



**HAL**  
open science

# Condensed-space methods for nonlinear programming on GPUs

François Pacaud, Sungho Shin, Alexis Montoison, Michel Schanen, Mihai  
Anitescu

► **To cite this version:**

François Pacaud, Sungho Shin, Alexis Montoison, Michel Schanen, Mihai Anitescu. Condensed-space methods for nonlinear programming on GPUs. 2024. hal-04875904

**HAL Id: hal-04875904**

<https://minesparis-psl.hal.science/hal-04875904v1>

Preprint submitted on 9 Jan 2025

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Condensed-space methods for nonlinear programming on GPUs

François Pacaud · Sungho Shin · Alexis  
Montoison · Michel Schanen · Mihai  
Anitescu

July 23, 2024

**Abstract** This paper explores two condensed-space interior-point methods to efficiently solve large-scale nonlinear programs on graphics processing units (GPUs). The interior-point method solves a sequence of symmetric indefinite linear systems, or Karush-Kuhn-Tucker (KKT) systems, which become increasingly ill-conditioned as we approach the solution. Solving a KKT system with traditional sparse factorization methods involve numerical pivoting, making parallelization difficult. A solution is to condense the KKT system into a symmetric positive-definite matrix and solve it with a Cholesky factorization, stable without pivoting. Although condensed KKT systems are more prone to ill-conditioning than the original ones, they exhibit structured ill-conditioning that mitigates the loss of accuracy. This paper compares the benefits of two recent condensed-space interior-point methods, HyKKT and LiftedKKT. We implement the two methods on GPUs using MadNLP.jl, an optimization solver interfaced with the NVIDIA sparse linear solver cuDSS and with the GPU-accelerated modeler ExaModels.jl. Our experiments on the PGLIB and the COPS benchmarks reveal that GPUs can attain up to a tenfold speed increase compared to CPUs when solving large-scale instances.

## 1 Introduction

Graphics processing units (GPUs) are driving the advancement of scientific computing, their most remarkable success being the capabilities to train and utilize large artificial intelligence (AI) models. GPUs offer two practical advantages: (1) massive parallel computing capability for applications that can exploit coarse-grain parallelism and high-memory bandwidth and (2) power efficiency due to requiring fewer transistors to process multiple tasks in parallel utilizing “single instruction, multiple data” (SIMD) parallelism.

---

Address(es) of author(s) should be given

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 second-order optimization methods for constrained optimization solve a form of Newton’s method using direct linear algebra as finding good iterative solvers for the Newton direction has proved elusive. 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 [24].
2. **Interest in batch optimization:** There is a growing interest in solving parametric optimization problems in batch mode, for problems sharing the same structure but with different parameters [2, 33].
3. **Advancements in automatic differentiation:** GPUs offer unparalleled performance for automatic differentiation, benefiting both machine learning [7] and scientific computing applications [30]. 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 [37].
4. **Role in exascale computing:** With the emergence of new exascale supercomputers (e.g., Frontier and Aurora), the capabilities to run on GPUs have become central for supercomputing.

### 1.1 Solving optimization problems on GPU: current state-of-the-art

For all the reasons listed before, there is an increased interest for solving optimization problems on GPUs. We now summarize the previous work on solving classical—large-scale, sparse, constrained—mathematical programs on GPUs.

*GPU for mathematical programming.* The factorization of sparse matrices encountered within second-order optimization algorithms has been considered to be challenging on GPUs. For this reason, practitioners often have resorted to using first-order methods on GPUs, leveraging level-1 and level-2 BLAS operations that are more amenable to parallel computations. 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 [22, 23] based on the primal-dual hybrid gradient method has demonstrated 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.

*GPU for batched optimization solvers.* 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 [2, 33]. This has led to the development of prototype code solving thousands of (small) optimization problems in parallel on the GPU. Furthermore, batched optimization solvers can be leveraged in decomposition algorithms, when the subproblems share the same structure [21]. 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 directions.

*GPU for nonlinear programming.* The success of first-order algorithms in classical mathematical programming relies on the convexity of the problem. Thus, this approach is nontrivial to replicate in general nonlinear programming: Most engineering problems encode complex physical equations that are likely to break any convex structure in the problem. Previous experiments on the alternating current (AC) optimal power flow (OPF) problem have shown that even a simple algorithm as the alternating direction method of multipliers (ADMM) has trouble converging as soon as the convergence tolerance is set below  $10^{-3}$  [21].

Thus, second-order methods remain a competitive option, particularly for scenarios that demand higher levels of accuracy and robust convergence. Second-order algorithms solve 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 [39, 40]. Fortunately, sparse solvers on GPUs are becoming increasingly better: NVIDIA has released in November 2023 the cuDSS sparse direct solver that implements different sparse factorization routines with remarkably improved performance. Our benchmark results indicate that cuDSS is significantly faster than the previous sparse solvers using NVIDIA GPUs (e.g., published in [37]). Furthermore, variants of interior point methods have been proposed that do not depend on numerical pivoting in the linear solves, opening the door to parallelized sparse solvers. 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 [37].

There exist a few alternatives to sparse linear solvers for solving the KKT systems on the GPU. On the one hand, iterative and Krylov methods depend only on matrix-vector products to solve linear systems. They often require non-trivial reformulation or specialized preconditioning of the KKT systems to mitigate the inherent ill-conditioning of the KKT matrices, which has limited their use within the interior-point methods [10, 35]. New results are giving

promising outlooks for convex problems [17], but nonconvex problems often need an Augmented Lagrangian reformulation to be tractable [8, 34]. In particular, [34] presents an interesting use of the Golub and Greif hybrid method [18] to solve the KKT systems arising in the interior-point methods, with promising results on the GPU. On the other hand, null-space methods (also known as reduced Hessian methods) reduce the KKT system down to a dense matrix, a setting also favorable for GPUs. Our previous research has shown that the approach is suitable for interior-point methods if the number of degrees of freedom in the problem remains relatively small [32].

## 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 [34, 37]. We re-use classical results from [43] 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 [37]. 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 [26], 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 examined in our previous work, we incorporate large-scale problems sourced from the COPS nonlinear benchmark [13]. 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 IPM enables a remarkable ten-time 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 by  $\hat{x}$  its floating point representation. We denote  $\mathbf{u}$  as the smallest positive number such that  $\hat{x} \leq (1 + \tau)x$  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 [31]. 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 Section 2.1, we give a brief description of a nonlinear program. We detail in Section 2.2 the Newton step computation within each IPM iteration.

### 2.1 Problem formulation and KKT conditions

We are interested in solving the following nonlinear program:

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

with  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  a real-valued function encoding the objective,  $g : \mathbb{R}^n \rightarrow \mathbb{R}^{m_e}$  encoding the equality constraints, and  $h : \mathbb{R}^n \rightarrow \mathbb{R}^{m_i}$  encoding the inequality constraints. In what follows, we suppose that the functions  $f, g, h$  are smooth and twice differentiable.

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

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

In (2), the inequality constraints are encoded inside the variable bounds on the slack variables.

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

$$L(x, s, y, z, v) = f(x) + y^\top g(x) + z^\top (h(x) + s) - v^\top s. \quad (3)$$

The KKT conditions of (2) are:

$$\nabla f(x) + \nabla g(x)^\top y + \nabla h(x)^\top z = 0, \quad (4a)$$

$$z - v = 0, \quad (4b)$$

$$g(x) = 0, \quad (4c)$$

$$h(x) + s = 0, \quad (4d)$$

$$0 \leq s \perp v \geq 0. \quad (4e)$$

The notation  $s \perp v$  is a shorthand for the complementarity condition  $s_i v_i = 0$  (for all  $i = 1, \dots, n$ ).

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

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

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

## 2.2 Solving the KKT conditions with the interior-point method

The interior-point method aims at finding a stationary point satisfying the KKT conditions (4). The complementarity constraints (4e) render the KKT conditions non-smooth, complicating the solution of the whole system (4). IPM uses a homotopy continuation method to solve a simplified version of (4), parameterized by a barrier parameter  $\mu > 0$  [31, Chapter 19]. For positive  $(x, s, v) > 0$ , we solve the system

$$F_\mu(x, s, y, z, v) = \begin{bmatrix} \nabla f(x) + \nabla g(x)^\top y + \nabla h(x)^\top z \\ z - v \\ g(x) \\ h(x) + s \\ Sv - \mu e \end{bmatrix} = 0. \quad (6)$$

We introduce in (6) the diagonal matrices  $X = \text{diag}(x_1, \dots, x_n)$  and  $S = \text{diag}(s_1, \dots, s_{m_i})$ , along with the vector of ones  $e$ . As we drive the barrier parameter  $\mu$  to 0, the solution of the system  $F_\mu(x, s, y, z, v) = 0$  tends to the solution of the KKT conditions (4).

We note that at a fixed parameter  $\mu$ , the function  $F_\mu(\cdot)$  is smooth. Hence, the system (6) can be solved iteratively using a regular Newton method. For a primal-dual iterate  $w_k := (x_k, s_k, y_k, z_k, v_k)$ , the next iterate is computed as  $w_{k+1} = w_k + \alpha_k d_k$ , where  $d_k$  is a descent direction computed by solving the linear system

$$\nabla_w F_\mu(w_k) d_k = -F_\mu(w_k). \quad (7)$$

The step  $\alpha_k$  is computed using a line-search algorithm, in a way that ensures that the bounded variables remain positive at the next primal-dual iterate:  $(x_{k+1}, s_{k+1}, v_{k+1}) > 0$ . Once the iterates are sufficiently close to the central path, the IPM decreases the barrier parameter  $\mu$  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  $d_k$ . By writing out all the blocks, the system in (7) expands as the  $6 \times 6$  *unreduced KKT system*:

$$\begin{bmatrix} W_k & 0 & G_k^\top & H_k^\top & 0 \\ 0 & 0 & 0 & I & -I \\ G_k & 0 & 0 & 0 & 0 \\ H_k & I & 0 & 0 & 0 \\ 0 & V_k & 0 & 0 & S_k \end{bmatrix} \begin{bmatrix} d_x \\ d_s \\ d_y \\ d_z \\ d_v \end{bmatrix} = - \begin{bmatrix} \nabla_x L(w_k) \\ z_k - v_k \\ g(x_k) \\ h(x_k) + s_k \\ S_k v_k - \mu e \end{bmatrix}, \quad (K_3)$$

where we have introduced the Hessian  $W_k = \nabla_{xx}^2 L(w_k)$  and the two Jacobians  $G_k = \nabla g(x_k)$ ,  $H_k = \nabla h(x_k)$ . In addition, we define  $X_k$ ,  $S_k$ ,  $U_k$  and  $V_k$  the diagonal matrices built respectively from the vectors  $x_k$ ,  $s_k$ ,  $u_k$  and  $v_k$ . Note that  $(K_3)$  can be symmetrized by performing simple block row and column operations. In what follows, we will omit the index  $k$  to simplify the notations.

*Augmented KKT system.* It is usual to remove in  $(K_3)$  the blocks associated to the bound multipliers  $v$  and solve instead the regularized  $4 \times 4$  symmetric system, called the *augmented KKT system*:

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

with the diagonal matrix  $D_s := S^{-1}V$ . The vectors forming the right-hand-sides are given respectively by  $r_1 := \nabla f(x) + \nabla g(x)^\top y + \nabla h(x)^\top z$ ,  $r_2 := z - \mu S^{-1}e$ ,  $r_3 := g(x)$ ,  $r_4 := h(x) + s$ . Once  $(K_2)$  is solved, we recover the updates on bound multipliers with  $d_v = -S^{-1}(Vd_s - \mu e) - v$ .

Note that we have added additional regularization terms  $\delta_w \geq 0$  and  $\delta_c \geq 0$  in  $(K_2)$ , to ensure the matrix is invertible. Without the regularization terms in  $(K_2)$ , the augmented KKT system is non-singular if and only if the Jacobian  $J = \begin{bmatrix} G & 0 \\ H & I \end{bmatrix}$  is full row-rank and the matrix  $\begin{bmatrix} W & 0 \\ 0 & D_s \end{bmatrix}$  projected onto the null-space of the Jacobian  $J$  is definite [4]. The condition is satisfied if the inertia (defined as the respective numbers of positive, negative and zero eigenvalues) of the matrix  $(K_2)$  is  $(n + m_i, m_i + m_e, 0)$ . We use the inertia-controlling method introduced in [41] to regularize the augmented matrix by adding multiple of the identity on the diagonal of  $(K_2)$  if the inertia is not equal to  $(n + m_i, m_e + m_i, 0)$ .

As a consequence, the system  $(K_2)$  is usually factorized using an inertia-revealing LBL<sup>T</sup> factorization [15]. Krylov methods are often not competitive when solving  $(K_2)$ , as the block diagonal terms  $D_s$  are getting increasingly ill-conditioned near the solution. Their use in IPM has been limited to linear and convex quadratic programming [19] (when paired with a suitable preconditioner). We also refer to [8] for an efficient implementation of a preconditioned conjugate gradient on GPU, for solving the Newton step arising in an augmented Lagrangian interior-point approach.



*Condensed KKT system.* The  $4 \times 4$  KKT system ( $K_2$ ) can be further reduced down to a  $2 \times 2$  system by eliminating the two blocks ( $d_s, d_z$ ) associated to the inequality constraints. The resulting system is called the *condensed KKT system*:

$$\begin{bmatrix} K & G^\top \\ G & -\delta_c I \end{bmatrix} \begin{bmatrix} d_x \\ d_y \end{bmatrix} = - \begin{bmatrix} r_1 + H^\top (D_H r_4 - C r_2) \\ r_3 \end{bmatrix} =: \begin{bmatrix} \tilde{r}_1 \\ \tilde{r}_2 \end{bmatrix}, \quad (K_1)$$

where we have introduced the *condensed matrix*  $K := W + \delta_w I + H^\top D_H H$  and the two diagonal matrices

$$C := (I + \delta_c (D_s + \delta_w I))^{-1}, \quad D_H := (D_s + \delta_w I) C. \quad (8)$$

Using the solution of the system ( $K_1$ ), we recover the updates on the slacks and inequality multipliers with  $d_z = -C r_2 + D_H (H d_x + r_4)$  and  $d_s = -(D_s + \delta_w I)^{-1} (r_2 + d_z)$ . Using Sylvester's law of inertia, we can prove that

$$\text{inertia}(K_2) = (n + m_i, m_e + m_i, 0) \iff \text{inertia}(K_1) = (n, m_e, 0). \quad (9)$$

*Iterative refinement.* Compared to ( $K_3$ ), the diagonal matrix  $D_s$  introduces an additional ill-conditioning in ( $K_2$ ), amplified in the condensed form ( $K_1$ ): the elements in the diagonal tend to infinity if a variable converges to its bound, and to 0 if the variable is inactive. To address the numerical error arising from such ill-conditioning, most of the implementations of IPM employ Richardson iterations on the original system ( $K_3$ ) to refine the solution returned by the direct sparse linear solver (see [41, Section 3.10]).

### 2.3 Discussion

We have obtained three different formulations of the KKT systems appearing at each IPM iteration. The original formulation ( $K_3$ ) has a better conditioning than the two alternatives ( $K_2$ ) and ( $K_1$ ) but has a much larger size. The second formulation ( $K_2$ ) is used by default in state-of-the-art nonlinear solvers [41, 42]. The system ( $K_2$ ) is usually factorized using a LBL<sup>T</sup> factorization: for sparse matrices, the Duff and Reid multifrontal algorithm [15] is the favored method (as implemented in the HSL linear solvers MA27 and MA57 [14]). The condensed KKT system ( $K_1$ ) is often discarded, as its conditioning is worse than ( $K_2$ ) (implying less accurate solutions). Additionally, condensation may result in increased fill-in within the condensed system ( $K_1$ ) [31, Section 19.3, p.571]. In the worst cases ( $K_1$ ) itself may become fully dense if an inequality row is completely dense (fortunately, a case rarer than in the normal equations commonly encountered in linear programming). Consequently, condensation methods are not commonly utilized in practical optimization settings. To the best of our knowledge, Artelys Knitro [42] 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 a new prominent computing hardware not only for graphics-related but also for general-purpose 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. Previous research has demonstrated that GPU-based linear solvers significantly lag behind their CPU counterparts [39,40]. One emerging strategy is to utilize sparse factorization techniques that do not necessitate numerical pivoting [34,37] by leveraging the structure of the condensed KKT system  $(K_1)$ . We present two alternative methods to solve  $(K_1)$ . On the one hand, HyKKT is introduced in §3.1 and uses the hybrid strategy of Golub & Greif [18,34]. On the other hand, LiftedKKT [37] uses an equality relaxation strategy and is presented in §3.2.

#### 3.1 Golub & Greif strategy: HyKKT

The Golub & Greif [18] strategy reformulates the KKT system using an Augmented Lagrangian formulation. It has been recently revisited in [34] to solve the condensed KKT system  $(K_1)$  on the GPU. For a dual regularization  $\delta_c = 0$ , the trick is to reformulate the condensed KKT system  $(K_1)$  in an equivalent form

$$\begin{bmatrix} K_\gamma & G^\top \\ G & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \end{bmatrix} = \begin{bmatrix} \bar{r}_1 + \gamma G^\top \bar{r}_2 \\ \bar{r}_2 \end{bmatrix}, \quad (10)$$

where we have introduced the regularized matrix  $K_\gamma := K + \gamma G^\top G$ . We note by  $Z$  a basis of the null-space of the Jacobian  $G$ . Using a classical result from [11], if  $G$  is full row-rank then there exists a threshold value  $\underline{\gamma}$  such that for all  $\gamma > \underline{\gamma}$ , the reduced Hessian  $Z^\top K Z$  is positive definite if and only if  $K_\gamma$  is positive definite. Using the Sylvester's law of inertia stated in (9), we deduce that for  $\gamma > \underline{\gamma}$ , if  $\text{inertia}(K_2) = (n + m_i, m_e + m_i, 0)$  then  $K_\gamma$  is positive definite.

The linear solver HyKKT [34] leverages the positive definiteness of  $K_\gamma$  to solve (10) using a hybrid direct-iterative method that uses the following steps:

1. Assemble  $K_\gamma$  and factorize it using sparse Cholesky ;
2. Solve the Schur complement of (10) using a conjugate gradient (CG) algorithm to recover the dual descent direction:

$$(GK_\gamma^{-1}G^\top)d_y = GK_\gamma^{-1}(\bar{r}_1 + \gamma G^\top \bar{r}_2) - \bar{r}_2. \quad (11)$$

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 [20]. 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  $K_\gamma$ . 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  $\gamma$  [34, Theorem 4], implying that  $\lim_{\gamma \rightarrow \infty} \kappa_2(S_\gamma) = 1$ . Because the convergence of the CG method depends on the number of distinct eigenvalues of  $S_\gamma$ , the larger the  $\gamma$ , the faster the convergence of the CG algorithm in (11). CR [20] and cAR [29] can also be used as an alternative to CG. Although we observe similar performance, these methods ensure a monotonic decrease in the residual norm of (11) at each iteration.

### 3.2 Equality relaxation strategy: LiftedKKT

For a small relaxation parameter  $\tau > 0$  (chosen based on the numerical tolerance of the optimization solver  $\varepsilon_{tol}$ ), the equality relaxation strategy [37] approximates the equalities with lifted inequalities:

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad -\tau \leq g(x) \leq \tau, \quad h(x) \leq 0. \quad (12)$$

The problem (12) has only inequality constraints. After introducing slack variables, the condensed KKT system  $(K_1)$  reduces to

$$K_\tau d_x = -r_1 - H_\tau^\top (D_H r_4 - C r_2), \quad (13)$$

with  $H_\tau = (G^\top H^\top)^\top$  and  $K_\tau := W + \delta_w I + H_\tau^\top D_H H_\tau$ . Using the relation (9), the matrix  $K_\tau$  is guaranteed to be positive definite if the primal regularization parameter  $\delta_w$  is adequately large. As such, the parameter  $\delta_w$  is chosen dynamically using the inertia information of the system in  $(K_1)$ . Therefore,  $K_\tau$  can be factorized with a Cholesky decomposition, satisfying the key requirement of stable pivoting for the implementation on the GPU. The relaxation causes error in the final solution. Fortunately, the error is in the same order of the solver tolerance, thus it does not significantly deteriorate the solution quality for small  $\varepsilon_{tol}$ .

While this method can be implemented with small modification in the optimization solver, the presence of tight inequality in (12) causes severe ill-conditioning throughout the IPM iterations. Thus, using an accurate iterative refinement algorithm is necessary to get a reliable convergence behavior.

### 3.3 Discussion

We have introduced two algorithms to solve KKT systems on the GPU. As opposed to classical implementations, the two methods do not require computing a sparse LBL<sup>T</sup> factorization of the KKT system and use instead alternate

reformulations based on the condensed KKT system  $(K_1)$ . Both strategies rely on a Cholesky factorization: HyKKT factorizes a positive definite matrix  $K_\gamma$  obtained with an Augmented Lagrangian strategy whereas Lifted KKT factorizes a positive definite matrix  $K_\tau$  after using an equality relaxation strategy. We will see in the next section that the ill-conditioned matrices  $K_\gamma$  and  $K_\tau$  have a specific structure that limits the loss of accuracy in IPM.

#### 4 Conditioning of the condensed KKT system

The condensed matrix  $K$  appearing in  $(K_1)$  is known to be increasingly ill-conditioned as the primal-dual iterates approach to a local solution with active inequalities. This behavior is amplified for the matrices  $K_\gamma$  and  $K_\tau$ , as HyKKT and LiftedKKT require the use of respectively large  $\gamma$  and small  $\tau$ . In this section, we analyze the numerical error associated with the solution of the condensed KKT system and discuss how the structured ill-conditioning makes the adverse effect of ill-conditioning relatively benign.

We first discuss the perturbation bound for a generic linear system  $Mx = b$ . The relative error after perturbing the right-hand side by  $\Delta b$  is bounded by:

$$\|\Delta x\| \leq \|M^{-1}\| \|\Delta b\|, \quad \frac{\|\Delta x\|}{\|x\|} \leq \kappa_2(M) \frac{\|\Delta b\|}{\|b\|}. \quad (14a)$$

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

$$\|\Delta x\| \lesssim \|M^{-1}\| \|\Delta M\| \|x\|, \quad \frac{\|\Delta x\|}{\|x\|} \lesssim \kappa_2(M) \frac{\|\Delta M\|}{\|M\|}. \quad (14b)$$

The relative errors are bounded above by a term depending on the conditioning  $\kappa_2(M)$ . Hence, it is legitimate to investigate the impact of the ill-conditioning when solving the condensed system  $(K_1)$  with LiftedKKT or with HyKKT. We will see that we can tighten the bounds in (14) by exploiting the structured ill-conditioning of the condensed matrix  $K$ . We base our analysis on [43], where the author has put a particular emphasis on the condensed KKT system  $(K_1)$  without equality constraints. We generalize their results to the matrix  $K_\gamma$ , which incorporates both equality and inequality constraints. The results extend directly to  $K_\tau$  (by letting the number of equalities to be zero).

##### 4.1 Centrality conditions

We start the discussion by recalling important results about the iterates of the interior-point algorithm. For  $p := (x, s, y, z)$ , we denote by  $(p, v)$  the current primal-dual iterate, and  $(p^*, v^*)$  a solution of the KKT conditions (4). We denote  $\mathcal{B} = \mathcal{B}(x^*)$  the active-set at the optimal solution  $x^*$ , and  $\mathcal{N} = \mathcal{N}(x^*)$  the inactive set. In this section, we are interested in the *local* convergence

behavior of the primal-dual iterate, and we suppose  $(p, v)$  is close enough to the solution  $(p^*, v^*)$ .

**Assumption 1** *Let  $(p^*, v^*)$  be a primal-dual solution satisfying the KKT conditions (4). Let the following hold:*

- *Continuity: The Hessian  $\nabla_{xx}^2 L(\cdot)$  is Lipschitz continuous near  $p^*$ ;*
- *Linear Independence Constraint Qualification (LICQ): the active Jacobian  $A(x^*)$  is full row-rank;*
- *Strict Complementarity (SCS): for every  $i \in \mathcal{B}(x^*)$ ,  $z_i^* > 0$ .*
- *Second-order sufficiency (SOSC): for every  $h \in \text{Ker}(A(x^*))$ ,  $h^\top \nabla_{xx}^2 L(p^*) h > 0$ .*

We denote  $\delta(p, v) = \|(p, v) - (p^*, v^*)\|$  the Euclidean distance to the primal-dual stationary point  $(p^*, v^*)$ . From [45, Theorem 2.2], if Assumption 1 holds at  $p^*$  and  $v > 0$ ,

$$\delta(p, v) = \Theta \left( \left\| \begin{bmatrix} \nabla_p L(p, v) \\ \min(v, s) \end{bmatrix} \right\| \right). \quad (15)$$

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

$$\Xi(s, v) = s^\top v / m_i, \quad (16)$$

where  $m_i$  is the number of inequality constraints. The duality measure encodes the current satisfaction of the complementarity constraints. For a solution  $(p^*, v^*)$ , we have  $\Xi(s^*, v^*) = 0$ . The duality measure can be used to define the barrier parameter in IPM.

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

$$\|\nabla_p \mathcal{L}(p, v)\| \leq C \Xi(s, v), \quad (17a)$$

$$(s, v) > 0, \quad s_i v_i \geq \alpha \Xi(s, v) \quad \forall i = 1, \dots, m_i, \quad (17b)$$

for some constants  $C > 0$  and  $\alpha \in (0, 1)$ . Conditions (17b) ensure that the products  $s_i v_i$  are not too disparate in the diagonal term  $D_s$ . This condition is satisfied (even if rather loosely) in the solver Ipopt (see [41, Equation (16)]).

**Proposition 1** ([45], **Lemma 3.2 and Theorem 3.3**) *Suppose  $p^*$  satisfies Assumption 1. If the current primal-dual iterate  $(p, v)$  satisfies the centrality conditions (17), then*

$$i \in \mathcal{B} \implies s_i = \Theta(\Xi), \quad v_i = \Theta(1), \quad (18a)$$

$$i \in \mathcal{N} \implies s_i = \Theta(1), \quad v_i = \Theta(\Xi). \quad (18b)$$

and the distance to the solution  $\delta(p, v)$  is bounded by the duality measure  $\Xi$ :

$$\delta(p, v) = O(\Xi). \quad (18c)$$

## 4.2 Structured ill-conditioning

The following subsection looks at the structure of the condensed matrix  $K_\gamma$  in HyKKT. All the results apply directly to the matrix  $K_\tau$  in LiftedKKT, by setting the number of equality constraints to  $m_e = 0$ . First, we show that if the iterates  $(p, v)$  satisfy the centrality conditions (17), then the condensed matrix  $K_\gamma$  exhibits a structured ill-conditioning.

### 4.2.1 Invariant subspaces in $K_\gamma$

Without regularization we have that  $K_\gamma = W + H^\top D_s H + \gamma G^\top G$ , with the diagonal  $D_s = S^{-1}V$ . We note by  $m_a$  the cardinality of the active set  $\mathcal{B}$ ,  $H_{\mathcal{B}}$  the Jacobian of active inequality constraints,  $H_{\mathcal{N}}$  the Jacobian of inactive inequality constraints and by  $A := [H_{\mathcal{B}}^\top G^\top]^\top$  the active Jacobian. We define the minimum and maximum active slack values as

$$s_{min} = \min_{i \in \mathcal{B}} s_i, \quad s_{max} = \max_{i \in \mathcal{B}} s_i. \quad (19)$$

We recall that  $m_e$  is the number of equality constraints, and define  $\ell := m_e + m_a$ .

We express the structured ill-conditioning of  $K_\gamma$  by modifying the approach outlined in [43, Theorem 3.2] to account for the additional term  $\gamma G^\top G$  arising from the equality constraints. We show that the matrix  $K_\gamma$  has two invariant subspaces (in the sense defined in [38, Chapter 5]), associated respectively to the range of the transposed active Jacobian (*large space*) and to the null space of the active Jacobian (*small space*).

**Theorem 2 (Properties of  $K_\gamma$ )** *Suppose the condensed matrix is evaluated at a primal-dual point  $(p, v)$  satisfying (17), for sufficiently small  $\Xi$ . Let  $\lambda_1, \dots, \lambda_n$  be the  $n$  eigenvalues of  $K_\gamma$ , ordered as  $|\lambda_1| \geq \dots \geq |\lambda_n|$ . Let  $[Y \ Z]$  be an orthogonal matrix, where  $Z$  encodes the basis of the null-space of  $A$ . Let  $\underline{\sigma} := \min(\frac{1}{\Xi}, \gamma)$  and  $\bar{\sigma} := \max(\frac{1}{s_{min}}, \gamma)$ . Then,*

- (i) *The  $\ell$  largest-magnitude eigenvalues of  $K_\gamma$  are positive, with  $\lambda_1 = \Theta(\bar{\sigma})$  and  $\lambda_\ell = \Omega(\underline{\sigma})$ .*
- (ii) *The  $n - \ell$  smallest-magnitude eigenvalues of  $K_\gamma$  are  $\Theta(1)$ .*
- (iii) *If  $0 < \ell < n$ , then  $\kappa_2(K_\gamma) = \Theta(\bar{\sigma})$ .*
- (iv) *There are orthonormal matrices  $\tilde{Y}$  and  $\tilde{Z}$  for simple invariant subspaces of  $K_\gamma$  such that  $Y - \tilde{Y} = O(\underline{\sigma}^{-1})$  and  $Z - \tilde{Z} = O(\underline{\sigma}^{-1})$ .*

*Proof* We start the proof by setting apart the inactive constraints from the active constraints in  $K_\gamma$ :

$$K_\gamma = W + H_{\mathcal{N}}^\top S_{\mathcal{N}}^{-1} V_{\mathcal{N}} H_{\mathcal{N}} + A^\top D_\gamma A, \quad \text{with} \quad D_\gamma = \begin{bmatrix} S_{\mathcal{B}}^{-1} V_{\mathcal{B}} & 0 \\ 0 & \gamma I \end{bmatrix}. \quad (20)$$

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

$$H_{\mathcal{N}}^{\top} S_{\mathcal{N}}^{-1} V_{\mathcal{N}} H_{\mathcal{N}} = O(\Xi). \quad (21)$$

Hence, for small enough  $\Xi$ , the condensed matrix  $K_{\gamma}$  is dominated by the block of active constraints:

$$K_{\gamma} = A^{\top} D_{\gamma} A + O(1). \quad (22)$$

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

$$\eta_1 = \Theta(\bar{\sigma}), \quad \eta_{\ell} = \Omega(\underline{\sigma}). \quad (23)$$

Using [43, Lemma 3.1], we deduce  $\lambda_1 = \Theta(\bar{\sigma})$  and  $\lambda_{\ell} = \Omega(\underline{\sigma})$ , proving the first result (i).

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

$$\begin{bmatrix} Z^{\top} \\ Y^{\top} \end{bmatrix} L_{\gamma} \begin{bmatrix} Z \\ Y \end{bmatrix} = \begin{bmatrix} L_1 & 0 \\ 0 & L_2 \end{bmatrix}, \quad (24)$$

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  $E := K_{\gamma} - L_{\gamma} = O(1)$ . By applying [43, Theorem 3.1, (ii)], the  $n - \ell$  smallest eigenvalues in  $K_{\gamma}$  differ by  $\Omega(\underline{\sigma}^{-1})$  from those of the reduced Hessian  $Z^{\top} K_{\gamma} Z$ . In addition, (21) 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  $\Xi$ , implying all its eigenvalues are  $\Theta(1)$ . Using again [43, Lemma 3.1], we get that the  $n - \ell$  smallest eigenvalues in  $K_{\gamma}$  are  $\Theta(1)$ , proving (ii). The results in (iii) can be obtained by combining (i) and (ii) (provided  $0 < \ell < n$ ). Finally, point (iv) directly follows from [43, Theorem 3.1 (i)].

**Corollary 1** *The condensed matrix  $K_{\gamma}$  can be decomposed as*

$$K_{\gamma} = U \Sigma U^{\top} = \begin{bmatrix} U_L & U_S \end{bmatrix} \begin{bmatrix} \Sigma_L & 0 \\ 0 & \Sigma_S \end{bmatrix} \begin{bmatrix} U_L^{\top} \\ U_S^{\top} \end{bmatrix}, \quad (25)$$

with  $\Sigma_L = \text{diag}(\sigma_1, \dots, \sigma_{\ell}) \in \mathbb{R}^{\ell \times \ell}$  and  $\Sigma_S = \text{diag}(\sigma_{\ell+1}, \dots, \sigma_n) \in \mathbb{R}^{(n-\ell) \times (n-\ell)}$  two diagonal matrices, and  $U_L \in \mathbb{R}^{n \times \ell}$ ,  $U_S \in \mathbb{R}^{n \times (n-\ell)}$  two orthogonal matrices such that  $U_L^{\top} U_S = 0$ . The diagonal elements in  $\Sigma_S$  and  $\Sigma_L$  satisfy

$$\frac{\sigma_1}{\sigma_{\ell}} \ll \frac{\sigma_1}{\sigma_n}, \quad \frac{\sigma_{\ell+1}}{\sigma_n} \ll \frac{\sigma_1}{\sigma_n}. \quad (26)$$

For suitably chosen basis  $Y$  and  $Z$ , spanning respectively the row space and the null space of the active Jacobian  $A$ , we get

$$U_L - Y = O(\underline{\sigma}^{-1}), \quad U_S - Z = O(\underline{\sigma}^{-1}). \quad (27)$$

*Proof* Using the spectral theorem, we obtain the decomposition as (25). According to Theorem 2, the  $\ell$  largest eigenvalues of  $K_\gamma$  are large and well separated from the  $n - \ell$  smallest eigenvalues, establishing (26). Using Theorem 2, part (iv), we obtain the result in (27).

Corollary 1 gives us a deeper insight into the structure of the condensed matrix  $K_\gamma$ . Using equation (27), we observe we can assimilate the large space of  $K_\gamma$  with  $\text{range}(A^\top)$  and the small space with  $\text{Ker}(A)$ . The decomposition (25) leads to the following relations

$$\begin{aligned} \|K_\gamma\| &= \|\Sigma_L\| = \Theta(\bar{\sigma}), & \Sigma_L^{-1} &= O(\underline{\sigma}^{-1}), \\ \|K_\gamma^{-1}\| &= \|\Sigma_S^{-1}\| = \Theta(1), & \Sigma_S &= \Theta(1). \end{aligned} \quad (28)$$

The condition of  $\Sigma_L$  depends on  $\kappa_2(A)$  and the ratio  $\frac{s_{max}}{s_{min}} = O(\Xi\bar{\sigma})$ . The condition of  $\Sigma_S$  reflects the condition of the reduced Hessian  $Z^\top WZ$ .

Three observations are due:

1. Theorem 2 (iii) tells us that  $\kappa_2(K_\gamma) = \Theta(\bar{\sigma})$ , meaning that if  $\gamma \geq \frac{1}{s_{min}}$ , then the conditioning  $\kappa_2(K_\gamma)$  increases linearly with  $\gamma$ , hence recovering a known result [34].
2. In early IPM iterations, the slacks are pushed away from the boundary and the number of active inequality constraints is  $m_a = 0$ . The ill-conditioning in  $K_\gamma$  is caused only by  $\gamma G^\top G$  and  $\underline{\sigma} = \bar{\sigma} = \gamma$ .
3. In late IPM iterations, the active slacks are converging to 0. We observe that if  $\frac{1}{\Xi} \leq \gamma \leq \frac{1}{s_{min}}$  the parameter  $\gamma$  does not increase the ill-conditioning of the condensed matrix  $K_\gamma$ .

#### 4.2.2 Numerical accuracy of the condensed matrix $K_\gamma$

In floating-point arithmetic, the condensed matrix  $K_\gamma$  is evaluated as

$$\begin{aligned} \widehat{K}_\gamma &= W + \Delta W + (A + \Delta A)^\top (D_\gamma + \Delta D_\gamma) (A + \Delta A) \\ &\quad + (H_N + \Delta H_N)^\top S_N^{-1} V_N (H_N + \Delta H_N), \end{aligned}$$

with  $\Delta W = O(\mathbf{u})$ ,  $\Delta H_N = O(\mathbf{u})$ ,  $\Delta A = \Theta(\mathbf{u})$ ,  $\Delta D_\gamma = O(\mathbf{u}\bar{\sigma})$ : most of the errors arise because of the ill-conditioned diagonal terms in  $D_\gamma$ .

**Proposition 2** *In floating-point arithmetic, the perturbation of the condensed matrix  $K_\gamma$  satisfies  $\Delta K_\gamma := \widehat{K}_\gamma - K_\gamma = O(\mathbf{u}\bar{\sigma})$ .*

*Proof* As  $A = \Theta(1)$ , we have:  $A^\top D_\gamma A = \Theta(\bar{\sigma})$  and  $A^\top \Delta D_\gamma A = O(\mathbf{u}\bar{\sigma})$ . Neglecting second-order terms, we get

$$\begin{aligned} \Delta K_\gamma &= \overbrace{\Delta W}^{O(\mathbf{u})} + \overbrace{\Delta A^\top D_\gamma A}^{O(\bar{\sigma}\mathbf{u})} + \overbrace{A^\top \Delta D_\gamma A}^{O(\bar{\sigma}\mathbf{u})} + \overbrace{A^\top \Delta D_\gamma A}^{O(\bar{\sigma}\mathbf{u})} \\ &\quad + \underbrace{\Delta H_N S_N^{-1} V_N H_N}_{O(\mathbf{u})} + \underbrace{H_N S_N^{-1} V_N \Delta H_N}_{O(\mathbf{u})}, \end{aligned}$$



where the terms in braces show the respective bounds on the errors. We deduce the error is dominated by the terms arising from the active Jacobian, all bounded by  $O(\bar{\sigma}\mathbf{u})$ , hence concluding the proof.

If it is large enough, the unstructured perturbation  $\Delta K_\gamma$  can impact the structured ill-conditioning in the perturbed matrix  $\widehat{K}_\gamma$ . We know that the smallest eigenvalue  $\eta_\ell$  of  $A^\top D_\gamma A$  is  $\Omega(\underline{\sigma})$ . As mentioned in [43, Section 3.4.2], the perturbed matrix  $\widehat{K}_\gamma$  keeps the  $p$  large eigenvalues bounded below by  $\underline{\sigma}$  if the perturbation is itself much smaller than the eigenvalue  $\eta_\ell$ :

$$\|\Delta K_\gamma\| \ll \eta_\ell = \Omega(\underline{\sigma}). \quad (29)$$

However, the bound given in Proposition 2 is too loose for (29) to hold without any further assumption (we have only  $\underline{\sigma} \leq \bar{\sigma}$ ). We note that for some constant  $C > 0$ ,  $\Delta K_\gamma \leq C\mathbf{u}\bar{\sigma}$ , implying  $\Delta K_\gamma/\underline{\sigma} \leq C\mathbf{u}\bar{\sigma}/\underline{\sigma}$ . Hence, if we suppose in addition the ratio  $\bar{\sigma}/\underline{\sigma}$  is close to 1, then  $\|\Delta K_\gamma\| = O(\mathbf{u}\bar{\sigma})$  can instead be replaced by  $\|\Delta K_\gamma\| = O(\mathbf{u}\underline{\sigma})$ , ensuring (29) holds.

#### 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  $K_\gamma$  is factorized using a backward-stable Cholesky decomposition. The computed solution  $\widehat{x}$  is solution of a perturbed system  $\widehat{K}_\gamma \widehat{x} = b$ , with  $\widehat{K}_\gamma = K_\gamma + \Delta_s K_\gamma$  and  $\Delta_s K_\gamma$  a symmetric matrix satisfying

$$\|\Delta_s K_\gamma\| \leq \mathbf{u}\varepsilon_n \|K_\gamma\|, \quad (30)$$

for  $\varepsilon_n$  a small constant depending on the dimension  $n$ . We need the following additional assumptions to ensure (a) the Cholesky factorization runs to completion and (b) we can incorporate the backward-stable perturbation  $\Delta_s K_\gamma$  in the generic perturbation  $\Delta K_\gamma$  introduced in Proposition 2.

**Assumption 3** *Let  $(p, v)$  be the current primal-dual iterate. We assume:*

- (a)  $(p, v)$  satisfies the centrality conditions (17).
- (b) The parameter  $\gamma$  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 (30) for a small constant  $\varepsilon_n$ .

Condition (a) implies that  $s_{\min} = \Theta(\Xi)$  and  $s_{\max} = \Theta(\Xi)$  (Proposition 1). Condition (b) supposes in addition  $\gamma = \Theta(\Xi^{-1})$ , making the matrix  $\Sigma_L$  well-conditioned with  $\underline{\sigma} = \Theta(\Xi^{-1})$ ,  $\bar{\sigma} = \Theta(\Xi^{-1})$  and  $\bar{\sigma}/\underline{\sigma} = \Theta(1)$ . Condition (c) ensures that  $\kappa_2(K_\gamma) = \Theta(\bar{\sigma})$  satisfies  $\kappa_2(K_\gamma)\mathbf{u} \ll 1$  (implying 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 (28) implies  $\|K_\gamma\| = \Theta(\Xi^{-1})$ , we can incorporate  $\Delta_s K_\gamma$  in the perturbation  $\Delta K_\gamma$  given in Proposition 2.

We are now ready to analyze the perturbation bound for the condensed system. We denote  $x$  the solution of the linear system  $K_\gamma x = b$  in exact arithmetic, and  $\hat{x}$  the solution of the perturbed system  $\hat{K}_\gamma \hat{x} = \hat{b}$  in floating-point arithmetic. We are interested in bounding the error  $\Delta x = \hat{x} - x$ . We recall that every vector  $x \in \mathbb{R}^n$  decomposes as

$$x = U_L x_L + U_S x_S = Y x_Y + Z x_Z . \quad (31)$$

*Impact of right-hand-side perturbation.* Using (25), the inverse of  $K_\gamma$  satisfies

$$K_\gamma^{-1} = [U_L \ U_S] \begin{bmatrix} \Sigma_L^{-1} & 0 \\ 0 & \Sigma_S^{-1} \end{bmatrix} \begin{bmatrix} U_L^\top \\ U_S^\top \end{bmatrix} . \quad (32)$$

Hence, if we solve the system for  $\hat{b} := b + \Delta b$ ,  $\Delta x = K_\gamma^{-1} \Delta b$  decomposes as

$$\begin{bmatrix} \Delta x_L \\ \Delta x_S \end{bmatrix} = \begin{bmatrix} \Sigma_L^{-1} & 0 \\ 0 & \Sigma_S^{-1} \end{bmatrix} \begin{bmatrix} \Delta b_L \\ \Delta b_S \end{bmatrix} , \quad (33)$$

which in turn implies the following bounds:

$$\|\Delta x_L\| \leq \|\Sigma_L^{-1}\| \|\Delta b_L\| , \quad \|\Delta x_S\| \leq \|\Sigma_S^{-1}\| \|\Delta b_S\| . \quad (34)$$

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

$$\|\Delta x\| \leq \|\Sigma_L^{-1}\| \|\Delta b_L\| + \|\Sigma_S^{-1}\| \|\Delta b_S\| = O(\|\Delta b\|) . \quad (35)$$

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

$$\begin{aligned} (K_\gamma + \Delta K_\gamma)^{-1} &= (I + K_\gamma^{-1} \Delta K_\gamma)^{-1} K_\gamma^{-1} , \\ &= K_\gamma^{-1} - K_\gamma^{-1} \Delta K_\gamma K_\gamma^{-1} + O(\|\Delta K_\gamma\|^2) . \end{aligned} \quad (36)$$

We decompose  $\Delta K_\gamma$  in two matrices  $\Gamma_L \in \mathbb{R}^{\ell \times n}$  and  $\Gamma_S \in \mathbb{R}^{(n-\ell) \times n}$  such that  $\Delta K_\gamma = \begin{bmatrix} \Gamma_L \\ \Gamma_S \end{bmatrix}$ . Using (32) the first-order error is given by

$$K_\gamma^{-1} \Delta K_\gamma K_\gamma^{-1} = U_L \Sigma_L^{-1} \Gamma_L \Sigma_L^{-1} U_L^\top + U_S \Sigma_S^{-1} \Gamma_S \Sigma_S^{-1} U_S^\top . \quad (37)$$

Using (28) and  $(\Gamma_L, \Gamma_S) = O(\Xi^{-1} \mathbf{u})$ , we obtain  $\Sigma_L^{-1} \Gamma_L \Sigma_L^{-1} = O(\Xi \mathbf{u})$  and  $\Sigma_S^{-1} \Gamma_S \Sigma_S^{-1} = O(\Xi^{-1} \mathbf{u})$ . 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 (34) and (37) to bound the error made when solving the condensed KKT system ( $K_1$ ) in floating-point arithmetic. In all this section, we assume that the primal-dual iterate  $(p, v)$  satisfies Assumption 3. Using [45, 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

$$\begin{cases} \bar{r}_1 = -\hat{r}_1 + \hat{H}^\top (\hat{D}_s \hat{r}_4 - \hat{r}_2), \\ \bar{r}_2 = -\hat{r}_3. \end{cases} \quad (38)$$

Using basic floating-point arithmetic, we get  $\hat{r}_1 = r_1 + O(\mathbf{u})$ ,  $\hat{r}_3 = r_3 + O(\mathbf{u})$ ,  $\hat{r}_4 = r_4 + O(\mathbf{u})$ . The error in the right-hand-side  $r_2$  is impacted by the term  $\mu S^{-1}e$ : under Assumption 3, it impacts differently the active and inactive components:  $\hat{r}_{2,\mathcal{B}} = r_{2,\mathcal{B}} + O(\mathbf{u})$  and  $\hat{r}_{2,\mathcal{N}} = r_{2,\mathcal{N}} + O(\Xi \mathbf{u})$ . Similarly, the diagonal matrix  $\hat{D}_s$  retains full accuracy only w.r.t. the inactive components:  $\hat{D}_{s,\mathcal{B}} = D_{s,\mathcal{B}} + O(\Xi^{-1} \mathbf{u})$  and  $\hat{D}_{s,\mathcal{N}} = D_{s,\mathcal{N}} + O(\Xi \mathbf{u})$ .

#### 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  $\gamma$ .

*Initial right-hand-side.* Let  $\hat{s}_\gamma := \bar{r}_1 + \gamma \hat{G}^\top \bar{r}_2$ . The initial right-hand side in (11) is evaluated as  $\hat{r}_\gamma := \hat{G} \hat{K}_\gamma^{-1} \hat{s}_\gamma - \bar{r}_2$ . The following proposition shows that despite an expression involving the inverse of the ill-conditioned condensed matrix  $K_\gamma$ , the error made in  $r_\gamma$  is bounded only by the machine precision  $\mathbf{u}$ .

**Proposition 3** *In floating point arithmetic, the error in the right-hand-side  $\Delta \hat{r}_\gamma$  satisfies:*

$$\Delta \hat{r}_\gamma = -\Delta \bar{r}_2 + \hat{G} \hat{K}_\gamma^{-1} \Delta s_\gamma = O(\mathbf{u}). \quad (39)$$

*Proof* Using (38), we have

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

The error decomposes as  $\Delta s_\gamma = Y \Delta s_Y + Z \Delta s_Z = U_L \Delta s_L + U_S \Delta s_S$ . We have  $\Delta s_Y = O(\Xi^{-1} \mathbf{u})$  and  $\Delta s_Z = O(\mathbf{u})$ . Using (27), we deduce  $\Delta s_L = U_L^\top \Delta s_\gamma =$

$O(\Xi^{-1}\mathbf{u})$  and  $\Delta s_S = U_S^\top \Delta s_\gamma = O(\mathbf{u})$ . Using (28) and (32), the error in the large space  $\Delta s_L$  annihilates in the backsolve:

$$K_\gamma^{-1} \Delta s_\gamma = U_L \Sigma_L^{-1} \Delta s_L + U_S \Sigma_S^{-1} \Delta s_S = O(\mathbf{u}). \quad (40)$$

Finally, using (36), we get

$$\widehat{G} \widehat{K}_\gamma^{-1} \Delta s_\gamma \approx \widehat{G}(I - K_\gamma^{-1} \Delta K_\gamma) K_\gamma^{-1} \Delta s_\gamma. \quad (41)$$

Using (40), the first term is  $\widehat{G} K_\gamma^{-1} \Delta s_\gamma = O(\mathbf{u})$ . We have in addition

$$G K_\gamma^{-1} \Delta K_\gamma (K_\gamma^{-1} \Delta s_\gamma) = [G U_L \Sigma_L^{-1} \Gamma_L + G U_S \Sigma_S^{-1} \Gamma_S] (K_\gamma^{-1} \Delta s_\gamma). \quad (42)$$

Using again (27):  $G U_L = G Y + O(\Xi)$  and  $G U_S = O(\Xi)$ . Hence  $G U_L \Sigma_L^{-1} \Gamma_L = O(1)$  and  $G U_S \Sigma_S^{-1} \Gamma_S = O(1)$ . Using (35), we have  $K_\gamma^{-1} \Delta G^\top = O(\mathbf{u})$ , implying  $\Delta G K_\gamma^{-1} \Delta K_\gamma (K_\gamma^{-1} \Delta s_\gamma) = O(\Xi^{-1} \mathbf{u}^2)$ . Assumption 3 implies that  $\Xi^{-1} \mathbf{u}^2 \ll \mathbf{u}$ , proving (39).

*Schur-complement operator.* The solution of the system (11) involves the Schur complement  $S_\gamma = G K_\gamma^{-1} G^\top$ . We show that the Schur complement has a specific structure that limits the loss of accuracy in the conjugate gradient algorithm.

**Proposition 4** *Suppose the current primal-dual iterate  $(p, v)$  satisfies Assumption 3. In exact arithmetic,*

$$S_\gamma = G Y \Sigma_L^{-1} Y^\top G^\top + O(\Xi^2). \quad (43)$$

*Proof* Using (32), we have

$$G K_\gamma^{-1} G^\top = G U_L \Sigma_L^{-1} U_L^\top G^\top + G U_S \Sigma_S^{-1} U_S^\top G^\top. \quad (44)$$

Using (27), we have  $G U_L = G Y + O(\Xi)$ , and  $G = O(1)$ , implying

$$G U_L \Sigma_L^{-1} U_L^\top G^\top = G Y \Sigma_L^{-1} Y^\top G^\top + O(\Xi^2). \quad (45)$$

Using again (27), we have  $G U_S = O(\Xi)$ . Hence,  $G U_S \Sigma_S^{-1} U_S^\top G^\top = O(\Xi^2)$ , concluding the proof.

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

**Proposition 5** *Suppose the current primal-dual iterate  $(p, v)$  satisfies Assumption 3. In floating-point arithmetic,*

$$\widehat{S}_\gamma = S_\gamma + O(\mathbf{u}). \quad (46)$$

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

$$\begin{aligned} \widehat{S}_\gamma &= \widehat{G} \widehat{K}_\gamma^{-1} \widehat{G}^\top, \\ &\approx (G + \Delta G)(K_\gamma^{-1} - K_\gamma^{-1} \Delta K_\gamma K_\gamma^{-1})(G + \Delta G)^\top, \\ &\approx S_\gamma - G(K_\gamma^{-1} \Delta K_\gamma K_\gamma^{-1})G^\top + K_\gamma^{-1} \Delta G^\top + \Delta G K_\gamma^{-1}. \end{aligned} \quad (47)$$

The second line is given by (36), the third by neglecting the second-order errors. Using (35), we get  $K_\gamma^{-1} \Delta G^\top = O(\mathbf{u})$  and  $\Delta G K_\gamma^{-1} = O(\mathbf{u})$ . Using (37), we have

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

Using (27), we have  $GU_S = O(\Xi)$ . 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

$$GU_L \Sigma_L^{-1} \Gamma_L \Sigma_L^{-1} U_L^\top G^\top = GY \Sigma_L^{-1} \Gamma_L \Sigma_L^{-1} Y^\top G^\top + O(\Xi^2\mathbf{u}). \quad (48)$$

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 (39) and (46) ensure that we can solve (11) using a conjugate gradient algorithm, as the errors remain limited in floating-point arithmetic.

#### 4.3.2 Solution with Lifted KKT system

The equality relaxation strategy used in LiftedKKT removes the equality constraints from the optimization problems, simplifying the solution of the condensed KKT system to (13). The active Jacobian  $A$  reduces to the active inequalities  $A = H_B$ , and we recover the original setting presented in [43]. Using the same arguments than in (4.3.1), the error in the right-hand-side is bounded by  $O(\Xi^{-1}\mathbf{u})$  and is in the range space of the active Jacobian  $A$ . Using (32), we can show that the absolute error on  $\widehat{d}_x$  is bounded by  $O(\Xi\mathbf{u})$ . 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 Richardson iterations.

#### 4.3.3 Summary

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

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

$$\widehat{d}_s = d_s + O(\varepsilon_K). \quad (49)$$

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:

$$\begin{aligned}\widehat{d}_{z,\mathcal{B}} &= -\widehat{r}_{2,\mathcal{B}} - \widehat{D}_{s,\mathcal{B}} \widehat{d}_{s,\mathcal{B}} = d_{z,\mathcal{B}} + O(\varepsilon_K \Xi^{-1}), \\ \widehat{d}_{z,\mathcal{N}} &= -\widehat{r}_{2,\mathcal{N}} - \widehat{D}_{s,\mathcal{N}} \widehat{d}_{s,\mathcal{N}} = d_{z,\mathcal{N}} + O(\varepsilon_K \Xi).\end{aligned}\tag{50}$$

Most of the error arises in the active components  $\widehat{d}_{z,\mathcal{B}}$ . To limit the loss of accuracy, the algorithm has to decrease the absolute precision  $\varepsilon_K$  as we are approaching to a local optimum. The impact remains limited if we have only few active inequality constraints.

## 5 Numerical results

We have implemented LiftedKKT and HyKKT on the GPU. First, we detail in §5.1 our implementation and assess in §5.2 the performance on the GPU of the two hybrid solvers. Then, we present in §5.3 the results reported on the PGLIB OPF benchmark, complemented in §5.4 by the COPS benchmark.

### 5.1 Implementation

All our implementation uses the Julia language [5]. We have used our local workstation to generate the results on the CPU, here equipped with an AMD EPYC 7443 processor (3.2GHz). For the results on the GPU, we have used an NVIDIA A100 GPU (with CUDA 12.3) on the Polaris testbed at Argonne National Laboratory <sup>1</sup>.

*IPM solver.* We have implemented the two condensed-space methods in our nonlinear IPM solver MadNLP [36]. This implementation utilizes the abstraction `AbstractKKTSystem` in MadNLP to represent the various formulations of the KKT linear systems. MadNLP can push most of the IPM algorithm to the GPU, except for basic IPM operations used for the coordination (e.g., the filter line-search algorithm). In particular, any operation that involves the manipulation of array entries is performed by GPU kernels without transferring data to host memory. We refer to [37] for a detailed description of the GPU implementation in MadNLP.

*Evaluation of the nonlinear models.* We use the ExaModels.jl modeling tool [37] to implement all the nonlinear programs utilized in our benchmark. ExaModels.jl harnesses the sparsity structure and provides custom derivative kernels for repetitive algebraic subexpressions of the constraints and objective functions to compute first and second-order derivatives on the GPU in parallel [6, 30]. This approach caters to the SIMD architecture of the GPU by assigning each expression to multiple threads responsible for computing derivatives for different values.

<sup>1</sup> <https://www.alcf.anl.gov/polaris>

*Linear solvers.* We solve the KKT systems assembled within MadNLP using various sparse linear solvers, chosen based on the KKT formulation ( $K_1$ ,  $K_2$ ,  $K_3$ ) and the device (CPU, GPU) being utilized. We utilize the following solvers:

- HSL MA27/MA57: Implement the LBL<sup>T</sup> factorization on the CPU [15]. It solves the augmented KKT system ( $K_2$ ). This solver serves as the reference when running on the CPU.
- CHOLMOD: Implements the Cholesky and LDL<sup>T</sup> factorizations on the CPU (using the AMD ordering [1] by default). It factorizes the condensed matrices  $K_\gamma$  and  $K_\tau$  appearing resp. in (10) and in (13). This solver is used to assess the performance of the hybrid solvers when running on the CPU.
- cuDSS: Implement Cholesky, LDL<sup>T</sup> and LU decompositions on an NVIDIA GPU. We use the LDL<sup>T</sup> factorization to factorize the ill-conditioned condensed matrices  $K_\gamma$  on GPU, as LDL<sup>T</sup> is more robust than the Cholesky factorization.
- Krylov.jl: Contains the CG method used in the Golub & Greif strategy to solve (11) on both CPU and GPU architectures.

CHOLMOD [9] is shipped with Julia. For the HSL linear solvers, we utilize libHSL [16] with the Julia interface HSL.jl [28]. HSL MA57 and CHOLMOD are both compiled with OpenBLAS, a multithreaded version of BLAS and LAPACK. The Julia package Krylov.jl [27] contains a collection of Krylov methods with a polymorphic implementation that can be used on both CPU and GPU architectures.

## 5.2 Performance analysis on a large-scale instance

We evaluate the performance of each KKT solver on a large-scale OPF instance, taken from the PGLIB benchmark [3]: `78484epigrids`. Our formulation with ExaModels has a total of 674,562 variables, 661,017 equality constraints and 378,045 inequality constraints. Our previous work has pointed out that as soon as the OPF model is evaluated on the GPU using ExaModels, the KKT solver becomes the bottleneck in the numerical implementation [37].

### 5.2.1 Individual performance of the linear solvers

Subsequently, we evaluate the individual performance of the cuDSS solver when factorizing the matrix  $K_\gamma$  at the first IPM iteration (here with  $\gamma = 10^7$ ). We compare the times to perform the symbolic analysis, the factorization and the triangular solves with those reported in CHOLMOD.

The results are displayed in Table 1. We benchmark the three decompositions implemented in cuDSS (Cholesky, LDL<sup>T</sup>, LU), and assess the absolute accuracy of the solution by computing the infinity norm of the residual  $\|K_\gamma x - b\|_\infty$ . We observe that the analysis phase is four times slower for cuDSS compared to CHOLMOD. Fortunately, this operation needs to be computed

only once in IPM, meaning that its cost is amortized if we run many IPM iterations. The factorization is about twenty times faster in cuDSS, with a time almost independent of the algorithm being used. The backward and forward sweeps are ten times faster: the triangular solves are harder to parallelize on a GPU. In terms of accuracy, the quality of the solution remains on par with CHOLMOD, except for the LDL<sup>T</sup> decomposition which lags behind by at least two orders of magnitude.

linear solver	analysis (s)	factorization (s)	backsolve (s)	abs. accuracy
CHOLMOD	1.18	$8.57 \times 10^{-1}$	$1.27 \times 10^{-1}$	$3.60 \times 10^{-13}$
cuDSS-Cholesky	4.52	$3.75 \times 10^{-2}$	$1.32 \times 10^{-2}$	$2.64 \times 10^{-13}$
cuDSS-LU	4.50	$3.72 \times 10^{-2}$	$1.49 \times 10^{-2}$	$2.58 \times 10^{-13}$
cuDSS-LDL <sup>T</sup>	4.50	$4.07 \times 10^{-2}$	$1.55 \times 10^{-2}$	$7.62 \times 10^{-11}$

**Table 1** Comparing the performance of cuDSS with CHOLMOD. The matrix  $K_\gamma$  is symmetric positive definite, with a size  $n = 674,562$ . The matrix is extremely sparse, with only 7,342,680 non-zero entries (0.002%).

### 5.2.2 Tuning the Golub & Greif strategy

In Figure 1 we depict the evolution of the number of CG iterations and relative accuracy as we increase the parameter  $\gamma$  from  $10^4$  to  $10^8$  in HyKKT.

On the algorithmic side, we observe that the higher the regularization  $\gamma$ , the faster 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 additional primal-dual regularizations.

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). Overall cuDSS is faster than CHOLMOD, leading to a 4x-8x speed-up in the total IPM solution time. We note also that the assembly of the condensed matrix  $K_\gamma$  parallelizes well on the GPU, with a reduction in the assembly time from  $\approx 8s$  on the CPU to  $\approx 0.2s$  on the GPU.

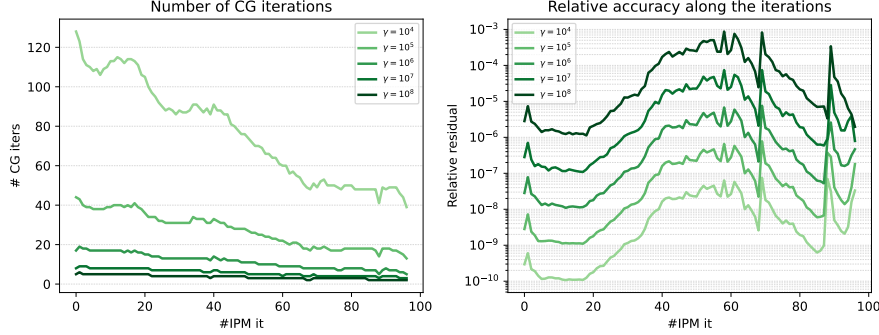
### 5.2.3 Tuning the equality relaxation strategy

We now analyze the numerical performance of LiftedKKT (§3.2). The method solves the KKT system (13) using a direct solver. The parameter  $\tau$  used in the equality relaxation (12) is set equal to the IPM tolerance  $\varepsilon_{tol}$  (in practice, it does not make sense to set a parameter  $\tau$  below IPM tolerance as the inequality constraints are satisfied only up to a tolerance  $\pm\varepsilon_{tol}$  in IPM).

We compare in Table 5.2.3 the performance obtained by LiftedKKT as we decrease the IPM tolerance  $\varepsilon_{tol}$ . We display both the runtimes on the CPU (using CHOLMOD-LDL<sup>T</sup>) and on the GPU (using cuDSS-LDL<sup>T</sup>). The slacks associated with the relaxed equality constraints are converging to a value below



$\gamma$	CHOLMOD-LDL <sup>T</sup> (CPU)					cuDSS-LDL <sup>T</sup> (CUDA)				
	# it	cond. (s)	CG (s)	linsol (s)	IPM (s)	# it	cond. (s)	CG (s)	linsol (s)	IPM (s)
$10^4$	96	8.15	463.86	536.83	<b>575.06</b>	96	0.17	113.27	114.52	<b>124.00</b>
$10^5$	96	8.33	163.35	235.61	<b>273.36</b>	96	0.17	53.37	54.62	<b>64.39</b>
$10^6$	96	8.22	68.69	139.86	<b>177.24</b>	96	0.17	14.53	15.78	<b>25.39</b>
$10^7$	96	8.24	35.12	107.17	<b>144.78</b>	96	0.17	7.95	9.20	<b>18.41</b>
$10^8$	96	7.89	21.68	93.85	<b>131.33</b>	96	0.17	5.36	6.62	<b>15.90</b>



**Fig. 1** Above: Decomposition of IPM solution time across (a) condensation time (cond.), (b) CG time, (c) total time spent in the linear solver (linsol.) and (d) total time spent in IPM solver (IPM). Below: Impact of  $\gamma$  on the total number of CG iterations and the norm of the relative residual at each IPM iteration. The peak observed in the norm of the relative residual corresponds to the primal-dual regularization performed inside the IPM algorithm, applied when the matrix  $K_\gamma$  is not positive definite.

$2\tau$ , leading to highly ill-conditioned terms in the diagonal matrices  $D_s$ . As a consequence, the conditioning of the matrix  $K_\tau$  in (13) can increase above  $10^{18}$ , leading to a nearly singular linear system. We observe cuDSS-LDL<sup>T</sup> is more stable than CHOLMOD: the factorization succeeds, and the loss of accuracy caused by the ill-conditioning is tamed by the multiple Richardson iterations that reduces the relative accuracy in the residual down to an acceptable level. As a result, cuDSS can solve the problem to optimality in  $\approx 20s$ , a time comparable with HyKKT (see Figure 1).

$\varepsilon_{tol}$	CHOLMOD-LDL <sup>T</sup> (CPU)		cuDSS-LDL <sup>T</sup> (CUDA)		accuracy
	#it	time (s)	#it	time (s)	
$10^{-4}$	115	268.2	114	19.9	$1.2 \times 10^{-2}$
$10^{-5}$	210	777.8	113	30.4	$1.2 \times 10^{-3}$
$10^{-6}$	102	337.5	109	25.0	$1.2 \times 10^{-4}$
$10^{-7}$	108	352.9	104	20.1	$1.2 \times 10^{-5}$
$10^{-8}$	-	-	105	20.3	$1.2 \times 10^{-6}$

**Table 2** Performance of the equality-relaxation strategy as we decrease the IPM tolerance  $\varepsilon_{tol}$ . The table displays the wall time on the CPU (using CHOLMOD-LDL<sup>T</sup>) and on the GPU (using cuDSS-LDL<sup>T</sup>).

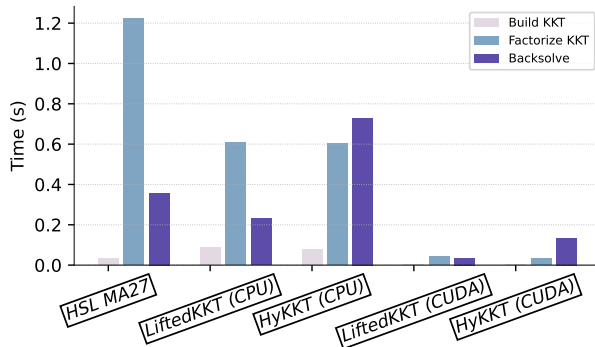
#### 5.2.4 Breakdown of the time spent in one IPM iteration

We decompose the time spent in a single IPM iteration for LiftedKKT and HyKKT. As a reference running on the CPU, we show the time spent in the

solver HSL MA27. We observe that HSL MA57 is slower than HSL MA27, as the OPF instances are super-sparse. Hence, the block elimination algorithm implemented in HSL MA57 is not beneficial there <sup>2</sup>.

When solving the KKT system, the time can be decomposed into: (1) assembling the KKT system, (2) factorizing the KKT system, and (3) computing the descent direction with triangular solves. As depicted in Figure 2, we observe that constructing the KKT system represents only a fraction of the computation time, compared to the factorization and the triangular solves. Using `cuDSS-LDLT`, we observe speedups of 30x and 15x in the factorization compared to MA27 and CHOLMOD running on the CPU. Once the KKT system is factorized, computing the descent direction with `LiftedKKT` is faster than with `HyKKT` (0.04s compared to 0.13s) as `HyKKT` has to run a CG algorithm to solve the Schur complement system (11), leading to additional backsolves in the linear solver.

	build (s)	factorize (s)	backsolve (s)	accuracy
HSL MA27	$3.15 \times 10^{-2}$	$1.22 \times 10^{-0}$	$3.58 \times 10^{-1}$	$5.52 \times 10^{-7}$
LiftedKKT (CPU)	$8.71 \times 10^{-2}$	$6.08 \times 10^{-1}$	$2.32 \times 10^{-1}$	$3.73 \times 10^{-9}$
HyKKT (CPU)	$7.97 \times 10^{-2}$	$6.02 \times 10^{-1}$	$7.30 \times 10^{-1}$	$3.38 \times 10^{-3}$
LiftedKKT (CUDA)	$2.09 \times 10^{-3}$	$4.37 \times 10^{-2}$	$3.53 \times 10^{-2}$	$4.86 \times 10^{-9}$
HyKKT (CUDA)	$1.86 \times 10^{-3}$	$3.38 \times 10^{-2}$	$1.35 \times 10^{-1}$	$3.91 \times 10^{-3}$



**Fig. 2** Breakdown of the time spent in one IPM iteration for different linear solvers, when solving `78484epigrids`

### 5.3 Benchmark on OPF instances

We run a benchmark on difficult OPF instances taken from the PGLIB benchmark [3]. We compare `LiftedKKT` and `HyKKT` with HSL MA27. The results are displayed in Table 3, for an IPM tolerance set to  $10^{-6}$ . Regarding `HyKKT`, we set  $\gamma = 10^7$  following the analysis in §5.2.2. The table displays the time spent in the initialization, the time spent in the linear solver and the total

<sup>2</sup> Personal communication with Iain Duff.

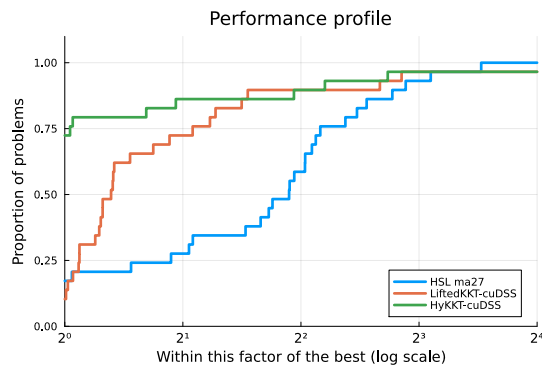
solving time. We complement the table with a Dolan & Moré performance profile [12] displayed in Figure 3. Overall, the performance of HSL MA27 on the CPU is consistent with what was reported in [3].

On the GPU, LiftedKKT+cuDSS is faster than HyKKT+cuDSS on small and medium instances: indeed, the algorithm does not have to run a CG algorithm at each IPM iteration, limiting the number of triangular solves performed at each iteration. Both LiftedKKT+cuDSS and HyKKT+cuDSS are significantly faster than HSL MA27. HyKKT+cuDSS is slower when solving `8387_pegase`, as on this particular instance the parameter  $\gamma$  is not set high enough to reduce the total number of CG iterations, leading to a 4x slowdown compared to LiftedKKT+cuDSS. Nevertheless, the performance of HyKKT+cuDSS is better on the largest instances, with almost an 8x speed-up compared to the reference HSL MA27.

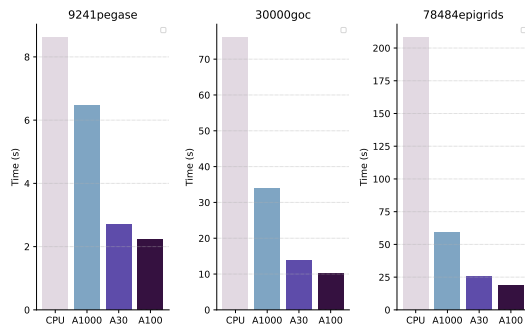
The benchmark presented in Table 3 has been generated using a NVIDIA A100 GPUs (current selling price: \$10k). We have also compared the performance with cheaper GPUs: a NVIDIA A1000 (a laptop-based GPU, 4GB memory) and a NVIDIA A30 (24GB memory, price: \$5k). As a comparison, the selling price of the AMD EPYC 7443 processor we used for the benchmark on the CPU is \$1.2k. The results are displayed in Figure 4. We observe that the performance of the A30 and the A100 are relatively similar. The cheaper A1000 GPU is already faster than HSL MA27 running on the CPU, but is not able to solve the largest instance as it is running out of memory.

Case	HSL MA27				LiftedKKT+cuDSS				HyKKT+cuDSS			
	it	init	lin	total	it	init	lin	total	it	init	lin	total
89_pegase	32	0.00	0.02	<b>0.03</b>	29	0.03	0.12	<b>0.24</b>	32	0.03	0.07	<b>0.22</b>
179_goc	45	0.00	0.03	<b>0.05</b>	39	0.03	0.19	<b>0.35</b>	45	0.03	0.07	<b>0.25</b>
500_goc	39	0.01	0.10	<b>0.14</b>	39	0.05	0.09	<b>0.26</b>	39	0.05	0.07	<b>0.27</b>
793_goc	35	0.01	0.12	<b>0.18</b>	57	0.06	0.27	<b>0.52</b>	35	0.05	0.10	<b>0.30</b>
1354_pegase	49	0.02	0.35	<b>0.52</b>	96	0.12	0.69	<b>1.22</b>	49	0.12	0.17	<b>0.50</b>
2000_goc	42	0.03	0.66	<b>0.93</b>	46	0.15	0.30	<b>0.66</b>	42	0.16	0.14	<b>0.50</b>
2312_goc	43	0.02	0.59	<b>0.82</b>	45	0.14	0.32	<b>0.68</b>	43	0.14	0.21	<b>0.56</b>
2742_goc	125	0.04	3.76	<b>7.31</b>	157	0.20	1.93	<b>15.49</b>	-	-	-	-
2869_pegase	55	0.04	1.09	<b>1.52</b>	57	0.20	0.30	<b>0.80</b>	55	0.21	0.26	<b>0.73</b>
3022_goc	55	0.03	0.98	<b>1.39</b>	48	0.18	0.23	<b>0.66</b>	55	0.18	0.23	<b>0.68</b>
3970_goc	48	0.05	1.95	<b>2.53</b>	47	0.26	0.37	<b>0.87</b>	48	0.27	0.24	<b>0.80</b>
4020_goc	59	0.06	3.90	<b>4.60</b>	123	0.28	1.75	<b>3.15</b>	59	0.29	0.41	<b>1.08</b>
4601_goc	71	0.09	3.09	<b>4.16</b>	67	0.27	0.51	<b>1.17</b>	71	0.28	0.39	<b>1.12</b>
4619_goc	49	0.07	3.21	<b>3.91</b>	49	0.34	0.59	<b>1.25</b>	49	0.33	0.31	<b>0.95</b>
4837_goc	59	0.08	2.49	<b>3.33</b>	59	0.29	0.58	<b>1.31</b>	59	0.29	0.35	<b>0.98</b>
4917_goc	63	0.07	1.97	<b>2.72</b>	55	0.26	0.55	<b>1.18</b>	63	0.26	0.34	<b>0.94</b>
5658_epigrids	51	0.31	2.80	<b>3.86</b>	58	0.35	0.66	<b>1.51</b>	51	0.35	0.35	<b>1.03</b>
7336_epigrids	50	0.13	3.60	<b>4.91</b>	56	0.45	0.95	<b>1.89</b>	50	0.43	0.35	<b>1.13</b>
8387_pegase	74	0.14	5.31	<b>7.62</b>	82	0.59	0.79	<b>2.30</b>	75	0.58	7.66	<b>8.84</b>
9241_pegase	74	0.15	6.11	<b>8.60</b>	101	0.63	0.88	<b>2.76</b>	71	0.63	0.99	<b>2.24</b>
9591_goc	67	0.20	11.14	<b>13.37</b>	98	0.63	2.67	<b>4.58</b>	67	0.62	0.74	<b>1.96</b>
10000_goc	82	0.15	6.00	<b>8.16</b>	64	0.49	0.81	<b>1.83</b>	82	0.49	0.75	<b>1.82</b>
10192_epigrids	54	0.41	7.79	<b>10.08</b>	57	0.67	1.14	<b>2.40</b>	54	0.67	0.66	<b>1.81</b>
10480_goc	71	0.24	12.04	<b>14.74</b>	67	0.75	0.99	<b>2.72</b>	71	0.74	1.09	<b>2.50</b>
13659_pegase	63	0.45	7.21	<b>10.14</b>	75	0.83	1.05	<b>2.96</b>	62	0.84	0.93	<b>2.47</b>
19402_goc	69	0.63	31.71	<b>36.92</b>	73	1.42	2.28	<b>5.38</b>	69	1.44	1.93	<b>4.31</b>
20758_epigrids	51	0.63	14.27	<b>18.21</b>	53	1.34	1.05	<b>3.57</b>	51	1.35	1.55	<b>3.51</b>
30000_goc	183	0.65	63.02	<b>75.95</b>	-	-	-	-	225	1.22	5.59	<b>10.27</b>
78484_epigrids	102	2.57	179.29	<b>207.79</b>	101	5.94	5.62	<b>18.03</b>	104	6.29	9.01	<b>18.90</b>

**Table 3** OPF benchmark, solved with a tolerance `tol=1e-6`.



**Fig. 3** Performance profile for the PGLIB OPF benchmark, solved with a tolerance  $\text{tol}=1\text{e-}6$ .

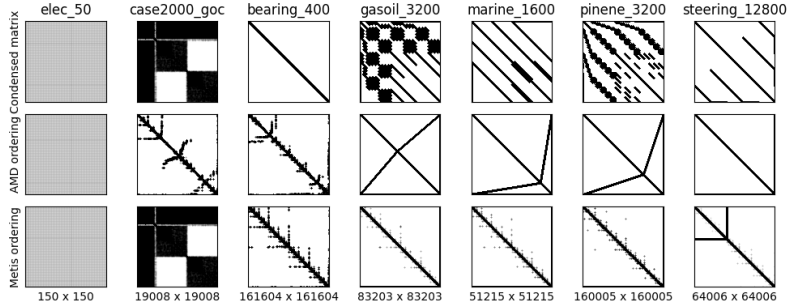


**Fig. 4** Comparing the performance obtained with various GPUs on three different OPF instances. We have used HyKKT to generate the results.

#### 5.4 Benchmark on COPS instances

We have observed in the previous section that both LiftedKKT and HyKKT outperform HSL MA27 when running on the GPU. However, the OPF instances are specific nonlinear instances. For that reason, we complement our analysis by looking at the performance of LiftedKKT and HyKKT on large-scale COPS instances [13]. We look at the performance we get on the COPS instances used in the Mittelmann benchmark [25]. To illustrate the heterogeneity of the COPS instances, we display in Figure 5 the sparsity pattern of the condensed matrices  $K_\gamma$  (10) 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**, **steering**) are highly sparse and can be reordered efficiently using AMD ordering [1].

The results of the COPS benchmark are displayed in Table 4. HSL MA57 gives better results than HSL MA27 for the COPS benchmark, and for that reason we have decided to replace HSL MA27 by HSL MA57. As expected, the results are different than on the OPF benchmark. We observe that LiftedKKT+cuDSS and HyKKT+cuDSS outperform HSL MA57 on the dense instance `elec` (20x speed-up) and `bearing` — an instance whose sparsity pattern is similar to the OPF. In the other instances, LiftedKKT+cuDSS and HyKKT+cuDSS on par with HSL MA57 and sometimes even slightly slower (`rocket` and `pinene`).



**Fig. 5** Sparsity patterns for one OPF instance and for various COPS problems. The first row displays the sparsity pattern of  $K_\gamma$ , after AMD reordering. The second row displays the sparsity pattern of  $K_\gamma$  after Metis reordering.

			HSL MA57				LiftedKKT+cuDSS				HyKKT+cuDSS			
	$n$	$m$	it	init	lin	total	it	init	lin	total	it	init	lin	total
bearing_400	162k	2k	17	0.14	3.42	<b>4.10</b>	14	0.85	0.07	<b>1.13</b>	14	0.78	0.76	<b>1.74</b>
camshape_6400	6k	19k	38	0.02	0.18	<b>0.29</b>	35	0.05	0.03	<b>0.19</b>	38	0.05	0.04	<b>0.23</b>
elec_400	1k	0.4k	185	0.54	24.64	<b>33.02</b>	273	0.46	0.97	<b>20.01</b>	128	0.48	0.75	<b>4.16</b>
gasoil_3200	83k	83k	37	0.36	4.81	<b>5.81</b>	21	0.54	0.24	<b>1.40</b>	20	0.59	0.21	<b>1.35</b>
marine_1600	51k	51k	13	0.05	0.41	<b>0.50</b>	33	0.38	0.58	<b>1.29</b>	13	0.37	0.12	<b>0.62</b>
pinene_3200	160k	160k	12	0.11	1.32	<b>1.60</b>	21	0.87	0.16	<b>1.52</b>	11	0.90	0.84	<b>2.02</b>
robot_1600	14k	10k	34	0.04	0.33	<b>0.45</b>	35	0.20	0.07	<b>0.76</b>	34	0.21	0.08	<b>0.80</b>
rocket_12800	51k	38k	23	0.12	1.73	<b>2.16</b>	37	0.74	0.06	<b>2.49</b>	24	0.25	1.70	<b>3.12</b>
steering_12800	64k	51k	19	0.25	1.49	<b>1.93</b>	18	0.44	0.06	<b>1.64</b>	18	0.46	0.07	<b>1.83</b>
bearing_800	643k	3k	13	0.94	14.59	<b>16.86</b>	14	3.31	0.18	<b>4.10</b>	12	3.32	1.98	<b>5.86</b>
camshape_12800	13k	38k	34	0.02	0.34	<b>0.54</b>	33	0.05	0.02	<b>0.16</b>	34	0.06	0.03	<b>0.19</b>
elec_800	2k	0.8k	354	2.36	337.41	<b>409.57</b>	298	2.11	2.58	<b>24.38</b>	184	1.81	2.40	<b>16.33</b>
gasoil_12800	333k	333k	20	1.78	11.15	<b>13.65</b>	18	2.11	0.98	<b>5.50</b>	22	2.99	1.21	<b>6.47</b>
marine_12800	410k	410k	11	0.36	3.51	<b>4.46</b>	146	2.80	25.04	<b>39.24</b>	11	2.89	0.63	<b>4.03</b>
pinene_12800	640k	640k	10	0.48	7.15	<b>8.45</b>	21	4.50	0.99	<b>7.44</b>	11	4.65	3.54	<b>9.25</b>
robot_12800	115k	77k	35	0.54	4.63	<b>5.91</b>	33	1.13	0.30	<b>4.29</b>	35	1.15	0.27	<b>4.58</b>
rocket_51200	205k	154k	31	1.21	6.24	<b>9.51</b>	37	0.83	0.17	<b>8.49</b>	30	0.87	2.67	<b>10.11</b>
steering_51200	256k	205k	27	1.40	9.74	<b>13.00</b>	15	1.82	0.19	<b>5.41</b>	28	1.88	0.56	<b>11.31</b>

**Table 4** COPS benchmark, solved with a tolerance `tol=1e-6`

## 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 is worth noting that the sparse Cholesky solver can be further customized to meet the specific requirements of the interior-point method [44]. Enhancing the two methods on the GPU would enable the resolution of large-scale problems that are currently intractable on classical CPU architectures such as multiperiod and security-constrained OPF problems.

## 7 Acknowledgements

This research used resources of the Argonne Leadership Computing Facility, a U.S. Department of Energy (DOE) Office of Science user facility at Argonne National Laboratory and is based on research supported by the U.S. DOE Office of Science-Advanced Scientific Computing Research Program, under Contract No. DE-AC02-06CH11357.

## References

1. Amestoy, P.R., Davis, T.A., Duff, I.S.: Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software (TOMS)* **30**(3), 381–388 (2004)
2. Amos, B., Kolter, J.Z.: Optnet: Differentiable optimization as a layer in neural networks. In: *International Conference on Machine Learning*, pp. 136–145. PMLR (2017)
3. Babaeinejadsarookolaee, S., Birchfield, A., Christie, R.D., Coffrin, C., DeMarco, C., Diao, R., Ferris, M., Fliscounakis, S., Greene, S., Huang, R., et al.: The power grid library for benchmarking AC optimal power flow algorithms. *arXiv preprint arXiv:1908.02788* (2019)
4. Benzi, M., Golub, G.H., Liesen, J.: Numerical solution of saddle point problems. *Acta numerica* **14**, 1–137 (2005)
5. Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: A fresh approach to numerical computing. *SIAM Review* **59**(1), 65–98 (2017). DOI 10.1137/141000671
6. Bischof, C., Griewank, A., Juedes, D.: Exploiting parallelism in automatic differentiation. In: *Proceedings of the 5th international conference on Supercomputing*, pp. 146–153 (1991)
7. Bradbury, J., Frostig, R., Hawkins, P., Johnson, M.J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., Zhang, Q.: JAX: composable transformations of Python+NumPy programs (2018). URL <http://github.com/google/jax>

8. Cao, Y., Seth, A., Laird, C.D.: An augmented lagrangian interior-point approach for large-scale NLP problems on graphics processing units. *Computers & Chemical Engineering* **85**, 76–83 (2016)
9. Chen, Y., Davis, T.A., Hager, W.W., Rajamanickam, S.: Algorithm 887: CHOLMOD, supernodal sparse cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software (TOMS)* **35**(3), 1–14 (2008)
10. Curtis, F.E., Huber, J., Schenk, O., Wächter, A.: A note on the implementation of an interior-point algorithm for nonlinear optimization with inexact step computations. *Mathematical Programming* **136**(1), 209–227 (2012). DOI 10.1007/s10107-012-0557-4
11. Debreu, G.: Definite and semidefinite quadratic forms. *Econometrica: Journal of the Econometric Society* pp. 295–300 (1952)
12. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. *Mathematical programming* **91**, 201–213 (2002)
13. Dolan, E.D., Moré, J.J., Munson, T.S.: Benchmarking optimization software with COPS 3.0. Tech. rep., Argonne National Lab., Argonne, IL (US) (2004)
14. Duff, I.S.: MA57—a code for the solution of sparse symmetric definite and indefinite systems. *ACM Transactions on Mathematical Software (TOMS)* **30**(2), 118–144 (2004)
15. Duff, I.S., Reid, J.K.: The multifrontal solution of indefinite sparse symmetric linear. *ACM Transactions on Mathematical Software (TOMS)* **9**(3), 302–325 (1983)
16. Fowkes, J., Lister, A., Montoison, A., Orban, D.: LibHSL: the ultimate collection for large-scale scientific computation. *Les Cahiers du GERAD G-2024-06, Groupe d’études et de recherche en analyse des décisions* (2024)
17. Ghannad, A., Orban, D., Saunders, M.A.: Linear systems arising in interior methods for convex optimization: a symmetric formulation with bounded condition number. *Optimization Methods and Software* **37**(4), 1344–1369 (2022)
18. Golub, G.H., Greif, C.: On solving block-structured indefinite linear systems. *SIAM Journal on Scientific Computing* **24**(6), 2076–2092 (2003)
19. Gondzio, J.: Interior point methods 25 years later. *European Journal of Operational Research* **218**(3), 587–601 (2012). DOI 10.1016/j.ejor.2011.09.017
20. Hestenes, M.R., Stiefel, E.: Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards* **49**(6), 409–436 (1952). DOI 10.6028/jres.049.044
21. Kim, Y., Pacaud, F., Kim, K., Anitescu, M.: Leveraging GPU batching for scalable nonlinear programming through massive Lagrangian decomposition (2021). DOI 10.48550/arXiv.2106.14995
22. Lu, H., Yang, J.: cuPDLP.jl: A GPU implementation of restarted primal-dual hybrid gradient for linear programming in julia. *arXiv preprint arXiv:2311.12180* (2023)
23. Lu, H., Yang, J., Hu, H., Huangfu, Q., Liu, J., Liu, T., Ye, Y., Zhang, C., Ge, D.: cuPDLP-C: A strengthened implementation of cuPDLP for linear programming by C language. *arXiv preprint arXiv:2312.14832* (2023)
24. Markidis, S., Der Chien, S.W., Laure, E., Peng, I.B., Vetter, J.S.: Nvidia tensor core programmability, performance & precision. In: 2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW), pp. 522–531. IEEE (2018)
25. Mittelmann, H.D.: Benchmark for optimization software (2002). URL <http://plato.asu.edu/bench.html>
26. Montoison, A.: CUDSS.jl: Julia interface for NVIDIA cuDSS. URL <https://github.com/exanauts/CUDSS.jl>
27. Montoison, A., Orban, D.: Krylov.jl: A julia basket of hand-picked Krylov methods. *Journal of Open Source Software* **8**(89), 5187 (2023)
28. Montoison, A., Orban, D., contributors: HSL.jl: A Julia interface to the HSL mathematical software library. <https://github.com/JuliaSmoothOptimizers/HSL.jl> (2021). DOI 10.5281/zenodo.2658672
29. Montoison, A., Orban, D., Saunders, M.A.: MinAres: An iterative solver for symmetric linear systems. *Les Cahiers du GERAD G-2023-40, Groupe d’études et de recherche en analyse des décisions* (2023)
30. Moses, W.S., Churavy, V., Paehler, L., Hückelheim, J., Narayanan, S.H.K., Schanen, M., Doerfert, J.: Reverse-mode automatic differentiation and optimization of GPU kernels via Enzyme. In: *Proceedings of the International Conference for High Perfor-*

- mance Computing, Networking, Storage and Analysis, SC '21. Association for Computing Machinery, New York, NY, USA (2021). DOI 10.1145/3458817.3476165. URL <https://doi.org/10.1145/3458817.3476165>
31. Nocedal, J., Wright, S.J.: Numerical optimization, 2nd edn. Springer series in operations research. Springer, New York (2006)
  32. Pacaud, F., Shin, S., Schanen, M., Maldonado, D.A., Anitescu, M.: Accelerating condensed interior-point methods on SIMD/GPU architectures. *Journal of Optimization Theory and Applications* pp. 1–20 (2023)
  33. Pineda, L., Fan, T., Monge, M., Venkataraman, S., Sodhi, P., Chen, R.T., Ortiz, J., DeTone, D., Wang, A., Anderson, S., et al.: Theseus: A library for differentiable nonlinear optimization. *Advances in Neural Information Processing Systems* **35**, 3801–3818 (2022)
  34. Regev, S., Chiang, N.Y., Darve, E., Petra, C.G., Saunders, M.A., Świrydowicz, K., Peleš, S.: HyKKT: a hybrid direct-iterative method for solving KKT linear systems. *Optimization Methods and Software* **38**(2), 332–355 (2023)
  35. Rodriguez, J.S., Laird, C.D., Zavala, V.M.: Scalable preconditioning of block-structured linear algebra systems using ADMM. *Computers & Chemical Engineering* **133**, 106478 (2020). DOI 10.1016/j.compchemeng.2019.06.003
  36. Shin, S., Coffrin, C., Sundar, K., Zavala, V.M.: Graph-based modeling and decomposition of energy infrastructures. *IFAC-PapersOnLine* **54**(3), 693–698 (2021)
  37. Shin, S., Pacaud, F., Anitescu, M.: Accelerating optimal power flow with GPUs: SIMD abstraction of nonlinear programs and condensed-space interior-point methods. arXiv preprint arXiv:2307.16830 (2023)
  38. Stewart, G.W., Sun, J.g.: Matrix perturbation theory. (No Title) (1990)
  39. Świrydowicz, K., Darve, E., Jones, W., Maack, J., Regev, S., Saunders, M.A., Thomas, S.J., Peleš, S.: Linear solvers for power grid optimization problems: a review of GPU-accelerated linear solvers. *Parallel Computing* p. 102870 (2021)
  40. Tasseff, B., Coffrin, C., Wächter, A., Laird, C.: Exploring benefits of linear solver parallelism on modern nonlinear optimization applications. arXiv preprint arXiv:1909.08104 (2019)
  41. Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming* **106**(1), 25–57 (2006)
  42. Waltz, R.A., Morales, J.L., Nocedal, J., Orban, D.: An interior algorithm for nonlinear optimization that combines line search and trust region steps. *Mathematical Programming* **107**(3), 391–408 (2006)
  43. Wright, M.H.: Ill-conditioning and computational error in interior methods for nonlinear programming. *SIAM Journal on Optimization* **9**(1), 84–111 (1998)
  44. Wright, S.J.: Modified cholesky factorizations in interior-point algorithms for linear programming. *SIAM Journal on Optimization* **9**(4), 1159–1191 (1999)
  45. Wright, S.J.: Effects of finite-precision arithmetic on interior-point methods for nonlinear programming. *SIAM Journal on Optimization* **12**(1), 36–78 (2001)