Questions may continue on the back. Type your text. You may write math formulas by hand. What we cannot read, we will not grade.

You may talk about this assignment with others, but do not write down anything while you talk. After that, do the assignment alone. What you hand in must be your work only.

Hand in your solution as a single, stapled paper document at the beginning of class on the due date.

1. Enter the name of a famous building you know well in Google images. Avoid buildings in Rome or Paris. Answer the questions below about the results that come back. Be brief but clear, and use full, grammatical English sentences.

   (a) State your exact query, and explain what building you are hoping to find through that (what is a common name for the building in English, where is the building, what is it: a church, a city hall, ...)

   (b) Give an approximate count of the images returned on the first page of the Google results. Just count images on a few of the rows and compute their average, then multiply by the number of rows. No need to be precise.

   (c) What fraction (as a percent) of the first 100 or so images contain, in your best judgement, photographs of at least some part of the outside of the building you were looking for? Let us call these the “valid” images.

   A drawing or a print of the building is not a valid image. Photographs of details of it (a bell, a statue, ...) should be included in your count if they are visible from outside the building.

   (d) Do you expect the answer to the previous question to be about the same for all queries of this type, regardless of which building you pick? Give a couple of reasons for your answer.

   (e) Describe very briefly a few of the images that are not “valid” and were returned in response to your query. What were they images of?

   (f) Pick one of the most surprising invalid results, and visit the web page that is linked to it. What did the image portray, and why was is part of the query results?

   (g) Are most of the “valid” images taken under similar weather conditions? Try to explain why or why not.

   (h) Ditto for time of day.

   (i) What difficulties do you expect to arise when applying the 3D reconstruction techniques described in the paper *Building Rome in a day* to images taken at different times of the day or the year, or in different weather? Explain in a couple of sentences.

2. This problem invites you to do a simple image coding exercise. The exercise requires almost no knowledge of images. Its intent is to encourage you to familiarize you with MATLAB, in preparation for further homework in this course.

   (a) What computer are you using for this problem? Specifically, what hardware, what operating system, including version? Is it your own computer, or if not where did you access it?

   (b) Download the image at http://tinyurl.com/dukeFlowers, convert it to black and white, and hand in the bar plot of a normalized histogram of the resulting gray-value image with bins centered at 0, 1, ..., 255. Explanations follow.

   To convert to black-and-white, it is best to use the Help feature in MATLAB to find a function that does the conversion for you. If you don't find it, it is OK to add up the red, green, blue components at each pixel, divide by 3, and round the result to the nearest integer. If you do so, keep in mind that the sum of three bytes does not fit in a byte, so you may need to do some data type conversions.

   The required histogram is a vector of 256 values $h_0, ..., h_{255}$. The value $h_i$ is the fraction (a number between 0 and 1) of pixels in the image that have pixel value $i$, divided by the total number of pixels in the image. Make sure that your histogram has appropriate labels and tick marks on the axes. Again, perusing the MATLAB Help is the better way to go.

   The first image (other than ads) at http://tinyurl.com/histExample is an example of such a histogram (for a different image), except that this does not have tick marks on the axes.
3. The three columns \( \mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3 \) of the matrix

\[
A = \begin{bmatrix}
\mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3 \\
1 & 3 & 0 \\
0 & x & 1 \\
1 & 1 & -2
\end{bmatrix}
\]

are linearly dependent on each other if there exist three numbers \( c_1, c_2, c_3 \), not all zero, such that

\[
c_1 \mathbf{u}_1 + c_2 \mathbf{u}_2 + c_3 \mathbf{u}_3 = 0.
\]  

(1)

The three vectors are linearly independent otherwise.

(a) Find a value of \( x \) that makes the vectors \( \mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3 \) linearly dependent on each other. You could do this by inspection, by trial and error, or by applying the definition of dependence and solving for \( x \) and the coefficients \( c_1, c_2, c_3 \). Just give the value of \( x \).

(b) Are there other values of \( x \) that achieve the same effect? Give another one, or explain precisely why there are no others. [Hint: spell out equation (1) and draw consequences about \( x \).]

(c) Find a value of \( x \) that makes the vectors \( \mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3 \) linearly independent of each other.

(d) Are there other values of \( x \) that achieve the same effect? Give another one, or explain precisely why there are no others.

4. The column rank of a matrix is the number of its linearly independent columns. The row rank of a matrix is the number of its linearly independent rows. Interestingly, the row rank of a matrix is always equal to its column rank, so we can remove the qualifiers “column” and “row” and just talk about the rank of a matrix. For instance, the \( 4 \times 3 \) matrix

\[
B = \begin{bmatrix}
1 & 3 & 0 \\
0 & 1 & 1 \\
2 & 5 & -1 \\
1 & 2 & -1
\end{bmatrix}
\]

has rank 2. The fact that row rank and column rank are the same number can be proven by transforming the given matrix with linear operations that are known not to change its rank and turn it into a matrix \( D \) for which both the row rank and the column rank are easy to determine by inspection. The row rank of \( D \) will turn out to be the same as the column rank of \( D \), regardless of the matrix you start from. We will perform this transformation with the following four types of operations:

- \( X_c(M, j) \) is the result of exchanging column 1 and column \( j \) of the matrix \( M \) with each other. For instance,

\[
X_c(B, 2) = \begin{bmatrix}
3 & 1 & 0 \\
1 & 0 & 1 \\
5 & 2 & -1 \\
2 & 1 & -1
\end{bmatrix}.
\]

- \( X_r(M, i) \) is the result of exchanging row 1 and row \( i \) of the matrix \( M \) with each other. For instance,

\[
X_r(B, 3) = \begin{bmatrix}
2 & 5 & -1 \\
0 & 1 & 1 \\
1 & 3 & 0 \\
1 & 2 & -1
\end{bmatrix}.
\]

- The first entry \( m_{11} \) of \( M \) is called the pivot. If the pivot is different from zero, let \( c_1 \) and \( c_j \) be the first and \( j \)-th column of \( M \), respectively. Then, \( Z_c(M, j) \) replaces column \( j \) of \( M \) with

\[
c_j' = c_j - \frac{m_{1j}}{m_{11}} c_1.
\]

For instance,

\[
Z_c(B, 2) = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 1 \\
2 & -1 & -1 \\
1 & -1 & -1
\end{bmatrix}.
\]

This operation always replaces entry \( m_{1j} \) with zero.
Assume that the pivot $m_{11}$ is different from zero, and let $r_1^T$ and $r_i^T$ be the first and $i$-th row of $M$, respectively. Then, $Z_c(M, i)$ replaces row $i$ of $M$ with

$$r_i^T = r_i^T - \frac{m_{i1}r_1^T}{m_{11}}.$$  

For instance,

$$Z_c(B, 3) = \begin{bmatrix}
1 & 3 & 0 \\
0 & 1 & 1 \\
0 & -1 & -1 \\
1 & 2 & -1
\end{bmatrix}.$$  

This operation always replaces entry $m_{i1}$ with zero.

None of these four operations changes the row rank or the column rank of the matrix it is applied to. This is because the operations $X_c$ and $X_r$ merely switch rows or columns around, so the number of linearly independent rows or columns does not change. Also, $Z_c$ does not change the space spanned by the columns of $M$, because if a vector $v$ can be written as

$$v = ac_1 + bc_j$$  

before the column exchange, then it can be written as

$$v = \left(a + \frac{m_{1j}}{m_{11}}\right)c_1 + bc'_j$$  

after the exchange (you may want to verify this).

Thus, any sequence of the four operations $X_c$, $X_r$, $Z_c$, $Z_r$ transforms a matrix without changing its row rank or column rank.

By using $X_c$ and $X_r$ once each, one can move the largest entry of an $m \times n$ matrix $M$ (that is, the entry with the biggest magnitude or absolute value) to position $(1, 1)$ in the matrix. Assuming that $M$ is not all zero, the first entry in the transformed matrix is now nonzero. One can then use $Z_c$ at most $n - 1$ times to zero all the entries in the first row of $M$ except the first, and use $Z_r$ at most $m - 1$ times to zero all the entries in the first column of $M$ except the first.

Thus, $M$ can be reduced to a matrix of the following form:

$$\begin{bmatrix}
* & 0 & \cdots & 0 \\
0 & & & \\
\vdots & & & M' \\
0 & & & 
\end{bmatrix}$$  

where the asterisk represents some nonzero number and $M'$ has one fewer row and one fewer column than $M$.

This procedure can be repeated on $M'$ to obtain a matrix of the following form:

$$\begin{bmatrix}
* & 0 & 0 & \cdots & 0 \\
0 & * & 0 & \cdots & 0 \\
0 & 0 & \cdots & M'' \\
\vdots & \vdots & \vdots & & \\
0 & 0 & & & 
\end{bmatrix}.$$  

One has to stop when the matrix in the lower-right corner is either empty (no rows or columns) or all zero, because then the pivot is zero and operations $Z_c$, $Z_r$ cannot be performed. If $r$ is the number of repetitions performed, one ends up with a diagonal matrix of the following form:

$$D = \begin{bmatrix}
d_1 & 0 & \cdots & 0 \\
0 & \ddots & & \\
\vdots & & \ddots & \\
0 & & \cdots & d_r
\end{bmatrix}$$  

(with or without a block of zeros in the lower-right corner). By construction, $D$ and $M$ have the same row rank and the same column rank. However, the column rank of $D$ is obviously $r$, because its first $r$ columns are linearly independent (even orthogonal) and its last $n - r$ columns are zero. By an analogous reasoning about the rows of $D$, the row rank of $D$ is $r$ as well.

Since we made no assumption about $M$, this argument proves that the row rank and the column rank of any matrix are the same number. So from now on we can just talk about rank, without specifying whether we mean the row rank or the column rank.
In addition, the procedure above can be used to find the rank of a matrix. This exercise guides you to do so. It also leads you to find out that the procedure thus found is numerically unsafe. Because of this, the MATLAB function `rank` computes the rank of a matrix in a very different way, which we will see in class soon.

If you are unfamiliar with MATLAB, you may want to look at the MATLAB Programming Tips at the end of this problem set.

(a) Write four instructions that implement the four operations $X_c(M, j)$, $X_r(M, i)$, $Z_c(M, j)$, $Z_r(M, i)$ on a matrix $M$. Use comments to tell which instruction implements which operation.

(b) Write a MATLAB function with header

```matlab
function [M nonzero] = reduce(M)
```

that takes any matrix $M$ with real entries and does the following: Let $m_{ij}$ be an entry of $M$ with greatest magnitude. If $m_{ij}$ is zero, do nothing and return $M$ itself. Otherwise, use the four operations $X_c$, $X_r$, $Z_c$, $Z_r$ repeatedly to first move $m_{ij}$ to row 1 and column 1 (the first entry of the matrix), and then zero all the other entries in row 1 and in column 1.

The function also returns a boolean variable `nonzero` that is true if the pivot is nonzero and false otherwise.

(c) Write a MATLAB function with header

```matlab
function [r M] = myrank(M)
```

that repeatedly calls `reduce` to transform the input matrix $M$ into a diagonal matrix, and then determines the rank $r$ of $M$ as explained above. The second output argument $M$ is the diagonal matrix.

Important: Even if you do everything right by these instructions, your function may return wrong rank values in $r$. Read on.

(d) Write down exactly MATLAB’s response to the calls

```matlab
[mr, D] = myrank(B)
r = rank(B)
```

where $B$ is the $4 \times 3$ matrix $B$ given in this problem. Do not panic if $mr$ and $r$ are different.

(e) The rank of $B$ is 2. If you got $mr = 3$ and you did everything by the instructions, then you encountered numerical problems. Just to make sure, $D$ should be close to the matrix

$$D = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 1.2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$ 

If you obtained something wildly different for $D$, please go back and review your previous answers. If you obtained this matrix exactly and $mr = 2$, then you probably already know about possible numerical problems, and you prevented them from happening in your code.

Write down MATLAB’s response to the call

```matlab
D(3, 3)
```

when performed just after the two calls in the previous question.

(f) The issue here concerns statements like the following, which you may have used (with $x$ replaced by something else) in your code to check whether some quantity is zero or not:

```matlab
if x == 0
```
or

```matlab
if x ~= 0
```
Keep in mind that statements like this may also be hidden inside others, for instance, in the MATLAB function `nnz` that counts how many entries in a matrix are nonzero.

The value of $x$ is the result of some other previous computation, such as one or more applications of the operators $Z_c$ or $Z_r$ (the operators $X_c$ and $X_r$ do not modify any numerical value, they just move values around). These computations are performed in finite arithmetic, and finite arithmetic produces numerical errors. For instance, double-precision floating-point numbers represent the value $1/3$ by $0.333333333333333$ (with 15 threes), and this is a little less than $1/3$. So if the exact value of $x$ is supposed to be zero, its actual value may be slightly different as a result of accumulated errors of this type.

A partial patch for this problem is to replace

```matlab
    if x == 0
```

with

```matlab
    if abs(x) < small
```

where `small` is some small positive number (and an analogous expression for $x \neq 0$). The smallest number MATLAB can represent in double-precision arithmetic is what it returns if you type `eps` to the prompt, a value close to $2 \times 10^{-16}$. So we can make `small` equal to a value that is much larger than `eps`, but much smaller than values we care about. For instance,

```matlab
    small = sqrt(eps)
```

is close to $1.5 \times 10^{-8}$, and tries to split the difference evenly in some sense, by reserving half of the digits in a double-precision floating-point number for numerical error and the other half for useful information.

This is only a patch, because the “values we care about” are hard to define in general. We will see a completely satisfactory solution to our problem in class, once we study MATLAB’s way to compute the rank of a matrix.

Please go back and apply the patch to your code. You may have to change an `if` statement in `reduce` and also how you count the number of nonzero elements (what does nonzero mean, given numerical errors?) in `myrank`. Do not hand in the new code listings. Just state what instructions you changed and how.

(g) Hand in MATLAB’s new response to the call

```matlab
    [mr, D] = myrank(B)
```

For your own peace of mind, you may also want to test your code with the script `testRank.m` available on the class homework page. Do not hand in the results of doing so.

**Matlab Programming Tips**

- The following code snippet finds the row index $i$ and column index $j$ of a largest entry of matrix $A$:

  ```matlab
  % Find index k of max entry within the column version A(:) of A
  [~, k] = max(abs(A(:)));
  [i, j] = ind2sub(size(A), k); % Convert k to row, column subscripts
  ```

- To access rows $i$ and $j$ in matrix $A$ use the following notation:

  ```matlab
  A([i j], :)
  ```

  and similarly for columns.

- To access the block starting at row $r$ and column $c$ in matrix $A$ use the following notation:

  ```matlab
  A(r:end, c:end)
  ```

- To access the diagonal of a matrix in the right-hand side of an expression, you can use the `diag` built-in function. However, you need to be a bit careful, because `diag` behaves differently for vectors and for matrices. In some cases, we want the “matrix” behavior even when the matrix happens to be a row vector or a column vector. The following code puts the diagonal of $A$ (regardless of its size) into the column vector $d$:

  ```matlab
  if isvector(A)
    d = A(1);
  else
    d = diag(A);
  end
  ```