Assignment 1

Problem 1:

a) Let us expand the derivative from $x \pm \delta$ and the derivative from $x \pm 2\delta$ of f:

$$\frac{df_2}{dx} = \frac{f(x+\delta) - f(x-\delta)}{2\delta} = \frac{\left(f(x) + f'(x)\cdot\delta + f''(x)\cdot\frac{\delta^2}{2} + \dots\right) - \left(f(x) - f'(x)\delta + f''(x)\cdot\frac{\delta^2}{2}\dots\right)}{2\delta}$$

$$= \frac{1}{\delta}\left(f'(x)\cdot\delta + f'''(x)\frac{\delta^3}{3!}\dots\right)$$

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

$$\frac{df_4}{dx} = \frac{f(x+2\delta) - f(x-2\delta)}{4\delta} = \frac{\left(f(x) + f'(x)2\delta + f''(x)\cdot\frac{4\delta^2}{2} + \dots\right) - \left(f(x) - f'(x)2\delta + f''(x)\cdot\frac{4\delta^2}{2}\dots\right)}{4\delta}$$

$$= \frac{1}{\delta}\left(\frac{f'(x)}{2}\cdot\delta + f'''(x)\frac{4\delta^3}{3!}\dots\right)$$

$$= f'(x) + f'''(x)\frac{4\delta^2}{3!} + \dots$$

We then want to combine those derivative such that

$$\frac{df_{4,2}}{dx} = \frac{4}{3}\frac{df_2}{dx} - \frac{1}{3}\frac{df_4}{dx} = \frac{4}{3}f'(x) + f'''(x)\frac{4\delta^2}{3\cdot 3!} + \dots - \frac{1}{3}f'(x) - f'''(x)\frac{4\delta^2}{3\cdot 3!} = O(\delta^4)$$

Hence our estimate of the derivative operator is:

$$\frac{df_{4,2}}{dx} = \frac{4}{3}\frac{df_2}{dx} - \frac{1}{3}\frac{df_4}{dx} = O(\delta^4)$$

b) We want δ:

As the derivative operator gives $O(\delta^4)$ then our error is of the form

$$Err \simeq \frac{\epsilon f(x)}{\delta} + f^{(5)}(x)\delta^4 \qquad \text{with } \epsilon = 10^{-16} \text{ (As seen in class)}$$

We want to find where Err is the closest to 0 as we differenciate

Hence $Err = 0 \iff -\frac{\epsilon f(x)}{dx^2} + 4f^{(5)}(x)\delta^3 = 0$

$$\iff -\epsilon f(x) + 4f^{(5)}(x)\delta^5 = 0$$

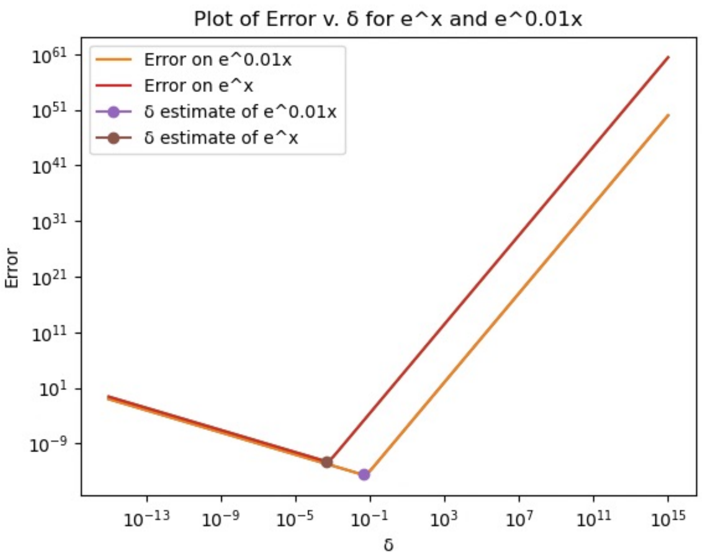
$$\iff \delta = \left(\frac{\epsilon f(x)}{4f^{(5)}(x)}\right)^{1/5}$$

for $f(x) = e^x$ this gives us

$$\delta = \left(\frac{\epsilon e^x}{4 e^x}\right)^{1/5} = \left(\frac{10^{-16}}{4}\right)^{1/5} = 4.8 \times 10^{-4}$$

for $f(x) = e^{0,01x}$ this gives us

$$\delta = \left(\frac{\epsilon e^{0,01 \cdot x}}{4 \cdot 0,01^5 \, e^{0,01x}}\right)^{1/5} = \left(\frac{\epsilon}{4 \cdot (0,01)^5}\right)^{1/5} = 4.8 \times 10^{-2}$$



Problem 2: We have:

$$f' = \frac{f(x+dx) - f(x-dx)}{dx}$$ as our derivative operator.

Hence as seen in class: $ERR \simeq \frac{f\epsilon}{dx} + f''' dx^2$

We want f''' hence we want to express f''' in terms of f:

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

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

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

Hence $f''' = \frac{1}{2dx^3}\left[f(x+2dx) - 2f(x+dx) + 2f(x-dx) - f(x-2dx)\right]$

Therefore $ERROR = \frac{f\epsilon}{dx} + \frac{1}{2dx}\left[f(x+2dx) - 2f(x+dx) + 2f(x-dx) - f(x-2dx)\right]$

for $f(x) = x^2$ we get $f'(5) = 9{,}999\,999999\,999\,996$

$dx = 5.2 \times 10^{-2}$

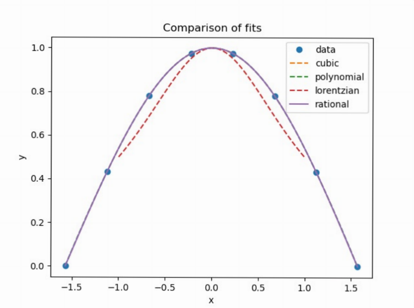
$Error = 4.79 \times 10^{-14}$

Problem 4:

The error for the Lorentzian from the rational function fit is supposed to be important as the lorentzian is not sinusoidal and is not a good estimate.

When we increase the order we get that the polynomial error decreases, the rational error increases and the cubic spline error decreases. It means for polynomial and the cubic spline more points allow better precision.

for $n = 4$ $m = 5$ we get

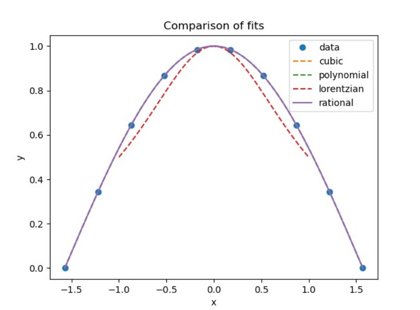


Rational error $= 3{,}48 \times 10^{-5}$

Polynomial error $= 9{,}06 \times 10^{-4}$

Cubic spline error $= 2{,}07 \times 10^{-4}$

for $n = 4$ $m = 7$ we get

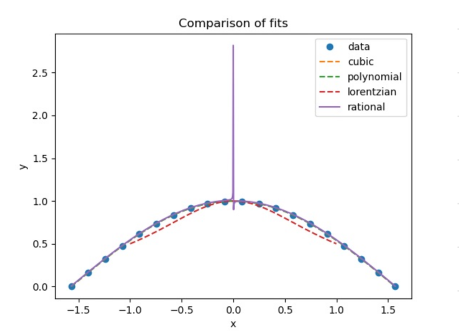


Rational error $= 1{,}5 \times 10^{-6}$

Polynomial error $= 8{,}83 \times 10^{-5}$

Cubic spline error $= 5{,}48 \times 10^{-5}$

for $n = 10$ $m = 11$ we get

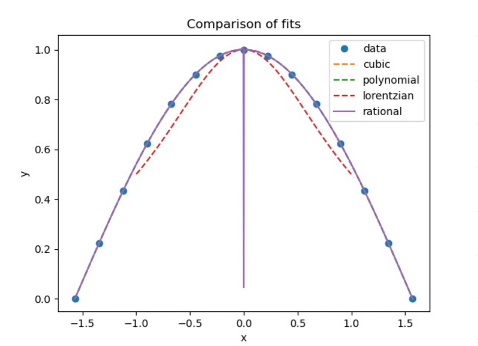


Rational error $= 5{,}76 \times 10^{-2}$

Polynomial error $= 4.40 \times 10^{-10}$

Cubic spline error $= 1.14 \times 10^{-6}$

for $n = 6$ $m = 10$ we get



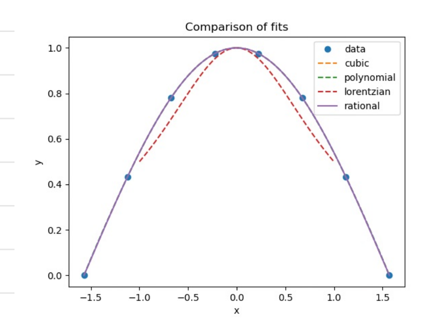
Rational error $= 3{,}01 \times 10^{-2}$

Polynomial error $= 2.5 \times 10^{-14}$

Cubic spline error $= 5.4 \times 10^{-6}$

When we modify with the np.linalg.pinv

for $n = 4$ $m = 5$ we get



Rational error = $3{,}48 \times 10^{-5}$

Polynomial error = $9{,}06 \times 10^{-4}$

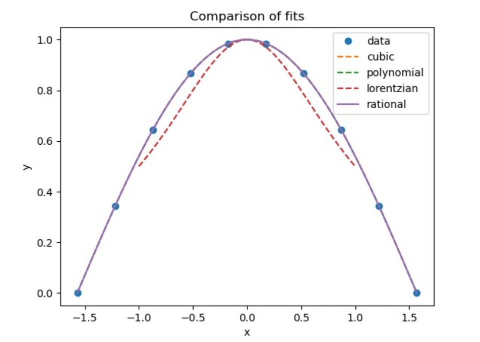
without modification

Cubic spline error = $2{,}07 \times 10^{-4}$

$p = [\, 9{,}9 \times 10^{-1} ;\ 0 ;\ -4.05 \times 10^{-1} ;\ 1.14 \times 10^{-13} \,]$

$q = [\, 0 ;\ 9.45 \times 10^{-2} ;\ 1.42 \times 10^{-14} ;\ 6.23 \times 10^{-3} \,]$

for $n = 4$ $m = 5$ we get



Rational error = $3{,}48 \times 10^{-5}$

with modification

Polynomial error = $9{,}06 \times 10^{-4}$

Cubic spline error = $2{,}07 \times 10^{-4}$

$p = [\, 9.9 \times 10^{-1} ;\ -2.27 \times 10^{-13} ;\ -4.05 \times 10^{-1} ;\ 5.68 \times 10^{-14} \,]$

$q = [\, 0 ;\ 9{,}45 \times 10^{-2} ;\ 1.42 \times 10^{-14} ;\ 6.23 \times 10^{-3} \,]$

Changing this parameter will allow to have 0 determinants matrices to have them inverted. Therefore we have less and more precise 0 coefficients.

We understand that by looking at p and q, p has increasing even terms whereas q does not change.

all_problems_code

September 19, 2022

Problem 1 : code

```
[ ]: #!/usr/bin/env python3
     # -*- coding: utf-8 -*-
     """
     Created on Tue Sep 13 18:24:36 2022

     @author: louis
     """



     import numpy as np
     import matplotlib.pyplot as plt

     #Machine python error epsilon
     epsilon = 10**(-16)

     #exponential function e^x
     def expo(x):
         return np.exp(x)

     #exponential function e^0.01x
     def expo_oo(x):
         return np.exp(0.01*x)

     #Error functions from the derivation done to find the minimum delta
     def error_plot(delta):
         return epsilon* np.exp(1) * (1/delta) + np.exp(1)*delta**4

     def error_plot_001(delta):
         return epsilon* np.exp(0.01*1) * (1/delta) + (0.01)**(5)*np.exp(0.01*1) *␣
      ↪delta**4

     #logscale set up
     logdx=np.linspace(-15,15,10000)
     dx=10**logdx

     plt.loglog(dx,error_plot_001(dx))
```

```
plt.plot(dx,error_plot_001(dx), label="Error on e^0.01x")

plt.loglog(dx,error_plot(dx))
plt.plot(dx,error_plot(dx),label="Error on e^x")

#delta estimate
dx_estimate_001 = 4.8*10**(-2)
dx_estimate = 4.8*10**(-4)

plt.plot(dx_estimate_001, error_plot_001(dx_estimate_001),␣
↪marker='o',label="\u03B4 estimate of e^0.01x")
plt.plot(dx_estimate, error_plot(dx_estimate), marker='o',label="\u03B4␣
↪estimate of e^x")
plt.legend()
plt.title("Plot of Error v. \u03B4 for e^x and e^0.01x")
plt.xlabel("\u03B4")
plt.ylabel("Error")
plt.show()
```

Problem 2 : code

[ ]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 13 18:24:36 2022

@author: louis
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import derivative
e = 10**(-16)#Here is the python internal error
def dx_finder(fun, x, dx): #function representing the error in terms of x and dx
    return fun(x) * e * (1/dx) + (1/dx) * (1/2) * (fun(x + 2*dx) -  2*fun(x +␣
↪dx) + 2*fun(x-dx) - fun(x-2*dx))
def find_min_dx(fun, x): #function that tries all the possible values of dx to␣
↪find the minimum error
    logdx=np.linspace(-15,-1,100)
    dx=10**logdx
    err = dx_finder(fun, x, dx)
    mini = np.abs(err[0])
    mini_index = 0
    for i in range(len(err)):
        if np.abs(err[i]) < mini:
            mini = np.abs(err[i])
            mini_index = i
```

```
    return dx[mini_index]
def ndiff(fun, x, full=False):
    dx = find_min_dx(fun, x)#compute the dx for the minimum error
    if full:
        derivative_estimate = (fun(x + dx) - fun(x - dx))/(2*dx) #computation␣
 ↪of our derivative
        error = dx_finder(fun, x, dx) #computation of our error
        return derivative_estimate, dx, error
    else:
        derivative_estimate = (fun(x + dx) - fun(x - dx))/(2*dx) #computation␣
 ↪of our derivative
        return derivative_estimate
```

Problem 3 : code

```
[ ]: #!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 13 18:24:36 2022

@author: louis
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
import collections.abc
dat=np.loadtxt('lakeshore.txt')
def lakeshore(V,data):
    V_points = []
    T_points = []
    #We reshape the (3*144) data into (144*33) to get the lists of V and T
    for line in data:
        T_points.append(line[0])
        V_points.append(line[1])
    #We interpolate
    f_interpolate = interp1d(V_points, T_points, kind='cubic')

        #We return the error and the interpolated value
    if isinstance(V, (collections.abc.Sequence, np.ndarray)):
        return list(f_interpolate(V)), np.std(T_points -␣
 ↪f_interpolate(V_points))
    else:
        #We return the error and the interpolated value
        return float(f_interpolate(V)), np.std(T_points -␣
 ↪f_interpolate(V_points))
```

Problem 4 : code

[ ]:

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 13 18:24:36 2022

@author: louis
"""
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d


#True cos(x) function
x = np.linspace(-np.pi/2, np.pi/2, num=8)
y = np.cos(x)
xnew = np.linspace(-np.pi/2,np.pi/2,1001)

#cubic spline interpolation function
cubic_spline_interpolation = interp1d(x, y, kind='cubic')

#Rational fit functions
def rat_eval(p,q,x):
    top=0
    for i in range(len(p)):
        top=top+p[i]*x**i
    bot=1
    for i in range(len(q)):
        bot=bot+q[i]*x**(i+1)
    return top/bot
def rat_fit(x,y,n,m):
    assert(len(x)==n+m-1)
    assert(len(y)==len(x))
    mat=np.zeros([n+m-1,n+m-1])
    for i in range(n):
        mat[:,i]=x**i
    for i in range(1,m):
        mat[:,i-1+n]=-y*x**i
    pars=np.dot(np.linalg.pinv(mat),y)
    p=pars[:n]
    q=pars[n:]
    return p,q

n=4
m=5
p,q=rat_fit(x,y,n,m)
rational_interpolation=rat_eval(p,q,xnew)
```

```
#polynomial interpolation function
fitp=np.polyfit(x,y,n+m-1)
polynomial_interpolation=np.polyval(fitp,xnew)
#polynomial_interpolation = np.poly1d(np.polyfit(x,y, 3))

#Lorentzian function
xnew2 = np.linspace(-1, 1, num=100, endpoint=True)
lorentzian = 1/(1 + xnew2**2)

plt.plot(x, y, 'o')
plt.plot( xnew, cubic_spline_interpolation(xnew), '--')
plt.plot( xnew, polynomial_interpolation, '--')
plt.plot( xnew2, lorentzian, '--')
plt.plot(xnew, rational_interpolation)
plt.legend(['data', 'cubic', 'polynomial', 'lorentzian', 'rational'],␣
↪loc='best')
plt.title("Comparison of fits")
plt.xlabel("x")
plt.ylabel("y")

#Error computation
print("q is " + str(q)+"and  p is " + str(p))
print('Rational error is ',np.std(rational_interpolation-np.cos(xnew)))
print('Polynomial interpolation error is ',np.std(polynomial_interpolation-np.
↪cos(xnew)))
print('Cubic spline interpolation error is ',np.
↪std(cubic_spline_interpolation(xnew)-np.cos(xnew)))
plt.show()
```