AnalisisNumerico

Dorian Moreno y David Molano

26 de Octubre, 2018

1 Ejercicio 1

Considere un cuerpo con temperatura interna T el cual se encuentra en un ambiente con temperatura constante T_e. Suponga que su masa m concentrada en un solo punto. Entonces la transferencia de calor entre el cuerpo y el entorno externo puede ser descrita con la ley de Stefan-Boltzmann:

$$v(t) = \epsilon\gamma S(T^4(t) - T_e^4)$$

Donde, t es tiempo y ϵ es la constante de Boltzmann ($\epsilon = 5.6x10^{-8} J/m^2K^2s$), γ es la constante de "emisividad" del cuerpo, S el área de la superficie y v es la tasa de transferencia del calor. La tasa de variación de la energía $\frac{dT}{dt} = \frac{-v(t)}{mC}$ (C indica el calor específico del material que constituye el cuerpo). En consecuencia,

$$\frac{dT}{dt} = -\frac{\epsilon\gamma S(T^4(t) - T_e^4)}{mC}$$

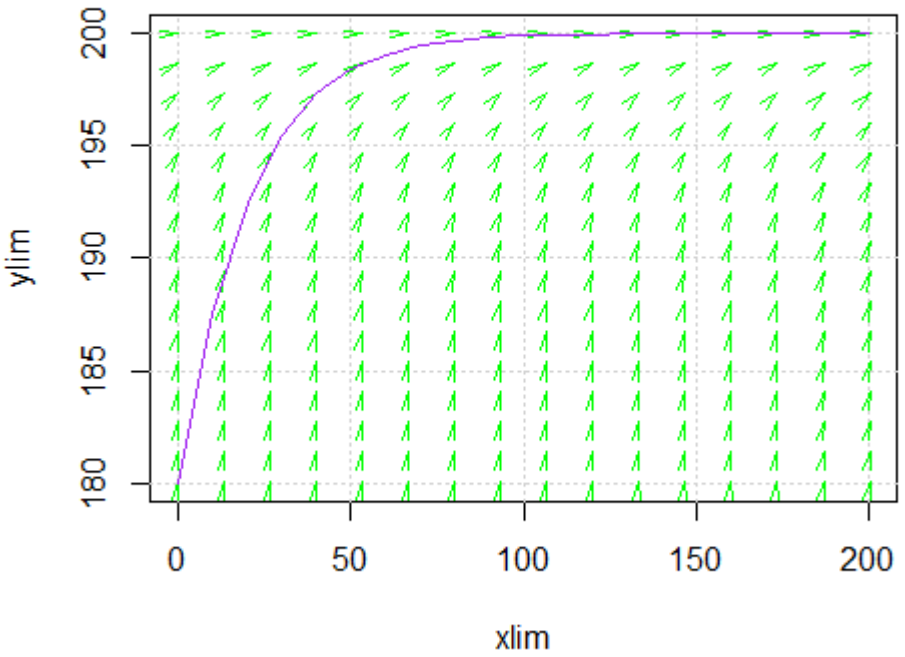
Usando el método de Euler (en R) y 20 intervalos iguales y t variando de 0 a 200 segundos, resuelva numéricamente la ecuación, si el cuerpo es un cubo de lados de longitud 1m y masa igual a 1Kg. Asuma que $T_0 = 180K$, $Te = 200K$, $\gamma = 0.5$ y $C = 100J/(Kg/K)$. Hacer una representación gráfica del resultado.

El código en R que utilizamos fue el siguiente:

```
metodoEuler <- function(f, h, xi, yi, xf)
{
  N = (xf - xi) / h
  x = y = numeric(N+1)
  x[1] = xi;
  y[1] = yi;
  i = 1
  while (i <= N)
  {
    x[i+1] = x[i]+h
    y[i+1] = y[i]+(h*f(x[i],y[i]))
    i = i+1
  }
  return (data.frame(X = x, Y = y))
}

f<-function(t, Tp, e = 5.6e-8, Te = 200, y = 0.5, m = 1, C = 100, S=6)
{
  top = -e*y*S*(Tp**4-Te**4)
  return(top/(m*C))
}

e1 = metodoEuler(f, 10, 0, 180, 20)

e1[nrow(e1),]

xx <- c(0, 200); yy <- c(180, 200)
vectorfield(f, xx, yy, scale = 0.1)
sol <- rk4(f, 0, 200, 180, 20)
data.frame(x = sol\$x, y = sol\$y)
lines(sol, col="purple")
```

Calculamos la función con los valores iniciales dados, lo que nos arroja los siguientes puntos y su gráfica, en conjunto con el campo vectorial:

x	y
0	180.0000
10	187.5804
20	192.4600
30	195.4897
40	197.3268
50	198.4246
60	199.0747
70	199.4576
80	199.6824
90	199.8142
100	199.8913
110	199.9365
120	199.9629
130	199.9783
140	199.9873
150	199.9926
160	199.9957
170	199.9975
180	199.9985
190	199.9991
200	199.9995



2 Ejercicio 2

Obtenga cinco puntos de la solución de la ecuación, utilizando el método de Taylor (los tres primeros términos) con $h = 0.1$. Implemente en R.

$$\frac{dy}{dx} - (x + y) = 1 - x^2; y(0) = 1$$

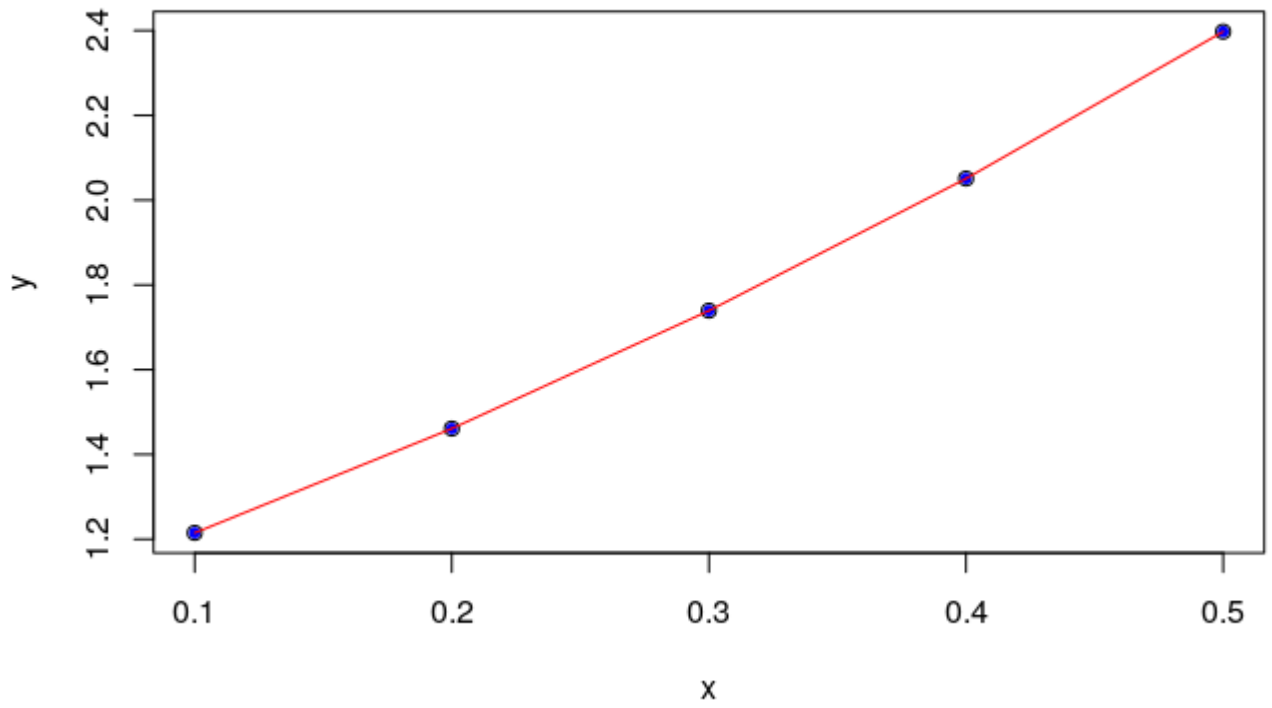
Grafique su solución y compare con la solución exacta. ¿Cual es el error de truncamiento en cada paso?

La siguiente fue la implementación del método de Taylor que se utilizó:

```
taylor<- function(fxy,dfxy,x,y,h,m){
  for (i in 1:m){
    y<-y+h*f(fxy,x,y)+h**2/2*f(dfxy,x,y)
    x<-x+h
    cat(x,y,"\n")
  }
}
```

Estos fueron los resultados obtenidos:

x	y
0.1	1.215
0.2	1.461025
0.3	1.739233
0.4	2.050902
0.5	2.397447



3 Ejercicio 3

Obtenga 20 puntos de la solución de la ecuación, utilizando el método de Euler con $h = 0.1$. Implemente en R.

$$\frac{dy}{dx} - (x + y) = 1 - x^2; y(0) = 1$$

Grafique su solución y compare con la solución exacta. ¿Cual es el error de truncamiento en cada paso?

La solución exacta es:

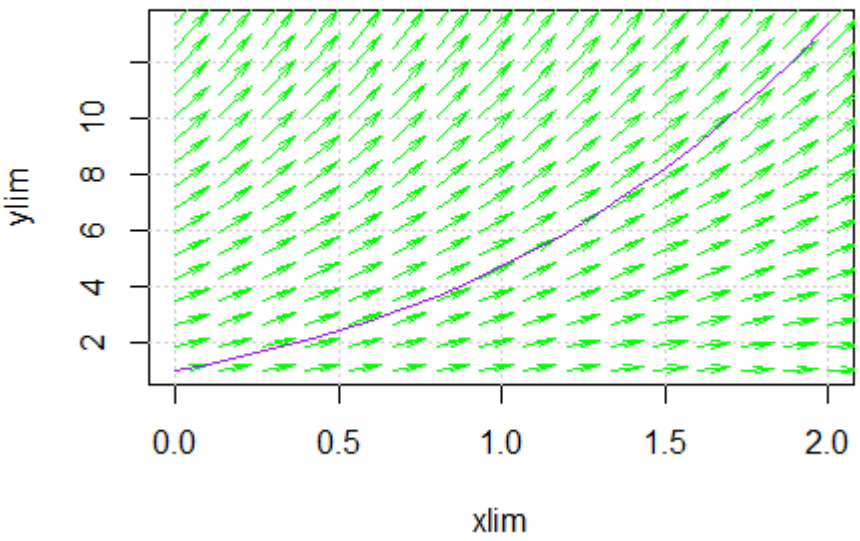
$$y = e^x(x^2e^{-x} + xe^{-x} + 1)$$

El código usado para resolver esta ecuación (de manera exacta y mediante Euler) es el siguiente:

```
metodoEuler <- function(f, h, xi, yi, xf)
{
  N = (xf - xi) / h
  x = y = numeric(N+1)
  x[1] = xi;
  y[1] = yi;
  i = 1
  while (i <= N)
  {
    x[i+1] = x[i]+h
    y[i+1] = y[i]+(h*f(x[i],y[i]))
    i = i+1
  }
  return (data.frame(X = x, Y = y))
}

truncamiento<-function(sol, fexacta)
{
  for(i in 1:(length(sol\$x)))
  {
    print(abs(sol\$y[i]-fexacta(sol\$x[i])))
  }
}

f<-function(x, y)
{
  return(1 + x+y-x**2)
}

fexacta<-function(x)
{
  return(exp(x)*((exp(-x)*x^2)+x*exp(-x)+1))
}

e1 = metodoEuler(f, 0.1, 0, 1, 20)

e1[nrow(e1),]

sol <- rk4(f, 0, 2, 1, 20)
data.frame(x = sol\$x, y = sol\$y)
xx <- c(min(sol\$x), max(sol\$x)); yy <- c(min(sol\$y), max(sol\$y))
vectorfield(f, xx, yy, scale = 0.1)
lines(sol, col="purple")
truncamiento(sol, fexacta)
```

Mientras que estos son los resultados utilizando el método de Euler, así como su error de truncamiento comparado con la solución exacta en cada paso:

x	y	error
0.0	1.000000	0
0.1	1.215171	2.9308e-07
0.2	1.461402	6.2589e-07
0.3	1.739858	1.0036e-06
0.4	2.051823	1.4318e-06
0.5	2.398719	1.9172e-06
0.6	2.782116	2.4668e-06
0.7	3.203750	3.0890e-06
0.8	3.665537	3.7929e-06
0.9	4.169599	4.5887e-06
1.0	4.718276	5.4881e-06
1.1	5.314160	6.5039e-06
1.2	5.960109	7.6509e-06
1.3	6.659288	8.9452e-06
1.4	7.415190	1.0405e-05
1.5	8.231677	1.2052e-05
1.6	9.113019	1.3907e-05
1.7	10.063931	1.5998e-05
1.8	11.089629	1.8353e-05
1.9	12.195873	2.1004e-05
2.0	13.389032	2.3988e-05



4 Ejercicio 4

Implemente en R el siguiente algoritmo y aplíquelo para resolver la ecuación anterior:

1) Defina f(x,y) y la condición incial (x_0, y_0)
2) Defina h y la cantidad de puntos a calcular m
3) Para i =1, 2, ..., m
4) $K_1 = hf(x_i, y_i)$
5) $K_2 = hf(x_i + h, y_i + K_1))$
6) $y_{i+1} = y_i + \frac{1}{2}(K_1 + K_2)$.
7) $x_{i+1} = x_i + h$
8) fin

Como no tendría sentido comparar los resultados de este método con el de Euler directamente, calcularemos el error de truncamiento con la solución exacta y luego compararemos este error con el obtenido mediante Euler (en el ejercicio 3).

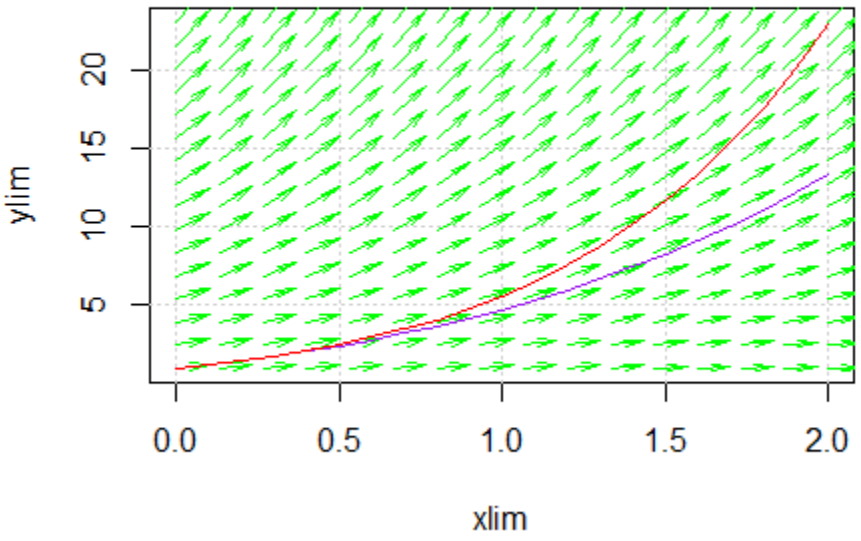
Este es el código que utilizamos:

```
siguiente<-function(f, x, y, h)
{
  k1 = h*f(x, y)
  k2 = h*f(x+h, y+k1)
  nexty = y + 0.5*(k1+k2)
  nextx = x + h
  return(list("x" = nextx, "y" = nexty))
}

punto4<-function(fexacta, f, x0, y0, h, n)
{
  x = y = error = numeric(n+1)
  x[1] = x0
  y[1] = y0
  error[1] = abs(y[1]-fexacta(x[1]))
  for(i in 1:n)
  {
    nex<- siguiente(f, x[i], y[i], h)
    y[i+1] = nex$y
    x[i+1] = nex$x
    error[i+1] = abs(y[i+1]-fexacta(x[i+1]))
  }
  return(data.frame("x" = x, "y" = y, "error" = error))
}

fexacta<-function(x)
{
  return(exp(x)*((exp(-x)*x^2)+x*exp(-x)+1))
}

f<-function(x, y)
{
  return(1+x+y+x^2)
}

punto4(fexacta, f, 0, 1, 0.1, 20)
```

Los datos que obtuvimos de esta forma son los siguientes (Euler a la izquierda y algoritmo nuevo a la derecha):

x	y	error
0.0	1.000000	0
0.1	1.215171	2.9308e-07
0.2	1.461402	6.2589e-07
0.3	1.739858	1.0036e-06
0.4	2.051823	1.4318e-06
0.5	2.398719	1.9172e-06
0.6	2.782116	2.4668e-06
0.7	3.203750	3.0890e-06
0.8	3.665537	3.7929e-06
0.9	4.169599	4.5887e-06
1.0	4.718276	5.4881e-06
1.1	5.314160	6.5039e-06
1.2	5.960109	7.6509e-06
1.3	6.659288	8.9452e-06
1.4	7.415190	1.0405e-05
1.5	8.231677	1.2052e-05
1.6	9.113019	1.3907e-05
1.7	10.063931	1.5998e-05
1.8	11.089629	1.8353e-05
1.9	12.195873	2.1004e-05
2.0	13.389032	2.3988e-05

x	y	error
0.0	1.000000	0.0000000000
0.1	1.215500	0.0003290819
0.2	1.466178	0.0047747418
0.3	1.757826	0.0179673299
0.4	2.096848	0.0450231843
0.5	2.490317	0.0915956388
0.6	2.946050	0.1639313847
0.7	3.472685	0.2689327470
0.8	4.079767	0.4142264987
0.9	4.777843	0.6082398959
1.0	5.578567	0.8602846943
1.1	6.494816	1.1806499837
1.2	7.540822	1.5807047657
1.3	8.732308	2.0730112982
1.4	10.086650	2.6714503353
1.5	11.623049	3.3913595136
1.6	13.362719	4.2496862608
1.7	15.329104	5.2651567554
1.8	17.548110	6.4584626182
1.9	20.048362	7.8524671990
2.0	22.861490	9.4724335147



Esta es la gráfica de ambos con los 20 puntos (El método de Euler es el morado y el nuevo es el rojo)

Podemos ver que con este algoritmo el error aumenta demasiado y se vuelve muy grande ya al final, siendo un error de casi el 50% del valor al que se lo sacamos. De esto podemos concluir que el método de Euler es mucho más preciso que este algoritmo a la hora de resolver esta ecuación diferencial ordinaria.

5 Ejercicio 5

Utilizar la siguiente variación en el método de Euler para resolver una ecuación diferencial ordinaria de primer orden, la cual calcula el promedio de las pendientes en cada paso:

$$y_{i+1} = y_i + \frac{h}{2} * (f(x_i, y_i) + f(x_{i+1}, y_{i+1}))$$

Implemente un código en R para este método y obtenga 10 puntos de la solución con $h = 0.1$. Grafíquela y compárela con el método de Euler.

$$\frac{dy}{dx} - x - y - 1 + x^2 = 0; y(0) = 1$$

Dado que la solución de este nuevo método se basa en tener ya calculado el método de Euler, vamos a utilizar los mismos datos que ya hallamos en el ejercicio 3 para esta ecuación que están guardados en 'euler'.

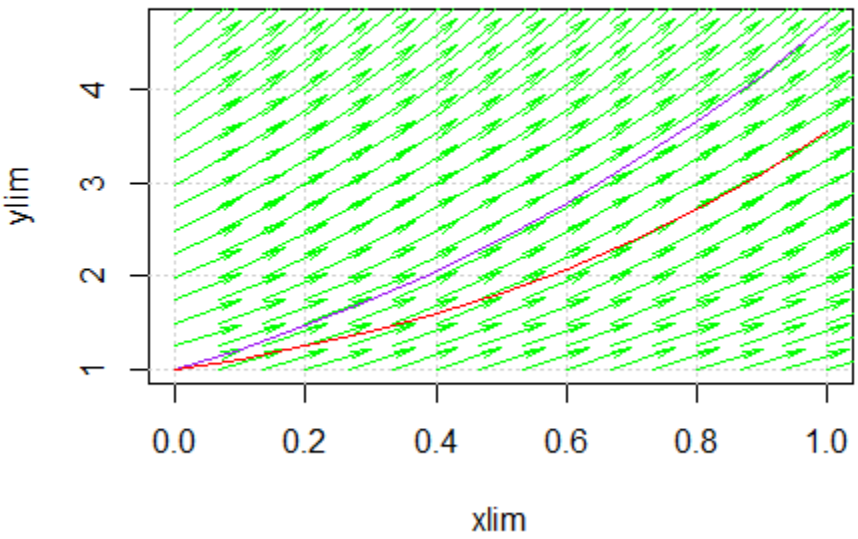
El código que utilizamos es el siguiente:

```
siguiente<-function(f, x, y, h)
{
  k1 = h*f(x, y)
  k2 = h*f(x+h, y+k1)
  nexty = y + (k1+k2)/2
  nextx = x + h
  return(list("x" = nextx, "y" = nexty))
}

punto4<-function(fexacta, f, x0, y0, h, n, sol)
{
  x = y = error = numeric(n+1)
  x[1] = x0
  y[1] = y0
  error[1] = abs(y[1]-fexacta(x[1]))
  for(i in 1:n)
  {

    y[i+1] = y[i] + h*(euler$y[i] + euler$y[i+1])/2
    x[i+1] = x[i] + h
    error[i+1] = abs(y[i+1]-fexacta(x[i+1]))
  }
  return(data.frame("x" = x, "y" = y, "error" = error))
}

fexacta<-function(x)
{
  return(exp(x)*((exp(-x)*x^2)+x*exp(-x)+1))
}

f<-function(x, y)
{
  return(1+x+y+x^2)
}

sol<-punto4(fexacta, f, 0, 1, 0.1, 10, euler)
lines(sol, col="red")
```

Y los resultados obtenidos son los siguientes (Euler a la izquierda y esta nueva versión de Euler):

x	y	error
0.0	1.000000	0
0.1	1.215171	2.9308e-07
0.2	1.461402	6.2589e-07
0.3	1.739858	1.0036e-06
0.4	2.051823	1.4318e-06
0.5	2.398719	1.9172e-06
0.6	2.782116	2.4668e-06
0.7	3.203750	3.0890e-06
0.8	3.665537	3.7929e-06
0.9	4.169599	4.5887e-06
1.0	4.718276	5.4881e-06

x	y	error
0.0	1.000000	0.0000000
0.1	1.110759	0.1044124
0.2	1.244587	0.2168156
0.3	1.404650	0.3352086
0.4	1.594234	0.4575905
0.5	1.816761	0.5819599
0.6	2.075803	0.7063157
0.7	2.375096	0.8286563
0.8	2.718561	0.9469802
0.9	3.110318	1.0592856
1.0	3.554711	1.1635705



Esta es la gráfica de ambos con 10 puntos (El método de Euler es el morado y el modificado es el rojo)

Podemos observar que esta vez pasa lo contrario que con la comparación pasada y esta vez es en general menor que el método de Euler. Sin embargo, la proporción de error es algo parecida (casi el 50%). Definitivamente podemos decir que para este caso el método de Euler es mucho más adecuado que cualquiera de estas modificaciones.

6 Ejercicio 6

Pruebe el siguiente código en R del método de Runge Kutta de tercer y cuarto orden. Obtenga 10 puntos de la solución con $h = 0.1$, grafíquela y compárela con el método de Euler:

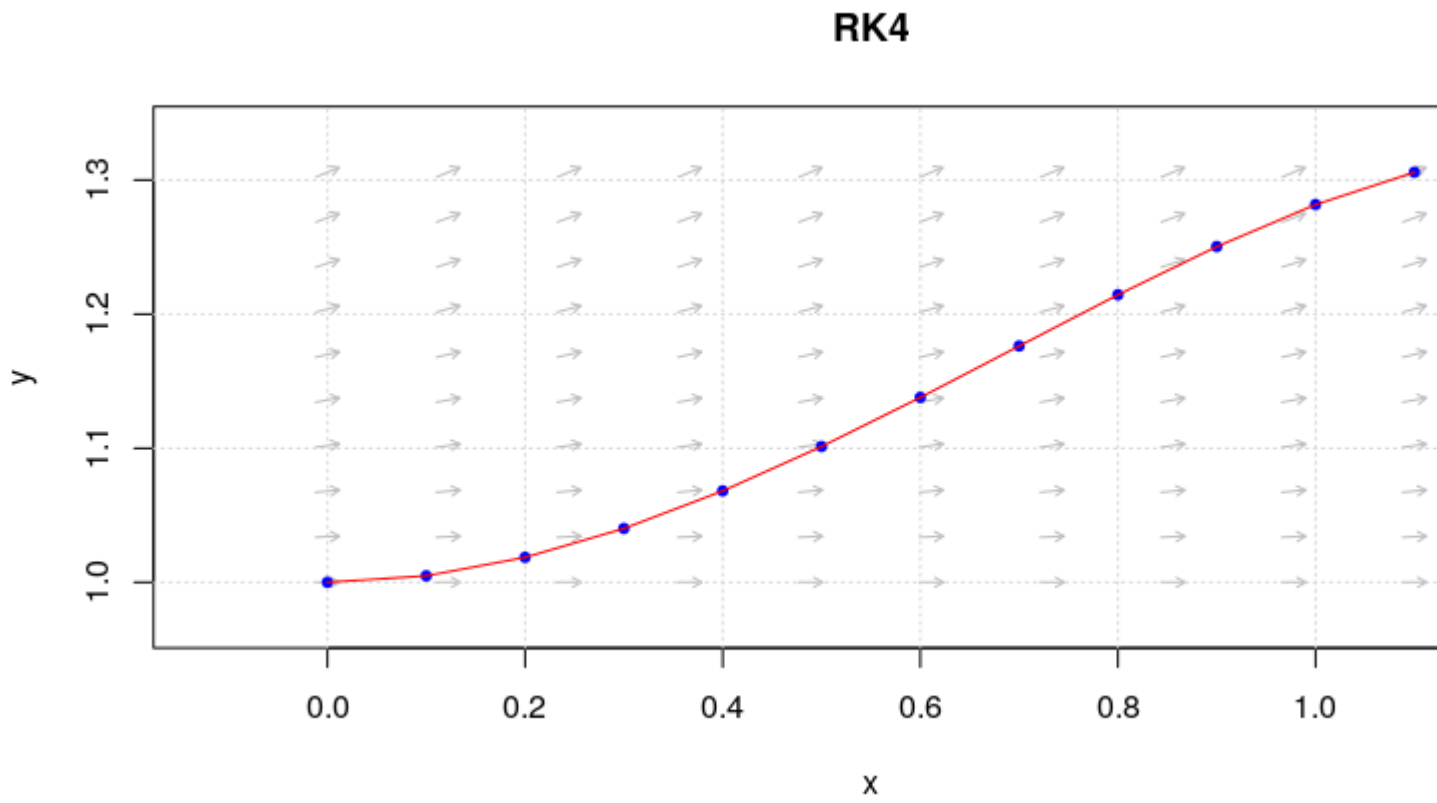
$$\frac{dy}{dx} - x - y - 1 + x^2 = 0; y(0) = 1$$

x	y	k1	k2	k3	k4	error
0	1	0	0.00475	0.0049875	0.00949875	0
0.1	1.004829	0.009482896	0.01370704	0.01391825	0.01787472	0.1055129
0.2	1.018597	0.0178597	0.02150268	0.02168483	0.02502818	0.2242085
0.3	1.040141	0.02501408	0.02801479	0.02816482	0.03083056	0.3595768
0.4	1.068175	0.03081748	0.03310835	0.0332229	0.03513977	0.5154746
0.5	1.101278	0.03512781	0.0366342	0.03670952	0.03779876	0.6961645
0.6	1.13788	0.03778804	0.03842744	0.03845941	0.03863398	0.9063572
0.7	1.176246	0.03862464	0.03830587	0.03828993	0.03745363	1.151259
0.8	1.214458	0.0374458	0.03606809	0.03599921	0.03404572	1.436624
0.9	1.250396	0.03403957	0.03149155	0.03136415	0.02817598	1.768811
1	1.281717	0.02817169	0.02433027	0.0241382	0.01958551	2.154847

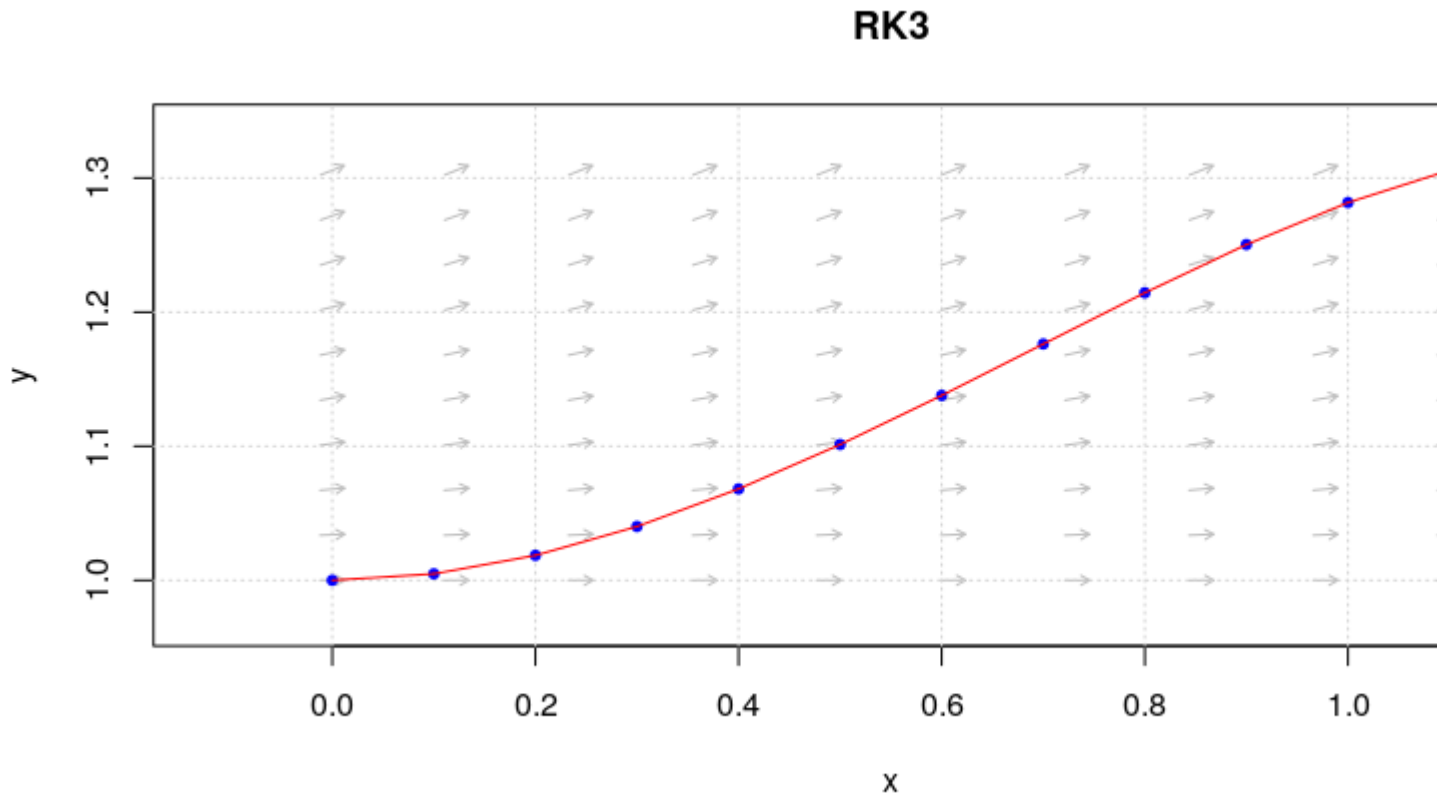
x	y	k1	k2	k3	error
0	1	0	0.00475	0.00995	0
0.1	1.004825	0.0094825	0.01370663	0.01827557	0.1055168
0.2	1.018589	0.01785891	0.02150186	0.02537339	0.2242164
0.3	1.040129	0.0250129	0.02801355	0.03111432	0.3595886
0.4	1.068159	0.03081593	0.03310673	0.03535568	0.5154901
0.5	1.101259	0.0351259	0.0366322	0.03793975	0.6961835
0.6	1.137858	0.03778581	0.0384251	0.03869225	0.9063795
0.7	1.176221	0.03862212	0.03830323	0.03742055	1.151284
0.8	1.21443	0.03744305	0.0360652	0.03391178	1.436651
0.9	1.250366	0.03403664	0.03148847	0.02793067	1.76884
1	1.281687	0.02816866	0.02432709	0.01921721	2.154877

La implementación del método de Rungge-Kutta utilizada fue la siguiente:

```
rk4<-function(dy, ti, tf, y0, h, graficar=TRUE, numpendientes=10){
  t<-seq(ti, tf, h)
  y<-c(y0)
  for(i in 2:length(t)){
    k1=h*f(dy, t[i-1], y[i-1])
    k2=h*f(dy, t[i-1]+h/2, y[i-1]+k1*(0.5))
    k3=h*f(dy, t[i-1]+h/2, y[i-1]+k2*(0.5))
    k4=h*f(dy, t[i-1]+h, y[i-1]+k3)
    y<-c(y, y[i-1]+1/6*(k1+2*k2+2*k3+k4))
  }
}
rk3<-function(dy, ti, tf, y0, h, graficar=TRUE, numpendientes=10){
  t<-seq(ti, tf, h)
  y<-c(y0)
  for(i in 2:length(t)){
    k1=h*f(dy, t[i-1], y[i-1])
    k2=h*f(dy, t[i-1]+h/2, y[i-1]+k1*(0.5))
    k3=h*f(dy, t[i-1]+h, y[i-1]-k1+2*k2)
    y<-c(y, y[i-1]+1/6*(k1+4*k2+k3))
  }
}
```



Este es el campo de pendientes de la solución obtenida con método de Runge-Kutta de cuarto orden.



Este es el campo de pendientes de la solución obtenida con método de Runge-Kutta de tercer orden.