uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Mecánica
2017-2018

Trabajo Fin de Grado

Dinámica por ordenador con Matlab.

Sergio Villar Alegría

Tutor:

Eduardo Corral Abad

Título: DINÁMICA POR ORDENADOR CON MATLAB.

Autor: Sergio Villar Alegría.

Tutor: Eduardo Corral Abad.

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Trabajo Fin de Grado el día ___ de _____ de 20___ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

SECRETARIO VOCAL PRESIDENTE

AGRADECIMIENTOS:

"A mis hermanos, Belén y Jaime, por ser mi vela, mi fuerza cuándo flaqueé, mis ganas de seguir. A Eduardo, por ser el viento, impulsándome con su ayuda y ánimos, siempre a mi lado cuando lo necesité. A Leticia, por ser la estrella que me guía. Este éxito no es mío, es nuestro."

RESUMEN.

El objetivo de este Trabajo de Fin de Grado en Ingeniería Mecánica, es el estudio y desarrollo de problemas mecánicos mediante ordenador. Para esto, se ha estudiado los diferentes métodos y soluciones que hoy en día se conocen, para modelar problemas físicos de cinemática y dinámica en lenguaje de programación de Matlab. Uniéndome al desarrollo de Mubodyna, un programa de ingeniería asistida por ordenador creado en la Universidad Carlos III, que permite el estudio y diseño de diversos problemas de sistemas mecánicos.

A su vez, se ha creado un manual para futuros estudiantes o investigadores, que pretende facilitar los primeros pasos en la dinámica de mecanismos por ordenador, y permita un aprendizaje sencillo y progresivo. Por último, se han desarrollado diversos sistemas mecánicos en Mubodyna, realizado simulaciones y analizado los resultados obtenidos, comprobando la correcta implementación de los problemas y el ajuste de las simulaciones a la realidad.

Palabras clave: Dinámica de mecanismos » Modelado » Simulación » Análisis.

ABSTRACT.

The goal on this project is the study and development of multibody systems by computer. To do this, we have studied different methods and solutions known to date, aiming to model cinematic and dynamic problems by coding it in Matlab. Joining to the development of Mubodyna, a CAE program created in Carlos III University, which allows the study and design of several mechanical problems.

Moreover, we have created a manual for future students and researchers, in order to make their first steps into multibody dynamics easier and progressive. In the end, several dynamic problems have been developed, creating simulations and analysing the obtained results, verifying the correct implementation of these problems and the approximation of the simulations to the reality.

Keywords: Multibody Dynamics » Modelling » Simulation » Analysis.

ÍNDICE DE CONTENIDOS:

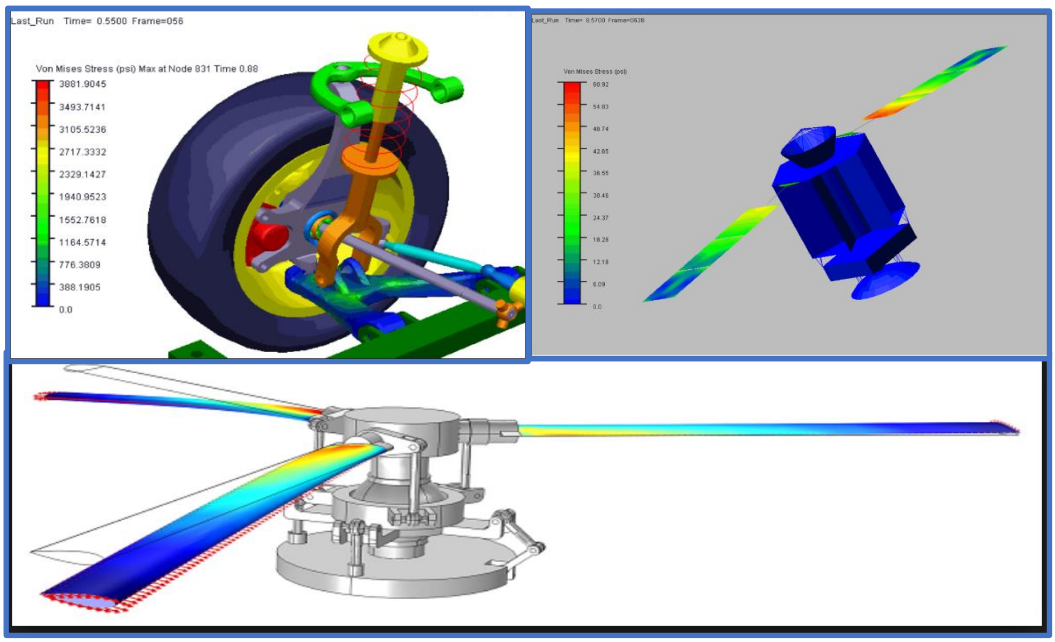
CAPÍTULO 0. MOTIVACIÓN Y OBJETIVOS.

Motivación:

El objetivo del proyecto es el estudio de problemas cinemáticos y dinámicos de Multibody Systems (sistemas mecánicos). Esto se hará mediante MUBODYNA, un programa implementado en Matlab por la universidad Carlos III, así como la creación de un documento que servirá como manual a futuros estudiantes o investigadores que quieran iniciarse en el trabajo y desarrollo de este programa CAD.

La cinemática y dinámica de mecanismos forman una parte muy importante del llamado CAD (Computer Aided Desing) y MCAE (Mechanical Computer Aided Engineering). Los sistemas denominados multicuerpo comprenden cantidad de sistemas mecánicos; sistemas de automovilismo como suspensiones o sistemas de dirección, robots, biomecánica, maquinaria pesada, sistemas aeroespaciales, maquinaria textil, herramientas de mecanizado automático, etc. El desarrollo de todos estos sistemas, así como sus velocidades de trabajo, fuerzas y pares capaces de desarrollar son motivo de una constante aparición de nuevos problemas dinámicos que deben ser controlados y analizados.

Los sistemas CAD y MCAE ofrecen la gran ventaja de predecir el comportamiento cinemático y dinámico de sistemas multicuerpo, mediante el análisis de simulaciones, desde los primeros conceptos de diseño, hasta los últimos prototipos más avanzados. En cualquier fase de diseño, los programas de análisis a ordenador pueden nutrir con gran cantidad de información al ingeniero en su estudio de los diferentes parámetros de diseño, logrando gran cantidad de simulaciones de una forma eficiente y económica.



1. Diseño en MCAE de sistema amortiguador, despliegue de satélite y hélices.

Estos programas de análisis simulan el comportamiento de los sistemas mecánicos una vez definidas las variables geométricas y dinámicas del problema. Estos programas

son hoy en día una de las herramientas de trabajo más importantes para un largo rango de problemas que la ingeniería debe resolver. A lo largo del proyecto, se irán desarrollando los puntos más importantes del análisis de problemas, desde los fundamentos teóricos y la modelización de problemas, hasta la simulación y obtención de datos con MUBODYNA.

Objetivos del proyecto.

El objetivo de este proyecto es el de ayudar al desarrollo de Mubodyna, un programa de análisis dinámico para sistemas mecánicos en Matlab. Para ello, se deberán estudiar y comprender los fundamentos físicos que rigen los problemas cinemáticos y dinámicos, para posteriormente familiarizarse con los procesos de modelado e implementación en Matlab. Tras esto, se crean estructuras de programa, a través de las cuales podremos hacer análisis y simulación de gran variedad de problemas que se presentan en diversos sistemas mecánicos industriales. Además, se crearán unos modelos que contienen gran variedad de problemas a los que se enfrenta la dinámica por ordenador, y se resolverán analizando los resultados y animaciones.

Además, el proyecto se ha estructurado de tal forma que pretende servir de guía para futuros alumnos o investigadores. Sea su objetivo, su objetivo el de continuar con el desarrollo del programa Mubodyna, o el de usarlo como herramienta para el desarrollo de otras metas, he pretendido que este proyecto sirva de puente, y facilite la introducción en este campo a cualquier persona que desee.

Resumen de objetivos:

- ✓ *Comprensión y estudio de problemas dinámicos y métodos de modelado por ordenador.*
- ✓ *Análisis y simulación de varios sistemas mecánicos con Mubodyna.*
- ✓ *Realización de simulaciones dinámicas con contacto/impacto.*
- ✓ *Ayudar al desarrollo e implementación de Mubodyna.*
- ✓ *Creación de guía/manual para futuros investigadores.*

CAPÍTULO I: FUNDAMENTOS.

1.0 Dinámica directa e inversa

La *dinámica directa* es aquella que, a partir de unas fuerzas aplicadas en el sistema, determina el movimiento del mismo. Por el contrario, la *dinámica inversa* es la que a partir de las ecuaciones de movimiento de un sistema y posiblemente alguna fuerza, pretende calcular las fuerzas que están actuando sobre el sistema.

1.1 Multibody Systems and Joints

Un *sistema mecánico* o *multibody system* se define como un conjunto formado por dos o más cuerpos rígidos o flexibles (también llamados elementos) unidos mediante *juntas* o *joints,* que permiten el movimiento relativo entre ellos. Esta unión de cuerpos rígidos es llamada *par cinemático.*

Los cuerpos del sistema, pueden ser considerados rígidos o flexibles. Se dice que un cuerpo es rígido cuando sus deformaciones son tan pequeñas que no afectan al movimiento general del cuerpo. Por tanto, podrá desplazarse o girar, pero en ningún caso deformarse.

El grado de libertad de un sistema mecánico es el número mínimo de coordenadas independientes necesarias para definir el sistema. Para un elemento libre en dos dimensiones el grado de libertad es 3, y en tres dimensiones es 6. Correspondientes a los desplazamientos y giros que estos pueden realizar. Para cuerpos flexibles será necesario incluir otras coordenadas generalizadas para definir las deformaciones. Aunque la naturaleza de los cuerpos nunca es absolutamente rígida, en una gran cantidad de casos los cuerpos son lo suficientemente rígidos como para despreciar las deformaciones, lo cual facilita enormemente el proceso de modelado de sistemas mecánicos. Por tanto y en este proyecto, analizaremos problemas suponiendo cuerpos perfectamente rígidos.

Según el tipo de par cinemático que une los cuerpos, estos permiten un cierto número de grados de libertad y a su vez, restringe otros. Se pueden clasificar en función de los grados de libertad que permiten, de este modo, un par de clase 1 tendrá un grado de libertad, uno de clase 2 permitirá dos grados de libertad y así sucesivamente. A su vez, el sistema podrá verse sometido a fuerzas sobre sus diversos componentes, debidas a elementos como muelles, amortiguadores, actuadores, motores o fuerzas externas. Fuerzas externas de diversa complejidad podrán actuar en nuestro sistema mecánico, con el fin de simular las interacciones entre nuestro sistema y los elementos que lo rodean.

Algunos de los principales pares cinemáticos usados en sistemas mecánicos y sus grados de libertad son:

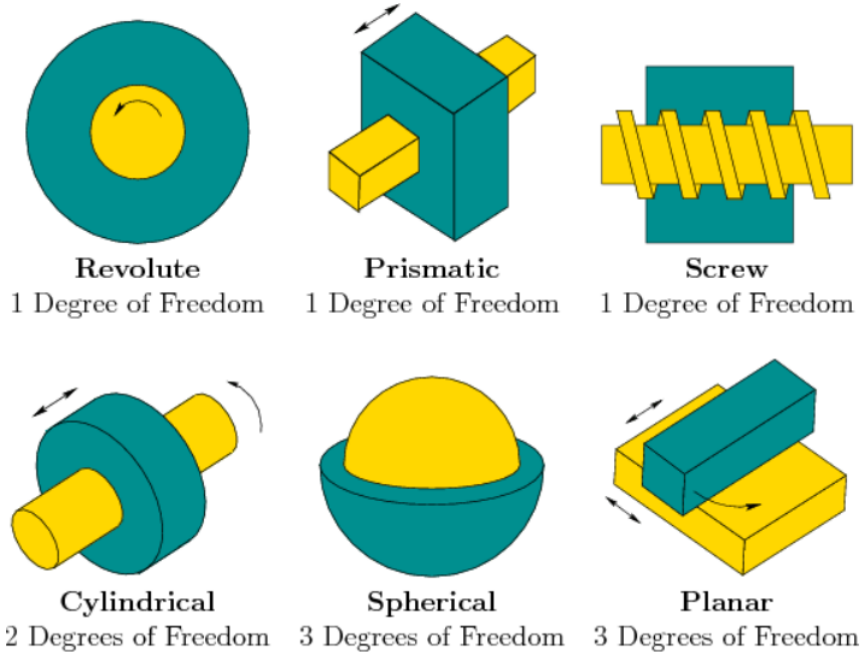


Ilustración 2. Principales pares cinemáticos: revolución, prismático, tornillo, cilíndrico, esférico y planar. (Planingalgorithms.com)

Por tanto, se hace evidente que, para el caso tridimensional, siendo 6 el número de grados de libertad, al introducir un par cinemático en el sistema, el número total de GDL será reducido en función de las restricciones impuestas por el tipo de par aplicado al sistema. Siendo las restricciones independientes entre ellas. El número de grados de libertad puede ser evaluado por la diferencia entre los grados de libertad del sistema y el número de restricciones independientes. La expresión matemática dada por Grüebler-Kutzback es:

$$g = 6N - r$$

Siendo $\boldsymbol{N}$ el número de cuerpos que componen el sistema mecánico y $\boldsymbol{r}$ el número de ecuaciones de restricción independientes. Este será el primer cálculo en el proceso de análisis del sistema mecánico. Si el número de grados de libertad fuese negativo, querría decir que el sistema está sobrerestringido, y por tanto es un problema irresoluble. Más adelante se tratará tal problema. Un grado de libertad igual a cero representa un sistema que no puede moverse, y si el grado es positivo, esto indicará que es un problema resoluble. [1]

1.2 Coordenadas

Las coordenadas son el conjunto de parámetros con el que se define perfectamente el sistema mecánico. En consecuencia, la variación de estos parámetros en función del tiempo describirá el movimiento.

La elección de estas coordenadas será de gran importancia ya que determinará aspectos fundamentales del análisis.

1.2.1 Coordenadas independientes

Es el número mínimo de coordenadas posible para definir la posición del sistema. Al trabajar con estas coordenadas, necesitaremos tantas como grados de libertad posea el mecanismo. Para explicarlas observaremos la figura de cadena cerrada con un grado de libertad:

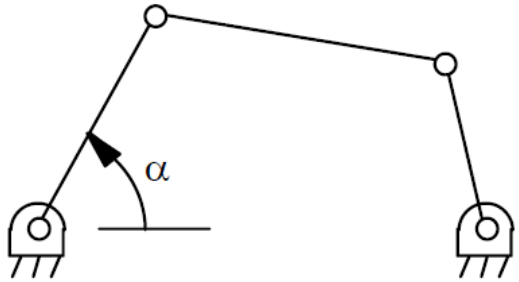


Ilustración 3. Coordenadas independientes.

Este sistema mecánico posee un único grado de libertad. Sólo necesitará por tanto una coordenada independiente para definir su movimiento por completo. Ese parámetro puede ser el ángulo α que forma la primera barra con la horizontal. En la ilustración 7 se puede observar un brazo robot, compuesto por tres barras que pueden girar independientemente. Por tanto, necesitaremos tres parámetros o coordenadas independientes para definir el movimiento del sistema en todo momento. Estas coordenadas podrán ser φ_1, φ_2 y φ_3, todos ellos independientes de la posición de los otros elementos.

La principal ventaja de estas coordenadas es el reducido número de parámetros a emplear. Son las más indicadas para resolver problemas de cadena abierta, muy utilizados en robótica. Sin embargo, para cadena cerrada, hay ocasiones en que las soluciones son múltiples al trabajar con coordenadas independientes, por lo que no son recomendables. Véase la figura 8. [2]

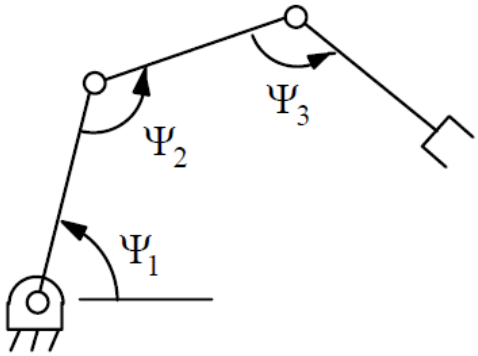


Ilustración 4. Brazo robot, coordenadas independientes.

Véase en la siguiente figura, para un mismo ángulo entre barras, pueden existir dos soluciones diferentes:

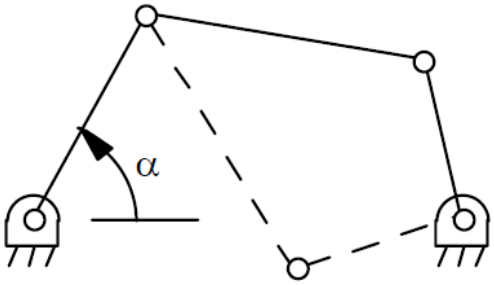


Ilustración 5. Soluciones múltiples para un mismo ángulo.

1.2.3 Coordenadas dependientes

En general podemos decir que un sistema se ha definido con coordenadas dependientes, cuando su número es mayor que el de grados de libertad de nuestro sistema mecánico. El objetivo será emplear los parámetros necesarios para definir la posición de cada elemento del mecanismo. Así evitaremos las dificultades mencionadas para coordenadas independientes.

Dado que existen más parámetros que grados de libertad, existen también unas ecuaciones de ligadura, llamadas ecuaciones de restricción. Podemos calcular el número de ecuaciones de restricción que serán necesarias definir en el problema, mediante la siguiente ecuación.

Ecuaciones de restricción r:

$$r = n - g$$

Siendo r, el número de ecuaciones de restricción, n el número de coordenadas dependientes, y g el número de grados de libertad del mecanismo.

Si bien, las coordenadas dependientes definen la posición del mecanismo perfectamente, existen tres tipos de coordenadas independientes en la mecánica clásica: coordenadas relativas, coordenadas de punto de referencia y coordenadas naturales.

1.2.4 Coordenadas relativas.

Las coordenadas relativas posicionan a cada elemento con respecto al anterior en la cadena cinemática. Están relacionadas con los pares cinemáticos, y en cada par se necesitarán tantas coordenadas como grados de libertad relativos permita el par entre sus elementos de unión.

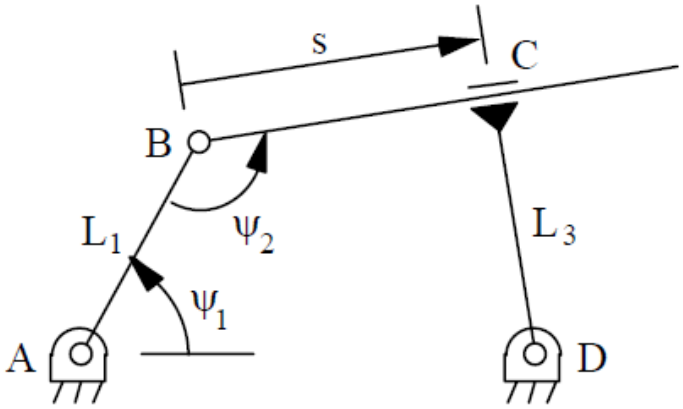


Ilustración 6. Coordenadas relativas.

Al trabajar con estas coordenadas, las ecuaciones de restricción se obtienen con la condición de cierre de lazos, es decir:

$$\overrightarrow{AB} + \overrightarrow{BC} + \overrightarrow{CD} + \overrightarrow{DA} = 0$$

Desarrollando esa ecuación, obtenemos dos ecuaciones de restricción, que son las que necesitamos, ya que r=n-g=3-1=2. De este modo se obtiene:

$$L_1 cos\Psi_1 + s.\cos(\Psi_1 + \Psi_2 - \pi) + L_3.sen(\Psi_1 + \Psi_2 - \pi) - L_4 = 0$$

$$L_1 sen\Psi_1 + s.\mathrm{sen}(\Psi_1 + \Psi_2 - \pi) - L_3.sen(\Psi_1 + \Psi_2 - \pi) - L_4 = 0$$

Teniendo así las ecuaciones de restricción en función de las coordenadas relativas Ψ_1, Ψ_{12} y s.

La ventaja de estas coordenadas es el reducido número de variables que quedan, facilitando un tamaño de problema menor. No obstante, tiene algunos inconvenientes importantes para automatizar los lazos a ordenador. Además, parecen funciones trigonométricas que dificultan la resolución de ecuaciones, aumentando el tiempo computacional.

1.2.5 Coordenadas de punto de referencia.

Estas coordenadas sitúan cada uno de los elementos con independencia del resto. De este modo, se definen las coordenadas de un punto cualquiera de cada elemento, generalmente el centro de masas, y la orientación del mismo. En el espacio tridimensional, definiremos el centro de masas de cada cuerpo como $r = (x_g, y_g, z_g)$ y su orientación mediante los parámetros de Euler $p = (e_0, e_1, e_2, e_3)$. Estas serán las usadas en nuestro programa MUBODYNA, objeto de este proyecto.

Para entender cómo se obtienen las ecuaciones de restricción trabajando con coordenadas de punto de referencia, veamos un ejemplo en dos dimensiones:

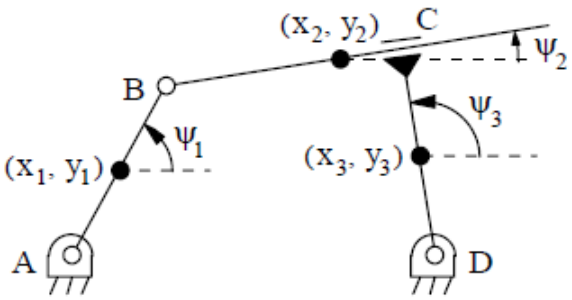


Ilustración 7. Coordenadas relativas

En el ejemplo de la ilustración 10, vemos como cada elemento se define con sus coordenadas de centro de masas y el ángulo que está girado respecto a la horizontal. En primer lugar, calcularíamos el número de restricciones que debemos obtener. Siendo n=9, g=1, obtendríamos $r = n - g = 8$ ecuaciones de restricción a imponer. Los pares de nuestro mecanismo son de grado 1, es decir restringen dos grados de libertad y por tanto habrá dos ecuaciones de restricción por cada par.

Las ecuaciones desde A hasta D quedan entonces:

$$(x_1 - x_A) - \frac{L_1}{2}\cos(\Psi_1) = 0 \qquad (1)$$

$$(y_1 - y_A) - \frac{L_1}{2}\text{sen}(\Psi_1) = 0 \qquad (2)$$

$$\left(x_1 + \frac{L_1}{2}cos\Psi_1\right) - \left(x_2 - \frac{L_2}{2}cos\Psi_2\right) = 0 \qquad (3)$$

$$\left(y_1 + \frac{L_1}{2}sen\Psi_1\right) - \left(y_2 - \frac{L_2}{2}sen\Psi_2\right) = 0 \qquad (4)$$

$$\Psi_3 - \left(\Psi_2 + \frac{\pi}{2}\right) = 0 \qquad (5)$$

$$(x_2 - x_3)cos\Psi_3 + (y_2 - y_3)sen\Psi_3 - \frac{L_3}{2} = 0 \qquad (6)$$

$$(x_3 - x_D) - \frac{L_3}{2}\ cos\Psi_3 = 0 \qquad (7)$$

$$(y_3 - y_D) - \frac{L_3}{2}\ sen\Psi_3 = 0 \qquad (8)$$

La ecuación (6) es la correspondiente al par prismático C, que establece que el centro de la barra 3 se encuentra a una distancia constante de la barra 2. Para obtenerla simplemente hay que proyectar el vector $(r_2 - r_3)$ sobre la barra 3, igual a una constante.

Las ventajas de estas coordenadas es que se pueden definir sistemáticamente, así como sus ecuaciones de restricción, lo que permite una sencilla automatización del proceso. Además de definir la posición de cada elemento del mecanismo sin necesidad de conocer el de los otros. Por todo esto, en MUBODYNA, definiremos la posición inicial de los cuerpos mediante estas coordenadas, como se indicará más adelante.

La única desventaja, es que ofrecen un elevado número de coordenadas y de ecuaciones de restricción. Además, incluyen ecuaciones trigonométricas, aumentando todo ello el tamaño del problema y los tiempos computacionales.

1.2.6 Coordenadas naturales

Las coordenadas naturales cada elemento mecánico con independencia de los demás, al igual que las de punto de referencia. De hecho, son una evolución de estas últimas, en las que los puntos de referencia pasan a ser los puntos donde se encuentran los pares cinemáticos. Véase la figura 11:

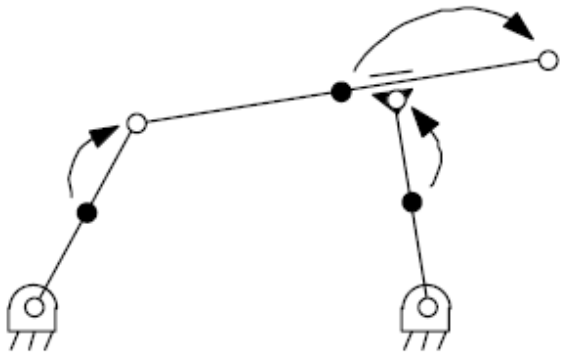


Ilustración 8. Evolución de las coordenadas de punto de referencia.

De este modo, los puntos de referencia quedan en los puntos de cada par. La principal ventaja que podemos observar es que ya no hace falta la definición del ángulo que forman entre sí, y por tanto podemos prescindir de variables angulares. Ventaja tremendamente importante en el caso tridimensional por las facilidades que ofrece al definir el problema.

Veamos ahora el mismo ejemplo anterior, en coordenadas naturales:

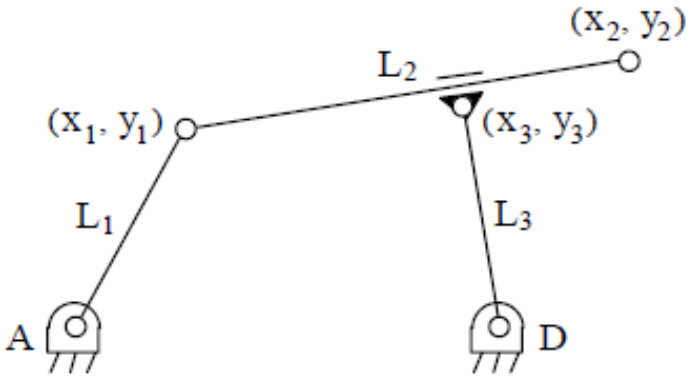


Ilustración 9. Coordenadas naturales.

Como vemos ahora nos quedan seis coordenadas para definir la posición del sistema. Siendo así; n=6, g=1 y r=n-g=5. Quedando nuestras ecuaciones de restricción:

$$(x_1 - x_A)^2 + (y_1 - y_A)^2 - L^2{}_1 = 0 \quad (9)$$

$$(x_2 - x_1)^2 + (y_2 - y_1)^2 - L^2{}_2 = 0 \quad (10)$$

$$(x_3 - x_D)^2 + (y_3 - y_D)^2 - L^2{}_3 = 0 \quad (11)$$

$$(x_2 - x_1)(x_3 - x_D) + (y_2 - y_1)(y_3 - y_D) = 0 \quad (12)$$

$$(x_3 - x_1)(y_2 - y_1) - (y_3 - y_1)(x_2 - x_1) = 0 \quad (13)$$

Las ecuaciones 9, 10 y 11 se obtienen, imponiendo condiciones de distancia constante entre puntos, condición necesaria de barras rígidas. Las ecuaciones 12 y 13 se obtienen gracias al par prismático, imponiendo la perpendicularidad de las barras 2 y 3 (ecuación 12), y la ecuación 13 queda imponiendo que el punto 3 es siempre perteneciente a la barra 2, es decir, alineado entre los puntos 1 y 2.

Las ventajas de las coordenadas naturales son su sencillez y su definición sistemática, independiente de la posición de cada elemento. A su vez las ecuaciones de restricción son fáciles de definir y su número es menor que con coordenadas de punto de referencia. Además, la interpretación de los resultados es mucho más intuitiva y fácil de ver que con las otras. Son por tanto las más ventajosas y conviene familiarizarse con ellas. [2]

1.3 Restricciones

A continuación, se expondrán algunos de los pares y restricciones más relevantes en la modelización de sistemas mecánicos, en coordenadas naturales. Aunque los principios físicos para obtenerlas pueden ser usados con cualquier tipo de coordenada, tal y como vimos en los apartados anteriores.

1.3.1 Restricciones de sólido rígido

Dado que un sólido rígido en *el plano* tiene tres grados de libertad, por tanto, para un sólido rígido en 2D definido por dos puntos tendremos n=4 variables, (x_1, y_1, x_1, y_2) y g=3. Necesitaremos r=n-g=1 ecuación de restricción.

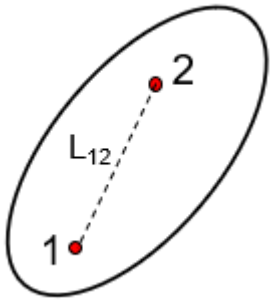


Ilustración 10. Sólido rígido

Por tanto, aplicando la condición de sólido rígido, la distancia L_{12} debe ser constante:

$$(x_1 - x_2)^2 + (y_1 - y_2)^2 - {L_{12}}^2 = 0 \qquad (14)$$

1.3.2 Restricción de sólido rígido modelizado con tres puntos básicos

Ahora tendremos seis variables, por tanto, r=n-g=6-3=3 ecuaciones de restricción. Viendo la figura 14 y aplicando la condición de sólido rígido obtendremos:

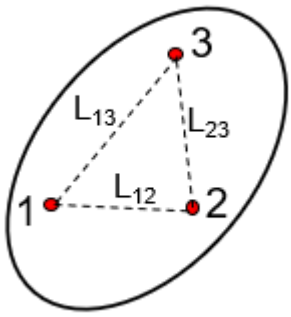


Ilustración 11. Restricciones con tres puntos básicos

$$(x_1 - x_2)^2 + (y_1 - y_2)^2 - {L_{12}}^2 = 0 \qquad (15)$$

$$(x_3 - x_2)^2 + (y_3 - y_2)^2 - {L_{23}}^2 = 0 \qquad (16)$$

$$(x_1 - x_3)^2 + (y_1 - y_3)^2 - {L_{13}}^2 = 0 \quad (17)$$

1.3.3 Sólido rígido con tres puntos básicos alineados

Análogo al caso anterior, y observando la figura 15 obtendremos nuestras tres ecuaciones de restricción necesarias, aplicando condición de sólido rígido:

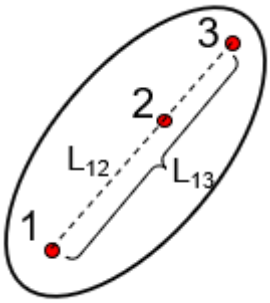


Ilustración 12. Restricciones con tres puntos alineados.

$$(x_2 - x_1)^2 + (y_2 - y_1)^2 - L_{12}^2 = 0 \quad (18)$$

$$(x_3 - x_1) - \frac{L_{13}}{L_{12}}(x_2 - x_1) = 0 \quad (19)$$

$$(y_3 - y_1) - \frac{L_{13}}{L_{12}}(y_2 - y_1) = 0 \quad (20)$$

1.3.4 Sólido rígido con cuatro puntos básicos.

Seleccionando tres puntos no alineados (base tridimensional), impondremos las condiciones de base rígida. El resto de puntos básicos podrán expresarse como función lineal de los segmentos de la base rígida. Veamos la figura 16:

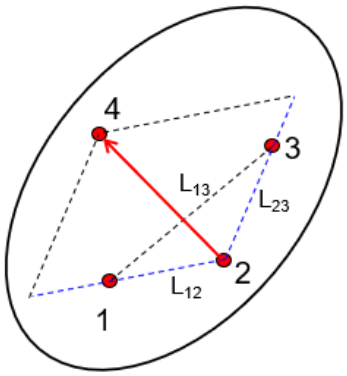


Ilustración 13. Restricciones con 4 puntos.

Con los puntos no alineados 1, 2 y 3, obtenemos:

$$(\mathrm{x}_2 - x_1)^2 + (y_2 - y_1)^2 - L_{12}^2 = 0 \quad (21)$$

$$(x_3 - x_2)^2 + (y_3 - y_2)^2 - {L_{23}}^2 = 0 \quad (22)$$

$$(x_1 - x_3)^2 + (y_1 - y_3)^2 - {L_{13}}^2 = 0 \quad (23)$$

Y ahora, las combinaciones lineales:

$$(x_4 - x_2) + \lambda(x_1 - x_2) - \mu(x_3 - x_2) = 0 \quad (24)$$

$$(y_4 - y_2) + \lambda(y_1 - y_2) - \mu(y_3 - y_2) = 0 \quad (25)$$

Para cualquier caso de sólido rígido definido con más puntos, el proceso a seguir es el mismo que el presentado, simplemente se definirán los puntos adicionales en función de la base rígida. [3]

1.4 Restricciones de par cinemático.

En general podemos decir, que el número de ecuaciones de restricción a definir será igual al número de grados de libertad que impide. A continuación, se presentan algunos de los pares más importantes.

1.4.1 Articulación.

Una articulación es un par cinemático de clase 1, ya que permite solo un grado de libertad, el giro. Por tanto, en caso plano necesitaremos r=3-1=2 ecuaciones de restricción. Observando la figura 17:

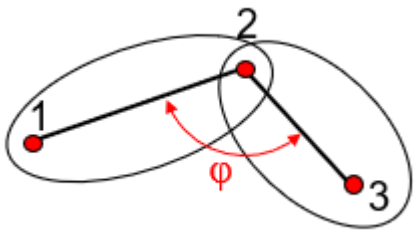


Ilustración 14. Articulación plana.

Para calcular la ecuación necesaria, se usa la condición de producto escalar o producto vectorial, estableciendo que:

$$(x_2 - x_1)(x_3 - x_2) + (y_2 - y_1)(y_3 - y_2) - L_{12}L_{23}cos\varphi = 0 \quad (26)$$

O bien, con el producto vectorial:

$$(x_2 - x_1)\,(y_3 - y_2) - (x_3 - x_2)\,(y_2 - y_1) - L_{12}L_{23}sen\varphi = 0 \qquad (27)$$

Ahora bien, estas ecuaciones presentan ángulos para los cuales, la ecuación 26 es más sensible y otros en los que la ecuación 27 dará el resultado más sensible. Concretamente si φ∈(45º, 135º) ó (225º, 315º) cosφ más sensible a Δφ, y si φ∈(-45º, 45º) ó (135º, 225º) senφ más sensible a Δφ. En un problema se pueden incluir ambas, aunque tendremos un sistema de ecuaciones redundante.

1.4.2 Restricción de par prismático.

Este es otro de los pares de clase 1, por lo tanto, debemos hallar dos ecuaciones de restricción. Para ello, estableceremos la condición de que el **vector 12**, por el producto vectorial del **vector 13**, siempre será nulo al ser paralelos, y que los vectores **12** y **34**, tendrán siempre un producto escalar constante, al formar siempre el mismo ángulo.

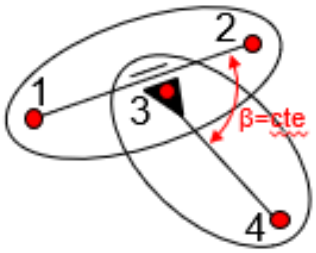


Ilustración 15. Par prismático.

Ecuación de producto vectorial:

$$(x_3 - x_1)\,(y_2 - y_1) - (y_3 - y_1)\,(x_2 - x_1) = 0 \qquad (28)$$

Ecuación de producto escalar:

$$(x_2 - x_1)(x_4 - x_3) + (y_2 - y_1)(y_4 - y_3) - c = 0 \qquad (29)$$

1.4.3 Par prismático de clase II.

Tenemos una única ecuación de restricción, que calcularemos con la condición de 3 perteneciente a la barra 12. Es decir, el producto vectorial de $\overrightarrow{12}$x$\overrightarrow{13}$=0

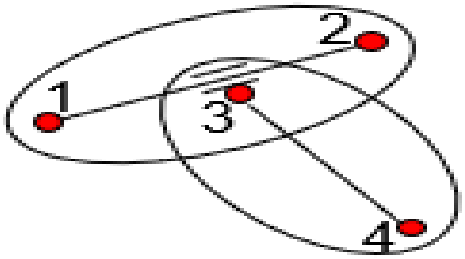


Ilustración 16. Par prismático clase II.

Quedando nuestra ecuación:

$$(x_3 - x_1)(y_2 - y_1) - (y_3 - y_1)(x_2 - x_1) = 0 \qquad (30)$$

1.5 Problemas cinemáticos.

Los problemas cinemáticos que se describirán en este punto, tienen por objetivo conocer el movimiento de los elementos que componen nuestro sistema mecánico sin atender a las fuerzas que actúen sobre él. Tiene como objetivo el estudio de la posición, velocidad y aceleraciones de las coordenadas de los puntos deseados de nuestro sistema, obtenidos a partir de la posición, velocidad y aceleración de los grados de libertad, que serán dato.

En lo sucesivo se resolverán los problemas para la figura 17, en coordenadas naturales.

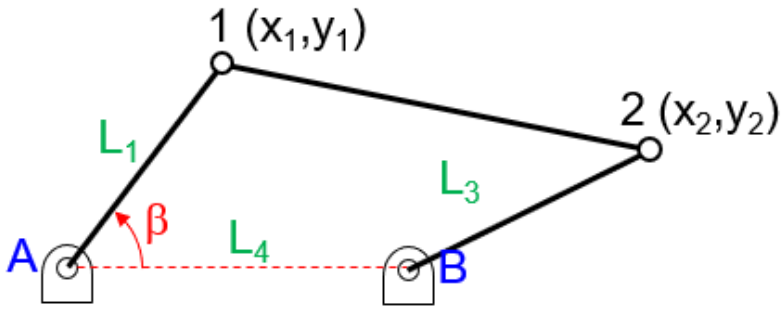


Ilustración 17. Posición inicial del sistema mecánico.

1.5.1 Problema de posición inicial.

En este problema la posición de los grados de libertad son los datos y la posición de los puntos de nuestro sistema las incógnitas, en nuestro ejemplo de la ilustración 17, puntos 1 y 2.

En nuestro ejemplo, el dato es nuestro único grado de libertad, el ángulo β. Por tanto, nuestro vector de coordenadas del mecanismo se define como sigue:

$$q^t = (x_1, y_1, x_2, y_2, \beta) \quad (31)$$

Para el caso general, con **n** variables, lo definimos como:

$$\vec{q}^t = (q_1, q_2, \dots, q_n) \quad (32)$$

Por tanto, el problema se reduce a calcular las coordenadas x_1, y_1, x_2 *e* y_2, en función de β. Siendo el vector columna $\vec{q}$, el vector de todas las coordenadas del sistema. Las ecuaciones de restricción nos darán relaciones entre las coordenadas definidas, entre las que se encuentran los grados de libertad. Estas ecuaciones se incluyen en el vector llamado vector de restricciones. Caso general, para r ecuaciones de restricción: [5]

$$\Phi(q) = \begin{pmatrix} \Phi_1 \\ \Phi_2 \\ . \\ . \\ . \\ \Phi_r \end{pmatrix} = 0 \quad (33)$$

Siendo cada Φ_i, las distintas ecuaciones de restricción del sistema. Para nuestro problema queda:

Ecuaciones de restricción, r=5-1=4:

$$\Phi(q) = \begin{pmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \end{pmatrix} = \begin{pmatrix} (x_1 - x_A)^2 + (y_1 - y_A)^2 - L_1^2) \\ (x_2 - x_1)^2 + (y_2 - y_1)^2 - L_2^2) \\ (x_2 - x_B)^2 + (y_2 - y_B)^2 - L_3^2) \\ (x_1 - x_A) - L_1 cos\beta \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad (34)$$

Como el problema es no-lineal, vamos a aplicar series de Taylor en torno a una posición inicial aproximada:

$$\bar{q}^0 = \begin{pmatrix} q_1^0 \\ . \\ . \\ . \\ q_n^0 \end{pmatrix} \qquad En\ el\ problema: \bar{q}^{0^t} = (x_1^0, y_1^0, x_2^0, y_2^0, \beta)$$

Linealización de Taylor:

$$\Phi(q) \cong \Phi(q_0) + \bar{\Phi}_q(q_0)(q - q_0) = 0 \quad (35)$$

$$\begin{cases} \phi_1(q_1,\ldots,q_n) \cong \phi_1(\bar{q}^0) + \left.\frac{\partial \phi_1}{\partial q_1}\right|_{\bar{q}^0} \cdot (q_1 - q_1^0) + \ldots + \left.\frac{\partial \phi_1}{\partial q_n}\right|_{\bar{q}^0} \cdot (q_n - q_n^0) \\ \vdots \\ \phi_r(q_1,\ldots,q_n) \cong \phi_r(\bar{q}^0) + \left.\frac{\partial \phi_r}{\partial q_1}\right|_{\bar{q}^0} \cdot (q_1 - q_1^0) + \ldots + \left.\frac{\partial \phi_r}{\partial q_n}\right|_{\bar{q}^0} \cdot (q_n - q_n^0) \end{cases}$$

Siendo el vector de coordenadas iniciales, aproximado, lógicamente no cumplirán las restricciones. Escribiendo la ecuación 35 en la forma:

$$\bar{\Phi}_q(q_0)(q - q_0) = -\Phi(q_0) \qquad (36)$$

Es un sistema lineal de **r** ecuaciones y **n-g** incógnitas, por lo que quedan también r=n-g incógnitas. Por lo que el sistema es cuadrado de tamaño **rxr**. La matriz del sistema es el Jacobiano de las ecuaciones de restricción. [5]

$$\Phi_q(q) = \begin{bmatrix} 2(x_1 - x_A) & 2(y_1 - x_A) & 0 & 0 & 0 \\ -2(x_2 - x_1) & -2(y_2 - y_1) & 2(x_2 - x_1) & 2(y_2 - y_1) & 0 \\ 0 & 0 & 2(x_2 - x_B) & 2(y_2 - y_B) & 0 \\ 1 & 0 & 0 & 0 & L_1 sen\beta \end{bmatrix}$$

Ahora solo queda resolver el sistema linealizado para $\bar{q}^0$ inicial y obtener el vector solución $\bar{q}_1^t = (x_1, y_1, x_2, y_2, \beta)$ que tampoco satisface las ecuaciones, pero se acerca más. El problema se itera hasta llegar a una solución más exacta del sistema de ecuaciones.

En la siguiente ilustración se puede ver la interpretación geométrica del método de Newton-Raphson, véase para la función f(x)=0 para una sola variable:

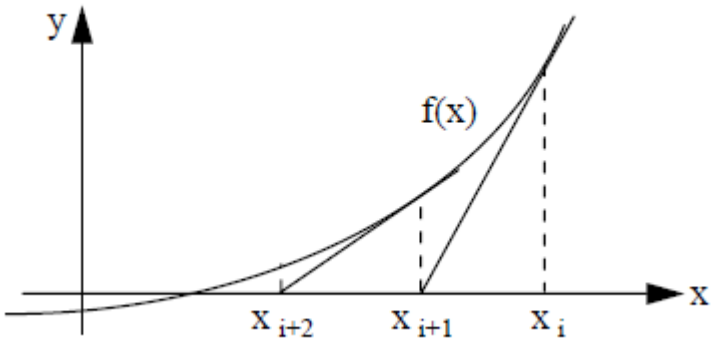


Ilustración 18. Interpretación Newton- Raphson, con una variable. [3]

El vector inicial y los vectores solución de las iteraciones, tienen en común los valores correspondientes a los grados de libertad, en nuestro ejemplo β, en concreto para las g componentes: $q_j^i = q_j^0, para\ j = r+1, \dots, n.$

De este modo podremos obtener un sistema cuadrado eliminando las columnas del Jacobiano de los grados de libertad. En nuestro ejemplo quedaría: [5]

$$\begin{bmatrix} 2(x_1^i - x_A) & 2(y_1^i - x_A) & 0 & 0 \\ -2(x_2^i - x_1^i) & -2(y_2^i - y_1^i) & 2(x_2^i - x_1^i) & 2(y_2^i - y_1^i) \\ 0 & 0 & 2(x_2^i - x_B) & 2(y_2^i - y_B) \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} x_1^{i+1} - x_1^i \\ y_1^{i+1} - y_1^i \\ x_2^{i+1} - x_2^i \\ y_2^{i+1} - y_2^i \end{Bmatrix} = \begin{Bmatrix} (x_1^i - x_A)^2 + (y_1^i - y_A)^2 - L_1^2 \\ (x_2^i - x_1^i)^2 + (y_2^i - y_1^i)^2 - L_2^2 \\ (x_2^i - x_B)^2 + (y_2^i - y_B)^2 - L_3^2 \\ (x_1^i - x_A) - L_1 \cos\alpha \end{Bmatrix}$$

Que, resolviendo para los parámetros de A=(0,0); B=(10,0), L1=2, L2=8, L3=5 y β=60º. Con una aproximación inicial de:

$$\vec{q}^0 = \begin{pmatrix} 1 \\ 2 \\ 8 \\ 3 \\ 60 \end{pmatrix}$$

Obtenemos para la iteración inicial: $\vec{q}^1 = \begin{pmatrix} 1 \\ 1.75 \\ 8.80114 \\ 5.53409 \\ 60 \end{pmatrix}$

La fórmula que nos da el error de la iteración i-ésima es:

$$e^i = \sqrt{\sum_i \left(\Phi_i(\vec{q_i})\right)^2)}$$

En nuestro ejemplo resolvemos para un $\mathbf{e<10^8}$, quedando los siguientes resultados:

Nº Iter. / Coord.	0	1	2	3	4	5
x_1	1	1	1	1	1	1
y_1	2	1.75	1.73214	1.73205	1.73205	1.73205
x_2	8	8.80114	8.42517	8.41258	8.41246	8.41246
y_2	3	5.53409	4.81447	4.74189	4.74128	4.74128
e	18.5	13.2	0.91	0.0077	5.48E-07	3.55E-15

Tabla 1. Resultados de posición inicial.

1.5.2 Problema de velocidad.

El problema de velocidad consiste en calcular las derivadas temporales de las variables de nuestro sistema, conocidas la posición inicial y las derivadas temporales de los grados de libertad.

Dichas variables deben cumplir:

$$\bar{\Phi}(\bar{q}) = 0 \qquad (37)$$

Por tanto, la derivada temporal debe quedar:

$$\overline{\Phi_q}(\bar{q}).\dot{\bar{q}} = 0 \qquad (38)$$

Siendo el vector $\bar{q}$ de posición, conocido. Por tanto, el jacobiano es conocido. De modo que el sistema de ecuaciones (38), es un sistema lineal de **r** ecuaciones y **n** incógnitas, pero de nuevo las velocidades de los grados de libertad son conocidas, al igual que ocurría con el problema de posiciones. De modo que despejando la ecuación (38), volvemos a tener un sistema de dimensión **rxr**. [6]

En resumen:

- Conocemos g=n-r
- $\dot{q}_1, \dots, \dot{q}_r$ velocidades a calcular, y $\dot{q}_{r+1}, \dots, \dot{q}_n$ velocidades de los gdl conocidas.
- Y las siguientes ecuaciones de restricción: $\Phi_1, \dots, \Phi_r$

$$\begin{cases} \left.\frac{\partial \phi_1}{\partial q_1}\right|_{\bar{q}} \cdot \dot{q}_1 + \dots + \left.\frac{\partial \phi_1}{\partial q_r}\right|_{\bar{q}} \cdot \dot{q}_r + \left.\frac{\partial \phi_1}{\partial q_{r+1}}\right|_{\bar{q}} \cdot \dot{q}_{r+1} + \dots + \left.\frac{\partial \phi_1}{\partial q_n}\right|_{\bar{q}} \cdot \dot{q}_n = 0 \\ \vdots \\ \left.\frac{\partial \phi_r}{\partial q_1}\right|_{\bar{q}} \cdot \dot{q}_1 + \dots + \left.\frac{\partial \phi_r}{\partial q_r}\right|_{\bar{q}} \cdot \dot{q}_r + \left.\frac{\partial \phi_r}{\partial q_{r+1}}\right|_{\bar{q}} \cdot \dot{q}_{r+1} + \dots + \left.\frac{\partial \phi_r}{\partial q_n}\right|_{\bar{q}} \cdot \dot{q}_n = 0 \end{cases} \Rightarrow \begin{pmatrix} \left.\frac{\partial \phi_1}{\partial q_1}\right|_{\bar{q}} & \cdots & \left.\frac{\partial \phi_1}{\partial q_r}\right|_{\bar{q}} \\ \vdots & \ddots & \vdots \\ \left.\frac{\partial \phi_r}{\partial q_1}\right|_{\bar{q}} & \cdots & \left.\frac{\partial \phi_r}{\partial q_r}\right|_{\bar{q}} \end{pmatrix} \begin{pmatrix} \dot{q}_1 \\ \vdots \\ \dot{q}_r \end{pmatrix} = - \begin{pmatrix} \left.\frac{\partial \phi_1}{\partial q_{r+1}}\right|_{\bar{q}} \cdot \dot{q}_{r+1} + \dots + \left.\frac{\partial \phi_1}{\partial q_n}\right|_{\bar{q}} \cdot \dot{q}_n \\ \vdots \\ \left.\frac{\partial \phi_r}{\partial q_{r+1}}\right|_{\bar{q}} \cdot \dot{q}_{r+1} + \dots + \left.\frac{\partial \phi_r}{\partial q_n}\right|_{\bar{q}} \cdot \dot{q}_n \end{pmatrix}$$

En nuestro ejemplo el problema de velocidad consistiría en calcular las velocidades de 1 y 2, conociendo, posición inicial β=60°, y la velocidad del grado de libertad $\dot{\beta}$=1 rad/s. Y ya calculado el vector de posición inicial:

$$\bar{q}^t = \{1, 1.73205, 8.41246, 4.7412, \pi/3\}$$

Queda el siguiente sistema a resolver:

$$\begin{pmatrix} 2 & 3.4614 & 0 & 0 \\ -14.8249 & -6.0185 & 14.8249 & 6.0185 \\ 0 & 0 & -3.1751 & 9.4826 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \dot{x_1} \\ \dot{y_1} \\ \dot{x_2} \\ \dot{y_2} \end{pmatrix} = - \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1.73205 \end{pmatrix}$$

Obteniendo el vector velocidades solución:

$$\dot{\bar{q}} = \begin{pmatrix} -1.73205 \\ 1 \\ -1.1674 \\ -0.3909 \\ 1 \end{pmatrix}$$

1.5.3 Problema de aceleración.

Consiste en calcular las aceleraciones de nuestros puntos, conocidas la posición y velocidad del mecanismo. Para ello derivamos respecto al tiempo la ecuación de velocidades (38):

$$\overline{\Phi_q}(\bar{q}).\ddot{\bar{q}} + \overline{\Phi_{\dot{q}}}(\bar{q}).\dot{q} = 0 \qquad (39)$$

Ahora al ser las posición y velocidades conocidas, conocemos el jacobiano y el segundo término de la ecuación. De nuevo, conociendo las g aceleraciones de los grados de libertad, el sistema queda de **r** ecuaciones y **r** incógnitas: [6]

$$\begin{cases} \sum_i \left.\frac{\partial \phi_1}{\partial q_i}\right|_{\bar{q}} \cdot \ddot{q}_i + \sum_{i,j} \left.\frac{\partial^2 \phi_1}{\partial q_i \partial q_j}\right|_{\bar{q}} \cdot \dot{q}_i \dot{q}_j = 0 \\ \vdots \\ \sum_i \left.\frac{\partial \phi_r}{\partial q_i}\right|_{\bar{q}} \cdot \ddot{q}_i + \sum_{i,j} \left.\frac{\partial^2 \phi_r}{\partial q_i \partial q_j}\right|_{\bar{q}} \cdot \dot{q}_i \dot{q}_j = 0 \end{cases} \Rightarrow \begin{pmatrix} \left.\frac{\partial \phi_1}{\partial q_1}\right|_{\bar{q}} & \cdots & \left.\frac{\partial \phi_1}{\partial q_r}\right|_{\bar{q}} \\ \vdots & \ddots & \vdots \\ \left.\frac{\partial \phi_r}{\partial q_1}\right|_{\bar{q}} & \cdots & \left.\frac{\partial \phi_r}{\partial q_r}\right|_{\bar{q}} \end{pmatrix} \begin{pmatrix} \ddot{q}_1 \\ \vdots \\ \ddot{q}_r \end{pmatrix} = - \begin{pmatrix} \sum_{i>r} \left.\frac{\partial \phi_1}{\partial q_i}\right|_{\bar{q}} \cdot \ddot{q}_i \\ \vdots \\ \sum_{i>r} \left.\frac{\partial \phi_r}{\partial q_i}\right|_{\bar{q}} \cdot \ddot{q}_i \end{pmatrix} - \begin{pmatrix} \sum_{i,j} \left.\frac{\partial^2 \phi_1}{\partial q_i \partial q_j}\right|_{\bar{q}} \cdot \dot{q}_i \dot{q}_j \\ \vdots \\ \sum_{i,j} \left.\frac{\partial^2 \phi_r}{\partial q_i \partial q_j}\right|_{\bar{q}} \cdot \dot{q}_i \dot{q}_j \end{pmatrix}$$

En nuestro ejemplo, tendremos que calcular las aceleraciones de 1 y 2, conociendo:

- La posición de β=60°, la velocidad $\dot{\beta}$= 1rad/s y la aceleración $\ddot{\beta}$=1rad/s^2 del grado de libertad.
- Los vectores de coordenadas y velocidades calculados en los anteriores apartados $\overline{\boldsymbol{q}}$ y$\dot{\overline{\boldsymbol{q}}}$.

Resolviendo el sistema nos queda:

$$\ddot{\overline{q}} = \begin{pmatrix} -2.73205 \\ 0.73205 \\ -2.8201 \\ -1.2639 \\ 1 \end{pmatrix}$$

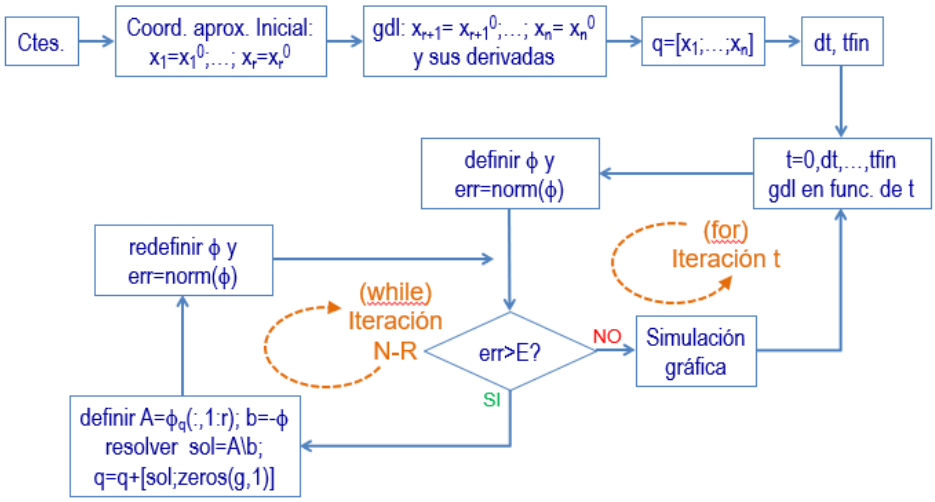
1.5.4 Problema de desplazamientos finitos y simulación cinemática.

Ahora mediante teoría de desplazamientos finitos, podemos calcular la nueva posición a la que se llega, si se incrementan una pequeña cantidad la posición de los gados de libertad. Es una repetición del problema de posición, con la diferencia de que al tener una posición inicial de iteración calculada anteriormente, los valores de partida son una buena aproximación, cosa que no ocurría en el problema de posición, con valores inventados.

Repitiendo este proceso, podemos conocer la evolución del movimiento de un mecanismo para unas variaciones de los grados de libertad.

Ahora, para lograr una simulación debemos incluir la variable tiempo al problema de los desplazamientos finitos y tendremos definida la evolución cinemática de nuestro mecanismo. Esto consiste en resolver la variación de los grados de libertad, que ahora varían en función del tiempo, pudiendo conocer en todo instante las posiciones, velocidades y aceleraciones de nuestro sistema y realizar por tanto la simulación del movimiento. [6]

Diagrama de flujo para implementar el problema de simulación:



1.6 Análisis dinámico. Integradores y métodos de resolución.

En nuestro problema, a partir de unas condiciones iniciales de posición y fuerzas aplicadas, podremos predecir el comportamiento del sistema.

1.6.1 Conjuntos libre y ligado:

Un conjunto de coordenadas se llama libre cuando sus coordenadas son independientes, es decir n=g. Por el contrario, será ligado cuando las coordenadas sean dependientes y por tanto existan ecuaciones de restricción r=n-g.

1.6.3 Ecuaciones de la dinámica.

Generalmente para el sólido rígido trabajaremos con seis ecuaciones, dadas por:

$$\sum \vec{F} = m\vec{a}_G \qquad (40)$$

$$\sum \overrightarrow{M_G} = [I_G]\vec{\alpha} + \vec{\omega} * [I_G]\vec{\omega} \qquad (41)$$

En un mecanismo, aplicaremos las ecuaciones a cada eslabón por separado. Consideraremos las fuerzas $\vec{F}_{ij}$ y los pares $\vec{T}_{ij}$ o esfuerzos de ligadura, ejercidos por eslabones j distintos al i, además de los externos al mecanismo $\vec{F}_i, \vec{T}_i$ o esfuerzos aplicados.

Además, consideraremos la ley de acción reacción: $\overrightarrow{F_{k\imath}} = -\overrightarrow{F_{\imath k}}$, ; $\overrightarrow{T_{k\imath}} = -\overrightarrow{T_{\imath k}}$. De modo que, eliminando los esfuerzos de los ki, con k>i:

$$\overrightarrow{F_\imath} + \sum_{j<i} \overrightarrow{F_{j\imath}} - \sum_{k>i} \overrightarrow{F_{\imath k}} = m_i \overrightarrow{a_{G_\imath}}$$

$$\overrightarrow{T_\imath} + \sum_{j<i} \overrightarrow{T_{j\imath}} - \sum_{k>i} \overrightarrow{T_{\imath k}} + \sum_{j<i} \overrightarrow{r_{\imath j}} \times \overrightarrow{F_{j\imath}} - \sum_{j<i} \overrightarrow{r_{\imath k}} \times \overrightarrow{F_{\imath k}} = \left[I_{G_i}\right]\overrightarrow{\alpha_\imath} + \overrightarrow{\omega_\imath} \times \left[I_{G_i}\right]\overrightarrow{\omega_\imath}$$

No obstante, este método no es el más conveniente ya que obtenemos seis ecuaciones por cada cuerpo rígido, dando sistemas de ecuaciones de gran dimensión. Por este motivo se pueden hacer otros planteamientos más convenientes, con dimensiones de problema menores. Uno de estos métodos es el planteado mediante las ecuaciones de Lagrange: [7]

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_j}\right) - \frac{\partial \mathcal{L}}{\partial q_j} - Q_j^{NC} = 0; j = 1, \dots, g \quad (42)$$

$$\mathcal{L} = T - V \quad (43)$$

Siendo T la energía cinética, V la energía potencial escrita en función de las coordenadas generalizadas, y Q_j^{NC} las fuerzas generalizadas no conservativas asociadas a las coordenadas q_j.

Si el sistema de coordenadas $\{\boldsymbol{q_j}\}$ es ligado con j=1…n los desplazamientos virtuales no son arbitrarios y existirán r=n-g ecuaciones de ligadura. Quedando la variación virtual de ellos nulo:

$$\delta\phi_k = \sum_j \frac{\partial \phi_k}{\partial q_j}\delta q_j = 0, \ \forall k = 1 \dots r \quad (44)$$

Por tanto, para cualquier conjunto de valores $\lambda_1 \dots \lambda_r$ también se cumplirá:

$$\sum_k \sum_j \lambda_k \frac{\partial \phi_k}{\partial q_j}\delta q_j = 0 \quad (45)$$

$$\sum_j \delta q_j \left[\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_j}\right) - \frac{\partial T}{\partial q_j} - \sum_{k=1}^{r} \lambda_k \frac{\partial \phi_k}{\partial q_j} - Q_j\right] = 0 \quad (46)$$

Los se eligen de forma que los primeros r sumandos sean nulos. Los restantes n-r desplazamientos si son independientes, de modo que también quedan nulos, quedando nuestra ecuación:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_j}\right) - \frac{\partial T}{\partial q_j} - \sum_{k=1}^{r} \lambda_k \frac{\partial \phi_k}{\partial q_j} - Q_j = 0; \ j = 1 \dots n \quad (47)$$

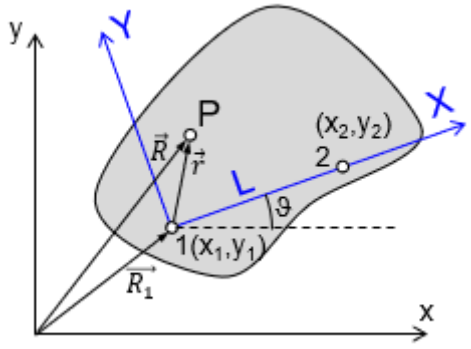
Con lo que nos queda un sistema de n ecuaciones diferenciales, con n+r incógnitas. El sistema se resuelve añadiendo nuestras r ecuaciones de restricción:

$$\phi_k(q_j) = 0; \ k = 1 \dots r \quad (48)$$

Quedando un sistema de n ecuaciones diferenciales y r ecuaciones algebraicas.

1.6.4 Matriz de masas del sólido:

Obsérvese el ejemplo, el vector de posición $\vec{R}$ del punto P se puede escribir como combinación lineal de los vectores de coordenadas de un punto básico 1 $\overrightarrow{R_1}$, y su vector de posición local $\vec{r}$ en coordenadas locales: [6]



$$\vec{R} = \overrightarrow{R_1} + \vec{r} \Rightarrow \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + M_\vartheta \begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{pmatrix} \cos\vartheta & -\sin\vartheta \\ \sin\vartheta & \cos\vartheta \end{pmatrix} \begin{pmatrix} X \\ Y \end{pmatrix} =$$

$$= \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \frac{1}{L} \begin{pmatrix} x_2 - x_1 & y_1 - y_2 \\ y_2 - y_1 & x_2 - x_1 \end{pmatrix} \begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} x_1 + (x_2 - x_1)\frac{X}{L} + (y_1 - y_2)\frac{Y}{L} \\ y_1 + (y_2 - y_1)\frac{X}{L} + (x_2 - x_1)\frac{Y}{L} \end{pmatrix} =$$

$$= \begin{pmatrix} x_1\left(1 - \frac{X}{L}\right) + y_1\frac{Y}{L} + x_2\frac{X}{L} - y_2\frac{Y}{L} \\ -x_1\frac{Y}{L} + y_1\left(1 - \frac{X}{L}\right) + x_2\frac{Y}{L} + y_2\frac{X}{L} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 - \frac{X}{L} & \frac{Y}{L} & \frac{X}{L} & -\frac{Y}{L} \\ -\frac{Y}{L} & 1 - \frac{X}{L} & \frac{Y}{L} & \frac{X}{L} \end{pmatrix}}_{\bar{\bar{C}}} \underbrace{\begin{pmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \end{pmatrix}}_{\bar{q}}$$

$$\vec{R} = \bar{\bar{C}}\bar{q}$$

La matriz C no depende del tiempo ya que solo depende del punto del sólido, por tanto, podemos obtener las derivadas temporales, en la forma:

$$\dot{\vec{R}} = \bar{\bar{C}}\dot{\bar{q}} \quad y \quad \ddot{\vec{R}} = \bar{\bar{C}}\ddot{\bar{q}}$$

1.6.5 Energía cinética:

Se puede escribir como:

$$T = \frac{1}{2}\dot{\bar{q}}^T M \dot{\bar{q}}$$

Siendo M la matriz de masas del sólido. Siendo la energía cinética del mecanismo la suma de las energías de cada sólido rígido:

$$T_{\text{tot}} = \sum_i T_{(i)} = \sum_i \frac{1}{2}\dot{\bar{q}}_{(i)}{}^T M_{(i)} \dot{\bar{q}}_{(i)}$$

Además, se puede calcular la matriz de masas del sistema como suma de las matrices de cada elemento eslabón:

$$M_{Tot} = \sum_i M_i$$

Con lo que las ecuaciones de Lagrange quedarían:

$$T = \frac{1}{2}\dot{\bar{q}}^T M \dot{\bar{q}} = \sum_{i,j} M_{ij}\dot{q}_i\dot{q}_j$$

$$T = \frac{1}{2}\dot{\bar{q}}^T M \dot{\bar{q}} = \sum_{i,j} M_{ij}\dot{q}_i\dot{q}_j$$

Siendo la ecuación compacta de Lagrange para un sistema ligado: [6]

$$\begin{cases} \bar{\bar{M}}\ddot{\bar{q}} + \bar{\bar{\phi}}_q{}^T \bar{\lambda} = \bar{Q} \\ \bar{\bar{\phi}} = 0 \end{cases} \qquad (49)$$

Por último, para un vector de fuerza puntual aplicado en el sólido, podríamos escribir:

$$\text{Por desplazamientos virtuales:} \quad \delta W = \bar{Q}^T \delta\bar{q}$$

Y, por tanto:

$$\vec{R} = \bar{\bar{C}}\bar{q} \Rightarrow \delta\vec{R} = \bar{\bar{C}}\delta\bar{q} \;\Rightarrow \delta W = \bar{F}^T\delta\vec{R} = \bar{F}^T\bar{\bar{C}}\delta\bar{q}$$

1.6.6 Dispositivos mecánicos fundamentales:

Resorte: $f_s = -k(s - L_0)\; y\; resorte\; angular\; f_\beta = -k(\beta - \beta_0)$

Amortiguador: $f_s = -c\dot{s} \quad y \quad amortiguador\; angular\; f_\beta = -c\dot{\beta}$

Actuador: $f_s = a \quad y \quad el\; angular\; f_\beta = a$

1.7 Integradores y Métodos numéricos de resolución.

A continuación, se presentan los métodos de resolución comúnmente utilizados para resolver las ecuaciones de movimiento de mecanismos restringidos. En particular, la aproximación estándar, el método de Baumgarte, el "penalti method" y el "augmented method" de la formulación Lagrangiana.

Cómo vimos anteriormente, el sistema de ecuaciones de Newton-Euler para un sistema mecánico restringido es:

$$\mathrm{M}\dot{\mathrm{v}} - \mathrm{D}^{\mathrm{T}}\lambda = \mathrm{g} \qquad (50)$$

En el análisis dinámico, para las ecuaciones de aceleración podemos escribir la segunda derivada de las ecuaciones de restricción de la siguiente forma:

$$\mathrm{D}\dot{\mathrm{v}} = \gamma \qquad (51)$$

Las dos ecuaciones anteriores pueden escribirse en la forma:

$$\begin{bmatrix} \mathbf{M} & \mathbf{D^T} \\ \mathbf{D} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \dot{\mathbf{v}} \\ \boldsymbol{\lambda} \end{Bmatrix} = \begin{Bmatrix} \mathbf{g} \\ \boldsymbol{\gamma} \end{Bmatrix} \qquad (52)$$

Este sistema diferencial de ecuaciones se resuelve para las aceleraciones ($\dot{\mathrm{v}}$), y los multiplicadores de Lagrange λ**.** Luego para cada paso de tiempo de integración, el vector de aceleraciones y el de velocidades (v y $\dot{\mathrm{v}}$), deben ser integrados para obtener esos mismos vectores para el siguiente paso de tiempo o "integration time step". Y este proceso debe repetirse hasta alcanzar el final del análisis.

Por tanto, nos queda ver como resolver el sistema lineal de ecuaciones (49). Éste, puede ser resuelto aplicando cualquier método de resolución de ecuaciones lineales algebraicas. Dado que el sistema puede contener ecuaciones redundantes de restricción optaremos por resolverlo analíticamente. Para esto despejamos el vector de aceleraciones:

$$\dot{\mathrm{v}} = M^{-1}(g + D^T\lambda) \qquad (53)$$

En este proceso, asumiendo que ningún cuerpo tiene masa o inercia nula, la matriz inversa de masas existe. Por tanto, operando las ecuaciones (53) y (52):

$$\lambda = [\mathrm{DM^{-1}D^{T}}]^{-1}(\gamma - \mathrm{DM^{-1}g}) \qquad (54)$$

Y ahora sustituyendo en la ecuación (52):

$$\dot{\mathrm{v}} = M^{-1}g + M^{-1}D^{T}\{[\mathrm{DM^{-1}D^{T}}]^{-1}(\gamma - \mathrm{DM^{-1}g})\} \qquad (55)$$

Como esta ecuación puede resolverse para v̇, las velocidades y posiciones pueden ser obtenidas mediante un proceso de integración. Esta forma de resolver las ecuaciones de dinámica se le llama comúnmente método estándar de los multiplicadores de Lagrange. Por lo que en nuestro programa está designado con el nombre de 'standard'. La siguiente ilustración presenta un diagrama de flujo, en el que se muestra el algoritmo de la solución estándar de las ecuaciones de movimiento. [7]

El algoritmo de la ilustración 19, puede ser resumido en los siguientes pasos:

1. Para el tiempo inicial $\mathbf{t^0}$ con las condiciones iniciales de posición y velocidad dadas $\boldsymbol{q^0}$ y $\mathbf{v^0}$.

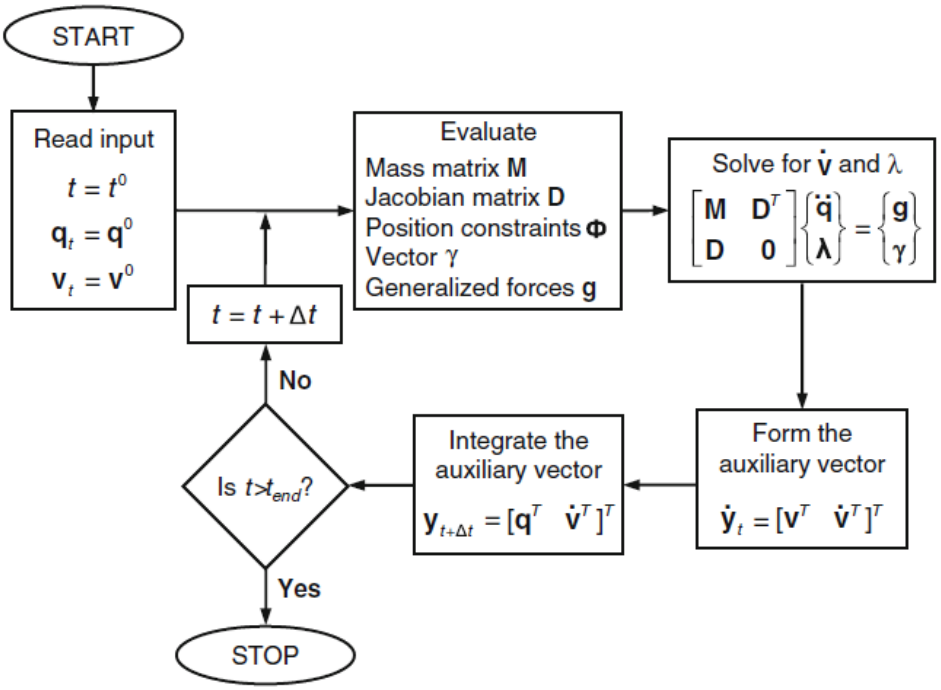
2. Construir la matriz de masas del sistema **M**, evaluar la matriz Jacobiana **D**, construir la matriz de ecuaciones de restricción **Φ**, determinar el termino de aceleraciones $\boldsymbol{\gamma}$ y calcular el vector de fuerzas **g**.

3. Resolver el sistema lineal de ecuaciones del movimiento (53) para el sistema mecánico restringido, con el fin de obtener el vector de aceleraciones $\dot{\mathbf{v}}$ en el instante t y los multiplicadores $\boldsymbol{\lambda}$ de Lagrange.

4. Ensamblar el vector $\dot{y}_t$ que contiene las velocidades generalizadas v y aceleraciones v̇ para el instante t.

5. Integrar numéricamente v y v̇, para los pasos de tiempo $t + \Delta t$ y obtener la nueva posición y velocidad.

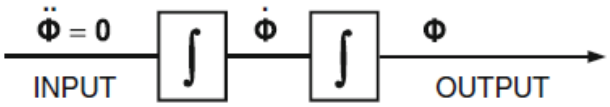
6. Actualizar la variable tiempo, ir al paso (2) y repetir el proceso para el nuevo paso de tiempo, hasta que el tiempo final del análisis.



19. Diagrama de flujo para el algoritmo del método estándar de Lagrange. [6]

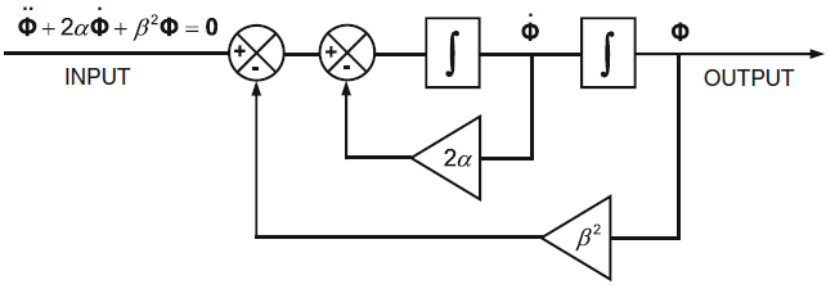
El sistema de ecuaciones (52) no usa explícitamente las ecuaciones de posición y velocidad asociadas a las restricciones cinemáticas, por lo que las ecuaciones de restricción comienzan a ser violadas debido al proceso de integración. Para solventar el problema, es necesario el uso de métodos de corrección de errores. Para este cometido elegimos el método de estabilización de Baumgarte. Este método permite que las ecuaciones sean violadas para luego tomar medidas correctivas para compensar el error.

Lo que hace este método es sustituir la ecuación $\ddot{\Phi} = \mathrm{D}\dot{\mathrm{v}} + \dot{\mathrm{D}}\mathrm{v} = 0$ por la siguiente ecuación:



20. Sistema de bucle abierto (inestable). [6]

Se sustituye por el siguiente:



21. Sistema de bucle cerrado (estable). [6]

Quedando la siguiente ecuación:

$$\ddot{\Phi} + 2\alpha\dot{\Phi} + \beta^2\Phi = 0 \qquad (56)$$

La ecuación (56) es de bucle cerrado y los dos últimos sumandos $2\alpha\dot{\Phi}\ y\ \beta^2\Phi$ son términos de control. Usando la aproximación de Baumgarte las ecuaciones de movimiento son estables, para valores de α y β mayores que cero la estabilidad de la solución general está garantizada. Es necesario resaltar que los parámetros α y β deben ser elegidos por experimentos numéricos para la optimización del resultado, no obstante, Baumgarte no ofrece una forma de calcularlos, por lo que la mejor o peor elección de estos coeficientes será crucial en nuestro resultado. (Flores, Concepts and Formulations for Spatil Multibody Dynamics).

1.7.1 Método de penalización o Penalty method.

El método de penalización presentado por Jalón y Bayo (1994) es una alternativa para resolver las ecuaciones de movimiento. Estas se modelan como ecuaciones diferenciales de segundo orden, de la siguiente forma:

$$m_c\ddot{\Phi} + d_c\dot{\Phi} + k_c\Phi = 0 \qquad (57)$$

Sustituyendo la ecuación $\ddot{\Phi} = \mathrm{D}\dot{\mathrm{v}} + \mathrm{D}\mathrm{v} = 0$, en (57):

$$m_c(\mathrm{D}\dot{\mathrm{v}} + \mathrm{D}\mathrm{v}) + d_c\dot{\Phi} + k_c\Phi = 0 \qquad (58)$$

Operando y teniendo en cuenta las ecuaciones de Newton-Euler ($M\dot{\mathrm{v}} = g$) obtenemos:

$$(M + \alpha D^T D)\dot{\mathrm{v}} = g - \alpha D^T(-\gamma + 2\mu\omega\dot{\Phi} + \omega^2\Phi) \qquad (59)$$

Donde:

$$\alpha = m_c, \qquad \frac{d_c}{m_c} = 2\mu\omega \ \ y\frac{k_c}{mc} = \omega^2 \qquad (60)$$

Esta ecuación (59) puede ser resuelta para v̇. El método dará buenos resultados si α tiende a infinito. Valores típicos de α, μ y ω, son 1x10^7 y 1, respectivamente (Jalón y Bayo 1994). Además, este método de resolución puede resolver sistemas de ecuaciones redundantes. [7]

1.7.2 Augmented method.

La formulación Lagrangiana aumentada, es una metodología que penaliza las violaciones de ecuaciones de restricción, de forma similar al método de estabilización de Baumgarte (Baumgarte 1972). Es un método de integración iterativa con ciertas ventajas, ya que permite resolver conjuntos de ecuaciones de menor dimensión, así como la resolución de ecuaciones redundantes, manteniendo una buena precisión en los resultados. [7]

El método de formulación Lagrangiana aumentada (augmented method), consiste en resolver un sistema de ecuaciones de movimiento en un proceso iterativo. En las siguientes formulas, el índice 'i' hará referencia a la i-ésima iteración. La evaluación del sistema de aceleraciones comienza con:

$$M\dot{\mathrm{v}}_i = g \quad (i = 0) \tag{61}$$

El proceso iterativo para evaluar el sistema de aceleraciones procede con la con la evaluación de:

$$(M + \alpha D^T D)\dot{\mathrm{v}}_{i+1} = M\dot{\mathrm{v}}_i - \alpha D^T(-\gamma + 2\mu\omega\dot{\Phi} + \omega^2\ddot{\Phi}) \tag{62}$$

El proceso iterativo continua hasta:

$$\|\dot{\mathrm{v}}_{i+1} - \dot{\mathrm{v}}_i\| = \varepsilon \tag{63}$$

Donde ε es la tolerancia especificada.

1.8 Fuerzas y problemas de contacto.

Dada la complejidad del problema de contacto y la modelización de los fenómenos que aparecen en él, trataremos y describiremos la forma en que se estudian y modelizan los problemas en MUBODYNA. Si bien hay diferentes modelos de resolución, nosotros presentaremos algunos de los más relevantes y que son utilizados en nuestro programa.

El impacto es un fenómeno físico complejo, dado que en él intervienen características como corta duración, altas magnitudes de fuerza, rápida disipación de energía y grandes y rápidos cambios en las velocidades de los cuerpos. De modo simplificado, en el problema de contacto ocurre cuando dos cuerpos que están separados entran en contacto. Para su modelado y análisis, tanto el método de elementos finitos como como el sistema de simulación de mecanismos pueden ser utilizados. Si bien el método de elementos finitos es el más preciso, pero requiere un alto grado de conocimiento y tiempos computacionales mayores, por lo que sin duda el método de sistemas mecánicos (multibody system) es el más eficiente para hacer una aproximación del movimiento y estudio general del comportamiento mecánico del sistema.

Sea cual sea el método utilizado, el problema siempre presenta dos fases principales, que son: (1) la detección del contacto y (2) la evaluación de las fuerzas de contacto debidas a la colisión. La detección del contacto es uno de los principales problemas, y ha de determinar cuándo, dónde y que puntos entran en contacto. El problema es evaluar los puntos que son candidatos, si entran o no en contacto, y puede ser de gran dificultad dependiendo de la complejidad de las superficies.

1.8.1 Métodos de modelado mediante fuerzas de contacto.

Son también conocidos como “penalty method” o “compliant method”, y son de gran importancia dada su simplicidad computacional y eficiencia. En estos métodos, las fuerzas de contacto son expresadas como funciones continuas de la penetración entre cuerpos. Esta aproximación es muy simple y directa de implementar y de alta eficiencia computacional. Además, la pérdida de contacto puede ser determinada fácilmente mediante la posición y la velocidad de los cuerpos. Las principales desventajas radican en la complejidad de elegir bien las propiedades de los materiales y los parámetros de contacto, así como la evaluación de situaciones complejas con materiales no metálicos. Básicamente, lo que plantea este método es tratar a los cuerpos como si cada región de contacto estuviese compuesta por un sistema de amortiguación, compuesto por un muelle y un amortiguador. La fuerza normal impide la penetración, por lo que no hacen falta ecuaciones de restricción, sino simples fuerzas de reacción. Las magnitudes de rigidez del amortiguador son calculadas en función de la penetración, los materiales y la geometría de los cuerpos que colisionan.

1.8.2 Modelos de Fuerzas de Contacto disipativo: Modelo de Hunt y Crossley.

Si bien, hay numerosos modelos que estudian la colisión, nos centraremos en aquellos que tienen en cuenta la disipación de energía que se produce en la colisión (frente a los modelos puramente elásticos), y más concretamente en el modelo de Hunt y Crossley, que será el implementado en MUBODYNA.

Con este modelo se propone la siguiente formula para las fuerzas de contacto:

$$F_N = K\delta^n + D\dot{\delta} \tag{64}$$

Dónde K es la rigidez del muelle, δ representa la penetración relativa o deformación, n es el exponente no lineal y D es el coeficiente del amortiguador y $\dot{\delta}$ es la velocidad normal de contacto relativa. La selección de D se hace de la siguiente forma:

$$D = x^{\delta^n} \tag{65}$$

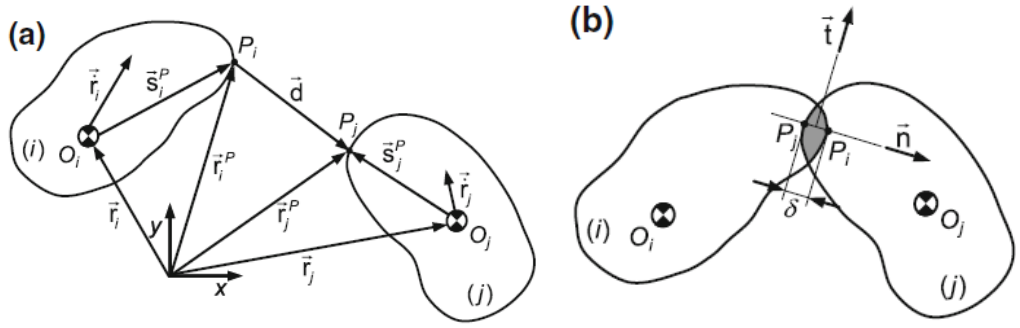
Donde x es el coeficiente de histéresis calculado como:

$$x = \frac{3K(1-c_r)}{2\dot{\delta}^-} \tag{66}$$

Siendo c_r el coeficiente de restitución y $\dot{\delta}^-$la velocidad inicial de contacto. Operando podemos llegar a la siguiente expresión final:

$$F_N = K\delta^n\left[1 + \frac{3(1-c_r)}{2}\frac{\dot{\delta}}{\dot{\delta}^-}\right] \tag{67}$$

En este modelo, la perdida de energía se considera asociada al material, mediante el elemento amortiguador entre los cuerpos. Ahora se muestra una ilustración para el mejor entendimiento del problema, siendo el estado (a) el anterior a la colisión y el (b) la fase de contacto.



Para el modelado se estudia primero la distancia $\vec{d}$para evaluar el momento de impacto, y en la ilustración (b) δ representa la penetración o deformación, ya que se asume una penetración ficticia en función de la cual se evaluarán las fuerzas. A continuación, se muestran los resultados para un modelo de Hunt Crossley, en el que dos esferas impactan, mostrando la evolución de penetración o deformación y la fuerza de contacto, para un coeficiente de rigidez de $2.4x10^9$ y un coeficiente de amortiguación de 3000 Ns/m:

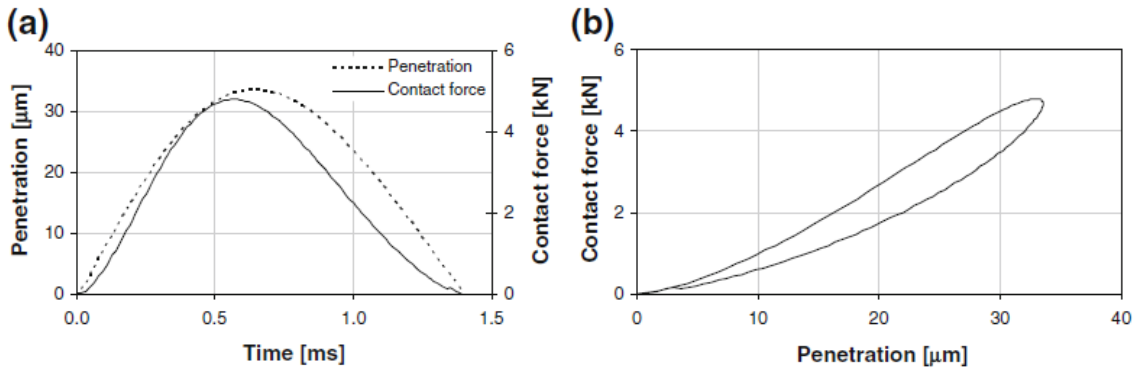


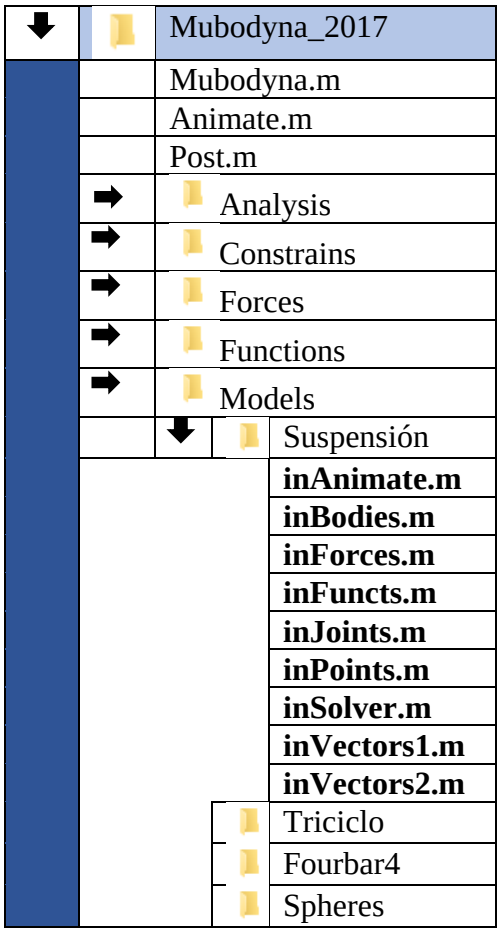
Ilustración 22. Modelo de colisión de esferas mediante la ley de Hunt y Crossley. (P.Flores y Hamid M.Lankarani, Contact Force Models for Multibody Dynamics) [7]

1. CAPÍTULO II: DINÁMICA POR ORDENADOR. MANUAL MUBODYNA.

El presente trabajo se ha desarrollado sobre el programa MUBODYNA, un programa de análisis dinámico directo, desarrollado entre varias universidades, siendo dos de las principales, la Universidad Carlos III de Madrid en colaboración con la Universidad de Minho (Portugal). Este programa está implementado mediante el lenguaje de programación de Matlab, y es desarrollado con motivos educacionales y de investigación, por lo que en ocasiones la eficiencia computacional no se tomará como un objetivo fundamental en su desarrollo.

Este programa, está construido siguiendo una estructura modular, es decir, a través de la nomenclatura y formulación de trabajo de MUBODYNA, se deben construir los diferentes módulos que serán inputs para la simulación de cada modelo. En este trabajo uno de los objetivos principales es facilitar a futuros estudiantes o investigadores la comprensión de la estructura del programa, de modo que se puedan desarrollar nuevos modelos, herramientas y trabajos dentro del entorno.

2.1 Estructura del programa.



El programa está compuesto de una carpeta con tres archivos de Matlab o M-files y otras carpetas contenedoras de más M-files. Los tres archivos principales son mubodyna.m, animate.m y post.m.

Entre las otras carpetas podemos distinguir dos tipos, la carpeta **models**, donde se guardan los modelos de los diferentes problemas que se hayan modelado para su simulación. Y las demás carpetas como: analysis, constraints, forces, functions… que contienen la estructura de los diferentes elementos usados en MUBODYNA, y que dictan como han de definirse posteriormente por el usuario para su correcta síntesis en el programa, así como el modo en que interactuarán en el problema.

InAnimate.m- Es el archivo implementado para la realización de simulaciones visuales en 3D de los diferentes modelos.

Post.m- Contiene la información del postprocesador, una herramienta para analizar datos tras la simulación.

Más adelante hablaremos detenidamente de cómo trabajar con estos archivos y las posibilidades que aportan al usuario.

Mubodyna.m- Contiene la estructura básica para el análisis y simulación de problemas, así como la captación de datos que le proporcionarán los 9 archivos .m a programar de cada modelo. Algunas de sus funciones son:

- ✓ Introduce un código llamado include_global.m que contiene la mayía parte de los comandos globales que serán introducidos en otros archivos "M-file" también.

- ✓ Presenta la interfaz del programa al reproducirlo en Matlab, y le pregunta al usuario por la información necesaria para el análisis, por ejemplo; el nombre del modelo que se desea ejecutar, si se desea corregir las condiciones iniciales o no, el método de resolución, etc.

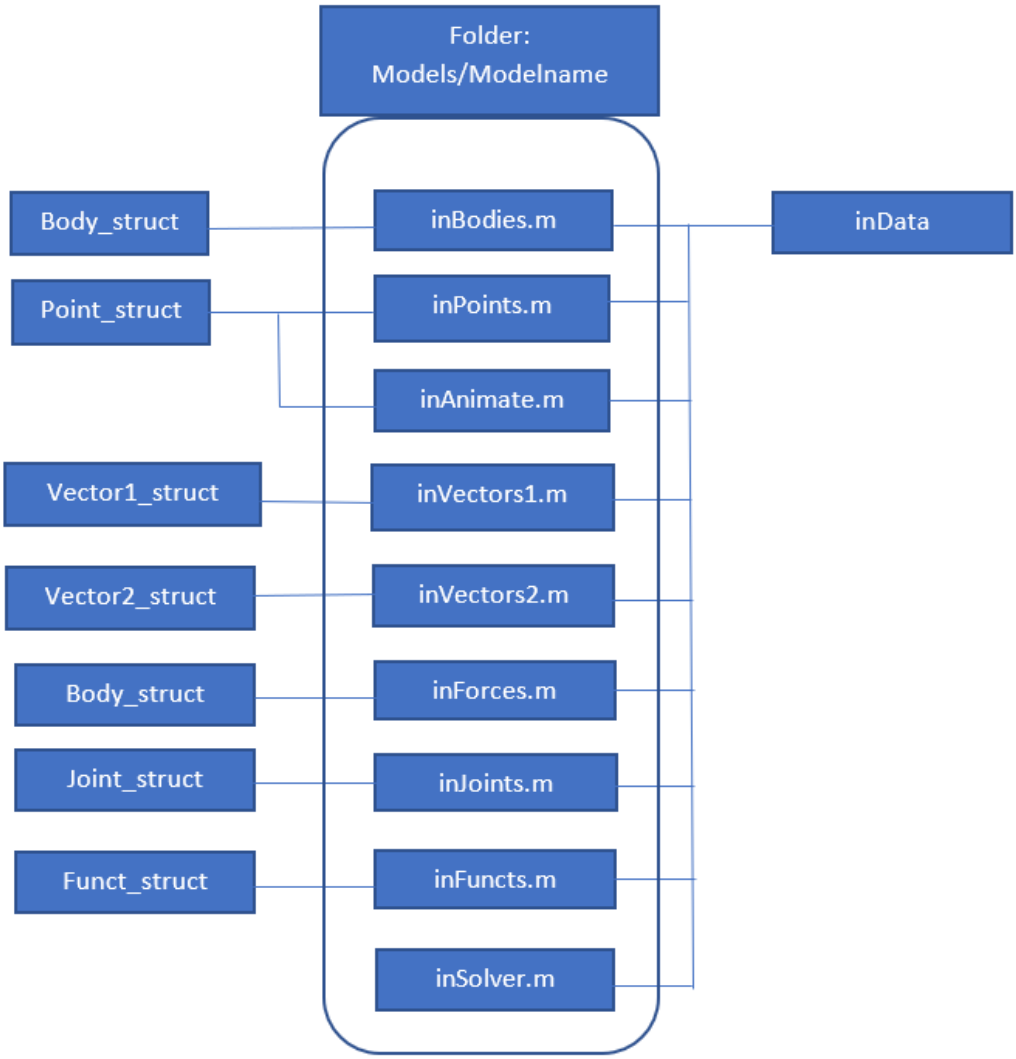
- ✓ Comprueba la redundancia de las ecuaciones de restricción. Y en caso de haber ecuaciones redundantes en el sistema, el programa usará el "penalti method" en lugar del método estándar para resolverlo.

- ✓ Establece la matriz de integración basado en los valores iniciales de las coordenadas y velocidades.

- ✓ Ofrece el informe con los resultados del problema, tales como tiempos, velocidades y coordenadas, que son guardados en la matriz T y uT para cada paso de tiempo. Estas matrices serán usadas posteriormente para realizar animaciones o para analizar resultados en el postprocesador.

2.2 Descripción del modelo.

En este apartado se explicará como el usuario debe introducir y describir todas las variables del problema, para que MUBODYNA pueda tomar los datos y realizar la simulación del problema. Para ello el usuario debe proveer los datos en las diferentes matrices "structure_arrray". Para ello, cada modelo debe introducir las variables en las secciones mostradas en el siguiente diagrama de flujo, que en la práctica consistirá en ir definiendo los distintos elementos del sistema mecánico en cada uno de los nueve m-files que componen un modelo:



Cada uno de los nueve m-files a describir por el usuario serán descritos y analizados, tal y como se deberán utilizar, con su estructura y nomenclatura correspondientes:

2.2.1 inBodies.m

En este archivo se definen las propiedades de los cuerpos de nuestro sistema mecánico, tales como masa, matriz de inercia, posición, velocidad inicial, velocidad angular, puntos de interés del cuerpo, etc.

Las características que nunca deben faltar son, la masa, la matriz de inercia, la posición del centro de masas, la matriz de inercia y la orientación del cuerpo mediante los parámetros de Euler. El resto de parámetros puede definirse si así lo requiere el problema, en caso contrario, el programa asignará los valores por defecto. Por ejemplo, para velocidad lineal y angular, estos valores por defecto son nulos. A continuación, se adjunta la nomenclatura para definir la estructura de un cuerpo:

```
function Body = Body_struct

Body = struct ( ...
    'r'       , [0;0;0]   , ... % x, y, z coordinates
    'p'       , [1;0;0;0] , ... % four Euler parameters
    'A'       , eye(3)    , ... % transformation matrix
    'r_dot'   , [0;0;0]   , ... % x_dot, y_dot, z_dot
    'w'       , [0;0;0]   , ... % omega_x, omega_y, omega_z
    'p_dot'   , [0;0;0;0] , ... % time derivative of Euler parameters
    'r_dot2'  , [0;0;0]   , ... % x_double_dot, ...
    'w_dot'   , [0;0;0]   , ... % alpha_x, ...
    'irc'     , 0         , ... % index of the 1st element of r in u or u_dot
    'irv'     , 0         , ... % index of the 1st element of r_dot in u or u_dot
    'ira'     , 0         , ... % index of the 1st element of r_dot2 in v_dot
    'mass'    , 1         , ... % mass
    'm_inv'   , 1         , ... % mass inverse
    'M'       , eye(3)    , ... % 3x3 mass matrix
    'M_inv'   , eye(3)    , ... % 3x3 mass inverse
    'Jp'      , eye(3)    , ... % inertia matrix in xi-eta-zeta
    'J'       , eye(3)    , ... % inertia matrix in x-y-z
    'Jp_inv'  , eye(3)    , ... % inverse inertia matrix in xi-eta-zeta
    'J_inv'   , eye(3)    , ... % inverse inertia matrix in x-y-z
    'f'       , [0;0;0]   , ... % sum of forces that act on the body
    'n'       , [0;0;0]   , ... % sum of moments that act on the body
    'wp'      , [0;0;0]   , ... % omega_xi, omega_eta, omega_zeta
    'wp_dot'  , [0;0;0]   , ... % alpha_xi, ...
    'radius'  , 0         , ... % if sphere impact exists
    'modulus' , 0         , ... % Young's modulus
    'poisson' , 0         , ... % Poisson's ratio
    'color'   , 'k'       , ... % default color for the body
    'pts'     , []          ... % point indexes associated with this body
);
```

Para definir cualquiera de las características descritas arriba, el usuario deberá escribir la referencia del "cuerpo i" mediante `Bi.´ y añadir después del punto la el comando que hace referencia a la variable deseada. Por ejemplo:

Para definir la posición del centro de masas del cuerpo 3 en el punto (3,-1,5) en coordenadas globales, deberemos escribir:

B3.r=[3;-1;5]

Para asociar al cuerpo 5 la velocidad angular (0,0,25) en ejes locales, escribiremos:

B5.w= [0;0;25]

Es importante mencionar que esta velocidad angular describe el vector en ejes locales. **Ver ejemplo de péndulo simple**

Para definir los angulos de Euler del cuerpo 2, cuyos ejes están alineados con los globales, escribiríamos:

B2.p=[1;0;0;0]

Es importante mencionar que el valor [1;0;0;0] es el valor por defecto de los parámetros de Euler, es decir, solo habrá que definir los parámetros de Euler, en aquellos cuerpos en los que la orientación de los ejes locales no coincida con la de los ejes globales.

Una vez definidos los cuerpos, generalmente cualquier otro elemento que describamos en el futuro y sea asociado a uno de los cuerpos, la posición o dirección de ese elemento se describirá en los ejes locales de dicho cuerpo. Ahora veremos un ejemplo, al definir los puntos de un cuerpo.

Por último, queda incluir los cuerpos definidos que se enviarán mediante le vector de cuerpos. Por tanto, la última línea de código siempre será:

Bodies=[B1;B2…;Bn];

2.2.2 inPoints.m

Todos los puntos de un modelo deben definirse en este archivo, con excepción de puntos que solo tienen un interés visual que se definirán en inAnimate.m. Todos los puntos de interés en el mecanismo, como puntos usados en las uniones de pares cinemáticos, puntos a los que se fija un vector, puntos desde los que se fijan resortes, amortiguadores, etc, y en general cualquier punto que se vaya a utilizar para definir un elemento de nuestro análisis, debe ser definido en este apartado.

El centro de masas de cada cuerpo será visualizado sin necesidad de definirlo aquí ya que, al definir la posición del cuerpo en inBodies mediante Bi.r ya quedó definido este punto. Sin embargo, si ese punto va a usarse para definir otro elemento también deberá definirse expresamente en inPoints.m.

También hay que comentar que los puntos del suelo que sean de interés deberán definirse aquí.

Estructura para definir el punto:

➢ Definir Pi como punto	Pi= Point;
➢ Definir el cuerpo al que pertenece	Pi.Bindex = 3
➢ Definir sus coordenadas en sistema local	Pi.sPp= [3;-1;5];

***Nota*:** el número '0' es el asignado al suelo, por lo que para un punto j que pertenece al suelo podríamos poner Pj.Bindex =0. No obstante, el valor asignado por defecto a los puntos ya es el '0', por lo que podríamos saltarnos esa línea de código.

Además, existen otros comandos que pueden ser de interés, a continuación, se adjunta la nomenclatura asociada a inPoints:

```
function Point = Point_struct

Point = struct ( ...
    'Bindex' , 0       , ... % body index
    'sPp'    , [0;0;0] , ... % s_prime; xsi, eta, zeta coordinates
    'sP'     , [0;0;0] , ... % x, y, z components of vector s
    'rP'     , [0;0;0] , ... % x, y, z coordinates of the point
    'sP_dot' , [0;0;0] , ... % s_P_dot
    'rP_dot' , [0;0;0] , ... % r_P_dot
    'rP_dot2', [0;0;0]   ... % r_P_dot2
);
```

2.2.3 inAnimate.m

En este apartado también se pueden definir puntos por motivos de interés en la visualización de animaciones. Como estos puntos no son empleados en el análisis, deben ser definidos aquí. La estructura para definir los puntos es exactamente la misma que la definida anteriormente. Estos puntos serán enviados a la animación mediante Points_anim = [P1;…;Pn].

En el caso de que no se incluyan puntos en inAnimate, se deberá asignar el vector vacío: Points_anim = [ ];

Además, se deben incluir algunos parámetros para definir como se visualizará la animación del mecanismo:

- Dimensiones del cubo de visualización mediante xmin, xmax, ymin, ymax, zmin y zmax.
- El ángulo de visualización de la animación se puede indicar mediante AZ y EL por ejemplo, AZ=60 girará el cubo 60º respecto al eje 'z', y EL=25 rotara el cubo sobre el eje horizontal 'y'.

Ejemplo de archivo inAnimate:

```
function inAnimate
    include_global
    Point = Point_struct;

P3 = Point;
P3.Bindex = 1;
P3.sPp = [0; -0.1; 0.1];

P4 = Point;
P4.Bindex = 1;
P4.sPp = [0.1; -0.1; 0];

P5 = Point;
P5.Bindex = 1;
P5.sPp = [0; -0.1; -0.1];

P6 = Point;
P6.Bindex = 1;
P6.sPp = [-0.1; -0.1; 0];

Points_anim = [P3; P4; P5; P6];

% This contains the variables for defining the 3D animation axes
    xmin = -0.4; xmax =  0.4;
    ymin = -0.4; ymax =  0.4;
    zmin = -0.4; zmax =  0.4;
% Rotation of coordinate axes
% AZ rotates about the Z axis,
%EL rotates about some horizontal axis.
    AZ = 60; EL = -10;
```

2.2.4 inVectors1.m

En este archivo se definen los vectores de tipo 1, estos son vectores unidos a un cuerpo. Los parámetros para definir estos vectores son:

Vi.Bindex	Indica el cuerpo al que va ligado el vector
Vi.sP	Vector en coordenadas locales del cuerpo al que pertenece

El output que ofrece los vectores de tipo 1, es la matriz Vectors1, al igual que otras matrices output, si no se han definido elementos, deberá incluirse vacía en el final del archivo: "Vectors1=[ ];"

Estructura y nomenclatura para definir inVectors1:

```
function Vector = Vector1_struct

Vector = struct ( ...
    'Bindex', 0       ,  ... % body index
    'sp'    , [0;0;0] ,  ... % s_prime; xi, eta, zeta components
    's'     , [0;0;0] ,  ... % x, y, z components
    's_dot' , [0;0;0]    ... % s_dot
);
```

Ejemplo: definir un vector perteneciente al cuerpo 3, con coordenadas locales (1,0,0):

```
function inVectors1
    include_global
    Vector = Vector1_struct;

    V1 = Vector;
    V1.Bindex = 3;
    V1.sP = [1;0;0];


Vectors1 = [V1];
```

2.2.5 inVectors2.m

En este archivo.m definimos los vectores de tipo 2, es decir, los que están conectados entre dos cuerpos. Para describir un vector de tipo 2 se requiere describir los índices de los dos puntos que conecta. **Si el vector está conectado entre el suelo y un cuerpo, el final del vector o flecha siempre debe apuntar al suelo, es decir el punto P_j deberá ser el ligado al suelo.

Comandos esenciales para definir un vector tipo 2:

Vx.iPindex	Punto inicial del vector
Vx.jPindex	Punto final o flecha

Estructura y nomenclatura de Vectores tipo 2:

```
function Vector = Vector2_struct

Vector = struct ( ...
    'iPindex' , 0       , ... % point Pi index (tail of vector)
    'jPindex' , 0       , ... % point Pj index (head of vector)
    'iBindex' , 0       , ... % body i index
    'jBindex' , 0       , ... % body j index
    'd'       , [0;0;0] , ... % x, y, z components
    'd_dot'   , [0;0;0]   ... % d_dot
);
```

Ejemplo de vector ligado a los puntos 1 y 4:

```
function inVectors2
    include_global
    Vector = Vector2_struct;

V1 = Vector;
V1.iPindex = 1;  %Inicio del vector en punto 1
V1.jPindex = 4;  % Final del vector en punto 2

Vectors2 = [V1];  %matriz output Vectors2
```

2.2.6 inForces.m

Los distintos tipos de fuerza que intervienen en un problema deben definirse en este archivo. Generalmente, utilizamos la letra 'S' para nombrar cada fuerza, seguida del índice 'i' que la identifica: 'Si'. Primero se describe el tipo de fuerza que es, y después su estructura según corresponda. A continuación, se describe la estructura general de los tipos de fuerza más importantes:

Gravedad	
Si.type = 'weight'	Tipo de fuerza
Si.gravity = 9.81 (**valor por defecto)	Valor de la gravedad
Si.wgt = [0;0;1] (**valor por defecto)	Dirección de la gravedad

Fuerza de muelle por vector	
Si.type = 'ptp'	Tipo de fuerza
Si.V2index=3	Vector de tipo 2, que une los dos puntos de actuación del muelle.
Si.k = 50	Rigidez del muelle
Si.el = 8	Elongación natural del muelle
Si.c = 30	Coeficiente de amortiguación.
Si.f_a = 0	Fuerza constante del actuador.

**La fuerza muelle se describe desde la estructura de un sistema muelle-amortiguador como el de la ilustración:

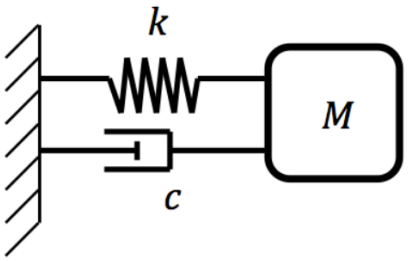


Ilustración 23.

Fuerza muelle-amortiguador definida entre puntos	
Si.type = 'zu'	Tipo de fuerza
Si.iPindex = 2	Define al punto 2 como uno de los extremos
Si.jPindex=5	Define al punto 5 como uno de los extremos
Si.k = 50	Rigidez del muelle
Si.c = 80	Coeficiente de amortiguación

**Si para este tipo de fuerzas uno de los puntos pertenece al suelo, ese deberá ser ligado al punto “j”: “Si.jPindex=0”

Par motor	
Si.type = ‘np’	Aplica un par constante
Si.iBindex = 3	Indica donde cuerpo en el que actúa el par
Si.np = [0 ; 10; 2.5]	Componentes locales del par

2.2.6.1 Modelos de fuerzas para simulación de impactos.

Actualmente se dispone de algunos modelos para simular ciertos problemas de impacto y estudiar estos fenómenos. Actualmente se siguen desarrollando diferentes modelos de fuerzas que respondan a este complejo fenómeno. A continuación, se muestran dos de los más importantes con los que se ha trabajado:

Impacto Esfera-Plano	
Si.type = ‘sph_pln’	Tipo esfera-plano
Si.iPindex=2;	Punto central de la esfera 5.
Si.iBindex = 5	Ligadura al índice de esfera 5.
Si.jBindex = 3	Ligadura al índice de plano 3.
Si.plane_normal_vector= [0;1;1]	Vector normal al plano
Si.sphere_radius = 1	Radio de la esfera
Si.restitution = 1.0	Coeficiente de restitución
Si.force_model = ‘lankarani’	Modelo físico de impacto empleado
Si.n_exponent = 1.5	Coeficiente ‘n’ (ver modelos de impacto en apartado de fundamentos).
Si.friction = true	True añade fuerzas de fricción/False descarta estas fuerzas
Si.fric_model = ‘linear’	Modelo lineal de fricción
Si.mu = 0.5	Valor de μ
Si.mus = 0.6	Valor de μ estático

Contacto entre esferas.	
Pi	Centro de esfera i.
Pj	Centro de esfera j.
Bi	Índice de la esfera i.
Bj	Índice de la esfera j.
V2	Vector de excentricidad.
Ri	Radio esfera i.
Rj	Radio esfera j.
K	Coeficiente de rigidez.
Cr	Coeficiente de restitución.
n	Exponente no lineal.
F_model	Modelo de fuerza de contacto.

A continuación, vemos un ejemplo, en el que se modela una esfera (Body 2) de radio 1 y centro Point=1, cayendo sobre un plano inclinado 30º (Body 1), mediante el modelo de fuerza de contacto de Lankarani con exponente no lineal n=1.5, y no se tiene en cuenta la fricción entre esfera y plano:

$$F_N = K\delta^n + D\dot{\delta}$$

```
function inForces
    include_global
    Force = Force_struct;

S1 = Force;
S1.type = 'weight';    % include the weight
S1.gravity = 9.81;

S2 = Force;
S2.type = 'sph_pln';
S2.iPindex = 1;
S2.iBindex = 2;
S2.jBindex = 1;
S2.plane_normal_vector = [0;sqrt(1)/2;sqrt(3)/2]; %plano inclinado 30°
S2.sphere_radius = 1;
S2.restitution = 1.0;
S2.force_model = 'lankarani';
S2.n_exponent =  1.5;
S2.friction = false;
S2.fric_model = 'linear';
S2.mu = 0.5;
S2.mus = 0.6;
%Friction variables (mu, mus, vs, z, sigma0, sigma1, sdw) se
%utiliza los valores por defecto

Forces = [S1;S2];
```

Nota: para poder describir esta fuerza, es importante describir como se define un plano. Éste queda definido mediante un punto y el vector normal al plano. Además, es muy importante que el vector normal apunte a la sección del espacio en la que está la esfera o el cuerpo que interactúe con él. El plano será definido como otro cuerpo en inBodies.m al igual que la esfera. Veamos como está definido para el problema del ejemplo anterior:

```
function inBodies
    include_global
    Body = Body_struct;


B1 = Body;                  % Plano asociado a B1!!
B1.r = [0; 0; 0];           %Punto perteneciente al plano
B1.mass = 100;
B1.Jp = [0.1    0           0     % el vector normal será definido
         0      0.1         0   % en inForces al incluir la fuerza sph-pln
         0      0           0.1];
B1.poisson = 0.3;
B1.modulus = 290e9;

B2 = Body;                % Definición de la esfera
B2.r = [0; -3; 4.5];
B2.mass = 500;
B2.Jp = [200        0               0
         0              200          0
         0              0              200];
B2.poisson = 0.3;
B2.modulus = 290e9;

Bodies = [B1; B2];
```

2.2.7 inJoints.m

En este archivo han de describirse las uniones o juntas, que forman los pares cinemáticos entre elementos del mecanismo. Estas uniones se definen entre dos cuerpos 'i' y 'j', o entre un cuerpo 'i' y el suelo'j=0'.

A continuación, se muestran los pares cinemáticos más relevantes de los que disponemos en MUBODYNA, así como los parámetros que el usuario deberá introducir para su definición:

Junta esférica (s,3):

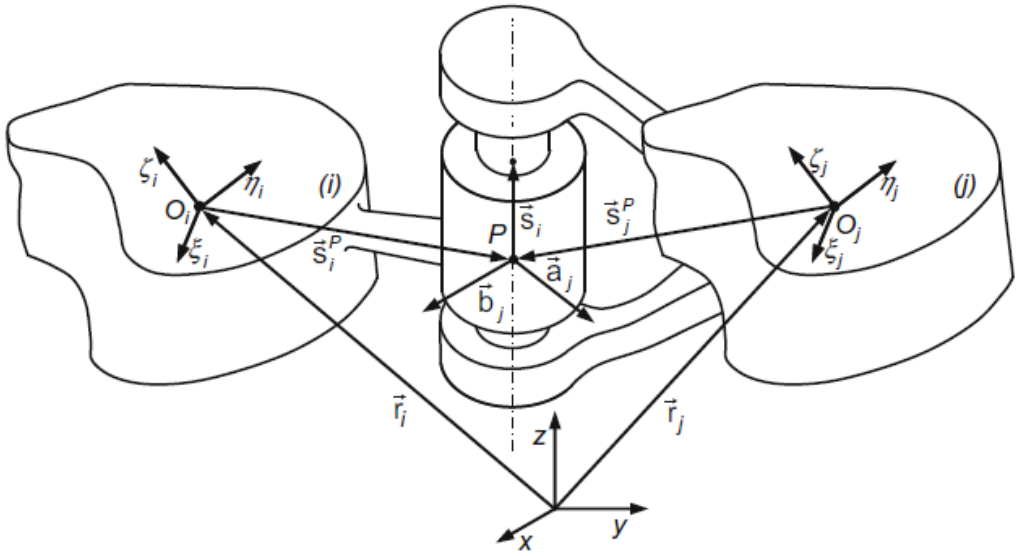
Esta junta tiene **tres** restricciones, que el programa definirá con sus correspondientes ecuaciones de restricción en la matriz Φ. Dejando libres los giros en las tres direcciones del espacio. Para definirla, se dan los puntos coincidentes de los dos cuerpos involucrados de la siguiente manera:

Sphereical joint	
Requerimientos para definición	P_i, P_j
Definición de inpunts	Jx.type = 'sph'; Jx.iPindex = 2; Jx.jPindex = 3;

Las líneas de código en tabla, indicarían una junta esférica entre los puntos 2 y 3. Los cuerpos a los que pertenece cada punto ya fueron definidos en inPoints.

Junta de revolución o giro (r,5):

Este tipo de junta restringe 5 grados de libertad, permitiendo solo el giro en una dirección entre dos cuerpos. Para definirla han de darse dos vectores, ambos con la dirección del giro, uno perteneciente al cuerpo 'i' y otro al 'j'. Obsérvese la ilustración:



24. Junta de revoluta entre cuerpos 'i' y 'j'. [6]

De modo que habrá que dar dos vectores, $\bar{s_i}$ y $\bar{s_j}$ que tendrán la dirección del giro y serán paralelos entre ellos, uno perteneciente a cada cuerpo, y los puntos en los que están aplicados:

Revolute joint.	
Requerimientos:	$P_i, P_j, \bar{a}_i, \bar{a}_j$
Definición de inputs:	Jx.type = 'rev'; Jx.iPindex =1; Jx.jPindex = 8; Jx.iVindex_a=1; Jx.jVindex_a=3;

El ejemplo en tabla indicaría una junta de revolución entre dos cuerpos, con vectores 1 y 3 aplicados en los puntos 1 (del cuerpo i) y 8 (del cuerpo j) respectivamente.

Par cilíndrico (c,4):

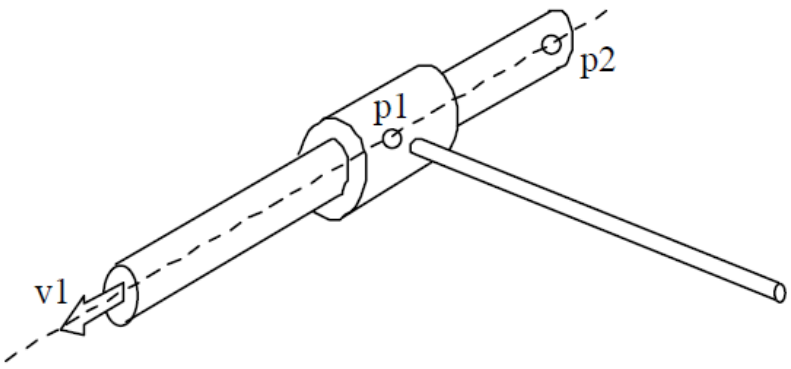
La junta cilíndrica restringe cuatro grados de libertad, dejando una dirección de desplazamiento y otra de giro libres entre dos cuerpos.

25. Junta cilíndrica.

Para definirla, se requieren dos vectores en dirección paralela al giro permitido, uno perteneciente a cada cuerpo, y otro vector de tipo 2 que une los puntos de ambos cuerpos en el instante inicial:

Cylindrical joint	
Requerimientos:	$\bar{v}_2, \bar{a}_i, \bar{a}_j$
Definición de inputs:	Jx.type = 'cyl'; Jx.iVindex_a= 3; Jx.jVindex_a= 1; Jx.V2index = 2;

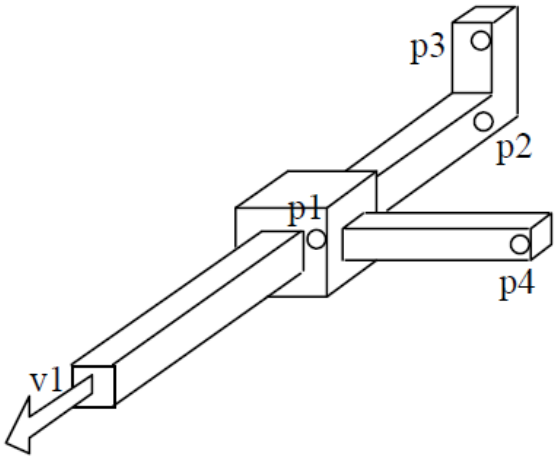
El vector V2, de tipo 2, une los puntos básicos de cada cuerpo. Para el ejemplo de la ilustración 25, crearíamos un vector de tipo 2 que uniese el punto p1 y el punto p2:



26. Par cilíndrico. [2]

Par prismático o traslacional (t,4).

Como vimos en el apartado 1.5.2 para modelizar este par, debemos definir tres vectores para establecer las ecuaciones de restricción. Observando la ilustración 26, necesitaríamos los vectores $\overline{21}, \overline{41}$ *y el vector unitario* $\overline{V}_1$, para establecer las restricciones explicadas en 1.5.2. Sin embargo, la condición de ángulo constante entre barras también se puede definir mediante una matriz de transformación entre los sistemas de ambas barras.



27. Par prismático. [2]

Quedando los siguientes inputs:

Par cilíndrico	
Requerimientos	El vector 41: $\overrightarrow{a_i}$ El vector 21 (de tipo2): $\vec{a}_j$ La matriz de transformación entre cuerpos que establece el ángulo α entre cuerpos.
Definición de inputs	Jx.type = 'tran'; Jx.jVindex_a = 1; Jx.V2index =2; Jx.Ac = [cos α 0 sin α 0 1 0 -sin α 0 cos α];

Rotación-relativa (-,1):

Esta relación de par sirve para establecer:

- Una rotación relativa entre un cuerpo y el suelo.
- Una rotación relativa entre dos cuerpos.
- Simular un par motor aplicado en un cuerpo.

Para definir cada caso se darán los índices de los cuerpos y un vector que tenga la dirección del giro relativo o el par motor, además para simular el motor se deberá aportar la función directora en inFunction.m:

Relative rotation	
Restricciones entre un cuerpo (2) y el suelo.	Jx.type = ‘rel-rot’; Jx.iBindex =2; Jx.jBindex_a =3; (Jx.jVindex = 0 no es necesario definirlo, ya que es valor por defecto ‘0’)
Restricciones entre cuerpos (1) y (2).	Jx.type = ‘rel-rot’; Jx.iBindex =2; Jx.jBindex=1; Jx.jVindex_a =3;
Implementar un par motor de rotación relativa entre cuerpos.	Jx.type = ‘rel-rot’; Jx.iBindex =2; Jx.jBindex=1; Jx.jVindex_a =3; Jx.iFunct = 1; La función 1 deberá ser definida en inFunctions y será la rectora del par motor.

Translación relativa (-,1):

Sus funciones son análogas al par anterior, pero con traslación, el vector a definir debe tener la dirección del movimiento relativo:

Relative rotation	
Restricciones entre un cuerpo (2) y el suelo.	Jx.type = ‘rel-tran’; Jx.iBindex =2; Jx.jVindex_a =3; (Jx.jBindex = 0 no es necesario definirlo, ya que es valor por defecto ‘0’)
Restricciones entre cuerpos (1) y (2).	Jx.type = ‘rel-tran’; Jx.iBindex =2; Jx.jBindex=1; Jx.jVindex_a =3;
Implementar un motor de traslación relativa entre cuerpos.	Jx.type = ‘rel-tran’; Jx.iBindex =2; Jx.jBindex=1; Jx.jVindex_a =3; Jx.iFunct = 1;

Comando Fix.

Este comando puede ser muy útil para fijar un cuerpo al suelo, veamos su definición:

Fix	
Definición:	Jx = Joint; Jx.type = 'fix'; Jx.iBindex = 2;

Esta restricción fijaría el cuerpo 2 al suelo. El comando también se utiliza cuando necesitamos definir un plano distinto del suelo, se definirá en inBodies y después deberá ser fijado en inJoints. (*Ver ejemplo de esfera y plano inclinado*).

Elementos disco.

Este tipo de restricciones asumen un elemento circular de tipo disco, sin grosor, con un punto de contacto con el suelo (**plano z=0**). Se pueden definir tres tipos de disco en función de que deslizamientos permitamos al punto en contacto con el suelo. Siendo las coordenadas locales del disco $\{\xi, \eta, \varsigma\}$, es importante que el eje 'η' esté orientado en la perpendicular al plano del disco.

DISK tipo 'xyz':

Se define así aquel disco cuyo punto instantáneo de rotación no tiene permitido el desplazamiento en la dirección 'z'.

Además, este tipo de discos no pueden deslizar en el plano x-y.

DISK tipo 'nz' y 'tz':

Se define así aquel disco cuyo punto instantáneo de rotación (CIR) no tiene permitido el desplazamiento en la dirección 'z'.

Además, este tipo de discos no pueden deslizar en dirección normal (n) del eje de la rueda (es decir solo puede deslizar lateralmente).

DISK tipo 'z':

Se define así aquel disco cuyo punto instantáneo de rotación (CIR) no tiene permitido el desplazamiento en la dirección 'z'. Si podrá deslizar en el plano z=0.

Definición y nomenclatura:

Tipo 'z' (*disk 1*)	
Definición de inputs	Jx = Joint; Jx.type = 'disk'; Jx.disk = `z`;

	Jx.iBindex = 5; Jx.R = 0.3;

Tipo ‘nz’ o ‘tz’ (*disk 2*)	
Definición de inputs	Jx = Joint; Jx.type = ‘disk’; Jx.disk = `nz`; Jx.iBindex = 3; Jx.R = 0.8;

Type ‘xyz’ (*disk 3*)	
Definición de inputs	Jx = Joint; Jx.type = ‘disk’; Jx.disk = `xyz`; Jx.iBindex = 1; Jx.R = 0.3;

Nota: a la hora de trabajar con discos habrá que ser especialmente cuidadoso con las restricciones que cada uno de los tipos introduce al sistema mecánico. En caso de que no se eligiesen correctamente el tipo de ‘disk joints’ para el conjunto de ruedas de un sistema, el sistema de ecuaciones a resolver será incompatible, dado que el deslizamiento en el plano de algunas ruedas es necesario para variaciones de dirección de ciertos móviles. En esta situación, al ejecutar el programa en MUBODYNA, éste entrará en un bucle infinito, intentando resolver un sistema de ecuaciones incompatible. El programa da la siguiente respuesta:

“**Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RECOND=3.12332e-18”**

En los ejemplos desarrollados más adelante se especificará como elegir el tipo de disco, para más información, ver el ejemplo resuelto de triciclo.

Por último, para enviar la información de las juntas o pares cinemáticos definidos en inJoints.m, deberemos escribir:

Joints = [J1; J2; … ; Jn];

2.2.8 inFunctions.

En este archivo podremos definir funciones y aplicarlas al sistema, en elementos como función del par transmitido por un motor, funciones no lineales de un muelle, etc. Las funciones definidas en el programa son tres:

Funciones tipo 'a'. Función y derivadas primera y segunda:

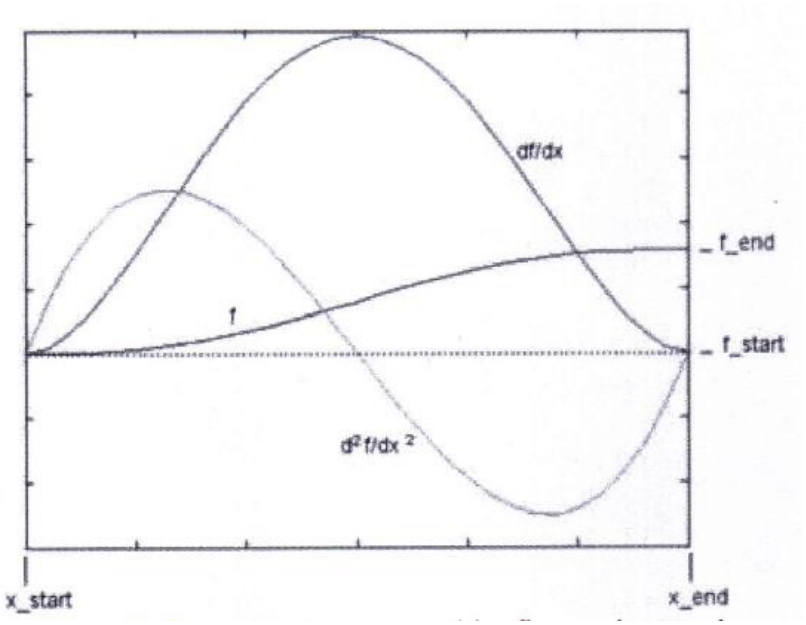
- $f = c_1 + c_2 x + c_3 x^2$
- $\frac{df}{dx} = c_2 + 2c_1 x$
- $\frac{d^2 f}{dx^2} = 2c_3$

El usuario deberá elegir los valores de $\boldsymbol{c_1, c_2\ y\ c_3}$.

Función tipo 'b'. Función y derivadas primera y segunda:

- $f = c_1 x^3 + c_2 x^4 + c_3 x^5$
- $\frac{df}{dx} = c_1 3x^2 + 4c_2 x^3 + 5c_3 x^4$
- $\frac{d^2 f}{dx^2} = 6c_1 x + 12c_2 x^2 + 20c_3 x^3$

La función y sus derivadas tienen la siguiente forma:

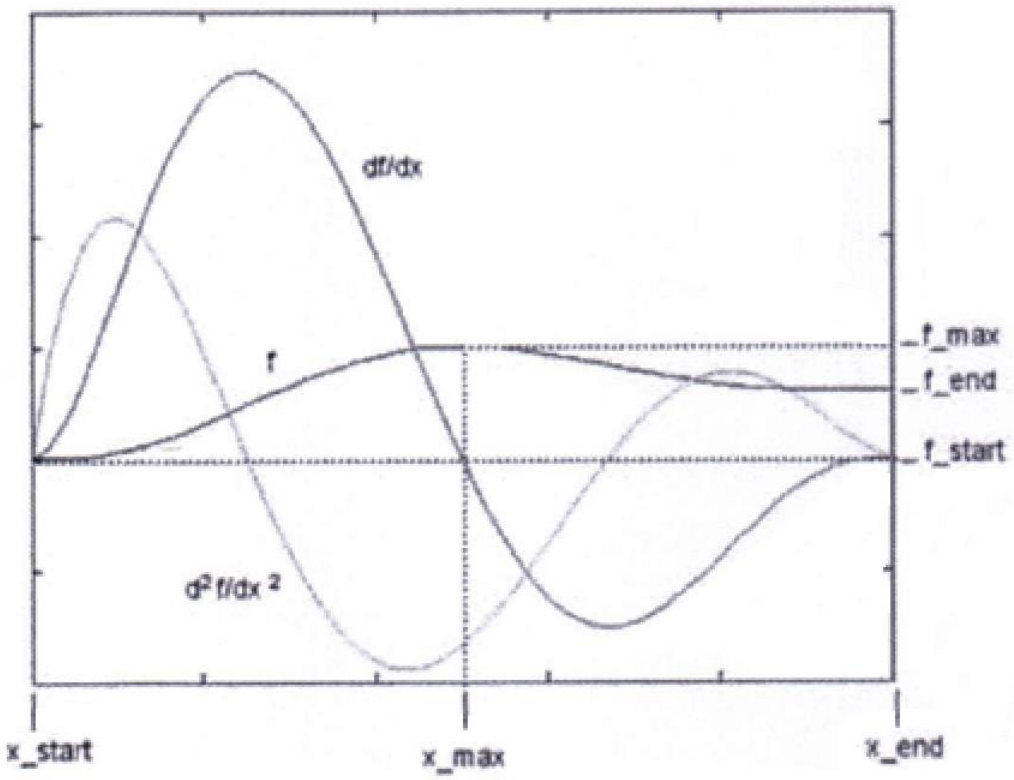


28. Función de tipo 2 y sus derivadas primera y segunda. [3]

Función tipo 'c'. Función y derivadas primera y segunda:

- $f = c_1x^3 + c_2x^4 + c_3x^5 + c_4x^6 + c_5x^7$
- $\frac{df}{dx} = 3c_1x^2 + 4c_2x^3 + 5c_3x^4 + 6c_4x^5 + 7c_5x^6$
- $\frac{d^2f}{dx^2} = 6c_1x + 12c_2x^2 + 20c_3x^3 + 30c_4x^4 + 42c_5x^5$

Su representación gráfica es:



29. Función tipo 'c' y sus derivadas primera y segunda. [3]

Para definir esta función es necesario dar los valores de comienzo y final (start/end) y los valores máximos de (x,f(x)). Según esas variables, el programa computa los coeficientes (c1,c2,c3,c4,c5) para formar la función de tipo c.

A continuación, se muestra un ejemplo de archivo inFunction.m en el que se define la función $f(x) = 5 + 2x + 1.5x^2$:

```
function inFuncts
    include_global
    Funct = Funct_struct;

C1 = Funct;
C1.coeff = [5 2 1.5];

Functs = [C1];
```

Como hemos hecho en los archivos anteriores, mandamos las funciones implementadas mediante Funct = [C1; C2; … ;Cn].

Además, en las funciones de tipo ‘b’ y ‘c’ se pueden dar los puntos (x,f(x)) inicial y final. Se adjunta la nomenclatura empleada para implementar funciones:

```
function Funct = Funct_struct
    include_global
    Funct = Funct_struct;
%Función tipo 'a'
C1 = Funct;
C1.type = 'a';
C1.coeff = [0 2*pi 0];
%Función tipo 'b'
C2 = Funct;
C2.type = 'b';
C2.x_start=0;
C2.f_start=0;
C2.x_end=0.8;
C2.f_end=0.8;
%Función tipo 'c'
C3 = Funct;
C3.type = 'c';
C3.x_start=0;
C3.f_start=0;
C3.x_end=0.8;
C3.f_end=0.8;
C3.f_max = 1.0;
...
    Functs=[C1;C2;C3];
```

2.2.9 inSolver.m

En este archivo debemos especificar la forma en que se resolverá el modelo propuesto por el usuario. Para ello deberemos indicar:

- ✓ Si se hace una corrección inicial, escribiendo 'n' (no) o 'y' (yes). Esta corrección inicial trata de comprobar que todos los puntos de interés, como los involucrados en definición de pares cinemáticos y juntas están situados donde deben, o si coinciden, si así es el caso por ejemplo dos puntos de dos cuerpos en una rótula. En caso de que uno esté descolocado (por algún error de definición en sus coordenadas) el programa lo colocará automáticamente, salvo en casos en los que el error sea muy grande.
- ✓ Indicar el tiempo final y los diferenciales de tiempo (reporting time-steps) en los cuales quedarán grabados los resultados de posición, velocidad, ect, de nuestro mecanismo.
- ✓ Indicar el método de resolución; standard o penalty method.

Veamos como se definiría este archivo, para un caso en el que deseamos que MUBODYNA corrija los errores iniciales, el tiempo de calculo sea de 8 segundos y los pasos de tiempo en los que se graban los resultados es de 0.2 segundos. El método de resolución deseado será el standard:

```
function inSolver
    include_global
%-------------------------------------------------------------------------
% This m-file contains all the necessary ingredients about the resolution
% of the equations of motion, such as integrator used, final time, etc.
%-------------------------------------------------------------------------
  correct_ic = 'y';         % flag used to correct the initial conditions
  t_final = 8;              % final time of simulation
  dt = 0.02;                % reporting time step

  method = 'standard';      % method used to solve the equations of motion

%-------------------------------------------------------------------------
```

Ahora sin corrección de errores y resolución mediante "penalty method":

```
function inSolver
    include_global
%-------------------------------------------------------------------------
% This m-file contains all the necessary ingredients about the resolution
% of the equations of motion, such as integrator used, final time, etc.
%-------------------------------------------------------------------------
    correct_ic = 'n';         % flag used to correct the initial conditions
    t_final = 2;              % final time of simulation
    dt = 0.02;                % reporting time step
    method = 'penalty';      % method used to solve the equations of motion
%-------------------------------------------------------------------------
```

Estos nueve archivos o m-files, son los que componen la estructura de un modelo. El usuario deberá crear en su carpeta contenedora del modelo estos nueve 'archivos.m'**. Es muy importante dar siempre los nueve archivos, incluso si uno o varios de ellos quedase en blanco porque no hiciese falta definir los elementos, deberá escribirse el archivo en blanco y dar la matriz output vacía. En caso contrario el programa dará un error al ejecutarse.**

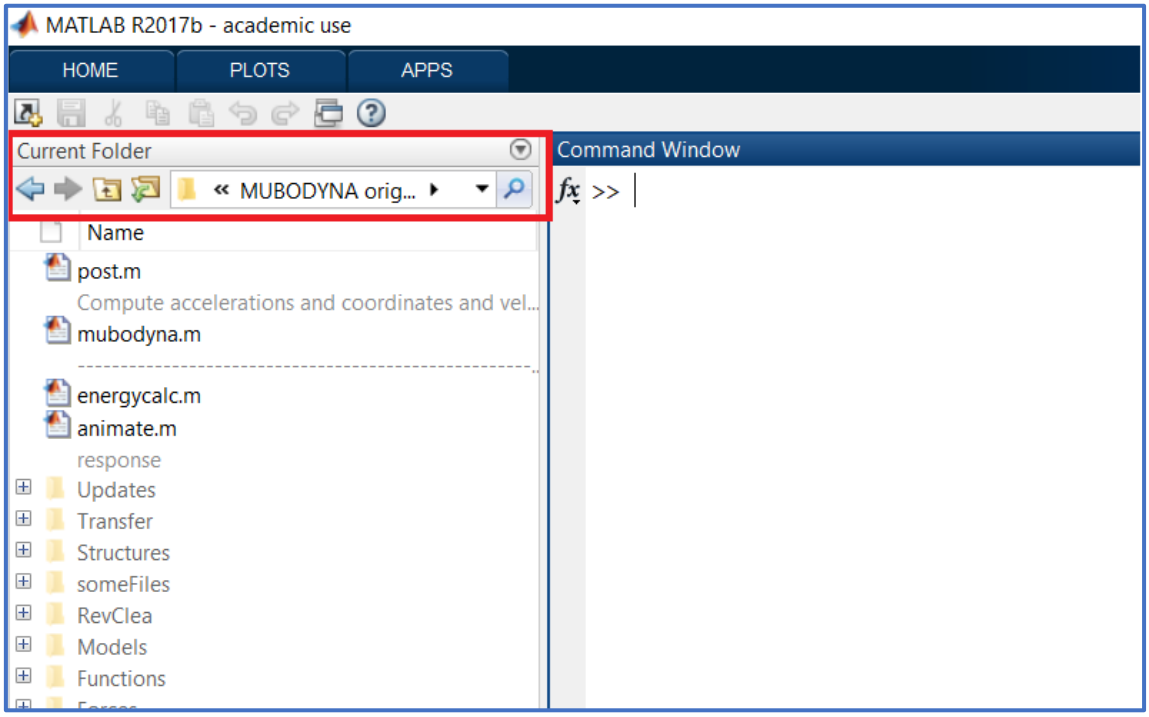
Por ejemplo, si en un sistema mecánico no nos hacía falta definir vectores de tipo 2, nuestro archivo inVectors2.m deberá quedar así:

```
function inVectors2
    include_global
    Vector = Vector2_struct;
                                %cuerpo del archivo en blanco

Vectors2 = [];                  %matriz output vacía
```

2.3 Análisis en MUBODYNA.

Para realizar el análisis de un modelo, deberemos seguir los siguientes pasos:

- Abrir Matlab, en el buscador de archivos de la ventana "current folder" debemos abrir la carpeta contenedora de MUBODYNA, y los diferentes archivos (models, Structures, Updates…) de la siguiente forma:



- Una vez dentro de la carpeta, podemos ejecutar el programa, para ello escribimos "mubodyna" en la ventana de comandos. Matlab ejecutará el programa y automáticamente nos preguntará por el modelo a analizar.

```
Command Window
>> mubodyna

   Which folder contains the model? sergio2esferasplano

   The total number of constraints is: 6
   The initial conditions were corrected!
   The final time of simulation is: 6 s
   The reporting time step is: 0.01 s

   Solving the equations of motion using standard method ...
   Solving the equations of motion using ODE Integrator ...

   The number of function evaluations is: 1303
   The CPU time consumed was: 63.4219 s

   Normal termination of MUBODYNA3D program!
```

- Ahora abrimos la carpeta "Models" y escribimos el nombre del modelo que deseamos analizar. Una vez hecho esto el programa comenzará a simular el modelo, y generar datos. Cuando ha terminado, nos ofrece un resumen con el número de restricciones que tiene el sistema mecánico, si se corrigieron las condiciones iniciales, el tiempo de simulación y los tiempos de informe o "reporting time steps", método de resolución y el tiempo computacional.


```
Command Window
>> mubodyna

   Which folder contains the model? sergio2esferasplano

   The total number of constraints is: 6
   The initial conditions were corrected!
   The final time of simulation is: 6 s
   The reporting time step is: 0.01 s

   Solving the equations of motion using standard method ...
   Solving the equations of motion using ODE Integrator ...

   The number of function evaluations is: 1303
   The CPU time consumed was: 63.4219 s

   Normal termination of MUBODYNA3D program!
```


En este ejemplo, se ha resuelto el modelo sergio2esferasplano. Que tiene 6 restricciones, se han corregido las condiciones iniciales, se ha simulado el sistema para una evolución de 6 segundos, los informes de datos han sido grabados en intervalos de

0.01 segundos. El método de resolución ha sido el estándar (ODE Integrator), y se ha consumido un tiempo para su análisis de 63.4219 segundos.

Cuando hacemos el análisis, a la derecha queda la ventana "workspace" con todas las matrices que contienen los datos y resultados de la simulación. Para ver sus datos podremos hacer doble click sobre la matriz deseada o bien analizarlas en el postprocesador como se verá más adelante. Conviene destacar dos matrices que serán de gran importancia, estas son **'T'** y **'uT'**.

La matriz 'T' guarda los diferentes pasos de tiempo (dt), en los cuales el programa ha guardado los datos del mecanismo, tales como posiciones y velocidades. Por ejemplo, para un problema cuyo tiempo de simulación es t=5s y el "recording time step" es de dt=0.01, (teniendo en cuenta que hay +1 por el t=0) tendremos nt=5/0.01 +1=501 pasos de tiempo y por tanto, la matriz 'T' será de dimensión 501x1.

Por otro lado, la matriz '**uT**' guarda los datos de posición y velocidad para cada paso de tiempo y cuerpo. Para un sistema con nB cuerpos (Bodies) la matriz uT será de **ntx(13*nB)**, siendo nt el número de pasos de tiempo. Recordemos que son 13 las variables que guarda de posición 'uT':

- La posición del centro de masas u origen de nuestro sistema local de coordenadas, dado por el vector r=[x,y,z] (3 coordenadas) y los parámetros de Euler p=[e_0, e_1, e_2, e_3] (4 coordenadas), 7 en total.
- Las velocidades del cuerpo tres para las velocidades lineales y tres para las velocidades angulares, 6 en total.

De modo que las primeras 7*nB columnas contienen las posiciones de los cuerpos y las siguientes 6*nB contienen las velocidades. Por tanto, en un sistema con 5 cuerpos, para saber la posición 'y_2' del cuerpo 3, en el time-step 36. Debemos escribir:

uT=(36,16);

Para saber la velocidad $\dot{w}_1$ del cuerpo 4 en time-step 12:

uT=(12,57);

Para facilitar el cálculo he desarrollado las siguientes formulas:

- Para posición: **uT=(dt, $\mathbf{7}(\boldsymbol{NB}-\mathbf{1})+\boldsymbol{nc}$);** Siendo **NB** el número de cuerpos del sistema y **'nc'** el número de coordenada que quieres saber del cuerpo, siendo 'x' la primera, y el parámetro de Euler e_3 la séptima.
- Para velocidad: **uT=($\boldsymbol{dt}, \mathbf{7}.\boldsymbol{NB}+\mathbf{6}(\boldsymbol{n}'-\mathbf{1})+\boldsymbol{nv}$)**; siendo **NB** número de cuerpos, **n'** el número del cuerpo del que deseamos saber la velocidad, y **'nv'** el número de velocidad del cuerpo que deseamos saber, siendo $\dot{x}$ la primera y $\dot{w}_3$ la sexta.

Dado que no es demasiado conveniente trabajar con T y uT, será aconsejable operar con el programa de post-procesado que se tratará un poco más adelante.

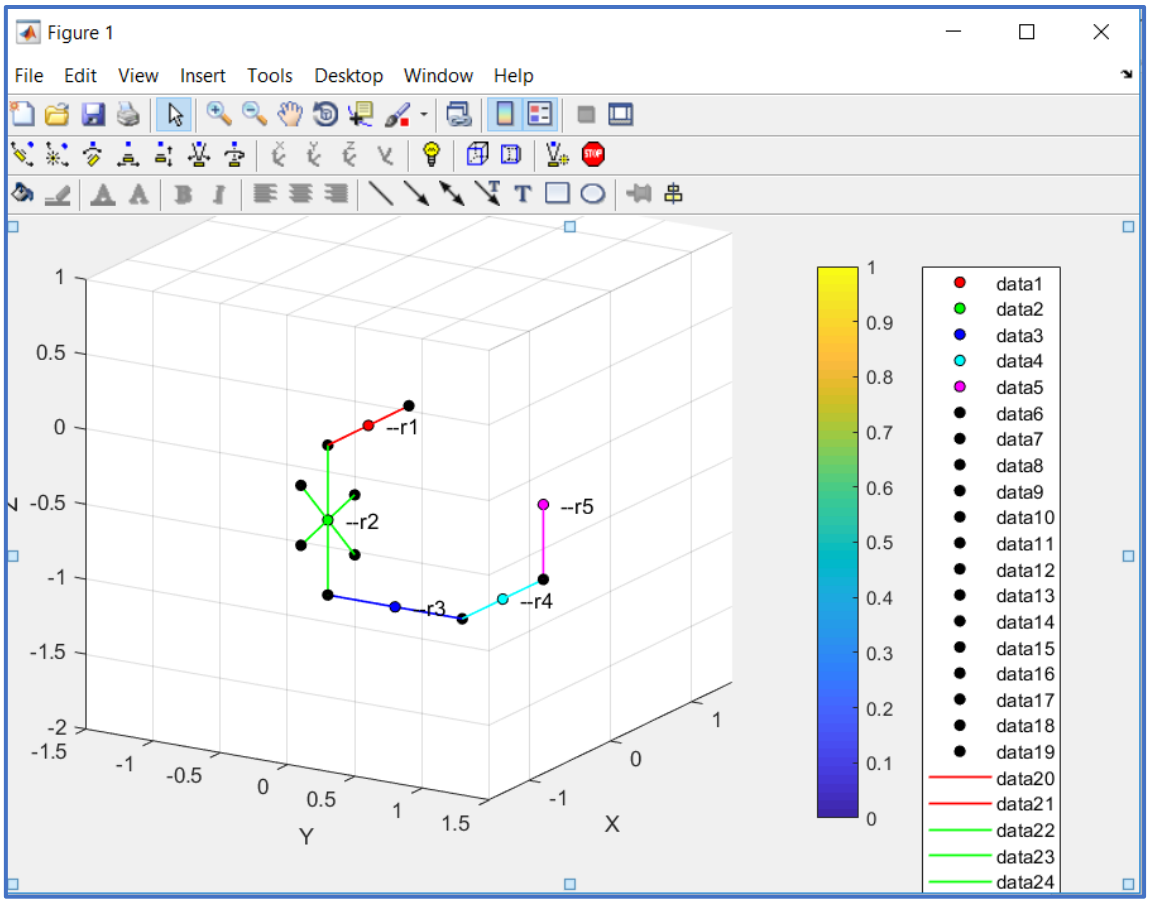
2.3.1 Animaciones: inAnimate.

Para realizar una simulación de el problema resuelto, el usuario deberá ejecutar el programa **"animate.m"**. Para ello simplemente se deberá tener la carpeta contenedora de los programas abierta, tal y como hicimos para ejecutar **mubodyna.m**, y escribir en la ventana de comandos: "animate", y presionamos enter.

Este programa accede a las coordenadas de los cuerpos desde la matriz **uT**, que podremos encontrar en la ventana Workspace, creando una animación desde los datos tomados en cada intervalo de tiempo. Además, si en el archivo **inAnimate.m** se definieron puntos, estos aparecerán dibujados en la animación junto con los otros definidos en **inPoints.m**, y los centros de masas de los cuerpos que se tomaron como origen de coordenadas locales. En cada paso de tiempo, se dibujan las líneas desde el origen del cuerpo (centro de masas) hasta los puntos definidos pertenecientes a dicho cuerpo. Para cada cuerpo se usan diferentes colores, lo que facilita la visualización. Estos dibujos son repetidos para cada intervalo de tiempo, creando la animación. Si la animación se desarrolla para un t=0 definido, es decir, sin evolución del movimiento en el tiempo, la animación crea una foto de la posición inicial del modelo. Esto puede ser muy útil para comprobar la estructura de nuevos modelos.

En el ejemplo podemos ver una "foto" realizada para el modelo Pendulo5, compuesto por cinco barras guiadas por un movimiento oscilatorio. Para observar su correcto montaje y definición de elementos, se hace una simulación para t=0.

```
>> mubodyna

   Which folder contains the model? Pendulo5

   The total number of constraints is: 25
   The initial conditions were not corrected!
   The final time of simulation is: 0 s
   The reporting time step is: 0.001 s


   The number of function evaluations is: 1
   The CPU time consumed was: 0.125 s

   Normal termination of MUBODYNA3D program!

>> animate
```



30. Animación de un mecanismo con cinco barras.

Si t>0, en lugar de ver una foto, Matlab reproducirá una animación, mostrando la evolución del movimiento del mecanismo a lo largo del tiempo t. Dentro de esta ventana de animación disponemos de varias herramientas para facilitar una mejor visualización. Conviene destacar la herramienta de cámara, mediante la que podremos rotar el cubo de visualización, hacer zoom, etc. Para acceder a ella: View/Camera Toolbar y aparecerán los comandos de control:

Además, se podrán seleccionar los elementos que aparecen en la animación mediante el Plot Browser (View/Plot Browser), siendo especialmente útil para visualizar mecanismos con gran numero de cuerpos, o si se quiere estudiar el movimiento de un elemento por separado.

2.3.1 Post-Processor.

El programa **post.m** se puede ejecutar después de hacer una simulación o una animación. Este programa, nos permite analizar los resultados guardados. Recoge los datos contenidos en la matriz 'uT' para cada paso de tiempo o "recording time step", y guarda los resultados en un formato con el que será más cómodo trabajar. Estas son las matrices y los resultados que guardan:

Matriz	Tamaño	Contenido
r	(nt,nB,3)	Coordenadas del cuerpo
p	(nt,nB,4)	Parámetros de Euler
rd	(nt,nB,3)	Velocidades lineales
w	(nt,nB,3)	Velocidad angular en coordenadas globales (x-y-z)
wp	(nt,nB,3)	Velocidades angulares en coordenadas locales (ξ-η-ς)
rdd	(nt,nB,3)	Aceleración traslacional
wd	(nt,nB,3)	Aceleración angular en coordenadas globales.
wpd	(nt,nB,3)	Aceleración angular en coordenadas locales.
rP	(nt,nB,3)	Coordenadas de los puntos
rPd	(nt,nB,3)	Velocidades de los puntos

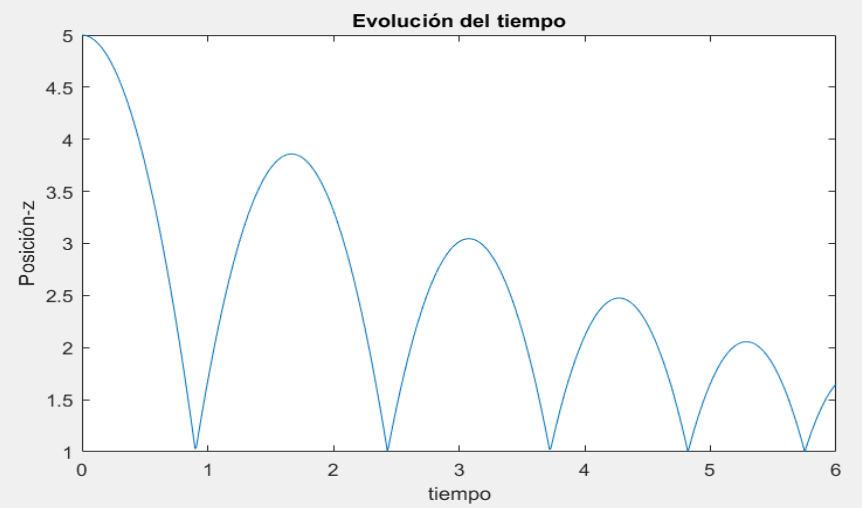
Por ejemplo, para saber la velocidad $\dot{x}$, del cuerpo 3 en el 6º paso de tiempo, lo obtendríamos mediante: $\boldsymbol{rd(6,3,1)}$. Y para representar esa velocidad del cuerpo 3 frente al tiempo, escribiríamos:

$$\boldsymbol{Plot(T, rd(:,3,1))}$$

Para obtener la 3ª velocidad angular del cuerpo 2, en el paso de tiempo nt=56, escribiríamos: wp=(56,2,3). Y para obtener su representación gráfica frente al tiempo de simulación escribiríamos:

$$\boldsymbol{Plot(T, w(:,2,3))}$$

La figura representa la evolución de la altura (z_1) de una pelota asociada al cuerpo B1 a lo largo del tiempo. La pelota se deja caer desde t=0 a una altura de h=5m. Su radio es de R=1m. Una vez hecho el análisis, se ejecuta el post-procesador escribiendo "post" y presionando enter. Después ejecutamos el comando **plot(T,r(:,1,3))**. Obteniendo:



3. CAPÍTULO III: Modelos desarrollados en MUBODYNA.

En el siguiente punto, se muestran un conjunto de modelos analizados en MUBODYNA, donde se pretende explicar como se formula y desarrollan distintos modelos de sistemas mecánicos para su análisis y simulación. A su vez se explicará la metodología y razonamientos a seguir a la hora de enfrentarse a estos problemas, para y analizar los resultados, sacando todo el partido a nuestro programa.

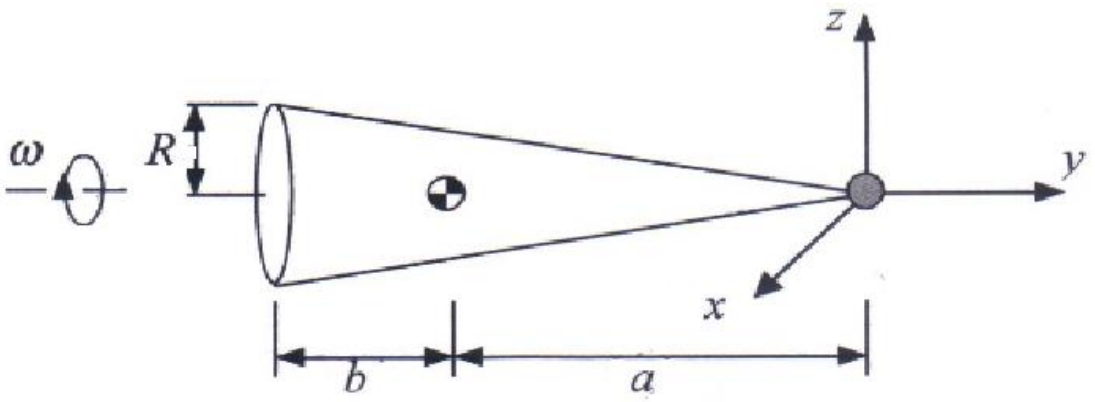
Si bien, para describir los diferentes archivos.m no es necesario seguir un orden concreto, en este trabajo se ha seguido un orden concreto, que ayuda a crear una metodología que para organizar las ideas a la hora de describir los diferentes elementos. Por tanto, los archivos se han creado en el orden que se presenta en los diferentes modelos.

3.1 Modelo 1: péndulo simple.

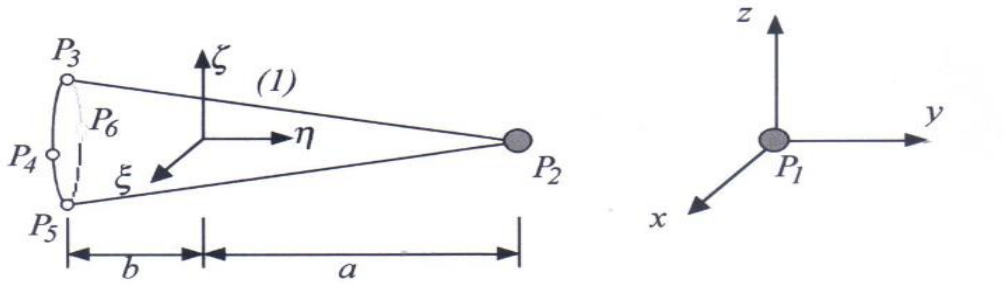
En este modelo, se va a analizar el movimiento de un cono como el que se muestra en la figura. Este cono está sujeto en su vértice mediante un par esférico junto al suelo, pudiendo girar libremente pero no desplazarse. El objeto se suelta libremente desde la posición horizontal con una velocidad angular de -5rad/s en la dirección 'η', eje del cono. Las dimensiones, masa y momentos de inercia del objeto se describen a continuación:

$a = 0.3\ m, b = 0.1\ m\ , R\ = 0.15, g = 9.8\ m/s^2$

$m = 4kg, j_{\xi\xi} = j_{\varsigma\varsigma} = 0.0375\ kg.m^2, j_{\eta\eta} = 0.0270\ \ kg.m^2$



Para resolver este modelo, en primer lugar, debemos asignar números a los diferentes elementos que vamos a usar. En las siguientes figuras se muestra un esquema de los puntos y cuerpos que vamos a usar, es conveniente hacer estos esquemas para facilitar la descripción de los diferentes elementos, a lo largo de los '***archivos.m***'. Los cuerpos y puntos usados quedarán numerados de la siguiente forma:



Como tenemos los números asignados para el suelo (0) y el cuerpo, podemos proceder a definirlos en el archivo inBodies.m.

inBodies.m:

En este archivo vamos a definir el cono como cuerpo 1, y sus propiedades de masa, momento de inercia, y posición mediante la posición de su centro de masas y los parámetros de Euler, teniendo en cuenta el ángulo que forman los ejes locales con los globales, siendo paralelos ($\Theta = 0$) en esta ocasión. En el programa todas las unidades se darán en sistema internacional.

```
function inBodies
    include_global
    Body = Body_struct;

B1 = Body;
B1.mass = 4.0;
B1.Jp = [0.0375 0      0
         0    0.0270   0
         0      0    0.0375];
B1.r = [0;-0.3;0];
B1.p = [1;0;0;0];
B1.w = [0;-5;0];

Bodies = [B1];
```

Tal y como se explicó en la estructura y nomenclatura de este archivo, comenzamos definiendo el cono como cuerpo 1, e introducimos la masa y matriz de inercia en unidades del sistema internacional. Después se da la posición del centro de masas expresado en ejes globales, mediante B1.r, y los ángulos iniciales que forman los ejes locales con los globales mediante los parámetros de Euler B1.p=[$e_0; e_1; e_2; e_3$]. Como en este caso los ejes son paralelos para t=0, los parámetros quedan B1.p=[1;0;0;0].

Por último, se dan las velocidades del cuerpo en sistema local de referencia, en este caso la rotación de -5rad/s, en el eje-η.

El suelo no es necesario definirlo ni asociarlo, ya que por defecto el programa ya lo asocia al cuerpo '0'.

Finalmente, definimos el vector Bodies con los cuerpos que deseamos dar como dato, y cuya información será enviada al programa cuando se ejecute.

El siguiente paso será definir los puntos clave del sistema, que nos permitirán describir el resto de elementos, como vectores, juntas o pares cinemáticos, etc. Además estos puntos serán representados en la animación.

inPoints.m

En primer lugar, hay que definir la estructura del archivo creando las llamadas de función punto e Include_global, y llamando a la estructura de punto. Tras esto se pueden definir los distintos puntos en coordenadas locales

```
function inPoints
    include_global
    Point = Point_struct;

P1 = Point;
P1.Bindex = 0;
P1.sPp = [0;0;0];

P2 = Point;
P2.Bindex = 1;
P2.sPp = [0;0.3;0];

Points = [P1; P2];
```

Cómo se puede observar, solo se definen los puntos 1 y 2, que serán utilizados para definir nuestra junta esférica (o “spherical joint”), siendo los únicos puntos necesarios para la descripción de otros elementos. Por tanto, los puntos 3,4,5 y 6, son definidos únicamente con fines de visualización, y deben ser introducidos en el archivo **inAnimate.m**.

Para definir los puntos, será suficiente con asociarlos al cuerpo que pertenecen y dar sus coordenadas locales.

inVectors1.m

En este problema no es necesario definir ningún vector, aún así como se apuntó anteriormente, se deberá dar un vector vacío para evitar un error del programa.

```
function inVectors1
    include_global
    Vector = Vector1_struct;

Vectors1 = [];
```

inVectors2.m

Dado que tampoco son necesarios, dejamos el archivo vacío de la siguiente forma:

```
function inVectors2
    include_global
    Vector = Vector2_struct;

Vectors2 = [];
```

inJoints.m

Ahora procedemos a definir las juntas que definen los pares cinemáticos. En este caso solo hay una que es la junta de tipo esférico, y que debemos definir con los puntos 1 y 2 de ambos cuerpos.

Tal como vimos en el punto 2.2.7, la junta esférica se define especificando su tipo (`sph') y los dos puntos 'i', 'j' pertenecientes a los dos cuerpos unidos mediante esta junta.

```
function inJoints
    include_global
    Joint = Joint_struct;

J1 = Joint;
J1.type = 'sph';
J1.iPindex = 2;
J1.jPindex = 1;

Joints = [J1];
```

**Recordar que siempre que haya un punto perteneciente al "suelo", deberá ir asociado al índice 'j', ya sea para describir juntas cinemáticas, fuerzas, vectores o cualquier otro elemento.

inForces.m

La fuerza que causa el movimiento en nuestro sistema es únicamente la gravedad, por lo que nuestro archivo de fuerzas quedaría:

```
function inForces
    include_global
    Force = Force_struct;

S1 = Force;
S1.type = 'weight';  % fuerza tipo peso
S1.gravity = 9.81;   % valor por defecto
S1.wgt = [0; 0; -1]; % vector dirección de la gravedad por defecto

Forces = [S1];
```

Expresando el tipo de fuerza según la nomenclatura vista anteriormente, definimos la gravedad, y damos su output mediante el vector Forces.

inFunction.m

En este problema no nos hace falta ninguna función por lo que se define en blanco para evitar errores.

```
function inFuncts
    include_global
    Funct = Funct_struct;

Functs = [];
```

inAnimate.m

En este modelo, los puntos 3,4,5 y 6 tienen el único objetivo de mejorar la visualización, por lo que han de definirse aquí. Además, especificamos las dimensiones del cubo de animación, y su orientación mediante AZ y EL.

AZ= 60 indica un giro de 60 grados en torno al eje ‘z’ del cubo de visualización, y EL=-10, indica un giro de -10 en torno al eje ‘x’ horizontal, elevando el cubo.

```
function inAnimate
    include_global
    Point = Point_struct;

P3 = Point;
P3.Bindex = 1;
P3.sPp = [0; -0.1; 0.1];

P4 = Point;
P4.Bindex = 1;
P4.sPp = [0.1; -0.1; 0];

P5 = Point;
P5.Bindex = 1;
P5.sPp = [0; -0.1; -0.1];

P6 = Point;
P6.Bindex = 1;
P6.sPp = [-0.1; -0.1; 0];

Points_anim = [P3;P4;P5;P6];

% Definimos los vectores 3D que contiene el cubo de animación
    xmin = -0.4; xmax =  0.4;
    ymin = -0.4; ymax =  0.4;
    zmin = -0.4; zmax =  0.4;
% Rotacion del cubo:
% AZ rota al rededor del eje 'z', EL rota sobre el eje horizontal 'x'.
    AZ = 60; EL = -10;
```

inSolver.m

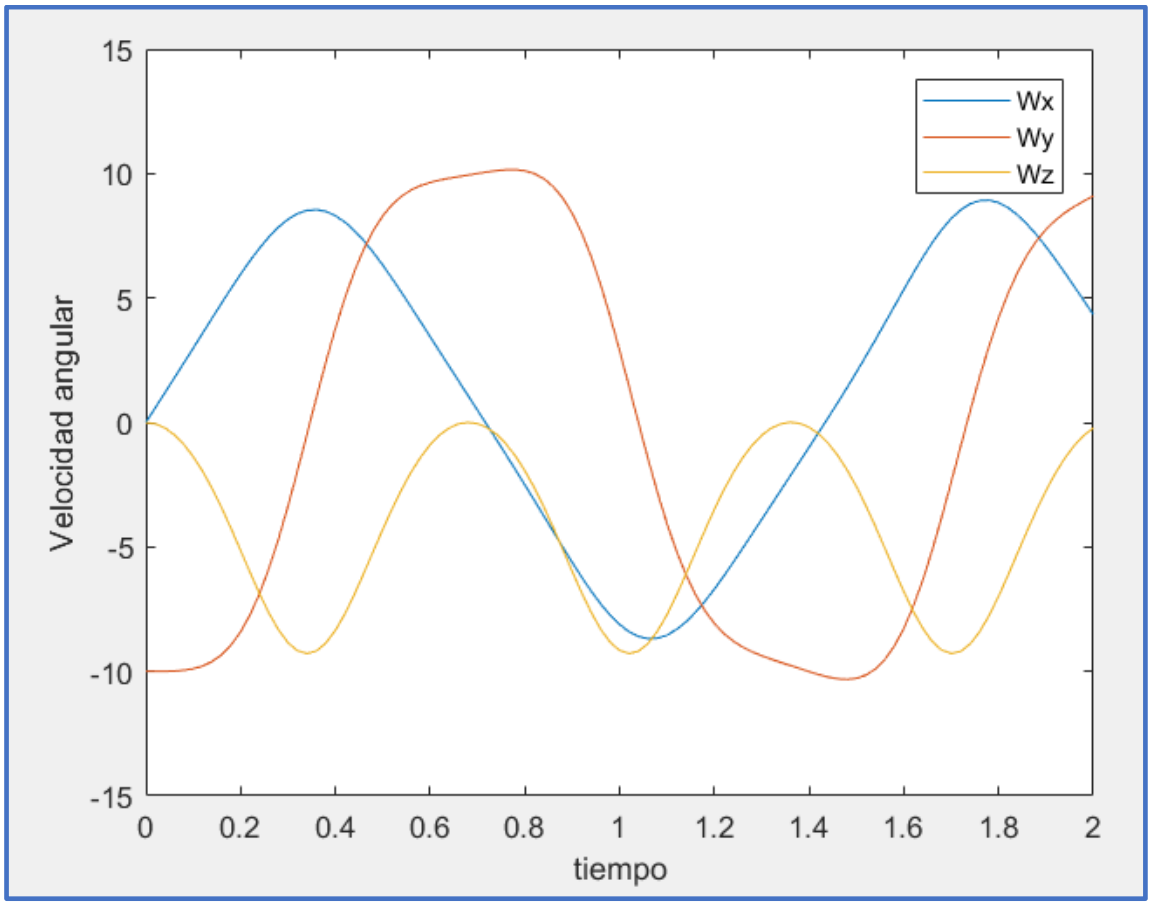
Ahora vamos a seleccionar los parámetros de resolución del modelo. En esta ocasión queremos que MUBODYNA nos resuelva el problema mediante el método de integración standard de los multiplicadores de Lagrange. Además, al ser un problema sencillo y fácil de ensamblar no vamos a hacer corrección de errores. Los pasos de tiempo de integración serán de 0.02 segundos y el tiempo de simulación de 5 segundos. Con lo que nuestro código quedaría:

```
function inSolver
    include_global

    correct_ic = 'n';        % no corregimos errores iniciales
    t_final = 5;             % tiempo final de simulación
    dt = 0.02;               % reporting time step
    method = 'standard';     % método de resolución de ecuaciones
```

Post-procesador.

Tras ejecutar Mubodyna y hacer el análisis, vamos a observar algunos resultados con el post-procesador. Para ello, escribimos "post" en la ventana de comandos de Matlab, esperamos a que procese la información y una vez termine, podremos ejecutar los comandos del post-procesador.

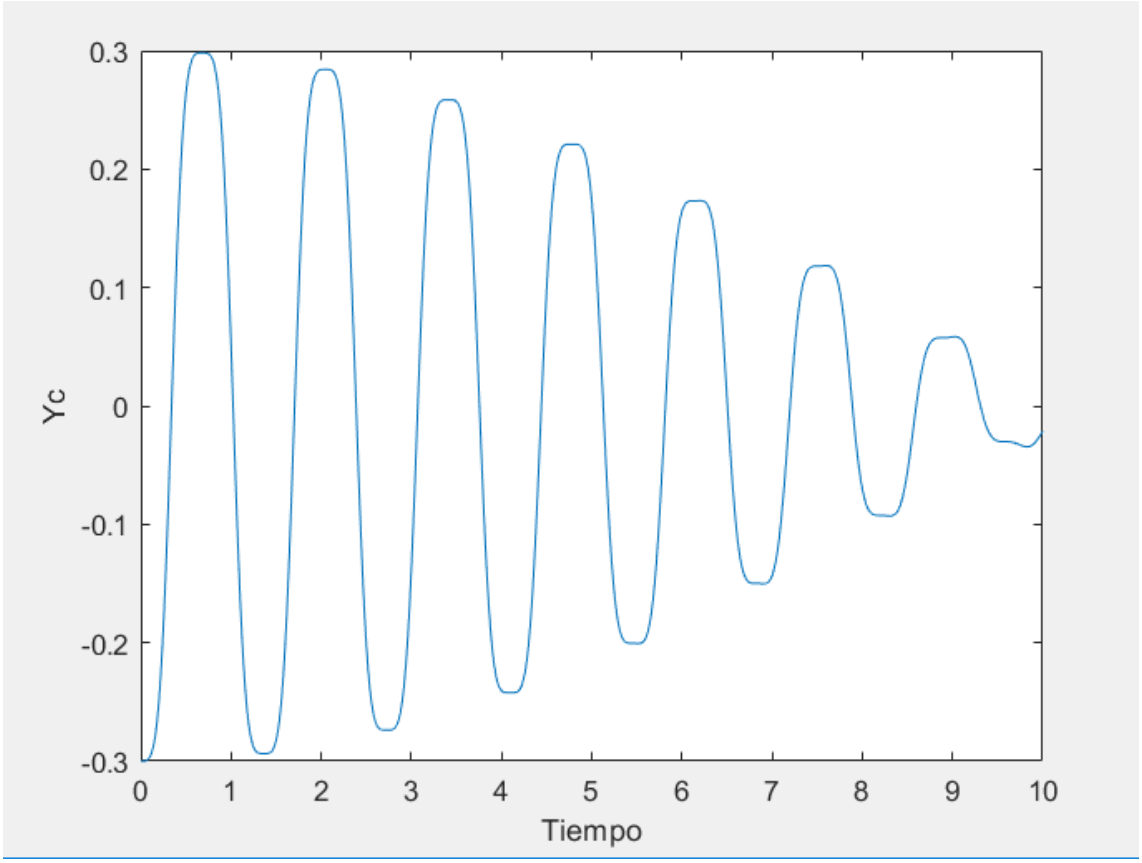
En primer lugar, analizamos la velocidad angular del cono, haciendo las gráficas de $\vec{w} = (w_x, w_y, w_z)$ frente al tiempo. Para ello, como vimos en el apartado 2.3.2, escribiendo plot(T,w(:,1,1),T,w(:,1,2),T,w(:,1,3)) obtendremos nuestra gráfica:



31.Velocidades angulares vs Tiempo. Péndulo simple.

Como podemos observar para el tiempo t=0, el cuerpo solo tiene la rotación inicial sobre el eje-η, que está paralelo a 'y' en el instante inicial. Por lo que $w_x = w_z = 0\ y\ w_y = -10$m/s.

También podemos observar el movimiento de oscilación del péndulo mediante la posición del centro de masas. Para una mejor observación hacemos la simulación de 10 segundos. Mediante: plot(T,r(:,1,2))



32. Movimiento oscilatorio del centro de masas.

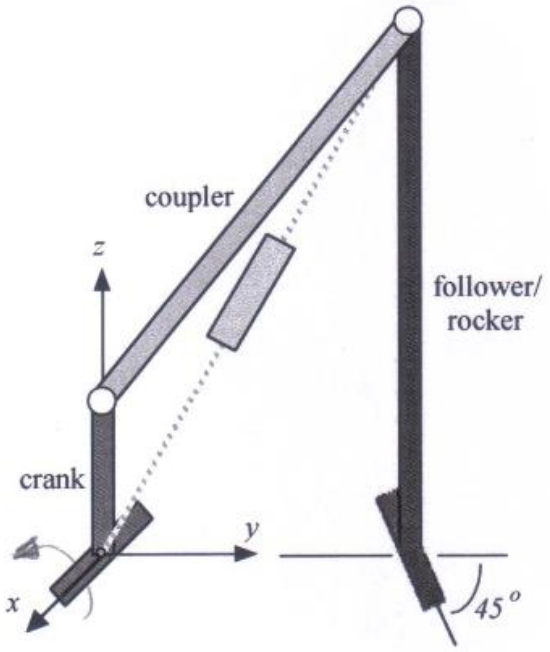
3.2 Modelo 2: mecanismo de cuatro barras.

El mecanismo llamado de cuatro barras, consiste en cuatro cuerpos llamados barras o eslabones, conectados por cuatro uniones en cadena. Generalmente las uniones se mueven en planos paralelos, aunque también pueden ser tridimensionales. Es el mecanismo que rige las máquinas de extracción de petróleo, conocidos como bomba de varilla o unidad de bombeo:

En nuestro problema vamos a modelar un mecanismo de cuatro barras que contiene tres cuerpos en movimiento, uno fijo que será el suelo, dos juntas de revolución y dos juntas esféricas. Además, habrá que implementar un motor entre el suelo y la manivela, que transmitirá el par a la manivela ("crank"), para empujar la biela ("coupler") ambos pares definidos mediante una junta de revolución. Además, entre los extremos de la biela y la manivela situaremos un sistema de amortiguación, tal y como muestra el esquema. El par de revolución que une el suelo con la manivela gira en torno a "x", mientras que el otro par ligado al suelo gira en torno a un eje de 45º entre los ejes "x" e "y".

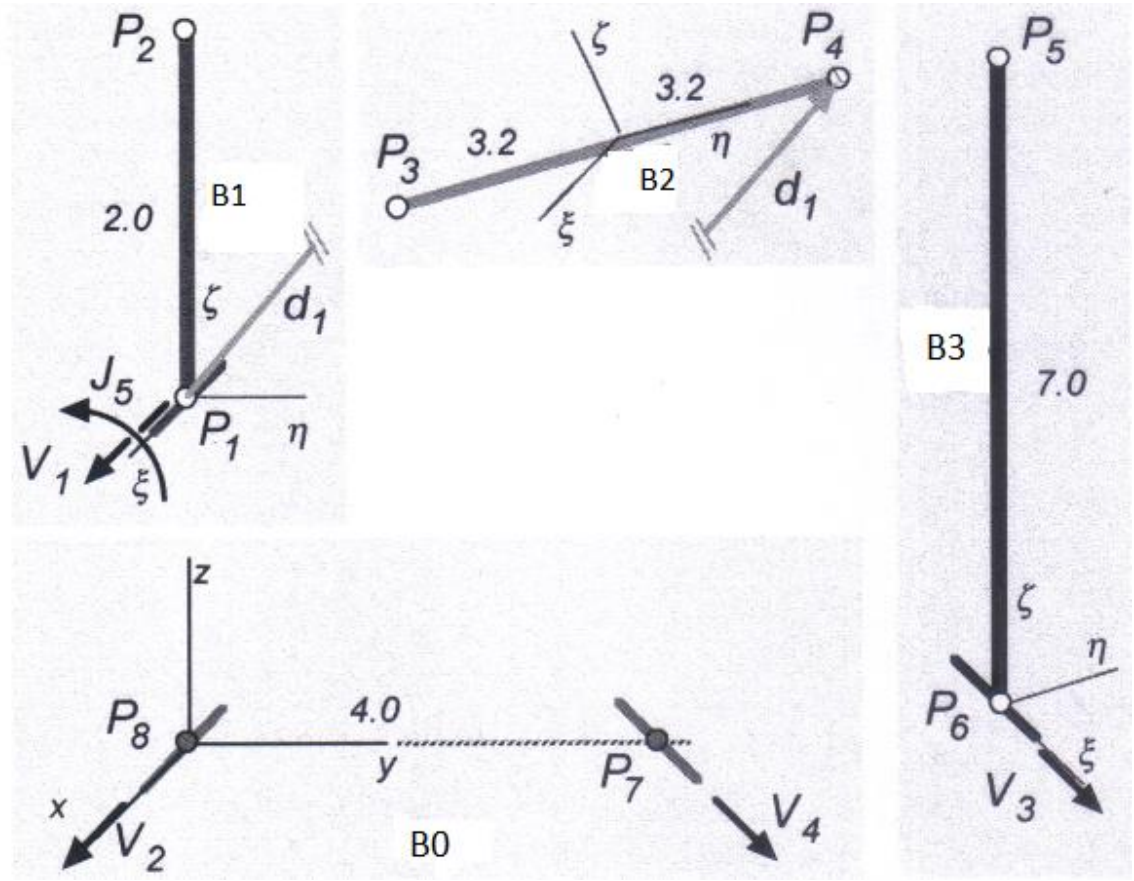
Las longitudes de los eslabones son: manivela=2.0m, biela =6.4m, seguidor=7.0m, distancia entre puntos del suelo=4.0m.

El motor gira la manivela con una velocidad constante de 2π rad/s. En la posición inicial la manivela está a lo largo del eje 'z' y los otros dos eslabones están contenidos en el plano 'y-z'. Al ser un sistema cuyo movimiento está dirigido por el motor, las masas no cambiarán nuestros resultados de movimiento (salvo que deseemos estudiar las fuerzas involucradas), por lo que asignamos unos valores aleatorios.



33. Mecanismo de cuatro barras.

El esquema de numeración usado para resolver el problema es el siguiente:



Una vez hecho el esquema y numeración de puntos, cuerpos, vectores y orientación de ejes locales de cada cuerpo. Podemos pasar a implementar los diferentes archivos:

inBodies.m

En este mecanismo tenemos cuatro cuerpos. Como uno de ellos es el suelo y por defecto ya está asignado en el programa al número '0', no hará falta definirlo. El archivo inBodies.m quedaría así:

```
function inBodies
    include_global
    Body = Body_struct;

B1 = Body;
B1.mass = 0.5;
B1.Jp = [0.03 0 0
         0 0.03 0
         0 0 0.03];
B1.w = [2*pi; 0; 0];

B2 = Body;
B2.r = [0;2;4.5];
phi = 51.34*pi/180;    %Como los ejes locales no son paralelos
    p2 = phi/2;        %hay que definir los parámetros de Euler.
    e0 = cos(p2);
    e1 = sin(p2);
B2.p = [e0;e1;0;0];
B2.mass = 0.15*10;
B2.Jp = [0.02 0 0
         0 0.02 0
         0 0 0.02];
B3 = Body;
B3.r = [0;4;0];
phi = 45*pi/180;
    p2 = phi/2;
    e0 = cos(p2);
    e3 = sin(p2);
B3.p = [e0;0;0;e3];
B3.mass = 0.15;
B3.Jp = [0.02 0 0
         0 0.02 0
         0 0 0.02];

Bodies = [B1; B2; B3];
```

Los sistemas locales de coordenadas de las barras 1,2 y 3, no son paralelos a los ejes globales y por tanto hay que definir su orientación mediante los parámetros de Euler, tal como se muestra en la imagen.

inPoints.m

Los puntos hay que definirlos cuidadosamente ya que muchos elementos como "joints" o vectores se definirán en función de ellos. En este mecanismo tenemos ocho puntos a definir, necesarios para hacer el modelado.

```
function inPoints
    include_global
    Point = Point_struct;

P1 = Point;
P1.Bindex = 1; %Por defecto sPp=[0;0;0].(P1, origen de s.local).

P2 = Point;
P2.Bindex = 1;
P2.sPp = [0;0;2];

P3 = Point;
P3.Bindex = 2;
P3.sPp = [0;-3.2016;0];

P4 = Point;
P4.Bindex = 2;
P4.sPp = [0;3.2016;0];

P5 = Point;
P5.Bindex = 3;
P5.sPp = [0;0;7];

P6 = Point;
P6.Bindex = 3;

P7 = Point;
P7.Bindex = 0;
P7.sPp = [0;4;0];

P8 = Point;
P8.Bindex = 0;
P8.sPp = [0;0;0];

Points = [P1; P2; P3; P4; P5; P6; P7; P8];
```

Damos las coordenadas locales de los puntos, en función de como se definieron en el esquema. Habitualmente el origen será el centro de masas, aunque en este caso se optó por tomar otros orígenes.

inVectors1.m

En este problema, se necesitan cuatro vectores para definir los dos pares cinemáticos de revolución. Como vimos en el apartado de uniones, una junta de revolución se define mediante dos vectores paralelos, uno perteneciente a cada cuerpo y anclados en los puntos de rotación.

```
function inVectors1
    include_global
    Vector = Vector1_struct;

V1 = Vector;
V1.Bindex = 1;
V1.sp = [1;0;0];

V2 = Vector;
V2.Bindex = 0;
V2.sp  = [1;0;0];

V3 = Vector;
V3.Bindex = 3;
V3.sp = [1;0;0];

V4 = Vector;
V4.Bindex = 0;
V4.sp  = [1;1;0]/sqrt(2);

Vectors1 = [V1; V2; V3; V4];
```

De este modo, los vectores 1 y 2, serán usados para definir la junta de revolución 3. Y los vectores 3 y 4, definirán la junta de revolución 4, como se verá más adelante.

inVectors2.m

Definiremos un vector de tipo 2, entre los puntos 1 y 4 que será necesario en la implementación del amortiguador que actúa entre las barras 1 y 2.

```
function inVectors2
    include_global
    Vector = Vector2_struct;

V1 = Vector;
V1.iPindex = 1;
V1.jPindex = 4;

Vectors2 = [V1];
```

inJoints.m

En este archivo vamos a implementar cinco juntas, dos de revolución, dos esféricas en los puntos 2/3 y 4/5, y necesitamos definir el par motor que guiará el movimiento y que se coloca entre el suelo y la barra 1. Siempre que se necesite un motor en un sistema se implementa mediante la junta “rel-rot”, tal y como se muestra a continuación:

```
function inJoints
    include_global
    Joint = Joint_struct;

J1 = Joint;
J1.type = 'sph';
J1.iPindex = 2;
J1.jPindex = 3;

J2 = Joint;
J2.type = 'sph';
J2.iPindex = 4;
J2.jPindex = 5;

J3 = Joint;
J3.type = 'rev';
J3.iPindex = 1;
J3.jPindex = 8;
J3.iVindex_a = 1;
J3.jVindex_a = 2;

J4 = Joint;
J4.type = 'rev';
J4.iPindex = 6;
J4.jPindex = 7;
J4.iVindex_a = 3;
J4.jVindex_a = 4;

J5 = Joint;
J5.type = 'rel-rot';
J5.iBindex = 1;
J5.jVindex_a = 2;
J5.iFunct  = 1;

Joints = [J1; J2; J3; J4; J5];
```

Como se observa el par motor está dirigido por la función 1, que habrá que definir con ese índice posteriormente.

inFunctions.m

Aquí se define la función rectora del motor. En este caso necesitaremos una función de tipo 'a', ya que el motor hace girar la manivela a 2π rad/s. Siendo una velocidad constante e independiente del tiempo, necesitamos implementar una función de tipo 'a':

$$\boldsymbol{f(t) = c_1 + c_2 x + c_3 x^3 \text{ (rad)}}$$

Para definir la velocidad angular constante, los coeficientes han de ser:

$c_1 = 0;\ c_2 = 2\pi;\ c_3 = 0$

```
function inFuncts
    include_global
    Funct = Funct_struct;

C1 = Funct;
C1.type = 'a';
C1.coeff = [0 2*pi  0];

Functs = [C1];
```

inForces.m

Las dos fuerzas que experimenta el sistema son la gravedad y la fuerza del muelle entre los puntos 1 y 4. La rigidez del muelle es de 50N/m y coeficiente de amortiguación c=0. Se definen de la siguiente forma:

```
function inForces
    include_global
    Force = Force_struct;

S1 = Force;
S1.type = 'ptp'; % "point to point"
S1.V2index = 1;
S1.k = 50;
S1.el_0 = 8;
S1.c = 0;
S1.f_a = 0;

S2 = Force;
S2.type = 'weight';  % tipo peso
S2.gravity = 9.81;   % por defecto
S2.wgt = [0; 0; -1]; % por defecto

Forces = [S1; S2];
```

Como vimos en la nomenclatura y estructura de fuerzas, la fuerza "tipo ptp", (point to point) que se utiliza para definir el muelle, necesita ser asociada a un vector de tipo 2 que una los puntos en los que actúa. Además, se proporcionan los valores de rigidez del muelle (k), elongación natural (el_0), coeficiente de amortiguación (c) y fuerza constante del actuador (f_a).

inAnimate.m

Se definen los parámetros para la animación, en este caso no se añaden puntos extra para visualización, por lo que el vector de puntos se envía vacío.

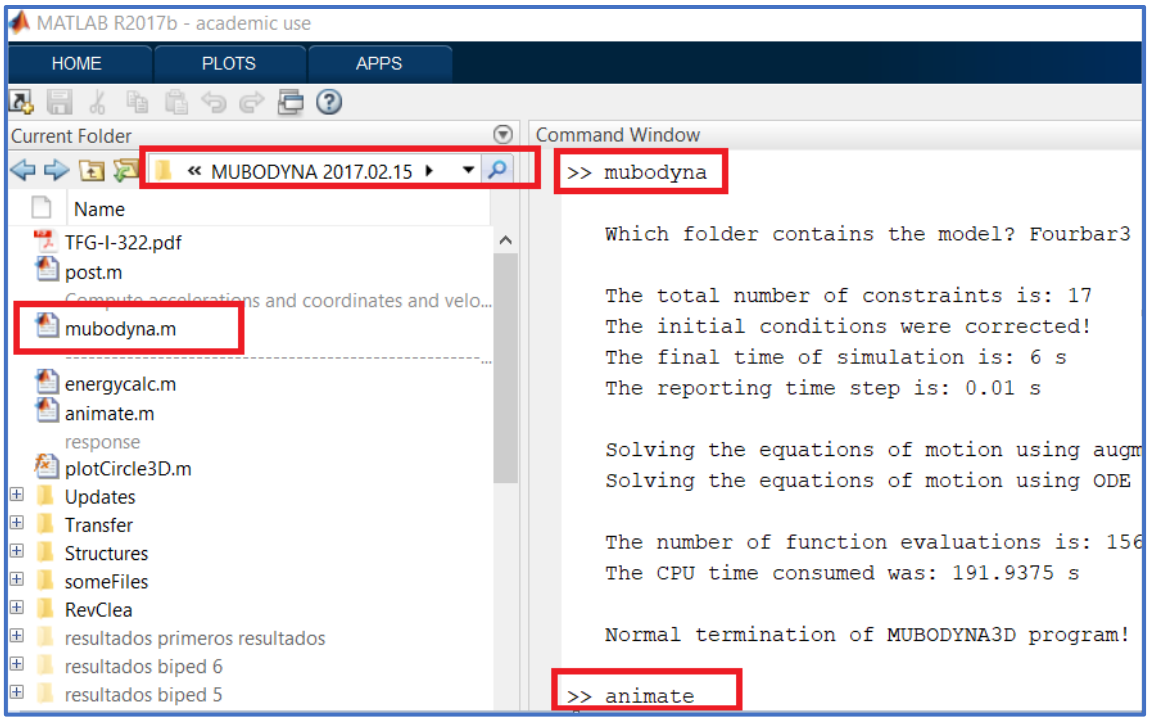
```
function inAnimate
    include_global
    Point = Point_struct;

Points_anim = [];   %no se definen puntos para visualización.

%Variables para la animación en 3D:

xmin = -5; xmax =  5;
ymin = -5; ymax =  5;
zmin = -2; zmax =  8;

%Rotación de los ejes de coordenadas. AZ rota sobre el eje Z, EL
%rota sobre el eje horizontal.
AZ = 130;
EL = -30;
```

inSolver.m

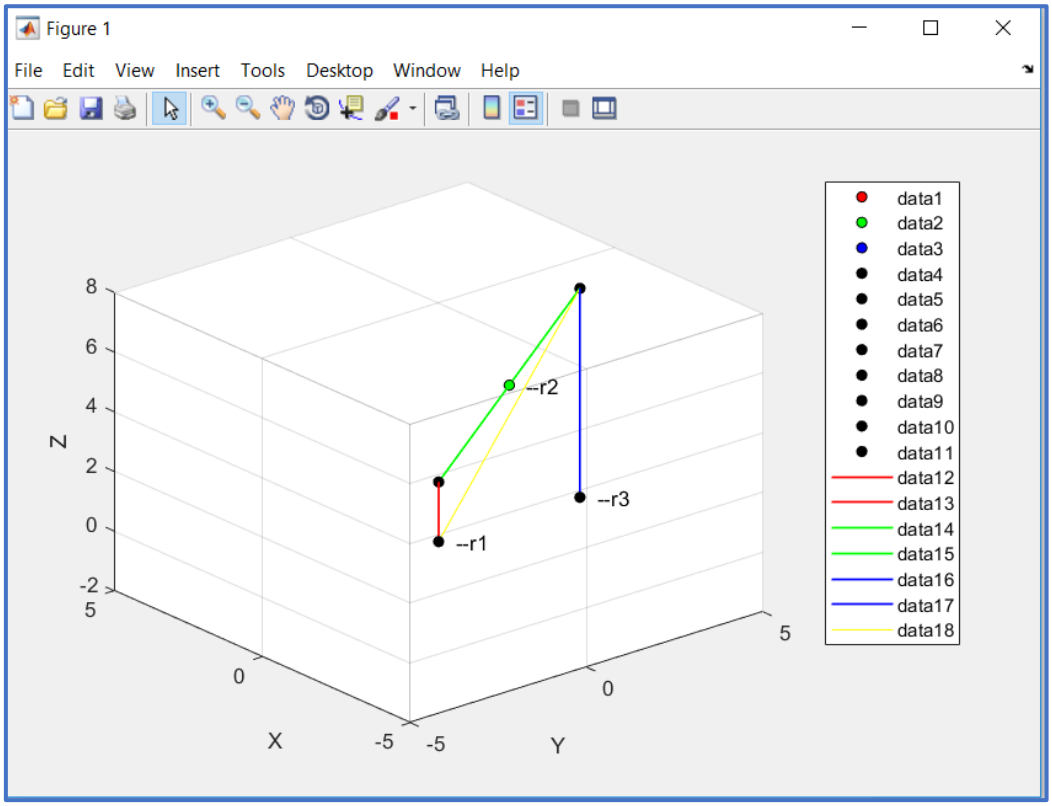
Para este problema vamos a hacer una corrección inicial de errores, el tiempo de simulación será de 6 segundos con intervalos de informe (time-steps) de 0.01 segundos. El método de integración será el de formulación Lagrangiana aumentada ("augmentedo method").

```
function inSolver
    include_global
%--------------------------------------------------------------------------
% This m-file contains all the necessary ingredients about the resolution
% of the equations of motion, such as integrator used, final time, etc.
%--------------------------------------------------------------------------
    correct_ic = 'y';         % corregir condiciones iniciales.
    t_final = 6;              % tiempo final de simulación.
    dt = 0.01;                % reporting time step
    method = 'augmented';      % método de resolución de ecuaciones.
%--------------------------------------------------------------------------
```

Para realizar la simulación, procedemos a abrir Matlab, abrimos la carpeta contenedora del programa y escribimos mubodyna.m. Una vez se ha ejecutado el programa, nos pregunta que modelo queremos resolver, en nuestro caso se llama "Fourbar3", y presionamos enter. Una vez el programa ejecuta el modelo y lo resuelve, hacemos la animación escribiendo, "animate".



El programa mostrará una animación del modelo, y nos permitirá su análisis de resultados tanto en la animación como con el post-procesador, si así se desea.



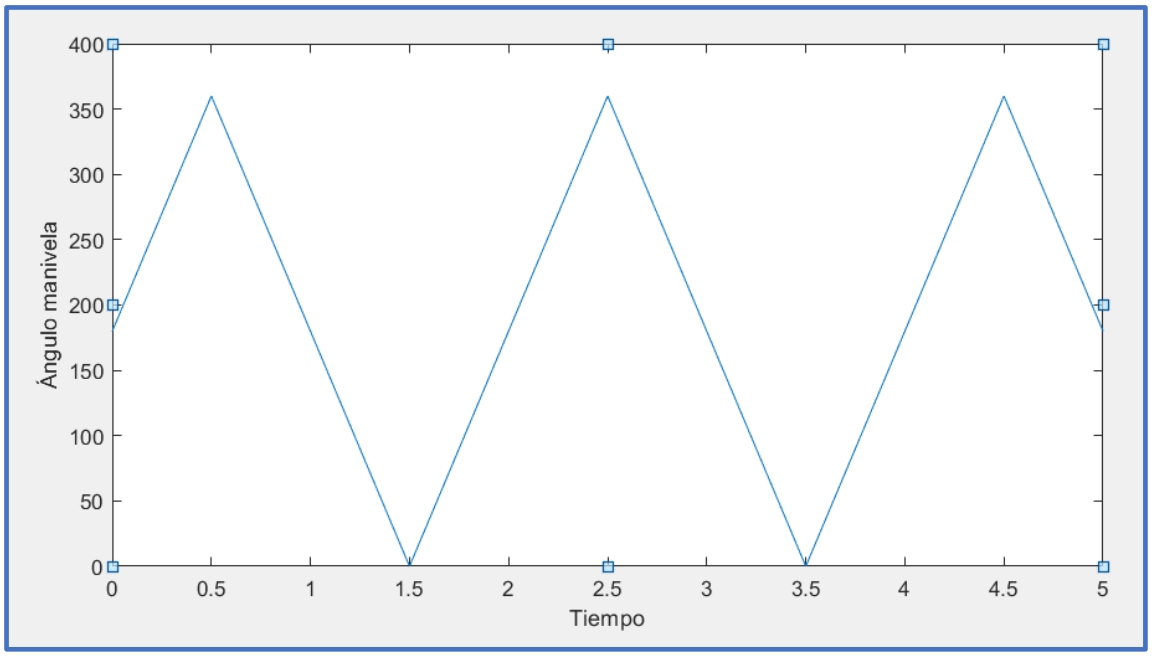
Post-procesador:

Para ejecutar el post-procesador simplemente escribimos "post" en la ventana de comandos de Matlab. Y procedemos a analizar la evolución de algunos parámetros:

Ángulo de giro de la manivela a lo largo del tiempo:

Para esto debemos ejecutar el post-procesador, escribiendo "post" en la ventana de comandos de Matlab, tras esto, teniendo en cuenta que los parámetros de Euler están en la matriz 'p', y que el parámetro $e_1 = \frac{cos\theta}{2}$, podemos hacer la conversión al ángulo en grados mediante:

$$\boldsymbol{plot(T, 2 * acos(p(:,1,2)) * 180/pi)}$$

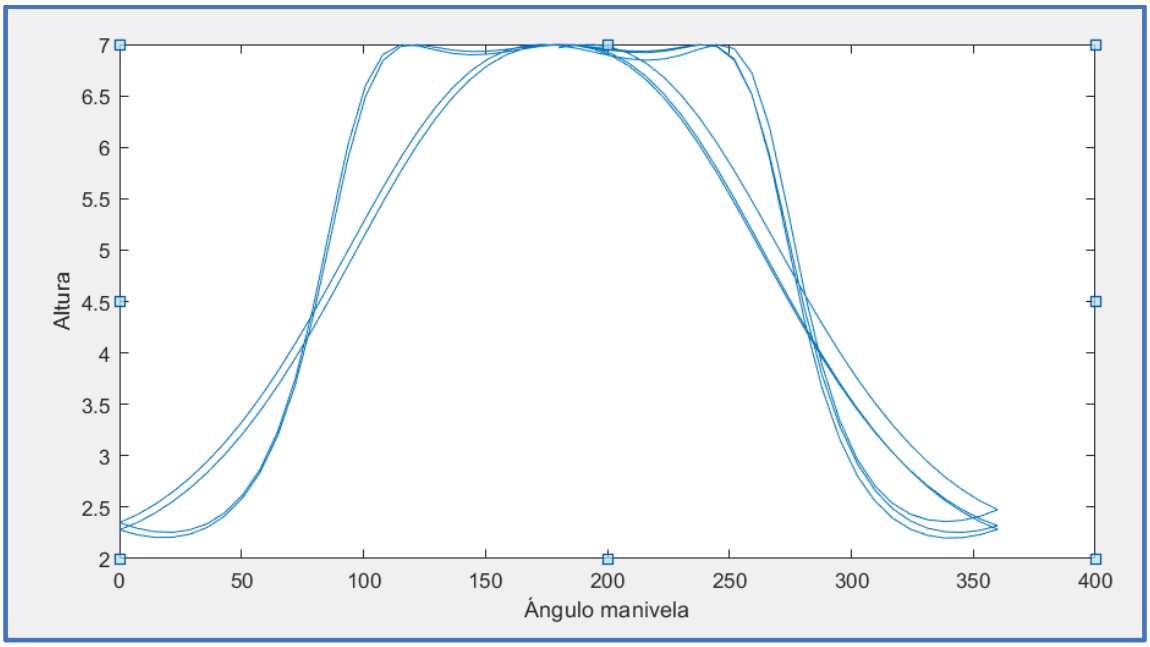


34. Ángulo manivela vs Tiempo.

Podemos observar que comienza a 180º tal y como lo describimos en inBodies.m. Y que tiene un periodo de revolución de 2 segundos.

Altura del punto 4 en función del ángulo de manivela:

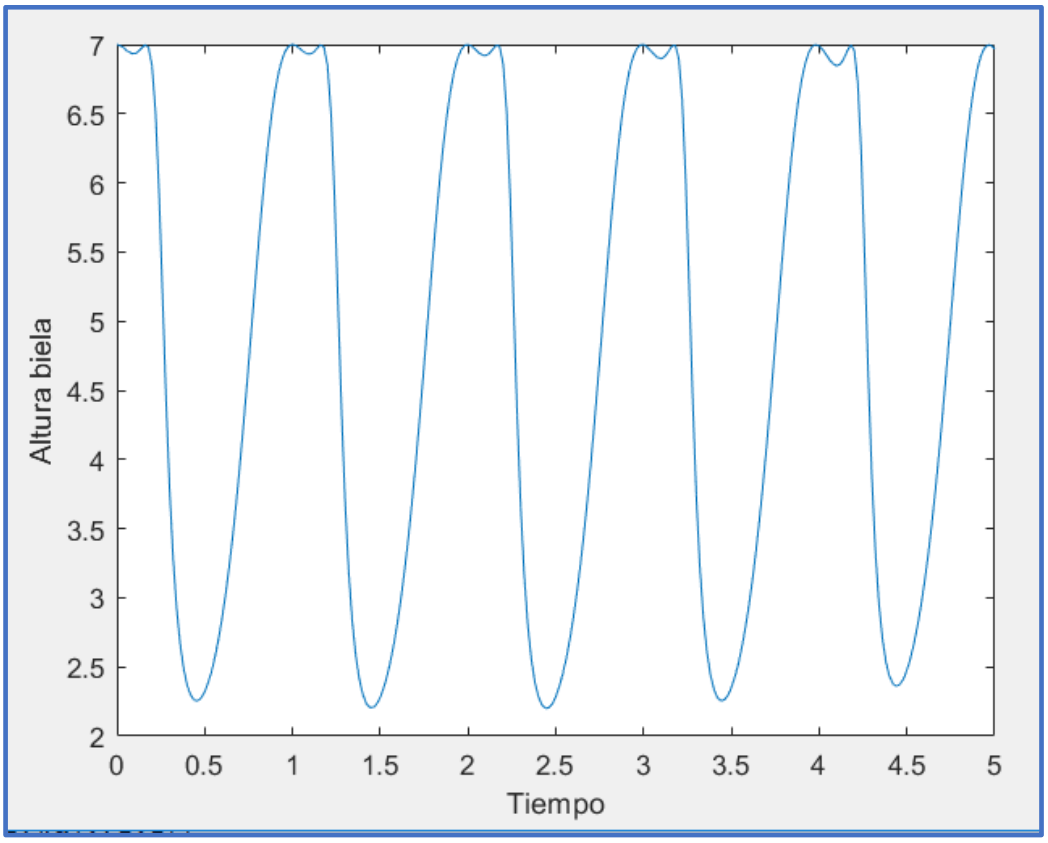
$$\boldsymbol{Plot(2 * acos(p(:,1,2)) * 180/pi, rP(:,5,3))}$$



35. Ángulo manivela vs Altura biela.

Posición de biela frente al tiempo:

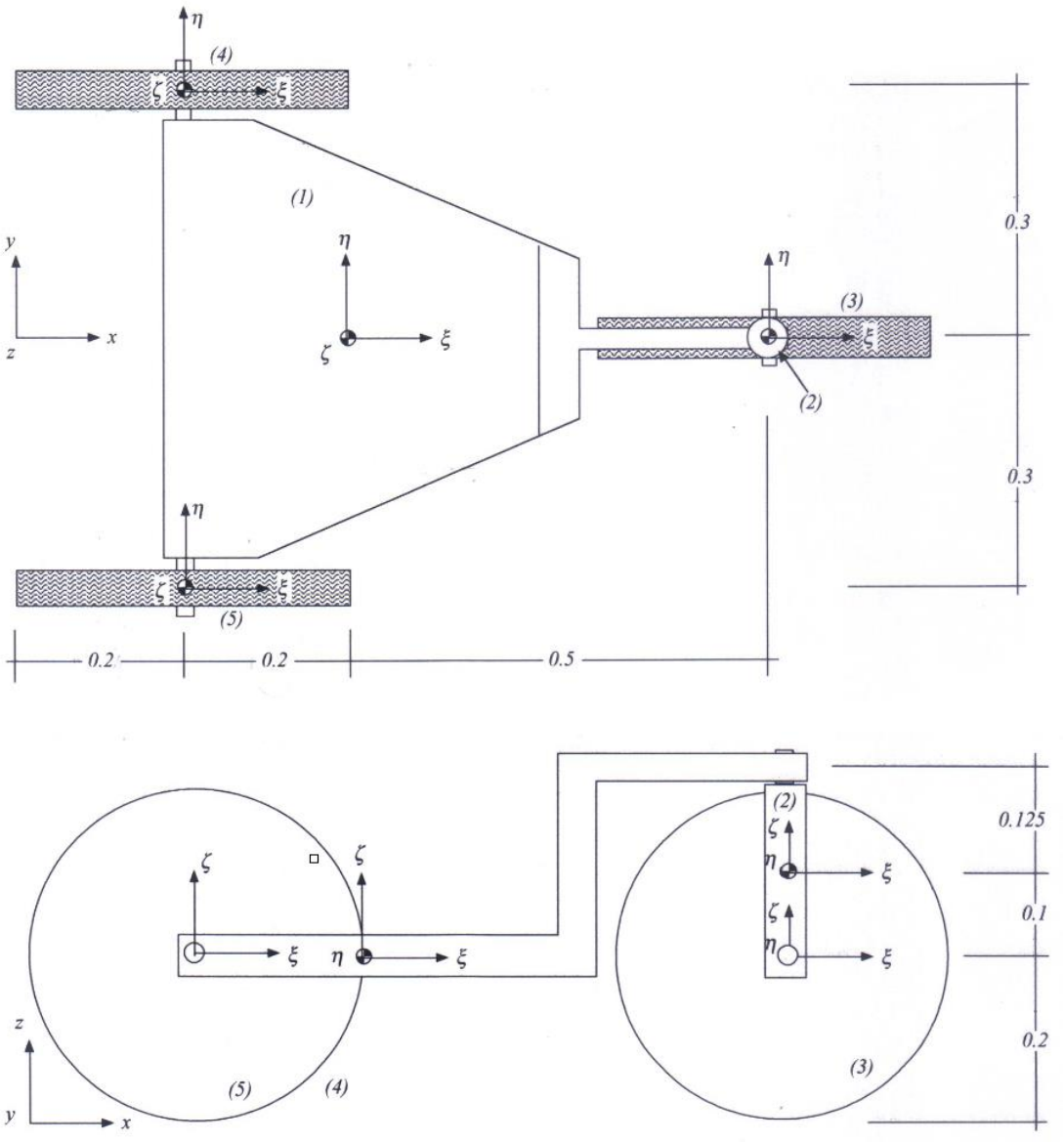
$$\boldsymbol{plot(T, rP(:, 5, 3))}$$



36. Posición biela vs Tiempo.

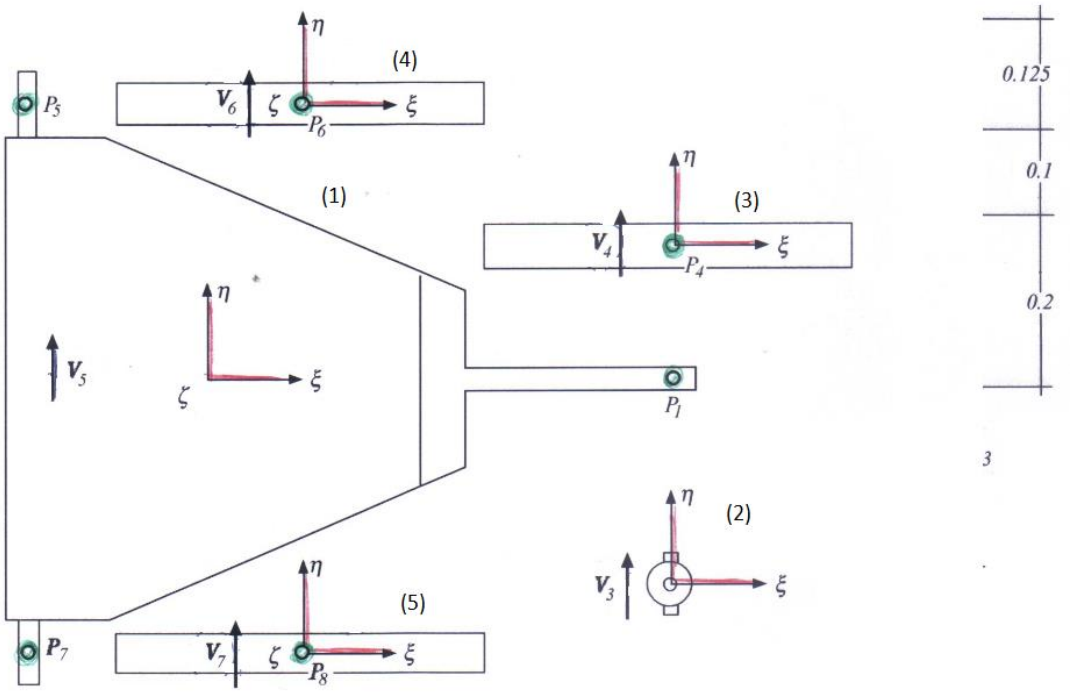
3.3 Modelo 3: Triciclo.

En este modelo se va a simular el movimiento de un triciclo que se compone de 5 cuerpos y 4 juntas de revolución. Los cuerpos se muestran en la ilustración. Un par de revolución une los cuerpos 1 y 2 confiriendo al triciclo el sistema de dirección. La rueda delantera está unida al cuerpo 2 mediante una junta de revolución, y las dos ruedas traseras al cuerpo 1 de la misma forma. El control de la dirección se hace mediante una junta de tipo "rel-rot", que aporta la función de 'tipo c' para definir la trayectoria. En la rueda tractora delantera se aplica un momento de 20 N.m para mover el triciclo.



Las masas y momentos de inercia de los cuerpos se indican en inBodies.m, se ponen valores representativos, ya que nos interesa la evolución cinemática.

Tal como se vio en el apartado para definir discos, es muy importante que el eje-η sea perpendicular al plano del disco. Una vez hemos colocado los ejes locales de los diferentes cuerpos, hacemos un esquema de los puntos y vectores que vamos a utilizar para el modelado:



Una vez numerado y hecho el esquema de los elementos más importantes podemos proceder al modelado en Matlab.

inBodies.m

En este modelo tenemos tres ruedas, estas deben ser definidas como discos, para observar el comportamiento cinemático del triciclo. Para ello, vamos a dar la posición de todos los cuerpos y los inputs estándar en inBodies.m y posteriormente se definirán los cuerpos 3,4 y 5 como discos en el archivo de restricciones inJoints.m. Los cuerpos quedan así definidos:

```
function inBodies
    include_global
    Body = Body_struct;

B1 = Body;     % cuerpo central
B1.mass = 50;
B1.Jp = [1  0  0
         0 10  0
         0  0  1];
B1.r = [0.4; 0; 0.2];
```

```
B2 = Body;     % dirección
B2.r = [0.9; 0; 0.3];
B2.mass = 0.5;
B2.Jp = [0.5 0 0
         0 0.5 0
         0 0 0.5];

B3 = Body;     % rueda delantera
B3.r = [0.9; 0; 0.2];
B3.mass = 1;
B3.Jp = [0.01 0 0
         0 0.02 0
         0 0 0.01];

B4 = Body;     % rueda trasera izquierda
B4.r = [0.2; 0.3; 0.2];
B4.mass = 1;
B4.Jp = [0.01 0 0
         0 0.02 0
         0 0 0.01];

B5 = Body;     % rueda trasera derecha
B5.r = [0.2;-0.3; 0.2];
B5.mass = 1;
B5.Jp = [0.01 0 0
         0 0.02 0
         0 0 0.01];

Bodies = [B1; B2; B3; B4; B5];
```

Nótese que, al estar todos los ejes locales paralelos a los ejes globales, todos los parámetros de Euler para los distintos cuerpos quedan Bi.p=[1;0;0;0], pero no es necesario definirlos ya que esos son los valores por defecto que asigna el programa. Solo es necesario definirlos cuando los ejes del cuerpo no son paralelos, es decir θ es distinto de 0.

inPoints.m

Definimos los puntos que participan en uniones, vectores, etc. El archivo queda así:

```
function inPoints
          include_global
       Point = Point_struct;
P1 = Point;
P1.Bindex = 1;
P1.sPp = [0.5; 0; 0.225];

P2 = Point;
P2.Bindex = 2;
P2.sPp = [0; 0; 0.125];

P3 = Point;
P3.Bindex = 2;
P3.sPp = [0; 0; -0.1];

P4 = Point;
P4.Bindex = 3;
P4.sPp = [0; 0; 0];

P5 = Point;
P5.Bindex = 1;
P5.sPp = [-0.2; 0.3; 0];

P6 = Point;
P6.Bindex = 4;
P6.sPp = [0; 0; 0];

P7 = Point;
P7.Bindex = 1;
P7.sPp = [-0.2;-0.3; 0];

P8 = Point;
P8.Bindex = 5;
P8.sPp = [0; 0; 0];

Points = [P1; P2; P3; P4; P5; P6; P7; P8];
```

inVectors1.m

Siguiendo nuestro esquema en el que están dibujados y numerados nuestros vectores, necesarios para definición de uniones de revolución, definición de la junta de dirección etc. Se implementa el archivo del siguiente modo:

```
function inVectors1
    include_global
    Vector = Vector1_struct;

V1 = Vector;
V1.Bindex = 1;
V1.sp = [0; 0; 1];

V2 = Vector;
V2.Bindex = 2;
V2.sp = [0; 0; 1];

V3 = Vector;
V3.Bindex = 2;
V3.sp = [0; 1; 0];

V4 = Vector;
V4.Bindex = 3;
V4.sp = [0; 1; 0];

V5 = Vector;
V5.Bindex = 1;
V5.sp = [0; 1; 0];

V6 = Vector;
V6.Bindex = 4;
V6.sp = [0; 1; 0];

V7 = Vector;
V7.Bindex = 5;
V7.sp = [0; 1; 0];

Vectors1 = [V1; V2; V3; V4;V5; V6; V7];
```

inVectors2.m

Este modelo no precisa ningún vector de tipo 2, de modo que se escribe el código mandando el vector Vectors2 vacío, o de lo contrario dará error tal y como se comentó.

```
function inVectors2
    include_global
    Vector = Vector2_struct;

Vectors2 = [];
```

inForces.m

Se definen las fuerzas y momentos que actúan en el sistema, en nuestro caso la fuerza de la gravedad y el par transmitido a la rueda delantera.

```
function inForces
    include_global
    Force = Force_struct;

S1 = Force;
S1.type = 'weight';    % incluimos el peso

S2 = Force;
S2.type = 'np';    % esta fuerza aplica un momento constante
S2.iBindex = 3;    %en el cuerpo 3
S2.np = [0; 20; 0];

Forces = [S1; S2];
```

inJoints.m

En este modelo, el archivo inJoints.m podemos dividirlo en dos secciones de código, en primer lugar, vamos a definir las juntas de revolución tal y como hemos hecho hasta ahora mediante los puntos de cuerpos que intervienen en el par y los dos vectores perpendiculares al giro definidos previamente. Los pares quedarían así definidos:

```
function inJoints
    include_global
    Joint = Joint_struct;

J1 = Joint;    % Junta de revolución entre cuerpos 1-2
J1.type = 'rev';
J1.iPindex = 1;
J1.jPindex = 2;
J1.iVindex_a = 1;
J1.jVindex_a = 2;
```

```
J2 = Joint;    % Junta de revolución entre cuerpos 2-3
J2.type = 'rev';
J2.iPindex = 3;
J2.jPindex = 4;
J2.iVindex_a = 3;
J2.jVindex_a = 4;

J3 = Joint;    % Junta de revolución entre cuerpos  1-4
J3.type = 'rev';
J3.iPindex = 5;
J3.jPindex = 6;
J3.iVindex_a = 5;
J3.jVindex_a = 6;

J4 = Joint;    % Junta de revolución entre cuerpos  1-5
J4.type = 'rev';
J4.iPindex = 7;
J4.jPindex = 8;
J4.iVindex_a = 5;
J4.jVindex_a = 7;
```

La segunda parte del archivo debe definir los “disk-joints”, recordemos que para definir los discos lo hacemos en este archivo ya que impone a las ruedas restricciones de desplazamientos.

Par definir los discos hay que ser especialmente cuidadoso y tener en cuenta cuantas restricciones impone cada tipo de disco. En caso de que no se eligiesen correctamente el tipo de ‘disk joints’ para el conjunto de ruedas de un sistema, el sistema de ecuaciones a resolver será incompatible, dado que el deslizamiento en el plano de algunas ruedas es necesario para variaciones de dirección de ciertos móviles. En esta situación, al ejecutar el programa en MUBODYNA, éste entrará en un bucle infinito, intentando resolver un sistema de ecuaciones incompatible. El sistema da la siguiente respuesta:

“Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RECOND=3.12332e-18”

En este modelo, es importante que la rueda tractora delantera sea la “disk xyz”, ya que, si por ejemplo se restringen los desplazamientos de las dos ruedas de atrás, el movimiento de giro será incompatible. Del mismo modo una de ellas deberá ser “disk nz” y la otra “disk z”, haciendo que una de las ruedas tenga tracción lateral y la otra deslizando en el plano libremente, ya que al ser paralelas y rígidas el movimiento sería imposible sin deslizamiento lateral de una de ellas.

La definición de “disk-joints” se muestra a continuación:

```
J5 = Joint;     % front wheel
J5.type = 'disk';
J5.disk = 'xyz';
J5.iBindex = 3;
J5.R = 0.2;

J6 = Joint;     % left-rear wheel
J6.type = 'disk';
J6.disk = 'nz';
J6.iBindex = 4;
J6.R = 0.2;

J7 = Joint;     % right-rear wheel
J7.type = 'disk';
J7.disk = 'z';
J7.iBindex = 5;
J7.R = 0.2;

J8 = Joint;     % steeting driver
J8.type = 'rel-rot';
J8.iBindex = 1;
J8.jBindex = 2;
J8.jVindex_a = 2;
J8.iFunct  = 1;

Joints = [J1; J2; J3; J4; J5; J6; J7; J8];
```

Además, la junta J8 se implementa para definir el giro en la dirección, que será dirigido mediante la función C1 descrita a continuación.

inJoints.m

Se define la función directora del volante, mediante una función de 'tipo c' que describe el ángulo que gira el volante. Para definir esta función es necesario dar los valores de comienzo y final (start/end) y los valores máximos de (x,f(x)). Según esas variables, el programa computa los coeficientes (c1,c2,c3,c4,c5) para formar la función de tipo c.

```
function inFuncts
    include_global
    Funct = Funct_struct;

C1 = Funct;
C1.type = 'c';
C1.x_start = 0;     % default
C1.x_max = 0.6;
C1.x_end   = 1.2;
C1.f_start = 0;     % default
C1.f_max   = pi/6;
C1.f_end   = 0;

Functs = [C1];
```

inAnimate.m

En este modelo vamos a incluir los puntos de animación P9 y P10, que simulan el manillar del triciclo para poder observar bien como se realiza el giro a lo largo del tiempo, además será necesario definir los parámetros típicos en que se muestra la animación.

```
function inAnimate
 include_global

    Point = Point_struct;
    P9=Point;
    P9.Bindex=2;            %Puntos del manillar.
    P9.sPp=[0;0.2;0.25];

    P10= Point;
    P10.Bindex=2;
    P10.sPp=[0;-0.2;0.25];

    Points_anim = [P9;P10]; %Vector output

xmin = -3;  xmax =  6;
ymin = -3;  ymax =  3;
zmin = -1;  zmax =  1;

%Parámetros de rotación del cubo de visualización.
AZ = 130;   %130 grados desde elfrente
EL = 20;    %elevación 20°
```

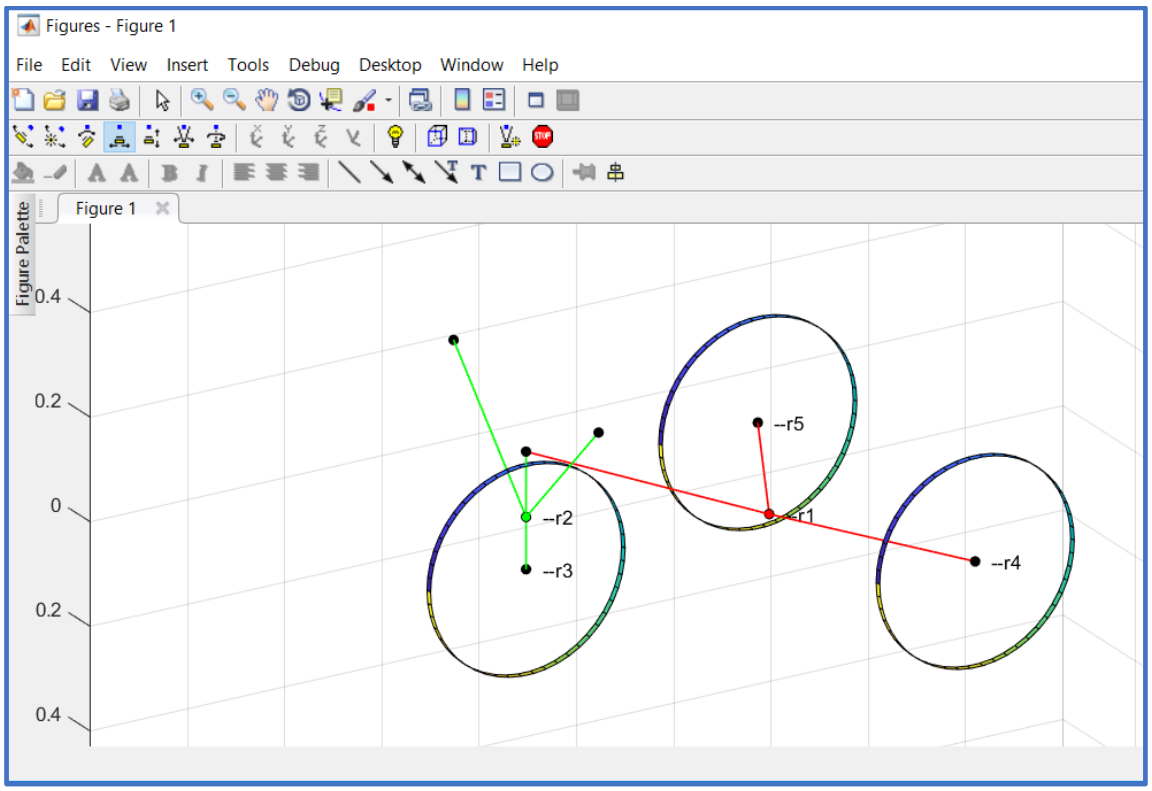
En función de el tiempo elegido para simulación, el triciclo podrá salirse del cubo tridimensional de animación. Por lo que el usuario deberá jugar con los valores máximos y mínimos de los ejes (x,y,z), con el objetivo de ver los giros correctamente.

inSolver.

En esta ocasión, no vamos a corregir las condiciones iniciales, el tiempo de simulación deseado es de 4 segundos, con intervalos de tiempo de informe (reporting time steps) de 0.02 segundos. El método de resolución será el estándar.

```
function inSolver
    include_global
%--------------------------------------------------------------------------
% This m-file contains all the necessary ingredients about the resolution
% of the equations of motion, such as integrator used, final time, etc.
%--------------------------------------------------------------------------
    correct_ic = 'n';        % sin corregir condiciones iniciales
    t_final = 4;             % tiempo final de simulación
    dt = 0.02;               % reporting time step
    method = 'standard';     % método estándar de resolución
%--------------------------------------------------------------------------
```

Con los archivos.m implementados vamos a Matlab y ejecutamos Mubodyna. Una vez el programa resuelve las ecuaciones podemos hacer una animación para confirmar el correcto montaje y observar la simulación del movimiento ejecutando el comando "animate".



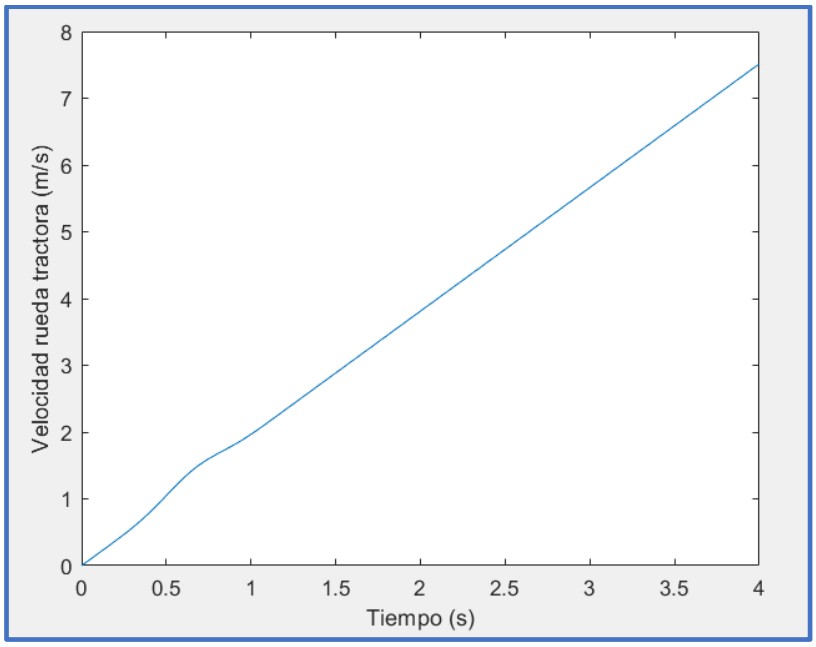
37. Simulación cinemática de triciclo.

Post-Procesador.

Ejecutando el post-procesador como vimos anteriormente, vamos a analizar algunos parámetros del triciclo:

Evolución de la velocidad de la rueda tractora: como el post-procesador nos permite sacar la velocidad V_x y V_y de la rueda tractora, podemos sacar el módulo de la velocidad mediante:

plot(T,(rd(:,3,1).^2+rd(:,3,2).^2).^0.5);



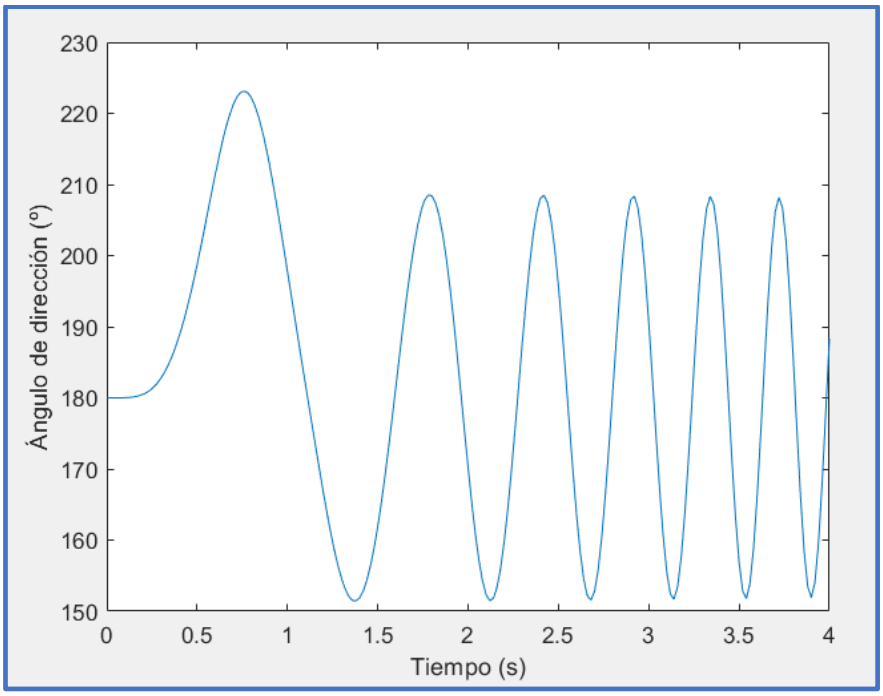
38. Velocidad rueda tractora vs Tiempo.

Observamos una evolución lineal de la velocidad, lo cual tiene sentido ya que el triciclo se mueve gracias a un par constante aplicado en la rueda tractora.

Ángulo de dirección: Lo calculamos mediante el ángulo del manillar (cuerpo 2), para ello graficamos el ángulo frente al tiempo, convirtiendo los parámetros de Euler:

plot(T,2*acos(p(:,2,2))*180/pi)

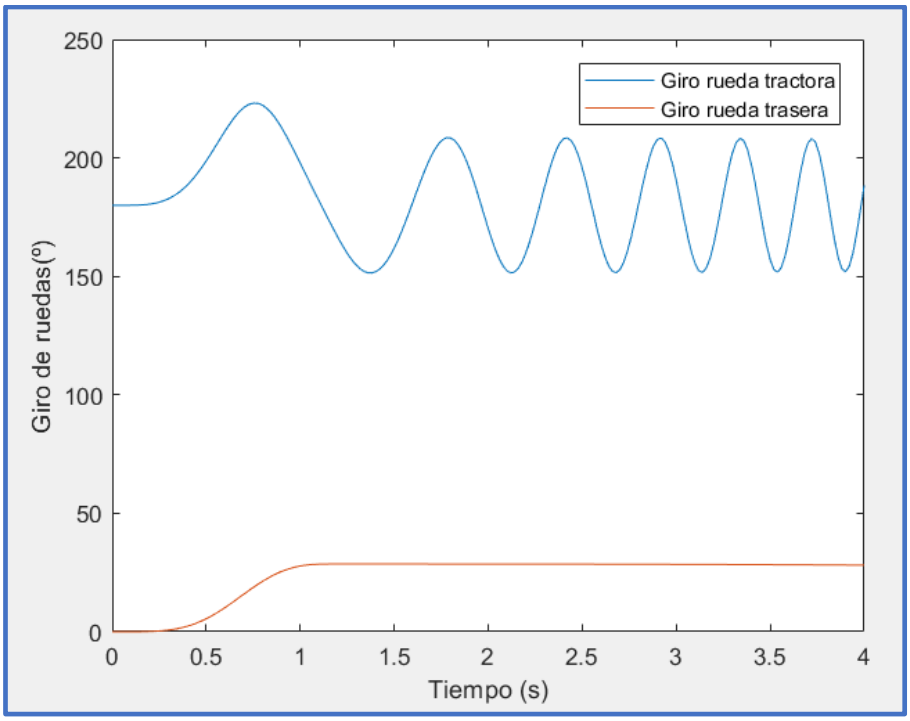
En la figura 39, podemos observar los distintos giros que hace el manillar frente al tiempo. Estos están dirigidos mediante la función de tipo ‘c’ implementada en inFunctions.m.



39. Giro manillar vs Tiempo.

Ángulo de las ruedas traseras vs Ángulo de tractora.

Podemos ver como se transmite el giro a las ruedas traseras mediante:

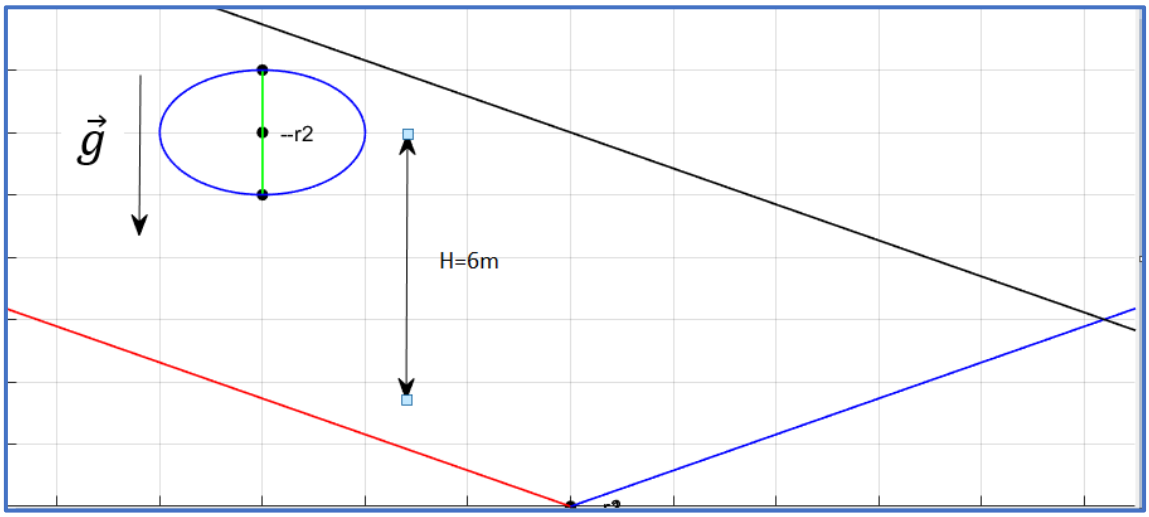


40. Relación de giro, rueda trasera y motriz.

Se observa que, para el primer giro, la rueda trasera hace un giro a izquierdas, pero en los pequeños giros siguientes apenas tiene efecto en la rueda trasera.

3.4 Modelo: Impacto entre esfera y suelo.

En el siguiente modelo se va a mostrar el modelado y simulación de una esfera cayendo libremente desde una altura, hasta impactar contra el suelo. Además, dentro de este problema se incluyen dos variaciones, una en la que el problema de choque es elástico y otro en el que es inelástico, para finalizar incluiremos fenómenos de rozamiento entre la esfera y el suelo, para ver cómo afecta al movimiento.



El sistema a resolver consiste en una esfera de radio R=1m, que cae sobre un plano desde una altura h=6m. La velocidad inicial es nula. Ambos planos forman 30º con la horizontal. Las propiedades de la esfera y planos son:

Esfera: $E = 290x10^9 \frac{N}{m^2}$, $v = 0.3$, *m=5kg.*

Planos: $E = 290x10^9 \frac{N}{m^2}$, $v = 0.3$, *m=100kg (aunque son planos, en el programa hay que definirlos como cuerpos y requieren una masa).*

inBodies.m

Definimos la esfera y los planos, con su posición inicial y propiedades. Dado que hay que definir los planos como cuerpos también deberemos darles una masa y una matriz de inercia. El archivo quedaría de la siguiente forma:

```
function inBodies
    include_global
    Body = Body_struct;


B1 = Body;                    % Plano asociado a B1!!
B1.r = [0; 0; 0];             %Punto perteneciente al plano
B1.mass = 100;
B1.Jp = [0.1    0          0     % el vector normal será definido
         0      0.1        0   % en inForces al incluir la fuerza sph-pln
         0      0          0.1];
```

```
B1.poisson = 0.3;
B1.modulus = 290e9;

B2 = Body;              % Definición de la esfera
B2.r = [-3; 0; 6];
B2.mass = 5;
B2.Jp = [2        0              0
         0              2           0
         0              0               2];
B2.poisson = 0.3;
B2.modulus = 290e9;

B3 = Body;                    % Plano asociado a B3!!
B3.r = [0; 0; 0];             %Punto perteneciente al plano
B3.mass = 100;
B3.Jp = [0.1    0         0    % el vector normal será definido
         0      0.1       0   % en inForces al incluir la fuerza sph-pln
         0      0         0.1];
B3.poisson = 0.3;
B3.modulus = 290e9;

Bodies = [B1; B2;B3];
```

inPoints.m

Definimos el punto 1, centro de la esfera, que será utilizado para definir las fuerzas de contacto, ya que el programa analiza el contacto entre cuerpos a partir de la distancia entre centro de la esfera y el plano, cuando la distancia es menor que el radio, aparecerán las fuerzas de impacto. Además, vamos a incluir puntos para mejorar la visualización, si bien dijimos que los puntos incluidos por este motivo deben usarse en inAnimate.m, como vamos a usarlos para definir vectores tipo2 que faciliten el análisis, debemos definirlos en el presente archivo.

```
function inPoints
    include_global
    Point = Point_struct;

P1 = Point;
P1.Bindex = 2;
P1.sPp = [0; 0; 0];

P2 = Point;
P2.Bindex = 2;
P2.sPp = [0; 0; 1];
```

```
P3 = Point;
P3.Bindex = 2;
P3.sPp = [0; 0; -1];

P4=Point;                   %Para dibujar un vector de plano150° y
P4.Bindex=1;                %mejorar visualización
P4.sPp=[-10*3^(0.5);0;10];

P5=Point;
P5.Bindex=1;
P5.sPp=[0;0;-10];

P6=Point;                   %para dibujar el plano 3 a 30°
P6.Bindex=3;
P6.sPp=[10*3^(0.5);0; 10];

P7=Point;
P7.Bindex=3;
P7.sPp=[0;0;0];

Points = [P1; P2; P3;P4;P5;P6;P7];
```

inVectors1.m

```
function inVectors1
    include_global
    Vector = Vector1_struct;

Vectors1 = [];
```

inVectors2.m

```
function inVectors2
    include_global
    Vector = Vector2_struct;

    V1=Vector;
    V1.iPindex=4; %vector para ver el plano a 150°
    V1.jPindex=5;

    V2=Vector;
    V2.iPindex = 6;  %Dibujo un vector para ver el plano30°
    V2.jPindex =7;

Vectors2 = [V1;V2];
```

Dibujamos esos dos vectores, de modo que en la animación podamos observar mejor los impactos con la esfera.

inForces.m

Las fuerzas que experimenta la esfera son tres; la gravedad, la fuerza de impacto esfera-plano1, y la fuerza entre esfera-plano2. La gravedad se define como siempre:

```
function inForces
    include_global
    Force = Force_struct;

S1 = Force;
S1.type = 'weight';    % include the weight
S1.gravity = 9.81;
```

Ahora para definir las fuerzas de contacto, vamos a utilizar el modelo de Lankarani para modelar la fuerza elástica. Para esto en la fuerza deberemos introducir los inputs indicados en el apartado de fuerzas de impacto. Obsérvese, que en estas fuerzas incluimos el radio de la esfera y el vector normal que define el plano. Además, se dan los coeficientes que participan en el modelo de impacto mencionado, n=1.5 y sin fricción (Si.friction='false') y coeficiente de restitución 1, por ser impacto elástico.

$$Lankarani\ el\acute{a}stico: \quad F_N = K\delta^n$$

La fuerza de choque entre esfera y plano 1, se define como sigue:

```
S2 = Force;
S2.type = 'sph_pln';
S2.iPindex = 1;
S2.iBindex = 2;
S2.jBindex = 1;
S2.plane_normal_vector = [sqrt(1)/2;0;sqrt(3)/2]; %plano inclinado 150°
S2.sphere_radius = 1;
S2.restitution = 1;
S2.force_model = 'lankarani';
S2.n_exponent =  1.5;
S2.friction = false;
S2.fric_model = 'linear';
S2.mu = 0.5;
S2.mus = 0.6;
%Variables de fricción(mu, mus, vs, z, sigma0, sigma1, sdw) se
%utiliza los valores por defecto
```

Para definir la fuerza entres esfera (cuerpo2), y el plano 3:

```
S3 = Force;
S3.type = 'sph_pln';
S3.iPindex = 1;
S3.iBindex = 2;
S3.jBindex = 3;
S3.plane_normal_vector = [-sqrt(1)/2;0;sqrt(3)/2]; %plano inclinado 30°
S3.sphere_radius = 1;
S3.restitution = 1;
S3.force_model = 'lankarani';
S3.n_exponent =  1.5;
S3.friction = false;
S3.fric_model = 'linear';
S3.mu = 0.5;
S3.mus = 0.6;
Forces = [S1;S2;S3];
```

inFunctions.m

```
function inFuncts
    include_global
    Funct = Funct_struct;

Functs = [];
```

inJoints.m

Cuando se definen planos distintos del suelo, es necesario fijarlos de modo que se reconozcan como un cuerpo inamovible. Fijamos los planos 1 y 3:

```
function inJoints
    include_global
    Joint = Joint_struct;

J1 = Joint;
J1.type = 'fix';
J1.iBindex = 1;

J2 = Joint;
J2.type = 'fix';
J2.iBindex =3;

Joints = [J1;J2];
```

inAnimate.m

Dado que hay un código implementado para dibujar la esfera dado su centro y radio siempre que existan fuerzas "tipo sph_pln", no vamos a añadir puntos extra para visualización. El archivo queda así:

```
function inAnimate
    include_global
    Point = Point_struct;

Points_anim = [];

xmin = -5.5;  xmax = 5.5;
ymin = -7.5;  ymax =  20;
zmin = 0;   zmax =  7;

AZ = 0;     %Angulo para mejor visualización del fenómeno
EL = 0;
```

Para verlo mejor hacemos que la simulación enfoque el cubo de animación de forma frontal al plano 'zx'.

inSolver.m

En este modelo vamos a hacer corrección de condiciones iniciales. Elegimos un tiempo de animación de 10 segundos, para observar que los rebotes son infinitos al

imponer rebotes perfectamente elásticos. Los "reporting time-steps" serán de 0.01 segundos y el método de resolución el estándar.

```
function inSolver
    include_global
%-------------------------------------------------------------------------
% This m-file contains all the necessary ingredients about the resolution
% of the equations of motion, such as integrator used, final time, etc.
%-------------------------------------------------------------------------
    correct_ic = 'y';         % Corrección de condiciones iniciales.
    t_final = 20;             % tiempo final de simulación
    dt = 0.01;                % reporting time step
    method = 'standard';      % method used to solve the equations of motion
```

Tras finalizar este archivo, el modelo estaría terminado y podemos ejecutarlo en Mubodyna.

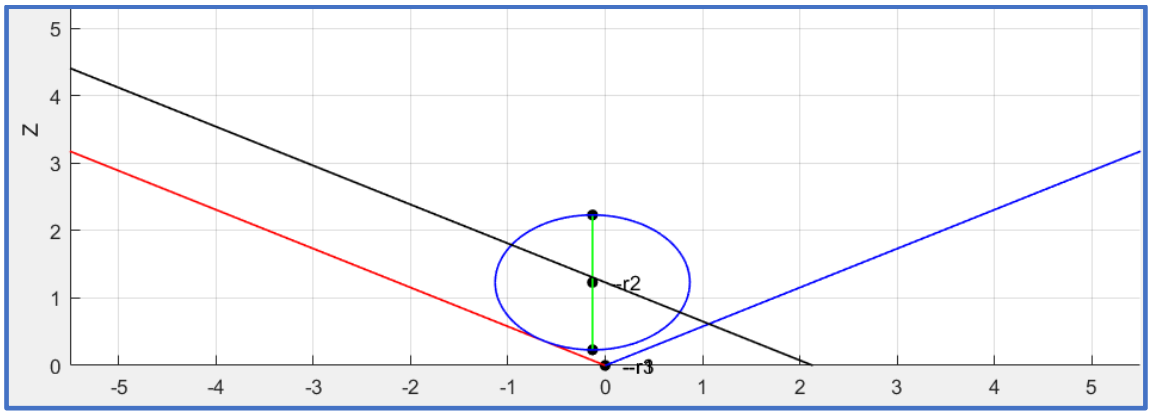
Dado que el coeficiente de restitución se definió como igual a 1, la energía cinética se conserva continuamente. Si hacemos la animación observaremos una pelota rebotando infinitamente entre los dos planos definidos, lo que indica que los rebotes son perfectamente elásticos.

3.4.1 Variaciones al problema: Choque inelástico.

Para simular choques inelásticos se cambió el coeficiente de restitución a un valor de 0.8, lo que se traduce en una pérdida de energía cinética, debida a las deformaciones. Para simular este fenómeno simplemente debemos cambiar en la definición de fuerzas de choque entre esfera y planos, el coeficiente asignado de restitución:

```
S2 = Force;
S2.type = 'sph_pln';
S2.iPindex = 1;
S2.iBindex = 2;
S2.jBindex = 1;
S2.plane_normal_vector = [sqrt(1)/2;0;sqrt(3)/2]; %
S2.sphere_radius = 1;
S2.restitution = 0.8;
S2.force_model = 'lankarani';
S2.n_exponent =  1.5;
S2.friction = false;
S2.fric_model = 'linear';
S2.mu = 0.5;
S2.mus = 0.6;
%Variables de fricción(mu, mus, vs, z, sigma0, sigm
%utiliza los valores por defecto
S3 = Force;
S3.type = 'sph_pln';
S3.iPindex = 1;
S3.iBindex = 2;
S3.jBindex = 3;
S3.plane_normal_vector = [-sqrt(1)/2;0;sqrt(3)/2];
S3.sphere_radius = 1;
S3.restitution = 0.8;
S3.force_model = 'lankarani';
S3.n_exponent =  1.5;
```

Ahora si hacemos la animación veremos que acaba cayendo al punto de convergencia de los dos planos, gracias al efecto de la gravedad:



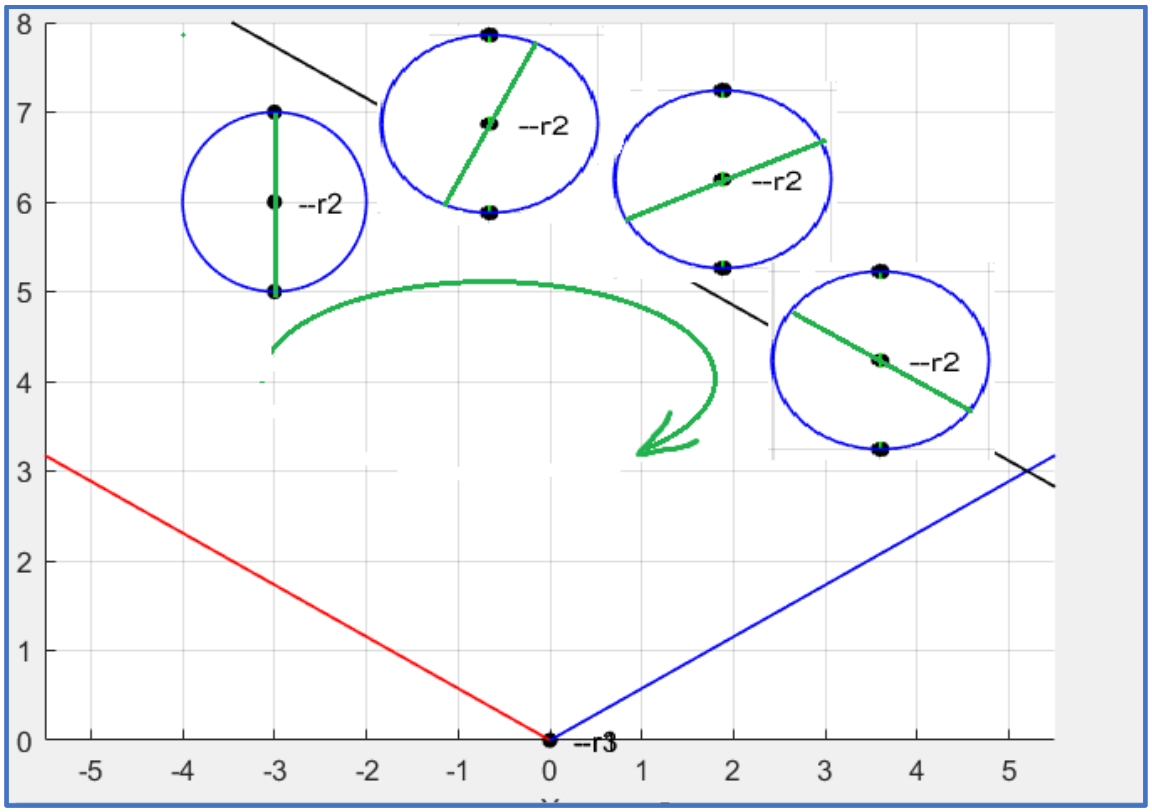
41. Posición final de la esfera en problema inelástico.

3.4.2 Variante2: rozamiento.

Otra posibilidad que ofrecen estas fuerzas es la de incluir rozamiento entre las superficies y la esfera. Para ello vamos a inForces.m y cambiamos el valor de fricción a 'true', de la siguiente manera:

```
S2 = Force;
S2.type = 'sph_pln';
S2.iPindex = 1;
S2.iBindex = 2;
S2.jBindex = 1;
S2.plane_normal_vector = [sqrt(1)/2;0;sqrt(3)/2];
S2.sphere_radius = 1;
S2.restitution = 0.1;
S2.force_model = 'lankarani';
S2.n_exponent =  1.5;
S2.friction = true;
S2.fric_model = 'linear';
S2.mu = 0.5;
S2.mus = 0.6;
```

Para comprobar el rozamiento sería una animación, en esta, observaríamos como la esfera rotaría gracias a las fuerzas tangenciales en el choque. Otra forma, sería analizar en el post-procesador la posición de los puntos de la esfera o los ángulos que adopta:



42. Evolución de modelo con fricción.

Post-procesador.

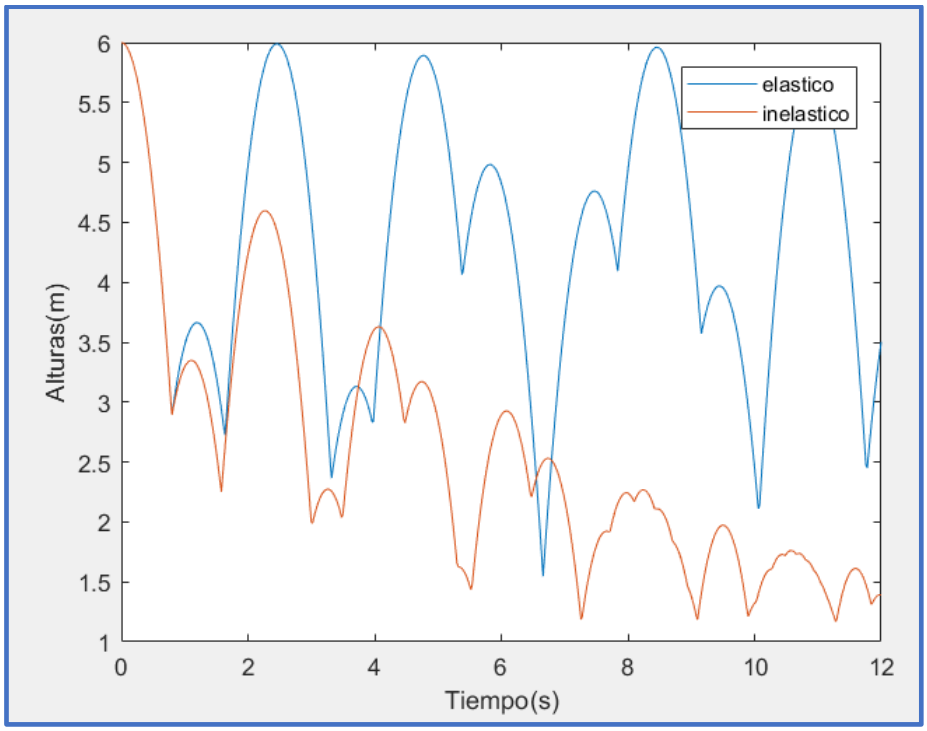
Vamos a comparar resultados entre el problema elástico y el inelástico, viendo analíticamente la diferencia entre evolución de alturas durante 12 segundos de animación, y la diferencia entre velocidades. Además, observaremos la rotación propia que adquiere la esfera cuando se introducen fuerzas de rozamiento tangenciales.

Para hacer esto primero vamos a simular el modelo elástico, y tras activar el post-procesador, en los vectores '**veloelast**' y '**alturaelast**', vamos a guardar las diferentes matrices solución de velocidad y altura mediante los siguientes comandos:

veloelast=((rd(:,2,2).^2)+rd(:,2,3).^2).^0.5;

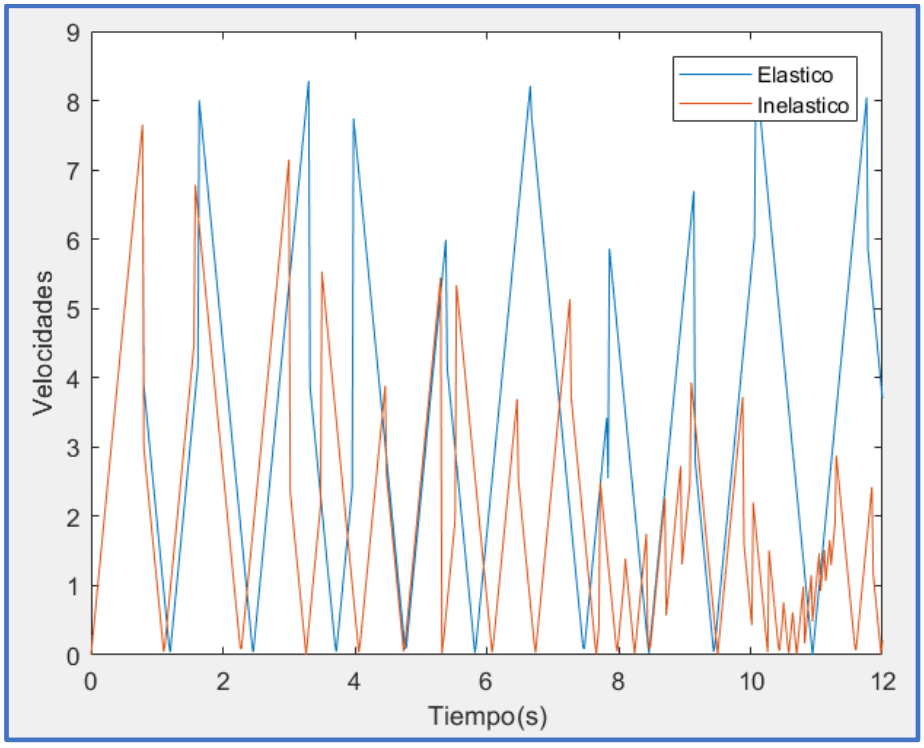
alturaelast=r(:,2,3);

Repetimos el proceso para el problema inelástico, almacenando los resultados en '**veloinelast**' y '**alturainelast**', y podremos realizar la comparación de resultados.



43. Alturas elástico/inelástico.

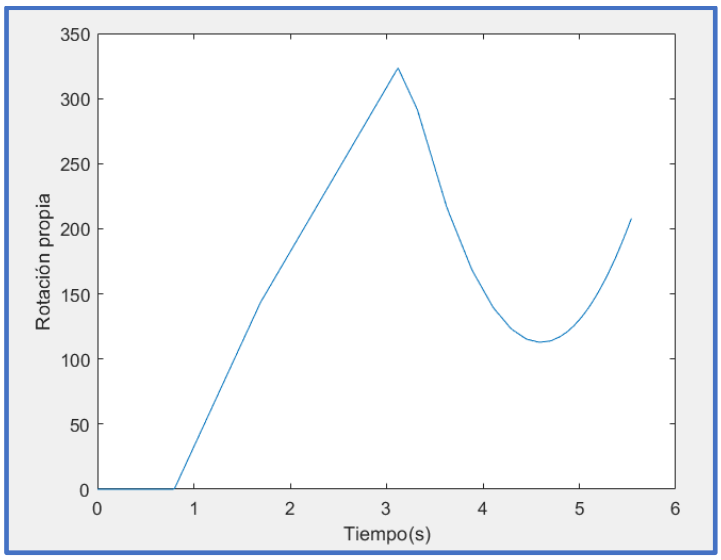
Se observa que la altura máxima del choque elástico se mantiene en 6 metros, altura desde donde se dejó caer, esto quiere decir que se conserva la energía cinética en los rebotes, por tanto, es lógico y concuerda con la definición de nuestro problema. Por otro lado, la energía cinética del problema inelástico va decreciendo progresivamente, ya que hay energía disipada en cada choque.



44. Velocidades elástico/inelástico.

Por último, vamos a comprobar analíticamente el modelo de fricción, en el que la pelota adquiere una rotación propia debido a las fuerzas tangenciales de rozamiento en el impacto. Para esto, simulamos el problema inelástico con rozamiento y ejecutamos el post-procesador. Para observar si tiene rotación propia, utilizaremos la evolución de los parámetros de Euler en el tiempo de simulación:

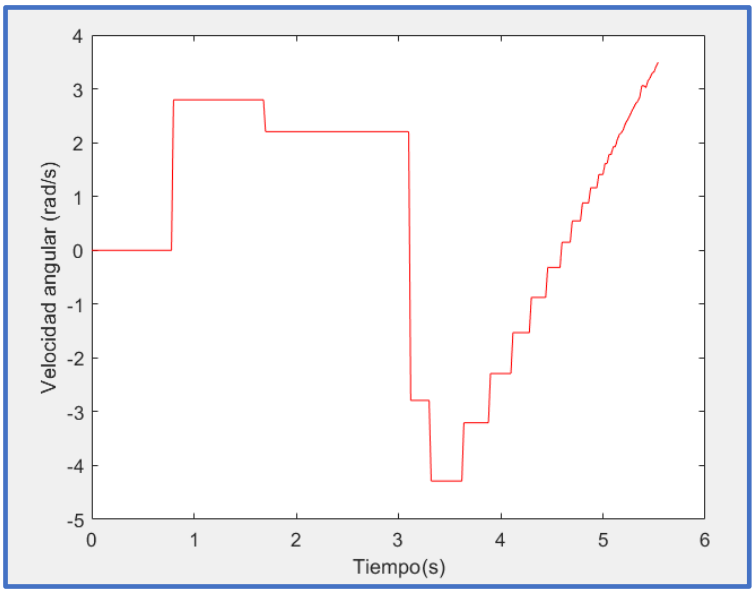
plot(T,(2*(acos(p(:,2,1)))*180/pi));



45. Rotación propia vs tiempo. Modelo con fricción.

En la gráfica podemos observar la evolución de la rotación de la esfera para los seis primeros segundos. Inicialmente la pelota no gira, hasta que recibe el primer choque contra el plano 1 para t=0.76s. En ese instante aparecen fuerzas tangenciales de rozamiento que hacen girar la esfera a la vez que se eleva.

También podemos observar los cambios de velocidad angular, para analizar el movimiento:



46. Velocidad angular. Modelo con fricción.

En la gráfica de velocidad angular, podemos observar perfectamente como la esfera comienza a rotar sobre su eje-ξ a partir de t=0.76s, momento en el que sufre el primer choque contra el suelo. Después para el segundo y tercer choque dado que choca en el plano 3, la fuerza de rozamiento genera un momento en la dirección opuesta al giro, por lo que hace decrecer la velocidad angular y posteriormente girar en el otro sentido.

Después vemos, que los pequeños rebotes de menor altura cada vez, aceleran angularmente la esfera, creando un aumento lineal de la velocidad angular, en el momento en que la pelota ya no rebota y acaba rodando sobre el plano, traduciéndose en un momento angular constante.

4. CONCLUSIONES:

Llegando al final del proyecto, podemos decir que hemos cumplido los objetivos que nos propusimos. Se han adquirido los conocimientos necesarios para modelar y desarrollar sistemas mecánicos de diferente índole exitosamente. Se han implementado las diferentes herramientas del programa y analizado la respuesta de este en diferentes situaciones.

A su vez, hemos creado un documento que servirá de manual a futuros estudiantes e investigadores que deseen continuar con el desarrollo de Mubodyna, o bien utilizarlo para sus intereses académicos.

Hemos comprobado el buen funcionamiento, y su ajuste a los fenómenos físicos de cinemática y dinámica, mediante la realización de varios modelos y el análisis de resultados. Podemos concluir así, que el programa se ajusta bien a la realidad y nos permite obtener buenos resultados, con tiempos computacionales no excesivos y buena precisión.

En resumen:

- ✓ Hemos comprendido y explicado los fundamentos físicos necesarios para el modelado de problemas dinámicos a ordenador.
- ✓ Se ha creado un manual para estudiantes y futuros investigadores.
- ✓ Hemos desarrollado diferentes modelos, analizando las diferentes posibilidades del programa.
- ✓ Se han analizado los resultados, creado gráficas y estudiado los resultados. Comprobando su ajuste a la realidad.
- ✓ Se han hecho diferentes animaciones que simulan los sistemas mecánicos presentados en este documento.

5. PRESUPUESTO DEL PROYECTO:

Ahora vamos a presentar un presupuesto, en el que se detallarán los diferentes gastos estimados en el desarrollo del presente trabajo. En las diferentes tablas se muestran los gastos según la categoría a la que pertenecen.

Recursos humanos:

Apellidos, Nombre	Categoría	Tiempo dedicado (h)	Sueldo (€/h)	Coste total (€)
Corral Abad, Eduardo	Dr. Ingeniero	100	25	2.500
Villar Alegría, Sergio	Ingeniero Junior.	480	10	4.800
TOTAL				7.300€

Tabla 2. Desglose presupuesto de personal cualificado.

Hardware:

Hardware	Unidades	Coste unitario (€)	Coste total (€)	Vida útil (años)	Amortización anual (%)	Coste anual amortización (€)
Ordenador portátil	1	900	900	4	25	225
Ordenador de torre	1	800	800	5	20	160
Pantalla ordenador	1	90	90	5	20	16
Ratón	1	25	25	8	12.5	3
Teclado	1	35	35	5	20	7
Disco de almacenamiento	1	30	30	3	33.33	10
TOTAL						421€

Tabla 3. Desglose de equipo utilizado.

Software:

Descripción	Unidades	Coste unitario (€)	Coste total (€)	Vida útil (años)	Amortización anual (%)	Coste anual amortización (€)
Matlab (Licencia académica)	1	180	180	1	100	180
TOTAL						180€

Tabla 4. Desglose del presupuesto por costes de software.

Presupuesto total:

PRESUPUESTO	
Concepto	Coste (€)
Personal	7300
Hardware	421
Software	180
TOTAL	7901 €

Tabla 5. Presupuesto general.

1. BIBLIOGRAFÍA.

[1] Javier García de Jalón, Eduardo Bayo, "*Kinematic and Dynamic Simulation of Multibody Systems*"-The Real Time Challenge-, Springer 2009.

[2] SHABANA, A.A., "*Computational Dynamics*". Wiley, 2001.

[3] SHABANA, A.A., "*Dynamics of Multibody Systems*" 4th ed. Cambridge University 2013.

[4] Parviz E. Nikravesh, "*Computer-Aided Analysis of Mechanism Systems*", Prentice Hall, 1988.

[5] Nikravesh PE "*Initial condition correction in multibody dynamics*". CRC, press,2007 London.

[6] Paulo Flores, "*Concept and Formulation for Spatial Multibody Dynamics*", Springer 2015.

[7] Paulo Flores, Hamid M. Lankarani, "*Contact Force Models for multibody Dynamics*". Springer, 2016.

[8] Nikravesh PE "*Planar multibody dynamics: formulation, programming and applications*" CRC, press, 2008 London.