Guaranteed Global Deterministic Optimization and Constraint Programming for Complex Dynamic Problems

Hugo Joudrier, Khaled Hadj-Hamou

Guaranteed Global Deterministic Optimization and Constraint Programming for Complex Dynamic Problems

Hugo Joudrier (student) and Khaled Hadj-Hamou (advisor)

Université de Grenoble, Laboratoire G-SCOP
46, avenue Félix Viallet - 38031 Grenoble Cedex 1 - France
hugo.joudrier@grenoble-inp.fr

Abstract. In this article we focus on particular multi-physics (mechanic, magnetic, electronic...) dynamic problems. These problems contain some differential constraints to model dynamic behaviors. The goal is to be able to solve it with guarantee, meaning to get a proof that all constraints are satisfied (without any approximation caused by binary representations or rounding modes from the unit core computing). The idea of getting guarantees on the arithmetic operations has been introduced via *Interval Arithmetic* [**?**]. Computers become faster gradually, increasing the rate of operations number computable in one time unit. The results computed are often rounded to the nearest representable values, then the global errors are increasing gradually as well without any control over it.

Keywords: Ordinary Differential Equation, Interval Arithmetic, Tube Arithmetic, Constraint Programming, Global Optimization

1 Introduction

In this paper, we try to solve some multi-physics dynamic problems, which are widely represented in the industries to model some complex design problems. The work we present here is motivated by the designing of an electromagnetic contactor. These problems are hardly solvable because of the types of variables (continuous, functionals) and constraints (algebraics, differentials and non-continuous). Usually some approximations over the mathematical model and/or the resolution process of these models are used, then the solution considered can be suboptimal and even worst, not feasible.

Once the multi-physics dynamic problems are presented, we will review some arithmetic aspects (*Interval Arithmetic* (IA) and *Tube Arithmetic* (TA)) used to compute safe operations results and the *Ordinary Differential Equation* (ODE) constraints describing the dynamic behavior of the problem. Then we will introduce the Multi-ODE constraints and an algorithm to solve and propagate these complex dynamical constraints. After a description of the integration in a more global *Interval Branch & Bound Algorithm* (IBBA) to compute the global optimal solution, the last sections will present the results we get and some perspectives.

2 Multi-Physics Dynamic Problem

The work related in this paper is motivated by the optimization of complex systems. It consists in finding the best real inputs $\mathbf{x} \in \mathbb{R}^p$ in order to minimize a cost function (Eq. 1).

$$\textit{Minimize } \mathbf{cost}(\mathbf{x}, \mathbf{u}) \tag{1}$$

where $\mathbf{u} \in (\mathbb{R} \to \mathbb{R})^n$ is the functional variables vector used to compute and control the dynamic behaviors of the problem described by ODEs (Eq. 3) and Multi-ODEs (Eq. 4). The model contains some algebraic constraints $\mathbf{f}$ from $\mathbb{R}^p \times (\mathbb{R} \to \mathbb{R})^n$ to $\mathbb{R}^k$, used to model static properties of the system (Eq. 2).

$$\mathbf{f}(\mathbf{x}, \mathbf{u}) \leq 0 \tag{2}$$

In order to model components evolving with use, we consider some differential constraints (Eq. 3 and 4).

$$\mathbf{u}'(t) = \mathbf{h}(t, \mathbf{x}, \mathbf{u}) \tag{3}$$

where $t \subset \mathbb{R}$ is a local variable used to control the chronological evolution and $\mathbf{h} \in \mathbb{R}^{p+1} \times (\mathbb{R} \to \mathbb{R})^n \to \mathbb{R}^n$ is the function describing the dynamical behavior of the model. The specificity of the problems we try to solve, is the hardly solvable multiple differential constraints (Eq. 4).

$$\mathbf{u}'(t) = \begin{cases} \mathbf{h_1}(t, \mathbf{x}, \mathbf{u}) \text{ if } g_1(t, \mathbf{x}, \mathbf{u}) \\ \vdots \\ \mathbf{h_m}(t, \mathbf{x}, \mathbf{u}) \text{ if } g_m(t, \mathbf{x}, \mathbf{u}) \end{cases} \tag{4}$$

where for all i in $\{1 \dots m\}$, $\mathbf{h_i} \in \mathbb{R}^{p+1} \times (\mathbb{R} \to \mathbb{R})^n \to \mathbb{R}^n$ is the description of the i-th dynamical behavior guarded by the boolean function $g_i \in \mathbb{R}^{p+1} \times (\mathbb{R} \to \mathbb{R})^n \to \mathbb{B}$ ($\mathbb{B}$ is the boolean set, $\mathbb{B} = \{False, True\}$).

3 Preliminaries

3.1 Interval Arithmetic (IA)

Interval Arithmetic (IA) [**?**] is a method to compute safe arithmetic operations. Values and results of operations are bounded by controlling the rounding (up, down) of the unit core computing and redefining basic operations and functions. The IA is also able to induce some reasoning on a continuous set of values, without enumerating all of them. This last point will be useful to do global optimization over real inputs. The set of intervals on the real line $\mathbb{R}$ is defined as below (Eq. 5) :

$$\mathbb{IR} = \left\{ [\underline{a}; \overline{a}] \;\middle|\; \begin{array}{c} (\underline{a}, \overline{a}) \in \mathbb{R}^2 \\ \underline{a} \leq \overline{a} \end{array} \right\} \tag{5}$$

In order to be able to compute on $\mathbb{IR}$, basic binary operators $\bullet \in \{+, -, \times, \div\}$ and usual functions $f \in \{abs,\ cos,\ sin,\ tan,\ exp,\ log,\ power,\ root \dots\}$ are extended on the interval arithmetic (Eq. 6). Let $([a], [b]) \in \mathbb{IR}^2$, then :

$$\begin{aligned} [a] \bullet [b] &= \{x \bullet y \mid x \in [a], y \in [b]\} \\ f([a]) &= \{f(x) \mid x \in [a]\} \end{aligned} \tag{6}$$

For each function $f : \mathbb{D} \to \mathbb{R}^n$ with $\mathbb{D} \subseteq \mathbb{R}$, we define the *range* of f over an interval $[a] \subseteq \mathbb{D}$ by :

$$range(f, [a]) = \{f(x) \mid x \in [a]\} \tag{7}$$

The interval-arithmetic evaluation of a complex function $f(x_1, \dots, x_n)$ over some intervals $[a_1] \dots [a_n]$, denoted by $f([a_1], \dots, [a_n])$ is obtained by replacing all the variables $(x_1 \dots x_n)$ by their respective interval value $([a_1] \dots [a_n])$ and evaluate each part of f (f is a composition of sub-functions) by including the range interval.

The IA is fast computable and doesn't require a lot of memory. Unfortunately for some cases results are too large and irrelevant. To illustrate, let us consider the interval $[a] = [-10; 10]$. Computing $[a] - [a]$ we get $[-10; 10] - [-10; 10] = [-20; 20]$ which is totally over-approximating 0. Thinner results can be obtained using piecewise evaluation over intervals or *Affine Arithmetic* (AA) developed a few years later [**?**] with some interesting properties but unfortunately some important memory and computing time requirements. To compute the results of the test presented in this article we used the IA efficiently implemented in the library BIAS [**?**].

3.2 Tube Arithmetic (TA)

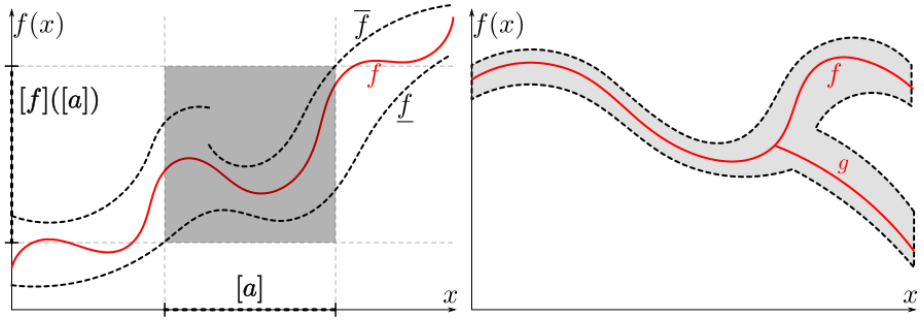
Tube Arithmetic (TA) [**?**] is a generalization of the IA to enclose unary functions. The TA will be used for internal representation of necessary functional variables which are to play with dynamic constraints (ODEs and Multi-ODEs). Let $\mathbf{f}$ a function from $\mathbb{R}$ to $\mathbb{R}^n$. We note $[\mathbf{f}]$ the tube defined by two functions $\underline{\mathbf{f}}$ and $\overline{\mathbf{f}}$ bounding $\mathbf{f}$ (Fig. 1a). We got some similarities with the IA like the evaluation (Eq. 8).

$$[\mathbf{f}]([t]) = range(\mathbf{f}, [t]) = \bigcup_{s' \in [t]} [\underline{\mathbf{f}}(s); \overline{\mathbf{f}}(s)] \tag{8}$$

Also, basic binary operations $\bullet$ on the tubes are defined as below (Eq. 9) :

$$([\mathbf{f}] \bullet [\mathbf{g}])([t]) = [\mathbf{f}]([t]) \bullet [\mathbf{g}]([t]) \tag{9}$$

We extended the TA to *Multiple-Tube Arithmetic* (Multi-TA) to be able to enclose multiple functions within the tiniest enclosure. A multi-tube (Fig. 1b) can be viewed as a set of tubes or a tube with holes. This trick is really efficient when enclosing Multi-ODEs trajectories.



(a) Evaluation of $[f]$ enclosing f over $[a]$ (b) Multi-tube enclosure of $f \cup g$

Fig. 1: Tube enclosures

3.3 Ordinary Differential Equation (ODE)

An *Ordinary Differential Equation* (ODE) of order n, is an equation of the form

$$u^{(n)}(t) = h(t, u, u', u'', \ldots, u^{(n-1)}) \tag{10}$$

which can be reduced to a n-dimensional first-order ODEs system $\boldsymbol{\Theta}$ (Eq. 11) by declaring new variables.

$$\boldsymbol{\Theta} \begin{cases} u_1'(t) = h_1(t, u_1, \ldots, u_n) \\ \quad \vdots \\ u_n'(t) = h_n(t, u_1, \ldots, u_n) \end{cases} \tag{11}$$

where $u_i \in \mathbb{R} \to \mathbb{R}$, and $h_i \in \mathbb{R} \times (\mathbb{R} \to \mathbb{R})^n \to \mathbb{R}$. We adopt the lighter vectorial notation $\mathbf{u}'(t) = \mathbf{h}(t, \mathbf{u})$.

The number of solutions for the equation system $\boldsymbol{\Theta}$ (Eq. 11) is infinite. The *Initial Value Problem* (IVP) is defined by adding an initial value $\mathbf{u}(t_0) = \mathcal{U}_\mathbf{0}$. The solution $\mathbf{u}^*$ is a function such that $\mathbf{u}^*$ satisfies $\boldsymbol{\Theta}$ and $\mathbf{u}^*(t_0) = \mathcal{U}_\mathbf{0}$.

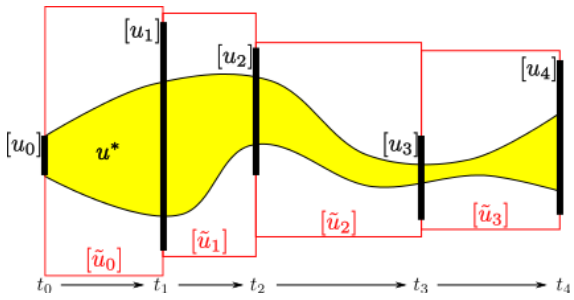


Fig. 2: Enclosure of an ODE solution $\mathbf{u}^*$ returned by Alg. 1

In the state of the art, the classical overall and efficient method to solve some IVP with a guaranteed enclosure (Fig. 2) is by using a successive integration scheme (Alg. 1). The sub-integration can be viewed as a composition of two phases.

Algorithm 1 : *Integrate*$(t_0 \in \mathbb{R}, [\mathbf{u_0}] \subseteq \mathbb{IR}^n, t_f \in \mathbb{R})$

1: **int** $i \leftarrow 0$; **real** $t \leftarrow t_0$
2: **while** $t \neq t_f$ **do**
3: $([\tilde{\mathbf{u}}_\mathbf{i}], t_{i+1}) \leftarrow$ *GlobalEnclosure*$(t_i, [\mathbf{u_i}], t_f)$
4: $[\mathbf{u_{i+1}}] \leftarrow$ *LocalEnclosure*$(t_i, [\mathbf{u_i}], [\tilde{\mathbf{u}}_\mathbf{i}], t_{i+1})$
5: $t \leftarrow t_{i+1}$; $i \leftarrow i + 1$
6: **end while**
7: **return** *Tube*$(t_0, [\mathbf{u_0}], [\tilde{\mathbf{u}}_\mathbf{0}], t_1, [\mathbf{u_1}], [\tilde{\mathbf{u}}_\mathbf{1}], \ldots, t_{i-1}, [\mathbf{u_{i-1}}], [\tilde{\mathbf{u}}_\mathbf{i-1}], t_i, [\mathbf{u_i}])$

The first one, *GlobalEnclosure* (Alg. 1, line 3) [**?**] [**?**], builds an a-priori enclosure $[\tilde{\mathbf{u}}_\mathbf{i}]$ of the solution on a range time step $[t_i; t_{i+1}]$ considering an initial enclosure $[\mathbf{u_i}]$ of $\mathbf{u}^*(t_i)$. The process based on the *Picard Existence and Uniqueness Theorem* and *Banach Fixpoint Theorem* is assured to return a safe enclosure of the solution and then a proof of the existence and uniqueness of the latter.

The *LocalEnclosure* method (Alg. 1, line 4) [**?**] computes a contraction $[\mathbf{u_{i+1}}]$ at t_{i+1} of the a-priori enclosure $[\tilde{\mathbf{u}}_\mathbf{i}]$ using the interval extension of *Taylor Model*. This enclosure can be eroded by calling a *PruneEnclosure* method [**?**] to cut unreachable domains. Then this enclosure is used as an initial value to integrate the ODE on $[t_{i+1}; t_{i+2}]$.

A lot of ODEs solvers are available like *AWA* [**?**], *COSY-VI* [**?**] and others. To realize the test presented later on, we used *VNODE-LP* [**?**] which is one of the best ODEs solver available.

4 Multiple Ordinary Differential Equation (Multi-ODE)

The problems we try to solve contain some complex differential constraints, which consist in a set of ODEs guarded by some logical expressions (Eq. 12).

$$\mathbf{u}'(t) = \begin{cases} \mathbf{h_1}(t, \mathbf{u}) \text{ if } g_1(t, \mathbf{u}) \\ \vdots \\ \mathbf{h_m}(t, \mathbf{u}) \text{ if } g_m(t, \mathbf{u}) \end{cases} \tag{12}$$

where $\mathbf{u} \in \mathbb{R} \to \mathbb{R}^n$, $\mathbf{h_i} \in \mathbb{R} \times (\mathbb{R} \to \mathbb{R}^n) \to \mathbb{R}^n$ and $g_i \in \mathbb{R} \times (\mathbb{R} \to \mathbb{R}^n) \to \mathbb{B}$ ($\mathbb{B}$ is the boolean set). Such systems are hardly solvable due to their non-continuous behavior as the solution can oscillate between many states using many differential functions. Moreover each guard couple g_{i_1} and g_{i_2} are not necessary disjoint, then a state set of the system can induce more than one dynamic. This case is

represented in the following Multi-ODE (Eq. 13) in which for all t in $\mathbb{R}$ such that $0 \leq \mathbf{u}(t) \leq 10$ we got $\mathbf{u}'(t) = \mathbf{h_1}(t, \mathbf{u}) \cup \mathbf{h_2}(t, \mathbf{u})$.

$$\mathbf{u}'(t) = \begin{cases} \mathbf{h_1}(t, \mathbf{u}) \text{ if } \mathbf{u}(t) \leq 10 \\ \mathbf{h_2}(t, \mathbf{u}) \text{ if } 0 \leq \mathbf{u}(t) \end{cases} \tag{13}$$

These dynamic systems are not treated in the state of the art. We developed an approach (Alg. 2) to solve them without any approximation, keeping the guarantee under certain conditions by bounding the solution.

Algorithm 2 : *Integrate*$(T_0 \in \mathbb{IR}, [\mathbf{u_0}] \subseteq \mathbb{IR}^n, T_f \in \mathbb{R}, k \in \mathbb{N}, mask \in \mathcal{P}(\mathbb{N}))$

1: $mask \leftarrow mask \cup \{k\}$
2: **int** $i \leftarrow 0$; **real** $t \leftarrow \underline{T_0}$; **Tube** $t_{res}(\emptyset)$
3: $[\mathbf{u_i}] \leftarrow Contract(g_k,\ t,\ [\mathbf{u_i}])$
4: **while** $(t \neq \overline{T_0} \vee t \neq T_f) \wedge [\mathbf{u_i}] \neq \emptyset$ **do**
5: **if** $t < \overline{T_0}$ **then** $([\tilde{\mathbf{u}}_\mathbf{i}], t_{i+1}) \leftarrow GlobalEnclosure(t_i, [\mathbf{u_i}], \overline{T_0})$
6: **else** $([\tilde{\mathbf{u}}_\mathbf{i}], t_{i+1}) \leftarrow GlobalEnclosure(t_i, [\mathbf{u_i}], t_f)$
7: $[\tilde{\mathbf{u}}_\mathbf{i}] \leftarrow Contract(g_k, [t_i, t_{i+1}], [\tilde{\mathbf{u}}_\mathbf{i}])$
8: **for all** $k' \in \{1 \dots m\} \setminus mask$ **do**
9: $t_{res} \leftarrow t_{res} \cup IntegrateMultiODE([t_i, t_{i+1}], [\tilde{\mathbf{u}}_\mathbf{i}], t_f, k', mask)$
10: **end for**
11: **if** $t < \overline{T_0}$ **then** $[\mathbf{u_{i+1}}] \leftarrow [\tilde{\mathbf{u}}_\mathbf{i}]$
12: **else** $[\mathbf{u_{i+1}}] \leftarrow LocalEnclosure(t_i, [\mathbf{u_i}], [\tilde{\mathbf{u}}_\mathbf{i}], t_{i+1})$
13: $t \leftarrow t_{i+1}$
14: $i \leftarrow i + 1$
15: $[\mathbf{u_i}] \leftarrow Contract(g_k, t_i, [\mathbf{u_i}])$
16: **end while**
17: **return** $t_{res} \cup \mathbf{Tube}(t_0, [\mathbf{u_0}], [\tilde{\mathbf{u}}_\mathbf{0}], t_1, [\mathbf{u_1}], [\tilde{\mathbf{u}}_\mathbf{1}], \dots, t_{i-1}, [\mathbf{u_{i-1}}], [\tilde{\mathbf{u}}_\mathbf{i-1}], t_i, [\mathbf{u_i}])$

The method consists in solving ODEs using the classical approach (Alg. 1) to propagate branches of ODE, coupled with a *Set Inversion Via Interval Analysis* procedure (SIVIA) [**?**]. This procedure (called by the method *Contract* in Alg. 2 at each integration step) has two different uses:

– Filter accessible domains reach by the integration of a branch. Let $[\mathbf{u_i}]$ an enclosure of the solution $\mathbf{u}^*$ at t_i. Before to compute the integration step over the k-th branch, we can reduce the initial value $[\mathbf{u_i}]$ used in this computation by filtering it with the k-th guard (as we know states which are false through the k-th guard are inconsistent with this branch and aren't involved in its integration) (Alg. 2, lines 7 and 15).

$$[\mathbf{u_i}] \leftarrow Contract(g_k, t_i, [\mathbf{u_i}])$$

The same process can be applied with $[\tilde{\mathbf{u}}_\mathbf{i}]$ over $[t_i; t_{i+1}]$.

– Detect behavior discontinuities. Let $[\tilde{\mathbf{u}}_\mathbf{i}]$ an enclosure of the solution $\mathbf{u}^*$ over $[t_i; t_{i+1}]$) on the current branch k. Then for all $k' \neq k$ we can compute the guard intersection (Alg. 2, line 3, using the recursive call).

$$[\mathbf{u}_\mathbf{i}^{\mathbf{k}'}] \leftarrow Contract(g_{k'}, [t_i; t_{i+1}], [\tilde{\mathbf{u}}_\mathbf{i}])$$

If $[\mathbf{u}_\mathbf{i}^{\mathbf{k}'}]$ is empty then the k'-th ODE is not activated by the branch k over $[t_i; t_{i+1}]$. Otherwise, $[\mathbf{u}_\mathbf{i}^{\mathbf{k}'}]$ is not empty, it has to be considered as an initial value to start the integration on the branch k' (a behavior discontinuity over $[t_i; t_{i+1}]$ could exist).

By computing the union of tubes built that way we get a multi-tube enclosing all the solutions of the integration. Unfortunately, we get an over-approximation of the solution, because of IA. In addition to the pessimistic evaluation issue, the fact of considering bounding boxes is problematic. Indeed, finding the intersection between a bounding box from a branch k and a guard k' we get an enclosure of this intersection containing some states for which the guard g'_k is false. The integration of these states with $\mathbf{h}_{\mathbf{k}'}$ will generate new ones for which the guard g_k is evaluated as true. Then, in order to keep the guarantee, we will have to consider these new states as initial values for the branch k. This recursive process will finally enclose all the guards hit by the enclosure which is a huge over-approximation. In order to get a tiny enclosure of the solution, we assume that when a trajectory from a branch k goes to another dynamic k', it will never come back to the previous branch k.

$$\texttt{Alg. 2, line 1 :}\ mask \leftarrow mask \cup \{k\}$$
$$\texttt{Alg. 2, line 8 :}\ k' \in \{1 \dots m\} \setminus mask$$

5 Interval Branch and Bound Algorithm (IBBA)

To optimize on such problems, we used an *Interval Branch & Bound Algorithm* (IBBA) [**?**] [**?**] extended with the Multi-TA. In order to get a faster convergence towards some feasible solutions, we coupled this algorithm with a constraint satisfaction process we developed, in which we included the ODEs and Multi-ODEs constraints propagators.

For each step of the IBBA, a node is selected and bisected. The sub-nodes are then contracted using the constraint satisfaction process through two steps. Firstly algebraic constraints are used to filter search domains, because the former are fast computable. If the nodes still contain some feasible domains then the differential constraints are propagated. This step costs memory and time computing. They are the main negative impacts from the methods previously mentioned (Alg. 1 and 2).

At the end of each iteration, if the node still contains some feasible solutions, a final step is usually used in the IBBA in order to get the lowest upper bound. This part consists in randomly finding a feasible point in the domain described by the node. Computing both the feasibility and the goal from one point limits

the over-approximation caused by IA, then we obtain a tinier enclosure of the goal.

6 Experiments

In this section we present the results of the experiment we conducted on an Intel Core i5-4200M processor with 2 cores and 2 gigabytes of RAM under the 32-bits ubuntu 12.04 operating system. The Multi-ODEs constraints resolution is the most important part in this work, and the recursive method we proposed to solve it is currently not faster enough to be efficient in an optimization process. Given that current limit, we just present some partial results about the constraint satisfaction module we developed.

Let the following complex second order multi-ODEs constraint with two dynamic components (Eq. 14). We choose the following coefficients $a = 0.01$, $b = 0.02$ and $c = -1.0$ in order to get the needed behavior (the first piece cross the guard from the second piece). The initial value is declared with an algebraic constraint.

$$\begin{aligned}(u_0, u_1)'(t) &= \begin{cases} (u_0(t) * u_1(t) + c, a * u_0(t) + b) \text{ if } u_0(t) < 11.5 \\ (-b, 0) \text{ if } u_0(t) > 11 \end{cases} \\ (u_0, u_1)(0) &= ([9.5; 10.5], 0)\end{aligned} \tag{14}$$

The solution enclosure of the integration over $t = [0; 5]$ returned by the propagator after 1400ms is presented below (Eq. 15) and drawn on Figure 3 where the black curves (respectively red boxes) represent the guards (respectively the solution enclosure).

$$\begin{aligned}u_0(5) &= [11, 11.101] \\ u_1(5) &= [0.276, 0.39]\end{aligned} \tag{15}$$

We introduce a single test, since it's difficult to provide some random tests. Indeed, the guard has to be crossed, and dynamic systems are really sensitive. We are not able to establish some comparisons with other approaches because these kind of constraints (without disjoint guards) are not treated in the state of the art.

7 Perspectives

Through this work we demonstrated the feasibility of solving some really complex dynamic systems with guarantees. The propagation algorithm requires a lot of time. It still can be improved to be used in an optimization process but it could be the price to pay to get guarantee. In the continuity of this work, it would be interesting to develop some strategies inside the IBBA to propagate differential constraints.

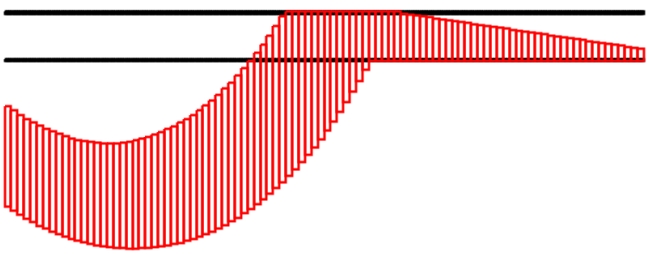


Fig. 3: enclosure of u_0 over $[0; 5]$