Fortran Homework Assignment 1

Solving Linear Systems with Gauss Elimination

Joris Tavernier, Pieter Appeltans & Emil Loevbak

October 12, 2018

1 Introduction

Solving linear equations is a computational kernel in many applications of scientific software, so it is important that these problems can be solved stably, accurately and efficiency. An example of this is the extensive work spent developing libraries such as LAPACK. In this first assignment you will implement code for solving systems of linear equations using LU factorization through Gauss elimination to get a feel for the issues that can occur when implementing such a routine in software.

2 Background information: Factoring with Gauss elimination

Gauss elimination is an algorithm for computing the LU-factorization of a matrix A, i.e., the matrices L and U so that L is lower triangular with ones on the diagonal, U is upper triangular and $A = LU$. This can be used to solve a linear system of the form

$$Ax = v,$$

with x the vector of unknowns by first solving a system of the form

$$Ly = v$$

for the unknown y. Then x can be computed by solving a system of the form

$$Ux = y.$$

Given that L and U are triangular matrices, these last two systems are easy to solve.[1]

Classical Gaussian elimination works by applying the following algorithm, where m is the dimension of A and I is the $m \times m$ identity matrix. Colons indicate ranges of indices, as is the case in Matlab code.

[1] Should these concepts be unfamiliar, a full descriptions of the intermediate steps can be found on Wikipedia: `https://en.wikipedia.org/wiki/LU_decomposition` and `https://en.wikipedia.org/wiki/Triangular_matrix#Forward_and_back_substitution`

```
1: procedure GAUSS
2:     U ← A, L ← I
3:     for k ← 1 ... m − 1 do
4:         for j ← k + 1 ... m do
5:             l_{j,k} ← u_{j,k} / u_{k,k}
6:             u_{j,k:m} ← u_{j,k:m} − l_{jk} u_{k,k:m}
```

Given that the diagonal elements of the matrix L are all equal to one, they do not have to be stored explicitly, meaning the resulting L and U can be stored in the memory that held A. This routine can thus be implemented in an efficient way, without using extra memory, by working directly in the memory of the matrix A.

For numerical stability reasons, it is often a good idea to include pivoting in Gauss elimination. This means that the matrix A is decomposed in such a way so that the expression

$$PA = LU$$

holds, with P a permutation matrix, meaning that each row and each column contains exactly one element one and for the rest zeros. This permutation of rows and columns is done so that the pivot (in the previous algorithm, the value $u_{k,k}$ is as large as possible. This can be done by applying the following modified algorithm:

```
1: procedure GAUSS_PIVOT
2:     U ← A, L ← I, P ← I
3:     for k ← 1 ... m − 1 do
4:         Select i ≥ k to maximize |u_{i,k}|
5:         u_{k,k:m} ↔ u_{i,k:m}                    ▷ Interchange two rows
6:         l_{k,1:k−1} ↔ l_{i,1:k−1}
7:         p_{k,:} ↔ p_{i,:}
8:         for j ← k + 1 ... m do
9:             l_{j,k} ← u_{j,k} / u_{k,k}
10:            u_{j,k:m} ← u_{j,k:m} − l_{jk} u_{k,k:m}
```

Although this algorithm uses a full matrix for P it is possible to store all of the necessary information in a vector of length m to reduce memory use.

The algorithms described here do not cover all possible pivoting techniques, but simply serve as an example for your implementations funther on. For a more detailed theoretical description and derivation of these algorithms, we refer to textbooks on numerical linear algebra, such as chapters 20 and 21 in [1].

3 Assignment

In this assignment you will implement code to solve linear equations using Gauss with, and without pivoting, as well as some helper routines for analysis of the results. You will then be expected to run some tests on your code to verify its correctness and discuss the issues that occur due to the implementation of the algorithms in floating

point arithmetic. To do this you should complete the following steps:

- Implement a module called `matrix_solve` which implements the subroutines `gauss(A, v, m)` and `gauss_pivot(A, v, m)`. In both cases the routine solves the system $Ax = v$, based on the inputs `A` and `v`, and stores the computed x in the vector `v` upon returning.

- Make sure the implementation can easily be adapted in terms of precision and motivate the method used to define the floating point precision. Make sure that this definition gives the same results on all of the Fortran compilers available in the computer class. (Hint: Think about defining a precision variable for use in your program. How should you do this?)

- Download the file `eigv_FC.o`, with `FC` the name of the compiler that you want to use, from Toledo. This file contains a subroutine `eig(A, lambdar, lambdai)` which computes the eigenvalues of the matrix `A` and returns them in the vectors `lambdar` and `lambdai`, representing respectively the real and imaginary parts. To link this file with your software you will need to add the flag `-llapack` to the end of your compilation command, e.g., `gfortran -o main main.o eigv_gfortran.o -llapack`. Using this functionality, implement a function `cond(A, m)` which calculates the condition number of a given matrix `A`.[2] Also implement a function `sum_ev(A, m)` which calculates the sum of the real parts of the eigenvalues of the matrix `A`.

- On Toledo you can find a zip file containing some test matrices and vectors, as well as a file `inout.f90` which can read these files. Using the routines you have developed so far, write a program that does the following:

  - Reads each matrix into the program
  - Solves the system with both algorithms
  - Prints the relative and absolute 2-norm of the backward error
  - Prints the 2-norm of the forward error
  - Prints the eigenvalues of the matrix
  - Prints the condition number of the matrix
  - Prints the sum of the eigenvalues

  Do this for both single and double precision and discuss the results: Where do both routines behave the same? Where do they fail? Is there a noticeable difference in accuracy in both algorithms? Why? Do you notice any differences between single and double precision? Do all of the results that you print make sense?

[2] If you need a reminder on how to do this, see `https://en.wikipedia.org/wiki/Condition_number`.

4 Practical information

The deadline for submission on Toledo is **Wednesday the 24th of October at 14:00**. This deadline is strict! Do not wait to the last minute to submit, as we will not accept technical issues as an excuse for late submissions.

We expect you to submit your code together with a PDF which, for each matrix, contains the output generated by your code, together with a discussion of the results achieved by the two algorithms in both single and double precision. Please also mention the total amount of time spent on the assignment in your report.[3] This should be submitted in a ZIP file with the name `lastname_firstname_studentnumber.zip`, i.e., if your name is John Smith and your student number is r0123456, your file should be called `smith_john_r0123456.zip`. Please name your main program file `main.f90`, and include the commands for compiling, linking and running of your program in comments at the top of the file. If this is more than a couple instructions, you can include a Makefile as an alternative.

Some specific points to pay attention to:

- In your discussion, we expect you to be specific. This means that besides from mentioning concepts such as stability, condition and so on, you should refer to your code (e.g. a specific operation in a given line of code) to explain why an algorithm does not work.
- This assignment is an opportunity to demonstrate that you have understood the concepts from the first two exercise sessions. Have a look at the bullet-point list at the top of the two assignment sheets to see if there are concepts that you have not yet discussed, which apply to this assignment. Try to specifically use the key words that appear in these lists.
- Think about the numbers you use to discuss your results. Does it make more sense to discuss things in terms of relative or absolute error? How is this effected by the dimension of the matrix?

References

[1] Lloyd N. Trefethen and David III Bau. Numerical Linear Algebra. SIAM, Philadelphia, 1997.

[3] We collect this information to assess the workload of the assignments for future years. The numbers you enter do not influence your grade.