Exercises03

February 21, 2015

In [1]:
```
import os
import sys
import glob
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
%matplotlib inline
plt.style.use('ggplot')
```

Exercise 1 (10 pts). Avoding catastrophic cancellation.
The tail of the standard logistic distributon is given by $1 - F(t) = 1 - (1 + e^{-t})^{-1}$.

- Define a function `f1` to calculate the tail probability of the logistic distribution using the formula given above
- Use `sympy` to find the exact value of the tail distribution (using the same symbolic formula) to 20 decimal digits
- Calculate the *relative error* of `f1` when $t = 25$ (The relative error is given by `abs(exact - approximate)/exact`)
- Rewrite the expression for the tail of the logistic distribution using simple algebra so that there is no risk of cancellation, and write a function `f2` using this formula. Calculate the *relative error* of `f2` when $t = 25$.
- How much more accurate is `f2` compared with `f1` in terms of the relative error?

In [2]:
```
# Your code here
```

Exercise 2 (10 pts). Ill-conditioned linear problems.
You are given a $n \times p$ design matrix X and a p-vector of observations y and asked to find the coefficients β that solve the linear equations $X\beta = y$.

```
X = np.load('x.npy')
y = np.load('y.npy')
```

The solution β can also be loaded as

```
beta = np.load('b.npy')
```

- Write a formula that could solve the system of linear equations in terms of X and y Write a function `f1` that takes arguments X and y and returns β using this formula.
- How could you code this formula using `np.linalg.solve` that does not require inverting a matrix? Write a function `f2` that takes arguments X and y and returns β using this.
- Note that carefully designed algorithms *can* solve this ill-conditioned problem, which is why you should always use library functions for linear algebra rather than write your own.

  ```
  np.linalg.lstsq(x, y)[0]
  ```

- What happens if you try to solve for β using `f1` or `f2`? Remove the column of X that is making the matrix singluar and find the $p-1$ vector b using `f2`.

- Note that the solution differs from that given by `np.linalg.lstsq`? This arises because the relevant condition number for `f2` is actually for the matrix $X^T X$ while the condition number of `lstsq` is for the matrix X. Why is the condition so high even after removing the column that makes the matrix singular?

```
In [3]: # Your code here
```

Exercise 3 (10 pts). Importance of using efficient algorihtms.

- Implement bubble sort
- Calculate its big $\mathcal{O}$ algorithmic complexity
- Time the performance of bubble sort on random uniform deviate vectors of sizes `range(100, 2000, 100)` using `time.time()` from the standard library
- Use `scipy.optimize.curve_fit` to fit an appropriate function to the collection of (size, execution time) data points. Extrapolate how long it would take to sort a random vector of size 1,000,000. Now time how long it takes for the system sort to sort a random vector of size 1,000,000.
- Plot the fits together with the data points uisng `matplotlib.pyplot` functions.

```
In [4]: # Your code here
```

Exercise 4 (20 pts). One of the goals of the course it that you will be able to implement novel algorihtms from the literature.

- Implement the mean-shift algorithm in 1D as described here.
  - Use the following function signature

    ```
    def mean_shift(xs, x, kernel, max_iters=100, tol=1e-6):
    ```

  - xs is the data set, x is the starting location, and kernel is a kernel function
  - tol is the difference in $||x||$ across iterations
- Use the following kernels with bandwidth h (a default value of 1.0 will work fine)
  - Flat - return 1 if $||x|| < h$ and 0 otherwise
  - Gaussian

$$\frac{1}{\sqrt{2\pi h}} e^{\frac{-||x||^2}{h^2}}$$

  - Note that $||x||$ is the norm of the data point being evaluated minus the current value of x
- Use both kernels to find all 3 modes of the data set in `x1d.npy`
- Modify the algorithm and/or kernels so that it now works in an arbitrary number of dimensions.
- Use both kernels to find all 3 modes of the data set in `x2d.npy`
- Plot the path of successive intermediate solutions of the mean-shift algorithm starting from `x0 = (-4, 10)` until it converges onto a mode in the 2D data for each kernel. Superimpose the path on top of a contour plot of the data density.

```
In [5]: # Your code here
```