

50 points total. 70+% correctness (35+ points) is needed to pass. Remember: you must pass all assignments to pass the class. The assignment is due at the beginning of the next class.

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 1000 powers (inclusive and starting from 0) of 2 in the following ways. Report 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()` function. To time multi-line statements, create the statement using triple quotes. Remember to import the necessary modules and create any necessary objects in the setup phase.

- (a) append to an initially empty list sequentially
- (b) create an list of zeros of size 1000 and assign elements sequentially
- (c) use a list comprehension
- (d) use the python map function
- (e) use vectorization (NumPy)

2. Copies and Views (10 points)

NumPy arrays are objects and follow the same assignment and copy rules as ordinary python objects. However, when we slice an array python returns a *view* on that same data, meaning that a new object is created but shares the same underlying data. Integer indexing and boolean indexing do not create views. Determine the value of the following statements and *very* briefly explain why.

- (a)

```
>>> def f(arr):
...     arr[0, 0] = 42
...     return arr
>>> A = np.ones((2, 5))
>>> B = f(A)
>>> A is B
```
- (b)

```
>>> A = np.ones((2, 5))
>>> B = A[:, 1:3]
>>> B[0,0] = 42.
>>> A[0, 1] == 1.
```
- (c)

```
>>> A = np.ones((2, 5))
>>> B = A[:, np.array([False, True, True, False, False])]
>>> B[0, 0] = 42
>>> A[0, 1] == 1
```
- (d)

```
>>> A = np.ones((2, 5))
>>> B = A[:, np.array([1, 3])]
>>> B[0, 0] = 42
>>> A[0, 1] == 1
```
- (e)

```
>>> A = np.ones((2, 5))
>>> B = A[:, 1:3]
>>> A += 1
>>> np.all(B == 2)
```

3. Kernel Density Estimate (KDE) (10 points)

Given a sample x_1, x_2, \dots, 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

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

Write a two line python function `kde(x, data, bw)` without list comprehension that takes the three parameters below and returns $\hat{f}(x)$, the KDE evaluated at x .

- (a) `x` the point to evaluate the KDE
- (b) `data` the sample of points from f ; above denoted x_1, x_2, \dots, x_n
- (c) `bw` the bandwidth of the kernel

4. SciPy Optimization (15 points)

The `scipy.optimize` module contains functions that perform numerical optimization. Use `scipy.optimize.minimize` to minimize the following function.

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

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 . That is, make sure $Ax^* = b$.

5. 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()
```