Introduction

Computational Intelligence, Lecture 1

by Sergei Savin

Spring 2024

Content

- Motivation
- Motivating example
  - fmincon
  - quadprog
  - SVD-based solution
  - CVX-based solution
- Homework

Motivation

A lot of modern methods, computations and algorithms are backed by numerical optimization tools. In this course we will study numerical optimization with an emphasis on convex methods.

What we want?

To go from "I hope it works" to a solid understanding of the mathematics and use-cases of those tools.

Why we want it?

It should allow us to solve a much wider range of problems, and solve them more effectively.

We have the following problem: find such $\mathbf{x}$ that minimizes $\mathbf{x}^\top \mathbf{M}\mathbf{x}$, while $\mathbf{C}\mathbf{x} = \mathbf{y}$. In other words:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \mathbf{x}^\top \mathbf{M}\mathbf{x}, \\ \text{subject to} \quad & \mathbf{C}\mathbf{x} = \mathbf{y}. \end{aligned} \tag{1}$$

More concrete:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 5 & 0 \\ 1 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \\ \text{subject to} \quad & \begin{bmatrix} 1 & 7 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 1. \end{aligned} \tag{2}$$

How do we solve it?

One very popular way of doing it is by use of a general-purpose local optimization solver, such as `fmincon` provided by MATLAB. Here is one possible solution:

```
M = [1 0 1; 0 5 0; 1 0 3];
C = [1 7 2];
y = 1;

fnc = @(x) x'*M*x;
con = @(x) deal([], C*x-y);
x = fmincon(fnc, zeros(3, 1), [],[],[],[],[],[], con)
```

Average solution time is **4.8** ms (this depends on many factors, so treat it only as a relative information). Solution is $\mathbf{x} = \begin{bmatrix} 0.0442 & 0.1239 & 0.0442 \end{bmatrix}$.

A more sophisticated, but still a very straightforward approach is to use a dedicated solver for this class of problems `quadprog` provided by MATLAB. Here is the solution:

```
M = [1 0 1; 0 5 0; 1 0 3];
C = [1 7 2];
y = 1;

x = quadprog(M,[],[],[],C,y)
```

Average solution time is **0.56** ms, an order of magnitude less than with `fmincon`.

We can use an algebraic solution, based on SVD decomposition (or its derivative methods - null space and pseudo-inverse), as follows:

```
M = [1 0 1; 0 5 0; 1 0 3];
C = [1 7 2];
y = 1;

tol = 10^(-5);
[P, N] = pinv_null(C, tol);
x = ( eye(3) - N*((N'*M*N) \ (N'*M)) ) * P*y
```

Where `pinv_null` is a function combining `pinv` and `null`, obtained from a single SVD decomposition.
Average solution time is **0.027** ms, $\sim$ 20 times faster than `quadprog` and $\sim$ 200 times faster than `fmincon`.

Motivating example
CVX-based solution

Finally, we can invoke one of the most powerful convex optimization tools with a user-friendly coding style - CVX:

```
M = [1 0 1; 0 5 0; 1 0 3];
C = [1 7 2];
y = 1;

cvx_begin
variables x(3);
minimize( x' * M * x );
subject to
    C*x == y;
cvx_end
```

However, we will see that the overhead for the call to the solver for this task is excessive. Average solution time is **282** ms, which is ~ 60 times slower than `fmincon`.

Lecture slides are available via Github, links are on Moodle:

github.com/SergeiSa/Computational-Intelligence-2024