Paul Simmerling
pws17001

Homework 3

1 Question 15

$$\text{Fit Parameters} = \begin{bmatrix} 18.4099405 \\ -0.00583313334 \end{bmatrix}$$

$\sigma = 1.1641716342135628$

2 Question 16

$$\text{Linear Fit} = \begin{bmatrix} -6.18989525 \\ 9.43854354 \end{bmatrix}$$

$\sigma_{linear} = 2.2435638279603114$

$$\text{Quadratic Fit} = \begin{bmatrix} 4.40567377 \\ -1.06889613 \\ 2.10811822 \end{bmatrix}$$

$\sigma_{quad} = 0.8129279610540698$

3 Question 17

$$\text{Fit} = \begin{bmatrix} -240.391746 \\ 0.137688935 \end{bmatrix}$$

$\sigma = 2.8552022884971544$

4 Question 18

$$\text{Roots} = \begin{bmatrix} -4.712388979739401 \\ -3.208838734015047 \\ 1.5707963275870451 \end{bmatrix}$$

5 Question 19

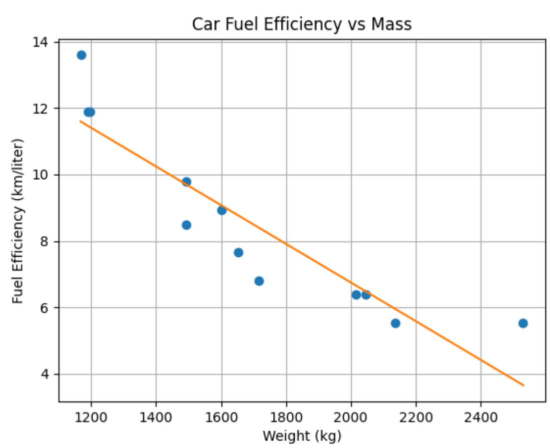
$$\text{Roots:} = \begin{bmatrix} -4.71238898038469 \\ -3.2088387319804816 \\ 1.5707963267948966 \end{bmatrix}$$

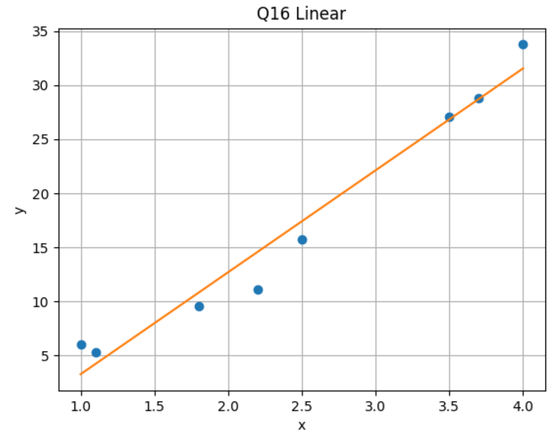
6 Question 20

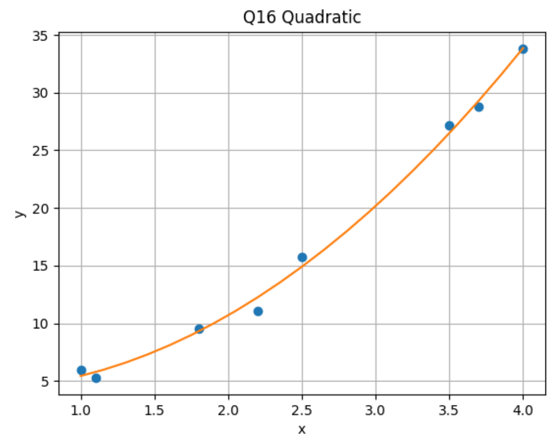
$$\text{Roots:} = \begin{bmatrix} 5.4125482419963555 \end{bmatrix}$$

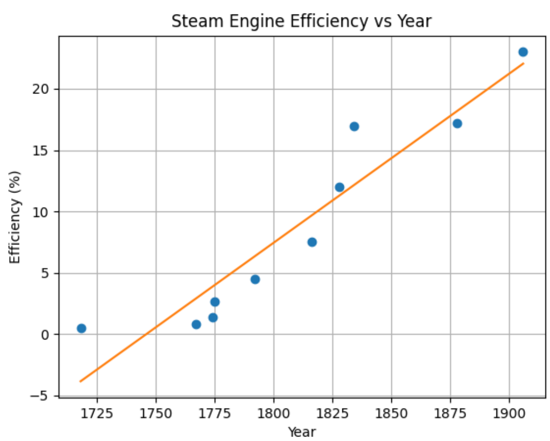
7 Question 21

$$\text{Roots:} = \begin{bmatrix} 0.881592594940857 \\ 1.329402126690394 \\ 1.4351760946763272 \\ 1.4748716035823155 \\ 1.4973496731626401 \end{bmatrix}$$










```
#P15: Page 142 Problem 6
import numpy as np
```

```
import math
from hw2 import gausPivot

def polyFit(xData, yData, m):
    a = np.zeros((m+1, m+1))
    b = np.zeros(m+1)
    s = np.zeros(2*m+1)
    for i in range(len(xData)):
        temp = yData[i]
        for j in range(m+1):
            #Sum data for b array -> Sum(x^j*y)
            b[j] = b[j] + temp
            #temp=x^j*y for j>0 else temp = y
            temp = temp*xData[i]
        temp = 1.0
        #make s matrix tat can be transformed into A
        for j in range(2*m+1):
            s[j] = s[j] + temp
            #make m^j for each loc
            temp = temp*xData[i]
    #reshape 1D S into 2d A
    for i in range(m+1):
        for j in range(m+1):
            a[i,j]=s[i+j]
    #solve for coeffts
    return gausPivot(a,b)

def evalPoly(c,x):
    #Evaluate polynomial with coeff c at point x
    m = len(c)-1
    p = c[m]
    for j in range(m):
        p = p*x + c[m-j-1]
    return p

#Standard stddev calculations
def stdDev(c,xData,yData):#Isn't this just Chi**2/NDF?
    n, m = len(xData) - 1, len(c) - 1
    sigma = 0.0
    for i in range(n+1):
        p = evalPoly(c,xData[i])
        #sigma = sum((y-f(x))^2)
        sigma = sigma + (yData[i] - p)**2
    #Divide by NDF (so you don't make an interpolating function)
    sigma = math.sqrt(sigma/(n - m))
```

```
    return sigma

import matplotlib.pyplot as plt

#plot data and poly
def plotPoly(xData, yData, coeff, title="", xlab='x',ylab='y'):
    plt.clf()
    m, x1, x2 = len(coeff), min(xData), max(xData)
    dx = (x2-x1)/20
    x = np.arange(x1, x2 + dx/10.0, dx)
    y = np.zeros((len(x)))*1.0
    for i in range(m):
        y = y + coeff[i]*x**i
    plt.plot(xData,yData,'o',x,y,'-')
    plt.xlabel(xlab); plt.ylabel(ylab)
    plt.title(title)
    plt.grid(True)
    plt.savefig("figures/"+title+".png")
    plt.show()


#P18: Page 166 Problem 10
from numpy import sign

def rootsearch(f, a, b, dx):
    x1 = a; f1 = f(a)
    x2 = a + dx; f2 = f(x2)
    #iterate between a and b using step
    #size dx searching for when f crosses x axis
    #                    => sign flips!
    while sign(f1) == sign(f2):
        #if a > b then no roots(or missed root!)
        if x1 >= b: return None, None
        x1 = x2; f1 = f2
        x2 = x1 +dx; f2 = f(x2)
    else:
        return x1, x2

def ridder(f, a, b, tol=1.0e-9):
    fa = f(a)
    if fa == 0.0: return a
    fb = f(b)
    if fb == 0.0: return b
    #if sign(f2) != sign(f3): x1 = x3; f1 = f3
    for i in range(30):
```

```
        c = 0.5*(a + b); fc = f(c)
        s = math.sqrt(fc**2 - fa*fb)
        if s == 0.0: return  None
        #create interpolant straight line
        dx = (c - a)*fc/2
        #make sure youre going in the right direction
        if (fa - fb) < 0.0: dx = -dx
        x = c + dx; fx = f(x)
        #if within tol, return x
        if i > 0:
            if abs(x - x01d) < tol*max(abs(x),1.0): return x
        x01d = x
        #create new bounds for next iteration
        if sign(fc) == sign(fx):
            if sign(fa) != sign(fx): b = x; fb = fx
            else: a = x; fa = fx
        else:
            a = c; b = x; fa = fc; fb = fx
    return None
    print('Too many iterations')


def multisearch_ridder(f, a, b, dx=0.01):
    roots = []
    x1, x2 = rootsearch(f, a, b, dx)
    while x2 != None:
        #use rootsearch to find bounds on many roots in interval a b
        roots.append([x1,x2])
        x1, x2 = rootsearch(f, x2, b, dx)
    if len(roots) > 0:
        for i in range(len(roots)):
            #use ridder to get root in each bound from rootsearch
            roots[i] = ridder(f, roots[i][0],roots[i][1])
        print("Roots:",roots)


def newtonRaphson(f, df, a, b, tol=1.0e-9):
    #Make sure you aren't already on root
    fa = f(a)
    if fa == 0.0: return a
    fb = f(b)
    if fb == 0.0: return b
    if sign(fa) == sign(fb): return None
    x = 0.5*(a + b)
    for i in range(30):
```

```python
        fx = f(x)
        if fx == 0.0: return x
        #make range smaller
        if sign(fa) != sign(fx): b =x
        else: a = x
        dfx = df(x)
        #use taylor series FO straight line to determine how far to move
        try: dx = -fx/dfx
        #if fail -> just move a out of bounds
        except ZeroDivisionError: dx = b - a
        x = x + dx
        #if outside of bounds use bisection
        if (b - x)*(x - a) < 0.0:
            dx = 0.5*(b-a)
            x = a + dx
        #if dx is small, close to root
        if abs(dx) < tol*max(abs(b),1.0): return x
    print('Too many iterations in Newton-Raphson')


def multisearch_newtonRaphson(f, df, a, b, dx=0.01):
    roots = []
    x1, x2 = rootsearch(f, a, b, dx)
    while x2 != None:
        #find multiple roots in bound a b
        roots.append([x1,x2])
        x1, x2 = rootsearch(f, x2, b, dx)
    if len(roots) > 0:
        for i in range(len(roots)):
            #find exact root for each sub-boundary
            roots[i] = newtonRaphson(f, df, roots[i][0], roots[i][1])
        print("Roots:",roots)

if __name__ == "__main__":
    m0 = [1198, 1715, 2530, 2014, 2136, 1492, 1652, 1168, 1492, 1602, 1192,
    p0 = [11.9, 6.80, 5.53, 6.38, 5.53, 8.50, 7.65, 13.6, 9.78, 8.93, 11.9,
    m = np.array(m0, dtype=float)
    p = np.array(p0, dtype=float)

    fit = polyFit(m, p, 1)
    sig = stdDev(fit, m, p)
    plotPoly(m,p,fit,"Car Fuel Efficiency vs Mass", "Weight (kg)", "Fuel Ef
    print("+-----------+")
    print("|    Q15    |")
    print("+-----------+")
```

```
    print("Fit: ", fit)
    print("Sig: ", sig)



    #P16: Page 143 Problem 9
    x0 = [  1.0,    2.5,    3.5,    4.0,   1.1,   1.8,    2.2,
3.7]
    y0 = [6.008, 15.722, 27.130, 33.772, 5.257, 9.549, 11.098, 28.828]
    x = np.array(x0, dtype=float)
    y = np.array(y0, dtype=float)

    fit1 = polyFit(x, y, 1)
    sig1 = stdDev(fit1, x, y)
    fit2 = polyFit(x, y, 2)
    sig2 = stdDev(fit2, x, y)
    plotPoly(x,y,fit1,"Q16 Linear")
    plotPoly(x,y,fit2, "Q16 Quadratic")
    print("")
    print("+-----------+")
    print("|    Q16    |")
    print("+-----------+")
    print("Fit1: ", fit1)
    print("Sig1: ", sig1)
    print("Fit2: ", fit2)
    print("Sig2: ", sig2)


    #P17: Page 143 Problem 10
    y0 = [1718, 1767, 1774, 1775, 1792, 1816, 1828, 1834, 1878, 1906]
    e0 = [ 0.5,  0.8,  1.4,  2.7,  4.5,  7.5, 12.0, 17.0, 17.2, 23.0]
    y = np.array(y0, dtype=float)
    e = np.array(e0, dtype=float)

    fit = polyFit(y, e, 1)
    sig = stdDev(fit, y, e)
    plotPoly(y,e,fit,"Steam Engine Efficiency vs Year","Year","Efficiency (%
    print("")
    print("+-----------+")
    print("|    Q17    |")
    print("+-----------+")
    print("Fit: ", fit)
    print("Sig: ", sig)

    print("")
```

```python
print("+-----------+")
print("|    Q18    |")
print("+-----------+")
def f(x): return x*math.sin(x) +3*math.cos(x) - x
multisearch_ridder(f, -6, 6)

#P19: Page 166 Problem 11
#Repeat Prob. 10 with the Newton-Raphson method.
print("")
print("+-----------+")
print("|    Q19    |")
print("+-----------+")
def df(x): return x*math.cos(x)-2*math.sin(x)-1
multisearch_newtonRaphson(f, df, -6, 6)

#P20: Page 169 Problem 20
# find TT that results in 30% -> eta=0.3
print("")
print("+-----------+")
print("|    Q20    |")
print("+-----------+")
def f20(TT): return (math.log(TT)-(1-1/TT))/(math.log(TT)+(1-1/TT)/(5/3-

#for i in range(0, 1000, 10):
#     multisearch_ridder(f, i, i+10)
multisearch_ridder(f20, 0.01, 100)

#P21: Page 170 Problem 25
#math.tan(x)-y=1
#math.cos(x)-3*sin(y)=0
#math.cos(x)-3*math.sin(1-math.tan(x))=0
print("")
print("+-----------+")
print("|    Q21    |")
print("+-----------+")
def f21(x): return math.cos(x)-3*math.sin(math.tan(x)-1)

multisearch_ridder(f21, 0, 1.5)
```