ETH zürich

Introduction to Scientific Computation
Lecture 8
Fall 2018

Symbolic computations, graph computations

Mikhail Usvyatsov November 28, 2018

SymPy

1. Evaluate expressions with arbitrary precision.
2. Perform algebraic manipulations on symbolic expressions.
3. Perform basic calculus tasks (limits, differentiation and integration) with symbolic expressions.

4. Solve polynomial and transcendental equations.
5. Solve some differential equations.

What is SymPy? SymPy is a Python library for symbolic mathematics. It aims to be an alternative to systems such as Mathematica or Maple while keeping the code as simple as possible and easily extensible. SymPy is written entirely in Python and does not require any external libraries.

3 basic numerical types

- Real
- Rational
- Integer

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

>>> a
1/2

>>> a*2
1
```

ETH zürich

Arbitrary precision of math constants

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

>>> sym.pi.evalf()
3.14159265358979

>>> (sym.pi + sym.exp(1)).evalf()
5.85987448204884
```

Algebraic manipulations

Expand

```
>>> sym.expand((x + y) ** 3)
 3      2           2    3
x  + 3*x *y + 3*x*y  + y
>>> 3 * x * y ** 2 + 3 * y * x ** 2 + x ** 3 + y ** 3
 3      2           2    3
x  + 3*x *y + 3*x*y  + y
```

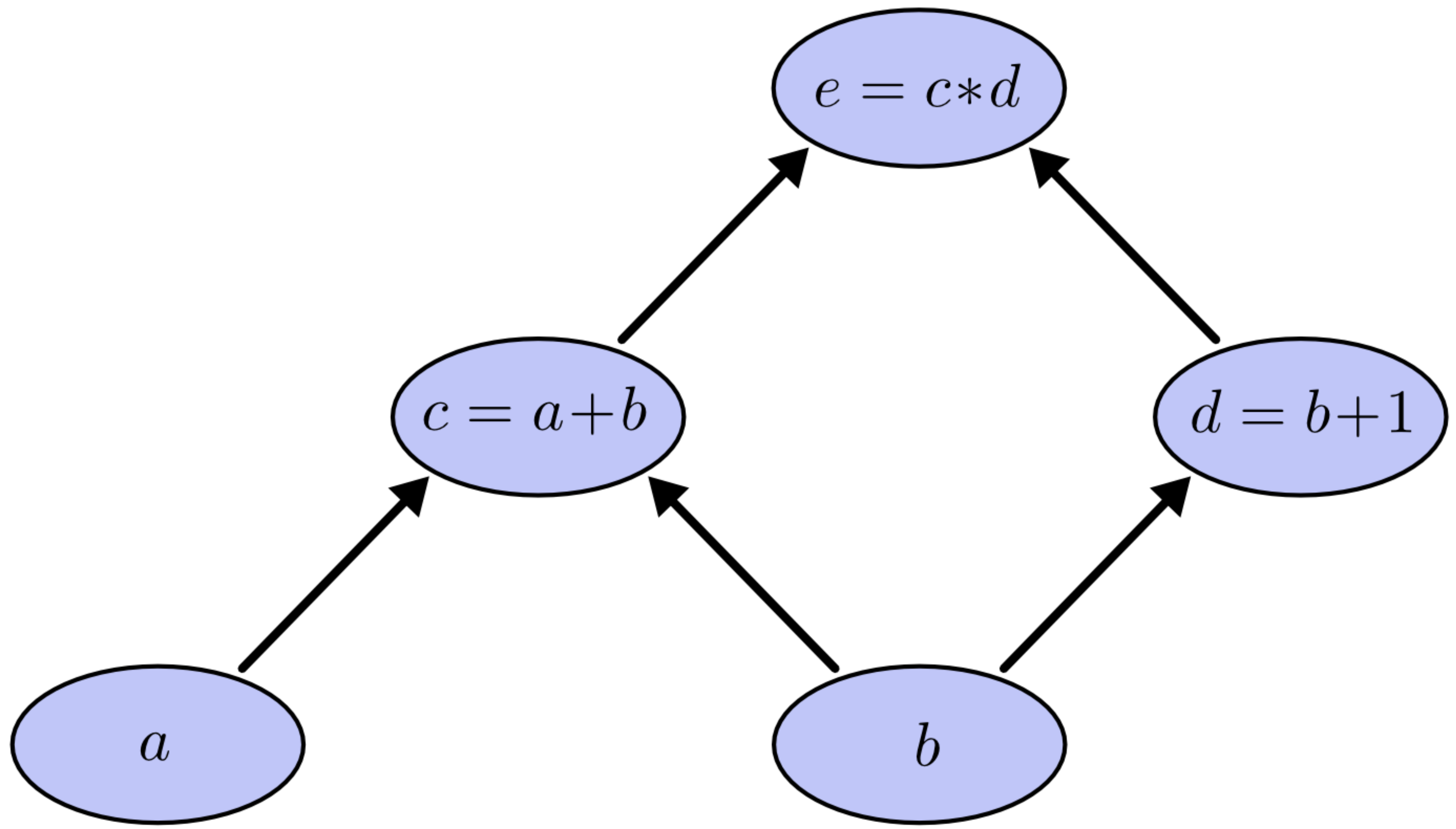
Simplify

```
>>> sym.simplify((x + x * y) / x)
y + 1
```

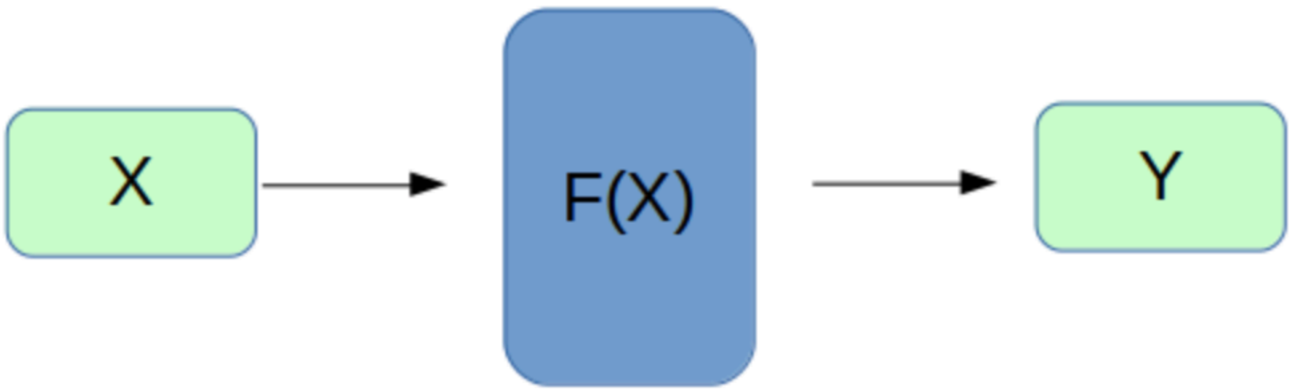
Calculus

- lim
- diff
- series
- Int
- eqs + system of eqs

Graph Computations



Graph Computations



1. Define the graph, that solves the problem you need
2. You provide the graph with the input

e.g.
```
x = 5
```
or
```
x = 5 * np.ones(100)
```

Tensorflow

Is a library that allows to perform symbolic computations.
Some important features:

- Use of GPU for computations
- Huge community
- Supported by Google

It allows

- Symbolically define mathematical functions
- Automatically derive gradient expressions
- Execute expression

Tensorflow

Datatypes

- Variable
- constant
- placeholder

Tensorflow math

In [3]:
```
import tensorflow as tf

# Note that tensorflow is fully typed
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)

z = x + y
w = z * x
a = tf.sqrt(w)
b = tf.exp(a)
c = a ** b
d = tf.log(c)
```

Tensorflow

Datatypes

- Variable
- constant
- placeholder

Tensorflow graph execution

In [4]:
```
import tensorflow as tf

def f(a, b):
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)

    with tf.Session() as session:
        # first arg is SYMBOLIC output
        # second arg is dict of SYMBOLIC inputs
        return session.run(x + y, {x : a,y : b})

# Call it with NUMERICAL values
# Get a NUMERICAL output
f(1., 2.) # => 3.0
```

Out[4]: 3.0

Tensorflow

Numpy	TensorFlow
`a = np.zeros((2,2)); b = np.ones((2,2))`	`a = tf.zeros((2,2)), b = tf.ones((2,2))`
`np.sum(b, axis=1)`	`tf.reduce_sum(a,reduction_indices=[1])`
`a.shape`	`a.get_shape()`
`np.reshape(a, (1,4))`	`tf.reshape(a, (1,4))`
`b * 5 + 1`	`b * 5 + 1`
`np.dot(a,b)`	`tf.matmul(a, b)`
`a[0,0], a[:,0], a[0,:]`	`a[0,0], a[:,0], a[0,:]`

Tensorflow

You just defined the recipe with the syntax close to numpy.
But one have to cook the recipe in the end!

Tensorflow

You just defined the recipe with the syntax close to numpy.
But one have to cook the recipe in the end!

```python
import tensorflow as tf

session = tf.Session()
session.run(expr, feed_dict)
```

Tensorflow

- All computations add nodes to global default
- Tf variables must be initialised before usage

```
W = tf.Variable(tf.zeros((2,2)), name="weights")
sess.run(tf.initialize_all_variables())
```

ETH zürich

Tensorflow

- One can also define a constant with a value assigned on creation with **tf.Constant**
- Placeholders are the variables which values should be provided by users in the **feed_dict**

Graph Computations

Pros:

- Optimised computation possible
- Faster execution
- Can reuse graphs

Cons:

- Need to maintain graph
- Hard to debug
- Different way of thinking