Unit V

Symbolic Programming Paradigm

Topics Covered

- Symbolic Programming Paradigm
- Symbolic Maths, algebraic manipulations, limits, differentiation, integration, series
- SymPy usage for symbolic maths :Equation Solving, Matrices
- Other languages: Aurora, LISP, Wolfram
- Demo: Symbolic Programming in Python

Reference Book: “Doing Math with Python “by Amit Saha

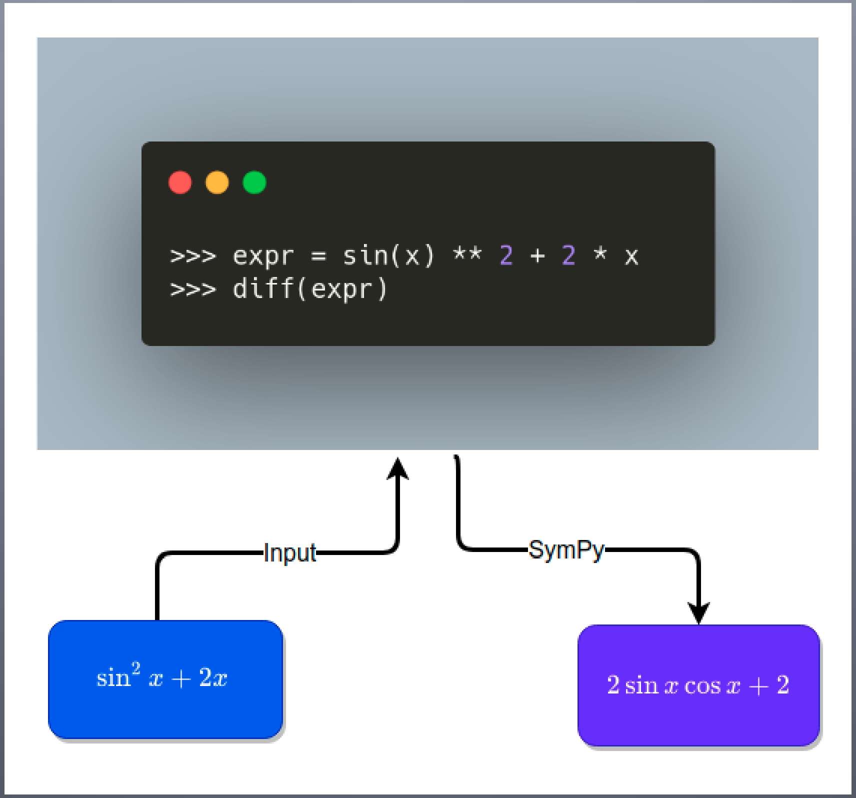
Symbolic Programming Paradigm

- Symbolic programming is a paradigm that describes programs able to manipulate formulas and program components as data.
- Complex processes can be developed that build other more intricate processes by combining smaller units of logic or functionality
- Programs can thus effectively modify themselves, and appear to "learn”
- Suitable for applications such as artificial intelligence, expert systems, natural-language processing and computer games.
- Languages that support this paradigm include Wolfram Language, Lisp and Prolog.

Symbolic vs. functional vs. logic programming

- **Functional programming**
  - Programming by defining functions (e.g., "the reciprocal of X is 1 divided by X") rather than by telling the computer what to do.
  - A functional program usually has lots of short function definitions rather than long stretches of sequential code.
  - LISP is both a symbolic and a functional language.
- **Logic programming**
  - Programming controlled by a model of logical reasoning (e.g., "I can conclude that Y is the reciprocal of X if I can do such-and-such…").
  - Prolog, is both a logic programming language and a symbolic programming language.

Symbolic Programming in Python



SymPy: Symbolic Computation in Python

- SymPy is the Python module to compute mathematical objects symbolically
- To install SymPy, type: pip install sympy
- Start with importing all methods provided by SymPy: from sympy import *

Basic Operations

- Normally, when computing a square root, we get a decimal:

```
18 ** (1 / 2)
```

executed in 5ms, finished 19:13:30 2021-07-01

```
4.242640687119285
```

- But with SymPy, we can get a simplified version of the square root instead:

```
sqrt(18)
```

executed in 186ms, finished 19:13:30 2021-07-01

$3\sqrt{2}$

- It is because SymPy tries to represent mathematical objects exactly instead of approximately.

Basic Operations…

- We will get a fraction instead of a decimal when dividing 2 numbers using SymPy.

```
25 / 15
```

executed in 5ms, finished 05:55:24 2021-07-02

```
1.6666666666666667
```

```
frac = Rational(25, 15)
frac
```

executed in 5ms, finished 05:55:24 2021-07-02

$$\frac{5}{3}$$

Exercises

- Calculate $\sqrt{2}$ with 100 decimals.
- Calculate $1/2 + 1/3$ in rational arithmetic.

Symbols

- The real power of SymPy is its ability to work with symbols.
- In contrast to other Computer Algebra Systems, in SymPy you have to declare symbolic variables explicitly
- To create symbols, use the method symbols()

```
x = sym.Symbol('x')

y = sym.Symbol('y')
```

- Then you can manipulate them:
  - >>> x + y + x - y
  - >>> Output :2*x

Symbols...

- **Example**

```
x, y = symbols("x y")
expr = 3 * x + y
expr
```

executed in 29ms, finished 19:13:42 2021-07-01

$$3x + y$$

```
expr + 2
```

executed in 14ms, finished 19:37:20 2021-07-01

$$3x + y + 2$$

Equations

- **Expand, Factor, and Simplify**

- To expand the equation

$$x(3x + y) = 3x^2 + xy$$

- SymPy allows us to expand an equation using expand :

```
expansion = expand(x*expr)
expansion
```

executed in 4ms, finished 14:44:02 2021-07-01

$$3x^2 + xy$$

Equations...

- Factor our expression by using factor :

```
factor(expansion)
```

executed in 6ms, finished 14:44:02 2021-07-01

$$x(3x + y)$$

- simplify an equation using simplify :

```
expr = (6*x**2 + 3*x)/(3*x)
expr
```

executed in 7ms, finished 14:44:02 2021-07-01

$$\frac{6x^2 + 3x}{3x}$$

```
simplify(expr)
```

executed in 11ms, finished 14:44:02 2021-07-01

$$2x + 1$$

Equations...

- **Solve an Equation :**To solve an equation, use solve :

```
eq = (2 * x + 1) * 3 * x
eq
```

executed in 29ms, finished 19:59:13 2021-07-01

$$x\,(6x + 3)$$

```
solve(eq, x)
```

executed in 10ms, finished 14:44:08 2021-07-01

```
[-1/2, 0]
```

Equations...

- **Systems of linear equations**
  - Sympy is able to solve a large part of polynomial equations, and is also capable of solving multiple equations with respect to multiple variables giving a tuple as second argument
  - Example

```
solution = sym.solve((x + 5 * y - 2, -3 * x + 6 * y - 15), (x, y))
solution[x], solution[y]
```

<u>Output:</u>(-3, 1)

Equations...

- **Substitution**
- If we substitute the equation $x(6x+3)$ by 2?
- We can figure that out using eq.subs(x, 2) :

```
eq.subs(x, 2)
```

executed in 11ms, finished 18:38:55 2021-07-01

30

- We can also substitute x with another variable to get an expression like below:

```
eq.subs(x**2, 2)
```

executed in 43ms, finished 18:39:15 2021-07-01

$x(6x+3)$

Exercises

- Calculate the expanded form of $(x+y)^6$
- Simplify the trigonometric expression

$$\sin(x)/\cos(x)$$

Trigonometric

- To simplify expressions using trigonometric identities, use trigsimp()

```
trigsimp(1/sec(x))
```

executed in 9ms, finished 14:53:51 2021-07-01

$$\cos(x)$$

```
trigsimp(sin(x)/cos(x))
```

executed in 16ms, finished 14:54:17 2021-07-01

$$\tan(x)$$

```
trigsimp(sin(x)**2 + cos(x)**2)
```

$$1$$

```
trigsimp(1 + cot(x) ** 2)
```

executed in 96ms, finished 20:09:08 2021-07-01

$$\frac{1}{\sin^2(x)}$$

Derivatives, Integrals, and Limit

- What is the derivative of the expression below?

```
expr = sin(x) ** 2 + 2 * x
expr
```

executed in 24ms, finished 20:22:25 2021-07-01

$$2x + \sin^2(x)$$

- use SymPy to figure it out.

```
res = diff(expr)
res
```

executed in 37ms, finished 20:22:25 2021-07-01

$$2\sin(x)\cos(x) + 2$$

Derivatives, Integrals, and Limit…

- Integral of the derivate

```
integrate(res)
```

executed in 35ms, finished 18:50:49 2021-07-01

$$2x + \sin^2(x)$$

Example #4 : Find derivative, integration, limits, quadratic equation.

```
# import everything from sympy module
from sympy import *
# make a symbol
x = Symbol('x')

# make the derivative of sin(x)*e ^ x
ans1 = diff(sin(x)*exp(x), x)
print("derivative of sin(x)*e ^ x : ", ans1)

# Compute (e ^ x * sin(x)+ e ^ x * cos(x))dx
ans2 = integrate(exp(x)*sin(x) + exp(x)*cos(x), x)
print("indefinite integration is : ", ans2)

# Compute definite integral of sin(x ^ 2)dx
# in b / w interval of ? and ?? .
ans3 = integrate(sin(x**2), (x, -oo, oo))
print("definite integration is : ", ans3)

# Find the limit of sin(x) / x given x tends to 0
ans4 = limit(sin(x)/x, x, 0)
print("limit is : ", ans4)

# Solve quadratic equation : x ^ 2?2 = 0
ans5 = solve(x**2 - 2, x)
print("roots are : ", ans5)
```

Output :

```
derivative of sin(x)*e^x :  exp(x)*sin(x) + exp(x)*cos(x)
indefinite integration is :  exp(x)*sin(x)
definite integration is :  sqrt(2)*sqrt(pi)/2
limit is :  1
roots are :  [-sqrt(2), sqrt(2)]
```

Series Expansion

- Consider the following series:

$$x + \frac{x^2}{2} + \frac{x^3}{3} + \frac{x^4}{4} + \ldots + \frac{x^n}{n}.$$

- Write a program that will ask a user to input a number, n, and print this series for that number.

Series Expansion…

```
from sympy import Symbol, pprint, init_printing
def print_series(n):
    # Initialize printing system with reverse order
    init_printing(order='rev-lex')
    x = Symbol('x') series = x
    for i in range(2, n+1):
        series = series + (x**i)/I
        pprint(series)
if __name__ == '__main__':
    n = input('Enter the number of terms you want in
    the series: ')
    print_series(int(n))
```

Series Expansion...

- Calculating the Value of a Series

```
# Evaluate the series at x_value
series_value = series.subs({x:x_value})
print('Value of the series at {0}: {1}'.format(x_value,
   seri es_value))
if __name__ == '__main__': n = input('Enter the
   number of terms you want in the series: ')
x_value = input('Enter the value of x at which you
   want to ev aluate the series: ')
print_series(int(n), float(x_value))
```

Matrices

- Matrices are created as instances from the Matrix class:

```
>>> sym.Matrix([[1, 0], [0, 1]])
[1  0]
[    ]
[0  1]
```

- You can also put Symbols in it:

```
>>> x, y = sym.symbols('x, y')
>>> A = sym.Matrix([[1, x], [y, 1]])
>>> A
[1  x]
[    ]
[y  1]
>>> A**2
[x*y + 1    2*x  ]
[                ]
[  2*y    x*y + 1]
```

Matrices...

- **Example**

```
def fibonacci(n):
    if n<1:
      return n
    iterator =sympy.Matrix([[1,1],[1,0]])
    base_case = sympy.Matrix([1,1])
    return (iterator**(n-1)*base_case)[0]
fibonacci(10**5)
```

Special Functions

- Find the factorial of a number

```
factorial(x)
```

executed in 18ms, finished 20:19:47 2021-07-01

$x!$

```
factorial(3)
```

executed in 10ms, finished 18:33:49 2021-07-01

6

- Rewrite the expression in terms of another

```
tan(x).rewrite(cos)
```

executed in 35ms, finished 18:35:13 2021-07-01

$$\frac{\cos\left(x - \frac{\pi}{2}\right)}{\cos\left(x\right)}$$

Example

- Program to find intersection of two line segments

```
from sympy import Point, Line
line1 = Line(Point(-3, 0.5), Point(1, 1))
line2 = Line(Point(-2,-5), Point(3, 3))
Intersect=line1.intersection(line2)
print(intersect)
```

Exercises

1. Calculate $\lim_{x \to 0} \sin(x)/x$
2. Calculate the derivative of $log(x)$ for *x*

3. Solve the system of equations ,

$$x + y = 2$$

$$2 \cdot x + y = 0$$

4. Are there boolean values x, y that make (~x | y) & (~y | x) true?