Short introduction to python
(based on last year's lecture by Marc Wiedermann)

Jasper Franke

$ ipython
In [1]: print 'Hello World!'
Hello World
Why python?

- Similar syntax as matlab → easy to learn
- Interactive
- Easy to run in parallel
- Open source
- Expendable with numerous packages for different applications
- Intelligent coding makes it almost as fast as C
Important extensions

numpy
• Fast numerics and statistics

scipy
• Tailored to scientific applications (solving ODE, interpolation, integration,...)

matplotlib (pyplot)
• Plotting data

pyunicorn
• Advanced nonlinear statistical methods for time series analysis
• Some methods covered during the school are part of pyunicorn

basemap
• Plotting geoscientific data on maps
Running your code

Two options (just like in matlab)

1. Interactive console (ipython)
   • start by typing `ipython` in shell and get:

   ![ipython shell output]

   • Then just start coding!

2. Alternatively write a script, class or a whole package and run from shell with:

   `python nameofyourscript.py`
Importing and using packages

```python
import numpy
print numpy.arange(10)

>>> [0 1 2 3 4 5 6 7 8 9]
```
Importing and using packages

```python
import numpy
print numpy.arange(10)

>>> [0 1 2 3 4 5 6 7 8 9]

import numpy as np
from matplotlib import pyplot as plt
print np.arange(10)
x = np.linspace(0, 2*np.pi, 1000)
y = np.sin(x)
plt.plot(x, y)
plt.show()

>>> [0 1 2 3 4 5 6 7 8 9]
```
For loops

```python
for i in range(10):
    print i,

>>> 0 1 2 3 4 5 6 7 8 9
```
For loops

```python
for i in range(10):
    print i,

>>> 0 1 2 3 4 5 6 7 8 9

for name in ['marc', 'reik']:
    print name

>>> marc
>>> reik
```
For loops

```python
for i in range(10):
    print i,

>>> 0 1 2 3 4 5 6 7 8 9

for name in ['marc', 'reik']:
    print name

>>> marc
>>> reik

for i in range(3):
    for j in range(2):
        print i*j

>>> 0 0 0 1 0 2

import numpy as np
i = np.arange(3)
for j in range(2):
    print i*j,

>>> [0 0 0] [1 0 2]
```
For loops

```python
for i in range(10):
    print i,

>>> 0 1 2 3 4 5 6 7 8 9

for name in ['marc', 'reik']:
    print name

>>> marc
>>> reik
```

Slow code:

```python
for i in range(3):
    for j in range(2):
        print i*j

>>> 0 0 0 0 0 1 0 2
```

Fast code:

```python
import numpy as np
i = np.arange(3)
for j in range(2):
    print i*j,

>>> [0 0 0] [1 0 2]
```

Always try to avoid nested for loops
If-elif-else clauses and Logical comparison

for i in range(10):
    if i<5:
        print i, 'is small.'

>>> 0 is small.
>>> 1 is small.
>>> 2 is small.
>>> 3 is small.
>>> 4 is small.
for i in range(10):
    if i<5:
        print i, 'is small.'
    elif i==5:
        print i, 'is 5.'

>>> 0 is small.
>>> 1 is small.
>>> 2 is small.
>>> 3 is small.
>>> 4 is small.
>>> 5 is 5.
If-elif-else clauses and Logical comparison

```python
for i in range(10):
    if i<5:
        print i, 'is small.'
    elif i==5:
        print i, 'is 5.'
    else:
        print i, 'is large.'

>>> 0 is small.
>>> 1 is small.
>>> 2 is small.
>>> 3 is small.
>>> 4 is small.
>>> 5 is 5.
>>> 6 is large.
>>> 7 is large.
>>> 8 is large.
>>> 9 is large.
```
If-elif-else clauses and Logical comparison

```python
for i in range(10):
    if i<5:
        print i, 'is small.'
    elif i==5:
        print i, 'is 5.'
    else:
        print i, 'is large.'

>>> 0 is small.
>>> 1 is small.
>>> 2 is small.
>>> 3 is small.
>>> 4 is small.
>>> 5 is 5.
>>> 6 is large.
>>> 7 is large.
>>> 8 is large.
>>> 9 is large.
```

```python
for animal in ['cat','fish','horse']:
    if animal is 'cat':
        print 'It is a', animal
    if animal is 'fish':
        print animal, 'has', len(animal), 'letters'
    if animal is not 'horse':
        print 'It is not a horse'
    else:
        print 'It is a', animal
```

The length of any object can be computed with the len() statement.
More on numpy

- Make use of arrays instead on lists (huge performance gain)
- Computations can be performed on entire array instead of individual elements

**Standard python:**

```python
a = range(10)
b = []
for i in range(len(a)):
    b.append(a[i] * 2)
print b
```

```python
>>> [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

**Numpy:**

```python
import numpy as np
a = np.arange(10)
b = a*2
print b
```

```python
>>> [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Lists can be converted to numpy array by typing:

```python
a = np.array(a)
```
More on numpy – Functions

- Initialize empty array to fill it with data
  
  ```python
  np.zeros(100)  # 1-dimensional array
  np.zeros((100, 200))  # 2-dimensional array
  ```

- Compute correlation between array of time series
  
  ```python
  np.corrcoef(time_series_array)
  np.corrcoef(time_series_a, time_series_b)
  ```

- Initialize array of evenly spaced values
  
  ```python
  np.arange(n0, n1, stepsize)
  np.linspace(n0, n1, number_of_steps)
  ```

- Standard deviation, mean and absolute values of a time series
  
  ```python
  np.mean(time_series)
  np.std(time_series)
  np.abs(time_series)
  ```

- Loading .txt files
  
  ```python
  np.loadtxt(filename)
  ```
More on numpy – Indexing

```python
a = np.arange(6).reshape(2,3)

print a
>>> array([[ 0,  1,  2],
          [ 3,  4,  5]])

print a[0]
>>> [ 0,  1,  2]

print a[1, 1]
>>> 4

print a[:, 1]
>>> [ 1  4]

print a[:, :2]
>>> array([[ 0,  1],
          [ 3,  4]])

print a.T
>>> [[ 0  3]
     [ 1  4]
     [ 2  5]]
```

Always check
http://docs.scipy.org/doc/numpy/reference/routines.html for help
Solving Ordinary Differential Equations

http://beavotron.deviantart.com/art/Fox-and-Rabbit-92840871
Solving Ordinary Differential Equations

- Described by the Lotka-Volterra model

\[
\frac{dx}{dt} = \alpha x - \beta xy \\
\frac{dy}{dt} = \delta xy - \gamma y
\]

http://beavotron.deviantart.com/art/Fox-and-Rabbit-92840871
from scipy.integrate import odeint
import numpy as np
import matplotlib.pyplot as plt

# Define the ODE
def LotkaVolterra(y,t,parameters):
    return [parameters[0]*y[0]-parameters[1]*y[0]*y[1],
            parameters[3]*y[0]*y[1]-parameters[2]*y[1]]

p = [0.6,0.1,1.5,0.75]  # parameter values
y0 = [1.0,1.0]  # initial conditions
t = np.linspace(0,20,1000)  # times for integration

res = odeint(LotkaVolterra,y0,t,args=(p,))  # solve the ODE

# plot the results
plt.plot(t,res[:,0],label="rabbits")
plt.plot(t,res[:,1],label="foxes")
plt.legend()
plt.show()
Solving Ordinary Differential Equations
Solving Stochastic Differential Equations

- In many real world applications there is stochastic noise.
- This is especially the case in a non-linear, multi-scale system as the earth system.
Solving Stochastic Differential Equations

• In many real world applications there is stochastic noise

• This is especially the case in a non-linear, multi-scale system as the earth system

• Thus we often need to work with Stochastic Differential Equations (SDE)

• In general we write:

\[ dX_t = a(X_t) \, dt + b(X_t) \, dW_t \]
Solving Stochastic Differential Equations

- In many real world applications there is stochastic noise
- This is especially the case in a non-linear, multi-scale system as the earth system
- Thus we often need to work with Stochastic Differential Equations (SDE)
- In general we write:

\[ dX_t = a(X_t) \, dt + b(X_t) \, dW_t \]

- drift term
- diffusion term
- Wiener process
Solving Stochastic Differential Equations

\[ \text{d}X_t = a(X_t) \text{d}t + b(X_t) \text{d}W_t \]

- This SDE can numerically be solved using the Euler-Maruyama scheme (alternatives are the Milstein or Runge-Kutta methods)

1) discretization of time into \( N \) intervals of length \( \Delta t \)

2) solve for each time step as:

\[
Y_n = Y_{n-1} + a(Y_{n-1}) \Delta t + b(Y_{n-1}) \Delta W \\
\mathcal{N}(0, \Delta t)
\]
Solving Stochastic Differential Equations

Example: Ornstein-Uhlenbeck process

\[ dX_t = \Theta \cdot (\mu - X_t) \, dt + \sigma \, dW_t \]
import numpy as np
import matplotlib.pyplot as plt

t_0 = 0        # define model parameters
t_end = 2
length = 1000
theta  = 1.1
mu     = 0.8
sigma  = 0.3

t = np.linspace(t_0, t_end, length)   # define time axis
dt = np.mean(np.diff(t))

y = np.zeros(length)
y0 = np.random.normal(loc=0.0, scale=1.0)  # initial condition

drift = lambda y, t: theta*(mu-y)       # define drift term, google to learn about lambda
diffusion = lambda y, t: sigma          # define diffusion term
noise = np.random.normal(loc=0.0, scale=1.0, size=length)*np.sqrt(dt) #define noise process

# solve SDE
for i in xrange(1, length):
    y[i] = y[i-1] + drift(y[i-1], i*dt)*dt + diffusion(y[i-1], i*dt)*noise[i]

plt.plot(t, y)
plt.show()
Solving Stochastic Differential Equations

one realization
Solving Stochastic Differential Equations

multiple realizations