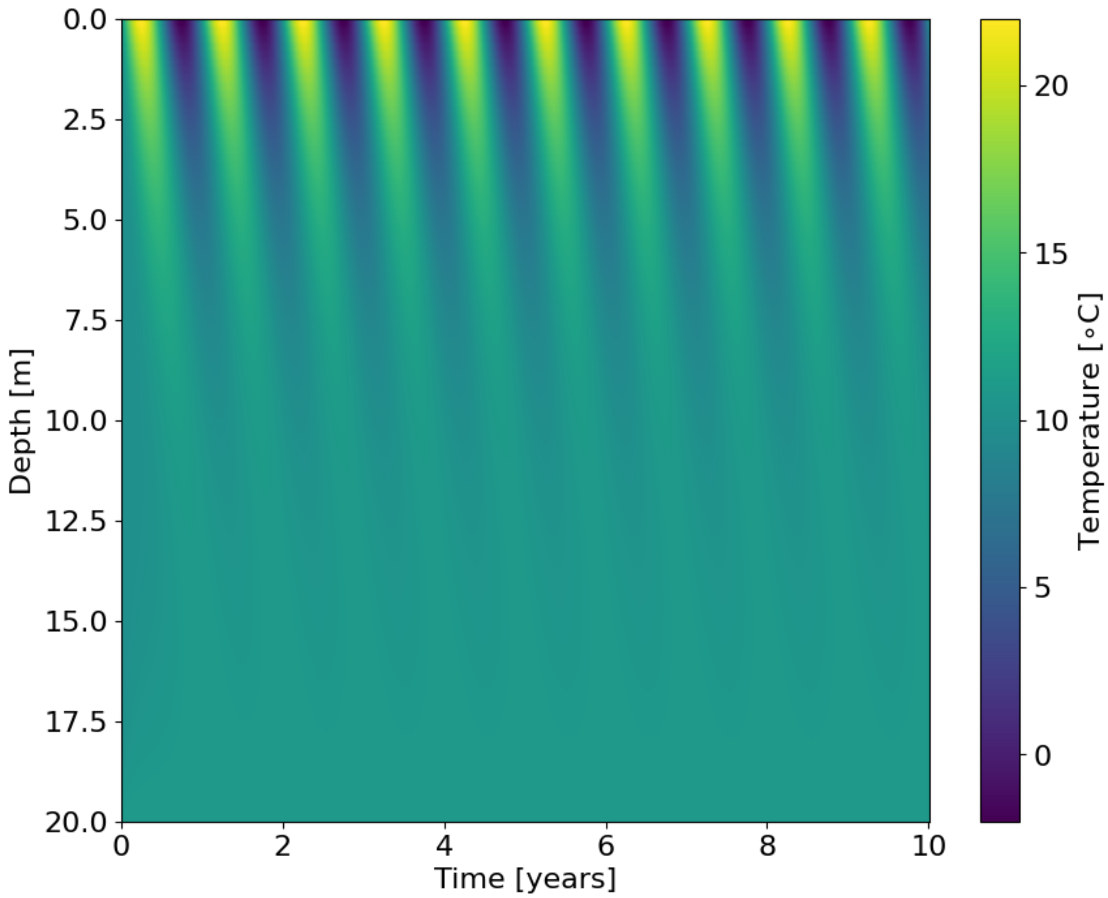
[8]:
```
# Set up initial conditions
T = np.zeros(N+1, float)
T[0:N] = A
T[N] = Tfixed
```

[9]:
```
# Main loop
x = np.linspace(0, L, N+1)
saved_soln = []
allT = np.empty((N+1,steps))
for k in range(steps):
    t = k*h
    T[0] = A + B*np.sin(2*np.pi*t/tau)
    T[1:N] += h*D*(T[0:N-1] + T[2:N+1] - 2*T[1:N]) / (a*a)
    allT[:,k] = T.copy()
    if k in [s1,s2,s3,s4]:
        saved_soln.append(T.copy())
```

[10]:
```
# Plot it
for i,soln in enumerate(saved_soln):
    plt.plot(x, soln, label='Period 9.%d' % (i))
plt.legend(loc='best')
plt.xlabel('Depth [m]')
plt.ylabel('Temperature (C)')
plt.show()
```



[11]:
```
plt.imshow(allT, aspect='auto', extent=[0, tmax/365, L, 0])
plt.xlabel('Time [years]')
plt.ylabel('Depth [m]')
cb = plt.colorbar(label=r'Temperature [$\circ$C]')
```



3 Problem 3

3.1 FTCS solution of the wave equation

[12]:
```
%matplotlib inline
from matplotlib import animation
```

[13]:
```
L = 1.0
N = 100
v = 100.0
C = 1.0
sigma = 0.3
d = 0.1
a = L/N
h = 1e-6
tfinal = 2e-2
steps = int(tfinal / h)
ainv = 1.0/a
```

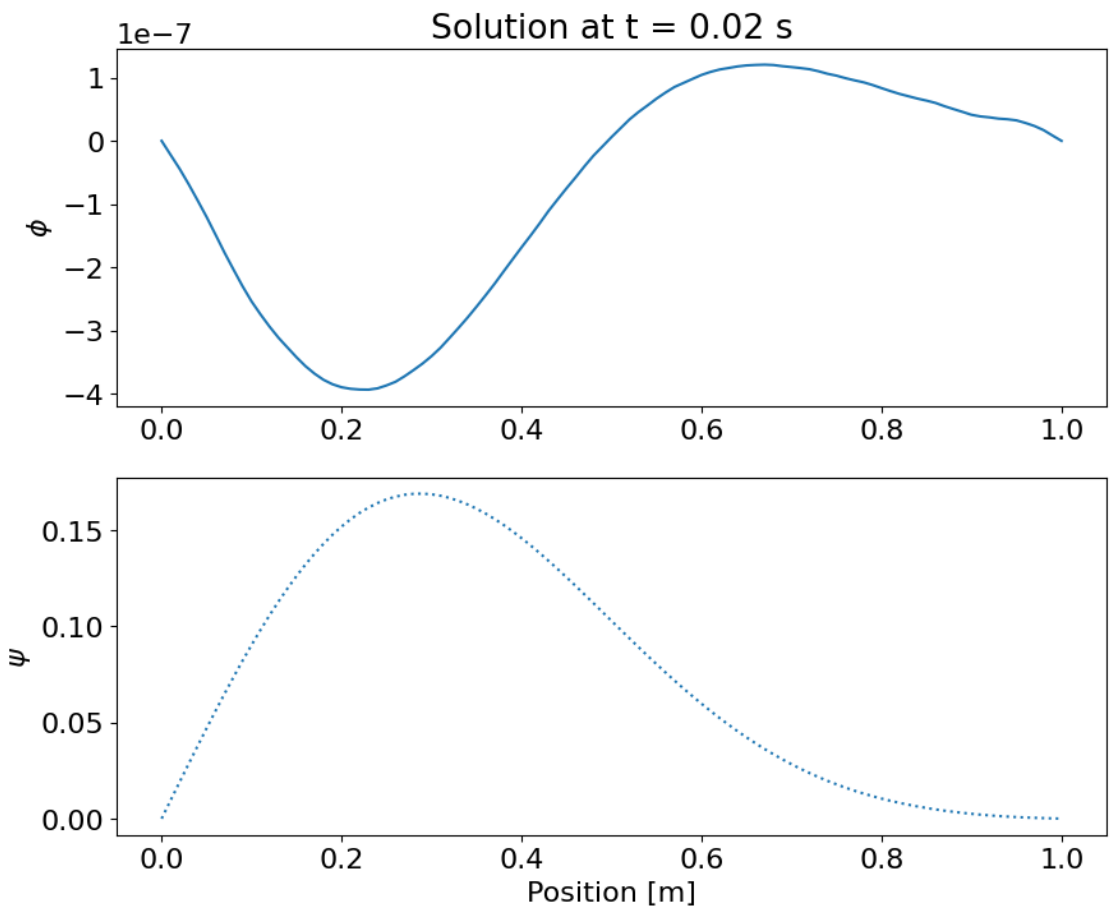
[14]:
```
def f(y):
    result = np.zeros(N+1)
    result[1:N] = (y[0:N-1] + y[2:N+1] - 2*y[1:N]) * v**2 * ainv**2
    return result
```

[15]:
```
# Initialize system
t = 0.0
x = np.linspace(0,L,N+1)
psi = np.zeros(N+1)
dpsi = np.zeros(N+1)
dpsi = C * x * (L - x) / L**2 * np.exp(-(x - d)**2 / (2*sigma**2))
```

3.2 Plot at t = tfinal

[16]:
```
plt.clf()
fig, ax = plt.subplots(2,1)
ax[0].set_title(f'Solution at t = {tfinal} s')
ax[0].set_ylabel(r'$\phi$')
ax[1].set_ylabel(r'$\psi$')
ax[1].set_xlabel('Position [m]')
for n in range(steps):
    psi, dpsi = psi + h*dpsi, dpsi + h*f(psi)
ax[0].plot(x, psi, ls='-')
ax[1].plot(x, dpsi, ls=':')
plt.show()
```

```
<Figure size 1000x800 with 0 Axes>
```



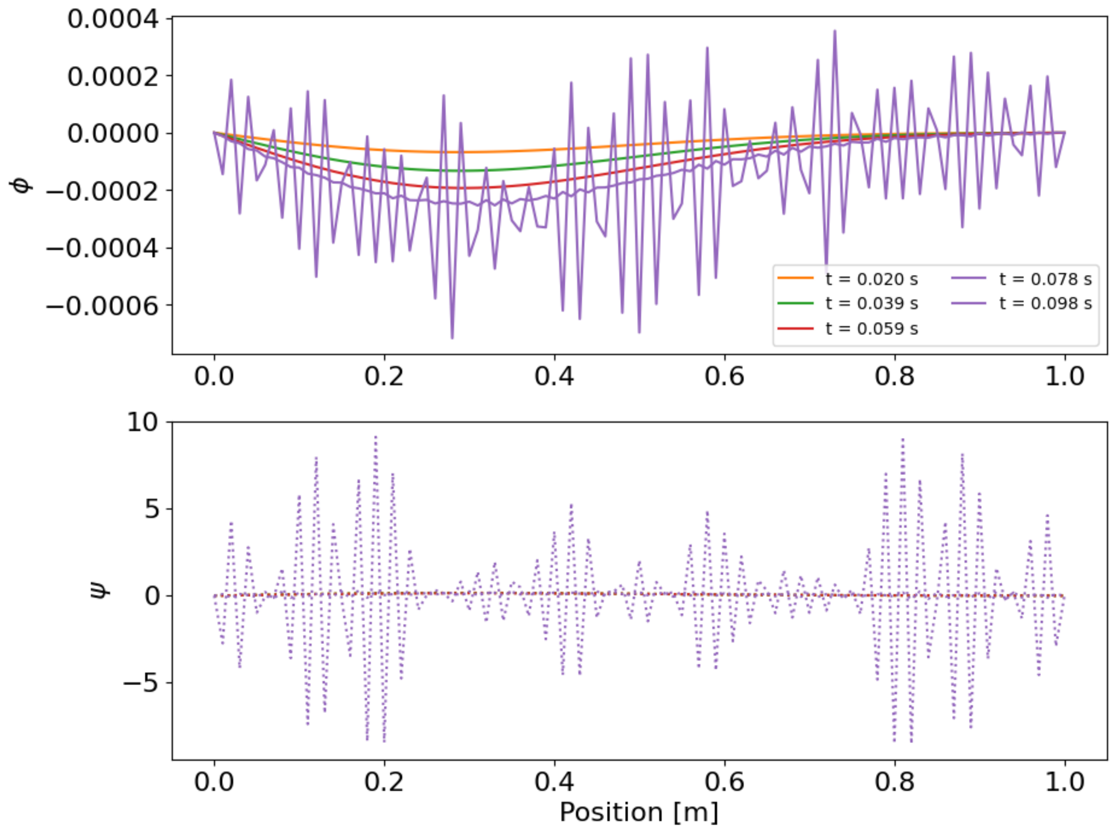
3.3 Plot of when the solution becomes unstable

[17]:
```
# Initialize system
t = 0.0
x = np.linspace(0,L,N+1)
psi = np.zeros(N+1)
dpsi = np.zeros(N+1)
dpsi = C * x * (L - x) / L**2 * np.exp(-(x - d)**2 / (2*sigma**2))
```

[18]:
```
plt.clf()
tfinal = 0.098
steps = int(tfinal/h)
nplots = 5
nev = steps // nplots
fig, ax = plt.subplots(2,1)
ax[0].set_ylabel(r'$\phi$')
ax[1].set_ylabel(r'$\psi$')
ax[1].set_xlabel('Position [m]')
```

```
for n in range(steps):
    if (n % nev == 0 and n > 0) or n == steps-1:
        ax[0].plot(x, psi, c='C%d' % ((n//nev) % 10), ls='-', label = f't =␣
 ↪{n*h:.3f} s')
        ax[1].plot(x, dpsi, c='C%d' % ((n//nev) % 10), ls=':')
    psi, dpsi = psi + h*dpsi, dpsi + h*f(psi)
ax[0].legend(loc='best', ncol=2, fontsize=10)
plt.show()
```

```
<Figure size 1000x800 with 0 Axes>
```



3.4 Just for fun: example with an animation

[ ]:
```
%matplotlib widget
from matplotlib import animation
from IPython import display
```

[ ]:
```
# Initialize system
t = 0.0
x = np.linspace(0,L,N+1)
psi = np.zeros(N+1)
```

```
dpsi = np.zeros(N+1)
#x0,x1 = int(0.2*N), int(0.3*N)
#dpsi[x0:x1] = 1.0
dpsi = C * x * (L - x) / L**2 * np.exp(-(x - d)**2 / (2*sigma**2))
```

[ ]:
```
show_every = 1000

# Initialize plot
fig = plt.figure()
ax = plt.axes(xlim=(0,L), ylim=(-1e-3,1e-3))
line, = ax.plot([], [], lw=2)
ax.set_xlabel('x [m]')
ax.set_ylabel('Displacement [m]')
title = ax.set_title('Time = %g s' % (t))

def initialize_plot():
    global psi, dpsi
    line.set_data([], [])
    title.set_text("")
    return line, title

def update_plot(n):
    global psi, dpsi, h, t
    psi, dpsi = psi + h*dpsi, dpsi + h*f(psi)
    t += h
    if n % show_every == 0:  # Show every Nth timestep
        line.set_data(x, psi)
        title.set_text('cycle = %d, Time = %g s' % (n, t))
        ax.set_ylim(-1e-3, 1e-3)
    return (line, title)

anim = animation.FuncAnimation(fig, update_plot, init_func=initialize_plot,
                               frames=steps, blit=True, interval=1,␣
 ↪repeat=False)
```

[ ]:
```
plt.close()
```