Using GMRES in a Constraint Optimization Algorithm

Nenna Giulio, Ornella Elena Grassi

Computational Linear Algebra For Large Scale Problems

The aim of this homework is to implement the GMRES algorithm (Generalized minimal residual method) for solving linear systems and test its effectiveness in a quadratic constraint optimization problem. In particular, we will introduce an Interior Point Method (IPM) for solving a quadratic constraint optimization problem which is an iterative method that requires to solve two sparse large scale linear systems at each iteration. We will use our implementation of the GMRES algorithm to solve those linear systems and measure both the accuracy and the speed of the method compared to other linear solvers.

1 Problem Setting

The prototype of constrained quadratic programming problem that we are dealing with is the following:

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathbb{R}^n} \quad & \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\mathbf{Q}\boldsymbol{x} + \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ \text{subject to} \quad & \mathbf{A}\boldsymbol{x} = \boldsymbol{b}, \\ & \boldsymbol{x} \geq \mathbf{0}, \end{aligned} \tag{1.1}$$

Where $\mathbf{Q} \in \mathbb{R}^{n\times n}$ is a symmetric positive semi-definite matrix, $\boldsymbol{c} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{K\times n}$ and $\boldsymbol{b} \in \mathbb{R}^K$ define K equality constraints. Problem 1.1 can be reformulated without equality constraints if we consider that:

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b} \iff \begin{cases} \mathbf{A}\boldsymbol{x} \geq \boldsymbol{b}, \\ \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}. \end{cases} \tag{1.2}$$

Hence we can write a new problem, equivalent to 1.1, in which there are only inequality constraints:

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathbb{R}^n} \quad & \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\mathbf{Q}\boldsymbol{x} + \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ \text{subject to} \quad & \hat{\mathbf{A}}\boldsymbol{x} \geq \hat{\boldsymbol{b}}, \end{aligned} \tag{1.3}$$

where $\hat{\mathbf{A}} \in \mathbb{R}^{(2K+n)\times n}$ and $\hat{\boldsymbol{b}} \in \mathbb{R}^{2K+n}$ are defined as

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{A} \\ -\mathbf{A} \\ \mathbf{I}_{n\times n} \end{bmatrix}, \qquad \hat{\boldsymbol{b}} = \begin{bmatrix} \boldsymbol{b} \\ -\boldsymbol{b} \\ \mathbf{0}_n \end{bmatrix}. \tag{1.4}$$

We will solve the constraint quadratic programming problem as formulated in 1.3 since this formulation is valid both for problems with and without equality constraints.

We can now define the corresponding Lagrangian function as:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) := \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\mathbf{Q}\boldsymbol{x} + \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + \hat{\boldsymbol{\lambda}}^{\mathrm{T}}(\hat{\mathbf{A}}\boldsymbol{x} - \hat{\boldsymbol{b}}) \tag{1.5}$$

where $\hat{\boldsymbol{\lambda}} \in \mathbb{R}^{2K+n}$ is the vector of Lagrangian multipliers associated with the inequality constraints and we can also write the corresponding KKT conditions:

$$\begin{cases} \mathbf{Q}\boldsymbol{x} + \boldsymbol{c} - \hat{\mathbf{A}}^{\mathrm{T}}\hat{\boldsymbol{\lambda}} = \mathbf{0} & \text{Stationarity condition,} \\ \hat{\mathbf{A}}\boldsymbol{x} - \hat{\boldsymbol{b}} \geq \mathbf{0} & \text{Primal feasibility,} \\ \hat{\boldsymbol{\lambda}} \geq \mathbf{0} & \text{Dual feasibility,} \\ \hat{\boldsymbol{\lambda}}^{\mathrm{T}}\left(\hat{\mathbf{A}}\boldsymbol{x} - \hat{\boldsymbol{b}}\right) = \mathbf{0} & \text{Complementary slackness condition.} \end{cases} \tag{1.6}$$

We can manipulate the system (1.6) in order to simplify the system of equations. In particular, we define a *slack variable* $\boldsymbol{y} := \hat{\mathbf{A}}\boldsymbol{x} - \hat{\boldsymbol{b}} \in \mathbb{R}^{2K+n}$ such that the KKT conditions becomes

$$\begin{cases} \mathbf{Q}\boldsymbol{x} + \boldsymbol{c} - \hat{\mathbf{A}}^{\mathrm{T}}\hat{\boldsymbol{\lambda}} = \mathbf{0} & \text{Stationarity condition,} \\ \boldsymbol{y} \geq \mathbf{0} & \text{Primal feasibility,} \\ \hat{\boldsymbol{\lambda}} \geq \mathbf{0} & \text{Dual feasibility,} \\ \hat{\mathbf{A}}\boldsymbol{x} - \hat{\boldsymbol{b}} - \boldsymbol{y} = \mathbf{0} & \text{Slack variable constraint,} \\ \hat{\boldsymbol{\lambda}}^{\mathrm{T}}\boldsymbol{y} = \mathbf{0} & \text{Complementary slackness condition.} \end{cases} \tag{1.7}$$

2 Predictor-Corrector IPM

Let's implement a Predictor-Corrector Interior Point Method for finding the solutions of the system (1.7). We can rewrite the problem as finding the zeros of the function $F : \mathbb{R}^{4K+3n} \to \mathbb{R}^{4K+3n}$

$$F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\lambda}) = \begin{bmatrix} \mathbf{Q}\boldsymbol{x} + \boldsymbol{c} - \hat{\mathbf{A}}^{\mathrm{T}}\hat{\boldsymbol{\lambda}} \\ \hat{\mathbf{A}}\boldsymbol{x} - \hat{\boldsymbol{b}} - \boldsymbol{y} \\ \mathbf{Y}\hat{\boldsymbol{\Lambda}}\boldsymbol{e} \end{bmatrix} = \mathbf{0} \tag{2.1}$$

with the inequality constraints $\boldsymbol{y} = \hat{\mathbf{A}}\boldsymbol{x} - \hat{\boldsymbol{b}} \geq \mathbf{0}$ and $\hat{\boldsymbol{\lambda}} \geq \mathbf{0}$, where we have defined

$$\mathbf{Y} = \begin{bmatrix} y_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & y_n \end{bmatrix} \in \mathbb{R}^{(2K+n)\times(2K+n)}, \quad \hat{\boldsymbol{\Lambda}} = \begin{bmatrix} \hat{\lambda}_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \hat{\lambda}_n \end{bmatrix} \in \mathbb{R}^{(2K+n)\times(2K+n)},$$

and $\boldsymbol{e} = [1, \ldots, 1]^{\mathrm{T}} \in \mathbb{R}^{2K+n}$.

The main idea about **Interior Point Methods** is that, given a starting point $(\mathbf{x_0}, \mathbf{y_0}, \lambda_\mathbf{0})$, we want to stay away from the boundary of the feasible set for as many iterations as possible since getting stuck on the boundary in early iterations kills the convergence speed of the algorithm. What we want instead is for the solution to gently approach the boundary in order to maximize the convergence speed.
This is accomplished in two main steps: the *Predictor* step and the *Corrector* step. In the Prediction step, given a current feasible solution $(\mathbf{x_k}, \mathbf{y_k}, \lambda_\mathbf{k})$, we compute a simple *Newton Step* to solve eq. 2.1. We then estimate how far away we are from the feasible set and compute the *Corrector* step which is simply a correction of the Newton step computed above needed to keep the boundary of the feasible set at a desired distance that will decrease during iterations.

Since we will solve Equation (2.1) with the Newton method, we will need the Jacobian of F, which is computed as

$$\mathbf{J}_F = \begin{bmatrix} \mathbf{Q} & \mathbf{0} & -\hat{\mathbf{A}}^{\mathrm{T}} \\ \hat{\mathbf{A}} & -\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \hat{\boldsymbol{\Lambda}} & \mathbf{Y} \end{bmatrix}.$$

The resulting Predictor-Corrector IPM algorithm is the following:

- We start from an initial point $(\boldsymbol{x}_0, \boldsymbol{y}_0, \hat{\boldsymbol{\lambda}}_0)$ with the only request that $\boldsymbol{y}_0$ and $\hat{\boldsymbol{\lambda}}_0$ satisfy the primal and dual strict feasibility conditions, i.e. $\boldsymbol{y}_0, \hat{\boldsymbol{\lambda}}_0 > 0$, and we choose $\boldsymbol{x}_0$ which solves the condition $\hat{\mathbf{A}}\boldsymbol{x}_0 - \hat{\boldsymbol{b}} - \boldsymbol{y}_0 = \mathbf{0}$ in the *least-square sense*.

- (**Predictor**) At step k, using the current three iterates $(\boldsymbol{x}_k, \boldsymbol{y}_k, \hat{\boldsymbol{\lambda}}_k)$, we compute the affine scaling step $(\Delta\boldsymbol{x}_k^{\text{aff}}, \Delta\boldsymbol{y}_k^{\text{aff}}, \Delta\hat{\boldsymbol{\lambda}}_k^{\text{aff}})$ using the following Newton step

$$\begin{bmatrix} \mathbf{Q} & \mathbf{0} & -\hat{\mathbf{A}}^{\mathrm{T}} \\ \hat{\mathbf{A}} & -\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \hat{\boldsymbol{\Lambda}}_k & \mathbf{Y}_k \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{x}_k^{\text{aff}} \\ \Delta\boldsymbol{y}_k^{\text{aff}} \\ \Delta\hat{\boldsymbol{\lambda}}_k^{\text{aff}} \end{bmatrix} = - \begin{bmatrix} \mathbf{Q}\boldsymbol{x}_k + \boldsymbol{c} - \hat{\mathbf{A}}^{\mathrm{T}}\hat{\boldsymbol{\lambda}}_k \\ \hat{\mathbf{A}}\boldsymbol{x}_k - \hat{\boldsymbol{b}} - \boldsymbol{y}_k \\ \mathbf{Y}_k\hat{\boldsymbol{\Lambda}}_k\boldsymbol{e} \end{bmatrix} \tag{2.2}$$

  where, again, we've defined $\mathbf{Y}_k = \operatorname{diag}((\boldsymbol{y}_k)_1, \ldots, (\boldsymbol{y}_k)_n)$ and $\hat{\boldsymbol{\Lambda}}_k = \operatorname{diag}((\hat{\boldsymbol{\lambda}}_k)_1, \ldots, (\hat{\boldsymbol{\lambda}}_k)_n)$.

- We then compute the affine step-length $\alpha_k^{\text{aff}} > 0$ in order to remain in the internal part of the feasible set by ensuring that $\boldsymbol{y}_k + \alpha_k^{\text{aff}}\Delta\boldsymbol{y}_k^{\text{aff}} > 0$ and $\hat{\boldsymbol{\lambda}}_k + \alpha_k^{\text{aff}}\Delta\hat{\boldsymbol{\lambda}}_k^{\text{aff}} > 0$. This is accomplished by setting the step-length as

$$\begin{aligned} \alpha_k^{\text{aff}} = \min\Bigg\{1,\ \min\Bigg\{ & -\frac{(\boldsymbol{y}_k)_i}{(\Delta\boldsymbol{y}_k^{\text{aff}})_i} \ :\ i = 1, \ldots, 2K+n,\ \ (\Delta\boldsymbol{y}_k^{\text{aff}})_i < 0 \Bigg\} \\ & \min\Bigg\{ -\frac{(\hat{\boldsymbol{\lambda}}_k)_i}{(\Delta\hat{\boldsymbol{\lambda}}_k^{\text{aff}})_i} \ :\ i = 1, \ldots, 2K+n,\ \ (\Delta\hat{\boldsymbol{\lambda}}_k^{\text{aff}})_i < 0 \Bigg\}\Bigg\} \end{aligned} \tag{2.3}$$

- We compute the affine complementarity measure μ_k^{aff} and the complementarity parameter σ_k as

$$\mu_k^{\text{aff}} = \frac{1}{n}\left(\boldsymbol{y}_k + \alpha_k^{\text{aff}}\Delta\boldsymbol{y}_k^{\text{aff}}\right)^{\mathrm{T}}\left(\hat{\boldsymbol{\lambda}}_k + \alpha_k^{\text{aff}}\Delta\hat{\boldsymbol{\lambda}}_k^{\text{aff}}\right) \tag{2.4}$$

$$\sigma_k = \left(\frac{\mu_k^{\text{aff}}}{\mu_k}\right)^3, \quad \mu_k = \frac{\boldsymbol{y}_k^{\mathrm{T}}\hat{\boldsymbol{\lambda}}_k}{n} \tag{2.5}$$

  Those are the measures that tell us "how far" we are from the boundary of the feasible set.

- (**Corrector**) We compute the affine corrector step $(\Delta\boldsymbol{x}_k, \Delta\boldsymbol{\lambda}_k, \Delta\boldsymbol{s}_k)$ using the following Newton step

$$\begin{bmatrix} \mathbf{Q} & \mathbf{0} & -\hat{\mathbf{A}}^{\mathrm{T}} \\ \hat{\mathbf{A}} & -\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \hat{\boldsymbol{\Lambda}}_k & \mathbf{Y}_k \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{x}_k \\ \Delta\boldsymbol{y}_k \\ \Delta\hat{\boldsymbol{\lambda}}_k \end{bmatrix} = - \begin{bmatrix} \mathbf{Q}\boldsymbol{x}_k + \boldsymbol{c} - \hat{\mathbf{A}}^{\mathrm{T}}\hat{\boldsymbol{\lambda}}_k \\ \hat{\mathbf{A}}\boldsymbol{x}_k - \hat{\boldsymbol{b}} - \boldsymbol{y}_k \\ \mathbf{Y}_k\hat{\boldsymbol{\Lambda}}_k\boldsymbol{e} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ -\Delta\mathbf{Y}_k^{\text{aff}}\Delta\hat{\boldsymbol{\Lambda}}_k^{\text{aff}}\boldsymbol{e} + \sigma_k\mu_k\boldsymbol{e} \end{bmatrix} \tag{2.6}$$

  where $\Delta\mathbf{Y}_k = \operatorname{diag}((\Delta\boldsymbol{y}_k)_1, \ldots, (\Delta\boldsymbol{y}_k)_n)$ and $\Delta\hat{\boldsymbol{\Lambda}}_k = \operatorname{diag}((\Delta\hat{\boldsymbol{\lambda}}_k)_1, \ldots, (\Delta\hat{\boldsymbol{\lambda}}_k)_n)$.

- We then compute the step-length $\alpha_k > 0$ as before

$$\begin{aligned} \alpha_k = \min\Bigg\{1,\ \min\Bigg\{ & -\frac{\tau_k(\boldsymbol{y}_k)_i}{(\Delta\boldsymbol{y}_k)_i} \ :\ i = 1, \ldots, n,\ \ (\Delta\boldsymbol{y}_k)_i < 0 \Bigg\} \\ & \min\Bigg\{ -\frac{\tau_k(\hat{\boldsymbol{\lambda}}_k)_i}{(\Delta\hat{\boldsymbol{\lambda}}_k)_i} \ :\ i = 1, \ldots, n,\ \ (\Delta\hat{\boldsymbol{\lambda}}_k)_i < 0 \Bigg\}\Bigg\} \end{aligned} \tag{2.7}$$

  where $\tau_k \in (0,1)$ controls how far we back off from the maximum step for which the inequality constraints are satisfied, i.e. $\boldsymbol{y}_k + \alpha_k\Delta\boldsymbol{y}_k \geq (1-\tau_k)\boldsymbol{y}_k$ and $\hat{\boldsymbol{\lambda}}_k + \alpha_k\Delta\hat{\boldsymbol{\lambda}}_k \geq (1-\tau_k)\hat{\boldsymbol{\lambda}}_k$.

- (**Update**) We then update the values for the next iteration:

$$\begin{cases} \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\Delta\boldsymbol{x}_k, \\ \boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \alpha_k\Delta\boldsymbol{y}_k, \\ \hat{\boldsymbol{\lambda}}_{k+1} = \hat{\boldsymbol{\lambda}}_k + \alpha_k\Delta\hat{\boldsymbol{\lambda}}_k. \end{cases} \tag{2.8}$$

2.1 Solution of the linear system

At each step of the algorithm, we have to solve the linear system

$$\begin{bmatrix} \mathbf{Q} & \mathbf{0} & -\hat{\mathbf{A}}^{\mathrm{T}} \\ \hat{\mathbf{A}} & -\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{\Lambda}}_k & \mathbf{Y}_k \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x} \\ \Delta \boldsymbol{y} \\ \Delta \hat{\boldsymbol{\lambda}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \\ \boldsymbol{r}_3 \end{bmatrix}. \tag{2.9}$$

Since the dimension of this linear system is very large, we need to split the equations in order to make the problem more manageable. From the third set of equations we get

$$\Delta \boldsymbol{y} = \hat{\mathbf{\Lambda}}^{-1} \boldsymbol{r}_3 - \mathbf{D}^{-1} \Delta \hat{\boldsymbol{\lambda}}. \tag{2.10}$$

where $\mathbf{D} = \mathbf{Y}^{-1} \hat{\mathbf{\Lambda}}$. So, substituting 2.10 into 2.9, we obtain the following augmented system:

$$\begin{bmatrix} \mathbf{Q} & -\hat{\mathbf{A}}^{\mathrm{T}} \\ \hat{\mathbf{A}} & \mathbf{D}^{-1} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x} \\ \Delta \hat{\boldsymbol{\lambda}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 + \hat{\mathbf{\Lambda}}^{-1} \boldsymbol{r}_3 \end{bmatrix}. \tag{Aug}$$

Moreover, we can exploit the new formulation to solve the system just by substitution. Indeed, if we apply $\mathbf{D}$ to the second equation, we get

$$\Delta \hat{\boldsymbol{\lambda}} = \mathbf{D} \left(\boldsymbol{r}_2 + \hat{\mathbf{\Lambda}}^{-1} \boldsymbol{r}_3 - \hat{\mathbf{A}} \Delta \boldsymbol{x} \right), \tag{2.11}$$

and, if we substitute back in the first equation of (Aug), we obtain the final system

$$\begin{aligned} \Delta \boldsymbol{x} &= \left(\mathbf{Q} + \hat{\mathbf{A}}^{\mathrm{T}} \mathbf{D} \hat{\mathbf{A}} \right)^{-1} \left(\boldsymbol{r}_1 + \hat{\mathbf{A}}^{\mathrm{T}} \mathbf{D} \left(\boldsymbol{r}_2 + \hat{\mathbf{\Lambda}}^{-1} \boldsymbol{r}_3 \right) \right) \\ &= \left(\mathbf{Q} + \hat{\mathbf{A}}^{\mathrm{T}} \mathbf{D} \hat{\mathbf{A}} \right)^{-1} \left(\boldsymbol{r}_1 + \hat{\mathbf{A}}^{\mathrm{T}} \left(\mathbf{D} \boldsymbol{r}_2 + \mathbf{Y}^{-1} \boldsymbol{r}_3 \right) \right). \end{aligned} \tag{2.12}$$

In (2.12) the matrix of the linear system is symmetric and positive semi-definite since $\mathbf{Q}$ and $\mathbf{D}$ are symmetric and positive semi-definite [1], so we will use the `pcg` solver. To summarize, this approach solves the system by solving for $\Delta \boldsymbol{x}$ and then by substitution in the other two equations as presented in the following system:

$$\begin{cases} \Delta \boldsymbol{x} &= \left(\mathbf{Q} + \hat{\mathbf{A}}^{\mathrm{T}} \mathbf{D} \hat{\mathbf{A}} \right)^{-1} \left(\boldsymbol{r}_1 + \hat{\mathbf{A}}^{\mathrm{T}} \left(\mathbf{D} \boldsymbol{r}_2 + \mathbf{Y}^{-1} \boldsymbol{r}_3 \right) \right), \\ \Delta \hat{\boldsymbol{\lambda}} &= \mathbf{D} \left(\boldsymbol{r}_2 + \hat{\mathbf{\Lambda}}^{-1} \boldsymbol{r}_3 - \hat{\mathbf{A}} \Delta \boldsymbol{x} \right), \\ \Delta \boldsymbol{y} &= \hat{\mathbf{\Lambda}}^{-1} \boldsymbol{r}_3 - \mathbf{D}^{-1} \Delta \hat{\boldsymbol{\lambda}}. \end{cases} \tag{Sus}$$

3 GMRES Method

The Generalized Minimal RESiduals (GMRES) method is an iterative approach that utilizes Krylov subspaces to transform a high-dimensional problem into a series of smaller dimensional problems. It is commonly used to solve linear systems of the form $Ax = b$, where A is an invertible $n \times n$ matrix and b is a vector of length n. GMRES starts with an initial guess for the solution x_0 and constructs a sequence of approximations $x_0, x_1, x_2, \ldots$ that converge to the true solution.

Instead of solving the system $Ax = b$ directly, GMRES finds an approximation of the solution $x_m \in \mathcal{K}_m(A, r_0)$, where $\mathcal{K}_m(A, r_0)$ is the Krylov subspace of dimension m. Moreover x_m minimizes the residual $r_m = \|b - Ax_m\|_2$, iteratively improving the solution and reducing the residual.

Hence the GMRES algorithm searches for the solution in a particular space called **Krilov Space**, where

$$\mathcal{K}_m(A, r_0) = \mathrm{span}(r_0, Ar_0, A^2 r_0, ..., A^{m-1} r_0)$$

[1] $\mathbf{D} = \mathbf{Y}^{-1} \hat{\mathbf{\Lambda}}$ is indeed SPD because, by the primal and dual feasibility conditions, $y_k > 0$ and $\hat{\lambda}_k > 0$.

We use Krilov spaces to find the approximation of the solution since, thanks to the Cayley-Hamilton theorem, we know that the inverse of a matrix can be represented as a polynomial of such matrix:

$$A^{-1} = -\frac{1}{c_0}(A^{n-1} + c_{n-1}A^{n-2} + ... + c_1 I)$$

This property tells us that $\bar{x} = A^{-1}b$ can be approximated by a linear combination of basis vectors of the Krylov space, in particular if $x_0 = 0$, $r_0 = b$ and $\mathcal{K}_m(A, r_0) = \mathcal{K}_m(A, b)$ then $\bar{x} \sim x_m \in \mathcal{K}_m(A, r_0)$ with $m < n$.

This way, GMRES gradually converges towards the true solution of the linear system. The convergence criterion is typically based on the norm of the residual, and the algorithm terminates when the residual falls below a certain threshold or after a specified number of iterations. In many situations, this happens when m is much smaller than n.

The GMRES algorithm uses the Arnoldi iteration with Gram-Schmidt orthonormalization to generate an orthonormal basis for the Krylov subspace. Given a generic Krylov subspace $\mathbb{K}_m(A, v_1)$, with v_1 nonvanishing vector of norm 1, the Arnoldi's procedure allows to build an orthonormal basis of $\mathbb{K}_m$, producing the two key matrices V_m, the $(n) \times m$ matrix with column vectors $v_1, v_2, \ldots, v_m$ and $\bar{H}_m$, a $(m+1)\times m$ upper Hessenberg matrix. The relations $AV_m = V_{m+1}\bar{H}_m$ and $V_m^T A V_m = H_m$ holds true, H_m obtained by deleting the last row from $\bar{H}_m$. The solution at iteration m is obtained by $x_m = x_0 + V_m y_m$ for some $y_m \in \mathbb{R}^m$. In particular, the vector of coefficients y_m is obtained in order to minimize the euclidean norm of the residual

$$\begin{aligned} r_m = b - Ax_m &= b - A(x_0 + V_m y_m) = \\ &= b - Ax_0 - AV_m y_m = \\ &= r_0 - V_{m+1}\bar{H} y_m = \\ &= \beta v_1 - V_{m+1}\bar{H} y_m. \end{aligned}$$

The orthogonality condition $r_m \perp L = AK_m(A, r_0)$ allows to derive the following equality

$$(AV_m)^T(V_{m+1}(\beta v_1 - V_{m+1}\bar{H} y_m) = 0 \tag{3.1}$$

$$\bar{H}_m^T \beta e_1 = \bar{H}_m^T \bar{H}_m y_m. \tag{3.2}$$

Since solving this linear system would be computationally inefficient, the same condition is obtained by minimizing the objective function

$$J(y) = \|r_m\|_2^2 = \|b - Ax_m\|_2^2.$$

Thus the problem of finding the y_m coefficients is a least square errors problem of dimensions $m+1$:

$$y_m = \arg\min_{y \in \mathbb{R}^m} \|\beta e_1 - \bar{H}_m y\|_2. \tag{3.3}$$

To find the solution of the least square problems means to find the QR factorization of the matrix $\bar{H}_m$, i.e. $\bar{H}_m = Q_m^T \bar{R}_m$ where $Q_m \in \mathbb{R}^{(m+1)\times(m+1)}$ is an orthogonal matrix and $\bar{R}_m \in \mathbb{R}^{(m+1)\times m}$. The minimizing problem can be rewritten as

$$\begin{aligned} \|\beta e_1 - \bar{H}_m y\|_2 &= \|\beta e_1 - Q_m^T \bar{R}_m y\|_2 \\ &= \|Q_m^T Q_m \beta e_1 - Q_m^T \bar{R}_m y\|_2 \\ &= \|Q_m^T (Q_m \beta e_1 - \bar{R}_m y)\|_2 \\ &= \|\bar{g}_m - \bar{R}_m y\|_2. \end{aligned} \tag{3.4}$$

Here the $\bar{g}_m$ vector is given by $\bar{g}_m = Q_m \beta e_1 = \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_{m+1} \end{bmatrix}$.

As a result, the vector y_m is therefore obtained by

$$y_m = R_m^{-1} g_m \tag{3.5}$$

where R_m is $m \times m$ upper triangular matrix obtained by deleting the $(m+1) - th$ row of $\bar{R}_m$ and g_m is the $m - dimensional$ vector given by the first m rows of $\bar{g}_m$. Moreover, the r_m vector, at iteration m is given by

$$b - Ax_m = V_{m+1}(\beta e_1 - \bar{H}_m y_m) = V_{m+1} Q_m^T (\gamma_{m+1} e_{m+1})$$

and

$$\|b - Ax_m\| = |\gamma_{m+1}|. \tag{3.6}$$

We have seen that, in order to compute y_m we need to compute the QR-factorization of the matrix $\bar{H}_m$. The matrix $\bar{H}_m$ is generated iteratively column by column by the Arnoldi alorithm and its dimension is not known until convergence. Is there a way to compute the QR-factorization *on the fly* while the matrix $\bar{H}_m$ is being constructed? The answer is yes and this is done by computing a Givens Rotation matrix each time a new column of $\bar{H}_m$ is constructed and then storing it in the Q_m matrix. The Algorithm below shows a version of the GMRES algorithm that allows us to compute the QR-factorization on the fly.

Algorithm 1 GMRES with *On-the-fly* QR factorization

Input: A, b, x_0, `tol`, `maxiter`
$r_0 \leftarrow b - Ax_0$, `res` $\leftarrow \|r_0\|_2$, $v_1 \leftarrow r_0/$`res` $V \leftarrow v_1$
$\bar{R} \leftarrow (Av_1)^T v_1$, $Q \leftarrow 1$ ▷ Factorization matrices initialization
$m \leftarrow 1$
while `res`$/b \leq$ `tol` and $m - 1 \leq$ `maxiter` **do**
  $\omega_m = Av_m$ ▷ Initialize new basis vector for Krilov Space
  for $j = 1, \ldots, m$ **do** ▷ Orthogonalization through GS
    $h_{j,m} \leftarrow \langle \omega_m, v_j \rangle$
    $\omega_m \leftarrow h_{j,m} v_j$
  end for
  $h_{m+1,m} \leftarrow \|\omega_m\|$
  if $h_{m+1,m} = 0$ **then** ▷ Check if max dimension of basis is reached
    Break
  end if
  $v_{m+1} \leftarrow \omega_m / h_{m+1,m}$
  Append v_m as last column of V
  $h \leftarrow [h_{1,m}, h_{2,m}, ..., h_{m+1,m}]^T$ ▷ New column of H to be factorized
  Augment Q by 1
  Increase size of $\bar{R}$ by 1 (with zeros)
  Append Qh as last column of $\bar{R}$
  $G \leftarrow$ Givens rotation to cancel $\bar{R}_{m+1,m}$
  $\bar{R} \leftarrow G\bar{R}$ ▷ rotate to cancel unwanted element under diagonal
  $Q \leftarrow GQ$ ▷ Save the rotation
  $\bar{g} \leftarrow [\bar{g}, 0]^T$
  $\bar{g} \leftarrow G\bar{g}$
  `res` $\leftarrow \bar{g}($`end`$)$
  $m \leftarrow m + 1$
end while
$R \leftarrow \bar{R}(1 :$ `end` $- 1, :)$
$g \leftarrow \bar{g}(1 :$ `end` $- 1)$
$y = R^{-1} g$ ▷ it's a triangular system
return $\tilde{x} = x_0 + Vy$

4 Numerical experiments and Results

The chosen benchmark problem is the following:

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathbb{R}^n} \quad & \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n-1} x_i x_{i+1} + \sum_{i=1}^{n} x_i \\ \text{subject to} \quad & x_1 + x_{1+K} + x_{1+2K} + \cdots = 1, \\ & x_2 + x_{2+K} + x_{2+2K} + \cdots = 1, \\ & \qquad\qquad \vdots \\ & x_K + x_{2K} + x_{3K} + \cdots = 1, \\ & x_i \geq 0, \quad i = 1, \ldots, n, \end{aligned} \tag{4.1}$$

The problem above can be reformulated as in 1.1 if we consider the following matrices:

$$\mathbf{Q} = \begin{bmatrix} 2 & -1 & & & 0 \\ -1 & \ddots & \ddots & & \\ & \ddots & \ddots & & \\ & & & & -1 \\ 0 & & & -1 & 2 \end{bmatrix} \in \mathbb{R}^{n\times n} \qquad \boldsymbol{c} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^n,$$

$$\mathbf{A} = \begin{bmatrix} 1 & & 1 & & & 1 & \\ & \ddots & & \ddots & \ddots & & \ddots \\ & & 1 & & 1 & & 1 \end{bmatrix} \in \mathbb{R}^{K\times n} \qquad \boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^K. \tag{4.2}$$

Ultimately, the problem will be solved in the more general formulation 1.3 using $\hat{\mathbf{A}} \in \mathbb{R}^{(2K+n)\times n}$ and $\hat{\boldsymbol{b}} \in \mathbb{R}^{2K+n}$ as defined in 1.4.

We studied four size scenarios, with $n = 10^4, n = 10^5$ and $K = 100, K = 500$. We tested both methods for solving the linear system as shown in 2.1 using both formulations Aug and Sus. Moreover, we compared our implementation of the GMRES algorithm with its native implementation in MATLAB

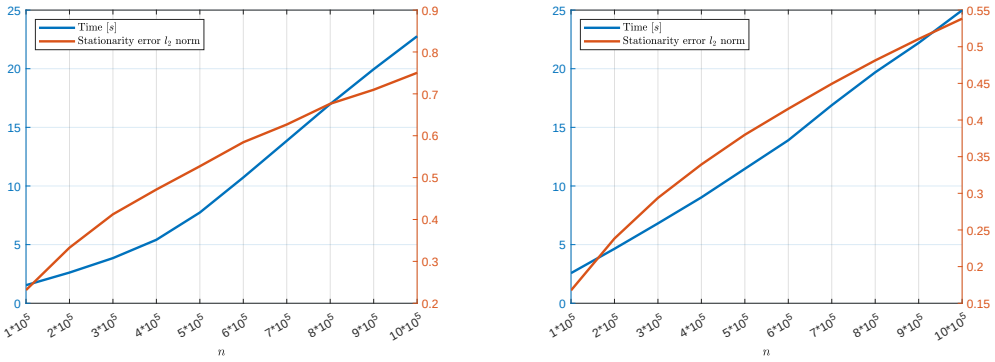
We initialized $\boldsymbol{y}_0 =$ `ones(2*K+n,1)`, $\hat{\boldsymbol{\lambda}}_0 =$ `ones(2*K+n,1)` and we obtain $\boldsymbol{x}_0$ by solving the slack variable constraint in (1.7) in the *least squares* sense. As seen in (2.7) we have chosen the parameter $\tau_k \in [0, 1]$ to be a function on μ_k because we wanted τ_k to converge to 1 as μ_k approaches to 0. After *tuning* the parameters, we have chosen the following expression:

$$\tau_k := \tau(\mu_k) = 0.3 \exp(-\mu_k) + 0.7. \tag{4.3}$$

In the following table computational times are shown comparing our implementation of the GMRES algorithm with the native implementation in MATLAB.

Dimensions	L.S. Formulation	Time $[s]$ Native	Time $[s]$ Our GMRES
$n = 10^4$, $K = 100$	Aug	0.561	0.439
$n = 10^4$, $K = 100$	Sus	0.592	0.386
$n = 10^4$, $K = 500$	Aug	1.191	0.262
$n = 10^4$, $K = 500$	Sus	2.383	1.407
$n = 10^5$, $K = 100$	Aug	1.663	2.965
$n = 10^5$, $K = 100$	Sus	2.383	2.408
$n = 10^5$, $K = 500$	Aug	1.311	1.229
$n = 10^5$, $K = 500$	Sus	5.413	5.601

As it is visible in the table above, performing the algorithm using the Aug formulation of the linear systems performs much quicker the majority of times. Let's investigate how the computation times scale with the dimension of the problem, fixing the value $K = 100$ and solving the Aug formulation.



(a) Native GMRES algorithm

(b) Our GMRES implementation

Figure 1: Time and stationarity error of the IPM algorithm using the native GMRES function implemented in MATLAB and our implementation shown for different values of n.

In 1 we can see that both computational time and accuracy of the method scale roughly linearly with the dimension of the problem. As a measure of accuracy we used the error on the stationarity condition as defined in 1.7, in particular:

$$\|Qx + c - \hat{A}^T \hat{\lambda}\|_2$$

A very important note is the fact that using an iterative linear system solver is **the only way to perform IPM at large scale**. We tried to solve the linear systems involved in the IPM iterations using the *Backslash operator* in MATLAB (which is the standard operator to solve linear systems) but this resulted in **failing the test** when the dimension of the problem approaches $n \sim 10^5$. Not only computational times are much higher ($\sim$ 1min for the backslash versus $\sim$ 1sec for the GMRES when $n \sim 10^5$) but the backslash operator cannot handle the dimension of the linear system returning `NaN` as results. This could be due to the conditioning of the matrices and the lack of enough RAM memory in the system.

The fact that we can solve such algorithm with enough accuracy only by using an iterative linear system solver is made possible because the solution of the linear systems at each iteration does not need to be much accurate itself. The GMRES algorithm in fact stops without reaching the desired tolerance most of the time, but this is not a problem for the accuracy of the final solution. Moreover, since the maximum iterations of the GMRES algorithm are fixed, bad conditioning of the matrix does not affect computational times unlike with the backslash operator.

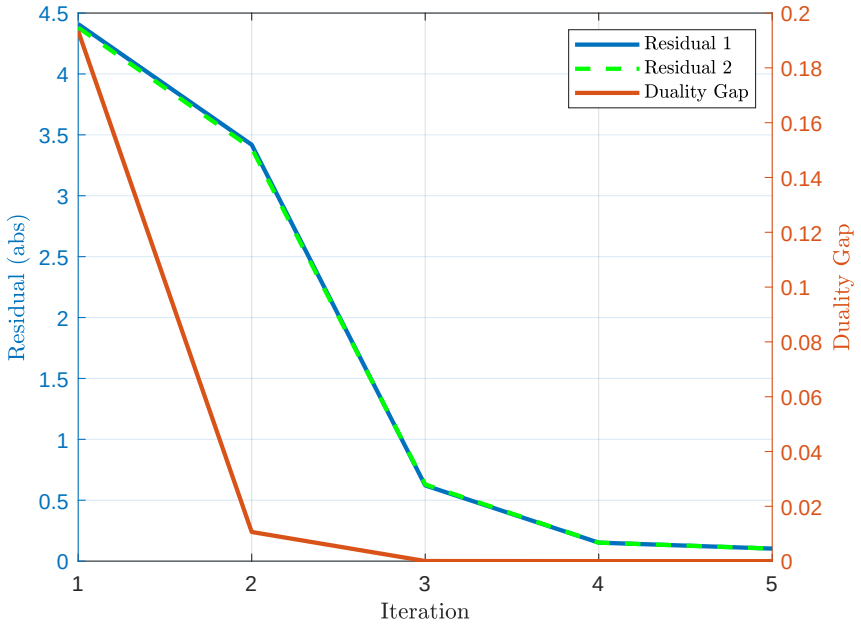


Figure 2: Residuals relative to the computation of both linear systems in each step of the IPM algorithm compared to the duality gap evolution. $n = 10^6$, $K = 100$

Since we can compute the duality gap at each step k of the IPM algorithm as:

$$\text{Duality Gap} = (2K + n)\mu_k$$

We can compare it to the residuals of the solutions of both linear systems in each iteration that can be easily retrieved from the GMRES algorithm as in INSERISCI REFERENCE A RESIDUO GMRES. As we can se in Figure 2 the the magnitude of the absolute residual of both linear systems is small enough to guarantee the convergence of the IPM algorithm in 5 iterations and the duality gap to be basically zero.