32. (a) Write a program in MATLAB or Octave which takes as input parameters:
    - an $n \times m$ ($n \geq m$) matrix A (assumed to be of full rank m),
    - an $n \times 1$ vector b and
    - a number of steps

    and performs the **CGN iteration** for $Ax = b$.

    (b) For testing, compare the output of your algorithm with the builtin `A\b` operator on some arbitrary testcases with square matrices A.

    (c) What does the algorithm converge to if A is not square, i.e. if $n > m$?

33. Try the Arnoldi iteration (example 28) on random square matrices of increasing size, and measure its convergence properties, i.e.:
    - how many iterations does it need to converge (to the first k eigenvalues, for different values of k),
    - how much time does it need to converge and
    - how does the convergence speed differ between the different eigenvalues,

    and how they evolve with increasing size n (in particular, whether they're asymptotically linear, quadratic, cubic etc.). Summarize your results.
    Hint: The `rand(n,m)` function of MATLAB or Octave can be used to generate a random $n \times m$ matrix. In particular, `rand(n,n)` (or just `rand(n)`) generates a random square matrix and `rand(n,1)` a random vector.

34. Try the Lanczos iteration (example 30) on random symmetric matrices of increasing size, and measure its convergence properties and how they evolve with increasing size n. Summarize your results.
    Hint: You can make arbitrary random matrices symmetric by just adding `A+A'`. (Do not use products like `A'*A` because those adversely affect the condition of the matrix.)

35. Try the GMRES iteration (example 29) and the CGN iteration (example 32) on (the same) random square matrices of increasing size, and measure the convergence properties of each, i.e.:
    - how many iterations does it need to converge and
    - how much time does it need to converge,

    how they evolve with increasing size n and how they compare to each other. Summarize your results.

36. Try the CG iteration (example 31) on matrices of the form $A^\mathsf{T}A + I$, where A is a random $n \times n$ matrix with values between 0 and 1 (as produced by `rand(n)`) and I is the $n \times n$ identity (as produced by `eye(n)`), and measure its convergence properties and how they evolve with increasing n. Summarize your results.
    Note: $A^\mathsf{T}A$ is symmetric and positive semidefinite, thus $A^\mathsf{T}A + I$ is symmetric and positive definite. It is also much less ill-conditioned than $A^\mathsf{T}A$. (Note that using CG on $A^\mathsf{T}A$ is the same as using CGN on A with a different right hand side.)

37. Consider the symmetric, positive definite matrix

$$A = \begin{pmatrix} 9 & 1 & 2 & 15 \\ 1 & 16 & 0 & 1 \\ 2 & 0 & 16 & 2 \\ 15 & 1 & 2 & 34 \end{pmatrix}$$

and the vector $b = \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix}^\mathsf{T}$. Using MATLAB or Octave and your CG implementation from example 31:

(a) Compute the exact solution $\hat{x} = A^{-1}b$ (i.e. `A\b`) of $Ax = b$.

(b) How many CG iterations on A and b do you need until the solution x_{CG} is so close to $\hat{x}$ that $\|x_{\text{CG}} - \hat{x}\|_2 < \frac{1}{8}\|\hat{x}\|_2$?

(c) Let

$$L = \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 5 & 0 & 0 & 3 \end{pmatrix}$$

and compute $M = LL^\mathsf{T}$. What can you say about the large entries ($|a_{ij}| > 2$) of A and M?
Note: M is an example of a **preconditioner**. Approximations $M \approx A$ (so that $M^{-1}A \approx I$) are good preconditioners for linear systems of equations. And M has known triangular factors L and L^T which are both sparse, so products $M^{-1}u$ are easy to compute (by solving 2 sparse triangular systems of equations), and would remain cheap to compute even for larger matrices $M = LL^\mathsf{T}$ of similar structure.

(d) Compute $C = M^{-1}A$ and $d = M^{-1}b$. What is the exact solution of $Cx = d$?
Note: In practice, the product $M^{-1}A$ is usually not computed explicitly, because it is often expensive to compute for large matrices M and A. Instead, the CG algorithm is reformulated using the associative law for multiplications $Cu = (M^{-1}A)u = M^{-1}(Au)$, resulting in 2 multiplications: Au, which we have to do anyway, and $M^{-1}v$ (where $v = Au$), which is cheap to compute as explained in the previous note.

(e) How many CG iterations on C and d do you need until the solution x_{CG} is so close to $\hat{x}$ that $\|x_{\text{CG}} - \hat{x}\|_2 < \frac{1}{8}\|\hat{x}\|_2$?

(f) What can you conclude?