情報解析技術の教育を革新しよう

- JupyterNotebookによる講義の試み -

2016年6月
釜江常好
（東大、スタンフォード大OB）

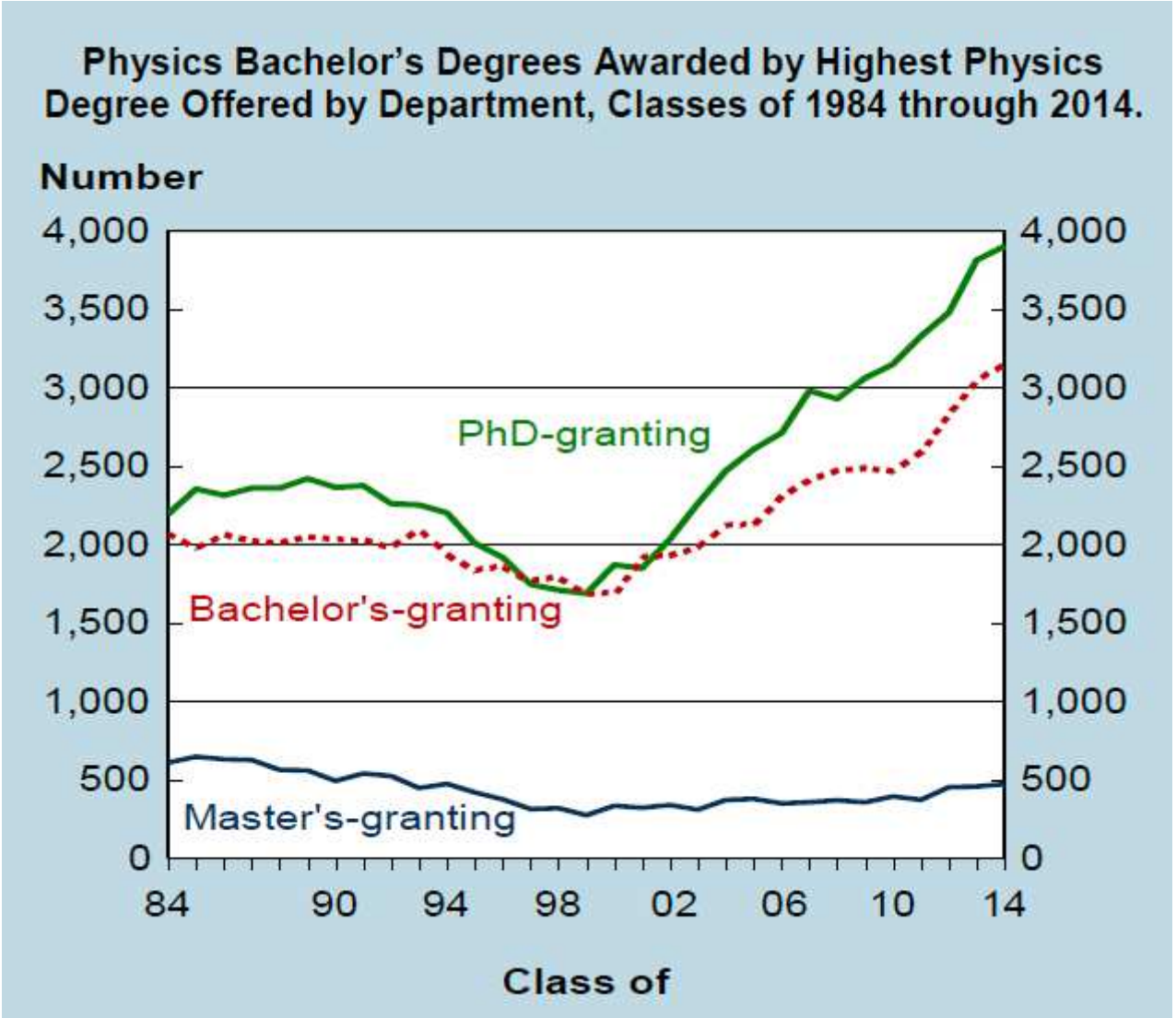
情報解析技術の習得は、産業界や研究教育分野だけでなく、スポーツ関係者、メディア企業などで、最重要視される資質となっている。その要請に答えるべく、大規模なデータを組み合わせ、手の込んだ情報解析を初心者にも可能にする、**画期的なソフトウェア**が開発されている。

しかるに、学生が細分化された専門分野に囲い込まれる日本の大学では、新しい情報解析技術に触れる機会が少ない。その結果、情報解析技術の習得に関しては、欧米の大学に大きく遅れを取っている。

私は情報解析技術の専門家ではないが、米国の大学で研究し、大学院生を指導する中で、新しい情報解析技術に触れてきた。その中で、急速に普及し始めた、汎用性が高い、**IPython Notebook**（**Jupyter Notebook**と改名した）を紹介したい。

American Inst Physの調査結果

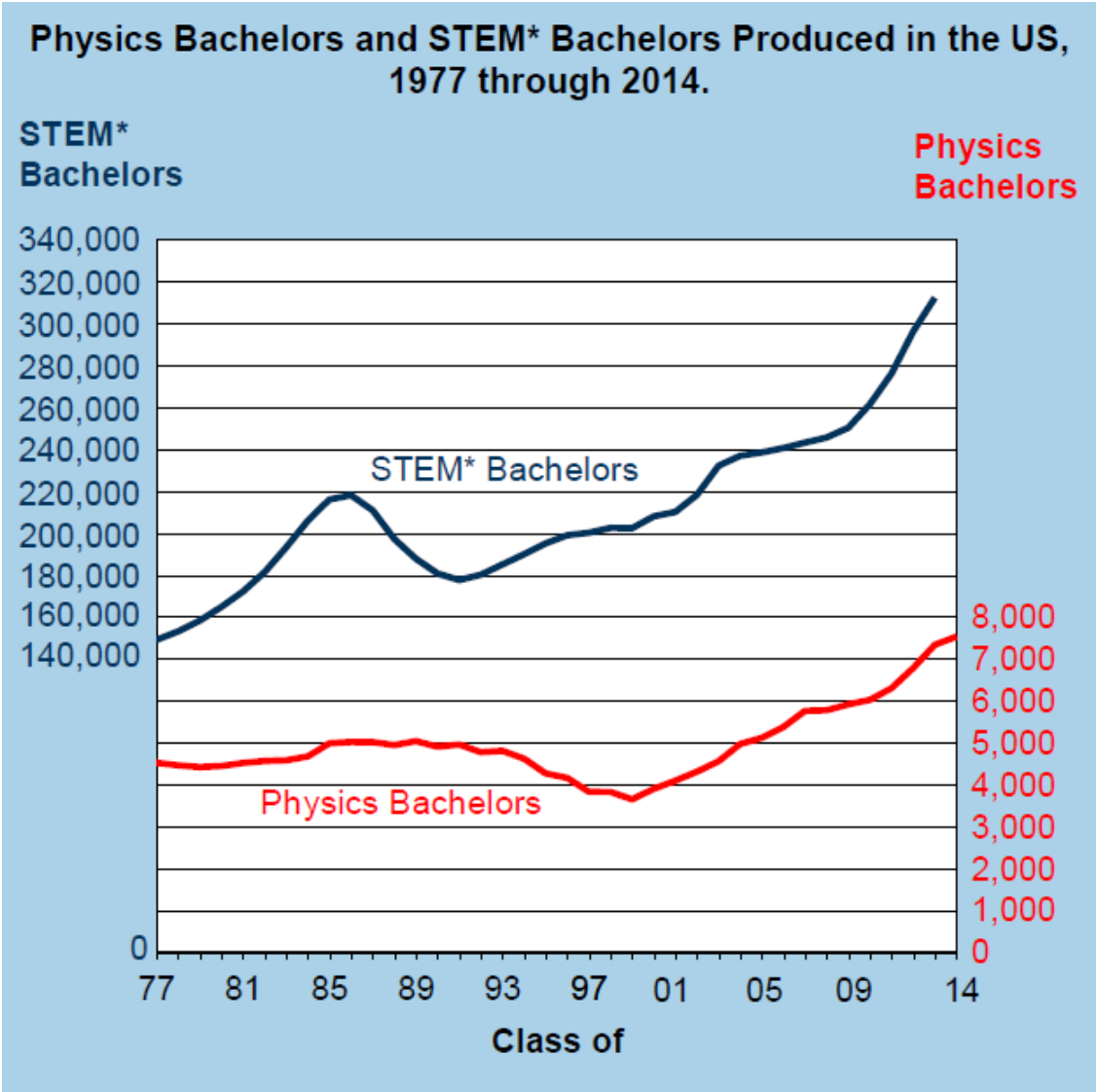
博士課程がある大学で学士号を取得する人が増えている



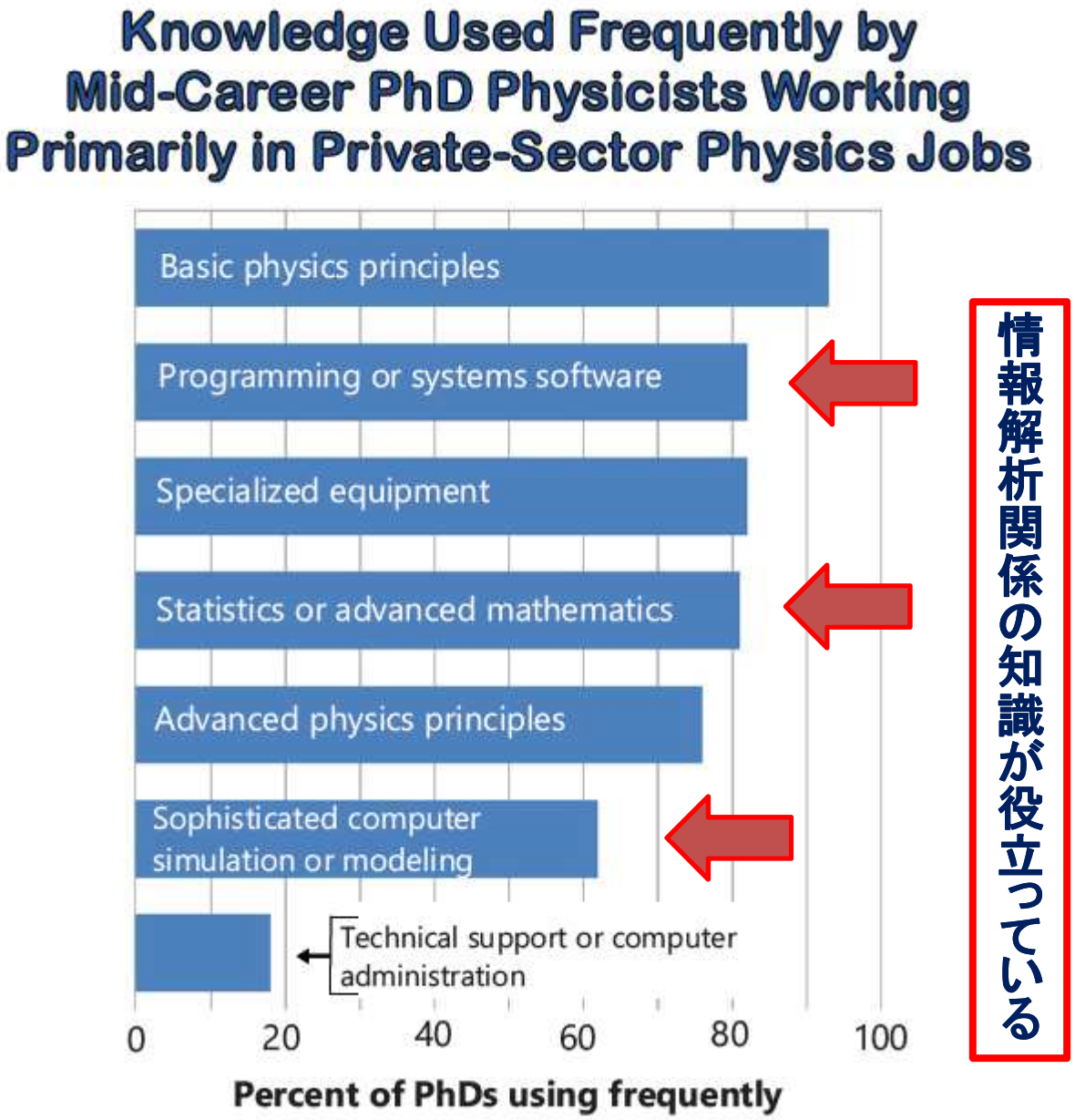
American Inst Physの調査結果

STEM学士の増加

Sci,Tech,Eng,Mathを統合した学士号取得者が増加



American Inst Physの調査結果

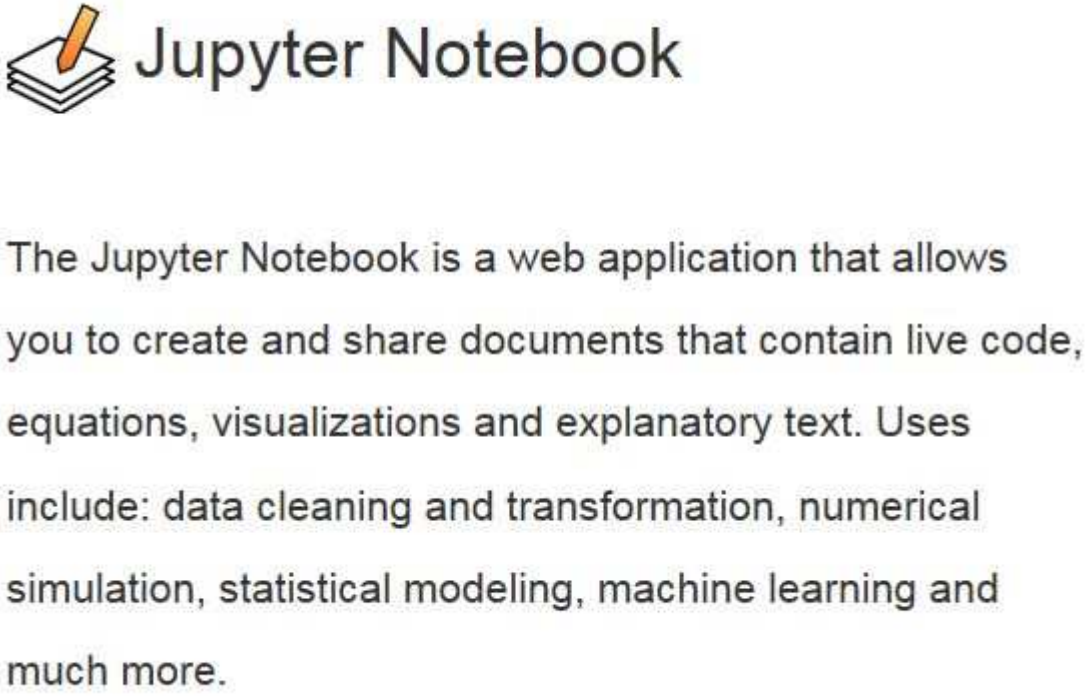


IPythonとは

- もちろん、Python です。
- IPython：Interactive Python の略で、通常はコマンドラインで使う対話的シェル環境のこと。
  - タブ補完
  - 便利なコマンド類(magic command)
    - %who, %whos, %who_ls
    - %reset
    - %magic
  - 容易なヘルプ表示
    - ?command or command? でのヘルプ表示
  - lsとかcd 等も使える：標準pythonでは無理

Jupyter Notebookとは

Jupyter Notebookとは**ブラウザ上で動くアプリケーション**で、
ローカルなLinux、Win、Macなどにインストールできると共に、
リモートでも動かすことが可能です。
Home: http://jupyter.org/



JupyterNotebook とは No.2

インタラクティブなセルと、LaTex、html などを含むマークダウン形式で記述した説明文を並べることができる。

Interactive widgets

Code can produce rich output such as images, videos, LaTeX, and JavaScript. Interactive widgets can be used to manipulate and visualize data in realtime.

多くの言語（40）をサポート

Language of choice

The Notebook has support for over 40 programming languages, including those popular in Data Science such as Python, R, Julia and Scala.

いろいろなサイト(Github、Dropbox等)で共有可能

Share notebooks

Notebooks can be shared with others using email, Dropbox, GitHub and the Jupyter Notebook Viewer.

ビッグデータを解析することが容易

Big data integration

Leverage big data tools, such as Apache Spark, from Python, R and Scala. Explore that same data with pandas, scikit-learn, ggplot2, dplyr, etc.

Youtube from PyCon on Jupyter Notebook

Presentations by Fernando Perez

https://youtu.be/2NSbuKFYyvc

https://youtu.be/26wgEsg9Mcc

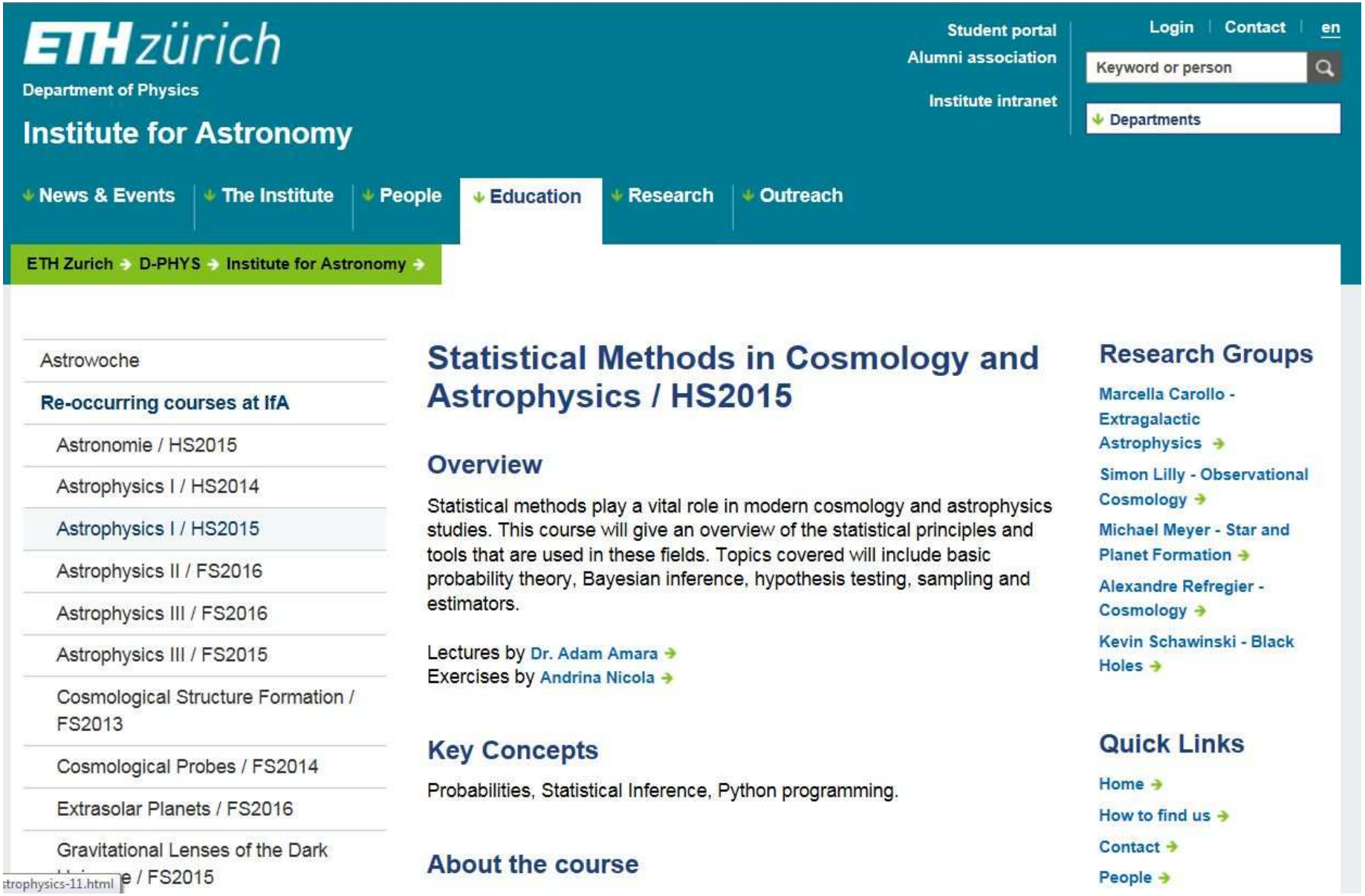
最近のQiitaの記事から

Jupyter NotebookでPythonを経験する

Jupyterのホームページで、Jupyter Notebookを動かしてみる

Jupyter NotebookでPythonに触れてみる

宇宙論や宇宙物理で広く使われている



Univ Southampton講義録

FEEG6002 Numerical Methods: Introduction

Learning Outcomes

After studying the Numerical Methods part of this module you should be able to:

- Solve simple linear equations numerically using an iterative method.
- Solve simple ODEs numerically using a Runge-Kutta method.
- Solve simple PDEs, such as the Laplace equation or the Heat equation in 1D or 2D, using the finite difference method.
- Solve simple eigenvalue problems numerically.
- Recognise the numerical methods that are suitable for solving a given engineering problem.

Assessment

- Coursework due on Wednesday 6 January 2016
- Counts for 100% of the mark for this part of the module, and 50% of the total module mark.

Course notes and delivery

- PDF of lecture notes http://www.southampton.ac.uk/~feeg6002/lecturenotes.html
- Jupyter / IPython notebooks https://jupyter.org/:
  - Useful for executing code live
  - Keeps both theory, implementation, results and analysis in a single place
  - You can easily modify and re-run
  - Can be exported to HTML or PDF
- Snippets: http://www.southampton.ac.uk/~feeg6002/snippets/numerical_methods

JupyterHub

https://github.com/jupyterhub/jupyterhub

JupyterHub: A multi-user server for Jupyter notebooks

Questions, comments? Visit our Google Group:

JupyterHub, a multi-user server, manages and proxies multiple instances of the single-user ~~IPython~~ Jupyter notebook server.

Three actors:

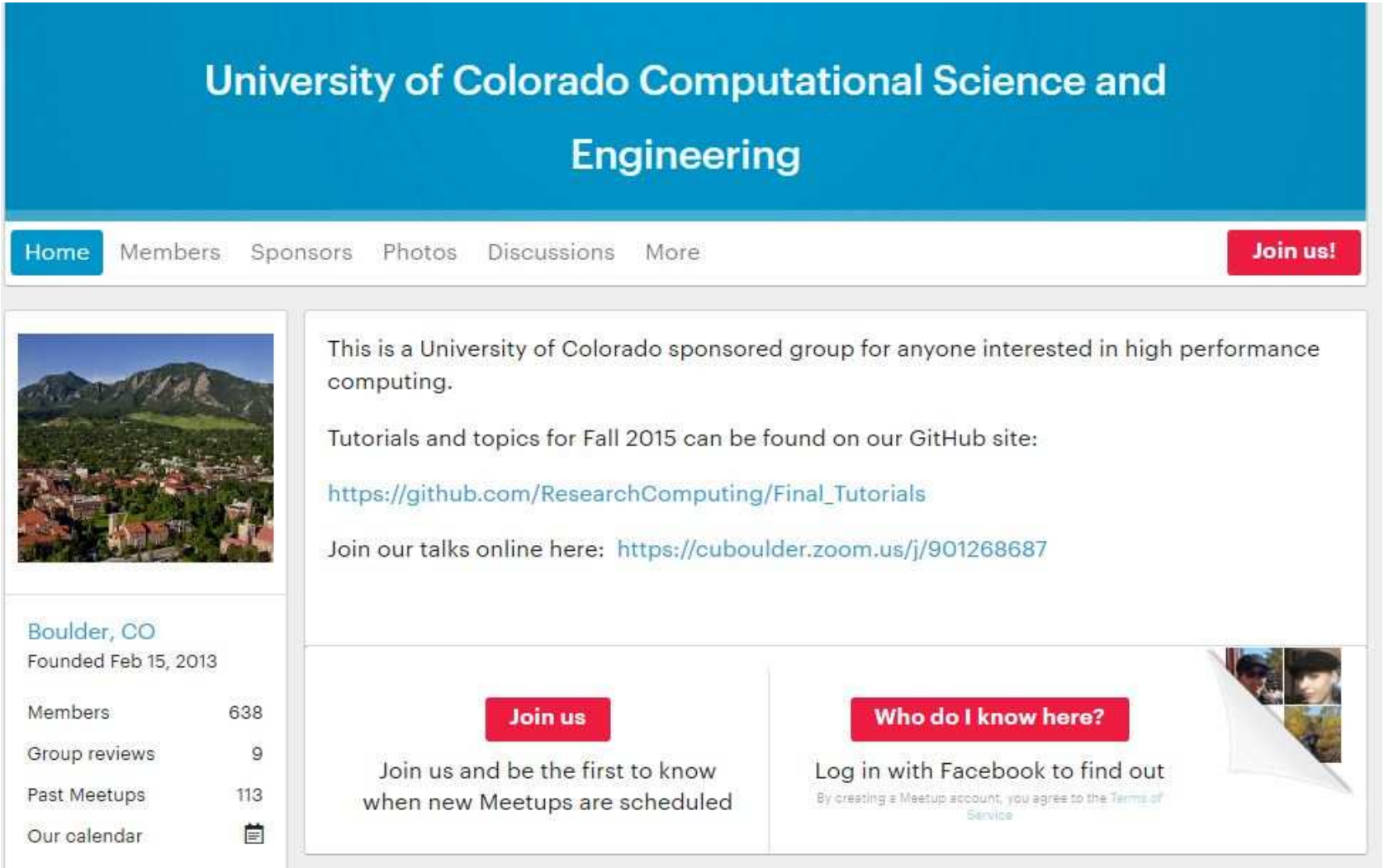
- multi-user Hub (tornado process)
- configurable http proxy (node-http-proxy)
- multiple single-user IPython notebook servers (Python/IPython/tornado)

Basic principles:

- Hub spawns proxy
- Proxy forwards ~all requests to hub by default
- Hub handles login, and spawns single-user servers on demand
- Hub configures proxy to forward url prefixes to single-user servers

U. Colorado Computational Science and Engineering

https://github.com/ResearchComputing/Meetup-Fall-2013



Pythonの国際会議、PyCon での講演

Youtube: Fernando Pérez,　Keynote talk in PyCon 2014

https://youtu.be/2NSbuKFYyvc

Youtube: Fernando Pérez , “Python at your fingertips” PyCon 2012

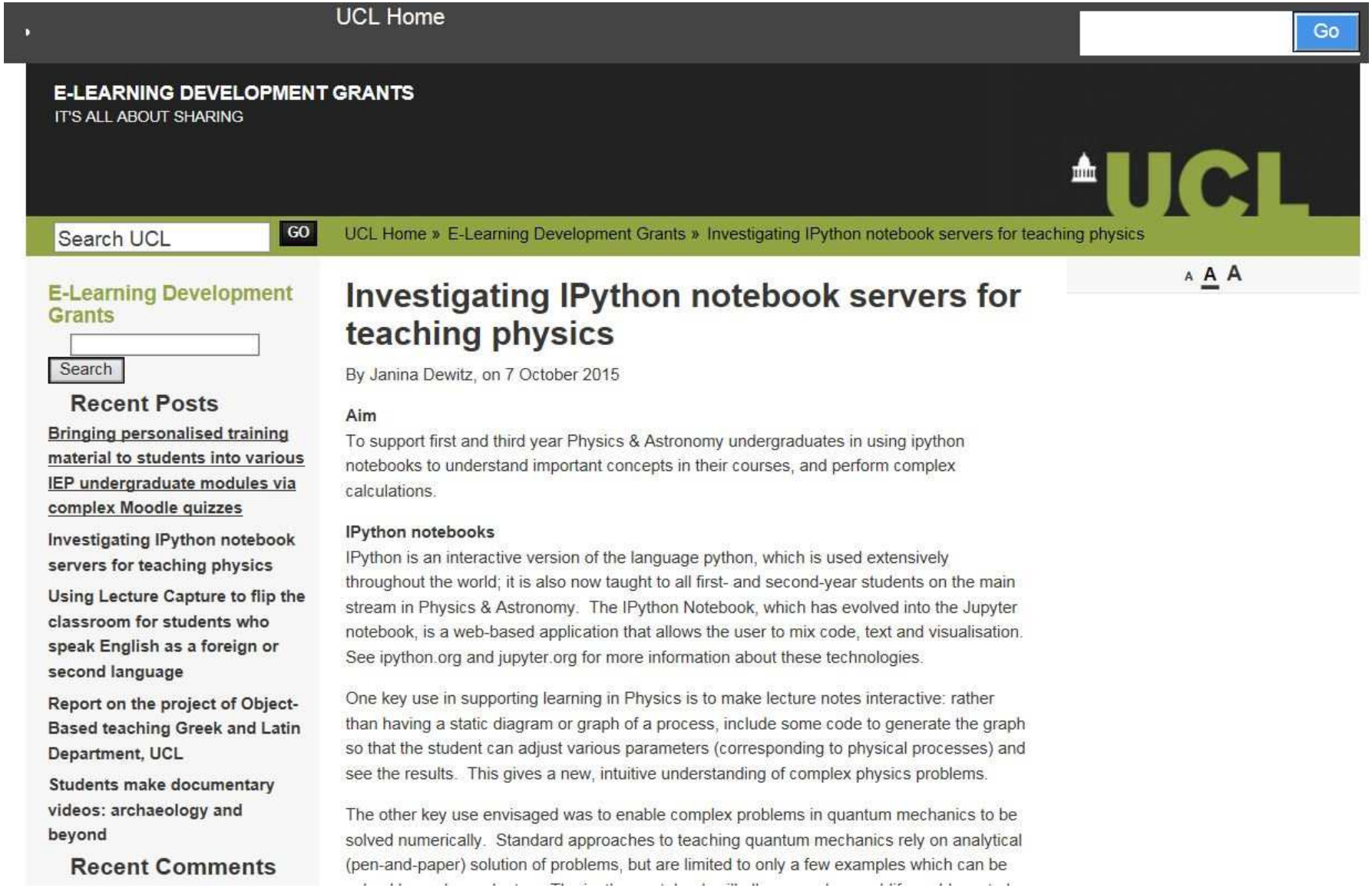
https://youtu.be/26wgEsg9Mcc

公開されている Jupyter Notebook 1

中井悦司さんのCTC教育サービス講演から(2016年5月）
サーバーオペレーションUIとしてのJupyterを使った例

中井悦司さんの本をJupyterNotebookに書き改めた例
「ITエンジニアのための機械学習理論入門」

公開されている Jupyter Notebook 2



JupyterのJuはJuliaからRはR言語から？

-「最速の言語?」Juliaとは -

Julia：スクリプト言語最速? 手軽さと速さを求めた科学技術計算向け言語

使い方 インストール Emacs Ruby 言語比較

一般的に Ruby, Python といったスクリプト言語は手軽に書けるけど、遅いという特徴があります。
今回はスクリプト言語でありながら、速度も求めた Julia という言語を紹介します。 Julia は科学技術計算向けですが、汎用的な用途にも使えると思います。

ちなみに計算時間は速いみたいですが、立ち上がりがすごく遅いので、 タイトルにはちょっと偽りがあります。

- The Julia Language

科学技術計算向け言語

先に科学技術計算の分野と、よく使われている言語について簡単に説明します。

科学技術計算というのは線形代数、数値解析、統計解析など専門的な数値計算をする分野です。 大抵のアプリではディスクアクセスやネットワーク通信が先にスピードネックになることが多いので、 純粋に言語としての速度が求められる分野でもあります。

数値計算に関しては **Fortran** という言語が大昔からあります。 これは簡単なものに限定すれば、いまでも最速の部類です。
ただし、総合的に見た場合は **C, C++ 言語**の方が速いことが多いです。 また、実用性も高いので、速いプログラムを作りたい場合には C, C++ 言語が使われるのが普通です。

しかし、 C, C++ 言語は実装が大変です。 手軽に作りたいといえば、スクリプト言語です。
スクリプト言語の中では比較的 **Python** が科学技術計算に強いと言われています。

Julia:Fortranに迫る高速性

High-Performance JIT Compiler

Julia's LLVM-based just-in-time (JIT) compiler combined with the language's design allow it to approach and often match the performance of C. To get a sense of relative performance of Julia compared to other languages that can or could be used for numerical and scientific computing, we've written a small set of micro-benchmarks in a variety of languages: C, Fortran, Julia, Python, Matlab/Octave, R, JavaScript, Java, Lua, Go, and Mathematica. We encourage you to skim the code to get a sense for how easy or difficult numerical programming in each language is. The following micro-benchmark results were obtained on a single core (serial execution) on an Intel(R) Xeon(R) CPU E7-8850 2.00GHz CPU with 1TB of 1067MHz DDR3 RAM, running Linux:

	Fortran gcc 5.1.1	Julia 0.4.0	Python 3.4.3	R 3.2.2	Matlab R2015b	Octave 4.0.0	Mathe-matica 10.2.0	JavaScript V8 3.28.71.19	Go go1.5	LuaJIT gsl-shell 2.3.1	Java 1.8.0_45
fib	0.70	2.11	77.76	533.52	26.89	9324.35	118.53	3.36	1.86	1.71	1.21
parse_int	5.05	1.45	17.02	45.73	802.52	9581.44	15.02	6.06	1.20	5.77	3.35
quicksort	1.31	1.15	32.89	264.54	4.92	1866.01	43.23	2.70	1.29	2.03	2.60
mandel	0.81	0.79	15.32	53.16	7.58	451.81	5.13	0.66	1.11	0.67	1.35
pi_sum	1.00	1.00	21.99	9.56	1.00	299.31	1.69	1.01	1.00	1.00	1.00
rand_mat_stat	1.45	1.66	17.93	14.56	14.52	30.93	5.95	2.30	2.96	3.27	3.92
rand_mat_mul	3.48	1.02	1.14	1.57	1.12	1.12	1.30	15.07	1.42	1.16	2.36

Figure: benchmark times relative to C (smaller is better, C performance = 1.0).

Pythonから異議：使われたPythonコードは最適化されてない！

Pythonはもっと早くできる！

原典

https://www.ibm.com/developerworks/community/blogs/jfp/entry/Python_Meets_Julia_Micro_Performance?lang=en

Let's summarize in a table where we are. We display the speedup we get between the original Python code and the optimized one. We also display the tools we used for each of the benchmark example used by the Julia team.

Time in micro seconds	Julia	Python Optimized	Python Original	Julia / Python Optimized	Numpy	Numba	Cython
Fibonacci 64 bits	80	24	NA	3.8			X
Fib BigInt	12,717	1,470	3,770	8.7			
quicksort	419	306	17,700	1.4	X		X
Mandelbrot	196	126	6,570	1.6	X	X	
pisum	34,783	20,400	926,000	1.7		X	
randmatmul	95,975	83,7000	83,700	1.1	X		
parse int	244	472	3,290	0.5	X		X
randmatstat	14,544	83,200	160,000	0.2	X		

Pythonを高速で走らせるには

Let me conclude with a list of interesting articles discussing the tools I used and more:

- How to optimize for speed A short optimization guide by scipy team. It also discusses memory profiling.
- A guide to analyzing Python performance. Short intro to various profilers.
- Numba vs. Cython: Take 2, Understanding the FFT Algorithm, and Optimizing Python in the Real World: NumPy, Numba, and the NUFFT . Three interesting posts from Jake Vanderplas. In the latter, he shows how Python plus Numba can yield code only 30% slower than highly optimized Fortran code.
- Enhancing Performance in pandas documentation. A useful guide on how to make pandas code faster.
- Faster code via static typing and Using Cython with NumPy in Cython documentation.
- Numba vs Cython: How to Choose . Title says it all.
- Python Is Not C: Take Two.

いろいろな Jupyter Notebookの例

- Git/Githubについて : KobeLectureJun2016GitGithub.ipynb
  出典 CodemapとTechacademyのブログとウェブから
- JupyterNotebook入門: KobeLectureJun2016Jupyter.ipynb
  出典 Ipythonのホーム、Cookpad有賀さん、KEK ObinaさんのWeb
- 顔認識を試みる: KobeLectureJun2016FaceDetection.ipynb
  出典　中井悦司さんのGithub（enakai00/jupyter_gcp）から
- 量子力学入門: KobeLectureJun2016FaceDetection.ipynb
  出典　Univ College LondonのDr.Janina Dewitzの講義録から
- ベイズ推定講義録: KobeLectureNakaiBeyesInference.ipynb
  出典　中井悦司さんのGithub/enakai00/jupyter_ml4se から
- LIGOが検出した重力波の解析: KobeLectureJun2016GW150914.ipynb
  出典 https://losc.ligo.org/s/events/GW150914/GW150914_tutorial.html

Jupyter Notebookの限界

Kirill Pomogajkonoポスティングと、それに対する議論

JupyterNotebookで、本格的なプログラム開発をするのは困難:

- It messes with your version control
- Code can only be run in chunks
- It's difficult to keep track
- Code often ends up very fragmented
- The output is incomplete
- Potential security risks?

しかし、新しい言語を学ぶときには、Jupyter Notebookが便利

やり残したこと

- Dockerを活用する　<=　ビッグデータ
- リモートでJupyter　Notebookを使う　<=　講義や演習
- Python以外の言語

付録：JupyterNotebookの画面例:Git/GitHub

Jupyter Notebook の紹介

2016年6月 釜江常好

テーマ：公開ソフトを利用するための初歩的な知識

-公開されたWebより-

プログラムのバージョン管理は、初心者にはわかりにくいものかもしれません。とは言え、公開されたプログラムをベースにデータ解析を進める、私たち科学者、技術者には、それらをダウンロードすることから仕事が始まります。従って、GitやGitHubは、我々になくてはならないツールであり、GitやGitHubについて知っておくことが、重要となります。本講義録を書いている私も、バージョン管理ツールのGitの仕組みや使い方を十分に理解していません。上のブログに出ている説明をコピーして、説明します。その後で、GitHubについての説明に移ります。こちらも、同じブログを参考にしました。

- [Git、Github説明1：https://blog.codemap.jp/git-github]
- [Git、Github説明2：http://techacademy.jp/magazine/6235]

Git とは

Git（ギット）とは、バージョン管理を行うためのツールのことです。複数人でプロ グラミングを行う場合、ソースコードを効率的に管理・運用する必要があります。例えば、「誰がどのファイルのどの部分を修正したのか？」や「リリース予定の機能追加で更新するのは、どれとどれか？」といったことです。Gitはこのような管理を行うためにつくられたツールで、システム開発の現場で使われているツールの一つです。公開ソフトをダウンロードして使う我々は、詳細を知る必要はありません。しかし、Gitを使った開発の流れを理解しておく必要があります。一般的な開発の流れは、以下のようになります。まずはじめに、開発プロジェクトのリーダが、Gitプロジェクトを立ち上げます。その開発に協力する人たちは、この大元のGitリポジトリ（共有リポジトリ）を、自分のリポジトリ（ローカルリポジトリ）にコピーします。この作業はクローンと呼ばれます。コマンドは以下のようになります。

```
$git clone git://github.com/schacon/grit.git
```

プログラムの開発は、クローンしたローカルリポジトリで行います。協力者は、仕事をを自分のパソコンで進めて行きます。キリの良いところまで開発が進んだら、ソースの保存をします。Gitではこれをコミットと呼んでいます。そして、ローカルリポジトリでの開発が完了したら、共有リポジトリにソースコードを送信して反映します。これをプッシュと呼びます。開発では、ここまでのフローを何度も繰り返して行くことになります。ローカルで行った変更はコミットしますが、その前にインデックスに登録しておく必要があります。下記のコマンドを実行して、該当するファイルを登録します。

$git add *.py

$git add README

そして以下のコマンドでコミットします。

$git commit -m "コメントを入力します"

Githubとは ¶

GitHub(ギットハブ)は、Gitの仕組みを利用して、世界中の人々が自分の作品(プログラムコードやデザインデータなど)を保存、公開することができるようにしたウェブサービスの名称です。この仕組みはGitHub社という会社によって運営されており、個人・企業問わず無料で利用することができます。GitHubに作成されたレポジトリは、基本的にすべて公開されますが、有料サービスを利用すると、指定したユーザーからしかアクセスができないプライベートなレポジトリを作ったりすることができます。複数人での開発をサポートしてくれる便利な機能が備わっていて、現在ではエンジニアにとって欠かせないツールの一つになっています。私はコード開発のプロジェクトをリードした経験がありません。しかしGithubで公開されているコードを、ダウンロードして使っています。Jupyter Notebookのサンプルも、多くの場合、Githubから取っています。

Githubの使い方の一例を、以下に示します。ここではRasberryPiで、Jupyter Notebook Serverを動かすのに必要なコードが、すべてダウンロードできるようになっています。複雑な手続きですが、順を追ってインストールできるように、丁寧に説明されています。

In [1]:
```
import webbrowser

# generate an URL
url = 'https://' + 'github.com/kleinee/jns'
webbrowser.open(url)
```

Out[1]: True

もしコード開発プロジェクトに参加したり、新しいプロジェクトを立ち上げたい場合には、Githubのアカウントをつくる必要があります。下記のGitHub公式ページより、すぐにアカウントが作成できます。

In [2]:
```
# generate an URL
url = 'https://' + 'github.com/'
webbrowser.open(url)
```

Out[2]: True

GitHubでの開発をはじめるには、新規にリポジトリを作成します。ここでは新規にリポジトリを作成してみましょう。GitHubにログインしたら、右下の「+ New repository」ボタンをクリックしてみましょう。リポジトリの内容を入力して、「Create repository」ボタンをクリック。これで新規リポジトリが作成できました。そしてリモートリポジトリをクローンするには、作成したリポジトリのURLをコピーして、下記のコマンドを実行します。

$git clone git://github.com/xxx/yyy.git

これでリモートリポジトリがコピーされ、ローカルリポジトリが作成されます。そして開発が完了したらリモートリポジトリに、以下のようにプッシュします。

$git push origin master

Githubリポジトリから直接Jupyter Notebookを動かすことが出来る

Githubのリポジトリから直接Jupyter Notebookを動かすことが出来るようです。これには「binder」と呼ばれるソフトをインストールしなければならないようです。しかしユーザは、何もする必要がないので、大学の講義などでは、使い易いでしょうね。誰か、試してみませんか。

url = 'http://' + 'mybinder.org' webbrowser.open(url)

宿題:

- 著者は経験がないのですが、皆さんで、グループを作って、Jupyter Notebookの講義録を集めて、Githublにプッシュして見ませんか。
- そうすると、Githubのリポジトリから直接Jupyter Notebookを動かせます。「binder」をインストールして、試してください。

付録：JupyterNotebookとは

Jupyter Notebook の紹介

2016年6月 釜江常好

テーマ：IPythonとJupyter Notebookについての概説

-公開されたurlより-

人文系、理工医農系の研究者、あらゆるICT関係の技術者は、数値計算やデータ処理に、持てる知力と時間の大きな部分を費やしています。これらの仕事に使える優れた計算機コードを探し出し、その効率よい利用方法を短時間で習得できればと願わない人は居ないと思います。それを手助けしてくれるのが、近年発達してきた「対話型の実行環境」です。一般的に「Notebook」と呼ばれています。これらは、説明文とコードがデータ処理の流れに沿ってまとめられ、ブラウザ上で、1ステップ毎に、結果を確かめながら実行できるのが、特長です。無料で公開された「Notebook」の最初に一つが、IPython notebookです。IPythonは、理工学の世界で広く使われてきたスクリプト言語、Pythonを、対話型に実行するための便利なシェル(Shell)です。タブ補完やユニックスのシェルコマンドなどが使えます。そのシェルをブラウザ上に順序良く並べて実行できるようにしたのが、IPython Notebookです。コードの中に残すコメント記述を、マーク付きで書け、Texフォーマットも理解します。NotebookはXMLフォーマットで、「ipynb」の拡張子で保存されます。そして、pdf、htmlなど、いろいろな形式に書き出すことが可能です。

- [IPythonのホーム：http://ipython.org/]
- [有賀 IPython Notebook：http://techlife.cookpad.com/entry/write-once-share-anywhare]
- [KEK Obina IPython Notebook紹介：http://cerldev.kek.jp/trac/EpicsUsersJP/raw-attachment/wiki/misc/ipynote/ipynote_cERL_pub.pdf]

IPython Notebook/Jupyter Notebookとは:

IPythonlには、古典的なRead-Evaluate-PrintのLoopと違って、以下のような実装方法が取られています。

- Evaluateを独立したプロセス(kernel)として実装
- 上の結果、IPythonlは、ターミナル、Qt console、Notebookなどのクライアント環境で実行可能になります
- また、IPython Notebookでは、Python、Ruby、R、Juliaなど、多くの言語が使えます
- ベースとなっているソフトは、IPythonの他、Tornado(web server)、jQuery、MathJaxなどである
- IPythonなどのkernelが起動されると、デフォルトでは、ポートlocalhost:8888でlistenする
- 並列化も可能である

有賀さんのブログの動画でJupyter Notebookを紹介する

]:
```
import webbrowser

# generate an URL
url = 'http://' + 'techlife.cookpad.com/entry/write-once-share-anywhare'
webbrowser.open(url)
```

Jupyter notebookの使い方を簡単にまとめて見ます

- セル(cell)と呼ばれる入力部分は、プルダウンメニューで、文章を入れる「Markdown」とプログラムの一部を入れる「Code」を選べる
- どちらのセルでも、編集することが可能である
- コードはShift+Enterで実行可能で、何度も修正して再実行することが可能
- 保存をしたければ、ボタンをクリックするか、Ctrl+sを入力する
- Matplotlibをimportすることで、多種多様なグラフを描くことができる
- 図や表を、inlineで示すことができる

宿題:

著者は経験がないのですが、皆さんで、グループを作って、Jupyter Notebookの講義録を集めて、Githublにプッシュして見ませんか。

付録：Face Detection

テーマ：Google Cloud Vision API を使った顔認識を試してみる

-中井悦司さんがGithubに出しているjupyter_gcpを元に作成-

このノートブックは、下記のチュートリアルの内容を噛み砕いて解説したものです。

- Label Detection Tutorial
- Face Detection Tutorial

Google Cloud Vision APIを利用するために、事前にGCP (Google Cloud Platform) のユーザー登録をしておいてください。(このノートブックの内容は、60日間の$300無償利用枠で試すことが可能です。)

必要な事前準備

GCPのAPI Managerで、Cloud Vision APIを有効化した後、「認証情報」のメニューからサービスアカウント「Compute Engine default service account」のアカウントキーをJSON形式でダウンロードします。ダウンロードしたファイルを、ファイル名を「Vision API Project-5912afb0bd62.json」と変更して、Jupyterを立ち上げるフォルダーに置いておくか、Jupyterの画面の右上に出るアップロードのボタンを押して、アップロードしておきます。

ライブラリーモジュールをインポートします。

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import base64
import httplib2
import os
```

```
from PIL import Image
from PIL import ImageDraw

from googleapiclient import discovery
from oauth2client.client import GoogleCredentials
```

先ほどアップロードしたアカウントキーファイルを利用して、APIサービスのハンドラーオブジェクトを取得します。

In [2]:
```
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = 'Vision API Project-5912afb0bd62.json'
DISCOVERY_URL='https://{api}.googleapis.com/$discovery/rest?version={apiVersion}'
credentials = GoogleCredentials.get_application_default()
service = discovery.build('vision', 'v1', credentials=credentials,
                          discoveryServiceUrl=DISCOVERY_URL)
```

分析に使用する画像ファイルをWebからダウンロードします。

ここでは、一般のWebページで公開されている画像を直接ダウンロードしています。

画像の引用元はこちらです。http://www.lawson.co.jp/campaign/akb/

In [3]:
```
!curl http://www.lawson.co.jp/campaign/akb/img/mv.png > sample01.png
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
 84  417k   84  352k    0     0  1129k      0 --:--:-- --:--:-- --:--:-- 1190k
100  417k  100  417k    0     0  1337k      0 --:--:-- --:--:-- --:--:-- 1410k
```

ダウンロードした画像を確認します。

In [4]:
```
Image.open("sample01.png")
```

Out[4]:

In [5]:
```
Image.open("ButsuriA.jpg")
```

Out[5]:

画像のラベリング、あるいはタグまたは分類

画像の内容を判別してラベルあるいはタグを取得する関数を用意します。

In [6]:
```
def get_labels(photo_file):
    with open(photo_file, 'rb') as image:
        image_content = base64.b64encode(image.read())
        service_request = service.images().annotate(body={
            'requests': [{
                'image': {
                    'content': image_content.decode('UTF-8')
                },
                'features': [{
                    'type': 'LABEL_DETECTION',
                    'maxResults': 5
                }]
            }]
        })
        response = service_request.execute()
        return response['responses'][0]['labelAnnotations']
```

先ほどアップロードした画像のラベルを取得します。ラベルあるいは分類の候補と確信度(%)を最大5つ表示します。

In [7]:
```
labels = get_labels('sample01.png')
for label in labels:
    print ('%s : %2d%%' % (label['description'], label['score']*100))
```

```
person : 94%
team : 80%
class : 78%
profession : 72%
```

人物(person)の画像であり、チーム(team)やクラス(class)の集合写真らしいと言い当てています。また、同じ服装をしているので、仕事姿(profession)と判断しています。

In [8]:
```
labels = get_labels("ButsuriA.jpg")
for label in labels:
    print ('%s : %2d%%' % (label['description'], label['score']*100))
```

```
social group : 98%
person : 94%
people : 87%
class : 78%
community : 68%
```

人物(person)の画像であると認識しています。そして制服でないので、ご近所(community)の写真と判断しています。

顔の部分を認識するサンプル

顔認識情報を取得する関数を用意します。

In [9]:
```
def get_faces(photo_file):
    with open(photo_file, 'rb') as image:
        image_content = base64.b64encode(image.read())
        service_request = service.images().annotate(body={
            'requests': [{
                'image': {
                    'content':  image_content.decode('UTF-8')
                },
                'features': [{
                    'type': 'FACE_DETECTION',
                    'maxResults': 100
                }]
            }]
        })
        response = service_request.execute()
        return response['responses'][0]['faceAnnotations']
```

さらに、取得した情報を用いて、顔の部分に枠を描く関数を用意します。joyLikelihood(笑顔)がVERY_LIKELY(非常に確からしい)ものについては、ピンク、その他は、グリーンの枠を描きます。

In [10]:
```
def highlight_faces(photo_file, responce):
    im = Image.open(photo_file)
    draw = ImageDraw.Draw(im)
    for face in faces:
        if face['joyLikelihood'] == 'VERY_LIKELY':
            color = '#ff8888'
        else:
            color = '#00ff00'
        box = [(v.get('x', 0.0), v.get('y', 0.0)) for v in face['fdBoundingPoly']['vertices']]
        draw.line(box + [box[0]], width=5, fill=color)
    return im
```

サンプル画像で試してみます。

In [11]:
```
faces = get_faces('sample01.png')
highlight_faces('sample01.png', faces)
```

Out[11]:

歯が見えていると笑顔だと認識されやすいようです。一人だけ認識に失敗していますが、この部分だけを切り出した場合なども調べると面白いかもしれません。

In [12]:
```
faces = get_faces("ButsuriA.jpg")
highlight_faces('ButsuriA.jpg', faces)
```

Out[12]:

この例では全員、認識に成功しています。一人だけ笑顔と認識されています。おそらく歯が見えているからでしょう。

宿題:

著者はよく理解していないのですが、Google Cloud Vision APIでは、サンプル写真を登録し、集合写真や景色の中から、登録した人物を探せるはずです。やってみませんか？

付録：LIGO重力波解析

テーマ：LIGOが検出した重力波信号の解析

-https://losc.ligo.org/s/events/GW150914/GW150914_tutorial.html-

Welcome! This ipython notebook (or associated python script GW150914_tutorial.py) will go through some typical signal processing tasks on strain time-series data associated with the LIGO GW150914 data release from the LIGO Open Science Center (LOSC):

- https://losc.ligo.org/events/GW150914/
- View the tutorial as a web page - https://losc.ligo.org/s/events/GW150914/GW150914_tutorial.html/
- Download the tutorial as a python script - https://losc.ligo.org/s/events/GW150914/GW150914_tutorial.py/
- Download the tutorial as iPython Notebook - https://losc.ligo.org/s/events/GW150914/GW150914_tutorial.ipynb/

To begin, download the ipython notebook, readligo.py, and the data files listed below, into a directory / folder, then run it. Or you can run the python script GW150914_tutorial.py. You will need the python packages: numpy, scipy, matplotlib, h5py.

On Windows, or if you prefer, you can use a python development environment such as Anaconda (https://www.continuum.io/why-anaconda) or Enthought Canopy (https://www.enthought.com/products/canopy/).

Questions, comments, suggestions, corrections, etc: email losc@ligo.caltech.edu

v20160208b

Intro to signal processing

This tutorial assumes that you know python well enough.

If you know how to use "ipython notebook", use the GW150914_tutorial.ipynb file. Else, you can use the GW150914_tutorial.py script.

This tutorial assumes that you know a bit about signal processing of digital time series data (or want to learn!). This includes power spectral densities, spectrograms, digital filtering, whitening, audio manipulation. This is a vast and complex set of topics, but we will cover many of the basics in this tutorial.

If you are a beginner, here are some resources from the web:

- http://101science.com/dsp.htm
- https://georgemdallas.wordpress.com/2014/05/14/wavelets-4-dummies-signal-processing-fourier-transforms-and-heisenberg/

Download the data

- Download the data files from LOSC:
- We will use the hdf5 files, both H1 and L1, with durations of 32 and 4096 seconds around GW150914, sampled at 16384 and 4096 Hz :
  - https://losc.ligo.org/s/events/GW150914/H-H1_LOSC_4_V1-1126259446-32.hdf5
  - https://losc.ligo.org/s/events/GW150914/L-L1_LOSC_4_V1-1126259446-32.hdf5
  - https://losc.ligo.org/s/events/GW150914/H-H1_LOSC_16_V1-1126259446-32.hdf5
  - https://losc.ligo.org/s/events/GW150914/L-L1_LOSC_16_V1-1126259446-32.hdf5
  - https://losc.ligo.org/s/events/GW150914/GW150914_4_NR_waveform.txt
- Download the python functions to read the data: https://losc.ligo.org/s/sample_code/readligo.py
- From a unix/mac-osx command line, you can use wget; for example,
  - wget https://losc.ligo.org/s/events/GW150914/H-H1_LOSC_4_V1-1126257414-4096.hdf5
- Put these files in your current directory / folder. Don't mix any other LOSC data files in this directory, or readligo.py may get confused.

Here,

- "H-H1" means that the data come from the LIGO Hanford Observatory site and the LIGO "H1" datector;
- the "*4*" means the strain time-series data are (down-)sampled from 16384 Hz to 4096 Hz;
- the "V1" means version 1 of this data release;
- "1126257414-4096" means the data starts at GPS time 1126257414 (Mon Sep 14 09:16:37 GMT 2015), duration 4096 seconds;
  - NOTE: GPS time is number of seconds since Jan 6, 1980 GMT. See http://www.oc.nps.edu/oc2902w/gps/timsys.html or https://losc.ligo.org/gps/
- the filetype "hdf5" means the data are in hdf5 format: https://www.hdfgroup.org/HDF5/

Note that the the 4096 second long files at 16384 Hz sampling rate are fairly big files (125 MB). You won't need them for this tutorial:

- https://losc.ligo.org/s/events/GW150914/H-H1_LOSC_4_V1-1126257414-4096.hdf5
- https://losc.ligo.org/s/events/GW150914/L-L1_LOSC_4_V1-1126257414-4096.hdf5
- https://losc.ligo.org/s/events/GW150914/H-H1_LOSC_16_V1-1126257414-4096.hdf5
- https://losc.ligo.org/s/events/GW150914/L-L1_LOSC_16_V1-1126257414-4096.hdf5

In [3]:
```
# Standard python numerical analysis imports:
import numpy as np
from scipy import signal
from scipy.interpolate import interp1d
from scipy.signal import butter, filtfilt, iirdesign, zpk2tf, freqz

# the ipython magic below must be commented out in the .py file, since it doesn't work.
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import h5py

# LIGO-specific readligo.py
import readligo as rl
```

NOTE that in general, LIGO strain time series data has gaps (filled with NaNs) when the detectors are not taking valid ("science quality") data. Analyzing these data requires the user to loop over "segments" of valid data stretches. In https://losc.ligo.org/segments/ we provide example code to do this.

However, the 4096 seconds of released data around GW150914 is one unbroken segment, with no gaps. So for now, we will read it all in and treat it as one valid data segment, ignoring the extra complexity mentioned above.

This won't work for other LOSC data releases! See https://losc.ligo.org/segments/ for a more general way to find valid data segments in LOSC data.

In [4]:
```
#-----------------------------------------------------------------
# Load LIGO data from a single file
#-----------------------------------------------------------------
# First from H1
fn_H1 = 'H-H1_LOSC_4_V1-1126259446-32.hdf5'
strain_H1, time_H1, chan_dict_H1 = rl.loaddata(fn_H1, 'H1')
# and then from L1
fn_L1 = 'L-L1_LOSC_4_V1-1126259446-32.hdf5'
strain_L1, time_L1, chan_dict_L1 = rl.loaddata(fn_L1, 'L1')

# sampling rate:
fs = 4096
# both H1 and L1 will have the same time vector, so:
time = time_H1
# the time sample interval (uniformly sampled!)
dt = time[1] - time[0]
```

Adding a numerical relativity template

Now let's also read in a theoretical (numerical relativity) template, generated with parameters favored by the output from the GW150914 parameter estimation (see the GW150914 detection paper, https://dcc.ligo.org/P150914/public).

This NR template corresponds to the signal expected from a pair of black holes with masses of around 36 and 29 solar masses, merging into a single black hole of 62 solar masses, at a distance of around 410 Mpc.

You can fetch the template time series from the following URL, and put it in your working directory / folder:

- https://losc.ligo.org/s/events/GW150914/GW150914_4_NR_waveform.txt

In [5]:
```
np.genfromtxt('GW150914_4_NR_waveform.txt')
```

Out[5]:
```
array([[ -6.20404109e-01,  -1.86912777e-22],
       [ -6.20159968e-01,  -1.77827879e-22],
       [ -6.19915828e-01,  -1.68582673e-22],
       ...,
       [  5.48888598e-02,   3.58407297e-26],
       [  5.51330004e-02,   3.87926155e-26],
       [  5.53771410e-02,   4.05305739e-26]])
```

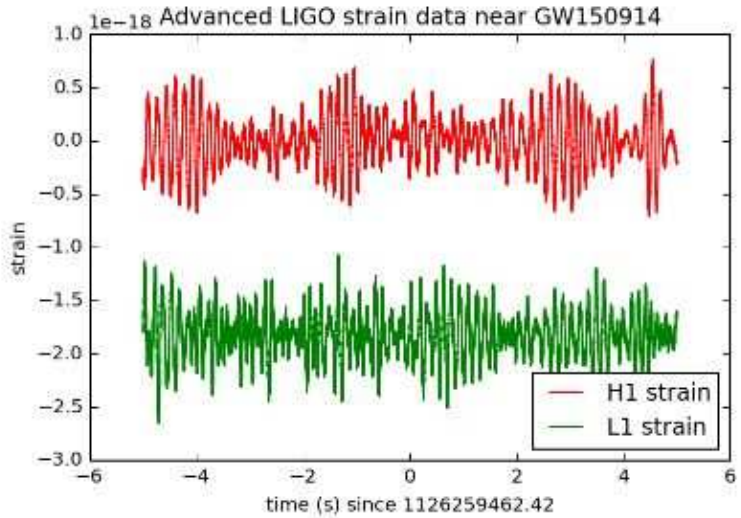
In [6]:
```
# read in the NR template
NRtime, NR_H1 = np.genfromtxt('GW150914_4_NR_waveform.txt').transpose()
```

First look at the data from H1 and L1

In [7]:
```
# First, let's look at the data and print out some stuff:

# this doesn't seem to work for scientific notation:
# np.set_printoptions(precision=4)

print '  time_H1: len, min, mean, max = ', \
   len(time_H1), time_H1.min(), time_H1.mean(), time_H1.max()
print 'strain_H1: len, min, mean, max = ', \
   len(strain_H1), strain_H1.min(),strain_H1.mean(),strain_H1.max()
print 'strain_L1: len, min, mean, max = ', \
   len(strain_L1), strain_L1.min(),strain_L1.mean(),strain_L1.max()

#What's in chan_dict? See https://losc.ligo.org/archive/dataset/GW150914/
bits = chan_dict_H1['DATA']
print 'H1     DATA: len, min, mean, max = ', len(bits), bits.min(),bits.mean(),bits.max()
bits = chan_dict_H1['CBC_CAT1']
print 'H1 CBC_CAT1: len, min, mean, max = ', len(bits), bits.min(),bits.mean(),bits.max()
bits = chan_dict_H1['CBC_CAT2']
print 'H1 CBC_CAT2: len, min, mean, max = ', len(bits), bits.min(),bits.mean(),bits.max()
bits = chan_dict_L1['DATA']
print 'L1     DATA: len, min, mean, max = ', len(bits), bits.min(),bits.mean(),bits.max()
bits = chan_dict_L1['CBC_CAT1']
print 'L1 CBC_CAT1: len, min, mean, max = ', len(bits), bits.min(),bits.mean(),bits.max()
bits = chan_dict_L1['CBC_CAT2']
print 'L1 CBC_CAT2: len, min, mean, max = ', len(bits), bits.min(),bits.mean(),bits.max()
print 'In both H1 and L1, all 32 seconds of data are present (DATA=1), '
print "and all pass data quality (CBC_CAT1=1 and CBC_CAT2=1)."
```

```
  time_H1: len, min, mean, max =  131072 1126259446.0 1126259462.0 1126259478.0
strain_H1: len, min, mean, max =  131072 -7.11996338709e-19 8.73279794057e-23 7.71483633765e-
19
strain_L1: len, min, mean, max =  131072 -2.6788089173e-18 -1.82870749189e-18 -7.69266177024e
-19
H1     DATA: len, min, mean, max =  32 1 1.0 1
H1 CBC_CAT1: len, min, mean, max =  32 1 1.0 1
```

In [8]:
```
# plot +- 5 seconds around the event:
tevent = 1126259462.422           # Mon Sep 14 09:50:45 GMT 2015
deltat = 5.                       # seconds around the event
# index into the strain time series for this time interval:
indxt = np.where((time_H1 >= tevent-deltat) & (time_H1 < tevent+deltat))

plt.figure()
plt.plot(time_H1[indxt]-tevent,strain_H1[indxt],'r',label='H1 strain')
plt.plot(time_L1[indxt]-tevent,strain_L1[indxt],'g',label='L1 strain')
plt.xlabel('time (s) since '+str(tevent))
plt.ylabel('strain')
plt.legend(loc='lower right')
plt.title('Advanced LIGO strain data near GW150914')
plt.savefig('GW150914_strain.png')
```



The data are dominated by **low frequency noise**; there is no way to see a signal here, without some signal processing.

There are very low frequency oscillations that are putting the mean of the L1 strain at -2.0e-18 at the time around this event, so it appears offset from the H1 strain. These low frequency oscillations are essentially ignored in LIGO data analysis (see bandpassing, below).

We will be "whitening" the data, below.

Data in the Fourier domain - ASDs

Plotting these data in the Fourier domain gives us an idea of the frequency content of the data. A way to visualize the frequency content of the data is to plot the amplitude spectral density, ASD.

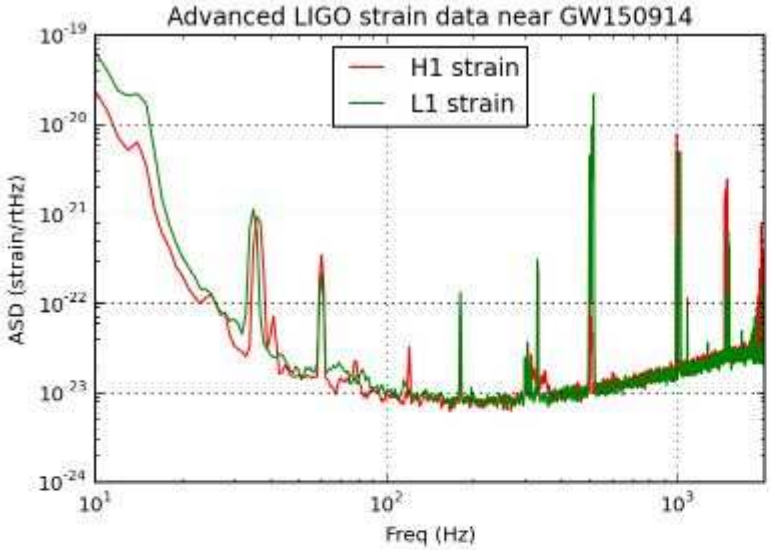
The ASDs are the square root of the power spectral densities (PSDs), which are averages of the square of the fast fourier transforms (FFTs) of the data.

They are an estimate of the "strain-equivalent noise" of the detectors versus frequency, which limit the ability of the detectors to identify GW signals.

They are in units of strain/rt(Hz). So, if you want to know the root-mean-square (rms) strain noise in a frequency band, integrate (sum) the squares of the ASD over that band, then take the square-root.

There's a signal in these data! For the moment, let's ignore that, and assume it's all noise.

In [9]:
```
# number of sample for the fast fourier transform:
NFFT = 1*fs
fmin = 10
fmax = 2000
Pxx_H1, freqs = mlab.psd(strain_H1, Fs = fs, NFFT = NFFT)
Pxx_L1, freqs = mlab.psd(strain_L1, Fs = fs, NFFT = NFFT)

# We will use interpolations of the ASDs computed above for whitening:
psd_H1 = interp1d(freqs, Pxx_H1)
psd_L1 = interp1d(freqs, Pxx_L1)

# plot the ASDs:
plt.figure()
plt.loglog(freqs, np.sqrt(Pxx_H1),'r',label='H1 strain')
plt.loglog(freqs, np.sqrt(Pxx_L1),'g',label='L1 strain')
plt.axis([fmin, fmax, 1e-24, 1e-19])
plt.grid('on')
plt.ylabel('ASD (strain/rtHz)')
plt.xlabel('Freq (Hz)')
plt.legend(loc='upper center')
plt.title('Advanced LIGO strain data near GW150914')
plt.savefig('GW150914_ASDs.png')
```



NOTE that we only plot the data between fmin = 10 Hz and fmax = 2000 Hz.

Below fmin, the data **are not properly calibrated**. That's OK, because the noise is so high below fmin that LIGO cannot sense gravitational wave strain from astrophysical sources in that band.

The sample rate is fs = 4096 Hz (2^12 Hz), so the data cannot capture frequency content above the Nyquist frequency = fs/2 = 2048 Hz. That's OK, because GW150914 only has detectable frequency content in the range 20 Hz - 300 Hz.

You can see strong spectral lines in the data; they are all of instrumental origin. Some are engineered into the detectors (mirror suspension resonances at ~500 Hz and harmonics, calibration lines, control dither lines, etc) and some (60 Hz and harmonics) are unwanted. We'll return to these, later.

You can't see the signal in this plot, since it is relatively weak and less than a second long, while this plot averages over 32 seconds of data. So this plot is entirely dominated by instrumental noise.

Later on in this tutorial, we'll look at the data sampled at the full 16384 Hz (2^14 Hz).

Whitening

From the ASD above, we can see that the data are very strongly "colored" - noise fluctuations are much larger at low and high frequencies and near spectral lines, reaching a roughly flat ("white") minimum in the band around 80 to 300 Hz.

We can "whiten" the data (dividing it by the noise amplitude spectrum, in the fourier domain), suppressing the extra noise at low frequencies and at the spectral lines, to better see the weak signals in the most sensitive band.

Whitening is always one of the first steps in astrophysical data analysis (searches, parameter estimation). Whitening requires no prior knowledge of spectral lines, etc; only the data are needed.

The resulting time series is no longer in units of strain; now in units of "sigmas" away from the mean.

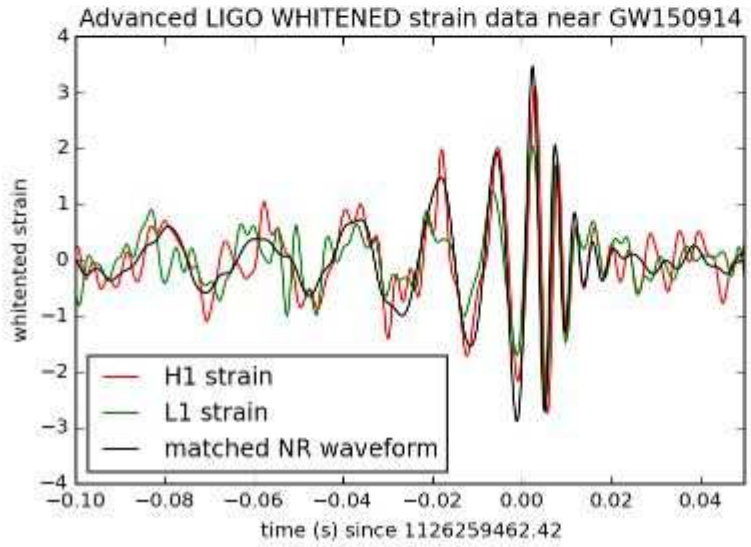
In [10]:
```
# function to writen data
def whiten(strain, interp_psd, dt):
    Nt = len(strain)
    freqs = np.fft.rfftfreq(Nt, dt)

    # whitening: transform to freq domain, divide by asd, then transform back,
    # taking care to get normalization right.
    hf = np.fft.rfft(strain)
    white_hf = hf / (np.sqrt(interp_psd(freqs) /dt/2.))
    white_ht = np.fft.irfft(white_hf, n=Nt)
    return white_ht

# now whiten the data from H1 and L1, and also the NR template:
strain_H1_whiten = whiten(strain_H1,psd_H1,dt)
strain_L1_whiten = whiten(strain_L1,psd_L1,dt)
NR_H1_whiten = whiten(NR_H1,psd_H1,dt)
```

Now plot the whitened strain data, along with the best-fit numerical relativity (NR) template.

To get rid of remaining high frequency noise, we will also bandpass the data (see bandpassing, below).

In [11]:

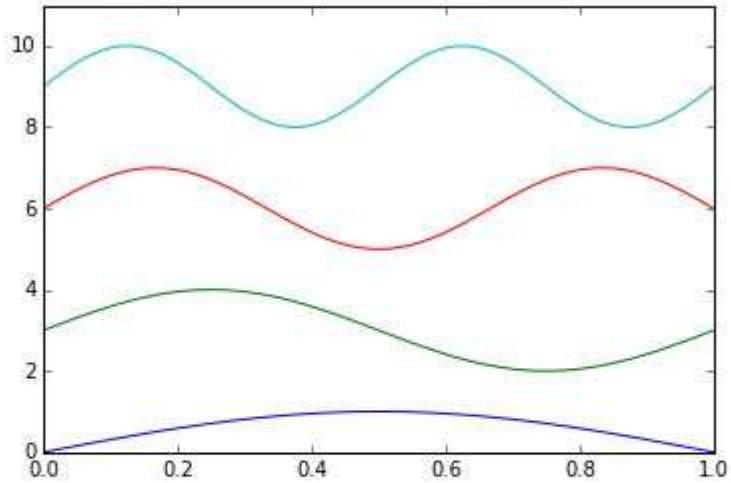
```
# We need to suppress the high frequencies with some bandpassing:
bb, ab = butter(4, [20.*2./fs, 300.*2./fs], btype='band')
strain_H1_whitenbp = filtfilt(bb, ab, strain_H1_whiten)
strain_L1_whitenbp = filtfilt(bb, ab, strain_L1_whiten)
NR_H1_whitenbp = filtfilt(bb, ab, NR_H1_whiten)

# plot the data after whitening:
# first, shift L1 by 7 ms, and invert. See the GW150914 detection paper for why!
strain_L1_shift = -np.roll(strain_L1_whitenbp,int(0.007*fs))

plt.figure()
plt.plot(time-tevent,strain_H1_whitenbp,'r',label='H1 strain')
plt.plot(time-tevent,strain_L1_shift,'g',label='L1 strain')
plt.plot(NRtime+0.002,NR_H1_whitenbp,'k',label='matched NR waveform')
plt.xlim([-0.1,0.05])
plt.ylim([-4,4])
plt.xlabel('time (s) since '+str(tevent))
plt.ylabel('whitented strain')
plt.legend(loc='lower left')
plt.title('Advanced LIGO WHITENED strain data near GW150914')
plt.savefig('GW150914_strain_whitened.png')
```



付録：UCL講義録-量子力学

In [1]:
```
# Import libraries and set up in-line plotting.
%matplotlib inline
import matplotlib.pyplot as pl
import numpy as np

# Define the eigenbasis - normalisation needed elsewhere
def eigenbasis_sw(n,width,norm,x):
    """The eigenbasis for a square well, running from 0 to a (width), sin(n pi x/a).
    N.B. requires a normalisation factor, norm."""
    fac = np.pi*n/width
    return norm*np.sin(fac*x)

# We will also define the second derivative for kinetic energy (KE)
def d2eigenbasis_sw(n,width,norm,x):
    """The second derivative of the eigenbasis for a square well, running from 0 to a, sin(n pi
    fac = np.pi*n/width
    return -fac*fac*norm*np.sin(fac*x)

# Define the x-axis
width = 1.0
num_x_points = 101
x = np.linspace(0.0,width,num_x_points)
dx = width/(num_x_points - 1)

# Integrate two functions over the width of the well
# NB this is a VERY simple integration routine: there are much better ways
def integrate_functions(f1,f2,size_x,dx):
    """Integrate two functions over defined x range with spacing dx"""
    sum = 0.0
    for i in range(size_x):
        sum = sum + f1[i]*f2[i]
    sum = sum*dx
    return sum


# Now plot the first few functions; we offset them vertically by adding 3(n-1)
# to make it clearer; remember that range(1,5) will give number 1 to 4. We don't
# normalise in this plot.
for m in range(1,5):
    pl.plot(x,eigenbasis_sw(m,width,1.0,x)+3*(m-1))
# Set y-axis so that we can see all the functions
pl.ylim((0,11))
```

Out[1]: (0, 11)



Do these make sense ? You should ask yourself if they match the boundary conditions, and if they fit with your expectations of the physical properties of the system.

We should now be confident that our function is correct (it's always worth checking this somehow, whether through plotting the output or a simple test). So we will create arrays to store the first ten basis states, properly normalised. I have chosen ten for no particular reason - we could use fewer or more states.

Matrix representation

What about the rest of the Hamiltonian matrix, i.e. the numbers $\langle \phi_n | \hat{H} | \phi_m \rangle$? We'll calculate these using two loops, and output the results. I will use two approaches: a formatted print written in the loop, and the internal numpy printing for arrays (though this requires some changes to the options to make it look nice).

In [4]:
```
Hmat = np.eye(num_basis)
print "Output of the Hamiltonian matrix elements as we calculate them"
# Loop over basis functions phi_n (the bra in the matrix element)
for n in range(num_basis):
    # Loop over basis functions phi_m (the ket in the matrix element)
    for m in range(num_basis):
        # Act with H on phi_m and store in H_phi_m
        H_phi_m = -0.5*d2basis_array[m]
        # Create matrix element by integrating
        H_mn = integrate_functions(basis_array[n],H_phi_m,num_x_points,dx)
        Hmat[m,n] = H_mn
        # The comma at the end prints without a new line; the %8.3f formats the number
        print "%8.3f" % H_mn,
    # This print puts in a new line when we have finished looping over m
    print

print
print "Now the numpy output with formatting"
print
# Another way to output is this, though it's commented out to avoid clutter
np.set_printoptions(precision=3,linewidth=100,suppress=True)
print Hmat
```

```
Output of the Hamiltonian matrix elements as we calculate them
   4.935   -0.000    0.000   -0.000    0.000    0.000    0.000   -0.000    0.000   -0.000
  -0.000   19.739    0.000   -0.000   -0.000   -0.000    0.000    0.000   -0.000   -0.000
   0.000    0.000   44.413    0.000    0.000    0.000   -0.000   -0.000    0.000    0.000
  -0.000   -0.000    0.000   78.957   -0.000   -0.000   -0.000   -0.000   -0.000   -0.000
   0.000   -0.000   -0.000    0.000  123.370    0.000    0.000    0.000   -0.000   -0.000
   0.000    0.000    0.000   -0.000    0.000  177.653    0.000   -0.000    0.000    0.000
   0.000    0.000   -0.000   -0.000    0.000   -0.000  241.805   -0.000    0.000   -0.000
  -0.000    0.000   -0.000   -0.000    0.000   -0.000   -0.000  315.827   -0.000    0.000
   0.000   -0.000    0.000   -0.000   -0.000    0.000    0.000   -0.000  399.719    0.000
  -0.000   -0.000    0.000   -0.000   -0.000    0.000   -0.000    0.000    0.000  493.480
```

In [5]:
```
# Define a coefficient array
coeff_array = np.zeros(num_basis)
coeff_array[0] = 1.0
coeff_array[2] = 2.0
coeff_array[3] = 1.0
# Prepare to make psi and its second derivative
psi = np.zeros(num_x_points)
d2psi = np.zeros(num_x_points)
for i in range(num_basis):
    psi = psi+coeff_array[i]*basis_array[i]
    d2psi = d2psi+coeff_array[i]*d2basis_array[i]
# Now we will normalise psi and apply the same factor to d2psi
integral = integrate_functions(psi,psi,num_x_points,dx)
psi = psi/np.sqrt(integral)
d2psi = d2psi/np.sqrt(integral)
# Plot psi
pl.plot(x,psi)
# Evaluate the energy from <psi|H|psi> - with no potential we just have
# the kinetic energy term. We set hbar = m = 1, so KE = -0.5 d^2 H/dx^2
Hpsi = -0.5*d2psi
print "Energy numerically is: ",integrate_functions(psi,Hpsi,num_x_points,dx)
# Find the energy from the sum over coefficient
energy_psi = 0.0
for i in range(num_basis):
    energy_psi = energy_psi + coeff_array[i]*coeff_array[i]*energy[i]/integral
print "Energy analytically is: ",energy_psi
```

```
Energy numerically is:  43.5907527715
Energy analytically is:  43.5907527715
```