Topic 4 : Solving calculus - Continuity and Derivatives

In [1]:
```
from datetime import date
today=date.today()
print("Today's date:",today)
```

```
Today's date: 2022-02-11
```

In [2]:
```
from math import *
import matplotlib.pyplot as plt
import numpy as np
```

A necessary, but not sufficient, condition for a function to be differentiable at a point is that it must be continuous at that point. That is, the function must be defined at that point and its left-hand limit and right hand limit must exist and be equal to the value of the function at that point. If f(x) is the function and x = a is the point we are interested in evaluating, this is mathematically stated as : $lim_{x\to a^+} f(x) = lim_{x\to a^-} f(x) = f(a)$

Consider a function f(x) = 1/x. As the value of x increases, the value of f(x) approaches 0. Using the limit notation, we'd write this as : $lim_{x\to\infty} \frac{1}{x} = 0$.

In [28]:
```
from sympy import Limit, Symbol, S
x= Symbol('x')
Limit(1/x, x, S.Infinity)
```

Out[28]:
$$\lim_{x\to\infty} \frac{1}{x}$$

In [29]:
```
l= Limit(1/x, x, S.Infinity)
l.doit()
```

Out[29]:
$$0$$

In [3]:
```
#Finding Right hand Limit
from sympy import Limit, Symbol, S
x= Symbol('x')
rlim=Limit(1/x, x, 0, dir='+').doit()
print(rlim)
```

```
oo
```

In [4]:
```
#Finding Left hand Limit
from sympy import Limit, Symbol, S
x= Symbol('x')
llim=Limit(1/x, x, 0, dir='-').doit()
print(llim)
```

```
-oo
```

Task for students : Create a code to find if the given function is continuous using the above two codes for right and left hand limits

(Hint : Use if-else condition)

In [8]:
```
# Simpler method to find continuity of a function f = 1/x
import sympy as sp
x = sp.Symbol("x")
f = 1/x
value = 0
def checkifcontinus(func,x,symbol):
    return (sp.limit(func, symbol, x).is_real)
print(checkifcontinus(f,value,x))
```

```
False
```

Q. Find the continuity of the following functions:

$1. f(x) = sinx$ at $x = 0$

$2. f(x) = x - 5$ at $x = 2$

$3. f(x) = 2x + 7$ at $x = 0$

$4. f(x) = \frac{5x}{3} + 7$ at $x = 3$

In [2]:
```
# Solution to (1)
import sympy as sp
from sympy import sin
x = sp.Symbol("x")
f = sin(x)
value = 0
def checkifcontinus(func,x,symbol):
    return (sp.limit(func, symbol, x).is_real)
print(checkifcontinus(f,value,x))
```

```
True
```

In [3]:
```
# Solution to (2)
import sympy as sp
from sympy import sin
x = sp.Symbol("x")
f = x-5
value = 2
def checkifcontinus(func,x,symbol):
    return (sp.limit(func, symbol, x).is_real)
print(checkifcontinus(f,value,x))
```

```
True
```

In [4]:
```
import sympy as sp
from sympy import sin
x = sp.Symbol("x")
f = 2*x+7
value = 0
def checkifcontinus(func,x,symbol):
    return (sp.limit(func, symbol, x).is_real)
print(checkifcontinus(f,value,x))
```

```
True
```

In [7]:
```
import sympy as sp
```

```
from sympy import sin
from fractions import Fraction
x = sp.Symbol("x")
f = Fraction(5, 3) *x +7
value = 3
def checkifcontinus(func,x,symbol):
    return (sp.limit(func, symbol, x).is_real)
print(checkifcontinus(f,value,x))
```

```
True
```

Definition of Derivatives

The derivative of a function y = f(x) expresses the rate of change in the dependent variable, y, with respect to the independent variable, x. It's denoted as either f'(x) or dy/dx. We can find the derivative of a function by creating an object of the Derivative class.

Example 1: $S(t) = 5t^2 + 2t + 8.$

In [1]:
```
# lim_{h->0} (f(x+h)-f(x))/h = f'(x), use the concept of limit to evaluate the deriv
from sympy import Symbol, Limit

t = Symbol('t')
St = 5*t**2 + 2*t + 8

t1 = Symbol('t1')
h = Symbol('h')

St1 = St.subs({t: t1})
St1_h = St.subs({t: t1 + h})

Limit((St1_h-St1)/h, h, 0).doit()
```

Out[1]: $10t_1 + 2$

In [3]:
```
from sympy import Symbol, Derivative
t = Symbol('t')
St = 5*t**2 + 2*t + 8
# We can find the derivative of a function by creating an object of the Derivative c
Derivative(St, t)
```

Out[3]: $\frac{d}{dt}\left(5t^2 + 2t + 8\right)$

In [4]:
```
d = Derivative(St, t)
d.doit() #We call the doit() method on the unevaluated Derivative object to find the
```

Out[4]: $10t + 2$

Now, if we want to calculate the value of the derivative at a particular value of t—say,t = t1 or t = 1—we can use the subs() method:

In [7]:
```
t1 = Symbol('t1') # defining symbol t1
d.doit().subs({t:t1})
```

Out[7]: $10t_1 + 2$

In [8]:
```
d.doit().subs({t:1})
```

Out[8]: 12

Example 2 : $(x^3 + x^2 + x)(x^2 + x)$.

In [13]:
```
from sympy import Derivative, Symbol
x = Symbol('x')
f = (x**3 + x**2 + x)*(x**2+x)
Derivative(f, x)
```

Out[13]: $\frac{d}{dx}\left(x^2 + x\right)\left(x^3 + x^2 + x\right)$

In [14]:
```
d = Derivative(f, x)
d.doit()
```

Out[14]: $(2x + 1)\left(x^3 + x^2 + x\right) + \left(x^2 + x\right)\left(3x^2 + 2x + 1\right)$

In [16]:
```
x1 = Symbol('x1')
d.doit().subs({x:x1})
```

Out[16]: $(2x_1 + 1)\left(x_1^3 + x_1^2 + x_1\right) + \left(x_1^2 + x_1\right)\left(3x_1^2 + 2x_1 + 1\right)$

In [38]:
```
d.doit().subs({x:1})
```

Out[38]: 21

Example 3: Find the derivatives of the following functions

a) $f(x) = 2x - \frac{3}{4}$

b) $f(x) = x^5(3 - 6x^{-9})$

c) $f(x) = sinxcosx$

d) $f(x) = 2tanx - 7secx$

Program to find the derivative of any function from the user.

In [7]:
```
from sympy import Symbol, Derivative, sympify,pprint
from sympy.core.sympify import SympifyError

def derivative(f, var):
    var = Symbol(var)
    d = Derivative(f, var).doit()
```

```
    pprint(d)

if __name__=='__main__':
    f = input('Enter a function: ')
    var = input('Enter the variable to differentiate with respect to: ')
    try:
        f = sympify(f)
    except SympifyError:
        print('Invalid input')
    else:
        derivative(f, var)
```

```
Enter a function: x**2 + 3*x
Enter the variable to differentiate with respect to: x
2·x + 3
```

1) We ask the user to input a function for which the derivative is to be found, and then we ask for the variable with respect to which the function is to be differentiated. At v, we convert the input function into a SymPy object using the $sympify()$ function.

2) We call this function in a $try...except$ block so that we can display an error message in case the user enters an invalid input. If the input expression is a valid expression, we call the derivative function passing the converted expression and the variable with respect to which the function is to be differentiated as arguments.

3) In the $derivative()$ function, we first create a $Symbol$ object that corresponds to the variable with respect to which the function is to be differentiated. We use the label var to refer to this variable. Next, we create a Derivative object that passes both the function to differentiate and the symbol object var. We immediately call the $doit()$ method to evaluate the derivative.

Example 4: Find the derivatives of the following functions

a) $f(x) = \frac{sinx}{x} + cosx$

b) $f(x) = e^x + e^y + e^{x+y}$

c) $f(x) = \frac{x}{1+x^2}$

d) $f(x) = 2tanx - 7secx$

Finding the Maxima and Minima

Find the maxima and minima of the function $x^5 - 30x^3 + 50x$.

In [1]:

```
from sympy import Symbol, solve, Derivative

x = Symbol('x')
f = x**5 - 30*x**3 + 50*x

d1 = Derivative(f, x).doit()
print(d1)

critical_points = solve(d1)
print(critical_points)
```

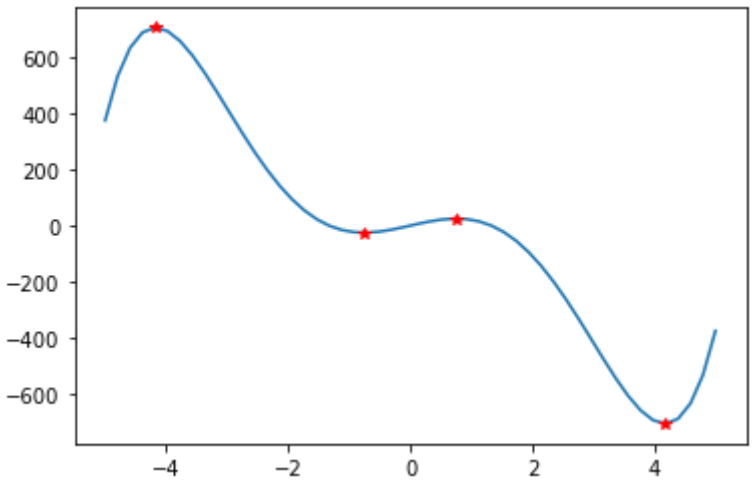
```
d2 = Derivative(f, x, 2).doit()
print(d2)

for i in range(len(critical_points)):
    print("critical point", i+1, "=", critical_points[i])

    print("second derivative at",critical_points[i], "=", d2.subs({x:critical_points
    #evalf() evaluates the final value of the expression
```

```
5*x**4 - 90*x**2 + 50
[-sqrt(9 - sqrt(71)), sqrt(9 - sqrt(71)), -sqrt(sqrt(71) + 9), sqrt(sqrt(71) + 9)]
20*x*(x**2 - 9)
critical point 1 = -sqrt(9 - sqrt(71))
second derivative at -sqrt(9 - sqrt(71)) = 127.661060789073

critical point 2 = sqrt(9 - sqrt(71))
second derivative at sqrt(9 - sqrt(71)) = -127.661060789073

critical point 3 = -sqrt(sqrt(71) + 9)
second derivative at -sqrt(sqrt(71) + 9) = -703.493179468151

critical point 4 = sqrt(sqrt(71) + 9)
second derivative at sqrt(sqrt(71) + 9) = 703.493179468151
```

Plot the curve and its critical points.

In [2]:

```
from matplotlib.pyplot import *
from numpy import *
a = linspace(-5,5)
b  = a**5 - 30*a**3 + 50*a
plot(a,b)

for i in range(len(critical_points)):
    plot(critical_points[i],f.subs({x:critical_points[i]}), color='r', marker = '*')
```



Find the maxima and minima of the function $5x^2 + 2x + 8$.

In [38]:

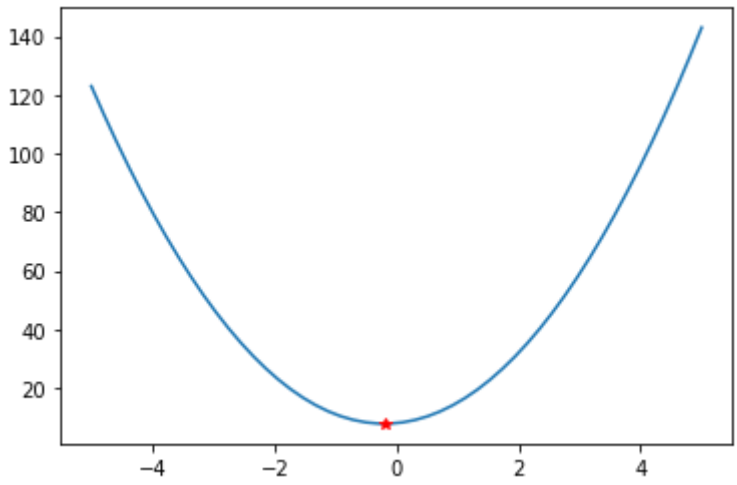
```
from sympy import Symbol, solve, Derivative

x = Symbol('x')
f = 5*x**2 + 2*x + 8
```

```
d1 = Derivative(f, x).doit()
print(d1)

critical_points = solve(d1)
print(critical_points)



d2 = Derivative(f, x, 2).doit()
print(d2)

for i in range(len(critical_points)):
    print("critical point", i+1, "=", critical_points[i])

    print("second derivative at",critical_points[i], "=", d2.subs({x:critical_points
    #evalf() evaluates the final final value of the expression
```

```
10*x + 2
[-1/5]
10
critical point 1 = -1/5
second derivative at -1/5 = 10.0000000000000
```

In [39]:

```
from matplotlib.pyplot import *
from numpy import *
a = linspace(-5,5)
b  = 5*a**2 + 2*a + 8
plot(a,b)

for i in range(len(critical_points)):
    plot(critical_points[i],f.subs({x:critical_points[i]}), color='r', marker = '*')
```



Find the maxima and minima of the function $(x^3 + x^2 + x)(x^2 + x)$.

In [6]:

```
from sympy import Symbol, solve, Derivative

x = Symbol('x')
f = (x**3 + x**2 + x)*(x**2+x)

d1 = Derivative(f, x).doit()
print("First derivative f' = ", d1,"\n")
```

```
critical_points = solve(d1)
print(critical_points,"\n")



d2 = Derivative(f, x, 2).doit()
print("Second derivative f'' = ", d2,"\n")

for i in range(len(critical_points)):
    print("critical point", i+1, "=", critical_points[i].evalf())

    print("second derivative at",critical_points[i].evalf(), "=", d2.subs({x:critica
    #evalf() evaluates the final final value of the expression
```

```
First derivative f' =  (2*x + 1)*(x**3 + x**2 + x) + (x**2 + x)*(3*x**2 + 2*x + 1)

[0, -8/15 + 26/(75*(-1/2 - sqrt(3)*I/2)*(107/125 + 3*sqrt(129)/25)**(1/3)) - (-1/2 -
sqrt(3)*I/2)*(107/125 + 3*sqrt(129)/25)**(1/3)/3, -8/15 - (-1/2 + sqrt(3)*I/2)*(107/
125 + 3*sqrt(129)/25)**(1/3)/3 + 26/(75*(-1/2 + sqrt(3)*I/2)*(107/125 + 3*sqrt(129)/
25)**(1/3)), -8/15 - (107/125 + 3*sqrt(129)/25)**(1/3)/3 + 26/(75*(107/125 + 3*sqrt
(129)/25)**(1/3))]

Second derivative f'' =  2*(x**3 + x**2 + x*(x + 1)*(3*x + 1) + x + (2*x + 1)*(3*x**
2 + 2*x + 1))

critical point 1 = 0
second derivative at 0 = 2.00000000000000

critical point 2 = -0.448840351216779 + 0.606699150112732*I
second derivative at -0.448840351216779 + 0.606699150112732*I = 0.71909379753823 -
2.92341284744047*I

critical point 3 = -0.448840351216779 - 0.606699150112732*I
second derivative at -0.448840351216779 - 0.606699150112732*I = 0.71909379753823 +
2.92341284744047*I

critical point 4 = -0.702319297566442
second derivative at -0.702319297566442 = -1.51818759507646
```

Plotting the above curve is possible with the below program, but all the critical points cannot be

plotted as few of them are complex numbers. Thus on executing the below given program we obtain an

error that 'cannot convert complex to float.'

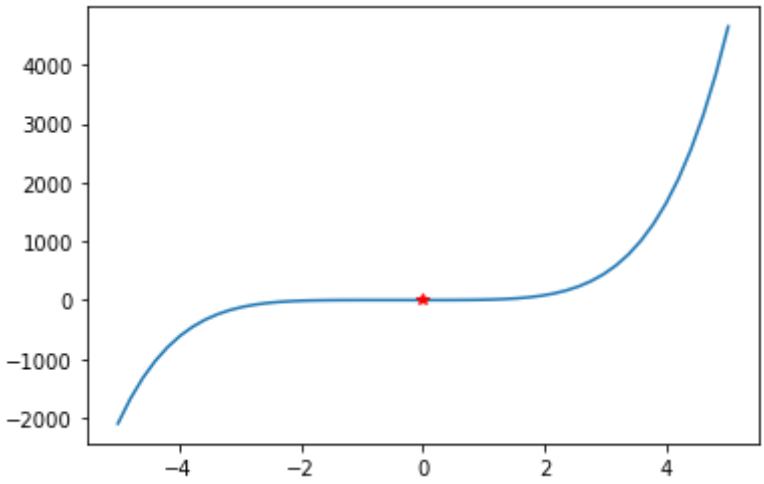
In [7]:

```
from matplotlib.pyplot import *
from numpy import *
a = linspace(-5,5)
b  = (a**3 + a**2 + a)*(a**2+a)
plot(a,b)

for i in range(len(critical_points)):
    plot(critical_points[i],f.subs({x:critical_points[i]}), color='r', marker = '*')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
C:\Users\JAMESJ~1\AppData\Local\Temp/ipykernel_4112/2330410110.py in <module>
      6
      7 for i in range(len(critical_points)):
```

```
----> 8     plot(critical_points[i],f.subs({x:critical_points[i]}), color='r', marke
r = '*')

~\anaconda3\lib\site-packages\matplotlib\pyplot.py in plot(scalex, scaley, data, *ar
gs, **kwargs)
   3017 @_copy_docstring_and_deprecators(Axes.plot)
   3018 def plot(*args, scalex=True, scaley=True, data=None, **kwargs):
-> 3019     return gca().plot(
   3020         *args, scalex=scalex, scaley=scaley,
   3021         **({"data": data} if data is not None else {}), **kwargs)

~\anaconda3\lib\site-packages\matplotlib\axes\_axes.py in plot(self, scalex, scaley,
data, *args, **kwargs)
   1605         lines = [*self._get_lines(*args, data=data, **kwargs)]
   1606         for line in lines:
-> 1607             self.add_line(line)
   1608         self._request_autoscale_view(scalex=scalex, scaley=scaley)
   1609         return lines

~\anaconda3\lib\site-packages\matplotlib\axes\_base.py in add_line(self, line)
   2099             line.set_clip_path(self.patch)
   2100
-> 2101         self._update_line_limits(line)
   2102         if not line.get_label():
   2103             line.set_label('_line%d' % len(self.lines))

~\anaconda3\lib\site-packages\matplotlib\axes\_base.py in _update_line_limits(self,
 line)
   2121         Figures out the data limit of the given line, updating self.dataLim.
   2122         """
-> 2123         path = line.get_path()
   2124         if path.vertices.size == 0:
   2125             return

~\anaconda3\lib\site-packages\matplotlib\lines.py in get_path(self)
   1020         """
   1021         if self._invalidy or self._invalidx:
-> 1022             self.recache()
   1023         return self._path
   1024

~\anaconda3\lib\site-packages\matplotlib\lines.py in recache(self, always)
    661         if always or self._invalidx:
    662             xconv = self.convert_xunits(self._xorig)
--> 663             x = _to_unmasked_float_array(xconv).ravel()
    664         else:
    665             x = self._x

~\anaconda3\lib\site-packages\matplotlib\cbook\__init__.py in _to_unmasked_float_arr
ay(x)
   1331         return np.ma.asarray(x, float).filled(np.nan)
   1332     else:
-> 1333         return np.asarray(x, float)
   1334
   1335

~\anaconda3\lib\site-packages\numpy\core\_asarray.py in asarray(a, dtype, order, lik
e)
    100         return _asarray_with_like(a, dtype=dtype, order=order, like=like)
    101
--> 102     return array(a, dtype, copy=False, order=order)
    103
    104
```

```
~\anaconda3\lib\site-packages\sympy\core\expr.py in __float__(self)
    356             return float(result)
    357         if result.is_number and result.as_real_imag()[1]:
--> 358             raise TypeError("can't convert complex to float")
    359         raise TypeError("can't convert expression to float")
    360

TypeError: can't convert complex to float
```



In [ ]: