The auto-grader will run tests against your code, all of which must pass for you to pass the assignment. Remember: you must pass all assignments to pass the class. Do not use the function np.vectorize anywhere in this assignment.

1. **Profiling and Vectorization** (15 points)
   In this question we are going to use Python profiling tools to determine the efficiency of using NumPy's vectorization capabilities. To do so we will compute the first 100 powers (exclusive and starting from 0) of 2 in the following ways. Use the `main()` method in `prof.py` to investigate the amount of time it takes to run each of these methods 100,000 times.

   To profile each of the statements you will create, use the `timeit` module. There are many ways to use this module (see http://docs.python.org/2.7/library/timeit.html) but you should only need to use the `timeit.timeit()` function which is implemented in the `main()` method.

   (a) append to an initially empty list sequentially in `empty_list()`

   (b) create an list of zeros of size 100 and assign elements sequentially in `preallocated_list()`

   (c) use a list comprehension in `lst_comp()`

   (d) use the python map function in `map_twos()`

   (e) use vectorization (NumPy) in `twos_numpy()`

2. **SciPy Optimization** (15 points)

   The `scipy.optimize` module contains functions that perform numerical optimization. Use `scipy.optimize.fmin` to minimize the following function with starter code in `scipy_opt.py` using the function `test_opt()`. If your version of scipy does not have `scipy.optimize.fmin`, use `scipy.optimize.fmin_bfgs`.

$$f(x) = \frac{1}{2}x^T Ax - b^T x$$

   When $A = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$ and $b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$. Verify that your solution x^* is a solution to the system of linear equations defined by A and b by passing your solution x^*, A and b to `verify_opt()`. That is, make sure $Ax^* = b$.

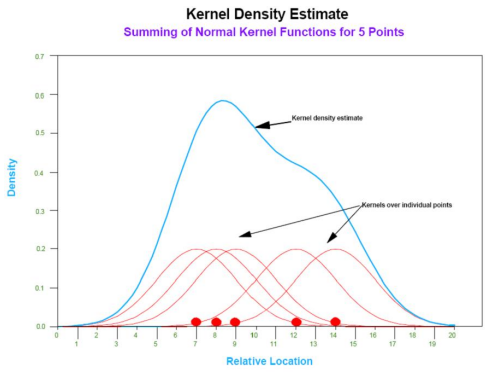
3. **Kernel Density Estimate (KDE)** (20 points)

Given a sample $x_1, x_2, \ldots, x_n$ from an unknown distribution f the kernel density estimate of f at point x with kernel K_b is defined as

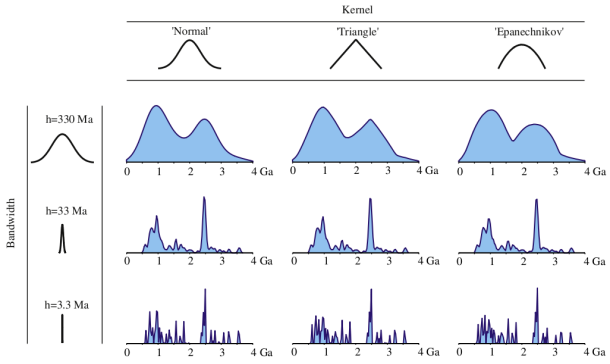
$$\hat{f}(x; b) = \frac{1}{n} \sum_{i=1}^{n} K_b(x - x_i)$$

where the kernel must satisfy $\int_{-\infty}^{\infty} K(u)du = 1$ and $K(-u) = K(u)$. The parameter b is known as the bandwidth and controls the width of the kernel used. There are many possible choices for kernels, and we will use the triangular kernel

$$K_b(z) = \frac{1}{b}(1 - |\frac{z}{b}|) * \mathbb{I}(|\frac{z}{b}| \leq 1)$$



(a) Gaussian Kernel Density Estimate



(b) Bandwiths and Kernels

Write a ONE line python function without list comprehension or the keyword `for` that takes the three parameters below and returns $\hat{f}(x)$, the KDE evaluated at x. Implement this in `kde.py`.

(a) `x` the point to evaluate the KDE

(b) `data` the sample of points from f ; above denoted $x_1, x_2, \ldots, x_n$

(c) `bw` the bandwidth of the kernel

4. **matplotlib** (0 points)

It will be useful to have matplotlib installed for lecture 5. Please install `matplotlib` on your system and make sure the following lines produce a graph

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(np.arange(10))
>>> plt.show()
```