TAYLOR SERIES EXPANSION for **asin(x)** function

```
## -*- coding: Taylor series expansion for asin-*-
import sympy as sy
import numpy as np
from sympy.functions import asin,cos
import matplotlib.pyplot as plt
```

We have specified what style we want to use in plotting, Matplotlib comes with a few styles already.

```
plt.style.use("ggplot")
```

Title for the 1st part

```
print("Taylor series expansion of asin(x)")
```

```
Taylor series expansion of asin(x)
```

Define the variable and the function to approximate

```
x = sy.Symbol('x')
f = asin(x)
```

Factorial function of given number

```
def factorial(n):
     if n <= 0:
          return 1
     else:
          return n*factorial(n-1)
```

Taylor approximation at x0 of the function 'function'

```
def taylor(function,x0,n):
    i = 0
    p = 0
```

```
    while i <= n:
        p = p + (function.diff(x,i).subs(x,x0))/(factorial(i))*(x-x0)**i
        i += 1
    return p
```

Plot results

The np.linspace() function returns evenly spaced numbers over the specified interval

```
def plot():
    x_lims = [-np.pi/2,np.pi/2]
    x1 = np.linspace(-1,1,800)
    y1 = []
```

Approximate up until 10 starting from 1 and using steps of 2

Python built-in function range() generates the integer numbers between the given start integer(1) to the stop integer(10)

```
for j in range(1,10,2):
        func = taylor(f,0,j)
        print('n='+str(j),func)
        for k in x1:
            y1.append(func.subs(x,k))
        plt.plot(x1,y1,label='order '+str(j))
        y1 = []
```

Plot the function to approximate (asin)

```
    plt.plot(x1,np.arcsin(x1),label='asin of x')
    plt.xlim(x_lims)
    plt.ylim([-np.pi/2,np.pi/2])
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.grid(True)
    plt.title('Taylor series of asin')
    plt.show()
plot()
```

CALCULATION of **asin(x)** value

```
from math import asin
print("Taylor series calculation of asin(x)")
def my_asin(x):
   k = 0
   a = x
   S = a
   print("a0 = %6.2f S0 = %6.2f"%(a,S))
```

The format specifier in the above print slot says how to format the value: %6.2f means:

1. "6" indicates 6 spaces allocated to this value.
2. "f" indicates treat is as a floating-point number.
3. "2" indicates places to the right of the decimal point (i.e.,)only two digits after the decimal point

while loop repeats the sequence of actions many times until some condition evaluates to False, here *(k<4)*

From previous commond where (k=0) the value of a and S are saved as:

- a0 = x (input = 0.75) = 0.75
- S0 = a0 = 0.75

```
while k < 4:
    k = k + 1
    R = ((x**(2))*((2*k-1)**(2)))/((2*k)*(2*k+1))
    a = a * R
    S = S + a
    print("a%d = %6.2f S%d = %6.2f"%(k,a,k,S))
  print("The Final answer")
  return S
```

From the above commands where (k<4), we obtain the values of a and S as follows:

1. k = 1, x = 0.75
   a1 = a0 * R = x * ((x**(2))*((2k-1)**(2)))/((2*k*)(2*k+1))
   a1 = 0.07
   S1 = a1 + S0 = 0.07 + 0.75
   S1 = 0.82

2. k = 2, x = 0.75
   a2 = a1 * R = x * ((x**(2))*((2k-1)**(2)))/((2*k*)(2*k+1)) a2 = 0.017 = 0.02
   S2 = a2 + S1 = 0.02 + 0.82
   S2 = 0.84

3. k = 3, x = 0.75
   a3 = a2 * R = x * ((x**(2))*((2k-1)**(2)))/((2*k*)(2*k+1))
   a3 = 0.006 = 0.01
   S3 = a3 + S2 = 0.01 + 0.84
   S3 = 0.85

4. k = 4, x = 0.75
   a4 = a3 * R = x * ((x**(2))*((2k-1)**(2)))/((2*k*)(2*k+1))
   a4 = 0.003 = 0.00
   S4 = a4 + S3 = 0.00 + 0.85
   S4 = 0.85

the above calculations explains the print function in the above code:

print("a%d = %6.2f S%d = %6.2f"%(k,a,k,S))

- a%d = %6.2f %(k,a):
  indicates that "a" of k value, (for example, if k = 1, a = a1 and the a1 must be of 6 digit but after the decimal point there should be only two digits. As we see in the above calculations actual a2=0.017 but "%6.2f" changes the a2=0.02).
- S%d = %6.2f %(k,S):
  indicates that "S" of k value, (for example, if k = 1, S = a1 and the S1 must be of 6 digit but after the decimal point there should be only two digits).

```
x = float(input("Enter argument (x): "))
y = asin(x)
print("standard asin(%.2f) = %6.2f"%(x,y))
yy = my_asin(x)
print("my asin(%.2f) = %6.2f"%(x,yy))
```

- The float()built-in function convert the input string to float type.

- *%.2f* ensures that if the input entered is "0.746" it convert the input to "0.75" because *%.2f* indicates that only two digits after the decimal point. So, it round up the input from 0.746 to 0.75.

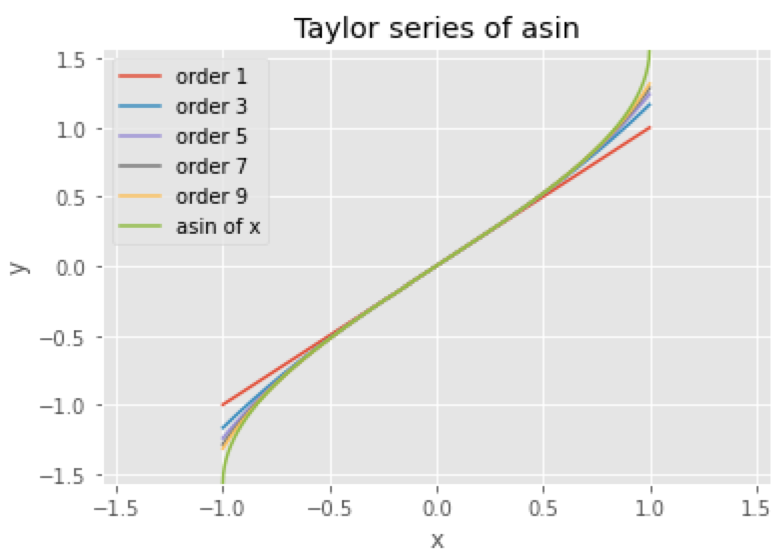
RESULTS

Taylor series expansion of asin(x)

- n=1 x
- n=3 x**3/6 + x
- n=5 3x*5/40 + x**3/6 + x
- n=7 5x**7/112 + 3x**5/40 + x**3/6 + x
- n=9 35x**9/1152 + 5x**7/112 + 3*x**5/40 + x**3/6 + x

Image description:

As the degree of the Taylor polynomial rises, it approaches the correct function. This image shows asin(x) and its Taylor approximations, polynomials of degree 1, 3, 5, 7 and 9.



Taylor series calculation of asin(x)

Enter argument (x): 0.75
Standard asin(0.75) = 0.85

k	k=0	k=1	k=2	k=3	k=4
a	a0=0.75	a1=0.07	a2=0.02	a3=0.01	a4=0.00
S	S0=0.75	S1=0.82	S2=0.84	S3=0.84	S4=0.85

The Final answer

```
my asin(0.75) =   0.85
```