Physics 129L: Problem Set 5

Problem Set Submission Guideline

A) Github Submissions We will use GitHub for problem set submissions. To access the problem set, please **fork** and **clone** the **forked** repository to your local virtual machine. **Please complete the problem set in this forked directory.** Submit **a pull request** for merging into the main branch before the problem set due date.

B) .tar.gz File compression and submission on Github For each problem set, you are asked to submit the compressed version of the problem set to GitHub via git operation. Here is a step-by-step guideline:

1. Use the **tar** command to compress the problem set directory into a **single** ".tar.gz" file.
2. Obtain the sha256sum by running "sha256sum P2.tar.gz".
3. Echo the **full sha256sum** to a text file named "sha25sum_problem_set.txt".
4. Initialize a git repository named "Archive_P# (#: problem set number) on your local machine, and move both the "tar.gz" file and the "sha25sum_problem_set.txt" file to the repository.
5. Create an empty **public** directory under the **same name** in **your own GitHub account**.
6. **Push** this local repository to the remote repository.

Imports and built-in functions

In [ ]:
```
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
```

Problem 1: Quadrature

Let's looked at different quadrature methods and rules.

Midpoint rule

$$\int_a^b f(x)\,dx \approx (b-a)\cdot f\left(\frac{a+b}{2}\right)$$

Trapezoidal Rule

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}[f(a)+f(b)]$$

Simpson's Rule

$$\int_a^b f(x)\,dx \approx \frac{b-a}{3}\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right]$$

A)

Define a class for all quadrature techniques.

In [ ]:
```
class Quad_:
    def __init__(self,fuc,N,a,b):
        '''The class Quad tkaes four inputs:
         fuc: a function input,
         N: number of grid points,
         a,b: left and right points'''
        pass
```

B)

Write the above three quadrature rules as class methods **without any pre-defined function from any package** 1) Midpoint rule 2) Trapezoidal Rule 3) Simpson's Rule. You need to figure out what additional inputs each rule needs.

In [ ]:
```
# Add the following to the class in part A).
def mid_quad(self,...):
    pass
def trapz_quad(self,...):
    pass
def simpson_quad(self,...):
    pass
```

Gauss-Legendre Quadrature

$$\int_{-1}^{1} f(x)\,dx \approx \sum_{i=1}^{N} w_i \cdot f(x_i)$$

C)

Since the range is from a to b, let's make a variable change,

$$x_i \rightarrow \frac{b-a}{2}x_i + \frac{a+b}{2}$$

Calculate the above condition **analytically**,
''**write you answer below**''

Steps:

'' **write you answer above**''

You should get something like this:

$$\int_a^b f(x)dx = \frac{b-a}{2}\sum_i w_i f(\frac{b-a}{2}x_i + \frac{a+b}{2})$$

Legendre polynomials

To find the position and weights of an order M Gauss-Legendre Quadrature, we need to calculate the roots of an order-M Legendre polynomial,

$$(1-x^2)\frac{d^2P_M(x)}{dx^2} - 2x\frac{dP_M(x)}{dx} + M(M+1)P_M(x) = 0,$$

and the solutions are given by the following:

$$P_M(x) = \frac{1}{2^n n!}\frac{d^M}{dx^M}\left[(x^2-1)^M\right]$$

D)

Write a child class named Gauss_Quad_ inherits methods from its parent class Quad_. This class takes an additional input, called order. Define a new method that outputs an order-M Legendre polynomial. Plot the following Legendre polynomials: M=[1,2,3,4,5].

In [ ]:
```
class Gauss_Quad_:
    def __init__(self,order):
        '''The class take an additional input, called order.
        make sure you look into super(). '''
        pass
    def legendre_poly(self,x):
        # range from -1 to 1
        pass
```

Newton's Method

The positions of an order M Gauss-Legendre Quadrature are calculated by finding roots of an order-M Legendre polynomial,

$$P_M(x) = 0.$$

To numerically find those roots (M of them), we can use the Newton's method:

$$x_{n+1} = x_n - \frac{P_M(x_n)}{P'_M(x_n)}.$$

You should be careful on the initial guess.
The weights w_i for Gauss-Legendre Quadrature are calculated as:

$$w_i = \frac{2}{(1-x_i^2)[P'_M(x_i)]^2}$$

where $P'_M(x_i)$ is the derivative of the Legendre polynomial of degree M evaluated at each root x_i.

E)

Calculate 'M' position and weights for Gaussian quadrature integration between 'a' and 'b' with the Newton's method. Returns a tuple of 2 arrays, the first array is the position of points and second array is the corresponding weights. Output an text file that contains the roots and weights for M=[1,2,3,4,5].

In [ ]:
```
# Add the following to the class in part D).
def newton_method_root(self,...):

    return root,weights
```

F)

Using the the following information, https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.roots_legendre.html (https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.roots_legendre.html) , calculate the roots and weights for M=[1,2,3,4,5]. Do they agree with what you calculate in E)?

In [ ]:
```
# Add the following to the class in part D).
from scipy.special import roots_legendre
def scipy_method_root(self,...):

    return root,weights
```

G)

Based on the roots and weights, write a method that output the Gauss-Legendre Quadrature.

In [ ]:
```
# Add the following to the class in part D).
def gauss_quad(self,...):
    pass
```

Quadrature on test functions

In the following question, we want to Use the above four methods, mid_quad, trapz_quad, simpson_quad, gauss_quad, to numerically calculate the quarature on two test functions ($k \geq 0$)

A general k-th order polynomial with a quadrature,

$$I_{\text{true}}^{\text{T=A}} = \int_a^b dx x^k = \frac{1}{k+1}(b^{k+1} - a^{k+1})$$

A Fermi–Dirac distribution with a quadrature,

$$I_{\text{true}}^{\text{T=B}} = \int_a^b dx \frac{1}{1+e^{-kx}} = \frac{1}{k}(\log(e^{kb}+1) - \log(e^{ka}+1))$$

Let's define the following: for each order k and N, we have the quadrature value and the relative error,

1. mid_quad: $M(k, N)$,

   $\Delta M(k, N) = 2\frac{I_{\text{true}} - M(k,N)}{I_{\text{true}} + M(k,N)}$,

2. trapz_quad: $T(k, N)$,

   $\Delta T(k, N) = 2\frac{I_{\text{true}} - T(k,N)}{I_{\text{true}} + T(k,N)}$,

3. simpson_quad: $S(k, N)$,

   $\Delta S(k, N) = 2\frac{I_{\text{true}} - S(k,N)}{I_{\text{true}} + S(k,N)}$,

4. gauss_quad: $G(k, N)$,

   $\Delta G(k, N) = 2\frac{I_{\text{true}} - G(k,N)}{I_{\text{true}} + G(k,N)}$,

Let's make the following heatmap for each quadrature method above, 1-4), over the range $[0, 1]$:

y-axis: k from 0 to 10

x-axis: N from 10 to 10^5

value: relative error

H) Polynomial

In [ ]:
```
'''--------------------Write your code below this line-----------------------------''

'''--------------------Write your code above this line-----------------------------''
```

H) Fermi–Dirac

In [3]:
```
'''--------------------Write your code below this line-----------------------------''

'''--------------------Write your code above this line-----------------------------''
```

Out[3]:
```
'--------------------Write your code above this line-----------------------------'
```

Harmonic Oscillator

This question is adapted from Prof.Uros Seljak (Berkeley) problem set 1 question 1. For more information, please visit [https://phy151-ucb.github.io/seljak-phy151-fall-2018/#course-syllabus (https://phy151-ucb.github.io/seljak-phy151-fall-2018/#course-syllabus)](https://phy151-ucb.github.io/seljak-phy151-fall-2018/#course-syllabus)

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

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

A)

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 per

    return ...
```

B)

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'?

In [ ]:
```
'''--------------------Write your code below this line-----------------------------''

'''--------------------Write your code above this line-----------------------------''
```

C)

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 [ ]:
```
'''--------------------Write your code below this line-----------------------------''

'''--------------------Write your code above this line-----------------------------''
```

D)

Calculate the time period by using 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)) for Romberg integration. A simplistic usage with romberg(func, 0, a), where a is the amplitude, will probably give error or 'nan'. Why?

In [4]:
```
'''--------------------Write your code below this line-----------------------------''

'''--------------------Write your code above this line-----------------------------''
```

Out[4]:
```
'--------------------Write your code above this line-----------------------------'
```

E)

Assume that we can tolerate the uncertainitiy of 10^{-5} in the position. Show and output of 'keyword' show = True for 'a = 2'. Use this to estimate error for divmax = 10.

In [5]:
```
'''--------------------Write your code below this line-----------------------------''

'''--------------------Write your code above this line-----------------------------''
```

Out[5]: '--------------------Write your code above this line-----------------------------'

F)

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

In [6]:
```
'''--------------------Write your code below this line-----------------------------''

'''--------------------Write your code above this line-----------------------------''
```

Out[6]: '--------------------Write your code above this line-----------------------------'

G)

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

In [7]:
```
'''--------------------Write your code below this line-----------------------------''

'''--------------------Write your code above this line-----------------------------''
```

Out[7]: '--------------------Write your code above this line-----------------------------'

Black Body Radiation

This question is adapted from Prof.Uros Seljak (Berkeley) problem set 1 question 2. For more information, please visit [https://phy151-ucb.github.io/seljak-phy151-fall-2018/#course-syllabus (https://phy151-ucb.github.io/seljak-phy151-fall-2018/#course-syllabus)](https://phy151-ucb.github.io/seljak-phy151-fall-2018/#course-syllabus)

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

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

A)

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

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

or equivalently

In [8]:
```
#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 ...
```

Out[8]: '--------------------Write your code above this line-----------------------------'

B)

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

$$W = \sigma T^4$$

. Use the integral to calculate the value of Stefan Boltzmann constant σ. Use 'fixed_quad' function to do the integral.

In [ ]:
```
'''--------------------Write your code below this line-----------------------------''

'''--------------------Write your code above this line-----------------------------''
```

C)

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 [9]:
```
'''--------------------Write your code below this line-----------------------------''

'''--------------------Write your code above this line-----------------------------''
```

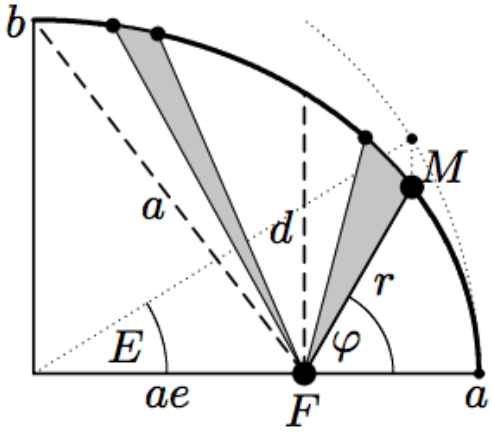
Out[9]: '--------------------Write your code above this line-----------------------------'

Planetary Orbit Integration

This question is adapted from Prof.Uros Seljak (Berkeley) problem set 1 question 4. For more information, please visit [https://phy151-ucb.github.io/seljak-phy151-fall-2018/#course-syllabus (https://phy151-ucb.github.io/seljak-phy151-fall-2018/#course-syllabus)](https://phy151-ucb.github.io/seljak-phy151-fall-2018/#course-syllabus)

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.

A)

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 $T_f = 200$.

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

In [ ]:
```
'''--------------------Write your code below this line-----------------------------''

'''--------------------Write your code above this line-----------------------------''
```

B)

Using 400000 steps, use the symplectic Euler method.

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

or

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

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 A) and B) by plotting both solutions in the same figure.

In [10]:
```
'''--------------------Write your code below this line------------------------------''

'''--------------------Write your code above this line------------------------------''
```

Out[10]:
```
'--------------------Write your code above this line------------------------------'
```

In [ ]:
```

```