2ο Σετ Ασκήσεων

Υπολογιστικά Μαθηματικά

Ψαρράς Δημήτριος

ΑΕΜ: 4407

Επίλυση σε Python 3

Άσκηση 1η

Εκφώνηση

Represent the linear system

$$\begin{aligned} x_1 - x_2 + 2x_3 - x_4 &= -8 \\ 2x_1 - 2x_2 + 3x_3 - 3x_4 &= -20 \\ x_1 + x_2 + x_3 &= -2 \\ x_1 - x_2 + 4x_3 + 3x_4 &= 4 \end{aligned}$$

as an augmented matrix and use Gaussian Elimination to find its solution.

Λύση

In [5]:

```python
import numpy as np

def Gaussian_Elimination(Aug,r,c):
    X = np.zeros(r)
    for i in range(c-1):
        Aug = Pivoting(Aug,i)
        for j in range(i+1,r):
            xn = Aug[j][i]/Aug[i][i]
            Aug[[j]]=C[[j]]-(Aug[[i]]*xn)

    for i in range(r-1,-1,-1):
        b = Aug[i][-1]
        for j in range(c):
            if i!=j:
                b = b - (Aug[i][j]*X[j])
        X[i] = b/Aug[i][i]
    return X

def Pivoting(Aug,c_c):
    Aug_P = np.absolute(Aug)
    l = len(Aug_P)
    max_v = Aug_P[c_c][c_c]
    p = c_c
    for i in range(c_c+1,l):
        if Aug_P[i][c_c]>max_v:
            max_v = Aug_P[i][c_c]
            p = i
    temp = Aug[[c_c]]
    Aug[[c_c]] = Aug[[p]]
    Aug[[p]] = temp
    return Aug

A = np.array([[1.0, -1.0, 2.0, -1.0],[2.0, -2.0, +3.0, -3.0],[1.0, 1.0, 1.0, 0.0],[1.0, -1.0, +4.0, +3.0]])
B = np.array([[-8.0],[-20.0],[-2.0],[4.0]])
C = np.append(A,B, axis=1)

rows = len(A)
columns = len(A[0])

X = Gaussian_Elimination(C, rows, columns)

print('The solution vector is: ')
print(X)
print('\nAnalytically the solution is: ')
for i in range(len(X)):
    print(f'x_{i} = {round(X[i],5)}')
```

```
The solution vector is:
[-7.  3.  2.  2.]

Analytically the solution is:
x_0 = -7.0
x_1 = 3.0
x_2 = 2.0
x_3 = 2.0
```

Άσκηση 2η

Εκφώνηση

Values for $f(x) = xe^x$ are given in the following table. Use numerical differentiation (aim for errors of $O(h^2)$) to complete the table and compare your results with the actual values.

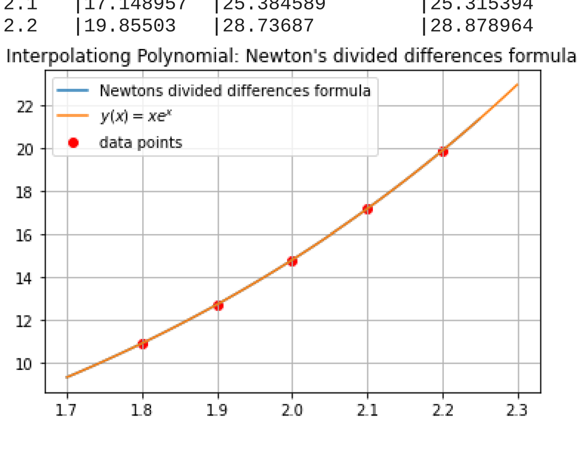
x	f(x)	f'(x)
1.8	10.889365	
1.9	12.703199	
2.0	14.778112	
2.1	17.148957	
2.2	19.855030	

Λύση

In [6]:

```python
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt
import math

def truncate(number, digits) -> float:
    stepper = 10.0 ** digits
    return math.trunc(stepper * number) / stepper

def Forward_diff(y,h):
    i = 0
    y_diff = (-y[i+2]+4*y[i+1]-3*y[i])/(2*h)
    return y_diff

def Backward_diff(y,h):
    i = len(y)-1
    y_diff = (y[i-2]-4*y[i-1]+3*y[i])/(2*h)
    return y_diff

def Central_diff(y,h):
    y_diff = np.zeros(len(y)-2)
    for i in range(1,len(y)-1):
        y_diff[i-1] = (y[i+1]-y[i-1])/(2*h)
    return y_diff

def Differentiation(x,y,y_actual):
    col = len(y)
    y_diff = np.zeros(col)
    h = round(x[1] - x[0],3)
    for i in range(1,col):
        if round(x[i]-x[i-1]-h,1)!=0.0:
            print('Warning: Data not equally spaced')
    y_diff[0] = Forward_diff(y_points,h)
    y_diff[col-1] = Backward_diff(y_points,h)
    y_diff[1:col-1] = Central_diff(y_points,h)
    P = Newton(x,y,y_actual)
    Diff_Pol = Polynomial_Diff(P,x,col)
    Diff_Actual = Actual_Diff(y_actual,col)
    Diffs = np.concatenate((x.reshape((col,1)),y.reshape((col,1)),y_diff.reshape((col,1)),Diff_Pol.reshape((col,1)),Diff_Actual.reshape((col,1))),axis = 1)
    return Diffs

def div_diff(x,y,n):
    table = np.zeros((n,n))
    for i in range(0,n):
        for j in range(0,n-i):
            if i!=0:
                table[j][i] = (table[j+1][i-1]-table[j][i-1])/(x[i+j]-x[j])
            else:
                table[j][i] = y[j]
    a = table[np.arange(0,n)][0]
    return a

def Newton(x, y,y_a):
    sym_x = sp.symbols('x')
    n = len(y)
    P = 0
    L1 = 1
    a = div_diff(x,y,n)
    for i in range(0,n):
        for j in range(0,i):
            l1 = sym_x - x[j]
            L1 = L1*l1
        L = a[i]*L1
        P = P + L
        L1 = 1
    P = sp.expand(P)
    Plot_Polynomial(P,y,x,n,y_a)
    return P

def Plot_Polynomial(Pol,y_val,x_val,n,y):
    x = sp.symbols('x')

    p1 = sp.plot(Pol,(x,x_val[0]-0.1,x_val[n-1]+0.05), show = False)
    x1, y1 = p1[0].get_points()

    p2 = sp.plot(y,(x,1.7,2.3), ylabel='y', xlabel='x', show = False)
    x2, y2 = p2[0].get_points()

    plt.figure()
    plt.plot(x1,y1,label=r'Newtons divided differences formula')
    plt.plot(x2,y2,label=r'$y(x)=xe^{x}$')
    plt.scatter(x_val,y_val,label=r'data points', c ="red")
    plt.title('Interpolationg Polynomial: Newton\'s divided differences formula')
    plt.legend()
    plt.grid()
    return

def Polynomial_Diff(Pol,x_points,n):
    P_dot = sp.diff(Pol,x)
    y_dot_num = np.zeros(n)
    for i in range(n):
        y_dot_num[i] = P_dot.subs(x,x_points[i])
    return y_dot_num

def Actual_Diff(y,n):
    y_dot = sp.diff(y,x)
    y_dot_anal = np.zeros(n)
    for i in range(n):
        y_dot_anal[i] = y_dot.subs(x,x_points[i])
    return y_dot_anal

def Print_Table(Table):
    print('\n---------------------------Table of Differentiation---------------------------')
    print("{:<5} |{:<10} |{:<15} |{:<18} |{:<18}".format("x","f(x)","f'(x) Num. Diff.","f'(x) Interpolating Pol.","f'(x) actual"))
    print('------------------------------------------------------------------------------')
    for i in range(len(Table)):
        print("{:<5} |{:<10} |{:<16} |{:<24} |{:<18}".format(Table[i][0], Table[i][1], truncate(Table[i][2],6), truncate(Table[i][3],6), truncate(Table[i][4],6)))
    return

x = sp.symbols('x')
y = x*sp.exp(x)

x_points = np.array([1.8, 1.9, 2.0, 2.1, 2.2])
y_points = np.array([10.889365, 12.703199, 14.778112, 17.148957, 19.855030])

Table_diff = Differentiation(x_points,y_points,y)

Print_Table(Table_diff)
```

```
---------------------------Table of Differentiation---------------------------
x     |f(x)       |f'(x) Num. Diff. |f'(x) Interpolating Pol. |f'(x) actual
------------------------------------------------------------------------------
1.8   |10.889365  |16.832944        |16.938014                |16.939012
1.9   |12.703199  |19.443735        |19.389349                |19.389093
2.0   |14.778112  |22.228789        |22.166999                |22.167168
2.1   |17.148957  |25.384589        |25.315394                |25.315126
2.2   |19.85503   |28.73687         |28.878964                |28.880043
```



Συμπεράσματα

Ο στόχος να συμπληρωθεί ο πίνακας της εκφώνησης είναι δυνατό να επιτευχθεί με δύο τρόπους. Αρχικά χρησιμοποιήθηκε αριθμητική παραγώγιση. Συγκεκριμένα forward αριθμητική παραγώγιση χρησιμοποιήθηκε για το πρώτο σημείο και backward για το τελευταίο σημείο. Τέλος, για τα κεντρικά σημεία χρησιμοποιήθηκε central αριθμητική παραγώγιση. Ο δεύτερος τρόπος ήταν να πραγματοποιηθεί αριθμητική παραγώγιση μέσω συμπτωτικού πολυωνύμου, το οποίο παραγωγίζεται και υπολογίζεται η τιμή της παραγώγου για κάθε τιμή του x. Από τις δύο μεθόδους φαίνεται τις πραγματικές τιμές να προσεγγίζονται από την αριθμητική παραγώγιση μέσω συπτωτικού πολυωνύμου.

Άσκηση 3η

Εκφώνηση

Use the power method to determine the largest eigenvalue of

$$\begin{bmatrix} 4 & 1 & 2 & 1 \\ 1 & 7 & 1 & 0 \\ 2 & 1 & 4 & 1 \\ 1 & 0 & 1 & 3 \end{bmatrix}$$

What is the corresponding eigenvector?

Λύση

In [7]:

```python
import numpy as np
import math

def truncate(number, digits) -> float:
    stepper = 10.0 ** digits
    return math.trunc(stepper * number) / stepper

def Power_Method(A_k,k,a):
    l_prev = 0
    for i in range(1,k+1,1):
        X_k_prev = A_k.dot(X)
        A_k = A_k.dot(A)
        X_k = A_k.dot(X)
        l = np.amax(X_k)/np.amax(X_k_prev)
        if abs(l - l_prev) < pow(10,-a):
            break
        l_prev = l

    max_v = np.amax(X_k)
    l_vector = X_k/max_v

    return l, l_vector, i


A = np.array([[4.0, 1.0, 2.0, 1.0],
              [1.0, 7.0, 1.0, 0.0],
              [2.0, 1.0, 4.0, 1.0],
              [1.0, 0.0, 1.0, 3.0]])
X = np.array([[8.0],[9.0],[8.0],[4.0]])
ac = 5
max_iter = 100

eigenvalue, eigenvector, iteration = Power_Method(A,max_iter,ac)

print(f'The largest eigenvalue of the matrix\n{A}\nis: {truncate(eigenvalue,ac-1)}')
print(f'\nCalculation was completed after {iteration} iterations.')
print(f'\nThe corresponding eigenvector is:\n{eigenvector}')
```

```
The largest eigenvalue of the matrix
[[4. 1. 2. 1.]
 [1. 7. 1. 0.]
 [2. 1. 4. 1.]
 [1. 0. 1. 3.]]
is: 8.1413

Calculation was completed after 26 iterations.

The corresponding eigenvector is:
[[0.57067418]
 [1.        ]
 [0.57067418]
 [0.22199596]]
```