FMINSDP – a code for solving optimization problems with matrix inequality constraints

Carl-Johan Thore

August 15, 2014

This document is a theoretical and practical introduction to the Matlab code `fminsdp` designed to find local solutions to (small-scale) non-linear, non-convex optimization problems (NLPs) with both scalar constraints and matrix inequality constraints.

Notation. A matrix $\boldsymbol{A} \in \mathbb{R}^{m\times m}$ is said to be *positive semi-definite* if $\boldsymbol{y}^\mathsf{T}\boldsymbol{A}\boldsymbol{y} \geq 0$ for all $\boldsymbol{y} \in \mathbb{R}^m$. It is convenient to introduce the notation "$\boldsymbol{A} \succeq \boldsymbol{0}$" to indicate that $\boldsymbol{A}$ is positive semi-definite. A *matrix inequality* is here defined as an expression of the form

$$\boldsymbol{\mathcal{A}}(\boldsymbol{x}) \succeq \boldsymbol{0}, \tag{1}$$

where $\boldsymbol{\mathcal{A}}$ is a map from $\Omega \subset \mathbb{R}^n$ to the space of symmetric matrices of size $m \times m$ with real entries, $\mathbb{S}^m$.

1 The optimization problem

`fminsdp` attempts to find a local solution to non-linear, non-convex optimization problems of the form

$$\begin{aligned}
&\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimize}} \;\; f(\boldsymbol{x}) \\
&\text{subject to} \begin{cases}
\boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq} & \text{linear equality constraints} \\
\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} & \text{linear inequality constraints} \\
\boldsymbol{c}_{eq}(\boldsymbol{x}) = \boldsymbol{0} & \text{nonlinear equality constraints} \\
\boldsymbol{c}(\boldsymbol{x}) \leq \boldsymbol{0} & \text{nonlinear inequality constraints} \\
\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} & \text{box constraints} \\
\boldsymbol{\mathcal{A}}_i(\boldsymbol{x}) \succeq \boldsymbol{0}, \quad i = 1,...,q & \text{matrix inequality constraints,}
\end{cases}
\end{aligned} \tag{2}$$

where $\boldsymbol{A}_{eq}$, $\boldsymbol{A}$, $\boldsymbol{b}_{eq}$, $\boldsymbol{b}$, $\boldsymbol{l}$ and $\boldsymbol{u}$ are constant matrices and vectors, respectively. The functions f, $\boldsymbol{c}$, $\boldsymbol{c}_{eq}$ and $\boldsymbol{\mathcal{A}}_i : \Omega_i \to \mathbb{S}^{m_i}$, $i = 1,...,q$, can be non-linear and are (preferably) at least twice continuously differentiable. Users familiar with `fmincon` from the Optimization Toolbox in Matlab [9] should recognize the form of problem (2) — the novelty here is the addition of q matrix inequality constraints.

As it stands, problem (2) is not in a form that can be treated directly by currently available NLP-solvers. Therefore, `fminsdp` works with a reformulation of problem (2) described next.

2 Theoretical background

To handle constraints of the form (1), `fminsdp` reformulates them into sets of scalar equality constraints based on the fact that a matrix is positive semi-definite if and only if it admits a Cholesky decomposition. In other words,

Theorem 1. A symmetric matrix $\boldsymbol{A} \succeq \boldsymbol{0}$ if and only if

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^{\mathsf{T}}$$

for some lower triangular matrix $\boldsymbol{L}$.

By default in `fminsdp`, $\boldsymbol{L}$ is restricted to the space of lower triangular $m \times m$-matrices with non-negative diagonal elements, referred to as $\mathcal{L}^m$. Non-negativity of the diagonal elements is not required by the theory, but enforcement of this condition is often important for computational efficiency.

`fminsdp` makes use of the function $\mathsf{svec} : \mathbb{R}^{m\times m} \to \mathbb{R}^p$, where p denotes the number of non-zero elements in the Cholesky factor $\boldsymbol{L}$ of a given matrix; i.e., the number of elements in the sparsity pattern of $\boldsymbol{L}$, obtained by a *symbolic* Cholesky factorization of the given matrix. svec takes, columnwise, the elements of the input matrix corresponding to potential non-zeros of $\boldsymbol{L}$ and stacks them on top of each other to form a vector of length p. If the lower triangular part of $\boldsymbol{L} \in \mathcal{L}^m$ is full we have $p = m(m+1)/2$ and

$$\mathsf{svec}(\boldsymbol{A}) = (A_{11}, A_{21}, \ldots, A_{m1}, A_{22}, \ldots, A_{m2}, \ldots, A_{mm})^{\mathsf{T}};$$

i.e., the vector $\mathsf{svec}(\boldsymbol{A})$ contains the elements of the lower triangular part of $\boldsymbol{A}$.

Now, based on Theorem 1, problem (2) is reformulated into the following problem:

$$\begin{aligned}
&\underset{\boldsymbol{x}\in\mathbb{R}^n,\, \boldsymbol{\ell}_1\in\mathbb{R}^{p_1},\, \ldots,\, \boldsymbol{\ell}_q\in\mathbb{R}^{p_q}}{\text{minimize}} \quad f(\boldsymbol{x}) \\
&\text{subject to} \begin{cases}
\boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq} \\
\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\
\boldsymbol{c}_{eq}(\boldsymbol{x}) = \boldsymbol{0} \\
\boldsymbol{c}(\boldsymbol{x}) \leq \boldsymbol{0} \\
\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \\
\tilde{\boldsymbol{l}} \leq (\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_q) \leq \tilde{\boldsymbol{u}} \\
\mathsf{svec}\left(\boldsymbol{\mathcal{A}}_i(\boldsymbol{x}) - \boldsymbol{L}_i(\boldsymbol{\ell}_i)\boldsymbol{L}_i(\boldsymbol{\ell}_i)^{\mathsf{T}}\right) = \boldsymbol{0}, \qquad i = 1, ..., q \\
\mathsf{diag}\{\boldsymbol{L}_i(\boldsymbol{\ell}_i)\} \geq \boldsymbol{0}, \qquad i = 1, ..., q,
\end{cases}
\end{aligned} \tag{3}$$

where the variables $\boldsymbol{\ell}_i$ defining the non-zero elements of the Cholesky factors are referred to as *auxiliary variables*, $\tilde{\boldsymbol{l}}$ are $\tilde{\boldsymbol{u}}$ are constant vectors, and diag returns a vector containing the diagonal elements of a matrix. This problem is in a form amenable to direct treatment by an NLP-solver, and this is the problem to which `fminsdp` attempts to find a local solution.

Note. Any local minimum of problem (3) is also a local minimum for (2), and vice versa (assuming that bounds on the auxiliary variables do not prevent this). However, problem (3) may have additional stationary points (KKT-points) not present in (2). The author has not experienced any difficulties obviously attributed to this fact, but it cannot be ruled out that it might cause trouble on some problems.

A key motivation behind `fminsdp` is to abstract away the exact treatment of the matrix inequality constraints so that the user only ”sees” a problem of the form (2); that is, the user should not have to deal with the Cholesky factors and the auxiliary variables. Thus, when using `fminsdp` one only works with the primary variables $\boldsymbol{x}$, and user-supplied derivatives are only with respect to $\boldsymbol{x}$. The examples in Section 5 and the folder `examples` shows how this is done in practice.

2.1 Problem sizes

It was mentioned, parenthetically, in the first line of this document that `fminsdp` seeks local solutions to *small-scale* problems. If the constraint matrices are of small sizes or/and very sparse, then `fminsdp` might in fact be able to solve large-scale problems within reasonable time. In general however, one can expect a fairly large number of auxiliary variables and thus many non-zero elements in the constraint Jacobian (and Hessian of the Lagrangian), resulting in much memory and CPU time being devoted to solution of linear systems by the NLP solver used to solve (3).

2.2 Infeasible initial points

It is recommended (but not necessary!) that the user supply an initial point which is feasible with respect to the constraints of problem (2), in particular the matrix inequality constraints. If the latter is not possible, an option can be set (see section 4) so that `fminsdp` attempts to solve the following problem in place of (3):

$$\begin{aligned} &\underset{\boldsymbol{x}\in\mathbb{R}^n,\, \boldsymbol{\ell}_1\in\mathbb{R}^{p_1},\, ...,\, \boldsymbol{\ell}_q\in\mathbb{R}^{p_q},\, s\in\mathbb{R}}{\text{minimize}} \quad f(\boldsymbol{x}) + cs \\ &\text{subject to} \begin{cases} \boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq} \\ \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\ \boldsymbol{c}_{eq}(\boldsymbol{x}) = \boldsymbol{0} \\ \boldsymbol{c}(\boldsymbol{x}) \leq \boldsymbol{0} \\ \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \\ \tilde{\boldsymbol{l}} \leq (\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_q) \leq \tilde{\boldsymbol{u}} \\ \mathsf{svec}\left(\boldsymbol{\mathcal{A}}_i(\boldsymbol{x}) + s\boldsymbol{I}^{m_i} - \boldsymbol{L}_i(\boldsymbol{\ell}_i)\boldsymbol{L}_i(\boldsymbol{\ell}_i)^{\mathsf{T}}\right) = \boldsymbol{0}, \qquad i = 1, ..., q \\ \mathsf{diag}\{\boldsymbol{L}_i(\boldsymbol{\ell}_i)\} \geq \boldsymbol{0}, \qquad i = 1, ..., q \\ \underline{s} \leq s \leq \overline{s}. \end{cases} \end{aligned} \tag{4}$$

Here s is an auxiliary variable, $\boldsymbol{I}^{m_i}$, $i = 1, ..., q$, are identity matrices of size $m_i \times m_i$, and $\underline{s}$ and $\overline{s}$ are constants. The constant c appearing in the objective should be set to some large positive number (finding a suitable value might require experimenting a bit) such that s becomes close to zero at a solution.

3 Alternatives

There are many important special cases of problem (2) for which efficient specialized solvers are available. Although this is possible, `fminsdp` is not intended to replace such solvers, which, when applicable, can be orders of magnitude faster.

The following is an incomplete list of currently available solvers:

- Problems with linear matrix inequality (LMI) constraints:
  SeDuMi, SDPT3, SDPA, LMI Lab, PENSDP, BMISolver, PENBMI, PENNON
- Problems with bilinear matrix inequality (BMI) constraints:.
  BMISolver, PENBMI, PENNON
- General problems of type (2):
  PENNON

(Except for PENNON, there are additional constraints on the structure of the problems treated by the listed solvers). Note that PENNON [6] (and its open-source version PENLAB) handles the same class of problems as `fminsdp` and may therefore be considered as an alternative to this solver.

The Matlab code YALMIP [7] provides a convenient unified interface to the solvers listed above (except BMISolver).

One could of think of other criteria, not without drawbacks of course, for positive semi-definiteness besides the Cholesky factorization that might be used to obtain a standard NLP formulation of (2). Non-negativity of the smallest eigenvalue or of the leading principle minors of a matrix are both necessary and sufficient criteria. If one is satisfied with sufficiency, diagonal dominance could also be considered.

4 Using `fminsdp`

A user of `fminsdp` is expected to provide two functions: one for evaluating the objective function (1) and one for evaluating the non-linear constraints (2).

1. `[fval,`*`grad`*`] = objfun(x)`
2. `[cineq,ceq,`*`cineqgrad,ceqgrad`*`] = nonlcon(x)`

(Here and in the following it is assumed that the reader is familiar with the Matlab syntax.) The return arguments written in italics are optional and only needed if the user wishes to provide gradients for the objective and constraints. The matrix inequality constraints are defined in the non-linear constraints function (even linear matrix inequalities). The values of the constraint matrices, vectorized using `svec`, are returned as the last elements in the output vector `ceq`; see Section 5 for an example.

In addition to objective and non-linear constraints functions, the user may optionally provide a function for evaluating the Hessian of the Lagrangian:

- `H = hessian(x,lambda)`,

where `lambda` is a struct with two fields `ineqnonlin` and `eqnonlin` containing values of the Lagrange multipliers associated with the non-linear inequality and equality constraints, respectively, and one field `sigma` which only used when running with NLP-solver Ipopt.

The calling syntax of `fminsdp` is similar to that of `fmincon`:

```
>> [x,fval,exitflag,output,lambda,grad,hessian] = ...
                        fminsdp(objfun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
```

Unlike fmincon, at least 9 input arguments are required. If your problem has no linear constraints and no bounds, the corresponding input arguments can be set to empty matrices. Note that `fminsdp` does not support passing additional arguments to the objective or constraint functions through an 11-th input argument. To pass additional arguments to the user-supplied functions one should use anonymous or nested functions.

4.1 Options

A number of options are available when calling `fminsdp`. These are passed to the code in the form of a struct containing parameter-value pairs. The most convenient way to specify options is to use the function `sdpoptionset`:

```
>> options = sdpoptionset('MatrixInequalities','on');
```

In addition to the options accepted by the function `optimset` from the Matlab Optimization Toolbox, the following options are available when calling `fminsdp`:

`MatrixInequalities` logical scalar
Indicates whether or not the problem has any matrix inequality constraints. If not, `fminsdp` will simply call the NLP solver specified using options.NLPsolver.
Default: true

`Aind` scalar or numeric array
Marks the beginning of each matrix constraint in the vector `ceq` returned from the non-linear constraints function. If the option `sp_pattern`, described below, is used it is only necessary to mark the beginning of the `first` matrix constraint.
Default: 1

`NLPsolver` {'fmincon', 'ipopt', 'snopt', 'knitro', 'mma', 'gcmma'}
Select NLP-solver for the reformulated problem. Currently `fminsdp` provides interfaces to fmincon, Ipopt [10], SNOPT [4], KNITRO [2], MMA/GCMMA [8]. NOTE: If you run Ipopt older than 3.11.0,

make sure to modify the file ipopt_main.m appropriately.
Default: 'fmincon'

`sp_pattern` (sparse) matrix or cell array of matrices
Sparsity patterns of the Cholesky factors of the matrix constraints. These can be generated by a call to `symbol_fact`. If this option is used, one must provide one matrix for each matrix constraint.
Default: []

`L0` (sparse) matrix or cell array of matrices
Initial guess for the Cholesky factors of the matrix constraints. If the user does not provide an initial guess, `fminsdp` will generate one by attempting a Cholesky factorization of the constraint matrices at the initial point. If this fails, `fminsdp` will add a multiple of the identity matrix to the constraint matrices until all of them are positive definite and use the Cholesky factorizations of these matrices as an initial guess.
Default: []

`Ldiag_low` scalar or numeric array
Lower bound(s) on the diagonal elements of the Cholesky factors.
Default: 0

`L_low` scalar or array of doubles
Lower bound(s) on the off-diagonal element of the Cholesky factors
Default: -inf

`L_upp` scalar or array of doubles
Upper bound(s) on the elements of the Cholesky factors (including the diagonal elements).
Default: inf

`c` non-negative scalar
By setting $c > 0$ `fminsdp` will attempt to solve problem (4) instead of (3). The user may in this case also let the objective function be empty; i.e., the first input argument to `fminsdp` can be set to '[]'. This is useful when one wants to check feasibility of one or more matrix inequalities.
Default: 0

`s_low` scalar
Lower bound on the auxiliary variable s in problem (4).
Default: 0

`s_upp` scalar
Upper bound on the auxiliary variable s in problem (4).
Default: inf

`HessianCheck` {'on', 'off'}
Simple check of Hessian of the Lagrangian against finite differences at the initial point. Assumes you have the code DERIVEST, which must be obtained separately, on your Matlab path. This check can be very time consuming and should preferably be carried out on small instances of a problem.
Default: 'off'

`HessMult` {function_handle, 'on'}
If using fmincon with `options.SubProblemAlgorithm = 'cg'`, you can work with Hessian times vector products directly, thereby avoiding the formation of the full Hessian of the Lagrangian. Set to a function handle or simply to 'on' if the Hessian of the Lagrangian with respect to the primary variables is zero.
Default: []

`ipopt` struct
Options to be passed on to NLP solver Ipopt. Please refer to the Ipopt documentation for a list of available options.
Default: []

`eigs_opts` struct
Options passed to the Matlab function `eigs` used for eigenvalue computations.
Default: struct('isreal',true,'issym',true)

`KnitroOptionsFile` character array
Name of an options file to be read by NLP solver KNITRO. Please refer to the KNITRO documentation for a list of available options.
Default: []

`SnoptOptionsFile` character array
Name of an options file to be read by NLP solver SNOPT. Please refer to the SNOPT documentation for a list of available options.
Default: []

`GradPattern` numeric array
Sparsity pattern for the gradient of the objective function. Only used by NLP solver SNOPT and only effective if you also set `options.JacobPattern` (see the fmincon documentation for details on the latter).
Default: []

4.2 Output

The available output arguments from `fminsdp` are the same as those of fmincon:

```
>> [x,fval,exitflag,output,lambda,grad,hessian] = fminsdp(...)
```

The only difference is that the struct `output`, the fourth output argument, contains some additional fields:

`A` cell array of matrices
Constraint matrices evaluated at the solution `x`.

`L` cell array of matrices
Cholesky factors of the constraint matrices evaluated at the solution `x`. These are computed by assembling each $\boldsymbol{L}_i$ from the variable vector $\boldsymbol{\ell}_i$ and not by a Cholesky factorization of the corresponding constraint matrix.

`L0` cell array of matrices
Cholesky factors of the constraint matrices evaluated at the initial point `x0`.

`nxvars` numeric scalar
Number of primary variables.

`nLvars` numeric scalar
Number of auxiliary variables.

`A_size` numeric array
Size of the constraint matrices; i.e., m_i in (3).

`nMatrixConstraints` numeric scalar
Number of matrix constraints.

`NLPsolver` string
Selected NLP solver.

It the user has set `options.c` to some positive number in order to solve problem (4), then two additional fields are available:

`s0` double scalar
Initial value for the auxiliary variable s.

`s` double scalar
Value of s at the solution.

5 A tutorial example

To aid the user, a very simple tutorial example is provided here. For more details, please refer to the examples found in the `examples`-folder.

Consider the following (non-sense) problem:

$$\begin{aligned} &\underset{\boldsymbol{x}\in\mathbb{R}}{\text{minimize}}\ \ x^2 \\ &\text{subject to} \begin{cases} x^3 = 0 \\ x^2 \leq 0 \\ \begin{pmatrix} x & x^2 \\ x^2 & 0 \end{pmatrix} \succeq \mathbf{0} \end{cases} \end{aligned} \tag{5}$$

To solve this problem using `fminsdp` we need to implement at least two functions:

1. The objective function:

```
function [fval,grad] = objfun(x)
fval = x^2;
if nargout>1
   grad = 2*x;
end
```

2. The non-linear constraints function:

```
function [cineq,ceq,cineqgrad,ceqgrad] = nonlcon(x)
cineq = x^2;
ceq = [x^3; svec([x x^2; x^2 0])];
if nargout>2
   cineqgrad = 2*x;
   ceqgrad = [3*x^2 svec([1 2*x; 2*x 0])'];
end
```

A function for evaluating the Hessian of the Lagrangian is optional, but recommended. One such function is given here:

```
function H = hessian(x, lambda)
H = lambda.ineqnonlin*2 + lambda.eqnonlin(1)*6*x  + ...
    lambda.eqnonlin(2:end,1)'*svec([0 2; 2 0]));
```

Using the functions specified above, a simple script to solve problem (5) can now be written:

```
% Mark the beginning of the matrix inequality constraints in the vector ceq
% returned from nonlcon
options.Aind = 2;

% Specify that analytical gradients should be used
options.GradObj = 'on';
options.GradConstr = 'on';

% Specify that the function "hessian" should be used for the
% Hessian of the Lagrangian
options.Hessian = 'user-supplied'
options.HessFcn = @(x,lambda) hessian(x,lambda);

% Specify initial point
x0 = 1;

% Call fminsdp
[x,fval] = fminsdp(objfun,x0,[],[],[],[],[],[],nonlcon,options);
```

6 Practicalities

For a given problem there are usually a number of things that can be done to improve numerical performance. One can, for instance, try out various scalings or introduce new variables and constraints to reduce the degree of non-linearity. Here are some additional tips (the first one obviously specific to `fminsdp`):

Exploit sparsity

In many cases, the functions $\boldsymbol{\mathcal{A}}_i$, $i = 1, ..., q$ are sparse in the sense that the matrix $\boldsymbol{\mathcal{A}}_i(\boldsymbol{x})$ is sparse for every $\boldsymbol{x}$. This fact can be exploited to reduce the number of auxiliary variables and nonlinear constraints.

To take advantage of sparsity, the user should compute the sparsity pattern of the Cholesky factor for each matrix constraint (or an (pessimistic) estimate thereof) and supply this to `fminsdp` via the option `sp_pattern`. For the dummy problem in Section 5 we simply add three lines to the driver script:

```
A = [1 1; 1 0]             % Sparsity pattern of the constraint matrix
sp = symbol_fact(A)        % Symbolic Cholesky factorization
options.sp_pattern = sp;

...

[x,fval] = fminsdp(@(x) objfun(x),x0,[],[],[],...
   [],[],[],@(x) nonlcon(x),options);
```

Add artificial upper and lower bounds

Even though some of the variables in your problem has no ”natural” bounds on them, it is usually wise to add upper and lower bounds. These bounds should of course be loose enough to not exclude any solution of interest, but even if they don't, setting them too tight might have a negative impact on the solution process. Therefore, a bit of experimenting is recommended.

Bounds on the auxiliary variables can be specified by setting `options.L_low` and `options.L_upp`.

Experiment with various solvers and algorihms

`fminsdp` can use fmincon, SNOPT, KNITRO and Ipopt to solve problem (3). In addition, fmincon and KNITRO lets the user choose between three different algorithms (two interior-point and one sqp). It is unlikely that a single solver and/or algorithm is best suited for all types of problems so it is recommended that the user experiment with various alternatives. The author's experience is that the interior-point method of fmincon (selected by setting `options.Algorithm='interior-point'` and `options.SubProblemAlgorithm = 'ldl-factorization'`) works well in terms of the number of function evaluations, but that more efficient handling of the linear algebra makes the other solvers better suited for larger problems.

Provide gradients and Hessian of the Lagrangian

Providing code that computes gradients and the Hessian of the Lagrangian is usually a good idea (if you use Ipopt you *must* provide code for evaluating gradients; SNOPT does not make use of code for evaluating the Hessian of the Lagrangian). Due to the homogeneity of `svec` one has simply

$$\frac{\partial \mathsf{svec}(\boldsymbol{\mathcal{A}})}{\partial x_i} = \mathsf{svec}\left(\frac{\partial \boldsymbol{\mathcal{A}}}{\partial x_i}\right),$$

and for a function $L(\boldsymbol{x}) = \boldsymbol{\lambda}^{\mathsf{T}}\mathsf{svec}(\boldsymbol{\mathcal{A}}(\boldsymbol{x}))$,

$$\frac{\partial^2 L}{\partial x_i \partial x_j} = \boldsymbol{\lambda}^T \mathsf{svec}\left(\frac{\partial^2 \boldsymbol{\mathcal{A}}}{\partial x_i \partial x_j}\right).$$

The user is strongly recommended to check the correctness of the derivatives by comparing against finite-difference approximations. This can be done by setting `options.DerivativeCheck='on'` and `options.HessianCheck='on'`. If necessary, the accuracy of the finite-difference approximations in `fmincon` can be increased by setting `options.FinDiffType='central'`.

If you use NLP solvers SNOPT, Ipopt or KNITRO, you can also provide sparsity patterns for the constraint Jacobian, and Ipopt and KNITRO can also exploit a sparsity pattern for the Hessian of the Lagrangian. Said patterns are passed to the solvers by setting `options.JacobPattern` and `options.HessPattern` to sparse matrices.

As an alternative to deriving and implementing derivatives one might consider using automatic differentiation, which provides derivatives with accuracy to machine precision, and, at least in principle, does so completely automatically.

7 Examples

Implementations of three example problem can be found in the folder `examples`. This section comprise a brief description of these examples, all of which are concerned with structural optimization of (plane) trusses. Please note that there are often many different ways to formulate such problems [3, 1], and formulations other than those given here may certainly be better suited to one's needs. Here, however, we are specifically interested in solving problems with matrix inequality constraints.

Let $n_d = 2$ be the number of spatial dimensions, N the number of nodes in the truss, and n_{fixed} the number of prescribed (to zero) displacement components. The number of displacement degrees of freedom is $n = n_d N - n_{fixed}$ and there are m potential bars in the truss. The m bar volumes are collected in a vector $\boldsymbol{x} \geq \boldsymbol{0}$ which is used to parametrize the design of the truss. Assuming (infinitesimally) small deformations and a quasi-static situation, the nodal displacement vector $\boldsymbol{u} \in \mathbb{R}^n$ satisfies the equilibrium equation

$$\boldsymbol{K}(\boldsymbol{x})\boldsymbol{u} = \boldsymbol{f}, \tag{6}$$

where $\boldsymbol{K}(\boldsymbol{x})$ is known as the stiffness matrix and $\boldsymbol{f} \in \mathbb{R}^n$ contains forces applied to the nodes. The stiffness matrix is given by

$$\boldsymbol{K}(\boldsymbol{x}) = \sum_{i=1}^{m} x_i E_i \boldsymbol{b}_i \boldsymbol{b}_i^{\mathsf{T}},$$

where x_i is the volume of the i:th bar, E_i is Young's modulus of said bar, and the vector $\boldsymbol{b}_i$ depends on the geometry of the undeformed truss. Clearly, $\boldsymbol{K}(\boldsymbol{x})$ is positive semi-definite and symmetric. Assuming rigid body motions are prevented, by appropriate support conditions, and $\boldsymbol{x} > \boldsymbol{0}$, it is even positive definite.

1. The first problem in the `examples`-folder is to minimize the volume of a truss subject to the equilibrium condition (6) and an upper bound c on the compliance $\boldsymbol{f}^{\mathsf{T}}\boldsymbol{u}$. This can be formulated as a problem involving a single LMI:

$$\begin{aligned} &\underset{\boldsymbol{x}\in\mathbb{R}^m}{\text{minimize}} \quad \sum_{i=1}^{m} x_i \\ &\text{subject to} \quad \begin{cases} \begin{pmatrix} c & \boldsymbol{f}^{\mathsf{T}} \\ \boldsymbol{f} & \boldsymbol{K}(\boldsymbol{x}) \end{pmatrix} \succeq \boldsymbol{0} \\ x_i \geq 0, \quad i = 1,\ldots,m. \end{cases} \end{aligned} \tag{7}$$

2. A drawback of problem (7) is that the optimized structures may be unstable in the sense that they are prone to global buckling. One way to avoid many such designs is to impose an additional constraint [5] requiring that

$$\boldsymbol{K}(\boldsymbol{x}) + \boldsymbol{G}(\boldsymbol{u}(\boldsymbol{x}),\boldsymbol{x}) \succeq \boldsymbol{0}, \tag{8}$$

where, the geometrix stiffness matrix,

$$\boldsymbol{G}(\boldsymbol{u}(\boldsymbol{x}),\boldsymbol{x}) = \sum_{i=1}^{m} x_i E_i \boldsymbol{b}_i^{\mathsf{T}}\boldsymbol{u}(\boldsymbol{x})\boldsymbol{\gamma}_i\boldsymbol{\gamma}_i^{\mathsf{T}},$$

in which $\boldsymbol{u}(\boldsymbol{x})$ denotes a solution to (6) and $\boldsymbol{\gamma}_i$, $i = 1,\ldots,m$, depend on the geometry of the undeformed truss.

Adding (8) to (7) leads to the following problem involving both a linear and a non-linear matrix inequality[1]:

$$\begin{aligned} &\underset{\boldsymbol{x}\in\mathbb{R}^m}{\text{minimize}} \quad \sum_{i=1}^{m} x_i \\ &\text{subject to} \quad \begin{cases} \begin{pmatrix} c & \boldsymbol{f}^{\mathsf{T}} \\ \boldsymbol{f} & \boldsymbol{K}(\boldsymbol{x}) \end{pmatrix} \succeq \boldsymbol{0} \\ \boldsymbol{K}(\boldsymbol{x}) + \boldsymbol{G}(\boldsymbol{u}(\boldsymbol{x}),\boldsymbol{x}) \succeq \boldsymbol{0} \\ x_i \geq \epsilon, \quad i = 1,\ldots,m, \end{cases} \end{aligned} \tag{9}$$

where ϵ is a small positive number introduced to avoid singularity of the stiffness matrix. Kočvara [5] showed that a solution to this problem corresponds to a truss that will not exhibit global, linear buckling for loads of the form $\tau\boldsymbol{f}$, $\tau \in [0,1)$.

3. Finally, the third example problem is an alternative formulation of problem (9) obtained by treating the displacements as explicit variables in the optimization problem:

$$\begin{aligned} &\underset{\boldsymbol{x}\in\mathbb{R}^m,\boldsymbol{u}\in\mathbb{R}^n}{\text{minimize}} \quad \sum_{i=1}^{m} x_i \\ &\text{subject to} \quad \begin{cases} \boldsymbol{f}^{\mathsf{T}}\boldsymbol{u} \leq c \\ \boldsymbol{K}(\boldsymbol{x})\boldsymbol{u} = \boldsymbol{f} \\ \boldsymbol{K}(\boldsymbol{x}) + \boldsymbol{G}(\boldsymbol{u},\boldsymbol{x}) \succeq \boldsymbol{0} \\ x_i \geq \epsilon, \quad i = 1,\ldots,m. \end{cases} \end{aligned}$$

In this formulation, the upper bound on the compliance is a linear constraint, the equilibrium equation a set of bilinear equality constraints, and the stability constraint is a BMI.

To see how these problems can be solved with `fminsdp`, and what optimized designs might look like, please check out the Matlab codes found in the `examples`-folder.

[1] In practice it is perhaps better to replace the LMI in (9) by the non-linear constraint $\boldsymbol{f}^{\mathsf{T}}\boldsymbol{u}(\boldsymbol{x}) \leq c$ but we keep it to illustrate solution of a problem with more than one matrix inequality.

References

[1] Bendsøe M, Ben-Tal A, Zowe J (1994) Optimization methods for truss geometry and topology design. Structural Optimization 7:141–159

[2] Byrd R, Nocedal J, Waltz R (2006) KNITRO: An integrated package for nonlinear optimization. In: Large-Scale Nonlinear Optimization, Springer, pp 35–59

[3] Christensen P, Klarbring A (2009) An Introduction to Structural Optimization. Springer

[4] Gill P, Murray W, Saunders M (2002) SNOPT: an SQP algorithm for large-scale constrained optimization. SIAM journal on optimization 12:979–1006

[5] Kočvara M (2002) On the modeling and solving of the truss design problem with global stability constraints. Structural and Multidisciplinary Optimization 23:189–203

[6] Kočvara M, Stingl M (2003) PENNON - A code for convex nonlinear and semidefinite programming. Optimization Methods and Software 18:317–333

[7] Löfberg J (2004) YALMIP : A toolbox for modeling and optimization in MATLAB. In: Proceedings of the CACSD Conference, Taipei, Taiwan, URL `http://users.isy.liu.se/johanl/yalmip`

[8] Svanberg K (2007) MMA and GCMMA, versions September 2007. URL `http://www.math.kth.se/\char‘~krille/gcmma07.pdf`

[9] The MathWorks Inc (2011) Optimization Toolbox - User's Guide. Natick, MA, 2000

[10] Wächter A, Biegler L (2006) On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. Mathematical programming 106:25–57