Scientific Software
Session 3: Fortran, arrays and Lapack

Emil Løvbak, Wouter Baert & Pieter Appeltans Leuven, October 22, 2019

Introduction

This exercise session focusses on working with arrays in Fortran. More specifically, this session aims for you to develop an understanding of the following concepts which will come back in the homework assignments:

- array manipulations in Fortran;
- dynamic memory and allocatable arrays in Fortran;
- valgrind, a programming tool to monitor memory usage;
- working with linear algebra libraries such as BLAS and LAPACK.

The syntax you need for the session can be found in the preparatory reading, but you can ask further questions to the assistants in the session. If you have questions later, please post them on the Toledo forum. That way other students can see the questions that have already been asked, as well as their answer.

Exercises

Question 1: Array initialization

Write a short Fortran program and test it with several compilers to answer the following questions.

1. Declare an array of length 10 using `real a(10)`. Then, loop over the elements and print them, what do you observe?
2. Next, put all elements to zero using `a = 0.0`. Then, print the elements again, what do you observe?
3. What happens when you try to print an element outside of the array, eg. `print *,a(12)`?
4. What happens when you compile the program with one of the following flags: `-fbounds-check` (gfortran), `-C=array` (nagfor) or `-CB`(ifort)?

Link preparation: Scientific Software Development with Fotran - 4.1 (Declaring Array Variables), 4.3 (Element-wise Operations)

Question 2: Dynamically allocated arrays

For a lot of problems the needed size of an array is not known *at compile time*. For this purpose, Fortran uses allocatable arrays, the size of which can be specified *at run time*.

Write a program `tensor` that asks the user for an integer number N. Allocate subsequently a $N \times N \times N$ tensor T of single precision floating-point numbers. Print whether the allocation was successful and if successful, fill the tensor with numbers $N^3, \ldots, 1$. Try to avoid explicit loops (*Hint:* use implied loops and `reshape`). Print the element on position $(1, N, N)$. Don't forget to deallocate the tensor at the end of the program to avoid memory leaks.

Link preparation: Scientific Software Development with Fortran - 4.2 (initializing arrays); 4.5 (allocatable arrays)

Question 3: BLAS & LAPACK

In this exercise we will work with BLAS and LAPACK, which implement several common linear algebra operations in both single and double precision. The BLAS documentation can be found on `http://www.netlib.org/blas/#_documentation`. The LAPACK documentation can be found on `http://www.netlib.org/lapack/explore-html/`. In this exercise, only single precision matrices and operations will be used. Execute the following steps.

1. Construct a random 10×10 matrix A in single precision (Hint: `call random_number(A)`).
2. Copy this matrix into another matrix B with the same dimensions.
3. Use `SGETRF` to compute the LU factorization of the left upper 5×5-block of A. Carefully read the documentation of `SGETRF` to understand the effect of this subroutine.
4. Create the matrices L and U from A using `SLACPY` and `SLASET`.
5. Calculate $C = PLU$ using `SGEMM` and `SLASWP` using L, U and the pivot elements that `SGETRF` returned. (Hint: should `INCX` be +1 or -1? Complete the following step, and check which of the two gives the correct result.)
6. Check whether the resulting C matches the left upper block of B. Print for example the maximal element-wise difference.
7. Use `SGETRI` to compute the inverse of the left upper block of A. For this operation, a workspace must be allocated. The optimal size of this workspace is however not known a priori but can be determined by first calling `SGETRI` with WORK an array of length 1 and LWORK equal to -1. On return, the first element of the array WORK contains the optimal workspace length. Allocate subsequently an array of this size and call `SGETRI` again with this new workspace array. For more information, consult the documentation.
8. Check whether the left upper block of B times the resulting left upper block of A returns the identity matrix (of size 5).
9. Compile the code and link it using the flags `-lblas` and `-llapack`

Link preparation: Scientific Software Development with Fortran - 4.8 (array intrinsics); Writing Scientific Software - 12.8 (BLAS); 12.9 (LAPACK); BLAS and LAPACK documentation

Question 4: Power iterations method

In this exercise we implement the power iterations method (also known as the method of von Mises[1]) to find the dominant eigenvalue and associated eigenvector of a matrix A. Execute following steps.

- Read the 3x3 double precision matrix A, which can be found on Toledo in the file `'A.in'`, into your program with `open` and `read`; **Do not forget to close the file afterwards**.
- Implement algorithm 1 (*tip:* use the builtin Fortran functions `dot_product` and `matmul`).
- Test your implementation using A. **Note**: A is stored in column-major form in `'A.in'`. If you want to print A row by row, you will have to write a loop.

[1] named after the Austrian scientist and mathematician Richard Edler von Mises (1883-1953)

Input: A ($n \times n$ matrix), K_{max} (integer), ϵ (scalar)
Output: λ (scalar), x (n dimensional vector)
Initialize x with a random vector of length n;
$x := x/||x||_2$;
$k := 0$;
while $k < K_{max}$ **do**
  $z := Ax$;
  $\lambda := x^T z$;
  $R := z - \lambda x$;
  if $||R|| < \epsilon$ **then**
    STOP
  end
  $x := z/||z||$;
  k := k +1 ;
end

Algorithm 1: Pseudocode for the power iterations method

Link preparation: Scientific Software Development with Fortran - 4.3 (Element-wise operations); 4.8 (Array intrinsics); 5.5 (Opening and closing files).

Question 5

When you dynamically allocate an array, it is important to deallocate it when it is no longer needed. Otherwise you could create memory leaks [2]. These leaks can be traced using `valgrind`. Write a program that allocates memory for an array without deallocating it. Compile it with `gfortran`. First, try to detect the leaks using the compiler warnings and runtime errors. Then, use valgrind. When and how do you detect the leaks?

Question 6

Answer following questions. To this end, write a short (or multiple) Fortran program(s) and compile it (them) with several compilers.

1. Declare an allocatable array using `real, allocatable :: a(:)`. Then try to print the first element of this array, without allocating the array. What do you observe?
2. Allocate the array using `allocate(a(10))`. Then, print an element in the array, what do you observe?
3. What happens when you try to assign a value to an index outside the array?
4. What happens when you try to print an element outside the array?
5. What happens when you execute the program using `valgrind`?
6. Write a program that allocates memory twice without deallocating in between. Run the program using `valgrind` what do you observe?

[2] https://en.wikipedia.org/wiki/Memory_leak