Lecture 15
Symbolic Computations

Instructors: Claus Fuhrer and Alexandros Sopasakis, LTH- Lund University

Let us look at a quick example of symbolic computations with Python before we get into the details!
Suppose we would like to integrate a given function \( f(x) \) from \( x=1 \) to \( x=4 \). We already know by now that we can find the answer if we do the following,

In [26]: from scipy.integrate import quad
In [29]: quad(lambda x : 1/(x**2+x+1),a=0, b=4)

Out[29]:

(0.9896614396122965, 1.17356634422835e−08)

Note: in order to continue it might be necessary for some to restart the kernel!
Do that now please!
Finally we now import the Python symbolic package as shown below.

In [1]: from sympy import *  # importing the symbolic package
In [2]: # Use one of the lines below
   # (depending on your version of python)
   # in order to produce better printing
   #init_session()  # OR
   init_printing()

To perform symbolic arithmetic we must explicitly define our symbols as follows

In [3]: x = symbols('x')
In [4]: type(x)  # to check what type x really is
Out[4]: sympy.core.symbol.Symbol

With this symbolic package we even get all the trigonometric (and other) identities for free! For example,

In [5]: simplify(cos(x)**2 + sin(x)**2)
We can define functions as follows below.  
This is an example of how to define a function in x using the Lambda keyword.

In [6]: f = Lambda(x, 1/(x**2 + x + 1))

Note the similarity with lambda which we used previously to define the one line function previously.  
Let us look at that function f(x) we just defined. To do that simply write,

In [14]: f

Out[14]:

\(\left( x \mapsto \frac{1}{x^2 + x + 1}\right)\)

Integrals, derivatives etc...

In [15]: # The indefinite integral of f(x) can be found by,
    integrate(f(x),x)

Out[15]:

\(\frac{2\sqrt{3}}{3} \tan \left( \frac{2x}{3} \sqrt{\frac{3}{\sqrt{3} + \frac{\sqrt{3}}{3}}} \right)\)

To further manipulate the above expression it would be best to make it depend on the variable x. We can define therefore a new function pf(x) depending on x as follows:

In [16]: pf = Lambda(x, integrate(f(x),x)) # Defining the integral of f(x) in a new function pf

Out[16]:

\(\left( x \mapsto \frac{2\sqrt{3}}{3} \tan \left( \frac{2x}{3} \sqrt{\frac{3}{\sqrt{3} + \frac{\sqrt{3}}{3}}} \right) \right)\)

Note that the above gives the equation in x of the general integral of our function f(x).
We can now compute the derivative with respect to x of that new function pf(x)

In [17]: diff(pf(x),x) # Taking the derivative of pf(x)

Out[17]:

\(\frac{4}{3 \left( \left( \frac{2x}{3} \sqrt{\frac{3}{\sqrt{3} + \frac{\sqrt{3}}{3}}} \right)^2 + 1 \right)}\)

Great thing about symbolic computations is that you can really re-produce all the work in analysis and linear algebra you would typically do by hand.  
For instance we can now simplify the complicated looking expression we obtained above:
In [18]: simplify(diff(pf(x),x))

Out[18]:
\[
\frac{1}{x^2 + x + 1}
\]

Note that after integrating \(f(x)\) and differenting that integral we indeed obtain the original function \(f(x)\) back (the function we started with) just as we ought to.

**Latex support**  An extra benefit included within this symbolic package is that it includes commands to provide us the syntax for latex as well! So we do not have to ever again manually write down the latex command for some complicated math expressions. For instance:

In [19]: print(latex(pf(x)))

\[
\frac{2 \sqrt{3}}{3} \text{atan} \left( \frac{2 x}{3} \sqrt{3} + \frac{\sqrt{3}}{3} \right)
\]

**Definite Integrals**  Let us now get back into the original example at the beginning of this session where we numerically computed the definite integral of \(f(x)\) for \(x\) from 0 to 4.

How would we do this with symbolic computations? Can we do this with sybmolic computations?

Remember that we already defined the function \(pf(x)\) to be equal to the integral of \(f(x)\). Let us look at that integral first,

In [20]: pf(x)

Out[20]:
\[
\frac{2 \sqrt{3}}{3} \text{atan} \left( \frac{2 x}{3} \sqrt{3} + \frac{\sqrt{3}}{3} \right)
\]

The way to compute the definite integral of \(f(x)\) from 0 to 4 is to use the fundamental theorem of calculus in order to obtain the definite integral from it.

In [21]: simplify(pf(4)-pf(0))

Out[21]:
\[
\frac{\sqrt{3}}{9} ( -\pi + 6 \text{atan} \left( 3 \sqrt{3} \right) )
\]

Furthermore, if we would like a number as a result we can evaluate the expression above using the command `evalf()`

In [22]: (pf(4)-pf(0)).evalf()

Out[22]:
0.9896614396123

If you check with the numerical value we obtain at the top of this notebook the results agree.
0.0.1 More details about symbolic objects and their definitions

In [23]: x,y,z,r,t = symbols('x y z rotation translation')

In [24]: r

Out[24]:

\textit{rotation}

In [25]: t**2

Out[25]:

\textit{translation}^2

If we look closely at the command below

In [26]: x,y,z,r,t = symbols('x y z rotation translation')

What is really happening internally in python is equivalent to the following two commands: first we load the symbols in a temporary list as seen below:

\texttt{templist=[symbols(n) for n in \textquoteleft x y z rotation translation\textquoteright .split()]} and then we unpack the list into respective variable names as follows:

\texttt{x, y, z, r, t = *templist}

In [27]: # a symbolically defined function
   
   f = symbols('f', Function=True)
   
   f

Out[27]:

\textit{f}

In [28]: F = integrate(f(x),x)
   
   F

Out[28]:

\int f(x) \, dx

0.0.2 Specifying more information about the symbols

In [29]: r = symbols('rotation',integer=True) # variable name and string representation do not match
   
   r**2

Out[29]:

\textit{rotation}^2

We can also specify a lot more information for each of our symbols in order to enforce the proper mathematical operations to take place.
In [30]: z = symbols('z', complex=True)
    
    r = symbols('rotation', negative=True)

In [31]: r.assumptions0 # to see what symbol r is really made from

Out[31]: {'negative': True,
             'nonzero': True,
             'imaginary': False,
             'zero': False,
             'nonpositive': True,
             'positive': False,
             'composite': False,
             'prime': False,
             'commutative': True,
             'nonnegative': False,
             'real': True,
             'complex': True,
             'hermitian': True}

0.0.3 Compact ways to define symbols

In [32]: vars = symbols('i:k')

In [33]: vars

Out[33]:

(i, j, k)

In [34]: A = symbols('A1:3(0:4)') # to define symbols with subindexes

Again, note that the above would take for ever to write in latex format.
Instead we can simply type the command below to do the job for us!

In [35]: print(latex(A))

\left ( A_{10}, \quad A_{11}, \quad A_{12}, \quad A_{13}, \quad A_{20}, \quad A_{21}, \quad A_{22}, \quad A_{23}\right )

0.0.4 Symbolic computations in several variables

A function which is not explicitly defined can be defined as follows.

In [36]: f, g = symbols('f g', cls=Function)

Such non explicitly defined functions can then be used to obtain the general rules of calculus
in terms of how to differentiate products (the product rule) or integrate using substitution or by parts methods.

In [37]: diff(f(g(x)),x) # remember the chain rule?
We can 'evaluate' our functions using our previously defined symbols thus defining functions of several variables.

$$\frac{d}{dx} g(x) \frac{d}{d\zeta_1} f(\zeta_1) \bigg|_{\zeta_1 = g(x)}$$

Now we can also obtain the gradient for this function as follows

$$\left[ \frac{\partial}{\partial x_0} f(x_0, x_1, x_2), \frac{\partial}{\partial x_1} f(x_0, x_1, x_2), \frac{\partial}{\partial x_2} f(x_0, x_1, x_2) \right]$$

or alternatively,

$$\left[ \frac{\partial}{\partial x_0} f(x_0, x_1, x_2), \frac{\partial}{\partial x_1} f(x_0, x_1, x_2), \frac{\partial}{\partial x_2} f(x_0, x_1, x_2) \right]$$

0.0.5 Taylor series expansions

Let us consider the cosine function,

$$f(x) = \cos(x)$$

and expand it as a MacLaurin (i.e. expand around x=0) series. Let's look at 10 terms of this series,
Out[46]:
\[
1 - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720} + O(x^8)
\]

If we would like we can even look at every single term of this series expansion by using list
comprehension. We can do this with the method .taylor_term.

**In [48]:** [f.taylor_term(n,x) for n in range(10)]

**Out[48]:**
\[
[1, 0, -\frac{x^2}{2}, 0, \frac{x^4}{24}, 0, -\frac{x^6}{720}, 0, \frac{x^8}{40320}, 0]
\]

**Taylor expansion of an abstract function f(x)**

**In [49]:** x = symbols('x')
    f = Function('f')
    dx = symbols('delta_x')
    f(x).series(x,0,n=4)

**Out[49]:**
\[
f(0) + x \frac{d}{dx} f(x) \bigg|_{x=0} + \frac{x^2}{2} \frac{d^2}{dx^2} f(x) \bigg|_{x=0} + \frac{x^3}{6} \frac{d^3}{dx^3} f(x) \bigg|_{x=0} + O(x^4)
\]

**Defining equations**

Defining a function for air resistance depending on speed v

**In [96]:** C, rho, A, v = symbols('C rho A v')
    # C drag coefficient,
    # A cross-sectional area,
    # rho density
    # v speed
    f_drag = Lambda(v, -Rational(1,2)*C*rho*A*v**2)
    f_drag

**Out[96]:**
\[
\left(v \mapsto -\frac{AC \rho v^2}{2}\right)
\]

As usual to evaluate this function we can simply plug in a value for the variable v. For instance
the value of the function when the velocity is 2 is given by,

**In [98]:** f_drag(5)
if $v$ is to depend on another variable $x$ then we can simply include that dependence as well! In the example below we let $v = x/3$ which now produces a function for velocity which changes with $x$!

```python
In [99]: x = symbols('x')
   :   v = x/3

    : f drag(v).subs({x:5}) # now evaluating the drag function at x = 5 (i.e. v = 5/3)

Out[99]:

\[-\frac{25A}{2}C\rho\]

As we have seen before we can create functions in several variables. Using Lambda we can do this with the following notation.

```python
In [107]: x, y = symbols('x y')
   : g = Lambda((x, y), sin(x) + cos(2*y))
   : g

Out[107]:

$((x, y) \mapsto \sin(x) + \cos(2y))$

The evaluation in this case would require 2 arguments $x$ and $y$ to be provided,

```python
In [106]: g(pi, pi/2)

Out[106]:

$-1$

Alternatively, a similar evaluation can be performed if we first define an input variable $p$ with the entries,

```python
In [121]: p = (pi, pi/2)
   : g(*p)

Out[121]:

$-1$


1 Basic 2D and 3D plotting for symbolic functions

Let us plot the function \( g \) defined above as a function of \( x \) only. In other words we will fix \( y \) to a specific value.

In [117]: # necessary if you would like to include plot in this notebook
    %matplotlib inline

    x, y = symbols('x y')
    h = sin(x)+cos(2*y)
    sety = {y:3} # freezing y to be always 3
    plot(h.subs(sety), (x, 0, 10))

Out[117]: <sympy.plotting.plot.Plot at 0x1cd7bd5a400>

In [120]: from sympy.plotting import plot3d # needed for 3d plotting
    plot3d(h, (x,0,6), (y,0,2))
1.1 Linear Algebra Computations

Defining a vector function in several variables

In [122]: F=Lambda((x,y),Matrix([sin(x) + cos(2*y), sin(x)*cos(y)]))
F

Out[122]

\[
\begin{pmatrix}
x & y \\
\end{pmatrix} 
\mapsto 
\begin{pmatrix}
sin(x) + cos(2 y) \\
sin(x) \cos(y)
\end{pmatrix}
\]

The analytic Jacobian is available via,

In [123]: F(x,y).jacobian((x,y))

Out[123]

\[
\begin{bmatrix}
\cos(x) & -2 \sin(2y) \\
\cos(x) \cos(y) & -\sin(x) \sin(y)
\end{bmatrix}
\]

alternatively we can also produce this very same result as follows,

In [124]: x=symbols('x:2')
F=Lambda(x,Matrix([sin(x[0]) + cos(2*x[1]),sin(x[0])*cos(x[1])]))
F

Out[124]:
\[
\left( x_0, x_1 \right) \mapsto \begin{bmatrix}
\sin(x_0) + \cos(2x_1) \\
\sin(x_0) \cos(x_1)
\end{bmatrix}
\]

In [125]: F(*x)

Out[125]:
\[
\begin{bmatrix}
\sin(x_0) + \cos(2x_1) \\
\sin(x_0) \cos(x_1)
\end{bmatrix}
\]

In [126]: F(*x).jacobian(x)

Out[126]:
\[
\begin{bmatrix}
\cos(x_0) & -2 \sin(2x_1) \\
\cos(x_0) \cos(x_1) & -\sin(x_0) \sin(x_1)
\end{bmatrix}
\]

1.1.1 Rotation matrix

In [127]: phi=symbols(\'phi\')
rotation=Matrix([[\cos(phi), -\sin(phi)],[\sin(phi), \cos(phi)]])
rotation

Out[127]:
\[
\begin{bmatrix}
\cos(\phi) & -\sin(\phi) \\
\sin(\phi) & \cos(\phi)
\end{bmatrix}
\]

In [128]: simplify(rotation.transpose()*rotation)

Out[128]:
\[
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}
\]

In [129]: simplify(rotation.transpose()*rotation - eye(2))

Out[129]:
\[
\begin{bmatrix}
0 & 0 \\
0 & 0
\end{bmatrix}
\]

In [130]: simplify(rotation.T - rotation.inv())

Out[130]:
\[
\begin{bmatrix}
0 & 0 \\
0 & 0
\end{bmatrix}
\]
1.1.2 Solving and manipulating $Ax = b$

In [131]: $M = \text{Matrix}(3,3, \text{symbols}(\text{\'M:3(3)\'})$)

\[
M
\]

Out[131]:

\[
\begin{bmatrix}
M_{00} & M_{01} & M_{02} \\
M_{10} & M_{11} & M_{12} \\
M_{20} & M_{21} & M_{22}
\end{bmatrix}
\]

Changing a specific element of the matrix by substituting a value in its place

In [132]: $M._{2,1}=0$

Out[132]:

\[
\begin{bmatrix}
M_{00} & M_{01} & M_{02} \\
M_{10} & M_{11} & M_{12} \\
M_{20} & 0 & M_{22}
\end{bmatrix}
\]

In [133]: $M\_{[0,2]}=0$

# changes one element

M[[1,1]]=Matrix(1,3,[[1,2,3]])

# changes an entire row

Out[133]:

\[
\begin{bmatrix}
M_{00} & M_{01} & 0 \\
1 & 2 & 3 \\
M_{20} & M_{21} & M_{22}
\end{bmatrix}
\]

In [134]: $A = \text{Matrix}(3,3, \text{symbols}(\text{\'A1:4(1:4)\'})$

$b = \text{Matrix}(3,1, \text{symbols}(\text{\'b1:4\'})$

$x = A.LU\text{solve}(b)$

# The analytic solution of $Ax = b$ is found here

Out[134]:

\[
\begin{bmatrix}
\frac{1}{A_{11}} \left( - \frac{A_{12} A_{22}}{A_{11}} \right) \left( b_2 - \frac{A_{23} - \frac{A_{13} A_{21}}{A_{11}}}{A_{33} - \frac{A_{23} A_{32}}{A_{11}}} \left( b_3 - \frac{A_{32} - \frac{A_{12} A_{31}}{A_{11}}}{A_{33} - \frac{A_{23} A_{32}}{A_{11}}} - \frac{A_{31} b_1}{A_{11}} \right) \right) - \frac{A_{13} b_1 b_3}{A_{11}} \\
\frac{1}{A_{22} - \frac{A_{12} A_{21}}{A_{11}}} \left( b_2 - \frac{A_{23} - \frac{A_{13} A_{21}}{A_{11}}}{A_{33} - \frac{A_{23} A_{32}}{A_{11}}} \left( b_3 - \frac{A_{32} - \frac{A_{12} A_{31}}{A_{11}}}{A_{33} - \frac{A_{23} A_{32}}{A_{11}}} - \frac{A_{31} b_1}{A_{11}} \right) \right) - \frac{A_{23} b_1 b_3}{A_{11}} \\
\frac{1}{A_{33} - \frac{A_{23} A_{32}}{A_{11}}} \left( b_3 - \frac{A_{32} - \frac{A_{12} A_{31}}{A_{11}}}{A_{33} - \frac{A_{23} A_{32}}{A_{11}}} - \frac{A_{31} b_1}{A_{11}} \right) - \frac{A_{32} b_1 b_3}{A_{11}}
\end{bmatrix}
\]

12
It is a rather complicated solution but it can hopefully be simplified using the simplify command.

```
In [135]: simplify(x)
```

```
Out[135]:
```

Regardless, now we can provide values for all the variables in order to evaluate this solution.
Let us check that indeed the above general solution for x is correct! Let us multiply A with x and check if it is equal to b...

```
In [136]: A*x-b
```

```
Out[136]:
```

If all went well the above should be equal to the vector [0,0,0]. This is clearly not what it looks like...
Perhaps if we simplify this result?

```
In [137]: simplify(A*x-b)
```

```
Out[137]:
```

There are several other commands available for us to use.
For instance here is how to compute the determinant of a matrix,

```
In [138]: A.det()
```

```
Out[138]:
```

\[
\begin{align*}
A_{11}A_{22}A_{33} - A_{11}A_{23}A_{32} - A_{12}A_{21}A_{33} + A_{12}A_{23}A_{31} + A_{13}A_{21}A_{32} - A_{13}A_{22}A_{31}
\end{align*}
\]