University of Costa Rica **Juan Ignacio Padilla B.**
School of Mathematics ID: B55272
MA-501 Numerical Analysis Prof. Juan Gabriel Calvo
Homework 3 January 29, 2026

Homework 3

Problem 1.

(Vandermonde Matrix) Consider the Lagrange interpolation problem given by: Find a polynomial p of degree at most n whose graph includes the points $\{(x_i, y_i)\}_{i=0}^n$ (which we know has a unique solution). In this exercise we will formulate and solve this problem using a system of equations. Write

$$p(x) = \sum_{k=0}^{n} a_k x^k$$

The interpolation problem is equivalent to finding the coefficients $\{a_k\}$ that satisfy the $n+1$ equations

$$p(x_i) = \sum_{k=0}^{n} a_k x_i^k = y_i, \quad \forall i = 0, \dots n,$$

which can be written in matrix form as $A\boldsymbol{x} = \boldsymbol{b}$, where

$$\boldsymbol{x} = \begin{bmatrix} a_0 \\ \vdots \\ a_n \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} y_0 \\ \vdots \\ y_n \end{bmatrix}$$

and A is the Vandermonde matrix

$$A = \begin{bmatrix} 1 & x_0 & \dots & x_0^n \\ 1 & x_1 & \dots & x_1^n \\ \vdots & & \ddots & \vdots \\ 1 & x_n & \dots & x_n^n \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}$$

In this exercise we will use equidistant nodes in the interval $[-1, 1]$, that is,

$$x_j = -1 + 2\frac{i}{n} \quad (i = 0, \dots, n).$$

a) Prove that A is invertible

Solution: Consider

$$A_n = \begin{vmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-2} & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-2} & x_2^{n-1} \\ 1 & x_3 & x_3^2 & \dots & x_3^{n-2} & x_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-2} & x_{n-1}^{n-1} \\ 1 & x_n & x_n^2 & \dots & x_n^{n-2} & x_n^{n-1} \end{vmatrix}$$

We can first consider A_n as a function of x_n. In particular, $A_n(x_n)$ is a polynomial of degree $n-1$, since expanding the last row we have

$$A_n(x_n) = \begin{vmatrix} x_1 & x_1^2 & \dots & x_1^{n-2} & x_1^{n-1} \\ x_2 & x_2^2 & \dots & x_2^{n-2} & x_2^{n-1} \\ x_3 & x_3^2 & \dots & x_3^{n-2} & x_3^{n-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-2} & x_{n-1}^{n-1} \end{vmatrix} + x_n \begin{vmatrix} 1 & x_1^2 & \dots & x_1^{n-2} & x_1^{n-1} \\ 1 & x_2^2 & \dots & x_2^{n-2} & x_2^{n-1} \\ 1 & x_3^2 & \dots & x_3^{n-2} & x_3^{n-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_{n-1}^2 & \dots & x_{n-1}^{n-2} & x_{n-1}^{n-1} \end{vmatrix} + \dots$$
$$+ x_n^{n-1} \begin{vmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-2} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-2} \\ 1 & x_3 & x_3^2 & \dots & x_3^{n-2} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-2} \end{vmatrix}$$

Observe that if we evaluate $A_n(x)$ at any of $x_1, x_2, \dots, x_n$ we get that all determinants will have a repeated row, so $A_n(x_i) = 0$ for $i = 1, \dots, n-1$. This tells us that

$$\det(A) = A_{n-1}(x_n - x_1)(x_n - x_2)\cdots(x_n - x_{n-1})$$

Since by the previous expansion, the leading coefficient is A_{n-1}. We can then repeat this argument, considering the polynomial $A_j(x_j)$ successively for $x_{n-1}, \dots, x_1$, to obtain

$$\begin{aligned} \det(A) &= \prod_{1 \le i < n} (x_n - x_i) A_{n-1} \\ &= \prod_{1 \le i < n} (x_n - x_i) \prod_{1 \le i < n-1} (x_{n-1} - x_i) A_{n-2} \\ &= \cdots \\ &= \prod_{1 \le i < j \le n} (x_j - x_i) \neq 0 \end{aligned}$$

Which proves that A is invertible.

b) For $n = 40$, construct the matrix A. Calculate the condition number of A, its rank and its determinant. Is the matrix invertible? Suggestion: you can use the commands `vander, rank, det, cond` in `MATLAB`.

Solution: When generating the matrix A for the equidistant nodes x_j, with $n = 40$, we numerically obtain

- **Condition number:** 3.4987×10^{17}
- **Rank:** 36
- **Determinant:** -3.6836×10^{-243}

Numerically, the Vandermonde matrix in this case will not be invertible.

c) To solve the system of equations $A\boldsymbol{x} = \boldsymbol{b}$, it suffices to execute the command $\tilde{\boldsymbol{x}} =$ `A \b`. Consider $f(x) = (1+25x^2)^{-1}$. Estimate $||f(\texttt{xx}) - p(\texttt{xx})||_\infty$ where xx = `linspace(-1,1,1e3)`

Solution: The polynomial with coefficients $a_i = \{(A^{-1})b\}_{i+1} (i = 0, \dots, 40)$ was considered, which was evaluated numerically. When calculating the error we obtained

$$||f(\texttt{xx}) - p(\texttt{xx})||_\infty \approx 1.9270 \times 10^5$$

Which is enormous. Solving the interpolation problem by "brute force" will result in terrible results for matrices with high condition number.

d) Calculate the singular value decomposition of A using the command

$$\texttt{[U,S,V]} = \texttt{svd}(A).$$

In this way,

$$\tilde{x} = VS^{-1}U^T b.$$

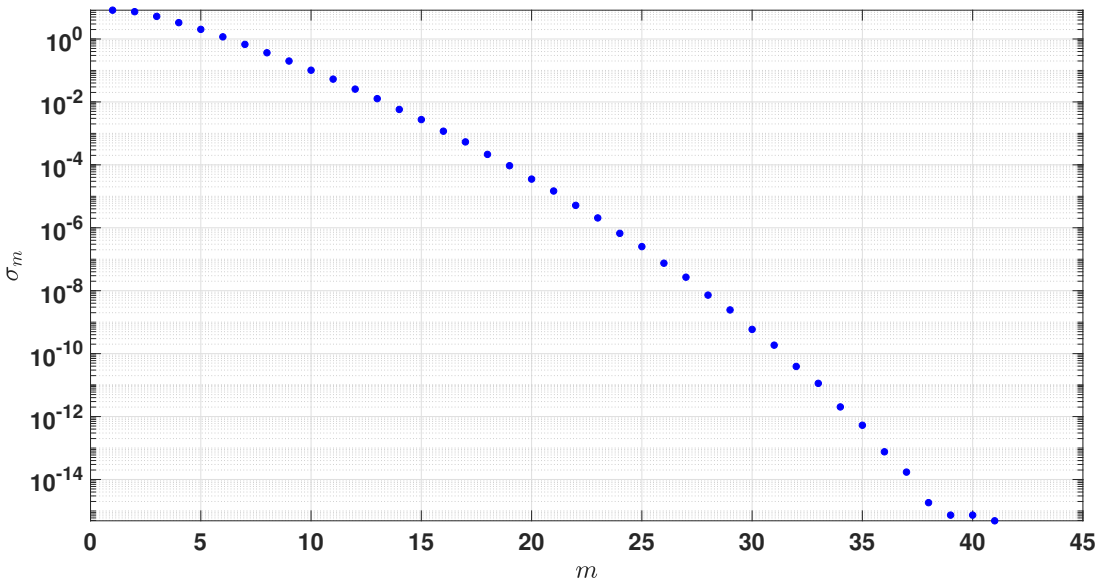
Estimate again $||f(\texttt{xx}) - p(\texttt{xx})||_\infty$. Is there any improvement in the approximation of the solution? Plot the singular values of A on a semilogarithmic scale. Comment on the behavior. What is the numerical rank of the matrix? Suggestion: the function `diag` may be useful.

Solution: This time we obtain

$$||f(\texttt{xx}) - p(\texttt{xx})||_\infty \approx 7.2127 \times 10^3$$

Which is better, but is still a huge error. The singular values of A are shown in graph 1.

Gráfica 1



It can be seen that the last values approach machine precision. Also, the numerical rank of S is still

Rank: 36.

e) Consider a matrix with singular value decomposition $A = USV^T$. The *pseudoinverse* of A is defined as

$$A^+ = VS^+U^T$$

where

$$S^+ = \begin{bmatrix} 1/\sigma_1 & & & \\ & 1/\sigma_2 & & \\ & & \ddots & \\ & & & 1/\sigma_p \end{bmatrix}$$

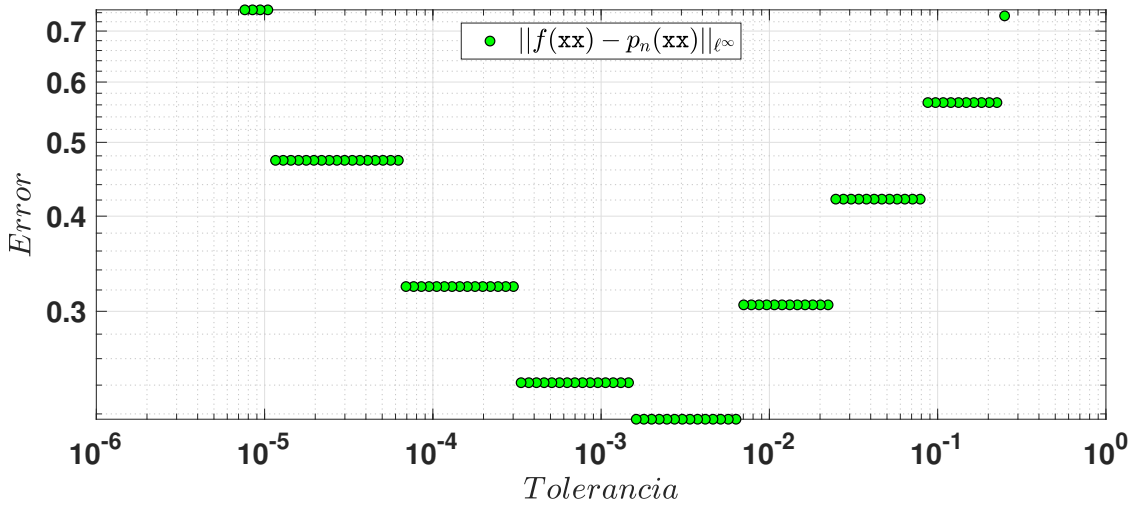
is the diagonal matrix obtained by inverting the positive singular values of S. Define a vector of tolerances `tol = 10.^(-linspace(2,17))`. For each entry `j` of `tol`, define S_j as the matrix obtained from S, replacing the singular values smaller than `tol(j)`$*\sigma_1$ by zero. Calculate

$$\texttt{x\_svd} = (VS_j^+U^T)b$$

and calculate the norm $||f(\texttt{xx}) - p(\texttt{xx})||_\infty$. Note that we are approximating the pseudoinverse with only some singular values of A. Plot the error as a function of the tolerance vector. Comment on your results and explain the behavior of the graph.

Solution: When applying the approximation using only some singular values of A, we obtain the error as a function of the chosen tolerance, and it is summarized in Graph 2

Gráfica 2



It can be observed that as the tolerance decreases, the error **grows rapidly**. The reason for this is the following: by minimizing the tolerance, more singular values are chosen, so the pseudoinverse of the matrix A numerically approaches the inverse of A. However, as we saw, when calculating A^{-1} numerically to solve systems, the results have very large errors. Therefore, by minimizing the tolerance too much, the error will grow. So, in this case, it is better to approximate the solution using fewer singular values. **Note:** The smallest errors in Graph 2 are approximately 0.01.

f) For a tolerance value that minimizes the error from the previous part, plot on the same graph $f(\texttt{xx})$, $p(\texttt{xx})$ and the interpolation nodes. How good is the approximation?

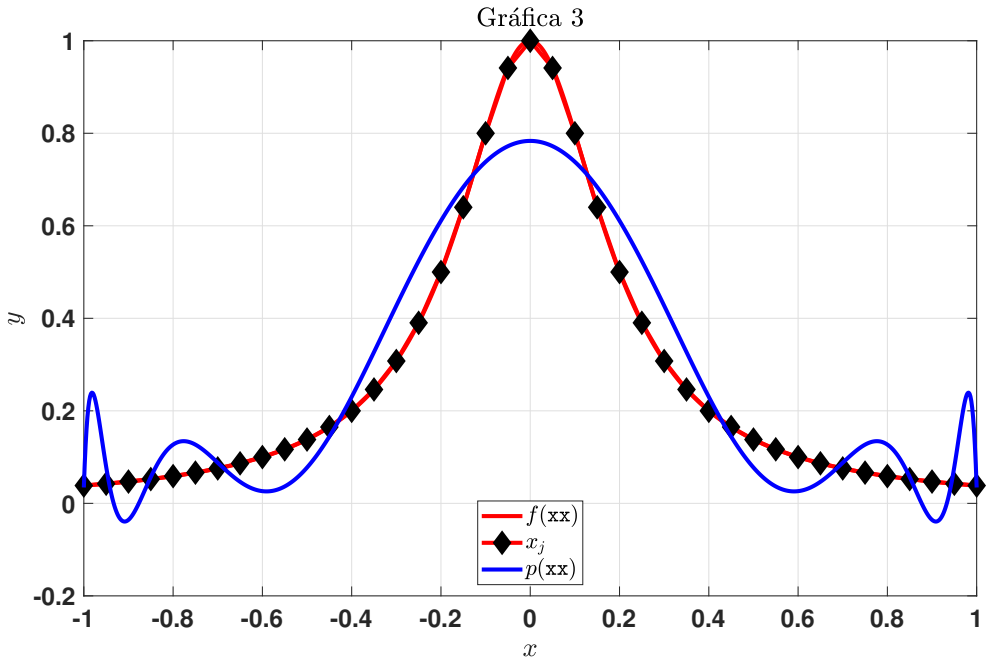
Solution: The tolerance value that minimizes the error from the previous part is $t \approx 0.0025$ (this was calculated in the attached script). When plotting f and $p(\texttt{xx})$ with 1000 equidistant nodes we obtain Graph 3

It is observed that the approximation is not very good; moreover, oscillations appear near the endpoints, so we conclude that this method exhibits the Runge phenomenon.

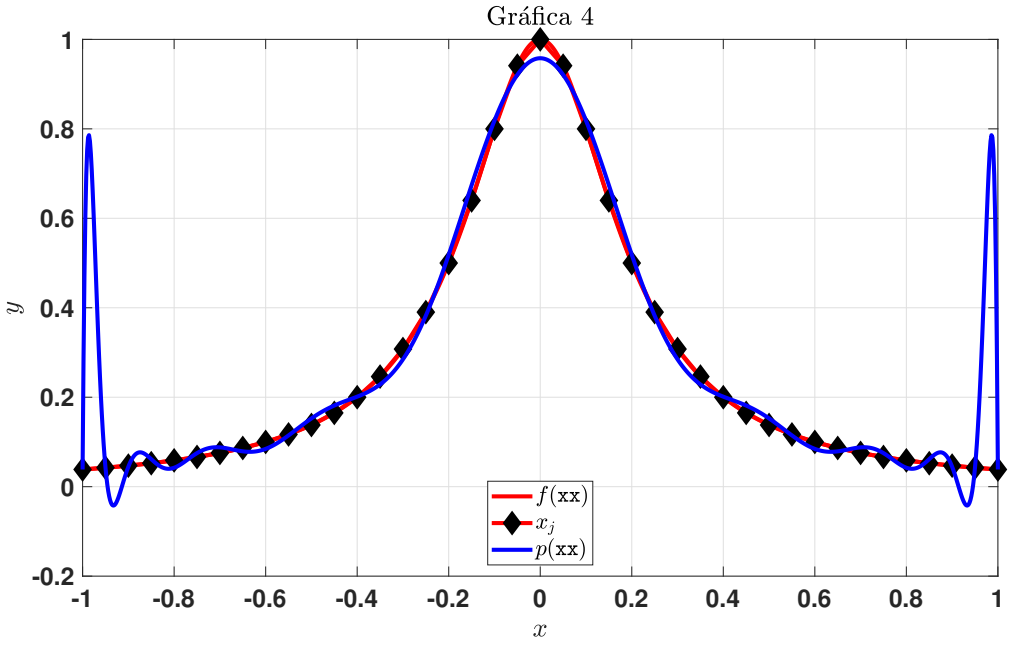
g) Now use `tol = 1e-5`. Plot on the same graph $f(\texttt{xx})$, $p(\texttt{xx})$, and the interpolation nodes. Is the Runge phenomenon observed? Explain why.

Solution: For the specified tolerance, we repeat the previous part, to obtain Graph 4.

Although the approximation appears to be better at first glance, in this case a greater Runge phenomenon occurs, which increases the error $||f(\texttt{xx}) - p(\texttt{xx})||_\infty$. The reason



for this is that, once again, by decreasing the tolerance, more singular values of A are chosen, and the final approximation of a_i has considerable errors, due to the high condition number. According to the infinity norm, this approximation is **worse** than the previous one.



Problem 2.

(Image Compression) In this exercise we use the singular value decomposition to compress images in a very basic way. An image of dimension $m \times n$ consists of mn pixels (the resolution refers to the number of pixels used). In a color image, each pixel contains three integer values $(r_p, g_p, b_p) \in [0, 255]^3$, corresponding to the red, green, and blue components. In this way, the image is stored as a matrix of size $m \times n \times 3$.

a) (Reading the image) Consider the image `'photo.bmp'`, provided. To read the image execute the command `RGB=imread('photo.jpg')` (*image read*). How many pixels does the image stored in `RGB` have? What type of entries does this matrix have? Suggestion: print the command `whos RGB` and comment on the result.

Solution: The matrix `RGB` was stored as a $3648 \times 2736 \times 3$ matrix, which stores the data type `uint8`, which corresponds to integers between 0 and 255.

b) (Converting the image to double format) To be able to use the SVD, convert the matrix `RGB` to double format using the command `RGB =im2double(RGB)`.

Solution: The action was executed in the script `Exercise2.m`.

c) (Compressing the image) Calculate the singular value decomposition of `RGB`$= USV^T$ (for each red, green, and blue color matrix). Calculate the rank 5 approximation given by

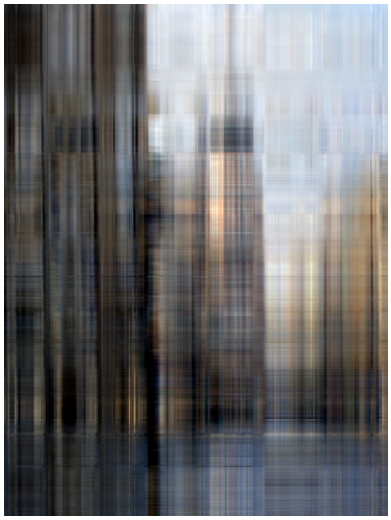
$$A_5 = \sum_{i=1}^{5} \sigma_i u_i v_i^T$$

In the same window (using the subplot command) show the original image, the image stored in A_5 and the error $|\texttt{RGB} - A_5|$. Comment on your results. Suggestion: the command `imshow(RGB)` displays the image given in `RGB`.

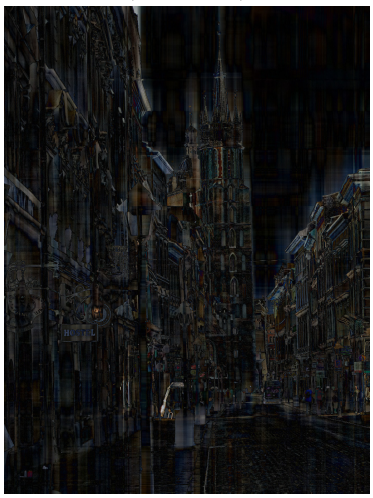
Solution: When executing the approximation $A5$, we obtain the following images 1.

RGB

A5



$|RGB - A5|$



The image produced by A_5 can be seen to be very distorted, although at least it preserves the general lighting and coloring, which is quite good for only taking 5 singular values out of 2736.

The image generated by the error captures the pixels that A_5 does not approximate correctly; most belong to regions of the photo with many color changes, something that is difficult to capture with few numerical entries.

d) For each matrix `RGB(:,:,i)` determine the number r_i of singular values σ_k such that $\sigma_k > 0.01\sigma_1$ and take $R_1 = \max r_i$. For this value of R_1 calculate the best rank R_1 approximation A_{R_1} of the matrix `RGB`. Repeat for $\sigma_k > 0.005\sigma_1$, denote this new rank R_2 and the respective approximation as A_{R_2}. Draw both approximations in the same window. Comment on the differences between both images. Suggestion: the command `diag(S)` extracts the elements of the diagonal of the matrix `S`.

Solution: The script that performs the instructions in `MATLAB` is attached. The comparison between the images generated by A_{R_1} and A_{R_2} is attached (page 8).

A_{R_1}

A_{R_2}

It can be observed that both images resemble the original photo. However, when slightly enlarging the image, it can be seen that A_{R_1} loses fine details, and appears pixelated under magnification. Meanwhile, A_{R_2} preserves more details, and only becomes pixelated if the image is enlarged significantly. The number of singular values for A_{R_1} and A_{R_2} are 75 and 172, respectively. (The images can be better viewed by running the script).

e) Finally save the image A_{R_2}, as files `.bmp` and `.jpg` using the command `imwrite(A_R2,'photo_comp.bmp')` and `imwrite(A_R2,'photo_comp.jpg')`. Compare the size of these two files with the original size of the initial photo. (Note that the `.jpg` format by default compresses the image and is optimized for that purpose).

Solution: The images were generated in `.jpg` and `.bmp` formats (they are attached in the files). The original size of the image was $29,242$ Kb. It can be observed that the size of the image in `.bmp` format is the same, from which we conclude that this image format simply counts the pixels (the size of the matrix) that was used, regardless of whether it has many zeros or is of much lower rank than the number of rows it has. Meanwhile, the size of the image in `.jpg` format is only 989 Kb, 3% of the original size. Clearly if memory efficiency is desired, the .jpg format is the right choice.

Problem 3.

(Stability of QR factorization) In the notes three algorithms for computing the QR factorization of a matrix $A \in \mathbb{C}^{m\times n}$ are discussed, which we present below:

Data: Matrix $A \in \mathbb{C}^{m\times n}$
Result: Unitary matrix Q and upper triangular R such that $A = QR$.
for $j = 1 : n$ **do**
  $\boldsymbol{v}_j = \boldsymbol{a}_j$;
  for $i = 1 : j-1$ **do**
    $r_{ij} = \boldsymbol{q}_i^* \boldsymbol{a}_j$;
    $\boldsymbol{v}_j = \boldsymbol{v}_j - r_{ij}\boldsymbol{q}_i$;
  end for
  $r_j j = ||\boldsymbol{v}_j||_2$;
  $\boldsymbol{q}_j = \boldsymbol{v}_j / r_{jj}$;
end for

Algorithm 1: Gram-Schmidt Orthogonalization (unstable).

Data: Matrix $A \in \mathbb{C}^{m\times n}$
Result: Unitary matrix Q and upper triangular R such that $A = QR$.
$V = A$;
for $j = 1 : n$ **do**
  $r_{ii} = ||\boldsymbol{v}_i||_2$;
  $\boldsymbol{q}_i = \boldsymbol{v}_i / r_{ii}$;
  for $j = i+1 : n$ **do**
    $r_{ij} = \boldsymbol{q}_i^* \boldsymbol{v}_j$;
    $\boldsymbol{v}_j = \boldsymbol{v}_j - r_{ij}\boldsymbol{q}_i$;
  end for
end for

Algorithm 2: Gram-Schmidt Orthogonalization (stable).

Data: Matrix $A \in \mathbb{C}^{m\times n}$
Result: Unitary matrix Q and upper triangular R such that $A = QR$.
for $j = 1 : n$ **do**
  $\boldsymbol{x} = A_{j:m,j}$
  $\boldsymbol{v}_j = \text{sign}(x_1)||\boldsymbol{x}||_2 \boldsymbol{e}_1 + \boldsymbol{x}$;
  $\boldsymbol{v}_j = \boldsymbol{v}_j \backslash ||\boldsymbol{v}_j||_2$;
  $A_{j:m,j:n} = A_{j:m,j:n} - 2\boldsymbol{v}_j(\boldsymbol{v}_j^* A_{j:m,j:n})$;
end for
$R = A$;
$Q = I$;
for $j = n : -1 : 1$ **do**
  $Q_{j:m,j:n} = Q_{j:m,j:n} - 2\boldsymbol{v}_j(\boldsymbol{v}_j^* A_{j:m,j:n})$
end for

Algorithm 3: Householder Triangularization

a) Write three functions that compute each of the algorithms.

Solution: The three functions `GS_unstable, GS_stable, Householder` are attached in the files.

b) Define the matrix A using the following lines:

```
m=80;
[U,~] = qr(randn(m));
[V,~] = qr(randn(m));
S=diag(2.^(-1:-1:-80));
A=U*S*V;
```

Solution: The matrix was defined in `MATLAB`.

c) Calculate the QR factorization of A using the three mentioned algorithms. For each one calculate $||A - QR||_2$ and $||Q'Q - I||_2$. Comment on your results. Which algorithm is better? Compare your results with the `MATLAB` command $[q, r] = \texttt{qr}(A)$.

Solution: The results are summarized in the following table

Table 1. Error in using algorithms for QR factorization

Algorithm	**Average error** $\|\|A - QR\|\|_2$	**Average error** $\|\|Q^*Q - I\|\|_2$
Unstable Gram-Schmidt	6.6472×10^{-17}	52.012
Stable Gram-Schmidt	6.61492×10^{-17}	0.99528
Householder	1.61738×10^{-16}	1.90284×10^{-15}
`qr`	1.573642×10^{-16}	1.98785×10^{-15}

Note: To calculate the average error, each script was run 5 times. It can be observed that the average factorization error is very low in all cases, being slightly better for the Gram-Schmidt methods. However, the first two methods do not produce a completely unitary matrix Q. The Q obtained from the first method is far from being unitary, while the one obtained from the second method still has a considerable error. Meanwhile, the Householder method and the software method produce very good approximations of unitary matrices. The `MATLAB` code does not allow examining the `qr` command, but it is presumed to coincide with Householder. In summary, the best method appears to be **Householder.**