

**Homework 0**  
**Parts 1-3: Due Tuesday, September 8**  
**Parts 4-6: Due Friday, September 11**

Welcome to CS205. In this class, we will be using a variety of tools that will require some initial configuration. To ensure everything goes smoothly moving forward, we will setup the majority of those tools in this homework. While some of this will likely be dull, doing it now will enable us to do more exciting work in the weeks that follow without getting bogged down in further software configuration. Some questions in this assignment will refer to provided skeleton code and/or data sets.

|   |           |
|---|-----------|
| <b>Problem 1 - Accounts, Class Survey, and Piazza Forum</b>   | <b>3</b>  |
| Create a GitHub account . . . . .   | 3         |
| Class Survey . . . . .  | 3         |
| Class Piazza Forum . . . . .  | 3         |
| Sign up for AWS credits . . . . .   | 3         |
| <b>Problem 2 - Set up your Virtual Workstation</b>  | <b>4</b>  |
| Install an Ubuntu VirtualBox . . . . .  | 4         |
| Add the Virtual Box Guest Additions (Optional) . . . . .  | 5         |
| Troubleshooting . . . . .   | 5         |
| Configure and Initialize . . . . .  | 5         |
| Make the VM a multicore machine . . . . .   | 5         |
| Set up Git . . . . .  | 5         |
| Set up SSH keys for GitHub access . . . . .   | 6         |
| Other personalizations . . . . .  | 6         |
| Linux Essentials . . . . .  | 6         |
| <b>Problem 3 - Fork/clone the cs205-homework repository, create a branch, do some work, and submit it via a Pull Request. [10%]</b> | <b>7</b>  |
| TL;DR for those familiar with git . . . . .   | 7         |
| Fork & clone the cs205-homework repository . . . . .  | 7         |
| Fork the repository to your GitHub account . . . . .  | 8         |
| Clone the repository to your VirtualBox . . . . .   | 8         |
| Create a HW0a branch . . . . .  | 8         |
| Do some work . . . . .  | 9         |
| Commit & Push . . . . .   | 9         |
| Open a Pull Request to submit your work . . . . .   | 10        |
| <b>Problem 4 - Getting Your Feet Wet with Python [40%]</b>  | <b>12</b> |
| Before you start: Move to a new branch (HW0b) . . . . .   | 12        |
| Welcome to Python . . . . .   | 12        |
| Executing Python Code . . . . .   | 12        |
| Lists . . . . .   | 13        |

|  |           |
|--|-----------|
| Arrays and Matrices . . . . .                              | 13        |
| Loops . . . . .  | 14        |
| Functions . . . . .  | 14        |
| Plotting with Python . . . . .                             | 15        |
| Putting It All Together . . . . .                          | 15        |
| Ground Truth . . . . .                                     | 16        |
| Measurements . . . . .                                     | 17        |
| Simple Model . . . . .                                     | 17        |
| Kalman Filtering . . . . .                                 | 18        |
| <b>Problem 5 - Introduction to Parallel Thinking [20%]</b> | <b>20</b> |
| <b>Problem 6 - The Multiprocessing Module [30%]</b>        | <b>21</b> |
| Count to Ten . . . . .                                     | 21        |
| How Much Faster? . . . . .                                 | 21        |
| <b>Submission instructions</b>                             | <b>22</b> |

# Problem 1 - Accounts, Class Survey, and Piazza Forum

## Create a GitHub account

We will be using [GitHub](#) for distributing and handing in homeworks. You will need to create a GitHub account, if you don't already have one. If you are not familiar with git, you will almost surely want to work through the [interactive tutorial](#) and probably read through some of the git-related resources on the [CS205 webpage](#).

## Class Survey

Please complete the mandatory course survey [located here](#). It should only take a few moments of your time.

## Class Piazza Forum

This class will use Piazza which you can use to communicate with and have your questions answered by both your fellow classmates and the course staff. [Register for this forum here](#).

Once registered, please create a post to introduce yourself to your classmates and course staff. In this post, include your name/nickname, your affiliation within Harvard, some of your interests/research, and tell us something interesting about yourself (e.g., an unusual hobby, past travels, or a cool project you did, etc.). This will help us get to know you and help you and your classmates when you look for partners and collaborators.

## Sign up for AWS credits

If you haven't already done so for another class, sign up for [AWS Educate](#) as a student. This will get you AWS credits that we will need for future parts of the course. You will have to sign up for an AWS account, as well, if you don't already have one.

## Problem 2 - Set up your Virtual Workstation

Throughout the class, we recommend everyone use a virtual machine to ensure an identical, preconfigured work environment. This makes it much easier for the course staff to provide assistance in case you have difficulties related to installation or configuration of software and libraries.

### Install an Ubuntu VirtualBox

To install a clean, virtual Linux system, follow the steps below.

- Download the [VirtualBox binary for your operating system here](#).
- Download the [Ubuntu 14.04 ISO image from here](#) (approx 1 GB).

Install and launch the virtual box executable. To install a virtual machine click **New** and use the recommended install parameters:

- **Name:** CS205
- **Operating System:** Linux
- **Version:** Ubuntu (32 bit)
- **Base memory size:** 1024 MB
- **Create a virtual hard drive now.** (default)
- **Hard drive file type:** VDI (default)
- **Storage on physical hard drive:** Dynamically allocated (default)
- **File location:** CS205 (default)
- **File size:** 8GB (default)

This will initialize the a virtual machine called **CS205** in the main selection window. Now we may install the Ubuntu image. Select the **CS205** machine and click **Start**:

- **Media Source:** /path/to/ubuntu-14.04.1-desktop-i386.iso

This will launch you into the Ubuntu OS install screen:

- **Install Ubuntu**
- **Preparing to install:** No need to check these boxes (default)
- **Erase Disk and Install Ubuntu.** This will “format” your virtual hard drive. Don’t worry, all your files are safe.
- **Select drive:** SCSI1 (default)
- **Install Now.**

To complete the set up, we will be installing a number of packages that CS205 will be using during this course. Open a terminal window by clicking **Dash home** → **Terminal** or by pressing Ctrl+Alt+T. Enter the following commands (# is a comment – do not enter these lines)

```
# Update the package index we are installing from
```

```
$ sudo apt-get update
# Upgrade the installed packages to the most recent versions
$ sudo apt-get upgrade -y
# Install packages we will be using in this course: python, numpy, scipy, Cython, git,
$ sudo apt-get install python-numpy python-scipy python-matplotlib -y
$ sudo apt-get install ipython python-setuptools cython git -y
# Install common editors
sudo apt-get install emacs24 vim -y
```

Restart the Ubuntu virtual machine by clicking the top-right corner to finalize the updates.

## Add the Virtual Box Guest Additions (Optional)

To improve the VirtualBox interface (i.e. resizable window, better graphics support, etc) [follow the instructions here](#) to install VirtualBox Guest Additions. Restart the VirtualBox. You may need to enable **Shared Clipboard** in the **Devices** menu of VirtualBox.

## Troubleshooting

If the VirtualBox freezes after the installation when the virtual machine is attempting to reboot, follow these steps:

- Force close the CS205 machine.
- Right-click the CS205 machine in VirtualBox, and go to **Settings** → **Storage**.
- Remove the Ubuntu iso image from the **Controller: IDE**.
- Start the CS205 machine normally.

## Configure and Initialize

### Make the VM a multicore machine

- Shut down the CS205 machine.
- Right-click the CS205 machine in VirtualBox, and go to **Settings** → **System**.
- In the **Processor** tab, drag the slider to give the VM 4 CPUs (use fewer if your physical machine has fewer actual CPUs).
- Close **Settings** and restart the CS205 machine.
- In the **Terminal**, if you type the command

```
$ less /proc/cpuinfo
```

you should see 4 processors listed.

## Set up Git

Git records your name and email address with every commit you make, so the first step is to tell Git who you are. Run these commands on your VirtualBox in the Terminal:

```
$ git config --global user.name "Your Name"
$ git config --global user.email "you@harvard.edu"
```

## Set up SSH keys for GitHub access

You may want to set up [SSH keys for GitHub access](#) to simplify things in the future. You should run the commands on your VirtualBox.

## Other personalizations

At this point, feel free to customize your system how you like. This may include:

- Installing Dropbox (use the Ubuntu Software Center).
- Installing your preferred IDE or editor (emacs, vim, gedit will already be there, but you may prefer Eclipse, Sublime Text, or something else).
- ??? - Share on Piazza if you know of tools to make your Linux experience more pleasant!

## Linux Essentials

We'll be running a lot of commands at the command line in the Terminal, particularly when working with git. If you are not familiar with the Linux/Unix command line, you may want to work through the [Command Line Interface Crash Course](#) or [this Unix tutorial](#).

Some basics:

- `cp file1 file2` - Copies `file1` into the location specified by `file2`.
- `rm myfile` - Removes the file `myfile`.
- `mkdir directory` - Creates a folder titled `directory`.
- `ls` - Lists all the files in the current directory. The `-a` flag shows hidden files and directories. The `-l` flag shows additional information.
- `clear` - Clears the screen.
- `cd directory` - Switches the present working directory to `directory`. Use `cd ..` to go up a directory and `cd ~` to go to your home directory.
- `pwd` - Prints the present working directory to the screen.
- `mv file1 file2` - Moves a file from `file1` to `file2`.
- `man XYZ` - Displays the help (manual) for the command `XYZ`. You will use this a lot.

## Problem 3 - Fork/clone the cs205-homework repository, create a branch, do some work, and submit it via a Pull Request. [10%]

This subproblem is long, but it is critical that you gain an understanding of working with git and GitHub. It is strongly suggested that if you do not know git already, you go through the [Git Tutorial](#).

We will use the [GitHub Flow](#) model for homework, including submitting your work via Pull Requests. The overall process will be:

- Setup: Fork the `cs205-homework` repository to your account on GitHub, and clone it to your VirtualBox.
- For each homework:
  - Create a new branch for your work.
  - While you still have something to do:
    - \* `HACK HACK HACK...` to get something done.
    - \* `git commit ...` your work.
    - \* `git push ...` your commits to GitHub
  - When you are ready to submit your work: open a pull request to the class's `harvard-cs205/cs205-homework` repository.

The inner HACKing loop is important! Committing is useful as you complete steps in your work, and pushing it to GitHub makes sure you have your work backed up somewhere. If you're not familiar with the idea of "Commit early, commit often" idea, [read this](#).

### TL;DR for those familiar with git

If you are already familiar with git and GitHub, here's what you need to do for this problem, which is primarily to make sure you can work with git and related tools, and submit your work to us.

- Fork the `cs205-homework` repository to your account on GitHub, and clone it to your VirtualBox.
- Create a `HW0a` branch in your local VirtualBox repository.
- Add your name to `HW0/README`
- `git commit` your changes.
- `git push` the `HW0a` branch to your GitHub repo.
- Open a pull request to the upstream `harvard-cs205/cs205-homework` repo with your work.

Everything beyond this point is the "non-TL;DR" version.

## Fork & clone the cs205-homework repository

### Fork the repository to your GitHub account

Go to the [harvard-cs205/cs205-homework](#) repository on GitHub. On this page, you'll see a Fork button in the upper right:  .

Click that button, and choose your own account (if given a choice). This will create a copy of the repository that you can modify, and will forward your browser to that page. The URL of your new repository should be:


`https://github.com/<USERNAME>/cs205-homework`

with your GitHub username substituted, if everything is functioning as expected.

### Clone the repository to your VirtualBox

On the page for your repository, find the clone URL, in the right column:

#### SSH clone URL

`git@github.com:thou:` 

You can clone with [HTTPS](#), [SSH](#),  
or [Subversion](#). ⓘ

If you set up SSH keys, this URL will start with "`git@github.com:...`". If not, it will start with "`https://github.com/...`". In your VirtualBox, in the **Terminal**, enter this command:

```
$ git clone <URL>
```

replacing `<URL>` with the URL from the github page. You may want to make sure you are in your home directory, first (run the `cd` command in the **Terminal**. If you then execute these commands in the **Terminal**:

```
$ cd cs205-homework
```

```
$ ls
```

You should see the top level files from the repository have been copied to your VirtualBox.



## Create a HW0a branch

Work in the [GitHub Flow](#) model is done on branches. Short-lived branches (such as we'll be using for most homeworks) are often referred to as "topic branches".<sup>1</sup>

To make it easier for us to track your work, we'll suggest branch names for each assignment (or piece of one) that we expect you to submit, using the pattern established here, i.e., **HWxy**, where **x** is the homework number, and **y** is a letter (starting with "a").

Let's create a topic branch for the first part of HW0. In the **Terminal**, within your **cs205-homework** directory, run these commands:

```
$ git checkout -b HW0a
$ git branch
```

You should see this:

```
* HW0a
  master
```

The asterisk indicates which branch you are working on.

## Do some work

In your VirtualMachine, edit **README** file in the **HW0** subdirectory and add your name. Ubuntu comes with a default editor, **gedit** that you can use for this task:

```
$ gedit HW0/README
```

add your name to the file, save it, and close gedit.

Now run these commands, to see the state of your work:

```
$ git status
$ git diff
```

The first command lists which files have been modified (and which **git** doesn't know about, i.e. "untracked"). The second shows the actual changes you've made. You should see something like:

```
diff --git a/HW0/README b/HW0/README
index e69de29..81bbfcb 100644
--- a/HW0/README
+++ b/HW0/README
@@ -0,0 +1,2 @@
+My name is Ray
```

---

<sup>1</sup>As opposed to longer-lived branches, such as a **dev** branch running parallel but separate to the default **master** branch.

## Commit & Push

To move changes from our working tree to git's repository, we need to commit them. Committing proceeds in two parts:

```
$ git add README           # 1- Tell git which files' changes
                           # we're going to commit.
$ git status               # 2- Not needed, but compare to previous result.
$ git commit -m "Added my name" # 3- Commit with an explanation message.
$ git log                  # 4- You should see the commit you made.
$ git status               # 5- Compare, again.
```

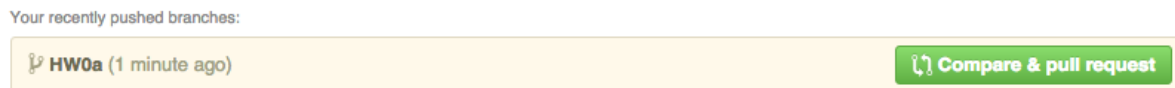
After these steps, your work has been “committed,” but is still only on your VirtualBox. To make it available for others to see (and us to grade it), we have to `git push` it to GitHub.

```
$ git push origin HW0a
```

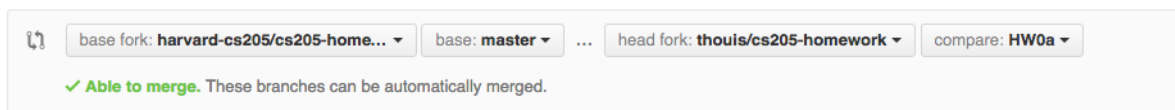
In this command, `origin` is the name of the remote repository to push to. When we cloned the repository with the `git clone` command, your fork of `cs205-homework` on GitHub was given the name `origin` in your local repository (on the VirtualBox).

## Open a Pull Request to submit your work

If you now go to your GitHub repository page in the browser, you should see something like this:



Click the “Compare & pull request” button. This will take you to a page that shows you the commits you’ve made, relative to the code in the class’s upstream `harvard-cs205/cs205-homework` repository. At the top of this page, you should see something like this:



The four entries should be:

- base fork: `harvard-cs205/cs205-homework`
- base: `master`
- head fork: `<USERNAME>/cs205-homework`
- compare: `HW0a`

The first two are where you’re going to submit your changes (i.e., your work) in the form of a pull request. The latter two are where that work currently resides (your repository, and

your branch). Below this, you should see that you've made changes to one file (HW0/README), and a summary of those changes.

Click the "Create pull request" button.

That's it! You've successfully submitted your work for the first part of this assignment.

**IMPORTANT:** Before you start on parts 4-6, you should move to a new branch. See the first part of problem 4 below.

## Problem 4 - Getting Your Feet Wet with Python [40%]

### Before you start: Move to a new branch (HW0b)

First, check that all of your work from parts 1-3 is saved, committed, and pushed. Run these commands:

```
$ git fetch
$ git status
```

You should check the output of `git status` for these things:

- At the top, look for the phrase: `Your branch is up-to-date with 'origin/HW0a'..` (Otherwise: you didn't push.)
- No files listed in `Changes to be committed`. (Otherwise: you didn't commit.)
- No files listed in `Changes not staged for commit`. (Otherwise: you didn't `git add` before committing.)
- Any files in `Untracked files` are expected, and don't contain any work you expected to have handed in. (Otherwise: you didn't `git add` before committing.)

If these aren't all true, you want to resolve them before going on, or you risk **losing your work**.

If everything is as expected, you can move to a new branch for more work:

```
$ git checkout -b HW0b
$ git branch
```

The second command should show you three branches (`master`, `HW0a`, `HW0b`), with the current branch being `HW0b`.

## Welcome to Python

We will be using [Python](#) as our language of choice in this class. For those of you unfamiliar with it, we will be holding weekly sessions for the first few weeks to introduce you to the language. In the meantime, feel free to browse the language's [vast documentation and tutorials](#).

This problem will get you started with Python and give you a hint of what's to come.

## Executing Python Code

Being an interpreted language, Python requires an interpreter to execute a script. The interpreter is called by typing `python` at the command line. If a file is passed as the first argument (i.e. `python myfile.py`), the interpreter will execute the specified script. If no parameters are given, the interpreter will launch in interactive mode where you can type individual

commands one at a time. Interactive mode is similar to using the prompt in Matlab in the sense that variables can be created, functions can be called, etc. In general, most things that can be done via script can be done in interactive mode.

You should now familiarize yourself with basics of using numbers and strings in Python. [Sections 3.1.1 and 3.1.2 in the tutorial here](#) should suffice.

## Lists

Lists are one of the data types commonly seen in Python. They are 0-indexed and declared by enclosing a comma-separated series of values in square brackets. A single list can hold a variety of different data types (including other lists) and are quite flexible as seen in this line below.

---

```
1 mylist = [1, 2.5, "Hello", [3, "World"]]
2 print(mylist[0])           # Prints 1
3 mylist[3][1] = ("World!!!").upper()
4 print(mylist[3])           # Prints [3, 'WORLD!!!']
5 mylist.reverse()
6 print(mylist)              # Prints [[3, 'WORLD!!!'], 'Hello', 2.5, 1]
7 mylist[0].append(7)
8 print(mylist[0])           # Prints [3, 'WORLD!!!', 7]
```

---

A few common ways to manipulate lists include:

---

```
1 a = range(0,5,2)           # [0,2,4], range(begin,end,step), step is int
2 a.append(5)                 # [0,2,4,5]
3 b = [2*i for i in a]        # [0,4,8,10]
4 c = a + b                   # [0,2,4,5,0,4,8,10], Concat, NOT addition
5 d = a * 2                   # [0,2,4,5,0,2,4,5], Concat, NOT mult
6 e = a[1:3]                  # [2,4], Excludes last index
7 g = a[2:]                   # [4,5], Index 2 through last
8 h = a[-1]                   # [5], Index backwards from end
9 i = a[:-2]                  # [0,2], All but the last two elements
```

---

For more information and operations on lists, [see this documentation](#).

## Arrays and Matrices

While a 2D numerical dataset can be constructed as a list of lists, the **numpy** module contains two objects that better suit this purpose: arrays and matrices. While the elements of lists may be any type, the elements of arrays and matrices are all of the same type. While arrays and matrices have many of the same operations and syntax as lists, there are some differences, they are geared toward numerical computation. For example, in the above list code, `*` and `+` are concatenation operators, but for arrays and matrices these are element-wise mathematical operations.

If you are familiar with Matlab arrays already, we suggest reading [this list of differences between Python arrays and Matlab arrays](#).

---

```

1 import numpy as np      # Use numpy package with alias np
2
3 a = np.arange(0,1,0.3) # [0,0.3,0.6,0.9], step is float
4 b = np.array([[0,1], [2,3]]) # 2x2 array
5 B = np.matrix([[0,1], [2,3]]) # 2x2 matrix
6 C = np.matrix('4 5; 6 7')    # 2x2 matrix
7 z = np.zeros([2,2])          # 2x2 ARRAY of zeros
8 Z = np.asmatrix(z)          # 2x2 MATRIX of zeros
9
10 blockMat = np.bmat('B C;C Z') # 4x4 matrix from blocks
11
12 b2 = b*b                  # Element-wise multiplication
13 B2 = B*B                  # Matrix multiplication

```

---

## Loops

Unlike in C, C++, Java, and a variety of other languages, **whitespace plays an important role in Python**. When writing a loop, instead of using { and } to denote the section of code that should be repeated, in Python one denotes such code via indentation. While the amount is arbitrary, it must be consistent throughout the section of code. Whitespace is preceded by a full colon to indicate the beginning of the loop's body. See below for a brief example.

---

```

1 # Print 0 through 9
2 i = 0
3 while i < 10:
4     print(i)
5     i += 1
6
7 # Infinite Loop!
8 i = 0
9 while i < 10:
10    print(i)
11    i += 1

```

---

## Functions

When defining functions, there are a few key points to remember. Declarations are specified using the `def` keyword. Function parameters are enclosed in parentheses and comma-separated. In addition, just as with loops, a colon is used to denote the beginning of a new block, and the function's body is indicated through indentation. Anonymous functions can also be created using lambda functions. These can be created with other functions to create functions on the fly. See below for examples of each.

---

```
1 def add1(x):
2     return x+1
3
4 add2 = lambda x: x+2
5
6 def functor_add(x):
7     return lambda y: y+x
8
9 add3 = functor_add(3)
10 add4 = functor_add(4)
11
12 print(add1(1)) # Prints 2
13 print(add2(1)) # Prints 3
14 print(add3(1)) # Prints 4
15 print(add4(1)) # Prints 5
```

---

## Plotting with Python

Matplotlib is a commonly used tool for plotting data in Python. A basic plot can be constructed as follows.

---

```
1 import matplotlib.pyplot as plt
2 import math
3 import numpy as np
4
5 x = np.linspace(-2, 3, 51)
6 y1 = [math.exp(i) for i in x]
7 y2 = [math.exp(2*i) for i in x]
8
9 plt.plot(x, y1, '-b')
10 plt.plot(x, y2, '--g')
11 plt.yscale('log')
12 plt.xlabel('x')
13 plt.ylabel('y')
14 plt.title('sample plot')
15 plt.show()
```

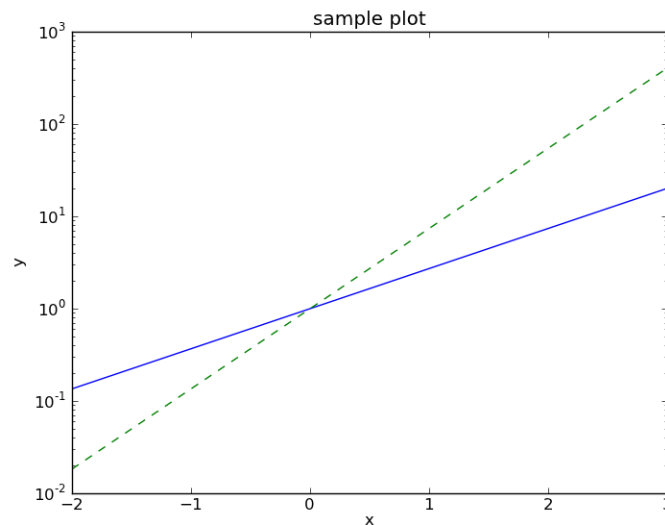
---

Produces:

[The gallery on the matplotlib website](#) provides code for constructing a variety of plots.

## Putting It All Together

You are on a robotics team and are tasked with predicting the path of a ball so the robot can predict where it will be and move to catch it. The robot has a (pretty bad) camera that estimates the position of the ball at each time step. Your task is to use this data to



approximate the true path of the ball. The skeleton files for this problem are in the HW0/P4 directory.

## Ground Truth

An oracle gives you the the exact location and velocity of the ball at each time step so that we can see how good any measurements and predictions actually are. In `P4_trajectory.txt`, each line is a comma-delimited list of floating-point values:

$$x, y, z, v_x, v_y, v_z$$

where  $\mathbf{x} = (x, y, z)$  is the position vector and  $\mathbf{v} = (v_x, v_y, v_z)$  is the velocity vector of the ball. Together, the  $k$ th line of the file defines the exact state of the ball at the  $k$ th time step:

$$\mathbf{s}^k = \begin{bmatrix} \mathbf{x}^k \\ \mathbf{v}^k \end{bmatrix} = \begin{bmatrix} x^k \\ y^k \\ z^k \\ v_x^k \\ v_y^k \\ v_z^k \end{bmatrix}$$

Complete the code in `P4.py` for Part 1 by reading in the data in `P4_trajectory.txt` into a variable called `s_true` and plotting the positions  $\mathbf{x}^k$  (the first three columns) using the commented `ax.plot` command.

HINT: Consider using `numpy.loadtxt`. To figure out how to use it, you can either Google or use the built-in documentation by executing the following commands:

```
$ ipython
```



```
: import numpy
: numpy.loadtxt?
```

## Measurements

Your robot has a simple tracker that will tell you the position of the ball at each time step. These measurements are listed in `P4_measurements.txt` in the same format as above. Note we only have position measurements (velocity is harder to measure). In addition, our tracker is rather inaccurate and stretches the dimensions! Since each measurement obviously depends on the true state of the ball,  $\mathbf{p}^k$ , we decide the we can model the measurements like so:

$$\mathbf{m}^k = \begin{bmatrix} x_m^k \\ y_m^k \\ z_m^k \end{bmatrix} = \begin{bmatrix} r_x & 0 & 0 & 0 & 0 & 0 \\ 0 & r_y & 0 & 0 & 0 & 0 \\ 0 & 0 & r_z & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x^k \\ y^k \\ z^k \\ v_x^k \\ v_y^k \\ v_z^k \end{bmatrix} + \text{error} = \mathbf{C}\mathbf{s}^k + \varepsilon^k$$

where  $\mathbf{m}^k = (x_m^k, y_m^k, z_m^k)$  is the position vector measured at the  $k$ th time step (and is the  $k$ th line in the file) and  $\varepsilon$  is a vector of (unknown) random errors. Here  $\mathbf{C}$  is accounting for the dimensional stretching and the fact that we're only measuring position.

We don't know what the errors are, but we can at least fix the stretching when we plot the measurements. Read the data from `P4_measurements.txt` and plot the approximate position points

$$\tilde{\mathbf{x}}^k = \begin{bmatrix} 1/r_x & 0 & 0 \\ 0 & 1/r_y & 0 \\ 0 & 0 & 1/r_z \end{bmatrix} \begin{bmatrix} x_m^k \\ y_m^k \\ z_m^k \end{bmatrix} = \begin{bmatrix} 1/r_x & 0 & 0 \\ 0 & 1/r_y & 0 \\ 0 & 0 & 1/r_z \end{bmatrix} \mathbf{m}^k$$

using the commented `ax.plot` command. The parameter values for constructing the matrix can be found in the provided skeleton code `P4.py`.

## Simple Model

Since the physics of a ball is pretty simple, your intern decides that he can predict the path of the ball from just the first measurement. From Newton's Law,

$$m \frac{\partial^2 \mathbf{x}}{\partial t^2} = \mathbf{F}$$

Noting that  $\mathbf{v} = \frac{\partial \mathbf{x}}{\partial t}$ , we can rewrite this as the system of equations

$$\frac{\partial}{\partial t} \begin{bmatrix} \mathbf{x} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ \mathbf{F}/m \end{bmatrix}$$

Let  $\mathbf{F}/m = \mathbf{g} - c\mathbf{v}$  be the acceleration due to gravity and a drag due to air resistance. Gravity is in the  $z$ -direction so  $\mathbf{g} = [0 \ 0 \ g]^T$  where  $g = -9.81$ . Using two time steps to approximate the time derivative, he gets

$$\frac{1}{\Delta t} \left( \begin{bmatrix} \mathbf{x}^{k+1} \\ \mathbf{v}^{k+1} \end{bmatrix} - \begin{bmatrix} \mathbf{x}^k \\ \mathbf{v}^k \end{bmatrix} \right) = \begin{bmatrix} \mathbf{v}^k \\ \mathbf{g} - c\mathbf{v}^k \end{bmatrix}$$

which can be rewritten as

$$\mathbf{s}^{k+1} = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 - c\Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 - c\Delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 - c\Delta t \end{bmatrix} \mathbf{s}^k + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ g\Delta t \end{bmatrix} = \mathbf{A}\mathbf{s}^k + \mathbf{a} \quad (1)$$

where  $\mathbf{A}$  is called the **propagation matrix**.

Therefore, if we know the initial state,  $\mathbf{s}^0$ , then we can determine all of the states,  $\mathbf{s}^k$ , from applications of the matrix  $\mathbf{A}$ ! He tries it with the data in `P4_measurements.txt`, but decides it's too inaccurate and useless. Instead, he convinces your advisor to buy some really expensive sensors to determine  $\mathbf{s}^0$  very accurately. He gets

$$\mathbf{s}^0 = [0 \ 0 \ 2 \ 15 \ 3.5 \ 4.0]^T$$

(which amazingly matches the first line of the oracle data exactly!).

Use the initial condition  $\mathbf{s}^0$  above and the propagation matrix  $\mathbf{A}$  to compute all of the  $\mathbf{s}^k$  for  $k = 0, 1, \dots, K-1$ . Plot the predicted positions,  $\mathbf{x}^k$ , using the commented `ax.plot` command.

HINT: Construct a large  $6 \times K$  matrix and fill it in one column at a time. When complete, convert it to an array before plotting the first three rows.

## Kalman Filtering

Despite your intern's best efforts, the predicted path and the real path still don't line up well. You recognize that there must be error in the physical propagation as well:

$$\mathbf{s}^{k+1} = \mathbf{A}\mathbf{s}^k + \mathbf{a} + \begin{bmatrix} b_x & 0 & 0 & 0 & 0 & 0 \\ 0 & b_y & 0 & 0 & 0 & 0 \\ 0 & 0 & b_z & 0 & 0 & 0 \\ 0 & 0 & 0 & b_{vx} & 0 & 0 \\ 0 & 0 & 0 & 0 & b_{vy} & 0 \\ 0 & 0 & 0 & 0 & 0 & b_{vz} \end{bmatrix} \boldsymbol{\epsilon}^k = \mathbf{A}\mathbf{s}^k + \mathbf{a} + \mathbf{B}\boldsymbol{\epsilon}^k$$

where  $\boldsymbol{\epsilon}$  is a vector of (unknown) random fluctuations in the ball's state (maybe due to measurement error, air currents, or spin) and  $\mathbf{B}$  is a matrix describing the scale of these errors.

We don't know what these errors are so computing the next state  $\mathbf{s}^{k+1}$  as before won't work. But it will work as a good initial guess. The propagated state from your intern then becomes only an intermediate guess:

$$\tilde{\mathbf{s}} = \mathbf{A}\mathbf{s}^k + \mathbf{a} \quad (2)$$

that we want to fix using the next measurement,  $\mathbf{m}^{k+1}$ , and our confidence in the current state  $\mathbf{s}^k$ . Then, we introduce a covariance matrix,  $\Sigma^k$  which is like a measure of how confident we are in the current state  $\mathbf{s}^k$ . If we compute the matrix

$$\tilde{\Sigma} = (\mathbf{A}\Sigma^k\mathbf{A}^T + \mathbf{B}\mathbf{B}^T)^{-1} \quad (3)$$

for reasons that are beyond this explanation, then we want to solve the system

$$\tilde{\Sigma}\mathbf{s}^{k+1} + \mathbf{C}^T\mathbf{C}\mathbf{s}^{k+1} = \tilde{\Sigma}\tilde{\mathbf{s}} + \mathbf{C}^T\mathbf{m}^{k+1}$$

which makes sure that  $\mathbf{s}^{k+1}$  is close to the predicted  $\tilde{\mathbf{s}}$  (see the first term on the left and the first term on the right) and that  $\mathbf{C}\mathbf{s}^{k+1}$  (the value we would measure for the next state) is close to  $\mathbf{m}^{k+1}$  (the actual next measurement). Solving the system for  $\mathbf{s}^{k+1}$  is easy, and we update the confidence matrix  $\Sigma^{k+1}$  at the same time:

$$\Sigma^{k+1} = (\tilde{\Sigma} + \mathbf{C}^T\mathbf{C})^{-1} \quad (4)$$

$$\mathbf{s}^{k+1} = \Sigma^{k+1} (\tilde{\Sigma}\tilde{\mathbf{s}} + \mathbf{C}^T\mathbf{m}^{k+1}) \quad (5)$$

Equations (2) and (3) are called the **prediction steps** and equations (4) and (5) are called the **update steps**. Define four python functions:

- **predictS**: Implements equation (2) and returns  $\tilde{\mathbf{s}}$ .
- **predictSig**: Implements equation (3) and returns  $\tilde{\Sigma}$ .
- **updateSig**: Implements equation (4) and returns  $\Sigma^{k+1}$ .
- **updateS**: Implements equation (5) and returns  $\mathbf{s}^{k+1}$ .

For initial conditions, use  $\mathbf{s}^0$  from Part 3 and  $\Sigma^0 = 0.01\mathbf{I}$ , where  $\mathbf{I}$  is the identity matrix.

Save each of the  $\mathbf{s}^k$  in a  $6 \times K$  matrix (like was done in Part 3) and plot the positions (the first three rows) to see the filtered predicted path.

**HINT:** You'll want to turn the measurement data you read in Part 2 into a matrix to get matrix-vector multiplications. Note that you do not need to save  $\tilde{\mathbf{s}}$ ,  $\tilde{\Sigma}$ , or even all of the  $\Sigma^k$ .

## Submission Requirements

- P4/P4.py: Completed Script

## Problem 5 - Introduction to Parallel Thinking [20%]

Frustrated with the rising cost of textbooks, you have decided to enact revenge on the campus bookstore; you will pay for your textbooks with a suitcase full of small bags of pennies. Although each bag is correctly labeled with the number of pennies inside it, there are quite a few bags, and the cashiers must sum their totals to verify you are providing the correct amount of money. Armed with their calculators, it takes them 1 second to add two numbers (assume one person can only add two numbers at a time).

1. If one cashier were to add the totals of 256 bags, how long would it take?
2. Seven of the cashier's coworkers approach to see this ridiculous sight. The bright (and perhaps a bit opportunistic) cashier recruits them to help count. By working together, can they verify the total sum faster than the original cashier by herself? If so, how fast can they do this?
3. Imagine the store is big. . . very big. As a matter of fact, the store has an infinite number of employees. How fast can the employees verify the sum?
4. If you were feeling especially frustrated, perhaps you may use more than 256 bags. Instead, assume there are  $N$  bags. How long will it take the infinite number of employees to count the total? Using Python, plot the result with bag count on the horizontal and counting time on the vertical. On the same graph, plot the time it would take the lone cashier to do this by herself.
5. We've been neglecting any overhead associated with people communicating. How does the answer for 256 bags change, if it takes one second to communicate a number between two employees?
6. Also assume it takes one second to hand a bag (or any number of bags) between two people (including from you to a cashier). How long will it take to verify 256 bags worth of coins if each cashier comes to you and takes 1 bag? What about 2 bags? Is there a better strategy? Assume you can't be handing bags to more than one person at a time.

### Submission Requirements

- P5/P5.txt: Answers.
- P5/P5.png: Plot for fourth question.

## Problem 6 - The Multiprocessing Module [30%]

One of the simplest ways to do things in parallel in Python is through the [multiprocessing module](#). Here, you'll reason through two simple examples that show the oddities of parallelism.

### Count to Ten

The multiprocessing module allows a pool of threads to complete a batch of jobs in parallel. The script P6A.py demonstrates this functionality.

---

```
1 import multiprocessing as mp
2 import time
3
4 def burnTime(k):
5     print "Hi Job %d" % k
6     time.sleep(0.25)
7     print "Bye Job %d" % k
8     return k
9
10 if __name__ == '__main__':
11     pool = mp.Pool(4); # Create a pool of 4 processes
12
13     # Apply burnTime to this list of "job numbers" using the pool
14     result = pool.map(burnTime, range(10))
15     print(result)
```

---

If you run the above code a number of times, you may see some unexpected results. Explain these results. How could this affect how we program in parallel? Describe a scenario where this would be important.

### How Much Faster?

In P6B.py, the burnTime has been changed to simply sleep for a parameterized amount of time. Using the Pool.map functionality, call burnTime 16 times in parallel using 4 processes and 16 times in serial using a standard loop. Use time.time() to determine how many seconds each takes and use various sleep times (ranging from  $10^{-6}$  to  $10^0$  seconds) for each timing. Plot the ratio of serial to parallel execution time against sleep time.

Try to explain the trend you observe. Is it possible that a parallel program could take longer than its serial version? Under what conditions?

### Submission Requirements

- P6/P6.txt: Provide example output, and answer all questions.
- P6/P6.png: Plot for P6B.py portion.
- P6/P6B.py: Completed Script

## Submission instructions

Submit your work the same way you did HW0a.

As you work, commit and push the files you've been working on (or have completed).

```
$ git add <file1> <file2> ...  
$ git commit -m "Work on <file1> <file2> ... for HW0b"  
$ git push origin HW0b
```

You can (and should!) do this multiple times! Don't worry about committing and pushing work in progress. It's good practice for two reasons: it breaks up more complex work into easier-to-understand chunks, and it allows you to back out changes in case you go take a wrong turn somewhere in your coding.

When you've completed your work, open a Pull Request, using the same method as for HW0a. Make sure your pull request is from HW0b, and contains the work you are expecting to hand in. Check this on the pull request page.

If you notice you forgot something after opening the Pull Request, or need to make a change, don't worry. Any work you commit and push to the **HW0b** branch on GitHub will automatically update the Pull Request (in other words, just repeat the edit/add/commit/push loop with your updated work).