A User Manual for cuHALLaR: A GPU Accelerated Low-Rank Semidefinite Programming Solver [*]

Jacob M. Aguirre[†] Diego Cifuentes[‡] Vincent Guigues[§] Renato D.C. Monteiro[¶]
Victor Hugo Nascimento[‖] Arnesh Sujanani[**]

August 21, 2025

Abstract

We present a Julia-based interface to the precompiled HALLaR and cuHALLaR binaries for large-scale semidefinite programs (SDPs). Both solvers are established as fast and numerically stable, and accept problem data in formats compatible with SDPA and a new enhanced data format taking advantage of Hybrid Sparse Low-Rank (HSLR) structure. The interface allows users to load custom data files, configure solver options, and execute experiments directly from Julia. A collection of example problems is included, including the SDP relaxations of the Matrix Completion and Maximum Stable Set problems.

Keywords: semidefinite programming, augmented Lagrangian, low-rank methods, GPU acceleration, Frank–Wolfe method.

1 Introduction

This document serves as a user guide for HALLaR [6] and cuHALLaR [1]. Their binaries can be downloaded from `https://github.com/OPTHALLaR`.

Let $\mathbb{S}^n$ be the set of $n \times n$ symmetric matrices. The notation $A \succeq B$ means that $A - B$ is positive semidefinite. cuHALLaR and HALLaR solve the primal-dual pair of semidefinite programs (SDPs)

$$P_* := \min_X \quad \{C \bullet X \quad : \quad \mathcal{A}(X) = b, \quad \mathrm{Tr}(X) \leqslant \tau, \quad X \succeq 0\} \tag{P}$$

and

$$D_* := \max_{(p,S,\theta)} \quad \{-b^\top p - \tau\theta \quad : \quad C + \mathcal{A}^*(p) + \theta I - S = 0, \quad S \succeq 0, \quad \theta \geqslant 0\} \tag{D}$$

where $b \in \mathbb{R}^m$, $C \in \mathbb{S}^n$, and $\mathcal{A} : \mathbb{S}^n \to \mathbb{R}^m$ and $\mathcal{A}^* : \mathbb{R}^m \to \mathbb{S}^n$ are linear maps such that

$$\mathcal{A}(X) = \begin{bmatrix} A_1 \bullet X \\ A_2 \bullet X \\ \vdots \\ A_m \bullet X \end{bmatrix}, \quad \mathcal{A}^*(p) = \sum_{i=1}^m p_i A_i \tag{1}$$

where $A_\ell \in \mathbb{S}^n$ for $\ell = 1, \dots m$. We refer to τ as the **trace bound**. We also define Δ^n to be the spectraplex, i.e.,

$$\Delta^n := \{X \in \mathbb{S}^n : \mathrm{Tr}(X) \leqslant \tau, X \succeq 0\}. \tag{2}$$

[*] **Funding**: Jacob M. Aguirre is supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-2039655. Diego Cifuentes is supported by U.S. Office of Naval Research, N00014-23-1-2631. Renato D.C. Monteiro is supported by AFOSR Grant FA9550-25-1-0131.

[†] H. M. Stewart School of Industrial and Systems Engineering, Georgia Tech, Atlanta, GA, 30332-0205. aguirre@gatech.edu

[‡] H. M. Stewart School of Industrial and Systems Engineering, Georgia Tech, Atlanta, GA, 30332-0205. dfc3@gatech.edu

[§] School of Applied Mathematics, FGV, Praia de Botafogo, Rio de Janeiro, Brazil. vincent.guigues@fgv.br

[¶] School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, 30332-0205. rm88@gatech.edu

[‖] School of Applied Mathematics, FGV, Praia de Botafogo, Rio de Janeiro, Brazil. vhn@fgv.br

[**] Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON, N2L 3G1. a3sujana@uwaterloo.ca

Authors are listed in alphabetical order.

Both solvers are based on an augmented Lagrangian framework with hybrid low-rank updates, incorporating the Frank–Wolfe method and an adaptive accelerated inexact proximal-point method (ADAP-AIPP) which uses ideas from [2, 4, 5, 7, 8]. cuHALLaR is a GPU-accelerated variant intended for high-throughput computation on modern CUDA-capable devices, while HALLaR is a CPU-based implementation suitable for systems without GPU support. The solvers are numerically stable, memory-efficient, and support SDPA and Hybrid Sparse Low Rank (HSLR) input formats (discussed more in Section 3).

The guide that follows explains how to prepare problem data in the required HSLR format, how to configure solver parameters either via command-line options or configuration files, and how to interpret the output files produced.

2 Installation & Running

Requirements. cuHALLaR is distributed as a precompiled binary. No Julia installation is required. A CUDA–enabled NVIDIA GPU, the matching NVIDIA driver, and a compatible CUDA runtime must be available on the system. All Julia dependencies (`CUDA.jl`, `LinearAlgebra`, `SparseArrays`, `Parameters.jl`, `KrylovKit.jl`) are bundled. HALLaR is the CPU version of our code and requires no CUDA libraries to run. We tested cuHALLaR with CUDA 12.9 on GPUs with compute capability $\geqslant 5.0$.

Obtain the binary. Download the precompiled executable from `https://github.com/OPTHALLaR`, place it in a working directory, and ensure that the NVIDIA driver and CUDA runtime libraries are discoverable by the dynamic loader.

Quick check. Verify the installation by running the built–in tests:

```
$ ./cuHallar --run_tests
```

This executes several small example instances shipped with the binary and reports if everything was installed correctly and the code successfully terminated.

Directory layout. It is not necessary to keep HALLaR and cuHALLaR in separate folders. Each folder contains its own executable, default configuration file, and output structure. Invoke the solver from its folder so that the relative paths for the input, output, and configuration files are correctly resolved.

Required inputs.

- **config and output files**

  The config file is a text file that contains the parameter options of cuHALLaR. The options are organized in the `key = value` format. For examples of parameter options that can be specified by the user in the text file, the user should refer to Table 1 in Subsection 3.2.

  The output file is a text file, similar to a CSV file, where a dual solution $p_* \in \mathbb{R}^m$ and a low-rank factor $Y_* \in \mathbb{R}^{n \times r}$ of the primal solution X_* are saved. We make no assumptions about the naming or the extensions of either the configurations or output file. If extensions such as `-c`, `-p`, or `-d` are omitted, compiled defaults are used. However, the correct files must be provided immediately after each option flag. For instance, if the user passes the wrong file type to the `-c` option flag, it will get the error listed below.

  ```
  ERROR: ArgParse.ArgParseError("unrecognized option --3 4")
  Stacktrace:
   [1] argparse_error(x::Any)
     @ ArgParse ~/.julia/packages/ArgParse/mpp98/src/parsing.jl:9
   [2] parse_long_opt!(state::ArgParse.ParserState, settings::ArgParse.ArgParseSettings)
     @ ArgParse ~/.julia/packages/ArgParse/mpp98/src/parsing.jl:842
   [3] parse_args_unhandled(args_list::Vector, settings::ArgParse.ArgParseSettings,
      truncated_shopts::Bool)
  ```

  Here, `3 4` is not a parameter option for the config file.

- **model files: HSLR or SDPA format**

  - **HSLR (Hybrid Sparse Low-Rank) format**

    When solving large-scale SDP problem instances with dense cost and constraint matrices, the model size often becomes an issue in terms of storage and RAM. For instance, describing such large SDP instances in standard SDPA format often requires several GBs of storage/RAM. As a result, we introduce the new HSLR format which allows the users to input cost and constraint matrices, C and A_ℓ, $\ell = 1, \ldots m$, that are sums of sparse and low-rank matrices. For the sparse component of a matrix, the user only needs to specify the values and indices of the nonzero entries of the upper triangular part of that component. For the low-rank component, the user only needs to input the factors that make-up the low-rank factorization. The sparse and low-rank components are stored internally as CSC (sparse) and dense column–major (low–rank) arrays, respectively.

    HSLR format also requires the user to input dimension pair (m, n), the right-hand side vector b, and a trace bound τ for the trace constraint in (P). For more details and concrete examples that display how to input these quantities and the cost and constraint matrices correctly for HSLR format, see Subsection 3.1.

  - **Sparse SDPA format**

    Standard sparse SDPA `.dat-s` files developed in [3] are also accepted by our cuHALLaR. Each constraint matrix A_ℓ that the user provides for SDPA format is imported as a sparse matrix so its low–rank factor is assumed to be zero.

    As the standard SDPA format lacks support for inputting a trace bound τ for the trace constraint in (P), users who wish to use this format **must** specify the field `--trace_bound` in either the command line or in the config file. cuHALLaR will not run if a user does not provide τ as an input and the interface will also provide a warning.

Optional input

- **initial point**

  An initial primal iterate $Y_0 \in \mathbb{R}^{n \times r}$ can be supplied by the user as a dense CSV file with no header:

  ```
  $ ./cuHallar -i model.hslr -w <path_to_Y0_file>
  ```

  The initial primal iterate Y_0 that the user provides should satisfy $\|Y_0\|_F^2 \leqslant \tau$ since this ensures that $\mathrm{Tr}(Y_0 Y_0^\top) \leqslant \tau$. If the user does not specify an initial point Y_0, then the code internally generates its own Y_0 which satisfies $\|Y_0\|_F^2 \leqslant \tau$. The user does not need to provide an initial dual iterate p_0 since the code always internally sets p_0 to be the vector of all zeros.

Overriding parameters inline. Any parameter option of cuHALLaR may be also supplied on the command line instead of the config file. For example, the following line may be used in the command line

```
$ ./cuHallar -i model.hslr --L_inc_fista 3.1 --eps_gap 1e-6 -o out.csv
```

If a parameter value was specified both on the command line and the config file, the value supplied on the command line takes precedence over the value specified on the config file.

Basic run.

- If the user has prepared data in HSLR format and created the required config and output files, the user should then execute the command

  ```
      ./cuHallar -i <path_to_HSLR_file> [-c <path_to_config_file>] [-o <path_to_output_file>]
  ```

  to call cuHALLaR to solve the SDP instance.

- If the user has prepared data in sparse SDPA format, created the required config and output files, and computed an appropriate trace bound τ, the user should then execute the command

  ```
  ./cuHallar -i <path_to_SDPA_file> --trace_bound τ [-c <path_to_config_file>] [-o <path_to_output_file>]
  ```

  to call cuHALLaR to solve the SDP instance.

Runtime reporting. During each of cuHALLaR's outer augmented Lagrangian iterations, the current violation in KKT residuals, the current penalty parameter, and the current objective value are all printed for the user to see.

3 Interface

This section describes the interface and inputs needed to run cuHALLaR using either HSLR or SDPA format.

3.1 Data Input

HALLaR and cuHALLaR accept problem data in either HSLR or SDPA format; SDPA inputs are converted in memory to the hybrid layout. HSLR allows each matrix A_ℓ to be written as the sum of a sparse matrix and a low rank matrix, i.e. (under the convention that $A_0 = C$):

$$A_\ell = A_\ell^{\rm sp} \; + \; A_\ell^{\rm lr}, \qquad A_\ell^{\rm lr} = P_\ell D_\ell P_\ell^\top, \qquad \ell = 0, 1, \ldots, m, \tag{3}$$

where $A^{\rm sp} \in \mathbb{S}^n$ is the sparse component of A_ℓ, and $P_\ell \in \mathbb{R}^{n \times r_\ell}$ and $D_\ell \in \mathbb{S}^{r_\ell}$ are the factors of the low rank component of A_ℓ. Only the upper triangular part of $A_\ell^{\rm sp}$ is stored by specifying triplets (i, j, val) with $1 \leqslant i \leqslant j \leqslant n$ corresponding to its nonzero entries; (duplicate triplets are not allowed). Each column of P_ℓ and the corresponding column of D_ℓ are provided in a single line of the input file, with the vector for the column of P_ℓ separated from the vector for the column of D_ℓ by a semicolon '; '. For example, in the maximum stable set formulation of Section A.2, the objective matrix $C = -ee^\top$ is dense but has rank one. Representing this matrix in SDPA format would require storing all $\mathcal{O}(n^2)$ nonzero entries, whereas the HSLR format requires only $\mathcal{O}(n)$ storage for the low-rank factor, enabling the solution of much larger instances than would otherwise be feasible. The first, second, and third lines of the HSLR file specifies the number of constraints m and matrix size n, the right–hand side vector $b \in \mathbb{R}^m$, and the trace bound $\tau > 0$, respectively. Next, each A_ℓ is specified by entering "ℓ `SP`" followed by its sparse component description, and/or by entering "ℓ `LR`" followed by its low rank description. Each A_ℓ can be entered in any order but, if A_ℓ has both sparse and low rank components, the sparse should precede the low rank one.

The following example is used to illustrate how the input data should be provided. We first note that all numerical data for matrix entries and vector components are parsed as floating-point values, accepting integer, decimal, and scientific notation allowing for instances like 1e5. In contrast, the problem dimensions m, n, and all sparse matrix indices must be provided as integers.

Consider the SDP problem given by

$$\min \, \langle I + ee^\top, X \rangle \quad \text{s.t.} \quad \langle 0.5\, I, X \rangle = 2, \;\; \langle ee^\top, X \rangle = 4, \;\; \langle A_3^{\rm sp} + A_3^{\rm lr}, X \rangle = 7, \;\; \mathrm{Tr}(X) \leqslant 5, \;\; X \succeq 0, \tag{4}$$

where I denotes the 4×4 identity matrix, $X \in \mathbb{S}^4$, $e := (1, 1, 1, 1)^\top \in \mathbb{R}^4$, and

$$A_3^{\rm sp} = \begin{bmatrix} 0 & 0 & 0.7 & 0 \\ 0 & 1 & 0 & -0.5 \\ 0.7 & 0 & 0 & 0 \\ 0 & -0.5 & 0 & -1 \end{bmatrix}, \qquad P_3 = \begin{bmatrix} 1.0 & 2.0 \\ 2.0 & 1.0 \\ 1.0 & 1.0 \\ 2.0 & 1.0 \end{bmatrix}, \qquad D_3 = \begin{bmatrix} 1.0 & -0.5 \\ -0.5 & -2.0 \end{bmatrix}.$$

Since the trace constraint is encoded by the line containing τ, the file records $m = 3$ equality constraints and $n = 4$, and uses $b = (2, 4, 7)^\top$.

The complete HSLR file[1] is:

Listing 1: HSLR file for Problem (4).

```
# m n
3 4
# b vector
2.0 4.0 7.0
# Trace bound
5.0

# Matrix 0: C = I + ee^T
0 SP
```

[1] Line spacing is added below for readability but is optional.

```
1 1 1.0
2 2 1.0
3 3 1.0
4 4 1.0
0 LR
1.0 1.0 1.0 1.0 ; 1.0

# Matrix 1: 0.5 * I
1 SP
1 1 0.5
2 2 0.5
3 3 0.5
4 4 0.5

# Matrix 2: ee^T
2 LR
1.0 1.0 1.0 1.0 ; 1.0

# Matrix 3: A3_sp + A3_lr
3 SP
1 3   0.7
2 2   1.0
2 4  -0.5
4 4  -1.0
3 LR
1.0 2.0 1.0 2.0 ;  1.0   -0.5
2.0 1.0 1.0 1.0 ; -0.5   -2.0
```

3.2 Parameter Input

Users supply options either on the command line or via a text configuration file. Precedence is strict:

$$\text{command line} \succ \text{configuration file} \succ \text{compiled defaults.}$$

When both are present, the value given on the command line overrides the value in the configuration file, and any value not specified by the user is taken from the solver's compiled defaults. Paths may be absolute or relative to the current working directory at invocation. Options and their defaults are listed in Table 1 and grouped as Basic, Intermediate, and Advanced to mirror the solver's control flow.

The command line accepts short flags for core I/O (`-i`, `-p`, `-d`, `-c`) and long GNU–style flags for all parameters (e.g., `--eps_gap 1e-6`). Boolean flags appear without a value when enabled (e.g., `--run_tests`). Numeric quantities use SI units: `time_limit` is in seconds; tolerances such as `eps_gap` and `eps_pfeas` are dimensionless; iteration limits are integers; penalty and Lipschitz parameters are real and must be positive. For SDPA inputs, a trace bound must be provided by the user via `--trace_bound` unless encoded in the hybrid HSLR file; for HSLR inputs, the trace bound is read from the file header.

A configuration file is plain text. Each nonempty line assigns a single option by either

`key = value` or `key value`,

with optional surrounding whitespace. Blank lines are ignored. Lines beginning with `#` are treated as comments and ignored. Keys must match the option names in Table 1. Values follow the same typing and units as on the command line.

Examples. The following two invocations are equivalent due to precedence:

```
# Using a configuration file and overriding one option on the command line
$ cuHallar -i model.hslr -c options.cfg --L_inc_fista 3.1

# A minimal options.cfg (values not listed fall back to defaults)
# I/O
input_path = model.hslr
output_path = out.csv
# Stopping criteria
eps_gap = 1e-5
eps_pfeas = 1e-5
time_limit = 3600
# Penalty schedule
beta0 = 10.0
beta_inc = 1.1
beta_min = 10.0
beta_max = 1e11
# ADAP-FISTA
maxiter_fista = 10000
L0_fista = 1.0
L_inc_fista = 2.0
mu_fista = 0.5
chi_fista = 1e-4
sigma_fista = 0.3
err_tol_fista = 1e-8
# AIPP and HLR
maxiter_aipp = 5
lam0_aipp = 0.1
maxiter_hlr = 10
maxiter_hallar = 10000
# Scaling and verbosity
scale_A = 1.0
scale_C = 1.0
verbosity = 1
```

In practice, place persistent choices in the configuration file and use the command line to override a small number of run–specific parameters (e.g., `--trace_bound` for SDPA input or a one–off change to `--L_inc_fista`). This keeps runs reproducible while preserving exact control through the precedence rule.

Basic Settings. These options control the fundamental behavior of the solver. The user must specify the input file path (`-i`), the primal output file path (`-p`), the dual output file path (`-d`), and an optional configuration file (`-c`). The main termination criteria are also basic settings: the maximum number of outer iterations (`maxiter_hallar`), the relative duality gap tolerance (`eps_gap`), the primal feasibility tolerance (`eps_pfeas`), and the maximum runtime (`time_limit`). In the case `time_limit` is reached, cuHALLaR returns the last known iteration information. Finally, the verbosity level (`verbosity`) controls the amount of information printed to the console. In the case of verbosity equal zero, no output is given to allow for users to implement the models within their own subroutines.

Intermediate Settings. This group of parameters allows for fine-tuning the solver's scaling and penalty updates. The `scale_A` and `scale_C` options apply uniform scaling to the constraint and cost matrices, respectively, which can improve numerical stability. More details can be found in Subsection 3.3. The penalty parameter β is controlled by its initial value, `beta0`, its increment factor `beta_inc`, and its lower and upper bounds `beta_min` and `beta_max`. This category also includes controls for the main inner loop, such as the maximum number of iterations for the Hybrid Low-Rank subroutine which consists of an ADAP-AIPP call + potential Frank-Wolfe steps (`maxiter_hlr`) and the initial λ_0 parameter for the AIPP subroutine (`lam0_aipp`).

Advanced Settings. These parameters are intended for expert users who wish to control the low-level behavior of the innermost subroutines. This group includes all parameters for the ADAP-FISTA method, such as its iteration limit (`maxiter_fista`), step-size parameters (`mu_fista`, `chi_fista`, `sigma_fista`), and Lipschitz constant controls (`L0_fista`, `L_inc_fista`). It also contains the tolerances for FISTA (`err_tol_fista`) and the eigenvalue solvers (`eps_eig`, `err_tol_eig`), as well as the AIPP iteration limit (`maxiter_aipp`). Adjusting these values can impact the trade-off between solution speed and accuracy, but they are generally left at their default values.

Option	Default Value	Description
Input / Output		
`-i`	`none (required)`	Path to the input file in HSLR format; SDPA `.dat-s` also accepted.
`-p`	`"primal_out.txt"`	Path for the output file containing the primal solution.
`-d`	`"dual_out.txt"`	Path for the output file containing the dual solution.
`-c`	`""`	Path to a configuration file to load options.
`--initial_solution`, `-w`	`""`	Path to CSV with dense $Y_0 \in \mathbb{R}^{n \times r}$ (no header) for primal warm start; if empty, a default Y_0 is used.
`--run_tests`	`false (flag)`	Run test routine with example instances.
FISTA Parameters		
`--maxiter_fista`	`1e4`	Maximum number of ADAP-FISTA iterations.
`--mu_fista`	`0.5`	FISTA parameter μ.
`--chi_fista`	`1e-4`	FISTA parameter χ.
`--L0_fista`	`1.0`	Initial Lipschitz constant for ADAP-FISTA.
`--L_inc_fista`	`2.0`	Lipschitz constant increment factor.
`--sigma_fista`	`0.3`	FISTA parameter σ.
`--err_tol_fista`	`1e-8`	Error tolerance for ADAP-FISTA.
AIPP Parameters		
`--maxiter_aipp`	`5`	Maximum number of AIPP iterations.
`--lam0_aipp`	`0.1`	AIPP initial parameter λ_0.
Hybrid Low-Rank & Hallar		
`--maxiter_hlr`	`10`	Maximum iterations for the hybrid low-rank method.
`--maxiter_hallar`	`1e4`	Maximum number of outer HALLaR iterations.
Stopping Criteria		
`--eps_pfeas`	`1e-5`	Primal feasibility tolerance (ϵ_{feas}).
`--eps_gap`	`1e-5`	Relative duality gap tolerance (ϵ_{gap}).
Penalty Parameters		
`--beta0`	`10.0`	Initial penalty parameter β_0.
`--beta_inc`	`1.1`	Increment factor for β.
`--beta_min`	`10.0`	Minimum value for β.
`--beta_max`	`1e11`	Maximum value for β.
Scaling		
`--scale_A` (τ_a)	`1.0`	Positive scaling factor for constraint matrices.
`--scale_C` (τ_c)	`1.0`	Positive scaling factor for the cost matrix.
Miscellaneous		
`--verbosity`	`1`	Verbosity level (0: silent, 1: summary steps, 2: detailed, 3: debug).
`--time_limit`	`3600.0`	Time limit in seconds.

Table 1: Parameter options grouped by category.

3.3 Scaling

This subsection explains the roles of the scaling parameters `scale_A` and `scale_C`, which are denoted by τ_a and τ_c in the discussion below. Recall that the original problem (P) has the trace constraint $\mathrm{Tr}(X) \leqslant \tau$ and that its optimal value is denoted by v_*. Let

$$\tilde{C} := \tau_c C, \qquad \tilde{\mathcal{A}} := \tau_a \mathcal{A}, \qquad \tilde{b} := \frac{\tau_a}{\tau} b\,, \tag{5}$$

and define the primal and dual scaled variables

$$\tilde{X} := \frac{1}{\tau} X, \qquad \tilde{p} := \frac{\tau_c}{\tau_a}\, p, \qquad \tilde{\theta} := \tau_c \theta, \qquad \tilde{S} := \tau_c S. \tag{6}$$

Then, (P) and (D) are equivalent to the following scaled primal-dual pair SDPs:

$$\begin{aligned} \tilde{v}_* &= \min\{\ \tilde{C} \bullet \tilde{X}\ :\ \tilde{\mathcal{A}}(\tilde{X}) = \tilde{b},\ \operatorname{Tr}(\tilde{X}) \leqslant 1,\ \tilde{X} \succeq 0\ \} \\ &= \max\{\ -\tilde{b}^\top \tilde{p} - \tilde{\theta}\ :\ \tilde{S} = \tilde{C} + \tilde{\mathcal{A}}^*(\tilde{p}) + \tilde{\theta} I \succeq 0,\ \tilde{\theta} \geqslant 0\ \}, \end{aligned} \tag{7}$$

and there holds

$$v_* = \frac{\tau}{\tau_c}\, \tilde{v}_*. \tag{8}$$

The augmented–Lagrangian subproblem associated with (P),

$$\min\Big\{\ C \bullet X + \langle p, \mathcal{A}X - b\rangle + \frac{\beta}{2}\, \|\mathcal{A}X - b\|^2\ :\ \operatorname{Tr}(X) \leqslant \tau,\ X \succeq 0\Big\}, \tag{9}$$

transforms, after multiplying the objective by τ_c/τ and using (5)–(6), into

$$\min\Big\{\ \tilde{C} \bullet \tilde{X} + \langle \tilde{p}, \tilde{\mathcal{A}}\tilde{X} - \tilde{b}\rangle + \frac{\tilde{\beta}}{2}\, \|\tilde{\mathcal{A}}\tilde{X} - \tilde{b}\|^2\ :\ \operatorname{Tr}(\tilde{X}) \leqslant 1,\ \tilde{X} \succeq 0\Big\}, \tag{10}$$

with penalty coupling

$$\tilde{\beta} = \frac{\tau \tau_c}{\tau_a^2}\, \beta. \tag{11}$$

Relations (5)–(11) make explicit how choices of (τ_c, τ_a) relocate magnitude across the objective, constraints, and trace bound.

4 Output and Interpretation

This section describes the output produced by cuHALLaR and HALLaR, including the criteria used for termination, the information reported in the solver logs (console output), and the structure of the solution file.

4.1 Termination Criteria

The solvers implement an augmented Lagrangian framework that seeks to satisfy the optimality conditions for the primal-dual pair (P) and (D). The algorithm terminates when the normalized residuals associated with these conditions fall below specified tolerances.

For user-defined tolerances ϵ_{feas} (`eps_pfeas`) and ϵ_{gap} (`eps_gap`), cuHALLaR stops successfully when an iterate $(X, p, \theta) \in \Delta^n \times \mathbb{R}^m \times \mathbb{R}_+$, satisfies the following two conditions:

1. **Primal Feasibility:** The relative error in the equality constraints must be sufficiently small.

$$\frac{\|\mathcal{A}(X) - b\|_2}{1 + \|b\|_1} \leqslant \epsilon_{\text{feas}}. \tag{12}$$

2. **Relative Duality Gap:** The difference between the primal objective value (pval $= C \bullet X$) and the dual objective value (dval $= -b^\top p - \tau\theta$) must be relatively small.

$$\frac{|\text{pval} - \text{dval}|}{1 + |\text{pval}| + |\text{dval}|} \leqslant \epsilon_{\text{gap}}. \tag{13}$$

Our returned solution further satisfies the following properties:

3. **Exact Dual Feasibility:** The dual slack matrix is constructed as $S = C + \mathcal{A}^*(p) + \theta I$. Hence, the dual feasibility condition $C + \mathcal{A}^*(p) + \theta I - S = 0$ is satisfied exactly.

4. **Exact Primal PSD:** The primal matrix is computed in factorized form $X = YY^\top$. Hence, X is always positive semidefinite.

5. **Exact Dual PSD:** The choice of dual variable $\theta = \max\{0, -\lambda_{\min}(C + \mathcal{A}^*(p))\}$ guarantees that $S = C + \mathcal{A}^*(p) + \theta I$ is always positive semidefinite.

If cuHALLaR reaches the iteration limit (`maxiter_hallar`) or the time limit (`time_limit`) before satisfying these criteria, it terminates and returns the best solution found so far.

4.2 Console Output (Solver Log)

During execution, cuHALLaR prints a log to the console detailing the progress of the algorithm. The level of detail is controlled by the `verbosity` parameter. At the default level, the log provides a header summarizing the parameter settings and the problem dimensions, followed by a table reporting the status of each outer Augmented Lagrangian (AL) iteration.

A sample console output is shown below.

Listing 2: Sample console output from cuHALLaR.

```
---------- Basic Settings ------------------
input_path = examples/mc_3.dat-s
output_path = out.txt
...
Reading SDPA file: examples/mc_3.dat-s
Problem dimensions:
  - Matrix size: 3000 x 3000
  - Number of constraints: 216172
  - Trace bound: 51601.0

Solving SDP problem with GPU acceleration...

##############################################################################
  #   rank        gap      feas     pval      dval      pnlty    steps
  0    1           -      2.9e-03   9.690e-06    NaN      1.0e+01  A
  1    1         NaN      2.9e-03   8.201e-06  2.500e-03  1.0e+01  A
  2    1         NaN      2.9e-03   6.903e-06  6.250e-03  1.0e+01  A
...
 41    3       9.5e-06   2.1e-08   8.357e-02  8.357e-02  6.0e+03  A
 42    3       8.8e-06   1.3e-08   8.357e-02  8.357e-02  6.6e+03
Final Results
Primal Obj              = 0.08356806847402057
Dual Obj                = 0.08356659006982121
PD Gap                  = 8.844561680506419e-6
Primal infeasibility      = 1.3353696237066644e-8

#ADAP FISTA Calls: 44
#ACG Iterations: 262
#FW Calls: 2
Primal val unscaled = 4312.195901327936
Run time = 2.718115 seconds
Writing output
Output written to primal_out.txt and dual_out.txt.
```

The columns in the iteration table are interpreted as follows:

- `#`: The outer AL iteration count.
- `rank`: The rank r of the current primal iterate $X = YY^\top$, where $Y \in \mathbb{R}^{n\times r}$.
- `gap`: The current relative duality gap, computed according to (13).
- `feas`: The current primal feasibility residual, computed according to (12).
- `pval`: The current primal objective value $C \bullet X$.
- `dval`: The current dual objective value $-b^\top p - \tau\theta$.
- `pnlty`: The current value of the penalty parameter β.
- `steps`: Indicates the type of inner-loop steps taken during the iteration. ‘A’ denotes a call to the ADAP-AIPP subroutine, and ‘F’ denotes a Frank–Wolfe step.

The solution `rank` typically increases after an 'F' step, reflecting the addition of a new rank-one component (atom) to the factorization. The penalty parameter `pnlty` is adjusted adaptively; it generally increases to enforce feasibility but may decrease if subproblem residuals permit.

Upon termination, the log reports the final objective values, gap, and feasibility, followed by statistics on the total number of calls to subroutines (AIPP, FISTA, FW), the unscaled primal objective value (if scaling was applied; see Subsection 3.3), and the total runtime.

4.3 Solution File Output

cuHALLaR saves the primal and dual solutions to separate files, whose paths are specified by the user. The final low-rank primal factor $Y \in \mathbb{R}^{n \times r}$ is saved to the path given by `--primal_output_path` (or `-p`). The file is formatted as a standard comma-separated value (CSV) text file without a header; each of its r columns corresponds to a column of Y. The full primal solution matrix is recovered as $X = YY^\top$.

The final dual variables (p, θ) are saved to a path specified by the flag `--dual_output_path`. The file is formatted as a single comma-separated value (CSV) line; the first field contains the scalar $\theta \geqslant 0$, and the subsequent m fields contain the components of the vector $p \in \mathbb{R}^m$. The dual slack matrix S is not saved, as it is uniquely determined by the relation $S = C + \mathcal{A}^*(p) + \theta I$ and is guaranteed by construction to be positive semidefinite.

Consider a problem with $n = 4$, $m = 3$, which terminates with a rank-2 solution. The output files would appear as follows.

Listing 3: Example primal output file for a rank-2 solution ($Y \in \mathbb{R}^{4\times 2}$).

```
# File specified by --primal_output_path out_Y.csv
0.8561,-0.0152
-0.0152,0.9998
-0.5163,0.0021
0.1005,-0.1009
```

Listing 4: Example dual output file for $m = 3$.

```
# File specified by --dual_output_path out_p.csv
0.5873,-0.5873,3.4121,-1.2345
```

All CSV fields are comma–separated with no embedded whitespace; numeric fields are written in floating–point format. Parsing is unambiguous: Y is read from the $n \times r$ data block in the primal file, and (θ, p) are read from the single comma-separated line in the dual file.

A Additional Examples of HSLR Format

This section displays how to construct HSLR format for several structured SDPs such as the SDP relaxations of the Matrix Completion and Maximum Stable Set problems. The main quantities that are needed for HSLR format are the dimension pair (m, n), the cost matrix C, the data matrices A_ℓ, $\ell = 1, \ldots m$, the right-hand side vector b, and the tracebound τ. By convention we consider C to be matrix 0, i.e., $C = A_0$. Recall from Subsection 3.1 that the matrices A_ℓ are assumed to have the structure

$$A_\ell = A_\ell^{\mathrm{sp}} + A_\ell^{\mathrm{lr}}, \qquad A_\ell^{\mathrm{lr}} = P_\ell D_\ell P_\ell^\top, \qquad \ell = 0, 1, \ldots, m, \tag{14}$$

where $A^{\mathrm{sp}} \in \mathbb{S}^n$ is the sparse component of A_ℓ, and $P_\ell \in \mathbb{R}^{n \times r_\ell}$ and $D_\ell \in \mathbb{S}^{r_\ell}$ are the factors of the low rank component of A_ℓ. Subsections A.1 and A.2 below display how the user should prepare HSLR format for the Matrix Completion and Maximum Stable Set SDP relaxations, respectively.

A.1 Matrix Completion

Given integers $n_2 \geqslant n_1 \geqslant 1$, the goal of the matrix completion problem is to recover a low-rank matrix $M \in \mathbb{R}^{n_1 \times n_2}$ by observing a subset of its entries $\{M_{ij} : (i,j) \in \Omega\}$. A standard convex relaxation replaces the rank function with the nuclear norm:

$$\min_{Y \in \mathbb{R}^{n_1 \times n_2}} \{ \; \|Y\|_* \; : \; Y_{ij} = M_{ij}, \; \forall (i,j) \in \Omega \; \}.$$

Using the semidefinite representation of the nuclear norm, this optimization problem is equivalent to the following SDP

$$\min_{X\in\mathbb{S}^{n_1+n_2}} \left\{ \tfrac{1}{2}\operatorname{Tr}(X) \ : \ X = \begin{pmatrix} W_1 & Y \\ Y^\top & W_2 \end{pmatrix} \succeq 0,\ Y_{ij} = M_{ij}\ \forall (i,j)\in\Omega \right\}. \tag{15}$$

cuHALLaR solves the formulation in (15).

In the above formulation, the size of the matrix variable X is $n = n_1 + n_2$ and the number of constraints is $m = |\Omega|$. The right hand side vector b is a vector in $\mathbb{R}^m$ and the ℓ-th component of b is just M_{ij} where (i,j) is the ℓ-th index pair in Ω. To compute a suitable tracebound τ, we generate $\hat{Y} \in \mathbb{R}^{n_1\times n_2}$ so that $\hat{Y}_{ij} = M_{ij}$ for indices $(i,j)\in\Omega$ and $\hat{Y}_{ij} = 0$ for all other indices. The tracebound τ is then computed to be $2\sqrt{n_1}\|\hat{Y}\|_F$.

All data matrices which encode the SDP in (15) are sparse. Clearly, $C = A_0 = 0.5I$. All constraint matrices A_ℓ, $l = 1,\ldots m$, have exactly 2 nonzero entries. To see this, consider the constraint $Y_{ij} = M_{ij}$ where $1 \leqslant i \leqslant n_1$ and $1 \leqslant j \leqslant n_2$. The A_ℓ matrix corresponding to this constraint then simply takes value 0.5 in its positions $(i, n_1 + j)$ and $(n_1 + j, i)$ and value 0 elsewhere since this enforces that $A_\ell \bullet X = Y_{ij}$. The following example illustrates how to prepare a HSLR data file for a small Matrix Completion problem where $n_1 = 2$ and $n_2 = 2$.

Example: Suppose that $n_1 = 2$, $n_2 = 2$, $\Omega = \{(1,1),(2,2)\}$, and $M_{11} = 5$ and $M_{22} = 3$. Then $m = 2$ and $n = 4$.

The cost matrix is $C = A_0 = 0.5I$. It is easy to see that the constraint matrices A_1 and A_2 and the right-side vector b are:

$$A_1 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad b = \begin{bmatrix} 5 \\ 3 \end{bmatrix}.$$

To compute the trace bound, let $\hat{Y} = \begin{bmatrix} 5 & 0 \\ 0 & 3 \end{bmatrix}$. The trace bound τ is then computed to be $\tau = 2\sqrt{2}\|\hat{Y}\|_F \approx 16.50$. The HSLR data file corresponding to this example is:

Listing 5: HSLR data file for a small Matrix Completion problem.

```
# m n
2 4
# b vector
5.0 3.0
# Trace bound
16.50

# Matrix 0: C = 0.5*I
0 SP
1 1 0.5
2 2 0.5
3 3 0.5
4 4 0.5

# Matrix 1: A1_sp
1 SP
1 3 0.5

# Matrix 2: A2_sp
2 SP
2 4 0.5
```

Several remarks about the above HSLR data file are now given. The entries "2 4" refer to the dimension pair (m,n) while the entries "5.0 3.0" refer to b_1 and b_2, respectively. The trace bound τ is 16.50. The line "Matrix 0: C = 0.5*I" refers to the fact that the cost matrix is just $0.5 * I$. The line "0 SP" just means that this matrix is the 0-th matrix and it is a sparse matrix. The entries "1 1 0.5", … "4 4 0.5" mean that $C_{11} = 0.5, \ldots C_{44} = 0.5$. The line "Matrix 1: A_sp" means that we are now writing the first constraint matrix A_1. The line "1 SP" means that this is the first constraint matrix and it is a sparse matrix. The line "1 3 0.5" means that $(A_1)_{13} = 0.5$. Note that this is the only entry that needs to be encoded for A_1 since it is the only nonzero entry of the upper triangular part of A_1. Finally, the line “Matrix 2: A2_sp” means that we are now writing the second constraint matrix A_2. The line “2 SP” means that this is the second constraint matrix and it is a sparse matrix. The line “2 4 0.5” means that $(A_2)_{24} = 0.5$.

A.2 Maximum Stable Set

For a given undirected graph $G = (V, E)$ with $|V| = n$ vertices and $|E|$ edges, the stability number $\alpha(G)$ is the maximum size of a stable set (a subset of vertices where no two vertices are adjacent). The Lovász ϑ-function provides an upper bound on $\alpha(G)$ and is defined via the following semidefinite program:

$$\vartheta(G) = \max_{X \in \mathbb{S}^n} \{ \ J \bullet X \ : \ \mathrm{Tr}(X) \leqslant 1, \ X_{ij} = 0 \ \forall \{i, j\} \in E, \ X \succeq 0 \ \}, \tag{16}$$

where $J = ee^\top$ is the $n \times n$ matrix of all ones, and $e \in \mathbb{R}^n$ is the vector of all ones.

In this formulation, the size of the matrix variable X is $n = |V|$ and the number of constraints is $m = |E|$. The trace bound τ is thus set to be 1. The right hand side vector $b \in \mathbb{R}^m$ is simply the vector of all zeros.

To adapt this formulation to the minimization format required by our HSLR format, we minimize the negative of the objective function, setting the cost matrix to $C = -J$. The cost matrix $C = A_0 = -J$ is dense but has rank one. It is represented efficiently using only its low-rank component:

$$A_0^{\mathrm{sp}} = 0, \quad A_0^{\mathrm{lr}} = P_0 D_0 P_0^\top, \quad \text{where } P_0 = e \in \mathbb{R}^{n\times 1} \text{ and } D_0 = [-1] \in \mathbb{S}^1.$$

In HSLR format, only the components D_0 and P_0 need to be specified. This representation is significantly more storage-efficient than storing the fully dense matrix $-J$.

All constraint matrices A_ℓ, $\ell = 1, \dots m$, are sparse and have exactly two nonzero entries. To see this, consider the constraint $X_{ij} = 0$ where $\{i, j\}$ is an edge. The A_ℓ matrix corresponding to this constraint is then just $A_\ell = 0.5(E_{ij} + E_{ji})$, where E_{ij} denotes the matrix with 1 in position (i, j) and zeros elsewhere. This construction enforces that $A_\ell \bullet X = X_{ij}$. Since HSLR format only requires the user to specify the nonzero components of the upper triangular part of a sparse matrix, the user only needs to specify one entry for each of the constraint matrices A_ℓ, $\ell = 1, \dots m$. The following example illustrates how to prepare a HSLR data file for a small Maximum Stable Set problem where $n = 4$ and $m = 4$.

Example: Consider the 4-cycle graph C_4, with $V = \{1, 2, 3, 4\}$ and $E = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{1, 4\}\}$. We have $|V| = 4$ and $|E| = 4$, so $n = 4$ and $m = 4$. The trace bound is $\tau = 1.0$. The cost matrix $C = A_0 = -J$. Its low-rank factorization uses:

$$P_0 = (1, 1, 1, 1)^\top, \quad D_0 = [-1].$$

The constraint matrices $A_1, \dots, A_4$ and the right-hand side vector b are:

$$A_1 = \begin{bmatrix} 0 & 0.5 & 0 & 0 \\ 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \tag{17}$$

$$A_3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 \end{bmatrix}, \quad A_4 = \begin{bmatrix} 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{18}$$

The HSLR data file corresponding to this example is:

Listing 6: HSLR data file for the Maximum Stable Set problem on C_4.

```
# m n
4 4
# b vector
0.0 0.0 0.0 0.0
# Trace bound
1.0

# Matrix 0: C = −J (Low Rank)
```

```
0 LR
1.0 1.0 1.0 1.0 ; −1.0

# Matrix 1: Edge (1,2)
1 SP
1 2 0.5

# Matrix 2: Edge (2,3)
2 SP
2 3 0.5

# Matrix 3: Edge (3,4)
3 SP
3 4 0.5

# Matrix 4: Edge (1,4)
4 SP
1 4 0.5
```

Several remarks about the above HSLR data file are now given. The first line “4 4” specifies the dimension pair (m, n). The second line specifies the vector b which in this case is just $b = (0, 0, 0, 0)^\top$. The third line sets the trace bound $\tau = 1.0$. The block starting with “0 LR” defines the cost matrix $A_0 = C$. The line “1.0 1.0 1.0 1.0 ; -1.0” defines the single column of P_0 (the vector e) and the corresponding entry in D_0 (the scalar -1), separated by “;”. The block starting with “1 SP” defines A_1. The line “1 2 0.5” specifies that $(A_1)_{12} = 0.5$. The block starting with “2 SP” defines A_2. The line “2 3 0.5” specifies that $(A_2)_{23} = 0.5$. The blocks starting with “3 SP” and “4 SP” define matrices A_3 and A_4, respectively, in a similar-like fashion.

Acknowledgments

This research was supported in part through research cyberinfrastructure resources and services provided by the Partnership for an Advanced Computing Environment (PACE) at Georgia Tech, Atlanta, Georgia, USA.

References

[1] Jacob M Aguirre, Diego Cifuentes, Vincent Guigues, Renato DC Monteiro, Victor Hugo Nascimento, and Arnesh Sujanani. cuhallar: A gpu accelerated low-rank augmented lagrangian method for large-scale semidefinite programming. *arXiv preprint arXiv:2505.13719*, 2025.

[2] Y. Carmon, J. C. Duchi, O. Hinder, and A. Sidford. Accelerated methods for nonconvex optimization. *SIAM J. Optim.*, 28(2):1751–1772, 2018.

[3] K Fujisawa, Yoshiaki Futakata, M Kojima, Satoshi Matsuyama, S Nakamura, K Nakata, and M Yamashita. Sdpa-m (semidefinite programming algorithm in matlab) user’s manual—version 6.2. 0. *Research Reports on Mathematical and Computing Sciences, Series B: Operation Res., Dep. Math. and Computing Sci., Tokyo Institute of Technol., Japan*, 10, 2000.

[4] W. Kong, J.G. Melo, and R.D.C. Monteiro. Complexity of a quadratic penalty accelerated inexact proximal point method for solving linearly constrained nonconvex composite programs. *SIAM J. Optim.*, 29(4):2566–2593, 2019.

[5] W. Kong, J.G. Melo, and R.D.C. Monteiro. An efficient adaptive accelerated inexact proximal point method for solving linearly constrained nonconvex composite problems. *Comput. Optim. Appl.*, 76(2):305–346, 2019.

[6] Renato DC Monteiro, Arnesh Sujanani, and Diego Cifuentes. A low-rank augmented lagrangian method for large-scale semidefinite programming based on a hybrid convex-nonconvex approach. *arXiv preprint arXiv:2401.12490*, 2024.

[7] C. Paquette, H. Lin, D. Drusvyatskiy, J. Mairal, and Z. Harchaoui. Catalyst for gradient-based nonconvex optimization. In *AISTATS 2018-21st International Conference on Artificial Intelligence and Statistics*, pages 1–10, 2018.

[8] A. Sujanani and R.D.C. Monteiro. An adaptive superfast inexact proximal augmented Lagrangian method for smooth nonconvex composite optimization problems. *J. Scientific Computing*, 97(2), 2023.