# Homework 1

Note: Since this is the first homework, detailed template (imports, function declarations, comments etc) are provided. Hence, if you are new to python, you are highly encouaged to play with things (for instance inbuilt functions used in the template) beyond what is asked in the homework.

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.

## Link Okpy

In [ ]:

```
from client.api.notebook import Notebook
ok = Notebook('hw1.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
```

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 s
econd
    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 s
econd
    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

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

1. 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. Use it to calculate time period of 'a = 2'. Use this amplitude for next 2 parts.
2. Use inbuilt fixed_quad function or the manual 'gaussxwab' function above 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 $10^{-6}$ for 'a = 2'
   - Use inbuilt 'quad' function that returns an error estimate and compare your answer for 'a = 2' (quad uses a more advanced integration technique)
3. Use the inbuilt romberg function to use Romberg integration.
   - Note that a simplistic usage with romberg(func, 0, a) will probably give error or 'nan'. Why?
   - 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. Show your calculation and compare it with the python warning.
   - Change divmax to change the number of divisions. How does the accuracy change on going from 10 to 15 divisions.
4. Use the function to make a graph of the period for amplitude rangeing from a=0 to a=2.

In [ ]:

```
#Part 1

def V(x):
    'Potential'

    return ...

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

    return ...
```

In [ ]:

```
#Part 2
a = 2

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

n =
tquadn = fixed_quad(..., args = (a, ), n = ...)[0]
tquadn2 = fixed_quad(..., args = (a, ), n = ...)[0]
print('\nFor n = %d, the time period is %0.3f, with error = %0.3e'%(n, tquadn, a
bs(tquadn2 - tquadn)))
```

***Part 3***

The romberg gives error on simplistic call of the romberg(func, 0, a) where a is the amplitude because <...>

In [ ]:

```
#Part 3
#The position tolerance is 10**-5

tromb = romberg(timep, 0, a-10**-5, args = (a,), divmax=10,show=True)
```

In [4]:

```
#Calculate Error
...
```

In [5]:

```
#When divmax = 15
...
```

In [6]:

```
#The new error is
...
```

In [ ]:

```
#Part 4
#Calculate points here
#Create 'a' values in the given range
avals = np.linspace(1e-1, 2, 100)
tvals = np.zeros_like(avals)

#signature for the manual function above
#Feel free to replace it with call to inbuilt function instead
N =                     #Number of GQ points
for foo in range(avals.size):
    a = avals[foo]
    x, w = gaussxwab(N, 0, a)
    t = timep(x, a)
    tvals[foo] = ...



#Draw Figure here
fig, ax = plt.subplots(1, 1)
ax.plot(..., ...)
ax.set_xlabel('Amplitude')
ax.set_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

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

1. 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?
2. 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 $\sigma$. Use 'fixed_quad' function to do the integral. How accurate you think the answer is?
3. Inbuilt 'quad' function can support an infinite range for integration. Write another function to do the integration from 0 to $\infty$ and compare your answer.

**Solution**

Part 1

The variable to go from range 0 to $\infty$ to a finite range of [...] is - ...

The change of the function under this change of variables is - ...

In [ ]:

```
#Part 1-2
def blackbody_var(z):
    'Blackbody spectrum after change of variables'

    return ...

#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
```

In [ ]:

```
#Part 2

#Gaussian Quadrature
N = ...
...fixed_quad(...)
...
...
#Calculate value
stefan = ...
...
error ...

print("Evaluated value = ", stefan, " with error = ",error)

print("Evaluated value = ", stefan, "\nTrue value = ",stfconst, "\nResidual from
 truth = ", (stfconst-stefan)/stfconst)
```

In [ ]:

```
def blackbody(x):
    'Blackbody Spectrum'
    return ...

stefan2 = quad(..., 0, np.inf)
print(...)
```

**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 $\sigma$ is the mass per unit area.

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 - 'gaussxwab' and 'gaussx' defined above.

- Study how the number of integration points 'N' affects the integral here.
- Based on the above, what do you expect the correct curve to look like. Why do you physically expect that to be the case? Try to make connection with what you might have learnt in electromagnetism.

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()
```

**Answer:**

Write your observation about different 'N' values here. What do you expect the correct curve to look like. Why do you physically expect that to be the case? Try to make connection with what you might have learnt in electromagnetism.

---

## 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 [ ]:

```
```