

CALCULUS USING



Suhas.P.K

Contents

1	Limit of a function	4
2	Piecewise Function-1	5
3	Piecewise Function-2	5
4	Derivative of a polynomial using Symplot	6
5	Derivative of a polynomial using Matplotlib	7
6	Derivative of a trigonometric function using Symplot	8
7	Derivative of a trigonometric function using Matplotlib	9
7.1	Exercise-1	10
8	Tangent Line for the given point.	11
9	Critical points of a function.	12
9.1	To find the critical points of a function using empirical method.	12
9.2	To find the critical points of a function using symbolic method.	13
10	Partial derivative	14
11	Integration	15
11.1	Integration of a polynomial.	15
11.2	Integration of a trigonometric function. . .	16
11.3	Definite Integration.	17

11.4	Trapezoidal Method.	18
11.5	Simpson's $\frac{1}{3}$ Method.	19
12	The Fundamental Theorem of Calculus.	20
13	Area under two curves.	21

1 Limit of a function

The Python program for this section:

```
1 # LIMIT OF A FUNCTION
2
3 # LIBRARIES
4 import matplotlib.pyplot as plt
5 import sympy as sym
6 import numpy as np
7 from IPython.display import display, Math
8
9 x = sym.symbols('x')
10
11 fx = x**3
12
13 lim_pnt = 1.5
14
15 lim = sym.limit(fx,x,lim_pnt)
16 print(lim)
17
18 display(Math('\\lim_{x\\to %g} %s = %g'%(lim_pnt,sym.latex(fx),lim)))
19
20
21 # plotting
22 fxx = sym.lambdify(x,fx)
23 xx = np.linspace(-5,5,500)
24
25 plt.plot(xx,fxx(xx),label='f(x)= $x^{3}$')
26 plt.plot(lim_pnt,lim,'ro')
27 plt.axis('square')
28 plt.axis((-7,7,-7,7))
29 plt.grid()
30 plt.legend()
31 plt.show()
```

2 Piecewise Function-1

The Python program for this section:

```
1 # PIECEWISE FUNCTION
2
3 # LIBRARIES
4 import numpy as np
5 import sympy as sym
6 import matplotlib.pyplot as plt
7
8 x = sym.symbols('x')
9 piece1 = 0
10 piece2 = -2*x
11 piece3 = (x**3)/10
12
13 fx = sym.Piecewise((piece1,x<0),(piece2, (x>=0)&(x<10)),(piece3,x>=10 )
14 )
15 fxx = sym.lambdify(x,fx)
16 xx = np.linspace(-3,20,1234)
17
18 plt.plot(xx,fxx(xx))
19
20 plt.show()
```

3 Piecewise Function-2

The Python program for this section:

```
1 # PIECEWISE FUNCTION
2
3 # LIBRARIES
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import sympy as sym
7
8 x = sym.symbols('x')
9 f1 = x**3
10 f2 = sym.log(x,2)
11
12
13
14 fx = sym.Piecewise( (f1, x<=0) , (f2, x>0 ) )
15
16 fxx = sym.lambdify(x,fx)
17 xx = np.linspace(-3,15,1000)
18
19 with plt.xkcd():
20     plt.plot(xx,fxx(xx))
21
22 plt.xlim([-2,2])
23 plt.ylim([-10,3])
24 plt.show()
```

4 Derivative of a polynomial using Symplot

The Python program for this section:

```
1 # TO FIND THE DERIVATIVE OF THE POLYNOMAIL
2
3 # LIBRARIES
4 import sympy as sym
5 import sympy.plotting.plot as symplot
6 import numpy as np
7
8 x = sym.symbols('x')
9 fx = x**2
10
11 dfx = sym.diff(fx)
12
13 fxx = sym.lambdify(x,fx)
14 dfxx = sym.lambdify(x,dfx)
15 xx = np.linspace(-5,5,1000)
16
17
18 p = symplot(fx,(x,-5,5),show=False)
19 p.extend(symplot(dfx,(x,-5,5),show=False))
20 p[1].line_color = 'r'
21 p[0].label='$f(x) = %s$'%sym.latex(fx)
22 p[1].label='$f\''(x) = %s$'%sym.latex(dfx)
23
24 p.legend = True
25 p.ylim = [-10,10]
26 p.xlim = [-3,3]
27 p.show()
```

5 Derivative of a polynomial using Matplotlib

The Python program for this section:

```
1 # DERIVATIVE OF A POLYNOMIAL USING MATPLOTLIB
2
3 # LIBRARIES
4 import sympy as sym
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8
9 x = sym.symbols('x')
10 fx = x**2
11
12 dfx = sym.diff(fx)
13
14 fxx = sym.lambdify(x,fx)
15 dfxx = sym.lambdify(x,dfx)
16 xx = np.linspace(-5,5,1000)
17
18
19 with plt.xkcd():
20     plt.plot(xx,dfxx(xx), label = '$f\'(x) = %s$'%(sym.latex(dfx)))
21     plt.plot(xx,fxx(xx),'r',label = '$f(x) = %s$'%(sym.latex(fx)))
22
23     plt.legend()
24 plt.show()
```

6 Derivative of a trigonometric function using Symplot

The Python program for this section:

```
1 # DERIVATIVE OF TRIGONOMETRIC FUNCTIONS USING SYMPLOT
2
3 # LIBRARIES
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import sympy as sym
7 import sympy.plotting.plot as symplot
8
9 x = sym.symbols('x')
10
11 f = sym.cos(x)
12 df = sym.diff(f)
13
14 p = symplot(f,(x,0,10),show=False)
15 p.extend(symplot(df,(x,0,10),show=False))
16 p[1].line_color = 'r'
17 p[0].label = '$f(x) = %s$'%sym.latex(f)
18 p[1].label = '$f\'(x) = %s$'%sym.latex(df)
19
20 p.legend = True
21 p.show()
```


7 Derivative of a trigonometric function using Matplotlib

The Python program for this section:

```
1 # DERIVATIVE OF TRIGONOMETRIC FUNCTIONS USING SYMPLOT
2
3 # LIBRARIES
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import sympy as sym
7 import sympy.plotting.plot as symplot
8
9 x = sym.symbols('x')
10
11 fx = sym.cos(x)
12 dfx = sym.diff(fx)
13
14 xx = np.linspace(0,10,1000)
15 fxx = sym.lambdify(x,fx)
16 dfxx = sym.lambdify(x,dfx)
17
18 with plt.xkcd():
19     plt.plot(xx,fxx(xx),'b',label = '$f(x)=s$'%(sym.latex(fx)))
20     plt.plot(xx,dfxx(xx),'r',label = '$f\'(x)=s$'%(sym.latex(dfx)))
21
22     plt.legend()
23
24 plt.axis((0,10,-1.1,1.1))
25 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:

```
1 # EXERCISE-1
2
3 # LIBRARIES
4 import numpy as np
5 import matplotlib.pyplot as plt
6 import sympy as sym
7 import sympy.plotting.plot as symplot
8
9 x,a = sym.symbols('x,a')
10 color = 'brkm'
11 fx = sym.sin(x + sym.cos(x)) + a
12
13
14
15 for ai in range(0,4):
16     if ai==0:
17         p = symplot(fx.subs(a,ai),show=False,label='a='+str(ai),
18                     line_color=color[0])
19     else:
20         p.extend(symplot(fx.subs(a,ai),show=False,label='a='+str(ai),
21                         line_color=color[ai]))
22 p.title = 'The functions'
23 p.legend = True
24 p.show()
25
26 for ai in range(0,4):
27     if ai==0:
28         g = symplot(sym.diff(fx.subs(a,ai)),show=False,label='a='+str(
29             ai),line_color=color[0])
30     else:
31         g.extend(symplot(sym.diff(fx.subs(a,ai)) , show=False , label='
32             a='+str(ai) , line_color=color[ai]))
33
34 g.title = 'Their derivatives'
35 g.legend = True
36 g.show()
```

8 Tangent Line for the given point.

The Python program for this section:

```
1 # TANGENT LINES
2
3 # LIBRARIES
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import sympy as sym
7
8
9 x = sym.symbols('x')
10
11 # defining the function.
12 fx = x**3
13
14 # taking the derivative of the function.
15 dfx = sym.diff(fx)
16
17 # the value at which to compute the tangent line.
18 a = 1
19
20 # get the function and derivative value at the point 'a'.
21 fa = fx.subs(x,a)
22 dfa = dfx.subs(x,a)
23
24
25 xx = np.linspace(-2,2,500)
26 f_fun = sym.lambdify(x,fx)(xx)
27 df_fun = sym.lambdify(x,dfx)(xx)
28
29 # the tangent line.
30 tanline = dfa * (xx-a) + fa
31
32 # plotting
33 with plt.xkcd():
34     plt.plot(xx,f_fun,label='$f(x)=s^3$'%sym.latex(fx))
35     plt.plot(xx,tanline,label='tangent')
36     plt.plot(a,fa,'ro',label='a=%g'%a)
37
38     plt.legend(loc='best')
39
40
41 plt.title('The Tangent line ')
42
43
44 plt.axis('square')
45 plt.axis([-2,2,-0.5,2])
46
47 ax = plt.gca()
48 plt.plot(ax.get_xlim(),[0,0], 'k--')
49 plt.plot([0,0],ax.get_ylim(), 'k--')
50
51 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:

```
1 # FINDING CRITICAL POINTS OF A FUNCTION
2
3 # LIBRARIES
4 import matplotlib.pyplot as plt
5 import numpy as np
6 from scipy.signal import find_peaks
7
8
9
10 x = np.linspace(-4,4,1001)
11 fx = x**2 * np.exp(-x**2)
12 dfx = np.diff(fx)/(x[1]-x[0])
13
14 localmax = find_peaks(fx)[0]
15 localmin = find_peaks(-fx)[0]
16
17 with plt.xkcd():
18     plt.plot(x,fx,label='f(x)')
19     plt.plot(x[0:-1],dfx,label='$f\''(x)$')
20     plt.plot(x[localmax],fx[localmax], 'ko',label='maxima')
21     plt.plot(x[localmin],fx[localmin], 'ro',label='minima')
22     plt.legend()
23
24 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:

```
1 # TO FIND CRITICAL POINTS OF A GIVEN FUNCTION - SUMBOLIC METHOD
2
3 # LIBRARIES
4 import numpy as np
5 import sympy as sym
6 import matplotlib.pyplot as plt
7 from scipy.signal import find_peaks
8
9 x = sym.symbols('x')
10 fx = x**2 * sym.exp(-x**2)
11 dfx = sym.diff(fx)
12
13 crit_pnts = sym.solve(dfx)
14
15 xx = np.linspace(-4,4,1001)
16 fxx = sym.lambdify(x,fx)(xx)
17 dfxx = sym.lambdify(x,dfx)(xx)
18
19 print('critical points are ' + str(crit_pnts))
20
21 # EXERCISE : PLOT THE CRITICAL POINTS ON THE GIVEN FUNCTION CURVE.
22
23 plt.plot(xx,fxx,label='f(x)')
24 plt.plot(xx,dfxx,label='$f\''(x)$')
25
26 plt.legend()
27 plt.show()
```

10 Partial derivative

The Python program for this section:

```
1 # PARTIAL DERIVATIVE
2
3 # LIBRARIES
4 import sympy as sym
5 import matplotlib.pyplot as plt
6 import numpy as np
7
8 x,y = sym.symbols('x y')
9
10 fxy = x**3 + (x**2)*(y**4)
11
12 # partial derivative with respect to x.
13 fdx = sym.diff(fxy,x)
14
15 # partial derivative with respect to y.
16 fdy = sym.diff(fxy,y)
17
18 # plotting
19
20 p = sym.plotting.plot3d(fxy,(x,-3,3),(y,-3,3),title = '$f(x,y)=%s$'%(
    sym.latex(fxy)) )
21
22 p = sym.plotting.plot3d(fdx,(x,-3,3),(y,-3,3),title = '$f(x,y)=%s$'%(
    sym.latex(fdx)) )
23
24 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:

```
1 # INTEGRATION OF A POLYNOMIAL
2
3 # LIBRARIES
4 import sympy as sym
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 x = sym.symbols('x')
9 fx = x**2
10
11 # INDEFINITE INTEGRATION
12 idi = sym.integrate(fx)
13
14 xx = np.linspace(-4,4,1000)
15 fxx = sym.lambdify(x,fx)(xx)
16 idi_xx = sym.lambdify(x,idi)(xx)
17
18 plt.plot(xx,fxx,label='$f(x)=%s$'%sym.latex(fx))
19 plt.plot(xx,idi_xx,label='$ \int %s dx = %s+C $'%(sym.latex(fx),sym.
    latex(idi)))
20
21 plt.xlabel('x')
22 plt.ylabel('y')
23 plt.title('Indefinite integration')
24 plt.grid()
25 plt.legend()
26 plt.show()
```

11.2 Integration of a trigonometric function.

The Python program for this section:

```
1 # INTEGRATION OF A TRIGONOMETRIC FUNCTION
2
3 # LIBRARIES
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import sympy as sym
7
8
9 x = sym.symbols('x')
10 fx = sym.cos(x)
11
12 idi = sym.integrate(fx)
13
14 xx = np.linspace(0,2*np.pi,1000)
15 fxx = sym.lambdify(x,fx)(xx)
16 idi_xx = sym.lambdify(x,idi)(xx)
17
18 plt.plot(xx,fxx, label='$f(x)=%s$'%(sym.latex(fx)))
19 plt.plot(xx,idi_xx, label='$ \int %s dx = %s+C$'%(sym.latex(fx),sym.
    latex(idi)))
20
21 plt.xlabel('x')
22 plt.ylabel('y')
23
24 plt.grid()
25 plt.legend()
26 plt.show()
```


11.3 Definite Integration.

The Python program for this section:

```
1 # DEFINITE INTEGRATION
2
3 # LIBRARIES
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import sympy as sym
7 import sympy.plotting.plot as symplot
8
9 x = sym.symbols('x')
10 fx = x**3
11
12 di = sym.integrate(fx,(x,1,2))
13
14 p = symplot(fx,show=False)
15
16
17 p[0].label = '$f(x)=%s$ '%sym.latex(fx)
18
19 p.title = '$\int_{1}^{2} %s dx = %g$'%(sym.latex(fx),di)
20 p.xlim = [-3,3]
21 p.ylim = [-10,10]
22 p.legend = True
23 p.show()
```

11.4 Trapezoidal Method.

The Python program for this section:

```
1 # TRAPEZOIDAL METHOD
2
3 # LIBRARIES
4
5 # to define a function.
6 def f(x):
7     return 1/(1 + x**2)
8
9 def trapezoidal(x0,xn,n):
10     # calculating step size
11     h = (xn - x0) / n
12
13     # Finding sum
14     integration = f(x0) + f(xn)
15
16     for i in range(1,n):
17         k = x0 + i*h
18         integration = integration + 2 * f(k)
19
20     # Finding final integration value
21     integration = integration * h/2
22
23
24
25
26     return integration
27
28
29 lower_limit = float(input("lower limit = "))
30 upper_limit = float(input("upper limit = "))
31 sub_interval = int(input("sub interval = "))
32
33
34 result = trapezoidal(lower_limit,upper_limit,sub_interval)
35
36 print("Integration result by Trapezoidal method is = %0.6f"%(result))
```

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

The Python program for this section:

```
1
2 # Simpson's 1/3 Rule
3
4 # Define function to integrate
5 def f(x):
6     return 1/(1 + x**2)
7
8 # Implementing Simpson's 1/3
9 def simpson13(x0,xn,n):
10     # calculating step size
11     h = (xn - x0) / n
12
13     # Finding sum
14     integration = f(x0) + f(xn)
15
16     for i in range(1,n):
17         k = x0 + i*h
18
19         if i%2 == 0:
20             integration = integration + 2 * f(k)
21         else:
22             integration = integration + 4 * f(k)
23
24     # Finding final integration value
25     integration = integration * h/3
26
27     return integration
28
29 # Input section
30 lower_limit = float(input("Enter lower limit of integration: "))
31 upper_limit = float(input("Enter upper limit of integration: "))
32 sub_interval = int(input("Enter number of sub intervals: "))
33
34 # Call trapezoidal() method and get result
35 result = simpson13(lower_limit, upper_limit, sub_interval)
36 print("Integration result by Simpson's 1/3 method is: %0.6f" % (result))
37 )
```

12 The Fundamental Theorem of Calculus.

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

The Python program for this section:

```
1 # FUNDAMENTAL THEOREM OF CALCULUS
2
3 # LIBRARIES
4 import sympy as sym
5
6
7 x = sym.symbols('x')
8 fx = 2*x + sym.cos(x)
9
10 # integrate(differentiate(function))
11 dfx1 = sym.diff(fx)
12 idf1 = sym.integrate(dfx1)
13
14 # differentiate(integrate(function))
15 idf2 = sym.integrate(fx)
16 dfx2 = sym.diff(idf2)
17
18 if idf1==fx and dfx2==fx :
19     print('The fundamental theorem of calculus holds true.')
20 else:
21     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,

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

The Python program for this section:

```
1 # AREA BETWEEN TWO CURVES
2
3 # LIBRARIES
4 import numpy as np
5 import sympy as sym
6 import matplotlib.pyplot as plt
7
8
9 x = sym.symbols('x')
10 f = x**2
11 g = x+2
12
13 h = f-g
14
15
16 xx = np.linspace(-3,3,200)
17 y1 = sym.lambdify(x,f)(xx)
18 y2 = sym.lambdify(x,g)(xx)
19
20 # this gives the intersecting points.
21 idx = np.argwhere(np.diff(np.sign(y1-y2)) != 0)
22
23 x1 = xx[idx][0]
24 x2 = y1[idx][1]
25 print(x1,x2)
26
27 A = abs(sym.integrate(h,(x,x1,x2))) # this give the area.
28
29 with plt.xkcd():
30     fig,ax = plt.subplots()
31     ax.fill_between(xx,y1,y2,where=y2>=y1,facecolor = 'green') # shades
32     the area.
33
34     plt.plot(xx,y1,label='$f(x)=%s$'%sym.latex(f))
35     plt.plot(xx,y2,label='$g(x)=%s$'%sym.latex(g))
36     plt.plot(xx[idx],y1[idx],'ro')
37     plt.legend()
38     plt.title('The area between the two curves is %.4f square units.'%A)
39
40 plt.show()
```