Course 02635, November 17th, 2021.

s194119 – Martin Ægidius

Group members:

s194120, Frederik Gade

s194146, Nikoline Mai Bøgely Rehn

s190566, Oliver Zacho

Assignment 1: call_dgels – solver for least squares problem $\min||Ax-b||_2^2$ by means of QR-factorization

Design: The function takes input: pointer to an array $\boldsymbol{A}$ of type **array2d_t** with dimensions $m \times n$, where $m \geq n$, and a pointer to an array $\boldsymbol{b}$ of type **array_t** with m entries**.** The input-matrix $\boldsymbol{A}$ is assumed nonsingular. The function solves the least squares problem where x is a vector of length n by using QR-factorization with LAPACKs routine dgels. The b-array is overwritten with the least-square-solution for x, and is reduced to length n. The function has input-checks: 1. Do $\boldsymbol{A}$ and $\boldsymbol{b}$ exist? If not, the function returns -12. 2. Does $\boldsymbol{A}$ fulfill having more rows than columns, i.e. is $m > n$? Else, return -13. 3. Is row-count of $\boldsymbol{A}$ compatible with length of $\boldsymbol{b}$? Else, return -14. 4. Is $\boldsymbol{A}$ stored using row-major storage order? If yes, the storage is converted to column-major-storage. 4.1. The conversion uses memory allocation, and thus it is checked if a temporary data-storage matrix is allocated successfully. 5. Is the work-array in **dgels** allocated successfully? In case of memory allocation errors, return -15. In order to use **dgels,** sufficient input-pointers must be initialized. The constant ones are: $\mathrm{nrhs} = 1$ (as call_dgels only is supposed to solve this type of least-squares problem) and $\mathrm{trans} = '\mathrm{N}'$, as the system only involves $\boldsymbol{A}$. The variable inputs are found using a ternary macro implementation for finding the maximal value of two variables: $\mathrm{lwork} = \max(1, \min(m,n) + \max(\min\,(m,n), \mathrm{nrhs})) = \max(1, 2n)\;\;\mathrm{for}\; m > n$, which is the needed length for the work-array. Thus, the work-array needs to be allocated with memory of size $\mathrm{lwork} * \mathrm{sizeof(double)}$ to hold all elements. The leading dimension of $\boldsymbol{A}$**,** LDA, is $\max(1, m)$. The leading dimension of $\boldsymbol{b}$ must fulfill $\mathrm{LDB} \geq \max(1, m, n) \rightarrow \mathrm{LDB} \geq \max(1, m)$ for a vector. LAPACKs dgels_() is called using only pointers, i.e. for these variables the addresses. The function overwrites $\boldsymbol{b}$, and returns info, which is 0 in case of a successful function call. Info may evaluate to a negative value $-i$ if the i'th input-argument of the function call had an illegal value, or positive i if the ith diagonal element of the triangular factorization of $\boldsymbol{A}$ is 0, which implicates division by zero for obtaining the solution.

Numerical aspects: in practice, dgels rarely will return $\mathrm{info} > 0$, even if matrix $\boldsymbol{A}$ is singular. This is due to the check for diagonal elements in the triangular matrix being equal to *exactly* zero, which is very rare while operating in floating point precision due to rounding. We could check if the matrix is singular beforehand, but the assignment assumes full rank. nan- and inf-values force the solution to be nan or inf. This could be checked for before finding the solution, but the function is only designed to work for **healthy arrays**. If factors in the QR-factorization are **very** small or large, loss of precision may occur.

Assignment 2: Command-line tool lssolve

Design: lssolve has 3 input-arguments defined when calling the function: the input-matrix file, the input vector file and the solution-output file. All are in txt-format, and entries space-seperated. Files are loaded using the **msptools** header-functions array2d_from_file and array_from_file. call_dgels() is called with the initialized pointers, and the solution is written to a file using the header-function array_to_file. The program checks if: 1. If call_dgels() fails, due to the checks in assignment 1 (which includes loading-errors). 2. If writing to the output-file is successful. In case of failure, an error-message is printed to stderr (also added to call_dgels), and the return value is set to EXIT_FAILURE; else return is EXIT_SUCCESS (both are defined in the header stdlib.h). Functions in msptools may print to stdout, which is not permitted, i.e. when loading a matrix with inconsistent column numbers. One could redirect error messages more elegantly using unistd.h (appendix 2), but it seems we may not use additional headers. We did not find a portable solution for this.

Testing: The least-square fitting is tested using different matrices with rank n using the command-line-tool. Randomly generated matrices with dimensions $m \times n$ have full rank with a probability of 1 using machine precision (Feng & Zhang, 2007). Thus, we can use a random seed to generate test-matrices, and cross-reference the least squares solution obtained from MATLAB to a certain tolerance with lssolve. A MATLAB implementation is in appendix for generating and saving random matrices, calculating the ls-solution and comparing. Valid comparisons to the 12$^{\text{th}}$ decimal are seen using nonsingular matrices. Singular matrices are generated using a sum for creating linear dependance in the last row. The singular matrices do not give an error code in lssolve due to rounding i.e. a solution is still output from the function even though it should not exist. Therefore, caution is advised when using lssolve. The command-line-tool is also tested with non-existent files, erroneous files e.g. variable row/column-counts, nan/inf values, and by trying to write to a read-only file. Of course, nan- and inf-values break solution, but this is expected.

Bibliography

Feng, X., & Zhang, Z. (2007). The rank of a random matrix. *Applied Mathematics and Computation 185*, s. 689-694.

Appendix 1 – MATLAB implementations for testing

Code is written inefficiently for illustrational purposes. Could easily be automated.

Program for creating and saving random singular matrices with m>n:

```
n = 100;
A = randn(n,floor(n/2)); %random matrix with m>n
A(end+1,:) = sum(A); %create linear dependance in row n+1 -> singular
b = randn(n+1,1); %random vector of size m
writematrix(A,'ASingmatrix.txt','Delimiter',' ');
writematrix(b,'BSingmatrix.txt','Delimiter',' ');
x = lsqr(A,b); %establish least-squares solution
writematrix(x,'XSingmatrix.txt','Delimiter',' '); %save solution
```

Program for creating and saving random non-singular matrices with m>n:

```
n = 100;
A = randn(n,7);
b = randn(n,1);
writematrix(A,'Amatrix.txt','Delimiter',' ');
writematrix(b,'Bmatrix.txt','Delimiter',' ');
x = lsqr(A,b);
writematrix(x,'Xmatrix.txt','Delimiter',' ');
```

Program for comparing output-files of lssolve.c and matlab lsqr to a certain tolerance:

```
alsqr = load('Xmatrix.txt'); %load matlab-solution-vector
blsqr = load('XmatrixC.txt'); %load c LAPACKE solution vector
valCmp = abs(alsqr-blsqr); %calculate residual between matlab and c result
if valCmp<=1e4*eps('double')%check if difference smaller than 2.22e-12
    disp("Agreement in least-squares solution to 12th decimal.");
else
    disp("!Different results!");
end
```

Appendix 2 – suppressing/redirecting stdout output from msptools functions

Redirecting stdout to Unix-/Unix-like null-device:

```
#include <stdlib.h>
#include <stdio.h>
#include "msptools.h"
#include <unistd.h> //for duplicating stream to dev/null ie dup2()
#include <fcntl.h> //for file control options ie O_RDWR (open read-write)
int call_dgels(array2d_t *A, array_t *b);
int main(int argc, char *argv[])
{
  if (argc != 4)
  {
    fprintf(stderr, "Usage: %s A b x\n", argv[0]);
    return EXIT_FAILURE;
  }

  /* Insert your code here */
  int fd = open("/dev/null",O_RDWR); //define null device on UNIX systems
  dup2(fd,1); //writing stdout-output from subfunctions to null device
  if(fd>2)close(fd); //open returns the lowest numbered unused file descriptor.
Bigger than two indicates error

  array2d_t *A =array2d_from_file(argv[1]);
  array_t *b = array_from_file(argv[2]);
  //rest of program...
  //.....
```

NOTE: the null device is defined differently on Unix-systems than on Windows systems, where it is defined as nul. Thus, this implementation should be used carefully, depending on system.

For keeping the output from stdout error-messages, one could instead redirect it to stderr using solely dup2(2,1) (as stdout has file descriptor 1, and stderr has file descriptor 2).