Object-Oriented Parametric Design of Constrained Motion Laws

R.GARZIERA, A.TASORA
Dipartimento di Ingegneria Industriale
Università degli Studi di Parma,
Parco Area delle Scienze, 43100 Parma,
ITALY
garziera@ied.unipr.it, tasora@ied.unipr.it, http://ied.unipr.it/

Abstract: - In this paper, a new method to design motion laws for automatic devices is proposed, featuring the ability of imposing constraints over sequences of functions, each defined by parameters. The optimal value for function parameters is obtained applying a Newton-Lagrange process over a constrained optimization problem. The outer Newton loop adopts a Bunch-Parlett linear solver which exploits the sparsity of the coefficient matrix, thus allowing high computational efficiency. The resulting method has been implemented into a custom multibody software which provides a powerful yet expandable way to design motion laws for mechatronic devices.

Keywords: - Functions, motion laws, robots, parametric design, optimization.

1 Introduction

The development of automatic devices and robots often requires the cumbersome process of designing the motion laws for the controlled degrees of freedom.

Many kinds of motion laws are currently adopted for this purpose (polynomial, cycloidal, etc.), usually joined into a sequence of multiple functions if complex trajectories are required [1].

Dealing with long sequences of motion laws, the many degrees of freedom offered by the parameters of the function segments could be subject to user-defined constraints: simplest cases can be solved immediately by analytical methods, if constraints are as simple as imposed continuity between segments or total movement over a fixed time span.

However, introducing more complex constraints (such as imposing acceleration or position at generic time values, setting whatever formula relationship between parameters, etc.) requires a generic, non-heuristic approach to the problem. Here we discuss a method which can be used to impose arbitrary constraints to the motion laws, that is a constrained non-linear optimization problem over the space of function's parameters.

In order to define sequences of functions, we adopted an object-oriented approach where each function segment is an instance of a specific class. Instancing and sequencing of functions can be done at compile-time (with C++ language) or at runtime (using Javascript interpreted language).

Finally, also an interactive graphical user-interface has been developed so that constraints over motion laws can be placed easily into the graph of the function in form of dimensioning symbols, just like most variational-parametric CAD software does for geometric shapes.

2 The problem

Let consider a motion law as a piecewise function of the independent variable t, that is a $y = f(t)$ function $f : t \in \mathbb{R}^+ \rightarrow y \in \mathbb{R}$.

Most often the shape of such function can be defined by a finite set of parameters (for example the position and weights of knots, in case $f(t)$ is a spline), whose value can be decided in order to achieve some specific goal. Hence all n function parameters can be collected into a $\mathbf{q}$ vector, with $\mathbf{q} \in \mathcal{Q}$, $\mathcal{Q} \subset \mathbb{R}^n$ that is

$$\mathbf{q} = \{q_1, q_2, ..., q_n\}^T . \tag{1}$$

Note that in case of functions which depend on w sub-functions, such as in the case of sequences or operations between children functions, a vector of design parameters can still be defined, as a collection of the parameter vectors $\mathbf{q}_{Fi}^T$ of the w children functions:

$$\mathbf{q} = \left\{\mathbf{q}_{F1}^T, \mathbf{q}_{F2}^T, ..., \mathbf{q}_{Fw}^T\right\}^T . \tag{2}$$

In general, we have a $f : \mathcal{T}, \mathcal{Q} \rightarrow \mathcal{Y}$ mapping:

$$y = f(t, \mathbf{q}) \qquad f : t \in \mathbb{R}, \mathbf{q} \in \mathbb{R}^n \rightarrow y \in \mathbb{R}. \tag{3}$$

This $f(t, \mathbf{q})$ function can be subject to m non-linear constraints, each with mapping $C_i : \mathcal{Q} \rightarrow \mathcal{C}_i$, which can be collected into a compact vectorial constraint:

$$\mathbf{C}(\mathbf{q}) = \mathbf{0} \qquad \mathbf{C} : \mathbf{q} \in \mathbb{R}^n \rightarrow \mathbf{C}(\mathbf{q}) \in \mathbb{R}^m, \tag{4}$$

for $\mathbf{C} \in \mathcal{C}$. Different approaches for the solution of $\mathbf{q}$ will be discussed heretoafter.

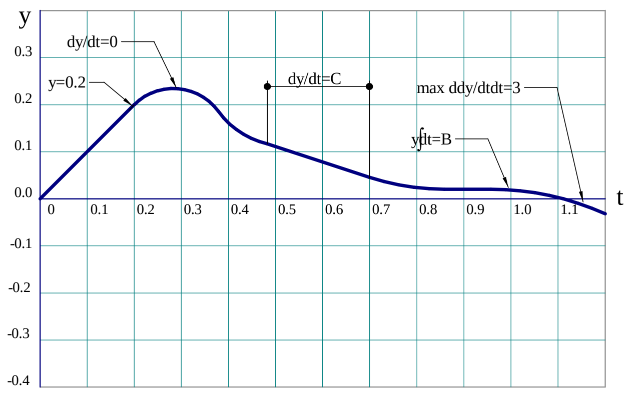


Figure 1: Motion law with user-imposed constraints.

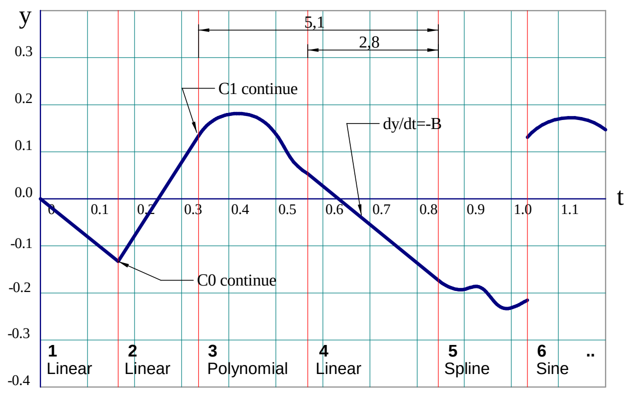


Figure 2: Constrained sequence of basic segments.

2.1 Constraints

Depending on the features required to the motion law $y = f(t)$, appropriate constraint equations can be introduced in eq.4. For example, the function may have specific values of y at given instants of time t, as well as specific values of speed and acceleration, not necessarily at once (Figure 1).

Also, in most complex situations, constraints could put into relation either some parameters q and function values (or derivative, or integrals),

When dealing with functions made by a sequence of segments, the requirement of Cn-continuity between such sub-functions can be represented by a constraint, acting on the parameters of these functions (Figure 2). For example one could build a sequence of polynomial functions, then require that all are tangent at the interfaces.

Following is a list of most useful constraints, whose meaning in a context of motion design is self-explaining.

- Prescribed value Y_a at given T_a time instant:
  $y|_{T_a} - Y_a = 0$
- Prescribed n-th derivative D_a at given T_a time instant[1]:
  $(\partial^n y/\partial t^n)|_{T_a} - D_a = 0$
- Prescribed integral I_a at given T_a time instant:
  $(\int ydt)|_{T_a} - I_a = 0$
- C0-continuity between segments:
  $y_{T_a+} - y_{T_a-} = 0$, with T_a interface between two segments,
- Cn-continuity between segments:
  $(d^n y/dt^n)|_{T_a+} - (d^n y/dt^n)|_{T_a-} = 0$.
- Prescribed value Q of a parameter q_i:
  $q_i - Q = 0$
- Most generic constraint:
  $C(\mathbf{q}) = 0$

All the equations above can be written as $C_i(\mathbf{q}) = 0$, and these can be collected in a vector of equations, that is $\mathbf{C}(\mathbf{q}) = \mathbf{0}$ as needed in Equation 4.

2.2 Case of unique solution(s)

In the case $m = n$, where $\dim(\mathcal{C}) - \dim(\mathcal{Q}) = 0$, nonlinear equation 4 can be solved for a root $\mathbf{q} \in \mathcal{Q}$. This problem can be faced with the straightforward application of a Newton-Raphson method, which converges to a numerical approximation $\widetilde{\mathbf{q}}$ of the exact root $\overline{\mathbf{q}}$ after few iterations:

$$\Delta \mathbf{q}_{i+1} = -\left[\frac{\partial \mathbf{C}(\mathbf{q}_i)}{\partial \mathbf{q}}\right] \mathbf{C}(\mathbf{q}_i), \tag{5}$$

$$\mathbf{q}_{i+1} = \mathbf{q}_i + \Delta \mathbf{q}_{i+1} \tag{6}$$

where $[\partial \mathbf{C}/\partial \mathbf{q}]$ is the jacobian of $\mathbf{C}$ respect to the $\mathbf{q}$ parameter vector, and we will write it $[\mathrm{C}_\mathrm{q}]$ heretofore.

However it often happens that the system has more degrees of freedom than the applied constraints. This is the case of $m < n$, that is $\dim(\mathcal{C}) < \dim(\mathcal{Q})$, and poses a different challenge. This is the case which we will deal with, in search of the highest generality.

2.3 Under-constrained case

Since an underconstrained problem leaves $n - m$ degrees of freedom, we can try to chase the solution which minimizes some useful function, for example the least change from the initial state $\mathbf{q}_0$ given at the beginning of the Newton Raphson process (this can be an useful default behavior of the solver because each time the user modifies a constraint, he would like to see the smallest change in other parts of the function under inspection).

Otherwise, introducing more freedom of control from the user point of view, the minimization could search the least value of some user-defined function, for example when one aims at least acceleration, least jerk, smallest average speed and so on [2].

In either cases, one ends with a constrained minimization problem [3] of the type:

$$\min |_q F(\mathbf{q}) \tag{7}$$
$$\mathbf{C}(\mathbf{q}) = \mathbf{0} \tag{8}$$

where $F : \mathcal{Q} \to \mathcal{F}$, $\mathcal{F} \subset \mathbb{R}$, $\mathcal{Q} \subset \mathbb{R}^n$ is a generic objective function which should be minimized, $\mathbf{q} \in \mathcal{Q}$ are the optimization variables, and $\mathbf{C} : \mathcal{Q} \to \mathcal{C}$ are the constraints.

The Lagrangian,

$$\mathcal{L}(\mathbf{q}, \lambda) := F(\mathbf{q}) + \lambda^T \mathbf{C}(\mathbf{q}) \tag{9}$$

can be used to convert the constrained optimization problem into a system of nonlinear equations, in fact the first order optimality conditions state that at *KKT points* (ex: a local minimum, saddle or maximum) the Lagrangian gradient must vanish:

$$\left\{ \begin{array}{c} \partial\mathcal{L}/\partial\mathbf{q} \\ \partial\mathcal{L}/\partial\lambda \end{array} \right\} (\mathbf{q}, \lambda) = \mathbf{0}, \tag{10}$$

$$\left\{ \begin{array}{c} \partial F/\partial\mathbf{q} + [\partial\mathbf{C}/\partial\mathbf{q}]^T \lambda \\ \mathbf{C}(\mathbf{q}) \end{array} \right\} = \mathbf{0} \tag{11}$$

Equation 11 can be written in the compact form $\mathbf{G}(\mathbf{y}) = \mathbf{0}$, where $\mathbf{G}$ is the lagrangian gradient (a nonlinear function of variables $\mathbf{y} = \left\{\mathbf{q}^T, \lambda^T\right\}^T$) The solution of the nonlinear problem $\mathbf{G}(\mathbf{y}) = \mathbf{0}$ may come from a Newton Raphson method, that is:

$$\Delta\mathbf{y}_{i+1} = -\left[\frac{\partial\mathbf{G}(\mathbf{y}_i)}{\partial\mathbf{y}}\right]^{-1} \mathbf{G}(\mathbf{y}_i), \tag{12}$$
$$\mathbf{y}_{i+1} = \mathbf{y}_i + \Delta\mathbf{y}_{i+1}. \tag{13}$$

The method may converge in few steps if a good initial estimate is provided, but the solution of the linear system in the step of equation 12 may be be difficult [4], either for the huge dimension of the jacobian matrix, either because such matrix may be ill conditioned, either because some parts of the matrix must be obtained by numerical differentiation, prone to roundoff and truncation.

In fact the linear system for the Newton step (equation 12) can be written as $[\mathrm{K}]\Delta\mathbf{y}_{i+1} = -\mathbf{G}(\mathbf{y}_i)$:

$$\left[\begin{array}{cc} [\mathrm{W}] & [\mathrm{C_q}]^T \\ [\mathrm{C_q}] & [0] \end{array} \right] \left\{ \begin{array}{c} \Delta\mathbf{q}_{i+1} \\ \Delta\lambda_{i+1} \end{array} \right\} = \left\{ \begin{array}{c} -\mathbf{g}_i + [\mathrm{C_q}]^T\lambda_i \\ -\mathbf{C}_i \end{array} \right\} \tag{14}$$

where the terms have the following meaning:

- $[\mathrm{W}] := \left[\left[\partial^2 F/\partial\mathbf{q}^2 + \sum_j \lambda_j \partial_{qq} c_j\right]\right]$
- $[\mathrm{C_q}] := [\partial\mathbf{C}/\partial\mathbf{q}]$
- $\mathbf{g} := \partial F/\partial\mathbf{q}$

The block-structured $[\mathrm{K}]$ matrix,

$$[\mathrm{K}] = \left[\begin{array}{cc} [\mathrm{W}] & [\mathrm{C_q}]^T \\ [\mathrm{C_q}] & [0] \end{array} \right] \tag{15}$$

often called *KKT (Karush-Kuhn-Tucker) matrix*, is highly sparse, symmetric and indefinite. However it may be singular if constraints are ill conditioned, that is when $\mathrm{rank}([\mathrm{C_q}]) < \dim(\mathcal{C})$, or when the Hessian block is near singularity, that is when $\det[\mathrm{W}] \approx 0$.

Also, recalling that $\lambda_{i+1} = \Delta\lambda_{i+1} + \lambda_i$, equation 14 can be written in an equivalent form which requires less floating point operations:

$$\left[\begin{array}{cc} [\mathrm{W}] & [\mathrm{C_q}]^T \\ [\mathrm{C_q}] & [0] \end{array} \right] \left\{ \begin{array}{c} \Delta\mathbf{q}_{i+1} \\ \lambda_{i+1} \end{array} \right\} = \left\{ \begin{array}{c} -\mathbf{g}_i \\ -\mathbf{C}_i \end{array} \right\} \tag{16}$$

Let consider the following example. If one wants the $\mathbf{q}$ solution which minimizes the objective function $F = \frac{1}{2}\mathbf{q}^T\mathbf{q}$, subject to constraints $\mathbf{C}$, the hessian becomes a simple diagonal matrix [I]. Also, the first step of the Newton procedure turns into the following system, for $\mathbf{q}_0 = \mathbf{0}$:

$$\left[\begin{array}{cc} [\mathrm{I}] & [\mathrm{C_q}]^T \\ [\mathrm{C_q}] & [0] \end{array} \right] \left\{ \begin{array}{c} \Delta\mathbf{q}_{i+1} \\ \lambda_{i+1} \end{array} \right\} = \left\{ \begin{array}{c} \mathbf{0} \\ -\mathbf{C}_i \end{array} \right\} \tag{17}$$

Special methods can solve the linear system of equation 17 by exploiting the block-sparsity of the $[\mathrm{C_q}]$ matrix, for example the decomposition of [K] into a Bunch-Parlett $[\mathrm{L}][\mathrm{D}][\mathrm{L}]^T$ form can be very efficient in this circumstance [5], esspecially when adopting custom algorithms for sparse matrices factorization [6].

Also, if $\mathbf{g}_i$ is kept updated as $\mathbf{g}_i = \partial F/\partial\mathbf{q}$, given objective

$$F = \frac{1}{2}[\mathbf{q} - \mathbf{q}_0]^T[\mathbf{q} - \mathbf{q}_0] \tag{18}$$

we get:

$$\left[\begin{array}{cc} [\mathrm{I}] & [\mathrm{C_q}]^T \\ [\mathrm{C_q}] & [0] \end{array} \right] \left\{ \begin{array}{c} \Delta\mathbf{q}_{i+1} \\ \lambda_{i+1} \end{array} \right\} = \left\{ \begin{array}{c} -\mathbf{q}_i + \mathbf{q}_0 \\ -\mathbf{C}_i \end{array} \right\} \tag{19}$$

whose meaning can be more intuitive: at the end of the Newton loops, the new state will satisfy the constraints, with the smallest change from the initial guess $\mathbf{q}_0$. This is useful in an interactive context, where an user may change the constraints on the functions in successive steps, testing different design solutions: each time a constraint is changed/added, the last state is used as $\mathbf{q}_0$ and the Newton loop with equation 19 is performed, hence providing a new function whose shape is not too far from the last before the tweaking.

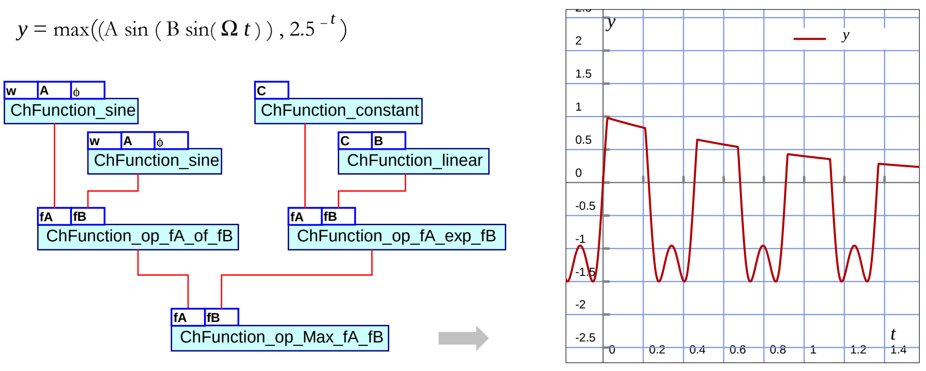


Figure 3: Example of a complex function as a tree of basic building blocks.

3 Implementation

The above theory has been implemented by using the C++ programming language, either in form of a stand-alone library, either integrated into our multibody simulation software [7].

3.1 Software design

Driven by a modular approach, the features of our library allow the building of complex motion laws starting from basic function objects (called *ChFunctions* heretoafter), which can be subject to operations (sequencing, multiplication, time warping, integration, etc.) by linking them to other *ChFunction* objects.

This building-block architecture means that a complex function can be seen as an hierarchical tree of simplier functions (Figure 3).

Also constraints are objects which can be linked to functions as needed, unlimited in number or type.

Instancing of functions and constraint objects (as well as setting up their hierarchy) can be done at C++ level, with compiled statements, or at scripting level, using an interactive shell and the compiled language Javascript (ECMA-262 specification, ISO-16262 standard). In fact, all C++ classes have a Javascript wrapper which allows an object-oriented control of *ChFunctions* and constraint objects using a scripting syntax which is very similar to C++.

Finally, an interactive graphical user interface has been implemented in our C++ multibody software, thus allowing an intuitive creation and manipulation of *ChFunctions* while designing the motion-laws of robotic devices (Figure 4 and 5).

3.2 Function classes

The *ChFunction* base class implements the common features of all inherited functions, for example a default numerical differentiation is provided, to get $\partial^n y/\partial t^n$. Then, all inherited functions must at least provide the $y = y(t)$, but they could also overload / override the base methods when necessary, for example the *ChFunction_sine* overrides the default numerical differentiation because the analytical derivative is known.

A partial view of the inheritance tree can be seen in Figure 6. Among the many specialized classes, we stress the importance of the *ChFunction_sequence* class, which is used to collect many sub-functions (maybe other sequences too) into a sequence of segments.

There are also functions which can modify other functions, for example *ChFunction_op_Fa_of_Fb* can be used to build functions of the $f_a(f_b(t))$ type; *ChFunction_repeat* creates periodic functions, and so on.

Another important class is *ChFunction_recorder* which can handle functions represented by sampled data, for example coming from instrumentation, from other programs or from numerical simulations. This class also implements a custom scheme for a robust and reliable numerical differentiation of sampled data, as exposed in [8].

Note that, in order to support the constrained optimization method presented in these pages, all functions must provide a method which exposes their optimization parameters, that is the vector of variables which can be collected into the $\mathbf{q}$ vector of equation 2.

3.3 Constraint classes

All constraints are handled as C++ objects. Each *ChFunction* can own an unlimited number of constraints, as they are inserted in a linked list.

Each specialized constraint class must implement the base methods which compute its $\mathbf{C}(\mathbf{q})$ residual, to be used in equation 19 for the Newton Raphson process. If possible, also the $[\mathrm{C_q}]$ jacobian should be provided by member overriding, if an analytical method is known, otherwise the base class can still compute it by using a default numerical differentiation

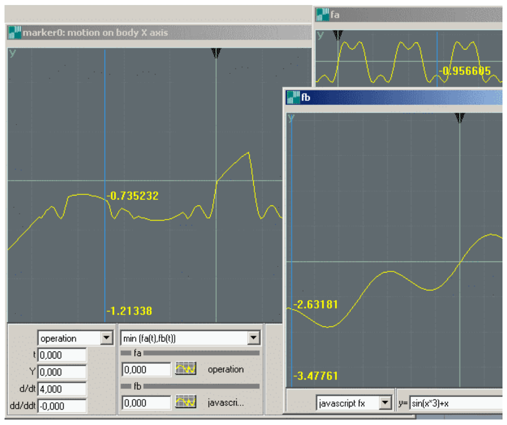
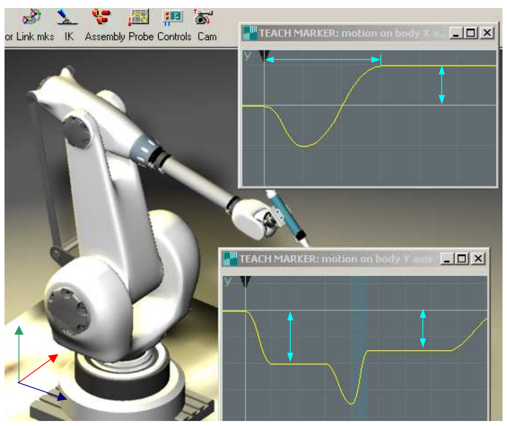


Figure 4: Graphical user interface for easy function creation and manipulation.



Link mks
IK
Assembly
Probe
Controls
Cam
TEACH MARKER: motion on body X axis
TEACH MARKER: motion on body Y axis

Figure 5: Constrained parametric design of motion laws as a tool for multibody simulation of robotic devices.

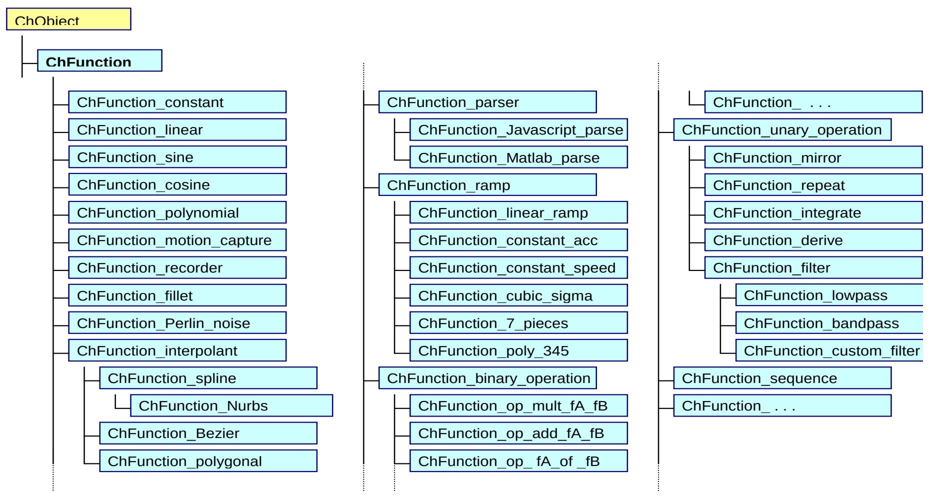


Figure 6: Class hierarchy of motion laws.

method.

Redundant constraints are detected as soon as they are inserted into the list, by using the following method: a single solution of equation 19 is performed just after the constraint insertion, giving null pivot during the factorization of the $[K]$ matrix if the constraint can be given as a linear combination of two other preexisting constraints.

4 Examples

Basic applications of the method are now discussed.

4.1 Constant acceleration

Motion law ramps with constant acceleration (and deceleration) are well known, being made of two parabolic pieces, with durations t_v and $t_s - t_v$, each with own constant acceleration A and B. If the ramp height h is imposed, for zero start-speed and zero end-speed there is a simple analytical solution: $A = 2h/t_v t_s$, $B = 2h/(t_s(t_s - t_v))$. The same result may be obtained automatically with the method presented in this paper, even if one does not remember the analytical solution.

We starts with a function of type *sequence*, containing two sub-functions of type *constant*. Let the duration and values of these constant be arbitrary values at the beginning. Now, put the *sequence* function inside a function of type *integrator*, then again into another *integrator*: we get the data tree of figure 7. Hence the root function is the motion ramp $y(t)$, for generic constant accelerations whose value is unknown. Finally, adding constraints such as $y|_0 = 0$, $y|_{t_s} = h$, $(\partial y/\partial t)_0 = 0$, $(\partial y/\partial t)_{t_s} = 0$, $t_v/t_s = 0.5$, the method will automatically find the right parameters $\mathbf{q}$ which satisfy all requirements, in detail the A and B values of the acceleration will correspond exactly to the analytical formulas.

4.2 Complex sequence

Now a more complex motion law is presented, where the values of function parameters which can satisfy the constraints are not as trivial as in the previous

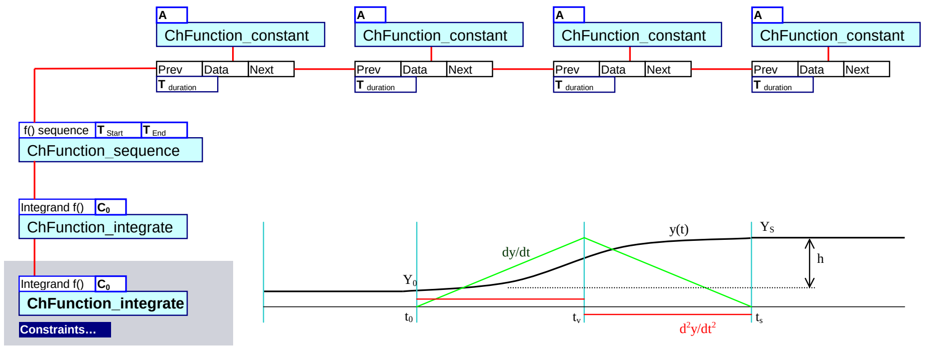


Figure 7: Example of a block-structured parametric function which reproduces a constant-acceleration motion law.

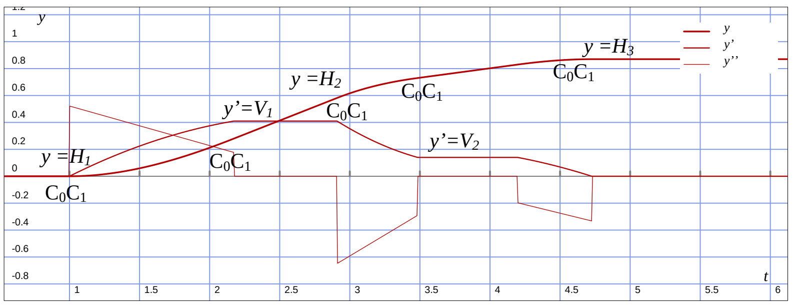


Figure 8: Motion law with multiple constraints. Function segments are of polynomial type.

example. While designing the motion for a tool of an automatic packaging device, a request was that the motion law must have two constant-speed segments (at given time intervals) before reaching the final ramp height. Also, at a given instant of time, the function must pass through a specific y value. The segments between the two constant-speed intervals are of polynomial type, and the resulting motion law (after application of constraints) can be seen in Figure8.

5 Conclusion

A new method to design motion laws has been proposed. It is based on the ability of imposing constraints over sequences of functions, each defined by parameters. The optimal value for these parameters is obtained after the application of a modified Newton-Lagrange process over a constrained optimization problem of KKT type. The outer Newton loop is endorsed by a Bunch-Parlett linear solver which exploits the sparsity of the coefficient matrix, hence allowing high computational efficiency even in case of many constraints.

The resulting theory has been implemented into our software for multibody simulation, thus providing a powerful yet expandable way to design motion laws for robots and automatic devices.

References

[1] G.Ruggieri, P.Magnani, *Progettazione meccanica funzionale*, Ed. UTET, Torino, Italy, 1990.

[2] R. Faglia, *Progetto del movimento per sistemi a camma tramite algoritmo genetico*, XII AIMETA, Napoli, Ottobre 1995.

[3] S. Rao, *Engineering Optimization: Theory and Practice*, Ed. John Wiley and Sons, New York.

[4] J.C. Haws and C.D. Meyer *Preconditioning KKT systems*, Numer. Linear Algebra Appl. 2001; 00:16

[5] J.R. Bunch, L.Kaufman, and B.N.Parlett, *Decomposition of a symmetric matrix*, Numer. Math., 27 (1976), pp. 95–109

[6] A.Tasora, *An optimized lagrangian multiplier approach for interactive multibody simulation in kinematic and dynamical digital prototyping*, VII ISCSB, Ed. CLUP, Milano, 2001.

[7] P.Righettini, A.Tasora *Implementazione object-oriented delle strutture dati di un codice di calcolo multibody general-purpose*, GIMC 2000 - XIII Convegno Italiano di Meccanica Computazionale, Brescia, 13-15 november 2000, Italy

[8] A. Cuccio, R. Garziera, S. Mauro, M. Silvestri, P. Righettini, R. Riva *Un linguaggio generale per la descrizione di leggi di moto*, XIV Congresso Aimeta, 6-9 Ottobre 1999, Como, Italy.