# Project 1: direct methods in optimization with constraints NLA

Clàudia Valverde

November 8, 2024

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

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

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

This exercise is intended to teach strategies for identifying problems with flow, transition and organization across writing.

### 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 $s = C^T x - d \in \mathbb{R}^m$, $s \geq 0$. Then, the Lagrangian is given by

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

where $\lambda \in \mathbb{R}^p$ and $\lambda \in \mathbb{R}^m$ are the vectors containing the Lagrangian multipliers for the inequality and inequality constrains respectively.
The optimality conditions can be rewritten as

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

where we also require the components $v_i = (\lambda, s) \in \mathbb{R}^{2m}$ 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. $\Lambda$) be the diagonal matrix with $s$ (resp. $\lambda$) on 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.

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

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 $i = 1, \dots, 2m$.

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 ideas is to modify the Newton steps so that the iterates $z_k$ verify that $F(z_k)$ 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 $e = (1, \dots, 1) \in \mathbb{R}^m$. These points verify the same optimality conditions except the last one, which is $s_i\lambda_i = \tau$ instead (hence when $\tau = 0$ we get the solution of the problem).

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 $z_1 = (x_1, \gamma_1, \lambda_1, s_1)$. A step of the algorithm to compute $z_1$ consists of the following substeps:

1. **Predictor substep:** We compute the standard Newton step $\delta z$ (by solving the KKT system).

2. **Step-size correction substep:** Compute $\alpha$ so that the step-size $\alpha\delta z$ guarantees feasibility. Below we explain how to do this.

3. Compute $\mu = \frac{s_0^T \lambda_0}{m}$, $\tilde{\mu} = \frac{(s_0 + \alpha\delta s)^T(\lambda_0 + \alpha\delta\lambda)}{m}$, and $\sigma = \left(\frac{\tilde{\mu}}{\mu}\right)^3$.

4. **Corrector substep:** Solve the linear system with the same KKT matrix $M_{KKT}$ and right-hand vector
$$(-\hat{r}_L, -\hat{r}_A, -\hat{r}_C, -\hat{s}) = (-r_L, -r_A, -r_C, -r_s - D_s D_\lambda e + \sigma\mu e),$$
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.

**Stopping Criterion:** Fix $\epsilon = 10^{-16}$ and iterate the previous algorithm until either

$$|r_L| < \epsilon, \quad |r_A| < \epsilon, \quad |r_C| < \epsilon, \quad \text{or} \quad |\mu| < \epsilon,$$

or until the number of iterations of the algorithm exceeds 100.

**Step-size Correction:** The following routine can be used to compute $\alpha$ for the step-size correction substeps:

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

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

    return alpha
```

## 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 reduce to solving a linear system with matrix $M_{KKT}$.

To solve a minimization problem with equality and inequality constraints, we apply the Karush-Kuhn-Tucker (KKT) conditions, which provide necessary conditions for optimality. Let $F : \mathbb{R}^N \to \mathbb{R}^N$, where $N = n + p + 2m$, and define $z = (x, \gamma, \lambda, s)$ as the vector of variables. Our objective is to find an optimal $z$ such that $F(z) = 0$, using the Newton method to approximate a solution iteratively.

We solve $F(z) = 0$ by linearizing $F(z)$ around the current iterate $z_0$ to obtain a sequence of approximations that converge to the optimal $z$. The Newton iteration is:

$$DF(z_0) \cdot \delta z = -F(z_0),$$

where $\delta z = z_{n-1} - z_n$ represents the Newton step.

The Lagrangian function $L$ is:

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

where the optimal solution satisfies the partial derivatives (first-order conditions) of $L$ with respect to each variable:

$$\frac{\partial L}{\partial x} = 0, \quad \frac{\partial L}{\partial \gamma} = 0, \quad \frac{\partial L}{\partial \lambda} = 0, \quad \frac{\partial L}{\partial s} = 0.$$

These conditions lead to the KKT conditions, which are necessary for optimality and are defined above.

The Jacobian of $F(z)$, corresponds to $DF(z)$ in this context. When evaluated at $z_0$, it forms a specific structure known as the KKT matrix (denoted $M_{KKT}$), which reflects the relationships between the variables $x$, $\gamma$, $\lambda$, and $s$ based on the KKT conditions.

The matrix $M_{KKT}$ is a block-structured matrix that includes the Hessian of the Lagrangian (second derivatives with respect to $x$), as well as the gradients of the constraint functions. Its structure is as follows:

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

This matrix consolidates the derivatives of the Lagrangian with respect to the variables $(x, \gamma, \lambda, s)$, capturing the linearized system of equations that we need to solve in each Newton step.

Knowing that, the predictor step involves solving the following linear system:

$$M_{KKT} \cdot \delta z = -r,$$

where $r = F(z_0)$ represents the residual, measuring how far the current $z_0$ is from satisfying the KKT conditions. Solving this system provides the update $\delta z$, which is then used to iteratively bring $z$ closer to the solution that satisfies all KKT conditions.

In summary, by constructing the KKT matrix $M_{KKT}$ (the Jacobian or Jaccard matrix in this context) and solving the linearized system at each step, the Newton method iteratively refines the estimate of $z$, moving closer to the optimal solution that satisfies $F(z) = 0$.

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

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

The step-size substep is given by the provided function named 'Newton_step' described in the statement of the project which takes the initial values of lamda, delta_lambda, s and delta_s as atributes.

**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.

The algorithm is written in the function named "main_algorithm()", all the steps and necessary functions are described there as well. Following what it is explained above, we have to recreate the algorithm taking a few considerations into account as we are working only with inequalities. In this section we are detailing the main differences to perform the algorithm with inequalities, in C5 the whole algorithm is explained in more detail.

Since we are dealing only with inequality constraints, the system simplifies because there is no matrix $A$ (equality constraint coefficient matrix) or $\gamma$ (equality constraint dual variable).

Optimality Condition (Gradient of the Lagrangian) Residual $r_L$:

$$r_L = Gx + g + C^T s$$

where:

- $G$ is the Hessian of the objective function,
- $x$ is the variable vector,
- $g$ is the gradient vector of the objective function, and
- $C^T s$ incorporates the inequality constraints and their respective dual variables $s$.

Since we only have inequality constraints, this residual is simplified to include only $G$, $g$, and $C$, with no contributions from equality terms.

Constraint Residual:

$$r_C = s + d - Cx$$

This ensures that the inequality constraints $s + d = Cx$ are satisfied, where:

- $s$ is the slack variable associated with inequality constraints, and
- $d$ is a constant vector from the inequality constraint definition $Cx \geq d$.

Complementarity Condition Residual $r_s$

$$r_s = Se - s$$

where $S$ is a diagonal matrix with slack variables on the diagonal, and $e$ is a vector of ones. This residual ensures that the complementarity condition holds, verifying that slack variables remain non-negative.

We will use the `numpy.linalg.solve` function to solve the linear system for both predictor and corrector steps of the algorithm.

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

Under the funciton test_different_dimensions() we iterate over the algorithm in C2 in order to test it for different values of n. We have test it every 20 n from 10 to 500, but in the table 1 we only show some selected dimensions to get the idea of the performance.

| **Dimension (n)** | **Exec. Time (s)** | **Iterations** | **Optimal** $f(x)$ | **Theor. Optimal** $f(x)$ | $k_1$ |
|---|---|---|---|---|---|
| 10 | 0.0282 | 18 | 5.693022 | 5.693022 | 23.300590 |
| 30 | 1.3062 | 18 | 13.579378 | 13.579378 | 23.391767 |
| 50 | 1.4629 | 18 | 18.736786 | 18.736786 | 24.694641 |
| 70 | 0.3945 | 20 | 33.518188 | 33.518188 | 24.384718 |
| 90 | 0.6052 | 19 | 48.031915 | 48.031915 | 27.167120 |
| 190 | 1.5256 | 20 | 89.830572 | 89.830572 | 24.510761 |
| 290 | 5.5364 | 19 | 138.784344 | 138.784344 | 25.489691 |
| 390 | 12.2501 | 19 | 192.484047 | 192.484047 | 25.936782 |
| 490 | 18.2215 | 19 | 273.101959 | 273.101959 | 25.647024 |

Table 1: Summary of Results for C3 with Different Dimensions

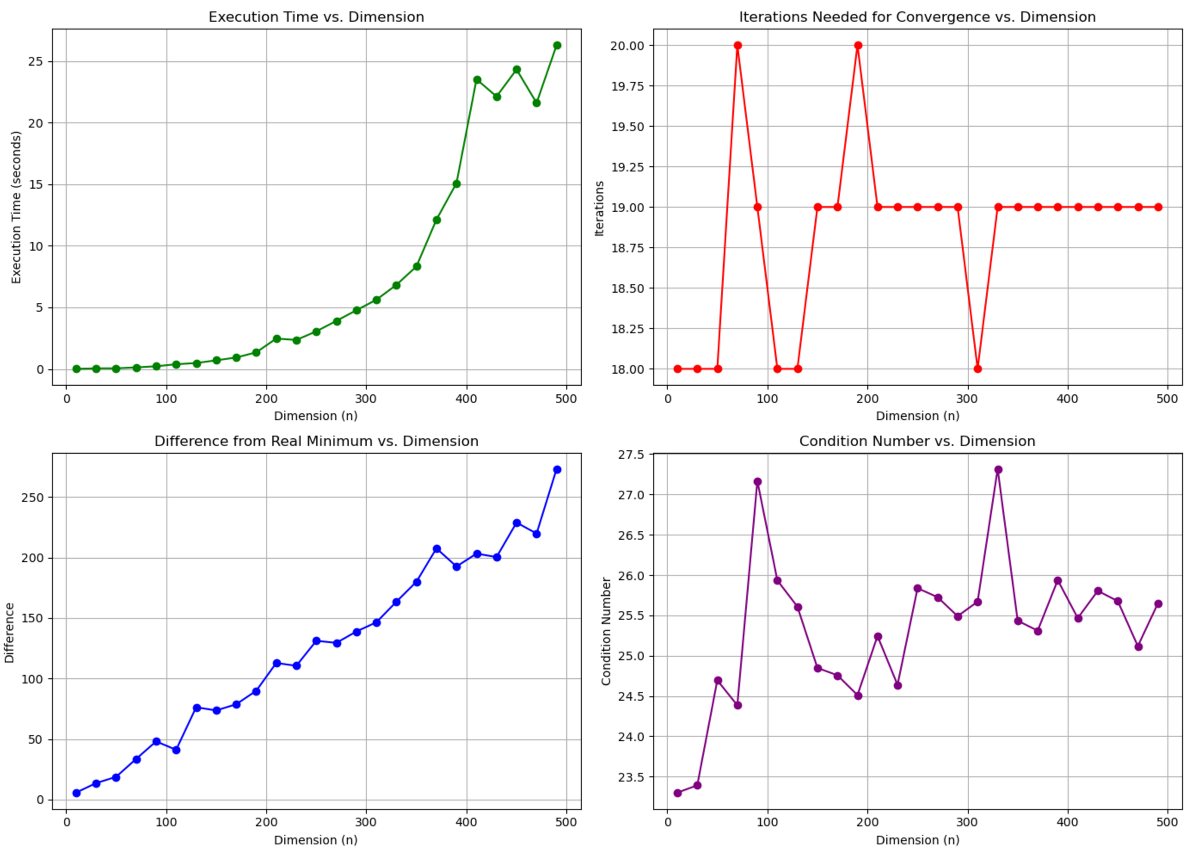

Figure 1: Performance metrics for varying dimensions $n$.

Oberving both the Table 1 as well as the Figure 1 we can get an idea of the performance of the general algorithm for A=0. We have plotted different metrics that we thought were interesting enough to compare based on the dimension of the iteration. Most interestingly we can see that the convergance tends to happen at iteration 19 for most of the dimensions. The Condition Number starts smaller and augments as n augments, but after a while it normalizes around 25. We also observe that the Execution Time is quite proportional to the dimension of the matrix.

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

Given the KKT matrix:

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

and the system to solve $M_{KKT} \begin{pmatrix} \delta x \\ \delta \lambda \\ \delta s \end{pmatrix} = - \begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix}$, our goal is to reduce the dimensionality of this system by eliminating certain variables, allowing us to use more efficient factorization techniques.

**Strategy 1: Isolate $\delta s$ from the 3rd Row of the KKT System**

In the first strategy, we isolate $\delta s$ directly from the third row of the KKT system. This row is given by:

$$S\delta\lambda + \Lambda\delta s = -r_3.$$

1. **Isolate $\delta s$**: We rearrange this equation to express $\delta s$ in terms of $\delta\lambda$:

   $$\delta s = \Lambda^{-1}(-r_3 - S\delta\lambda).$$

   Here, we assume that $\Lambda$ is invertible. Since $\Lambda$ is a diagonal matrix with the elements of $\lambda$ on its diagonal, $\Lambda$ is invertible if and only if each component $\lambda_i \neq 0$. This assumption is critical because if any $\lambda_i = 0$, $\Lambda$ would not be invertible, preventing us from isolating $\delta s$ in this way.

2. **Substitute $\delta s$ into the 2nd Row**: Next, we substitute this expression for $\delta s$ into the second row of the KKT system:

   $$-C^T\delta x + \delta s = -r_2.$$

   Plugging in the expression for $\delta s$ from above, we obtain:

   $$-C^T\delta x - \Lambda^{-1}S\delta\lambda = -r_2 + \Lambda^{-1}r_3.$$

   This gives us a system in terms of $\delta x$ and $\delta\lambda$ only.

3. **Resulting Reduced System**: After the substitution, we have a new linear system that depends only on $\delta x$ and $\delta\lambda$:

   $$\begin{pmatrix} G & -C \\ -C^T & -\Lambda^{-1}S \end{pmatrix} \begin{pmatrix} \delta x \\ \delta\lambda \end{pmatrix} = \begin{pmatrix} -r_1 \\ -r_2 + \Lambda^{-1}r_3 \end{pmatrix}.$$

4. **LDL$^T$ Factorization**: The resulting matrix is symmetric since $G$ is symmetric by hypothesis, and $\Lambda^{-1}S$ is diagonal. Because of this structure, we can apply LDL$^T$ factorization, a method suitable for symmetric matrices, which allows us to efficiently solve the system via forward and backward substitution with the resulting triangular matrices.

**Assumptions for Strategy 1**:

- **Invertibility of $\Lambda$**: We assume that each $\lambda_i \neq 0$ so that $\Lambda$ is invertible. This ensures we can isolate $\delta s$ from the third row.
- **Symmetry of the Reduced Matrix**: The reduced matrix is symmetric because $G$ is symmetric by assumption, and $\Lambda^{-1}S$ is a diagonal matrix.
- **Non-singularity for LDL$^T$ Factorization**: For LDL$^T$ factorization to work, the matrix needs to be nonsingular. This requires that the reduced matrix, formed after substitutions, is nonsingular.

**Strategy 2: Isolate $\delta s$ from the 2nd Row of the KKT System**

In this strategy, we isolate $\delta s$ from the second row of the KKT system first, then use this expression in subsequent rows to simplify the system.

1. **Isolate $\delta s$**: Starting with the second row of the KKT system:

   $$-C^T\delta x + \delta s = -r_2,$$

   we rearrange to express $\delta s$ in terms of $\delta x$:

   $$\delta s = -r_2 + C^T\delta x.$$

2. **Substitute $\delta s$ into the 3rd Row**: We now substitute this expression for $\delta s$ into the third row:

$$S\delta\lambda + \Lambda\delta s = -r_3.$$

   Substituting for $\delta s$, we get:

$$S\delta\lambda + \Lambda(-r_2 + C^T\delta x) = -r_3.$$

   Solving for $\delta\lambda$, we have:

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

   Here, we assume that $S$ is invertible. Since $S$ is a diagonal matrix with the elements of $s$ on its diagonal, this assumption implies that each $s_i \neq 0$.

3. **Resulting Reduced System**: Now that we have $\delta\lambda$ and $\delta s$ in terms of $\delta x$, we substitute both into the first row of the KKT system:

$$G\delta x - C\delta\lambda = -r_1.$$

   Substituting the expression for $\delta\lambda$, we obtain:

$$(G + CS^{-1}\Lambda C^T)\delta x = -r_1 - CS^{-1}(-r_3 + \Lambda r_2).$$

   This is a reduced system in terms of $\delta x$ alone, where we define $\hat{G} = G + CS^{-1}\Lambda C^T$ and $\hat{r} = -CS^{-1}(-r_3 + \Lambda r_2)$.

4. **Cholesky Factorization:** The reduced matrix $\hat{G}$ is symmetric since $G$ is symmetric and $CS^{-1}\Lambda C^T$ is also symmetric. Therefore, we can apply Cholesky factorization to solve this system if $\hat{G}$ is also positive definite.

**Assumptions for Strategy 2**:

- **Invertibility of $S$**: We assume $s_i \neq 0$ for all $i$ so that $S$ is invertible, which allows us to isolate $\delta\lambda$.
- **Symmetry of the Reduced Matrix $\hat{G}$**: The symmetry of $\hat{G}$ holds because $G$ is symmetric, and $CS^{-1}\Lambda C^T$ is symmetric as well. This property is required for Cholesky factorization.
- **Positive Definiteness of $\hat{G}$**: Cholesky factorization also requires that $\hat{G}$ be positive definite. Positive definiteness of $\hat{G}$ is guaranteed if $G$ is positive definite and all elements of $S$ and $\Lambda$ are positive, which implies that both $s_i$ and $\lambda_i$ are positive for all $i$.

**Summary of the Approaches**
- **Strategy 1**: We isolate $\delta s$ from the third row, substitute into the second row, and solve a reduced system using LDL$^T$ factorization. This approach requires invertibility of $\Lambda$ and symmetry of the reduced matrix.
- **Strategy 2**: We isolate $\delta s$ from the second row, substitute into the third row to solve for $\delta\lambda$, and finally solve for $\delta x$ using Cholesky factorization. This approach requires invertibility of $S$, symmetry, and positive definiteness of the reduced ma

**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.

We have tested both strategies for several dimensions once again following the methodology in C3. Figure 2 shows an overall overview of the comparison of the strategies. We can see that both augment execution time as n augments although Cholseky performs faster. The mean number of iterations to reach convergence is also around 19 for both and the difference from Real Minimum is approximately also the same. Cholesky shows condition number of 1 always whereas the condition number of $LDL^T$ augments a lot, thus the log scale.

See table 2 for further comparison of the condition number for the 3 strategies:

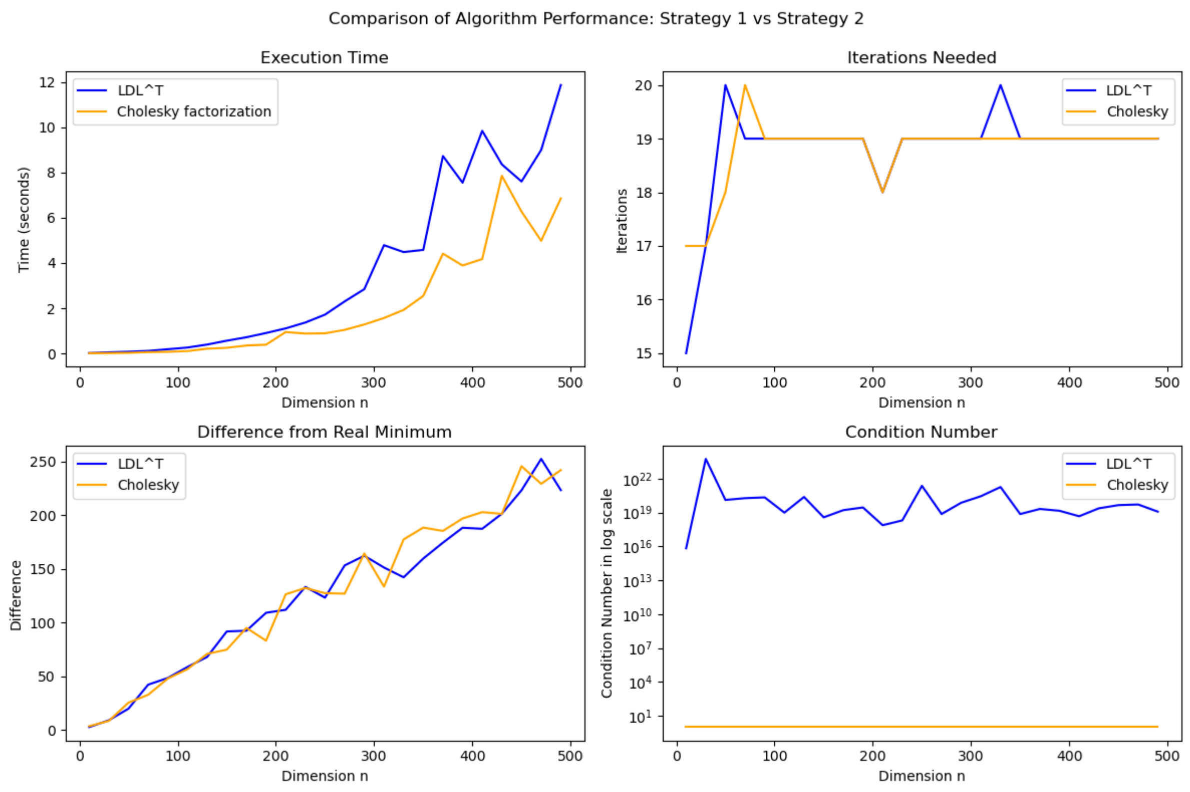

Figure 2: Performance metrics for varying dimensions $n$ comparing the two strategies.

Table 2: Comparison of Condition Numbers $k$ for C3, LDLT, and Cholseky

| **Dimension (n)** | $k_1$ **(C3 Results)** | $k_1$ $(LDL^T)$ | $k_1$ **(Cholesky)** |
|---|---|---|---|
| 10 | 23.300590 | $6.74 \times 10^{15}$ | 1.000000 |
| 30 | 23.391767 | $5.48 \times 10^{23}$ | 1.000000 |
| 50 | 24.694641 | $1.27 \times 10^{20}$ | 1.000000 |
| 70 | 24.384718 | $1.84 \times 10^{20}$ | 1.000000 |
| 90 | 27.167120 | $2.14 \times 10^{20}$ | 1.000000 |
| 190 | 24.510761 | $2.75 \times 10^{19}$ | 1.000000 |
| 290 | 25.489691 | $7.26 \times 10^{19}$ | 1.000000 |
| 390 | 25.936782 | $1.40 \times 10^{19}$ | 1.000000 |
| 490 | 25.647024 | $1.18 \times 10^{19}$ | 1.000000 |

## 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. Take as initial condition x0 $= (0, \dots, 0)$ and s0 $= \gamma_0 = \lambda_0 = (1, \dots, 1)$ for all problems.

To solve the general optimization problem, we implement a predictor-corrector approach using the KKT (Karush-Kuhn-Tucker) conditions, solved through direct linear system resolution. This implementation utilizes the `numpy.linalg.solve` function for efficient handling of dense matrix systems.

The data for each optimization problem is loaded from a collection of files, specifically:

- `G.dad` for the Hessian matrix $G$,
- `g.dad` for the gradient vector $g$,
- `A.dad` and `b.dad` for equality constraint matrices $A$ and $b$,
- `C.dad` and `d.dad` for inequality constraint matrices $C$ and $d$.

**Steps of the Predictor-Corrector Algorithm**

The following key steps outline the main components of the algorithm:

1. KKT Matrix Construction:

In each iteration, we construct the KKT matrix, using the function `create_KKT_matrix`. This matrix is essential for both the predictor and corrector steps.

2. Residual Computation:

Using the function `compute_residuals`, we generate the right-hand side (RHS) for the KKT system. These residuals are used to calculate the direction in the predictor step.

3. Predictor Step:

We solve for $\Delta z$ using the predictor substep:

$$\Delta z = \texttt{predictor\_substep}(M_{\text{KKT}}, \text{rhs})$$

In this step, `numpy.linalg.solve` should be used within the function `predictor_substep` to solve the system directly with $M_{\text{KKT}}$ and the RHS.

4. Step-Size Correction:

The step size $\alpha$ and the central path parameter $\sigma$ are computed to adjust the progress of the algorithm.

5. Corrector Step:

To handle the central path adjustment, we set up a corrector RHS, `rhs_corrector`:

$$\Delta z_{\text{corr}} = \texttt{corrector\_substep}(M_{\text{KKT}}, \text{rhs\_corrector})$$

This function should also use `numpy.linalg.solve` to solve the linear system directly.

After obtaining the correction direction, we update the variable $z_0$ based on the computed step size $\alpha_{\text{corr}}$ from the corrector step:

$$z_1 = \texttt{update\_step}(z_0, \Delta z_{\text{corr}}, \alpha_{\text{corr}})$$

Using the stopping criteria the algorithm checks if the residuals have fallen below a small threshold $\epsilon$, which serves as the convergence criterion.

The results obtained by the code using the vectors and matrices given in “optpr1” and “optpr2” are the following:

Table 3: Results of C5 for OPTPR1 and OPTPR2

| **Dataset** | **Dimension (n)** | **Execution Time (s)** | **Iterations** | **Optimal $f(x)$** | $k_1$ |
|---|---|---|---|---|---|
| **OPTPR 1** | 100 | 8.08 | 26 | 11590.72 | $5.37 \times 10^{18}$ |
| **OPTPR 2** | 1000 | 317.90 | 29 | 1087511.57 | $1.45 \times 10^{18}$ |

Some conclusions that we can extract from C5 are that:

Computational Time: Optpr2 requires significantly more computational time than Optpr1, highlighting its higher computational complexity.

Objective Function Value: Optpr1 achieves a lower minimum objective function value compared to Optpr2, suggesting that Optpr1 is a comparatively easier optimization problem to solve.

Conditioning of the KKT Matrix: The KKT matrix for Optpr2 has a much higher condition number than that of Optpr1, indicating that Optpr2 is a more ill-conditioned problem, which may pose additional challenges during optimization.

Convergence: Although both problems converge within the specified tolerance, Optpr2 requires slightly more iterations to reach convergence than Optpr1.

**T3:** Isolate $\delta_s$ from the 4th row of MKKT and substitute into the 3rd row. Justify that this procedure leads to a linear system with a symmetric matrix.

To solve the general KKT system in the presence of both equality and inequality constraints, we begin with the following matrix form, where $A \neq 0$:

$$M_{KKT} = \begin{pmatrix} G & -A & -C & 0 \\ -A^T & 0 & 0 & 0 \\ -C^T & 0 & 0 & I \\ 0 & 0 & S & \Lambda \end{pmatrix} \begin{pmatrix} \delta x \\ \delta\gamma \\ \delta\lambda \\ \delta s \end{pmatrix} = - \begin{pmatrix} r_L \\ r_A \\ r_C \\ r_s \end{pmatrix}$$

Writing this system as equations, we have:

$$\begin{cases} G\delta x - A\delta\gamma - C\delta\lambda = -r_L, \\ -A^T\delta x = -r_A, \\ -C^T\delta x + \delta s = -r_C, \\ S\delta\lambda + \Lambda\delta s = -r_s. \end{cases}$$

**Step 1:** Isolate $\delta s$ from the 4th Row
We start by isolating $\delta s$ from the fourth equation:

$$\delta s = -\Lambda^{-1}(r_s + S\delta\lambda).$$

This step assumes that $\Lambda$ is invertible, which requires $\lambda_i \neq 0$ for all $i$.

**Step 2:** Substitute $\delta s$ into the 3rd Row
Substituting the expression for $\delta s$ from above into the third equation, we obtain:

$$-C^T\delta x - \Lambda^{-1}(r_s + S\delta\lambda) = -r_C.$$

Rearranging, this becomes:

$$-C^T\delta x - \Lambda^{-1}S\delta\lambda = -r_C + \Lambda^{-1}r_s.$$

The resulting system in matrix form is:

$$\begin{pmatrix} G & -A & -C \\ -A^T & 0 & 0 \\ -C^T & 0 & -\Lambda^{-1}S \end{pmatrix} \begin{pmatrix} \delta x \\ \delta\gamma \\ \delta\lambda \end{pmatrix} = \begin{pmatrix} -r_L \\ -r_A \\ -r_C + \Lambda^{-1}r_s \end{pmatrix}.$$

This matrix is symmetric, which can be shown by examining the individual blocks:

- $G$ is symmetric by assumption.
- The diagonal block $-\Lambda^{-1}S$ is symmetric as it arises from the product of two diagonal matrices $\Lambda^{-1}$ and $S$.
- The symmetry of $M_{KKT}$ is thus preserved in the reduced system.

Given that the reduced matrix is symmetric, we can apply $LDL^T$ factorization. This method allows us to decompose the matrix into a product of a lower triangular matrix $L$, a diagonal matrix $D$, and the transpose $L^T$. Once factorized, we can solve the system efficiently using forward and backward substitution.

This approach demonstrates how we can reduce the system's dimensionality, maintain symmetry, and solve the system using efficient matrix factorization techniques.

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

In table 4 we can see the results of the different metrics using the matrices and vectors gieven in the project minimized using the LDLT factorization. We can observe that the computation time has decreased significantly.

In this section we compare two algorithms: C5, which uses the `numpy.linalg.solve` factorization function, and C6, which relies on LDLT factorization. The following observations summarize their relative performance and characteristics:

Table 4: Results of C6 for OPTPR1 and OPTPR2

| **Dataset** | **Dimension (n)** | **Execution Time (s)** | **Iterations** | **Optimal** $f(x)$ | $k_1$ |
|---|---|---|---|---|---|
| **OPTPR 1** | 100 | 0.4157 | 39 | 11590.72 | $1.22506 \times 10^{24}$ |
| **OPTPR 2** | 1000 | 61.68 | 29 | 1087511.57 | $9.295 \times 10^{24}$ |

1. The C5 algorithm generally ran faster than C6, as the LDLT factorization steps in C6 introduced extra computational cost.

2. Both algorithms produced nearly identical minimum values, demonstrating that each method reliably converges to a similar solution.

3. A significant difference was observed in the condition number of the KKT matrix: C6 often yielded a higher condition number, suggesting that the matrix may be more ill-conditioned. This increases sensitivity to small variations in input, meaning that C6 solutions could potentially be less stable for some problems.

4. C6 also required more iterations to converge than C5, likely due to its additional LDLT factorization steps and step-size correction sub-steps, which contributed to longer computation times.

In summary, both the `numpy.linalg.solve` and LDLT factorizations are effective for solving optimization problems, though they each offer distinct advantages:

- The C5 algorithm is faster and may be preferred when computational efficiency is critical.
- The C6 algorithm, employing LDLT factorization, may be better suited to problems requiring greater numerical stability, particularly where the condition number of the KKT matrix is a concern.

Finally, it is noteworthy that the Cholesky factorization method was found to be both the fastest and most accurate approach, yielding a solution with a better condition number overall.