Lecture 20 -- Jupyter and Scipy Integrals

1. Prayer/Quiz/AMA
2. Jupyter Notebook
3. Scipy Quad and Simps
4. Symbolic Calculus in Python

I. Jupyter Notebook

- Jupyter notebook is another way of interacting with Python. It is an alternative to Spyder. It is often a convenient way to do your homework, and several professors in the department use it, so I wanted to introduce you to it.
- Just like Spyder, you open Jupyter using Anaconda. Jupyter will open in a broswer window.
- Unlike *.py files, you need to open a *.ipynb file in Jupyter, not a text editor. If you open *.ipynb files with a text editor, you will see that it isn't a simple python file.

A. Cells

A notebook is divided into *cells* of different types.

- **Code** cells for executable Python code.
- **Markdown** cells for formatting text and writing descriptions.
- **Raw** cells for plain text.

You can change the cell type using the drop down menu above.

- You can insert and delete cells above or below a selected cell using the "Insert" and "Edit" menus.
- Executing a cell:
  - Click the "play" button.
  - Press **`shift-enter`** or **`ctrl-enter`**
- You can execute a cell and then insert a cell below by pressing **`alt-enter`**.
- Code in cells below are "aware" of the results as if the whole document were one big Python script.
  - Note that a cell has to be executed for cells below to be "aware" of the results.
  - Sometimes you may have to re-execute things if you change cells above.
  - If in doubt, just go to "Cell" --> "Run All"
- You can restart with the "kernel" menu

Deleting Variables

- Because Jupyter "knows" about all of the variables, it is sometimes useful to delete variables.
- `%reset` deletes *all* of the variables
- `del x` deletes the single variable `x`.

B. Code cells

Code cells are where we write python code.

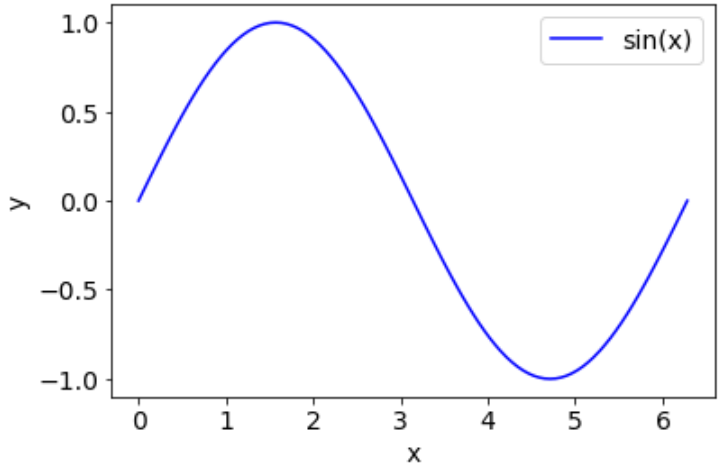
The code is the same syntax whether we are using the Jupyter Notebook, or running Python in a Python interpreter like Spyder.

Example:

In [10]:
```
# This is a code cell
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2*np.pi, 101)
y = np.sin(x)

# Plots are nice, because they show up in the cell
plt.rc('font', size=14)
plt.plot(x, y, 'b', label='sin(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```



C. Markdown Cells

Markdown is a markup language (think html) used for formatting text cells.

- Wikipedia page (https://en.wikipedia.org/wiki/Markdown)
- Official website (http://daringfireball.net/projects/markdown/)
- Markdown Cheat Sheet (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet)

Double click on the markdown cells in this document (including this cell) for examples.

Examples:

1. Text Formatting and Links

Italic Text

Bold Text

You can also put html code, images, pdfs, etc. directly into markdown, e.g. Colored Text.

Make hyperlinks like this `[Link text](http://somesite.com)`
For example: Google it (http://www.google.com)

2. Headings and Paragraphs

Headings are preceeded by the pound sign #.

Heading 1

Heading 2

Heading 3

Etc.

New paragraphs are separated by empty lines.

The quick brown fox jumped over the lazy dog.

New lines are initiated by giving two trailing spaces.
The quick brown fox jumped over the lazy dog.

3. Lists

Make a bulleted list with asterisks and tabs/spaces like this:

- Item one (note the space after the *)
- Item two (note the symbol becomes a circle)
    - Sub-bullet 1 (note the sub-bullet is a square)
    - Sub-bullet 2
        - Sub-bullet 3 (the sub-sub-bullet is an open circle)

Use numbers to make a numbered list:

1. Item 1
2. Item 2
    A. Sub-item (the "1." in the code becomes "A." in output)
    B. Sub-item
        a. Sub-Sub-item (The "1." becomes "a." in output)

4. Equations and Math Formatting

- In-line equations are surrounded with a dollar sign like this: `$a=b$`.

Example: `$a^2 + b^2 = c^2$` becomes: $a^2 + b^2 = c^2$.

- To put an equation on its own line use double dollar signs: `$$a=b$$`.

Example: `$$\frac{x+1}{y} + \frac{x^2}{y} = \frac{x^2 + x + 1}{y}$$` becomes:

$$\frac{x+1}{y} + \frac{x^2}{y} = \frac{x^2 + x + 1}{y}$$

Fraction: `$\frac{num}{den}$` becomes $\frac{num}{den}$

Superscript: `$a^b$, or $a^{multiple characters}$` becomes a^b, or $a^{multiplecharacters}$

Subscript: `$a_b$, or $a_{multiple characters}$` becomes a_b, or $a_{multiplecharacters}$

Space: `\,` gives a space. `$a^{multiple\,characters}$` becomes: $a^{multiple\,characters}$

Partial derivative: `$\frac{\partial y}{\partial x}$` becomes: $\frac{\partial y}{\partial x}$

Special functions: use \sin, \exp, etc.: $x^2 \sin(x+3)$, $y = c_0 \exp(-t_{i,j})$

Greek symbols are spelled out like `\alpha`, `\beta`, etc: `$$\alpha \, \beta \, \gamma \, \Delta$$` becomes:

$$\alpha \, \beta \, \gamma \, \Delta$$

Big parentheses with `\left(` and `\right)`:

`$$\left(\frac{a}{b}\right) = \left[\frac{x+y}{2}\right]^{\kappa/RT}$$` becomes:

$$\left(\frac{a}{b}\right) = \left[\frac{x+y}{2}\right]^{\kappa/RT}$$

Equations are entered using a typesetting language called LaTeX. More info about LaTeX can be found here:

- LaTeX Tutorial (https://www.sharelatex.com/learn/Learn_LaTeX_in_30_minutes#Adding_math_to_LaTeX)
- LaTeX Cheat Sheet (https://users.dickinson.edu/~richesod/latex/latexcheatsheet.pdf)

II. Scipy Quad and Simps

A. Scipy Quadrature

- Just like we have Scipy functions for solving non-linear equations, there are scipy functions for doing numerical integrations.
- There are two relevant functions for us:

    1. Scipy.integrate.quad
    2. Scipy.integrate.simps

Quad

- More accurate/fastest
- Can only use if you know the function
- Uses an advanced method that we didn't learn called Gaussian quadrature

Simps

- Can use if you only have data points, but don't know the equation
- Uses composite Simpson's rule
- Best if data is at equally spaced points

More info (https://docs.scipy.org/doc/scipy/reference/integrate.html) about the scipy integrate module.

B. Scipy.integrate.quad

- The "best" general quadrature function in scipy is the scipy.integrate.quad function. To use it, you must know the function (not just the data points).
- Documentation for quad (https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.quad.html)

Steps to numerically integrate a function with quad:

```
1. Import quad from the scipy.integrate module
2. Define the function
3. Evaluate the integral using quad:
```

```
from scipy.integrate import quad
integral, error = quad(fn_name, lower_bound, upper_bound)

# In:
# fn_name: name of the function defined in (2)
# lower bound: the lower bound of the integral
# upper bound: upper bound of the integral

# Out:
# integral: value of the integral
# error: estimate of the error of the integral
```

Example:

Evaluate $I = \int_0^1 3x^2 + 1\, dx$

(The exact answer is $I = 2$.)

In [15]:
```
# Integrate f(x) = 3 x^2 + 1 from x = 0 to x = 1
#
# (The exact value is: x^3 + x |_0^1 == 2)

from scipy.integrate import quad

def f(x) :
    return 3.0*x*x + 1.0

xlo = 0
xhi = 1

I_quad, err = quad(f, xlo, xhi)
I_quad, _ = quad(f, xlo, xhi)
I_quad = quad(f, xlo, xhi)[0]

print("I_quad     = ", I_quad)
print("error = ", err)
```

```
I_quad     =  2.0
error =  2.220446049250313e-14
```

C. Scipy.integrate.simps

- If you don't have a function, but a set of {x,y} data points, your next best option is to use Simpson's rule. You already know how to code this, but Scipy has done it for you!
- Documentation for simps (https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.simps.html#scipy.integrate.simps)

Steps to numerically integrate a function with simps:

1. Import simps from the scipy.integrate module
2. Read in or define your x and y data
3. Evaluate the integral using simps

```
from scipy.integrate import simps
integral = simps(y_data, x_data)
# In
# y_data: y data (usually given)
# x_data: x data (usually given)
# Out
# integral: value of the integral
```

Example

Integrate the curve described by the set of ten {x,y} data points given. The value of the integral should be close to $\frac{2}{3\pi}$. (This is the same problem you had last time for the trapezoidal rule.)

In [16]:
```
from scipy.integrate import simps

x_data = np.array([0.000000e+00, 1.111111e-01, 2.222222e-01, 3.333333e-01, \
                   4.444444e-01, 5.555556e-01, 6.666667e-01, 7.777778e-01, \
                   8.888889e-01, 1.000000e+00])

y_data = np.array([0.000000e+00, 3.020114e-01, 3.772033e-01, 2.165064e-01, \
                   2.969559e-02, 2.969559e-02, 2.165064e-01, 3.772033e-01, \
                   3.020114e-01, 1.224647e-16])

I_simps = simps(y_data, x_data)
print("\nI_simps = ", I_simps)
print("error = ", np.abs(I_simps - 2/(3*np.pi)))

# The error is quite small, approx 9.6e-4. This is a bit more than an order
# of magnitude better than the trapezoidal rule which had an error of
# 6.5e-3 when N=10.
```

```
I_simps =  0.211240935769
error =  0.00096565502021
```

III. Symbolic Math

A. Sympy Module

Sympy is a Python module for symbolic math (i.e. computer algebra system) similar to Maple or Mathematica.

Here are some useful links:

- Sympy Tutorial (http://docs.sympy.org/dev/tutorial/index.html)
- Sympy Documentation (http://docs.sympy.org/latest/index.html)

One import Sympy as follows:

In [18]:
```
import sympy as sp
sp.init_printing() # To get pretty output

# Use display() instead of print() for nice output
from IPython.display import display
```

B. Symbolic Variables

Sympy works by defining symbolic variables. Those variables are then used to create expressions which can be used to do algebra, linear algebra or calculus.

In [21]:
```
# Symbol variables are defined
x, y, z = sp.symbols('x, y, z')

# The name of the variable doesn't have
# to be the same as the symbol
Ell = sp.symbols('L')

display(x)
display(Ell)

# Expressions are defined just like regular
# python, except we use sympy math functions.
ex1 = x*sp.cos(x)
ex2 = x**2 + y**2 + z
ex3 = sp.sqrt(x+y/2)

display(ex1)
display(ex2)
display(ex3)
```

x

L

$x \cos(x)$

$x^2 + y^2 + z$

$\sqrt{x + \frac{y}{2}}$

C. Algebra

Symbolic Math is really useful for doing algebra.

Examples:

1. Simplify

In [24]:
```
ex = x**3 / x**2
sp.simplify(ex)
```

Out[24]: x

2. Expand

In [11]:
```
ex = (x+2)*(x-3)
sp.expand(ex)
```

Out[11]: $x^2 - x - 6$

3. Factor

In [12]:
```
ex = x**3 - x**2 + x - 1
sp.factor(ex)
```

Out[12]: $(x-1)\left(x^2+1\right)$

4. Substitute a number into an expression

In [13]:
```
ex = x**3 - x**2 + x - 1

ans1 = ex.subs(x, 2.)   # evaluates numerically
ans2 = ex.subs(x, 2)    # treats as a symbol

display(ans1)
display(ans2)
```

5.0

5

5. Substite a symbol into an expression

In [14]:
```
ans3 = ex.subs(x, y)
ans4 = ex.subs(x, y**2+z)

display(ans3)
display(ans4)
```

$y^3 - y^2 + y - 1$

$y^2 + z + \left(y^2+z\right)^3 - \left(y^2+z\right)^2 - 1$

6. Other Algebra Functions

- There are many more things that sympy can do! It is very powerful.
- More algebraic manipulation functions: `collect`, `cancel`, `apart`, `trigsimp`, `expand_trig`, `powsimp`, `expand_power_exp`, `expand_power_base`, `pow_denest`, `expand_log`, `logcombine`
- Matrix math functions: `Matrix`, `shape`, `row`, `col`, `col_del`, `row_del`, `col_insert`, `row_insert`, `T`, `zeros(m,n)`, `ones(m,n)`, `diag`, `nullspace`, `eigenvals`, `diagonalize`, `rref`, `inv()`
- The `sp.solve()` function can be used to find symbolic solutions of both linear or non-linear equations.

D. Calculus

Finally, Sympy can also do symbolic derivatives or integrals.

- Derivatives are done using `sp.diff( expr, var, [order])`.
- Indefinite Integrals are done using `sp.integrate(expr, var)`.
- Definite Integrals are done using `sp.integrate(expr, (var, lb, ub))`, where `lb` is the lower bound and `ub` is the upper bound.

Examples:

1. Derivatives

In [26]:
```
ex = sp.exp(x**4)

sp.diff(ex, x) # single derivative
```

Out[26]: $4x^3 e^{x^4}$

In [17]:
```
sp.diff(ex, x, 3) # third derivative
```

Out[17]: $8x\left(8x^8 + 18x^4 + 3\right)e^{x^4}$

In [18]:
```
ex = sp.exp(x*y*z)
dxdy = sp.diff(ex, x, y) # mixed partial derivatives

display(dxdy)
```

$z\left(xyz + 1\right)e^{xyz}$

2. Integrals

In [19]:
```
sp.integrate(x**2, x)  # indefinite integral
```

Out[19]: $\frac{x^3}{3}$

In [20]:
```
sp.integrate(x**2, (x, 0, 1))  # definite integral
```

Out[20]: $\frac{1}{3}$

In [27]:
```
sp.integrate(sp.exp(-x), (x,0,sp.oo)) # infinity is "oo": double o's
```

Out[27]: 1

In [22]:
```
sp.integrate(x*sp.exp(-x), (x,y,sp.oo)) # y is in the limit
```

Out[22]: $-\left(-y - 1\right)e^{-y}$

3. Other Calculus Functions

- Limits: **`limit()`**
- Solve Differential Equations: **`dsolve()`**
- Taylor Series: **`series()`**
- Finite Differences: **`differentiate_finite()`**