Available at
www.ElsevierMathematics.com
POWERED BY SCIENCE DIRECT®
ELSEVIER Applied Mathematics and Computation 152 (2004) 561–580
APPLIED MATHEMATICS AND COMPUTATION
www.elsevier.com/locate/amc

Complete automation of the generalized inverse method for constrained mechanical systems of particles

C. Itiki [a,*,1], J. José Neto [b]

[a] *Department of Telecommunication and Control Engineering, University of São Paulo, São Paulo, SP, Brazil*
[b] *Department of Computer Engineering and Digital Systems, University of São Paulo, São Paulo, SP, Brazil*

Abstract

A completely automated method for simulating constrained mechanical systems of particles is presented. A compiler generates MATLAB® program files that are specific to the problem presented by the user. These automatically generated programs call several subroutines that perform numerical differentiation, acceleration computation and numerical integration, providing the accelerations, velocities and coordinates of all the particles of the specific mechanical system.

Keywords: Generalized inverse method; Constrained mechanical systems; Automation; Simulation; Numerical routines

1. Introduction

One of the central problems in analytical mechanics is the derivation of equations of motion for constrained mechanical systems. Lagrange [1], Gauss [2], Gibbs [3] and Appell [4] proposed analytical methods to derive the equations

* Corresponding author. EPUSP-PTC, Av. Prof. Luciano Gualberto, trav.3, n.158, Cidade Universitária, São Paulo, SP 05508-900, Brazil.
E-mail addresses: cinthia@leb.usp.br (C. Itiki), joao.jose@poli.usp.br (J. José Neto).
[1] Supported by FAPESP grant 98/07653-8.

0096-3003/$ - see front matter
doi:10.1016/S0096-3003(03)00577-0

of motion. However, the choice of a reduced set of independent variables and the assumption of independent constraint equations constitute a hindrance to the automatic derivation of the equations of motion.

More recently, Udwadia and Kalaba [5] introduced the generalized inverse form of the acceleration for constrained mechanical systems. Their solution is equivalent to Gauss, Lagrange and Gibbs–Appell equations of motion [5–7]. One advantage of the generalized inverse form is that the actual acceleration is given explicitly. Besides it, non-holonomic constraints are dealt as easily as holonomic constraints, and dependent constraint equations may be included, as long as they are consistent with each other. Another advantage is that the method does not require the choice of a reduced set of variables. Its analytical solution still demands a lot of algebraic work for the determination of the actual acceleration. However, the automatic solution of the equations of motion can be obtained. A completely automated numerical method is presented in this work. Given the constraint equations, masses and external forces, the generalized inverse solution can be obtained automatically. The automation of the generalized inverse method makes the simulation of constrained mechanical systems of particles straightforward.

2. Methods

Consider an unconstrained mechanical system with n particles. The free acceleration vector $(x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_n, y_n, z_n)$ is related to the total external force $\mathbf{f} = [fx_1, fy_1, fz_1, \ldots, fx_n, fy_n, fz_n]^{\mathrm{T}}$ applied to the particles, and can be expressed as

$$\mathbf{a} = \boldsymbol{M}^{-1} \cdot \mathbf{f}, \tag{1}$$

where $\boldsymbol{M}$ is the mass matrix, a positive definite diagonal matrix, with masses $\{m_1, m_1, m_1, m_2, m_2, m_2, \ldots, m_n, m_n, m_n\}$ on its main diagonal.

Let us introduce m independent constraints, holonomic or non-holonomic, in the mechanical system. A set of m linear relationships on the acceleration components $(\ddot{x}_1, \ddot{y}_1, \ddot{z}_1, \ldots, \ddot{x}_n, \ddot{y}_n, \ddot{z}_n)$ is obtained by differentiating the constraint equations. Holonomic constraint equations $h_k(t, \mathbf{x}(t)) = 0$ should be differentiated twice, while non-holonomic constraints $g_k(t, \mathbf{x}(t), \dot{\mathbf{x}}(t)) = 0$ should be differentiated only once. The set of linear relationships can be written in matrix notation as

$$\boldsymbol{A}(t, \mathbf{x}, \dot{\mathbf{x}}) \cdot \ddot{\mathbf{x}} = \mathbf{b}(t, \mathbf{x}, \dot{\mathbf{x}}). \tag{2}$$

According to Udwadia and Kalaba [5], the generalized inverse form of the actual acceleration is given in terms of matrix $\boldsymbol{A}$ and vector $\mathbf{b}$

$$\ddot{\mathbf{x}} = \mathbf{a} + \boldsymbol{M}^{-1/2} \cdot (\boldsymbol{A} \cdot \boldsymbol{M}^{-1/2})^{+} \cdot (\mathbf{b} - \boldsymbol{A} \cdot \mathbf{a}), \tag{3}$$

where $(\boldsymbol{A}\cdot\boldsymbol{M}^{-1/2})^{+}$ is the Moore–Penrose generalized inverse matrix [8] of the product $(\boldsymbol{A}\cdot\boldsymbol{M}^{-1/2})$.

The automation of the generalized inverse method can be divided into four steps [9]. In the first step, the compiler writes specific programs that call general routines. The second step consists of the differentiation of constraint equations. The third step is the determination of matrix $\boldsymbol{A}$ and vector $\mathbf{b}$. The fourth step is the computation of the actual acceleration, given matrix $\boldsymbol{A}$ and vector $\mathbf{b}$ and numerical integration. Each step of the generalized inverse method can be implemented automatically.

In the main program, the user provides constraint equations, masses and external forces as inputs. The output (first step) is a specific set of MATLAB® source files that call general routines for numerical differentiation (second step), matrix $\boldsymbol{A}$ and vector $\mathbf{b}$ construction (third step), acceleration computation and numerical integration (fourth step). The word specific is used here to express that the files are generated for the specific problem that was defined by the user. The generation of the specific set of files using the given inputs is performed by the main program implemented in MATLAB®, which is called "compiler". Fig. 1 illustrates the relationship between the inputs and outputs of the compiler.

The general routines and the compiler are described in the following sections. Then, an example is given to illustrate how the system may be used for the simulation of constrained mechanical systems of particles.

2.1. General routines for differentiation

The second automation step involves differentiation of constraint equations. One differentiation of any non-holonomic constraint of the form $g_k(t,\mathbf{x}(t),\dot{\mathbf{x}}(t))=0$ results in a linear relationship on the acceleration components, given by Eq. (4).

$$\begin{aligned}&\frac{\partial g_k}{\partial \dot{x}_1}\ddot{x}_1+\frac{\partial g_k}{\partial \dot{y}_1}\ddot{y}_1+\frac{\partial g_k}{\partial \dot{z}_1}\ddot{z}_1+\cdots+\frac{\partial g_k}{\partial \dot{x}_n}\ddot{x}_n+\frac{\partial g_k}{\partial \dot{y}_n}\ddot{y}_n+\frac{\partial g_k}{\partial \dot{z}_n}\ddot{z}_n\\&\quad=-\left(\frac{\partial g_k}{\partial t}+\frac{\partial g_k}{\partial x_1}\dot{x}_1+\frac{\partial g_k}{\partial y_1}\dot{y}_1+\frac{\partial g_k}{\partial z_1}\dot{z}_1+\cdots+\frac{\partial g_k}{\partial x_n}\dot{x}_n+\frac{\partial g_k}{\partial y_n}\dot{y}_n+\frac{\partial g_k}{\partial z_n}\dot{z}_n\right).\end{aligned}\tag{4}$$

One may notice that the coefficients of acceleration components are partial derivatives. As a result, the differentiation of constraint equations involves the calculation of partial derivatives. The fast and efficient evaluation of derivatives (FEED) method [10] evaluates derivatives in an automatic and efficient manner. It expresses a given function as a combination of simple functions available in a library of numerical routines. Each FEED routine calculates the value of a specific simple function and its first and second partial derivatives.

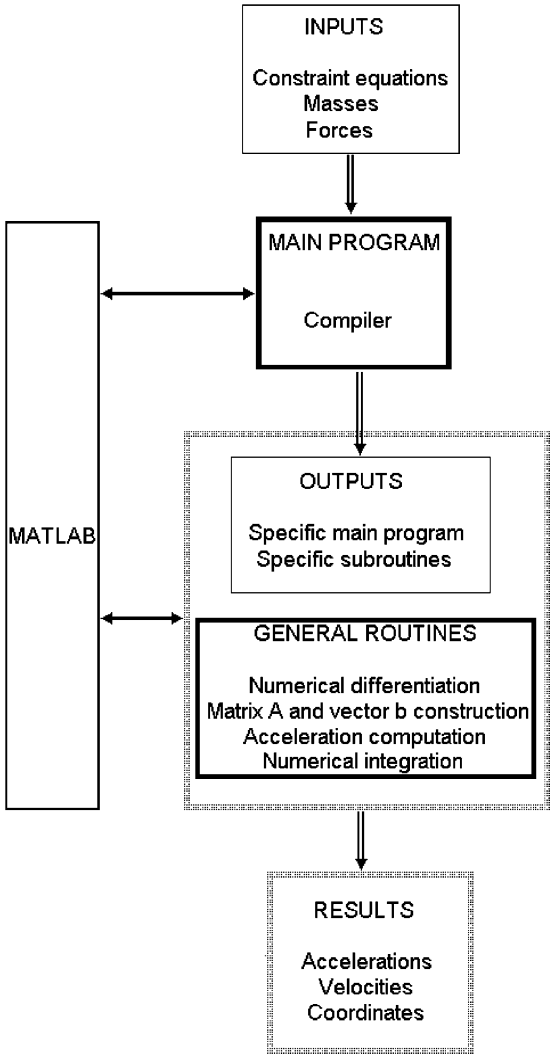


Fig. 1. Automation system.

Several FEED routines were implemented [9] in MATLAB® and adapted for this work.

For any function $g_k(t, \mathbf{x}, \dot{\mathbf{x}})$, vector G_k contains the function's value, and its first- and second-order partial derivatives. The first element of vector G_k is the value of function $g_k(t, \mathbf{x}, \dot{\mathbf{x}})$. The second element is the first partial derivative of the function g_k regarding time t. Rows 3 to $3n + 2$ contain the first partial derivatives regarding coordinates $(x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_n, y_n, z_n)$. The first

partial derivatives regarding velocity components $(\dot{x}_1, \dot{y}_1, \dot{z}_1, \dot{x}_2, \dot{y}_2, \dot{z}_2, \ldots, \dot{x}_n, \dot{y}_n, \dot{z}_n)$ are on rows $3n+3$ to $6n+2$.

Two differentiations of holonomic constraints $h_k(t, \mathbf{x}(t)) = 0$ result in linear relationships on the acceleration components. General FEED routines implement the two differentiations required for the second automation step. For holonomic constraints, vector H_k contains function $h_k(t, \mathbf{x})$, and its first- and second-order partial derivatives. The first element of vector H_k is the value of function $h_k(t, \mathbf{x}(t))$. Rows 2 to $3n+2$ contain the first partial derivatives regarding time t and coordinates $(x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_n, y_n, z_n)$. Second-order partial derivatives are on the last rows of vector H_k.

2.2. General routine for identification of matrix *A* and vector *b*

The third step of the generalized inverse method is the identification of matrix $\boldsymbol{A}$ and vector $\mathbf{b}$, in terms of the calculated partial derivatives. Each constraint equation brings a new row $\mathbf{A}_k$ to matrix $\boldsymbol{A}$, and a new element b_k to vector $\mathbf{b}$. For non-holonomic constraints $g_k(t, \mathbf{x}(t), \dot{\mathbf{x}}(t)) = 0$, the kth rows of matrix $\boldsymbol{A}$ and vector $\mathbf{b}$ may be obtained through Eq. (4), and are given in matrix notation by Eqs. (5) and (6).

$$\mathbf{A}_k = \left[\frac{\partial g_k}{\partial \dot{x}_1} \quad \frac{\partial g_k}{\partial \dot{y}_1} \quad \frac{\partial g_k}{\partial \dot{z}_1} \quad \cdots \quad \frac{\partial g_k}{\partial \dot{x}_n} \quad \frac{\partial g_k}{\partial \dot{y}_n} \quad \frac{\partial g_k}{\partial \dot{z}_n} \right], \tag{5}$$

$$b_k = -\left[\frac{\partial g_k}{\partial t} \quad \frac{\partial g_k}{\partial x_1} \quad \frac{\partial g_k}{\partial y_1} \quad \frac{\partial g_k}{\partial z_1} \quad \cdots \quad \frac{\partial g_k}{\partial x_n} \quad \frac{\partial g_k}{\partial y_n} \quad \frac{\partial g_k}{\partial z_n} \right] \begin{bmatrix} 1 \\ \dot{x}_1 \\ \dot{y}_1 \\ \dot{z}_1 \\ \vdots \\ \dot{x}_n \\ \dot{y}_n \\ \dot{z}_n \end{bmatrix}, \tag{6}$$

for $k = 1, 2, \ldots, m$.

For holonomic constraints, each row $\mathbf{A}_k$ of matrix $\boldsymbol{A}$ is given by Eq. (7) and each row b_k of vector $\mathbf{b}$ is given by Eq. (8).

$$\mathbf{A}_k = \left[\frac{\partial h_k}{\partial x_1} \quad \frac{\partial h_k}{\partial y_1} \quad \frac{\partial h_k}{\partial z_1} \quad \cdots \quad \frac{\partial h_k}{\partial x_n} \quad \frac{\partial h_k}{\partial y_n} \quad \frac{\partial h_k}{\partial z_n} \right], \tag{7}$$

$$b_k = -\begin{bmatrix} 1 & \dot{x}_1 & \dot{y}_1 & \cdots & \dot{z}_n \end{bmatrix} \times \begin{bmatrix} \dfrac{\partial^2 h_k}{\partial t^2} & \dfrac{\partial^2 h_k}{\partial t \partial x_1} & \dfrac{\partial^2 h_k}{\partial t \partial y_1} & \cdots & \dfrac{\partial^2 h_k}{\partial t \partial z_n} \\ \dfrac{\partial^2 h_k}{\partial x_1 \partial t} & \dfrac{\partial^2 h_k}{\partial x_1^2} & \dfrac{\partial^2 h_k}{\partial x_1 \partial y_1} & \cdots & \dfrac{\partial^2 h_k}{\partial x_1 \partial z_n} \\ \dfrac{\partial^2 h_k}{\partial y_1 \partial t} & \dfrac{\partial^2 h_k}{\partial y_1 \partial x_1} & \dfrac{\partial^2 h_k}{\partial y_1^2} & \cdots & \dfrac{\partial^2 h_k}{\partial y_1 \partial z_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial^2 h_k}{\partial z_n \partial t} & \dfrac{\partial^2 h_k}{\partial z_n \partial x_1} & \dfrac{\partial^2 h_k}{\partial z_n \partial y_1} & \cdots & \dfrac{\partial^2 h_k}{\partial z_n^2} \end{bmatrix} \begin{bmatrix} 1 \\ \dot{x}_1 \\ \dot{y}_1 \\ \vdots \\ \dot{z}_n \end{bmatrix}; \tag{8}$$

for $k = 1, 2, \ldots, m$.

In this work, a general routine that builds matrix $\boldsymbol{A}$ and vector $\mathbf{b}$ from the partial derivatives provided by FEED routines was implemented in MATLAB®.

2.3. General routines for computation of the actual accelerations, velocities and positions

Once matrix $\boldsymbol{A}$ and vector $\mathbf{b}$ are obtained, the fourth step of the generalized inverse method can be automated. The actual acceleration should be calculated by Eq. (3). However, there is a need for a numerical routine that calculates generalized inverse matrices. The numerical calculation of generalized inverse matrices may be performed in several ways [11,12]. In this work, a MATLAB® routine called ***pinv***$(\cdot)$ [13] was used. It calculates pseudo-inverse matrices numerically, through singular value decomposition routines. The actual acceleration is then obtained from the constraint equations, by an automated method.

One may also desire to obtain trajectories instead of accelerations. In this case, a numerical integration method may be used. In our work, a fourth-order Runge–Kutta method with constant step size is used for calculation of velocities and coordinates.

2.4. Compiler

The set of general routines described in the previous sections allows fluent users of MATLAB® to write programs that solve their specific problems, by calling those routines. However, in order for the system to be useful for all users, it is important that a man–machine interface be available, allowing data to be input by the user and automatically creating (first step) a specific set of programs, which calls general routines. This interface was implemented as a

compiler that converts input data into specific programs written in MATLAB® language. These programs, in turn, solve the specific problem.

As in any compiler development problem, it is advisable to specify rigorously the language to be accepted by the compiler, before its implementation. To do this, we use knowledge from compiling theory [14].

The compiler has three functional parts: a lexical analyzer, a syntax acceptor, and a set of semantic routines. The lexical analyzer processes the input chains (for example, the specifications of forces and constraint equations), decomposing them into basic elementary units (for example, variables, constants, functions and operation symbols), called "tokens". The syntax recognizer verifies the adherence of the sequence of tokens generated by the lexical analyzer to the grammar defining the syntax of the input language—the language that specifies the application problem to be solved. This grammar specifies the set of rules, by defining the valid sequences of tokens. Finally, the set of semantic routines implements the generation of the desired MATLAB® script text, by choosing the general routines to be called and delivering the needed error messages.

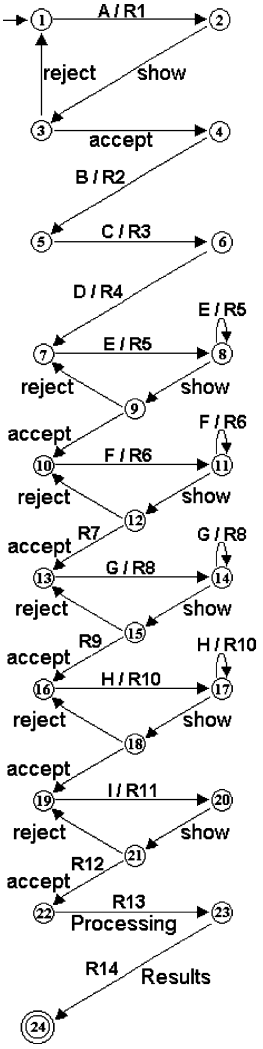
In this implementation, the compiler may be described as a sequence of nine dialogs:

Compiler

1. dialog A (subroutine name)
2. dialog B (system dimension)
3. dialog C (number of particles)
4. dialog D (number of constraint equations)
5. dialog E (masses)
6. dialog F (forces)
7. dialog G (constraint equations)
8. dialog H (initial values for numerical integration)
9. dialog I (parameters for numerical integration)

The automaton that represents the sequential logic of the compiler is illustrated in Fig. 2. Letters A through I represent dialogs between the compiler and its user. Actual dialogs are described in Table 1. Semantic routines are represented by R1 through R14 and are described by templates 1 to 4, in Appendix A.

According to Table 1, for each input type (file name, mass, force, constraint equation, etc.), there are corresponding valid atoms. For masses, any positive number is valid (integer, decimal, scientific). For system dimension, number of particles and number of constraint equations, the only valid atoms are natural numbers. For the sake of simplicity, we omit the descriptions of the lexical,



R1: Subroutine names [ARQ0, ARQ1] are stored. Default names are ["TEST", "TEST.M"]

R2: The natural number selected (l, 2 or 3) is stored in D (system dimension)

R3: The natural number selected (1,2, …, 10) is stored in F (number of particles)

R4: The natural number selected (1, 2, …, 10) is stored in *Neq* (number of constraint equations)

R5: For each particle, the corresponding mass is entered. In vector *Maux*, a group of D numbers is filled with the mass value of the corresponding particle i. For D=3, $Maux = [m_1, m_1, m_1, m_2, m_2, m_2, \ldots, m_p, m_p, m_p]^T$. Users are allowed to fix incorrect data if necessary. Vector *Maux* is copied into the main diagonal of a $(D \cdot F) \times (D \cdot F)$ diagonal matrix M.

R6: For each force component applied to each particle, the corresponding expression is entered. The correct input expressions are stored

R7: Subroutine template 1 is generated

R8: For each constraint equation, the corresponding expression is entered. The correct input expressions are stored.

R9: For each constraint equation, subroutine template 2 is generated.

R10: For each variable (coordinates and velocities), the corresponding initial conditions are entered. Values are stored.

R11: Integration step, initial and final times are entered. Values are stored.

R12: The main specific program (subroutine template 4) is generated.

R13: The main specific program is executed. Results are saved in disk files.

R14: Final results are displayed for users.

Fig. 2. Submachine *compiler*.

syntactical and semantic analyzers in the cases mentioned above, for they may be implemented directly as part of the host programming language.

However, in order to describe the more complex structures of the compiler, their analyzers are specified in detail.

File names assume the standard form, being denoted as a letter followed by an optional sequence of letters and/or decimal digits. The maximum length is 7,

Table 1
Compiler dialogs

System messages	Syntax of user's answer	Restrictions
A—Enter subroutine name (maximum 7 characters) ([a])	Name	Only the seven leftmost characters are used. Programs defined in this way are activated by a function call, identified by the letter c, followed by the name provided in this dialog (the resulting name is used for identifying the file corresponding to the specific main program).
B—Choose the system dimension	Select the desired option (a natural number)	Options: 1, 2 and 3
C—Choose the number of particles	Select the desired option (a natural number)	Options: natural numbers from 1 to 10
D—Choose the number of constraint equations	Select the desired option (a natural number)	Options: natural numbers from 1 to 10
E—Enter a value for m_i ([a])	Number ([b])	$i = 1, 2, \ldots$ number of particles
F—Enter the value of the force component ([a]) in direction d for particle p	Expression ([c])	$p \in \{1, \ldots n\}$, $d \in \{x, y, z\}$ all variables in the expression must have defined values at execution time.
G—Enter the ith constraint equation. ([a])	Expression ([c])	In case of syntax error, the user is notified and asked to reenter the ith constraint equation.
H—Enter the initial values for the coordinates and the velocity components of each particle	Number ([b])	
I—Enter the numerical integration step size	Number ([b])	
Enter the initial time	Number ([b])	
Enter the final time	Number ([b])	

[a] In these dialogs, all data are pre-filled with default values. The <ENTER> key may be pressed for their acceptance. Valid inputs, which are typed in by the user, replace default values. After the input of data, the system prompts the user for confirmation. Users answer by typing Y <ENTER> or <ENTER> in order to confirm. Any other key followed by <ENTER> causes the rejection of data. In this case, the dialog is restarted, with the previously typed data, presented to the user instead of the corresponding defaults.

[b] The syntax of numbers is verified at run-time by the subjacent MATLAB® environment.

[c] The syntax of the expressions provided by the user is verified, as described in Fig. 4.

that is, only the first seven characters of longer names will be considered. All grammars below are described in Wirth's notation for context-free languages.

Lexical Grammar for file names:

1. lettter = "A" | "B" | "C" | "D" | "E" | "F" |"G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" |"O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" |"W" | "X" | "Y" | "Z" | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" |"m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" |"u" | "v" | "w" | "x" | "y" | "z".
2. digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".

Syntax Grammar for file names:

1. name = letter {digit | letter}.

The lexical analyzer will extract the following valid atoms from force components and constraint equations: variables, functions, operators and numbers. Valid variables are: t, $x\diamondsuit$, $y\diamondsuit$, $z\diamondsuit$, $vx\diamondsuit$, $vy\diamondsuit$, $vz\diamondsuit$ where $\diamondsuit$ denotes a natural number in the range 1 to p for a system with p particles. For example, $vy14$ is valid, while $vy15$ is not valid for a system with 14 particles. Variables $t12$, y, vz and $z0$ are not valid, regardless of the system's dimension. The only valid names for functions are: sin, cos, tan, asin, acos, atan, exp, ln, log, sinh, cosh, tanh, asinh, acosh, atanh, and they are used in the conventional form. Valid operators are +, −, ∗, /, ^. They stand for the five usual arithmetic operations (sum, subtraction, multiplication, division, and power). Numbers may be denoted in natural, decimal or scientific notation.

The lexical grammar for constraint equations and forces may be described in the following way:

Lexical Grammar for force components and constraint equations:

1. variable = "t" | ("v" | ε) ("x" | "y" | "z") digit1 {digit}.
2. natural = digit {digit}.
3. decimal = natural "." {digit} | "." digit {digit}.
4. scientific = (natural ("." {digit} | ε) | "." digit {digit}) "e" ("+" | "−" | ε) digit {digit}.
5. function = "sin" | "cos" | "tan" | "asin" | "acos" | "atan" | "exp" | "ln" |"log" | "sinh" | "cosh" | "tanh" | "asinh" | "acosh" | "atanh".
6. digit1 = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
7. digit = "0" | digit1.

An automaton corresponding to the lexical analyzer associated with this grammar is illustrated in Fig. 3.

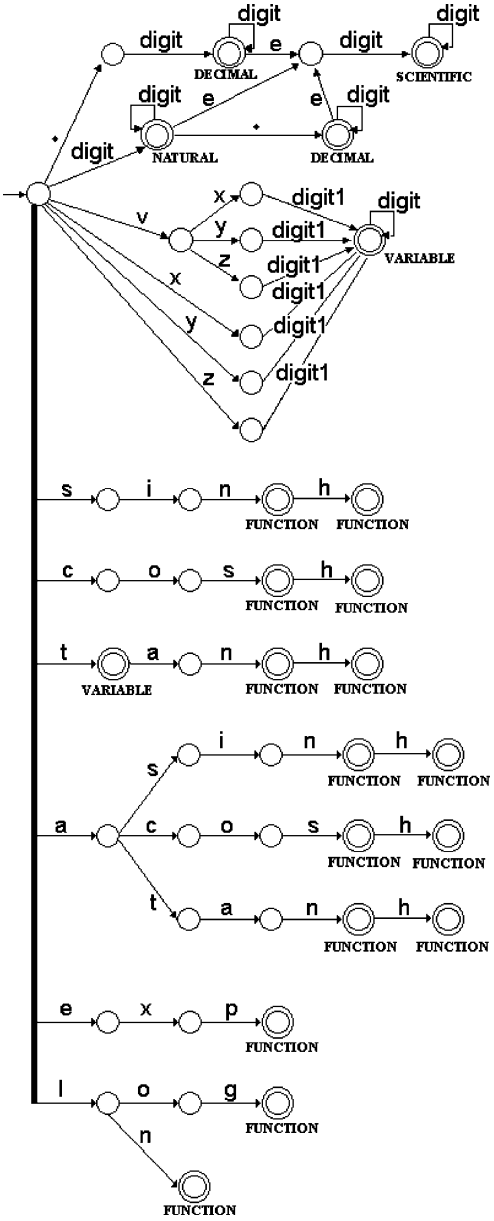


Fig. 3. Submachine *lexical analyzer* for force components and constraint equations.

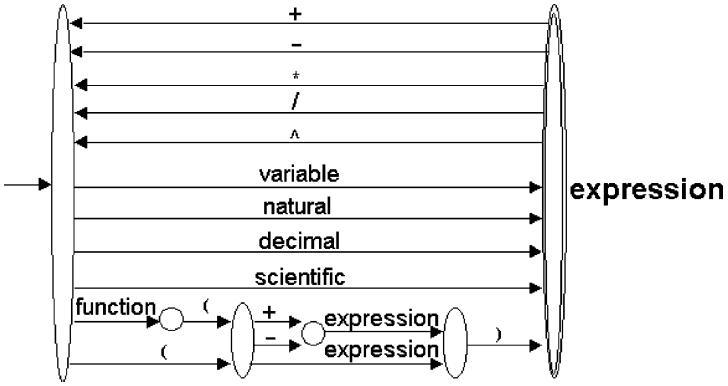


Fig. 4. Submachine *syntax analyzer* for force components and constraint equations.

From the syntactical point of view, atoms are arranged according to the following rule:

Syntax Grammar for force components and constraint equations:

1. expression = (variable | natural | decimal | scientific |
   "(" ("+" | "–" | ε) expression ")" | function "(" ("+" | "–" | ε) expression ")")
   {("+" | "–" | "*" | "/" | "^") (variable | natural | decimal | scientific |
   "(" ("+" | "–" | ε) expression ")" | function "(" ("+" | "–" | ε) expression")")}

A submachine corresponding to the non-terminal *expression* is depicted in Fig. 4.

3. Example

The simulation of a system with two particles is used as an application of the automated method. Coordinates are given in terms of an inertial Cartesian system $(x_1, y_1, z_1, x_2, y_2, z_2)$.

The first particle is subject to two holonomic constraints that describe the fact that the particle is moving in a meridian, over the surface of a ball rotating around the z-axis with a constant angular velocity.

These holonomic constraints are given by

$$h_1(t, \mathbf{x}) = x_1^2 + y_1^2 + z_1^2 - 1 = 0, \tag{9}$$

and

$$h_2(t, \mathbf{x}) = x_1 \cdot \sin(2\pi t) - y_1 \cdot \cos(2\pi t) = 0. \tag{10}$$

The second particle is subject to two non-holonomic equations. No external forces are applied to the particle. The two non-holonomic constraints are described by

$$g_3(t, \mathbf{x}, \dot{\mathbf{x}}) = z_2 \cdot \dot{x}_2 - \dot{y}_2 = 0, \tag{11}$$

and

$$g_4(t, \mathbf{x}, \dot{\mathbf{x}}) = z_2^2 \cdot \dot{x}_2 - \dot{y}_2 = 0. \tag{12}$$

The following set of values was given as input to the compiler: masses equal to 1 kg and an external force of −9.81 N applied to the first particle in the radial direction.

The compiler generated automatically the set of specific programs—included in Appendix B—that was the implementation of the generalized inverse method for this specific problem.

Numerical integration of the actual accelerations—performed by the set of specific programs—provided the trajectories of the particles. Integration was performed in MATLAB® for a 5-s time window, with and integration step size of 0.005 s. Initial conditions $(x_1, y_1, z_1, x_2, y_2, z_2, \dot{x}_1, \dot{y}_1, \dot{z}_1, \dot{x}_2, \dot{y}_2, \dot{z}_2)$ were consistent with the constraint equations and given by $(\sqrt{2}/2{,}0{,}\sqrt{2}/2{,}2{,}1{,}$ $-0.75{,}0{,}\sqrt{2} \cdot \pi{,}\ 0{,}\ 0{,}\ 0{,}\ 0.25)$.

The numerical integration results obtained through the automatically evaluated acceleration are shown in Figs. 5 and 6. Constraint functions h_1 and h_2—given by Eqs. (9) and (10)—were calculated for 5 s of numerical integration. Function h_1 assumed the maximum absolute value of 1.5×10^{-5}, while function h_2 had a maximum absolute value of 2.5×10^{-11}. Both values show that errors were negligible in comparison to the amplitude of the corresponding coordinates.

Fig. 6a–c shows the coordinates of the second particle. Coordinates x_2 and y_2 remain constant. However, z_2 has a linear response $z_2(t) = -0.75 + 0.25t$. Consequently, two of the velocity components remain equal to zero, and the third one is constant and equal to 0.25, as one may see in Fig. 6d–f. In Fig. 6g–i we may observe that all three acceleration components are equal to zero. With this second particle, there was a change in the rank of matrix A, for $z_2 = 0$. However, the results of numerical integration were quite satisfactory. Constraint functions g_3 and g_4 given by

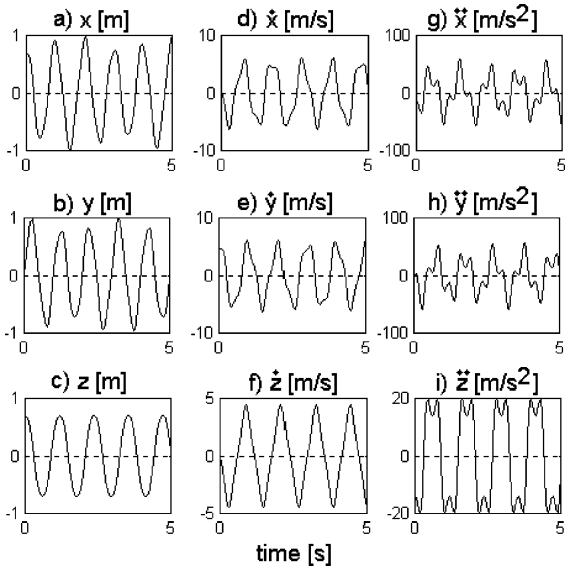


Fig. 5. (a–c) Coordinates, (d–f) velocities, and (g–i) accelerations of a particle subject to two holonomic constraints.

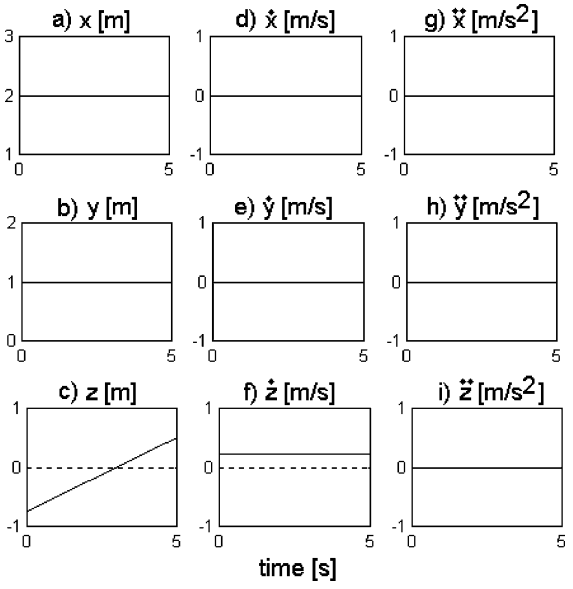


Fig. 6. (a–c) Coordinates, (d–f) velocities, and (g–i) accelerations of a particle subject to two non-holonomic constraints.

Eqs. (11) and (12), were exactly equal to zero, throughout the whole simulation interval.

4. Discussion

The automated method developed in this work deals with holonomic and non-holonomic constraints. It allows the use of rectangular coordinates instead of forcing the choice of generalized coordinates. Constraint equations do not need to be independent, as long as they are consistent. It also has flexibility to handle systems with several particles.

The compiler—a computer program that implements this method in MATLAB®—provides accelerations, velocities and coordinates of all particles of a constrained mechanical system, and can be easily used by people who are not experts in advanced topics of Mechanics. It was used for the simulation of a mechanical system with two independent particles and four constraint equations, by automatically creating a specific set of programs that calculated the acceleration in the generalized inverse form and performed numerical integration. The same compiler can be used to solve other problems involving constrained mechanical systems of particles, including particles that interact with each other, if the user provides suitable data—system dimension, number of particles, applied forces and constraint equations.

In summary, this work developed the complete automation of the generalized inverse method, for constrained mechanical systems of particles.

Acknowledgements

Authors thank Eduardo Nazima for the translation of Gauss's paper, and FAPESP for the research grant 98/07653-8.

Appendix A. Templates

Elements in rectangles are filled with appropriate values, according to the corresponding user-provided data.

Specific Subroutine Template 1

```
Function V = ARQ0 (t,U);
% Global variable definitions
global nU N Nhs Nhr Nns Nnr Nths Nthr Ntns Ntnr;
% Velocity vector
xdot=U(nU/2+1:nU);
% Initialization of matrix A and vector b
A=[ ]; b=[ ];
% Variables in terms of t and U
x1=U(1);y1=U(2);z1=U(3);
 ⋮
xP=U( D*P-2 );yP=U(D*P-1);zP=U(D*P);
vx1=U(D*P+1);vy1=U(D*P+2);vz1=U(D*P+3);
 ⋮
vxP=U(2D*P-2);vyP=U(2D*P-1);vzP=U(2D*P);
% Mass matrix
M=[ M ];
M2=M^(-0.5);
% Free acceleration vector
a=[ aX1(0); aY1(0); aZ1(0); … ; aXP(0); aYP(0); aZP(0)];
```

Specific Subroutine Template 2

```
% Time and velocity flags
tflag=TFLAG;  vflag=VFLAG;
% Number of variables and derivatives
Nv=Nhs; Nt=Nths;
% Variables in terms of parameters t and U
X1=f_var(U(1),1,Nt,tflag);
Y1=f_var(U(2),2,Nt,tflag);
Z1=f_var(U(3),3,Nt,tflag);
 ⋮
XP =f_var(U(D*P-2), D*P-2,Nt,tflag);
YP =f_var(U(D*P-1), D*P-1,Nt,tflag);
ZP =f_var(U(D*P), D*P,Nt,tflag);
% Subroutine call: implementation of the constraint equation.
F= EXPRESSION
Ain=A;bin=b;
% Update of matrix A and vector b
[A,b]=newAb(Ain,bin,F,xdot,N,tflag,vflag);
```

Specific Subroutine Template 3

```
% Actual acceleration vector
accel=a+M2*(pinv(A*M2))*(b-A*a);
% Velocity vector
V=[xdot(:); accel(:)];
```

Specific Subroutine Template 4

```
% PROGRAM ARQ1.M
clc; clear; clear functions;

% Initialization of variables
global nU N Nhs Nhr Nns Nnr Nths Nthr Ntns Ntnr;
N= D*P;
nU=2*N;
Nhs=N;      Nths=1+Nhs+Nhs*(Nhs+1)/2;
Nns=2*N;    Ntns=1+Nns;
Nhr=N+1;    Nthr=1+Nhr+Nhr*(Nhr+1)/2;
Nnr=2*N+1;  Ntnr=1+Nnr;

% Numerical integration variables
h= H;
t0= T0;
tf= TF;

% Initial conditions
w0=[ X1(0); Y1(0); Z1(0); … ; XP(0); YP(0); ZP(0); VX1(0); VY1(0); VZ1(0);…;
VXP(0); VYP(0); VZP(0) ];

% Numerical integration routine
[t,W]=ode45cte(' ARQ0 ',t0,tf,w0,h,0);

% Save data on file ARQ1.MAT
save ARQ1 t W;

% Plot graphics for particle 1
figure(1);clf;
subplot(311);
plot(t(:),W(:,1:3));
grid; ylabel('Coordinates');
subplot(312);
plot(t(:),W(:,7:9));
grid; ylabel('Velocities');
subplot(313);
plot(t(:),W(:,13:15));
grid; ylabel('Accelerations');
 ⋮
```

Appendix B. Specific programs

The specific main program defines global variables (number of coordinates and number of variables) and numerical integration variables (step size, initial time, end time, and initial conditions). It calls the routine ode45cte.m that performs numerical integration, and it saves the results in files.

The specific subroutine generated by the compiler defines the mass matrix and free acceleration vector. It also calls the numerical differentiation FEED routines, the matrix $\boldsymbol{A}$ and vector $\mathbf{b}$ construction routines, and the calculation of the actual acceleration vector. Only one of the four constraint equations is shown.

Constraint equations were differentiated by FEED routines f_const(c,Nt), f_var(x,i,Nt,tflag), f_mult(x,y,Nt,vflag), f_sin(x,Nv,vflag), f_cos(x,Nv,vflag), f_atan(x,Nv,vflag), and f_xttc(x,c,Nv,vflag). Matrix $\boldsymbol{A}$ and vector **b** were obtained through numerical routines for non-holonomic and holonomic constraint equations by the routine call [A,b] = newAb(A_{in}, b_{in}, G,xdot,N, tflag,vflag). The actual acceleration was calculated at each moment by MATLAB® built-in routines for addition, subtraction and multiplication of matrices, as well as computation of generalized inverse matrices.

Specific Main Program

```
clc; clear; clear functions;
global nU N Nhs Nhr Nns Nnr Nths Nthr Ntns Ntnr;
N = 6;
nU = 2*N;
Nhs = N; Nths = 1+Nhs+Nhs*(Nhs+1)/2;
Nns = 2*N; Ntns = 1+Nns;
Nhr = N+1; Nthr = 1+Nhr+Nhr*(Nhr+1)/2;
Nnr = 2*N+1; Ntnr = 1+Nnr;
h = 0.005;
t0 = 0;
tf = 5;
w0 = [sqrt(2)/2;  0;  sqrt(2)/2;  2;  1;  -0.75;  0;
4*atan(1)*sqrt(2); 0; 0; 0; 0.25];
[t, W] = ode45cte('artigo5', t0, tf, w0, h, 0);
save cartigo5 t W;
```

Specific Subroutine

```
function V = artigo5(t, U);
global nU N Nhs Nhr Nns Nnr Nths Nthr Ntns Ntnr;
xdot = U(nU/2+1:nU);
A = [ ]; b = [ ];
x1 = U(1); y1 = U(2); z1 = U(3); x2 = U(4); y2 = U(5); z2 =
U(6);
vx1 = U(7); vy1 = U(8); vz1 = U(9); vx2 = U(10); vy2 =
U(11); vz2 = U(12);


 M = [ 1.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000 ;
       0.00000000  1.00000000  0.00000000  0.00000000  0.00000000  0.00000000 ;
       0.00000000  0.00000000  1.00000000  0.00000000  0.00000000  0.00000000 ;
       0.00000000  0.00000000  0.00000000  1.00000000  0.00000000  0.00000000 ;
       0.00000000  0.00000000  0.00000000  0.00000000  1.00000000  0.00000000 ;
       0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  1.00000000 ];
```

```
M2 = M^(-0.5);
a = [-9.8l*xl; -9.8l*yl; -9.8l*zl; 0; 0; 0;];
⋮
tflag = l; vflag = 0;
Nv = Nhr; Nt = Nthr;
T = f_var(t,0,Nt,tflag);
Xl = f_var(U(l),l,Nt,tflag);
Yl = f_var(U(2),2,Nt,tflag);
Zl = f_var(U(3),3,Nt,tflag);
X2 = f_var(U(4),4,Nt,tflag);
Y2 = f_var(U(5),5,Nt,tflag);
Z2 = f_var(U(6),6,Nt,tflag);
F = f_mult(Xl,f_sin(f_mult(f_mult(f_const(8,Nt),
f_atan(f_const(l, Nt),Nv,vflag), Nv,vflag),
T,Nv,vflag),Nv,vflag),Nv,vflag)—f_mult(Yl,
f_cos(f_mult(f_mult (f_const(8,Nt),
f_atan(f_const(l,Nt),Nv,vflag),Nv,vflag),
T,Nv,vflag), Nv,vflag) ,Nv,vflag);
Ain = A; bin = b;
[A,b] = newAb(Ain,bin,F,xdot,N,tflag,vflag);
⋮
accel = a+M2*(pinv(A*M2))*(b-A*a);
V = [xdot(:); accel(:)];
```

References

[1] J.L. Lagrange, Mécanique Analytique. Jacques Gabay, Sceaux, France, 1989 (reprint of the 1788 original edition), pp. 158–428.

[2] C.F. Gauss, Über ein neues allgemeines Grundgesetz der Mechanik, Journal für die Reine und Angewandte Mathematik 4 (1829) 232–235.

[3] J.W. Gibbs, On the fundamental formulae of dynamics, American Journal of Mathematics 2 (1879) 49–64.

[4] P. Appell, Sur une Forme Générale des Équations de la Dynamique, Gauthier-Villars, Paris, France, 1925.

[5] F.E. Udwadia, R.E. Kalaba, A new perspective on constrained motion, Proceedings of the Royal Society of London A 439 (1992) 407–410.

[6] R. Kalaba, F. Udwadia, R. Xu, C. Itiki, The equivalence of Lagrange's equations of motion of the first kind and the generalized inverse form, Nonlinear World 2 (1995) 519–526.

[7] C. Itiki, R. Kalaba, F. Udwadia, Appell's equations of motion and the generalized inverse form, in: R.P. Agarwal (Ed.), Recent Trends in Optimization Theory and Applications, World Scientific Series in Applicable Analysis, vol. 5, World Scientific, Singapore, 1995, pp. 123–143.

[8] F.A. Graybill, Matrices with Applications in Statistics, second ed., Wadsworth, Pacific Grove, CA, 1983, pp. 105–148.
[9] C. Itiki, Constrained motion and generalized inverses with applications in biomechanics, Ph.D. Dissertation, University of Southern California, Los Angeles, CA, 1996.
[10] H. Kagiwada, R. Kalaba, N. Rasakhoo, K. Spingarn, Numerical Derivatives and Nonlinear Analysis, Plenum, New York, NY, 1986, pp. 1–13.
[11] T.N.E. Greville, Some applications of the pseudoinverse of a matrix, SIAM Review 2 (1960) 15–22.
[12] R. Kalaba, N. Rasakhoo, Algorithms for generalized inverses, Journal of Optimization Theory and Applications 48 (3) (1986) 427–435.
[13] MATLAB®, Reference Guide. The MathWorks Inc., Natick, MA, August 1992.
[14] J. José Neto, Introdução à Compilação, Livros Técnicos e Científicos, Rio de Janeiro, Brazil, 1987.