Approaches to nonlinear programming on GPU architectures

François Pacaud Sungho Shin Alexis Montoison
Michel Schanen Mihai Anitescu

March 23, 2024

Abstract

Solving nonlinear optimization problems requires the successive solutions of large symmetric indefinite linear systems, a task known to pose computational challenges. In particular, sparse factorization of such matrices relies on expensive numerical pivoting operations, which are difficult to parallelize on GPUs. A workaround is to reformulate the successive Karush-Kuhn-Tucker systems by condensing it into a symmetric positive-definite system suitable for an efficient Cholesky factorization on GPU. In this paper, we analyze the performance of two different condensed-space methods within an interior-point solver. We prove that the condensed matrices exhibit structured ill-conditioning, limiting the loss of accuracy when computing a descent direction. We implement the two methods on GPU, using the NVIDIA library cuDSS for solving sparse linear systems. We illustrate their strengths and weaknesses on large-scale instances taken from the PGLIB and the COPS benchmarks. We find that GPUs excel at computing results rapidly down to a medium convergence tolerance, showcasing a 10x speed-up compared to a state-of-the-art CPU version, yet demonstrating limited robustness.

1 Introduction

Graphical processing units (GPUs) have become the most popular parallel computing architecture for scientific computing, driven by their success in machine learning. GPUs offer massive parallel computing capability for applications that can exploit coarse-grain parallelism and high-memory bandwidth. As such, the library CUDA has revolutionized high-performance computing (HPC) by democratizing parallel capabilities previously exclusive to expensive supercomputers. In the post-Moore's law era, GPUs are becoming more power-efficient than CPUs for running parallel workloads, as they require fewer transistors to process multiple tasks in parallel. This efficiency is further enhanced by technologies like "single instruction, multiple data" (SIMD), which allow GPUs to perform the same operation on multiple data simultaneously, maximizing throughput and efficiency.

While GPUs have made significant strides in enhancing machine learning applications, their adoption in the mathematical programming community has been relatively limited. This limitation stems primarily from the fact that most optimization solvers were developed in the 1990s and are heavily optimized for CPU architectures. Additionally, the utilization of GPUs has been impeded by the challenges associated with sparse matrix factorization routines, which are inherently difficult to parallelize on SIMD architectures. Nevertheless, recent years have witnessed notable advancements that are reshaping this landscape.

1. **Improved sparse matrix operations**: The performance of sparse matrix operations has seen substantial improvements in the CUDA library, largely attributed to the integration of novel tensor cores in recent GPUs [16].

2. **Interest in batch optimization**: There is a growing interest in solving optimization problems in batch mode, for problems sharing the same structure but with different parameters [1, 24].

3. **Advancements in automatic differentiation**: GPUs offer unparalleled performance for automatic differentiation, benefiting both machine learning [4] and scientific computing applications [20]. Engineering problems often exhibit recurring patterns throughout the model. Once these patterns are identified, they can be evaluated in parallel within a SIMD framework, enabling near speed-of-light performance [27].

4. **Role in exascale computing**: With the emergence of new exascale architectures, GPUs have become indispensable in achieving high-performance computing, particularly in supercomputing environments.

For all the reasons listed before, there is an increasing interest to solve optimization problems on the GPU.

1.1 Current state-of-the-art on GPU

GPU for mathematical programming. The machine learning community has been a strong advocate for porting mathematical optimization on the GPU. One of the most promising applications is embedding mathematical programs inside neural networks, a task that requires batching the solution of the optimization model for the training algorithm to be efficient [1, 24]. This has led to the development of prototype code solving thousands of (small) optimization problems in parallel on the GPU. However, it is not trivial to adapt such code to solve large-scale optimization problems, as the previous prototypes are reliant on dense linear solvers to compute the descent direction.

For this reason, practitioners often resort to using first-order methods on GPUs, leveraging level-1 and level-2 BLAS operations that are more amenable to parallel computation. First-order algorithms depend mostly on (sparse) matrix-vector operations, that run very efficiently on modern GPUs. Hence, we can counterbalance the relative inaccuracy of the first-order method by running more

iterations of the algorithm. A recent breakthrough [14, 15] demonstrates that a first-order algorithm can surpass the performance of Gurobi, a commercial solver, in tackling large-scale linear programs. This performance gain is made possible by executing the first-order iterations solely on the GPU through an optimized codebase, thereby solving the LP problems to 10^{-8} accuracy.

GPU for nonlinear programming. The success of first-order algorithms in classical mathematical programming relies on the convexity of the problem. Thus, their approaches are nontrivial to replicate in nonlinear programming. Most engineering problems embed complex physical equations that are likely to break any convex structure in the problem. Previous experiments on the OPF problem have shown that even a simple algorithm as ADMM has trouble converging as soon as the convergence tolerance is set below 10^{-3} [13].

Thus, second-order methods continue to be a competitive option, particularly for scenarios that require higher levels of accuracy and robust convergence. Second-order algorithms require solving a Newton step at each iteration, an operation relying on non-trivial sparse linear algebra operations. The previous generation of GPU-accelerated sparse linear solvers were lagging behind their CPU equivalents, as illustrated in subsequent surveys [29, 28]. Fortunately, sparse solvers on GPUs are getting increasingly better with the newest generations of GPUs. In particular, NVIDIA has released in November 2023 a new sparse direct solver that implements different sparse factorization routines: `cuDSS`. Our preliminary benchmark has shown `cuDSS` is significantly faster than the previous sparse solvers using NVIDIA GPUs. The new NVIDIA Grace CPU architecture could also be a game changer in the future of sparse linear solvers, thanks to fast communication between the CPU and GPU. Furthermore, variants of interior point methods have been proposed that does not require the use of numerical pivoting. As these algorithms do not require expensive numerical pivoting operations, parallelized sparse solvers can be effectively exploited within the solution algorithms. Coupled with a GPU-accelerated automatic differentiation library and a sparse Cholesky solver, these nonlinear programming solvers can solve Optimal Power Flow (OPF) problems 10x faster than state-of-the-art methods [27].

An alternative thread of research is investigating the solution of the Newton steps on the GPU using iterative methods such as Krylov methods. Iterative methods often require non-trivial reformulation of the Newton step to avoid ill-conditioned matrices, which has limited their use inside interior-point algorithms. New results are giving promising outlooks for convex problems [10], but nonconvex problems often require an Augmented Lagrangian reformulation to be tractable [5, 25]. In particular, [25] presents an interesting use of the Golub and Greif hybrid method [11] to solve the KKT systems arising in the interior-point methods, with promising results on the GPU. The null-space method, also known as the reduced Hessian strategy, is also a good candidate for solving KKT systems on the GPU. The null-space method reduces the KKT system down to a medium-sized dense matrix, which then can be factorized efficiently on the

GPU. Our previous research has shown that the method plays nicely with the interior-point methods if the number of degrees of freedom in the problem is relatively small [23].

1.2 Contributions

In this article, we assess the current capabilities of modern GPUs to solve large-scale nonconvex nonlinear programs to optimality. We focus on the two condensed-space methods introduced respectively in [25, 27]. We re-use classical results from [32] to show that for both methods, the condensed matrix exhibits structured ill-conditioning that limits the loss of accuracy in the descent direction (provided the interior-point algorithm satisfies some standard assumptions). We implement both algorithms inside the GPU-accelerated solver MadNLP, and leverage the GPU-accelerated automatic differentiation backend ExaModels [27]. The interior-point algorithm runs entirely on the GPU, from the evaluation of the model (using ExaModels) to the solution of the KKT system (using a condensed-space method running on the GPU). We use CUDSS.jl [18], a Julia interface to the NVIDIA library `cuDSS` to solve the condensed KKT systems. We evaluate the strengths and weaknesses of both methods, in terms of accuracy and runtime. Extending beyond the classical OPF instances previously examined in our work, we incorporate large-scale problems sourced from the COPS nonlinear benchmark [7]. Our assessment involves comparing the performance achieved on the GPU with that of a state-of-the-art method executed on the CPU. The findings reveal that the condensed-space methods demonstrate a remarkable 10x acceleration in solving large-scale OPF instances when utilizing the GPU. However, performance outcomes on the COPS benchmark exhibit more variability.

1.3 Notations

By default, the norm $\|\cdot\|$ refers to the 2-norm. We define the conditioning of a matrix A as $\kappa_2(A) = \|A\|\|A^{-1}\|$. For any real number x, we denote $\widehat{x}$ as its floating point representation. We denote $\mathbf{u}$ as the smallest positive number such that $\widehat{x} \leq (1+\tau)x$ for $|\tau| < \mathbf{u}$. In double precision, $\mathbf{u} = 1.1 \times 10^{-16}$. We use the following notations to proceed with our error analysis. For $p \in \mathbb{N}$ and a positive variable h:

- We write $x = O(h^p)$ if there exists a constant $b > 0$ such that $\|x\| \leq bh^p$;
- We write $x = \Omega(h^p)$ if there exists a constant $a > 0$ such that $\|x\| \geq ah^p$;
- We write $x = \Theta(h^p)$ if there exists two constants $0 < a < b$ such that $ah^p \leq \|x\| \leq bh^p$.

2 Primal-dual interior-point method

The interior-point method (IPM) is among the most popular algorithms to solve nonlinear programs. The basis of the algorithm is to reformulate the Karush-Kuhn-Tucker (KKT) conditions of the nonlinear program as a smooth system of nonlinear equations using a homotopy method [22]. In a standard implementation, the resulting system is solved iteratively with a Newton method (used in conjunction with a line-search method for globalization). In this section, we give a brief description of a nonlinear program in section 2.1 and detail the Newton step solved at each IPM iteration in section 2.2.

2.1 Problem's formulation and KKT conditions

We are interested in solving the following nonlinear program:

$$\min_{x \in \mathbb{R}^n} \; f(x) \quad \text{subject to} \quad \begin{cases} g(x) = 0 \;,\; h(x) \leq 0 \;, \\ x \geq 0 \;, \end{cases} \tag{1}$$

with $f : \mathbb{R}^n \to \mathbb{R}$ a real-valued function encoding the objective, $g : \mathbb{R}^n \to \mathbb{R}^{m_e}$ encoding the equality constraints, and $h : \mathbb{R}^n \to \mathbb{R}^{m_i}$ encoding the inequality constraints. In addition, the variable x is subject to simple bounds $x \geq 0$. In what follows, we suppose that the functions f, g, h are smooth and twice differentiable.

We reformulate (1) using slack variables $s \geq 0$ into the equivalent formulation

$$\min_{x \in \mathbb{R}^n,\; s \in \mathbb{R}^{m_i}} \; f(x) \quad \text{subject to} \quad \begin{cases} g(x) = 0 \;,\; h(x) + s = 0 \;, \\ x \geq 0 \;,\; s \geq 0 \;. \end{cases} \tag{2}$$

In (2), the inequality constraints are directly encoded inside the variable bounds.

We denote by $y \in \mathbb{R}^{m_e}$ the multipliers associated to the equality constraints and $z \in \mathbb{R}^{m_i}$ the multipliers associated to the inequality constraints. Similarly, we denote by $(u, v) \in \mathbb{R}^{n+m_i}$ the multipliers associated respectively to $x \geq 0$ and $s \geq 0$. Using the multipliers (y, z, u, v), we define the Lagrangian of (2) as

$$L(x, s; y, z, u, v) = f(x) + y^\top g(x) + z^\top \big(h(x) + s\big) - u^\top x - v^\top s \;. \tag{3}$$

The KKT conditions of (2) are:

$$\nabla f(x) + \nabla g(x)^\top y + \nabla h(x)^\top z - u = 0 \tag{4a}$$
$$z - v = 0 \tag{4b}$$
$$g(x) = 0 \tag{4c}$$
$$h(x) + s = 0 \tag{4d}$$
$$0 \leq x \perp u \geq 0 \tag{4e}$$
$$0 \leq s \perp v \geq 0 \tag{4f}$$

The notation $x \perp u$ is a shorthand for the complementarity condition $x_i u_i = 0$ (for all $i = 1 \cdots, n$).

The set of active constraints at a point x is denoted by

$$\mathcal{B}(x) := \{i = 1, \cdots, m_i \mid h_i(x) = 0\} . \tag{5}$$

The inactive set is defined as the complement $\mathcal{N}(x) := \{1, \cdots, m_i\} \setminus \mathcal{B}(x)$. We note m_a the number of active constraints. The active Jacobian is defined as $A(x) := \begin{bmatrix} \nabla g(x) \\ \nabla h_{\mathcal{B}}(x) \end{bmatrix} \in \mathbb{R}^{(m_e+m_a)\times n}$.

Assumption 2.1. *Let $w^\star = (x^\star, s^\star, y^\star, z^\star, u^\star, v^\star)$ be a primal-dual solution satisfying the KKT conditions (4). Let the following hold:*

- *Continuity: The Hessian $\nabla^2_{xx} L(\cdot)$ is Lipschitz continuous near $w^\star$;*
- *Linear Independence Constraint Qualification: the active Jacobian $A(x^\star)$ is full row-rank;*
- *Strict Complementarity: for every $i \in \mathcal{B}(x^\star)$, $z_i^\star > 0$.*
- *Second-order sufficiency: for every $v \in \mathrm{Ker}\big(A(x^\star)\big)$, $v^\top \nabla^2_{xx} L(w^\star) v > 0$.*

2.2 Solving the KKT conditions with the interior-point method

The interior-point method aims at finding a stationary point associated to the KKT conditions (4). The complementarity constraints (4e)-(4f) render the KKT conditions non-smooth, complicating the solution of the whole system (4). Instead, IPM uses a homotopy continuation method to solve a simplified version of (4), parameterized by a barrier parameter $\mu > 0$ [22, Chapter 19]. For positive $(x, s, u, v) > 0$, we solve the system $F_\mu(x, s, y, z, u, v) = 0$, with

$$F_\mu(x, s, y, z, u, v) = \begin{bmatrix} \nabla f(x) + \nabla g(x)^\top y + \nabla h(x)^\top z - u \\ z - v \\ g(x) \\ h(x) + s \\ Xu - \mu e \\ Sv - \mu e \end{bmatrix} . \tag{6}$$

We introduce in (6) the diagonal matrices $X = \mathrm{diag}(x_1, \cdots, x_n)$ and $S = \mathrm{diag}(s_1, \cdots, s_{m_i})$. As we drive the barrier parameter μ to 0, we recover the original KKT conditions (4).

We note that at a fixed parameter μ, the function $F_\mu(\cdot)$ is smooth. Hence, the system (6) can be solved iteratively using a regular Newton method. For a given primal-dual variable $w_k := (x_k, s_k, y_k, z_k, u_k, v_k)$, the Newton step writes out $w_{k+1} = w_k + \alpha_k d_k$, with d_k a descent direction being solution of the linear system

$$\nabla_w F_\mu(w_k) d_k = -F_\mu(w_k) . \tag{7}$$

The step α_k is computed using a line-search algorithm, in a way that ensures that the bounded variables remain positive at the next primal-dual iterate (so

that $(x_{k+1}, s_{k+1}, u_{k+1}, v_{k+1}) > 0$). Once the iterates are sufficiently close to the central path, the IPM decreases the barrier term μ to find a solution closer to the original KKT conditions (4).

In IPM, the bulk of the workload is the computation of the Newton step (7), which involves assembling the Jacobian $\nabla_w F_\mu(w_k)$ and solving the linear system to compute the descent direction d_k. In order to avoid overloading the notation, we omitted the subscript k in the components of the KKT systems. However, it's important to remember that they are non-constant. The Newton step (7) expands as the 6×6 *unreduced KKT system*:

$$\begin{bmatrix} W & 0 & G^\top & H^\top & -I & 0 \\ 0 & 0 & 0 & I & 0 & -I \\ G & 0 & 0 & 0 & 0 & 0 \\ H & I & 0 & 0 & 0 & 0 \\ U & 0 & 0 & 0 & X & 0 \\ 0 & V & 0 & 0 & 0 & S \end{bmatrix} \begin{bmatrix} d_x \\ d_s \\ d_y \\ d_z \\ d_u \\ d_v \end{bmatrix} = - \begin{bmatrix} \nabla_x L(w_k) \\ z_k - v_k \\ g(x_k) \\ h(x_k) + s_k \\ X_k u_k - \mu e \\ S_k v_k - \mu e \end{bmatrix}, \qquad (K_3)$$

where we have introduced the Hessian $W = \nabla^2_{xx} L(w_k)$ and the two Jacobians $G = \nabla g(x_k)$, $H = \nabla h(x_k)$. In addition, we define X, S, U and V the diagonal matrices built from the vectors x_k, s_k, u_k and v_k. Note that (K_3) is not symmetric but can be symmetrized.

Augmented KKT system. It is usual to remove the blocks associated to the bound multipliers (u, v) and solve instead the equivalent 4×4 symmetric system, called the *augmented KKT system*:

$$\begin{bmatrix} W + D_x & 0 & G^\top & H^\top \\ 0 & D_s & 0 & I \\ G & 0 & 0 & 0 \\ H & I & 0 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_s \\ d_y \\ d_z \end{bmatrix} = - \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix}, \qquad (K_2)$$

with the diagonal matrices $D_x = X^{-1}U$ and $D_s = S^{-1}V$. The vectors forming the right-hand-side are given respectively by $r_1 = \nabla f(x_k) + \nabla g(x_k)^\top y_k + \nabla h(x_k)^\top z_k + \mu X^{-1} e$, $r_2 = z_k + \mu S^{-1} e$, $r_3 = g(x_k)$, $r_4 = h(x_k) + s_k$. Once (K_2) solved, we recover the updates on bound multipliers with $d_u = -X^{-1}(U d_x + X u_k - \mu e)$ and $d_v = -S^{-1}(V d_s + S v_k - \mu e)$.

The system (K_2) is usually factorized using an inertia-revealing LBL$^{\mathrm{T}}$ factorization. Because the block diagonal terms D_x and D_s are usually poorly conditioned, Krylov methods has difficulties to perform. Their effectiveness can be significantly enhanced when paired with suitable preconditioners. This synergy offers a promising alternative in interior point methods, especially for linear and convex quadratic programming [**?**]. Additionally, because Krylov methods are efficient on GPU architectures, they are experimented in nonlinear programming applications to speed-up computations [**?**].

Condensed KKT system. The 4×4 KKT system (K_2) can be further reduced down to a 2×2 system by eliminating the two blocks (d_s, d_z) associated

to the inequality constraints. The resulting system is called the *condensed KKT system*:

$$\begin{bmatrix} K & G^\top \\ G & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \end{bmatrix} = - \begin{bmatrix} r_1 + H^\top (D_s r_4 - r_2) \\ r_3 \end{bmatrix} =: \begin{bmatrix} \bar{r}_1 \\ \bar{r}_2 \end{bmatrix}, \qquad (K_1)$$

where we have introduced the *condensed matrix* $K := W + D_x + H^\top D_s H$. Using the solution of the system (K_1), we recover the updates on the slacks and inequality multipliers with $d_s = -r_4 - H d_x$ and $d_z = -r_2 - D_s d_s$.

Iterative refinement. Compared to (K_3), the diagonal matrices D_x and D_s in (K_2) introduce an additional ill-conditioning in (K_2), amplified in the condensed form (K_1). For that reason, it is recommended to refine the solution returned by the direct sparse linear solver by using Richardson iterations on the original system (K_3) (see [30, Section 3.10]).

2.3 Discussion

We have obtained three different formulations for the KKT system appearing at each IPM iteration. The original formulation (K_3) is not symmetric, but has a better conditioning than the two alternatives (K_2) and (K_1). The second formulation (K_2) is used by default in state-of-the-art nonlinear solvers [30, 31]. The system (K_2) is usually factorized using a LBL^T factorization: for sparse matrices, the Duff and Reid multifrontal algorithm [9] is the favored method (as implemented in the HSL linear solvers MA27 and MA57 [8]). On its end, the condensed KKT system (K_1) is often discarded, as its conditioning is higher than (K_2) (implying less accurate solutions) and the term $H^\top D_s H$ can lead to a dense condensed matrix if one column of the Jacobian is dense.

Additionally, condensation may result in increased fill-in within system K_1 [22, Section 19.3, p.571]. In the worst cases where an inequality row is completely dense, K_1 itself may become fully dense. Consequently, condensation methods are not commonly utilized in practical optimization settings. To the best our knowledge, Knitro [31] is the only solver that supports computing the descent direction with (K_1).

3 Solving KKT systems on the GPU

The GPU has emerged as the new prominent landscape for numerical computing. GPUs employ a SIMD formalism that yields excellent throughput for parallelizing small-scale operations. However, their utility remains limited when computational algorithms require global communication. Sparse factorization algorithms, which heavily rely on numerical pivoting, pose significant challenges for implementation on GPUs due to this limitation. Previous research has demonstrated that GPU-based linear solvers significantly lag behind their CPU counterparts [29, 28]. One emerging strategy to address this challenge is to utilize sparse factorization techniques that do not necessitate numerical pivoting [25, 27], leveraging the structure of the condensed KKT system (K_1).

3.1 Golub & Greif strategy

The Golub & Greif [11] strategy reformulates the KKT system using an Augmented Lagrangian formulation. It has been recently revisited in [25] to solve the condensed KKT system (K_1) on the GPU.

The trick is to reformulate the condensed KKT system (K_1) in the equivalent form

$$\begin{bmatrix} K_\gamma & G^\top \\ G & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \end{bmatrix} = \begin{bmatrix} \bar{r}_1 + \gamma G^\top \bar{r}_2 \\ \bar{r}_2 \end{bmatrix} \tag{8}$$

where we have introduced the regularized matrix $K_\gamma := K + \gamma G^\top G$. If SOSC holds, the matrix K_γ is positive definite for a large-enough parameter γ.

Proposition 3.1 ([25], Theorem 1, p.7)**.** *We suppose G is full row rank. Then there exists γ_{min} such that, for all $\gamma > \gamma_{min}$, K_γ is positive definite if and only if the reduced Hessian $Z^\top K Z$ is positive definite.*

If γ is large enough, we can prove that the conditioning of K_γ increases linearly with γ.

Proposition 3.2 ([25], Theorem 2, p.7)**.** *We suppose G is full row rank. Then there exists $\gamma_{max} \geq \gamma_{min}$ such that for all $\gamma \geq \gamma_{max}$, $\kappa_2(K_\gamma)$ increases linearly with γ.*

The linear solver HyKKT [25] leverages the positive definiteness of K_γ and solves (8) using a hybrid direct-iterative method that uses the following steps:

1. Assemble K_γ and factorize it using sparse Cholesky ;
2. Solve the Schur complement of (8) using a conjugate gradient (CG) algorithm to recover the dual descent direction:

$$(GK_\gamma^{-1}G^\top)d_y = GK_\gamma^{-1}(\bar{r}_1 + \gamma G^\top \bar{r}_2) - \bar{r}_2 \ . \tag{9}$$

3. Solve the system $K_\gamma d_x = \bar{r}_1 + \gamma G^\top \bar{r}_2 - G^\top d_y$ to recover the primal descent direction.

The method uses a sparse Cholesky factorization along with the conjugate gradient (CG) algorithm [12]. The sparse Cholesky factorization has the advantage of being stable without numerical pivoting, rendering the algorithm tractable on a GPU. Each CG iteration requires the application of sparse triangular solves with the factors of K_γ, operations that can be numerically demanding. For that reason, HyKKT is efficient only if the CG solver converges in a small number of iterations. Fortunately, the eigenvalues of the Schur-complement $S_\gamma := GK_\gamma^{-1}G^\top$ all converge to $\frac{1}{\gamma}$ as we increase the regularization parameter γ ([25, Theorem 4]), meaning that $\lim_{\gamma\to\infty} \kappa_2(S_\gamma) = 1$. Because the convergence of the CG method depends on the number of distinct eigenvalues of S_γ, when γ increases, the eigenvalues of S_γ become more clustered, resulting in fewer iterations being required to solve (9).

3.2 A novel equality relaxation strategy (SS)

TODO (Sungho)

3.3 Discussion

We have introduced two algorithms to solve the KKT system on the GPU. On the contrary to classical CPU algorithms, the two methods do not require computing a sparse LBL^T factorization of the KKT system by using alternate reformulation based on the condensed KKT system (K_1).

The first strategy uses an augmented Lagrangian formulation of the KKT system. It requires a sparse Cholesky factorization (stable without numerical pivoting, hence favorable for the GPU) and uses it to solve the Schur-complement system using the CG algorithm. The performance of the method depends on a parameter γ, that has to be tuned independently. The larger the γ, the faster is the convergence in the CG algorithm, but the worst is the accuracy of the linear solve. Hence, a trade-off has to be found. The second method uses an equality relaxation strategy to solve the condensed KKT system (K_1) directly. It just requires a sparse Cholesky solver to factorize (K_1). However, the method solves a relaxation of the original problem (1), limiting the accuracy of the solution.

4 Conditioning of the condensed KKT system

The condensed matrix K appearing in (K_1) is known to be ill-conditioned close to a local optimum solution. That behavior is amplified for the matrix K_γ as we increase the parameter γ. For a generic linear system $Mx = b$, the relative error after a perturbation Δb on the right-hand-side is bounded by

$$\|\Delta x\| \leq \|M^{-1}\|\|\Delta b\| \,, \quad \frac{\|\Delta x\|}{\|x\|} \leq \kappa_2(M)\, \frac{\|\Delta b\|}{\|b\|} \,. \tag{10a}$$

When the matrix is perturbed by ΔM, the perturbed solution $\widehat{x}$ satisfies $\Delta x = \widehat{x} - x = -(M + \Delta M)^{-1}\Delta M \widehat{x}$. If $\kappa_2(M) \approx \kappa_2(M + \Delta M)$, the error Δx satisfies $M\Delta x \approx -\Delta M x$ (neglecting second-order terms), giving the bounds

$$\|\Delta x\| \leq \|M^{-1}\|\|\Delta M\|\|x\| \,, \quad \frac{\|\Delta x\|}{\|x\|} \leq \kappa_2(M) \frac{\|\Delta M\|}{\|M\|} \,. \tag{10b}$$

The relative errors are bounded above by the conditioning $\kappa_2(M)$, meaning that without further assumptions the condensed matrix K amplify the errors made throughout the algorithm. Hence, it is legitimate to investigate the impact of the ill-conditioning when solving the condensed system (K_1) in both the equality-relaxation strategy and the HyKKT strategy. We will see that we can tighten the bounds in (10) by exploiting the structured ill-conditioning of the condensed matrix K. We base our analysis on [32], where the author has put a particular emphasis on the condensed KKT system (K_1), but without equality constraints.

We generalize her results to the matrix K_γ, which incorporates both equality and inequality constraints.

To alleviate the notations, we suppose that the primal variable x is unconstrained, leaving only the slack s with bounded values in (2). This is equivalent including the bounds on the variable x in the inequality constraints, as $\bar{h}(x) \leq 0$ with $\bar{h}(x) := (h(x), -x)$.

4.1 Centrality conditions

We start this paragraph by recalling important results porting on the iterates of the interior-point algorithm [34]. Let $p = (x, s, y, z)$ the current primal-dual iterate (the multiplier v is considered apart), and $p^\star$ a solution of the KKT conditions (4). We note $\delta(p, v) = \|(p, v) - (p^\star, v^\star)\|$ the Euclidean distance to the primal-dual stationary point $p^\star$. From [34, Theorem 2.2], if Assumptions 2.1 hold at $p^\star$ and $v > 0$,

$$\delta(p, v) = \Theta\left(\left\| \begin{bmatrix} \nabla_p L(p, v) \\ \min(v, s) \end{bmatrix} \right\|\right) . \tag{11}$$

For $(s, v) > 0$, we define the *duality measure* $\Xi(s, v)$ as the mapping

$$\Xi(s, v) = s^\top v / m_i . \tag{12}$$

We suppose the iterates (p, v) satisfies the *centrality conditions*

$$\|\nabla_p \mathcal{L}(p, v)\| \leq C\, \Xi(s, v) , \tag{13a}$$

$$(s, v) > 0 , \quad s_i v_i \geq \alpha\, \Xi(s, v) \quad \forall i = 1, \cdots, m_i , \tag{13b}$$

for some constant $C > 0$ and $\alpha \in (0, 1)$. Conditions (13b) ensures that the products $s_i v_i$ are not too disparate in the diagonal term D_s. We note that this condition is satisfied in the solver Ipopt (see [30, Equation (16)]).

Proposition 4.1 ([34], Lemma 3.2)**.** *Suppose $p^\star$ satisfies the assumptions 2.1, and the current primal-dual iterate (p, v) satisfies the centrality conditions* (13)*. Then,*

$$i \in \mathcal{B} \implies s_i = \Theta(\Xi) , \quad v_i = \Theta(1) , \tag{14a}$$

$$i \in \mathcal{N} \implies s_i = \Theta(1) , \quad v_i = \Theta(\Xi) . \tag{14b}$$

Using the centrality conditions (13), we can bound the distance to the solution $\delta(p, v)$ with the duality measure Ξ.

Theorem 4.2 ([34], Theorem 3.3)**.** *Suppose $p^\star$ satisfies the assumptions 2.1, and the current primal-dual iterate (p, v) satisfies the centrality conditions* (13)*. Then,*

$$\delta(p, v) = O(\Xi) . \tag{15}$$

4.2 Structured ill-conditioning

We show that if the iterates (p, v) satisfy the centrality conditions (13), then the condensed matrix K_γ has a structured ill-conditioning.

4.2.1 Invariant subspaces in K_γ

First, we follow the analysis presented in [32] and show that the condensed matrix K_γ can be decomposed as

$$K_\gamma = \begin{bmatrix} U_L & U_S \end{bmatrix} \begin{bmatrix} \Sigma_L & 0 \\ 0 & \Sigma_S \end{bmatrix} \begin{bmatrix} U_L^\top \\ U_S^\top \end{bmatrix} , \tag{16}$$

with U orthogonal matrix and Σ diagonal, where the two diagonal matrices Σ_L and Σ_S are both better conditioned than K_γ. The result is a consequence of the SVD decomposition, provided that the singular values satisfy $\frac{\sigma_1}{\sigma_p} \leq \frac{\sigma_1}{\sigma_n}$ and $\frac{\sigma_{p+1}}{\sigma_n} \leq \frac{\sigma_1}{\sigma_n}$.

We recall that $K_\gamma = W + H^\top D_s H + \gamma G^\top G$, with the diagonal matrix $D_s = S^{-1}V$. We note $\mathcal{B} = \mathcal{B}(x^\star)$ the active-set at the optimal solution $x^\star$, and $\mathcal{N} = \mathcal{N}(x^\star)$ the inactive set. In addition, we note m_a the cardinal of $\mathcal{B}$, and $p := m_a + m_e$. We denote by $H_\mathcal{B}$ the Jacobian of active inequality constraints, $H_\mathcal{N}$ the Jacobian of inactive inequality constraints and $A := \begin{bmatrix} G^\top & H_\mathcal{B}^\top \end{bmatrix}^\top$. We define the minimum and maximum active slack values as

$$s_{min} = \min_{i \in \mathcal{B}} s_i \ , \quad s_{max} = \max_{i \in \mathcal{B}} s_i \ . \tag{17}$$

To prove K_γ decomposes as in (16), we adapt [32, Theorem 3.2] by incorporating the additional term $\gamma G^\top G$ coming from the equality constraints.

Theorem 4.3 (Properties of K_γ)**.** *Suppose the condensed matrix is evaluated at a primal-dual point (p, ν) satisfying (13), for sufficiently small Ξ. Let $\lambda_1, \cdots, \lambda_n$ be the n eigenvalues of K_γ, ordered as $|\lambda_1| \geq \cdots \geq |\lambda_n|$. Let $\begin{bmatrix} Y & Z \end{bmatrix}$ be an orthogonal matrix, where the columns of Z span the null-space of A. Let $\underline{\sigma} = \min\{\frac{1}{\Xi}, \gamma\}$ and $\overline{\sigma} = \max\{\frac{1}{s_{min}}, \gamma\}$. Then,*

(i) The p largest-magnitude eigenvalues of K_γ are positive, with $\lambda_1 = \Theta(\overline{\sigma})$ and $\lambda_p = \Omega(\underline{\sigma})$.

(ii) The $n - p$ smallest-magnitude eigenvalues of K_γ are $\Theta(1)$.

(iii) If $0 < p < n$, then $\kappa_2(K_\gamma) = \Theta(\overline{\sigma})$.

(iv) There are orthonormal matrices $\widetilde{Y}$ and $\widetilde{Z}$ for simple invariant subspaces of K_γ, such that $Y - \widetilde{Y} = O(\underline{\sigma}^{-1})$ and $Z - \widetilde{Z} = O(\underline{\sigma}^{-1})$.

Proof. We start the proof by setting apart the inactive constraints from the active constraints in K_γ:

$$K_\gamma = W + H_\mathcal{N}^\top S_\mathcal{N}^{-1} V_\mathcal{N} H_\mathcal{N} + A^\top D_\gamma A \, , \quad \text{with} \quad D_\gamma = \begin{bmatrix} S_\mathcal{B}^{-1} V_\mathcal{B} & 0 \\ 0 & \gamma I \end{bmatrix} . \tag{18}$$

Using Assumption 2.1, Lipschitz continuity implies that the Hessian and the inactive Jacobian are bounded: $W = O(1)$, $H_{\mathcal{N}} = O(1)$. Proposition 4.1 implies that $s_{\mathcal{N}} = \Theta(1)$ and $v_{\mathcal{N}} = \Theta(\Xi)$. We deduce:

$$H_{\mathcal{N}}^{\top} S_{\mathcal{N}}^{-1} V_{\mathcal{N}} H_{\mathcal{N}} = O(\Xi) \; . \tag{19}$$

Hence, for small enough Ξ, the condensed matrix K_γ is dominated by the block of active constraints:

$$K_\gamma = A^{\top} D_\gamma A + O(1) \; . \tag{20}$$

Sufficiently close to the optimum $p^\star$, the constraints qualification in Assumption 2.1 implies that $A = O(1)$ and has rank p. The eigenvalues $\{\eta_i\}_{i=1,\cdots,n}$ of $A^{\top} D_\gamma A$ satisfy $\eta_i > 0$ for $i = 1, \cdots, p$ and $\eta_i = 0$ for $i = p+1, \cdots, n$. As $s_{\mathcal{B}} = \Theta(\Xi)$ and $v_{\mathcal{B}} = \Theta(1)$ (Proposition 4.1), the smallest diagonal element in D_γ is $\Omega(\min\{\frac{1}{\Xi}, \gamma\})$ and the largest diagonal element is $\Theta(\max\{\frac{1}{s_{min}}, \gamma\})$. Hence,

$$\eta_1 = \Theta(\overline{\sigma}) \; , \quad \eta_p = \Omega(\underline{\sigma}) \; . \tag{21}$$

Using [32, Lemma 3.1], we deduce $\lambda_1 = \Theta(\overline{\sigma})$ and $\lambda_p = \Omega(\underline{\sigma})$, proving the first result (i).

Let $L_\gamma = A^{\top} D_\gamma A$. We have that

$$\begin{bmatrix} Z^{\top} \\ Y^{\top} \end{bmatrix} L_\gamma \begin{bmatrix} Z & Y \end{bmatrix} = \begin{bmatrix} L_1 & 0 \\ 0 & L_2 \end{bmatrix} \; , \tag{22}$$

with $L_1 = 0$ and $L_2 = Y^{\top} L_\gamma Y$. The smallest eigenvalue of L_2 is $\Omega(\underline{\sigma})$ and the matrix $E := K_\gamma - L_\gamma$ is $O(1)$. By applying [32, Theorem 3.1, (ii)], the $n-p$ smallest eigenvalues in K_γ differ by $\Omega(\underline{\sigma}^{-1})$ from those of the reduced Hessian $Z^{\top} K_\gamma Z$. In addition, (19) implies that $Z^{\top} K_\gamma Z - Z^{\top} W Z = O(\Xi)$. Using SOSC, $Z^{\top} W Z$ is positive definite for small enough Ξ, implying all its eigenvalues are $\Theta(1)$. Using again [32, Lemma 3.1], we get that the $n-p$ smallest eigenvalues in K_γ are $\Theta(1)$, proving (ii). The third point (iii) is proved by combining (i) and (ii) (provided $0 < p < n$). Eventually, point (iv) is proven by using [32, Theorem 3.1 (i)]. □

Corollary 4.4. *The condensed matrix K_γ can be decomposed as* (16)*, with $\Sigma_L \in \mathbb{R}^{p \times p}$ and $\Sigma_S \in \mathbb{R}^{(n-p)\times(n-p)}$. In addition, for suitably chosen Y and Z,*

$$U_L - Y = O(\underline{\sigma}^{-1}) \; , \quad U_S - Z = O(\underline{\sigma}^{-1}) \; . \tag{23}$$

Proof. According to Theorem 4.3, the p largest eigenvalues of K_γ are large and well separated from the $n-p$ smallest eigenvalues. Using the spectral theorem, we obtain the decomposition as (16). Using Theorem 4.3, part (iv), we obtain the result in (23). □

Theorem 4.3 gives us a deeper insight into the structure of the condensed matrix K_γ. Using equation (23), we observe we can assimilate the large-space of

K_γ with range$(A^\top)$ and the small space with Ker(A). The decomposition (16) leads to the following relations

$$\begin{aligned} \|K_\gamma\| = \|\Sigma_L\| = \Theta(\overline{\sigma})\,, \qquad & \Sigma_L^{-1} = O(\underline{\sigma}^{-1})\,, \\ \|K_\gamma^{-1}\| = \|\Sigma_S^{-1}\| = \Theta(1)\,, \qquad & \Sigma_S = \Theta(1)\,. \end{aligned} \tag{24}$$

The condition of Σ_L depends on $\kappa_2(A)$, $\kappa_2(V)$ and the ratio $\frac{s_{max}}{s_{min}} = O(\Xi\overline{\sigma})$. The condition of Σ_S reflects the condition of the reduced Hessian $Z^\top W Z$.

Remark 1. Theorem 4.3 (iii) tells us that $\kappa_2(K_\gamma) = \Theta(\overline{\sigma})$, meaning that if γ is large-enough so that $\gamma \geq \frac{1}{s_{min}}$, then the conditioning $\kappa_2(K_\gamma)$ increases linearly with γ. In other words, the value $\frac{1}{s_{min}}$ gives us a lower-bound for γ_{max} in Proposition 3.2.

4.2.2 Numerical accuracy of the condensed matrix K_γ

We are now interested in bounding the error made in the computation of the condensed matrix K_γ in floating point arithmetic.

In direct contrast with [32, Section 4.1], the slack variable s appearing in the diagonal matrix $S^{-1}V$ is not subject to cancellation: the interior-point algorithm is passing exactly the value of the slack variables to the linear solver (this would not be the case if the slack s_i was replaced by the nonlinear function $h_i(x)$). Hence, the error arising from the multiplication by S^{-1} is only $O(\mathbf{u})$, which is significantly better than the error in $O(\mathbf{u}/s_{min})$ we would obtain in the cancellation case.

In floating-point arithmetic, the condensed matrix K_γ is evaluated as

$$\begin{aligned} \widehat{K}_\gamma = W + E + (A+\Delta A)^\top (D_\gamma + \Delta D_\gamma)(A+\Delta A) \\ + (H_{\mathcal{N}} + \Delta H_{\mathcal{N}})^\top S_{\mathcal{N}}^{-1} V_{\mathcal{N}} (H_{\mathcal{N}} + \Delta H_{\mathcal{N}})\,, \end{aligned}$$

with $\|\Delta H\| = O(\mathbf{u}\|H\|)$, $\|\Delta G\| = O(\mathbf{u}\|G\|)$ and E symmetric with $\|E\| = O(\mathbf{u}\overline{\sigma})$. We have

$$A^\top D_\gamma A = \Theta(\overline{\sigma})\,, \quad A^\top \Delta D_\gamma A = O(\mathbf{u}\overline{\sigma})\,. \tag{25}$$

We deduce that the perturbation ΔK_γ is bounded by multiple of $\|K_\gamma\|$.

Proposition 4.5. *In floating-point arithmetic, the perturbation of the condensed matrix ΔK_γ satisfies $\Delta K_\gamma := \widehat{K_\gamma} - K_\gamma = O(\mathbf{u}\overline{\sigma})$.*

The perturbation ΔK_γ displays no specific structure. Hence, we should determine under which conditions the perturbed matrix $\widehat{K}_\gamma$ keeps the structure highlighted in (16). We know that the smallest eigenvalue η_p of $A^\top D_\gamma A$ is $\Omega(\underline{\sigma})$. As mentioned in [32, Section 3.4.2], the perturbed matrix $\widehat{K}_\gamma$ has p large eigenvalues bounded below by $\underline{\sigma}$ if the perturbation is much smaller than the eigenvalue η_p:

$$\|\Delta K_\gamma\| \ll \eta_p\,. \tag{26}$$

However, the bound in Proposition 4.5 is too loose for (26) to hold without any further assumption. As $\eta_p = \Omega(\underline{\sigma})$, we deduce $\frac{\|K_\gamma\|}{\eta_p} = O(\overline{\sigma}\underline{\sigma}^{-1})$. In other words, If we suppose in addition the ratio $\overline{\sigma}\underline{\sigma}^{-1}$ is close to 1, then $\|\Delta K_\gamma\| = O(\mathbf{u}\overline{\sigma})$ can be replaced by $\|\Delta K_\gamma\| = O(\mathbf{u}\underline{\sigma})$ which in turns implies (26).

4.2.3 Numerical solution of the condensed system

We are interested in estimating the relative error made when solving the system $K_\gamma x = b$ in floating point arithmetic. We suppose K_γ is factorized using a backward-stable Cholesky decomposition. The computed solution $\widehat{x}$ is solution of a perturbed system $\widetilde{K}_\gamma \widehat{x} = b$, with $\widetilde{K}_\gamma = K_\gamma + \Delta_s K_\gamma$ and $\Delta_s K_\gamma$ a symmetric matrix satisfying, for ϵ_n a small constant depending on the dimension n:

$$\|\Delta_s K_\gamma\| \leq \mathbf{u}\varepsilon_n \|K_\gamma\| \,. \tag{27}$$

We need the following additional assumptions to ensure (i) the Cholesky factorization runs to completion and (ii) we can incorporate the backward-stable perturbation $\Delta_s K_\gamma$ in the generic perturbation ΔK_γ introduced in Proposition 4.5.

Assumption 4.6. *Let (p, v) be the current primal-dual iterate. We assume:*

(a) (p, v) satisfies the centrality conditions (13).

(c) The parameter γ satisfies $\gamma = \Theta(\Xi^{-1})$.

(c) The duality measure is large enough relative to the precision $\mathbf{u}$*: $\mathbf{u} \ll \Xi$.*

(d) The primal step $\widehat{x}$ is computed using a backward stable method satisfying (27) *for a small constant ε_n.*

Condition (a) implies that $s_{min} = \Theta(\Xi)$ and $s_{max} = \Theta(\Xi)$ (Proposition 4.1). Condition (b) supposes in addition $\gamma = \Theta(\Xi^{-1})$, making the matrix Σ_L well-conditioned with $\underline{\sigma} = \Theta(\Xi^{-1})$, $\overline{\sigma} = \Theta(\Xi^{-1})$ and $\overline{\sigma}/\underline{\sigma} = \Theta(1)$. Condition (c) ensures that the conditioning of $\kappa_2(K_\gamma) = \Theta(\overline{\sigma})$ satisfies $\kappa_2(K_\gamma)\mathbf{u} \ll 1$, ensuring the Cholesky factorization runs to completion. Condition (d) tells us that the perturbation caused by the Cholesky factorization is $\Delta_s K_\gamma = O(\mathbf{u}\|K_\gamma\|)$. As (24) implies $\|K_\gamma\| = \Theta(\Xi^{-1})$, we can incorporate $\Delta_s K_\gamma$ in the perturbation ΔK_γ given in Proposition 4.5.

We note x the solution of the linear system $K_\gamma x = b$ in exact arithmetic, and $\widehat{x}$ the solution of the perturbed system $\widehat{K}_\gamma \widehat{x} = \widehat{b}$ in floating-point arithmetic. We are interested in bounding the error $\Delta x = \widehat{x} - x$. We recall that every vector $x \in \mathbb{R}^n$ decomposes as

$$x = U_L x_L + U_S x_S = Y x_Y + Z x_Z \,. \tag{28}$$

Impact of right-hand-side perturbation. Using (16), the inverse of K_γ satisfies

$$K_\gamma^{-1} = \begin{bmatrix} U_L & U_S \end{bmatrix} \begin{bmatrix} \Sigma_L^{-1} & 0 \\ 0 & \Sigma_S^{-1} \end{bmatrix} \begin{bmatrix} U_L^\top \\ U_S^\top \end{bmatrix} . \tag{29}$$

Hence, if we solve the system for $\widehat{b} := b + \Delta b$, the error Δx made is $\Delta x = K_\gamma^{-1} \Delta b$, and using the structure (29) we get

$$\begin{bmatrix} \Delta x_L \\ \Delta x_S \end{bmatrix} = \begin{bmatrix} \Sigma_L^{-1} & 0 \\ 0 & \Sigma_S^{-1} \end{bmatrix} \begin{bmatrix} \Delta b_L \\ \Delta b_S \end{bmatrix} , \tag{30}$$

which in turn implies the following bounds:

$$\begin{aligned} \|\Delta x_L\| &\le \|\Sigma_L^{-1}\| \|\Delta b_L\| , \\ \|\Delta x_S\| &\le \|\Sigma_S^{-1}\| \|\Delta b_S\| . \end{aligned} \tag{31}$$

As $\Sigma_L^{-1} = O(\Xi)$ and $\Sigma_S^{-1} = \Theta(1)$, we deduce that the error Δx_L is smaller by a factor of Ξ than the error Δx_S. The total error $\Delta x = U_L \Delta x_L + U_S \Delta x_S$ is bounded by

$$\|\Delta x\| \le \|\Sigma_L^{-1}\| \|\Delta b_L\| + \|\Sigma_S^{-1}\| \|\Delta b_S\| = O(\|\Delta b\|) . \tag{32}$$

Impact of matrix perturbation. As $\|\Delta K_\gamma\| \ll \|K_\gamma\|$, we have that

$$\begin{aligned} (K_\gamma + \Delta K_\gamma)^{-1} &= (I + K_\gamma^{-1} \Delta K_\gamma)^{-1} K_\gamma^{-1} \\ &= K_\gamma^{-1} - K_\gamma^{-1} \Delta K_\gamma K_\gamma^{-1} + O(\|\Delta K_\gamma\|^2) . \end{aligned} \tag{33}$$

Using (29) and noting $\Delta K_\gamma = \begin{bmatrix} \Gamma_L \\ \Gamma_S \end{bmatrix}$, the first-order error is given by

$$K_\gamma^{-1} \Delta K_\gamma K_\gamma^{-1} = U_L \Sigma_L^{-1} \Gamma_L \Sigma_L^{-1} U_L^\top + U_S \Sigma_S^{-1} \Gamma_S \Sigma_S^{-1} U_S^\top . \tag{34}$$

Using (26) and $(\Gamma_L, \Gamma_S) = O(\Xi^{-1}\mathbf{u})$, we deduce that the error made in the large-space is $O(\Xi\mathbf{u})$ whereas the error in the small-space is $O(\Xi^{-1}\mathbf{u})$.

4.3 Solution of the condensed KKT system

We use the relations (31) and (34) to bound the error made when solving the original condensed KKT (K_1) in floating-point arithmetic. In all this section, we assume that the primal-dual iterate (p, v) satisfies Assumption 4.6. Using [34, Corollary 3.3], the solution (d_x, d_y) of the condensed KKT system (K_1) in exact arithmetic satisfies $(d_x, d_y) = O(\Xi)$.

In (K_1), the RHS $\bar{r}_1$ and $\bar{r}_2$ evaluate in floating-point arithmetic as

$$\begin{cases} \bar{r}_1 = -\widehat{r}_1 + \widehat{H}^\top \left(\widehat{D}_s \widehat{r}_4 - \widehat{r}_2 \right) , \\ \bar{r}_2 = -\widehat{r}_3 . \end{cases} \tag{35}$$

Using basic floating-point arithmetic, we get $\widehat{r}_1 = r_1 + O(\mathbf{u})$, $\widehat{r}_3 = r_3 + O(\mathbf{u})$, $\widehat{r}_4 = r_4 + O(\mathbf{u})$. The error in the right-hand-side r_2 is impacted by the term $\mu S^{-1}e$: under Assumption 4.6, it impacts differently the active and inactive components: $\widehat{r}_{2,\mathcal{B}} = r_{2,\mathcal{B}} + O(\mathbf{u})$ and $\widehat{r}_{2,\mathcal{N}} = r_{2,\mathcal{N}} + O(\Xi\mathbf{u})$. Similarly, the diagonal matrix $\widehat{D}_s$ retains full accuracy only w.r.t. the inactive components: $\widehat{D}_{s,\mathcal{B}} = D_{s,\mathcal{B}} + O(\Xi^{-1}\mathbf{u})$ and $\widehat{D}_{s,\mathcal{N}} = D_{s,\mathcal{N}} + O(\Xi\mathbf{u})$.

4.3.1 Solution with HyKKT

We analyze the accuracy achieved when we solve the condensed system (K_1) using HyKKT, and show that the error remains reasonable even for large values of the regularization parameter γ.

Initial right-hand-side. In floating-point arithmetic, the initial RHS in (9) is evaluated as $\widehat{r}_\gamma := \widehat{G}\widehat{K}_\gamma^{-1}(\bar{r}_1 + \gamma\widehat{G}^\top\bar{r}_2) - \bar{r}_2$. Using (35), we have

$$\begin{aligned}\bar{r}_1 + \gamma\widehat{G}^\top\bar{r}_2 &= -\widehat{r}_1 + \gamma\widehat{G}^\top\widehat{r}_3 + \widehat{H}^\top\big(\widehat{D}_s\widehat{r}_4 - \widehat{r}_2\big) \\ &= -\underbrace{\widehat{r}_1}_{O(\mathbf{u})} + \underbrace{\widehat{H}_\mathcal{N}^\top\big(\widehat{D}_{s,\mathcal{N}}\widehat{r}_{4,\mathcal{N}} - \widehat{r}_{2,\mathcal{N}}\big)}_{O(\Xi\mathbf{u})} + \underbrace{\widehat{A}^\top\begin{bmatrix}\widehat{D}_{s,\mathcal{B}}\widehat{r}_{4,\mathcal{B}} - \widehat{r}_{2,\mathcal{B}} \\ \gamma\widehat{r}_3\end{bmatrix}}_{O(\Xi^{-1}\mathbf{u})} . \quad (36)\end{aligned}$$

We note $\widehat{s}_\gamma = \bar{r}_1 + \gamma\widehat{G}^\top\bar{r}_2 = Y\widehat{s}_Y + Z\widehat{s}_Z$. The error decomposes as $\Delta s_\gamma = Y\Delta s_Y Y + Z\Delta s_Z = U_L\Delta s_L + U_S\Delta s_S$. Using (36), we have $\Delta s_Y = O(\Xi^{-1}\mathbf{u})$ and $\Delta s_Z = O(\mathbf{u})$. Using (23), we deduce $\Delta s_L = U_L^\top\Delta s_\gamma = O(\Xi^{-1}\mathbf{u})$ and $\Delta s_S = U_S^\top\Delta s_\gamma = O(\mathbf{u})$. We obtain using (24) and (29) that the error in the large space Δs_L annihilates in the backsolve:

$$K_\gamma^{-1}\Delta s_\gamma = U_L\Sigma_L^{-1}\Delta s_L + U_S\Sigma_S^{-1}\Delta s_S = O(\mathbf{u}) . \quad (37)$$

Using (33), we get

$$\widehat{G}\widehat{K}_\gamma^{-1}\Delta s_\gamma \approx \widehat{G}(I - K_\gamma^{-1}\Delta K_\gamma)K_\gamma^{-1}\Delta s_\gamma . \quad (38)$$

Using (37), the first term is $\widehat{G}K_\gamma^{-1}\Delta s_\gamma = O(\mathbf{u})$. We have in addition

$$GK_\gamma^{-1}\Delta K_\gamma(K_\gamma^{-1}\Delta s_\gamma) = \big[GU_L\Sigma_L^{-1}\Gamma_L + GU_S\Sigma_S^{-1}\Gamma_S\big](K_\gamma^{-1}\Delta s_\gamma) . \quad (39)$$

Using again (23): $GU_L = GY + O(\Xi)$ and $GU_S = GZ + O(\Xi) = O(\Xi)$. Hence $GU_L\Sigma_L^{-1}\Gamma_L = O(1)$ and $GU_S\Sigma_S^{-1}\Gamma_S = O(1)$. Using (32), we have $K_\gamma^{-1}\Delta G^\top = O(\mathbf{u})$, implying $\Delta GK_\gamma^{-1}\Delta K_\gamma(K_\gamma^{-1}\Delta s_\gamma) = O(\Xi^{-1}\mathbf{u}^2)$. Assumption 4.6 implies that $\Xi^{-1}\mathbf{u}^2 \ll \mathbf{u}$. All in all, the error in the right-hand-side $\Delta\widehat{r}_\gamma$ satisfies:

$$\Delta\widehat{r}_\gamma = -\Delta\bar{r}_2 + \widehat{G}\widehat{K}_\gamma^{-1}\Delta s_\gamma = O(\mathbf{u}) . \quad (40)$$

The result is remarkable: despite an expression involving the inverse of the ill-conditioned condensed matrix K_γ, the error made in r_γ is bounded only by the machine precision $\mathbf{u}$.

Schur-complement operator. The solution of the system (9) involves the Schur complement $S_\gamma = GK_\gamma^{-1}G^\top$. We show that the Schur complement has a specific structure that limits the loss of accuracy in the conjugate gradient algorithm.

Proposition 4.7. *Suppose the current primal-dual iterate (p, v) satisfies Assumption 4.6. In exact arithmetic,*

$$S_\gamma = GY\,\Sigma_L^{-1}\,Y^\top G^\top + O(\Xi^2)\,. \tag{41}$$

Proof. Using (29), we have

$$GK_\gamma^{-1}G^\top = GU_L\Sigma_L^{-1}U_L^\top G^\top + GU_S\Sigma_S^{-1}U_S^\top G^\top\,. \tag{42}$$

Using (23), we have $GU_L = GY + O(\Xi)$, and $G = O(1)$, implying

$$GU_L\Sigma_L^{-1}U_L^\top G^\top = GY\Sigma_L^{-1}Y^\top G^\top + O(\Xi^2)\,. \tag{43}$$

Using again (23), we have $GU_S = GZ + O(\Xi) = O(\Xi)$. Hence, $GU_S\Sigma_S^{-1}U_S^\top G^\top = O(\Xi^2)$, concluding the proof. □

We adapt the previous proposition to bound the error made when evaluating $\widehat{S}_\gamma$ in floating-point arithmetic.

Proposition 4.8. *Suppose the current primal-dual iterate (p, v) satisfies Assumption 4.6. In floating-point arithmetic,*

$$\widehat{S}_\gamma = S_\gamma + O(\mathbf{u})\,. \tag{44}$$

Proof. We note $\widehat{G} = G + \Delta G$ (with $G = O(\mathbf{u})$). Then

$$\begin{aligned}\widehat{S}_\gamma &= \widehat{G}\widehat{K}_\gamma^{-1}\widehat{G}^\top\,,\\ &\approx (G+\Delta G)\big(K_\gamma^{-1} - K_\gamma^{-1}\Delta K_\gamma K_\gamma^{-1}\big)(G+\Delta G)^\top\,,\\ &\approx S_\gamma - G\big(K_\gamma^{-1}\Delta K_\gamma K_\gamma^{-1}\big)G^\top + K_\gamma^{-1}\Delta G^\top + \Delta G K_\gamma^{-1}\,.\end{aligned} \tag{45}$$

The second line is given by (33), the third by neglecting the second-order errors. Using (32), we get $K_\gamma^{-1}\Delta G^\top = O(\mathbf{u})$ and $\Delta G K_\gamma^{-1} = O(\mathbf{u})$. Using (34), we have

$$G\big(K_\gamma^{-1}\Delta K_\gamma K_\gamma^{-1}\big)G^\top = GU_L\Sigma_L^{-1}\Gamma_L\Sigma_L^{-1}U_L^\top G^\top + GU_S\Sigma_S^{-1}\Gamma_S\Sigma_S^{-1}U_S^\top G^\top\,.$$

Using (23), we have $GU_S = O(\Xi)$, and as $\Sigma_S^{-1} = \Theta(1)$ and $\Gamma_S = O(\Xi^{-1}\mathbf{u})$, we get $GU_S\Sigma_S^{-1}\Gamma_S\Sigma_S^{-1}U_S^\top G^\top = O(\Xi\mathbf{u})$. Finally, as $\Sigma_L^{-1} = \Theta(\Xi)$ and $GU_L = GY + O(\Xi)$, we have

$$GU_L\Sigma_L^{-1}\Gamma_L\Sigma_L^{-1}U_L^\top G^\top = GY\Sigma_L^{-1}\Gamma_L\Sigma_L^{-1}Y^\top G^\top + O(\Xi^2\mathbf{u})\,. \tag{46}$$

We conclude the proof by using $GY\Sigma_L^{-1}\Gamma_L\Sigma_L^{-1}Y^\top G^\top = O(\Xi\mathbf{u})$. □

The two error bounds (40) and (44) ensure that we can solve (9) using a conjugate gradient algorithm, as the errors remain limited in floating-point arithmetic.

4.3.2 Solution with sparse-condensed KKT system

The sparse-condensed KKT system has removed the equality constraints from the optimization problems, simplifying the solution of the condensed KKT system to

$$K d_x = -r_1 + H^\top (D_s r_4 - r_2) \; . \tag{47}$$

Hence, the active Jacobian A reduces to the active inequalities $A = H_{\mathcal{B}}$. Using a similar analysis than in (36), the error in the right-hand-side is $O(\Xi^{-1}\mathbf{u})$ and is in the range space of the active Jacobian A. Using (29), we can show that the absolute error on $\widehat{d}_x$ is bounded by $O(\|d_x\|\mathbf{u}) = O(\Xi\mathbf{u})$. That implies the descent direction $\widehat{d}_x$ retains full relative precision close to optimality. In other words, we can refine the solution returned by the Cholesky solver accurately using iterative refinement.

4.3.3 Summary

Numerically, the primal-dual step $(\widehat{d}_x, \widehat{d}_y)$ is computed only with an (absolute) precision ε_K, greater than the machine precision $\mathbf{u}$ (for HyKKT, ε_K is the absolute tolerance of the CG algorithm, for SCKKT the absolute tolerance of the iterative refinement algorithm ε_K).

The errors $\widehat{d}_x - d_x = O(\varepsilon_K)$ and $\widehat{d}_y - d_y = O(\varepsilon_K)$ propagate further in $(\widehat{d}_s, \widehat{d}_z)$. According to (K_1), we have $\widehat{d}_s = -\widehat{r}_4 - \widehat{H}\widehat{d}_x$. By continuity, $\widehat{H} = H + O(\mathbf{u})$ and $\widehat{r}_4 = r_4 + O(\mathbf{u})$, implying

$$\widehat{d}_s = d_s + O(\varepsilon_K) \; . \tag{48}$$

Eventually, we obtain $\widehat{d}_z = -\widehat{r}_2 - \widehat{D}_s\widehat{d}_s$, giving the following bounds for the errors in the inactive and active components:

$$\begin{aligned} \widehat{d}_{z,\mathcal{B}} &= -\widehat{r}_{2,\mathcal{B}} - \widehat{D}_{s,\mathcal{B}}\widehat{d}_{s,\mathcal{B}} = d_{z,\mathcal{B}} + O(\max\{\mathbf{u}, \varepsilon_K \Xi^{-1}\}) \; , \\ \widehat{d}_{z,\mathcal{N}} &= -\widehat{r}_{2,\mathcal{N}} - \widehat{D}_{s,\mathcal{N}}\widehat{d}_{s,\mathcal{N}} = d_{z,\mathcal{N}} + O(\varepsilon_K \Xi) \; . \end{aligned} \tag{49}$$

The error arises mostly in the active components $\widehat{d}_{z,\mathcal{B}}$. We want to set $\varepsilon_K \ll \Xi$ to limit the loss of accuracy, but doing so is not trivial as we are approaching the optimum. For example, solving the condensed system with a very good tolerance $\varepsilon_K = 10^{-12}$ would only ensure that the absolute error is bounded by 10^{-4} if $\Xi = 10^{-8}$ (as it is typical in primal-dual IPM). The impact remains limited if we have only a few active inequality constraints.

5 Numerical results

5.1 Implementation

IPM solver. We have implemented the three KKT solvers in our nonlinear IPM solver MadNLP [26], using the `AbstractKKTSystem` abstraction. MadNLP

is able to deport most of the IPM algorithm on the GPU, at the exception of basic IPM operations used for the coordination (such as the filter line-search algorithm).

Evaluation of the nonlinear models. We use the modeler ExaModels.jl [27] to implement all the nonlinear programs used in our benchmark. ExaModels.jl leverages the implicit sparsity structure and provides custom derivative kernels for repeated algebraic subexpressions of the constraints and the objective kernels to compute the first and second-order derivatives of the GPU kernels in parallel [3, 21]. This caters towards the SIMD architecture of the GPU by assigning each expression to multiple threads that compute derivates for different values.

Linear solvers. We factorize the KKT systems assembled inside MadNLP using different sparse linear solvers, depending on the KKT formulation and the device we are using. Namely,

- `HSL ma27`: Implement the LBL factorization on the CPU. Solve the augmented KKT system (K_2). This solver is used as the reference running on the CPU.
- `CHOLMOD`: Implement Cholesky factorization on the CPU (using AMD ordering by default). Factorize the condensed matrix K_γ appearing in (8). This solver is used to assess the performance of the Golub & Greif strategy when running on the CPU.
- `cuDSS`: Unified interface to compute the Cholesky, LU or LDL decompositions on an NVIDIA GPU. We use the Cholesky factorization to decompose the same condensed matrix K_γ. This newly released solver is the contender in our benchmark.

For the Krylov iterative method, we rely on the implementation provided in the Julia package Krylov.jl [19], where all methods have been ported on the GPU.

5.2 Performance analysis on a large-scale instance

We first assess the performance of each KKT solver on a large-scale OPF instance, taken from the PGLIB benchmark [2]: `case9241pegase`. We have implemented the model in ExaModels.jl, to support the evaluation of the GPU: our formulation has a total of 85,568 variables, 82,679 equality constraints and 48,147 inequality constraints.jl. It is acknowledged that this particular instance is challenging to solve with IPM, as it requires a non-trivial amount of primal-dual regularization to achieve convergence. Our previous work has pointed out that the KKT solver is the current bottleneck in the numerical implementation [27].

5.2.1 Individual performance of the linear solvers

First, we assess the individual performance of the cuDSS solver when factorizing the matrix K_γ at the first IPM iteration (here with $\gamma = 10^7$). We compare the analysis time, the factorization time and the triangular solve time with CHOLMOD. The results are displayed in Table 1. We benchmark the three decompositions implemented in cuDSS (Cholesky, LU and LDL), and look at the accuracy obtained in the backsolve. We observe the analysis phase is almost 10 times slower for cuDSS, compared to CHOLMOD. That operation has to be computed only once, meaning that its cost is amortized if we run many IPM iterations. The refactorization is about twice as fast in cuDSS, with a time almost independent from the algorithm being used. The backsolve is 3 times slower, as the triangular solve algorithm does not parallelize well on a GPU. In terms of accuracy, the quality of the solution remains on par with CHOLMOD, except for the LDL decomposition which lags behind by at least 2 orders of magnitude.

	analysis (s)	factorization (s)	backsolve (s)	accuracy
cholmod	0.065	4.50×10^{-2}	2.23×10^{-3}	1.84×10^{-13}
cudss-cholesky	0.441	1.93×10^{-2}	7.66×10^{-3}	1.74×10^{-13}
cudss-lu	0.454	2.03×10^{-2}	6.69×10^{-3}	1.70×10^{-13}
cudss-ldl	0.454	2.04×10^{-2}	6.95×10^{-3}	3.36×10^{-11}

Table 1: Comparing the performance of cuDSS with CHOLMOD. The matrix K_γ is symmetric positive definite, with a size $n = 85,568$. The matrix is super sparse: the number of nonzeroes is $1,057,200$ (0.01%). (A30 GPU)

5.2.2 Tuning the Golub & Greif strategy

In Figure 1 we depict the evolution of the number of CG iterations and accuracy, while increasing the parameter γ from 10^4 to 10^8. The associated table details the time spent in the algorithm.

On the algorithmic side, we observe that the higher the regularization γ, the fastest the CG algorithm: we decrease the total number of iterations spent in CG by a factor of 10. However, we have to pay a price in term of accuracy: for $\gamma > 10^8$ the solution returned by the linear solver is not accurate enough and the IPM algorithm has to proceed to more primal-dual regularization, leading to an increase in the total number of iterations.

On the numerical side, the table in Figure 1 compares the time spent in the IPM solver on the CPU (using CHOLMOD) and on the GPU (using the solver `cuDSS`). We observe that overall `cuDSS` is faster than CHOLMOD, leading to a decrease in the total IPM solution time. We note that we decrease by a factor of 5 the time to assemble the condensed matrix K_γ on the GPU, meaning that this operation is not a bottleneck in the algorithm.

	CHOLMOD (CPU)					cuDSS (CUDA)				
γ	# it	cond. (s)	CG (s)	linsol (s)	**IPM (s)**	# it	cond. (s)	CG (s)	linsol (s)	**IPM (s)**
10^4	63	0.42	16.87	21.36	**23.85**	62	0.09	30.24	31.05	**33.51**
10^5	63	0.42	6.13	10.66	**13.42**	62	0.09	10.44	11.24	**13.64**
10^6	63	0.43	2.58	7.09	**9.53**	62	0.09	4.53	5.33	**7.69**
10^7	63	0.43	1.42	6.11	**8.81**	62	0.09	2.44	3.24	**5.62**
10^8	105	1.11	1.78	12.39	**16.77**	90	0.20	2.51	3.82	**7.90**

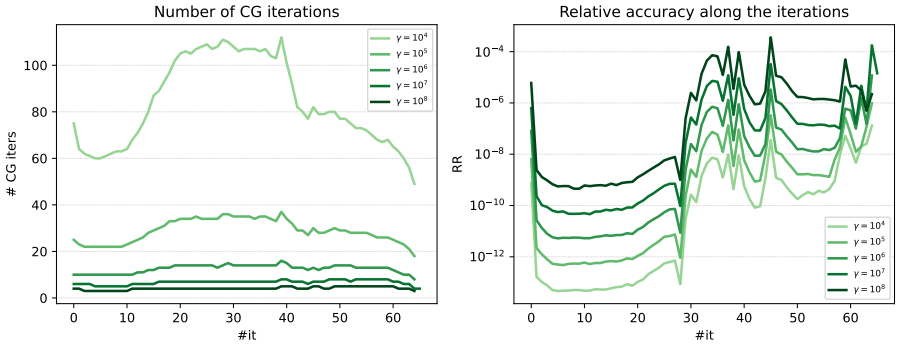

Figure 1: Above: Decomposition of IPM solution time across (a) condensation time (cond.), (b) CG time, (c) total time spent in linear solver (linsol.) and (d) total time spent in IPM solver (IPM). Below: Impact of γ on the total number of CG iterations and on the norm of the relative residual at each IPM iteration. The peak observe in the norm of the relative residual correspond to the primal-dual regularization performed inside the IPM algorithm, applied when the matrix K_γ is not positive definite. (A30 GPU)

5.2.3 Tuning the equality relaxation strategy

We now analyze the numerical performance of the equality relaxation strategy described in Section 3.2. The method solves the KKT system (K_1) using a direct solver, as the relaxed problem does not have any equality constraints. The parameter ε used in the equality relaxation $-\varepsilon \leq g(x) \leq \varepsilon$ is set equal to the IPM tolerance ε_{tol}: in practice, it does not make sense to set a parameter ε below the IPM tolerance as the inequality constraints are satisfied only up to a tolerance $\pm\varepsilon_{tol}$ in IPM.

We compare the performance obtained by the equality relaxation strategy in Table 2, for different values of ε_{tol}. We display both the runtimes on the CPU (using CHOLMOD) and on the GPU (using cuDSS), as well as the relative accuracy achieved as we decrease the tolerance ε_{tol}: we compare the solution returned by the equality relaxation strategy together with the solution returned by MadNLP when running on the CPU with HSL ma27 for a tight tolerance ($\varepsilon_{tol} = 10^{-8}$). We observe that we cannot solve the relaxed problem in MadNLP with a tolerance below 10^{-5}. Indeed, the slacks associated to the relaxed equality constraints $-\varepsilon \leq g(x) \leq \varepsilon$ are converging to a value below 2ε at the optimum, leading to highly ill-conditioned terms in the diagonal matrices Σ_s: despite being easier to solve, the conditioning of the relaxed problem is

preventing the IPM solver to converge to a solution more accurate than 10^{-5}.

This observation is corroborated by the results in Table 2. Overall, we observe that the method is converging fast if the tolerance ε_{tol} is above 10^{-4}. The method benefits from the GPU-accelerated linear solver cUDSS, as the running time decreases by a factor of 4 compared to CHOLMOD running on the CPU. However, the method remains inaccurate: for $\varepsilon_{tol} = 10^{-4}$, we get a 0.5% error on the objective, and a 10% error on the primal solution. To achieve convergence for the lower tolerance $\varepsilon_{tol} = 10^{-5}$, we had to decrease the tolerance of the iterative refinement to 10^{-12} to get a solution accurate enough in the linear solve: this explains the significant slowdown, as we have to perform more backsolve per iteration thanks to the additional iterative refinement iterations.

	CPU (CHOLMOD)		**CUDA (cuDSS)**		**accuracy**	
ε_{tol}	#it	time (s)	#it	time (s)	δ_{obj}	δ_x
10^{-2}	86	11.7	86	1.9	3.4×10^{-1}	0.66
10^{-3}	85	11.7	85	1.8	3.9×10^{-2}	0.22
10^{-4}	88	14.1	86	2.1	4.1×10^{-3}	0.10
10^{-5}	140	38.0	184	11.7	4.6×10^{-4}	0.06

Table 2: Performance of the equality-relaxation strategy as we decrease the IPM tolerance ε_{tol}. . The table displays the wall time on the CPU (using CHOLMOD) and on the GPU (using cuDSS). We display the relative errors on the objective $\delta_{obj} = (f(x^\sharp_{hsl}) - f(x^\sharp_{sc}))/f(x^\sharp_{hsl})$ and on the primal solution $\delta_x = \|x^\sharp_{hsl} - x^\sharp_{sc}\|_\infty / \|x^\sharp_{hsl}\|_\infty$. (A30 GPU)

5.2.4 Breakdown of the time spent in one IPM iteration

Now, we decompose the total wall running time spent at a single IPM iteration, for the different linear solvers we have at our disposal. We look at the 5th IPM iteration obtained when solving `case9241pegase`. The results are displayed in Figure 2. We observe that building the KKT system represents only a fraction of the computation time, compared to the factorization and the backsolve. The solver `cuDSS` is significantly faster to refactorize the KKT system, compared to the other linear solvers running on the CPU (ma27 and CHOLMOD). The backsolve is the bottleneck operation for HyKKT, as it has to run a full CG algorithm to solve the system (9). Hence, the performance of the HyKKT algorithm significantly improves if we manage to reduce the total number of iterations in the CG algorithm, e.g. by increasing the parameter γ.

5.3 Benchmark on OPF instances

The previous subsection has detailed the performance of the three KKT solvers on a specific instance, and highlighted the respective downsides of the null-space strategy, the Golub & Greif strategy and the equality relaxation strategy. Now, we run a full benchmark on difficult OPF instances taken from the PGLIB benchmark [2]. We compare our 3 GPU-accelerated KKT solvers with HSL

	build (s)	factorize (s)	backsolve (s)	accuracy
hsl	2.86e-03	4.78e-02	1.74e-02	7.80e-07
sckkt-cpu	5.41e-03	5.05e-02	1.96e-02	1.36e-04
hckkt-cpu	5.20e-03	5.10e-02	3.02e-02	3.28e-03
sckkt-cuda	1.55e-03	1.81e-02	1.01e-02	8.64e-06
hckkt-cuda	1.40e-03	1.43e-02	5.22e-02	2.66e-03

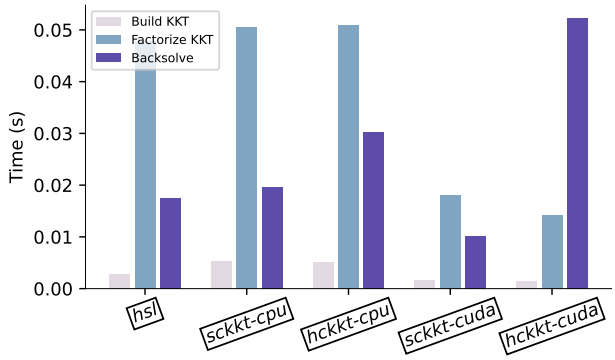

Figure 2: Breakdown of the time spent in one IPM iteration for different linear solvers, when solving `case9241pegase` (A30 GPU)

ma27 running on the CPU. The results are displayed in Table 3 for an IPM tolerance set to 10^{-4}. We complement the table with a Dolan & Moré performance profile [6] displayed in Figure 3.

Overall, the performance of HSL ma27 is consistent with what was reported in [2]. We note that HSL ma57 is slower than HSL ma27 on this particular benchmark, as the OPF instances are super-sparse: hence, the block elimitation algorithm implemented in HSL ma57 is not beneficial there [1]. In addition, we observe that the AD is not the bottleneck in that benchmark, as it contributes to less than 10% of the running time on the CPU, and it parallelizes effectively on the GPU when we use ExaModels.

Overall, the equality relaxation strategy gives good results and is faster than the Golub & Greif strategy on small and medium instances: indeed, the algorithm does not have to run a CG algorithm at each IPM iteration, limiting the number of backsolves to one. However, the method solves only a relaxation of the original problem, and suffers from the limited accuracy of the equality relaxation strategy: the algorithm converges in 811 iterations for `30000_goc`, leading to a slower solution time than with HSL ma27.

Regarding HyKKT, we manage to solve all the instances for $\gamma = 10^7$, limiting the need to tune the parameter γ on that particular benchmark. The algorithm is significantly faster than HSL ma27, but it remains on average slightly slower than the equality relaxation strategy: the algorithm's weakness is the multiple backsolves computed at each IPM iteration. For example, compared to the other instances, the solution of `8387_pegase` is impaired by a larger total number of

[1] Personal communication with Ian Duff.

CG iterations leading to a 4x slowdown compared to the equality relaxation strategy. Nevertheless, the performance of HyKKT is better on the largest instances, with almost an 8x speed-up compared to the reference HSL ma27.

	HSL ma27				eq. relaxation+cuDSS				HyKKT+cuDSS			
Case	it	AD	lin	**total**	it	AD	lin	**total**	it	AD	lin	**total**
89_pegase	30	0.00	0.02	**0.03**	28	0.02	0.03	**0.13**	30	0.03	0.06	**0.17**
179_goc	43	0.00	0.03	**0.05**	30	0.02	0.04	**0.14**	43	0.03	0.06	**0.20**
500_goc	35	0.01	0.08	**0.12**	36	0.02	0.04	**0.19**	35	0.03	0.06	**0.20**
793_goc	31	0.01	0.10	**0.15**	33	0.02	0.05	**0.20**	31	0.03	0.08	**0.23**
1354_pegase	42	0.03	0.29	**0.42**	44	0.04	0.08	**0.32**	42	0.04	0.14	**0.39**
2000_goc	38	0.05	0.57	**0.79**	36	0.03	0.07	**0.31**	38	0.03	0.13	**0.42**
2312_goc	38	0.05	0.50	**0.69**	38	0.03	0.10	**0.35**	38	0.03	0.18	**0.46**
2742_goc	121	0.27	3.52	**5.22**	-	-	-	**-**	-	-	-	**-**
2869_pegase	51	0.10	0.97	**1.34**	52	0.04	0.14	**0.50**	51	0.05	0.26	**0.65**
3022_goc	47	0.08	0.80	**1.11**	43	0.03	0.12	**0.44**	47	0.04	0.19	**0.54**
3970_goc	44	0.12	1.71	**2.35**	44	0.04	0.14	**0.57**	44	0.04	0.23	**0.70**
4020_goc	57	0.17	3.62	**4.41**	68	0.06	0.36	**1.00**	57	0.06	0.41	**1.02**
4601_goc	66	0.20	2.79	**3.70**	71	0.06	0.22	**0.82**	66	0.06	0.38	**1.01**
4619_goc	46	0.16	2.90	**3.67**	54	0.04	0.20	**0.77**	46	0.05	0.32	**0.89**
4837_goc	55	0.18	2.26	**3.12**	57	0.05	0.33	**0.89**	55	0.05	0.37	**0.93**
4917_goc	55	0.16	1.67	**2.45**	48	0.04	0.19	**0.64**	55	0.05	0.30	**0.80**
5658_epigrids	46	0.18	2.45	**3.35**	50	0.05	0.28	**0.86**	46	0.05	0.35	**0.94**
7336_epigrids	45	0.22	3.13	**4.18**	46	0.05	0.30	**1.00**	45	0.05	0.38	**1.07**
8387_pegase	70	0.44	4.99	**7.07**	67	0.08	0.82	**1.93**	70	0.08	6.86	**7.97**
9241_pegase	62	0.44	5.22	**7.23**	63	0.08	0.73	**1.86**	62	0.07	1.14	**2.27**
9591_goc	64	0.45	10.40	**12.31**	69	0.07	0.60	**1.78**	64	0.08	0.87	**2.03**
10000_goc	75	0.44	5.26	**7.31**	56	0.08	0.30	**1.10**	75	0.09	0.86	**1.88**
10192_epigrids	50	0.42	6.94	**8.80**	52	0.08	0.46	**1.55**	50	0.08	0.79	**1.87**
10480_goc	65	0.54	10.71	**13.15**	71	0.09	0.93	**2.37**	65	0.08	1.21	**2.57**
13659_pegase	57	0.55	6.28	**8.77**	65	0.09	1.70	**3.17**	57	0.07	1.16	**2.64**
19402_goc	70	1.20	31.11	**35.93**	108	0.18	4.95	**8.37**	70	0.12	2.38	**4.84**
20758_epigrids	46	0.80	12.36	**15.83**	2838	4.63	325.55	**407.69**	46	0.08	1.83	**3.65**
30000_goc	130	2.66	37.02	**45.88**	158	0.34	5.56	**9.19**	130	0.24	4.05	**7.23**
78484_epigrids	96	7.17	166.35	**192.70**	128	0.48	15.97	**27.49**	96	0.41	9.75	**19.58**

Table 3: OPF benchmark , solved with a tolerance `tol=1e-4`. (A30 GPU)

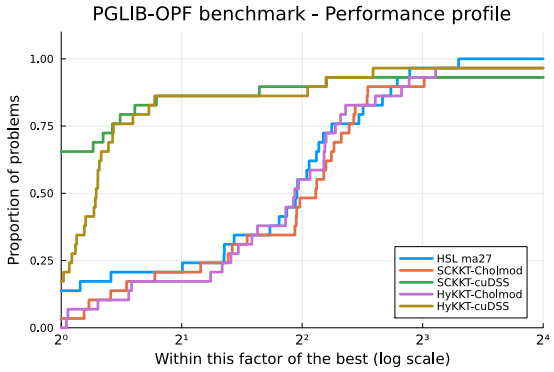

Figure 3: Performance profile for the PGLIB OPF benchmark, solved with a tolerance `tol=1e-4`.

5.4 Benchmark on COPS instances

We have observed in the previous section that both the equality relaxation and the HyKKT algorithm gives significant speed-up for large-scale OPF instances, compared to HSL ma27. However, the OPF instances are specific nonlinear problems. For that reason, we have analyzed further the performance of the equality relaxation and the HyKKT strategies on the COPS benchmark, which gathers generic nonlinear programs [7]. In this section, we look at the performance we get on the particular COPS instances used in the Mittelmann benchmark, widely used to benchmark nonlinear optimization solvers [17]. To illustrate the heterogeneity of the COPS instances compared to the previous OPF problems, we display in Figure 4 the sparsity pattern of the condensed matrices K_γ (8) for one OPF instance and for multiple COPS instances. We observe that some instances (`bearing`) have a sparsity pattern similar to the OPF instance on the left, whereas some are fully dense (`elec`). On the opposite, the optimal control instances (`marine`, `pinene`) are highly sparse and have highly structured Cholesky lower triangular factors. The results of the COPS benchmark are displayed in Table 4.

As expected, the results are different than the previous OPF benchmark. We observe that HyKKT+cuDSS and SCKKT+cuDSS outperform HSL ma27 on the dense instance `elec` (25x speed-up) and `bearing` (6x speed-up) — an instance whose sparsity pattern is closed to the OPF. In the other instances, both HyKKT+cuDSS and SCKKT+cuDSS are on par with HSL ma27 and sometimes even slightly slower (`steering`). In addition, the two solvers HyKKT+cholmod and SCKKT+cholmod offer approximatively the same performance as HyKKT+cuDSS and SCKKT+cuDSS, respectively, indicating that these particular instances are less amenable to GPU acceleration.

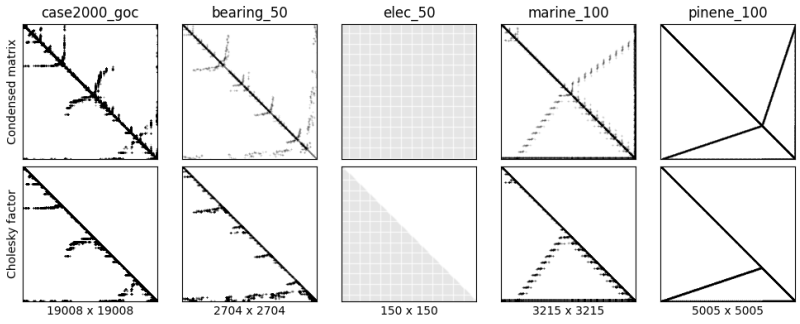

Figure 4: Sparsity patterns for one OPF instance and for various COPS problems. The first row displays the sparsity pattern of K_γ, after AMD reordering. The second row displays the sparsity pattern of the triangular factor computed by CHOLMOD.

	HSL ma27			SCKKT+cholmod			SCKKT+cuDSS			HyKKT+cholmod			HyKKT+cuDSS		
	it	lin	**total**	it	lin	**total**	it	lin	**total**	it	lin	**total**	it	lin	**total**
bearing_400	10	6.89	**7.26**	11	1.77	**2.16**	11	0.08	**0.83**	10	2.39	**2.72**	10	0.33	**1.10**
camshape_6400	21	0.09	**0.14**	17	0.02	**0.07**	15	0.01	**0.09**	21	0.03	**0.10**	18	0.02	**0.12**
elec_400	185	68.46	**77.85**	118	11.02	**14.29**	127	4.12	**6.99**	98	8.91	**11.75**	92	0.43	**3.10**
gasoil_3200	18	0.88	**7.30**	14	0.51	**0.96**	12	0.06	**0.82**	18	0.88	**1.36**	18	0.29	**1.22**
marine_1600	14	0.61	**0.94**	21	0.44	**0.69**	19	0.09	**0.51**	13	0.32	**0.48**	13	0.12	**0.51**
pinene_3200	10	0.95	**8.31**	30	2.25	**3.57**	23	0.16	**1.51**	10	2.71	**3.38**	10	1.28	**2.34**
robot_1600	61	0.94	**1.28**	-	-	**-**	-	-	**-**	57	0.58	**0.80**	49	0.13	**1.17**
steering_12800	17	0.73	**0.98**	15	0.25	**0.63**	13	0.04	**1.09**	12	0.48	**0.75**	11	0.17	**1.36**

Table 4: COPS benchmark , solved with a tolerance `tol=1e-4` (A30 GPU)

6 Conclusion

This article moves one step further in the solution of generic nonlinear programs on GPU architectures. We have compared two approaches to solve the KKT systems arising at each interior-point iteration, both based on a condensation procedure. Despite the formation of an ill-conditioned matrix, our theoretical analysis shows that the loss of accuracy is benign in floating-point arithmetic, thanks to the specific properties of the interior-point method. Our numerical results show that both methods are competitive to solve large-scale nonlinear programs. Compared to the state-of-the-art HSL linear solvers, we achieve a 10x speed-up on large-scale OPF instances and quasi-dense instances (`elec`). While the results are more varied across the instances of the COPS benchmark, our performance consistently remains competitive with HSL.

Looking ahead, our future plans involve enhancing the robustness of the two condensed KKT methods, particularly focusing on stabilizing convergence for small tolerances (below 10^{-8}). It's worth noting that the sparse Cholesky solver can be further customized to meet the specific requirements of the interior-point method [33]. Enhancing the two methods on the GPU would enable the resolution of large-scale problems that are currently intractable on classical CPU architectures. Examples include multiperiod and security-constrained OPF problems, particularly when combined with a Schur-complement approach.

References

[1] B. Amos and J. Z. Kolter, *Optnet: Differentiable optimization as a layer in neural networks*, in International Conference on Machine Learning, PMLR, 2017, pp. 136–145.

[2] S. Babaeinejadsarookolaee, A. Birchfield, R. D. Christie, C. Coffrin, C. DeMarco, R. Diao, M. Ferris, S. Fliscounakis, S. Greene, R. Huang, et al., *The power grid library for benchmarking AC optimal power flow algorithms*, arXiv preprint arXiv:1908.02788, (2019).

[3] C. Bischof, A. Griewank, and D. Juedes, *Exploiting parallelism in automatic differentiation*, in Proceedings of the 5th international conference on Supercomputing, 1991, pp. 146–153.

[4] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-

Milne, and Q. Zhang, *JAX: composable transformations of Python+NumPy programs*, 2018.

[5] Y. Cao, A. Seth, and C. D. Laird, *An augmented lagrangian interior-point approach for large-scale NLP problems on graphics processing units*, Computers & Chemical Engineering, 85 (2016), pp. 76–83.

[6] ———, *An augmented lagrangian interior-point approach for large-scale nlp problems on graphics processing units*, Computers & Chemical Engineering, 85 (2016), pp. 76–83.

[7] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Mathematical programming, 91 (2002), pp. 201–213.

[8] E. D. Dolan, J. J. Moré, and T. S. Munson, *Benchmarking optimization software with COPS 3.0.*, tech. rep., Argonne National Lab., Argonne, IL (US), 2004.

[9] I. S. Duff, *MA57—a code for the solution of sparse symmetric definite and indefinite systems*, ACM Transactions on Mathematical Software (TOMS), 30 (2004), pp. 118–144.

[10] I. S. Duff and J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear*, ACM Transactions on Mathematical Software (TOMS), 9 (1983), pp. 302–325.

[11] A. Ghannad, D. Orban, and M. A. Saunders, *Linear systems arising in interior methods for convex optimization: a symmetric formulation with bounded condition number*, Optimization Methods and Software, 37 (2022), pp. 1344–1369.

[12] G. H. Golub and C. Greif, *On solving block-structured indefinite linear systems*, SIAM Journal on Scientific Computing, 24 (2003), pp. 2076–2092.

[13] J. Gondzio, *Interior point methods 25 years later*, European Journal of Operational Research, 218 (2012), pp. 587–601.

[14] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, Journal of Research of the National Bureau of Standards, 49 (1952), pp. 409–436.

[15] Y. Kim, F. Pacaud, K. Kim, and M. Anitescu, *Leveraging GPU batching for scalable nonlinear programming through massive Lagrangian decomposition*, arXiv preprint arXiv:2106.14995, (2021).

[16] H. Lu and J. Yang, *cuPDLP.jl: A GPU implementation of restarted primal-dual hybrid gradient for linear programming in julia*, arXiv preprint arXiv:2311.12180, (2023).

[17] H. Lu, J. Yang, H. Hu, Q. Huangfu, J. Liu, T. Liu, Y. Ye, C. Zhang, and D. Ge, *cuPDLP-C: A strengthened implementation of cuPDLP for linear programming by C language*, arXiv preprint arXiv:2312.14832, (2023).

[18] S. Markidis, S. W. Der Chien, E. Laure, I. B. Peng, and J. S. Vetter, *Nvidia tensor core programmability, performance & precision*, in 2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW), IEEE, 2018, pp. 522–531.

[19] H. D. Mittelmann, *Benchmark for optimization software*, 2002.

[20] A. Montoison, *CUDSS.jl: Julia interface for NVIDIA cuDSS*.

[21] A. Montoison and D. Orban, *Krylov. jl: A julia basket of hand-picked krylov methods*, Journal of Open Source Software, 8 (2023), p. 5187.

[22] W. S. Moses, V. Churavy, L. Paehler, J. Hückelheim, S. H. K. Narayanan, M. Schanen, and J. Doerfert, *Reverse-mode automatic differentiation and optimization of GPU kernels via Enzyme*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '21, New York, NY, USA, 2021, Association for Computing Machinery.

[23] ———, *Reverse-mode automatic differentiation and optimization of gpu kernels via enzyme*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '21, New York, NY, USA, 2021, Association for Computing Machinery.

[24] J. Nocedal and S. J. Wright, *Numerical optimization*, Springer series in operations research, Springer, New York, 2nd ed., 2006.

[25] F. Pacaud, S. Shin, M. Schanen, D. A. Maldonado, and M. Anitescu, *Accelerating condensed interior-point methods on SIMD/GPU architectures*, Journal of Optimization Theory and Applications, (2023), pp. 1–20.

[26] L. Pineda, T. Fan, M. Monge, S. Venkataraman, P. Sodhi, R. T. Chen, J. Ortiz, D. DeTone, A. Wang, S. Anderson, et al., *Theseus: A library for differentiable nonlinear optimization*, Advances in Neural Information Processing Systems, 35 (2022), pp. 3801–3818.

[27] S. Regev, N.-Y. Chiang, E. Darve, C. G. Petra, M. A. Saunders, K. Świrydowicz, and S. Peleš, *HyKKT: a hybrid direct-iterative method for solving KKT linear systems*, Optimization Methods and Software, 38 (2023), pp. 332–355.

[28] S. Shin, C. Coffrin, K. Sundar, and V. M. Zavala, *Graph-based modeling and decomposition of energy infrastructures*, IFAC-PapersOnLine, 54 (2021), pp. 693–698.

[29] S. Shin, F. Pacaud, and M. Anitescu, *Accelerating optimal power flow with GPUs: SIMD abstraction of nonlinear programs and condensed-space interior-point methods*, arXiv preprint arXiv:2307.16830, (2023).

[30] K. Świrydowicz, E. Darve, W. Jones, J. Maack, S. Regev, M. A. Saunders, S. J. Thomas, and S. Peleš, *Linear solvers for power grid optimization problems: a review of GPU-accelerated linear solvers*, Parallel Computing, (2021), p. 102870.

[31] B. Tasseff, C. Coffrin, A. Wächter, and C. Laird, *Exploring benefits of linear solver parallelism on modern nonlinear optimization applications*, arXiv preprint arXiv:1909.08104, (2019).

[32] A. Wächter and L. T. Biegler, *On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming*, Mathematical Programming, 106 (2006), pp. 25–57.

[33] R. A. Waltz, J. L. Morales, J. Nocedal, and D. Orban, *An interior algorithm for nonlinear optimization that combines line search and trust region steps*, Mathematical Programming, 107 (2006), pp. 391–408.

[34] M. H. Wright, *Ill-conditioning and computational error in interior methods for nonlinear programming*, SIAM Journal on Optimization, 9 (1998), pp. 84–111.

[35] S. J. Wright, *Modified cholesky factorizations in interior-point algorithms for linear programming*, SIAM Journal on Optimization, 9 (1999), pp. 1159–1191.

[36] ———, *Effects of finite-precision arithmetic on interior-point methods for nonlinear programming*, SIAM Journal on Optimization, 12 (2001), pp. 36–78.