Paul Simmerling
pws17001

Homework 2

1 P7: Page 78 Problem 5

$$\mathbf{A} = \begin{bmatrix} 4 & -2 & 1 \\ -2 & 1 & -1 \\ -2 & 3 & 6 \end{bmatrix} \tag{1.1}$$

$$\mathbf{B} = \begin{bmatrix} 2 \\ -1 \\ 0 \end{bmatrix} \tag{1.2}$$

$$\mathbf{X} = \begin{bmatrix} 0.75 \\ 0.5 \\ -0.0 \end{bmatrix} \tag{1.3}$$

2 P8: Page 78 Problem 7

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & -1 & 2 & -1 \\ -1 & 2 & -1 & 0 \end{bmatrix} \tag{2.1}$$

$$\mathbf{B} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.2}$$

$$\mathbf{X} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{2.3}$$

3 P9: Page 79 Problem 12

$$\mathbf{A} = \begin{bmatrix} 60 & -10 & -20 \\ -10 & 20 & -10 \\ -20 & -10 & 30 \end{bmatrix} \tag{3.1}$$

$$\mathbf{B} = \begin{bmatrix} 200 \\ 100 \\ 200 \end{bmatrix} \tag{3.2}$$

$$\mathbf{X} = \begin{bmatrix} 16.667 \\ 26.667 \\ 26.667 \end{bmatrix} \tag{3.3}$$

4 P10: Page 82 Problem 19

$$\mathbf{A_2} = \begin{bmatrix} 0 & 1 \\ 1 & 4 \end{bmatrix} \tag{4.1}$$

$$\mathbf{B_2} = \begin{bmatrix} 1 \\ 5 \end{bmatrix} \tag{4.2}$$

$$\mathbf{X_2} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{4.3}$$

$$\mathbf{A_3} = \begin{bmatrix} 0 & 1 & 4 \\ 1 & 4 & 9 \\ 4 & 9 & 16 \end{bmatrix} \tag{4.4}$$

$$\mathbf{B_3} = \begin{bmatrix} 5 \\ 14 \\ 29 \end{bmatrix} \tag{4.5}$$

$$\mathbf{X_3} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{4.6}$$

$$\mathbf{A_4} = \begin{bmatrix} 0 & 1 & 4 & 9 \\ 1 & 4 & 9 & 16 \\ 4 & 9 & 16 & 25 \\ 9 & 16 & 25 & 36 \end{bmatrix} \tag{4.7}$$

$$\mathbf{B_4} = \begin{bmatrix} 14 \\ 30 \\ 54 \\ 86 \end{bmatrix} \tag{4.8}$$

Matrix is **singular**, this is because the columns of **A** are no longer linearly independent.

5 P11: Page 126 Problem 1

$$\begin{bmatrix} \mathbf{X} \\ \mathbf{Y} \end{bmatrix} = \begin{bmatrix} -1.2 & 0.3 & 1.1 \\ -5.76 & -5.61 & -3.69 \end{bmatrix} \tag{5.1}$$

Neville's Method - At x = 0: y = -6.0

Lagrange's Method - At x = 0: y = -6.0

6 P12: Page 126 Problem 6

$$\begin{bmatrix} \mathbf{X} \\ \mathbf{Y} \end{bmatrix} = \begin{bmatrix} -2 & 1 & 4 & -1 & 3 & -4 \\ -1 & 2 & 59 & 4 & 24 & -53 \end{bmatrix} \tag{6.1}$$

$$\text{Divided Diff} = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 \\ 59 & 10 & 3 & 0 & 0 & 0 \\ 4 & 5 & -2 & 1 & 0 & 0 \\ 24 & 5 & 2 & 1 & 0 & 0 \\ -53 & 26 & -5 & 1 & -0 & 0 \end{bmatrix} \tag{6.2}$$

Polynomial is of order **3** (cubic)

7 P13: Page 127 Problem 7

$$\begin{bmatrix} \mathbf{X} \\ \mathbf{Y} \end{bmatrix} = \begin{bmatrix} -3 & 2 & -1 & 3 & 1 \\ 0 & 5 & -4 & 12 & 0 \end{bmatrix} \tag{7.1}$$

$$\text{Newton Coef} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{7.2}$$

$$y = (x+3) + (x-2)(x+3) = x^2 + 2x - 3 \tag{7.3}$$

8 P14: Page 127 Problem 14

x = 1.1, y = 1.3262

x = 1.2, y = 1.3938

x = 1.3, y = 1.4693

Found different y for the last x, typo in book?

```python
#ECE_3431 HW2
from math import *
import numpy as np
from np2latex import *
from hw1 import swapRows, swapCols, transpose
#from bookcode import LUpivot


def gausPivot(a, b, tol=1.0e-12):
    n = len(b)

    #Set up scale factors based on max for each row
    s = np.zeros(n)
    for i in range(n):
        s[i] = max(np.abs(a[i, :]))

    for k in range(0, n-1):
        #Swap rows based on scaled max vals
        p = np.argmax(np.abs(a[k:n,k])/s[k:n]) + k
        #Make sure no zeros in diag
        if abs(a[p,k]) < tol:
            print('Matrix is singular')
            return 'Matrix is singular'
        if p != k:
            swapRows(b, k, p)
            swapRows(s, k, p)
```

```
            swapRows(a, k, p)

        #Standard Gauss Elimination
        for i in range(k+1, n):
            if a[i,k] != 0.0:
                lam = a[i,k]/a[k,k]
                a[i,k+1:n] = a[i,k+1:n] -  lam*a[k,k+1:n]
                b[i] = b[i] - lam*b[k]
    #Detect if the last row (and in turn all rows) has 0
    if abs(a[n-1,n-1]) < tol:
        print('Matrix is singular')
        return 'Matrix is singular'
    #Standard Backwards Sub
    b[n-1] = b[n-1]/a[n-1,n-1]
    for k in range(n-2, -1, -1):
        b[k] = (b[k] - np.dot(a[k,k+1:n],b[k+1:n]))/a[k,k]
    return b

def P7():
    print("P7: Page 78 Problem 5")

    Q7_A = np.array([[4, -2, 1],
                    [-2, 1, -1],
                    [-2, 3, 6]], dtype=float)
    Q7_B = np.array([2, -1, 0], dtype=float)
    np2latex(Q7_A, "A")
    np2latex(transpose(Q7_B), "B")
    Q7_X = gausPivot(Q7_A, Q7_B)
    np2latex(transpose(Q7_X), "X")
def P8():
    print("----------------------------------")
    print("P8: Page 78 Problem 7")

    Q8_A = np.array([[2, -1, 0, 0],
                    [0, 0, -1, 1],
                    [0, -1, 2, -1],
                    [-1, 2, -1, 0]], dtype=float)
    Q8_B = np.array([1, 0, 0, 0], dtype=float)
    np2latex(Q8_A, "A")
    np2latex(transpose(Q8_B), "B")
    Q8_X = gausPivot(Q8_A, Q8_B)
    np2latex(transpose(Q8_X), "X")
def P9():
    print("----------------------------------")
    print("P9: Page 79 Problem 12")
```

```
    #Use your program to solve the equations for k=10 and W=100.
    k = 10
    W = 100
    k1, k2, k3, k4, k5 = k, 2*k, k, k, 2*k
    W1, W2, W3 = 2*W, W, 2*W
    Q9_A = np.array([[k1+k2+k3+k5, -k3, -k5],
                    [-k3, k3+k4, -k4],
                    [-k5, -k4, k4+k5]], dtype=float)
    Q9_B = np.array([W1, W2, W3], dtype=float)
    np2latex(Q9_A, "A")
    np2latex(transpose(Q9_B), "B")
    Q9_X = gausPivot(Q9_A, Q9_B)
    np2latex(transpose(Q9_X), "X")

def genQ19_A(n):
    a = np.zeros([n,n])
    for i in range(n):
        #symmetric -> save time
        for j in range(i, n):
            a[i,j] = (i+j)**2
            a[j,i]=a[i,j]
    return a

#generates B
def genQ19_B(a):
    return np.sum(a, 0)

#Your program should return:
#1. Matrix A for the current value of 'n'
#2. Corresponding Matrix b
#3. Solution Matrix i.e., Matrix x
# (or return "SINGULAR" as string/array if that's the case)
    #Generates matrix
def P10():
    print("----------------------------------")
    print("P10: Page 82 Problem 19")
    for n in [2, 3, 4]:
        Q10_A = genQ19_A(n)
        Q10_B = genQ19_B(Q10_A)
        np2latex(Q10_A, "A_{"+str(n)+"}")
        np2latex(transpose(Q10_B), "B_{"+str(n)+"}")
        Q10_X = gausPivot(Q10_A, Q10_B)
        if Q10_X != "Matrix is singular":
            np2latex(transpose(Q10_X), "X_{"+str(n)+"}")
        print("")
```

```
def neville(xData,yData,x):
    m = len(xData)
    y = yData.copy()
    for k in range(1,m):
        #Recursively solve interpolants for
        #y_{k} = ((x-xd_{i+k})*yi+(xd_i-x)*y_{i+1})
        #          /(x_i-x_{i+k})
        y[0:m-k]= ((x - xData[k:m])*y[0:m-k] + \
                            (xData[0:m-k] - x)*y[1:m-k+1])/ \
                            (xData[0:m-k] - xData[k:m])
    return y[0]


def lagrange(xData, yData, x):
    n, f = len(xData), 0
    for i in range(n):
        l = 1
        #make lagrange coeffs parts
        #l = (x-xd_2)/(xd_i-xd_2)*(x-xd_3)/(xd_i-xd_3)*...
        for j in range(n):
            if i != j:#avoid divide by zero
                l = l*(x - xData[j])/(xData[i] - xData[j])
        f += float(yData[i]*l)
    return f

def P11():
    print("-------------------------------------")
    print("P11: Page 126 Problem 1")
    Q11_X = np.array([-1.2, 0.3, 1.1])
    Q11_Y = np.array([-5.76, -5.61, -3.69])
    np2latex(np.vstack((Q11_X, Q11_Y)), "Top:X,Bot:Y")

    print("  \\item Neville's Method")
    Q11_X0_neville = neville(Q11_X, Q11_Y, 0)
    print("    At x = 0: y = "+str(round(Q11_X0_neville, 5)))



    print("  \\item Lagrange's Method")
    Q11_X0_lagrange = lagrange(Q11_X, Q11_Y, 0)
    print("    At x = 0: y = "+str(round(Q11_X0_lagrange, 5)))

def P12():
```

```
    print("-----------------------------------")
    print("P12: Page 126 Problem 6")
    #The program should return "divided difference table"
    # and "degree of the polynomial" respectively.
    #Hint: Look at page 113 of the book to learn how to
    # find the degree of a polynomial from the
    # divided difference table.
    Q12_X = np.array([-2, 1, 4, -1, 3, -4], dtype=float)
    Q12_Y = np.array([-1, 2, 59, 4, 24, -53], dtype=float)

    np2latex(np.vstack((Q12_X, Q12_Y)), "Top:X,Bot:Y")

    n=len(Q12_X)
    #make divided difference table
    dd = np.zeros([n,n])
    dd[:,0] = Q12_Y
    for i in range(1,n):
        for o in range(1,i+1):
            #can recursively find divided difference from previous column
            dd[i,o] = (dd[i,o-1]-dd[o-1,o-1])/(Q12_X[i]-Q12_X[o-1])
    np2latex(dd, "\\text{Divided Diff}")
    #last nonzero is the order
    print("Polynomial is of order " + str(max([i for i, arg in enumerate(dd.


def evalPoly(a,xData,x):
    #evaluate using a as poly coeffs
    n = len(xData) - 1
    p = a[n]
    for k in range(1, n+1):
        p = a[n-k] + (x - xData[n-k])*p
    return p
def coeffts(xData,yData):
    m = len(xData)
    a = yData.copy()
    #recursively find p_k using p_k = a_{n-k}+(x-x_{m-k})*p_{k-1}
    #with p_0=a_m=y_m
    for k in range(1,m):
        a[k:m] = (a[k:m] - a[k-1])/(xData[k:m] - xData[k-1])
    return a

def P13():
    print("-----------------------------------")
    print("P13: Page 127 Problem 7")
    Q13_X = np.array([-3, 2, -1, 3, 1], dtype=float)
```

```
    Q13_Y = np.array([0, 5, -4, 12, 0], dtype=float)
    np2latex(np.vstack((Q13_X, Q13_Y)), "Top:X,Bot:Y")
    Q13_coef = coeffts(Q13_X, Q13_Y)
    np2latex(transpose(Q13_coef), "\\text{Newton Coef}")
    #print(evalPoly(Q13_coef, Q13_X, Q13_X))

def P14():
    print("-----------------------------------")
    print("P14: Page 127 Problem 14")

    Q14_X = np.array( [ -2.0, -0.1, -1.5, 0.5, -0.6, 2.2, 1.0, 1.8 ] )
    Q14_Y = np.array( [ 2.2796, 1.0025, 1.6467, 1.0635, 1.0920, 2.6291, 1.26

    for x in [1.1, 1.2, 1.3]:
        print("x = "+str(x)+", y = "+str(round(neville(Q14_X,Q14_Y,x), 4)))

    print("-----------------------------------")


if __name__ == "__main__":
    P7()
    P8()
    P9()
    P10()
    P11()
    P12()
    P13()
    P14()
```