564 Project: SQP in C++

Seth Brown

May 1, 2019

1 Introduction

SQP (sequential quadratic programming) methods are a useful way to approach nonlinear programs, in that they are able to apply efficient quadratic programming solution approaches to find approximate solutions to non-quadratic problems. Here, I have followed the Matlab code for SQP provided in class to develop a C++ implementation. I also use OpenMP to parallelize this implementation, though any performance gains, at least for the problem under consideration, are minimal. However, the performance gains over Matlab are significant even without parallelism. Here I solve Problem 18.3 from page 562 of Nocedal and Wright[1]:

$$\min e^{x_1x_2x_3x_4x_5} - (x_1^3 + x_2^3 + 1)^2/2$$

$$\text{s.t. } \sum_{i=1}^{5} x_i^2 = 10$$

$$x_2x_3 - 5x_4x_5 = 0$$

$$x_1^3 + x_2^3 + 1 = 0$$

and the problem of finding the analytic center for an equality constrained polyhedron:

$$\min -\sum_{i=1}^{n} \log(x_i)$$

$$\text{s.t. } \mathsf{A}\mathbf{x} = \mathbf{b}$$

In particular I use:

$$\mathsf{A} = \begin{bmatrix} -1.2141 & -0.7697 & -1.0891 & 1.5442 & 0.5377 & 0.3188 & 3.5784 & 0.7254 & -0.1241 & 0.6715 \\ -1.1135 & 0.3714 & 0.0326 & 0.0859 & 1.8339 & -1.3077 & 2.7694 & -0.0631 & 1.4897 & -1.2075 \\ -0.0068 & -0.2256 & 0.5525 & -1.4916 & -2.2588 & -0.4336 & -1.3499 & 0.7147 & 1.4090 & 0.7172 \\ 1.5326 & 1.1174 & 1.1006 & -0.7423 & 0.8622 & 0.3426 & 3.0349 & -0.2050 & 1.4172 & 1.6302 \end{bmatrix}$$

[1]Nocedal, J., & Wright, S. (2006). Numerical optimization. Springer Science & Business Media.

$$\mathbf{b} = \begin{bmatrix} 1.0933 \\ 1.1093 \\ -0.8637 \\ 0.0774 \end{bmatrix}$$

These do not correspond to any particular problem but were rather chosen at random.

2 How to run the code

One future improvement would be to make the code a bit easier to use for new problems. However, it shouldn't be too bad even now. To run in serial, follow these steps:

1. Write a new example problem `example.cpp` and corresponding header `example.h` file or select an existing one.
2. Update the include statements in `SQPhelp.h`, `SQPhelp.cpp`, and `SQPserial.cpp` to reference `example.h` and not the old example problem.
3. Let N and M be the dimensions of the problem selected. Build all via `g++ example.cpp SQPhelp.cpp SQPserial.cpp -D N=`N `-D M=`M `-D dim=`$N+M$ `-o sqps`.
4. Use `./sqps` to run.

Note that there is also a test function, `SQPtest.cpp`, which can be used in conjunction with the `nwsqp` example in place of the usual `SQPserial.cpp` in order to check if any changes to the main code have broken something. To run in parallel with 4 threads, the steps are similar:

1. Write a new (potentially parallelized) example problem `example.cpp` and corresponding header `example.h` file or select an existing one.
2. Update the include statements in `SQPhpar.h`, `SQPhpar.cpp`, and `SQPparallel.cpp` to reference `example.h` and not the old example problem.
3. Let N and M be the dimensions of the problem selected. Build all via `g++ -fopenmp -O3 example.cpp -O3 SQPhpar.cpp -O3 SQPparallel.cpp -D N=`N `-D M=`M `-D dim=`$N+M$ `-o sqpp`.
4. Use `./sqpp` to run.

Another useful improvement would be to make the number of threads to use be a command line option. Note that the current examples are:

- `nwsqp`, the serial version of the Nocedal and Wright problem
- `nwsqppar`, the parallel version of the Nocedal and Wright problem
- `ecac`, the serial version of the analytic center problem
- `ecacp`, the parallel version of the analytic center problem

3 Results

The analytic center test example runs in serial in 1800-2000 us. In parallel it takes 2300-2500 us - this is not particularly surprising given that the problem is not very large and the opportunities for parallelism are limited. The Nocedal and Wright problem runs in serial in about 500 us, or in parallel in 700-800 us - again, the increased time is not particularly surprising given the limited parallelism. On the other hand, the Matlab code you provided takes at least 270 ms to run on my machine for the Nocedal and Wright problem. If it were possible to parallelize the LU step, this would have significant value - but at least for the current algorithm this is not an option.