# Symbolic Programming Paradigm

# Introduction

Symbolic computation deals with the computation of mathematical objects symbolically. This means that the mathematical objects are represented exactly, not approximately, and mathematical expressions with unevaluated variables are left in symbolic form.

Concepts of symbolic programming paradigm:

- As A calculator and symbols
- Algebraic Manipulations - Expand and Simplify
- Calculus – Limits, Differentiation, Series , Integration
- Equation Solving – Matrices

# Calculator and Symbols

**Rational**

```
>>import sympy as sym
>>a = sym.Rational(1, 2)
>>a
```

**Output:** 1/2

**Constants like pi,e**

```
>>sym.pi**2
```

**Output:** pi**2

```
>>sym.pi.evalf()
```

**Output:** 3.14159265358979

**X AND Y**

```
>> x = sym.Symbol('x')
>>y = sym.Symbol('y')
>>x + y + x – y
```

**Output:** 2*x

# Algebraic Manipulations

**EXPAND**

```
>> sym.expand((x + y) ** 2)
```

**Output**: x**2+2*x*y+y**2

```
>> sym.expand((x + y) ** 3)
```

**Output**: x**3 + 3*x**2*y + 3*x*y**2 + y**3

**WITH TRIGNOMETRY LIKE SIN,COSINE**

eg. COS(X+Y)= - SIN(X)*SIN(Y)+COS(X)*COS(Y)

```
>> sym.expand(sym.cos(x + y), trig=True)
```

**Output**: -sin(x)*sin(y) + cos(x)*cos(y)

**SIMPLIFY**

```
from sympy import sin, cos, simplify
expr = sin(x) / cos(x)
expr = simplify(expr)
```

**Output**: tan(x)

# Calculus

**LIMITS**

limit(function, variable, point)

limit( sin(x)/x , x, 0) **Output**: 1

**Differentiation**

diff(func,var)

diff(sin(x),x) **Output**: cos(x)

**Series**

series(expr,var)

series(cos(x),x) **Output**: $1 – (x^2 / 2 !) + (x^4 / 4 !) – (x^6 / 6 !) +…$

**Integration**

Integrate(expr,var)

Integrate(sin(x),x) **Output**: -cos(x)

# Equation Solving

**Matrix.col():**

**Syntax**: sympy.Matrix().col() #return the column of the matrix

Example:

```
from sympy import *
val = Matrix([[1, 2], [2, 1]]).col(1)
print(val)                                  #[2],[1]
```

**Matrix().col_insert():**

**Syntax :** Matrix().col_insert() # Return a new matrix

**Example:**

```
from sympy import *
mat = Matrix([[1, 2], [2, 1]])
new_mat = mat.col_insert(1, Matrix([[3], [4]]))
print(new_mat)                              #[1,3,2],[2,4,1]
```

**Matrix().columnspace():**

**Syntax:** Matrix().columnspace() # Returns a list of column vectors that span the columnspace of the matrix.

Example:

```
from sympy import *
M = Matrix([[1, 0, 1, 3], [2, 3, 4, 7], [-1, -3, -3, -4]])
print("Matrix : {} ".format(M))
```

Output:

Matrix([[1, 0, 1, 3], [2, 3, 4, 7], [-1, -3, -3, -4]])

Columnspace of a matrix : [Matrix([

[ 1],

[ 2],

[-1]]), Matrix([

[ 0],

[ 3],

[-3]])]

**Square root**

Square root is a number which produces a specified quantity when multiplied by itself

```
from sympy import sqrt, pprint, Mul
x = sqrt(2)
y = sqrt(2)
pprint(Mul(x,  y, evaluate=False))
print('equals to ')
print(x * y)
```

Output:

```
2
```

# SymPy canonical form of expression

An expression is automatically transformed into a canonical form by SymPy.

```
from sympy.abc import a, b
expr = b*a + -4*a + b + a*b + 4*a + (a + b)*3
print(expr)
```

Output:

```
2*a*b + 3*a + 4*b
```