Project 1 NLA: direct methods in optimization with constraints

The goal of this project is to investigate some of the basic numerical linear algebra ideas behind optimization problems. For simplicity we consider convex optimization problems. Concretely, we look for $x \in \mathbb{R}^n$ that solves

$$\begin{aligned} \text{minimize} \quad & f(x) = \frac{1}{2} x^T G x + g^T x \\ \text{subject to} \quad & A^T x = b, \quad C^T x \geq d, \end{aligned} \tag{1}$$

where $G \in \mathbb{R}^{n \times n}$ is symmetric semidefinite positive, $g \in \mathbb{R}^n$, $A \in \mathbb{R}^{n \times p}$, $C \in \mathbb{R}^{n \times m}$, $b \in \mathbb{R}^p$ and $d \in \mathbb{R}^m$.

We first describe the general ideas to solve the problem (1) using an interior point algorithm. After this general mathematical description we ask for different implementations and theoretical exercises that complete the project work.

1 Description of the interior point algorithm

The interior point strategies are based on the solution of the so-called KKT system. Details of the derivation are given briefly below, and we refer to the optimization course for further details. Here we just point out that since our problem is convex, the KKT conditions guarantee that any solution is a minimum of $f(x)$.

1.1 The KKT system

The constrained minimization problem can be solved using Lagrange multipliers. The inequality constrains are translated into equality constrains by means of slack variables. We introduce $s = C^T x - d \in \mathbb{R}^m$. Then, the Lagrangian is given by

$$L(x, \gamma, \lambda, s) = \frac{1}{2} x^T G x + g^T x - \gamma^T (A^T x - b) - \lambda^T (C^T x - d - s^2)$$

where $\gamma \in \mathbb{R}^p$ and $\lambda \in \mathbb{R}^m$ are the vectors containing the Lagrangian multipliers for the inequality and inequality constrains respectively.

The optimality conditions are

$$\begin{aligned} Gx + g - A\gamma - C\lambda &= 0 \\ b - A^T x &= 0 \\ s + d - C^T x &= 0 \\ s_i \lambda_i &= 0, \quad i = 1, \ldots, m \end{aligned}$$

where we also require the components v_i of $v = (\lambda, s) \in \mathbb{R}^{2m}$ to verify $v_i \geq 0$ (feasibility region).

We introduce $z = (x, \gamma, \lambda, s)$ and $F : \mathbb{R}^N \to \mathbb{R}^N$, $N = n + p + 2m$, such that the previous (nonlinear) system of equations is translate into $F(z) = 0$. The solutions can be computed using a Newton method. The computation of an iterate of the Newton method reduces to solve a linear system, the so-called KKT system.

Let S (resp. Λ) be the diagonal matrix with s (resp. λ) in the diagonal. Then, one step of the Newton method to determine $\delta_z = (\delta_x, \delta_\gamma, \delta_\lambda, \delta_s)$ such that $z_1 = z_0 + \delta_z$ leads to the solution of a linear system defined by the KKT matrix

$$M_{\text{KKT}} = \begin{pmatrix} G & -A & -C & 0 \\ -A^T & 0 & 0 & 0 \\ -C^T & 0 & 0 & I \\ 0 & 0 & S & \Lambda \end{pmatrix}$$

and right-hand vector $-F(z_0) = (-r_L, -r_A, -r_C, -r_s)$.

1.2 Feasibility problem: Newton step-size using a central path

We note that:

1. The previous scheme shows that the optimization problem (1) is equivalent to the computation of z such that $F(z) = 0$ and such that verifies the feasibility condition: if $v = (s, \lambda)$ we want $v_i \geq 0$ for all $i = 1, \ldots, 2m$.
2. The use of a Newton method to solve $F(z) = 0$ does not guarantee that the solution verifies the feasibility condition.

There are different approaches to guarantee that the solution verifies the feasibility condition. One of the simplest one is to select the Newton steps so that the iterates z_k are close to a curve (the central path) of feasible points. We consider the path $(0, 0, 0, \tau e)$ parametrized by $\tau \in \mathbb{R}$ and where $e = (1, \ldots, 1) \in \mathbb{R}^m$.

Instead of computing iterates of the Newton method we will compute steps of the following algorithm. Assume we have computed $z_0 = (x_0, \gamma_0, \lambda_0, s_0)$ and we want to determine the next iterate $z_1 = (x_1, \gamma_1, \lambda_1, s_1)$. An step of the algorithm to compute z_1 consists of following substeps:

1. Predictor substep: We compute the standard Newton step δ_z (by solving the KKT system).
2. Step-size correction substep: Compute α so that the step-size $\alpha\delta_z$ guarantees feasibility. Below we explain how to do this.
3. Compute $\mu = \frac{s_0^T \lambda}{m}$, $\tilde{\mu} = \frac{(s_0 + \alpha\delta_s)^T(\lambda + \alpha\delta_\lambda)}{m}$ and $\sigma = (\tilde{\mu}/\mu)^3$.
4. Corrector substep: Solve the linear system with the same KKT matrix M_{KKT} and right hand vector $(-\hat{r}_L, -\hat{r}_A, -\hat{r}_C, -\hat{r}_s) = (-r_L, -r_A, -r_C, -r_s - D_s D_\lambda e - \sigma\mu e)$, where $D_s = \text{diag}(\delta_s)$ and $D_\lambda = \text{diag}(\delta_\lambda)$.
5. Step-size correction substep.
6. Update substep: Define $z_1 = z_0 + 0.95\,\alpha\,\delta_z$ and update M_{KKT} and right-hand parts.

Stop criterion: Fix $\epsilon = 10^{-16}$ and iterate the previous algorithm until either $|r_L| < \epsilon$, $|r_C| < \epsilon$ or $|\mu| < \epsilon$ or until the number of iterations of the algorithm exceeds 100.

Step-size correction: The following routine can be used to compute α for the step-size correction substeps.

```
def Newton_step(lamb0,dlamb,s0,ds):
    alp=1;
    idx_lamb0=np.array(np.where(dlamb<0))
    if idx_lamb0.size>0:
       alp = min(alp,np.min(-lamb0[idx_lamb0]/dlamb[idx_lamb0]))

    idx_s0=np.array(np.where(ds<0))
    if idx_s0.size>0:
       alp = min(alp,np.min(-s0[idx_s0]/ds[idx_s0]))

    return alp
```

2 Solving the KKT system

The project consists in the implementation of specific routines to solve the problem (1) based on different ways to solve the KKT system of the predictor and corrector substeps.

T1: Show that the predictor steps reduces to solve a linear system with matrix M_{KKT}.

C1: Write down a routine function that implements the step-size substep.

2.1 Inequality constrains case (i.e. with $A = 0$)

We shall consider first the case of inequality constrains only (i.e. with $A = 0$), and later we will discuss about the general case.

We propose different strategies to solve the KKT system. To test the routines use the *test problem* defined by

$$m = 2n, \quad G = I_{n\times n}, \quad C = \begin{pmatrix} I_{n\times n} & -I_{n\times n} \end{pmatrix}, \quad d = (-10, \ldots, -10),$$

and random components of g generated by a Gaussian distribution $N(0, 1)$. Use $x_0 = (0, \ldots, 0)$ and $s_0 = \lambda_0 = (1, \ldots, 1)$ as initial points of the algorithm. The solution of this test problem is given by $x = -g$.

C2: Write down a program that, for a given n, implements the full algorithm for the test problem. Use the `numpy.linalg.solve` function to solve the KKT linear systems of the predictor and corrector substeps directly.

C3: Write a modification of the previous program **C2** to report the computation time of the solution of the test problem for different dimensions n.

We suggest the following strategies to solve the KKT system more efficiently. Note that M_{KKT} is a 3×3 block matrix in this case. We denote by $(-r_1, -r_2, -r_3)$ the right-hand vector.

Strategy 1: Isolate δ_s from the 3rd row of the KKT system,

$$\delta_s = \Lambda^{-1}(-r_1 - Sd_\lambda),$$

and substitute it into the 2nd row. This leads to the system

$$\begin{pmatrix} G & -C \\ -C^T & -\Lambda^{-1}S \end{pmatrix} \begin{pmatrix} \delta_x \\ \delta_\lambda \end{pmatrix} = - \begin{pmatrix} r_1 \\ r2 - \Lambda^{-1} r_3 \end{pmatrix}$$

We can then use LDL^T factorization of the matrix to solve the system.

Strategy 2: Isolate δ_s from the 2nd row,

$$\delta_s = -r_2 + C^T \delta_x,$$

and substitute into the 3rd row to get

$$\delta_\lambda = S^{-1}(-r_3 + \Lambda r_2) - S^{-1}\Lambda C^T \delta_x.$$

Finally substitute into the 1st row to obtain the linear system $\hat{G}\delta_x = -r_1 - \hat{r}$, where $\hat{G} = (G + CS^{-1}\Lambda C^T)$ and $\hat{r} = -CS^{-1}(-r_3 + \Lambda r_2)$. One can apply Cholesky factorization to $\hat{G}$ to solve the system.

T2: Explain the previous derivations of the different strategies and justify under which assumptions they can be applied.

C4: Write down two programs (modifications of **C2**) that solve the optimization problem for the test problem using the previous strategies. Report the computational time for different values of n and compare with the results in **C3**.

The memory of the project should also contain information about the number of iterations performed, the precision of the results, the condition number of the matrices, etc, and also about any difficulties in the implementation you have found, improvements, other checks, etc.

2.2 General case (with equality and inequality constrains)

C5: Write down a program that solves the optimization problem for the general case. Use `numpy.linalg.solve` function. Read the data of the optimization problems from the files (available at the Campus Virtual). Each problem consists on a collection of files: `G.dad`, `g.dad`, `A.dad`, `b.dad`, `C.dad` and `d.dad`. They contain the corresponding data in coordinate format. [1] Take as initial condition $x_0 = (0, \ldots, 0)$ and $s_0 = \gamma_0 = \lambda_0 = (1, \ldots, 1)$ for all problems.

T3: Isolate δ_s from the 4th row of M_{KKT} and substitute into the 3rd row. Justify that this procedure leads to a linear system with a symmetric matrix.

C6: Implement a routine that uses LDL^T to solve the optimizations problems (in **C5**) and compare the results.

[1] The `optpr1` ($n = 100$, $p = 50$, and $m = 200$) has as solution a vector x such that $f(x) = 1.15907181 \times 10^4$. The `optpr2` ($n = 1000$, $p = 500$, and $m = 2000$) has as solution a vector x such that $f(x) = 1.08751157 \times 10^6$.