EPiC Series in Computing

Volume 104, 2024, Pages 221–242

Proceedings of 3rd International Workshop on
Mathematical Modeling and Scientific Computing

Linear Differential Games with Multi-Dimensional Terminal Target Set: Geometric Approach

Anton V. Mikhailov[1,2] and Sergey S. Kumkov[1,3]

[1] Krasovskii Institute of Mathematics and Mechanics,
Russian Academy of Sciences, Yekaterinburg, Russia
[2] tigr-mav@yandex.ru
[3] sskumk@gmail.com

Abstract

The paper deals with linear differential games with a fixed terminal instant, convex geometric constraints of the players' controls, and convex terminal target set. The first player tries to guide the system to the target set at the terminal instant, the second one hinders this. In the 1960's, L. S. Pontryagin proposed a theoretic geometric procedure for approximate constructing time sections of the maximal stable bridge for games of this type. This procedure is known as the second Pontryagin's method. At the beginning of the 1980's in the Krasovskii Institute of Mathematics and Mechanics (Yekaterinburg, Russia), a computational algorithm for the procedure has been suggested and implemented as a computer program. However, this algorithm is suitable only for games with two-dimensional equivalent phase vector. The authors suggest a procedure suitable for games with a multi-dimensional phase vector. For an implementation of this method, one needs implementations of convex hull construction, Minkowski sum and difference. The authors have taken known algorithms for convex hull construction and Minkowski sum. An algorithm for Minkowski difference as well as some procedures for conversion of different representations of multi-dimensional polytopes to each other have been suggested. All these algorithms have been implemented as a computer library in C# by the authors. A series of model differential games has been computed.

1 Introduction

The authors are interested in numerical study of differential games [11, 19, 13, 14]. More exactly, the main goal is games with multi-dimensional phase vector, linear dynamics, fixed terminal instant, geometric constraints for the players' controls, and convex terminal target set. In this case, the main element of the solution is the maximal stable bridge [13, 14] (the solvability set). Also, it is possible to consider problems with quasiconvex terminal payoff. (A function is called *quasiconvex* if it has convex level sets (Lebesgue sets).) In this case, the solution of the game is defined by its value function (the optimal result function).

Nowadays, the most widespread approaches for constructing the value function are of grid kind [4, 1, 2]. The algorithms of this type are very universal with respect to the class of the problems to be solved, to the dimension of the phase vector, to the type of dynamics, etc.

V.L. Turova, A.E. Kovtanyuk and J. Zimmer (eds.), MMSC 2024 (EPiC Series in Computing, vol. 104), pp. 221–242

But their main disadvantage is their performance, which decreases rapidly with growth of the dimension of the phase vector. Also, in this case, the memory demands are affected by the curse of dimension. The size of necessary memory is defined by the requirement to store the entire grid covering the domain where the problem is solved. Time expenditures are caused by recomputations at each iteration of the algorithm of the data over the entire grid, not only the area where some changes might happen indeed.

Potentially, geometric methods do not possess these disadvantages, totally or to some extent. In the framework of a geometric approach, the main solution element is the maximal stable bridge constructed for the given dynamics, players' constraints, and terminal set. To construct a time section of the maximal stable bridge for a linear differential game, in the second half of the 1960's, L.S. Pontryagin suggested [19] a theoretic procedure; he named the time section to be constructed as *alternating integral*. This procedure has a geometric nature and now is called *the second Pontryagin's method*. The convergence of this procedure for the case of convex terminal set (and linear dynamics) has been proved by B.N. Pshenichnyi in [20]. For the general case of non-convex target set, the convergence is not proved yet.

The second Pontryagin's method involves the operations of Minkowski sum and difference (called by L.S. Pontryagin as algebraic sum and geometric difference, respectively). When geometric methods are applied, one deals with the evolution of a set described by its boundary. Such a description potentially is much more saving than a grid one. (However, there are grid methods dealing with the boundary of an evolving set, see [21].) In general, performance of geometric procedures is higher, but their realization needs for much subtler algorithms and data structures.

In the case of a two-dimensional phase vector, in the 1980's, geometric methods have been developed for differential games of many classes. For games with dynamics linear on the phase vector and players' controls, two-dimensional phase vector, fixed terminal instant, geometric constraints for the players' controls, and convex terminal (two-dimensional) set, a quite effective procedure has been suggested [12] to construct the maximal stable bridges (the Lebesgue sets of the value function for games with quasiconvex terminal payoff). But the second dimension of the phase vector is constitutive for this procedure since some specifics of the Euclidean plane are used. Further attempts [24, 3] to transit these ideas to a multi-dimensional space, unfortunately, were not well enough.

In [24], algorithms for the Minkowski sum and difference are suggested for the case when the second summand and subtrahend are *zonotopes*, that is, they are representable as a Minkowski sum of several segments. So, actually, in this case, one needs algorithms for the Minkowski sum and difference with a segment, which are not general enough.

In [3], a general case of polytope operands is considered. But the suggested algorithms essentially involve the Fourier-Motzkin procedure for eliminating inequalities from a system, which is quite heavy-weight even for modern computers.

At the end of 1990's in Moscow State University (Moscow, Russia) [10] and in Moscow Institute of Physics and Technology (Moscow, Russia) [18], some algorithms similar to each other have been worked out. They implement some approximate construction of Minkowski sum and difference of two multi-dimensional polytopes. The (upper) approximation is obtained by replacing the true polytopes by ones with a fixed grid of outer normals to their facets.

For time-optimal problems in the plane, some algorithms have been suggested [16, 17] for constructing level sets of the value function. However, the concepts behind these algorithms are primarily conditioned by specific properties of the Euclidean plane. Further on their basis, a procedure for linear differential games in the plane with a non-convex terminal target set has been suggested [9] that uses algorithms for Minkowski sum and difference of a convex and

a non-convex polygons. But, eventually, these algorithms were connected too tightly with a problem, for which they were developed, to be used for arbitrary problems even in the plane. Similar ideas have been set forth in [7], but they work in the plane principally and cannot be propagated into higher dimensions.

Thus, now, the authors do not know any effective algorithms for constructing the maximal stable bridges for linear differential games with a multi-dimensional phase vector. Therefore, to have such a procedure, algorithms for Minkowski sum and difference of multi-dimensional polytopes have to be found in the literature and implemented or to be worked out independently. Also, a procedure for constructing the convex hull of a set of points in a multi-dimensional space is necessary.

Concerning the convex hull construction, it seems that only one family of effective algorithms exists; it is the GiftWrapping algorithm. The most detailed description of some classic version of this algorithm can be found in [22]. This algorithm has been implemented by the authors.

The only reasonable algorithm for the Minkowski sum has been found in [5, 6]. This algorithm is not ideal since it demands an extremely rich description of polytopes-summands as face lattices. Therefore, some preparation of the polytopes is necessary if they are obtained from other algorithms in some other form.

And, finally, the authors were unable to find in the modern literature a description of a general Minkowski difference procedure for multi-dimensional polytopes. So, the main result of this paper is a description of some version of such an algorithm.

All algorithms: convex hull construction, Minkowski sum and difference, as well as some supplementary procedures — have been implemented by the authors in C# 12.0 for .NET 8.0.

An essential point when performing geometric procedures in a computer is the calculation accuracy. During testing of the implemented algorithms, it became clear that the accuracy of the standard 8-byte floating type (`double`) is insufficient for these computations. So, the created library has been rewritten using template (generic) technology. With that, one of the type parameters is the numeric type used for computations. A 16-byte floating type `ddouble` [23] and longer types have been tested. However, in the latter case, the performance drastically decreases. Also, further usage of rational arithmetic is considered by own implementation or by third-party libraries.

The paper has the following structure. Section 2 contains a formulation of the original problem from the differential game theory, which stipulates geometric studies. Some basic definitions and denotations are given in Section 3. In Section 4, representations of multi-dimensional polytopes are discussed those are used in the involved geometric algorithms. The algorithms themselves are described in Section 5; those taken from literature have quite short descriptions, the author's one is set forth in more details. The structure of the created computer library and difficulties appeared during its creation are given in Section 6. Section 7 presents some results obtained for model differential games by means of the library written by the authors. The paper is finalized by a conclusion and a list of references.

2 Problem formulation

A linear differential game is considered:

$$\begin{aligned} &\dot z = A(t)z + B(t)u + C(t)v, \\ &t \in [t_0, T], \qquad z \in \mathbb{R}^n, \\ &u \in \mathbf{P} \subset \mathbb{R}^p, \quad v \in \mathbf{Q} \subset \mathbb{R}^q, \\ &z(T) \in \mathbf{M} + \mathbf{M}^\perp. \end{aligned} \tag{1}$$

The instant T of game termination is fixed. The controls u and v of the first and second players are constrained by convex compacta $\mathbf{P}$ and $\mathbf{Q}$ in their spaces. A terminal target set is given, which is an infinite cylinder with the base $\mathbf{M}$, which, in its turn, is a convex compact subset of a d-dimensional linear subspace of some d coordinates, $d \leqslant n$.

The objective of the first player is to guide the system at the terminal instant T to this target set. Vice versa, the second player tries to deviate the system from the target set at the terminal instant T.

Consider the variable change

$$x(t) = X_{s_1,s_2,\ldots,s_d}(T,t)z(t) \tag{2}$$

defined by the matrix combined of d rows of the fundamental Cauchy matrix $X(T,t)$ for the system $\dot{z} = A(t)z$. The chosen rows correspond to the components of the phase vector of the subspace where the base $\mathbf{M}$ of the target set is located.

After such a change, one obtains a differential game *equivalent* (in some sense) to the original one (1):

$$\begin{aligned}
&\dot{x} = D(t)u + E(t)v,\\
&t \in [t_0,T],\ x \in \mathbb{R}^d,\ u \in \mathbf{P},\ v \in \mathbf{Q},\\
&D(t) = X_{s_1,s_2,\ldots,s_d}(T,t)B(t),\\
&E(t) = X_{s_1,s_2,\ldots,s_d}(T,t)C(t),\\
&x(T) \in \mathbf{M}.
\end{aligned} \tag{3}$$

This new game (3) possesses the following pleasant properties:

- the dimension d of the new phase vector x is equal to the dimension of the set $\mathbf{M}$ defining the terminal target set in game (1);
- the phase vector x is absent in the right-hand side of dynamics (3);
- now, $\mathbf{M}$ itself is the target set.

The sense of the new coordinates x is following. If the original system (1) is at a position $\big(t, z(t)\big)$, then under zero controls of the players (along the free motion of the system), the d phase coordinates defining the subspace of $\mathbf{M}$ reach the value $x(t)$ at the terminal instant T. Due to this observation, these coordinates x are often called *forecast coordinates* or *zero effort miss*. The players' controls in game (3) actually affect the final position of the system.

Equivalence of the original and new coordinates z and x means the following. If the first player from a position $\big(t, z(t)\big)$ of game (1) is able to guide the system to the target set at the terminal instant T, then in the equivalent game (3), the first player also can achieve its goal from the corresponding position $\big(t, x(t)\big)$. And vice versa, if the first player in game (3) can guide the system to the target set at the instant T from the position $\big(t, x(t)\big)$, then in the original game (1), the first player can succeed from any position $\big(t, z(t)\big)$ such that $x(t)$ and $z(t)$ are connected by relation (2).

The main element of solution of a differential game with a target set is the *maximal stable bridge* [13, 14]. A stable bridge is a set in the space of time t and phase coordinates of the game consisting of positions such that the the first player can guide the system to the target set from any such a position despite of the second player's action. The maximal stable bridge is the stable bridge maximal by inclusion. Complement of the maximal stable bridge is the maximal set wherefrom the second player guarantees evasion from the target set. The maximal stable bridge for game (3) with the target set $\mathbf{M}$ is a subset in the space $\mathbb{R}^{d+1}$ and is denoted below by $\mathbf{W}$.

If a problem with a quasiconvex terminal payoff is considered, then maximal stable bridges constructed from level sets (Lebesgue sets) of the payoff function are the level sets of the value function of the game. Thus, if one can construct the maximal stable bridge for a given terminal target set, then for a problem with a terminal payoff, the value function can be approximated arbitrarily exactly by a family of maximal stable bridges constructed for a family of level sets of the payoff taken for a quite dense grid of payoff magnitudes.

So, one needs for a numerical procedure for constructing the maximal stable bridge $\mathbf{W}$ for game (3).

A *time section* $\mathbf{W}(t)$ of the maximal stable bridge $\mathbf{W}$ at an instant $t \in [t_0, T]$ is the set $\mathbf{W}(t) = \{x \in \mathbb{R}^d \,|\, (t,x) \in \mathbf{W}\}$.

L.S. Pontryagin has suggested in [19] a theoretic procedure for constructing a time section $\mathbf{W}(t)$ of a maximal stable bridge $\mathbf{W}$ at an instant $t \in [t_0, T]$. In that work, the time section is called *alternating integral*. Later, this procedure has been called *the second Pontryagin's method*.

Introduce a grid in the time interval of the game $[t_0, T]$: $\mathcal{T} = \{t_0 < t_1 < t_2 < \ldots < t_N = T\}$. Denote $\Delta_i = t_{i+1} - t_i$, $i = 0, \ldots, N-1$; $\operatorname{diam} \mathcal{T} = \max \Delta_i$. The grid can be both uniform and non-uniform; this does not affect the idea of the procedure. So, below, the gris is assumed to be uniform with the step Δ.

The idea of the procedure is the following. For the instants t_i of the chosen grid $\mathcal{T}$, some sets $W_i = W(t_i)$ are constructed those approximate the time sections $\mathbf{W}(t_i)$:

$$W_N = M, \quad W_i = \big(W_{i+1} + (-\Delta) D(t_i) P\big) \stackrel{*}{-} \Delta E(t_i) Q, \ i = N-1, \ldots, 0. \tag{4}$$

Here, M, P, Q are some convex approximations of the sets $\mathbf{M}$, $\mathbf{P}$, $\mathbf{Q}$. The symbols "+" and "$\stackrel{*}{-}$" denote the operations of Minkowski sum and difference respectively (the algebraic sum and geometric difference in terms of [19]):

$$A + B = \{a + b \in \mathbb{R}^d \,|\, a \in A, b \in B\}, \tag{5}$$

$$A \stackrel{*}{-} B = \{x \in \mathbb{R}^d \,|\, x + B \subset A\}. \tag{6}$$

In [20], convergence of procedure (4) is proven in the following sense.

Consider a sequence of procedures (4) with time grids $\mathcal{T}_n$ and approximations M_n, P_n, Q_n such that

$$\operatorname{diam} \mathcal{T}_n \to 0, \quad h(M_n, \mathbf{M}) \to 0, \quad h(P_n, \mathbf{P}) \to 0, \quad h(Q_n, \mathbf{Q}) \to 0$$

as $n \to \infty$. The symbol $h(\cdot, \cdot)$ denotes the Hausdorff distance between compacta.

Let $W(t_0; \mathcal{T}_n, M_n, P_n, Q_n)$ be the set W_0 build by procedure (4) with the corresponding time grid and approximations. In [20], it is proven that $h\big(W(t_0; \mathcal{T}_n, M_n, P_n, Q_n), \mathbf{W}(t_0)\big) \to 0$ as $n \to \infty$.

In the general case of non-convex target set $\mathbf{M}$, convergence of procedure (4) is not proved yet.

In the desired numerical procedure, the approximations M, P, Q can be taken as multi-dimensional polytopes. In this case, the resultant sets W_i are polytopes too. This is very convenient for computer calculations.

Thus, an implementation of procedure (4) demands to implement the Minkowski sum (5) and difference (6) operations as well as a convex hull construction for a set of points, which is necessary for constructing the polytopes $D(t_i)P$ and $E(t_i)Q$. In the text below, some realizations of these operations are discussed and some results of application of the corresponding implementation of procedure (4) to model differential games are given.

3 Definitions and denotations

We work both with vectors of linear spaces as well as with points. With that, we identify a point in $\mathbb{R}^d$ and its radius-vector. The operations of linear space, addition and multiplication by a scalar, are denoted in a conventional way: $x + y$ and $\alpha \cdot x$. Also, in a traditional way, the scalar (inner) product is introduced in $\mathbb{R}^d$: $\langle x, y \rangle = \sum_{i=1}^{d} x_i y_i$.

Definition 1. The *linear span* $\operatorname{span}(X)$ of a finite set $X = \{x^{(k)}\}_{k=1}^{m}, X \subset \mathbb{R}^d$ of vectors is the collection of points

$$\operatorname{span}(X) = \Big\{ \sum_{k=1}^{m} \alpha_k \cdot x^{(k)} \,\Big|\, \alpha_k \in \mathbb{R} \Big\} \subseteq \mathbb{R}^d.$$

Definition 2. Let $\mathcal{B} = \{x^{(k)}\}_{k=1}^{m} \subset \mathbb{R}^d$ be a set of m linearly independent vectors, $o \in \mathbb{R}^d$. The set

$$\mathbb{A}_m = \langle o; \mathcal{B} \rangle = \Big\{ o + \sum_{k=1}^{m} \alpha_k \cdot x^{(k)} \,\Big|\, \alpha_k \in \mathbb{R} \Big\}$$

is called *affine subspace* of the space $\mathbb{R}^d$ with the *origin* o and the *linear basis* $\mathcal{B}$.

The construction $\langle o; \mathcal{B} \rangle$ is called an *affine basis* also.

A linear basis is called *orthogonal* if the scalar product of any two different its vectors $x^{(k)}$ equals zero. An orthogonal linear basis is called *orthonormal* if the length of any its vector is equal to 1. Traditionally, the length of a vector is the square root of its scalar square.

The number m of vectors in a linear basis $\mathcal{B}$ of an affine space $\mathbb{A}_m$ is called the *dimension* of this space and is denoted as $\dim \mathbb{A}_m$.

Note that the same affine space can be defined by different affine bases with different origins and/or different linear bases.

Definition 3. The *dimension* $\dim M$ of a set $M \subseteq \mathbb{R}^d$ is called the dimension of the minimal by inclusion affine subspace that contains the set M. A set can be *of the complete dimension* if its dimension equals d, or *of an incomplete dimension* otherwise.

Definition 4. An affine subspace $\langle o; \mathcal{B} \rangle$ having the dimension $d-1$ in the space $\mathbb{R}^d$ is called *(affine) hyperplane*.

A hyperplane can be defined also as a set of points obeying the equation $\langle n, x \rangle = c$, where $n \in \mathbb{R}^d$, $c \in \mathbb{R}$. The vector n is called the *normal of the hyperplane*. With that, the vector n is orthogonal to all vectors of the basis $\mathcal{B}$, the constant $c = \langle o, n \rangle$.

Such a hyperplane is denoted by $\operatorname{hp}(n, c)$.

A hyperplane $\operatorname{hp}(n, c)$ separates the space $\mathbb{R}^d$ to two open half-spaces defined by the inequalities $\langle n, x \rangle > c$ and $\langle n, x \rangle < c$. These half-spaces are denoted as $\operatorname{hp}^+(n, c)$ and $\operatorname{hp}^-(n, c)$ and called *positive* and *negative*, respectively. Note that the hyperplanes $\operatorname{hp}(n, c)$ and $\operatorname{hp}(-n, c)$ coincide. With that, the positive and negative half-spaces of these hyperplanes are swapped.

Definition 5. A hyperplane $\operatorname{hp}(n, c) \subset \mathbb{R}^d$ is called *support* to a closed set M if one has $M \cap \operatorname{hp}(n, c) \neq \varnothing$ and $M \cap \operatorname{hp}^+(n, c) = \varnothing$, that is, the set M "touches" the hyperplane and the interior of M is located entirely in $\operatorname{hp}^-(n, c)$.

Definition 6. A *convex combination* of a finite set $\{x^{(k)}\}_{k=1}^{m}$ of vectors is some their linear combination, whose coefficients and non-negative and their sum equals 1:

$$\sum_{k=1}^{m} \lambda_k \cdot x^{(k)}, \quad \lambda_k \geqslant 0, \sum_{k=1}^{m} \lambda_k = 1.$$

Definition 7. A set $M \subset \mathbb{R}^d$ is *convex* if with any two points belonging to the set, it contains the entire segment with these endpoints too.

Definition 8. A *convex hull* $\operatorname{conv} M$ of a set M is the set of all convex combinations of points of all finite subsets of this set. An equivalent definition: a convex hull of a set M is the minimal by inclusion convex set containing M.

Definition 9. The convex hull of a finite number of points in $\mathbb{R}^d$ (in particular, in some affine subspace of $\mathbb{R}^d$) is called a *convex polytope*. Below, the word "convex" is omitted since we work with convex polytopes only.

Definition 10. A *face* of a polytope P is an intersection of P and some its support hyperplane. Also, it is assumed that the empty set and the polytope P itself are faces of P; these two faces are called *trivial*.

Note that any support hyperplane defines a unique face of P, and each non-trivial face is generated by a unique support hyperplane if the dimension of the face is $d-1$, or by infinite number of hyperplanes otherwise. Each face of P defines its own affine subspace: as the origin of the subspace one can take any point of the face, and the basis can be constructed by the maximal number of linearly independent vectors belonging to this face.

Definition 11. *k-face* of a polytope $P \subset \mathbb{R}^d$, $\dim P = m \leqslant d$, $0 \leqslant k \leqslant m$, is a face of P such that the dimension of its affine subspace equals k. $(d-1)$-faces of P are called *facets* or *hyperfaces*. Note that 0-faces are the *vertices* of the polytope P.

Definition 12. Let us say that a face f of a polytope P is a *subface* of a face g if $f \subset g$ as a set in $\mathbb{R}^d$.

Below, $\mathcal{N}(x)$ denotes the operation of *normalization* of a vector: $\mathcal{N}(x) = x/\|x\|$. Assume $\mathcal{N}(0) = 0$.

Denote by $\mathcal{ON}(X)$ the result of *orthonormalizing* a set X of vectors, that is an orthonormal set X' of vectors such that $\operatorname{span}(X') = \operatorname{span}(X)$. Generally speaking, after applications of different procedures, one might obtain different resultant sets X' obeying this condition.

4 Representations of polytope

The algorithms discussed below accept the polytopes-operands in some form and produce the polytope-result in some form. In this section, possible representations of polytopes are enlisted.

1. A vertex representation $\texttt{Vrep}\, P$ of a polytope P (Fig. 1). It is assumed that only vertices of a polytope P are given without any additional information. In the case of multi-dimensional space, such a representation is not too substantial because it does not include any information on the neighborhood of any elements of the polytope (*vertices, facets, faces). Such an information is vital for performing many operations with polytopes.

Note that in the plane, the vertex representation, vice versa, is quite informative since the vertex order easily can be reconstructed and neighborhood information can be obtained. This is the main reason for effectiveness of planar algorithms and difficulties appearing in higher dimensions.

One can also consider a redundant vertex representation $\texttt{Vrep}^*\, P$ of a polytope P (Fig. 2). In this case, the set may contain not the vertices only, but also some inner points of the polytope including inner points of its faces. The convex hull construction just removes the redundant points from the set if producing the vertex representation of a polytope.

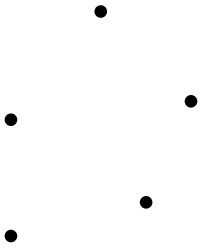


Figure 1: The vertex representation `Vrep` of a 5-gon in the plane.

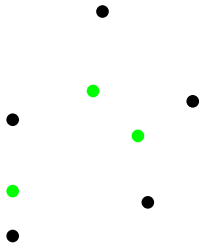


Figure 2: A redundant vertex representation $\overset{*}{\texttt{Vrep}}$ of a 5-gon in the plane. The green points are redundant.

2. Any convex set can be obtained by intersecting all its support half-spaces. The collection of support half-spaces is infinite. For a polytope, this infinite collection can be reduced to a finite one containing support half-spaces of its facets only. So, any polytope P has a finite hyperplane representation $\texttt{Hrep}\,P$ (Fig. 3). It is assumed that the descriptions of the support hyperplanes $\mathrm{hp}(n_i, c_i)$ include the normals n_i outer with respect to P (as it is shown in Fig. 3). Despite such a representation also does not include any neighborhood information, it is, nevertheless, more substantial than the vertex one because the neighborhood information can be restored in an easier way. The hyperplane representation is sufficient for many algorithms (however, for many, it is still not sufficiently substantial).

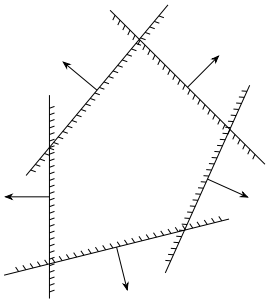


Figure 3: The hyperplane representation `Hrep` of a 5-gon in the plane.

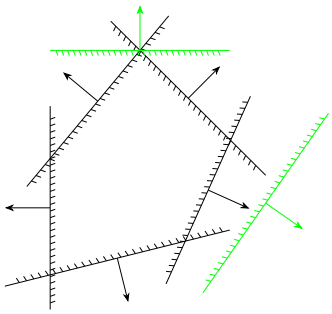


Figure 4: A redundant hyperplane representation $\overset{*}{\texttt{Hrep}}$ of a 5-gon in the plane. The green hyperplanes are redundant.

Also, a redundant hyperplane representation $\overset{*}{\texttt{Hrep}}\,P$ of a polytope P can be considered. Such a representation contains not the support hyperplanes of all facets only, but also some additional hyperplanes, whose hyperspaces do not participate efficiently in the intersection producing the polytope (Fig. 4). Such a representation can appear from an algorithm operating with hyperplanes in the hyperplane representation of a polytope.

Procedures for excluding redundant hyperplanes from a redundant hyperplane representation of a polytope are very important in computational geometry. Again, such procedures in the plane are relatively simple and efficient and, vice versa, quite complicated in higher dimensions.

3. The most complete description of a polytope P is a structure called *face lattice* $\texttt{FLrep}\,P$.

The face lattice of a polytope P is a complete lattice, whose elements one-to-one correspond to all faces of all dimensions of the polytope P. The partial order in this lattice is "to be a subface".

An example of the face lattice for a three-dimensional cube (Fig. 5) is given in Fig. 6.

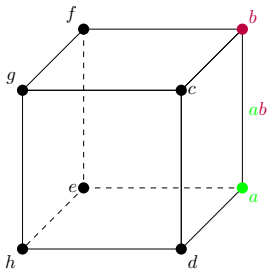


Figure 5: A three-dimensional cube.

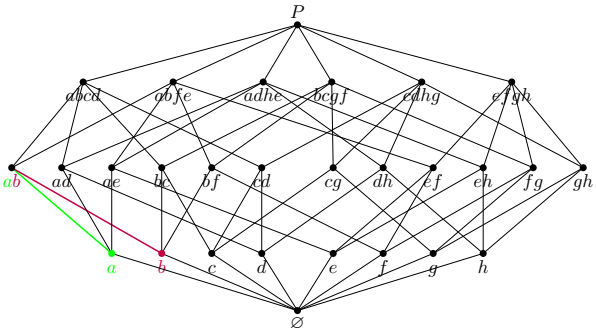


Figure 6: The face lattice of a three-dimensional cube. The color marks correspond to Fig. 5.

The representation $\texttt{FLrep}\,P$ contains two others described above. Indeed, the facets of P correspond to the lattice elements from the layer just below the maximal element corresponding to the entire P. So, descriptions of hyperplanes can be constructed on the basis of these elements of the lattice. In the same way, the vertices of P correspond the elements from the layer just above the minimal element corresponding to the empty set. So, the transformations $\texttt{FLrep}\,P \to \texttt{Hrep}\,P$ and $\texttt{FLrep}\,P \to \texttt{Vrep}\,P$ are trivial.

However, reverse transformations from $\texttt{Hrep}\,P$ or $\texttt{Vrep}\,P$ to $\texttt{FLrep}\,P$ are much less simple. The procedures $\texttt{Vrep}\,P \to \texttt{FLrep}\,P$ and $\overset{*}{\texttt{V}}\texttt{rep}\,P \to \texttt{FLrep}\,P$ can be provided by a convex hull construction. The transformations $\texttt{Hrep}\,P \to \texttt{FLrep}\,P$ and $\overset{*}{\texttt{H}}\texttt{rep}\,P \to \texttt{FLrep}\,P$ are even less evident and some their version is one of the results of this paper described in Subsection 5.3.

5 Geometric operation algorithms

5.1 Convex hull construction algorithm

The first algorithm that needs to be implemented is the procedure for constructing the convex hull of a set of points in $\mathbb{R}^d$. The algorithm from article [22] has been taken as a basis. The main idea of the algorithm is as follows:

1. Construct some $(d-1)$-face of the convex hull;
2. Construct all $(d-2)$-subfaces of the current $(d-1)$-face F; it may be necessary to move into the subspace of the face F and apply recursively the same algorithm for constructing convex hull;
3. Go from the current $(d-1)$-face F to an adjacent $(d-1)$-face through one of the constructed $(d-2)$-subfaces of F.

Repeat steps 2 and 3 until the entire polytope is constructed.

The complexity of the convex hull construction using the gift-wrapping method is

$$T(S,k) = O\big(k^2 d f_{d-1} + dn f_{d-1} + kd f_{d-2} \log f_{d-2} + \sum_i T(F'_i, k-1)\big),$$

where S is the initial set of points, k is the dimension of the space, $n = |S|$ is the number of points in the set, f_k is the number of k-faces in the polytope, and F_i' is the set of points lying on the i-th facet of the polytope.

Boundary conditions for this recursive formula is $T(S, 2) = O(n \log n)$. We use a two-dimensional convex hull algorithm based on the Graham's scan when the set of points to be processed is in the space $\mathbb{R}^2$. In the original work, it was proposed to perform one more step of the recursive part of the algorithm, and in the one-dimensional space, construct the convex hull by finding the minimum and maximum points.

The signature of the implemented algorithm looks as follows:

$$\text{conv} : \mathtt{V\overset{*}{rep}}\, A \to \mathtt{Vrep}\, A, \mathtt{Hrep}\, A, \mathtt{FLrep}\, A.$$

As a result of this procedure, all three polytope representations can be formed. If necessary, it is possible to limit the output to only the vertex representation without building the face lattice, which slightly speeds up the function execution; "slightly" because inner data structures actually store the face lattice of the polytope to be constructed.

5.2 Minkowski sum algorithm

The Minkowski sum (algebraic sum) of two sets A and B in $\mathbb{R}^d$ is defined as

$$A + B = \left\{ a + b \in \mathbb{R}^d \,|\, a \in A, b \in B \right\}.$$

The algorithm for constructing the Minkowski sum of two convex polytopes has been taken from works [5, 6]. The main idea of the algorithm is to iterate through all pairs of faces of the summands and check quite simply whether their Minkowski sum is a face of the resulting polytope or not. The face lattice is constructed from top to bottom.

The signature of the implemented algorithm is as follows:

$$\mathtt{sum} : (\mathtt{FLrep}\, A, \mathtt{FLrep}\, B) \to \mathtt{FLrep}(A + B).$$

The enumeration of face pairs of the summands has been somewhat optimized, but the asymptotic complexity remains the same as when enumerating all pairs of faces. Therefore, the complexity of the algorithm is $O\big(\mathtt{solveLS} \cdot |\,\mathtt{FLrep}\, A| \cdot |\,\mathtt{FLrep}\, B|\big)$ where $\mathtt{solveLS}$ is the complexity of solving a system of linear algebraic equations. Typically, it is $O(d^3)$, but can be faster. There are subtle algorithms with asymptotic complexity $O(d^{2.373})$ [15]. The symbols $|\,\mathtt{FLrep}\, P|$ denotes the number of nodes in the face lattice of the polytope P, that is, the total number of its faces.

However, we only need the $(d-1)$-level of the face lattice of the resulting polytope, which corresponds to the layer of facets. Since the algorithm constructs the face lattice from top to bottom, we stop the process after building the $(d-1)$-level, obtaining $\mathtt{Hrep}(A+B)$. This speeds up the algorithm, but in general, the entire complexity stays the same.

5.3 Minkowski difference algorithm

The third algorithm required for the implementation of the procedure for constructing time sections of maximal stable bridges using the second Pontryagin's method is the algorithm for computing the Minkowski difference (the geometric difference) of two convex polytopes.

The Minkowski difference of two sets A, $B \subset \mathbb{R}^d$ can be defined in two equivalent ways:

$$A \overset{*}{-} B = \left\{ x \in \mathbb{R}^d \,|\, x + B \subset A \right\} = \bigcap_{b \in B} \big(A + (-b)\big).$$

The first definition states that it is the largest set C such that $C+B \subset A$. The second definition says that the geometric difference is the intersection of copies of set A obtained by all possible shifts by elements of set B.

If B is a polytope, the latter representation can be rewritten as

$$A \stackrel{*}{-} B = \bigcap_{b \in \mathtt{Vrep}\, B} \big(A + (-b)\big).$$

No algorithms suitable for our purpose for computing the geometric difference of polytopes have been found in the literature, so the procedure for constructing the Minkowski difference has been developed by the authors themselves. The ideas from the work [3] are taken as a basis.

In our terms, it can be described as $\mathtt{diffBasic} : (\mathtt{Hrep}\, A, \mathtt{Vrep}\, B) \to \mathtt{H\overset{*}{r}ep}\, C \to \mathtt{Hrep}\, C$. To perform the transformation $\mathtt{H\overset{*}{r}ep}\, C \to \mathtt{Hrep}\, C$, that work proposes to use a procedure based on the Fourier-Motzkin elimination algorithm for redundant inequalities. Its complexity is $O\big(4(m/4)^{2^d}\big)$, where m is the number of inequalities and d is the number of variables to be eliminated. One can see that the complexity is polynomial on the number of inequalities and double exponential on the dimension of the space. Even for modern computing power, this algorithm becomes too heavy at relatively small values of m.

The problem of eliminating redundant inequalities is a well-known issue in computational geometry, for which other approaches also exist. One of these is proposed in book [8]. This algorithm has a complexity $O\big(m \times \mathtt{LP}(d,s)\big)$, where m is the total number of inequalities, d is the dimension of the space, and s is the number of non-redundant inequalities.

However, in the next iteration of the second Pontryagin's method (4), it is necessary to add the polytope obtained from the previous iteration with the polytope of the first player's actions (see (4)). Now, only a procedure is available, which demands the summands to be represented as face lattices. Therefore, even if to take the algorithm of difference from [3], an additional procedure $\mathtt{Hrep}\, C \to \mathtt{FLrep}\, C$ is required.

The authors are capable to do the latter transformation only by the GiftWrapping convexification procedure of the set of points $\mathtt{V\overset{*}{r}ep} \,/\, \mathtt{Vrep} \to \mathtt{FLrep}$. This, in its turn, requires the transformation from $\mathtt{H\overset{*}{r}ep}$ or $\mathtt{Hrep}$ to $\mathtt{Vrep}$ or, at least, $\mathtt{V\overset{*}{r}ep}$. Instead of a chain $\mathtt{H\overset{*}{r}ep} \to \mathtt{Hrep} \to \mathtt{Vrep}$, we propose to use a direct passage $\mathtt{H\overset{*}{r}ep} \to \mathtt{Vrep}$.

Therefore, we propose the following algorithm for computing the geometric difference.

$$\mathtt{diff} : (\mathtt{Hrep}\, A, \mathtt{Vrep}\, B) \to \mathtt{H\overset{*}{r}ep}\, C \to \mathtt{Vrep}\, C \to \mathtt{FLrep}\, C$$

Minkowski difference. First stage
$\mathtt{diffStg1}(\mathtt{Hrep}\, A, \mathtt{Vrep}\, B) \to \mathtt{H\overset{*}{r}ep}\, C$
This subprocedure is taken from [3].

1. For each $\mathrm{hp}(n_i, c_i) \in \mathtt{Hrep}\, A$, find the vertex $v_i^* \in \operatorname{Argmax}_{v \in B} \langle v, n_i \rangle$.
2. Update the hyperplane $\mathrm{hp}(n_i, c_i)$ by subtracting from c_i the value $\langle v_i^*, n_i \rangle$. The resultant set of hyperplanes is just $\mathtt{H\overset{*}{r}ep}\, C$.

Minkowski difference. Second stage. Naive algorithm
$\mathtt{diffStg2\_naive} : \mathtt{H\overset{*}{r}ep}\, A \to \mathtt{Vrep}\, A$

1. $\mathtt{Vrep}\, A \leftarrow \emptyset$;
2. Iterate over all sets of d hyperplanes from $\mathtt{H\overset{*}{r}ep}\, C$. For each set, solve the system of linear equations obtaining the candidate point v to be a vertex of C. If the system has a unique solution and $v \in \mathrm{hp}^-(n,c) \cup \mathrm{hp}(n,c)$ for all $\mathrm{hp}(n,c) \in \mathtt{H\overset{*}{r}ep}\, C$, then the obtained point v indeed is a vertex of C: $\mathtt{Vrep}\, A \leftarrow v$.

The advantage of the naive algorithm is that it can work with a redundant representation of the polytope. However, its computational complexity is extremely high: $O\big(d^3\binom{H}{d}\big) = O\big(\frac{H^d}{d!}\big)$, where $H = |\overset{*}{\mathtt{Hrep}}\, A|$. That is, the complexity is polynomial on the number of facets of the minuend polytope, which can be very large even in three dimensional space.

Minkowski difference. Second stage. Geometric algorithm

$\mathtt{diffStg2\_more\_optimal} : \overset{*}{\mathtt{Hrep}}\, A \to \mathtt{Hrep}\, A$

The idea of this algorithm has a clear geometric sense. At first, find some vertex of C. For example, by the naive approach. Then, knowing what hyperplanes are incident at this vertex, the 1-faces (that is, one-dimensional edges of the polytope C) incident to this vertex can be found. Along them, the procedure can go to neighbor vertices repeating there the search of 1-faces and spreading. Formally, this algorithm can be written as follows.

For brevity, denote $\mathcal{H} \triangleq \overset{*}{\mathtt{Hrep}}\, A$, $H = |\mathcal{H}|$.

Stage 1. Find any vertex v of the polytope C and a set of hyperplanes $\mathcal{H}_v$ those are incident to it. The search of vertex can be done by solving a linear programming problem once or by using the naive procedure until one point is found. The search of hyperplane set is done simultaneously by the naive algorithm of by substituting v into all hyperplanes from $\mathcal{H}$. The complexity depends on the certain procedure used for this construction and is denoted $O(\mathtt{VInit})$.

Stage 2. Find the remaining vertices of the polytope. We suggest do it by width-first search over the graph whose vertices and edges are the vertices and 1-faces of the polytope C.

Create a queue of pairs (vertex z, set of hyperplanes $\mathcal{H}_z$ incident to the vertex z). Initialize it with the data found in Stage 1.

1. While the queue is not empty, take a pair $(z, \mathcal{H}_z)$ from the queue.
2. Iterate over all possible sets $\mathcal{H}'_z$ of $d-1$ hyperplanes from $\mathcal{H}_z$. For each such a set $\mathcal{H}'_z$:
   (a) Build the direction vector l of the one-dimensional line obtained by intersection of hyperplanes from $\mathcal{H}'_z$. This can be done by finding a vector orthogonal to all normal vectors of the hyperplanes in $\mathcal{H}'_z$, what requires to solve an underdetermined system of linear equations.
   (b) Check the vector l. For each hyperplane $\mathrm{hp}(n, c) \in \mathcal{H}_z$, compute the scalar product $\langle l, n\rangle$. If it equals zero, ignore this hyperplane and continue to the next hyperplane. The first non-zero scalar product determines the orientation of l: if it is positive, reverse the direction of l. For all subsequent non-zero products, ensure that they are non-negative.
   (c) If the vector l passes the filtering, that is, for all hyperplanes $\mathrm{hp} \in \mathcal{H}_z$ it is not directed towards hp^+ (outside the polytope C), it together with the current vertex z, defines a ray containing an 1-face of C. If the vector fails the check, the current set $\mathcal{H}'_z$ does not define a polytope edge, so move on to the next set of hyperplanes, that is, go to step 2.
   (d) Find all hyperplanes from $\mathcal{H}\backslash\mathcal{H}_z$ closest to z in the direction of l:

$$J = \operatorname{Arg\,min}\left\{ t_i = \frac{c_i - \langle n_i, z\rangle}{\langle n_i, l\rangle} \;\middle|\; \mathrm{hp}(n_i, c_i) \in \mathcal{H}\backslash\mathcal{H}_z,\ t_i > 0 \right\}.$$

   Also, a set $\mathcal{H}_\parallel$ is collected including all hyperplanes from $\mathcal{H}\backslash\mathcal{H}_z$, which contain the constructed line. In the case of redundancy of the set $\mathcal{H}$, such hyperplanes might exist; an edge might be defined by more than $(d-1)$ hyperplanes.

The set J together with the sets $\mathcal{H}'_z$ and $\mathcal{H}_\parallel$ defines the new vertex.

The algorithm for constructing J is as follows: iterate through $\mathcal{H}\backslash\mathcal{H}_z$ keeping the current minimum t^* of t_i and the set of hyperplanes, whose t_i equals t^*.

i. Initially, set $t^* = +\infty$, $J = \emptyset$, $\mathcal{H}_\parallel = \emptyset$.
ii. For the next hyperplane $\mathrm{hp}_i = \mathrm{hp}(n_i, c_i)$, compute its t_i. If the denominator of the fraction is zero, that is, if the plane is parallel to the direction vector l, assume $t_i = +\infty$; this plane does not participate in the search of the minimum. In this situation, if $z \in \mathrm{hp}_i$, that is, if the plane contains the constructed one-dimensional line; add hp_i to $\mathcal{H}_\parallel$. If all hyperplanes are enumerated, go to 2(e).
iii. If $t_i \leqslant 0$ or $t_i > t^*$, proceed to the next hyperplane, that is, go to 2(d)ii.
iv. If $t_i = t^*$, add hp_i to J and go to 2(d)ii.
v. If $t_i < t^*$, update $t^* \leftarrow t_i$, $J = \{\mathrm{hp}_i\}$ and compute the new candidate vertex $z^* = z + t^* \cdot l$. If z^* is already in the set of known vertices, stop the algorithm with no new vertex found; go to 2(f). Otherwise go to 2(d)ii.

(e) Add the found pair $(z^*, \mathcal{H}'_z \cup J \cup \mathcal{H}_\parallel)$ to the queue as a new polytope vertex and the set of hyperplanes incident to it. Add z^* to the set of vertices constructed already.
(f) Pass to spreading along other 1-faces, go to 2.

3. Go to spreading from other vertex, go to 1.

The complexity of the algorithm is $O\big(\texttt{VInit} + V \cdot \binom{k}{d-1} \cdot d^3 \cdot kd \cdot Hd\big)$. Here,

- $O(\texttt{VInit})$ is the complexity of the procedure for finding some vertex of the polytope C;
- V is the number of vertices of the polytope C; it is the number of iterations of the loop at the step 1;
- k is the maximal number of facets incident to a vertex;
- $\binom{k}{d-1}$ is the number of sets $\mathcal{H}'_z$ enumerated during search of 1-faces for a vertex; it is the number of iterations of the loop at the step 2;
- $O(d^3)$ is the complexity of constructing a direction vector of next 1-face;
- $O(kd)$ is the complexity of checking whether next found vector l indeed defines an 1-face (computation of k scalar products with normals of hyperplanes from $\mathcal{H}'_z$);
- $O(Hd)$ is the complexity of constructing the set J at the step 2(d); essentially, this construction enumerates (almost) all hyperplanes from $\mathcal{H}$ and computes some scalar products.

Usually, $k \ll H$, what provides significantly better performance compared to the naive procedure.

6 Realization of computational library

Before the algorithms have been implemented, a basic library for geometric computations has been developed. Necessary geometric primitives and corresponding functions have been implemented within this library.

6.1 General library architecture

All computations in the library are performed using floating point numbers. The library is designed to work with a variety of such types. More specifically, the type must support the following interfaces:

```
struct, INumber<TNum>, ITrigonometricFunctions<TNum>, IPowerFunctions<TNum>,
IRootFunctions<TNum>, IFloatingPoint<TNum>, IFormattable
```

which guarantee that necessary classes of functions are available for numbers of this type. Thus, the library is *generic* with respect to the numerical type used, which is denoted by `TNum`.

6.2 On calculation accuracy

Since the library operates with floating point numbers, the problem of comparison of these numbers arises in a natural way. Within the library, all comparisons are performed with a predefined precision ε.

Let x and y be two numbers. It is assumed that $x < y$ if $x - y < \varepsilon$, $x > y$ if $x - y > \varepsilon$, and otherwise x and y are considered equal. In this way, the comparison method `CompareTo(TNum` x, `TNum` y) of the interface `IComparable<TNum>` is defined for two numbers x and y of the type `TNum`.

An alternative variant of the basic numeric type can be a type based on rational arithmetic. Its advantage is that calculations are performed "precisely" meaning that there is almost no loss of precision during basic arithmetic operations unlike floating point types. However, these types have significantly lower performance. Now, the library has not been tested with rational numerical types. The possibility of including rational arithmetic into the library is being considered for the future.

Our algorithms require data structures with fast insertion, deletion, and access to elements. Of course, they are not arrays or lists, but either hash table based structures, or tree based structures.

The usage of the hash based structures needs computation of hash (an integer) for each element inserted into the structure. If two objects are considered equal, then their hashes must be the same. The hash can be equal for non-equal objects, such a situation is called *collision*. But as less collisions appear, the more efficient the hash is and more fast insertion, deletion, and access operations are.

The tree based structures arrange elements using a comparison function. Potentially, hash-based structures are faster. That is why we chose them initially.

The main object, for which the hash should be computed, is a floating point number. Initially, for an arbitrary $x \in \mathbb{R}^d$, the following hash function was proposed: `Round`$(\frac{x}{\varepsilon})$. For example, let $x = 0.1234567$ and $\varepsilon = 10^{-4}$, then the hash would be 1235. However, this function is unsatisfiable: let $y = 0.1234444$, the hash is 1234. Thus, there are two different hashes, but the objects are equal for a given precision ε.

Attempts have been made to construct other hash functions overcoming the mentioned problem, but they were unsuccessful. The main problem is that transitivity of equality is violated in the terms of comparison by a precision: $1 \approx 1 - \varepsilon$ and $1 \approx 1 + \varepsilon$, but $1 - \varepsilon \not\approx 1 + \varepsilon$, where "$\approx$" is equality by a precision.

So, eventually, we started to use tree-based structures. Although they offer lower performance, they are much more stable. As a result, the current implementation of the library uses tree based structures when the elements in the case of sets or keys in the case of dictionaries (maps) are floating point numbers. In the basic library of C#, these structures are `SortedSet<T>` and `SortedDictionary<TKey,TValue>`, respectively. This does not solve the problem of equality transitivity absence, but makes algorithms more stable.

6.3 Basic computing library

Consider the main data structures and functions provided by the library.

6.3.1 Multidimensional vector class

A vector $x \in \mathbb{R}^d$ is represented as an array of its coordinates. Two vectors $x, y \in \mathbb{R}^d$ are compared in the lexicographic order of their coordinates, where component comparison is performed using `CompareTo` for numbers of the basic numeric type `TNum`. The class supports vector addition and subtraction, multiplication by a scalar, and division by a non-zero number. Functions for finding the length, normalization, and computing the inner and outer products are also defined.

6.3.2 Matrix class

Matrices are stored in a "dense" representation in a two-dimensional array. The class defines the operations of matrix addition and subtraction, multiplication by a scalar, division by a non-zero number, as well as the ability to get a row or column by index. These operations on matrices are sufficient for the algorithms under consideration.

6.3.3 Linear basis class

A linear basis $\mathcal{B}$ is represented by a $k \times d$ matrix, where $0 \leqslant k \leqslant d$, and its columns form a set of linearly independent vectors. The basis is stored in an orthonormalized form.

The class supports operations such as projecting a vector onto the subspace spanned by $\mathcal{B}$, orthonormalizing a vector with respect to the basis $\mathcal{B}$, constructing the orthogonal complement basis of $\mathcal{B}$, and adding vectors to the basis. Vectors are added one at a time, and if a vector already lies within the subspace of the current basis, it is not added. A key operation is the orthonormalization of a vector against the current basis. Initially, the Gram–Schmidt orthogonalization algorithm was used, but it was soon discovered that it is unstable and lead to significant loss of calculation accuracy. Therefore, a reflection algorithm, which does not have such drawbacks, was implemented.

Alternatively, the basis can be specified as a list of vectors; however, the reflection algorithm relies on matrix operations, and converting data between formats is inefficient.

6.3.4 Affine basis class

An affine basis consists of a vector o and a linear basis $\mathcal{B}$. The operations are similar to those for a linear basis, but taking into account its affine structure, meaning the work is done with the vector $x - o$. The class supports operations such as projecting a vector onto the subspace spanned by $\mathcal{B}$, orthonormalizing a vector with respect to the basis $\mathcal{B}$, constructing the orthogonal complement basis of $\mathcal{B}$, and adding vectors to the basis.

6.3.5 Face lattice class

The data structure that stores the face lattice is introduced. It holds a reference to the topmost node of the face lattice as well as a list of nodes, where the i-th position contains all nodes of the i-th layer. A face lattice node stores a strictly interior point and an affine basis of the corresponding face as well as lists of immediate parent nodes and child nodes in the sense of the face lattice order.

6.3.6 Auxiliary points class

During the convex hull construction algorithm, it is often necessary to move into the subspace of a facet and perform the convex hull construction there. After this, one needs to return to the original space. That is why the class of auxiliary points is introduced as a heritor of the point class. It stores a reference to the parent point, from which this point is projected. When returning from the subspace, no calculations are performed; instead, the current point is simply replaced by its parent.

6.4 Stable bridge construction algorithm

The following algorithm is proposed.

$$\texttt{doNextSection}(\texttt{FLrep}\,W_{i-1}, \texttt{FLrep}\,\mathcal{P}_i, \overset{*}{\texttt{Vrep}}\,\mathcal{Q}_i) \to \texttt{FLrep}\,W_i,$$

$\mathcal{P}_i = (-\Delta)D(t_i)P$, $\mathcal{Q}_i = \Delta E(t_i)Q$
Step by step:

1. $\texttt{sum}(\texttt{FLrep}\,W_{i-1}, \texttt{FLrep}\,\mathcal{P}_i) \to \texttt{FLrep}\,S_i$
2. $\texttt{FLrep}\,S_i \to \texttt{Hrep}\,S_i$
3. $\texttt{diff}(\texttt{Hrep}\,S_i, \texttt{Vrep}\,\mathcal{Q}_i) \to \overset{*}{\texttt{Hrep}}\,W_i$
4. $\overset{*}{\texttt{Hrep}}\,W_i \to \texttt{Vrep}\,W_i$
5. $\texttt{conv}(\texttt{Vrep}\,W_i) \to \texttt{FLrep}\,W_i$.

Let us explain how the polytopes $\mathcal{P}_i$ and $\mathcal{Q}_i$ are obtained. Initially, the control constraint polytopes of the players are located in their spaces $\mathbb{R}^p$ and $\mathbb{R}^q$. The fundamental Cauchy matrix $X(T,t)$ for the system $dX(T,t)/dt = A(t)X(T,t)$, $X(T,T) = E$, is integrated. From it the variable change matrix $X_{s_1,s_2,\ldots,s_d}(T,t)$ is created.

After that, the matrices $D_i = X(T,t_i)B(t_i)$ and $E_i = X(T,t_i)C(t_i)$ are calculated. Then, the capabilities of the first player at the ith step are computed as $\mathcal{P}_i = (-\Delta)D_iP$ and of the second player as $\mathcal{Q}_i = \Delta E_iQ$. For $\mathcal{P}_i$, all vertices of P are multiplied by $D(t_i)$, and then the convex hull construction algorithm is performed producing $\texttt{FLrep}\,\mathcal{P}_i$. For $\mathcal{Q}_i$, only projection is performed. Since we need $\texttt{Vrep}\,\mathcal{Q}_i$ only, we can take $\overset{*}{\texttt{Vrep}}\,\mathcal{Q}_i$. It is faster to work with redundant unconvexified set of points, than to try run the GiftWrapping algorithm.

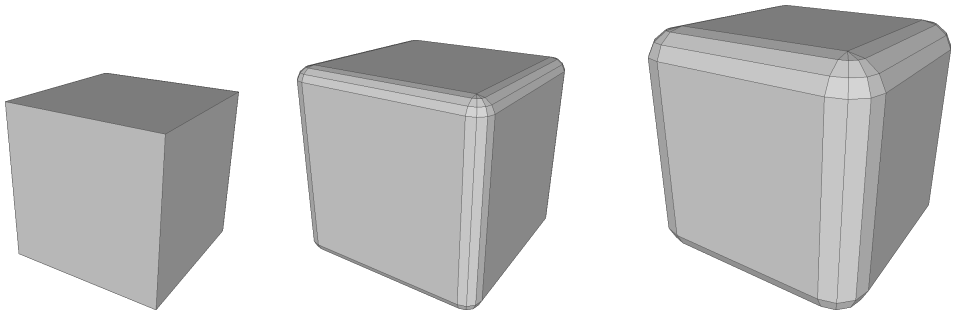
7 Examples

The computations were performed on a computer running Windows 11 64-bit equipped with an Intel(R) Core(TM) i5-10400F CPU 2.90 GHz max and 32 GB of 2400 MHz RAM.

7.1 Simple motion

Consider the following differential game with a three-dimensional phase vector:

$$\begin{aligned}
&\dot{x} = u + v,\\
&\mathbf{M} = [-1,1] \times [-1,1] \times [-1,1],\\
&t \in [t_0, T] = [0,4], \quad x \in \mathbb{R}^3,\\
&u \in \mathbf{B}_2(0,1), \quad v \in \mathbf{B}_2(0,0.9),
\end{aligned}$$



(a) $t = 4$ (b) $t = 2$ (c) $t = 0$

Figure 7: The game with dynamics of simple motion. A side view of three-dimensional time sections of the 4-dimensional maximal stable bridge.

where $\mathbf{B}_2(0, R)$ is the Euclidean ball with the center at the origin and radius R. The program uses some approximation of these balls by polytopes.

In Figure 7, one can see the evolution of the three-dimensional time sections of the corresponding maximal stable bridge. Some section are drawn. All three-dimensional figures have been visualized in `MeshLab` after preparation in a popular 3D-format `OBJ` on the basis of numerical results produced by the computational program.

Since the first player has the dynamic advantage, that is, $\mathcal{Q}_i \subset \mathcal{P}_i$ for all i, the time section grows in the backward time. All three figures are presented in the same scale.

7.2 Mass point

The second example is connected with a classic model differential game describing a conflict controlled mass point in a line:

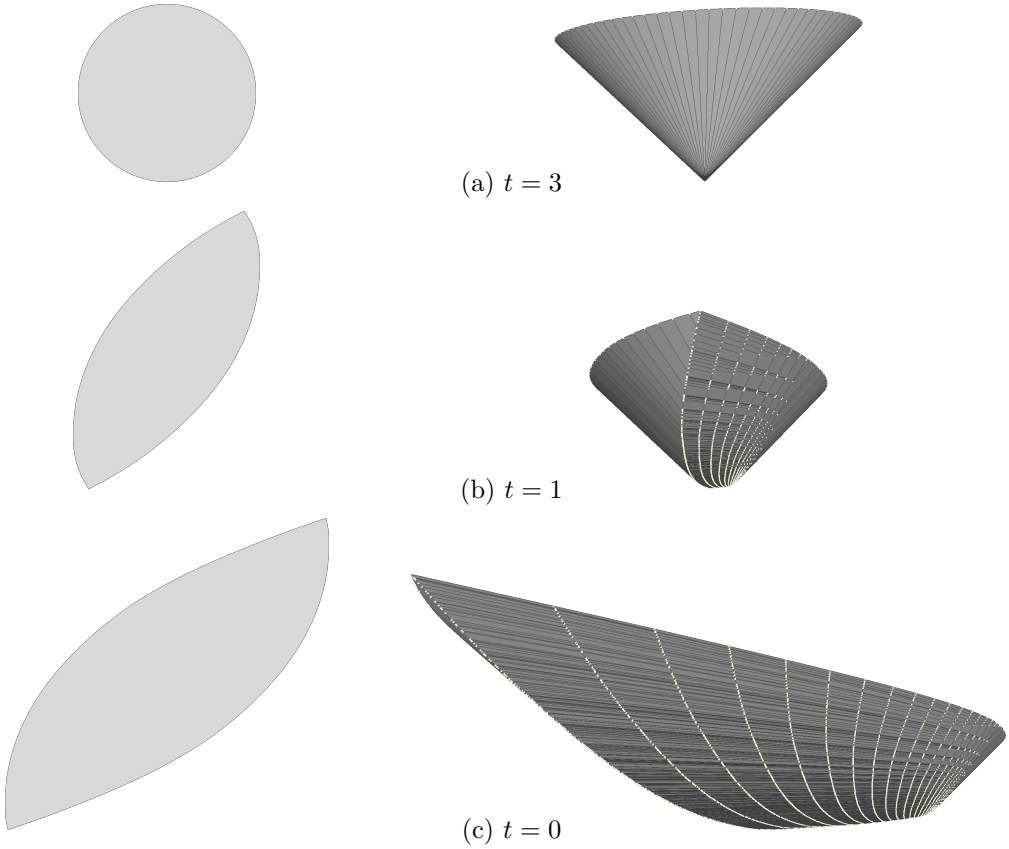
$$\begin{aligned}
&\dot{x}_1 = x_2 + v,\\
&\dot{x}_2 = u,\\
&t \in [t_0, T] = [0, 3], \quad x = (x_1, x_2)^{\mathrm{T}} \in \mathbb{R}^2,\\
&u \in [-1, 1], \quad v \in [-0.9, 0.9],\\
&\varphi\big(x(T)\big) = \|x(T)\| \to \min_u \max_v .
\end{aligned}$$

Here, x_1 represents the coordinate of the point in the line, and x_2 represents its velocity. There is the terminal payoff function corresponding to the objective of the first player to bring the point to the origin at the terminal instant with a small velocity.

This is a game with a two-dimensional phase vector. It is formally out of the class, to which the algorithm worked out is directed. However, such a game can be transformed into a three-dimensional game with a terminal target set.

To do this, a new variable x_3 is introduced, which represents the value of the payoff function on a trajectory. It is known that for games with the terminal payoff, this value does not change along the optimal trajectory. So, the dynamics of the new coordinate is very simple: $\dot{x}_3 = 0$.

So, the new coordinates are (old phase vector, payoff value). The epigraph epi φ of the payoff function is the terminal target set $\mathbf{M}$. This set is unbounded, so only some of its bounded part $\overline{\mathbf{M}} = \mathbf{M} \cap \{x_3 \leqslant 2\}$ is considered as the target set.

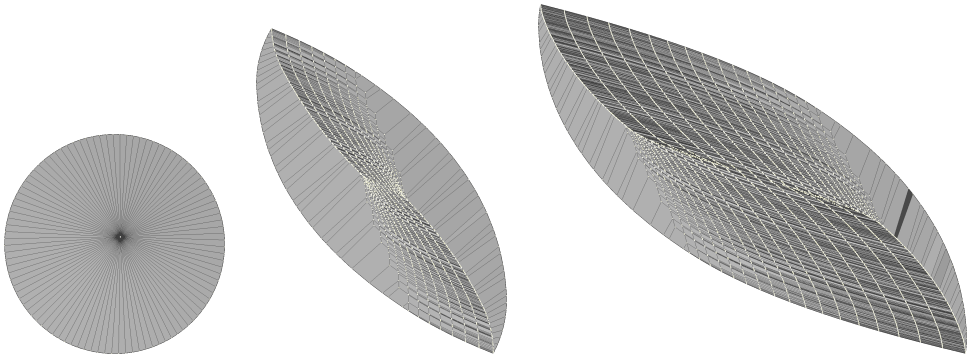


(a) $t = 3$

(b) $t = 1$

(c) $t = 0$

Figure 8: Time section of the maximal stable bridges for the mass point. *At the left:* the maximal stable bridge constructed in the original 2D game from the level set of the payoff 2. *At the right:* the maximal stable bridge constructed in the 3D game from a part of the epigraph of the payoff as the terminal target set. All 3D-figures have the same point of view.



(a) $t = 3$ (b) $t = 1$ (c) $t = 0$

Figure 9: The mass point. A bottom view of the three-dimensional time sections.

The new game is described as follows:

$$\begin{aligned}
&\dot{x}_1 = x_2 + v,\\
&\dot{x}_2 = u,\\
&\dot{x}_3 = 0,\\
&t \in [t_0, T] = [0, 3], \quad x = (x_1, x_2, x_3)^{\mathrm{T}} \in \mathbb{R}^3,\\
&u \in [-1, 1], \quad v \in [-0.9, 0.9],\\
&x(T) \in \overline{\mathbf{M}}.
\end{aligned}$$

For computations, the circular cone of $\overline{\mathbf{M}}$ is approximated by a right pyramid with 100 side facets. The time sections have been constructed for instants from a uniform grid with the time step $\Delta = 0.1$. Number of vertices and faces of the polytope of the time section at $t = 0$ equal 9690 and 19376, respectively. The computations took about a half of an hour.

Figure 8 shows comparison of time sections of stable bridges in the two- and three-dimensional variants of the game. At the left, time sections of the maximal stable bridges computed for the target set taken as the level set of the payoff corresponding to the value 2. They are obtained as a top view of the time sections of the maximal stable bridge in the three-dimensional game. The two-dimensional sets coincide well (in numerical comparison, not a visual one only) with the ones obtained by the algorithm for two-dimensional games.

At the right, one can see a side view of the section in the three-dimensional game. Thin black and thick white lines show boundaries of faces of the polytope. The scale and the view point are the same for all three 3D-figures.

In Figure 9, images of the time sections are given from the bottom (from the negative direction of the axis x_3). These figures demonstrates the structure of the faces appearing during the evolution of the time sections.

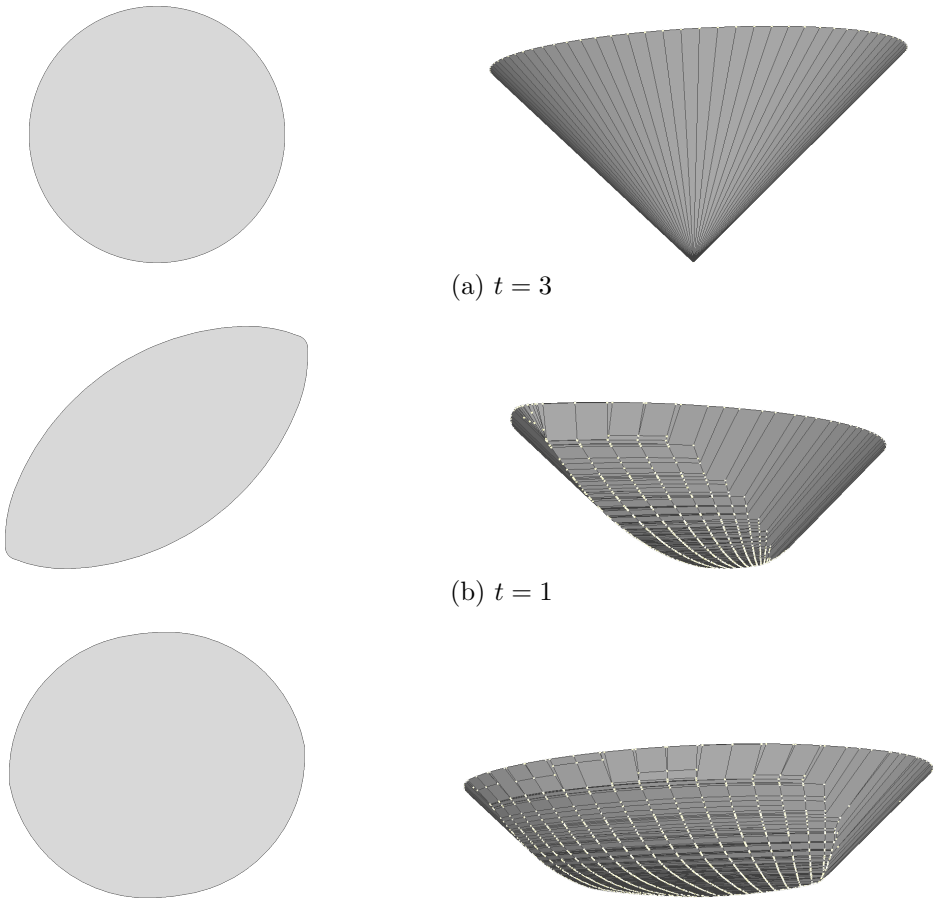
7.3 Oscillator

Another classic model differential game is the conflict controlled oscillator. It has the dynamics of the mathematical pendulum, where the first player affects the acceleration of the pendulum, whereas the second player has its control in the velocity row:

$$\begin{aligned}
&\dot{x}_1 = x_2 + v,\\
&\dot{x}_2 = -x_1 + u,\\
&t \in [t_0, T] = [0, 3], \quad x = (x_1, x_2)^{\mathrm{T}} \in \mathbb{R}^2,\\
&u \in [-1, 1], \quad v \in [-0.9, 0.9],\\
&\varphi\big(x(T)\big) = \|x(T)\| \to \min_u \max_v .
\end{aligned}$$

Again, x_1 is the coordinate of the pendulum, that is, its deviation from the vertical line (which is supposed to be small to consider a linear dynamics), and x_2 is the velocity.

The game has the same terminal payoff as in the previous example. Thus, the objectives of the players are the same. The first one tries to calm down the system, bringing it as close to the origin at the terminal instant as possible. The second player, vice versa, tries to swing the pendulum away.

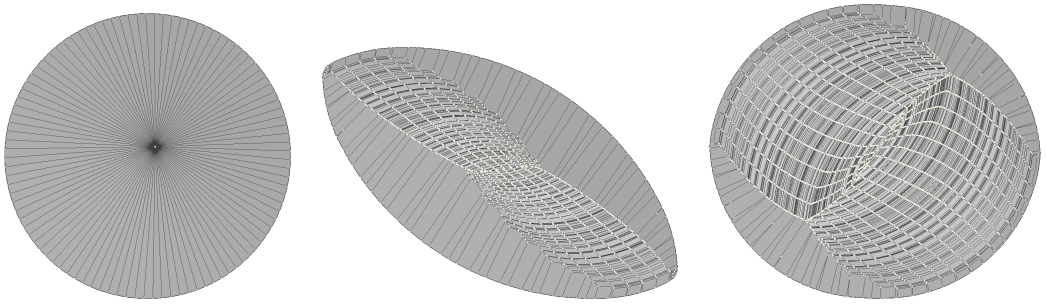


(a) $t = 3$

(b) $t = 1$

(c) $t = 0$

Figure 10: Time section of the maximal stable bridges for the oscillator. *At the left:* the maximal stable bridge constructed in the original 2D game from the level set of the payoff 2. *At the right:* the maximal stable bridge constructed in the 3D game from a part of the epigraph of the payoff as the terminal target set. All 3D-figures have the same point of view.



(a) $t = 3$ (b) $t = 1$ (c) $t = 0$

Figure 11: The oscillator model game. A bottom view of the three-dimensional time sections.

In the same way, one can pass to a three-dimensional game with the third coordinate corresponding to the payoff:

$$\begin{aligned}
&\dot{x}_1 = x_2 + v,\\
&\dot{x}_2 = -x_1 + u,\\
&\dot{x}_3 = 0,\\
&t \in [t_0, T] = [0, 3], \quad x = (x_1, x_2, x_3)^{\mathrm{T}} \in \mathbb{R}^3,\\
&u \in [-1, 1], \quad v \in [-0.9, 0.9],\\
&x(T) \in \overline{\mathbf{M}} = \operatorname{epi} \varphi \cap \{x_3 \leqslant 2\}.
\end{aligned}$$

For computations, the circular cone of $\overline{\mathbf{M}}$ is approximated by a right pyramid with 100 side facets. The time sections have been constructed for instants from a uniform grid with the time step $\Delta = 0.1$. The number of vertices and faces of the polytope of the time section at $t = 0$ are equal 4354 and 8704, respectively. The computations took about 8 minutes.

In Figure 10, time sections of the maximal stable bridges in the two- and three-dimensional games are given. In contrast to the mass point game, the diameter of the time section does not grow so fast. The scale and point of view of the 3D-figures is the same as in Fig. 8. Again, one has good coincidence of the two-dimensional sets computed by the our and two-dimensional algorithms.

In Figure 11, the structure of the faces of the three-dimensional time sections is visible.

8 Conclusion

The paper addresses linear differential games with a fixed terminal instant, geometric constraints on the players' controls, and a convex terminal target set. Algorithms are implemented for convex hull construction, Minkowski sum and difference, which are necessary for the second Pontryagin's method to construct approximations of time sections of a maximal stable bridge. Existing methods from the literature have been utilized for the convex hull and Minkowski sum, while a new algorithm has been developed for the Minkowski difference. These algorithms have been incorporated into a C# library, and some model examples have been computed.

9 Acknowledgements

The work was performed as part of research conducted in the Ural Mathematical Center with the financial support of the Ministry of Science and Higher Education of the Russian Federation (Agreement number 075-02-2024-1377).

References

[1] M. Bardi, M. Falcone, and P. Soravia. Numerical methods for pursuit-evasion games via viscosity solutions. In M. Bardi, T. Parthasarathy, and T. E. S. Raghavan, editors, *Annals of the International Society of Dynamic Games, Vol. 6: Stochastic and Differential Games*, pages 105–175. Birkhäuser, Boston, 1999.

[2] N. Botkin, K.-H. Hoffmann, N. Mayer, and V. Turova. Computation of value functions in nonlinear differential games with state constraints. In D. Hömberg and F. Tröltzsch, editors, *System Modeling and Optimization. CSMO 2011. IFIP Advances in Information and Communication Technology*, volume 391, pages 235–244, 2013.

[3] N. D. Botkin and E. A. Ryazantseva. An algorithm of constructing the solvability set for linear differential games of high dimension. *Trudy Inst. Mat. i Mekh.*, 2:128–134, 1992. (in Russian).

[4] P. Cardaliaguet, M. Quincampoix, and P. Saint-Pierre. Some algorithms for differential games with two players and one target. *RAIRO-Modélisation-Matématique-et-Analyse-Numérique*, 28(4):441–461, 1994.

[5] S. Das, S. R. Dev, and S. Sarvottamananda. A worst-case optimal algorithm to compute the Minkowski sum of convex polytopes. In A. Mudgal and C. R. Subramanian, editors, *Algorithms and Discrete Applied Mathematics. CALDAM 2021. Lecture Notes in Computer Science*, volume 12601, pages 179–195. Springer, Cham., 2021.

[6] S. Das, S. R. Dev, and S. Sarvottamananda. A worst-case optimal algorithm to compute the Minkowski sum of convex polytopes. *Discrete Applied Mathematics*, 350:44–61, 2024.

[7] P. E. Dvurechensky and G. E. Ivanov. Algorithms for computing Minkowski operators and their application in differential games. *Computational Mathematics and Mathematical Physics*, 54(2):235–264, 2014.

[8] K. Fukuda. *Polyhedral Computation*. Department of Mathematics, Institute of Theoretical Computer Science ETH Zurich, 2020-07-10.

[9] S. A. Ganebny, S. S. Kumkov, S. Le Ménec, and V. S. Patsko. Model problem in a line with two pursuers and one evader. *Dynamic Games and Applications*, 2(2):228–257, 2012.

[10] N. L. Grigorenko, Yu. N. Kiselev, N. V. Lagunova, D. B. Silin, and N. G. Trin'ko. Solution methods for differential games. *Computational Mathematics and Modeling*, 7(1):101–116, 1996.

[11] R. Isaacs. *Differential Games*. John Wiley and Sons, New York, 1965.

[12] E. A. Isakova, G. V. Logunova, and V. S. Patsko. Computation of stable bridges for linear differential games with fixed time of termination. In A. I. Subbotin and V. S. Patsko, editors, *Algorithms and Programs for Solving Linear Differential Games*, pages 127–158. Institute of Mathematics and Mechanics, Ural Scientific Center, Academy of Sciences of USSR, Sverdlovsk, 1984. (in Russian).

[13] N. N. Krasovskii and A. I. Subbotin. *Positional Differential Games*. Nauka, Moscow, 1974. (in Russian).

[14] N. N. Krasovskii and A. I. Subbotin. *Game-Theoretical Control Problems*. Springer-Verlag, New York, 1988.

[15] F. Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, pages 296–303, New York, NY, USA, 2014. Association for Computing Machinery.

[16] V. S. Patsko and V. L. Turova. Numerical study of the homicidal chauffeur game. In D. Bainov, editor, *Proceedings of the 8th International Colloquium on Differential Equations*, pages 363–371. VSP, Utrecht, 1998.

[17] V. S. Patsko and V. L. Turova. Level sets of the value function in differential games with the homicidal chauffeur dynamics. *International Game Theory Review*, 3(1):67–112, 2001.

[18] E. S. Polovinkin, G. E. Ivanov, M. V. Balashov, R. V. Konstantinov, and A. V. Khorev. An algorithm for the numerical solution of linear differential games. *Sbornik: Mathematics*, 192(10):1515–1542, 2001.

[19] L. S. Pontryagin. Linear differential games. 2. *Soviet Math. Dokl.*, 8:910–912, 1967.

[20] B. N. Pschenichnyi and M. I. Sagaidak. Differential games of prescribed duration. *Cybernetics*, 6(2):72–80, 1970.

[21] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proc. Natl. Acad. Sci.*, 93(4):1591–1595, 1996.

[22] G. Swart. Finding the convex hull facet by facet. *J. Algorithms*, 6(1):17–48, 1985.

[23] T. Yoshimura. DoubleDouble, 2023. `github.com/tk-yoshimura/DoubleDouble`.

[24] M. A. Zarkh and A. G. Ivanov. Construction of the value function in the linear differential game with the fixed terminal time. *Trudy Inst. Mat. i Mekh.*, 2:140–155, 1992. (in Russian).