XXV CONGRESO NACIONAL DE INGENIERÍA MECÁNICA
Santander, 10-13 junio 2025
UC Universidad de Cantabria
IFToMM

Comparativa de Lenguajes de Programación para el Análisis Dinámico de Sistemas Multicuerpo: Julia vs. Matlab

Raúl Gismeros Moreno[1,2], David Zapata Castillo[1], Eduardo Corral Abad[1,2], María Jesús Gómez García[1,2], Jesús Meneses Alonso[1,2]

[1] *Grupo de Investigación MaqLab, Departamento de Ingeniería Mecánica, Universidad Carlos III de Madrid, rgismero@ing.uc3m.es, zapatacastillodavid@gmail.com, ecorral@ing.uc3m.es, mjggarci@ing.uc3m.es, meneses@ing.uc3m.es*

[2] *Instituto de Desarrollo Tecnológico y Promoción de la Innovación Pedro Juan de Lastanosa, Universidad Carlos III de Madrid*

Julia es un lenguaje de programación de alto rendimiento y código abierto que ha ganado una gran popularidad en los últimos años, tanto en el ámbito académico como en la industria, por su idoneidad para aplicaciones de análisis numérico, "data science" y "machine learning". Julia combina la velocidad propia de lenguajes de bajo nivel como C con la facilidad de aprendizaje y uso de los códigos de alto nivel como Python, convirtiéndolo en una opción atractiva para ciertas aplicaciones computacionalmente pesadas. Se considera un lenguaje "compilado just-in-time", lo que lo ubica a medio camino entre los códigos interpretados y aquellos estrictamente compilados.

Actualmente, Julia está ampliando su comunidad mayoritariamente en ámbitos científicos e investigadores debido a que su sintaxis resulta familiar para los usuarios, lo que hace que su aprendizaje sea relativamente sencillo. En este contexto, se ha planteado su aplicación al análisis dinámico de sistemas multicuerpo complejos, en los que se consideran fenómenos de contacto/impacto entre sus diferentes elementos constitutivos. Para ello, se ha adaptado un código "in-house" existente, definido en lenguaje Matlab®, utilizado por diferentes grupos de investigación, tanto nacionales como internacionales, en una amplia variedad de temas de estudio. Uno de los principales cuellos de botella identificados en la resolución de este tipo de problemas radica en la capacidad de los integradores destinados a resolver las Ecuaciones Diferenciales Ordinarias (ODE, en inglés). La librería de Julia a este respecto destaca por su alto nivel de optimización y capacidad de modificación frente a los "ODE solver" de Matlab®, que resultan más amigables para los usuarios principiantes.

En este trabajo, se han implementado varios modelos en ambos lenguajes para su comparación. Estos modelos incluyen diferentes características presentes de forma general en los sistemas multicuerpo, como pueden ser los elementos muelle-amortiguador-actuador, las juntas cinemáticas o las fuerzas de contacto/impacto. Los resultados obtenidos muestran reducciones considerables del coste computacional en Julia con respecto a Matlab®, manteniendo altos niveles de precisión.

1. Introducción

La complejidad creciente de los sistemas mecánicos actuales requiere de herramientas computacionales cada vez más robustas y precisas, que permitan simular una variedad creciente de condiciones de trabajo. Se busca obtener simulaciones cada vez más realistas y eficientes, considerando la optimización de cualquier característica que pueda tener impacto en el resultado final. Las herramientas computacionales utilizadas para simular las condiciones operativas de estos sistemas se encuentran en constante evolución, buscando mejorar la eficiencia al mismo tiempo que intentan mantener la precisión de los resultados. Esta tendencia se encuentra fuertemente influenciada por las exigencias del usuario, cada vez mayores, y que habitualmente implican la implementación de nuevas funciones que se traducen en un mayor coste computacional. Por este motivo, el estudio de cualquier posibilidad que contribuya a reducir el tiempo de computación es de interés para la industria y los investigadores.

Esto es crítico en sistemas complejos como aeronaves, barcos o aerogeneradores, los cuales están diseñados para trabajar durante largos periodos de tiempo en ambientes inhóspitos con condiciones de operación muy exigentes [1]. Todos estos ejemplos tienen una característica común: son sistemas mecánicos en los que los análisis dinámicos debe caracterizar los fenómenos de contacto de forma muy precisa. Existen diferentes técnicas para ello [2], [3], principalmente clasificadas en dos grupos: el FEM (Método de Elementos Finitos, en inglés) y la MSD (Dinámica de Sistemas Multicuerpo, en inglés) [4], [5], [6]. El FEM es considerado generalmente como la opción más poderosa en términos numéricos, siendo especialmente recomendada para requerimientos de alta precisión. Sin embargo, su alto coste computacional a la hora de modelar fenómenos de contacto/impacto puede hacerlo inadecuado para ciertos escenarios, en los cuales la MSD puede caracterizar dichos fenómenos con una precisión razonable y a un coste computacional contenido.

El presente trabajo describe el proceso de conversión de Matlab® a Julia de un código desarrollado *in-house* enfocado al análisis dinámico de sistemas multicuerpo con especial interés en la caracterización de los fenómenos de contacto/impacto. Para ello, se han definido y simulado varios modelos sencillos con diferentes funcionalidades, tanto en la versión consolidada en Matlab® como en la desarrollada en Julia. Se ha prestado especial atención a la precisión de los resultados y al coste computacional. Los resultados preliminares obtenidos muestran una mejora sustancial del tiempo de cálculo en la versión de Julia, con un grado de precisión similar o incluso mayor.

2. Metodología: sistemas multicuerpo y formulación del contacto

2.1. Sistemas multicuerpo

La formulación adoptada en este trabajo para definir las ecuaciones de movimiento en sistemas multicuerpo con juntas cinemáticas sigue la nomenclatura desarrollada por Nikravesh para sistemas planares, la cual hace uso de coordenadas generalizadas para describir la configuración de los sistemas mecánicos [7]. Un sistema multicuerpo se compone de varios cuerpos (rígidos o flexibles) que puede experimentar traslaciones y rotaciones apreciables. Si se consideran únicamente cuerpos rígidos, la posición del centro de masas de un cierto cuerpo *i* puede definirse como

$$\mathbf{r}_i = \begin{Bmatrix} x_i & y_i \end{Bmatrix}^{\mathrm{T}} \tag{1}$$

donde x_i y y_i son las coordenadas cartesianas del centro de masas del cuerpo *i* con respecto al sistema de referencia global. Así, la localización de cualquier punto *P* de un cierto cuerpo *i* puede determinarse en coordenadas del sistema local de ese mismo cuerpo

$$\mathbf{r}_i^P = \mathbf{r}_i + \mathbf{A}_i \mathbf{s}_i'^P \tag{2}$$

siendo $\mathbf{s}_i'^P$ el vector que define la posición del punto *P* con respecto al centro de masas del cuerpo *i*, en coordenadas del sistema local del cuerpo, siendo esta magnitud constante en el caso de cuerpos rígidos. $\mathbf{A}_i$ representa la matriz de rotación del cuerpo *i*, que describe la orientación de su sistema local con respecto al global. Esta matriz de transformación es función del ángulo ϕ_i, que define la orientación del cuerpo *i*. Por lo tanto, este valor, junto con las dos coordenadas definidas anteriormente, constituyen el vector de coordenadas que permite describir de forma unívoca la posición y orientación del cuerpo *i* en un espacio bidimensional

$$\mathbf{q}_i = \begin{Bmatrix} \mathbf{r}_i & \phi_i \end{Bmatrix}^{\mathrm{T}} = \begin{Bmatrix} x_i & y_i & \phi_i \end{Bmatrix}^{\mathrm{T}} \tag{3}$$

El movimiento relativo entre los cuerpos puede restringirse mediante la definición de juntas cinemáticas entre ellos. La función de estos componentes mecánicos puede representarse mediante restricciones cinemáticas de carácter holonómico, descritas mediante un conjunto de ecuaciones algebraicas de la forma

$$\boldsymbol{\Phi} \equiv \boldsymbol{\Phi}(\mathbf{q}) = \mathbf{0} \tag{4}$$

La primera derivada temporal de la Ecuación 4 da lugar a las ecuaciones de restricción a nivel de velocidades, mientras que la segunda derivada proporciona las restricciones de las aceleraciones

$$\ddot{\mathbf{\Phi}} \equiv \mathbf{D}\ddot{\mathbf{q}} + \dot{\mathbf{D}}\dot{\mathbf{q}} = \mathbf{0} \tag{5}$$

donde $\dot{\mathbf{q}}$ y $\ddot{\mathbf{q}}$ son los vectores de velocidades y aceleraciones, respectivamente, y **D** es la matriz jacobiana de dimensión $k \cdot m$, siendo k el número total de restricciones del sistema y m el número de coordenadas generalizadas (tres por cuerpo). Este conjunto de ecuaciones se suele reescribir de la forma

$$\mathbf{D}\ddot{\mathbf{q}} = \boldsymbol{\gamma} \tag{6}$$

siendo $\boldsymbol{\gamma}$ el término habitualmente denominado como "el lado derecho" de las ecuaciones de restricción a nivel de aceleraciones" (*right-hand side constraint equations*, en inglés) e incluye aquellos términos de la Ecuación 5 que dependen de forma explícita de las posiciones, las velocidades y el tiempo.

Por otro lado, las ecuaciones de movimiento de un sistema multicuerpo planar con juntas cinemática basadas en la formulación Netwon-Euler se definen como [8]

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{D}^{\mathrm{T}}\boldsymbol{\lambda} = \mathbf{g} \tag{7}$$

donde **M** es la matriz de masas del sistema, **g** representa el vector de fuerzas generalizadas que contiene todas las cargas externas (fuerzas y momentos) aplicados sobre el sistema como, por ejemplo, campos gravitacionales o fuerzas de contacto [9], y el término $\mathbf{D}^{\mathrm{T}}\boldsymbol{\lambda}$ denota las fuerzas y momentos de reacción que actúan sobre los cuerpos debido a las juntas cinemáticas, calculadas en base a los multiplicadores de Lagrange. Combinando las Ecuaciones 6 y 7 se obtiene un sistema de Ecuaciones Diferenciales Algebraicas (EDA)

$$\begin{bmatrix} \mathbf{M} & \mathbf{D}^{\mathrm{T}} \\ \mathbf{D} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \ddot{\mathbf{q}} \\ \boldsymbol{\lambda} \end{Bmatrix} = \begin{Bmatrix} \mathbf{g} \\ \boldsymbol{\gamma} \end{Bmatrix} \tag{8}$$

Este sistema de ecuaciones define la dinámica de un sistema multicuerpo con juntas cinemáticas en el que, dadas las condiciones iniciales, se obtienen los valores de $\ddot{\mathbf{q}}$ y $\boldsymbol{\lambda}$, con ellos, la evolución dinámica del sistema. En cada paso del proceso de integración, el vector de aceleraciones $\ddot{\mathbf{q}}$, junto con el de velocidades $\dot{\mathbf{q}}$, son integrados para obtener los valores de velocidad y posición que servirán de condiciones de entrada del siguiente diferencial de integración.

Sin embargo, el sistema definido en la Ecuación 8, conocido como el método estándar de los multiplicadores de Lagrange, no tiene en cuenta de forma explícita las ecuaciones de restricción de las juntas cinemáticas a niveles de posición y velocidad. Esto da lugar al incumplimiento de las ecuaciones de restricción en simulaciones más o menos largas, debido al error acumulado durante el proceso de integración y/o a condiciones iniciales imprecisas. Existen diferentes métodos para eliminar esta propagación de errores y mantener los errores de las ecuaciones de restricción a niveles de posición y velocidad dentro de unos márgenes definidos [8]. En este trabajo se ha utilizado el método de estabilización propuesto por Baumgarte para contener el error en el proceso de integración [10]. Partiendo de la expresión definida en la Ecuación 5, se añaden dos términos de control, de forma que el sistema de la Ecuación 8 queda rescrito de la forma

$$\begin{bmatrix} \mathbf{M} & \mathbf{D}^{\mathrm{T}} \\ \mathbf{D} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \ddot{\mathbf{q}} \\ \boldsymbol{\lambda} \end{Bmatrix} = \begin{Bmatrix} \mathbf{g} \\ \boldsymbol{\gamma} - 2\alpha\dot{\mathbf{\Phi}} - \beta^2\mathbf{\Phi} \end{Bmatrix} \tag{9}$$

siendo α y β los términos de control que acompañan a las desviaciones de posiciones y velocidades, respectivamente, que aparecen de forma explícita en el control del incumplimiento de las ecuaciones de restricción a nivel de aceleraciones. Se ha comprobado que se garantiza la estabilidad del sistema si los valores elegidos para los parámetros α y β son ambos positivos. Para este trabajo se han elegido el par de valores $\alpha = \beta = 5$.

2.2. Modelado de eventos de contacto en sistemas multicuerpo

Una interacción de contacto/impacto entre dos cuerpos puede ser caracterizada mediante dos tipos de estrategias: los métodos *non-smooth* y las formulaciones *smooth* [11]. Tres elementos distinguen estas dos metodologías: (i) la localización de los puntos de contacto; (ii) las ecuaciones que definen las fuerzas; y (iii) el uso del valor de la penetración [12]. Las soluciones *non-smooth* consideran el contacto como un proceso instantáneo, caracterizado por el concepto de impulso, en el que los cuerpos en contacto experimentan deformaciones muy pequeñas en comparación con su geometría. Por el contrario, los modelos *smooth* cuantifican la interacción entre los cuerpos mediante fuerzas, definidas estas como funciones continuas en el tiempo de la indentación relativa (y su derivada temporal) de los sólidos en contacto, que se consideran deformables. Estas fuerzas evitan que la penetración entre los cuerpos tenga lugar, haciendo innecesaria la definición de restricciones unilaterales. Asimismo, los métodos

smooth han demostrado funcionar bien en sistemas mecánicos con múltiples fenómenos de contacto de forma simultánea [13], [14].

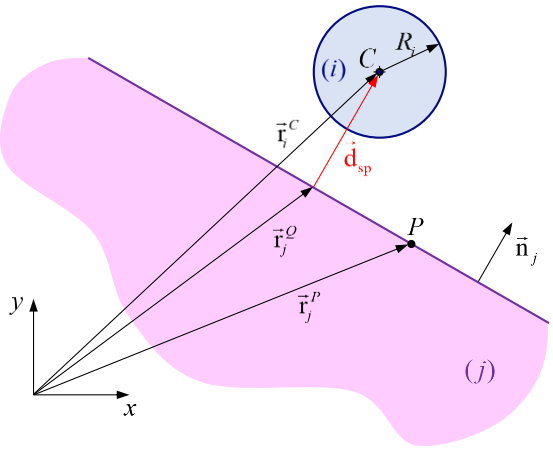


Figura 1: Parámetros asociados al algoritmo de detección del contacto entre un círculo y un plano.

Los modelos descritos en las siguientes secciones estarán basados en gran medida en los algoritmos descritos a continuación, aplicables a interacciones círculo-plano y círculo-círculo [15]. Para definir el contacto entre una superficie circular y una planar, se propone el sistema mostrado en la Figura 1, en el que se disponen un círculo *i* con centro en el punto *C* y radio *R* y un plano infinito definido por el punto *P* y su vector normal $\mathbf{n}_j$. Cualquier punto $\mathbf{r}_j$ del plano se puede obtener mediante la siguiente ecuación implícita

$$\mathbf{n}_j^{\mathrm{T}}\left(\mathbf{r}_j-\mathbf{r}_j^P\right)=0 \tag{10}$$

La distancia del centro del círculo al plano, d_{sp}, descrita en la Figura 2, se calcula de la forma

$$d_{\mathrm{sp}}=\mathbf{n}_j^{\mathrm{T}}\left(\mathbf{r}_i^C-\mathbf{r}_j^P\right) \tag{11}$$

La proyección del centro del círculo sobre el plano define el punto potencial de contacto *Q*, cuyas coordenadas en el plano *j* vienen definidas por la expresión

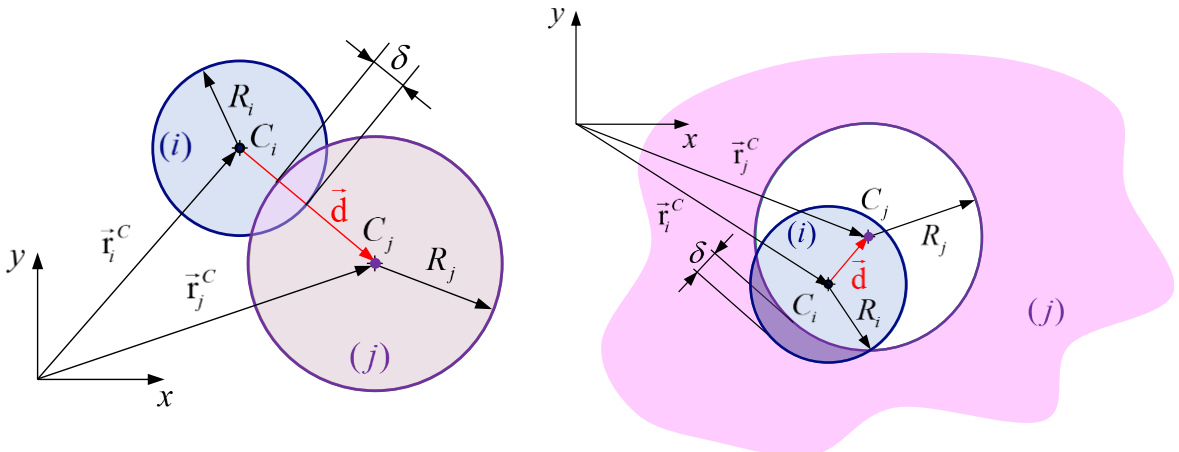
$$\mathbf{r}_j^Q=\mathbf{r}_i^C-\mathbf{d}_{\mathrm{sp}} \tag{12}$$

Por otro lado, las coordenadas del punto *Q* en el círculo *i* se calculan como

$$\mathbf{r}_i^Q=\mathbf{r}_i^C-R_i\mathbf{n}_j \tag{13}$$

Por último, se obtiene el valor de la penetración relativa entre las dos superficies, δ

$$\delta=R_i-d_{\mathrm{sp}} \tag{14}$$



(a) Contacto externo (b) Contacto interno

Figura 2: Parámetros asociados al algoritmo de detección del contacto entre dos círculos.

donde valores positivos indican que el contacto está teniendo lugar, calculándose en tal caso los valores de las velocidades relativas entre los cuerpos (normal y tangencial). El instante inicial de contacto tiene lugar cuando $\delta = 0$. Esta interacción podría ser también implementada para contactos en los que intervengan planos finitos y superficies parcialmente redondeadas.

La segunda interacción descrita corresponde al contacto entre dos superficies circulares, tal y como se muestra en la Figura 2, donde dos círculos *i* y *j*, con centros en los puntos C_i y C_j y radios R_i y R_j, respectivamente, entran en contacto. En este caso, dos posibles situaciones pueden tener lugar, según la disposición de los cuerpos: un contacto externo (Figura 2(a)) o un contacto interno (Figura 2(b)). Los puntos potenciales de contacto se determinan como

$$\mathbf{r}_i^Q = \mathbf{r}_i^C + R_i\mathbf{n} \tag{15}$$

$$\mathbf{r}_j^Q = \mathbf{r}_j^C - R_j\mathbf{n} \tag{16}$$

donde **n** es el vector unitario normal al plano de contacto entre las superficies

$$\mathbf{n} = \frac{\mathbf{d}}{d} \tag{17}$$

siendo *d* y **d** la magnitud y la dirección del vector definido entre los centros de los dos círculos, respectivamente. Este último se calcula mediante la siguiente expresión

$$\mathbf{d} = \mathbf{r}_j^C - \mathbf{r}_i^C \tag{18}$$

Finalmente, se obtiene la penetración relativa entre los dos cuerpos, distinguiendo según cada caso

$$\delta = \begin{cases} R_i + R_j - d & \text{contacto externo} \\ R_i - R_j + d & \text{contacto interno} \end{cases} \tag{19}$$

Como en el caso anterior, el contacto tiene lugar con valores positivos de la indentación, comenzando cuando $\delta = 0$.

3. Lenguajes de programación en el contexto de la dinámica multicuerpo: alternativas a Matlab®

La clasificación de los softwares dedicados a dinámica multicuerpo puede realizarse en base a diferentes criterios, siendo los más habituales su ubicación física, el tipo de licencia o el campo de aplicación. Respecto al primero, los programas pueden estar instalados localmente en los equipos de los usuarios o estar alojados en los servidores del proveedor, siendo entonces accedidos a través de distintas interfaces. Respecto a las licencias, la gama de software comercial de dinámica multicuerpo es amplia (por ejemplo, Ansys, Adams o RecurDyn), aunque muchos grupos de investigación han optado por desarrollar sus propias soluciones "in-house" [16], [17], [18], [19]. Mientras que las herramientas comerciales suelen estar desarrolladas como recursos de uso general, las propuestas desarrolladas por los investigadores suelen estar ajustadas a aplicaciones específicas [20], [21]. Los programas comerciales son especialmente ventajosos cuando se busca un tiempo de desarrollo mínimo y facilidad de uso. Sin embargo, si el usuario desea comprender a fondo la estructura interna y controlar el funcionamiento del software, así como implementar/modificar diferentes características para escenarios específicos, las soluciones "in-house" son la opción más favorable.

Centrando la atención en estos últimos, se observa en la literatura el uso de diferentes lenguajes de programación, incluyendo Fortran [17], C++ [16], Matlab [22], Python [19], y, más recientemente, Julia [23]. En los últimos tiempos, lenguajes tradicionales como Fortran o C++ se han ido dejando atrás en favor de opciones más sofisticadas y de código abierto como Julia o Python. Mientras que Python es ampliamente utilizado debido a su aplicación en contextos de IA, Julia sigue siendo relativamente desconocido para el gran público. Julia es un lenguaje de programación de código abierto y alto rendimiento que ha ganado popularidad en los últimos años, tanto en el mundo académico como en la industria, por su idoneidad para aplicaciones de análisis numérico, ciencia de datos y aprendizaje automático. Julia combina la eficiencia computacional de lenguajes de bajo nivel como C++ con la accesibilidad y facilidad de uso, características de códigos de alto nivel como Python, lo que lo convierte en una opción atractiva para determinadas tareas de cálculo intensivo. Se considera un lenguaje "compilado *just-in-time*", lo que lo ubica a medio camino entre los códigos interpretados y aquellos estrictamente compilados.

En la Tabla 1 se muestra una comparativa de los lenguajes de programación más utilizados en el contexto de la dinámica multicuerpo. Aunque Matlab® destaca por su facilidad de aprendizaje y uso en una aplicación de por sí compleja, presenta limitaciones relacionadas con su rendimiento que Julia compensa, con la ventaja de tener una

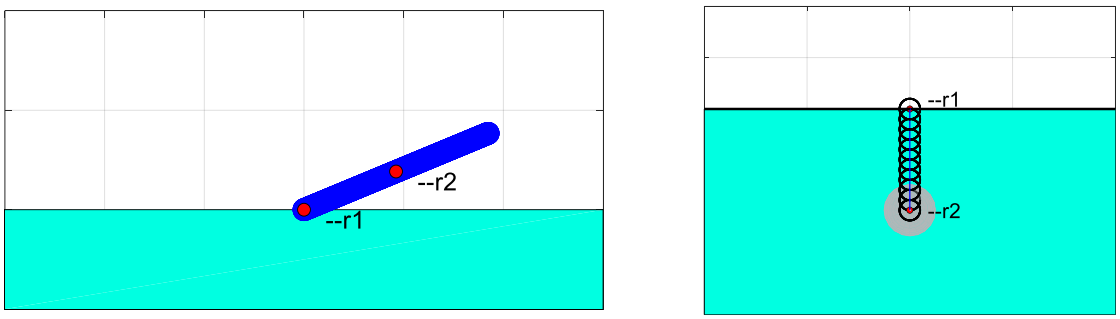
ventana de mejora mayor y ser un recurso de código abierto. Por estos motivos, se ha decidido explorar la posibilidad de migrar el código utilizado por los autores a este nuevo lenguaje.

Tabla 1: Comparativa de los principales lenguajes utilizados en dinámica multicuerpo computacional.

Lenguaje	Ventajas	Desventajas
Fortran	▪ Optimización para cálculos numéricos intensivos. ▪ Gestión eficaz de matrices. ▪ Amplia trayectoria ▪ Ejecución estable y previsible	▪ Sintaxis y ecosistema obsoletos. ▪ Curva de aprendizaje pronunciada. ▪ Comunidad actual de usuarios limitada. ▪ Soporte limitado de Programación Orientada a Objetos (POO).
C++	▪ Alto rendimiento y optimización de recursos. ▪ Altas capacidades para POO. ▪ Opción de computación en paralelo.	▪ Curva de aprendizaje pronunciada ▪ Sintaxis compleja. ▪ Gestión de memoria manual, lo que afecta al rendimiento.
Matlab	▪ Altamente amigable para el usuario. ▪ Herramientas de visualización de alta calidad. ▪ Cálculo simbólico y numérico. ▪ Amplia biblioteca de *ODE solvers*.	▪ Más lento que los lenguajes compilados. ▪ Licencia de pago. ▪ Capacidad de computación en paralelo limitada.
Python	▪ Fácil aprendizaje y uso. ▪ Comunidad de usuarios grande. ▪ Integración sencilla con otros lenguajes como Fortran o C++.	▪ Menor velocidad de ejecución comparado con lenguajes compilados. ▪ Capacidad de simulación en tiempo real limitada. ▪ Alta dependencia en bibliotecas externas para los *ODE solvers*.
Julia	▪ Alto rendimiento y optimización de recursos. ▪ Computación en paralelo integrada. ▪ *ODE solvers* altamente optimizados.	▪ Ecosistema relativamente joven. ▪ Tiempos de arranque mayores en comparación con lenguajes pre-compilados. ▪ Comunidad de usuarios reducida.

El software, conocido como MUBODYNA (acrónimo de MUltiBOdy DYNAmics), ha sido utilizado por diferentes grupos de investigación en distintas aplicaciones [14], [22], [24]. Este código tiene su origen en la herramienta DAP3D, desarrollada en lenguaje Fortran por Nikravesh [25]. Uno de los principales cuellos de botella detectados durante su funcionamiento actual radica en la capacidad de los integradores propios de Matlab® para resolver las Ecuaciones Diferenciales Ordinarias (ODE, en inglés) en determinadas aplicaciones. Este problema es especialmente notable en escenarios de contacto/impacto múltiple-simultáneo, en los que la elección y el ajuste adecuados de estos *ODE solvers* se vuelven críticos para la obtención de resultados precisos a un coste computacional contenido. En este sentido, la biblioteca de Julia destaca por su alto nivel de optimización y capacidad de modificación en comparación con los integradores de Matlab®, que son más fáciles de utilizar para los usuarios noveles.

Actualmente, Julia está expandiendo su comunidad de usuarios principalmente en campos de investigación debido a su sintaxis amigable. En este contexto, se ha explorado su aplicación al análisis dinámico de sistemas multicuerpo complejos con fenómenos de contacto. Para ello, se ha decidido realizar el proceso de migración de una versión 2D de MUBODYNA de Matlab® a Julia. Los principales aspectos que se han considerado durante la conversión han sido la adaptación del proceso de definición de variables y estructuras globales y la modificación de algunos de los scripts existentes para adecuarlos a las características propias de este lenguaje (por ejemplo, la sentencia *switch* no está definida por defecto en Julia, sino que debe implementarse a través de paquetes adicionales). En esta fase, la correcta comprensión, tanto de la definición de las ecuaciones de Newton-Euler, incluyendo la formulación de las expresiones de las ecuaciones de restricción, como de la evaluación y el cálculo de las fuerzas de contacto, fueron fundamentales para completar la traducción al nuevo lenguaje con éxito.

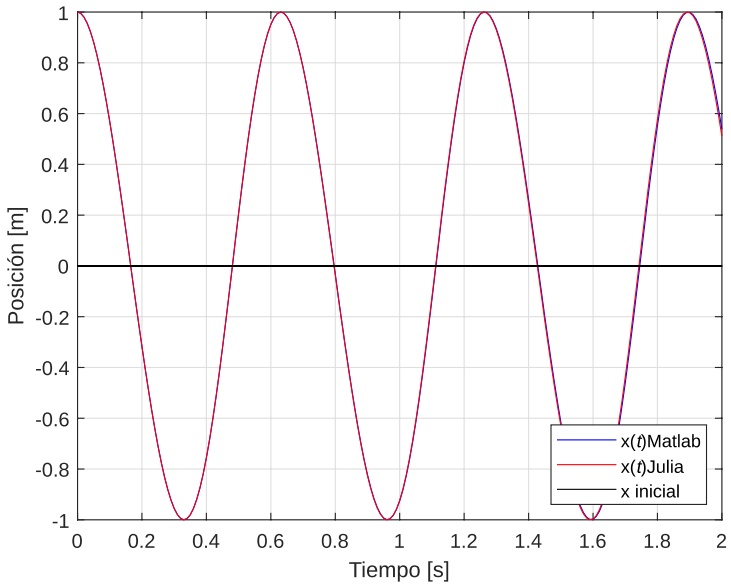


(a) (b)

Figura 3: Modelos de prueba desarrollados: (a) Junta cinemática de revolución; (b) Elemento muelle.

4. Modelos de prueba desarrollados

En una primera fase se optó por definir modelos relativamente sencillos, con distintas funcionalidades, buscando validar la correcta implementación de todas las características existentes en la versión consolidada de Matlab®. En la Figura 3 se muestran los dos modelos de ajuste desarrollados. En el primero, se buscó verificar la correcta definición de las ecuaciones de restricción y el cálculo de las fuerzas asociadas a las reacciones en la junta cinemática, mientras que en el segundo se comprobó la consistencia de los resultados con respecto a la versión original.



(a)

```
Which folder contains the model? juntarev_ajuste

The initial conditions were not corrected!
The final time of simulation is: 2 s
The reporting time step is: 0.01 s

Solving the equations of motion using standard method with Baumgarte ...
Integrator used: ODE45 ...

The number of function evaluations is: 235
The CPU time consumed was: 31.1875 s

Normal termination of RDYNA program!
```

(b)

```
Which folder contains the model?

The final time of simulation is: 2 s
The reporting time step is: 0.01 s

   Solving the equations of motion using standard method with Baumgarte ...
   Integrator used: ODE45 ...

  3.762820 seconds (7.09 M allocations: 476.080 MiB, 4.12% gc time, 99.25% compilation time)
La función tomó 14.2303896  segundos en ejecutarse.
```

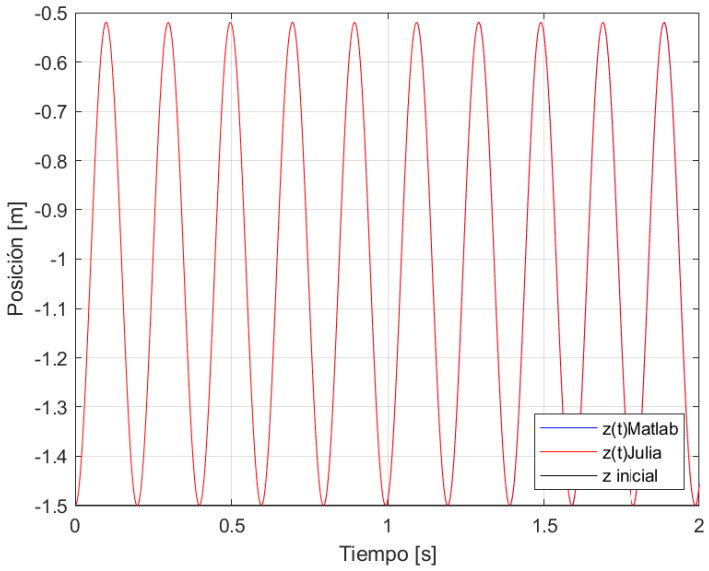
(c)

Figura 4: Resultados obtenidos del primer modelo de ajuste: (a) Evolución de la coordenada *x* del centro de masas de la manivela; (b) Coste computacional en Matlab®; (c) Tiempo de cálculo empleado por Julia.

En la Figura 4 se muestran tanto la comparativa de los resultados para el primer modelo, comparando la evolución de la trayectoria del centro de masas de la manivela a lo largo del análisis, para el cual se ha fijado un tiempo de integración de 2 s, con un diferencial de las salidas cada 0,01 s. Como esquema de integración se ha elegido el método de Runge-Kutta (4,5), el cual en Matlab se implementa a través del *solver* "ode45", mientras que en Julia la función empleada es "Tsit5()". Como se puede observar, los resultados obtenidos son prácticamente idénticos,

confirmando la correcta conversión del código, mientras que el coste computacional se ha reducido en más del 50%.

Una mejora aún mayor se ha confirmado con el segundo modelo de prueba (ver Figura 5), en el cual se coloca una bola en el extremo de un muelle dispuesto verticalmente, el cual se encuentra inicialmente comprimido. La evolución de la coordenada vertical del centro de masas de la bola es prácticamente idéntica, mientras que el coste computacional se ha reducido en torno a un 75%.



(a)

```
Which folder contains the model? muelle_ajuste

The final time of simulation is: 2 s
The reporting time step is: 0.001 s

Solving the equations of motion using standard method with Baumgarte ...
Integrator used: ODE45 ...

The number of function evaluations is: 511
The CPU time consumed was: 61.9375 s

Normal termination of RDYNA program!
```

(b)

```
Which folder contains the model?
The final time of simulation is: 2 s
The reporting time step is: 0.001 s

   Solving the equations of motion using standard method with Baumgarte ...
   Integrator used: ODE45 ...

  3.896597 seconds (6.41 M allocations: 405.511 MiB, 4.23% gc time, 89.72% comp
ilation time)
La función tomó 15.1975338  segundos en ejecutarse.
```

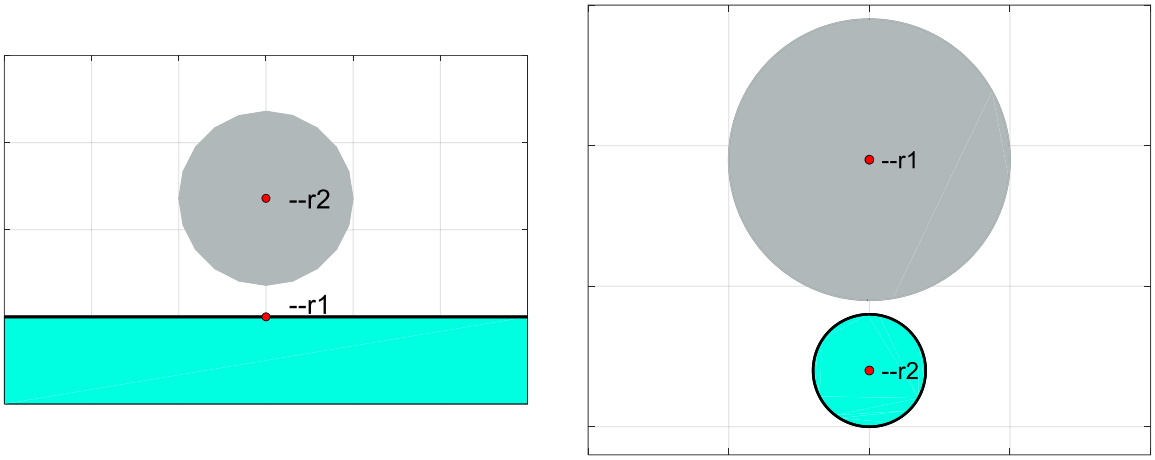
(c)

Figura 5: Resultados obtenidos del segundo modelo de ajuste: (a) Evolución de la coordenada *z* del centro de masas de la bola; (b) Coste computacional en Matlab®; (c) Tiempo de cálculo empleado por Julia.

5. Modelos con fuerzas de contacto

Una vez comprobada la correcta conversión de las diferentes funcionalidades del código desarrollado, se procedió a ensayar modelos con fuerzas de contacto. El estudio de sistemas con estas características es de especial interés, ya que suelen ser modelos computacionalmente más exigentes, especialmente cuando se implementan varias de estas interacciones de forma simultánea. Los integradores utilizados, tanto de Matlab® como de Julia, suelen emplear pasos de integración variables, de forma que el *step* se reduce automáticamente en momentos críticos del

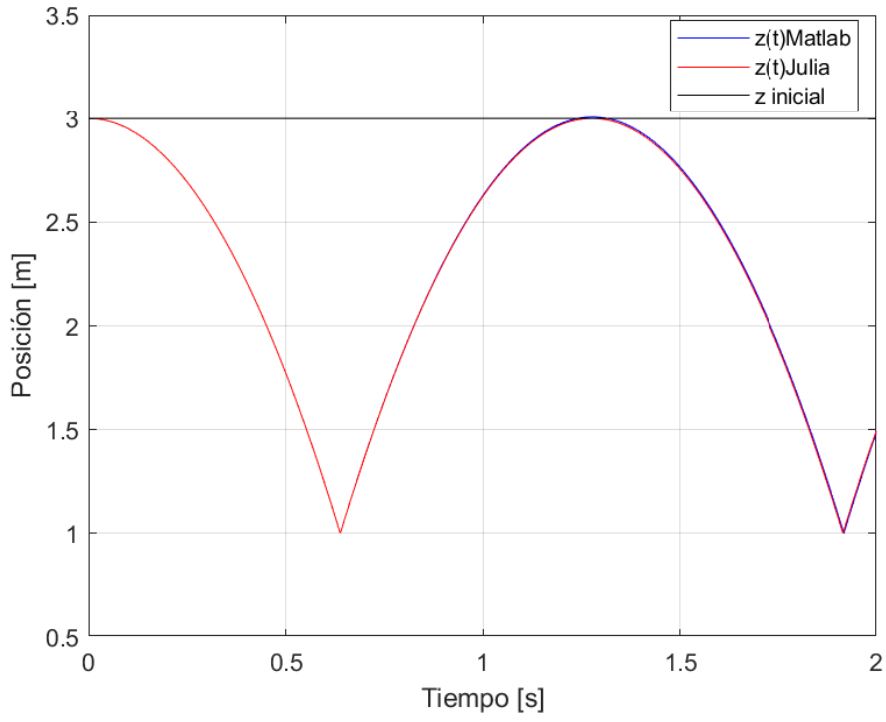
análisis, como puede ser el inicio de un contacto entre dos cuerpos. Si el sistema es demasiado rígido (*stiff*), el integrador puede reducir de forma excesiva el paso de integración en dichos instantes, haciendo que el análisis se ralentice o incluso se estanque [26]. Este problema es aún más crítico en sistemas con múltiples eventos de contacto de forma simultánea, en los que la configuración de cada interacción afecta a las demás [14], [18]. En una primera fase, se ha decidido ensayar las interacciones más elementales, correspondientes a las descritas en la Sección 2.2. En la Figura 6 se muestran los dos modelos testados. En ambos casos, la fuerza de contacto utilizada corresponde al modelo de Hertz no disipativo [11]. Además, se ha omitido la existencia de fuerzas de fricción y se han considerado los efectos de la gravedad.



(a) (b)

Figura 6: Modelos con fuerzas de contacto ensayados: (a) Interacción círculo-plano; (b) Contacto entre dos círculos.

En la Figura 7 se representan los resultados obtenidos para el modelo círculo-plano, para un análisis de 2 segundos, con un diferencial entre salidas de 0,001 segundos. Como se puede observar, para una configuración por defecto de los parámetros del *solver*, se observa una reducción del tiempo de cálculo superior al 60%. Además, se advierte una ligera inconsistencia en el comportamiento del modelo ensayado en Matlab®, pues el círculo supera la altura inicial tras el primer choque con la superficie. Por otro lado, se observa también un ligero desfase entre ambos casos, lo que sugiere que los integradores de Matlab® y Julia trabajan de forma distinta a la hora de tratar fenómenos de contacto. Se decidió ampliar el tiempo de análisis hasta los 20 segundos, manteniendo el valor del diferencial entre salidas, obteniendo unos tiempos de cálculo para Matlab® y Julia de 449 s y 37 s, respectivamente.



(a)

```
Which folder contains the model? circle_plane

The final time of simulation is: 2 s
The reporting time step is: 0.001 s

Solving the equations of motion using standard method with Baumgarte ...
Integrator used: ODE45 ...

The number of function evaluations is: 439
The CPU time consumed was: 56.0156 s

Normal termination of RDYNA program!
```

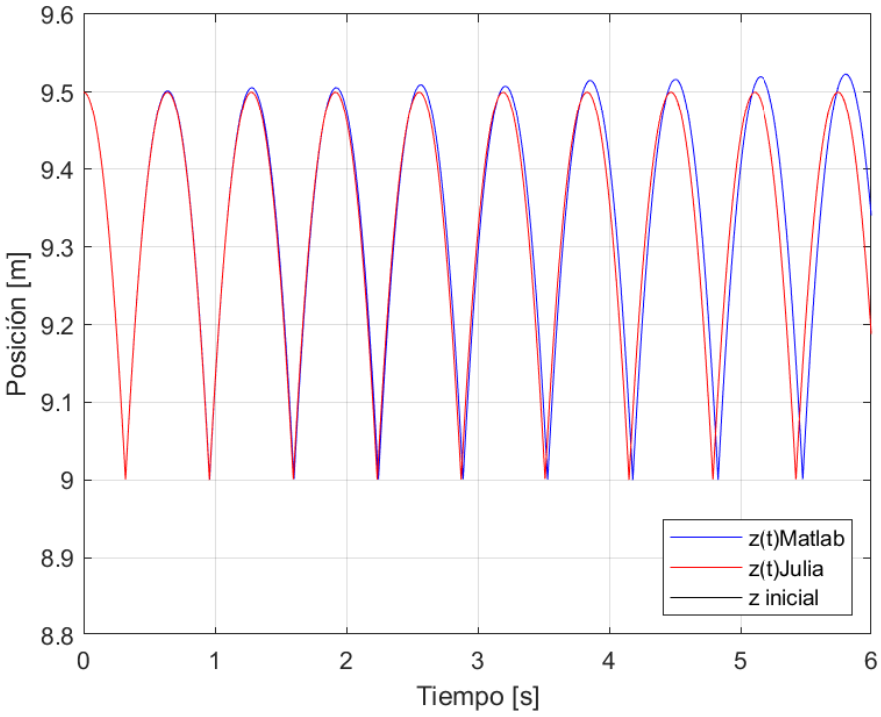
(b)

```
Which folder contains the model?
The final time of simulation is: 2 s
The reporting time step is: 0.001 s

   Solving the equations of motion using standard method with Baumgarte ...
   Integrator used: ODE45 ...

  3.599082 seconds (5.78 M allocations: 375.807 MiB, 4.06% gc time, 95.59% compilation time)
La función tomó 15.5319268  segundos en ejecutarse.
```

(c)

Figura 7: Resultados obtenidos con el primer modelo de fuerzas de contacto (círculo-plano): (a) Evolución de la coordenada *z* del centro del círculo; (b) Coste computacional en Matlab®; (c) Tiempo de cálculo empleado por Julia.

Por último, en la Figura 8 se muestran los resultados obtenidos para el modelo círculo-círculo, en este caso para una simulación de 6 s y un step entre salidas de 0,001 s, en la que se ha fijado el círculo inferior, mientras se deja caer el superior. El comportamiento observado es similar al del primer modelo, con una propagación de las tendencias descritas anteriormente: a medida que se van produciendo los sucesivos impactos, el círculo sobrepasa la posición inicial desde la cual se deja caer. Esto indica que, para la configuración por defecto de los integradores, Matlab® proporciona resultados ligeramente peores que Julia. Respecto al tiempo de cálculo, la mejora es más que considerable. De forma similar al modelo anterior, se ha decidido repetir el análisis, duplicando el tiempo de simulación hasta los 12 s y manteniendo el diferencial de las salidas. En este nuevo ensayo, los costes computacionales de Matlab® y Julia han sido de 465 s y 17 s, respectivamente.



(a)

```
Which folder contains the model? circle_circle

The final time of simulation is: 6 s
The reporting time step is: 0.001 s

Solving the equations of motion using standard method with Baumgarte ...
Integrator used: ODE45 ...

The number of function evaluations is: 1981
The CPU time consumed was: 227.2969 s

Normal termination of RDYNA program!
```

(b)

```
Which folder contains the model?
The final time of simulation is: 6 s
The reporting time step is: 0.001 s

   Solving the equations of motion using standard method with Baumgarte ...
   Integrator used: ODE45 ...

  3.753612 seconds (7.21 M allocations: 447.994 MiB, 4.31% gc time, 86.58% compilation time)
La función tomó 15.1915769  segundos en ejecutarse.
```

(c)

Figura 8: Resultados obtenidos con el segundo modelo de fuerzas de contacto (círculo-círculo): (a) Evolución de la coordenada *z* del centro del círculo gris; (b) Coste computacional en Matlab®; (c) Tiempo de cálculo empleado por Julia.

No obstante, es necesario realizar un análisis más exhaustivo de los resultados obtenidos para confirmar que la mejora observada es real, para lo cual se debería lanzar el post-procesado de las simulaciones, de forma que se puedan estudiar los valores de las fuerzas experimentadas por los diferentes cuerpos del sistema y las variaciones de energía de los mismos. Una opción a valorar podría ser la exportación de los resultados generados en Julia para realizar el post-procesado en Matlab®. Por otro lado, sería de interés el ensayo de un modelo más complejo, con múltiples eventos de contacto de forma simultánea, para comprobar la robustez de los *ODE solver* de Julia.

6. Conclusiones

En este trabajo se ha abordado el estudio de diversos modelos multicuerpo con distintas funcionalidades como juntas cinemáticas, elementos tipo muelle o fuerzas de contacto, con el objetivo de comparar el rendimiento de los lenguajes Julia y Matlab®. Los resultados obtenidos muestran que Julia presenta un rendimiento computacional significativamente mejor en comparación con Matlab®, especialmente en modelos que involucran fuerzas de contacto. Esta mejora se debe, en parte, a la mayor flexibilidad de los *ODE solver* de Julia, que permiten un mejor ajuste de los parámetros de integración para cada problema específico. Además, se ha observado que Julia es capaz de mantener la precisión de los resultados incluso con pasos de integración por defecto más grandes, lo que se traduce en una reducción del tiempo de cálculo. En el caso de Matlab®, se ha detectado una ligera inconsistencia en los resultados para configuraciones por defecto de los parámetros de integración, lo que sugiere que los integradores de Julia podrían ser más robustos ante la presencia de fuerzas de contacto. En cualquier caso, sería necesario un análisis más exhaustivo de los resultados para confirmar que la mejora observada es real. Por otro lado, sería de interés el ensayo de un modelo más complejo, con múltiples eventos de contacto de forma simultánea, para comprobar la robustez de los *ODE solver* de Julia en aplicaciones de mayor complejidad.

7. Agradecimientos

Esta publicación ha sido desarrollada como parte de los proyectos MEMRIAAP-CM-UC3M (financiado por la Comunidad de Madrid), PID2020-116984RB-C22 y TED2021-131372A-I00 (financiados por el Ministerio de Ciencia e Innovación a través de la Agencia Estatal de Investigación).

8. Referencias

[1] Tao, F., Zhang, M., Liu, Y. y Nee, A. Y. C., "Digital twin driven prognostics and health management for complex equipment", *CIRP Annals* **67**, 169-172 (2018).

[2] Altintas, Y., Brecher, C., Weck, M. y Witt, S., "Virtual Machine Tool", *CIRP Annals* **54**, 115-138 (2005).

[3] Kim, C.-J., Oh, J.-S., Park, C.-H. y Lee, C.-H., "Fast flexible multibody dynamic analysis of machine tools using modal state space models", *CIRP Annals* **72**, 341-344 (2023).

[4] Ebrahimi, S. y Eberhard, P., "Aspects of Contact Problems in Computational Multibody Dynamics", *Multibody Dynamics: Computational Methods and Applications*, *vol. 2*, Springer Netherlands, Dordrecht, 23-47 (2007).

[5] Rui, X., Bestle, D., Zhang, J. y Zhou, Q., "A new version of transfer matrix method for multibody systems", *Multibody Syst Dyn* **38**, 137-156 (2016).

[6] Koutsoupakis, J., Giagopoulos, D. y Chatziparasidis, I., "AI-based condition monitoring on mechanical systems using multibody dynamics models", *Eng Appl Artif Intell* **123**, 106467 (2023).

[7] Nikravesh, P. E., *Planar Multibody Dynamics: Formulation, Programming with MATLAB®, and Applications*, CRC Press, Boca Raton (2018).

[8] Flores, P., *Concepts and Formulations for Spatial Multibody Dynamics*, Springer International Publishing, Cham (2015).

[9] Flores, P., Ambrósio, J., Pimenta Claro, J. C. y Lankarani, H. M., "Kinematics and Dynamics of Multibody Systems with Imperfect Joints", *Lecture Notes in Applied and Computational Mechanics, vol. 34*, Springer Berlin Heidelberg, Berlin, Heidelberg (2008).

[10] Baumgarte, J., "Stabilization of constraints and integrals of motion in dynamical systems", *Comput Methods Appl Mech Eng* **1**, 1-16 (1972).

[11] Corral, E., Gismeros Moreno, R., Gómez García, M. J. y Castejón, C., "Nonlinear phenomena of contact in multibody systems dynamics: a review", *Nonlinear Dyn* **104**, 1269-1295 (2021).

[12] Machado, M., Moreira, P., Flores, P. y Lankarani, H. M., "Compliant contact force models in multibody dynamics: Evolution of the Hertz contact theory", *Mech Mach Theory* **53**, 99-121 (2012).

[13] Gismeros Moreno, R., Corral Abad, E., Meneses Alonso, J., Gómez García, M. J. y Castejón Sisamón, C., "Modelling multiple-simultaneous impact problems with a nonlinear smooth approach: pool/billiard application", *Nonlinear Dyn* **107**, 1859-1886 (2022).

[14] Gismeros Moreno, R., Marques, F., Corral Abad, E., Meneses Alonso, J., Flores, P. y Castejón, C., "Enhanced modelling of planar radial-loaded deep groove ball bearings with smooth-contact formulation", *Multibody Syst Dyn* **60**, 121-159 (2024).

[15] Corral, E., Gismeros Moreno, R., Meneses, J., Gómez García, M. J. y Castejón, C., "Spatial Algorithms for Geometric Contact Detection in Multibody System Dynamics", *Mathematics* **9**, 1359 (2021).

[16] Mazhar, H., et al., "CHRONO: A parallel multi-physics library for rigid-body, flexible-body, and fluid dynamics", *Mechanical Sciences* **4**, 49-64 (2013).

[17] Zhu, Y., Dopico, D., Sandu, C. y Sandu, A., "MBSVT: Software for Modeling, Sensitivity Analysis, and Optimization of Multibody Systems at Virginia Tech", *Proceedings of the ASME 2014 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. *2nd Biennial International Conference on Dynamics for Design; 26th International Conference on Design Theory and Methodology* **7**, Buffalo, Nueva York, EE.UU. (2014).

[18] Gismeros Moreno, R., Corral Abad, E., Meneses Alonso, J., Gómez García, M. J. y Castejón Sisamón, C., "Modelling multiple-simultaneous impact problems with a nonlinear smooth approach: pool/billiard application", *Nonlinear Dyn* **107**, 1859-1886 (2022).

[19] Gerstmayr, J., "Exudyn – a C++-based Python package for flexible multibody systems", *Multibody Syst Dyn* **60**, 533-561 (2024).

[20] Rajagopal, A., Dembia, C. L., DeMers, M. S., Delp, D. D., Hicks, J. L. y Delp S. L., "Full-Body Musculoskeletal Model for Muscle-Driven Simulation of Human Gait", *IEEE Trans Biomed Eng* **63**, 2068-2079 (2016).

[21] Bosso, N., Magelli, M. y Zampieri, N., "Development and validation of a new code for longitudinal train dynamics simulation", *Proc Inst Mech Eng F J Rail Rapid Transit* **235**, 286-299 (2021).

[22] Corral, E., et al., "Dynamic Modeling and Analysis of Pool Balls Interaction", *Multibody Dynamics 2019. ECCOMAS 2019. Computational Methods in Applied Sciences* **53**, 79-86, Cham (2020).

[23] Verulkar, A., Sandu, C., Sandu, A. y Dopico, D., "Simultaneous optimal system and controller design for multibody systems with joint friction using direct sensitivities", *Multibody Syst Dyn*, 1-31 (2024).

[24] Rodrigues da Silva, M., Marques, F., Tavares da Silva, M. y Flores, P., "An improved methodology to restrict the range of motion of mechanical joints", *Nonlinear Dyn* **112**, 4227-4256 (2024).

[25] Nikravesh, P. E., *Computer-aided Analysis of Mechanical Systems*, Prentice-Hall (1988).

[26] Flores, P. y Ambrósio, J., "On the contact detection for contact-impact analysis in multibody systems", *Multibody Syst Dyn* **24**, 103-122 (2010).