DESHMUKH-HW4

October 31, 2017

```
In [1]: import numpy as np
        import sympy as sp
        import math as math
        import matplotlib.pyplot as plt
        #np.set_printoptions(precision=8,suppress=True)
```

Problem 1:

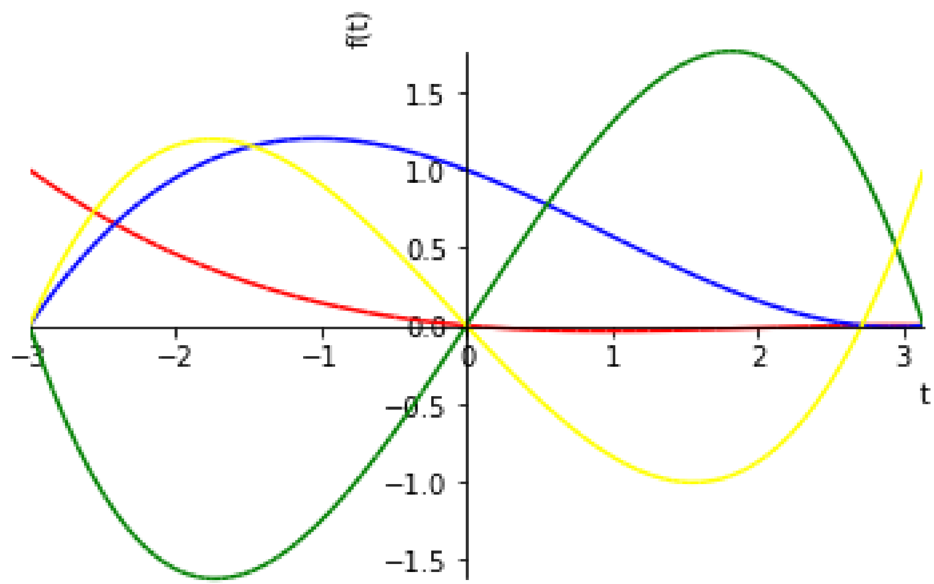
a)

```
In [2]: x=np.array([-3,0,np.e,np.pi])
        def Lag_basis(i,t):
            x = sp.symbols('t')
            L=1
            for j in range(0,len(t)):
                if i != j:
                    L=L*(x-t[j])/(t[i]-t[j])
                    #print(L)
            return(L)
        L30 = Lag_basis(0,x)
        L31 = Lag_basis(1,x)
        L32 = Lag_basis(2,x)
        L33 = Lag_basis(3,x)
        print('L3,0(x) is '+str(sp.expand(L30))+'\n')
        print('L3,1(x) is '+str(sp.expand(L31))+'\n')
        print('L3,2(x) is '+str(sp.expand(L32))+'\n')
        print('L3,3(x) is '+str(sp.expand(L33))+'\n')
```

```
L3,0(x) is -0.00949144162819572*t**3 + 0.0556186565949202*t**2 - 0.0810543888948115*t

L3,1(x) is 0.0390332210162128*t**3 - 0.111630112736439*t**2 - 0.3528559940219*t + 1.0

L3,2(x) is -0.151977951886552*t**3 + 0.0215189614947589*t**2 + 1.43235845146325*t

L3,3(x) is 0.122436172498535*t**3 + 0.0344924946467603*t**2 - 0.998448068546537*t
```

b) Plot of Lagrange polynomials

```
In [3]: t = sp.symbols('t')
        p1=sp.plot(L30,L31,L32,L33,(t,-3,np.pi),show=0)
        p1[0].line_color = 'red'
        p1[1].line_color = 'blue'
        p1[2].line_color = 'green'
        p1[3].line_color = 'yellow'
        p1.show()
```

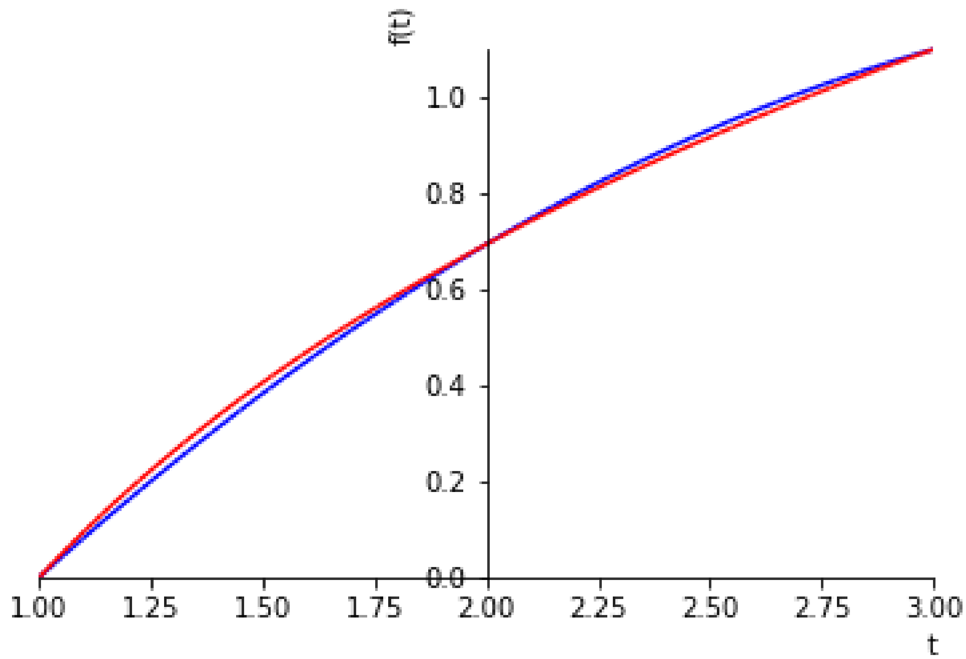


Problem 2:

```
In [4]: def f(x):
            return(np.log(x))

In [5]: x=np.array([1,2,3])
        y=f(x)

In [6]: def Lpoly(x,y):
            poly=0
            for i in range(0,len(x)):
                poly = poly+(Lag_basis(i,x)*y[i])
            return(poly)
        poly = Lpoly(x,y)
        print('The Lagrange form of interpolating polynomial is:\n'
              +str(sp.expand(poly)))
```

```
The Lagrange form of interpolating polynomial is:
-0.14384103622589*t**2 + 1.12467028923762*t - 0.980829253011726
```

In [7]:
```
t = sp.symbols('t')
p1=sp.plot(poly,sp.log(t),(t,1,3),show=0)
p1[0].line_color = 'blue'
p1[1].line_color = 'red'
p1.show()
```



c)

In [8]:
```
def p(t):
    return(-0.14384103622589*t**2 + 1.12467028923762*t - 0.980829253011726)
```

In [9]:
```
a=1.5;b=2.4
fa=f(a);fb=f(b)
pa=p(a);pb=p(b)
ea=np.abs(pa-fa)
eb=np.abs(pb-fb)
print('For x='+str(a)+'\n'+'The estimated value is '+str(pa)+'\n'+
      'The exact value is '+str(fa)+'\n'+'The error is '+str(ea)+'\n')
print('For x='+str(b)+'\n'+'The estimated value is '+str(pb)+'\n'+
      'The exact value is '+str(fb)+'\n'+'The error is '+str(eb)+'\n')
```

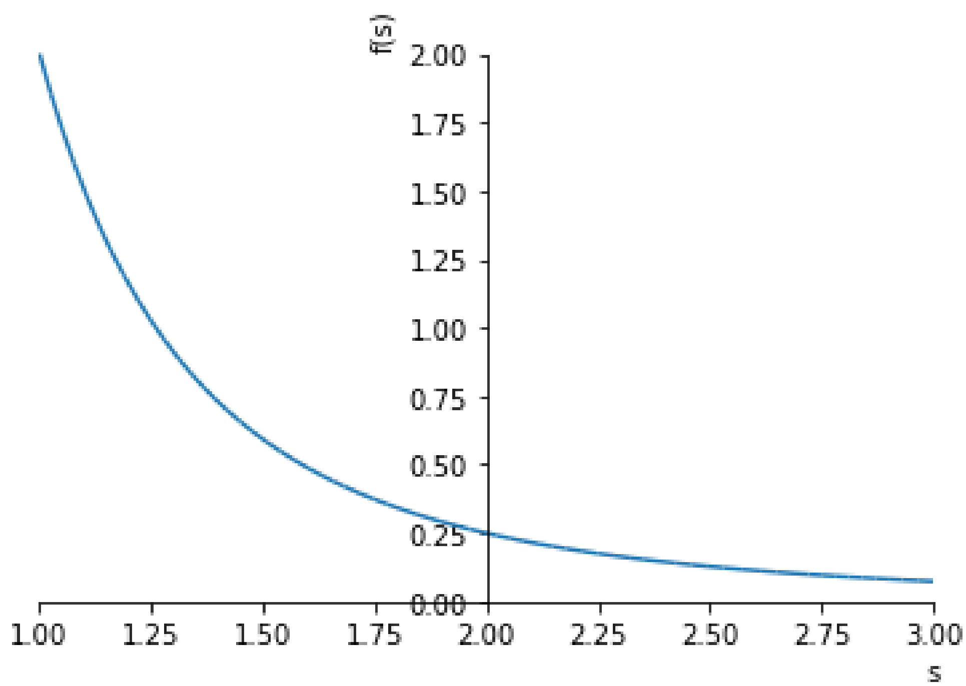
```
For x=1.5
The estimated value is 0.3825338493364516
The exact value is 0.405465108108
The error is 0.0229312587717
```

```
For x=2.4
The estimated value is 0.8898550724974352
The exact value is 0.875468737354
The error is 0.0143863351435
```

d) The theoretical error will be given by $\frac{f^{iii}(z)(t-t_0)(t-t_1)(t-t2)}{(3)!}$ where $z \in [1,3]$

for $f(x) = log(x)$ the third derivative is $f^{iii}(x) = \frac{2}{x^3}$

```
In [10]: #s = sp.symbols('s') #for finding the deriavtive of the function
         #df=sp.log(s)
         #for i in range(0,len(x)):
         #    df = sp.diff(df)
         #print(str(df))
         s = sp.symbols('s')
         sp.plot(2/(s**3),(s,1,3))
```




```
Out[10]: <sympy.plotting.plot.Plot at 0x7fadcc4e5d68>
```

From the above plot of $f^{iii}(x)$ it can be observed that the function is striclty decreasing in $\in [1,3]$ and therefore its maximum value will be at z=1

In [11]:
```
pr=1
for i in range(0,len(x)):
    pr = pr*(a-x[i])
th_er=(2/(1**3))*(pr/math.factorial(len(x))) #z=1
print('The theoretical error is '+str(np.abs(th_er))+
      ' which is greater than the actual error '+str(ea))
```

```
The theoretical error is 0.125 which is greater than the actual error 0.0229312587717
```

Therefore the actual error is within bounds of the theoretical error
Problem 3:

In [12]:
```
x=np.array([1,4,16])
y=np.sqrt(x)
c=9
```

In [13]:
```
def Nev(x,y,c):
    A = np.zeros((len(x),len(x)))
    A[:,0]=y
    for i in range(1,len(x)):
        for j in range(i,len(x)):
            A[j,i]=((c-x[j-i])*(A[j,i-1])-(c-x[j])*(A[j-1,i-1]))/(x[j]-x[j-i])
    #print(A)
    return(A[-1,-1])
value = Nev(x,y,c)
print('The approximate value of the function f(x) at x='+str(c)
      +' is '+str(value)+'\n')
```

```
The approximate value of the function f(x) at x=9 is 3.22222222222
```

Problem 4:

In [14]:
```
x=np.array([0.005, 0.010, 0.020, 0.050, 0.100, 0.200, 0.500, 1.000, 2.000])
logx=np.log(x)
y=np.array([0.924, 0.896, 0.859, 0.794, 0.732, 0.656, 0.536, 0.430, 0.316])
logy=np.log(y)
a=0.032;b=1.682
coef_a=Nev(x,y,a)
coef_b=Nev(x,y,b)
print('For molality of '+str(a)+' The estimated coefficient is '+str(coef_a)+'\n')
print('For molality of '+str(b)+' The estimated coefficient is '+str(coef_b)+'\n')
```

```
For molality of 0.032 The estimated coefficient is 0.830670159374

For molality of 1.682 The estimated coefficient is -216711.826899
```

```
In [15]: #N=100
         #p=np.zeros(N)
         #c=np.linspace(x[0],x[-1],N)
         #for i in range(0,N):
         #    p[i]=Nev(x,y,c[i])
         #plt.plot(c,p)
         #plt.plot(x,y,'x')
         #plt.plot(a,coef_a,'*')
         #plt.plot(b,coef_b,'*')
         #plt.show()
```

Problem 5:

```
In [16]: def f(x):
             return(np.cos(x))
         x=np.array([1,2,3])
         y =f(x)
```

```
In [17]: def Newton_int(t,f):
             x = sp.symbols('x')
             N=len(t)
             A = np.zeros((N,N))
             A[:,0]=np.copy(f)
             for i in range(1,N):
                 for j in range(i,N):
                     A[j,i]=(A[j,i-1]-A[j-1,i-1])/(t[j]-t[j-i])
             poly=0
             for j in range(0,N):
                 m=1
                 for i in range(0,j):
                     m =m*(x-t[i])
                 poly= poly+A[j,j]*m
             return(poly)
         p=Newton_int(x,y)
         print('The Newton form of interpolating polynomial is:\n')
         print(str(sp.expand(p))+'\n')
```

```
The Newton form of interpolating polynomial is:

0.19130174118099*x**2 - 1.53035436595825*x + 1.8793549306454
```

Problem 6:

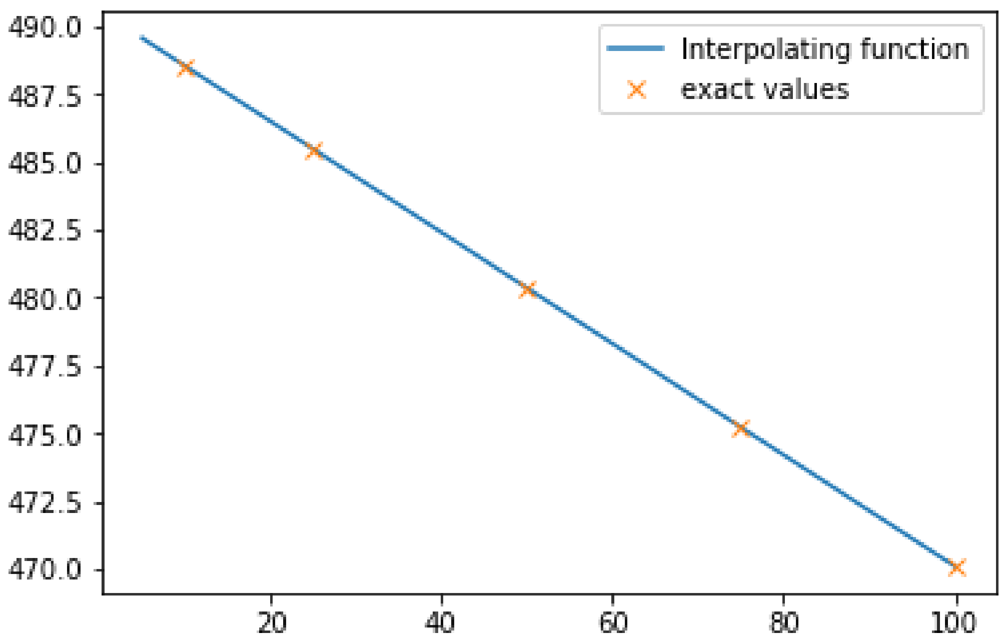
```
In [18]: x=np.array([10, 25, 50, 75, 100])
         y = np.array([488.55, 485.48, 480.36, 475.23, 470.11])
         p=Newton_int(x,y)
         print('The Newton form of interpolating polynomial is:\n')
         print(str(sp.expand(p))+'\n')
```

```
The Newton form of interpolating polynomial is:

3.16809116808524e-9*x**4 - 5.78689458688163e-7*x**3 + 2.93019943018995e-5*x**2 - 0.2052084045584
```

```
In [19]: def f(x):
             return(3.16809116808524e-9*x**4 - 5.78689458688163e-7*x**3 +
                    2.93019943018995e-5*x**2 - 0.205208404558402*x +
                    490.599700854701)
         t=np.arange(5,105,5)
         yt=f(t)
         print('Temp\tSurface Tension\n')
         for i in range(0,len(t)):
             print(str(t[i])+'\t'+str(yt[i])+'\n')
         plt_a,=plt.plot(t,yt,label='Interpolating function')
         plt_b,=plt.plot(x,y,'x',label='exact values')
         plt.legend(handles=[plt_a,plt_b])
         plt.show()
```

```
Temp        Surface Tension

5         489.574321026

10         488.55

15         487.526375043

20         486.50313094

25         485.48

30         484.456762051

35         483.433244444

40         482.409322051

45         481.384917265

50         480.36

55         479.334587692

60         478.308745299

65         477.282585299
```

```
70          476.256267692

75          475.23

80          474.204037265

85          473.178682051

90          472.154284444

95          471.131242051

100          470.11
```



As can be seen from the above plot the interpolating function is a linear function and coincides with the exact values of the given data

Problem 7:

In [20]:
```
x=np.array([100, 200, 300, 400, 500, 600])
y = np.array([9.4, 18.4, 26.2, 33.3, 39.7, 45.7])
p=Newton_int(x,y)
print('The Newton form of interpolating polynomial is:\n')
print(str(sp.expand(p)))
```

```
The Newton form of interpolating polynomial is:
```

```
6.66666666666631e-13*x**5 - 1.20833333333327e-9*x**4 + 8.58333333333295e-7*x**3 - 0.00033291666
```

```
In [21]: def f(x):
             return(6.66666666666631e-13*x**5 - 1.20833333333327e-9*x**4 +
                    8.58333333333295e-7*x**3 - 0.000332916666666655*x**2 +
                    0.145849999999998*x - 2.59999999999993)
         a=240;b=485;
         pa=f(a);pb=f(b)
         print('For Temp '+str(a)+'K The estimated conductivity is '+str(pa)+' mW/mk\n')
         print('For Temp '+str(b)+'K The estimated conductivity is '+str(pb)+' mW/mk')
```

```
For Temp 240K The estimated conductivity is 21.615481599999917 mW/mk

For Temp 485K The estimated conductivity is 38.781375821875024 mW/mk
```