# Diferenciación Automática Aplicada en Simulación Multicuerpo

D. Gómez[1], J. García de Jalón[2]

[1]*SOCOIN*
*Paseo Club Deportivo, 1, Edificio 5,28223, Pozuelo de Alarcón, Madrid,*
*+34-912577051, davidgomezjim@gmail.com*

[2]*Universidad Politécnica de Madrid*
*INSIA y ETSII, José Gutiérrez Abascal, 2, 28006, Madrid, España*
*+34-913363213, javier.garciadejalon@upm.es*

Área Temática: Dinámica de Sistemas Multicuerpo

**Resumen**
La Diferenciación Automática (DA) es una nueva técnica que permite obtener, de forma automática y exacta, las derivadas de funciones definidas mediante código de ordenador. En este trabajo la DA se ha aplicado al cálculo de las Jacobianas de las ecuaciones de restricción. Como ejemplos concretos se presentan los casos de un robot y de un turismo.

**Palabras Clave:** Diferenciación Automática, Simulación, Derivadas, Sistemas Multicuerpo.

**Abstract**
Automatic Differentiation (AD) is a new technique that allows the programmer to obtain derivatives from functions defined trough source code, in an automatic and accurate form. It has been used for the calculation of constraint Jacobians and applied to a robot and a car.

**Keywords:** Automatic Differentiation, Simulation, Derivatives, Multibody Systems.

## 1. Introducción

La Diferenciación Automática (DA, o AD en inglés) es una nueva técnica matemático-informática, que posibilita la obtención de las derivadas de funciones escritas en código de ordenador, por complejas o extensas que éstas sean, con la mayor precisión posible y en un tiempo razonable, que en la mayor parte de los casos prácticos no excede de un pequeño múltiplo del propio tiempo de evaluación de la función [1]. Aunque pueda parecer que estos objetivos ya se habían logrado con la derivación numérica y la derivación simbólica, la DA es mucho más precisa que la primera y más versátil que la segunda, ya que está pensada para funciones definidas por el usuario de un modo informático y no analítico. Estas funciones pueden ser funciones escalares o vectoriales, recursivas, con ciertas discontinuidades y que por lo general van más allá de los límites que impone la derivación simbólica. La DA es una técnica que en su uso se parece más a las técnicas numéricas basadas en diferencias finitas, pero que es mucho más precisa que éstas. Sus

resultados, por lo general, consiguen la misma precisión que la propia evaluación numérica de la función original.

Existen dos tipos de herramientas que posibilitan la DA. Las herramientas del tipo *Sobrecarga de Operadores* determinan las derivadas en tiempo de ejecución. Para ello se manejan unas variables ampliadas (estructuras) que contienen tanto el valor real como las derivadas y se sobrecargan todas las funciones para que sean capaces de manejar esas variables ampliadas. Por otro lado las herramientas del tipo *Transformación de Código*, a partir de la función original, generan el código de una función que evalúa las derivadas, y que debe ser llamada por el usuario. Por lo tanto, la derivación se realiza antes de ejecutar el programa [2].

En simulación dinámica de sistemas multicuerpo existen numerosas situaciones en las que es necesario la obtención de derivadas, en especial Jacobianas, como son la resolución de sistemas no lineales por Newton-Raphson o la obtención de la Jacobiana de las ecuaciones de restricción, normalmente denotada por $\mathbf{\Phi_q}$. Aparte de la derivación manual, hasta ahora la única técnica para obtener estas derivadas computacionalmente era la aproximación por diferencias finitas, que tiene los citados problemas de falta de precisión por errores numéricos.

En este artículo, aparte de explicar los fundamentos de esta técnica, se ha aplicado la DA para la obtención de la Jacobiana $\mathbf{\Phi_q}$ en la simulación del robot Kawasaski ZZX, y de un vehículo turismo de 15 gdl, empleando para ello Matlab y C. El objetivo era comprobar la aplicabilidad de estas técnicas y comparar su eficiencia. La aplicación de las técnicas de DA se realiza mediante herramientas disponibles en Internet. En el caso de Matlab se empleó ADiMAT [3] (*Transformación de Código*), mientras que en C se utilizó ADOL-C [4] (*Sobrecarga del Operadores*).

## 2. Fundamentos teóricos de la Diferenciación Automática

La DA está basada en una definición informática de la función a derivar. En un caso general se puede suponer una función o programa de ordenador que produce *m* resultados agrupados en un vector **y** (variables dependientes), que dependen de *n* variables agrupadas a su vez en un vector **x** (variables independientes) y de un conjunto de parámetros, que se suponen constantes y respecto a los cuáles nunca hay que derivar. La

función informática puede contener todos los elementos habituales en cualquier lenguaje de programación: expresiones matemáticas, bifurcaciones y bucles, llamadas a funciones de librería y a otras funciones de usuario, llamadas recursivas a la propia función, etc. Sea cual sea la forma en que esté programada, esta función informática contiene la definición matemática de la función a derivar.

### 2.1. Árboles computacionales

Para formalizar el concepto de definición informática de una función matemática, se va a introducir la notación propuesta por Griewank [1], para denotar el algoritmo de descomposición computacional de una función $F : X \subset \mathbb{R}^n \to \mathbb{R}^m$, $\mathbf{y} = \mathbf{F}(\mathbf{x})$. En general $\mathbf{F}(\mathbf{x})$ será una función que se podrá descomponer en *N–n* sentencias o funciones elementales (denotadas por $f_i < x_j >$), y que dependerán de las *n* variables independientes:

$$\begin{aligned} &\text{for } i = n+1, n+2, ..., N+m \\ &\qquad x_i = f_i < x_j >_{j \in \mathcal{J}i} \\ &\text{end} \\ &\text{for } i = 1, 2, ..., m \\ &\qquad y_i = x_{N+i} \\ &\text{end} \end{aligned} \tag{1}$$

donde $\mathcal{J}_i$ es el conjunto de subíndices anteriores al de la variable considerada $x_i$:

$$\mathcal{J}_i \subset \{1, 2, ..., i-1\} \quad \text{for} \quad i = n+1, n+2, ..., N \tag{2}$$

Con esto se quiere decir que las funciones elementales sólo pueden depender de variables independientes, con subíndices $(1,2,\ldots,n)$, o variables intermedias anteriores, es decir, que han sido calculadas antes que $x_i$, con subíndices $(n+1,\ldots,i-1)$.

Las *n* variables independientes se denotan desde $x_1$ hasta $x_n$. El algoritmo consiste en una secuencia de cálculos de variables intermedias $x_i$ por medio de las funciones elementales $f_i$, $i = n+1, n+2, ..., N$. Cada $f_i$ puede depender de una o dos variables, intermedias o independientes, que tienen un índice menor que *i*. Los índices de las variables de las que $f_i$ depende pertenecen al conjunto $\mathcal{J}_i$. Las variables finales que tienen que ser calculadas son los valores de la función $\mathbf{y} = \mathbf{F}(\mathbf{x}) = \begin{bmatrix} x_{N+1} & x_{N+2} & \cdots & x_{N+m} \end{bmatrix}^T$, también llamado vector de variables dependientes. Como ya se ha dicho, las funciones

$f_i$ pueden ser operaciones elementales (suma, resta, producto, división, cambio de signo, …) entre sólo una o dos de las variables anteriores $x_j$, o llamadas a funciones de librería (seno, exponencial,…). Por ejemplo el árbol computacional de la función:

$$y = F(x) = x \cdot \sin(x) + \sin(x) \tag{3}$$

está representado en la Figura 1. Esta función se puede descomponer en las siguientes funciones elementales:

$$\begin{aligned} x_1 &\equiv x \\ x_2 &= f_2 < x_1 >= \sin(x_1) \\ x_3 &= f_3 < x_1, x_2 >= x_1 \cdot x_2 \\ x_4 &= f_4 < x_2, x_3 >= x_2 + x_3 \\ y &\equiv x_4 \end{aligned} \tag{4}$$

En este caso se manejan en total $N = 4$ variables, de las cuales $n = 1$ son independientes ($x_1$), y la función *F(X)* se descompone en $N - n = 3$ funciones elementales ($f_2, f_3$ y $f_4$)

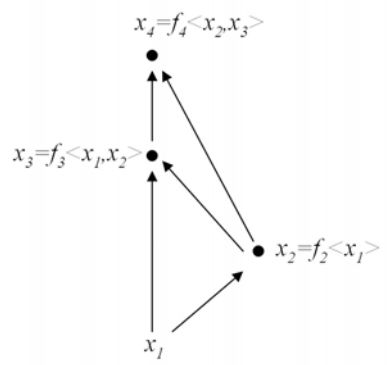


***Figura 1.*** *Representación de un árbol computacional*

Dependiendo de cómo se calcula la derivada en relación al gráfico o árbol computacional, aparecen los dos principales modos de diferenciación automática. Son los modos *forward* y *reverse* de DA [5] (en español, hacia adelante y hacia atrás, respectivamente, lo que da cierta idea de cómo se recorre el árbol computacional). A continuación sólo se va a explicar el modo *forward* por ser el más sencillo.

### 2.2. El modo *forward*

El modo *forward* de DA está basado en la aplicación literal de la regla de la cadena. Se empieza por hallar las derivadas de las propias variables independientes y se continúa propagando los cálculos de la función junto con los cálculos de las derivadas a través

del árbol computacional, aplicando las reglas básicas de diferenciación del cálculo, hasta que finalmente se obtienen las derivadas de las variables dependientes. El proceso se puede resumir por el siguiente algoritmo [1], orientado a la evaluación simultánea de la función **F** y de su derivada $\mathbf{D_F}$:

$$
\begin{aligned}
&\text{for } i = 1, 2, ..., n \\
&\qquad \mathbf{D}x_i = \mathbf{e}_i \qquad \text{//derivada de las variables independientes} \\
&\text{end} \\
&\text{for } i = n+1, n+2, ..., N \\
&\qquad x_i = f_i < x_j >_{j \in \mathcal{J}_i} \\
&\qquad \text{for } k = 1, 2, ..., n \\
&\qquad\qquad \left(\mathbf{D}x_i\right)_k = \sum_{j \in \mathcal{J}_i} \frac{\partial f_i}{\partial x_j} \left(\mathbf{D}x_j\right)_k \\
&\qquad \text{end} \\
&\text{end} \\
&\mathbf{F} = \mathbf{x_N} \\
&\mathbf{D_F} = \mathbf{D}\mathbf{x_N}
\end{aligned}
\tag{5}
$$

Para la función (3) del ejemplo anterior la función a derivar $y = F(x)$ es una función escalar **y** (variable dependiente) de una única variable independiente **x**. Esta función se debe descomponer en funciones elementales, y a partir de ellas la derivada se calcula siguiendo el algoritmo (5) de la siguiente forma:

$$
\begin{aligned}
&y = F(x) = x_1 \cdot \sin(x_1) + \sin(x_1) && \left[Dx_1 = 1\right] \\
&x_2 = f_2 < x_1 >= \sin(x_1) && \left[Dx_2 = \cos(x_1) \cdot Dx_1 = \cos(x_1)\right] \\
&x_3 = f_3 < x_1, x_2 >= x_1 \cdot x_2 && \left[Dx_3 = x_2 \cdot Dx_1 + x_1 \cdot Dx_2 = \sin(x_1) + x_1 \cdot \cos(x_1)\right] \\
&x_4 = f_4 < x_2, x_3 >= x_2 + x_3 && \left[Dx_4 = 1 \cdot Dx_2 + 1 \cdot Dx_3 = \cos(x_1) + \sin(x_1) + x_1 \cdot \cos(x_1)\right] \\
&y = x_4 && \left[D_F = Dx_4 = \cos(x_1) + \sin(x_1) + x_1 \cdot \cos(x_1)\right]
\end{aligned}
\tag{6}
$$

Como se puede ver, las operaciones siguen el orden del árbol computacional, de la raíz a las hojas, desde el principio hasta el final y por eso el nombre de modo *forward*, hacia delante.

### 3. Aplicaciones y Resultados

Como es sabido, el espacio de posibles movimientos de un sistema multicuerpo viene dado por el núcleo de la matriz Jacobiana de las ecuaciones de restricción [6], que es la derivada de las ecuaciones de restricción $\mathbf{\Phi}$ con respecto al vector de coordenadas **q** que definen la posición del sistema. La Diferenciación Automática se ha empleado para

obtener esta matriz Jacobiana $\mathbf{\Phi}_q$, comparando con la Jacobiana calculada manualmente [7]-[9]. Se han utilizado dos herramientas, una de ellas para Matlab, ADiMAT, que es del tipo de *Transformación de Código*, y otra para C, llamada ADOL-C, del tipo *Sobrecarga del Operador*. Los casos analizados han sido el robot Kawasaki ZZX y un vehículo turismo. Los tiempos de CPU para ambos casos pueden verse en la Tabla 1.

***Tabla 1.*** *Comparación de tiempos para los casos del Robot y del vehículo*

| Caso | Tamaño de la Jacobiana | Lenguaje y Herramienta DA | Tiempo derivada manual (s) | Tiempo DA (s) |
|---|---|---|---|---|
| Robot Kawasaki ZZX | 31×45 | Matlab + ADiMAT | 5.657 | 272.047 |
| | | C + ADOL-C | 1.344 | 1.172 |
| Vehículo turismo | 186×201 | Matlab + ADiMAT | 72.034 | 2385.67 |
| | | C + ADOL-C | 14.25 | 17.96 |

## 4. Conclusiones

Se ha visto que la Diferenciación Automática es una técnica válida para la obtención precisa de la Jacobiana de las ecuaciones de restricción.

En Matlab se produce una gran ineficacia que no la hace recomendable para su uso en tiempo de ejecución. Sin embargo si que puede ser de ayuda en el desarrollo de los programas, ya que proporciona resultados correctos con los que poder comparar.

Sin embargo en C con ADOL-C, no existe pérdida de eficiencia, incluso es más rápida la integración empleando DA que la derivada manual, y ello ahorrando el tiempo de desarrollo empleado en el cálculo manual de la Jacobiana. Por ello su uso está totalmente recomendado. La DA abre nuevas vías de trabajo, no sólo en simulación, también en optimización donde los mejores algoritmos requieren derivadas precisas y eficientes.

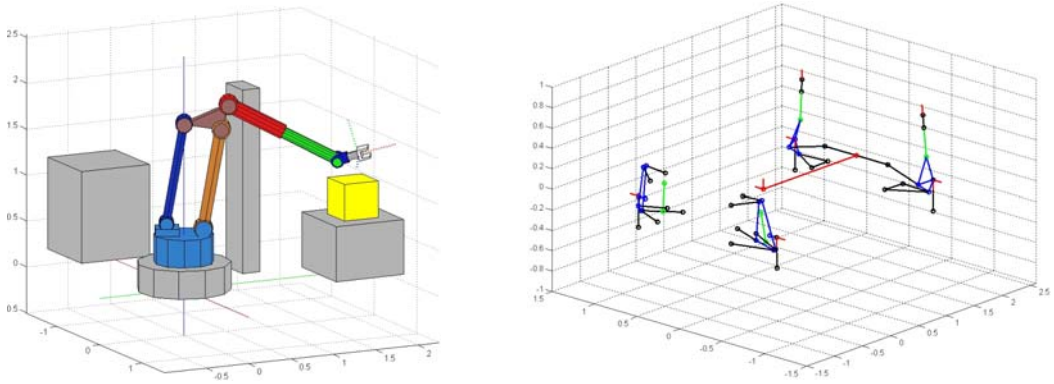

***Figura 2****. Robot Kawasaki ZZX y vehículo turismo de 15 gdl*

## 5. Referencias


[1] A. Griewank. *On automatic differentiation*, In M. and K. Tanabe editors, Mathematical Programming: Recent Developments and Applications. Kluwer Academic Publishers, Dordech, (1989), 83-108.

[2] L. B. Rall, *Automatic Differentiation: Techniques and Applications*, G. Goos and J. Hartamis editors, Lecture Notes in Computer Science. Springer-Verlag, New York, (1981)

[3] C. H. Bischof, B. Lang and A. Vehreschild, *Automatic Differentiation for Matlab Programs*, Article in Proc. Appl. Math. Mech., 2, (2003) 50-53.

[4] A. Walther, A. Kowarz and A. Griewank, *ADOL-C: A Package for the Automatic Differentiation of Algorithms Written in C/C++* Version 1.10.0, (2005).

[5] A. Griewank, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, Frontiers in Applied Mathematics, SIAM, (2000)

[6] J. García de Jalón and E. Bayo, *Kinematic and Dynamic Simulation of Multibody Systems. The Real-Time Challege*, Springer-Verlag, (1994).

[7] C.H. Bischof, *On the Automatic Differentiation of Computer Programs and An Application to Multibody Systems*, High-Performance Computing in the Geosciences, (1995), 59-80.

[8] A. Griewank, G. Corliss, C. Faure, L. Hascoet, *Automatic Differentiation of Algorithms: From Simulation to Optimization*, Springer-Verlag, (2002).

[9] P. Eberhard and W. Stadler, *Jacobian motion and its derivatives*, Pergamon, Mechatronics 11, (2000) 563-593,.