

データサイエンス

第2回

情報理工学系研究科
創造情報学専攻
中山 英樹

講義の準備： 環境構築 （再）

- 資料を一部ずつとってください
 - USBを回覧しています
 - 必要なファイルだけコピーして回してください
 - 講義のwebページ
 - <http://www.nlab.ci.i.u-tokyo.ac.jp/~nakayama/ds13>
（「中山英樹、個人HP」で検索）
 - ID: dslecture Pass: d@tsc¥13
（¥はバックスラッシュ）
-

本日の内容

- 道具の紹介 & 使い方
 - Python
 - IPython, NumPy, etc.
 - R
 - Rstudio
 - 統計基礎
-

Perl

- 1987年 Lally Wall
- Practical Extraction and Report Language
- インタープリタ型
- 大量テキストデータの抽出・操作・集計が得意
 - 正規表現などは非常に書きやすい
 - 現在でも十分使える場合も多い
- コーディングの制限が緩い。フリーダム。
 - *There's More Than One Way To Do It*
 - 逆に嫌われる場合も…
- マルチプラットフォーム (UNIX, Win, Mac)
- GPL, Artistic License

Python

- 1990年頃, Guido van Rossuma
- インタープリタ型、オブジェクト指向
- オープンソース
- マルチプラットフォーム
- インデント(字下げ) が必須文法
 - 適当にやると実行時エラー。。
 - 自然に整ったコードが書ける印象

c.f. 空飛ぶモンティ・パイソン (英BBC)

Ruby

- 1993年, まつもとゆきひろ氏
- 日本発のオープンソース
- オブジェクト指向スクリプト言語
 - 手軽にオブジェクト指向
 - データ型は全てオブジェクト
- Perlを代替可能であることを最初から重要視
 - テキスト処理は同じくらい得意
 - *Diversity is Good*
- Ruby on Rails (Webアプリケーションフレームワーク)の登場でブレイク

Python for scientific programming

- 2000年頃から、科学計算・データ分析用のライブラリが急速に普及
 - NumPy
 - SciPy
 - scikit-learn
 - Matplotlib
 - IPython
 - Pandas
 - Theano
- Matlabを置き換える流れ
 - (↑ライセンス高い…)

なぜPython ?

- インタプリタ型、コーディングがしやすい
- 言語自体がもともと世界的に人気、オープンソース
- C, C++, FORTRANとの統合が容易
 - 比較的簡単にモジュール化できる（ラッパー）
 - Cython: Pythonのコードを自動でCへ変換、コンパイルしてモジュール化
- 研究からプロダクト開発まで幅広く対応
 - MatlabやRはプロトタイピングは楽だが、スケールしない

弱点もある

- 実行速度はJavaやC++より遅い
 - ループが入ると絶望的
- マルチスレッディングが苦手
 - global interpreter lock (GIL)
 - 単一のプロセスで実行できるPythonスレッドは一つだけ
- パフォーマンスを上げようと思ったらC, C++との連携は必須
 - OpenMP
 - Cython を使えば楽？
 - 本講義ではやりません

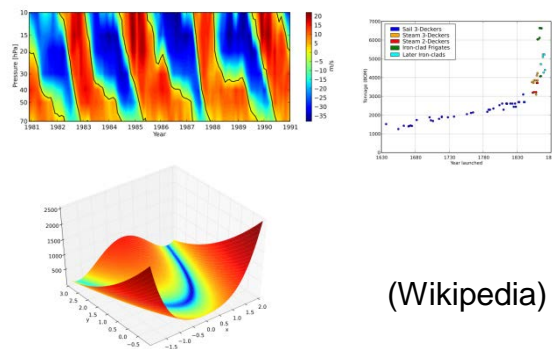
便利なライブラリ/ツールたち

- NumPy (Numerical Numpy)
 - ndarray 高速な多次元配列 (行列)
 - 行列演算
 - 線形代数、フーリエ変換など
 - 実装はCやFortran
- SciPy (Scientific Python)
 - 高度な数値計算
- scikit-learn
 - 機械学習、データマイニング

便利なライブラリ/ツールたち

- matplotlib

- グラフ描画ツール
- 使いこなすとかなりいろいろできる



- IPython

- 前述のライブラリを統合的に扱うのに便利なシェル
- データ処理とmatplotlibによる可視化 (Qtコンソール)
- HTML Notebook

- Spyder

- Matlabライクな統合開発環境

おまけ

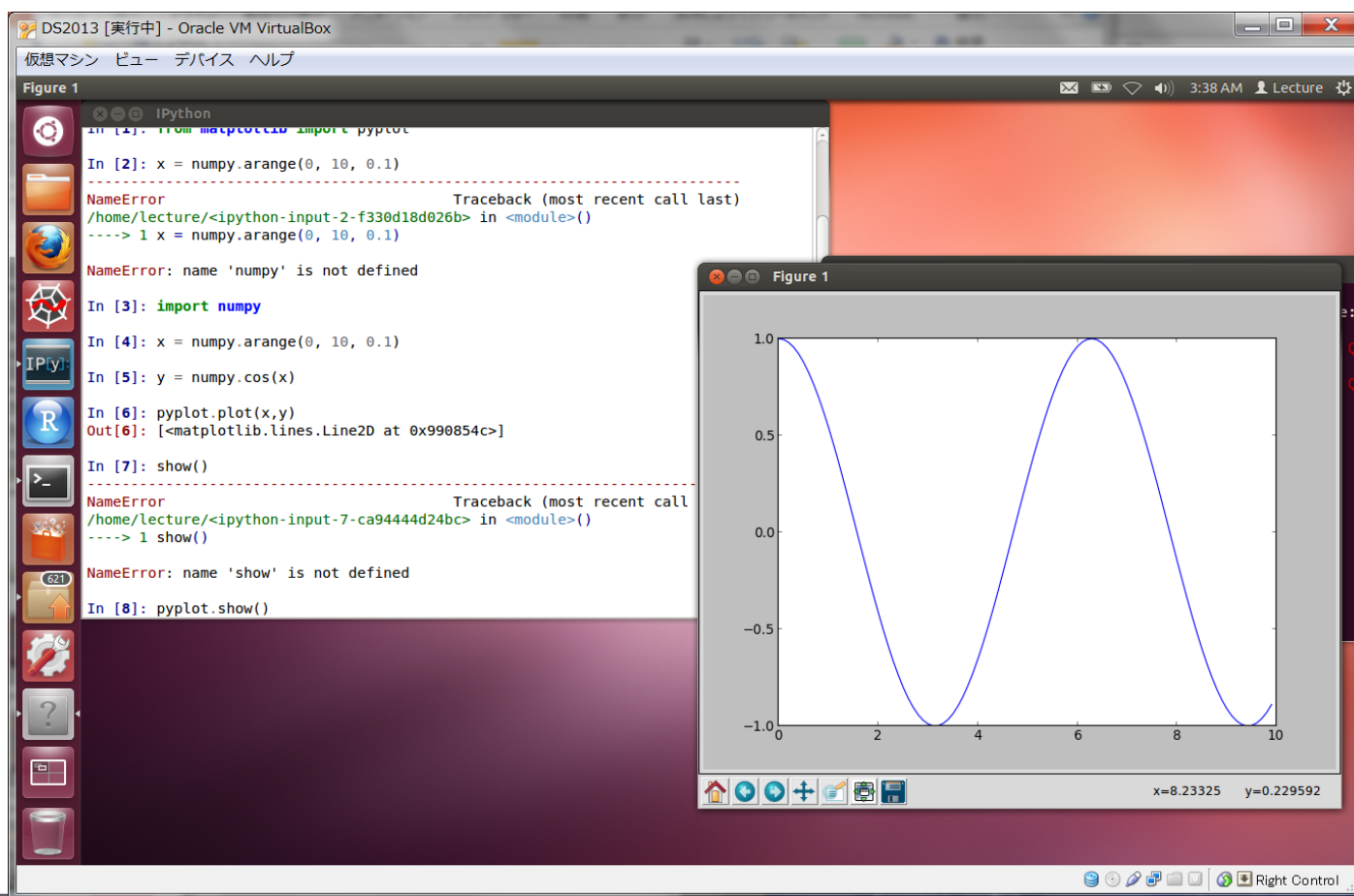
- SciRuby
 - <http://sciruby.com/>
 - 2013年から機械学習のコードなども実装



- Ruby has for some time had no equivalent to the beautifully constructed NumPy, SciPy, and matplotlib libraries for Python. We believe that **the time for a Ruby science and visualization package has come.**

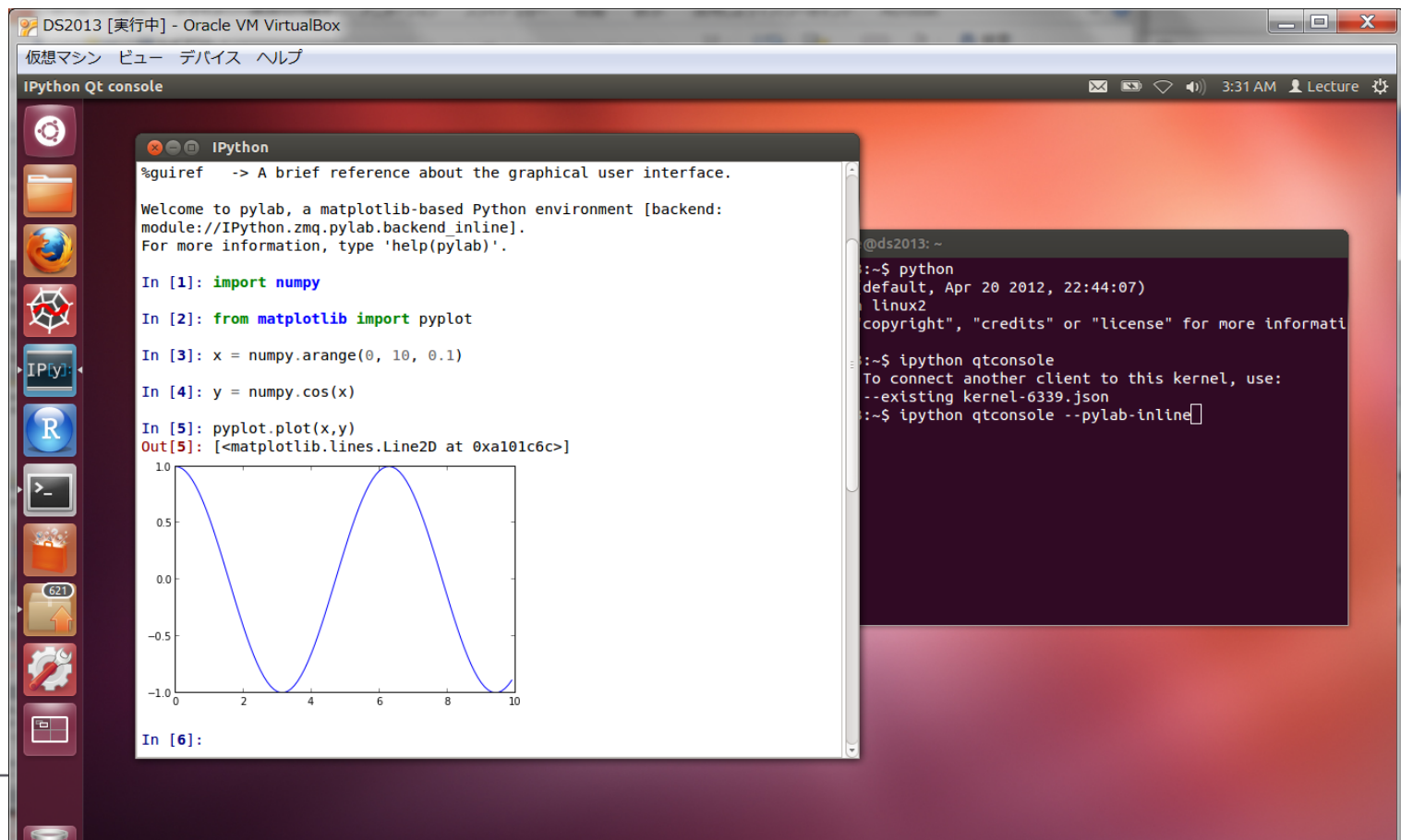
IPythonインタプリタ

- matplotlibでグラフを書いてみた例



IPython: インライン表示

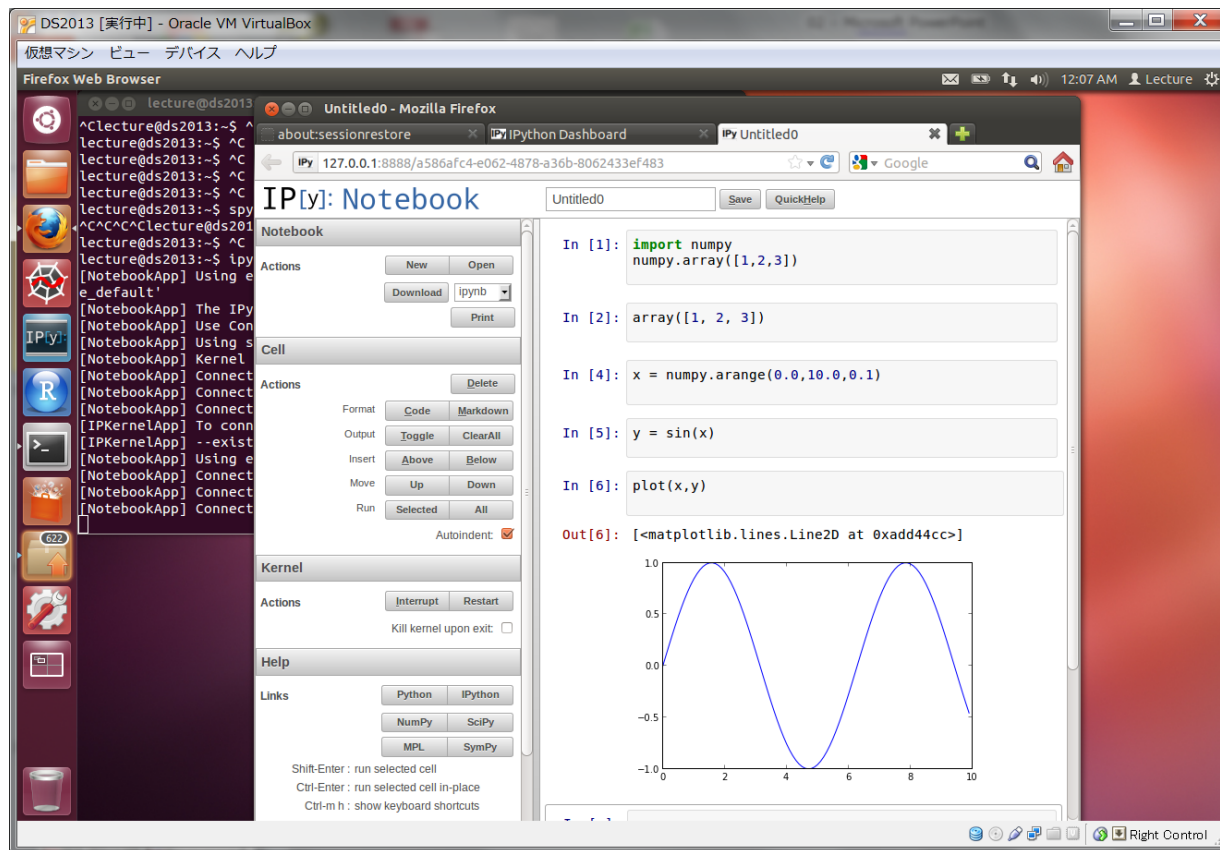
- グラフがコンソール内に表示
- しばらくこれで練習します



(参考) IPython HTML Notebook

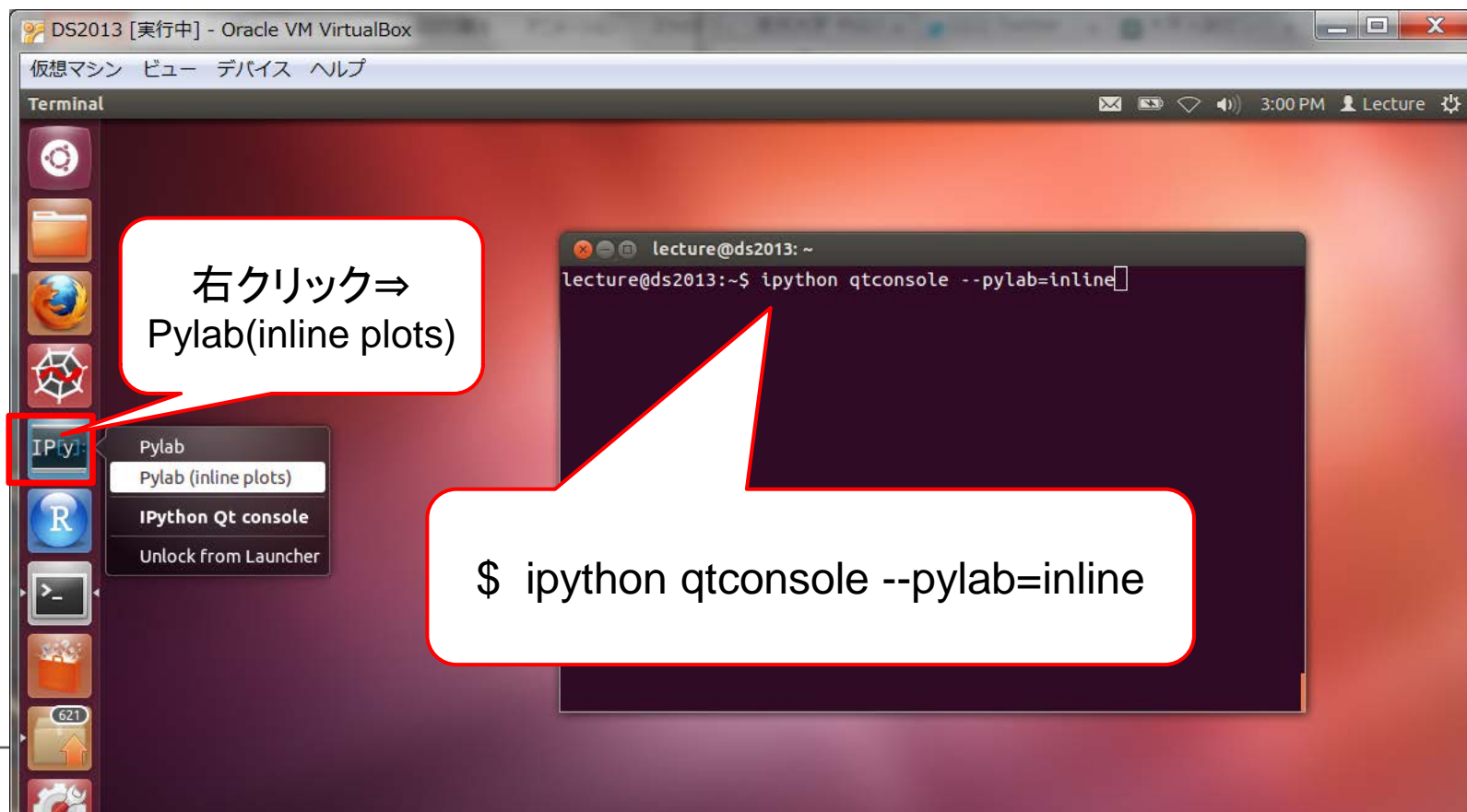
- ブラウザベースでも使える

\$ ipython notebook --pylab=inline (← linuxコンソールから)



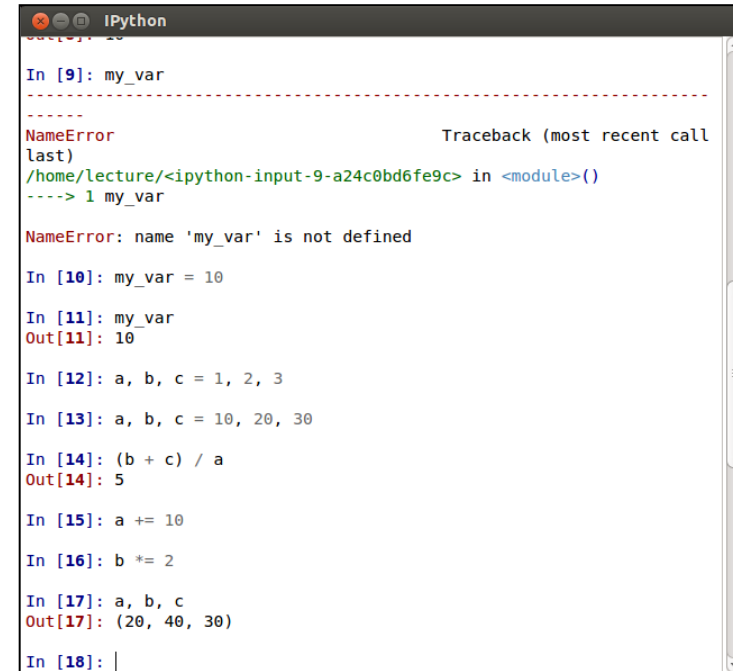
IPython: 起動

- インライン指定を忘れずに



電卓替わり

In [1]: 1 + 1	#コメント文
Out[1]: 2	
In [2]: 1.0 + 1.0	#不動小数点数
Out[2]: 2.0	
In [3]: 7.0 / 2	#浮動小数点数の割り算
Out[3]: 3.5	
In [4]: 7 / 2	#整数の割り算
Out[4]: 3	
In [5]: 7.0 // 2.0	#ダブルスラッシュは整数商を強制
Out[5]: 3.0	
In [6]: 7 % 2	#剰余
Out[6]: 1	
In [7]: 3 ** 2	#累乗
Out[7]: 9	
In [8]: _ + 1	# _は最後の値を受ける
Out[8]: 10	



```
IPython
In [9]: my_var
-----
NameError                                Traceback (most recent call
last)
/home/lecture/<ipython-input-9-a24c0bd6fe9c> in <module>()
----> 1 my_var

NameError: name 'my_var' is not defined

In [10]: my_var = 10

In [11]: my_var
Out[11]: 10

In [12]: a, b, c = 1, 2, 3

In [13]: a, b, c = 10, 20, 30

In [14]: (b + c) / a
Out[14]: 5

In [15]: a += 10

In [16]: b *= 2

In [17]: a, b, c
Out[17]: (20, 40, 30)

In [18]:
```

変数

- 代入時に作られる

```
In [9]: my_var
```

```
-----  
NameError                                Traceback (most recent call last)  
/home/lecture/<ipython-input-9-a24c0bd6fe9c> in <module>()  
----> 1 my_var
```

```
NameError: name 'my_var' is not defined
```

```
In [10]: my_var = 10
```

```
In [11]: my_var
```

```
Out[11]: 10
```

- やりかたはいろいろ

```
In [13]: a, b, c = 10, 20, 30
```

```
In [14]: (b + c) / a
```

```
Out[14]: 5
```

```
In [15]: a += 10
```

```
In [16]: b *= 2
```

```
In [17]: a, b, c
```

```
Out[17]: (20, 40, 30)
```

NumPy

- 行列計算、数値演算のライブラリ
- 書き方に慣れておく必要がある
- ndarray (n-dimensional array)

```
In [20]: import numpy    #numpy モジュールの機能をインポート
```

```
In [21]: data1 = [6, 7.5, 8, 0, 1]
```

(IPythonを--pylab=inlineで起動した場合は必要ない)

```
In [22]: arr1 = numpy.array(data1)
```

```
In [23]: arr1
```

```
Out[23]: array([ 6. ,  7.5,  8. ,  0. ,  1. ])
```

ndarray (続き)

```
In [7]: data2 = [[1,2,3,4],[5,6,7,8]]    # 二次元配列
```

```
In [8]: arr2 = numpy.array(data2)        # ndarrayへ変換
```

```
In [9]: arr2
```

```
Out[9]:  
array([[1, 2, 3, 4],  
       [5, 6, 7, 8]])
```

```
In [10]: arr2.ndim    #次元数
```

```
Out[10]: 2
```

```
In [11]: arr2.shape   #配列サイズ
```

```
Out[11]: (2, 4)
```

```
In [12]: arr2.dtype   #型チェック
```

```
Out[12]: dtype('int32')    #型はいろいろ (代入した値に依存)
```

```
In [13]: arr2 = numpy.array(data2, dtype=numpy.float64)
```

```
In [14]: arr2.dtype
```

```
Out[14]: dtype('float64')
```

```
In [15]: numpy.zeros(10)    #ゼロベクトル
```

```
Out[15]: array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
```

```
In [16]: numpy.zeros((2,3)) #二行三列のゼロ行列
```

```
Out[16]:  
array([[ 0.,  0.,  0.],  
       [ 0.,  0.,  0.]])
```

```
In [17]: numpy.eye(3)    #単位行列
```

```
Out[18]:  
array([[ 1.,  0.,  0.],  
       [ 0.,  1.,  0.],  
       [ 0.,  0.,  1.]])
```

```
In [19]: arr3 = numpy.arange(15)
```

```
In [20]: arr3
```

```
Out[20]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11,  
                12, 13, 14])
```

```
In [21]: arr3.reshape((3,5))
```

```
Out[21]:  
array([[ 0,  1,  2,  3,  4],  
       [ 5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14]])
```

行列演算

- 通常の演算子では、基本的に要素ごとの演算を行うので注意

```
In [22]: arr =  
numpy.array([[1.,2.,3.],[4.,5.,6.]])
```

```
In [23]: arr
```

```
Out[23]:  
array([[ 1.,  2.,  3.],  
       [ 4.,  5.,  6.]])
```

```
In [24]: arr + arr
```

```
Out[24]:  
array([[ 2.,  4.,  6.],  
       [ 8., 10., 12.]])
```

```
In [25]: arr - arr
```

```
Out[25]:  
array([[ 0.,  0.,  0.],  
       [ 0.,  0.,  0.]])
```

```
In [26]: arr * arr
```

```
Out[26]:  
array([[ 1.,  4.,  9.],  
       [16., 25., 36.]])
```

```
In [27]: 1 / arr
```

```
Out[27]:  
array([[ 1.        ,  0.5        ,  0.33333333],  
       [ 0.25       ,  0.2       ,  0.16666667]])
```

```
In [28]: arr ** 2
```

```
Out[28]:  
array([[ 1.,  4.,  9.],  
       [16., 25., 36.]])
```

行列積

- `numpy.dot` を使う

```
In [2]: x = numpy.array([[1., 2., 3.], [4., 5., 6.]])
```

```
In [5]: y = numpy.array([[6., 23.], [-1., 7.], [8., 9.]])
```

```
In [6]: x
```

```
Out[6]:
```

```
array([[ 1.,  2.,  3.],  
       [ 4.,  5.,  6.]])
```

```
In [7]: y
```

```
Out[7]:
```

```
array([[ 6., 23.],  
       [-1.,  7.],  
       [ 8.,  9.]])
```

```
In [8]: numpy.dot(x,y)
```

```
Out[8]:
```

```
array([[ 28.,  64.],  
       [ 67., 181.]])
```

```
In [9]: x.dot(y)      #numpy.dot(x,y)と同じ
```

```
Out[9]:
```

```
array([[ 28.,  64.],  
       [ 67., 181.]])
```

線形代数

```
In [29]: from numpy.linalg import inv
```

```
In [30]: X = numpy.random.randn(3,3) #ランダム行列
```

```
In [31]: X
```

```
Out[31]:
```

```
array([[ 1.079221 ,  0.33981537,  0.83211947],  
       [-0.8495429, -1.02950923, -0.34767011],  
       [ 0.34320046, -1.45251741, -1.44951243]])
```

```
In [32]: X.T #転置
```

```
Out[32]:
```

```
array([[ 1.079221 , -0.8495429 ,  0.34320046],  
       [ 0.33981537, -1.02950923, -1.45251741],  
       [ 0.83211947, -0.34767011, -1.44951243]])
```

```
In [33]: mat = X.T.dot(X) #転置行列と元の行列の積
```

```
In [34]: inv(mat) #逆行列
```

```
Out[34]:
```

```
array([[ 0.54729351, -0.06832945, -0.06630383],  
       [-0.06832945,  1.44209051, -1.34252336],  
       [-0.06630383, -1.34252336,  1.62397906]])
```

```
In [35]: mat.dot(inv(mat))
```

```
Out[35]:
```

```
array([[ 1.00000000e+00,  1.51354623e-16, -  
        5.59448321e-17],  
       [-4.15384957e-17,  1.00000000e+00, -  
        7.19042881e-16],  
       [-1.55583012e-17,  3.58870919e-16,  
        1.00000000e+00]])
```

SciPy

- 基本的な使い方は同じ
- より高度な数値計算をサポート

```
In [9]: from scipy import linalg
```

```
In [10]: scipy.linalg.svd(numpy.eye(3)) #特異値分解（特に意味はない例だが）
```

```
Out[10]:
```

```
(array([[ 1.,  0.,  0.],  
       [ 0.,  1.,  0.],  
       [ 0.,  0.,  1.]]),  
array([ 1.,  1.,  1.]),  
array([[ 1.,  0.,  0.],  
       [ 0.,  1.,  0.],  
       [ 0.,  0.,  1.])))
```

SciPy

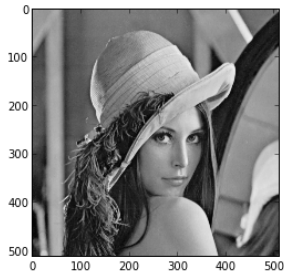
```
In [1]: import scipy
```

```
In [2]: img = scipy.misc.lena()
```

```
In [3]: gray()
```

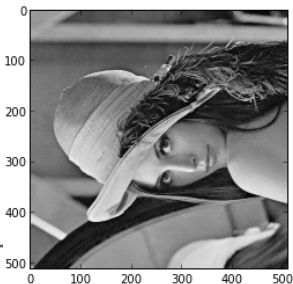
```
In [4]: imshow(img)
```

```
Out[4]: <matplotlib.image.AxesImage at 0x95fb28c>
```



```
In [5]: imshow(img.T)
```

```
Out[5]: <matplotlib.image.AxesImage at 0x96786ac>
```



matplotlib

- グラフ描画ライブラリ

In [1]: `import numpy`

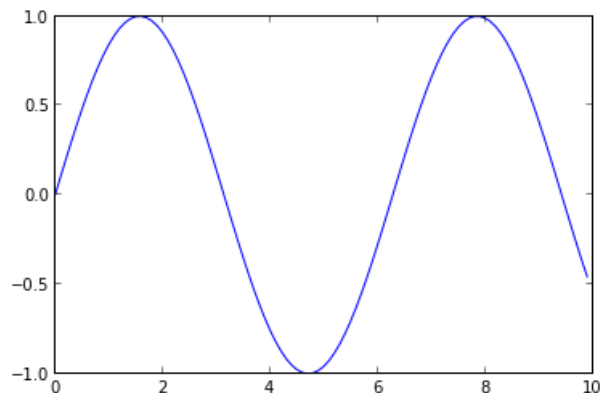
In [2]: `import matplotlib`

In [3]: `x = numpy.arange(0.0, 10.0, 0.1)` #0.0から10.0まで0.1刻みで配列を作成

In [4]: `y = sin(x)`

In [5]: `matplotlib.pyplot.plot(x,y)`

Out[5]: [`<matplotlib.lines.Line2D at 0xa105a2c>`]



Webドキュメントを活用すべし

- **Numpy and Scipy Documentation**
<http://docs.scipy.org/doc/>
- **matplotlib**
<http://matplotlib.org/contents.html>
- 超多機能です
 - こんなのないかな、と思って探したら大体見つかる（と思う）

import について

- ```
>>> import numpy
>>> array([1,2,3]) ×
>>> numpy.array([1,2,3]) ○
```
- ```
>>> import numpy as np #以後npという名前で参照
>>> numpy.array([1,2,3]) ×
>>> np.array([1,2,3]) ○
```

```
numpy.linalg.dot
numpy.linalg.inter
numpy.linalg.outer
numpy.linalg.inv
numpy.fft.fft
numpy.fft.fft2
numpy.distutils
...
```

置き換える名前は経験的に決まっている（お作法）

```
import numpy as np
import scipy as sp
import matplotlib as mpl
import matplotlib.pyplot as plt
```

- ```
>>> from numpy import * #numpyの下にあるものを全部インポート
>>> array([1,2,3]) ○
```

# PyLab

- NumPy, SciPy, matplotlibを配下に持つメタパッケージ
- `from pylab import *`  
とすれば全ての関数がグローバル名前空間に配置
- IPythonは、インラインモード(`--pylab=inline`)の場合これをやっている
  - なので、“array”、“plot” などそのまま使える

```
In [1]: import numpy

In [2]: import matplotlib

In [3]: x = numpy.arange(0.0, 10.0, 0.1)

In [4]: y = sin(x)

In [5]: matplotlib.pyplot.plot(x,y)
```



```
In [1]: x = arange(0.0, 10.0, 0.1)

In [2]: y = sin(x)


In [3]: plot(x,y)
```

実はこっちでもOK

# 実際は…

- 対話的に使う時は便利なので `import *` でもよい
- スクリプト（プログラム）を書くときは必要なものだけ `import` するのが望ましい
  - 名前が衝突する恐れがあるし、無駄


```
from pylab import *
svd(eye(d))
```



```
from numpy import eye, array
from numpy.linalg import svd
```

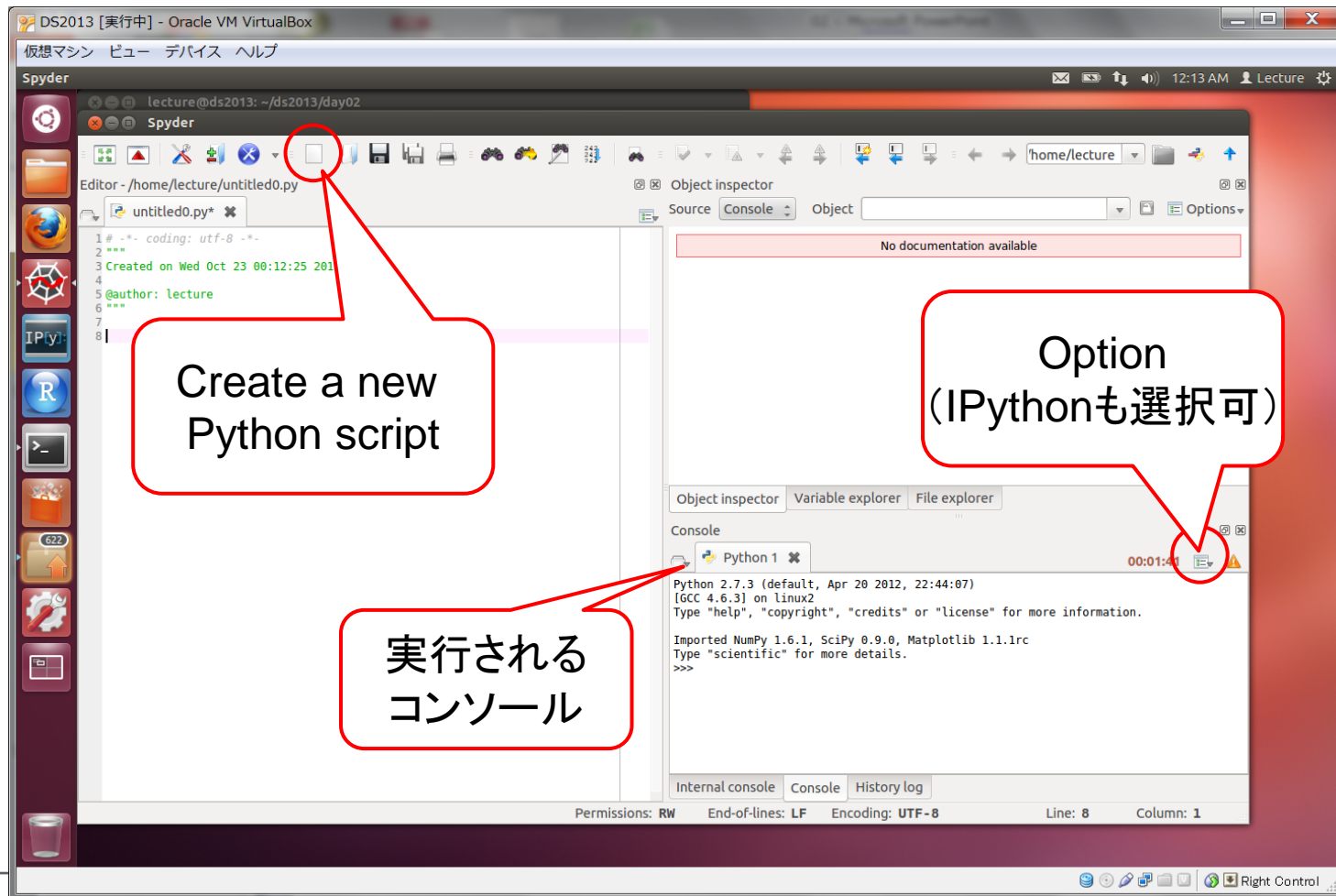


```
import numpy as np
np.linalg.svd(np.eye(3))
```



# Spyder

- matlabライクな統合開発環境



# Spyder

Figure 1

Editor - /home/lecture/ds2013/day02/test.py

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Wed Oct 23 00:12:25 2013
4
5 @author: lecture
6 """
7
8 import numpy as np
9 import matplotlib.pyplot as plt
10
11 x = np.arange(0.0, 10.0, 0.1)
12 y = np.sin(x)
13
14 plt.plot(x,y)
15
16 plt.show()
```

F5 or F6  
で実行

Ctrl + S  
で保存

import numpy as np  
import matplotlib.pyplot as plt

x = np.arange(0.0, 10.0, 0.1)  
y = np.sin(x)

plt.plot(x,y)

plt.show()

Figure 1

Console

```
>>> runfile(r'/home/lecture/ds2013/day02/test.py', wdir=r'/home/lecture/ds2013/day02')
>>> runfile(r'/home/lecture/ds2013/day02/test.py', wdir=r'/home/lecture/ds2013/day02')
>>> runfile(r'/home/lecture/ds2013/day02/test.py', wdir=r'/home/lecture/ds2013/day02')
>>> runfile(r'/home/lecture/ds2013/day02/test.py', wdir=r'/home/lecture/ds2013/day02')
>>> runfile(r'/home/lecture/ds2013/day02/test.py', wdir=r'/home/lecture/ds2013/day02')
>>> runfile(r'/home/lecture/ds2013/day02/test.py', wdir=r'/home/lecture/ds2013/day02')
>>>
```

Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 7 Column: 1  
lecture@ds2013:~/ds2013/day02\$ spyder

# Pythonまとめ

- 対話的に使うならIPython (pylab)、プログラム開発環境にはSpyderがおすすめ
- importや名前空間の仕組みを理解しましょう
- ndarrayを使った行列演算、数値計算に慣れましょう
- 細かいことは随時やっていきます

# R言語

- 1996年, Ross Ihaka & Robert Gentleman
- 統計解析、データマイニングに適した環境
- ベクトル処理言語
- 高速な組みこみ関数群
  - 更に、C, C++, Fortranなどの外部プログラムと動的リンク可能
- 優れたグラフ機能
- オープンソース, CRANネットワーク
  - 最新の研究成果も比較的すぐ実装されたりする

SAS, SPSS, S言語等の商用解析ツールを代替

# R言語：欠点

- 大規模データが処理できない
  - 基本的にオンメモリ前提
- 並列化の実装が難しい

(ただし、拡張ツールなどはいろいろある)

- アルゴリズムの中身をいじるには向いていない

# 基本操作

- 起動、終了

```
$ R
> q() #終了
```

- オブジェクトへ代入

```
> x<-1
> x #オブジェクトの中身を表示
[1] 1
> x<-c(1,2,3,4,5,6)
[1] 1 2 3 4 5 6
```

- ベクトル演算

```
> x^2
[1] 1 4 9 16 25 36
```

- 関数の例

```
> mean(x)
[1] 3.5
> round(mean(x),0)
[1] 4
```

# データ型

- マトリックス (数値のみ)

```
> mat<-matrix(c(1,2,0,3), 2, 2) #2行2列の行列
> mat
 [,1] [,2]
[1,] 1 0
[2,] 2 3
> class(mat) #型
[1] "matrix"
```

- データフレーム (表みたいなもの)

```
> df1<-data.frame(cbind(LETTERS[1:4],3:0)) #アルファベット、数値をバインド
> colnames(df1)<-c("X","Y") #列名をつける
> df1
 X Y
1 A 3
2 B 2
3 C 1
4 D 0
> class(df1)
[1] "data.frame"
```

# グラフプロット

```
> iris #フィッシャーのアヤメのデータ (データフレーム型)
```

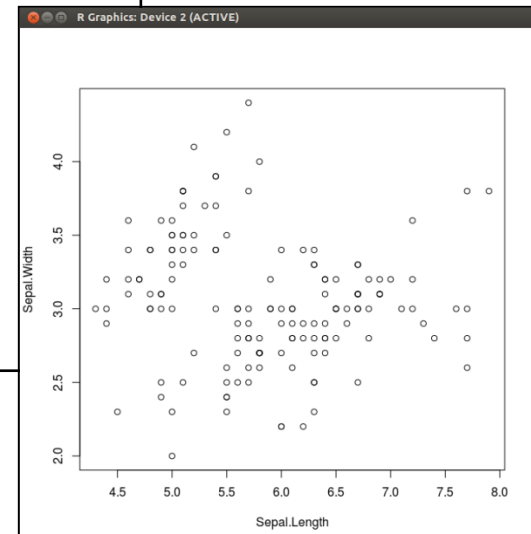
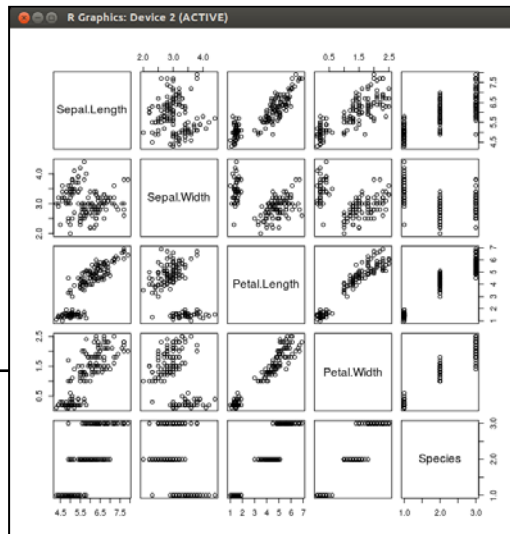
|  | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|--|--------------|-------------|--------------|-------------|---------|
|--|--------------|-------------|--------------|-------------|---------|

|   |     |     |     |     |        |
|---|-----|-----|-----|-----|--------|
| 1 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 2 | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 3 | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 4 | 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 5 | 5.0 | 3.6 | 1.4 | 0.2 | setosa |

(省略)

```
> plot(iris)
```

```
> plot(iris[,1:2])
```



# パッケージ

- さまざまなツールをまとめたもの
- 必要に応じて読み込んで使う

```
> library() #インストール済みのパッケージを表示
Packages in library '/usr/lib/R/library':
base The R Base Package
boot Bootstrap Functions (originally by Angelo Canty for S)
class Functions for Classification
(省略)

> search() #読み込み済みのパッケージを表示
[1] ".GlobalEnv" "package:stats" "package:graphics"
[4] "package:grDevices" "package:utils" "package:datasets"
[7] "package:methods" "Autoloads" "package:base"
```

- 例

```
> library(MASS) #MASSライブラリを読み込む
> help(package="MASS") #MASS
> truehist(c(1:10))
```

# パッケージ

- ないものはインストール
  - rootユーザでないと、インストール先はユーザのホームディレクトリ以下になる

```
> install.packages("randomForest")
> library("randomForest")
> randomForest(formula=Species~., data=iris)
```

Call:

```
randomForest(formula = Species ~ ., data = iris)
```

    Type of random forest: classification

    Number of trees: 500

No. of variables tried at each split: 2

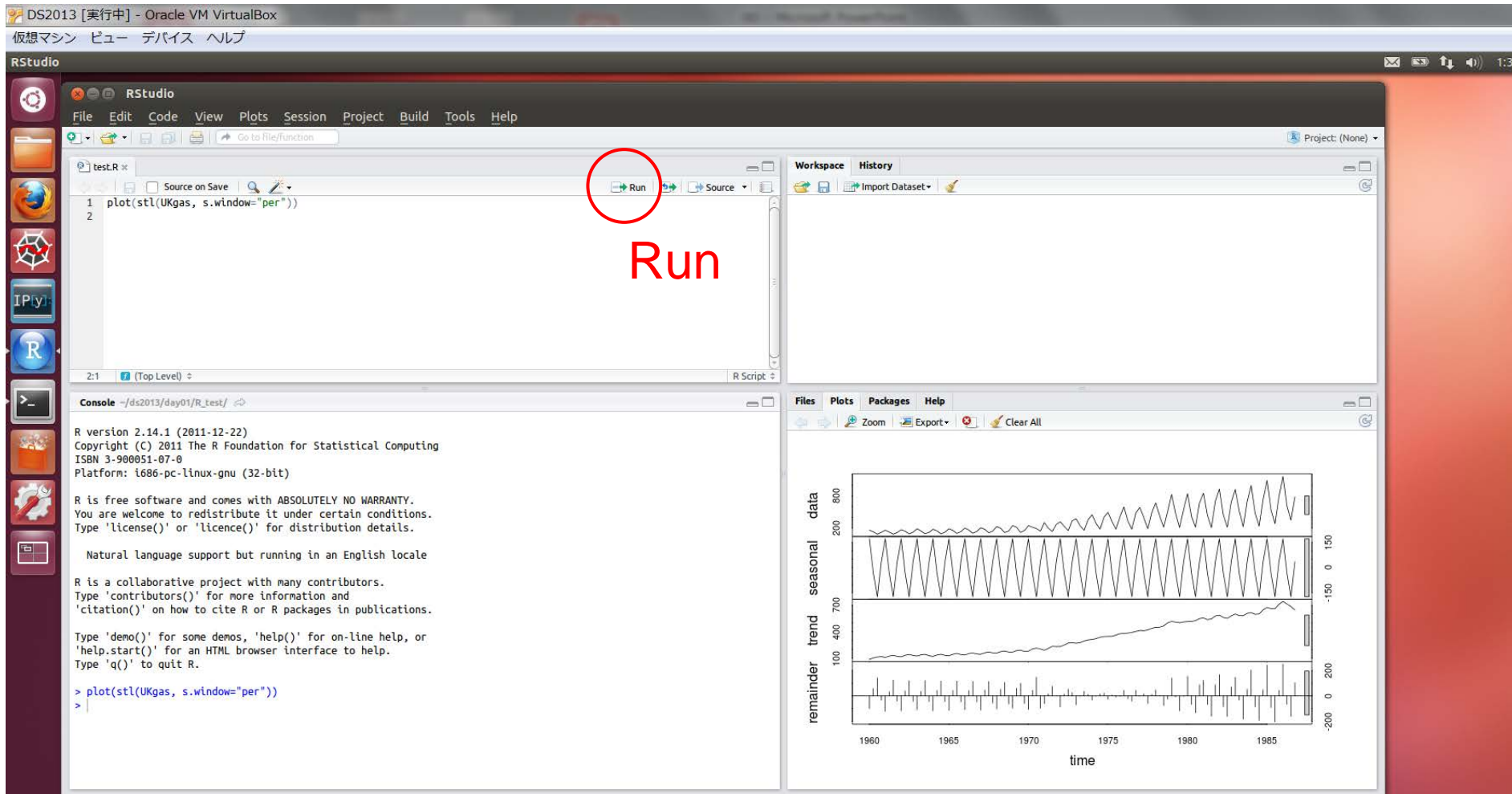
    OOB estimate of error rate: 4.67%

Confusion matrix:

|            | setosa | versicolor | virginica | class.error |
|------------|--------|------------|-----------|-------------|
| setosa     | 50     | 0          | 0         | 0.00        |
| versicolor | 0      | 47         | 3         | 0.06        |
| virginica  | 0      | 4          | 46        | 0.08        |

# Rstudio

- 統合開発環境  
「Run」で一行ずつ実行



# 統計基礎

- データサイエンティスト = ビッグデータを扱う人？
- 全データ (N=全数) を利用すべき？
  - データが大きいとご利益があるらしい？
  - ランダムサンプリングじゃだめなの？
- 正しくデータを使い、判断を行うためには**推定と検定**の概念が重要

# 身近でよくあるパターン

- 修論締切まであとx日…
    - 「新しいシステム作ったけど何か評価しないと。とりあえずラボのみんなに頼んで主観評価しよう！」
    - 従来システム：“2, 3, 1, 2, 3”（平均2.2点）
    - 提案システム：“3, 2, 4, 3, 3”（平均3.0点）
- 勝った！（完）

# 2つの標本の平均を比較する

2つの母集団からそれぞれ選びだした標本がある。これらから、2つの母集団の平均が同じになりえるかどうかを知りたい

- t.test 関数(R)

- $p$ 値を求める。慣例的に、 $p < 0.05$ の場合は平均がおそらく異なる（同じになる可能性は十分小さい）ことを意味し、 $p > 0.05$ の場合は平均が異なる証拠にならない

```
> x<-c(2, 3, 1, 2, 3)
> y<-c(3, 2, 4, 3, 3)
> t.test(x,y,paired=TRUE)
```

```
> t.test(x,y,paired=TRUE)
```

Paired t-test

data: x and y

t = -1.206, df = 4, **p-value = 0.2943**

alternative hypothesis: true difference in means is not equal to 0

# ディスプレイ広告（再）



Apache  
log



ページビュー予測

## 広告商品(契約)の例

2012/4/1~4/15の間、**ファイナンス**のページを訪れた、  
**東京在住で不動産に興味ある30代の男性**に**100万回**表示

# 割合の予測

- ログ中で、ある属性を持ったユーザの割合が重要
    - 考慮すべき属性の組み合わせ数は膨大（数100億～兆！）
      - 性別
      - 年齢
      - 掲載場所（サービスの種類）（ $10^3$ ）
      - 地域（ $10^3$ ）
      - 興味カテゴリ（ $10^3$ ）
      - ドメイン、時間帯、・・・
-

# ログの数え上げ

- 毎日30億ページビュー
- 全部見る必要はなさそうだが、どれくらいサンプリングすべき？
- 直感的には
  - “男”, “女”など, 大きい属性を見るには少しでよさそう
  - “男+30代+車+〇〇県〇〇町+…”  
小さい場合は大丈夫だろうか？

# 比率の信頼区間を求める

TrueとFalseからなる母集団の値の標本がある。この標本データに基づき、母集団のTrueの本当の比率の信頼区間を求めたい

- `prop.test(x, n)` #nが標本サイズ、xがTrueの個数
  - 95%信頼水準の信頼区間を求める

```
> prop.test(1e+09, 3e+09)
0.3333 ~ 0.3334
```

```
> prop.test(1e+05, 3e+05)
0.3316 ~ 0.3350
```

```
> prop.test(1e+05, 3e+09)
3.313e-05 ~ 3.354e-05
```

```
> prop.test(10, 3e+05)
1.694e-05 ~ 6.352e-05
```

```
> prop.test(1e+9, 3e+9)
```

1-sample proportions test with continuity  
correction

data: 1e+09 out of 3e+09, null probability 0.5  
X-squared = 333333333, df = 1, p-value < 2.2e-16  
alternative hypothesis: true p is not equal to 0.5

95 percent confidence interval:  
0.3333165 0.3333502

# Sample-based approach

- Forecasting high-dimensional data [Agarwal et al., SIGMOD'10]
  - ランダムにログをサンプリングし、メモリ上に保持
    - 2000万サンプル、8GB
  - 問い合わせの際、条件を満たすログをon-siteに数え上げる
    - 数十msec
- Bitmap indexing <https://sdm.lbl.gov/fastbit/>

|     | 1 | 2 | 3 | 4 | 5 |  |
|-----|---|---|---|---|---|--|
| 男性  | 1 | 0 | 0 | 0 | 1 |  |
| 女性  | 0 | 0 | 1 | 1 | 0 |  |
| 東京  | 1 | 0 | 0 | 0 | 0 |  |
| 大阪  | 0 | 1 | 1 | 0 | 0 |  |
| 自動車 | 1 | 1 | 0 | 0 | 0 |  |
| 化粧品 | 0 | 0 | 1 | 1 | 0 |  |



count  
“女性=1 AND 化粧品=1”

# 実験

- Yahoo! Inc. のログデータ
  - 一週間分のログで割合計算
  - 2000万サンプル、8GB
- 提案手法（ログカウント）が最もよい予測精度
- 全数に比べるとごくわずかのサンプルであるが、このビジネス要件の範囲内においては十分

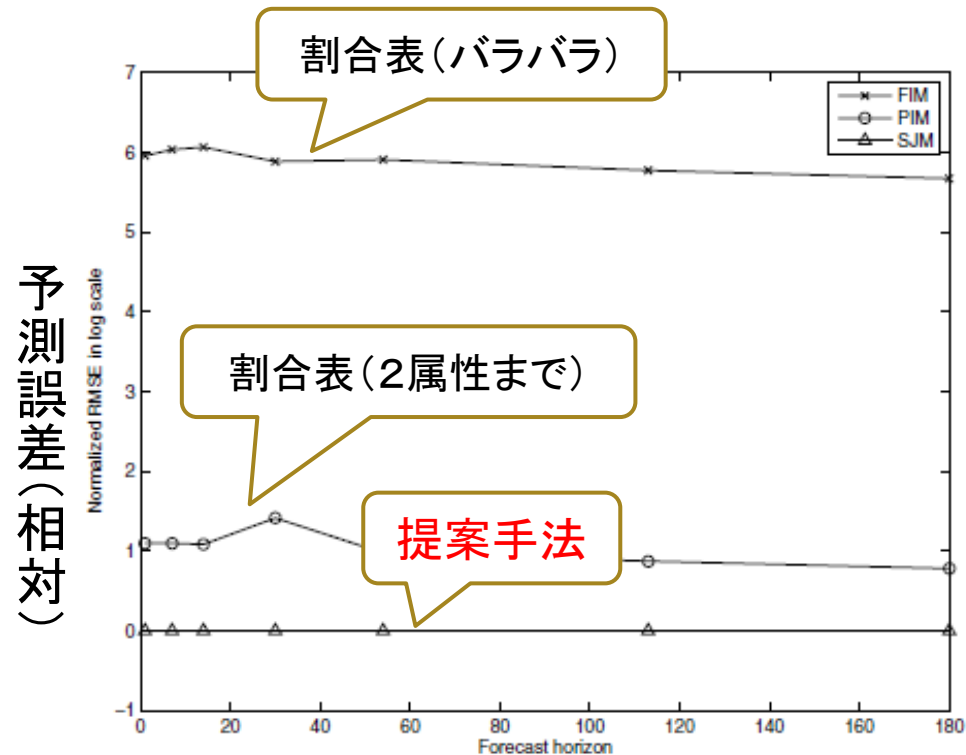


Figure 2: Count Forecast Accuracy: Varying Forecast Horizon (Y-axis shown in logscale, base  $e$ )

# ビッグデータ

- データ量の重要さは目的によって変わる
  - とにかく全部使えばいい、というわけではない
- 特定のアプリケーションに利用することが目的であれば、往々にして費用対効果が薄い場合が多い（と思う）
- データドリブンに知識発見をしたいのであれば、データは大いにこしたことはない
  - 10/1000 と 10万/1000万は全く違う！
  - 多ければ新しいものが見つかる可能性が増える