Taller EDO

Juliana García y Gabriel Niño

Problema

8.2 Considere el problema de valor inicial $y' = t\exp(3t) - 40y$, $t \in [1,2]$, $y(1) = 10$.

a.) Usando el método de Euler, aproximar $y(0.4)$ con $h = 0.1$.

b.) Usando el método de Taylor de orden 4, aproximar $y(0.4)$ con $h = 0.2$

c.) Usando el método de Runge-Kutta de orden 4, aproximar $y(0.4)$ con $h = 0.2$

Ecuación Diferencial – Solución por WolframAlpha

$$Y' = xe^{3x} - 40y, x \in [1,2], y(1)=10$$

WolframAlpha computational intelligence.

[//math:y'(x)=x*e^(3*x)-40*y, y(1)=10//]

Extended Keyboard Upload

Input:

$\{y'(x) = x\,e^{3x} - 40\,y(x),\ y(1) = 10\}$

ODE classification:

first-order linear ordinary differential equation

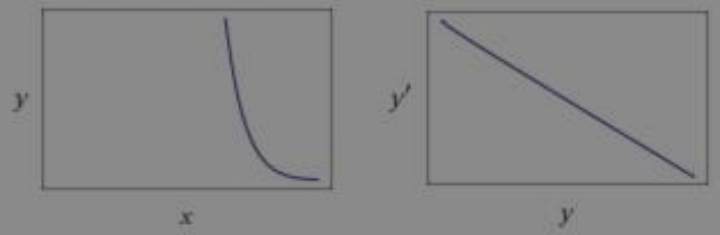
Alternate form:

$\{e^{3x}\,x = y'(x) + 40\,y(x),\ y(1) = 10\}$

Differential equation solution:

$$y(x) = \frac{e^{-40x}\left(e^{43x}(43x-1) - 42\,e^{43} + 18\,490\,e^{40}\right)}{1849}$$

Plots of the solution:



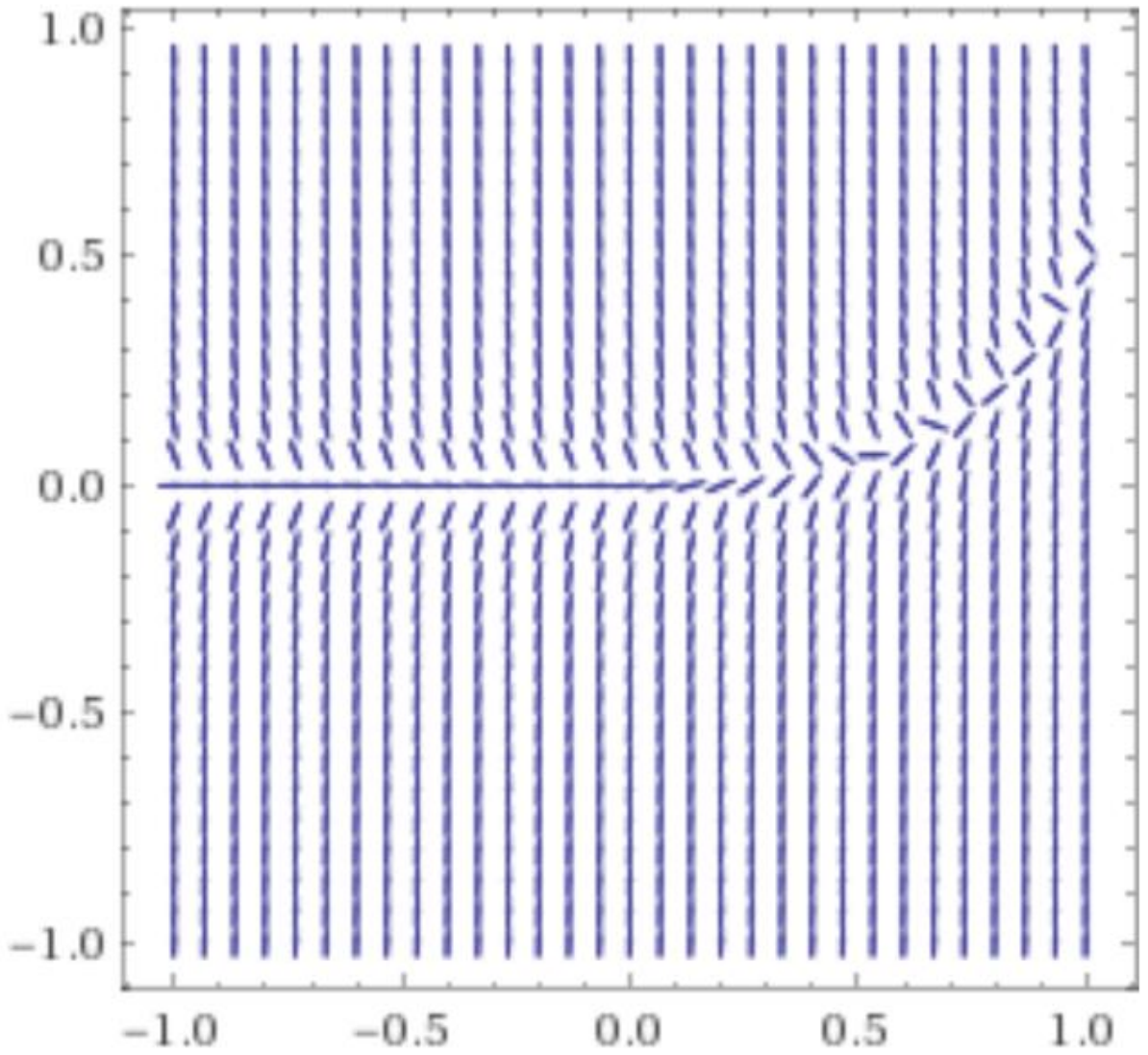
Taylor orden 4

Campo de Pendientes

WolframAlpha computational intelligence

taylor series y'=t*exp(3*t)-40*y

Slope field:



Algoritmo parte 1

```
rm(list=ls())
options(digits = 16)
#Funcion que implementa la aproximacion Taylor donde:
#f es la expresion de la EDO, t0 y y0 son los valores iniciales de t y y respectivamente,
#h representa la magnitud del paso, a y b son el dominio de t tal que t pertenece a [a,b]
#m es el orden de la aproximacion Taylor
taylor<-function(f,t0,y0,h,m,a,b){
  #Derivadas parciales de f respecto a t y y
  fdt<-c(f,D(f,"t")); fdy<-c(f,D(f,"y"))
  for(i in 3:m){
    fdt<-c(fdt,D(fdt[[i-1]],"t")); fdy<-c(fdy,D(fdy[[i-1]],"y"))
  }
  t<-t0 ; y<-y0; E<-c(abs(y0 - g(t0))) #Error absoluto y0 y la funcion g en t0
  valt<-c(t) ; valy<-c(y) ; valg<-c(g(t0))
  miny <- y ; maxy<-0 ; n <- (b-a)/h #n es la cantidad de pasos
  #La aproximacion de Taylor se calcula para cada paso hasta n
  for(k in 1:n){
    resp<-y + h*eval(f) #Se inicializa para aproximar Taylor
    #Se completa la aproximacion de Taylor
    for(i in 2:m){
      resp<-resp+((h^(i))/factorial(i)) * (eval(fdt[[i]]) + eval(fdy[[i]]) * eval(f))
    }
```

Algoritmo parte 2

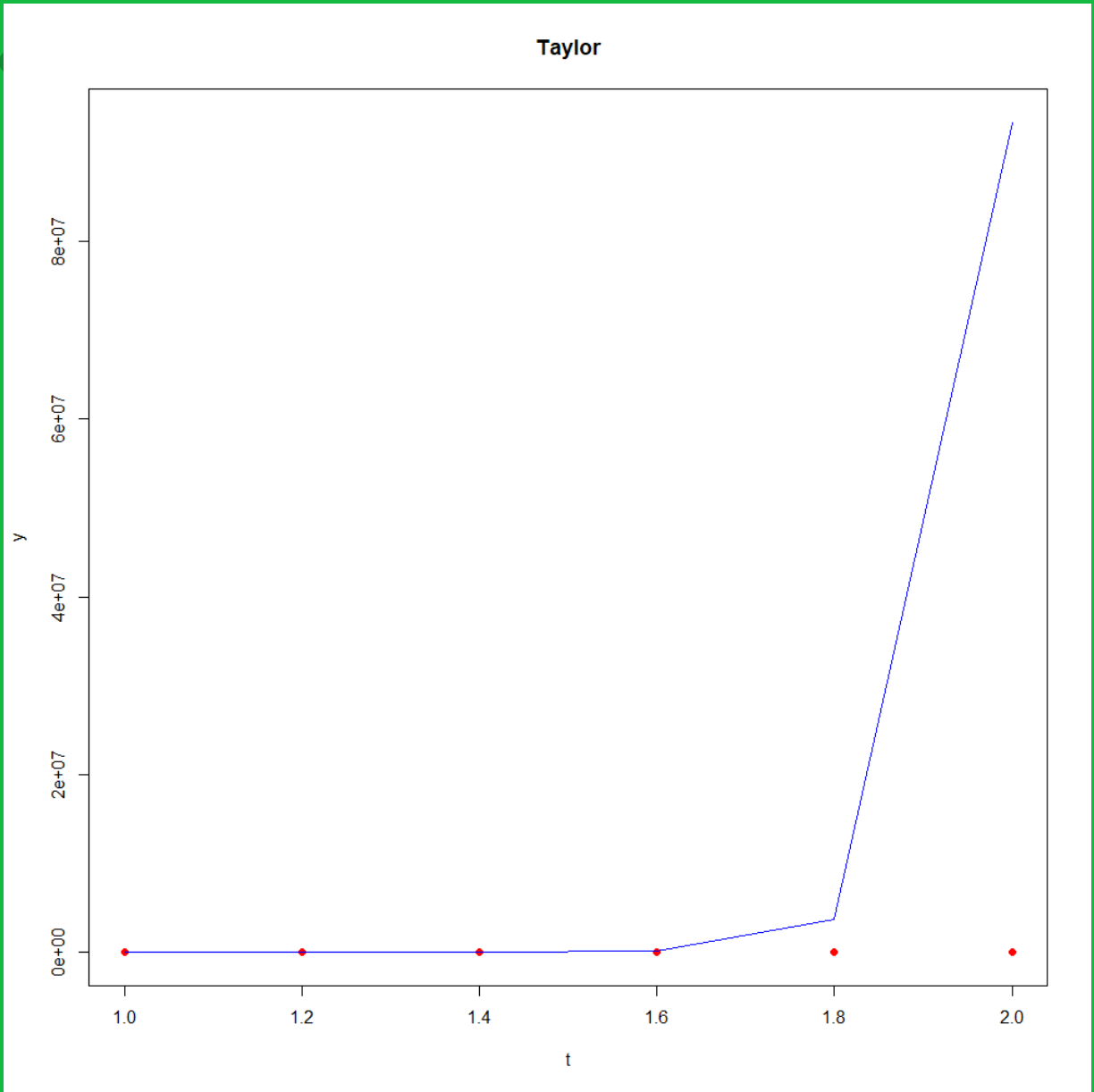
```
    }
    t<-t + h ; y<-resp ; valt<-c(valt,t) ; valy<-c(valy,y)
    E<-c(E,abs(y - g(t))) #Error absoluto entre la aproximacion de Taylor y la solucion
    valg<- c(valg, g(t)) ; if(y > maxy){ maxy<-y } ; if(y < miny){ miny<-y }
  }
  #Los puntos rojos representan la solucion del problema
  plot(valt,valg,main=paste("Taylor"),xlab="t"
       ,ylab="y",xlim = c(a,b),ylim = c(miny,maxy),col="red", pch = 16)
  #La linea azul representa la aproximacon de Taylor
  points(valt,valy, type="l", col = "blue")
  resultados<-data.frame("i"=(0:n),"t"=valt,"y"=valy, "Solucion" = valg ,"Error"=E)
  resultados
}
```

Resultados de Taylor

```
#Expresion de la EDO del problema
f = expression(t*exp(3*t) - 40*y)
#Ecuacion solucion calculada con Wolfram
g = function(t){
  exp(-40*t)*(exp(43*t)*(43*t-1)-42*exp(43)+18490*exp(40))/1849
}
#Prueba de la funcion Taylor
taylor(f,1,10,0.2,4,1,2)
```

i	t	y	Solucion	Error
0	1.0	1.000000000000000e+01	10.000000000000000	0.000000000000000e+00
1	1.2	2.400295394713297e+02	1.004754122804794	2.390247853485249e+02
2	1.4	5.978719525012551e+03	2.135117784458787	5.976584407228092e+03
3	1.6	1.494208488916276e+05	4.455601032145178	1.494163932905954e+05
4	1.8	3.735422541250415e+06	9.148431691725241	3.735413392818723e+06
5	2.0	9.338536042187043e+07	18.545942372570295	9.338534187592806e+07

Gráfica Resultante



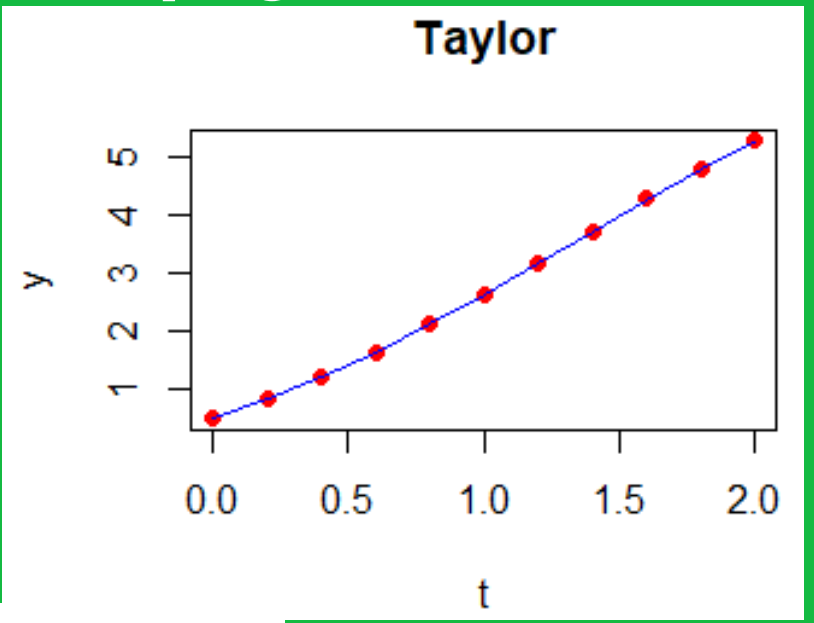
Los puntos rojos representan la solución de la EDO y la línea azul representa la aproximación de Taylor.

Ejercicios Adicionales de Taylor

Hechos con el objetivo de verificar de que la función “taylor” logra aproximar la respuesta. Fueron tomados de las referencias [1] y [2].

Resultado ejercicio 1 [1] pg.7

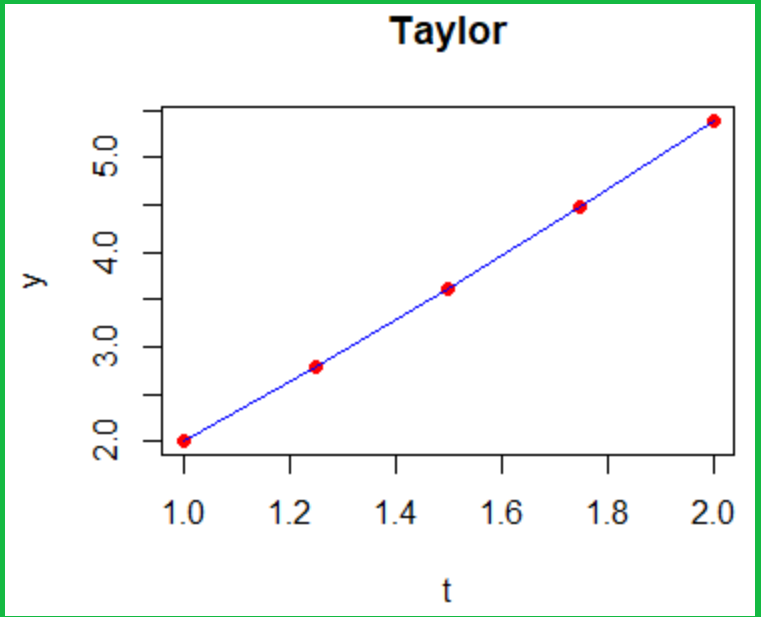
```r
#Expresion de la EDO del problema
f = expression(y - t^2 + 1)
#Ecuacion solucion calculada con Wolfram
g = function(t){(t+1)^2 - 0.5*exp(t)}
#Prueba de la funcion Taylor
taylor(f,0,0.5,0.2,4,0,2)
```



i	t	y	Solucion	Error
0	0.000000000000000	0.500000000000000	0.500000000000000	0.000000000000000
1	0.200000000000000	0.827333333333334	0.829298620919915	0.001965287586581654
2	0.400000000000000	1.209880000000001	1.214087651179365	0.004207651179364502
3	0.600000000000001	1.642186933333334	1.648940599804746	0.006753666471412512
4	0.800000000000000	2.117601392000001	2.127229535753766	0.009628143753766416
5	1.000000000000000	2.628007031573330	2.640859085770478	0.012852054197144636
6	1.200000000000000	3.163501911852796	3.179941538631727	0.016439626778927519
7	1.399999999999999	3.712005665793746	3.732400016577663	0.020394350783913939
8	1.599999999999999	4.258780245601706	4.283483787802441	0.024703542200733786
9	1.799999999999998	4.785845232967416	4.815176267793527	0.029331034826110880
10	1.999999999999998	5.271264517553581	5.305471950534676	0.034207432981093966

Resultado ejercicio 2 [2] pg.20

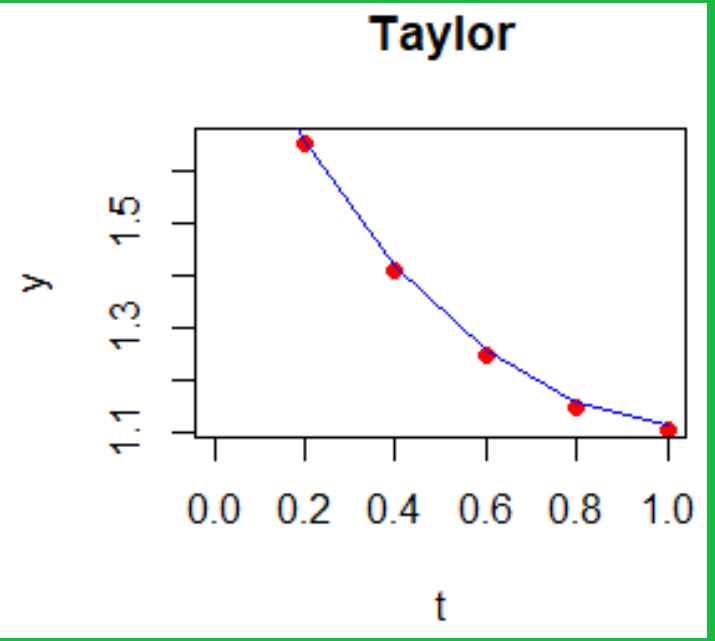
```r
#Expresion de la EDO del problema
f = expression(1 + y/t)
#Ecuacion solucion calculada con Wolfram
g = function(t){t*(log(t) + 2)}
#Prueba de la funcion Taylor
taylor(f,1,2,0.25,2,1,2)
```



i	t	y	Solucion	Error
0	1.00	2.000000000000000	2.000000000000000	0.000000000000000000
1	1.25	2.781250000000000	2.778929439142762	0.002320560857238085
2	1.50	3.612500000000000	3.608197662162247	0.004302337837752912
3	1.75	4.485416666666667	4.479327628886990	0.006089037779676509
4	2.00	5.394047619047619	5.386294361119891	0.007753257927728185

Resultado ejercicio 3 [1] pg.5

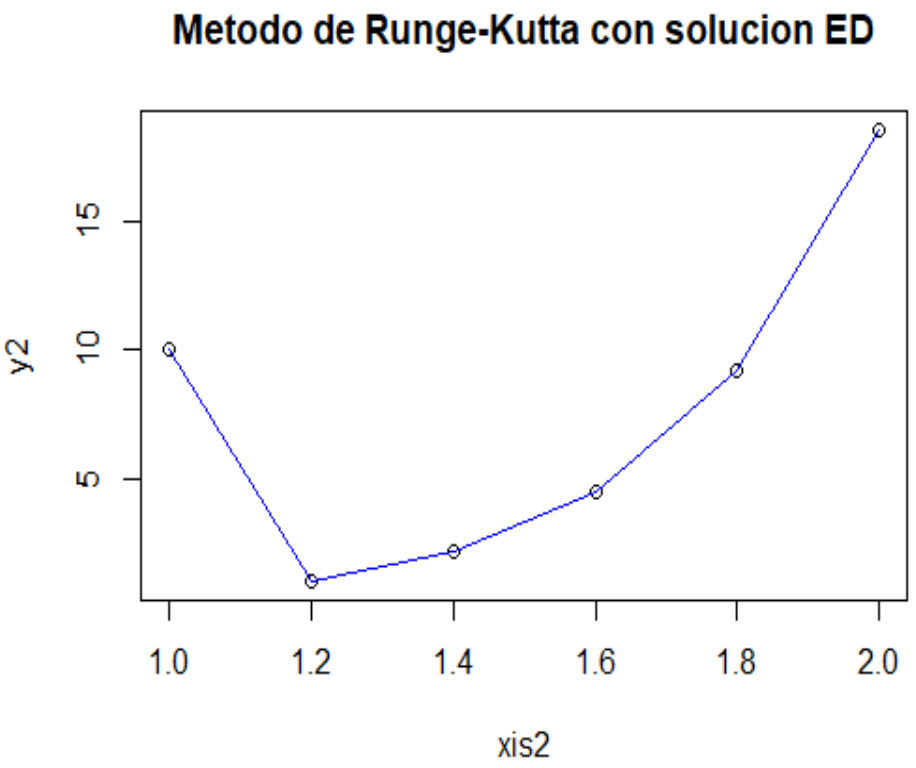
```r
#Expresion de la EDO del problema
f = expression(t - y)
#Ecuacion solucion calculada con Wolfram
g = function(t){t + 3*exp(-t) - 1}
#Prueba de la funcion Taylor
taylor(f,0,2,0.2,4,0,1)
```



i	t	y	Solucion	Error
0	0.000000000000000	2.0000000000	2.00000000000000	0.00000000000000000
1	0.200000000000000	1.6600000000	1.65619259233946	0.00380774076605439
2	0.400000000000000	1.4172000000	1.41096013810691	0.00623986189308234
3	0.600000000000001	1.2541040000	1.24634908282079	0.00766909171792074
4	0.800000000000000	1.1563652800	1.14798689235166	0.00837838764833498
5	1.000000000000000	1.1122195296	1.10363832351432	0.00858120685673143

Runge-Kutta orden 4

Graficando la Solución de la E.D

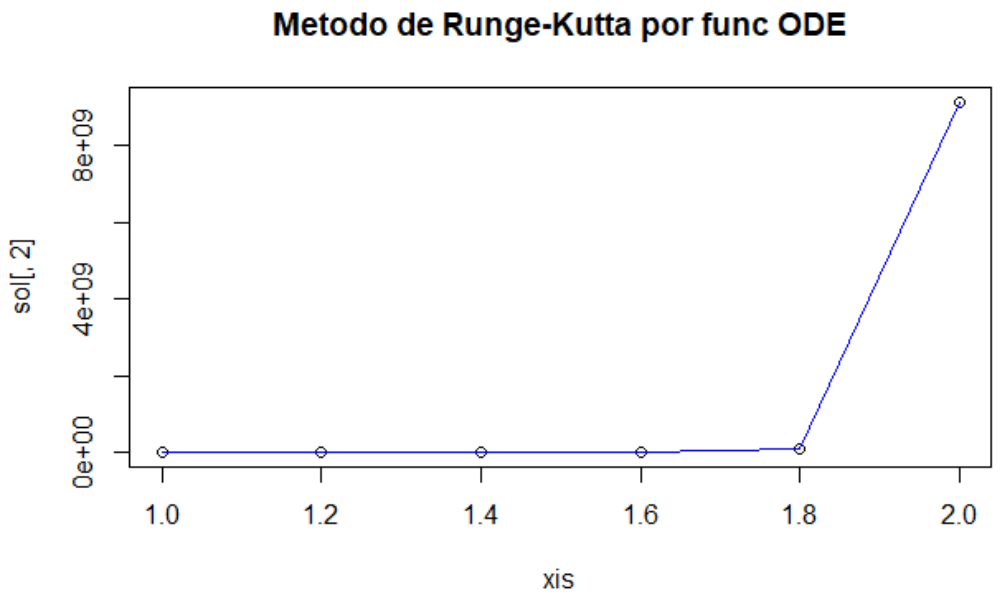



```
> solucionED = function(x){(exp(1)^(-40*x)*(exp(1)^(43*x)*(43*x-1)-42*exp(1)^(43)+18490*exp(1)^(4
0)))/1849}
> options(digits = 16)
> xis2 = seq(from = 1, to = 2,by = 0.2)
>
> y2=c()
> h2=1
> i=1
> while(i <= 6){
+   y2[i] = solucionED(h2)
+   h2 = h2+0.2
+   i = i + 1
+ }
> data.frame (x=xis2, y=y2)
    x                  y
1 1.0 10.000000000000000
2 1.2  1.004754122804794
3 1.4  2.135117784458786
4 1.6  4.455601032145178
5 1.8  9.148431691725239
6 2.0 18.545942372570288
>
> plot(xis2, y2, type = "p", main="Metodo de Runge-Kutta con solucion ED")
> lines(xis2, y2, col= "blue")
```

Se toma como referencia la ayuda en la pg.69 del libro “Cómo utilizar R en métodos numéricos” [3]

Usando la función **ode()** del paquete **deSolve** el código sería,

```
#install.packages("deSolve")
 require(deSolve)
# La función ode() requiere obligatoriamente, varias cosas que debemos agregar
# ode(valores iniciales, tis , func, parms, method, ...)

fp = function(t,y, parms){
 s = -1.68*10^(-9)*y^4+2.6880
 return(list(s))                          # ode requiere salida sea una  lista
}

tis= seq(0,200,200/20)
# Usamos la función ode()
sol = ode(c(180), tis, fp, parms=NULL, method = "rk4") # método Runge Kutta orden 4

# Salida
tabla = cbind(xis, solnumerica[,2] )
```



Resultado por consola

Console | Terminal | Jobs

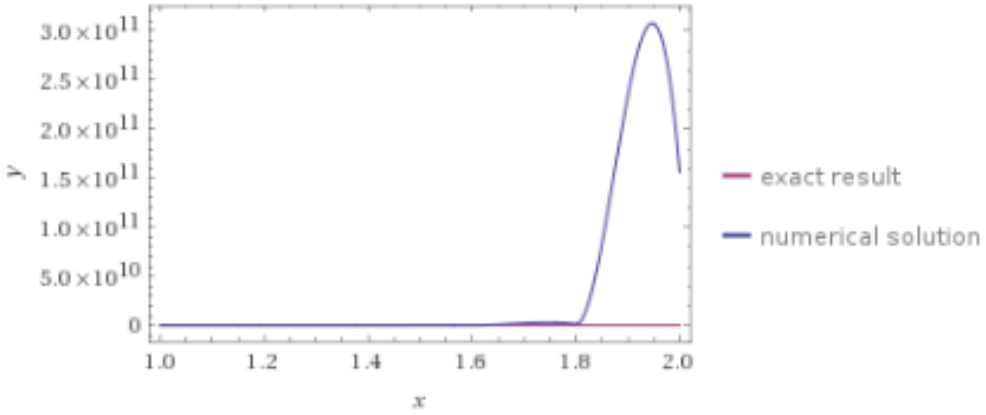
C:/Users/gabri/Desktop/Analisis Número/

```
> rm(list=ls())
>
> require(deSolve)
> require(PolynomF)
>
>
> fp = function(x,y, parms){
+   s = ((x*exp(1)^(3*x))-40*y)
+   return(list(s)) # ode requiere salida sea una lista
+ }
>
> xis = seq(from = 1, to = 2,by = 0.2)
>
> sol = ode(c(1,10),xis,fp,parms = NULL, method = "rk4")
> options(digits = 16)
> tabla = cbind(xis, sol[,2])
> tabla
     xis
[1,] 1.0 1.000000000000000e+00
[2,] 1.2 6.271952114995185e+01
[3,] 1.4 6.815196592310100e+03
[4,] 1.6 7.517193160739151e+05
[5,] 1.8 8.293922955214156e+07
[6,] 2.0 9.150960697755255e+09
> plot(xis, sol[,2], type = "p", main="Metodo de Runge-Kutta por func ODE")
> lines(xis, sol[,2], col= "blue")
>
```

Input interpretation:

solve	$y'(x) = e^{3x} x - 40\, y(x)$ $y(1) = 10$	using fourth-order Runge-Kutta method	with a stepsize of 0.2 from $x = 1$ to 2

Solution plot: Show error plot



Stepwise results:

step	x	y	local error	global error
0	1	10	0	1.77636×10^{-15}
1	1.2	1055.72	−1054.71	−1054.71
2	1.4	116 376.	−3.86551	−116 374.
3	1.6	1.28399×10^{7}	−7.07778	-1.28399×10^{7}
4	1.8	1.41667×10^{9}	−14.1315	-1.41667×10^{9}
5	2	1.56306×10^{11}	−28.	-1.56306×10^{11}

Definitions »

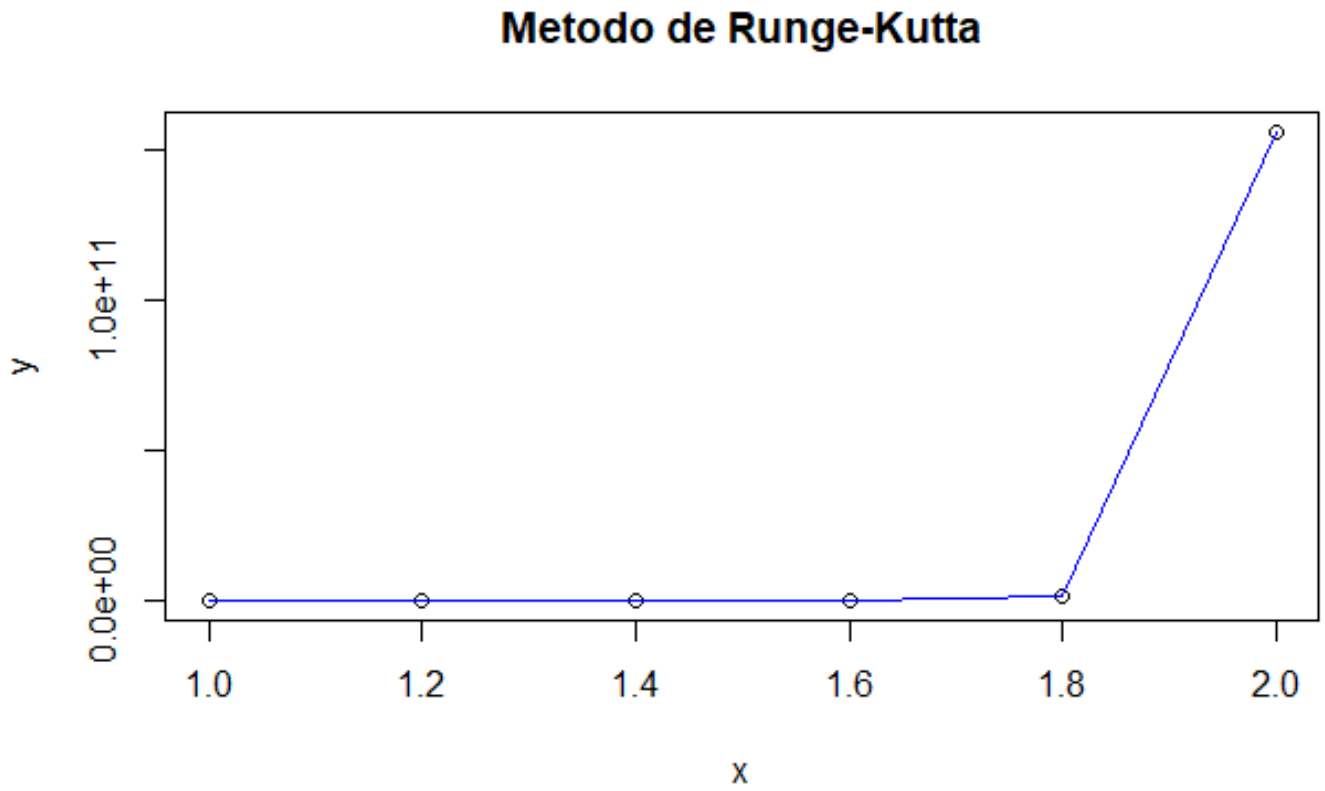
Solución de WolframAlpha

Ahora de la forma “manual”

Console | Terminal | Jobs

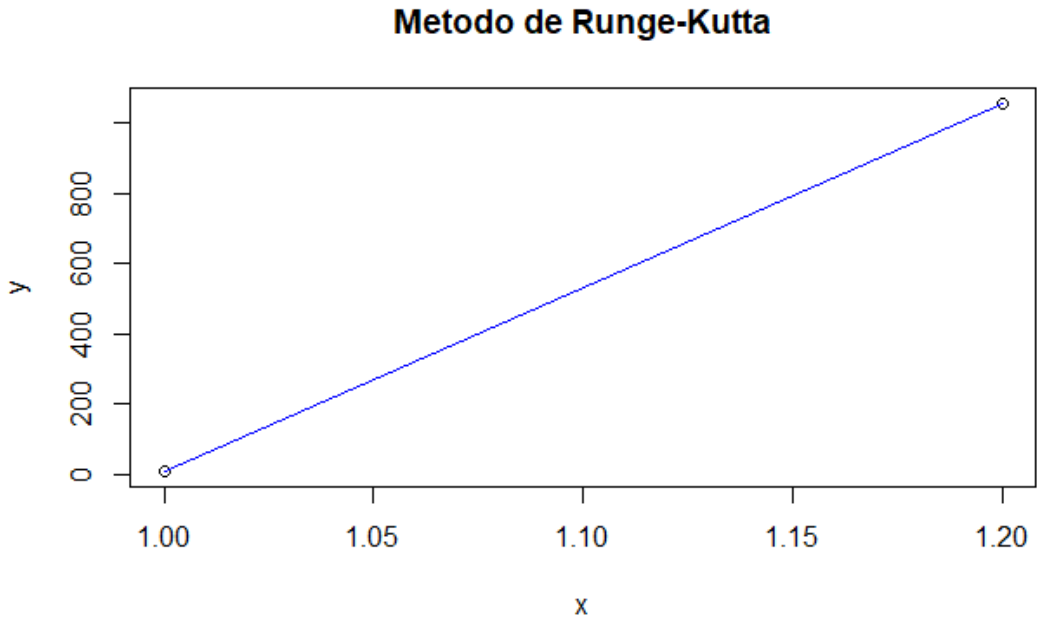
C:/Users/gabri/Desktop/Analisis Número/

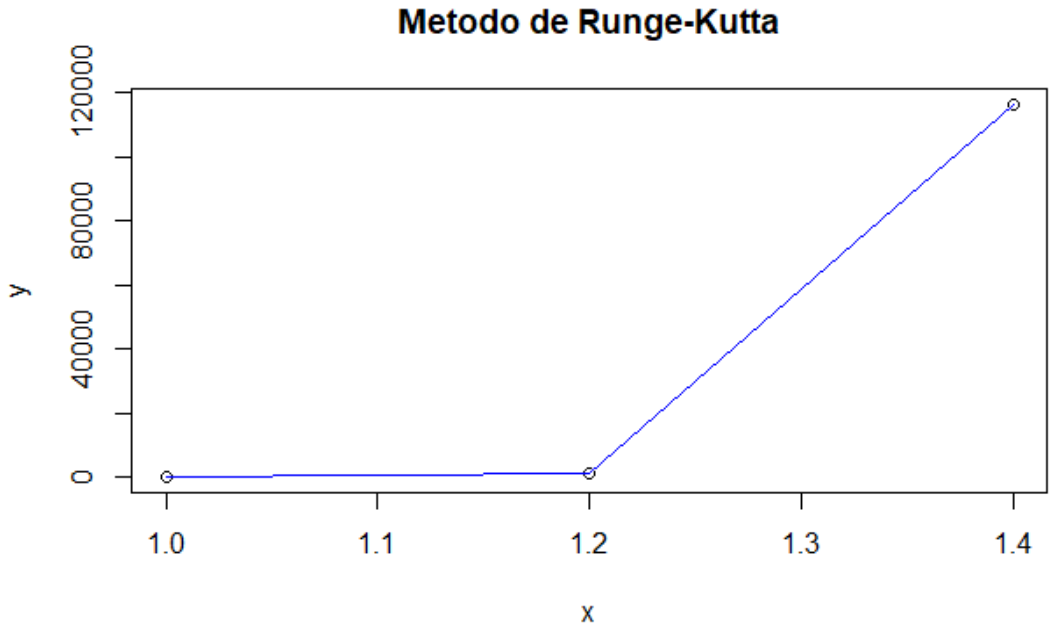
```
> f<-function(x,y){x*exp(1)^(3*x)-40*y}
> x<-c(1)
> y<-c(10)
> h<-0.2
> error<-c(0)
>
> for(i in 1:5)
+ {
+   k1<-h*f(x[i],y[i])
+   k2<-h*f(x[i]+h/2,y[i]+(k1/2))
+   k3<-h*f(x[i]+h/2,y[i]+(k2/2))
+   k4<-h*f(x[i]+h, y[i]+k3)
+   y<-c(y,y[i]+1/6*(k1+2*k2+2*k3+k4))
+   x<-c(x,x[i]+h)
+   #error<-c(error, abs(y[i]-sol[i,2]))
+ }
> data.frame (x=x, y=y)
    x                   y
1 1.0 1.000000000000000e+01
2 1.2 1.055719521149952e+03
3 1.4 1.163761965923101e+05
4 1.6 1.283994964940725e+07
5 1.8 1.416673976329920e+09
6 2.0 1.563063610922368e+11
>
> plot(x, y, type = "p", main="Metodo de Runge-Kutta")
> lines(x, y, col= "blue")
>
```



¿Qué sucede en la gráfica?

Aunque entre los intervalos 1.0 a 1.8 la grafica se vea plana, en realidad lo que sucede es que pega unos saltos muy pero muy grandes (observar graficas a la derecha) y es debido a esto que la grafica se ve de aquella forma.



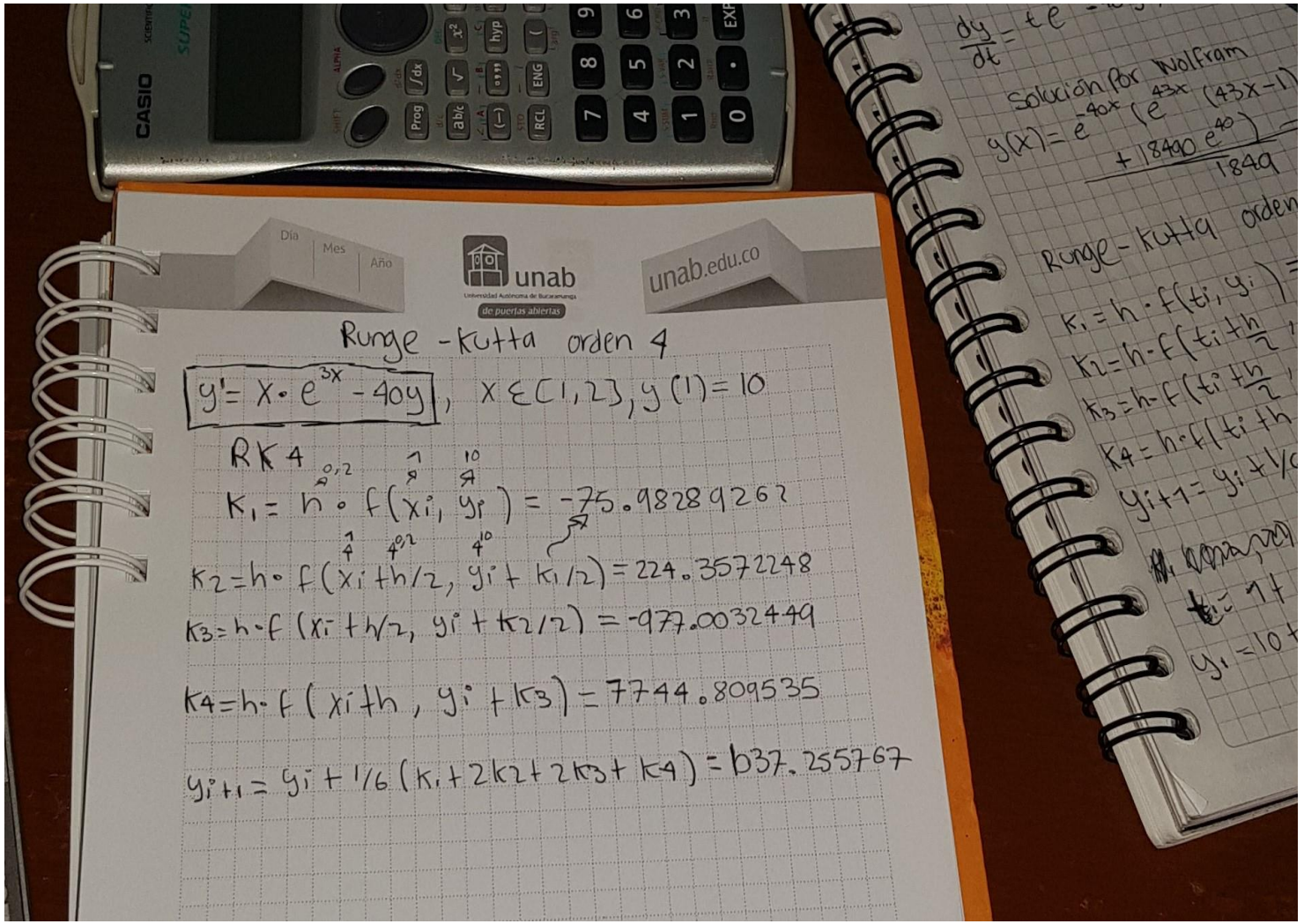


Comparación Rstudio vs Casio fx-3950P

Console | Terminal | Jobs

C:/Users/gabri/Desktop/Analisis Númerico/

```
imprimiendo iteracion:  1

imprimiendo y[i] 10
imprimiendo k1 -75.98289261536247
imprimiendo k2 229.8963510239946
imprimiendo k3 -993.6206235334338
imprimiendo k4 7877.748564533952
-----------------
imprimiendo iteracion:  2

imprimiendo y[i] 1055.719521149952
imprimiendo k1 -8436.972592933133
imprimiendo k2 25314.97883930035
imprimiendo k3 -109692.8268896336
imprimiendo k4 869115.5311205607
-----------------
imprimiendo iteracion:  3

imprimiendo y[i] 116376.1965923101
imprimiendo k1 -930990.9005657895
imprimiendo k2 2792981.034664067
imprimiendo k3 -12102906.70625536
imprimiendo k4 95892282.96063803
-----------------
imprimiendo iteracion:  4

imprimiendo y[i] 12839949.64940725
imprimiendo k1 -102719558.3119244
imprimiendo k2 308158691.8198881
imprimiendo k3 -1335354308.707362
imprimiendo k4 10580114952.16995
-----------------
imprimiendo iteracion:  5

imprimiendo y[i] 1416673976.32992
imprimiendo k1 -11333391730.93305
imprimiendo k2 34000175226.66245
```



Observaciones

- Es importante mencionar que los resultados que coincidieron con la respuesta dada por el programa WolframAlpha (diapositiva 18), fueron los resultados de la forma “manual” (diapositiva 19) pues como se observa en ambas tablas las cifras coinciden a pesar que WolframAlpha quita algunas cifras decimales.
- También es importante preguntarnos porque los resultados arrojados por la función ODE son distintos a aquellos arrojados de la forma manual, sin embargo, ambas graficas son iguales.
- Adicional a esto, se quiso hacer una pequeña comparación frente a los resultados obtenidos por medio del programa Rstudio y los resultados obtenidos hechos a mano utilizando una calculador común y corriente.

Referencias

[«INTRODUCCIÓN A LOS MÉTODOS NUMÉRICOS,» 8 octubre 2013. [En línea]. Available: 1http://matema.ujaen.es/jnavas/web_modelos/problemas/ptema7.pdf.
]
[Palacios, Francisco; Escuela Politécnica Superior de Ingeniería de Manresa; Universidad Politécnica de Cataluña, «Métodos 2Numéricos: Resumen y ejemplos,» Marzo 2008. [En línea]. Available:
] http://www.eupm.upc.edu/~fpq/numerico/resum/num-edos-resum.pdf.
[«Cómo utilizar R en métodos numéricos,» p. 69.
3
]


Muchas Gracias