Implementación de métodos numéricos en *Mathematica* de forma funcional

José Luis Gómez Muñoz

Los comandos NestWhileList y Function

- **Se duplica el número, comenzando con 10, mientras el número sea menor que 100**

```
NestWhileList[
 Function[numero, 2 * numero],
 10,
 Function[numero, numero < 100]
]
```

```
{10, 20, 40, 80, 160}
```

- **Se le suma dos al número, comenzando con 6, mientras el número sea menor que 15**

```
NestWhileList[
 Function[z, z + 2],
 6,
 Function[w, w < 15]
]
```

```
{6, 8, 10, 12, 14, 16}
```

- **A la pareja de números se les suma {10,200} respectivamente, comenzando con la pareja {3,7}, mientras el primer elemento de la pareja sea menor que 50**

```
NestWhileList[
 Function[pareja, pareja + {10, 200}],
 {3, 7},
 Function[pareja, pareja[[1]] < 50]
]
```

```
{{3, 7}, {13, 207}, {23, 407}, {33, 607}, {43, 807}, {53, 1007}}
```

Método de Euler implementación funcional

- **Ecuación diferencial del tipo:**

$$\begin{cases} \frac{dy}{dx} = f(x, y) \\ y(x_0) = y_0 \end{cases}$$

Método de Euler para obtener solución aproximada:

$$x_{j+1} = x_j + h$$

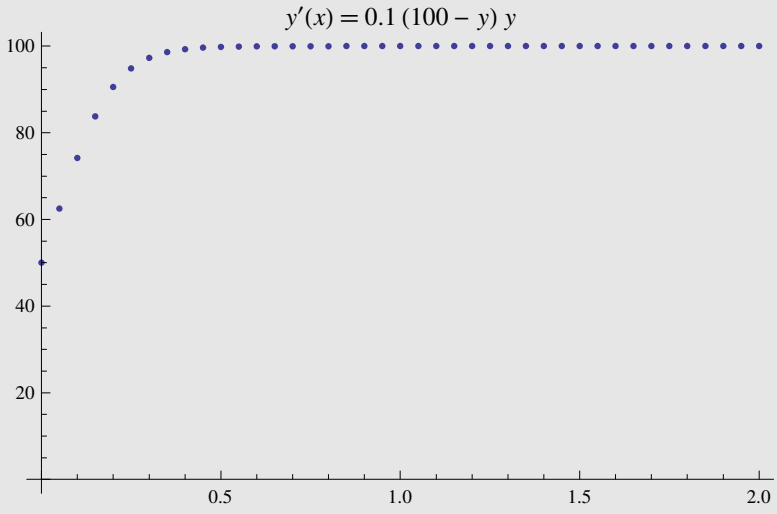
$$y_{j+1} = y_j + h * f(x_j, y_j)$$

Un programa de *Mathematica* que implementa este método:

```
euler[f_, {xo_, yo_}, xend_, h_] :=
 NestWhileList[
   (*Método numérico: *)
   Function[ parejaxy, parejaxy + {h, h * Apply[f, parejaxy]} ],
   (*Valores iniciales: *)
   {xo, yo},
   (*Condición para seguir calculando: *)
   Function[ parejaxy, parejaxy[[1]] < xend] ]
```

- **Ejemplo:**

```
f[x_, y_] := 0.1 * y * (100 - y);
datos = euler[f, {0, 50}, 2, 0.05];
ListPlot[datos, AxesOrigin → {0, 0}, PlotLabel → y'[x] == f[x, y]]
```

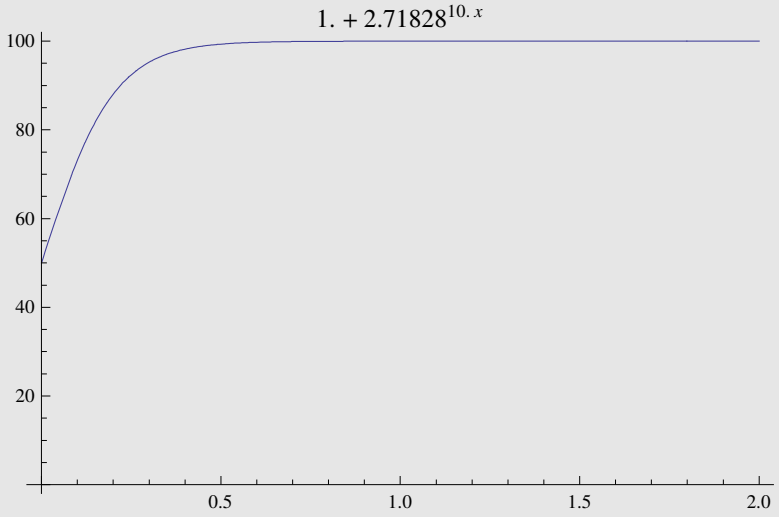


Solución exacta

```
f[x_, y_] := 0.1 * y * (100 - y);
sol = DSolve[{y'[x] == f[x, y[x]], y[0] == 50}, y, x];
exacta[x_] := y[x] /. sol[[1]];
Plot[exacta[x], {x, 0, 2}, AxesOrigin → {0, 0}, PlotLabel → exacta[x]]
```

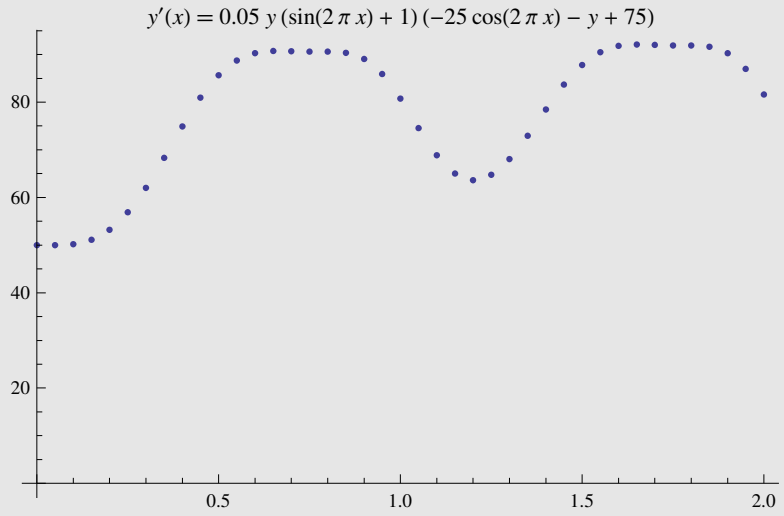
Solve::ifun: Inverse functions are being used by Solve, so
some solutions may not be found; use Reduce for complete solution information. >>

$$\frac{100. \times 2.71828^{10.\, x}}{1. + 2.71828^{10.\, x}}$$



Ejemplo:

```
k[x_] := 0.1 * (1 + Sin[2 * π * x / 1]) / 2;
ymax[x_] := 75 - 25 * Cos[2 * π * x / 1];
f[x_, y_] := k[x] * y * (ymax[x] - y);
datos = euler[f, {0, 50}, 2, 0.05];
ListPlot[datos, AxesOrigin → {0, 0}, PlotLabel → y'[x] == f[x, y]]
```



Método “Euler Mejorado”, implementación funcional

- **Ecuación diferencial del tipo:**

$$\begin{cases} \frac{dy}{dx} = f(x, y) \\ y(x_0) = y_0 \end{cases}$$

Método de Euler para obtener solución aproximada:

$$x_{j+1} = x_j + h$$

$$k_1 = h \cdot f(x_j, y_j)$$

$$k_2 = h \cdot f(x_j + h, y_j + k_1)$$

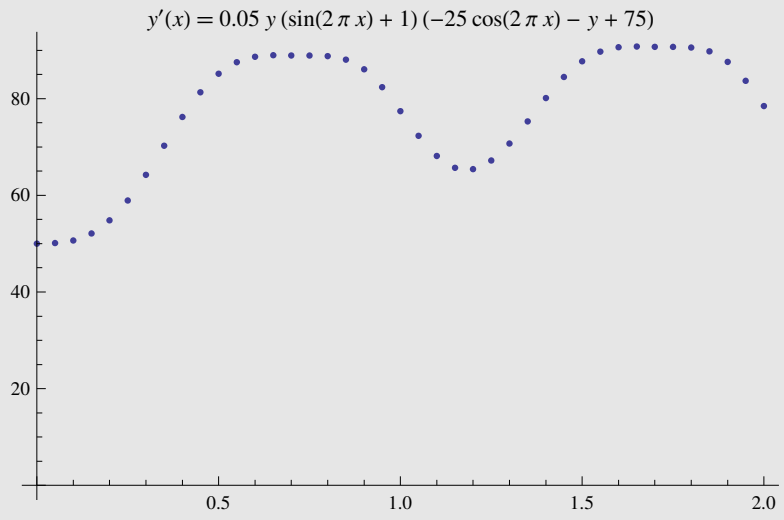
$$y_{j+1} = y_j + \frac{k_1 + k_2}{2}$$

Un programa de *Mathematica* que implementa este método. Nótese el uso de Module adentro de Function:

```
eulermejorado[f_, {xo_, yo_}, xend_, h_] :=
 NestWhileList[
  (*Método numérico: *)
  Function[ parejaxy,
   Module[{k1, k2},
    k1 = h * Apply[f, parejaxy];
    k2 = h * Apply[f, parejaxy + {h, k1}];
    parejaxy + {h, 0.5 * (k1 + k2)}] ],
  (*Valores iniciales: *)
  {xo, yo},
  (*Condición para seguir calculando: *)
  Function[ parejaxy, parejaxy[[1]] < xend] ]
```

- **Ejemplo Euler Mejorado:**

```
k[x_] := 0.1 * (1 + Sin[2 * π * x / 1]) / 2;
ymax[x_] := 75 - 25 * Cos[2 * π * x / 1];
f[x_, y_] := k[x] * y * (ymax[x] - y);
datos = eulermejorado[f, {0, 50}, 2, 0.05];
ListPlot[datos, AxesOrigin → {0, 0}, PlotLabel → y'[x] == f[x, y]]
```



- **Ejercicio para los estudiantes: De una manera similar a como se implementó Euler mejorado, implementar el Método de Runge-Kutta de cuarto orden:**

$$x_{j+1} = x_j + h$$

$$k_1 = h \cdot f(x_j, y_j)$$

$$k_2 = h \cdot f\left(x_j + \frac{h}{2}, y_j + \frac{k_1}{2}\right)$$

$$k_3 = h \cdot f\left(x_j + \frac{h}{2}, y_j + \frac{k_2}{2}\right)$$

$$k_4 = h \cdot f(x_j + h, y_j + k_3)$$

$$y_{j+1} = y_j + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}$$

Método de Euler de Orden Superior, implementación funcional

- **Ecuación diferencial del tipo:**

$$\begin{cases} y^{(n)} = f(x, y, y', y'', \ldots, y^{(n-1)}) \\ y(x_0) = y_0 \\ y'(x_0) = y'_0 \\ \quad \vdots \\ y^{(n-1)}(x_0) = y_0^{(n-1)} \end{cases}$$

Método de Euler (de orden superior) para obtener solución aproximada:

$x_{j+1} = x_j + h$

$y_{j+1} = y_j + h \cdot y'_j$

$y'_{j+1} = y_j + h \cdot y''_j$

$y''_{j+1} = y_j + h \cdot y'''_j$

$\vdots$

$y^{(n-1)}_{j+1} = y_j + h \cdot y^{(n)}_j$

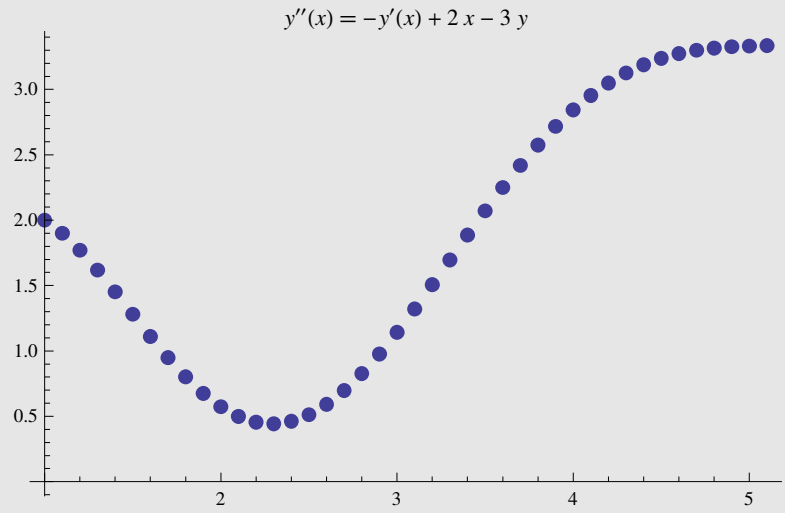
$y^{(n)}_{j+1} = y_j + h \cdot f\left(x, y, y', y'', \ldots, y^{(n-1)}\right)$

Un programa de *Mathematica* que implementa este método:

```
eulerOrdenSuperior[f_, listaini_, xend_, h_] :=
 NestWhileList[
  (*Método numérico:*)
  Function[ listaxyderivs,
   Module[{listaincrementos},
    listaincrementos =
     Flatten[{h,
       h * listaxyderivs[[ 3 ;; Length[listaxyderivs] ]],
       h * Apply[f, listaxyderivs]}];
    listaxyderivs + listaincrementos] ],
  (*Valores iniciales:*)
  listaini,
  (*Condición para seguir calculando: *)
  Function[ listaxyderivs, listaxyderivs[[1]] <= xend] ]
```

Euler de orden superior:

```
npuntos = 40;
x0 = 1.0;
xfin = 5.0;
y0 = 2.0;
v0 = -1.0;
f[x_, y_, v_] := -v + 2 x - 3 y;
aproximados = eulerOrdenSuperior[f, {x0, y0, v0}, xfin, (xfin - x0)/npuntos];
mispuntos = aproximados[[All, 1 ;; 2]];
ListPlot[mispuntos, PlotStyle → PointSize[Large],
 PlotLabel → y''[x] == f[x, y, y'[x]]]
```



Euler de orden superior:

```
npuntos = 40;
x0 = 1.0;
xfin = 5.0;
y0 = 2.0;
v0 = -1.0;
a0 = 3.0;
f[x_, y_, v_, a_] := a - v + 2 x - 3 y;
aproximados = eulerOrdenSuperior[f, {x0, y0, v0, a0}, xfin, (xfin - x0)/npuntos];
mispuntos = aproximados[[All, 1 ;; 2]];
ListPlot[mispuntos, PlotStyle → PointSize[Large],
 PlotLabel → y'''[x] == f[x, y[x], y'[x], y''[x]]]
```