

On the feasibility for the system of quadratic equations MATLAB Library

Anatoly Dymarsky, Elena Gryazina, Boris Polyak, Sergei Volodin

1 Notations

The goal of the project is to solve a number of tasks for quadratic maps, which are

1. (Real case) The map $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$ s.t.

$$f_i(x) = x^T A_i x + 2b_i^T x, A_i = A_i^T$$

2. (Complex case) The map $f: \mathbb{C}^n \rightarrow \mathbb{R}^m$ s.t.

$$f_i(x) = x^* A_i x + b_i^* x + x^* b_i, A_i = A_i^*$$

Where \cdot^* is Hermitian conjugate.

From this point on, X denotes \mathbb{R}^n for real case or \mathbb{C}^n for complex case.

We use the following notations:

Definition 1.1. For a vector $c \in \mathbb{R}^n$ and tuple of matrices (A_1, \dots, A_n) (or vectors) the dot product is defined as following:

$$c \cdot A = \sum_{i=1}^n c_i A_i$$

Definition 1.2. The image of f is denoted as F :

$$F = f(X)$$

Definition 1.3. The convex hull of F is denoted as G :

$$G = \text{conv } F$$

Definition 1.4. The boundary points of F touched by a tangent hyperplane with normal vector $c \in \mathbb{R}^m$:

$$\partial F_c = \arg \min_{y \in F} (c, y)$$

Definition 1.5. The boundary points of G touched by a tangent hyperplane with normal vector $c \in \mathbb{R}^m$:

$$\partial G_c = \arg \min_{y \in G} (c, y)$$

2 Functions

The library consists of a number of functions defined in separate .m files. Input format for the map is the following:

- The number $A(i, j, k)$ denotes i 'th row and j 'th column of the matrix A_k
- The number $b(i, j)$ denotes i 'th element of the vector $b_j \in \mathbb{R}^m$

1. **Feasibility membership oracle**
Given:

- The map f as matrices A and vectors b
- A point $y \in \mathbb{R}^m$.

Determine: if $y \in F$

`is_infeasible = infeasibility_oracle(A, b, y)`

This function tries to separate the point y from the convex hull G with a hyperplane. See Theorem 3.2 from the article.

Return value:

- 1 means that the separation was successful and the point $y \notin G$. This implies $y \notin F$.
- 0 means that the feasibility is uncertain.

2. Boundary oracle

Given:

- The map f as matrices A and vectors b
- A point $y \in F$
- A direction $d \in \mathbb{R}^m$

The following two tasks are considered:

(a) `[t, is_in_F] = boundary_oracle(A, b, y, d)`

This function finds the point $y + td$ on the boundary ∂G with the largest t :

$$t = \sup\{\tau \mid y + \tau d \in G\}$$

Return value:

- t is the largest step in direction d
- `is_in_F` is a binary variable indicating if the resulting point $y + td$ belongs to F

Exception: if optimization task failed

(b) `c = get_c_from_d(A, b, y, d)`

This function obtains the normal vector c at the boundary point $y + td$ using dual problem (5) from the article.

Return value: the normal vector c s.t. $y + td \in \partial G_c$

Exception: if optimization task failed

3. Nonconvexity certificate

Given:

- The map f as matrices A and vectors b
- A point $y \in F$
- Number of iterations k

The following two tasks are considered:

(a) `c = get_c_minus(A, b, y, k)`

This function runs k iterations of the following procedure:

- Generate random direction d
- Obtain a normal vector c using `get_c_from_d()`
- Check if ∂F_c is nonconvex using Theorem 3.4 from the article
- Return c if so, continue otherwise

Return value: c s.t. ∂F_c is nonconvex

Exception: if c was not found in k iterations

(b) `is_nonconvex = nonconvexity_certificate(A, b, y, k)`

This function runs `get_c_minus()` and outputs 1 if the normal vector c was found. In this case the image F is guaranteed to be nonconvex.

Return value: 1 if F is nonconvex, 0 if result is uncertain

4. Positive-definite $c \cdot A$

Given:

- The map f as matrices A and vectors b
- The initial normal vector p

The following three tasks are considered:

(a) `c_plus = get_near_c_plus(A, p, gamma);`

This function finds the nearest to p vector c_+ such that

$$c_+ \cdot A \succeq 0$$

via solving the following optimization task ($c_{+\perp}$ is the part of c_+ orthogonal to p):

$$\begin{aligned} \min \quad & \gamma \|c_+\|^2 + c_{+\perp}^2 \\ \text{s.t.} \quad & c_+ \cdot A - I \succeq 0 \\ & (c_+, p) \geq 0 \end{aligned}$$

Return value: c_+ s.t. $c_+ \cdot A \succeq 0$

Exception: if c_+ was not found

(b) `c_plus = get_c_plus(A, k)`

This function generates a random vector p and then finds c_+ nearest to it. Function generates at most k vectors p .

Return value: c_+ s.t. $c_+ \cdot A \succeq 0$

Exception: if c_+ was not found

(c) `c_plus = get_best_plus(A)`

This function returns the "best" vector c_+ s.t. $c_+ \cdot A \succeq 0$ via the following problem:

$$\begin{aligned} \max \quad & \lambda_{\min}(c_+ \cdot A) \\ \text{s.t.} \quad & \|c\|^2 \leq 1 \end{aligned}$$

The spectrum of the resulting matrix $c_+ \cdot A$ is separated from 0 the most.

Return value: c_+ s.t. $c_+ \cdot A \succeq 0$

Exception: if c_+ was not found

5. Convex subpart

Given:

- The map f as matrices A and vectors b
- The point $y \in F$
- Number of iterations k
- Vector c_+ s.t. $c_+ \cdot A \succeq 0$
- Number of iterations `k_c_minus` for the nonconvexity certificate

`z_max = get_z_max(A, b, y, k, c_plus, k_c_minus)`

This function performs the following procedure k times:

- Changing basis with `change_basis`
- Obtaining $c \in C_-$ using `get_c_minus`
- Minimizing $z(c)$ using c as a starting point

The resulting z_{\max} is a minimum over all obtained z 's

Return value: Minimal value z_{\max} or Inf if no nonconvexities were found

Exception: None

Other functions

Name	Input	Call	Description	Return value	Exception
Random map	Dimensions n, m	get_random_f(n, m, is_complex)	Generates random map f	[A, b]	None
Value at x	The point $x \in X$	quadratic_map(A, b, x)	Calculates $f(x)$	$y = f(x)$	None
Product $c \cdot A$	Normal vector c	get_Ac(A, c)	Calculates $c \cdot A$	$A_c = c \cdot A$	None
Get H_c	$c, y \in \mathbb{R}^m$	get_H_c(A, b, c, y)	$H_c = \begin{pmatrix} A_c & b_c \\ b'_c & -(c, y) \end{pmatrix}$	H_c	None
Minimize $z(c)$	c, c_+ , step β	minimize_z_c(A, b, c, c_plus, beta_initial, max_step)	Calculates $\inf_{c \in C_-} z(c)$	[z, c_array, z_array]	If failed
\mathbb{R}^n projection		project(A, b, c, x_0, delta_c, normal, search_area_size)	Projects $c + \Delta c$ to C_-	[c_new, lambda]	If failed
\mathbb{C}^n projection		project_descent(A, b, c, normal_1, normal_2)	Projects c to C_-	[c_new, distance]	If failed
Gradient $\frac{\partial z}{\partial c}$	Normal c	get_dz_dc(A, b, c)	Calculates $Q, \nabla z(c)$, normal vectors n_1, n_2	[Q, Q_inv, k, v, lambda_min, z, dz_dc, normal_re, normal_im, drho_dc]	None
Change of basis	c_+	change_basis(A, b, c_plus)	$\begin{cases} x = S(x' + x_0) \\ y = y' + y_0 \end{cases} \quad \text{s.t.} \quad \begin{cases} c_+ \cdot A_0 = I \\ c_+ \cdot b_0 = 0 \end{cases}$	[A_new, b_new, x0, y0]	None