Fortran Homework Assignment 2

Solving KKT systems

Joris Tavernier, Emil Loevbak & Pieter Appeltans

October 26, 2018

1 Introduction

Note: This section does not need to be fully understood to complete the assignment, but is only here as background information. The summary of this section is that we want to solve systems in the form given by (3) *and we will do this with an algorithm given in section 2.*

Optimization is all around us. Nature generally is in peace, e.g., equilibrium. People tend to optimize their world, from investors maximizing their rate of return; to students minimizing their effort and still finish courses; and teachers letting students pass exams to maximize their vacation time.

This report is based on the notation and contents of Nocedal and Wright [1]. The following symbols are used:

- x is a vector of variables $\in \mathbb{R}^n$, also called unknowns or parameters
- q is the objective function which we want to minimize or maximize

We will focus on equality constrained positive definite quadratic problems, which can be formulated as:

$$\min_x \quad q(x) = \frac{1}{2}x^T G x + x^T c \tag{1a}$$

$$\text{subject to} \quad Ax = b \tag{1b}$$

where we define $G \in \mathbb{R}^{n\times n}, c \in \mathbb{R}^n, A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$ with m constraint rows and all matrices are of full rank. Introducing a vector of Lagrange multipliers λ

$$\begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -c \\ b \end{bmatrix} \tag{2}$$

we can derive the Karush-Kuhn-Tucker (KKT) system

$$\begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} -p \\ \lambda^* \end{bmatrix} = \begin{bmatrix} g \\ h \end{bmatrix} \tag{3}$$

where the optimal solution is rewritten as a current estimate x and a search direction p

$$x^* = x + p \tag{4}$$

$$h = Ax - b \tag{5}$$

$$g = x + Gx \tag{6}$$

Denoting $Z \in \mathbb{R}^{n\times(n-m)}$ of rank $n-m$ as the nullspace of A, e.g. $AZ = 0$. The KKT system is nonsingular if A is of full row rank and the reduced-Hessian matrix Z^TGZ is positive definite. The solution x^* is then a global minimizer of (1).

For a full discussion of the theoretical background to optimization, we refer to the book by Nocedal and Wright [1] and to courses on optimization. In the remainder of this assignment we will focus on solving sets of equations of the form given in (3).

2 Schur-complement method for KKT systems

A simple way to approach the problem of solving such system in the form given by (3) is through the application of a general solver for linear systems. An example is LU-factorization through Gauss as implemented in the previous assignment. However the system we are solving has a special structure which we should take advantage of. A method that does this is the Schur-complement method.

First we will solve the equation

$$(AG^{-1}A^T)\lambda^* = (AG^{-1}g - h) \tag{7}$$

to recover λ^*. This equation is derived by multiplying the first equation in the system (3) with AG^{-1} and subtracting the second equation from it. Next we can use the expression

$$Gp = A^t\lambda^* - g \tag{8}$$

to calculate p. This method assumes that G is positive definite, but this is not an issue as we assumed that our problem was convex. For more background information on this method a good reference is chapter 16 of Nocedal and Wright [1].

As practical point for the implementation of (7), it should be noted that there is no need to explicitly compute the inverse of G. Once a LU-decomposition for G has been computed, the product $G^{-1}A^T$ can be computed through a series of M upper and lower triangular solves. This LU-decomposition can also be re-used in the computation of the right-hand side.

3 Assignment

This assignment builds further on your code from the previous assignment. This means that you have the opportunity to make changes to your existing code to show what you have learned from your feedback. In addition you will need to expand on your existing implementation. Specifically you will implement the Schur-complement method and compare it with your implementation of Gauss' method.

- Adapt your code from the `matrix_solve` module so that you no longer need to specify the dimension, i.e., the interfaces should be `gauss(A, v)`, `gauss_pivot(A, v)`, `cond(A)` and `sum_ev(A)`. Make any changes in your code that this modification requires.

- Also avoid hard coding the dimensions of any matrices. To this end a new file `inout2.f90` can be found on Toledo for this assignment, that contains subroutines `matrix_dims(dims, path)` and `vector_length(length, path)`.

- Re-run your code from the previous assignment to check that everything still works.

- Add a subroutine to the `matrix_solve` module called `schur(G, A, gg, hh)` that solves KKT systems according to equations (7) and (8). To solve the linear systems in this algorithm, you should re-use the code for your implementation of Gauss with pivoting. (Hint: Think about how you can do this in an efficient way. Do you have to recompute everything for each system you solve?)

- Replace the file `eigv_FC.o` that you received for the previous homework assignment with your own implementation which you can call with the interface

```
subroutine eig(A, lambdar, lambdai)
    real(wp), dimension(:,:), intent(in) :: A
    real(wp), dimension(:), intent(out) :: lambdar, lambdai
```

  where `wp` is the resulted of `selected_real_kind()` for inputs 6 or 15. You can use two Lapack routines to do this `SGEEV` and `DGEEV`. (Hint: Write a subroutine to deal with calling Lapack for each precision and think about how you can call both using the same name.)

- Run your software using Valgrind to check for any memory leaks and give an explanation for any remaining leaks. Note: You should not spend too much time fixing these leaks.

- As in the previous assignment, you can find some sets of test KKT-systems on Toledo. Solve them using the Schur complement method and Gauss with pivoting for both single and double precision. For each test you should print the following output:

  - the relative 2-norm of the forward error;

  - the condition number of the full KKT matrix;

  - the CPU-time it takes to solve the given system for the method. (Think about how you should do this in a reliable way.)

- After completing the previous step, run the double precision versions both of your algorithms for second test system one at a time using Valgrind and check the overall memory consumption. (You can use your existing main program to do this where you comment out unneeded lines of code each time you recompile.)

- Compare these methods using the generated data and discuss how they compare.

- (Extra) Test your code on other Fortran-compilers to check that everything is compiler independent. Document which changes were necessary.

- (Extra) Test the effect of using different compiler optimization levels on the run-time and memory usage of your code.

4 Practical information

The deadline for submission on Toledo is **11 November at 14:00**. This deadline is strict! Do not wait until 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.[1] 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.

The bullet points in the previous section marked "extra" are not necessary to achieve a reasonable passing grade for this assignment. However if you wish to go for extra marks, you can complete one or both of these points.

Some specific points to pay attention to:

- In your discussion, we expect you to be specific. This means that besides mentioning concepts such as stability, condition, computational complexity and so on, you should refer to your code (e.g. a specific operation in a given line of code, number of nested loops, ...) to explain why an algorithm behaves in a certain way.

- This assignment is an opportunity to demonstrate that you have understood the concepts from the first four exercise sessions, with a focus on sessions three and four. 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.

- Motivate your implementation decisions. Which concepts from the exercise sessions have you applied to your program?

- In this assignment you will be writing data in multiple arrays. Think about which compiler flags you can use to help avoid errors, e.g., with writing out of bounds. Mention your compiler flags and why you use them in your report.

References

[1] Stephen Wright and Jorge Nocedal. *Numerical optimization*. Springer-Verlag New York, 2nd edition, 2006.

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