Computational Physics
Lecture 3

sieversj@ukzn.ac.za

git clone https://github.com/ukzncompphys/lecture3.git

Tutorial Problems - Questions?

- How can you list all files in a directory, with the most recent one displayed last? (5)
- How can you display the first 25 lines of a file? How can you display just lines 16 through 25? (5)
- How can you search for all commands relating to python? Hint - it will be a use of "man". (5)
- Make a new text file with your name, email address, what you're working on for your honours project, plus a few things you'd like to learn in the course. Initialize a git repository and commit this file. (5)
- Put your answers to the other tutorial questions into another text file. Add this to the repository also. (5)
- Make a github account and push the repository onto github. Email me your github name so I can have a look at your file/answers. (10)

Python

- Now that we're all comforable in Unix (right?), let's learn basic python
- Python is *interpreted* - not compiled.
- Blocks are set off by indents - no END command.
- Python is *object oriented* - things know about themselves
- Python is highly extensible - just import stuff!
- Namespaces are distinct, functions remember where they came from - unless you specify otherwise.

Hello World

```
Jonathans-MacBook-Pro:lecture3 sievers$ python
Python 2.7.5 (default, Sep 12 2013, 21:33:34)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'hello world'
hello world
>>> print "hello world"
hello world
>>> ^D
Jonathans-MacBook-Pro:lecture3 sievers$
```

- Strings in python can be specified with either single or double quotes
- to show a string, just type print it!
- To start python, type "python" from a prompt
- to quit, hit ctrl-d

```
Jonathans-MacBook-Pro:lecture3 sievers$ cat hello_world.py
print "hello world"
print 'hello world'
Jonathans-MacBook-Pro:lecture3 sievers$ python hello_world.py
hello world
hello world
Jonathans-MacBook-Pro:lecture3 sievers$
```

You can also execute scripts

Variables

```
Jonathans-MacBook-Pro:lecture3 sievers$ cat simple_variables.py
x=5
y=3.0
z=x+y
print 'z is ' + repr(z)
print 'z is a ' + repr(type(z))
print 'x is a ' + repr(type(x))
print 'y is a ' + repr(type(y))

Jonathans-MacBook-Pro:lecture3 sievers$ python simple_variables.py
z is 8.0
z is a <type 'float'>
x is a <type 'int'>
y is a <type 'float'>
Jonathans-MacBook-Pro:lecture3 sievers$
```

- Python has several built-in variable types. These include strings, floats, ints, boolean.
- You can create variables by assigning to them.
- You can see what type variables are with the *type* command.
- You can print their values with the *print* command. *repr* will return value as a string
- strings can be combined with '+'. Types may be automatically converted.

Lists

```
Jonathans-MacBook-Pro:lecture3 sievers$ cat list_example.py
mylist=['a','bc',4.0,5,['d',9]]
print mylist[0]
print mylist[2]
print mylist[-1]
Jonathans-MacBook-Pro:lecture3 sievers$ python list_example.py
a
4.0
['d', 9]
Jonathans-MacBook-Pro:lecture3 sievers$
```

- Many variables can be put together into a list
- Each member of a list is a variable (possibly another list!)
- Lists are indexed with []
- Index starts from 0
- Negative indices go from end of list

Tuples

```
Jonathans-MacBook-Pro:lecture3 sievers$ cat tuple_example.py
mylist=('a','bc',4.0,5,['d',9])
print mylist[0]
print mylist[2]
print mylist[-1]
mylist[2]=5.0
Jonathans-MacBook-Pro:lecture3 sievers$ python tuple_example.py
a
4.0
['d', 9]
Traceback (most recent call last):
  File "tuple_example.py", line 5, in <module>
    mylist[2]=5.0
TypeError: 'tuple' object does not support item assignment
Jonathans-MacBook-Pro:lecture3 sievers$
```

- Tuples are like lists, except you can't change them.
- Tuples are defined with ().
- Normally used to return values from functions.
- Note error message telling you what broke!

For Loop

```
Jonathans-MacBook-Pro:lecture3 sievers$ cat for_example.py
mylist=['a','bc',4.0,5,['d',9]]
# this is a comment!
#Note the colon to start the for loop
for x in mylist:
    print x
#when we stop indenting, we're done
print "That's all folks"

Jonathans-MacBook-Pro:lecture3 sievers$ python
Python 2.7.5 (default, Sep 12 2013, 21:33:34)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> execfile('for_example.py')
a
bc
4.0
5
['d', 9]
That's all folks
>>>
```

- *For* loops iterate over items. list, tuples, (some) others valid
- Indenting in python is meaningful, blocks of code *defined* by indents.
- *execfile* will run a script from within the interpreter

```
>>> for x in range(0,5):
...   print x
...
0
1
2
3
4
>>>
```

range command will make a list of a sequence of numbers. Use to make traditional *for* loop.

Quick Example

- How would I print the first 10 squares?
- note - x*x or x**2 are both the square of x

```
>>> for x in range(0,10):
...   print x**2
...
0
1
4
9
16
25
36
49
64
81
>>>
```

Let's do sin(x) instead

```
Jonathans-MacBook-Pro:lecture3 sievers$ cat math_example.py
import math
from math import sin

#you *can* do this.  But don't!
from math import sin as cos
for x in range(0,10):
    print [x,math.sin(x),sin(x),cos(x)]

Jonathans-MacBook-Pro:lecture3 sievers$ python math_example.py
[0, 0.0, 0.0, 0.0]
[1, 0.8414709848078965, 0.8414709848078965, 0.8414709848078965]
[2, 0.9092974268256817, 0.9092974268256817, 0.9092974268256817]
[3, 0.1411200080598672, 0.1411200080598672, 0.1411200080598672]
[4, -0.7568024953079282, -0.7568024953079282, -0.7568024953079282]
[5, -0.9589242746631385, -0.9589242746631385, -0.9589242746631385]
[6, -0.27941549819892586, -0.27941549819892586, -0.27941549819892586]
[7, 0.6569865987187891, 0.6569865987187891, 0.6569865987187891]
[8, 0.9893582466233818, 0.9893582466233818, 0.9893582466233818]
[9, 0.4121184852417566, 0.4121184852417566, 0.4121184852417566]
Jonathans-MacBook-Pro:lecture3 sievers$
```

- *sin* is not a build-in python function
- Instead, we'll have to *import* it from the math library
- Python gives you great power over how much to import, and what to call it.

More on import

- There is a *huge* variety of stuff available in python
- See, e.g., https://docs.python.org/2/library/ for list of standard library. Many, many more things available online.
- Some commonly used things: os, sys, pickle, datetime, ctypes, threading, readline, urllib…
- *help* will (usually) print more info, like *man* in Unix. e.g. help(math.sin)
- You can use *pip* to install more python packages. If you don't have sudo powers, *pip install —user <package>* should work.
- readline/rlcompleter can be used to get tab-completion working in python.

Numpy

- Python has no built in arrays. Use NUMerical PYthon (numpy) for math functionality

- Many functions work on numpy arrays - usually much (much) faster than going through interpreter

- *numpy.arange* works like *range*, but returns numpy array instead of a list.

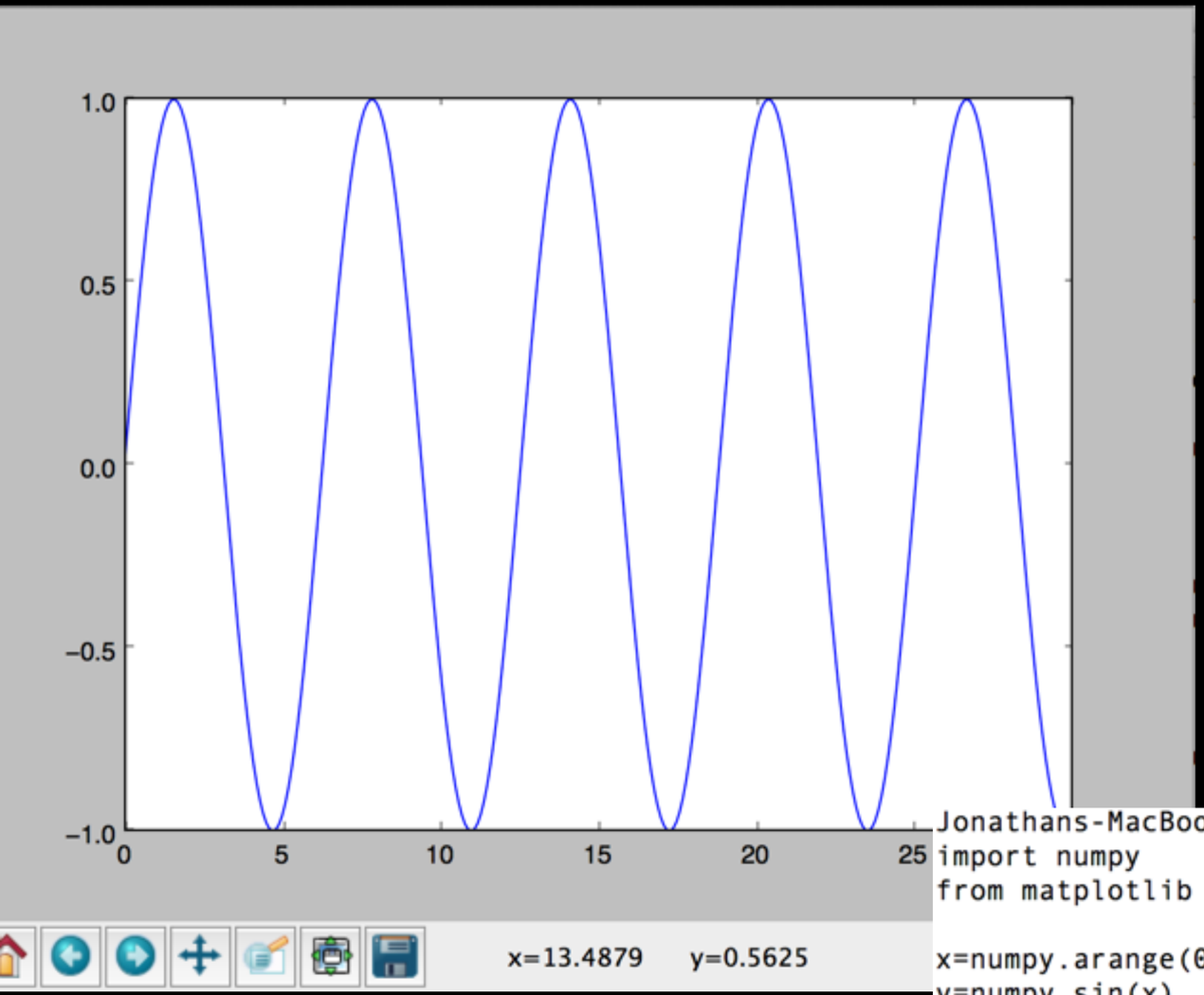
note - numpy.arange with three arguments will space values by 3rd argument.

```
Jonathans-MacBook-Pro:lecture3 sievers$ cat numpy_example.py
import numpy
x=numpy.arange(0,10)
print x**2
print 'type(x) is ' + repr(type(x))

x=range(0,10)
print 'type(x) is now ' + repr(type(x))
print x**2

Jonathans-MacBook-Pro:lecture3 sievers$ python numpy_example.py
[ 0  1  4  9 16 25 36 49 64 81]
type(x) is <type 'numpy.ndarray'>
type(x) is now <type 'list'>
Traceback (most recent call last):
  File "numpy_example.py", line 8, in <module>
    print x**2
TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
Jonathans-MacBook-Pro:lecture3 sievers$
```

Plotting

- Let's make a plot! Say, sin(x) from 0 to 30
- How would we do this? To the tubes!

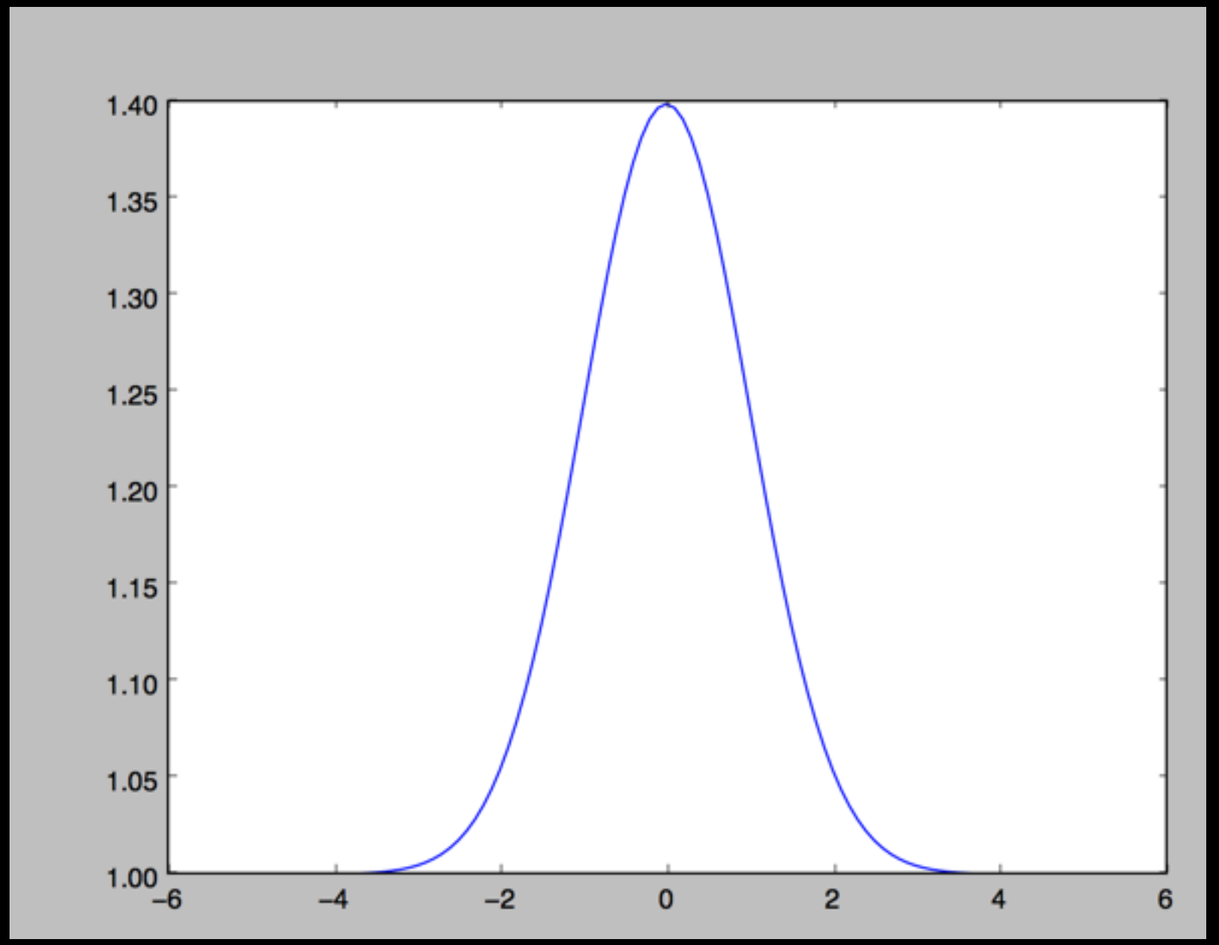
Example




```
Jonathans-MacBook-Pro:lecture3 sievers$ cat matplotlib_example.py
import numpy
from matplotlib import pyplot as plt

x=numpy.arange(0,30,0.1)
y=numpy.sin(x)

plt.plot(x,y)
plt.show()
Jonathans-MacBook-Pro:lecture3 sievers$ python matplotlib_example.py
```

Functions in Python

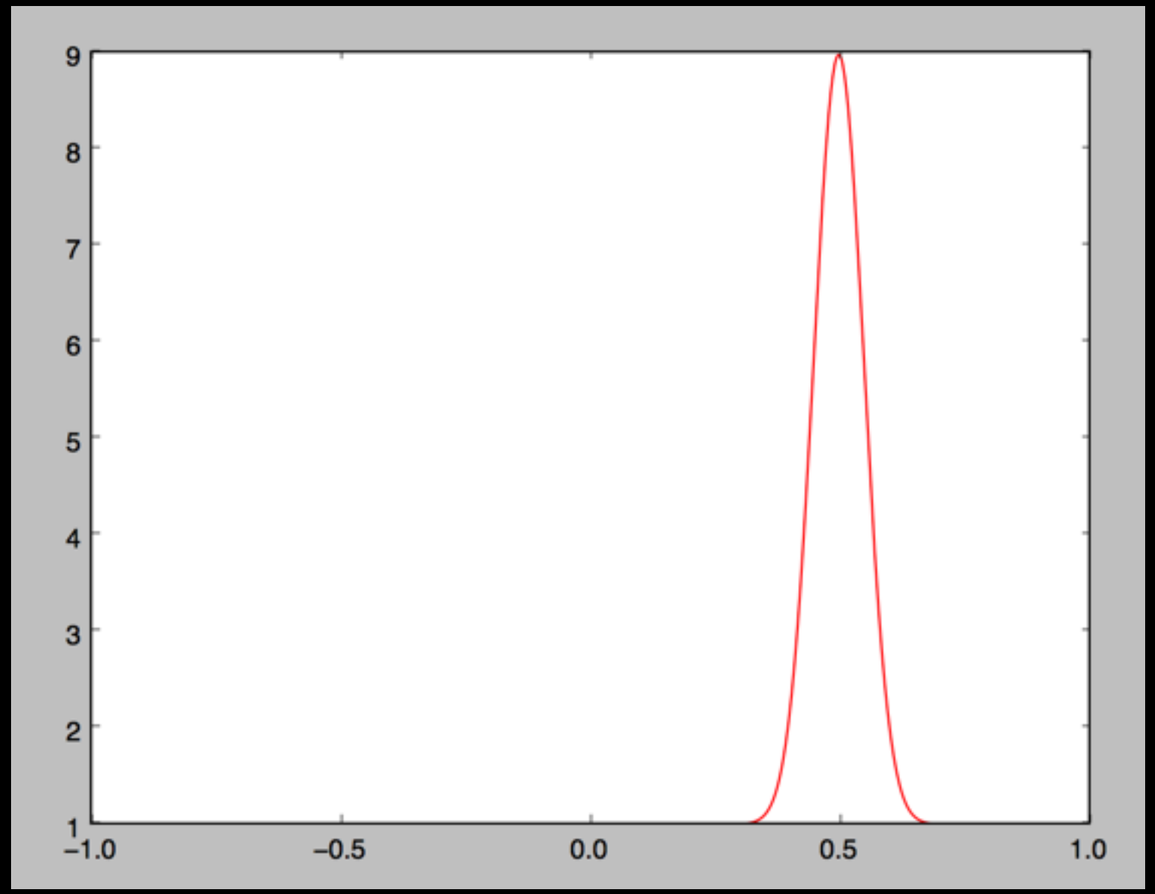
```
import numpy
from matplotlib import pyplot as plt
#functions defined with def
#arguments can be given default values
def mygauss(x,cent=0,sig=0.1):
    y=numpy.exp(-0.5*(x-cent)**2/sig**2)
    #pick this normalization so area under
    #gaussian is one.
    y=1+y/numpy.sqrt(2*numpy.pi*sig**2)
    return y

#only run this part if script is executed
if __name__ == "__main__":
    dx=0.1
    x=numpy.arange(-5,5,dx)
    y=mygauss(x,0,1)
    y2=mygauss(x,sig=1)
    print 'y total is ' +repr(y.sum()*dx)

    plt.plot(x,y)
    plt.show()
```



numpy arrays know how do to a lot of things. Can sum all elements with e.g. y.sum(). arrays also know their dimensions (shape), min,max, etc.

Functions Ctd.

```
import numpy
from matplotlib import pyplot as plt
#we can import functions we just wrote!
import func_example

x=numpy.arange(-1,1,0.002)
#functions are referenced by the file they're in
y=func_example.mygauss(x,cent=0.5,sig=0.05)
#we can assign the function to a variable
gg=func_example.mygauss
y2=gg(x,cent=0.5,sig=0.05)
delt=numpy.abs(y2-y)
print 'error is ' + repr(delt.sum())
#will output:
#error is 0.0
plt.plot(x,y,'r')
plt.show()
```



Integration

- We know have the tools to do some simple definite integrals
- Recall fundamental definition - sum ($f(x_i)*dx$) as dx-> 0
- How would we approximate integral of sin from 0 to pi?
- Before we do that, what *should* the answer be?

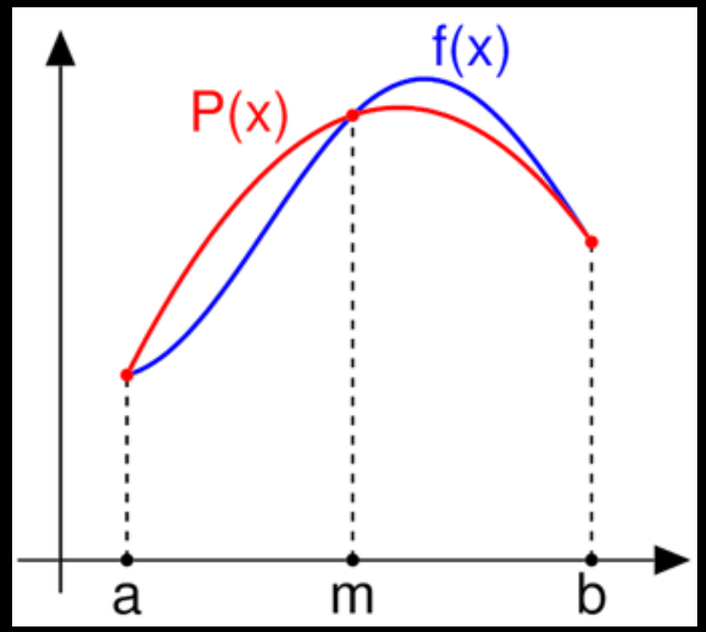
Example

- Here's some code that does the numerical integral of sin while varying the step size. How well does it work?

```
import numpy

x0=0
x1=numpy.pi

mydelts=[0.5,0.1,0.03,0.01,0.003,0.001]
for dx in mydelts:
    x=numpy.arange(x0,x1,dx)
    y=numpy.sin(x)
    tot=y.sum()*dx
    print 'integral is ' + repr(tot) + ' with dx=' + repr(dx)
```

```
Jonathans-MacBook-Pro:lecture3 sievers$ python sin_integral.py
integral is 1.9836405445028298 with dx=0.5
integral is 1.9995479597125 98 with dx=0.1
integral is 1.9999407675824561 with dx=0.03
integral is 1.9999900283082466 with dx=0.01
integral is 1.9999992133611066 with dx=0.003
integral is 1.9999999540409921 with dx=0.001
Jonathans-MacBook-Pro:lecture3 sievers$
```

Simpson's Rule



- Let's integrate a quadratic over three points.
- Draw a straight line between the left and right points. The middle point is now off the straight line by $y_{mid}-0.5*(y_{left}+y_{right})$
- What is the average value of $(1-x^2)$ between -1 and 1?
- Area is now $1/2*(y_{left}+y_{right})+2/3*(y_{mid}-0.5*(y_{left}+y_{right}))$
- simplify: area=$1/6*y_{left}+2/3*y_{mid}+1/6*y_{right}$.
- for a bunch of points, string together segments, y_{right} become y_{left} of the next segment.
- Simpson's rule: integral=$dx*(1/6\ y_0 + 2/3y_{odd} + 1/3y_{even} + 1/6\ y_{last})$

4th Order Runge-Kutte

- Sometimes we want to integrate ODE's, $dy/dx=f(x,y)$
- Tricker than simple integration, because we can't evaluate at arbitrary points since y is changing.
- One standard technique is 4th-order Runge-Kutta, analagous to Simpson's rule. Make estimates of function using left edge, 2 in center, and one at right.
- RK4 reduces to Simpson's rule if $dy/dx=f(x)$ only (not $f(x,y)$). For smooth functions, RK4 should be accurate to 4th order.

RK4 Recipe

- for step size h, let k_1=h*f(x,y) (left edge)
- k_2=h*f(x+h/2,y+k_1/2) (first mid-point estimate)
- k_3=h*f(x+h/2,y+k_2/2) (second mid-point estimate)
- k_4=h*f(x+h,y+k_3) (right edge)
- y(x+h)=y(x)+(k_1+2k_2+2k_3+k_4)/6

In Practice

```
import numpy
def myfun(x,y,a):
    #evaluate dydx=a*x*y
    return x*y*a
def rkstep(x,y,h,a,func):
    k1=h*func(x,y,a);
    k2=h*func(x+0.5*h,y+0.5*k1,a);
    k3=h*func(x+0.5*h,y+0.5*k2,a);
    k4=h*func(x+h,y+k3,a);
    dy=(k1+2*k2+2*k3+k4)/6
    return dy
y0=2.0
x0=4.0
x1=10
a=0.5
h=0.01
y=y0
for x in numpy.arange(x0,x1,h):
    #print "(x,y)="+repr(x)+' '+repr(y)
    y=y+rkstep(x,y,h,a,myfun)
print "at end (x,y)=" + repr(x+h) + ' ' + repr(y)
#can solve analytically:
#dy/y=axdx, log(y)=0.5*ax^2+c, y=c*exp(0.5*ax^2)
#at (x0,y0)=c=y0/exp(0.5*ax0**2)
c=y0/numpy.exp(0.5*a*x0**2)
print "predicted: " + repr(c*numpy.exp(0.5*a*(x+h)**2))
```

```
Jonathans-MacBook-Pro-3:lecture3_2015 sievers$ python rk4.py
at end (x,y)=9.999999999998721 2637630368.8623924
predicted: 2637631468.9647431
```

Tutorial

- Write a python script to make a vector of n evenly spaced numbers between 0 and pi/2. i.e. x[0]=0, x[-1]=pi/2 (5)

- Use this vector to integrate cos(x) from 0 to pi/2 for a range # of points using the simple method. include 10,30,100,300,1000 points between 0 and pi/2. How does error scale with # of points? (5)

- Python supports array slicing - x[5:10:2] will take points 5,7,9 from x. x[5::2] will take points 5,7,9… from x. How can I take all odd points from an array? How can I take all even points from an array, but skipping the first and last points? (5)

- Write a python function to integrate this vector using Simpson's rule. How does error scale with # of points? How many points did we need to use in part 2 to get same accuracy as 11 points with Simpson's rule? (10)

- Plot the errors as a function of # of points using Simpson's rule and standard sum. You will want to use a log scale here - look at logplot.py in the github distribution (5)

Bonus Points

- the scipy module has built in integration functions in scipy.integrate. The quad routine will do numerical integrals. quad will try to put its effort where the function changes quickly.

- Look at scipy_quad_example.py, which uses scipy to integrate our Gaussian function over two different ranges. The integrals should be (almost) identical - yet they are not. Can you figure out why? (5)

- Can you write another function that will always give the correct answer to this integral? (5) Hint - you may want to do two integrals instead of one.