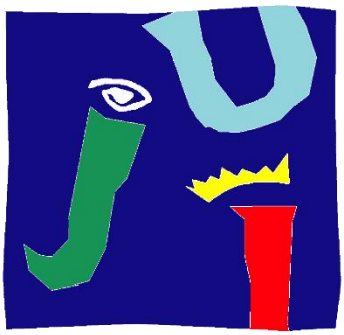
Escuela Superior de Tecnología y Ciencias Experimentales

Departamento de Matemáticas
Trabajo de Fin de Máster



UNIVERSITAT JAUME·I

RESOLUCIÓN NUMÉRICA DE ECUACIONES DIFERENCIALES EN WOLFRAM MATHEMATICA

Trabajo de fin de máster de: Jaime Rodrigo Segarra Escandón

Supervisada por: Fernando Casas Pérez

Este trabajo de investigación se presenta como Trabajo Final de Máster dentro del programa de Máster Universitario en Matemática Computacional para optar al título de Máster.

Castellón, 19 de Octubre de 2018

Fernando Casas Pérez

Índice

Capítulo 1. Introducción a Wolfram Mathematica.

1.1 ¿Qué es Wolfram Mathematica?

Mathematica es un programa desarrollado por Wolfram Research (www.wolfram.com) y utilizado en áreas científicas, de matemáticas, ingeniería y computacionales. Se trata de un sistema de álgebra computacional y de un lenguaje de programación de propósito general. Se divide en dos partes: el kernel o núcleo que desempeña los cálculos, y el front end o interfaz que despliega los resultados y permite al usuario interactuar con el núcleo como si fuera un documento [28].

Como las características más importantes de Mathematica podemos destacar: la capacidad de solucionar sistemas de ecuaciones (ordinarias, parciales o diferenciales); la disponibilidad de bibliotecas de funciones elementales y especiales para matemáticas; soporte de matrices; herramientas numéricas y simbólicas para cálculo de variable continua o discreta; herramientas de visualización de datos en 2D y 3D; una colección de bases de datos, etc.

Las principales funcionalidades más recientes incorporadas en Mathematica son las siguientes:

- ApplySides aplica operaciones algebraicas a ambos lados de ecuaciones y desigualdades y FindEquationalProof encuentra pruebas para teoremas lógicos ecuacionales desde axiomas.
- GeoSmoothHistogram crea intensidades alisadas en un mapa.
- FeatureSpacePlot3D para el trazado de espacios de funciones de dimensiones reducidas en 3D.
- Tiene capacidades de vanguardia para la construcción, capacitación y despliegue de sistemas de aprendizaje automático de redes neuronales.
- Navegue, importe o genere modelos de sistema listos para la simulación, para la extracción, análisis y visualización de datos.

1.2 Breve historia de Wolfram Mathematica.

Wolfram Research fue fundada por Stephen Wolfram en 1987 esta es una de la empresas más solventes en informática en el desarrollo de software y software para la nube.

El lanzamiento de Mathematica se produjo durante el año de 1988, siendo esta la primera versión que salió a la venta este mismo año, recién en el 2015 Wolfram Research hace el lanzamiento de Mathematica 10.3.

En la actualidad, la última versión disponible de Mathematica es la 11.3 y fue lanzado en marzo de 2018.

La versión 11.3 expande las funciones de Mathematica y Wolfram Language en computación matemática, audio y procesamiento de imágenes, aprendizaje automático y redes neuronales, modelado de sistemas y más; asimismo introduce nuevas propiedades de interfaz [27].

1.3 Entrada y salida de datos.

Para la entrada y salida de datos se debe tener cuidado ya que Mathematica es sensible con el uso de mayúsculas y minúsculas, una regla muy importante para la entrada y salida de los datos es que Mathematica utiliza paréntesis para las operaciones y no corchetes, para las operaciones básicas se utilizan los símbolos estándares, en la figura 1.1 se presenta un ejemplo:

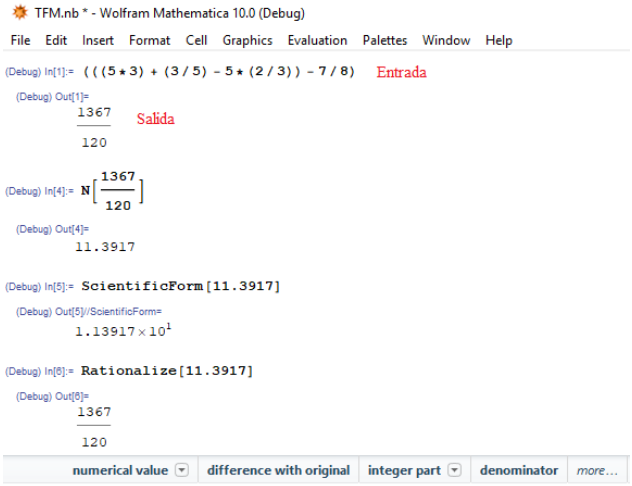


Figura 1.1: Entradas y salidas básicas en Mathematica

En la figura 1.1 se puede observar que al realizar una operación, Mathematica automáticamente presenta un menú de ayuda con comandos relacionados al tema.

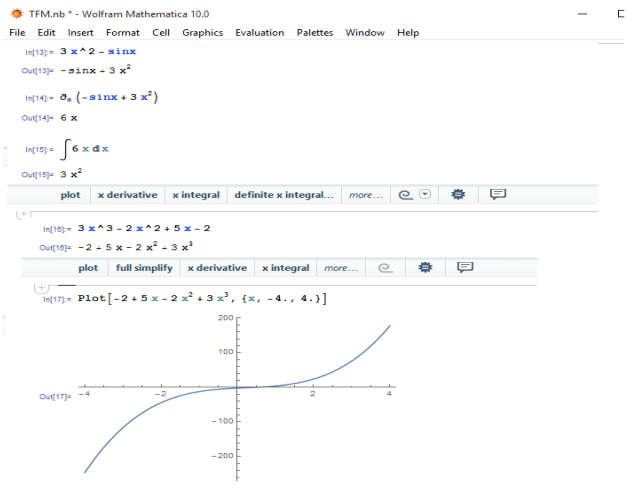


Imagen 1.2: Resolución de una integral

En la figura 1.2 se presenta algunos ejemplos con el desarrollo de funciones y se aplica derivadas e integrales con la ayuda del menú de comandos, se puede observar la salida de una gráfica de un polinomio de tercer grado.

1.4 Programación básica.

Mathematica soporta los clásicos comandos If, Do, For, While para el control de ciclos, que caracterizan los tradicionales lenguajes de programación, permitiendo escribir programas al estilo de Pascal y C [36].

Mathematica tiene funciones que también se puede aplicar en la programación, diseñada para la nueva generación de programadores, Wolfram Language posee una alta gama de algoritmos y conocimiento incorporado, todos accesibles de forma automática en su elegante y unificado lenguaje simbólico. Escalable desde programas pequeños hasta grandes, con implementación inmediata local y en la nube, Wolfram Language construye sobre la base de principios claros, y tres décadas de desarrollo, para crear lo que promete ser el lenguaje de programación más productivo del mundo, así como el primer lenguaje de comunicación computacional del mundo tanto para humanos como para AI (inteligencia artificial) [26].

En la figura 1.3 se presenta una entada utilizando el condicional If.

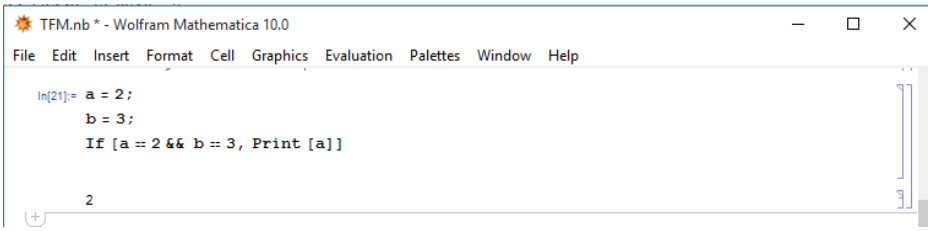


Figura 1.3: Condicional if

En la figura 1.4 se presenta la utilización de bucles (for), se presenta con un ejemplo simple de calcular el factorial del número 6.

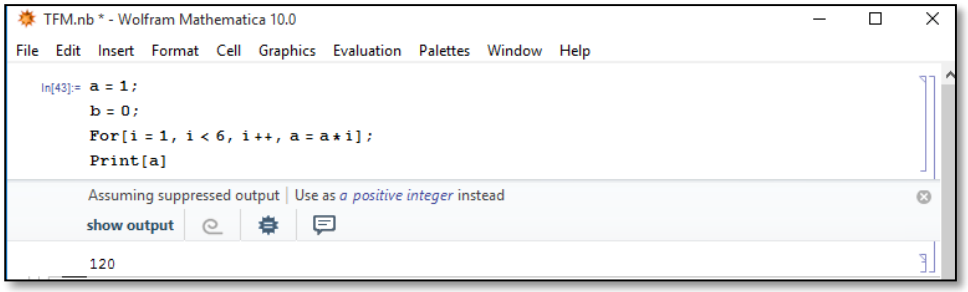


Figura 1.4: Bucle for

En la figura 1.5 se utiliza el bucle (while) para presentar los 10 primeros números pares.

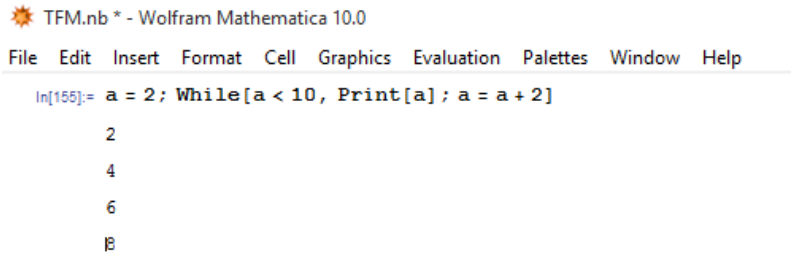


Figura 1.5: Bucle While

1.5 Componentes y estructuras de datos.

Mathematica tiene una extensa variedad de manejo de datos, además de los números convencionales como pueden ser los reales y complejos en operaciones de cálculo, álgebra y otros campos [32].

Se pueden encontrar enteros de diferentes tamaños (digamos 4 u 8 bytes), números reales en precisión simple y doble, y números complejos compuestos por pares de números reales. Mathematica nos brinda un conjunto de herramientas mucho más rico:

- Enteros de tamaño arbitrario
- Números racionales de tamaño arbitrario
- Números reales de doble precisión
- Números complejos
- Listas

Entre la gran cantidad de forma de representar datos en Mathematica, ya sea de forma convencional o en forma de estructura, en esta sección se trata de presentar una lista, se muestra cómo se debe trabajar con este tipo de datos, su sintaxis y ejemplos de aplicación. En la figura 1.6 se presenta dos listas, la primera con números enteros y la segunda con fracciones, se puede observar que se puede realizar operaciones con los elementos de la lista, inclusive podemos calcular la media, máximo o mínimo de los datos de las listas.

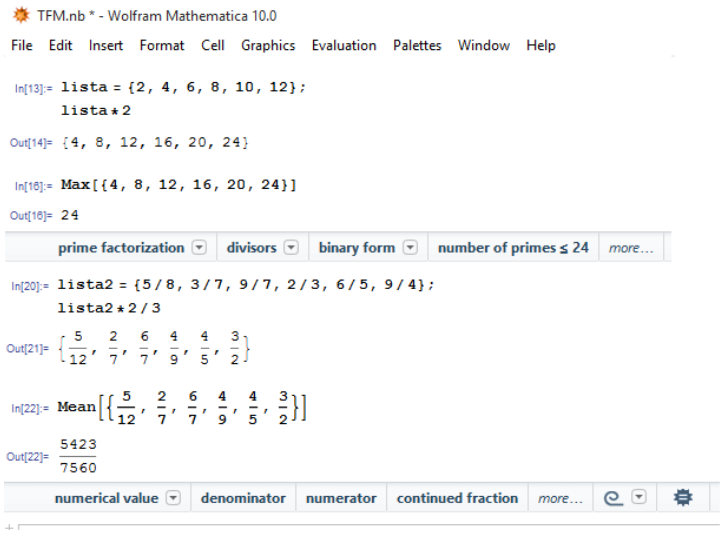


Figura 1.6: Estructura de datos

La figura 1.7 muestra el manejo de listas aplicando Estadística Descriptiva, como ejemplo se calcula la covarianza, la desviación típica, la correlación y la mediana, esto se presenta como ejemplo ya que Mathematica tiene incorporado una serie de comandos para el cálculo de la Estadística Descriptiva e Inferencial.

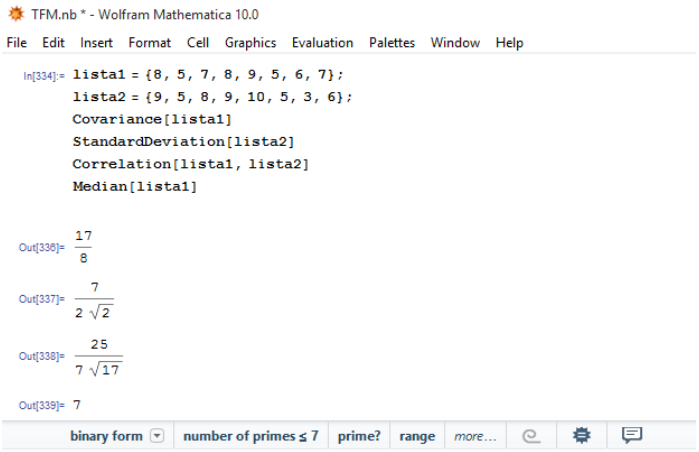


Figura 1.7: Listas

1.6 Implementación de Paquetes.

Para el correcto funcionamiento de algunas funciones de Mathematica es necesario realizar la instalación de paquetes, los cuales contiene una gran variedad de herramientas para el trabajo en diferentes categorías.

Mathematica ofrece una gran cantidad de paquetes de complementos estándar para extender sus capacidades. Muchos usuarios ignoran esta lista y terminan intentando duplicar las funciones. Mathematica es un sistema que puede representar y presentar conocimiento científico y técnico. Ciertos paquetes están incluidos con Mathematica para proporcionar un acceso fácil a los datos científicos de uso común [32].

Como se indica en el párrafo anterior existe una gran cantidad de paquetes como complementos, una de las ventajas que presenta este programa es que el usuario puede crear su propio paquete, y luego utilizarlo en el momento que se necesite.

Para cargar un paquete en Mathematica se utiliza la siguiente sintaxis:

<<paquete o categoría nombre del paquete de Mathematica

Otra manera de instalar un paquete es con la función Needs, como se muestra a continuación:

Needs[Nombre del paquete]

Como ejemplo se instala el paquete de estadística descriptiva, en el que están todas las funciones necesarias para trabajar con este tema, la instalación se realiza como se muestra en la figura 1.8.

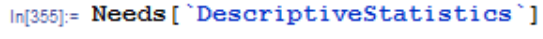


Figura 1.8: Instalación de paquetes

En la figura 1.9 se muestra todas las funciones que se instala en el paquete DescriptiveStatistics.

Nombre	Función
Media	Mean[]
Moda	Mode[]
Mediana	Median[]
Media geométrica	GeometricMean[]
Media armónica	HarmonicMean[]
Cuantiles n (entre 0 y 1)	Quantile[datos,n]
Cuartiles	Quartiles[]
Rango	SampleRange[]
Varianza	VarianceMLE[]
Cuasivarianza	Variance[]
Desviación estándar	StandardDeviationMLE[]
Cuasidesviación estándar	StandardDeviation[]
Coeficiente de variación	CoefficientOfVariation[]
Momento central de orden n	CentralMoment[datos,n]
Coeficiente de asimetría	Skewness[]
Coeficiente de Curtosis	KurtosisExcess[]

Figura 1.9: Funciones del paquete DescriptiveStatistics [14].

Mathematica posee un conjunto de paquetes para la resolución numérica de ecuaciones diferenciales, para ello se debe realizar la instalación de este paquete y de esa manera se puede acceder a todas las funciones.

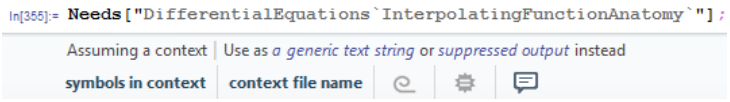


Figura 1.10: Instalación de un paquete.

El paquete DifferentialEquations`InterpolatingFunctionAnatomy` proporciona una interfaz para los datos en un objeto InterpolatingFunction que se mantendrá para futuras versiones de Wolfram Language.

Una situación común en la que el paquete InterpolatingFunctionAnatomy es útil es cuando NDSolve no puede calcular una solución en toda la gama de valores que ha especificado, y desea trazar toda la solución que se calculó para intentar comprender mejor lo que podría haber salido mal [22].

Capítulo 2. Resolución numérica de ecuaciones diferenciales en Wolfram Mathematica

2.1 DSolve.

El comando DSolve permite resolver sistemas de ecuaciones de diferentes tipos, estas pueden ser lineales con varias incógnitas, también puede desarrollar ecuaciones de diferentes potencias.

Resolver una ecuación diferencial consiste esencialmente en encontrar la forma de una función desconocida de manera que se satisfaga una cierta ecuación que involucra también a sus derivadas.

La función que se utiliza en Mathematica para obtener soluciones simbólicas de ecuaciones diferenciales ordinarias es:

$$DSolve\left[eqn, y\left[x\right], x\right]$$

Se realizan tres ejemplos utilizando el comando DSolve y NSolve.

Ejemplo 2.1

En este ejemplo se resuelve una ecuación diferencial con la utilización del comando DSolve.

```
(Debug) In[68]:=
DSolve[y'[x] + y[x] == 3 Cos[x] + 6, y[x], x]
(Debug) Out[68]=
{{y[x] → e^x C[1] + 3/2 (4 + Cos[x] + Sin[x])}}
```

Ejemplo 2.2:

En este ejemplo calculamos los puntos de equilibrio utilizando el comando NSolve.

$$f\left[u\_, v\_\right] := u*(1-u-v)$$

$$g\left[u\_, v\_\right] := v*(-3+u)$$

```
Introduciomos las funciones :

In[28]:= f[u_, v_] := u * (1 - u - v)
         g[u_, v_] := v * (-3 + u)

In[31]:= respuesta = NSolve[{f[x, y] == 0, g[x, y] == 0}, {x, y}]
Out[31]= {{x → 1., y → 0.}, {x → 0.333333, y → 0.666667}, {x → 0., y → 0.}, {x → 0., y → 0.}, {x → 0., y → 0.}}
```

Ejemplo 2.3:

Determinar y clasificar los puntos de equilibrio del sistema dinámico:

$$\frac{dx}{dt} = x(2-x-2y)$$
$$\frac{dy}{dt} = y(2-2x-y)$$

Realizar una representación del mapa de fases, que incluya las órbitas significativas.

```
f[x_, y_] := x * (2 - x - 2 * y);
g[x_, y_] := y * (2 - 2 x - y);
```

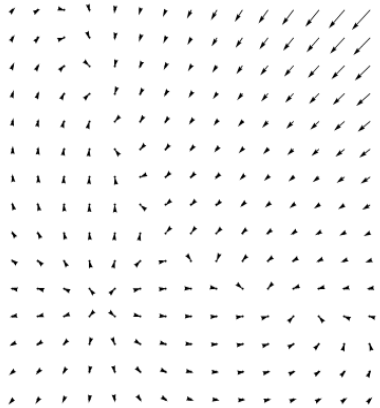
Se calcula los puntos de equilibrio.

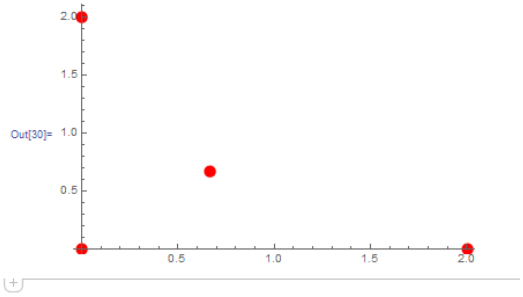
```
In[9]:= puntoseq = NSolve[{f[x, y] == 0, g[x, y] == 0}, {x, y}]
Out[9]= {{x → 2., y → 0.}, {x → 0., y → 2.}, {x → 0.666667, y → 0.666667}, {x → 0., y → 0.}}
```

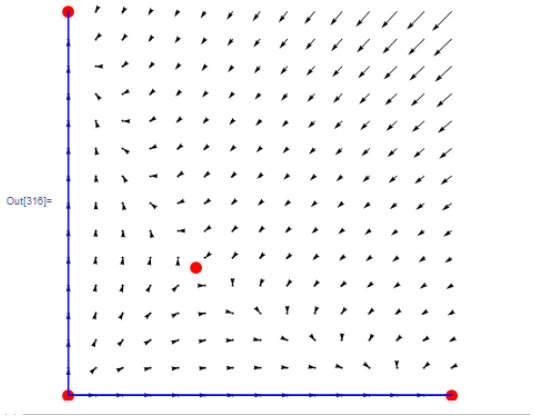
Se realiza los calculos convenientes para obtener de que tipo son los puntos :

```
In[11]:= P1 = {x, y} /. puntoseq[[1]]
         P2 = {x, y} /. puntoseq[[2]]
         P3 = {x, y} /. puntoseq[[3]]
         P4 = {x, y} /. puntoseq[[4]]
Out[11]= {2., 0.}
Out[12]= {0., 2.}
Out[13]= {0.666667, 0.666667}
Out[14]= {0., 0.}

In[15]:= J[{x_, y_}] = {{D[f[x, y], x], D[f[x, y], y]}, {D[g[x, y], x], D[g[x, y], y]}}
Out[15]= {{2 - 2 x - 2 y, -2 x}, {-2 y, 2 - 2 x - 2 y}}

In[18]:= J1 = J[P1]
Out[18]= {{-2., -4.}, {0., -2.}}

In[17]:= J2 = J[P2]
Out[17]= {{-2., 0.}, {-4., -2.}}

In[19]:= J3 = J[P3]
Out[19]= {{-0.666667, -1.33333}, {-1.33333, -0.666667}}

In[16]:= J4 = J[P4]
Out[16]= {{2., 0.}, {0., 2.}}

In[20]:= Eigensystem[J1]
Out[20]= {{-2., -2.}, {{1., 0.}, {0., 0.}}}

In[21]:= Eigensystem[J2]
Out[21]= {{-2., -2.}, {{0., 1.}, {0., 0.}}}

In[22]:= Eigensystem[J3]
Out[22]= {{-2., 0.666667}, {{0.707107, 0.707107}, {-0.707107, 0.707107}}}

In[23]:= Eigensystem[J4]
Out[23]= {{2., 2.}, {{-1., 0.}, {0., 1.}}}

In[33]:= gra1 = VectorFieldPlot[{f[x, y], g[x, y]}, {x, -1, 3}, {y, -1, 3}]
         gra2 = ListPlot[{{0, 2}, {2/3, 2/3}, {2, 0}, {0, 0}}, PlotStyle → {RGBColor[1, 0, 0], PointSize[0.03]}]
```

Out[33]=







En el ejemplo 2.2 se realiza el siguiente proceso:

- Se introduce las funciones.
- Utilizando el comando NSolve se calcula los puntos de equilibrio.
- Se realiza los cálculos pertinentes para verificar el tipo de puntos.
- Dibujamos el mapa de fase del sistema.
- Se grafica el campo de vectores.
- Se grafica los puntos de equilibrio.
- Se grafica los puntos de equilibrio y el campo de vectores sobre el mismo plano.

2.2 NDSolve.

La función NDSolve de Wolfram Mathematica proporciona aproximaciones numéricas de ecuaciones diferenciales, ésta tiene incorporada una serie de parámetros que facilita el cálculo de ecuaciones diferenciales ordinaria y en derivadas parciales.

La función NDSolve construye aproximaciones numéricas a la solución de ecuaciones diferenciales numéricas generales que se aproxima a la solución. Las ecuaciones diferenciales parciales implican dos o

más variables independientes. NDSolve también pueden resolver algunas ecuaciones diferenciales algebraicas (DAE), que suelen ser una combinación de ecuaciones diferenciales y algebraicas [21].

NDSolve tiene una serie de formas de expresión, a continuación se detallan algunas de ellas:

Está opción permite calcular una solución numérica para ecuaciones diferenciales ordinarias, para una función μ con la variable independiente x en el rango $x_{\min}$ a $x_{\max}$

$$NDSolve\left[eqn, \mu, \{x, x_{\min}, x_{\max}\}\right]$$

Sirve para resolver ecuaciones diferenciales parciales sobre un área regular.

$$NDSolve\left[eqn, \mu, \{x, x_{\min}, x_{\max}\}, \{t, t_{\min}, t_{\max}\}\right]$$

Permite resolver ecuaciones diferenciales en derivadas parciales sobre Ω.

$$NDSolve\left[eqn, \mu, \{x, y\} \in \Omega\right]$$

Resuelve ecuaciones diferenciales parciales que dependen del tiempo sobre la región Ω.

$$NDSolve\left[eqn, \mu, \{t, t_{\min}, t_{\max}\}, \{x, y\} \in \Omega\right]$$

NDSolve puede resolver funciones subíndice:

$$NDSolve\left[eqn, \{u_1, u_2, u_3.....\}, ...\right]$$

Con el comando Options [NDSolve] se puede acceder a las opciones que tiene incorporado NDSolve.

```
(Debug) In[65]:=
Options[NDSolve]
(Debug) Out[65]=
{AccuracyGoal → Automatic, Compiled → Automatic,
 DependentVariables → Automatic, DiscreteVariables → {},
 EvaluationMonitor → None, InterpolationOrder → Automatic,
 MaxStepFraction → 1/10, MaxSteps → Automatic, MaxStepSize → Automatic,
 Method → Automatic, NormFunction → Automatic,
 PrecisionGoal → Automatic, StartingStepSize → Automatic,
 StepMonitor → None, WorkingPrecision → MachinePrecision}
```

A continuación se presenta dos ejemplos aplicando NDSolve.

Ejemplo 2.3.

```
In[8]:= sol = NDSolve[{y'[x] == y[x] Sin[2 x + y[x]], y[0] == 1}, y, {x, 0, 15}]
Out[8]= {{y → InterpolatingFunction[ Domain: {{0., 15.}}  Output: scalar ]}}
```

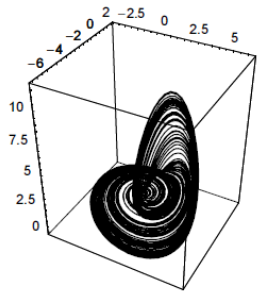
Se presentan algunas funcionalidades y algunos ejemplos utilizando NDSolve [39].

Ejemplo 2.4. Sistema de Rossler [39],

$$x'(t) = -y(t) - z(t)$$
$$y'(t) = x(t) + ay(t)$$
$$z'(t) = b - cz(t) + x(t)z(t)$$

Como se puede observar es un sistema de tres ecuaciones diferenciales no lineales, se procede a asignar valores a los parámetros $a = 1$, $b = 3$ y $c = 6$.

```
In[243]:=
x'[t] := y[t] + z[t];
y'[t] := x[t] + a * y[t];
z'[t] := b - c * z[t] + x[t] * z[t];

In[245]:= Module[{a = 1/2, b = 2, c = 4, T = 500},
 ndsolRössler =
  NDSolve[{x'[t] == -y[t] - z[t], y'[t] == x[t] + a y[t], z'[t] == b + x[t] z[t] - c z[t],
    (*start at the origin*)x[0] == 0, y[0] == 0, z[0] == 0}, {x, y, z}, {t, 0, T},
   MaxSteps → 10^5];
 (*show trajectory*)ParametricPlot3D[Evaluate[{x[t], y[t], z[t]} /. ndsolRössler[[1]]],
 {t, 0, T}, PlotPoints → 1000, PlotRange → All]];
```

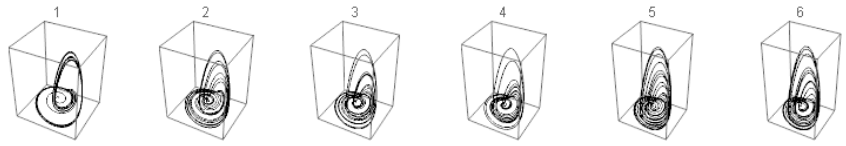


Se puede observar que la función NDSolve puede resolver fácilmente las ecuaciones diferenciales planteadas.

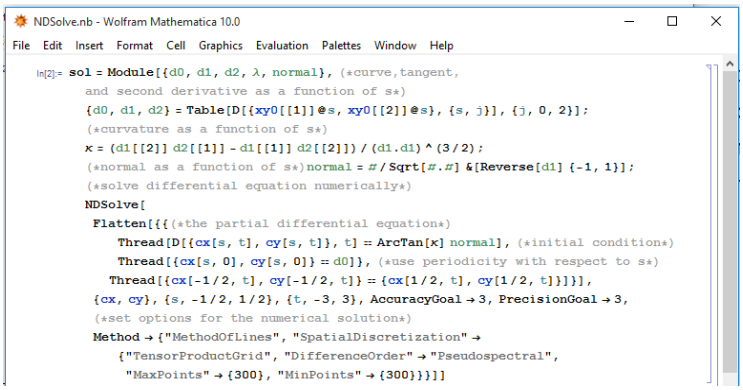
Ejemplo 2.5 Discretización de Euler. La siguiente serie de cinco gráficos muestra la solución obtenida al usar la discretización de Euler, este procedimiento se estudiará más adelante.

```
In[247]:= hpEuler[n_, h_, xyz0_ : {0, 0, 0}] :=
 Module[{a = 1/2, b = 2, c = 4, x = xyz0[[1]], y = xyz0[[2]], z = xyz0[[3]]},
  Table[{x, y, z} = {x + h (-y - z), y + h (x + a y), z + h (b + x z - c z)}, {n}]]
Show[
 GraphicsArray[
  Table[Graphics3D[Line[N[hpEuler[Round[10^4/k], N[k 10^-2, 10^3]]]], PlotRange → All,
    PlotLabel → k], {k, 6}]]]
GraphicsArray::obs : GraphicsArray is obsolete. Switching to GraphicsGrid. >>
```

Out[248]=

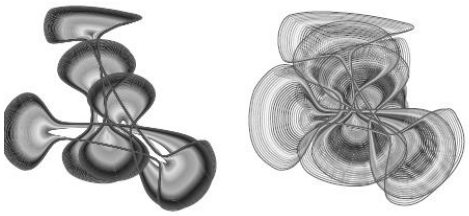


Ejemplo 2.6. Resuelve las ecuaciones diferenciales parciales numéricamente, este procedimiento numérico es el método de líneas.

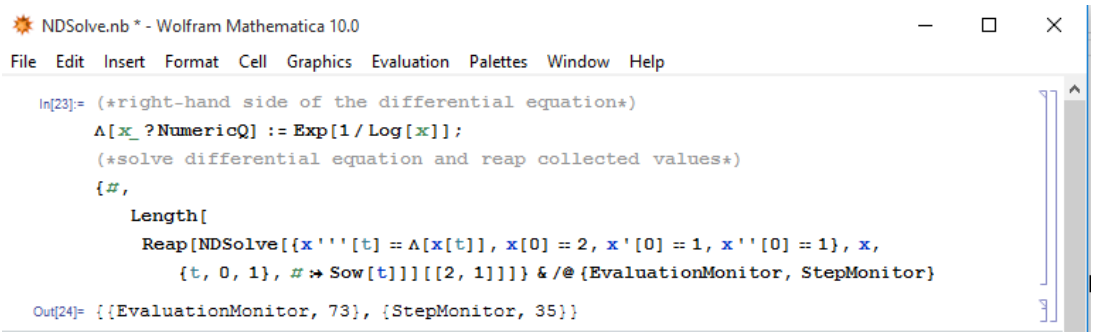


$$cx\left[-\frac{1}{2}, t\right] == cx\left[\frac{1}{2}, t\right], cy\left[-\frac{1}{2}, t\right] == cy\left[\frac{1}{2}, t\right]\Big\}, \{cx, cy\},$$
$$\left\{s, -\frac{1}{2}, \frac{1}{2}\right\}, \{t, -3, 3\}, AccuracyGoal \to 3, PrecisionGoal \to 3,$$
$$Method \to \{MethodOfLines, SpatialDiscretization \to \{TensorProductGrid,$$
$$DifferenceOrder \to Pseudospectral, MaxPoints \to \{300\}, MinPoints \to \{300\}\}\}\Big]$$

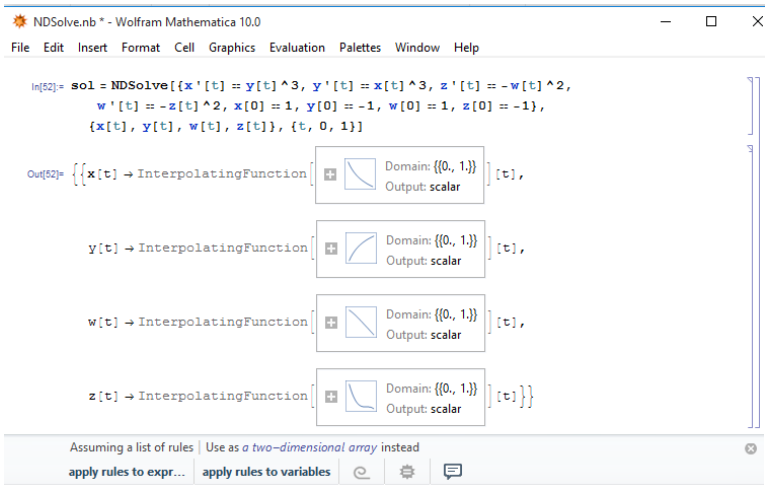
```
In[22]:= Show[
  GraphicsArray[
   Show[Table[ParametricPlot[Evaluate[{cx[s, t], cy[s, t]} /. ndsol[[1]]],
      {s, -1/2, 1/2}, PlotPoints → 100, Axes → False, AspectRatio → Automatic,
      PlotStyle → Hue[0.8 t], DisplayFunction → Identity],
     {t, #1, #2, (#2 - #1)/60}]] & @@@ {{0, 1}, {-2, 2}}]]
```



Ejemplo 2.7. Utilizando el comando NDSolve se desarrolla la ecuación diferencial $x'''(t) = \exp\left(\frac{1}{In(x(t))}\right)$.



Cuando se utiliza NDSolve para resolver un sistema de ecuaciones diferenciales, se genera cuatro objetos de regla, una por cada variable que presenta la ecuación.



2.3 El comando NDSolve y Plot.

El comando Plot representa la gráfica de funciones, elementos geométricos o ecuaciones diferenciales, su sintaxis en Mathematica es:

$$Plot\left[f,\ \{ x,\ xmin,\ xmax\}\right]$$
$$Plot\left[\ \{ f1,\ f2,\ldots\},\ \{x,\ xmin,\ xmax\}\right]$$

El comando NDSolve y Plot se pueden utilizar conjuntamente para la evaluación de ecuaciones diferenciales, en el ejemplo 2.8 proporciona una solución numérica para una EDO no lineal como la ecuación de Van der Pol (el oscilador de Van der Pol es un sistema dinámico consistente en un circuito eléctrico no lineal).

Ejemplo 2.8.

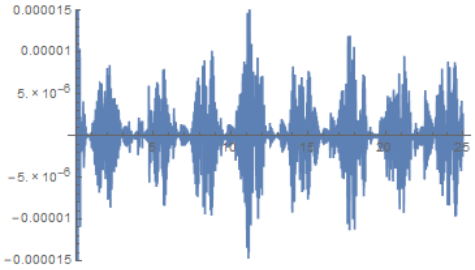
Ecuación de Van der Pol [30].

$$x'' - \varepsilon\left(1 - x^2\right)x' + x = 0$$

$$x'' - \frac{1}{3}\left(1 - x^2\right)x' + x = 0$$

```
(Debug) In[26]:=
solucion =
 NDSolve[{x''[t] - 1/3 * (1 - x[t]^2) x'[t] + x[t] == 0, x[0] == 1,
   x'[0] == 0}, x, {t, 0, 25}]
(Debug) Out[26]=
{{x → InterpolatingFunction[ Domain: {{0., 25.}}
                             Output: scalar ]}}
```

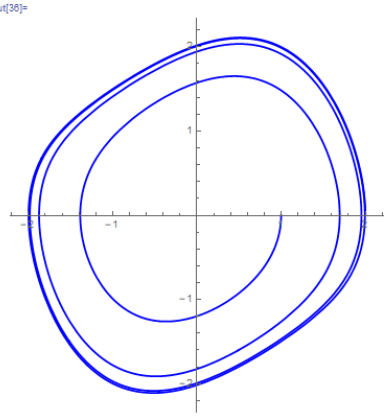
```
(Debug) In[32]:=
Plot[N[Evaluate[x''[t] - 1/3 * (1 - x[t]^2) x'[t] + x[t] /. solucion],
  3] , {t, 0, 25}, PlotRange → {-0.000015, 0.000015}]
(Debug) Out[32]=
```



Utilizando NDSolve y Plot podemos verificar que la solución numérica obtenida tiene bastante precisión. La solución numérica verifica la ecuación diferencial en el intervalo $[0,25]$.

ParametricPlot es un comando que se utiliza conjuntamente con NDSolve y nos permite presentar el plano de fase. El ejemplo permite observar el plano de fases de la Ecuación de Van der Pol.

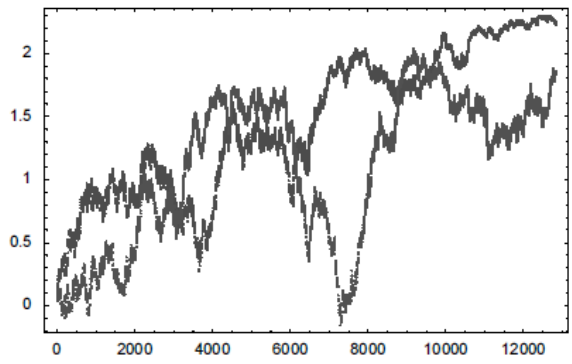
```
(Debug) In[38]:=
ParametricPlot[Evaluate[{x[t], x'[t]} /. solucion[[1]]],
 {t, 0, 25}, PlotStyle → {Thickness[0.005], RGBColor[0, 0, 1]}]
(Debug) Out[38]=
```



2.4 NDSolve utilidades y aplicación.

Utilización de NDSolve para resolver ecuaciones diferenciales para la posición y la velocidad de un oscilador anarmónico $x''(t)+x(t)^3=\frac{1}{2}\sin\left(\omega(t)+\varphi(t)\right)x(t)$ [40], se debe tomar en cuenta que frecuencia omega y el defasaje phi son funciones aleatorias.

```
(Debug) In[1]:= (*extract extrema positions from a curve segment*)
extrema[Line[l_]] :=
 Point[#[[2]]] & /@ Select[Partition[l, 3, 1],
   (Abs[#[[2, 2]]] > Max[Abs[{#[[1, 2]], #[[3, 2]]}]]) &]
Module[{ω0 = 10, x0, v0, ϕ, ω, δT, T0, ndsol, pl, o = 2, n = 20000},
 Show[(*two random realizations of the time evolution*)
  Table[(*seed random number generator*)SeedRandom[ ];
   (*initial time*)T0 = 0;
   (*start position and velocity*){x0, v0} = {0., 1.};
   (*show graphics of maximal elongations*)
   Graphics[{PointSize[0.005], Hue[( -1) / (o - 1) 0.78],
      (*follow oscillator over n periods of the field oscillations*)
      Table[If[(*exit gracefully in case of a solution blow-up*)
        MatchQ[{x0, v0}, {_Real, _Real}], (*random phase and frequency*)
        ϕ = Random[Real, {0, 2 Pi}];
        ω = Random[Real, {3 / 4 ω0, 5 / 4 ω0}];
        (*one period*)δT = 2 Pi / ω;
        (*solve differential equations of motion*)
        ndsol =
         Check[NDSolve[{v[t] == x'[t], v'[t] + x[t]^3 + 1 / 2 Sin[ω t + ϕ] x[t] == 0,
            x[T0] == x0, v[T0] == v0}, {x, v}, {t, T0, T0 + δT}], $Failed];
        (*extract final position and velocity*)
        If[ndsol === $Failed, {x0, v0} = {$Failed, $Failed},
         {x0, v0} = {x[T0 + δT], v[T0 + δT]} /. ndsol[[1]];
         (*plot x[t]*)pl = Plot[Evaluate[x[t] /. ndsol[[1]]], {t, T0, T0 + δT},
           PlotPoints → Ceiling[5 + 10 δT], DisplayFunction → Identity];
         (*extract extremas from lines from the plot*)T0 = T0 + δT;
         extrema /@ Cases[pl, _Line, Infinity]]], {n}]} /.
     (*show on logarithmic scale*)Point[{t_, x_}] ↦ Point[{t, Log[Abs[x]]}]],
   { , o}], Frame → True, PlotRange → All]]
```

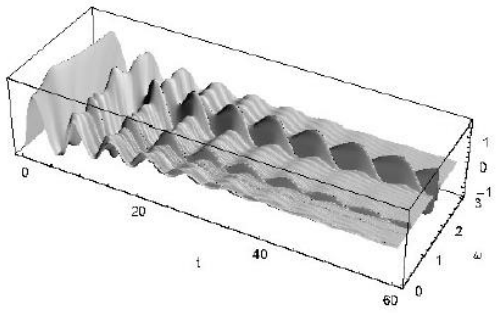


Utilización de NDSolve para desarrollar la siguiente ecuación diferencial:

$$\frac{\partial\left(l(t)^2\,\theta'(t)\right)}{\partial t}+\ell l(t)^2\,\theta'(t)+l(t)\sin\left(\theta(t)\right)=0$$

para la elongación $\theta'(t)$ más de 200 veces para $l(t)$ de la forma $l(t)=1+\varepsilon\sin(\omega t+9/8\pi)^7$ Esta ecuación diferencial describe un péndulo de varilla sin masa con longitud de varilla dependiente del tiempo y una masa.

```
Module[{γ = 0.16, ∂ = 0.2, ω = 1, T = 60, ppω = 240, ppT = 480,  , data},
 (*time dependent periodic rod length with main frequency ω*)
  [t_] := 1 + ∂ Sin[ω t + 9 / 8 Pi] ^ 7;
 data = Table[(*solve differential equation for fixed ω*)
   ndsol = NDSolve[{D[  [t] ^ 2 θ'[t], t] + γ  [t] ^ 2 θ'[t] +  [t] Sin[θ[t]] == 0,
      θ[0] == -1, θ'[0] == 1}, θ, {t, 0, T}];
   (*elongation data for fixed ω*)
   Table[Evaluate[{t, ω, θ[t]} /. ndsol[[1]]], {t, 0, T, T / ppT}], {ω, 0, 3, 2 / ppω}];
 (*show elongation over the t,ω-plane*)
 Show[Graphics3D[{EdgeForm[], (*form polygons*)
    Cases[ListSurfacePlot3D[data, DisplayFunction → Identity], _Polygon,
     Infinity]}], PlotRange → All, BoxRatios → {2.8, 1, 0.6}, Axes → True,
  AxesLabel → {"t", "ω", None}]]
```



A partir de un punto genérico $z_0=z(0)$ y resolver la ecuación diferencial $z'(t)=-p(z(t))/p'(z(t))$

```
(Debug) In[15]:=
f[x_] = Sum[Random[Integer, {-4, 4}] x^i, {i, 0, 6}]
(Debug) Out[15]=
-3 + 4 x - 4 x^2 - 4 x^3 - 4 x^4 - x^5 + 3 x^6
```

```
(Debug) In[7]:= SeedRandom[555];
(Debug) In[15]:=
f[x_] = Sum[Random[Integer, {-4, 4}] x^i, {i, 0, 6}]
(Debug) Out[15]=
2 - 4 x^2 + 4 x^3 - x^4 + 2 x^5 - 2 x^6
(Debug) In[16]:=
df[x_] = -f[x] / D[f[x], x]
(Debug) Out[16]=
-2 + 4 x^2 - 4 x^3 + x^4 - 2 x^5 + 2 x^6
-----------------------------------------
-8 x + 12 x^2 - 4 x^3 + 10 x^4 - 12 x^5
```

```
(Debug) In[17]:=
polyRoots = x /. NSolve[f[x] == 0, x]
(Debug) Out[17]=
{-0.57532 - 1.1936 i, -0.57532 + 1.1936 i, -0.537685, 0.739338 - 0.573674 i, 0.739338 + 0.573674 i, 1.20965}
```

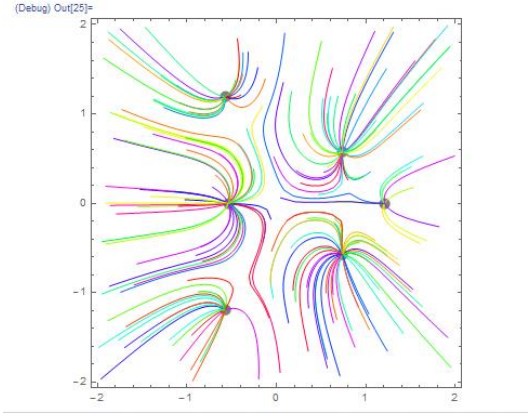
Se resuelve numéricamente la ecuación diferencial correspondiente, a partir de 175 valores elegidos al azar.

```
(Debug) In[22]:=
Off[NDSolve::mxst];
flowLines =
  Table[
   NDSolve[{z'[t] == df[z[t]], (*use random complex numbers as initial
      conditions*) z[0] == Random[Complex, {-2 - 2 I, 2 + 2 I}]}, z, {t, 0, 1000},
    MaxStepSize → 1], {175}];
On[NDSolve::mxst];
```

A continuación se presenta las líneas de flujo resultantes.

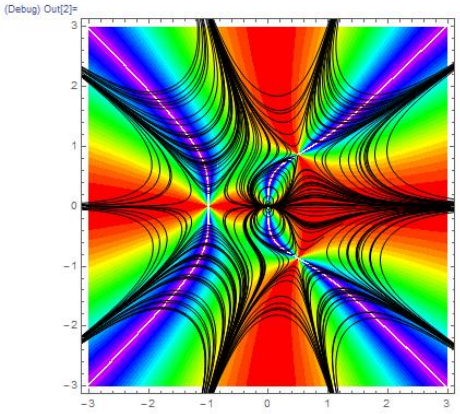
```
(Debug) In[25]:=
Show[
 {Graphics[{PointSize[0.03], (*the roots*)GrayLevel[1/2],
    Point[{Re[#], Im[#]}] & /@ polyRoots}], (*the flow lines*)
  ParametricPlot[Evaluate[{Re[z[t]], Im[z[t]]} /. #], {t, 0, #[[1, 1, 2, 1, 1, 2]]},
    PlotStyle → {Hue[Random[]], Thickness[0.002]}, PlotPoints → 200,
    DisplayFunction → Identity] & /@ flowLines}, DisplayFunction → $DisplayFunction,
 PlotRange → All, Frame → True, Axes → False, AspectRatio → Automatic]
```

(Debug) Out[25]=



Para las ecuaciones diferenciales que describen trayectorias es conveniente usar parametrizaciones que representa la longitud del arco de la curva, esto puede lograrse fácilmente mediante la normalización del lado derecho de la ecuación diferencial, como se puede observar a continuación.

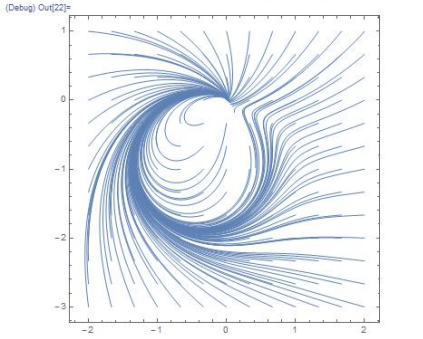
```
(Debug) In[3]:= SeedRandom[123];
Module[{n = 200, df, newtonFlow, flowLines},
 df[x_] = Function[f, -f'[x]/f[x]][Exp[1/(#^3 + 1)] &];
 newtonFlow[x_?InexactNumberQ] := #/Abs[#] &[df[x]];
 Off[NDSolve::mxst]; Off[NDSolve::ndcf];
 flowLines = Table[NDSolve[{z'[t] == newtonFlow[z[t]],
     z[0] == Random[Complex, 2 {-1 - I, 1 + I}]}, z, {t, -3, 3}], {n}];
 On[NDSolve::mxst]; On[NDSolve::ndcf];
 Show[{ContourPlot[Evaluate[Arg[newtonFlow[x + I y]]^2], {x, -3, 3},
    {y, -3, 3}, PlotPoints → 200, ColorFunction → (Hue[0.8 #] &),
    ContourLines → False, Contours → 30, DisplayFunction → Identity],
   ParametricPlot[Evaluate[{Re[z[t]], Im[z[t]]} /. #],
     Evaluate[Flatten[{t, #[[1, 1, 2, 1, 1]]}]],
     PlotStyle → {GrayLevel[0], Thickness[0.002]}, DisplayFunction → Identity] & /@
    flowLines}, DisplayFunction → $DisplayFunction, PlotRange → {{-3, 3}, {-3, 3}}]]
```



Se presenta un sistema de dos ecuaciones diferenciales acopladas, los lados derechos son funciones racionales. El siguiente sistema modela la formación de ondas espirales en reacciones química.

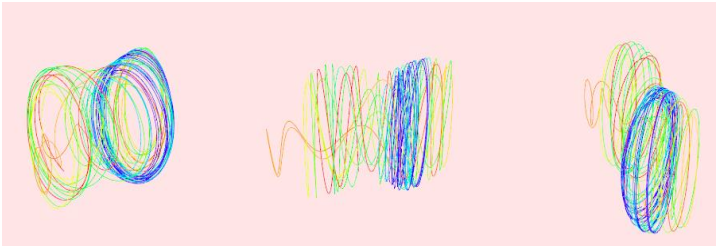
```
phasePortrait[{l_, ω_, p_, d_, q_, e_, c_}, tMax_, pp_] :=
 Module[{x0, y0, x, y, t, nsol},
  Do[(*Ecuacioiones diferenciales*)
   nsol[x0, y0] =
    NDSolve[
     Flatten[
      {x'[t] == l x[t] (1 - x[t]^2 - (q + (1 + p) y[t])^2) -
         ω y[t] (1 - q^2 - (d ((1 + p)^2 - c (1 - q)) y[t] (1 - e + e y[t])) /
              (1 + p - e (p + q)) - c (x[t]^2 + y[t]^2)),
       y'[t] == l y[t] (1 - x[t]^2 - (q + (1 + p) y[t])^2) +
         ω x[t] (1 - q^2 - (d ((1 + p)^2 - c (1 - q)) y[t] (1 - e + e y[t])) /
              (1 + p - e (p + q)) - c (x[t]^2 + y[t]^2)), x[0] == x0, y[0] == y0}],
     {x, y}, {t, 0, tMax}], {x0, -2, 2, 4 / pp}, {y0, -3, 1, 4 / pp}];
  (*genera visualización*)
  Show[Table[ParametricPlot[Evaluate[{x[t], y[t]} /. nsol[x0, y0][[1]]],
     {t, 0, tMax}, PlotStyle → {Thickness[0.002]}, DisplayFunction → Identity],
    {x0, -2, 2, 4 / pp}, {y0, -3, 1, 4 / pp}],
   DisplayFunction → $DisplayFunction, AspectRatio → Automatic, PlotRange → All,
   Frame → True, Axes → False, PlotRange → {{-3, 1}, {-3, 3}}]]
```

```
(Debug) In[22]:=
phasePortrait[{1, 1/2, 0, 1/2, 11/10, 0, 1}, 10, 12]
```



Se presenta un sistema de cuatro ecuaciones diferenciales ordinarias no lineales acopladas con polinomios de lado derecho. La curva de solución forma un atractor caótico la que se muestra en proyecciones de 3D.

```
Module[{α, β, x0, y0, vx0, vy0, T = 200, ndsol, path4D, colors, ℝ, viewGraphics},
 (*parámetros*){α, β, x0, y0, vx0, vy0} =
  {770 / 471, 24 / 187, -312 / 583, -18 / 37, -324 / 403, 104 / 785};
 ndsol = NDSolve[{x'[t] == vx[t], y'[t] == vy[t], vx'[t] == -(α + y[t]^2) x[t] + y[t],
    vy'[t] == -(β + x[t]^2) y[t] + x[t], x[0] == x0, y[0] == y0, vx[0] == vx0,
    vy[0] == vy0}, {x, y, vx, vy}, {t, 0, T}, MaxSteps → 10^5];

 path4D = Table[Evaluate[{x[t], y[t], vx[t], vy[t]} /. ndsol[[1]]],
   {t, 0, T, 1 / 20}];

 colors = Table[Hue[0.78 x], {x, 0, 1, 1 / (Length[path4D] - 2)}];

 ℝ[ϕ_] := N@{{Cos[ϕ], Sin[ϕ], 0, 0}, {-Sin[ϕ], Cos[ϕ], 0, 0}, {0, 0, 1, 0},
    {0, 0, 0, 1}};

 viewGraphics[ϕ_] :=
  With[{rm = ℝ[ϕ]},
   Graphics3D[{Thickness[0.002],
     Transpose[{colors, Line /@ Partition[Take[rm.#, 3] & /@ path4D, 2, 1]}]},
    PlotRange → All, SphericalRegion → True, Boxed → False, BoxRatios → {1, 1, 1}]];

 Show[GraphicsArray[viewGraphics /@ {0, Pi / 4, Pi / 2}], GraphicsSpacing → -0.1]]
```



NDSolve también resuelve sistemas de ecuaciones diferenciales grandes, se plantea un sistema de 15 ecuaciones diferenciales de segundo orden que se acoplan de forma no lineal entre sí, se elige las constantes y los exponentes al azar.

```
(Debug) In[30]:=
num = 15;
SeedRandom[968];
(diffEqn =
   Table[
    x[i]''[t] == Sum[Random[Integer, {-2, 2}] * x[j][t]^Random[Integer, {0, 3}],
      {j, num}], {i, num}]) // Short[#, 16] &
```

```
(Debug) Out[32]//Short=
{x[1]''[t] == 2 + 2 x[1][t] + 2 x[2][t] - 2 x[5][t]^3 -
    x[7][t]^3 - x[8][t]^2 - x[9][t]^3 + x[12][t]^2 + 2 x[13][t] + 2 x[15][t]^3,
 x[2]''[t] == -4 + 2 x[3][t] + x[4][t] - x[5][t]^2 + 2 x[9][t] + x[11][t]^2 -
    2 x[12][t]^2 + x[14][t] + x[15][t], x[3]''[t] == x[1][t] + 2 x[5][t]^2 -
    x[7][t]^3 - x[9][t] + 2 x[12][t]^2 + 2 x[13][t]^2 + 2 x[14][t] + x[15][t],
 x[4]''[t] == -8 - 2 x[4][t]^2 - 2 x[10][t]^3 - x[13][t]^3,
 x[5]''[t] == 6 - x[2][t] - x[3][t] - 2 x[6][t] +
    2 x[7][t] - 2 x[8][t]^3 - x[10][t]^2 + x[11][t]^3 + x[13][t]^3,
 x[6]''[t] == 1 - x[1][t]^3 - 2 x[2][t]^3 + x[5][t]^3 - x[7][t]^3,
 x[7]''[t] == 2 - 2 x[1][t]^2 - x[2][t]^2 - x[5][t]^2 - x[6][t]^3 - x[8][t]^3 + x[9][t]^2 +
    2 x[10][t]^3 + x[11][t]^3 - x[12][t]^2 - x[14][t]^2 + x[15][t], x[8]''[t] ==
  -2 - x[2][t]^2 - 2 x[4][t]^2 - x[10][t] + x[11][t]^2 + x[12][t] + x[14][t]^2 - 2 x[15][t]^2,
 x[9]''[t] == 2 - 2 x[1][t]^2 + 2 x[2][t] + x[3][t] + 2 x[5][t] + 2 x[6][t]^2 - x[7][t] +
    2 x[8][t] - 2 x[9][t] + x[10][t]^3 - x[12][t]^2 - 2 x[13][t] - x[14][t]^2 - x[15][t],
 x[10]''[t] == -2 - x[3][t]^3 + x[4][t]^3 + 2 x[6][t]^2 + x[7][t]^2 +
    x[10][t]^2 + 2 x[11][t] + 2 x[12][t]^3 - x[13][t]^2,
 x[11]''[t] == -2 + 2 x[1][t] - x[2][t] - x[5][t] - 2 x[6][t] + x[7][t] +
    2 x[10][t] + x[11][t] + 2 x[12][t]^2, x[12]''[t] == -1 - x[1][t] - 2 x[3][t]^2 -
    x[6][t]^2 + 2 x[7][t]^2 + x[8][t] - x[9][t]^2 + x[11][t]^2 + x[13][t],
 x[13]''[t] == 4 - 2 x[1][t] + 2 x[3][t]^2 + 2 x[4][t] + 2 x[5][t] - 2 x[6][t]^2 -
    2 x[7][t] + x[9][t] + x[10][t]^3 + 2 x[12][t]^3 - x[13][t] + 2 x[14][t] + 2 x[15][t],
 x[14]''[t] == -5 - 2 x[1][t]^2 + 2 x[2][t] - x[3][t]^2 + 2 x[7][t]^3 + x[8][t]^2 -
    2 x[9][t]^2 - 2 x[10][t]^2 + x[11][t]^3 - 2 x[13][t]^3 + 2 x[14][t]^3 + x[15][t]^2,
 x[15]''[t] == 2 x[3][t]^2 + 2 x[6][t] + 2 x[7][t] - 2 x[9][t]^3 - 2 x[13][t]^3 - x[15][t]^3}
```

```
(Debug) In[33]:=
eqn = Join[diffEqn, Table[x[i][0] == 1, {i, num}], Table[x[i]'[0] == 1, {i, num}]];

(Debug) In[34]:=
Timing[sol = NDSolve[eqn, Table[x[i], {i, num}], {t, 0, 1/2}]]
```

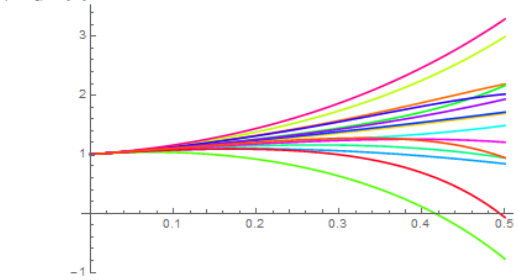
(Debug) Out[34]=

{0., {{x[1] → InterpolatingFunction[Domain: {{0., 0.5}} Output: scalar],
x[2] → InterpolatingFunction[Domain: {{0., 0.5}} Output: scalar],
x[3] → InterpolatingFunction[Domain: {{0., 0.5}} Output: scalar],
x[4] → InterpolatingFunction[Domain: {{0., 0.5}} Output: scalar],
x[5] → InterpolatingFunction[Domain: {{0., 0.5}} Output: scalar],
x[6] → InterpolatingFunction[Domain: {{0., 0.5}} Output: scalar],
x[7] → InterpolatingFunction[Domain: {{0., 0.5}} Output: scalar],
x[8] → InterpolatingFunction[Domain: {{0., 0.5}} Output: scalar],
x[9] → InterpolatingFunction[Domain: {{0., 0.5}} Output: scalar],
x[10] → InterpolatingFunction[Domain: {{0., 0.5}} Output: scalar],
x[11] → InterpolatingFunction[Domain: {{0., 0.5}} Output: scalar],
x[12] → InterpolatingFunction[Domain: {{0., 0.5}} Output: scalar],
x[13] → InterpolatingFunction[Domain: {{0., 0.5}} Output: scalar],
x[14] → InterpolatingFunction[Domain: {{0., 0.5}} Output: scalar],
x[15] → InterpolatingFunction[Domain: {{0., 0.5}} Output: scalar]}}}

(Debug) In[35]:=

```
Plot[Evaluate[Table[x[i][t] /. sol, {i, num}], {t, 0, 1/2},
  PlotStyle → Table[Hue[j/10 0.7], {j, num}], PlotRange → All]
```

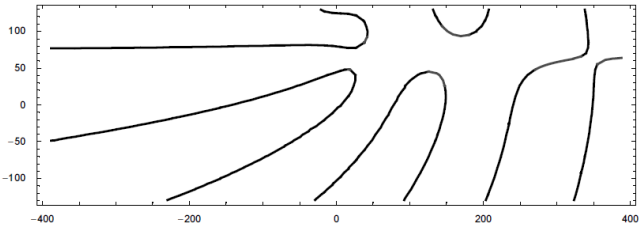
(Debug) Out[35]=



NDSolve con ecuaciones diferenciales de valores complejos [39].

```
In[52]:= energyBandGraphic[d_, VMax_, seed_] :=
     Module[{δ = 4 VMax, ppε = 36,

       randomTrigPoly, cp, lineSegments, allowedLineSegments},
 (* random complex 1D potential (rational function in trigs) *)
 randomTrigPoly[d_] := Sum[Random[] Exp[2 Pi I Random[]]*
                          Cos[k 2Pi x + 2Pi Random[]], {k, d}];
 V[x_] = VMax randomTrigPoly[d]/randomTrigPoly[d];
 (* show curves Im[Δ[ε]] == 0 *)
 cp = ContourPlot[Im[Δ[εr + I εi]], {εr, -δ, δ}, {εi, -δ/3, δ/3},
                  Contours -> {0}, DisplayFunction -> Identity,
                  ContourShading -> False, PlotPoints -> {2, 1} ppε];
 (* extract linesegments that form the curves *)
 lineSegments = Line /@ Flatten[Partition[First[#], 2, 1]& /@
                               Cases[Graphics[cp], _Line, Infinity], 1];
 (* extract linesegments corresponding to allowed energy bands *)
 allowedLineSegments =  Select[lineSegments,
                          Abs[Re[Δ[{1, I}.(Plus @@ #[[1]]/2)]]] < 1&];
(* show allowed energy bands in red *)
 Show[Graphics[{{Thickness[0.004], GrayLevel[0],
                 Complement[lineSegments, allowedLineSegments]},
                Thickness[0.004], Hue[0], allowedLineSegments}],
      PlotRange -> All, AspectRatio -> Automatic, Frame -> True]]
```

```
In[53]:= energyBandGraphic[10, 97, 269889295]
```



Capítulo 3. Métodos numéricos para ecuaciones diferenciales.

3.1 Método de Euler.

Euler propuso uno de los primeros métodos para resolver ecuaciones diferencias con problemas de valores iniciales, cabe destacar que este método es muy sencillo en sus cálculos.

Este método permite aproximar soluciones del problema de valor inicial.

$$y' = f(x, y) \qquad y(x_0) = y_0 \qquad (3.1)$$

Se puede analizar $y' = f(x, y)$ como un campo de direcciones en el plano $x - y$ con la condición inicial $y(x_0) = y_0$, se puede aproximar la solución $y(x)$ por medio de la recta tangente a la misma que pasa por ese punto:

$$y(x) \cong y_0 + f(x_0, y_0)(x - x_0) \qquad (3.2)$$

Por lo tanto se tiene la pendiente de la tangente $m = y'(x_0)$, entonces se deduce que $m = f(x_0, y_0)$.

Se aproxima el valor de la solución y en el punto de abscisa x_1 como:

$$y(x_1) \cong y_1 + f(x_0, y_0)(x_1 - x_0) \qquad (3.3)$$

Con el punto aproximado, se puede repetir el método para otro punto aproximado (x_2, y_2), entonces se tiene:

$$y(x_2) \cong y_2 = y_1 + f(x_1, y_1)(x_2 - x_1) \qquad (3.4)$$

de esta manera se va obteniendo los valores aproximados.

Para abscisas equiespaciadas, se tiene las fórmulas:

$$x_n = x_{n-1} + h$$

$$y_n = y_{n-1} + f(x_{n-1,} y_{n-1})h \qquad (3.5)$$

h es el paso del método.

Otra forma de interpretar este método consiste en invertir la aproximación de orden más bajo para la derivada, normalmente conocido como el método de Euler Mejorado.

$$f\left(x_n, y_n\right) = y_n' \approx \frac{y_{n+1} - y_n}{h} + O\left(h\right) \qquad (3.6)$$

por tanto se obtiene

$$y_{n+1} = y_n + hf\left(x_n, y_n\right) + O\left(h^2\right) \qquad (3.7)$$

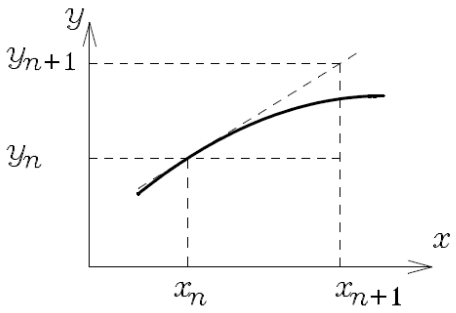
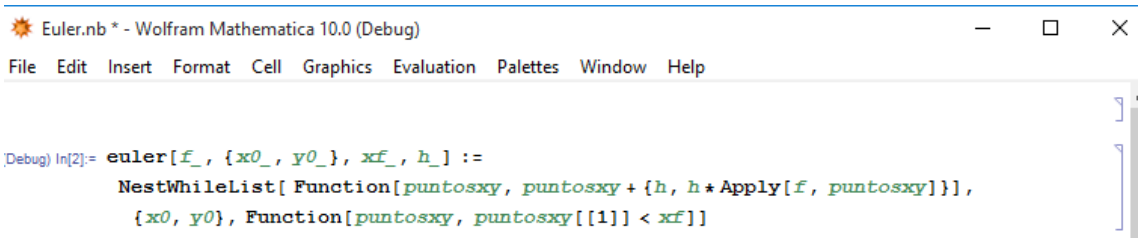


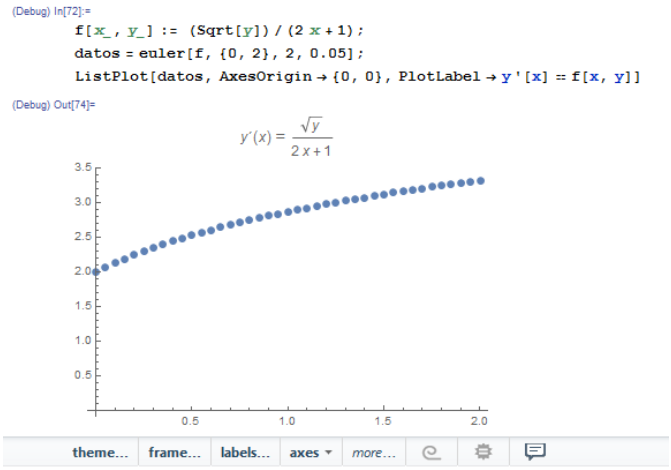
Figura 4.1 Método de Euler [1].

Como se observa en la figura 3.1, el método consiste en aproximar la solución entre x_n y x_{n+1} por el segmento tangente a la solución que pasa por $\left(x_n, y_n\right)$ [1].

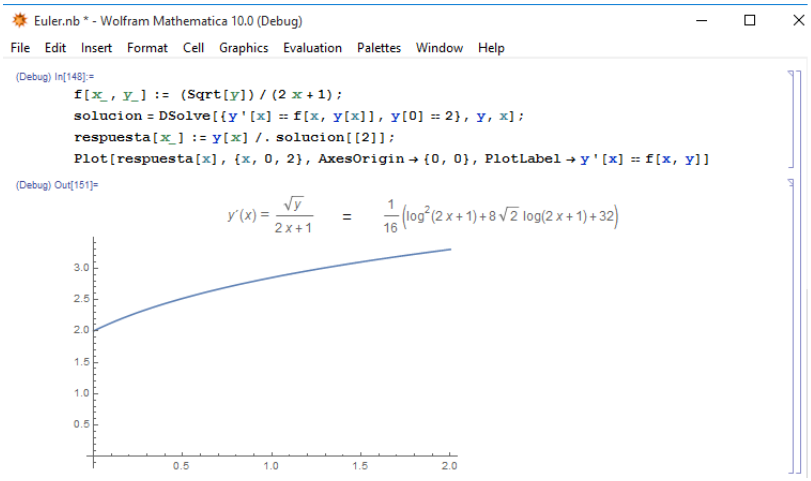
A continuación se presenta una función en Mathematica para resolver ecuaciones diferenciales por el método de Euler.



Se aplica la función creada anteriormente para calcular ecuaciones diferenciales, en este caso ingresamos la ecuación que se desee resolver y realizamos el respectivo llamado a la función.



También se puede utilizar el comando DSolve para calcular la solución:



3.2 Métodos Runge-Kutta.

Para la resolución numérica de ecuaciones diferenciales tenemos entre la variedad de métodos a los métodos de Runge-Kutta que son un conjunto de métodos desarrollados por Carl David Tolmé Runge y Martin Wilhelm Kutta. Carl Runge fue un físico matemático alemán nacido el 30 de agosto de 1856 y fallece el 3 de enero de 1927, en 1880 se graduó como Doctor en Matemática en Berlín. Martin Wilhelm Kutta un físico matemático alemán nacido el 3 de noviembre de 1867 y fallece el 25 de diciembre de 1944, fue un gran matemático alemán. Estos dos autores en 1901 desarrollan varios métodos de tipo Runge-Kutta para resolver ecuaciones diferenciales.

Los métodos de Runge-Kutta forman una clase importante de métodos para la integración de ecuaciones diferenciales. Una subclase especial, los métodos de colocación, permite un particular acceso elegante al orden, simplicidad y producción continua [15].

Los métodos Runge-Kutta sirven para la integración de ecuaciones diferenciales utilizando varias derivadas o tangentes intermedias.

Es importante mencionar el método de Euler, es un procedimiento de integración numérica que se utiliza para poder resolver ecuaciones diferenciales ordinarias a partir de un valor inicial dado (3.1), es decir, con el método de Euler se logra obtener una solución aproximada en un conjunto finito de puntos partiendo de la ecuación de la recta [8].

A partir de la ecuación de la recta que se utiliza el método de Euler se puede definir ecuaciones diferenciales de primer orden.

$$\dot{y} = f(t, y), y(t_0) = \mathrm{y}_0 \qquad (3.8)$$

La integración de esta ecuación da $y(t_1) = y_0 + \int_{t_0}^{t_1} f(t, y(t))dt$ y reemplazando la integral por la regla trapezoidal, se obtiene [4]:

$$y_1 = y_0 + \frac{h}{2}\left(f(t_0, y_0) + f(t_1, y_1)\right) \qquad (3.9)$$

A continuación se muestra la fórmula de Runge-Kutta de segundo orden, se presenta además la fórmula análoga para la regla del punto medio:

Punto pendiente

$$\begin{aligned} k_1 &= f(t_0, y_0) \\ k_2 &= f(t_0 + h, y_0 + hk_1) \\ y_1 &= y_0 + \frac{h}{2}(k_1 + k_2) \end{aligned} \qquad \begin{aligned} k_1 &= f(t_0, y_0) \\ k_2 &= f\left(t_0 + \frac{h}{2}, y_0 + \frac{h}{2}k_1\right) \\ y_1 &= y_0 + (hk_2) \end{aligned} \qquad (3.10)$$

Además se presenta la fórmula de Runge-Kutta de cuarto orden:

$$\begin{aligned} k_1 &= f(t_0, y_0) \\ k_2 &= f\left(t_0 + \frac{h}{2}, y_0 + \frac{h}{2}k_1\right) \\ k_3 &= f\left(t_0 + \frac{h}{2}, y_0 + \frac{h}{2}k_2\right) \\ k_4 &= f(t_0 + h, y_0 + hk_3) \\ y_1 &= y_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \qquad (3.11)$$

Para resultados más exactos se utiliza el método de Runge Kutta de quinto orden de butcher (1964), donde,

$$k_1 = f\left(x_i, y_i\right)$$

$$k_2 = f\left(x_i + \frac{1}{4}h, y_i + \frac{1}{4}k_1 h\right)$$

$$k_3 = f\left(x_i + \frac{1}{4}h, y_i + \frac{1}{8}k_1 h + \frac{1}{8}k_2 h\right)$$

$$k_4 = f\left(x_i + \frac{1}{2}h, y_i - \frac{1}{2}k_1 h + k_3 h\right)$$

$$k_5 = f\left(x_i + \frac{3}{4}h, y_i + \frac{3}{16}k_1 h + \frac{9}{16}k_4 h\right)$$

$$k_6 = f\left(x_i + h, y_i - \frac{3}{7}k_1 h + \frac{2}{7}k_2 h + \frac{12}{7}k_3 h - \frac{12}{7}k_4 h + \frac{8}{7}k_5 h\right)$$

$$y_{i+1} = y_i + \frac{1}{90}\left(7k_1 + 32k_3 + 12k_4 + 32k_5 + 7k_6\right)h$$

3.2.1 Formulación general.

$$\dot{x} = f(t, x)\ ,\ x(\mathrm{t}_0) = x_0\ ,\ x \in \mathbb{R}^d \qquad (3.12)$$

$$\begin{aligned} k_s &= f(t_n + c_s h, x_n + h(a_{s1}k_1 + \ldots + a_{s,s-1}k_{s-1})) \\ x_{n+1} &= x_n + h(b_1 k_1 + \ldots + b_s k_s) \end{aligned} \qquad (3.13)$$

Es posible considerar métodos aún más generales (implícitos) para el sistema diferencial ordinario del primer orden (3.12) dentro de esta familia simplemente permitiendo en la expresión (3.13), en otras palabras, la clase general de los métodos de Runge-Kutta en etapa s se caracteriza por los números reales [4]:

$a_{ij}, b_i (i, j = 1, \ldots, s)$ y $c_i = \sum_{j=1}^{s} a_{ij}$ como

$$k_i = f\left(t_n + c_i h, x_n + h\sum_{j=1}^{s} a_{ij}k_j\right), \qquad i = 1, \ldots, s \qquad (3.14)$$

$$x_{n+1} = x_n + h\sum_{i=1}^{s} b_i k_i$$

Para simplificar, los coeficientes asociados generalmente se muestran con el tablero de Butcher [4]:

$$\begin{array}{c|ccc} c_1 & a_{11} & \dots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \dots & a_{ss} \\ \hline & b_1 & \dots & b_s \end{array} \qquad (3.15)$$

$a_{ij} = 0,\ j \geq i$, entonces el método es explícito

Método de Heun.

$$x_{n+1} = x_n + \frac{h}{4}\left(f(t_n, x_n) + 3 f(\mathrm{t}_n + 2\mathrm{h}/3, x_n + (2h/3) f(\mathrm{t}_n, x_n))\right) \qquad (3.16)$$

Tenga en cuenta que el cálculo de un paso con un método RK explícito por lo tanto requiere s evaluaciones de la función f , con la notación (3.15), el método de Heun (3.16) y el método de Runge Kutta de cuarto orden, se puede expresar como:

$$\begin{array}{c|cc} 0 & & \\ \frac{2}{3} & \frac{2}{3} & \\ \hline & \frac{1}{4} & \frac{3}{4} \end{array}, \qquad \begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2} & & \\ \frac{1}{2} & 0 & \frac{1}{2} & \\ 1 & 0 & 0 & 1 \\ \hline & \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6} \end{array}, \qquad (3.17)$$

Si algún $a_{ij} \neq 0$, $j \geq i$, el esquema es implícito y requiere resolver numéricamente un sistema de ecuaciones no lineales en cada paso, si expresamos el método (3.14) en la forma alternativa se tiene:

$$Y_i = x_n + h\sum_{j=1}^{s} a_{ij} f(t_n + c_j h, Y_j), \qquad i = 1, \dots, s \qquad (3.18)$$

$$x_{n+1} = x_n + h\sum_{i=1}^{s} b_i f\left(t_n + c_i h, Y_i\right)$$

Y_i se determina resolviendo un sistema de ecuaciones no lineales sd de la forma:

$$y = X_n + hG(h, y) \qquad (3.19)$$

Donde $y=\left(Y_1,...,Y_s\right)^T$, $X_n=\left(x_n,...,x_n\right)^T\in\mathbb{R}^{sd}$ y G es una función que depende del método, un método estándar para obtener $Y_1,...,Y_s$ de (3.19) es aplicar la interacción del punto fijo:

$$y^{[j]}=X_n+hG\left(h,y^{[j-1]}\right),\quad j=1,2..... \tag{3.20}$$

x_{n+1} se calcula con la última formula de 3.18.

Se dice que usualmente los métodos Runge Kutta son de orden r para los problemas regulares (3.12), si el error local satisface

$$x_1-x\left(t_0+h\right)=\vartheta\left(h^{r+1}\right)\ \text{ con }\ h\to 0 \tag{3.21}$$

La orden puede verificarse calculando la expansión de la serie de Taylor de $x\left(t_0+h\right)$ y x_1 con $h=0$.

Al igualar los coeficientes de las potencias sucesivas de h en ambas expansiones, da como resultado el siguiente orden condiciones [4]:

para la orden 1 $\sum_{i=1}^{s}b_i=1$

para la orden 2 $\sum_{i=1}^{s}b_ic_i=1/2$

para la orden 3 $\sum_{i=1}^{s}b_ic_i^2=1/3$, $\sum_{i=1}^{s}b_ic_i^2=1/3$

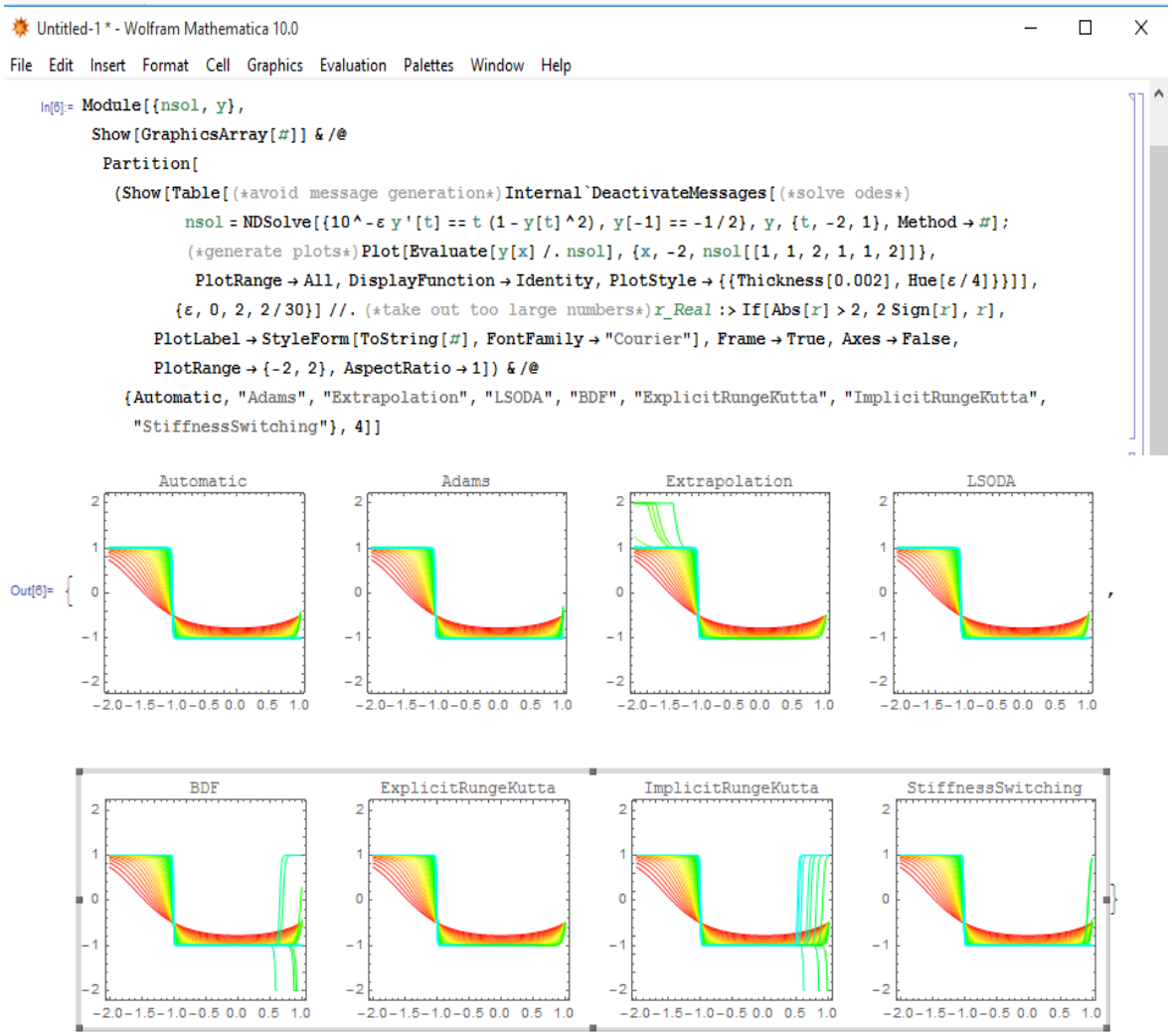
En Mathematica existen algunos métodos disponibles para trabajar con el método Runge-Kutta, a continuación se presenta una lista de estos y sus utilidades.

- Para sistemas grandes no rígidos.
  "Adams"
- Para sistemas rígidos.
  "BDF"
- Para obtener soluciones de problemas potencialmente rígidos.
  "LSODA"
- Para sistemas no rígidos.
  "ExplicitRungeKutta"
- Para sistemas rígidos.
  "ImplicitRungeKutta"
- Para obtener soluciones de muy alta precisión.
  "Extrapolation"

- Para obtener alta precisión de problemas potencialmente rígidos
  "StiffnessSwitching"
- Para problemas donde se mantiene invariantes importantes.
  "Projection"
- Para problemas que surgen de los sistemas hamiltonianos.
  "SymplecticRungeKutta"
- Para problemas de valores límite.
  "Chasing"
- Para sistemas algebraicos diferenciales.
  "IDA"

Para la mayoría de los propósitos, uno quiere usar la opción de método que establece Automático. En este modo, Mathematica alterna entre los diversos métodos de tal manera que la solución sea rápida y confiable al mismo tiempo [18]. A continuación se tiene el resultado al resolver $\varepsilon y'(t) = t\left(1 - y(t)^2\right)$.

Al analizar las gráficas se puede realizar una comparación de los diferentes métodos.



Se presenta ejemplos del comportamiento de los diversos métodos disponibles que funcionan en el siguiente sistema de Ecuaciones diferenciales de primer orden [39].

$$x_1'(t) = \frac{7}{5} x_2(t)^2 x_4(t)^3 - \frac{3}{4} x_2(t)$$
$$x_2'(t) = -\frac{81}{2} x_1(t) x_2(t)^2 x_4(t) x_3(t)^2 - \frac{1}{2} x_2(t)$$
$$x_3'(t) = 5 x_1(t) x_3(t) x_4(t)^3 + \frac{25}{36} x_3(t)$$
$$x_4'(t) = -\frac{16}{9} x_1(t) + \frac{78}{29} x_2(t) - \frac{38}{83} x_3(t)$$

Para resolver estas ecuaciones en Mathematica se debe primero ingresar las ecuaciones en una variable, en este caso la variables que se propone en el ejemplos es eqs.

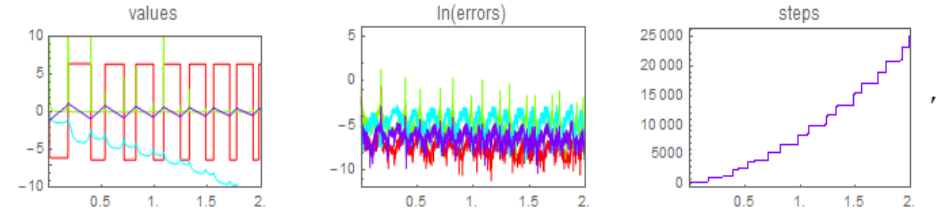
```
eqs = {x[1]'[t] == (-3/5 x[2][t] + 7/5 x[2][t]^2 x[4][t]^3),
    x[2]'[t] == (-1/2 x[2][t] - 81/2 x[1][t] x[2][t]^2 x[3][t]^2 x[4][t]),
    x[3]'[t] == (25/36 x[3][t] + 5 x[1][t] x[3][t] x[4][t]^3),
    x[4]'[t] == (-16/9 x[1][t] + 78/29 x[2][t] - 38/83 x[3][t])};
```

La función solveAndAnalyze resuelve el sistema utilizando el método y sus opciones de configuración y muestra un matriz de tres gráficos: las cuatro curvas de solución, el residuo y los valores t que se usaron en la integración (se recopila estos valores t utilizando la opción EvaluationMonitor) [39].

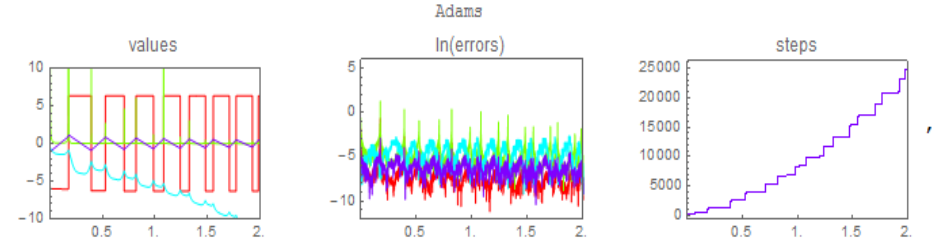
```
solveAndAnalyze[method_, otherNDSolveOptions___] :=
 Module[{tMax = 2, nsol, tValues, plotOpts}, (*solve ODEs and extract t-values used*)
  {nsol, tValues} = Internal`DeactivateMessages[
    MapAt[First, #, 2] &@
     Reap[NDSolve[Join[eqs, {x[1][0] == -2/17, x[2][0] == 92, x[3][0] == -1/2, x[4][0] == -9/7}],
       {x[1], x[2], x[3], x[4]}, {t, 0, tMax}, otherNDSolveOptions, Method → method,
       MaxSteps → 20000, EvaluationMonitor :→ Sow[t]]]];
(*a function for frequent use in the following*)
  plotOpts[label_, pr_] = Sequence[Frame → True, Axes → False, PlotLabel → label,
    PlotStyle → Table[{Thickness[0.002], Hue[x]}, {x, 0, 3/4, 1/4}],
    FrameTicks → {{0.5, 1., 1.5, 2.}, Automatic, None, None}, PlotRange → {{0, tMax}, pr}];
(*maximal time achieved for the numerical solution*)
  T = nsol[[1, 1, 2, 1, 1, 2]];
(*graphs of functions,logarithms of errors,and function evaluations*)
  GraphicsArray[Block[{$DisplayFunction = Identity},
    {Plot[Evaluate[{x[1][t], x[2][t], x[3][t], x[4][t]} /. nsol], {t, 0, T},
      Evaluate[plotOpts["values", {-10, 10}]]],
     Plot[Evaluate[Log[10, Abs[(Subtract @@@ eqs) /. nsol]]], {t, 0, T},
      Evaluate[plotOpts["ln(errors)", {-12, 6}]]],
     ListPlot[MapIndexed[{#1, #2[[1]]} &, Sort[Flatten[tValues]]],
      Evaluate[plotOpts["steps", All]]]}]]]
```

Out[7]=

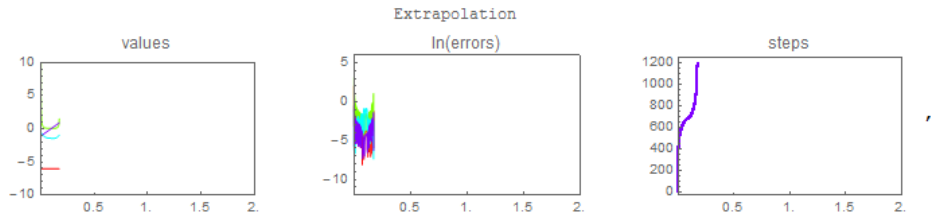
Automatic



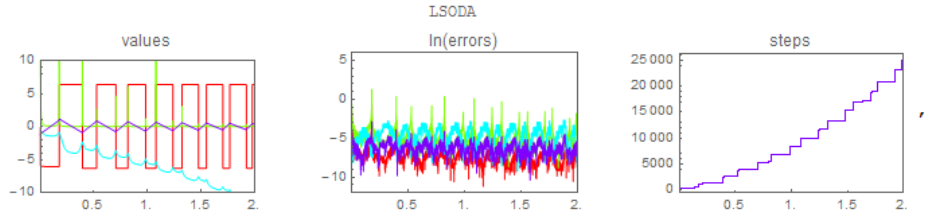
Adams



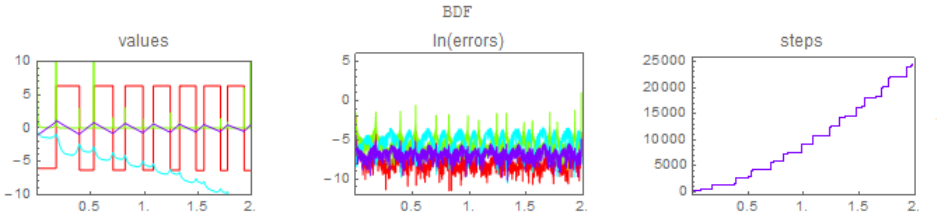
Extrapolation



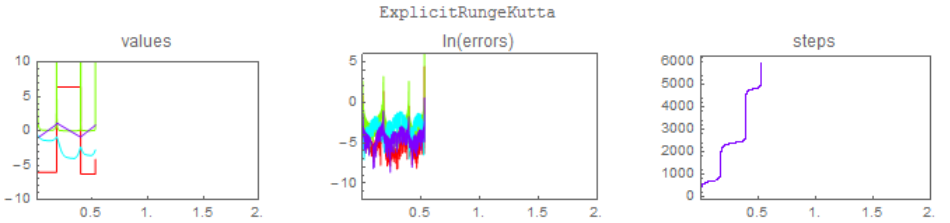
LSODA

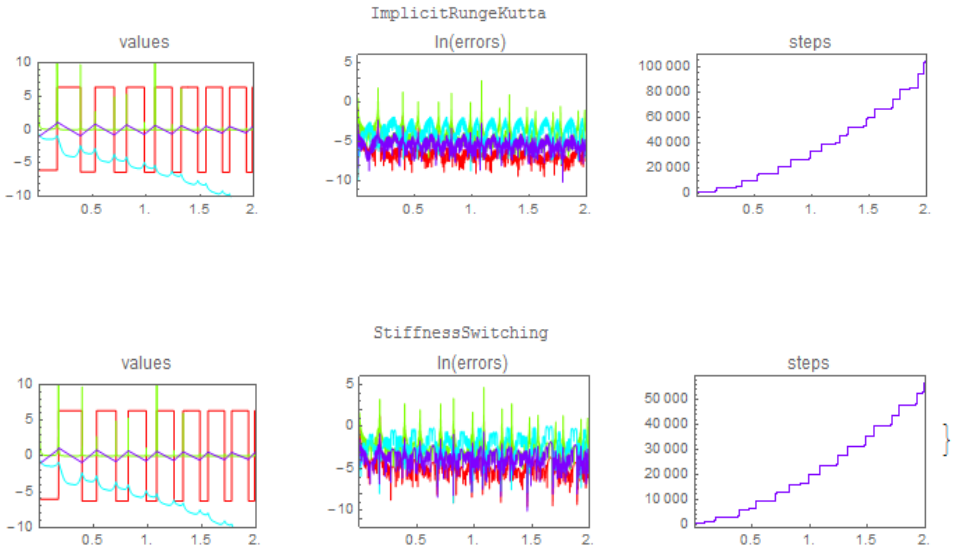


BDF



ExplicitRungeKutta





En el ejemplo se puede observar la utilización de los métodos de Runge Kutta, este nos da los resultados de los valores, el error y el paso utilizado.

Utilizando Extrapolación con un salto menor a 0.5 obtenemos valores aproximados a 1200 y el rango de error se encuentra entre [3,-8].

Utilizando LSODA con un salto en el rango de [0,2] obtenemos valores aproximados a 25000 y el rango de error se encuentra entre [2,-11].

Utilizando BDF con un salto en el rango de [0,2] obtenemos valores aproximados a 25000 y el rango de error se encuentra entre [0,-15] inclusive menores a -15.

Utilizando ExplicitRungeKutta con un salto en el rango de [0,0.6] obtenemos valores aproximados a 6000 y el rango de error se encuentra entre [5,-8].

Utilizando ImplicitRungeKutta con un salto en el rango de [0,2] obtenemos valores aproximados a 100000 y el rango de error se encuentra entre [4,-10].

Utilizando StiffnessSwitching con un salto en el rango inferiores a [0,2] obtenemos valores aproximados a 50000 y el rango de error se encuentra entre [5,-10].

Es importante tomar en cuenta los resultados del método ImplicitRungeKutta que tiene valores muy altos, también es importante mencionar que los métodos BDF tienen valores negativos y no positivos en el error producido.

3.3 Métodos de Multipaso

3.3.1 Una clase general de procedimientos de varios pasos

Conociendo las aproximaciones numéricas $x_n, x_{n+1}, ..., x_{n+k-1}$ a la solución exacta $x(t_n), ..., x(t_{n+k-1})$ de la ecuación diferencial (3.12) por lo tanto se puede calcular los valores $f_j \equiv f(t_j, x_j)$ y el polinomio de interpolación $f(t_j, x_j)$ para $j = n, ..., n+k-1$.

Explícitamente,

$$P_{k-1}(t) = \sum_{i=0}^{k-1} L_{k-1}, i(t) f_{n+i} \qquad (3.22)$$

donde

$$L_{k-1,i}(t) = \prod_{j=0, j\neq i}^{k-1} \frac{t - t_{n+j}}{t_{n+1} - t_{n+j}} \qquad (3.23)$$

son polinomios de interpolación de Lagrange [31].

Es sabido que $P_{k-1}(t_{n+i}) = f_{n+1}$ para $i = 0,1,...,k-1$. Entonces se puede avanzar desde t_{n+k-1} a t_{n+k}, como se detalla a continuación:

$$x(t_{n+k}) = x(t_{n+k-1}) + \int_{t_{n+k-1}}^{t_{n+k}} f(t, x(t))dt \qquad (3.24)$$

$$\simeq x_{n+k-1} + \int_{t_{n+k-1}}^{t_{n+k}} P_{k-1}(t)dt = x_{n+k-1} + \sum_{i=0}^{k-1} \alpha_{k-1,i} f_{n+i}$$

Donde $\alpha_{k-1}, i = \int_{t_{n+k-1}}^{t_{n+k}} L_{k-1,\mathrm{i}}(t)dt$

Para $k = 1$ recuperamos el método de Euler explícito, mientras que para $k = 2,3$ obtenemos

$$k = 2: x_{n+2} = x_{n+1} + \frac{h}{2}(3 f_{n+1} - f_n), \qquad (3.25)$$

$$k = 3: x_{n+3} = x_{n+2} + \frac{h}{24}(9 f_{n+3} + 19 f_{n+2} - 5 f_{n+1} + f_n). \qquad (3.26)$$

Dada una secuencia de puntos t_n con un tamaño de paso h, los métodos lineales de pasos múltiples son definidos por

$$\sum_{j=0}^{k} \alpha_j x_{n+j} = h \sum_{j=0}^{k} \beta_j f(t_{n+j}, x_{n+j}) \qquad (3.27)$$

donde los coeficientes α_j, β_j son constantes reales, $\alpha_k \neq 0$ y α_0, β_0 no son ambas igual a cero. Si $\beta_k = 0$, entonces x_{n+k} se obtiene explícitamente de los valores anteriores x_{n+j} y $f\left(t_{n+j}, x_{n+j}\right)$, entonces el método es explícito, de lo contrario el método es implícito.

Una caracterización algebraica equivalente se obtiene con los polinomios generadores de los coeficientes

$$\rho(z) = \sum_{j=0}^{k} \alpha_j z^j, \sigma(z) = \sum_{j=0}^{k} \beta_j z^j \qquad (3.28)$$

El método lineal de pasos múltiples (3.27) es cero-estable si y solo si todas las raíces de $\rho(z)$ verifican que $|z| \leq 1$, los método se llama estrictamente estable si todas las raíces están dentro del círculo unitario, excepto $z = 1$[30].

El método (3.27) es de orden r si, cuando se aplica con valores iniciales exactos a la ecuación $\dot{x} = t^a \left(0 \leq a \leq r\right)$ tenemos una solución exacta [18], esto es equivalente a

$$\rho\left(e^h\right) - h\sigma\left(e^h\right) = \vartheta\left(h^{r+1}\right) \text{ con } h \to 0 \qquad (3.29)$$

Si un método multipaso es estable a cero y de orden $r \geq 1$, entonces es convergente de orden r.

Para ecuaciones diferenciales particionadas $\dot{q} = f(q, p), \dot{p} = g(q, p)$, se puede aplicar diferentes métodos de varios pasos para los diferentes componentes,

$$\sum_{j=0}^{k} \alpha_j q_{n+j} = h\sum_{j=0}^{k} \beta_j f\left(q_{n+j}, p_{n+j}\right), \quad \sum_{j=0}^{\hat{k}} \hat{\alpha}_j p_{n+j} = h\sum_{j=0}^{\hat{k}} \hat{\beta}_j g\left(q_{n+j}, p_{n+j}\right) \qquad (3.30)$$

Análogamente, son de orden r si sus correspondientes polinomios generadores comprueba que $\rho\left(e^h\right) - h^2\sigma\left(e^h\right) = \vartheta\left(h^{r+2}\right)$ con $h \to 0$ y son estables si todas la raíces de $\rho(z)$ cumplen $|z| \leq 1$.

El método es estrictamente estable si todas las raíces están dentro del círculo unitario, excepto $z = 1$ [18] y los coeficientes satisfacen,

$$\alpha_{k-j} = \alpha_j, \qquad \beta_{k-j} = \beta_j, \qquad j = 0, ..., k$$

3.3.2 Métodos predictor–corrector

Estos métodos utilizan un método implícito, sin tener que resolver ecuaciones lineales implícitas, se considera el método multipaso lineal explícito:

$$\sum_{i=0}^{k} \alpha_i^* y_{n+1} = h\sum_{i=0}^{k-1} \beta_i^* f_{n+1,} \; n = 0 \; \ldots, N-k \;, \; k > 0, \; y_0, y_1, y_2 \ldots\ldots y_{k-1} \qquad (3.31)$$

y el implícito:

$$\sum_{i=0}^{k} \alpha_i \, y_{n+1} = h \sum_{i=0}^{k-1} \beta_i \; f_{n+1,} \; n = 0 \; \ldots, \; N-k \; , \; k > 0 , \; y_0, y_1, y_2 \ldots\ldots y_{k-1} \quad (3.32)$$

El predictor-corrector denominado PEC(predictor – evaluador -corrector) tiene la siguiente forma:
Predictor:

$$y_{n+k}^{[0]} = \frac{1}{\alpha_k^*} \left(h \sum_{i=0}^{k-1} \beta_i^* f_{n+i} - \sum_{i=0}^{k-1} \alpha_i^* y_{n+i} \right) \quad (3.33)$$

evaluador:

$$f\left(t_{n+k}, y_{n+k}^{[0]}\right) \quad (3.34)$$

corrector:

$$y_{n+k}^{[0]} = \frac{1}{\alpha_k^*} \left(h \beta_k f\left(t_{n+k}, y_{n+k}^{[0]}\right) + h \sum_{i=0}^{k-1} \beta_i f_{n+i} - \sum_{i=0}^{k-1} \alpha_i \, y_{n+i} \right) \quad (3.35)$$

Teorema. En un método PEC si el corrector es de orden p entonces es suficiente que el predictor sea de orden $p-1$ para que el PCE sea convergente de orden.

3.3.3. Métodos de Adams

Métodos de Adams explícito.

Consideramos P(s) un polinomio de grado k−1 que interpole las k pendientes calculadas, teniendo:

$$P\left(t_j\right) = f\left(t_j, y_i\right) = f_j, \; j = 0,1,2,3\ldots., k-1 \quad (3.36)$$

por tanto,

$$P(\mathrm{s}) = \sum_j f_j L_j\left(s\right) \in \mathrm{P}^{k-1} \quad (3.37)$$

Obtenemos entonces el valor y_k

$$y_k = y_{k-1} + \int_{t_{k-1}}^{t_k} \mathrm{P(s)} \, \mathrm{ds}$$

$$y_k = y_{k-1} + \sum_{j=0}^{k-1} c_j \, f\left(\mathrm{t}_j, y_j\right) \quad (3.38)$$

donde

$$c_j = \int_{t_{k-1}}^{t_k} \mathrm{L}_j(\mathrm{s})\,\mathrm{ds} \qquad (3.39)$$

Se puede ver que los coeficientes c_j no dependen del intervalo $[t_{k-1}, t_k]$, tienen la forma $c_j = h\bar{c}_j$ (cambio en la variable de integración al [0, 1]) una vez computados sirven para cualquier otro intervalo. A continuación se traslada el proceso un índice para así obtener y_{k+1}, y_{k+2} …., en este caso se genera lo que se conoce como métodos de Adams explícitos de k pasos y se describen por [4]:

Dados $y_0, y_1, ..., y_{k-1}$ se obtiene

$$y_{k+n} = y_{k+n-1} + h\sum_{j=0}^{k-1} c_j f(\mathrm{t}_{j+n}, y_{j+n}), n \geq 0 \qquad (3.40)$$

Métodos de Adams implícito.

Para $k \geq 1$ usamos los valores $y_0, y_1, \mathrm{y}_2 ..., y_{k-1}, y_k$, nótese que incluimos el valor que buscamos y utilizamos el polinomio $P(t) \in \mathrm{P}^k$ que interpole en los puntos t_j los valores f_j para $j = 0,1,2,3.....,k$, por tanto:

$$P(\mathrm{t}_j) = f(\mathrm{t}_j, \mathrm{y}_j), j = 0,1,2,3.....,k-1,k \qquad (3.41)$$

De aquí se genera los que se conoce como métodos de Adams implícito de $k+1$ pasos

$$y_k = y_{k-1} + \sum_{j=0}^{k} d_j f(\mathrm{t}_j, \mathrm{y}_j) \qquad (3.42)$$

Generamos para otros coeficientes diferentes

$$d_j = \int_{t_{k-1}}^{t_k} L_j(\mathrm{s})\,\mathrm{ds} \qquad (3.43)$$

Se puede ver que los coeficientes d_j no dependen del intervalo $[t_{k-1}, t_k]$, tienen la forma $d_j = h\bar{d}_j$ (cambio en la variable de integración al [0, 1]) y una vez computados sirven para cualquier otro intervalo. A continuación se traslada el proceso un índice para así obtener y_{k+1}, y_{k+2} ...., se describen por [4]:

Dados $y_0, y_1, ..., y_{k-1}$ se obtiene

$$y_{k+n} = y_{k+n-1} + h\sum_{j=0}^{k} c_j f(\mathrm{t}_{j+n}, y_{j+n}), n \geq 0 \qquad (3.44)$$

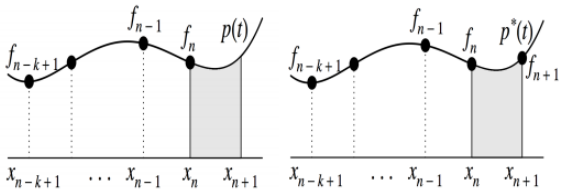


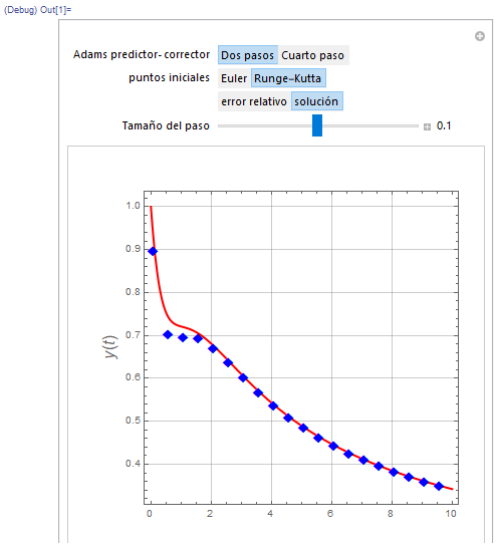
Figura 3.6: Interpretación geométrica en los métodos de Adams explícito e implícito.

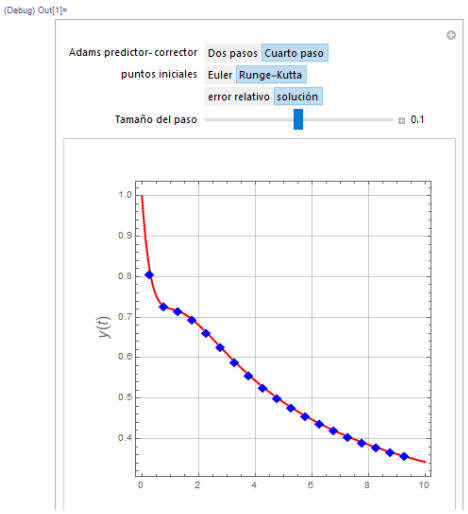
Se presenta un ejemplo del Método Predictor-Corrector Adams de dos pasos y cuatro pasos [19].

```
Manipulate[
 Module[{sol, p1, p2, y, i, t, f, tbl, tbl1, tbl2, tbl3, time, tbl4,
   yapc2, RungeKutta},
  is = .85 {450, 450};
  ip = {{60, 10}, {20, 10}};
  RungeKutta[f_, t_, h_, y_, yp_] := Block[{deltay, k1, k2, k3, k4},
    k1 = yp;
    k2 = f[t + 1 / 2 h, y + 1 / 2 h k1];
    k3 = f[t + 1 / 2 h, y + 1 / 2 h k2];
    k4 = f[t + h, y + h k3];
    deltay = h (1 / 6 k1 + 1 / 3 k2 + 1 / 3 k3 + 1 / 6 k4);
    deltay];
  Switch[ctrl2, 1,
   sol = NDSolve[{y'[t] == -y[t]^2 + t / (t^2 + 1), y[0] == 1}, y[t],
     {t, -0.3, 10}];
   p1 = Plot[y[t] /. sol, {t, 0, 10}, Frame → True,
     FrameLabel → {Style["t", Italic, 16],
       Row[{Style["y", Italic, 16], Style["(", 16], Style["t", Italic, 16],
         Style[")", 16]}]}, GridLines → Automatic, AspectRatio → 1,
     PlotStyle → {Thick, Red}, ImagePadding → ip, ImageSize → is];
   i = 1;


   f[t_, y_] := -y^2 + t / (t^2 + 1);
   t[0] = -h; t[1] = 0;
   yapc2[1] = 1;
   Switch[ctrl3, 1, yapc2[0] = yapc2[1] - h f[t[1], yapc2[1]], 2,
    yapc2[0] = yapc2[1] + RungeKutta[f, t[1], -h, yapc2[1],
       f[t[1], yapc2[1]]]];
   While[i ≤ 10 / h, {t[i + 1] = t[i] + h,
     yapc2[i + 1] =
      yapc2[i] +
       1 / 2 h
        (f[t[i + 1], yapc2[i] + 3 / 2 h f[t[i], yapc2[i]] -
            1 / 2 h f[t[i], yapc2[i]]] + f[t[i - 1], yapc2[i - 1]]), i++}];
   tbl = Table[{t[j], yapc2[j]},
     {j, 2, IntegerPart[10 / h], IntegerPart[10 / (20 h)]}];
   p2 = ListPlot[tbl, PlotMarkers → Style["•", 16, Blue],
     ImagePadding → ip, ImageSize → is];


    tbl1 = Table[yapc2[j], {j, 2, IntegerPart[10 / h],
       IntegerPart[10 / (20 h)]}];
    tbl2 = Table[First[y[t] /. sol /. t → t[j]],
      {j, 2, IntegerPart[10 / h], IntegerPart[10 / (20 h)]}];
    tbl3 = Quiet@Drop[(tbl1 - tbl2) / tbl2, 1];
    time = Table[t[j], {j, 2, IntegerPart[10 / h], IntegerPart[10 / (20 h)]}];
    tbl4 = Table[{time[[j]], Abs@tbl3[[j]]}, {j, 2, Length[time] - 1, 1}];
    Switch[
     ctrl, 1,
     ListLogPlot[tbl4, PlotRange → All, PlotMarkers → Style["•", 16, Blue],
      PlotRange → {{0, 10}, Automatic}, Frame → True, AspectRatio → 1,
      GridLines → Automatic,
      FrameLabel → {Style["t", Italic, 16], Style["error relativo", 16]},
      PlotLabel → Row[{Style["10^4 norma Euclidiana:  ", Red, 16],
         Style[10^4 Sqrt[Sum[Abs[(tbl1 - tbl2)[[i]]]^2,
             {i, 1, Length[time], 1}]], Red, 16]}], ImagePadding → ip,
      ImageSize → is], 2, Show[p1, p2]], 2,
    sol = NDSolve[{y'[t] == -y[t]^2 + t / (t^2 + 1), y[0] == 1}, y[t],
      {t, -0.7, 10}];
```

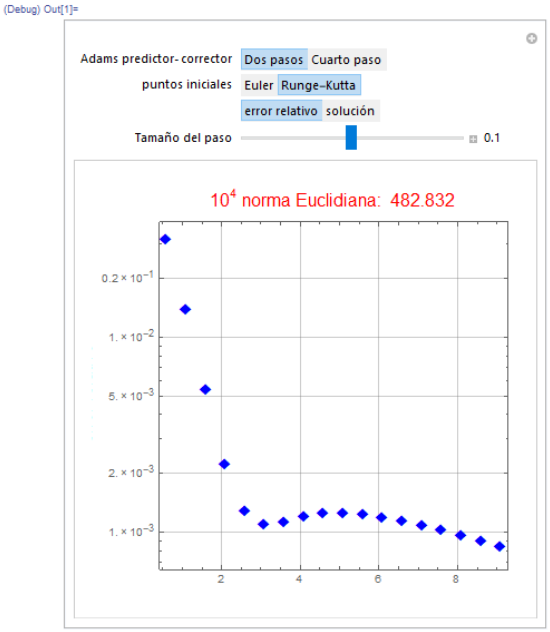
```
p1 = Plot[y[t] /. sol, {t, 0, 10}, Frame → True,
  FrameLabel → {Style["t", Italic, 16],
    Row[{Style["y", Italic, 16], Style["(", 16], Style["t", Italic, 16],
      Style[")", 16]}]}, GridLines → Automatic, AspectRatio → 1,
  PlotStyle → {Thick, Red}, ImagePadding → ip, ImageSize → is];
i = 3; t[0] = -3 h; t[1] = -2 h; t[2] = -h; t[3] = 0;
f[t_, y_] := -y^2 + t/(t^2 + 1);
yapc2[3] = 1;
Switch[ctrl3, 1, yapc2[2] = yapc2[3] - h f[t[3], yapc2[3]];
 yapc2[1] = yapc2[2] - h f[t[2], yapc2[2]];
 yapc2[0] = yapc2[1] - h f[t[1], yapc2[1]], 2,
 yapc2[2] = yapc2[3] + RungeKutta[f, t[3], -h, yapc2[3],
    f[t[3], yapc2[3]]];
 yapc2[1] = yapc2[2] + RungeKutta[f, t[2], -h, yapc2[2],
    f[t[2], yapc2[2]]];
 yapc2[0] = yapc2[1] + RungeKutta[f, t[1], -h, yapc2[1],
    f[t[1], yapc2[1]]]];


While[i ≤ 10/h, {t[i + 1] = t[i] + h,
  yapc2[i + 1] =
   yapc2[i] +
    1/24 h
     (9 f[t[i + 1], yapc2[i] + 1/24 h (55 f[t[i], yapc2[i]] -
              59 f[t[i - 1], yapc2[i - 1]]
              + 37 f[t[i - 2], yapc2[i - 2]] - 9 f[t[i - 3], yapc2[i - 3]])] +
       19 f[t[i], yapc2[i]] - 5 f[t[i - 1], yapc2[i - 1]] +
       f[t[i - 2], yapc2[i - 2]]), i++}];
tbl = Table[{t[j], yapc2[j]},
  {j, 1, IntegerPart[10/h], IntegerPart[10/(20 h)]}];
p2 = ListPlot[tbl, PlotMarkers → Style["•", 16, Blue],
  ImagePadding → ip, ImageSize → is];
tbl1 = Table[yapc2[j], {j, 3, IntegerPart[10/h],
   IntegerPart[10/(20 h)]}];
tbl2 = Table[First[y[t] /. sol /. t → t[j]],
  {j, 3, IntegerPart[10/h], IntegerPart[10/(20 h)]}];
tbl3 = Quiet@Drop[(tbl1 - tbl2)/tbl2, 1];
time = Table[t[j], {j, 3, IntegerPart[10/h], IntegerPart[10/(20 h)]}];
tbl4 = Table[{time[[j]], Abs@tbl3[[j]]}, {j, 1, Length[time] - 1, 1}];


Switch[ctrl, 1,
 ListLogPlot[tbl4, PlotRange → All, PlotMarkers → Style["•", 16, Blue],
  PlotRange → {{0, 10}, Automatic}, Frame → True, AspectRatio → 1,
  GridLines → Automatic,
  FrameLabel → {Style["t", Italic, 16], Style["error relativo", 16]},
  PlotLabel → Row[{Style["10^4 norma Euclidiana:  ", Red, 16],
     Style[10^4 Sqrt[Sum[Abs[(tbl1 - tbl2)[[i]]]^2,
         {i, 2, Length[time] - 1, 1}]], Red, 16]}], ImagePadding → ip,
  ImageSize → is], 2, Show[p1, p2]]],
{{ctrl2, 2, "Adams predictor-corrector"},
 {1 → "Dos pasos", 2 → "Cuarto paso"}},
{{ctrl3, 2, "puntos iniciales"}, {1 → "Euler", 2 → "Runge-Kutta"}},
{{ctrl, 1, ""}, {1 → "error relativo", 2 → "solución"}},
{{h, 0.1, "Tamaño del paso"}, 0.0005, 0.2, 0.0001,
 Appearance → "Labeled"}]
```

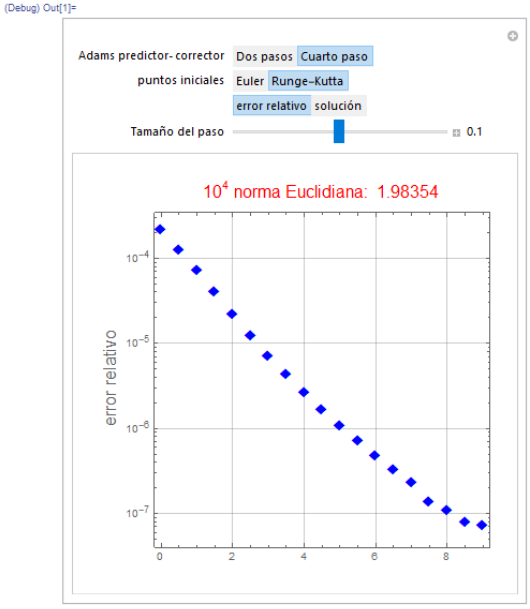
En la salida se puede observar la solución del método Adams Predictor—corrector de dos pasos y cuatro pasos, utilizando puntos iniciales con Runge-Kutta:





También se puede observar los errores relativos.





En el ejemplo anterior se plantea un problema $y'(t) = f(x,t) = -y^2 + t/(t^2+1)$ con valor inicial $y(0) = 1$, se calcula el error relativo que presentan por el método Adams, este resultado se calcula con la comparación del resultado obtenido entre el método Adams y la utilización del comando NDSolve.

Se desarrolla el método predictor-corrector Adams de dos pasos, para la resolución de este método procedemos a realizar: paso predictor (dos pasos Adams-Bashforth); paso corrector (dos pasos Adams–Moulton).

También se utiliza el método predictor-corrector Adams de cuatro pasos, este usa los métodos de cuatro pasos de los métodos Adams-Bashforth y Adams–Moulton.

Los métodos Adams de dos y cuatro pasos requieren dos y cuatro valores iniciales para comenzar el cálculo, respectivamente, para calcular los valores iniciales utilizamos el método Euler y Runge-Kutta de cuarto orden.

Analizando los resultados del ejemplo planteado con un paso de 0.1 en el Adams predictor corrector de dos pasos obtenemos mayor precisión al utilizar los métodos de Euler (método para calcular los valores iniciales), utilizando Range-Kutta obtenemos un error mayor.

Con un paso de 0.1 en el método de Adams predictor corrector de cuatro pasos y utilizando el método de Runge-Kutta obtenemos una precisión mayor al método de Euler.

Claramente se puede observar que el método de Adams predictor corrector de cuatro pasos y utilizando el método Runge-Kutta es el método con mayor precisión en el ejemplo planteado.

3.3.4 Métodos Adams-Bashforth

Estos métodos utilizan la identidad $y(x_{n+1}) - y(x_n) = \int_{x_n}^{x_{n+1}} f(\xi, y(\xi))\,\mathrm{d}\xi$, aquí se tiene el valor $f(x, y(x))$ como incógnita, por lo que el valor debe ser aproximado, por lo cual definimos la expresión de la siguiente manera:

$$y(x_{n+1}) - y(x_n) = \int_{x_n}^{x_{n+1}} f(\xi, y(\xi))\,\mathrm{d}\xi \approx \int_{x_n}^{x_{n+1}} P(x)dx \qquad (3.45)$$

En la expresión anterior tenemos que $\mathrm{P}(x)$ es el polinomio interpolador de $f(x, y(x))$ en los puntos $\{x_{n-q}, ..., x_n\}$, entonces:

$$y(x_{n+1}) - y(x_n) = \int_{x_n}^{x_{n+1}} P(x)dx = h\sum_{i=0}^{q} \gamma_i \nabla^i f_n \qquad (3.46)$$

Para calcular γ_i se debe realizar el siguiente proceso:

$$\gamma_i = (-1)^i \frac{1}{h} \int_{x_n}^{x_{n+1}} \binom{-s}{i} dx = (-1)^i \int_0^1 \binom{-s}{i} ds \qquad (3.47)$$

Para calcular los coeficientes de γ_i se utiliza la función generatriz $G(t) = -\dfrac{t}{(1-\mathrm{t})\log(1-\mathrm{t})}$, teniendo,

$$G(\mathrm{t})=\sum_{n=0}^{\infty}\gamma_n t^n=\sum_{n=0}^{\infty}(-t)^n\int_0^1\binom{-s}{n}ds=\int_0^1\sum_{n=0}^{\infty}(-\mathrm{t})^n\binom{-s}{n}ds=\int_0^1(1-\mathrm{t})^{-s}ds=\frac{-t}{(1-\mathrm{t})\log(1-\mathrm{t})} \quad (3.48)$$

Considerando que $|t|<1$ se calcula los coeficientes de $G(t)$ en un entorno del 0, de esta forma se hallan de forma recursiva los γ_i , que resultan ser [5]

i	0	1	2	3	4	5	6
γ_i	1	$\frac{1}{2}$	$\frac{5}{12}$	$\frac{3}{8}$	$\frac{251}{720}$	$\frac{95}{288}$	$\frac{19087}{60480}$
i	7	8	9	10	11	12	
γ_i	$\frac{5257}{17280}$	$\frac{1070017}{3628800}$	$\frac{25713}{89600}$	$\frac{26842253}{95800320}$	$\frac{4777223}{17418240}$	$\frac{703604254357}{703604254357}$	

También podemos expresar la ecuación (3.46) de la siguiente manera, $y_{n+1}-y_n=h\sum_{i=0}^{q}\gamma_{q,i}f_{n-i}$, donde ahora el subíndice de $\gamma_{q,i}$ se explica porque los coeficientes dependen tanto de q como de i , la siguiente tabla nos presenta los valores de dichos coeficientes con q variando entre 0 y 5 [19].

i	0	1	2	3	4	5
$\gamma_{0,i}$	1	0	0	0	0	0
$\gamma_{1,i}$	$\frac{3}{2}$	$\frac{-1}{2}$	0	0	0	0
$\gamma_{2,i}$	$\frac{23}{12}$	$\frac{-16}{12}$	$\frac{5}{12}$	0	0	0
$\gamma_{3,i}$	$\frac{55}{24}$	$\frac{-59}{24}$	$\frac{37}{24}$	$\frac{-9}{24}$	0	0
$\gamma_{4,i}$	$\frac{1901}{720}$	$\frac{-2774}{720}$	$\frac{2616}{720}$	$\frac{-1274}{720}$	$\frac{251}{720}$	0
$\gamma_{5,i}$	$\frac{4227}{1440}$	$\frac{-7673}{1440}$	$\frac{9482}{1440}$	$\frac{-6798}{1440}$	$\frac{2627}{1440}$	$\frac{-425}{1440}$

Se plantea un problema de valor inicial.

$y'(x)=\sin(x)$, $\quad y(0)=5$

El problema se desarrolla en Mathematica utilizando el método Adams-Bashforth de orden 2, con χ variando de 0 a 3 y con una longitud de paso h =0.1, h =0.01 y h =0.001

```
(Debug) In[114]:=
h = 0.1;
x1 = 0;
xf = 3;
niter = (xf - x1) / h;
x0 = x1 - h;
y1 = 5;
y0 = 6 - Cos[x0];
f[x_, y_] := Sin[x];
g[x_] := -Cos[x] + 6;
For[j = 1, j ≤ niter, j++,
 y2 = y1 + h * (3 / 2 * f[x1, y1] - 1 / 2 * f[x0, y0]);
 y0 = y1;
 y1 = y2;
 x0 = x1;
 x1 = x1 + h;
 Print[x1];
 Print[N[y1, 15]];
 Print[N[y1 - g[x1]]];]
```

Los datos que se obtiene al ejecutar el programa con h=0.1 se puede observar en la siguiente tabla:

x_n	**0.1**	**0.2**	**0.3**	**1.**	**2.**	**3.**
y_n	5.00499	5.01997	5.04478	5.4614	6.42181	6.99823
error	-4.16389×10^{-6}	0.0000332612	0.000111901	0.00170132	0.00566148	0.00823953

La tabla con h=0.01 es la siguiente:

x_n	**0.01**	**0.02**	**0.03**	**1.**	**2.**	**3.**
y_n	5.00005	5.0002	5.00045	5.4597	6.4162056	6.99008
error	-4.16664×10^{-10}	-3.33326×10^{-9}	-1.12494×10^{-8}	0.0000189433	0.00005878	0.00008288

La tabla con h=0.001 es la siguiente:

x_n	**0.001**	**0.002**	**0.003**	**1.**	**2.**	**3.**
y_n	5.	5.	5.	5.45967	6.41615	6.98999
error	-4.08562×10^{-14}	3.33955×10^{-13}	1.12532×10^{-12}	1.9133×10^{-7}	5.898×10^{-7}	8.2913×10^{-7}

3.3.5 Métodos Adams-Moulton.

Este método utiliza los puntos $\{x_{n-q},...,x_n\}$ en el polinomio interpolador, por lo cual se debe integrar en los puntos $\{x_{n-1},...,x_n\}$

$$y(x_n) - y(x_{n-1}) = \int_{x_{n-1}}^{x_n} f(\xi, y(\xi))\,\mathrm{d}\,\xi \qquad (3.49)$$

por lo tanto tenemos,

$$y_n - y_{n-1} = \int_{x_{n-1}}^{x_n} P(x)dx = h\sum_{i=0}^{q} \gamma_i^* \nabla^i f_n \qquad (3.50)$$

luego

$$\gamma_i^* = (-1)^i \frac{1}{h}\int_{x_{n-1}}^{x_n} \binom{-s}{i} dx = (-1)^i \int_{-1}^{0} \binom{-s}{n} ds \qquad (3.51)$$

Se calcula la función generatriz obteniendo los γ_i^*, dando como resultado la siguiente expresión.

$$G(t) = \sum_{n=0}^{\infty} \gamma_n^* t^n = \sum_{n=0}^{\infty} (-\mathrm{t})^n \int_{-1}^{0} \binom{-s}{n} ds = \int_{-1}^{0} \sum_{n=0}^{\infty} (-t)^n \binom{-s}{n} ds = \int_{-1}^{0} (1-\mathrm{t})^{-s} ds = \frac{-t}{\log(1-\mathrm{t})} \qquad (3.52)$$

y se pueden hallar de forma recursiva los γ_i^*, que resultan ser

i	0	1	2	3	4	5	6
γ_i^*	1	$\frac{-1}{2}$	$\frac{-1}{12}$	$\frac{-1}{24}$	$\frac{-19}{720}$	$\frac{-3}{160}$	$\frac{-863}{60480}$
i	7	8	9	10	11	12	
γ_i^*	$\frac{-275}{24192}$	$\frac{-33953}{3628800}$	$\frac{-8183}{1036800}$	$\frac{-3250433}{479001600}$	$\frac{-4671}{788480}$	$\frac{-13695779093}{2615348736000}$	

Se realiza el mismo problema que se desarrolló con el método Adams-Bashforth, ahora se va a realizar con el método Adams-Moulton de 2 pasos, de igual forma con χ variando de 0 a 3 y con una longitud de paso h=0.1, h=0.01 y h=0.001

```
(Debug) In[71]:=
h = 0.1;
x1 = 0;
xf = 3;
niter = (xf - x1) / h;
x0 = x1 - h;
y1 = 5;
y0 = 6 - Cos[x0];
f[x_, y_] := Sin[x];
g[x_] := -Cos[x] + 6;
For[j = 1, j ≤ 30, j++,
 y3 = y1 + h * (3 / 2 * f[x1, y1] - 1 / 2 * f[x0, y0]);
 y2 = y1 + h * (f[x1 + h, y3] + f[x1, y1]) / 2;
 y0 = y1;
 y1 = y2;
  x0 = x1;
 x1 = x1 + h;
 Print[x1];
 Print[N[y1, 15]];
 Print[N[y1 - g[x1]]];]
```

Al resolver con h=0.1 nos da la siguiente tabla:

x_n	**0.1**	**0.2**	**0.3**	**1.**	**2.**	**3.**
y_n	5.00499	5.01992	5.04463	5.45931	6.41497	6.988334
error	-4.16389×10^{-6}	-0.000016614	-0.0000372258	0.000383145	-0.00118032	-0.0016586

Al resolver con h=0.01 nos da la siguiente tabla:

x_n	**0.01**	**0.02**	**0.03**	**1.**	**2.**	**3.**
y_n	5.00005	5.0002	5.00045	5.45969	6.416135	6.9899759
error	$-4.16664 * 10^{-10}$	-1.66661×10^{-9}	-3.74972×10^{-9}	-3.83082×10^{-6}	-0.000012	-0.00001658

y por último al resolver con h=0.001 nos da la siguiente tabla:

x_n	**0.001**	**0.002**	**0.003**	**1.**	**2.**	**3.**
y_n	5.	5.	5.	5.4597	6.416147	6.98999
error	$-4.08562 * 10^{-14}$	-1.66089×10^{-13}	-3.74811×10^{-13}	-3.83081×10^{-8}	-1.18012×10^{-7}	-1.65833×10^{-7}

Al comparar los métodos Adams-Bashforth y Adams-Moulton con h=0.01 y con $x = 1$.

En el método Adams-Bashforth tenemos el siguiente resultado:

$x_n = 1.$
$y_n = 5.4597$
$error = 0.0000189433$

y con el método Adams-Moulton tenemos:

$x_n = 1.$
$y_n = 5.45969$
$error = -3.83082 \times 10^{-6}$

Podemos observar claramente que el método Adams-Moulton tiene un error inferior al calculado en el método Adams-Bashforth.

3.3.6 Método de diferenciación regresiva.

El proceso que se aplica en este método es muy similar al de los métodos Adams, se establece tres pasos para la aplicación de este método.

a. Primero se procede a identificar la nube de puntos y su polinomio interpolador de Newton en diferencias regresivas.
   Nube de puntos:

$$\left(t_n, y\left(t_n\right)\right),.....,\left(t_{n+k}, y\left(t_{n+k}\right)\right)$$

   Polinomio de grado k:

$$Q_{k,n}(t) = \sum_{i=0}^{k} (-1)^i \binom{-s}{i} \nabla^i y(\mathrm{t}_{n+k}) \qquad (3.53)$$

$$s = \frac{t - t_{n+k}}{h}, k \geq 1$$

b. Teniendo $y'\left(t\right) = f\left(t, y\left(t\right)\right)$ en t_{n+k} , se cambia $y'\left(t\right)$ por el polinomio interpolador $Q_{k,n}\left(t\right)$.

$$\left.\frac{dQ_{k+n}\left(t\right)}{dt}\right|_{t=t_{n+k}} \approx y'\left(t_{n+k}\right) = f\left(t_{n+k}, y\left(t_{n+k}\right)\right) \qquad (3.54)$$

   Al final tenemos:

$$\frac{1}{h}\sum_{i=1}^{k} \frac{\nabla^i y\left(t_{n+k}\right)}{i} \approx f\left(t_{n+k}, y\left(t_{n+k}\right)\right) \qquad (3.55)$$

c. Para obtener métodos de diferencias regresivas de k paso, se debe cambiar $y\left(t_{n+k}\right)$ por $y\left(t_{n+k}\right)$ y $f\left(t_{n+k}, y\left(t_{n+k}\right)\right)$ por $f_{n+k} = f\left(t_{n+k}, y_{n+k}\right)$. (3.56)

$$\sum_{i=1}^{k} \frac{\nabla^i y_{n+k}}{i} = h f_{n+k}$$

$$y_0, \ldots y_{k-1}$$

Teorema. Los métodos de diferencias regresivas de k pasos son convergentes si y sólo si $k \in \{1, 2, \ldots, 6\}$.

3.4 Métodos de escisión (splitting) y composición.

3.4.1 Métodos Splitting

Los métodos splitting integran ecuaciones diferenciales ordinarias (EDOs), formulando de la siguiente manera, dado el valor inicial del problema.

$$x' = f(x) \qquad x_0 = x(0) \in \mathbb{R}^D \qquad (3.57)$$

con $f : \mathbb{R}^D \to \mathbb{R}^D$ y solución $\vartheta_t(x_0)$, también f puede ser expresado como $f = \sum_{i=1}^{m} f^{[i]}$ entonces se puede definir a $f^{[i]} : \mathbb{R}^D \to \mathbb{R}^D$, de tal manera las ecuaciones se plantarán de la siguiente manera:

$$x' = f^{[i]}(x), \qquad x_0 = x(0) \in \mathbb{R}^D, \qquad i = 1, \ldots, m \qquad (3.58)$$

se puede integrar con soluciones $x(h) = \vartheta_h^i(0)$ a $t = h$

entonces se combina las soluciones, como:

$$X_h = \vartheta_h^{[m]} \circ \cdots \circ \vartheta_h^{[2]} \circ \vartheta_h^{[1]} \qquad (3.59)$$

y aplicando la serie de Taylor a X, se encuentra $X_h(x_0) = \vartheta_h(x_0) + \vartheta(h^2)$.

X_h proporciona una aproximación de primer orden a la solución exacta.

Las principales ventajas que poseen los métodos de splitting [5]:

- Por lo general, son fáciles de implementar.
- En general son explícitos.
- El algoritmo es secuencial y las soluciones en etapas intermedias se almacenan en los vectores de solución. Esta propiedad puede ser de gran interés cuando son aplicados a ecuaciones diferenciales parciales (EDP) previamente semidiscretizadas.
- Existe en la literatura una gran cantidad de métodos específicos adaptados para diferentes estructuras.
- Conservan las propiedades estructurales de la solución exacta, lo que confiere al esquema numérico una superioridad cualitativa con respecto a otros integradores estándar, especialmente cuando los intervalos de tiempo largos son considerados. Algunos ejemplos de estas características estructurales son la simplecticidad, preservación de volumen, simetría de tiempo y conservación de primeras

integrales. En este sentido, los métodos de splitting constituyen una clase importante de geometría integradores numéricos.

Es un método numérico de integración basadas en particiones temporales, una diferencia entre el método Runge-Kutta y Splitting, "la idea de Splitting produce un enfoque que es completamente diferente al método Runge-Kutta. Uno descompone el campo vectorial en piezas integrables y las trata por separado." [17]

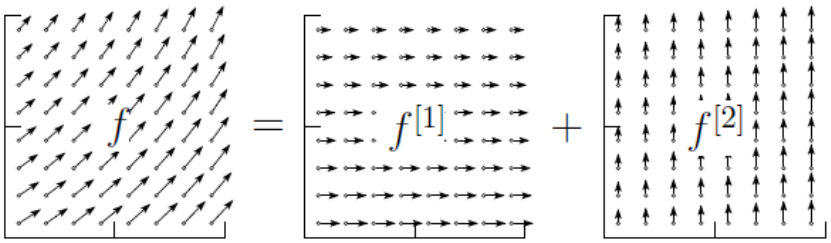


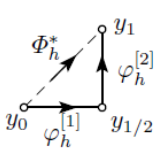
Figura 3.1. Splitting de un campo vectorial
Fuente: Ernst Hairer

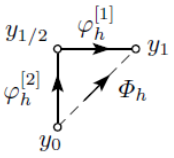
Si se tiene un sistema $\dot{x} = f_0(x) + f_1(x)$, $x(0) = x_0$ donde f_0, f_1 son campos de $\mathbb{R}^n$, en la figura 3.1 se observa el campo vectorial de Splitting correspondiente a $\dot{x}$.

Entonces se platea de forma alternada los flujos $\phi_t^{[1]}$ y $\phi_t^{[2]}$ del sistema f_0 y f_1 que se pueden calcular explícitamente, se puede dar valores iniciales y_0, resolvemos el primer sistema para obtener un valor $y_{1/2}$ y a partir de este valor integrar el segundo sistema para obtener y_1.

Se utiliza fórmulas del llamado método de splitting de Lie-Trotter [40].

$$\Phi_h^* = \varphi_h^2 \circ \varphi_h^1$$
$$\Phi_h = \varphi_h^1 \circ \varphi_h^2$$





También se puede utilizar otra nomenclatura, como se presenta a continuación:

$$\Phi_L^{(j)}(h) = \varphi_{1-j}(h) \circ \varphi_j(h)$$
$$\Phi_S^j(h) = \varphi_j(h/2) \circ \varphi_j(h/2) \qquad (3.60)$$

Por la expansión de Taylor se tiene $\left(\varphi_h^{[1]} \circ \varphi_h^{[2]}\right)(y_0) = \varphi_h(y_0) + O(h^2)$, estas fórmulas mencionadas son utilizadas en ecuaciones de orden 1.

Para métodos de orden 2 se debe utilizar las fórmulas de Strang Splitting $\varphi_h^{[2]} = \varphi_{h/2}^{[2]} \circ \varphi_{h/2}^{[2]}$

Se analiza dos ejemplos [19] de la utilización y aplicación de los métodos splitting.

Ejemplo 3.4.1 Simpléctico de Euler y método leapfrog.

Supongamos que tenemos un sistema hamiltoniano de la forma $H(q,p) = T(p) + V(q)$, donde $q \in \mathbb{R}^d$ son las coordenadas canónicas $p \in \mathbb{R}^d$ son los momentos conjugados, T representa la energía cinética y V es la energía potencial. Entonces las ecuaciones de movimiento leen [4]

$$q' = T_p(p), \qquad p' = -V_q(q) \qquad (3.61)$$

donde T_p y V_q denota los vectores de las derivadas parciales, la ecuación (3.61) pueden ser formuladas con (3.57) como: $x = (q,p)^T \quad f(x) = (T_p, -V_q)^T = J\nabla H(x)$ y $D = 2d$.

Aquí J la matriz canonical symplectic, denotado por $2d \times 2d$.

$$J = \begin{pmatrix} 0 & I_d \\ -I_d & 0 \end{pmatrix}$$

Id representa la matriz de identidad d-dimensional. En este caso, el flujo ϑ_t es simpléctico [25]. El método de Euler aplicado a este sistema proporciona la siguiente aproximación de primer orden para un paso de tiempo h:

$$\begin{aligned} q_{n+1} &= q_n + hT_p(p_n) \qquad (3.62) \\ p_{n+1} &= p_n - hV_q(q_n) \end{aligned}$$

Si consideramos H como la suma de dos hamiltonianos, el primero depende solo de p y el segundo solo de q, ecuaciones:

$q' = T_p(p)$ y $q' = 0$

$p' = 0$ y $p' = -V_q(q)$

con la condición inicial (q_0, p_0), puede ser resuelto fácilmente.

$$\vartheta_t^{[T]} : \begin{matrix} q(t) = q_0 + tT_p(P_0) \\ p(t) \quad\quad = p_0 \end{matrix} \qquad (3.63)$$

y

$$\vartheta_t^{[V]}: \begin{matrix} q(t) & = q_0 \\ p(t) & = p_0 - tV_q(q_0) \end{matrix} \qquad (3.64)$$

teniendo el tiempo $t = h$ por lo tanto se tiene $\vartheta_h^{[V]}$, con las condiciones iniciales (q_n, p_n) y con $\vartheta_h^{[T]}$, se tiene el siguiente esquema:

$$X \equiv \vartheta_h^{[T]} \circ \vartheta_h^{[V]}: \begin{matrix} p_{n+1} = p_n - hV_q(q_n) \\ q_{n+1} + hT_p(p_{n+1}) \end{matrix} \qquad (3.65)$$

Es posible composición de los mapas en el orden opuesto, $\vartheta_h^{[V]} \circ \vartheta_h^{[T]}$ obteniendo así otro esquema de Euler simpléctico de primer orden:

$$X_h^* \equiv \vartheta_h^{[V]} \circ \vartheta_h^{[T]}: \begin{matrix} q_{n+1} = q_n + hT_p(p_n) \\ p_{n+1} = p_n - hV_q(q_{n+1}) \end{matrix} \qquad (3.66)$$

Ejemplo 4.4.2 Oscilador armónico.

En este ejemplo se considera la función hamiltoniano $H(q, p) = \frac{1}{2}\left(p^2 + q^2\right)$, donde $q, p \in \mathbb{R}$, en donde las ecuaciones (3.66) son lineales y se pueden escribir como:

$$x' = \begin{pmatrix} q' \\ p' \end{pmatrix} = \left[\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ -1 & 0 \end{pmatrix} \right] \begin{pmatrix} q \\ p \end{pmatrix} = (A + B)x$$

teniendo

$$A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \quad \text{y} \quad B = \begin{pmatrix} 0 & 0 \\ -1 & 0 \end{pmatrix}$$

La solución numérica obtenida por el esquema de Euler es:

$$\begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & h \\ -h & 1 \end{pmatrix} \begin{pmatrix} q_n \\ p_n \end{pmatrix}$$

Mientras que el método simplépico de Euler conduce a:

$$\begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & h \\ -h & 1-h^2 \end{pmatrix} \begin{pmatrix} q_n \\ p_n \end{pmatrix} = e^{hB} e^{hA} \begin{pmatrix} q_n \\ p_n \end{pmatrix}$$

Las dos son aproximaciones de primer orden a la solución exacta, y se puede expresar como $x(t)=e^{h(A+B)}x_0$, la aproximación obtenida por el esquema de Euler simpléctico, verifica

$$\frac{1}{2}\left(p_{n+1}^2+hp_n+{}_1q_{n+1}+q_n^2+1\right)=\frac{1}{2}\left(p_n^2+hp_nq_n+q_n^2\right) \qquad (3.67)$$

Se puede demostrar que la solución exacta en $t=h$ de la perturbada del sistema hamiltoniano

$$\tilde{H}(q,p,h)=\frac{2\arcsin(h/2)}{h\sqrt{4-h^2}}\left(p^2+hpq+q^2\right) \qquad (3.68)$$

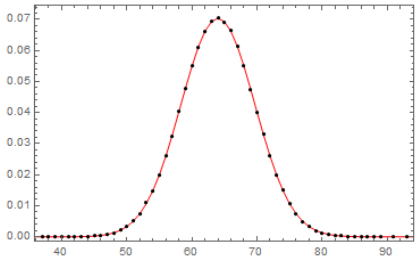
$$=\frac{1}{2}\left(p^2+q^2\right)+h\left(\frac{1}{2}pq+\frac{1}{12}h\left(p^2+q^2\right)+\cdots\right)$$

Es solo de primer orden para las trayectorias exactas del hamiltoniano, es la solución exacta del hamiltoniano perturbado

Se presenta un ejemplo desarrollado en Mathematica y con la aplicación de la función pileSplittingFunc esta genera una construcción Si anidada que, para un número aleatorio dado entre 0 y 1 y devuelve el tamaño de una de las dos pilas.

```
In[128]:= pileSplittingFunction[n_] :=
  pileSplittingFunction[n] = Module[{▯, if, half}, (*cumulative splitting probabilities*)
    MapIndexed[(▯[[#2[[1]] - 1]] = #1) &, FoldList[Plus, 0., Table[Binomial[n, k], {k, 0, n}]] / 2^n];
    (*approximately half a list*)half[{x_?NumberQ}] := x;
    half[{x_?NumberQ, y_?NumberQ}] := if[ξ < p[y], x, y];
    half[l_List] :=
     With[{v = Ceiling[Length[l] / 2]}, (*recursive calls to half*)if[ξ < p[l[[v + 1]]], half@Take[l, v], half@Take[l, {v + 1, Length[l]}]]] /.
      (*make nested If*)p → ▯ /. if → If;
    (*compile nested If construct*)Compile @@ {{ξ}, half[Range[0, n]]}]

In[134]:= pileSplittingFunction[8]

Out[134]= CompiledFunction[ ▯ | Argument count: 1 | Argument types: {_Real} ]

In[136]:= With[{n = 128, o = 10^6}, ListPlot[{First[#], Length[#] / o} & /@ Split[Sort[Table[pileSplittingFunction[n][Random[]], {o}]]],
    PlotStyle → {GrayLevel[0], PointSize[0.01]}, PlotRange → All, Frame → True, Axes → False, (*theoretical distribution*)
    Prolog → {Hue[0], Thickness[0.002], Line[Table[{k, Binomial[n, k] / 2^n}, {k, 0, n}]]}]]
```

Out[136]=



3.4.2 Métodos de composición

Los métodos de composición aplican la fórmula de BCH al producto de exponenciales de los campos de vectores, esto conduce a [4]:

$$\log\left(\Psi\left(h\right)\right)=F\left(h\right)=hw_1Y_1+h^2w_2Y_2+h^3\left(w_3Y_3+w_{12}\left[Y_1,Y_2\right]\right)$$
$$+h^4\left(w_4Y_4+w_{13}\left[Y_1,Y_3\right]+w_{112}\left[Y_1,\left[Y_1,Y_2\right]\right]\right)+\mathrm{O}\left(h^5\right) \qquad (3.69)$$

donde $w_{j_1.....j_m}$ son polinomios de grado $n=j_1+...+j_m$ con los parámetros $\alpha_1,.....,\alpha_{2_s}$.

Los primeros polinomios son:

$$w_1=\sum_{i=1}^{2s}\alpha, \qquad w_2=\sum_{i=1}^{2s}\left(-1\right)^i\alpha_i^2, \qquad w_3=\sum_{i=1}^{2s}\alpha_i^3 \qquad (3.70)$$

$$w_{12}=\sum_{j_2=1}^{2s}\left(-1\right)^{j_2}\alpha_{j_2}^2\sum_{j_1=1}^{j_2^*}\alpha_{j_1}-w_3-w_1w_2 \qquad w_4=\sum_{j=1}^{2s}\left(-1\right)^j\alpha_i^4$$

Se tiene que $j_2^*=j_2-1$ y $j_2^*=j_2$ si j es impar.

Un método de composición de orden 3 debe satisfacer $w_1=1,w_2=w_3=w_{12}=0$, el método simétrico de cuarto orden tiene que satisfacer $w_1=1,w_3=w_{12}=0$, teniendo las condiciones, $w_2=w_4=w_{13}=w_{112}=0$.

3.4.3 Integradores y series de operadores diferenciales.

Vamos a relacionar el número genérico integradores con series formales de ecuaciones diferenciales. Esta relación permite formular de una manera bastante simple las condiciones que debe cumplir un esquema de integración para lograr un orden de consistencia dado. Aumentar el orden de un integrador por composición.

Primero damos a conocer el integrador $\psi_n:\mathbb{R}^D\to\mathbb{R}^D$ para el sistema $x'=f\left(x\right)$, $x_0=x\left(0\right)\in\mathbb{R}^D$, se dice que es de orden r si para todo $x\in\mathbb{R}^D$

$$\psi_h\left(x\right)=\varphi_h\left(x\right)+\vartheta\left(h^{r+1}\right) \qquad (3.71)$$

Se sabe que, para cualquier función $g:\mathbb{R}^D \to \mathbb{R}$, formalmente contiene [33]

$$g\left(\varphi_h(x)\right) = g(x) + \sum_{n\geq 1} \frac{1}{n!} F^n[\mathrm{g}](x) = \exp(hF)[\mathrm{g}](x) \qquad (3.72)$$

Para cada $g \in C^\infty\left(\mathbb{R}^D, \mathbb{R}\right)$ y cada $x = (x_1, ..., x_D) \in \mathbb{R}^D$

$$F[\mathrm{g}](x) = \sum_{j=1}^{D} f_j(x) \frac{\partial g}{\partial x_j}(x) \qquad (3.73)$$

donde $f(x) = \left(f_1(x), ..., f_D(x)\right)^T$, consideramos que un integrador básico $X_h : \mathbb{R}^D \to \mathbb{R}^D$, los operadores diferenciales lineales $X_n (n \geq 1)$ actuando sobre funciones suaves $g \in C^\infty\left(\mathbb{R}^D, \mathbb{R}\right)$, como se muestra a continuación:

$$X_n[\mathrm{g}](x) = \frac{1}{n!} \frac{d^n}{dh^n} g\left(X_h(x)\right)|_{h=0} \qquad (3.74)$$

Para que $g\left(X_h(x)\right) = X(h)[g](x)$ donde

$$X(h) = I + \sum_{n\geq 1} h^n X_n \qquad (3.75)$$

y I denota el operador de identidad, por lo tanto, el integrador X_h es de orden r si:

$$X_n = \frac{1}{n!} F^n \ , \qquad 1 \leq n \leq r$$

También se puede considerar la serie de campos vectoriales

$$Y(h) = \sum_{n\geq 1} h^n Y_n = \log\left(X(h)\right) = \sum_{m\geq 1} \frac{(-1)^{m+1}}{m} \left(hX_1 + h^2 X_2 + \cdots\right)^m \qquad (3.76)$$

es decir

$$Y_n = \sum_{m\geq 1}^{n} \frac{(-1)^{m+1}}{m} \sum_{j_1 + \cdots + j_m = n} X_{j_1} \cdots X_{j_m} \qquad (3.77)$$

así que $X(h)=\exp\big(Y(h)\big)$ y formalmente $g\big(X_h(x)\big)=\exp\big(Y(h)\big)[g](x)$, el integrador básico es de orden r si

$Y_1=F\,,\quad Y_n=0\quad$ para $\quad 2\le n\le r$

Consideremos la serie de operadores diferenciales

$$F^{[2k]}(h)=hF+h^{2k+1}F_{2k+1}^{[2k]}+h^{2k+3}F_{2k+3}^{[2k]}+\cdots \qquad (3.78)$$

tal que $g\big(\delta_h^{2k}(x)\big)=\exp\Big(F^{[2k]}(h)\Big)[g](x)$, entonces se tiene

$$\exp\Big(F^{[2k+2]}(h)\Big)=\exp\Big(F^{[2k]}(\alpha h)\Big)\exp\Big(F^{[2k]}(\beta h)\Big)\exp\Big(F^{[2k]}(\alpha h)\Big) \qquad (3.79)$$

lo que implica

$$F^{[2k+2]}(h)=h(2\alpha+\beta)F+h^{2k+1}\left(2\alpha^{2k+1}+\beta^{2k+1}\right)F_{2k+1}^{[2k]}+\vartheta\left(h^{2k+3}\right) \qquad (3.80)$$

así S_h^{2k+2} es de orden $2k+2$ siempre que S_h^{2k} es de orden $2k$ y α y β satisfagan las ecuaciones

$2\alpha+\beta=1,\quad 2\alpha^{2k+1}+\beta^{2k+1}=0$

cuya única solución viene dada por

$$\alpha=\frac{1}{2-2^{1/(2k+1)}}\,,\qquad \beta=1-2\alpha \qquad (3.81)$$

Para el método de composición $\psi_h=X_{\alpha 2sh}\circ X^*_{\alpha 2s-1h}\circ\cdots\circ X_{\alpha 2h}\circ X^*_{\alpha 1h}$, tenemos

$$g\big(\psi_h(x)\big)=\Psi(h)[g]x \qquad (3.82)$$

donde $\Psi(h)=I+h\Psi_1+h^2\Psi_2+\cdots$ es una serie de operadores diferenciales que satisfacen

$$\Psi(h)=X(-\alpha h)^{-1}X(\alpha_2 h)\cdots X(-\alpha_{2s-1}h)^{-1}X(\alpha_{2sh}) \qquad (3.83)$$

donde la serie $X(h)$ es dada por (3.74)- (3.75) :

$$X(h)^{-1} = I + \sum_{m\geq 1} (-1)^{m+1} \left(hX_1 + h^2 X_2 + \cdots \right)^m \qquad (3.84)$$

Teorema. El integrador $\psi_h = \vartheta^{[b]}_{b_{s+1}h} \circ \vartheta^{[a]}_{a_s h} \circ \vartheta^{[a]}_{b_s h} \circ \cdots \circ \vartheta^{[b]}_{b_2 h} \circ \vartheta^{[a]}_{a_1 h} \circ \vartheta^{[b]}_{b_1 h}$ es de orden r para EDO de la forma (3.57) donde $f : \mathbb{R}^d \to \mathbb{R}^d$ es un Split $f = f^{[1]} + f^{[2]}$ si y solo si el integrador $f = f^{[a]} + f^{[b]}$ y $X_h = \vartheta^b_h \circ \vartheta^a_h$ (con coeficientes α_j obtenidos de $a_j = \alpha_{2j-1} + \alpha_{2j}$, $b_{j+1} = \alpha_{2j} + \alpha_{2j+1}$ es de orden r para arbitrario integradores consistentes.

3.4.4 NDSolve con los métodos de Splitting y Composición.

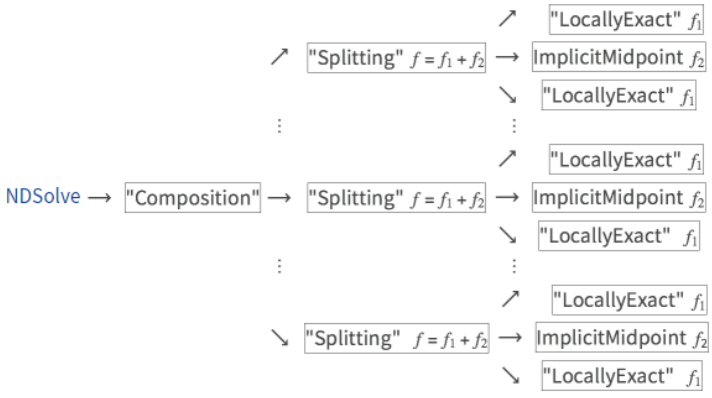
Opciones del método de composición.

Nombre de la opción option name	**Valor por defecto default value**	
"Coefficients"	Automatic	Especifica los coeficientes a usar en el método de composición
"DifferenceOrder"	Automatic	Especifica el orden de precisión local del método
Method	None	Especifica los métodos base para usar en la integración numérica

Opciones del método de splitting.

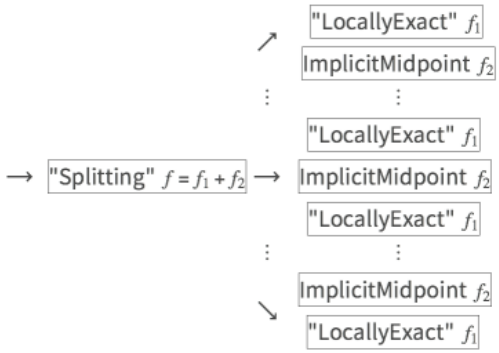
Nombre de la opción option name	**Valor por defecto default value**	
"Coefficients"	{}	Especifica los coeficientes a usar en el método de splitting
"DifferenceOrder"	Automatic	Especifica el orden de precisión local del método
"Equations"	{}	Especifica la forma en que las ecuaciones deberían dividirse
Method	None	Especifica los métodos base para usar en la integración numérica

Esquema del método de splitting de alto orden a partir de una división de bajo orden usando "Composición".



Esquema NDSolve [23]

Se puede obtener un integrador más eficiente en el esquema anterior usando la propiedad grupal de flujos y llamando directamente al método "Splitting".



Esquema Splitting [23]

Los siguientes ejemplos utilizarán un splitting simétrica de segundo orden conocida como Strang splitting [31], [37].

Cargamos el paquete necesario,

```
(Debug) In[1]:= Needs["DifferentialEquations`NDSolveProblems`"];
```

Symplectic Leapfrog.

"SymplecticPartitionedRungeKutta" es un método eficiente para resolver sistemas hamiltonianos separables $H(\mathrm{p},\mathrm{q}) = \mathrm{T}(\mathrm{p}) + \mathrm{V}(\mathrm{q})$

"Splitting" es un método de propósito más general, pero se puede usar para construir métodos simplécticos particionados

Consideremos el sistema hamiltoniano separable correspondiente al oscilador armónico $H(\mathrm{p},\mathrm{q}) = \frac{p^2}{2} + \frac{q^2}{2}$

```
(Debug) In[59]:=
system = GetNDSolveProblem["HarmonicOscillator"]
(Debug) Out[59]=
NDSolveProblem[{{Y1'[T] == Y2[T], Y2'[T] == -Y1[T]}, {Y1[0] == 1, Y2[0] == 0},
  {Y1[T], Y2[T]}, {T, 0, 10}, {}, {1/2 (Y1[T]^2 + Y2[T]^2)}, {}}]
```

Se divide el campo vectorial hamiltoniano en componentes independientes.

```
(Debug) In[60]:=
eqs = system["System"];
Y1 = eqs;
Part[Y1, 1, 2] = 0;
Y2 = eqs;
Part[Y2, 2, 2] = 0;
```

Su composición de pasos de integración de Euler ponderados (primer orden) corresponde al método leapfrog.

```
(Debug) In[65]:=
tfinal = 1;
time = {T, 0, tfinal};
qvars = {Subscript[Y, 1][T]};
splittingsol = NDSolve[system, time, StartingStepSize → 1 / 10,
  Method → {"Splitting", "DifferenceOrder" → 2, "Equations" → {Y1, Y2, Y1},
    "Method" → {"ExplicitEuler", "ExplicitEuler", "ExplicitEuler"}}]
(Debug) Out[68]=
{{Y1[T] → InterpolatingFunction[ Domain: {{0., 1.}} Output: scalar ][T],
 Y2[T] → InterpolatingFunction[ Domain: {{0., 1.}} Output: scalar ][T]}}
```

Este es el resultado al final del paso de integración.

```
(Debug) In[69]:=
InputForm[splittingsol /. T → tfinal]
Assuming a list of rules | Use as a two-dimensional array instead
apply rules to expr... | apply rules to variables
(Debug) Out[69]//InputForm=
{{Subscript[Y, 1][1] -> 0.5399512509335085, Subscript[Y, 2][1] ->
   -0.8406435124348495}}
```

Aplicando el método de integración incorporado correspondiente al integrador leapfrog simpléctico.

```
(Debug) In[70]:=
sprksol = NDSolve[system, time, StartingStepSize → 1 / 10, Method → SymplecticLeapfrog]
(Debug) Out[70]=
{{Y1[T] → InterpolatingFunction[ Domain: {{0., 1.}} Output: scalar ][T],
 Y2[T] → InterpolatingFunction[ Domain: {{0., 1.}} Output: scalar ][T]}}
```

El resultado al final del paso de integración es idéntico al resultado del método de splitting.

```
(Debug) In[71]:=
InputForm[sprksol /. T → tfinal]
Assuming a list of rules | Use as a two-dimensional array instead
apply rules to expr... | apply rules to variables
(Debug) Out[71]//InputForm=
{{Subscript[Y, 1][1] -> 0.5399512509335085, Subscript[Y, 2][1] ->
   -0.8406435124348495}}
```

Composición de leapfrog simpléctico.

Esto toma el esquema de Symplectic Leapfrog como el método de integración de base y construye un integrador simpléctico de cuarto orden usando una composición simétrica de Ruth-Yoshida [41]

```
(Debug) In[72]:=
YoshidaCoefficients =
  RootReduce[{1 / (2 - 2^(1/3)), -2^(1/3) / (2 - 2^(1/3)), 1 / (2 - 2^(1/3))}];
YoshidaCompositionCoefficients[4, p_] := N[YoshidaCoefficients, p];
splittingsol = NDSolve[system, time, StartingStepSize → 1/10,
  Method → {"Composition", "Coefficients" → YoshidaCompositionCoefficients,
    "DifferenceOrder" → 4, "Method" → {SymplecticLeapfrog}}]
```

```
(Debug) Out[74]=
{{Y1[T] → InterpolatingFunction[ Domain: {{0., 1.}} Output: scalar ][T],
 Y2[T] → InterpolatingFunction[ Domain: {{0., 1.}} Output: scalar ][T]}}
```

El resultado al final del paso de integración es:

```
(Debug) In[75]:=
InputForm[splittingsol /. T → tfinal]
```

Assuming a list of rules | Use as *a two-dimensional array* instead

apply rules to expr... | apply rules to variables

```
(Debug) Out[75]//InputForm=
{{Subscript[Y, 1][1] -> 0.5403078808898406, Subscript[Y, 2][1] ->
   -0.8414706295697821}}
```

Realiza el llamado al método de integración simpléctica que usa coeficientes para los métodos de cuarto orden de Ruth y Yoshida.

```
(Debug) In[76]:=
SPRK4[4, prec_] :=
  N[{{Root[-1 + 12 * #1 - 48 * #1^2 + 48 * #1^3 &, 1, 0],
     Root[1 - 24 * #1^2 + 48 * #1^3 &, 1, 0], Root[1 - 24 * #1^2 + 48 * #1^3 &, 1, 0],
     Root[-1 + 12 * #1 - 48 * #1^2 + 48 * #1^3 &, 1, 0]},
    {Root[-1 + 6 * #1 - 12 * #1^2 + 6 * #1^3 &, 1, 0],
     Root[1 - 3 * #1 + 3 * #1^2 + 3 * #1^3 &, 1, 0],
     Root[-1 + 6 * #1 - 12 * #1^2 + 6 * #1^3 &, 1, 0], 0}}, prec];
sprksol = NDSolve[system, time, StartingStepSize → 1/10,
  Method → {"SymplecticPartitionedRungeKutta", "Coefficients" → SPRK4,
    "DifferenceOrder" → 4, "PositionVariables" → qvars}]
```

```
(Debug) Out[77]=
{{Y1[T] → InterpolatingFunction[ Domain: {{0., 1.}} Output: scalar ][T],
 Y2[T] → InterpolatingFunction[ Domain: {{0., 1.}} Output: scalar ][T]}}
```

Como se observa el resultado al final del paso de integración es idéntico al resultado del método de composición.

```
(Debug) In[78]:=
InputForm[sprksol /. T → tfinal]
```

Assuming a list of rules | Use as *a two-dimensional array* instead

apply rules to expr... | apply rules to variables

```
(Debug) Out[78]//InputForm=
{{Subscript[Y, 1][1] -> 0.5403078808898405, Subscript[Y, 2][1] ->
   -0.8414706295697822}}
```

Capítulo 4. Modelo Lotka-Volterra y los métodos numéricos.

Las ecuaciones de Lotka-Volterra, son un par de ecuaciones diferenciales de primer orden no lineales que se usan para describir dinámicas de sistemas biológicos en el que dos especies interactúan, una como presa y otra como depredador, el sistema se expresa de la siguiente forma:

$$\frac{dx}{dt} = x(a - by)$$

$$\frac{dy}{dt} = y(-c + dx) \qquad (4.1)$$

donde:

y número de predadores ;

x número de presas;

dy/dt y dx/dt representa la variación de las dos poblaciones en el tiempo;

t representa el tiempo; y

a, b, c, d son parámetros que representan las interacciones de las dos especies.

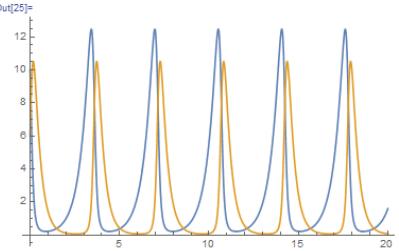
Una de las características demostrados por el modelo Lotka-Volterra es que, bajo ciertas condiciones, las poblaciones de depredadores y presas son cíclicas con un cambio de fase entre ellas. Aquí hay una demostración de este efecto [25].

```
(Debug) In[24]:=
lv = With[{a = 2., b = 1., c = 3., d = 1.},
  With[{x0 = 10., y0 = 6.},
   NDSolve[{x'[t] == a x[t] - b x[t] y[t], y'[t] == -c y[t] + d x[t] y[t], x[0] == x0, y[0] == y0},
    {x[t], y[t]}, {t, 0, 20}]]]

Plot[Evaluate[{x[t], y[t]} /. lv], {t, 0, 20}]
```

(Debug) Out[24]=

```
{{x[t] → InterpolatingFunction[ ⊞ Domain: {{0., 20.}} Output: scalar ][t],
 y[t] → InterpolatingFunction[ ⊞ Domain: {{0., 20.}} Output: scalar ][t]}}
```

(Debug) Out[25]=



Para el sistema (4.1) tomamos valores concretos para a, b, c, d , teniendo:

$$\begin{aligned} \dot{u} &= u(v-2) \\ \dot{v} &= v(1-u) \end{aligned} \qquad (4.2)$$

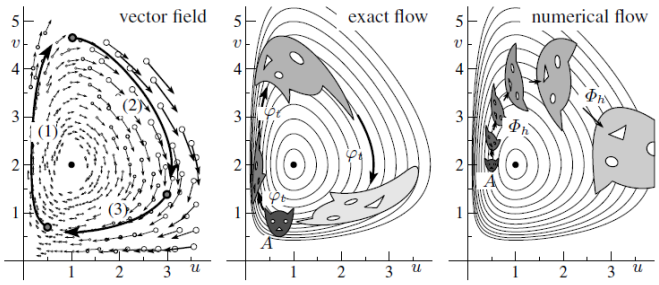


Figura 3.2. Modelo de Lotka-Volterra
Fuente: Ernst Hairer

Las ecuaciones (4.2) constituyen un sistema autónomo de ecuaciones diferenciales. Escribimos el sistema en la forma

$$\dot{y} = f(y) \qquad (4.3)$$

Cada y representa un punto en el espacio de fase, la función vectorial $f(y)$ representa un campo vectorial que, en cualquier punto del espacio de fase, prescribe la velocidad (dirección y velocidad) de la solución $y(t)$ que pasa por ese punto.

Si dividimos las dos ecuaciones de (4.2) entre sí, obtenemos una sola ecuación, entre las variables u y v. Después de la separación de las variables obtenemos [17]

$$0 = \frac{1-u}{u}\dot{u} - \frac{v-2}{v}\dot{v} = \frac{d}{dt} I(u,v) \qquad (4.4)$$

donde

$$I(u,v) = \log u - u + 2\log v - v \qquad (4.5)$$

Llamamos a I una invariante del sistemas (4.2), a que tomar en consideración que $I(u(t), v(t))$ es igual a una constante para todo t.

Realizando el cambio de variables $u = e^q$, $v = e^p$, transformamos este problema a un sistema Hamiltoniano separable $H(q,p) = I(u(q), v(p)) = p - e^p + q^2 - e^q$.

La solución del sistema Lotka-Volterra se desarrolla en Mathematica, para esto lo primero que realizamos es ingresar las ecuaciones (4.2) y declarar unas constantes que tienen los siguientes valores: las ecuaciones, invariantes, el tiempo, las variables dependientes y el valor de paso $\left(h=\frac{1}{10}\right)$; que servirá para todos los métodos que se utilizará en este capítulo.

```
solucion = NDSolveProblem[{{Y1'[T] == Y1[T] (-2 + Y2[T]), Y2'[T] == (1 - Y1[T]) Y2[T]},
    {Y1[0] == 3, Y2[0] == 1}, {Y1[T], Y2[T]}, {T, 0, 10}, {},
    {Log[Y1[T]] - Y1[T] + 2 Log[Y2[T]] - Y2[T]}, {}}];

ecualog = solucion["Invariants"];
tiempo = solucion["TimeData"];
var = solucion["DependentVariables"];
h = 1 / 10;
```

La solución del sistema Lotka-Volterra se plantea de la siguiente manera

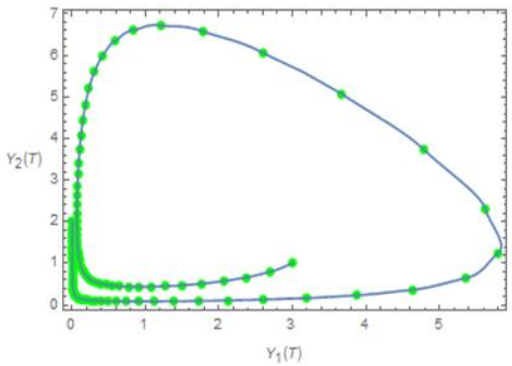
```
LotkaVolterra[sol_, var_, tiempo_, opc___ ?OptionQ] :=
  Module[{comp, datos, datos1, datos2, div, plot1, plot2}, div = First[var /. sol];
   datos1 = Part[div, 1, 0]["ValuesOnGrid"];
   datos2 = Part[div, 2, 0]["ValuesOnGrid"];
   datos = Transpose[{datos1, datos2}];
   comp = Sequence[Axes → False, Frame → True,
     FrameLabel → Join[Map[TraditionalForm, var], {None, None}], RotateLabel → False];
   plot1 = ListPlot[datos, Evaluate[FilterRules[{opc}, Options[ListPlot]]],
     PlotStyle → {PointSize[0.02], RGBColor[0, 1, 0]}, Evaluate[comp]];
   plot2 = ParametricPlot[Evaluate[div], tiempo,
     Evaluate[FilterRules[{opc}, Options[ParametricPlot]]], Evaluate[comp]];
   Show[plot1, plot2]];
```

En este capítulo analizaremos los métodos numéricos de resolución siguiendo las referencias [17] y [23], tomamos el valor de $h=\frac{1}{10}$ para todos los métodos.

4.1. Método de Euler explícito.

Se utiliza el método de Euler explícito para desarrollar el modelo (4.2),

```
solucion = NDSolve[solucion, Method → "ExplicitEuler", StartingStepSize → h];
LotkaVolterra[solucion , var, tiempo]
```



La solución del método de Euler Explícito para la ecuación (4.2) con un tamaño de paso de $h=\frac{1}{10}$ con valores iniciales en $(3,1)$ y con valores finales de $((0.001988, 2.020994))$,, traza las aproximaciones

numéricas de los primeros 100 pasos con la aplicación del método numérico anteriores, todos con tamaños de pasos constantes, observe que los métodos de Euler explícitos muestran un comportamiento cualitativo incorrecto ya que las soluciones numéricas son en espiral hacia afuera o hacia adentro.

Método de Euler implícito.

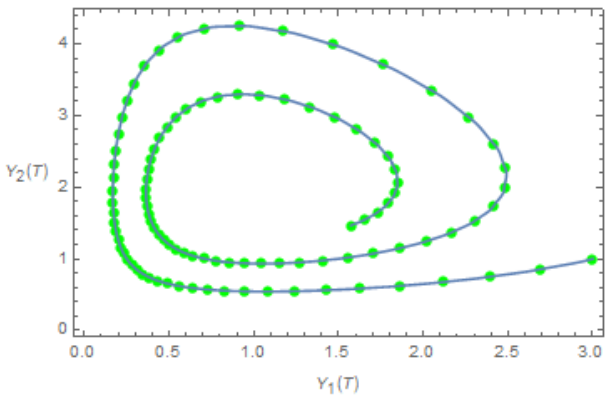
```
EulerImplicito =
  {"FixedStep",
   Method → {"ImplicitRungeKutta", "Coefficients" → "ImplicitRungeKuttaRadauIIACoefficients",
     "DifferenceOrder" → 1,
     "ImplicitSolver" → {"FixedPoint", AccuracyGoal → MachinePrecision,
       PrecisionGoal → MachinePrecision, "IterationSafetyFactor" → 1}}};
```

Luego resolvemos con el comando NDSolve

```
solucion1 = NDSolve[solucion, Method → EulerImplicito, StartingStepSize → h];
```

Aplicamos la función de Lotka-Volterra, sobre la función Runge Kutta

```
LotkaVolterra[solucion1, var, tiempo]
```



La solución del método de Euler Explícito para la ecuación (4.2) con un tamaño de paso de $h=\frac{1}{10}$ con valores iniciales en $(3,1)$ y con valores finales de $(1.57264,1.466899)$, al igual que el método de Euler Implícito se traza aproximaciones numéricas de los primeros 100 pasos, observe que los métodos de Euler implícito muestran un comportamiento cualitativo incorrecto ya que las soluciones numéricas son espirales.

La solución del método de Euler implícito para la ecuación (4.2) con un tamaño de paso de $h=\frac{1}{10}$ con valores iniciales en $(3,1)$ y con valores finales de $(1.57264,1.466899)$, traza las aproximaciones numéricas de los primeros 100 pasos con la aplicación del método numérico anteriores, todos con tamaños de pasos constantes, observe que los métodos de Euler implícito muestran un comportamiento cualitativo incorrecto ya que las soluciones numéricas son en espiral hacia afuera o hacia adentro.

4.2 Método de Runge Kutta 4.

Para la utilización del método Runge Kuttta de cuarto orden se crea la respectiva función,

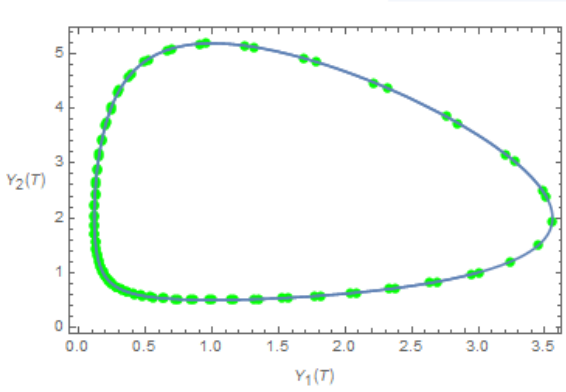
```
RungeKutta4[___]["Step"[f_, t_, h_, y_, fxy_]] := Block[{yt, k1, k2, k3, k4},
   k1 = fxy;
   k2 = f[t + 1/2 h, y + 1/2 h k1];
   k3 = f[t + 1/2 h, y + 1/2 h k2];
   k4 = f[t + h, y + h k3];
   yt = h (1/6 k1 + 1/3 k2 + 1/3 k3 + 1/6 k4);
   {h, yt}];
```

Luego resolvemos con el comando NDSolve

```
solucion3 = NDSolve[solucion, Method → RungeKutta4, StartingStepSize → h]
grafico3 = LotkaVolterra[solucion3, var, tiempo]
(*InputForm[solucion3/.T→tfinal]*)
```

```
{{Y1[T] → InterpolatingFunction[ Domain: {{0., 10.}} Output: scalar ][T], Y2[T] → InterpolatingFunction[ Domain: {{0., 10.}} Output: scalar ][T]}}
```

Aplicamos la función de Lotka-Volterra, sobre la función Runge Kutta 4.



La solución para la colección de los métodos de Runge Kutta para la ecuación (4.2) con un tamaño de paso de $h = \frac{1}{10}$ con valores iniciales en $(3,1)$ y con valores finales de $(3.50998, 2.38822)$, método de 4 orden, en el ejercicio ilustramos la idea de colocación con estos métodos para el problema de Lotka-Volterra (4.2), estos métodos son bastante satisfactorios.

4.3 Método de Runge Kutta 5.

Los métodos explícitos de Runge-Kutta se han utilizado ampliamente en la solución de problemas de valores iniciales. Si bien Kutta no brinda una solución general a las ecuaciones simultáneas involucradas, sí describe una familia de soluciones. Al elegir miembros particulares de estas soluciones de quinto orden, busca la simplicidad de los coeficientes. Exhibe dos miembros, cada uno de los cuales tiene algún error, uno es corregido por Fehlberg y uno por Nystrom [29]. Este último es el que se usa comúnmente y es:

$$y_{n+1} = y_n + \{23k_1 + 125\,\mathrm{k}_2 - 81\mathrm{k}_5 + 125\,\mathrm{k}_6\}/192$$

$$\mathrm{k}_1 = hf\left(x_n, y_n\right)$$

$$\mathrm{k}_2 = hf\left(x_n + \frac{h}{3}, y_n + k_1/3\right)$$

$$\mathrm{k}_3 = hf\left(x_n + \frac{2h}{5}, y_n + \{4k_1 + 6k_2\}/25\right)$$

$$\mathrm{k}_4 = hf\left(x_n + h, y_n + \{k_1 - 12k_2 + 15\,\mathrm{k}_3\}/4\right)$$

$$\mathrm{k}_5 = hf\left(x_n + \frac{2h}{3}, y_n + \{6k_1 + 90k_2 - 50\,\mathrm{k}_3 + 8\,\mathrm{k}_4\}/81\right)$$

$$\mathrm{k}_6 = hf\left(x_n + \frac{4h}{5}, y_n + \{6k_1 + 36k_2 10\,\mathrm{k}_3 + 8\,\mathrm{k}_4\}/75\right)$$

Para la utilización del método Runge Kuttta de quinto orden de Butcher (1964), de orden superior se crea la respectiva función,

```
RungeKutta5 /: NDSolve`InitializeMethod[RungeKutta5, __] := RungeKutta5[];
RungeKutta5[___]["Step"[f_, t_, h_, y_, fxy_]] := Block[{yt, k1, k2, k3, k4, k5, k6},
   k1 = fxy;
   k2 = f[t + 1/4 h, y + 1/4 h k1];
   k3 = f[t + 1/4 h, y + 1/8 h k1 + 1/8 h k2];
   k4 = f[t + 1/2 h, y - 1/2 h k2 + h k3];
   k5 = f[t + 3/4 h, y + 3/16 h k1 + 9/16 h k4];
   k6 = f[t + h, y - 3/7 h k1 + 2/7 h k2 + 12/7 h k3 - 12/7 h k4 + 8/7 h k5];
   yt = h (1/90 (7 k1 + 32 k3 + 12 k4 + 32 k5 + 7 k6));
   {h, yt}];
```

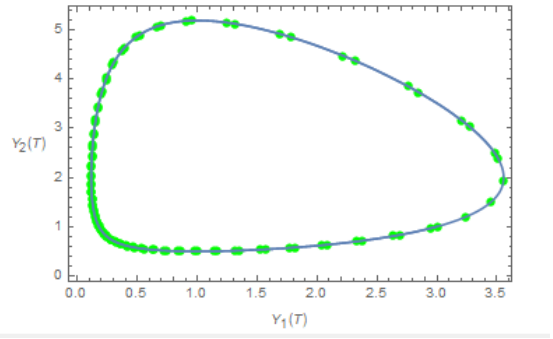
Luego resolvemos con el comando NDSolve

```
solucion3 = NDSolve[solucion, Method → RungeKutta5, StartingStepSize → h]
```

```
{{Y1[T] → InterpolatingFunction[ Domain: {{0., 10.}} Output: scalar ][T],
 Y2[T] → InterpolatingFunction[ Domain: {{0., 10.}} Output: scalar ][T]}}
```

Aplicamos la función de Lotka-Volterra, sobre la función Runge Kutta 5

```
grafica = LotkaVolterra[solucion3, var, tiempo]
```



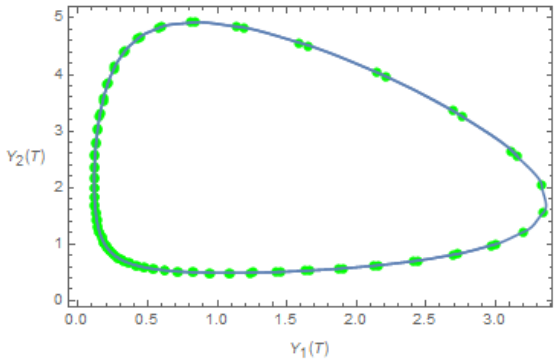
La solución para la colección de los métodos de Runge Kutta de orden 5 para la ecuación (4.2) con un tamaño de paso de $h = \frac{1}{10}$ con valores iniciales en $(3,1)$ y con valores finales de $(3.50993, 2.38777)$, para esto se creó una función que permita realizar el cálculo respectivo, este método tiene mayor precisión que el métodos RK4.

4.4 Método de Splitting y composición.

El método de Euler simpléctico en el método de Splitting, utilizamos los métodos de Euler explícito e implícito.

```
ecuacion = solucion["System"];
Y1 = ecuacion;
Part[Y1, 2, 2] = 0;
Y2 = ecuacion;
Part[Y2, 1, 2] = 0;
SimplecticoEuler = {"Splitting", "DifferenceOrder" → 1, "Equations" → {Y1, Y2},
   "Method" → {EulerImplicito, "ExplicitEuler"}};
solucion = NDSolve[solucion, Method → SimplecticoEuler, StartingStepSize → h];
```

```
LotkaVolterra[solucion, var, tiempo]
```

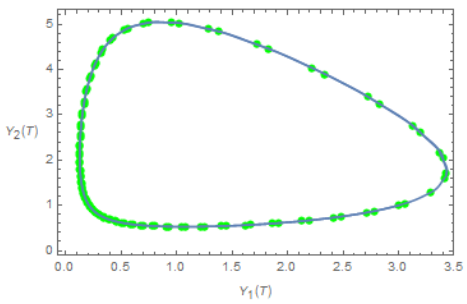


La solución para los métodos de Splitting con valores iniciales en $(3,1)$ y con valores finales de $(3.15546, 2.5649)$, observamos que el resultado de este método se aproxima a las soluciones de los métodos de Runge Kutta de orden 4 y orden 5.

Implementación del Método de LocallyExact, esta usa la evaluación numérica directa para avanzar localmente la solución.

```
DSolve[Y1, var, T]
{{Y2[T] → C[1], Y1[T] → e^(T (-2 C[1])) C[2]}}

DSolve[Y2, var, T]
{{Y1[T] → C[1], Y2[T] → e^(T (1 C[1])) C[2]}}

SplittingLotkaVolterra = {"Splitting", "DifferenceOrder" → 1, "Equations" → {Y1, Y2},
   "Method" → {"LocallyExact", "LocallyExact"}};
solucion = NDSolve[solucion, Method → SplittingLotkaVolterra, StartingStepSize → h];
```

```
LotkaVolterra[solucion, var, tiempo]
```

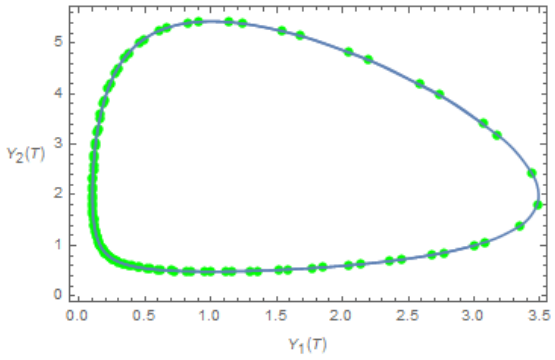


La solución para los métodos de Splitting LocallyExact con valores iniciales en $(3,1)$ y con valores finales de $(3.428, 1.69879)$, observamos que para desarrollar con este método, primero resolvemos las ecuaciones del sistema (4.2) con el comando DSolve, resolviendo de esta manera cada ecuación diferencial.

Método de composición de pasos de integración de Euler.

```
splittingsolucion = NDSolve[solucion, tiempo, StartingStepSize → h,
  Method → {"Splitting", "DifferenceOrder" → 2, "Equations" → {Y1, Y2, Y1},
    "Method" → {"ExplicitEuler", "ExplicitEuler", "ExplicitEuler"}}]
{{Y1[T] → InterpolatingFunction[ Domain: {{0., 10.}} Output: scalar ][T],
  Y2[T] → InterpolatingFunction[ Domain: {{0., 10.}} Output: scalar ][T]}}
```

```
LotkaVolterra[splittingsolucion, var, tiempo]
```

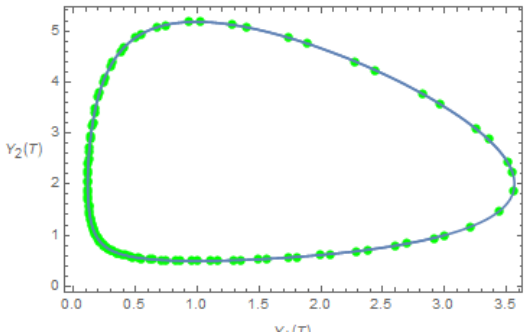


La solución para los métodos de Splitting y composición de pasos de integración de Euler, con valores iniciales en $(3,1)$ y con valores finales de $(3.0673, 3.40644269)$, observamos que para desarrollar con estos métodos, utilizamos la variables Y_1 y Y_2.

Regla implícita del punto medio en términos del método "ImplicitRungeKutta" incorporado.

```
ImplicitMetodo =
  {"FixedStep",
   Method → {"ImplicitRungeKutta", "Coefficients" → "ImplicitRungeKuttaGaussCoefficients",
     "DifferenceOrder" → 2,
     ImplicitSolver → {FixedPoint, AccuracyGoal → MachinePrecision,
       PrecisionGoal → MachinePrecision, "IterationSafetyFactor" → 1}}};

  splittingsolucion = NDSolve[solucion, StartingStepSize → h,
    Method → {"Splitting", "DifferenceOrder" → 2, "Equations" → {Y2, Y1, Y2},
      "Method" → {"LocallyExact", ImplicitMetodo, "LocallyExact"}}]

{{Y1[T] → InterpolatingFunction[ Domain: {{0., 10.}} Output: scalar ][T],

  Y2[T] → InterpolatingFunction[ Domain: {{0., 10.}} Output: scalar ][T]}}
```

```
LotkaVolterra[splittingsolucion, var, tiempo]
```

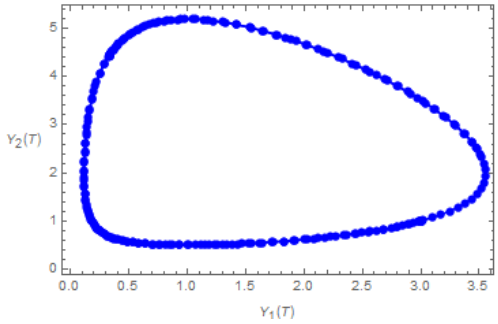


La solución para los métodos de Splitting y composición de ImplicitRungeKutta, con valores iniciales en $(3,1)$ y con valores finales de $(3.54615, 2.2494852)$, observamos que para desarrollar con estos métodos, utilizamos la variables Y_1 y Y_2, esta es una regla implícita del punto medio en términos del método "ImplicitRungeKutta" incorporado.

4.5 Comparación de métodos.

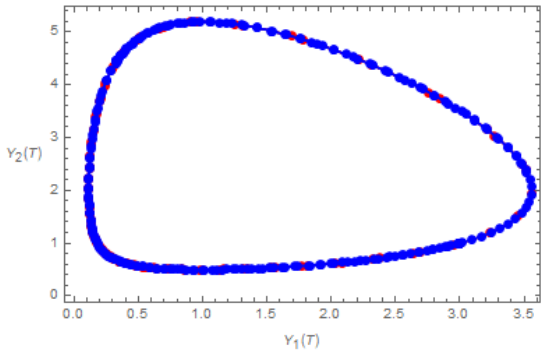
Para realizar el análisis primero se calcula la solución exacta con el comando NDSolve.

```
solucionr = NDSolve[{Y1'[T] == Y1[T] (-2 + Y2[T]), Y2'[T] == (1 - Y1[T]) Y2[T], Y1[0] == 3, Y2[0] == 1},
   {Y1[T], Y2[T]}, {T, 0, 10}];
```



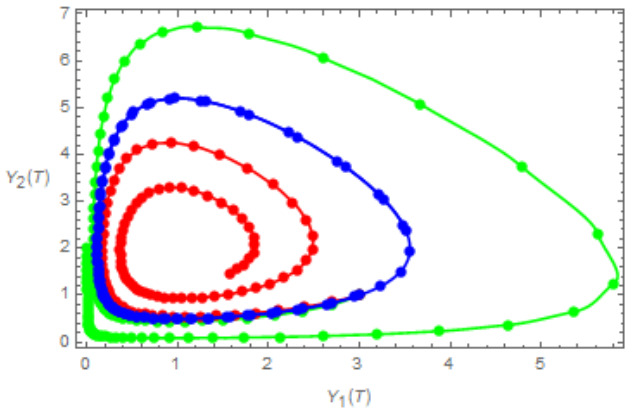
Plano de fase 4.1. Solución exacta

En el plano de fase 4.2 se observa la comparación de los métodos de Runge-Kutta 5 y la solución exacta, podemos concluir que la solución del método RK5 se aproxima en gran medida a la solución exacta.



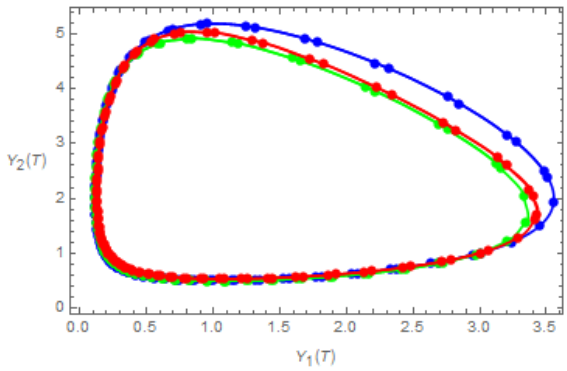
Plano de fase 4.2 Solución exacta y RK5

En el plano de fase 4.3 se observa la comparación de los métodos RK5 (azul), Euler explícito (verde) y Euler implícito (rojo), claramente se observa que los métodos de Euler se alejan demasiado a la solución de los métodos de Runge-Kutta 5 implicando así que se aleja a la solución exacta.



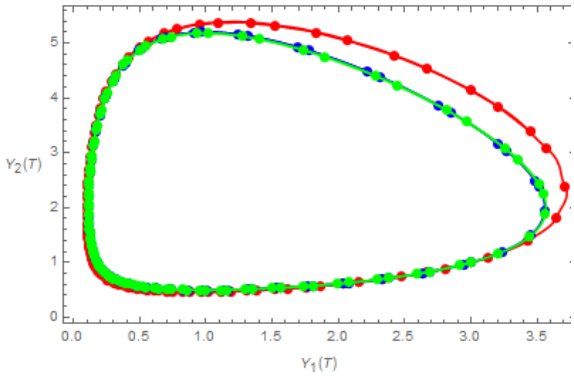
Plano de fase 4.3. RK5, Euler explícito y Euler implícito

En el plano de fase 4.4 se comparan los métodos RK5 (azul) y los métodos de Splitting LocallyExact (rojo) y el método Splitting Euler simpléctico (verde), estos métodos de Splitting y Composición se acercan al método de Runge-Kutta 5, por tanto se acercan a la solución exacta.



Plano de fase 4.4. RK5, LocallyExact y Euler simpléctico

En el plano 4.5 se comparan los métodos RK5 (azul) y los métodos de Splitting y Composición: Regla implícita del punto medio en términos del método "ImplicitRungeKutta" incorporado (verde) y método de composición de pasos de integración de Euler (rojo), se observa claramente que el método de Splitting y Composición ImplicitRungeKutta se acerca en gran medida al método RK5 por la tanto a la solución correcta, mientras que el método de integración de Euler está alejado de la solución en la mayoría de puntos.



Plano de fase 4.5. RK5, ImplicitRungeKutta y la integración de Euler.

En la tabla 4.1 se muestra algunas interacciones del plano de fase de los métodos de Runge-Kutta 5 y los valores exactos, se verifica la cercanía que existe entre los puntos de los dos métodos, en la mayoría, los puntos son los mismo y en algunos casos los puntos varían con una décima, para el análisis se redondeó a 2 decimales y se utilizó los métodos de Runge-Kutta y los valores exactos.

Y_1	**Exacto** Y_2	**Runge-Kutta 5** (Y_1, Y_2)	Y_1	**Exacto** Y_2	**Runge-Kutta 5** Y_2
2.08	0.63	0.63	0.35	0.65	0.65
0.86	0.49	0.49	0.24	0.80	0.80
0.63	0.52	0.52	0.21	0.87	0.85
0.31	0.68	0.68	0.15	1.11	1.13
0.17	1.00	1.02	0.14	1.21	1.23
0.14	1.19	1.19	0.13	1.31	1.33
0.13	1.29	1.28	1.12	1.43	1.42
0.12	1.41	1.41	0.12	1.57	1.56
0.12	1.54	1.56	0.11	1.71	1.69
0.11	2.01	2.05	0.114	1.87	1.83
0.11	2.20	2.23	0.113	2.04	2.01
0.12	2.62	2.60	0.115	2.23	2.21
0.16	3.39	3.33	0.118	2.44	2.42
0.19	3.68	3.69	0.12	2.66	2.62
3.48	2.49	2.49	0.13	2.91	2.83
3.55	1.93	1.92	0.24	4.03	4.07
2.32	0.69	0.70	0.39	4.64	4.64
2.03	0.61	0.61	1.77	4.86	4.87
1.71	0.56	0.55	2.31	4.37	4.36
1.52	0.53	0.52	2.84	3.73	3.80
0.53	0.55	0.55	3.27	3.03	2.99

Tabla 4.1. Tabla comparativa de los puntos del método Runge-Kutta y exacto.

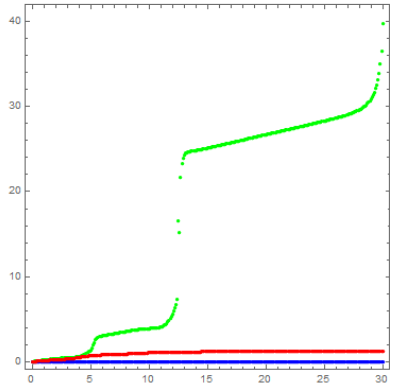
Según la investigación realizada los métodos de Runge-Kutta logran una exactitud mayor sobre otros métodos, estos datos también los validan otras investigaciónes [11,12,34] en el que determinan que los métodos de Runge-Kutta tienen una exactitud mayor.

Aunque el método de Runge-Kutta nos da resultados más precisos, se debe tener en cuenta que este método requiere un coste computacional ya que el número de etapas del método Runge-Kutta es el número de veces que se evalúa la función en cada paso i.

4.6 Invariantes.

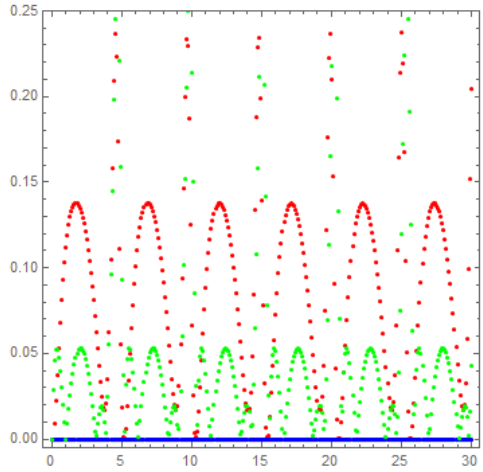
Conservación de los diferentes métodos de la función $I(u,v)$, esta es una constante del movimiento, por tanto siempre toma exactamente el mismo valor, mientras que cada método proporcionará un error diferente.

En el gráfico 4.1 se observa la comparación de los métodos RK5 (azul), Euler explícito (verde) y Euler implícito (rojo), observe que RK5 tiene una mejor conservación de I con respectos a los otros métodos, se puede observar que el método de Euler implícito tiene mejor conservación de I que el método de Euler explícito.



Gráfica 4.1 Invariante de RK5, Euler explícito y Euler implícito

En el gráfico 4.2 se comparan los métodos RK5 (azul) y los métodos de Splitting LocallyExact (rojo) y el método Splitting Euler simpléctico (verde), se observa cómo crece y disminuye el error del invariante en los métodos de Splitting LocallyExact y Euler simpléctico, están sobre el método de Runge-Kutta5.



Gráfica 4.2 Invariante de RK5, LocallyExact y Euler simpléctico

En el plano 4.5 se comparan los métodos RK5 (azul) y los métodos de Splitting y Composición: Regla implícita del punto medio en términos del método "ImplicitRungeKutta" incorporado (verde) y método de composición de pasos de integración de Euler (rojo), se observa claramente que el error invariante del método de integración de Euler tiene valores que aumentan y luego vuelven a disminuir, pero podemos observar que el método RK5 es el que tiene mejor conservación.

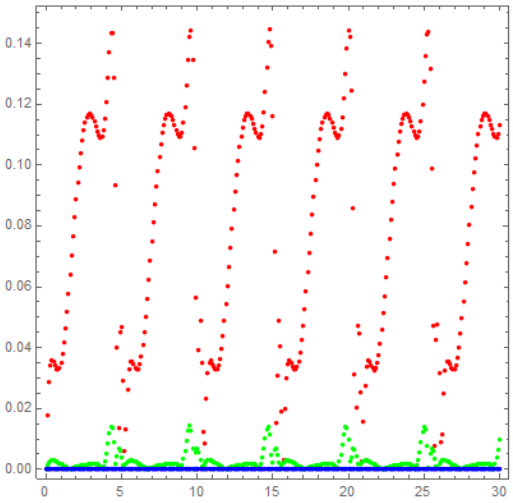


Gráfico 4.3 . Invariante de RK5, ImplicitRungeKutta y la integración de Euler.

En anexos se encuentran los gráficos de cada método analizado en esta investigación de función $I(u,v)$, para un mejor análisis de los resultados.

Conclusiones

En este trabajo investigativo se realiza una breve introducción de Wolfram Mathematica y con mayor énfasis el manejo del comando NDSolve ya que facilita la resolución de una ecuación diferencial numéricamente, se procede a analizar un grupo de métodos numéricos para hallar la solución numérica de ecuaciones diferenciales con condiciones iniciales dadas.

Se ha tratado el modelo presa-depredador con el método de Lotka-Volterra y se utilizó el modelo $\dot{u} = u(v-2)$, $\dot{v} = v(1-u)$, sobre este se aplicó los métodos de Euler explícito, Euler implícito, Runge-Kutta4, Runge Kutta 5 y splitting y composición. Estos métodos se realizaron con el apoyo de Mathematica.

Finalmente se realizó una comparación entre los diferentes métodos, para esto se calculó la solución con el comando NDSolve y se procedió a realizar la comparación con el método de Runge-Kutta5, analizando las interacciones del plano de fase 4.2 se puede determinar que el método de Runge-Kutta5 tiene muy buena precisión y se asemeja a la solución que se obtuvo con el comando NDSolve. El método Runge-Kutta muestra mejor precisión que los demás métodos estudiados en esta investigación, esto podemos determinar gracias al análisis de los diagramas de fase y la tabla de puntos de las interacciones de cada método (Anexos).

Bibliografía

[1] AGUIRREGABIRIA AGUIRRE, Juan María. *Ecuaciones diferenciales ordinarias para estudiantes de física*. Servicio Editorial de la Universidad del País Vasco/Euskal Herriko Unibertsitatearen Argitalpen Zerbitzua, 2009.

[2] ARNOL'D, Vladimir Igorevich. *Mathematical methods of classical mechanics*. Springer Science & Business Media, 2013.

[3] BANK, R., GRAHAM, R. L., STOER, J., & VARGA, R. *Solving Ordinary Differential Equations*, Springer Series in 8,1987.

[4] BLANES, Sergio; CASAS, Fernando. *A concise introduction to geometric numerical integration*, Chapman and Hall/CRC, 2016.

[5] BLANES, Sergio; CASAS, Fernando; MURUA, Ander. *Splitting and composition methods in the numerical integration of differential equations*. arXiv preprint arXiv:0812.0377, 2008.

[6]BUTCHER, John Charles. *The numerical analysis of ordinary differential equations*: Runge-Kutta and general linear methods. 1987.

[7] CANDY, J.; ROZMUS, W. *A symplectic integration algorithm for separable Hamiltonian functions. Journal of Computational Physics*, 1991, vol. 92, no 1, p. 230-256.

[8] CHAPRA, Steven C.; CANALE, Raymond P. *Métodos numéricos para ingenieros*. McGraw-Hill, 2007.

[9] CHERO, Robert Ipanaqué; LEÓN, Ricardo Velesmoro. *Breve manual de MATHEMATICA 5.1*, EumedNet, 2005.

[10] CHRISTIANTO, Victor; SMARANDACHE, Florentin. *An Economic Analogy with Maxwell Equations in Fractional Space.* Infinite Study, 2015.

[11] CHRISTODOULOU, Nikolaos S. *An algorithm using Runge–Kutta methods of orders 4 and 5 for systems of ODEs*. International journal of numerical methods and applications, 2009, vol. 2, no 1, p. 47-57.

[12] DI RADO, G. R.; DEVINCENZI, G. H.; PRESTA GARCÍA DANIEL, S. *Aplicación del Método de Integración Numérica de Ecuaciones Diferenciales Runge Y Kutta 4 (Rk4) a un Modelo de Simulación Longitudinal de Dinámica Vehicular Terrestre*. Mecánica Computacional, 2011, vol. 30, p. 2907-2927.

[13] EDWARDS, C. Henry; PENNEY, David E. *Ecuaciones diferenciales.* Pearson Educación, 2001.

[14] MARTEL, Eugenio Fedriani. *Guía rápida para el nuevo usuario de LATEX*. Juan Carlos Martínez Coll, 2004.

[15] GARCÍA GARROSA, Amelia, et al. *Métodos Numéricos tipo Runge-Kutta-Nyström para la integración eficiente de problemas oscilatorios*. 2001.

[16] HAIRER, Ernst; NØRSETT, Syvert Paul; WANNER, Gerhard. *Solving ordinary differential equations*. Springer, 1993.

[17] HAIRER, Ernst; LUBICH, Christian; WANNER, Gerhard. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*. Springer Science & Business Media, 2006.

[18] HAIRER, Ernst; NØRSETT, Syvert P.; WANNER, Gerhard. *Solving ordinary differential equations*. I, volume 8 of Springer Series in Computational Mathematics. 1993.

[19] http://demonstrations.wolfram.com/TwoStepAndFourStepAdamsPredictorCorrectorMethod/

[20] http://mathworld.wolfram.com/Predictor-CorrectorMethods.html

[21] http://reference.wolfram.com/language/tutorial/NDSolveIntroductoryTutorial.html

[22] http://reference.wolfram.com/language/tutorial/NDSolvePackages.html

[23] http://reference.wolfram.com/language/tutorial/NDSolveSplitting.html

[24] http://www.cfm.brown.edu/people/dobrush/am33/Mathematica/pc.html

[25] https://mathematica.stackexchange.com/questions/74214/the-lotka-volterra-predator-prey-model

[26] https://www.wolfram.com/language/index.es.html?footer=lang

[27] https://www.wolfram.com/mathematica/quick-revision-history.es.html?footer=lang

[28] LLONGO, Vicent Almenar; SANCHO, Francesc Hernández. *Excel como herramienta docente de las asignaturas de Microeconomía*. @ tic. revista d'innovació educativa, 2009, no 3, p. 108-114.

[29] LUTHER, H. A.; KONEN, H. P. *Some fifth-order classical Runge-Kutta formulas*. SIAM Review, 1965, vol. 7, no 4, p. 551-558.

[30] MANZANO GARCÍA, Fernando, et al. *Estudio de un tipo de extrapolación no lineal y sus aplicaciones*. 2011.

[31] MARCHUK, Gurij Ivanovich. *Some application of splitting-up methods to the solution of mathematical physics problems. Aplikace matematiky*, 1968, vol. 13, no 2, p. 103-132.

[32] MCMAHON, David; TOPA, Daniel M. *A beginner's guide to mathematica*. Chapman and Hall/CRC, 2006.

[33] OLVER, Peter J. *Applications of Lie groups to differential equations*. Springer Science & Business Media, 2000.

[34] RODRÍGUEZ, Carlos Mata. *Análisis comparativo de los métodos de Euler y Runge-Kutta en la solución numérica de ecuaciones diferenciales de primer orden mediante programación en mathcad*. Revista Ingeniería, Matemáticas y Ciencias de la Información, 2016, vol. 3, no 5.

[35] ROSAS, Jesús Javier Cortés. *Solución numérica de ecuaciones diferenciales con condiciones iniciales*, academia, 2011.

[36] SALAS, Carlos L. Arce. *Mathematica: potentes herramientas*. Revista Digital: Matemática, Educación e Internet, 2002, vol. 3, no 3.

[37] STRANG, Gilbert. *On the construction and comparison of difference schemes*. SIAM Journal on Numerical Analysis, 1968, vol. 5, no 3, p. 506-517.

[38] SÜLI, E.; MAYERS, D. F. An introduction to numerical analysis, Cambridge University Press, 2003.

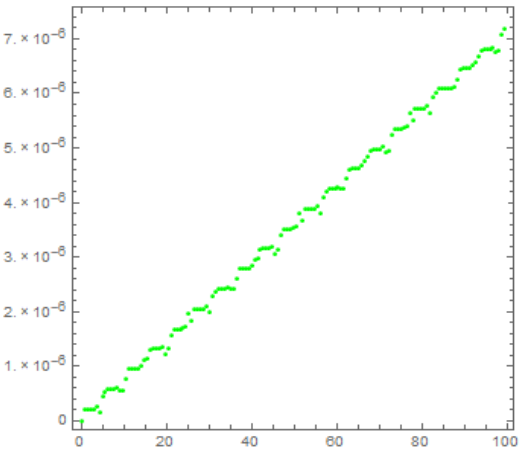
[39] TROTT, Michael. *The Mathematica guidebook for numerics*. Springer Science & Business Media, 2006.

[40] TROTTER, Hale F. *On the product of semi-groups of operators*. Proceedings of the American Mathematical Society, 1959, vol. 10, no 4, p. 545-551.

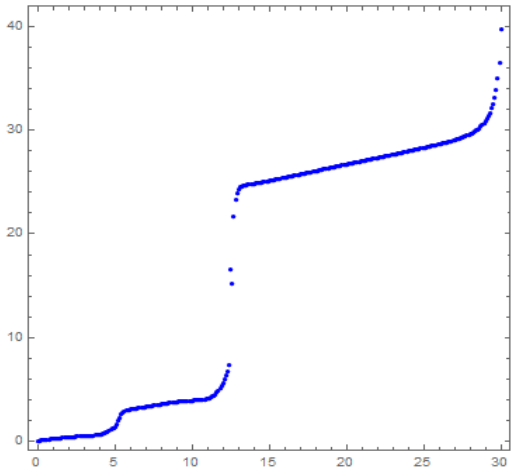
[41] YOSHIDA, Haruo. *Construction of higher order symplectic integrators*. Physics letters A, 1990, vol. 150, no 5-7, p. 262-268.

Anexos

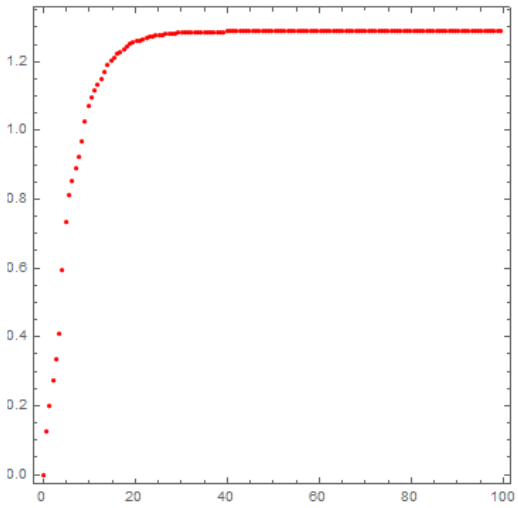
Método de Runge-Kutta5



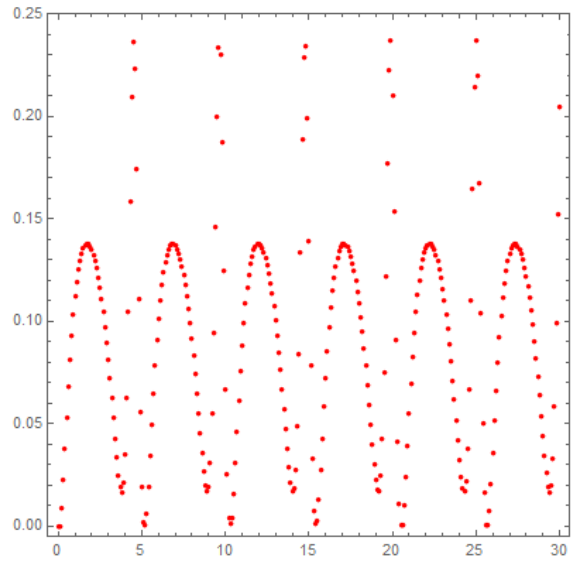
Método de Euler Explícito



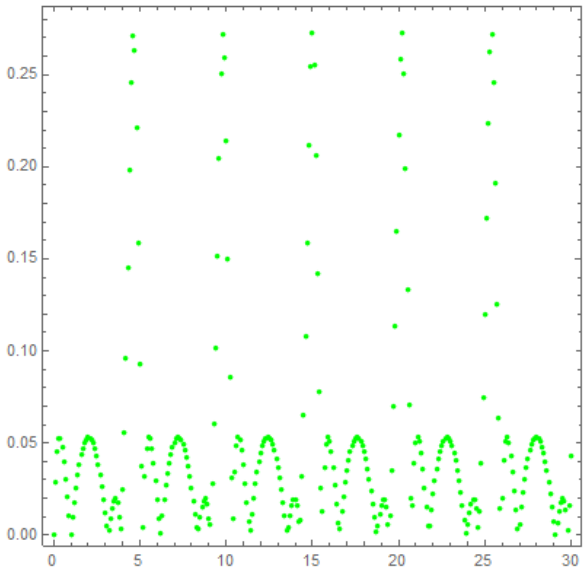
Método de Euler implícito



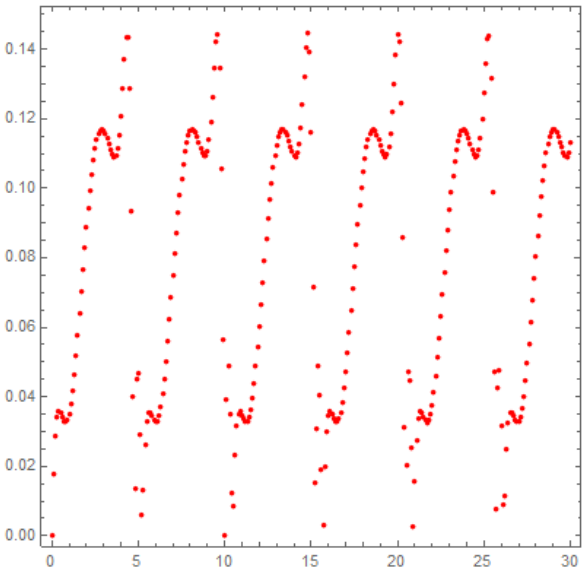
Método splitting y composición (LocallyExact)



Método splitting y composición (Euler simlpléctico)



Método splitting y composición (Integración de Euler)



Método splitting y composición (ImplicitRungeKutta)