EXAM

Exam 1

Math 4330, Spring 2017

Feb. 28, 2017

- Write all of your answers on separate sheets of paper. Do not write on the exam handout. You can keep the exam questions when you leave. You may leave when finished.
- You **must** show enough work to justify your answers. Unless otherwise instructed, give exact answers, not approximations (e.g., $\sqrt{2}$, not 1.414).
- This exam has 4 problems. There are **350 points total**.

Good luck!

100 pts.

Problem 1. Consider a system of floating point numbers with radix β, and precision p. Let x be a normalized floating point number. Show that the distances from x to the adjacent normalized floating point numbers are bounded below by $\beta^{-1}\varepsilon_M|x|$ and bounded above by $\varepsilon_M|x|$, where ε_M is the machine epsilon.

---

100 pts.

Problem 2. In this problem, you want to write a function `revdict` that takes a dictionary `D` and returns a dictionary `R` that reverses `D` by making the keys into values and the values into keys. For example if `D[k] = v` we would want `R[v] = k`.

There's a problem. The keys in a dictionary are unique while several keys may have the same value. To fix this, we'll let `R[v]` be a list of the keys that have value `v`. So, if

```
D = { 'A':1, 'B':2, 'C':1}
```

we'll have `R[2] = ['B']` and `R[1] = ['A', 'C']`.

Write the function `revdict`. Include sorce code for the function and and some test runs to show it works.

---

50 pts.

Problem 3. The description of this problem is probably longer than the source code. For simplicity, we won't include error checking, so illegal input will crash the program. As usual, turn in source code and tests to show your code works.

A. Write a class `Matrix`. Objects in this class maintain an $m \times n$ matrix. (We'll handle pairs of numbers as tuples). Store the matrix in a list or list of lists, whichever you prefer. Keep track of the shape $m \times n$. Include the following basic methods.

`Matrix((m,n))` Provide a constructor that sets the shape of the matrix and sets all the entries to a default value of zero.

newMatrix The call `newMatrix((m,n))` clears the existing data in the `Matrix` and sets it to a matrix of zeros of shape `(m,n)`

getShape `getShape()` returns a tuple giving the shape of the matrix.

getElement The call `getElement((i,j))` returns the element at row `i` and column `j`.

setElement The call `setElement((i,j), value)` sets the entry at position `(i, j)` to `value`.

The rest of the methods should use those above in order to be independent of the internal implementation of the matrix.

setElements Takes a dictionary `D` where the keys are tuples `(i,j)` and the values are numbers. For each key, it sets the element at position `(i,j)` to `D[i,j]`.

SetByRows Takes a list of lists. The inner lists give values for each row, so the outer list describes the matrix. For example `[[1,2], [3,4]]` sets the matrix to the matrix with first row $\begin{bmatrix}1 & 2\end{bmatrix}$ and second row $\begin{bmatrix}3 & 4\end{bmatrix}$. This method will clear the existing data and reset the shape.

For the matrix operations, we want the user to provide the target object. So, if we have `A` and `B`, the user should provide an existing matrix object `C` to store the result. Thus `C.add(A,B)` should reset the data in `C` (using methods above) to the matrix sum of `A` and `B`. For simplicity, `C` should not be `A` or `B`. Provide methods for scalar multiplication and matrix multiplication. For example `C.scalarMult(num, A)` should set `C` to the result of multiplying `A` by the scalar `num`. For matrix multiplication, `C.mult(A,B)` should set `C` to the matrix product of `A` and `B`.

Also provide a method `duplicate` so that `C.duplicate(A)` sets `C` to the same mathematical matrix as `A`.

B. In many applications, you have huge matrices, but most of the elements of the matrix are zeros. It's inefficient to store all those zeros, it's only necessary to store the nonzero elements. Here is one approach to this idea.

Define a subclass of `Matrix` called `SparseMatrix`. In a sparse matrix, we keep track of the nonzero elements in a dictionary. The keys are tuples `(i,j)` giving the location of the element and the value assigned to this key gives the value of the nonzero element. If a tuple `(i,j)` does not appear in the dictionary, the matrix element at that location is assumed to be zero. When creating or modifying a sparse matrix, we don't want to store zero entries.

In the class `SparseMatrix` override some (but as few as possible) of the parent class `Matrix` to make sparse matrices work anywhere you can put a `Matrix`.

---

100 pts.

Problem 4. In this problem, we'll modify the code from the previous problem.

Look up the Python Iterator protocol. Add code to what you had in the previous problem to provide an iterator that returns tuples `((i,j),v)` for each element of the matrix. Of course `(i,j)` is the location of the element and `v` is its value. The iteratrion should go along the rows of the matrix, i.e., return the elements in the first row from left to right, then the elements of the second row, and so on.
