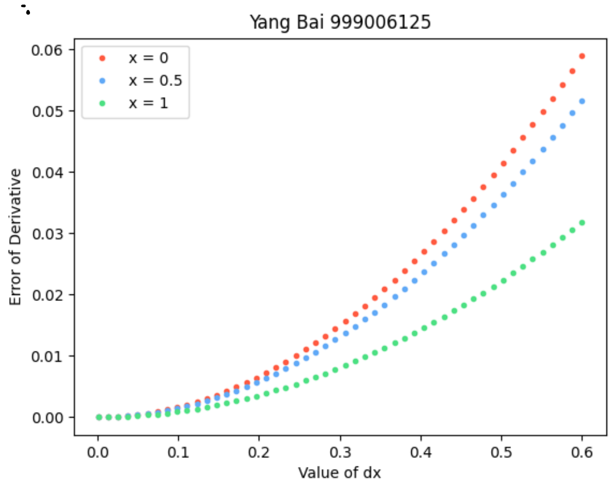
Q1

Figure :



As the spacing between sample points getting bigger, the Error increase.

Explaination :

Equation :

$$f'(x) \approx \frac{f(x+dx) - f(x-dx)}{2dx}$$

And the Error is :

$$R = \frac{1}{6} (dx)^2 f''(x) \quad \propto (dx)^2 \propto dx .$$

That Explain When $dx \uparrow \Rightarrow R \uparrow E$

CODES :

```
import numpy as np
import scipy.misc as mi
import scipy.integrate as ite
import matplotlib.pyplot as plt

#Q1
def f(x):
    return np.sin(x)

True_derivative = np.cos(0),np.cos(0.5),np.cos(1)

_dx = np.linspace(0.001, 0.6)

plt.title("Yang Bai 999006125")
plt.xlabel("Value of dx")
plt.ylabel("Error of Derivative")
plt.plot(_dx, True_derivative[0] - mi.derivative(f, 0, _dx), '.', color = '#FF5A40', label = 'x = 0')
plt.plot(_dx, True_derivative[1] - mi.derivative(f, 0.5, _dx), '.', color = '#60A9F7', label = 'x = 0.5')
plt.plot(_dx, True_derivative[2] - mi.derivative(f, 1, _dx), '.', color = '#4CE082', label = 'x = 1')
plt.legend()
plt.show()
```

Q2

Result:

```
4th order Gaussian : Result: 1.1311235558246302 Error : 1.8111317927828388
--------------
 4 order Newton-Cotes: Result: 3.244478294 Error : 3.02223e-01
 6 order Newton-Cotes: Result: 6.828016887 Error : 3.88576e+00
 8 order Newton-Cotes: Result: -1.051610482 Error : 3.99387e+00
10 order Newton-Cotes: Result: 11.896779084 Error : 8.95452e+00
```

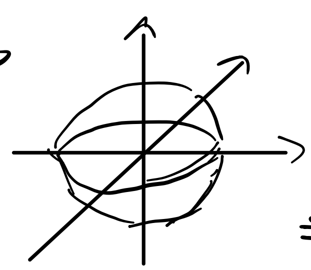
Compare to k-point Gaussian Quadrature. Error become bigger when order increase. It's because our Function $\frac{1}{1+x^2}$ is basically a polynomial with High degree. So when we try to use polynomial interpolation on it, It will Cause Runge's phenomenon. Making the Error become Bigger.

CODES:

```python
import numpy as np
import scipy.misc as mi
import scipy.integrate as ite
import matplotlib.pyplot as plt

#Q2
func = lambda x0: 1/(1+x0**2)
t = ite.fixed_quad(func, -10, 10, n=4)
print("4th order Gaussian : Result:", t[0],"Error :", abs(t[0] - 2*np.arctan(10)))
print("--------------")
for N in [4,6,8,10]:
    x=np.linspace(-10,10,N+1)
    weight,B=ite.newton_cotes(N,1)
    dx=20/N
    quad=dx*np.sum(weight*(1/(1+x**2)))
    error=abs(quad-2*np.arctan(10))
    print('{:2d} order Newton-Cotes: Result: {:10.9f} Error : {:.5e}'.format(N,quad,error))
```

Q3



$$\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}=1$$

$$dV = z(x,y)\cdot dxdy$$

$$\Rightarrow V = 2c\cdot\iint_D z(x,y)\,dxdy = \frac{4}{3}\pi abc$$

Multivarible quadrature $\Rightarrow \int_{-a}^{a}\int_{-b\sqrt{1-\frac{x^2}{a^2}}}^{b\sqrt{1-\frac{x^2}{a^2}}} c\sqrt{1-\frac{x^2}{a^2}-\frac{y^2}{b^2}}\,dx\,dy$

```
Result from nqaud : 251.32741228718373
Error is  2.842170943040407e-13
---------------------
Result from simpson : 251.3258379892207
Error is  0.0015742979627475506
Spacing is  0.0034944670937682005
```

From the Result, we can obtain that the error of Multivarible Quadrature ($\sim 10^{-13}$) is much smaller than Simpson ($\sim 10^{-3}$ with $N=1717$)

So Multivarible Quadrature is more accurate than Simpson when calculating Multiple Integrals

CODES :

```python
import numpy as np
import scipy.misc as mi
import scipy.integrate as ite
import matplotlib.pyplot as plt

#Q3
a, b, c = 3, 4, 5
volume = (4/3)*np.pi*a*b*c
def f1(y,x):
    return 2*c*np.sqrt(1 - ((x**2)/a**2) - ((y**2)/b**2))
def f2(x):
    return [(-1) * b * np.sqrt(1 - ((x**2)/a**2)), b * np.sqrt(1 - ((x**2)/a**2))]
nq = ite.nquad(f1, [f2, [-a , a]])
error = np.abs(nq[0] - volume)
print("Result from nqaud :",nq[0])
print("Error is ", error)
print("---------------------")

N = 1717
x = np.linspace(-a, a, N+1)
def f3(x):
    return b*np.sqrt(1-((x**2)/a**2))
ylist = np.zeros(N+1)
for i in range(len(x)):
    y = np.linspace(-f3(x[i]), f3(x[i]), int(2 * f3(x[i]) / (2*a/N))+1)
    z = 2*c*np.sqrt(abs(1-((x[i]**2)/a**2) - ((y**2)/b**2)))
    ylist[i] = ite.simpson(z,y)
ss = ite.simpson(ylist,x)
ss_error = abs(ss - volume)
print("Result from simpson :",ss)
print("Error is ", ss_error)
print("Spacing is ", 2*a/N)
```