Problem 1:

• Allow ourselves to evaluate function @ 4 points → $(x \pm \delta,\ x \pm 2\delta)$

(a) What should our estimate of the 1^{st} Derivative be?

↳ let's first expand... the derivs

✩ Inspired by Ch. 3 of "An Intro to Comp. Phys"
↳ Tao Pang

$$f(x) = f(x_0) + (x-x_0) f'(x_0) + \frac{(x-x_0)^2}{2!} f''(x_0) + \dots + \frac{(x-x_0)^n}{n!} f^{(n)}(x_0) + \dots$$

(I) $$f(x \pm \delta) = f(x) \pm \delta f'(x) + \frac{\delta^2}{2} f''(x) \pm \frac{\delta^3}{6} f^{(3)}(x) + \frac{\delta^4}{24} f^{(4)}(x) + O(\delta^5)$$

(II) $$f(x \pm 2\delta) = f(x) \pm 2\delta f'(x) + 2\delta^2 f''(x) \pm \frac{8}{6}\delta^3 f^{(3)}(x) + \frac{16}{24}\delta^4 f^{(4)}(x) + O(\delta^5)$$

$$f'(x) = \lim_{dx \to 0} \frac{f(x+dx) - f(x)}{dx}$$

$$f'(x) \simeq \frac{f(x+\delta) - f(x)}{\delta} + O(\delta)$$; for improved accuracy about smooth functions, we must apply a central difference formula...

two points:

$$f(x \pm \delta) = f(x) \pm \delta f'(x) + \frac{\delta^2}{2} f''(x) + O(\delta^3)$$

$$\rightarrow f(x+\delta) - f(x-\delta) = \emptyset + 2\delta f'(x) + O(\delta^3) \quad \Rightarrow \quad f'(x) \simeq \frac{f(x+\delta) - f(x-\delta)}{2\delta} + O(\delta^2)$$

four points:

✩ $$f(x \pm \delta) = f(x) \pm \delta f'(x) + \frac{\delta^2}{2} f''(x) \pm \frac{\delta^3}{6} f^{(3)}(x) + \frac{\delta^4}{24} f^{(4)}(x) + O(\delta^5)$$

$$f(x+\delta) - f(x-\delta) = 2\delta f'(x) + \frac{2}{6}\delta^3 f^{(3)}(x) + O(\delta^5) = 2\delta f'(x) + \frac{\delta^3}{3} f^{(3)}(x) + O(\delta^5)$$

✩✩ $$f(x \pm 2\delta) = f(x) \pm 2\delta f'(x) + 2\delta^2 f''(x) \pm \frac{8}{6}\delta^3 f^{(3)}(x) + \frac{16}{24}\delta^4 f^{(4)}(x) + O(\delta^5)$$

$$f(x+2\delta) - f(x-2\delta) = \left[\cancel{f(x) - f(x)}\right] + \left[2\delta f' + 2\delta f'\right] + \left[\cancel{2\delta^2 f'' - 2\delta^2 f''}\right] + \left[\frac{8}{6}\delta^3 f''' + \frac{8}{6}\delta^3 f'''\right] + \left[\cancel{\frac{16}{24}\delta^4 (f^{(4)} - f^{(4)})}\right] + O(\delta^5)$$

$$f(x+2\delta) - f(x-2\delta) = 4\delta f'(x) + \frac{8}{3}\delta^3 f'''(x) + O(\delta^5)$$

⇝ let's cancel the $f^{(3)}$ terms:

$$\left\{ f(x+\delta) - f(x-\delta) = 2\delta f'(x) + \frac{\delta^3}{3} f^{(3)}(x) + O(\delta^5) \right\} \cdot 8 \quad \rightsquigarrow \text{Multiply by } 8$$

$$8f(x+\delta) - 8f(x-\delta) - 16\delta f'(x) + O(\delta^5) = \frac{8\delta^3}{3} f^{(3)}(x) \quad \rightsquigarrow \text{Now, equate to } \star\star$$

$$8f(x+\delta) - 8f(x-\delta) - 16\delta f'(x) + O(\delta^5) = f(x+2\delta) - f(x-2\delta) - 4\delta f'(x) + O(\delta^5)$$

$$12\delta f'(x) = 8\left[f(x+\delta) - f(x-\delta)\right] - \left[f(x+2\delta) - f(x-2\delta)\right] + O(\delta^5)$$

$$\Rightarrow \boxed{f'(x) \simeq \frac{1}{12\delta}\left[f(x-2\delta) - f(x+2\delta) + 8f(x+\delta) - 8f(x-\delta)\right] + O(\delta^4)}$$

(b) Since the leading order for the error term is $f^{(5)}$

$$\Rightarrow \quad err = \frac{f\epsilon}{\delta} + f^{(5)}\delta^4 \quad \Rightarrow \quad 0 = -\frac{f\epsilon}{\delta^2} + 4f^{(5)}\delta^3$$

$$\Rightarrow \quad \delta = \left(\frac{f\epsilon}{4f^{(5)}}\right)^{1/5}$$

⇝ let $\epsilon = 10^{-15}$

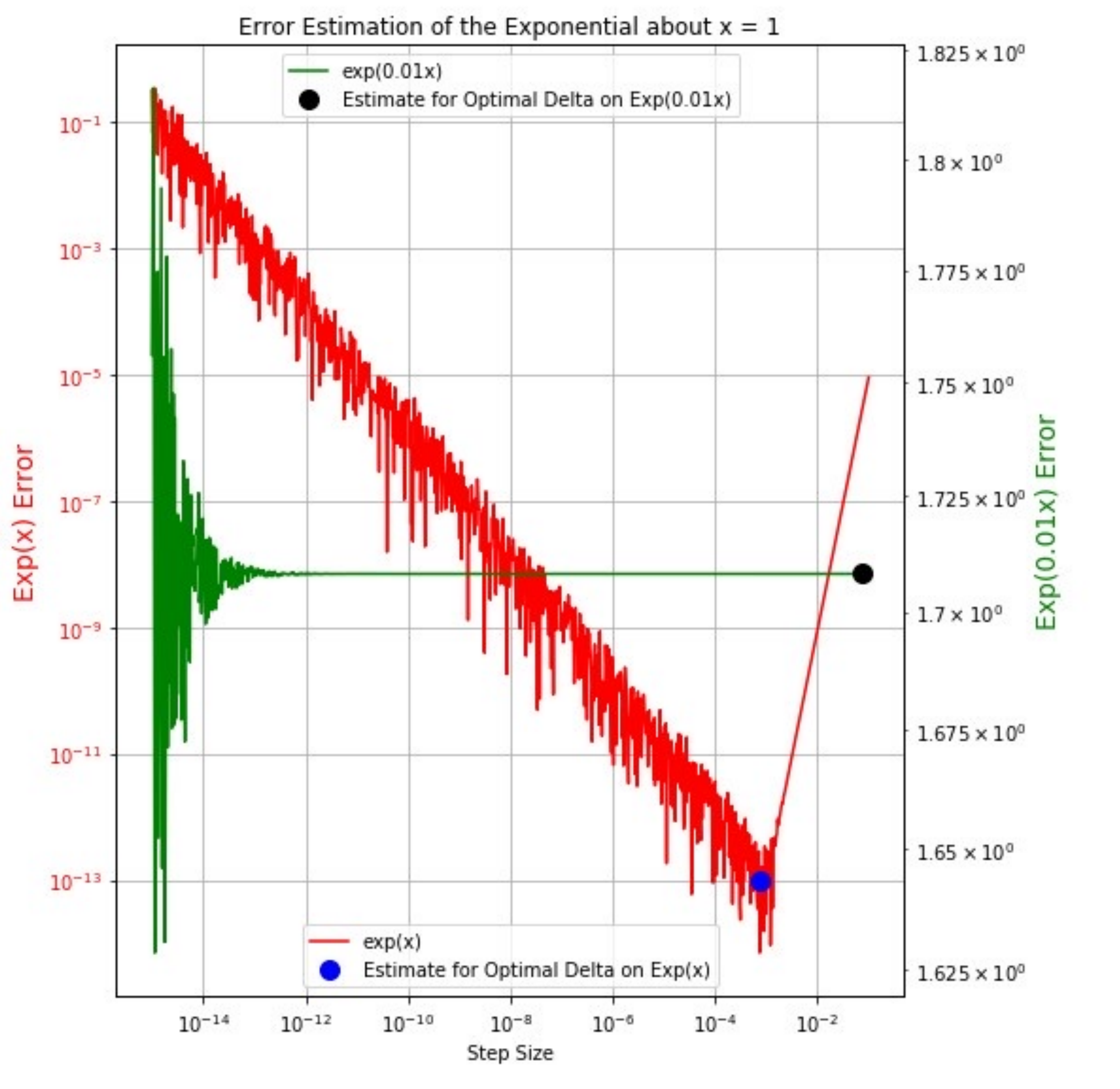
On the $f_1(x) = \exp[0.01x]$:

$$\delta = \left(\frac{e^{0.01x}\,\epsilon}{4(0.01)^5\,e^{0.01x}}\right)^{1/5} = \left(\frac{\epsilon}{4\times10^{-10}}\right)^{1/5} \quad \Rightarrow \delta_0 \approx 0.0758$$

On $f_2(x) = \exp[x]$:

$$\delta = \left(\frac{e^x\,\epsilon}{4\,e^x}\right)^{1/5} = \left(\frac{\epsilon}{4}\right)^{1/5} \quad \Rightarrow \delta_0 \approx 7.58\times10^{-4}$$

```
Error on Exp(0.01x) is  1.7082327736073493 at a delta of  0.0758
Error on Exp(x) Exp is  9.725553695716371e-14 at a delta of  0.000758
```



The response on Exp[0.01x] error is not especially sensible, yet it seems to agree with the prediction... I would expect the green fit to resemble the red one only shifted in step-size.

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 13 13:17:16 2022

@author: Yacine Benkirane
Solution to Q1b of PSET1
Comp Physics at McGill University: PHYS 512
"""

import numpy as np
from matplotlib import pyplot as plt

logdx = np.linspace(-15, -1, 1001)
dx = 10**logdx

func = np.exp
x0 = 1

y00 = func(x0)
y01 = func(x0 + dx)
y02 = func(x0 - dx)
y03 = func(x0 + 2*dx)
y04 = func(x0 - 2*dx)

y10 = func(x0*0.01)
y11 = func(x0*0.01 + dx)
y12 = func(x0*0.01 - dx)
y13 = func(x0*0.01 + 2*dx)
y14 = func(x0*0.01 - 2*dx)

d1_Norm = (y04 - 8*y02 + 8*y01 - y03)/(12*dx)
d1_Stretched = (y14 - 8*y12 + 8*y11 - y13)/(12*dx)


d1_Stretched_Optim = (func(x0*0.01 - 2*(0.0758)) - 8*func(x0*0.01 - (0.0758)) + 8*func(x0*0.01 + (0.0758)) - func(x0*0.01 + 2*(0.0758)))/(12*(0.0758))
d1_Norm_Optim = (func(x0 - 2*0.000758) - 8*func(x0 - (7.58 * 10**(-4))) + 8*func(x0 + (7.58 * 10**(-4))) - func(x0 + 2*(7.58 * 10**(-4))))/(12*(7.58 * 10**(-4)))

print('Error on Exp(0.01x) is ', np.abs(d1_Stretched_Optim - np.exp(x0)), 'at a delta of ', 0.0758)
print('Error on Exp(x) Exp is ', np.abs(d1_Norm_Optim - np.exp(x0)), 'at a delta of ', (7.58 * 10**(-4)))


fig, ax1 = plt.subplots(figsize=(8, 8))
ax2 = ax1.twinx()

ax1.loglog(dx, np.abs(d1_Norm - np.exp(x0)), label = 'exp(x)', color = 'red')
ax2.loglog(dx, np.abs(d1_Stretched - np.exp(x0)), label = 'exp(0.01x)', color = 'green')

ax1.set_xlabel('Step Size')
ax1.set_ylabel('Exp(x) Error', color = 'red', fontsize = 14)
ax1.tick_params(axis="y", labelcolor='red')

ax2.set_ylabel('Exp(0.01x) Error', color = 'green', fontsize = 14)
ax2.tick_params(axis="y", labelcolor='green')

#Estimated on paper
xNorm = [0.000758]
yNorm = [9.725553695716371* 10**(-14)]
xStretch = [(0.0758)]
yStretch = [1.7082327736073493]

ax1.plot(xNorm, yNorm, "o", color="blue", markersize=10, label='Estimate for Optimal Delta on Exp(x)')
ax2.plot(xStretch, yStretch, "o", color="black", markersize=10, label='Estimate for Optimal Delta on Exp(0.01x)')

plt.title('Error Estimation of the Exponential about x = 1')
fig.tight_layout()

ax1.grid(True)
ax1.legend(loc = 'lower center')
ax2.legend(loc='upper center')

plt.savefig('ErrorPlot.png')
plt.show()
```

Problem 2:

Results:

```
Exp(10x) about x=1
dx: 9.999833333172091e-07
Computational Deriv: 220264.65794665305 Analytical Deriv: 220264.65794806703
Computed Error: 6.608049872713116e-05 Analytical Error: 1.4139804989099503e-06
Ratio btwn Comp and Analyt Errors:  46.733670498336565

Exp(10x) about x=0
dx: 9.999833335124805e-07
Computational Deriv: 10.000000000147631 Analytical Deriv: 10
Computed Error: 3.0000500002958922e-09 Analytical Error: 1.4763124056571542e-10
Ratio btwn Comp and Analyt Errors:  20.321240875575203

Exp(x) about x=1
dx: 9.999998102099061e-06
Computational Deriv: 2.7182818284999564 Analytical Deriv: 2.718281828459045
Computed Error: 8.154847033086331e-10 Analytical Error: 4.091127436822717e-11
Ratio btwn Comp and Analyt Errors:  19.93300663207796

Exp(x) about x=0
dx: 9.999998428939069e-06
Computational Deriv: 1.0000000000160862 Analytical Deriv: 1
Computed Error: 3.000000471318357e-10 Analytical Error: 1.6086243448398818e-11
Ratio btwn Comp and Analyt Errors:  18.649478238606225
```

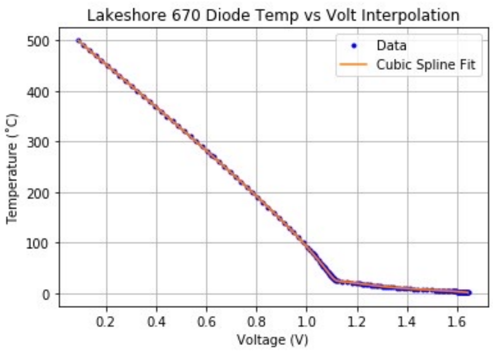
The computed and analytical errors are roughly one order of mag. within each other.

☆ ndiff was thusly tested and works as intended, optimizing dx and calculating reasonable derivative values...

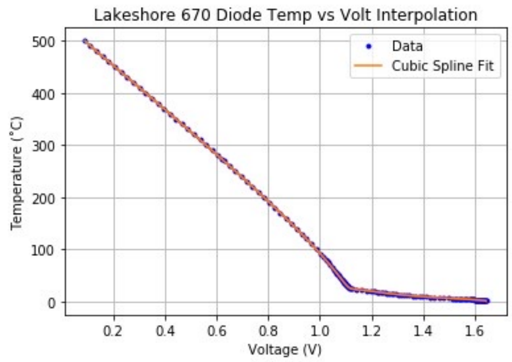
```python
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 13 15:21:25 2022

@author: Yacine Benkirane
Solution to Q2 of PSET1
Comp Physics at McGill University: PHYS 512
"""

import numpy as np

#Using exp functions as they are often presented as examples.
def funNorm(x):
    return np.e**(x)

def funStretch(x):
    return np.e**(10*x)

#Defining ndiff, function of interest...
def ndiff(fun,x,full):
    epsil = 1e-15 #Epsil used in Q1 for consistency (mentioned in class)
    Df3 = 1e-3 #Picked this for no particular reason

    f3 = (3*fun(x-Df3) - 3*fun(x+Df3) + fun(x+3*Df3) - fun(x-3*Df3)) * (1/(8*Df3**3))
    dx = ((epsil)*(fun(x)/f3))**(1/3)

    f1 = (fun(x + dx) - fun(x - dx))/(2*dx)
    error = epsil*(fun(x)/dx) + 2 * f3 * (dx**2)

    if full == True:
        return f1, dx, error
    elif full == False:
        return f1

#Above is the assignement, Lines under test the defined function(s)
fprime, dx, error_fPrime = ndiff(funStretch, 1, True)
print('Exp(10x) about x=1')
print('dx:', dx)
print('Computational Deriv:',fprime, 'Analytical Deriv:',10*np.e**10)
print('Computed Error:', error_fPrime, 'Analytical Error:', np.abs(fprime - 10*np.e**10))
print('Ratio btwn Comp and Analyt Errors: ', error_fPrime/np.abs(fprime - 10*np.e**10), '\n')


fprime, dx, error_fPrime = ndiff(funStretch, 0, True)
print('Exp(10x) about x=0')
print('dx:', dx)
print('Computational Deriv:', fprime, 'Analytical Deriv:', 10)
print('Computed Error:', error_fPrime, 'Analytical Error:', np.abs(fprime-10))
print('Ratio btwn Comp and Analyt Errors: ', error_fPrime/np.abs(fprime-10), '\n')


fPrimeValue, dx, error_fPrime = ndiff(funNorm, 1, True)
print('Exp(x) about x=1')
print('dx:', dx)
print('Computational Deriv:',fPrimeValue, 'Analytical Deriv:', np.e)
print('Computed Error:',error_fPrime, 'Analytical Error:', np.abs(fPrimeValue-np.e))
print('Ratio btwn Comp and Analyt Errors: ', error_fPrime/np.abs(fPrimeValue-np.e), '\n')


fPrimeValue, dx, error_fPrime = ndiff(funNorm, 0, True)
print('Exp(x) about x=0')
print('dx:', dx)
print('Computational Deriv:',fPrimeValue,'Analytical Deriv:', 1)
print('Computed Error:', error_fPrime, 'Analytical Error:', np.abs(fPrimeValue-1), )
print('Ratio btwn Comp and Analyt Errors: ', error_fPrime/np.abs(fPrimeValue-1), '\n')
```

Problem 3: Lakeshore 670 diodes.

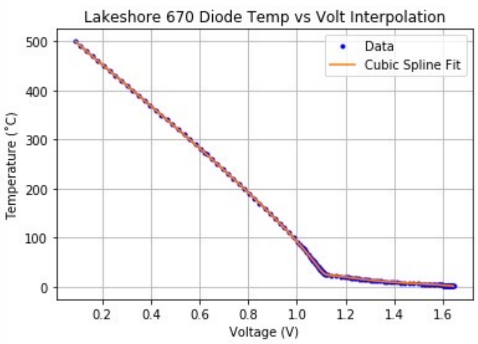
```
Hello! Input a voltage between 0.090681 and 1.64429 Volts:
0.1

 As per the Cubic Spline Fit, the Temperature is   495.6689391056888 Kelvin.
As per the Polynomial Fit, the Temperature is  495.5823627798449 Kelvin.
Polynomial Fit Error:  0.05008394883489402
Cubic Spline Fit Error:  0.005869676742489251
```



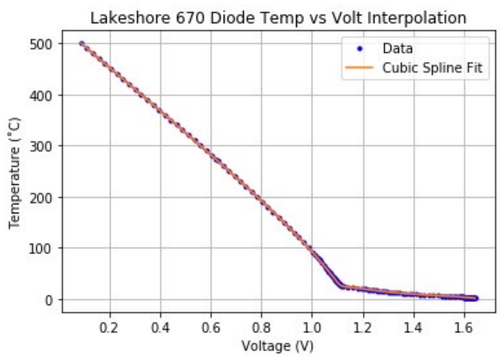

```
Hello! Input a voltage between 0.090681 and 1.64429 Volts:
0.6

 As per the Cubic Spline Fit, the Temperature is   282.4640460364362 Kelvin.
As per the Polynomial Fit, the Temperature is  282.4351831626149 Kelvin.
Polynomial Fit Error:  0.05008394883489402
Cubic Spline Fit Error:  0.005869676742489251
```




```
Hello! Input a voltage between 0.090681 and 1.64429 Volts:
1

 As per the Cubic Spline Fit, the Temperature is   92.8996409775585 Kelvin.
As per the Polynomial Fit, the Temperature is  92.89954807119172 Kelvin.
Polynomial Fit Error:  0.05008394883489402
Cubic Spline Fit Error:  0.005869676742489251
```




```
Hello! Input a voltage between 0.090681 and 1.64429 Volts:
1.6

 As per the Cubic Spline Fit, the Temperature is   3.464059110197756 Kelvin.
As per the Polynomial Fit, the Temperature is  3.46403263750881 Kelvin.
Polynomial Fit Error:  0.05008394883489402
Cubic Spline Fit Error:  0.005869676742489251
```

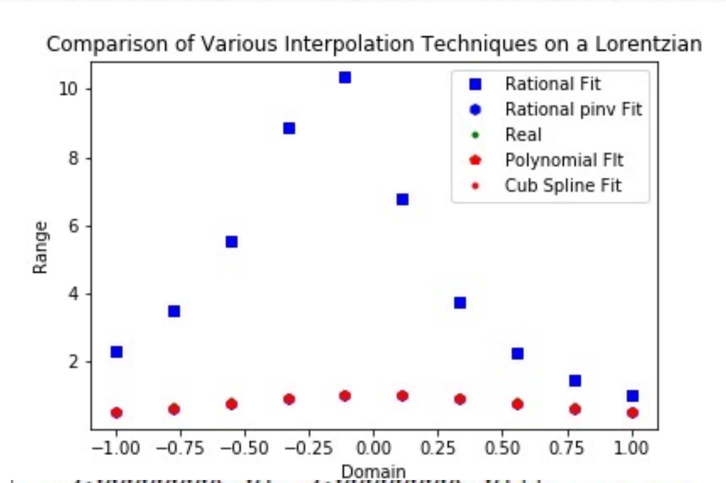


The results are highly sensible and agree with the data.

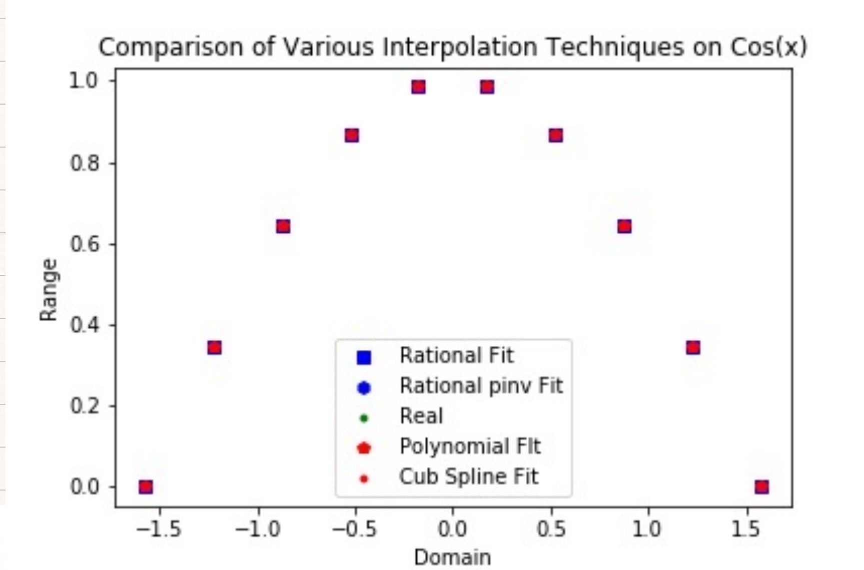
```python
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 14 11:09:13 2022

@author: Yacine Benkirane
Solution to Q3 of PSET1
Comp Physics at McGill University: PHYS 512
"""


import matplotlib.pyplot as plt
import numpy as np
import scipy.interpolate as sp

dat = np.loadtxt("lakeshore.txt")
Temperature = dat[:,0]
Voltage = dat[:,1]

#print(min(volt), max(volt))
plt.clf();
plt.plot(Voltage, Temperature, "b.", label = 'Data')

#Let's iterate through even and odd segments (Technique mentioend by Prof)
TempEven = Temperature[::2]
TempOdd = Temperature[1::2]
VoltEven = Voltage[::2]
VoltOdd = Voltage[1::2]

#Interpolate using Polynomial
def PlyInt(Volt,Temp,x):
    Ply = 0

    for m in range(0, len(Volt), 4):
        min = m
        max = m + 4

        if m == (len(Volt)-4):
            max= m + 3

        if x <= Volt[min] and x >= Volt[max]:
            for i in range(min, max):

                xx = np.append(Volt[min:i],Volt[i + 1:max])
                y0 = Temp[i]
                x0 = Volt[i]
                num = 1
                den = np.prod((x0 - xx))

                for j in xx:
                   num=num*(x - j)
                Ply = Ply + (y0*num)/den

    return Ply



interpolVolt = np.linspace(Voltage[1],Voltage[-1],501)
polyPlot = np.zeros(len(interpolVolt)) #Polynomial Interpolation Graphing

for j in range(len(interpolVolt)): #Looping through Voltage Domain
   polyPlot[j]=PlyInt(Voltage, Temperature, interpolVolt[j])
plt.plot(interpolVolt, polyPlot)


#Requesting User for Input (Arbitrary) Voltage to be interpolated
userResp = float(input('Hello! Input a voltage between 0.090681 and 1.64429 Volts: \n'))

print('\n', "As per the Cubic Spline Fit, the Temperature is  " , sp.splev(userResp, sp.splrep(Voltage[::-1], Temperature[::-1])), "Kelvin.")
print("As per the Polynomial Fit, the Temperature is ", PlyInt(Voltage, Temperature, userResp),"Kelvin.")

#def lakeshore(V,data):

PlyEvenDomain = np.zeros(len(VoltEven))
for i in range(len(VoltEven)):
    PlyEvenDomain[i]=PlyInt(VoltOdd, TempOdd, VoltEven[i])
plt.plot(VoltEven, PlyEvenDomain, color='red')
PlyError = np.mean(np.abs(PlyEvenDomain- TempEven)) #Error in Polynomial Fit

splineOdd =sp.splrep(VoltOdd[::-1],TempOdd[::-1])
CubSplnError = np.mean(np.absolute(sp.splev(VoltEven, splineOdd)- TempEven)) #Cubic Spline Error

print("Polynomial Fit Error: ", PlyError)
print("Cubic Spline Fit Error: ", CubSplnError, '\n')


plt.plot(VoltOdd, sp.splev(VoltOdd, splineOdd), label = 'Cubic Spline Fit') #Plotting Interpolation
plt.ylabel("Temperature ('C)")
plt.xlabel("Voltage (V)")
plt.title("Lakeshore 670 Diode Temp vs Volt Interpolation")
plt.grid()
plt.legend()
plt.savefig('lakeshoreDiagram.png')
plt.show()
```

Problem 4:

$\cos(x)$, $x \in [-\frac{\pi}{2}, +\frac{\pi}{2}]$ and $\frac{1}{(1+x^2)}$, $x \in [-1, 1]$




```
Rational Fit Summed Error: 37.709292139176824
Rational (pinv) Summed Error: 0.5322844071755793
Polynomial Fit Summed Error: 0.5322844071755091
Cubic Spline Summed Error: 0.5322844071755731
```




```
Rational Fit Summed Error: 3.4881802210287945e-10
Rational (pinv) Summed Error: 1.7732701051373824e-10
Polynomial Fit Summed Error: 5.82752664153685e-14
Cubic Spline Summed Error: 8.943282723798091e-16
```

★ The Rational fit using np.linalg.inv gives an abnormally large error; whereas np.linalg.pinv rational, Polynomial, and Spline are reasonable and well-fitted.

↳ As prof Sievers explained, pinv is critical in removing extreme spikes (highly relevant for Lorentzians).

↳ sets eigenvalues near zero at zero!!!

★ for $f(x) = \cos(x)$, all interpolation techniques give "good" fits; with Polynomial at the lowest error and Rat Fit at the highest.

All code is also on github.

```
"""
Created on Wed Sep 14 11:15:28 2022

@author: Yacine Benkirane
Solution to Q3 of PSET1
Comp Physics at McGill University: PHYS 512
"""


import numpy as np
from numpy.polynomial import polynomial
from scipy.interpolate import splev, splrep
from matplotlib import pyplot as plt



def Lorentzian(X):
    return 1/(1+X**2)

#Evaluating the Rational Expression
def RationalEval(p, q, x): #Directly from Prof Sievers  lines 20 to 29
    num=0
    for i in range(len(p)):
        num = num + p[i] * (x**i)
    den = 1

    for i in range(len(q)):
        den = den + q[i] * (x**(i+1))
    return num/den

#Fitting the Rational Model
def RationalFit(x,y,n,m):   #Directly from Prof Sievers lines 31 to 48

    assert(len(x)==n+m-1)
    assert(len(y)==len(x))
    conc=np.zeros([n+m-1,n+m-1])

    for i in range(n):
        conc[:,i]=x**i

    for i in range(1,m):
        conc[:,i-1+n]= -y * (x**i)

    print(conc)

    prod = np.dot( np.linalg.inv(conc), y)
    return prod[:n], prod[n:]

#pinv... Lecture Notes Inspired!
def rat_fit_pinv_version(x, y, n, m):
    assert(len(x)==n+m-1)
    assert(len(y)==len(x))
    conc = np.zeros([n+m-1,n+m-1])
    for i in range(n):
        conc[:,i]=x**i
    for i in range(1,m):
        conc[:, i-1+n]=-y*x**i

    print(conc)
    prod = np.dot(np.linalg.pinv(conc), y)
    return prod[:n], prod[n:]



def RoutineOutput(func, a, b):
    Ptns = 10
    X = np.linspace(a,b, Ptns)
    Y = func(X)

    #Fitting Polynomials
    PolyfitVals = polynomial.polyfit(X,Y,11)
    PolyFunc = polynomial.Polynomial(PolyfitVals)
    spl = splrep(X,Y)

    #Such order was recommended during conversation
    ratp, ratq = RationalFit(X,Y,6,5)
    ratp2, ratq2 = rat_fit_pinv_version(X,Y,6,5)

    RationalFitPoints = RationalEval(ratp, ratq, np.linspace(a, b, 10))
    RatPinvFitPoints = RationalEval(ratp2,ratq2,np.linspace(a, b, 10))
    realPoints = np.cos(np.linspace(a, b, 10))
    PolynomialFitPoints = PolyFunc(np.linspace(a, b, 10))
    CubicSplinePoints = splev(np.linspace(a, b, 10),spl)


    print("Rational Fit Summed Error:", np.sum(np.abs(RationalFitPoints - realPoints)))
    print("Rational (pinv) Summed Error:", np.sum(np.abs( RatPinvFitPoints - realPoints )))

    print("Polynomial Fit Summed Error:", np.sum(np.abs( PolynomialFitPoints - realPoints )))
    print("Cubic Spline Summed Error:", np.sum(np.abs( CubicSplinePoints - realPoints )))
    plt.plot(np.linspace(a, b, 10), RationalFitPoints, 'bs', label = "Rational Fit")
    plt.plot(np.linspace(a, b, 10), RatPinvFitPoints, 'bh', label = "Rational pinv Fit")
    plt.plot(np.linspace(a, b, 10), realPoints, 'g.', label = "Real")
    plt.plot(np.linspace(a, b, 10), PolynomialFitPoints, 'rp', label = "Polynomial FIt " )
    plt.plot(np.linspace(a, b, 10), CubicSplinePoints, 'r.', label = "Cub Spline Fit")
    plt.legend()
    plt.xlabel("Domain")
    plt.ylabel("Range")
    plt.title('Comparison of Various Interpolation Techniques on Cos(x)')
#    plt.savefig('CosineInterpolation.png')

#print("\n Lorentzian Fits:")
#RoutineOutput(Lorentzian, -1, 1)
#plt.show()

print("Cos(x) Fits")
RoutineOutput(np.cos, -np.pi/2, np.pi/2)
#plt.show()
```