CALCULUS USING python

Suhas.P.K

Contents

1 Limit of a function

The Python program for this section:

```python
#   LIMIT OF A FUNCTION

# LIBRARIES
import matplotlib.pyplot as plt
import sympy as sym
import numpy as np
from IPython.display import display,Math

x = sym.symbols('x')

fx = x**3

lim_pnt = 1.5

lim = sym.limit(fx,x,lim_pnt)
print(lim)

display(Math('\\lim_{x\\to%g} %s = %g '%(lim_pnt,sym.latex(fx),lim)))


# plotting
fxx = sym.lambdify(x,fx)
xx = np.linspace(-5,5,500)

plt.plot(xx,fxx(xx),label='f(x)= $x^{3}$')
plt.plot(lim_pnt,lim,'ro')
plt.axis('square')
plt.axis((-7,7,-7,7))
plt.grid()
plt.legend()
plt.show()
```

2 Piecewise Function-1

The Python program for this section:

```
#   PIECEWISE FUNCTION

#   LIBRARIES
import numpy as np
import sympy as sym
import matplotlib.pyplot as plt

x = sym.symbols('x')
piece1 = 0
piece2 = -2*x
piece3 = (x**3)/10

fx = sym.Piecewise((piece1,x<0),(piece2, (x>=0)&(x<10)),(piece3,x>=10 )
     )

fxx = sym.lambdify(x,fx)
xx = np.linspace(-3,20,1234)

plt.plot(xx,fxx(xx))

plt.show()
```

3 Piecewise Function-2

The Python program for this section:

```
#   PIECEWISE FUNCTION

#   LIBEARIES
import matplotlib.pyplot as plt
import numpy as np
import sympy as sym

x = sym.symbols('x')
f1 = x**3
f2 = sym.log(x,2)




fx = sym.Piecewise( (f1, x<=0) , (f2, x>0 ) )

fxx = sym.lambdify(x,fx)
xx = np.linspace(-3,15,1000)

with plt.xkcd():
    plt.plot(xx,fxx(xx))

plt.xlim([-2,2])
plt.ylim([-10,3])
plt.show()
```

4 Derivative of a polynomial using Symplot

The Python program for this section:

```
# TO FIND THE DERIVATIVE OF THE POLYNOMAIL

# LIBRARIES
import sympy as sym
import sympy.plotting.plot as symplot
import numpy as np

x = sym.symbols('x')
fx = x**2

dfx = sym.diff(fx)

fxx = sym.lambdify(x,fx)
dfxx = sym.lambdify(x,dfx)
xx = np.linspace(-5,5,1000)


p = symplot(fx,(x,-5,5),show=False)
p.extend(symplot(dfx,(x,-5,5),show=False))
p[1].line_color = 'r'
p[0].label='$f(x) = %s$'%sym.latex(fx)
p[1].label='$f\'(x) = %s$'%sym.latex(dfx)

p.legend = True
p.ylim = [-10,10]
p.xlim = [-3,3]
p.show()
```

5 Derivative of a polynomial using Matplotlib

The Python program for this section:

```python
# DERIVATIVE OF A POLYNOMIAL USING MATPLOTLIB

# LIBRARIES
import sympy as sym
import numpy as np
import matplotlib.pyplot as plt


x = sym.symbols('x')
fx = x**2

dfx = sym.diff(fx)

fxx = sym.lambdify(x,fx)
dfxx = sym.lambdify(x,dfx)
xx = np.linspace(-5,5,1000)


with plt.xkcd():
    plt.plot(xx,dfxx(xx), label = '$f\'(x) = %s$'%(sym.latex(dfx)))
    plt.plot(xx,fxx(xx),'r',label = '$f(x) = %s$'%(sym.latex(fx)))

    plt.legend()
plt.show()
```

6 Derivative of a trigonometric function using Symplot

The Python program for this section:

```python
# DERIVATIVE OF TRIGONOMETRIC FUNCTIONS USING SYMPLOT

# LIBRARIES
import matplotlib.pyplot as plt
import numpy as np
import sympy as sym
import sympy.plotting.plot as symplot

x = sym.symbols('x')

f = sym.cos(x)
df = sym.diff(f)

p = symplot(f,(x,0,10),show=False)
p.extend(symplot(df,(x,0,10),show=False))
p[1].line_color = 'r'
p[0].label = '$f(x) = %s $'%sym.latex(f)
p[1].label = '$f\'(x) = %s$'%sym.latex(df)

p.legend = True
p.show()
```

7 Derivative of a trigonometric function using Matplotlib

The Python program for this section:

```
# DERIVATIVE OF TRIGONOMETRIC FUNCTIONS USING SYMPLOT

# LIBRARIES
import matplotlib.pyplot as plt
import numpy as np
import sympy as sym
import sympy.plotting.plot as symplot

x = sym.symbols('x')

fx = sym.cos(x)
dfx = sym.diff(fx)

xx = np.linspace(0,10,1000)
fxx = sym.lambdify(x,fx)
dfxx = sym.lambdify(x,dfx)

with plt.xkcd():
    plt.plot(xx,fxx(xx),'b',label = '$f(x)=%s$'%(sym.latex(fx)))
    plt.plot(xx,dfxx(xx),'r',label = '$f\'(x)=%s$'%(sym.latex(dfx)))

    plt.legend()

plt.axis((0,10,-1.1,1.1))
plt.show()
```

7.1 Exercise-1

Plot the functions $f(x) = sin(x + cos(x) + a$ and also it's derivatives for $a = 1, 2, 3, 4$.
The Python program for this section:

```python
# EXERCISE-1

# LIBRARIES
import numpy as np
import matplotlib.pyplot as plt
import sympy as sym
import sympy.plotting.plot as symplot

x,a = sym.symbols('x,a')
color = 'brkm'
fx = sym.sin(x + sym.cos(x)) + a



for ai in range(0,4):
    if ai==0:
        p = symplot(fx.subs(a,ai),show=False,label='a='+str(ai),
    line_color=color[0])
    else:
        p.extend(symplot(fx.subs(a,ai),show=False,label='a='+str(ai),
    line_color=color[ai]))
p.title = 'The functions'
p.legend = True
p.show()

for ai in range(0,4):
    if ai==0:
        g = symplot(sym.diff(fx.subs(a,ai)),show=False,label='a='+str(
    ai),line_color=color[0])
    else:
        g.extend(symplot(sym.diff(fx.subs(a,ai)) , show=False , label='
    a='+str(ai) , line_color=color[ai]))

g.title = 'Their derivatives'
g.legend = True
g.show()
```

8 Tangent Line for the given point.

The Python program for this section:

```python
# TANGENT LINES

# LIBRARIES
import matplotlib.pyplot as plt
import numpy as np
import sympy as sym


x = sym.symbols('x')

# defining the function.
fx = x**3

# taking the derivative of the function.
dfx = sym.diff(fx)

# the value at which to compute the tangent line.
a = 1

# get the function and derivative value at the point 'a'.
fa = fx.subs(x,a)
dfa = dfx.subs(x,a)


xx = np.linspace(-2,2,500)
f_fun = sym.lambdify(x,fx)(xx)
df_fun = sym.lambdify(x,dfx)(xx)

# the tangent line.
tanline = dfa * (xx-a) + fa

# plotting
with plt.xkcd():
    plt.plot(xx,f_fun,label='$f(x)=%s$'%sym.latex(fx))
    plt.plot(xx,tanline,label='tangent')
    plt.plot(a,fa,'ro',label='a=%g'%a)


    plt.legend(loc='best')

plt.title('The Tangent line ')


plt.axis('square')
plt.axis([-2,2,-0.5,2])

ax = plt.gca()
plt.plot(ax.get_xlim(),[0,0],'k--')
plt.plot([0,0],ax.get_ylim(),'k--')

plt.show()
```

9 Critical points of a function.

9.1 To find the critical points of a function using empirical method.

The Python program for this section:

```python
# FINDING CRITICAL POINTS OF A FUNCTION

# LIBRARIES
import matplotlib.pyplot as plt
import numpy as np
from scipy.signal import find_peaks



x = np.linspace(-4,4,1001)
fx = x**2 * np.exp(-x**2)
dfx = np.diff(fx)/(x[1]-x[0])

localmax = find_peaks(fx)[0]
localmin = find_peaks(-fx)[0]

with plt.xkcd():
    plt.plot(x,fx,label='f(x)')
    plt.plot(x[0:-1],dfx,label='$f\'(x)$')
    plt.plot(x[localmax],fx[localmax],'ko',label='maxima')
    plt.plot(x[localmin],fx[localmin],'ro',label='minima')
    plt.legend()

plt.show()
```

9.2 To find the critical points of a function using symbolic method.

Verify the critical points from the above subsection.
The Python program for this section:

```
# TO FIND CRITICAL POINTS OF A GIVEN FUNCTION - SUMBOLIC METHOD

# LIBRARIES
import numpy as np
import sympy as sym
import matplotlib.pyplot as plt
from scipy.signal import find_peaks

x = sym.symbols('x')
fx = x**2 * sym.exp(-x**2)
dfx = sym.diff(fx)

crit_pnts = sym.solve(dfx)

xx = np.linspace(-4,4,1001)
fxx = sym.lambdify(x,fx)(xx)
dfxx = sym.lambdify(x,dfx)(xx)

print('critical points are ' + str(crit_pnts))

# EXERCISE : PLOT THE CRITICAL POINTS ON THE GIVEN FUNCTION CURVE.

plt.plot(xx,fxx,label='f(x)')
plt.plot(xx,dfxx,label='$f\'(x)$')

plt.legend()
plt.show()
```

10 Partial derivative

The Python program for this section:

```python
# PARTIAL DERIVATIVE

# LIBRARIES
import sympy as sym
import matplotlib.pyplot as plt
import numpy as np

x,y = sym.symbols('x y')

fxy = x**3 + (x**2)*(y**4)

# partial deerivative with respect to x.
fdx = sym.diff(fxy,x)

# partial derivative with respect to y.
fdy = sym.diff(fxy,y)

# plotting

p = sym.plotting.plot3d(fxy,(x,-3,3),(y,-3,3),title = '$f(x,y)=%s$'%(sym.latex(fxy)) )

p = sym.plotting.plot3d(fdx,(x,-3,3),(y,-3,3),title = '$f(x,y)=%s$'%(sym.latex(fdx)) )

p = sym.plotting.plot3d(fdy,(x,-3,3),(y,-3,3),title = '$f(x,y)=%s$'%(sym.latex(fdy)) )
```

11 Integration

11.1 Integration of a polynomial.

The Python program for this section:

```python
# INTEGRATION OF A POLYNOMIAL

# LIBRARIES
import sympy as sym
import numpy as np
import matplotlib.pyplot as plt

x = sym.symbols('x')
fx = x**2

# INDEFINITE INTEGRATION
idi = sym.integrate(fx)

xx = np.linspace(-4,4,1000)
fxx = sym.lambdify(x,fx)(xx)
idi_xx = sym.lambdify(x,idi)(xx)

plt.plot(xx,fxx,label='$f(x)=%s$'%sym.latex(fx))
plt.plot(xx,idi_xx,label='$ \int %s dx = %s+C $ '%(sym.latex(fx),sym.
    latex(idi)))

plt.xlabel('x')
plt.ylabel('y')
plt.title('Indefinite integration')
plt.grid()
plt.legend()
plt.show()
```

11.2 Integration of a trigonometric function.

The Python program for this section:

```
# INTEGRATION OF A TRIGONOMETRIC FUNCTION

# LIBRARIES
import matplotlib.pyplot as plt
import numpy as np
import sympy as sym


x = sym.symbols('x')
fx = sym.cos(x)

idi = sym.integrate(fx)

xx = np.linspace(0,2*np.pi,1000)
fxx = sym.lambdify(x,fx)(xx)
idi_xx = sym.lambdify(x,idi)(xx)

plt.plot(xx,fxx, label='$f(x)=%s$'%(sym.latex(fx)))
plt.plot(xx,idi_xx, label='$ \int %s dx = %s+C$'%(sym.latex(fx),sym.
    latex(idi)))

plt.xlabel('x')
plt.ylabel('y')

plt.grid()
plt.legend()
plt.show()
```

11.3 Definite Integration.

The Python program for this section:

```
# DEFINITE INTEGRATION

# LIBRARIES
import matplotlib.pyplot as plt
import numpy as np
import sympy as sym
import sympy.plotting.plot as symplot

x = sym.symbols('x')
fx = x**3

di = sym.integrate(fx,(x,1,2))

p = symplot(fx,show=False)


p[0].label = '$f(x)=%s$ '%sym.latex(fx)

p.title = '$\int_{1}^{2} %s dx = %g$'%(sym.latex(fx),di)
p.xlim = [-3,3]
p.ylim = [-10,10]
p.legend = True
p.show()
```

11.4 Trapezoidal Method.

The Python program for this section:

```
# TRAPEZOIDAL METHOD

# LIBRARIES

# to define a function.
def f(x):
    return 1/(1 + x**2)

def trapezoidal(x0,xn,n):
    # calculating step size
    h = (xn - x0) / n

    # Finding sum
    integration = f(x0) + f(xn)

    for i in range(1,n):
        k = x0 + i*h
        integration = integration + 2 * f(k)

    # Finding final integration value
    integration = integration * h/2




    return integration


lower_limit = float(input("lower limit = "))
upper_limit = float(input("upper limit = "))
sub_interval = int(input("sub interval = "))


result = trapezoidal(lower_limit,upper_limit,sub_interval)

print("Integration result by Trapeziodal method is = %0.6f"%(result))
```

11.5 Simpson's $\frac{1}{3}$ Method.

The Python program for this section:

```python

# Simpson's 1/3 Rule

# Define function to integrate
def f(x):
    return 1/(1 + x**2)

# Implementing Simpson's 1/3
def simpson13(x0,xn,n):
    # calculating step size
    h = (xn - x0) / n

    # Finding sum
    integration = f(x0) + f(xn)

    for i in range(1,n):
        k = x0 + i*h

        if i%2 == 0:
            integration = integration + 2 * f(k)
        else:
            integration = integration + 4 * f(k)

    # Finding final integration value
    integration = integration * h/3

    return integration

# Input section
lower_limit = float(input("Enter lower limit of integration: "))
upper_limit = float(input("Enter upper limit of integration: "))
sub_interval = int(input("Enter number of sub intervals: "))

# Call trapezoidal() method and get result
result = simpson13(lower_limit, upper_limit, sub_interval)
print("Integration result by Simpson's 1/3 method is: %0.6f" % (result)
    )
```

12 The Fundamental Theorem of Calculus.

$$\int \frac{df(x)}{dx} dx = \frac{d}{dx} \int f(x) dx \tag{1}$$

The Python program for this section:

```python
# FUNDAMENTAL THEOREM OF CALCULUS

# LIBRARIES
import sympy as sym


x = sym.symbols('x')
fx = 2*x + sym.cos(x)

# integrate(differentiate(function))
dfx1 = sym.diff(fx)
idf1 = sym.integrate(dfx1)

# differentiate(integrate(function))
idf2 = sym.integrate(fx)
dfx2 = sym.diff(idf2)

if idf1==fx and dfx2==fx :
    print('The fundamental theorem of calculus holds true.')
else:
    print('The fundamental theorem of calculus holds false.')
```

13 Area under two curves.

For given functions $f(x) = x^2$ & $g(x) = x + 2$, the area between the two curves whose boundary conditions are the intersection points when the two curves intersect (a & b) is given by,

$$\int_a^b [f(x) - g(x)]dx = Area \tag{2}$$

The Python program for this section:

```
# AREA BETWEEN TWO CURVES

# LIBRARIES
import numpy as np
import sympy as sym
import matplotlib.pyplot as plt


x = sym.symbols('x')
f = x**2
g = x+2

h = f-g


xx = np.linspace(-3,3,200)
y1 = sym.lambdify(x,f)(xx)
y2 = sym.lambdify(x,g)(xx)

# this gives the intersecting points.
idx = np.argwhere(np.diff(np.sign(y1-y2)) != 0)

x1 = xx[idx][0]
x2 = y1[idx][1]
print(x1,x2)

A = abs(sym.integrate(h,(x,x1,x2))) # this give the area.

with plt.xkcd():
    fig,ax = plt.subplots()
    ax.fill_between(xx,y1,y2,where=y2>=y1,facecolor = 'green') # shades
      the area.

    plt.plot(xx,y1,label='$f(x)=%s$'%sym.latex(f))
    plt.plot(xx,y2,label='$g(x)=%s$'%sym.latex(g))
    plt.plot(xx[idx],y1[idx],'ro')
    plt.legend()
    plt.title('The area between the two curves is %.4f square units.'%A
    )


plt.show()
```