CS 370 Fall 2023: Assignment 4

Instructors: Justin Wan (email: justin.wan@uwaterloo.ca)
Leili Rafiee Sevyeri (email: leili.rafiee.sevyeri@uwaterloo.ca)

Due December 4, Monday, at 10:00pm EDT

Submit all components of your solutions (written/analytical work, code/scripts, figures, plots, output, etc.) to Crowdmark in PDF form for each question (multiple pages are allowed). Note that you may resubmit as often as necessary until the due date - only the final submission will be marked.

You must also separately submit a single zip file containing any and all code/scripts you write to the Assignment 4 DropBox on LEARN, in runnable format (and if necessary, any data needed to reproduce your results).

Note: Only the solutions submitted to Crowdmark will be marked. If you submit solutions to Learn only, you will receive 0 mark. There is no exception whatsoever.

You have a number of options of how to prepare your solutions. You can typeset your solutions in a word-processing application (MS Word, LaTeX, etc.), you can write on a tablet computer, or you can write on paper and take photos. **It is your responsibility to ensure that your submission is sufficiently legible. This is particularly important if you are submitting photos of handwritten pages.** TAs have the right to take marks off for illegible answers.

Note that your solutions will be judged not only for correctness but also for the quality of your presentation and explanations.

1. **(10 marks)** Gaussian Elimination

   (a) (6 marks) By hand, use Gaussian elimination with row pivoting to find the $PA = LU$ factorization for the following matrix

   $$A = \begin{bmatrix} 2 & 3 & 0 \\ -2 & -7 & 8 \\ 4 & -4 & 23 \end{bmatrix}$$

   where L is unit lower triangular, U is upper triangular, and P is a permutation matrix. (You can use a calculator for individual calculations if you like.)

   (b) (4 marks) By hand, use the factorization you found above to solve for x in the linear system $Ax = b$ where

   $$b = \begin{bmatrix} -3 \\ 15 \\ 30 \end{bmatrix}.$$

2. **(6 marks)** LU Factorization

   Suppose a square $N \times N$ nonsingular matrix A has already been factored as

   $$A = LU,$$

where L is unit-diagonal and lower triangular, and U is upper triangular. Consider the system

$$A^2 x = b,$$

where b is a given vector and x is an unknown vector.

(a) (3 marks) Show how to use the L and U factors of A to solve the linear system *efficiently.*

(b) (2 marks) How many flops does your algorithm use to compute the solution, x? (Include the flops of the initial LU factorization of A.)

(c) (1 mark) Show that the number of multiplications to form A^2 is higher than the flop count for computing x in part (b).

3. **(7 marks)** Application of LU Factorization

   Let A be an $N \times N$ matrix and you have already computed the L, U factors of A; i.e. $A = LU$ where L is unit-diagonal and lower triangular, and U is upper triangular.

   Later on, some entries of A were found to be wrong. The new matrix can be written as $A + pq^T$ where p and q are $N \times 1$ vectors. It is known that

$$(A + pq^T)^{-1} = A^{-1} - A^{-1}p(1 + q^T A^{-1} p)^{-1} q^T A^{-1}.$$

   (a) (1 mark) Consider the linear system:

$$(A + pq^T)x = b.$$

   Using the above formula, write down the formula for computing the solution x.

   (b) (6 marks) Describe an efficient algorithm to compute x based on the formula in part (a). You should use the L and U factors of A. If you compute the L and U factors of $A + pq^T$, you will receive 0 mark for this part. Also, do not compute A^{-1} explicitly. You will also receive 0 mark if you do. Explain where and how many (if any) vector dot products, matrix-vector multiplies, forward and backward solves are needed. Perform a complexity analysis. The flop counts for your efficient algorithm should be given by

$$\text{Flops} = 4N^2 + O(N),$$

   in which you should ignore the flops for the LU factorization of A.

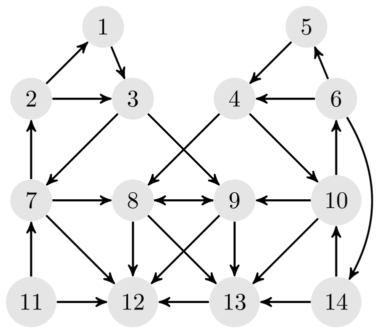
4. (**20 marks**) Implementing Google PageRank

   (a) (5 marks) Write a Python function

```
PageRank(G, alpha)
```

   which determines the pagerank for a network using the basic PageRank algorithm described in class. The inputs are the adjacency matrix `G` given as a standard 2D Numpy array, representing the directed graph of the network, and the scalar weight `alpha`. (As usual, the adjacency matrix is defined such that $G(i, j) = 1$ if there is a link from page j pointing to page i, otherwise $G(i, j) = 0$. You do not need to remove any self-links that may be present in the input graph G.) The function should output a tuple `[p, it]` consisting of the vector `p` of pagerank scores, and integer `it` representing the number of steps of pagerank required for the computation to finish. Use a tolerance value of 10^{-8}. For this part, you can implement the basic PageRank method in a naive/straightforward fashion; for example, explicitly forming the (dense) Google matrix is acceptable for this part.

(b) (5 marks) The following figure shows a small directed graph representing a set of 14 web pages.



Write Jupyter/Python code that constructs the adjacency matrix G and computes the pagerank vector for the web shown above using your PageRank function with $\alpha = 0.9$.

Print the vector of final pagerank scores (in ascending order of the node numbers).

List the indices of the pages in descending order of importance (from most to least), according to your results.

Use the matplotlib `spy` command to graph the sparsity (nonzero) pattern of the adjacency matrix G. Use the matplotlib `bar` command to plot the pagerank scores as a bar graph. Title your graphs appropriately.

(c) (6 marks) In order to efficiently handle larger network graphs, write a new function

`PageRankSparse(Gcsr, alpha)`

This new function should take in an adjacency matrix `Gcsr` which is our usual adjacency matrix but stored in an underlying sparse matrix representation known as compressed sparse row, or CSR. You may wish to refer to `https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html`. (Fortunately though, SciPy largely hides the underlying internal data structure details, so we essentially do not need to be concerned with them.) Your new function should also accept the same parameter `alpha` as before, and likewise return a tuple `[p, it]` consisting of the pagerank score vector `p` and iteration count `it`.

This function must *efficiently* compute the update for the pagerank vector by taking advantage of the sparsity of `Gcsr` and the inherent structure of the Google matrix, as discussed in class. Avoid creating any full (i.e., dense) matrices and do not explicitly form the Google matrix. The main PageRank iteration should consist of only a single outer loop structure (i.e., do not implement matrix or vector multiplication manually). Make use of SciPy's built in sparse matrix multiplication functionality.

A few tips: Sparse matrix multiplication can be performed in Python using the `dot` function. That is, if `A` is a CSR matrix and `b` is a (standard, dense) numpy vector, you can compute their matrix-vector product as `A.dot(b)`. `dot` can also be used for computing dot products between two vectors, `a.dot(b)` or sparse matrix-matrix products `A.dot(B)`. The `sum` function may be useful for computing column sums of `G`, i.e., outdegrees of nodes.

You can convert a dense matrix `G` to a CSR matrix using `Gcsr = sparse.csr_matrix(G)`. This may be useful for debugging your sparse method, by comparing against the results of your earlier dense implementation.

Refer to online Numpy/SciPy documentation for further details of their sparse matrix support.

(d) (4 marks) Write Python code to apply your efficient pagerank implementation to the internet graph data in the file `bbc.mat` provided with the assignment, again using $\alpha = 0.9$. This data represents a network of 500 webpages, originally generated starting from www.bbc.co.uk. The data consists of a sparse matrix G and a list of url's U. These two pieces of data can be loaded as follows:

```
import scipy.io
data = scipy.io.loadmat('bbc.mat')
Gcsr = data['G']
Gcsr = Gcsr.transpose() #data uses the reverse adjacency matrix convention.
U = data['U']}
```

Once again, use the `spy` command to graph the sparsity (nonzero) pattern of this adjacency matrix, titling the graph appropriately.

Using the results of running your efficient page rank routine on this data, your code should print the top 20 urls in descending order of importance. You may find the function `argsort` useful to determine the indexing that would sort the pagerank vector.