Homework 1

Numerical Integration and ODE/PDEs

This notebook is arranged in cells. Texts are usually written in the markdown cells, and here you can use html tags (make it bold, italic, colored, etc). You can double click on this cell to see the formatting.

The ellipsis (...) are provided where you are expected to write your solution but feel free to change the template (not over much) in case this style is not to your taste.

Hit "Shift-Enter" on a code cell to evaluate it. Double click a Markdown cell to edit.

Link Okpy

In [ ]:
```
from client.api.notebook import Notebook
ok = Notebook('hw1_U2.ok')
_ = ok.auth(inline = True)
```

Imports

In [2]:
```
import numpy as np
#Import in-built functions for different integration techniques
#For reference: https://docs.scipy.org/doc/scipy/reference/integrate.html
from scipy.integrate import quad, fixed_quad, romberg, dblquad
#For plotting
import matplotlib.pyplot as plt
%matplotlib inline
```

Gaussian Quadrature

The manual function for Gaussian quadrature integration.

In [ ]:
```

def gaussxw(N):
    '''Calculate 'N' position and weights for Gaussian quadrature integration
    Returns a tuple of 2 arrays, the first array is the position of points and second
    array is the corresponding weights.
    '''
    a = np.linspace(3, 4*N -1, N)/(4*N+2)
    x = np.cos(np.pi*a + 1/(8*N*N*np.tan(a)))
    eps = 1e-15
    delta = 1.
    #calc roots
    while delta>eps:
        p0 = np.ones(N)
        p1 = np.copy(x)
        for k in range(1, N):
            p0, p1 = p1, ((2*k +1)*x*p1 -k*p0)/(k+1)
        dp = (N+1)*(p0 -x*p1)/(1-x**2)
        dx = p1/dp
        x -= dx
        delta = max(abs(dx))

    #calc weights
    w = 2*(N+1)**2/(N*N*(1 - x**2)*dp**2)

    return x, w

def gaussxwab(N, a, b):
    '''Calculate 'N' position and weights for Gaussian quadrature integration
    between 'a' and 'b'
    Returns a tuple of 2 arrays, the first array is the position of points and second
    array is the corresponding weights.
    '''
    x, w = gaussxw(N)
    return 0.5*(b-a)*x + 0.5*(b+a), 0.5*(b-a)*w

```

Problem 1 - Harmonic Oscillator

The total energy of a harmonic oscillator is given by

$$E = \frac{1}{2}m\left(\frac{dx}{dt}\right)^2 + V(x)$$

Assuming that the potential $V(x)$ is symmetric about $x = 0$ and the amplitude of the oscillator is a. Then the equation for the time period is given by

$$T = \sqrt{8m}\int_0^a \frac{dx}{\sqrt{V(a) - V(x)}}$$

Q1. Suppose the potential is $V(x) = x^4$ and mass of the particle $m = 1$, write a function that calculates the period for a given amplitude.

In [ ]:
```
def V(x):
    'Potential'

    return ...

def timep(x, a):
    'Define the function that needs to be integrated (integrand) to calculate time period'

    return ...

```

Q2. Let $a = 2$. Use inbuilt 'fixed_quad' (https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.integrate.fixed_quad.html (https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.integrate.fixed_quad.html)) function to calculate the time period for different values of 'N' (number of integration points). Calculate the error in the integral by estimating the difference for 'N' & '2N'. Approximately, at what 'N' is the absolute error less than 10^{-4} for 'a = 2'?

Q3. Use inbuilt 'quad' (https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.quad.html (https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.quad.html)) function that returns an error estimate and compare your answer for 'a = 2' (quad uses a more advanced integration technique)

In [ ]:
```
a = ...

n = ...
tquadn = ...
tquadn2 = ...
print('Using fixed_quad: ')
print('\nFor n = %d, the time period is %0.3f, with error = %0.3e'%(n, tquadn, abs(tquadn2 - tquadn)))

tquad = ...
print('Using quad: ')
print('Inbuilt Gaussian Quadrature gives time period = ', tquad[0], ' with error = ', tquad[1])
```

Use the inbuilt romberg function (https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.romberg.html (https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.romberg.html)) to use Romberg integration.

Q4. A simplistic usage with romberg(func, 0, a), where a is the amplitude, will probably give error or 'nan'. Why?

...

Assume that we can tolerate the uncertainitiy of 10^{-5} in the position.

Q5. Show and output of 'keyword' show = True for 'a = 2'. Use this to estimate error for divmax = 10. Show your calculation and compare it with the python warning.

In [ ]:
```
tromb = romberg(...)
```

Q6. Change divmax to change the number of divisions. How does the accuracy change on going from 10 to 15 divisions.

In [4]:
```
...
```

Q7. Use the function to make a graph of the period for amplitude ranging from a=0 to a=2.

In [ ]:
```
...

#Draw Figure here
plt.plot(..., ...)
plt.xlabel('Amplitude')
plt.ylabel('Time Period')
plt.show()
```

Problem 2 - Black Body Radiation

The total rate at which energy is radiated by a black body per unit area over all frequencies is

$$W = \frac{2\pi k_B^4 T^4}{c^2 h^3} \int_0^\infty \frac{x^3}{e^x - 1} dx$$

Q1. Write a function to to evaluate the integral in this expression. You will need to change the variables to go from an infinite range to a finite range. What is the change of variable and new functional form?

Hint

The variable to go from range 0 to ∞ to a finite range of is

$$z = \frac{x}{1+x}$$

or equivalently

$$x = \frac{z}{1-z}$$

In [ ]:
```
#Constants
k = 1.38064852e-23
h = 6.626e-34
pi= np.pi
c = 3e8
hb = h /2/pi
prefactor = k**4/c**2/hb**3/4/pi**2
#True value
stfconst = 5.670367e-8

def blackbody_var(z):
    'Blackbody spectrum after change of variables'

    return ...
```

According to Stefan's law, the total energy given off by a black-body per unit area per second is given by

$$W = \sigma T^4$$

Q2. Use the integral to calculate the value of Stefan Boltzmann constant σ. Use 'fixed_quad' function to do the integral. How accurate you think the answer is?

In [ ]:
```
...
```

Q3. Inbuilt 'quad' function can support an infinite range for integration. Write another function to do the integration from 0 to ∞ and compare your answer.

In [ ]:
```
def blackbody(x):
    'Blackbody Spectrum (without the change of variable)'
    return ...

...
```

Problem 3 - Gravitational Pull of Uniform Sheet

The gravitational force due to a plate felt by a point mass of 1 kg a distnace z from the center of the square in the direction perpendicular to the sheet is given by

$$F_z = G\sigma z \int\int_{-L/2}^{L/2} \frac{dxdy}{(x^2+y^2+z^2)^{3/2}}$$

where $G = 6.674 \times 10^{-11} m^3 kg^{-1} s^{-2}$ and σ is the mass per unit area.

Q1. Write a program to calcualte and plot the force as a function of z from $z = 0$ to $z = 10$ for a sheet of 10 metric tonnes and the sheet is $10\ m$ on side. Use Gaussian quadrature for the double integral. Though there is a 'dblquad' routine in python, we will make use of the manual functions defined above ("gaussxwab"). Study how the number of integration points 'N' affects the integral here. (Try N = 500 and N = 1000 and compare results: i.e. plot F_z vs. z)

In [ ]:
```
def force(x, y, z):
    'Write the functional form of force here'
    return ...

#Factors
G = ...
M = 1e4
L = 10.

#points in z direction
zz = np.logspace(-2, 1, 100)
f, f2 = np.zeros_like(zz), np.zeros_like(zz)

#Number of points for the integral defining
#points in x,y direction

N1 = ...
xx1, w1 = gaussxwab(N1, ..., ...)

for foo in range(zz.size):
    z = zz[foo]
    ...
    f[foo] = ...

N2 = ...
...

for foo in range(zz.size):
    ...
    f2[foo] =

```

Hint:

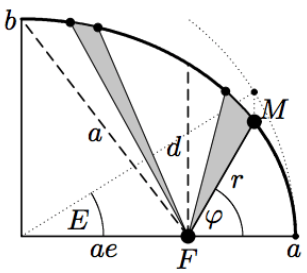
The loop is supposed to calculate the double integral.
To fill in the for loop, study what is returned by the 'gaussxwab' function and how do you use it to evaluate the integral. The easiest thing that can possibly be done is write a triple for loop. However for loops are quite slow in python. You should be able to reduce it to a double loop by using inbuilt functions on numpy array. Take inspiration from here https://docs.scipy.org/doc/numpy/reference/routines.math.html (https://docs.scipy.org/doc/numpy/reference/routines.math.html)
The next way to avoid loops in python is use broadcasting (https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html) and/or use of numpy.einsum. Though you are not required to use it here, its nevertheless a handy thing to know about and this problem is one of the simplest cases where you can use it. You will not need to add any new loops inside the one given (though you may need to declare new variables), and it should be faster than both 2 and 3 for loops.

In [ ]:
```
#Make plot
fig, ax = plt.subplots(1, 1)
ax.plot(..., label = ...)
ax.plot(...)
ax.plot(...)
ax.legend(loc = 0)
ax.set_xlabel('z')
ax.set_ylabel('Force')
ax.set_xscale('log')
ax.set_title("Force due to a sheet")
plt.show()
```

Problem 4 - Planetary Orbit Integration

One of the great achievements in the history of science was the discovery of the laws of J. Kepler, based on many precise measurements of the positions of Mars by Tycho Brahe and himself. The planets move in elliptic orbits with the sun at one of the foci (Kepler's first law).

Newton (Principia 1687) then explained this motion by his general law of gravitational attraction (proportional to $1/r^2$) and the relation between forces and acceleration. This then opened the way for treating arbitrary celestial motions by solving differential equations.



Consider the following two-body problem, wherein a single planet orbits around a large star. Stellar mass is much larger than planetary mass, so we choose the star as the center of our coordinate system. Now, consider the planet's two-dimensional elliptical orbit around the star. The position of the planet is given by the coordinates $q = (q_1, q_2)$, with the planet's velocity given by $p = \dot{q}$.

Newton's laws, with a suitable normalization, yield the following ordinary differential equations:

$$\ddot{q_1} = -\frac{q_1}{(q_1^2+q_2^2)^{3/2}}, \quad \ddot{q_2} = -\frac{q_2}{(q_1^2+q_2^2)^{3/2}}.$$

This is equivalent to a Hamiltonian system with the Hamiltonian:

$$H(p,q) = \frac{1}{2}(p_1^2+p_2^2) - \frac{1}{\sqrt{q_1^2+q_2^2}}$$

$$p_i = \dot{q_i}$$

We will consider the initial position and velocity of the planet to be:

$$q_1(0) = 1 - e, \quad q_2(0) = 0, \quad \dot{q_1}(0) = 0, \quad \dot{q_2}(0) = \sqrt{\frac{1+e}{1-e}}$$

Now determine q as a function of time t.

Q1. Using 400000 steps, use the explicit Euler method (Let $f(q) = \frac{dq}{dt}$. Then, $q(t + \Delta t) = \Delta t \cdot f(q)$ for small Δt) and plot the orbit of the planet. Assume $e = 0.6$ and integrate to a final time of $T_f = 200$.

Hint:

$$\begin{aligned} q_{n+1} &= q_n + \Delta t \cdot \dot{q_n} \\ \dot{q}_{n+1} &= p_{n+1} = p_n + \Delta t \cdot \dot{p_n} \end{aligned}$$

In [ ]:
```
...
```

Q2. Using 400000 steps, use the symplectic Euler method

$$\begin{aligned} p_{n+1} &= p_n - \Delta t H_q(p_{n+1}, q_n) \\ q_{n+1} &= q_n + \Delta t H_p(p_{n+1}, q_n) \end{aligned}$$

or

$$\begin{aligned} q_{n+1} &= q_n + \Delta t H_p(p_n, q_{n+1}) \\ p_{n+1} &= p_n - \Delta t H_p(p_n, q_{n+1}) \end{aligned}$$

where H_p and H_q denote the column vectors of partial derivatives of the Hamiltonian with respect to p and q, respectively. i.e. $H_{p_1} = p_1, H_{q_1} = \frac{q_1}{(q_1^2+q_2^2)^{3/2}}, H_{p_2} = p_2, H_{q_2} = \frac{q_2}{(q_1^2+q_2^2)^{3/2}}$.

Again plot the orbit of the planet. Compare your results in Q1 and Q2 by plotting both solutions in the same figure.

In [ ]:
```
...
```

To Submit

Execute the following cell to submit. If you make changes, execute the cell again to resubmit the final copy of the notebook, they do not get updated automatically.
We recommend that all the above cells should be executed (their output visible) in the notebook at the time of submission.
Only the final submission before the deadline will be graded.

In [ ]:
```
_ = ok.submit()

```

In [ ]:
```

```