Computational Linear Algebra for Coders

This course is focused on the question: **How do we do matrix computations with acceptable speed and acceptable accuracy?**

This course was taught in the University of San Francisco's Masters of Science in Analytics program, summer 2017 (for graduate students studying to become data scientists). The course is taught in Python with Jupyter Notebooks, using libraries such as Scikit-Learn and Numpy for most lessons, as well as Numba (a library that compiles Python to C for faster performance) and PyTorch (an alternative to Numpy for the GPU) in a few lessons.

Accompanying the notebooks is a playlist of lecture videos, available on YouTube. If you are ever confused by a lecture or it goes too quickly, check out the beginning of the next video, where I review concepts from the previous lecture, often explaining things from a new perspective or with different illustrations, and answer questions.

Getting Help

You can ask questions or share your thoughts and resources using the **Computational Linear Algebra** category on our fast.ai discussion forums.

Table of Contents

The following listing links to the notebooks in this repository, rendered through the nbviewer service. Topics Covered:

0. Course Logistics (Video 1)

- My background
- Teaching Approach
- Importance of Technical Writing
- List of Excellent Technical Blogs
- Linear Algebra Review Resources

1. Why are we here? (Video 1)

We start with a high level overview of some foundational concepts in numerical linear algebra.

- Matrix and Tensor Products
- Matrix Decompositions
- Accuracy
- Memory use
- Speed
- Parallelization & Vectorization

2. Topic Modeling with NMF and SVD (Video 2 and Video 3)

We will use the newsgroups dataset to try to identify the topics of different posts. We use a term-document matrix that represents the frequency of the vocabulary in the documents. We factor it using NMF, and then with SVD.

- Topic Frequency-Inverse Document Frequency (TF-IDF)
- Singular Value Decomposition (SVD)
- Non-negative Matrix Factorization (NMF)
- Stochastic Gradient Descent (SGD)
- Intro to PyTorch
- Truncated SVD

3. Background Removal with Robust PCA (Video 3, Video 4, and Video 5)

Another application of SVD is to identify the people and remove the background of a surveillance video. We will cover robust PCA, which uses randomized SVD. And Randomized SVD uses the LU factorization.

- Load and View Video Data
- SVD
- Principal Component Analysis (PCA)
- L1 Norm Induces Sparsity
- Robust PCA
- LU factorization
- Stability of LU
- LU factorization with Pivoting
- History of Gaussian Elimination
- Block Matrix Multiplication

4. Compressed Sensing with Robust Regression (Video 6 and Video 7)

Compressed sensing is critical to allowing CT scans with lower radiation-- the image can be reconstructed with less data. Here we will learn the technique and apply it to CT images.

- Broadcasting
- Sparse matrices
- CT Scans and Compressed Sensing
- L1 and L2 regression

5. Predicting Health Outcomes with Linear Regressions (Video 8)

- Linear regression in sklearn
- Polynomial Features
- Speeding up with Numba
- Regularization and Noise

6. How to Implement Linear Regression(Video 8)

- How did Scikit Learn do it?
- Naive solution
- Normal equations and Cholesky factorization
- QR factorization
- SVD
- Timing Comparison
- Conditioning & Stability
- Full vs Reduced Factorizations

- Matrix Inversion is Unstable

7. PageRank with Eigen Decompositions (Video 9 and Video 10)

We have applied SVD to topic modeling, background removal, and linear regression. SVD is intimately connected to the eigen decomposition, so we will now learn how to calculate eigenvalues for a large matrix. We will use DBpedia data, a large dataset of Wikipedia links, because here the principal eigenvector gives the relative importance of different Wikipedia pages (this is the basic idea of Google's PageRank algorithm). We will look at 3 different methods for calculating eigenvectors, of increasing complexity (and increasing usefulness!).

- SVD
- DBpedia Dataset
- Power Method
- QR Algorithm
- Two-phase approach to finding eigenvalues
- Arnoldi Iteration

8. Implementing QR Factorization (Video 10)

- Gram-Schmidt
- Householder
- Stability Examples

Why is this course taught in such a weird order?

This course is structured with a *top-down* teaching method, which is different from how most math courses operate. Typically, in a *bottom-up* approach, you first learn all the separate components you will be using, and then you gradually build them up into more complex structures. The problems with this are that students often lose motivation, don't have a sense of the "big picture", and don't know what they'll need.

Harvard Professor David Perkins has a book, Making Learning Whole in which he uses baseball as an analogy. We don't require kids to memorize all the rules of baseball and understand all the technical details before we let them play the game. Rather, they start playing with a just general sense of it, and then gradually learn more rules/details as time goes on.

If you took the fast.ai deep learning course, that is what we used. You can hear more about my teaching philosophy in this blog post or this talk I gave at the San Francisco Machine Learning meetup.

All that to say, don't worry if you don't understand everything at first! You're not supposed to. We will start using some "black boxes" or matrix decompositions that haven't yet been explained, and then we'll dig into the lower level details later.

To start, focus on what things DO, not what they ARE.