Package 'metnum'

November 13, 2018

Type Package

Title MetodosNumericos

Version 1.0.0

Author Valentina Escobar, Pedro Escobar (Métodos de Simpson)
Santiago Chaparro, Briam Agudelo (NDerivación de N Puntos)
Diego Barajas, Brandonn Cruz (Métodos Richardson)
Andrés Felipe Mariño, Javier Esteban Marín (Spline Cúbico)
David molano, Dorian Moreno (Interpolación Parámetrica de Lagrange)
Maria Fernanda Garces, Juan Sebastian Leon (SMRoot)
Andrés Mendoza, Camilo serrano (Método Newton) <biseccion.R, newton.R>
Daniela Beltran Saavedra, Nicolas Duarte Ospina (Punto Fijo, Posición Falsa, Secante)
Jonathan Esteban Molina Castañeda, Felipe Andrés Gutierrez Naranjo (Euler para EDO)
Pablo Millan, Oscar Fonseca (Euler para EDO No Lineal)

Maintainer There is no actual mantainer <yourself@somewhere.any>

Description La funcion de este paquete es contener métodos númericos para hallar raices, derivar, integrar, interpolar y resolver ecuaciones diferenciales.

License LGPL (>=v3)

Encoding UTF-8

LazyData true

RoxygenNote 6.1.0

Imports ggplot2, rSymPy, PolynomF, pracma, graphics, phaseR

R topics documented:

`cincoPuntos` *Halla el valor de la derivada en un punto*

Description

Halla el valor de la derivada por el método de cinco puntos de una expresion en un punto dado, con la restricción de un error o un h específico(según sea el caso).

Usage

```
cincoPuntos(funcion, punto, dato, tipo)
```

Arguments

`punto`	punto en el que se evaluará la derivada
`dato`	puede representar el valor de la variación x(h) o el error con el que se quiere calcular el valor de la derivada.
`tipo`	especifíca si el anterior parametro(dato) corresponde a cualquiera de los dos datos que puede representar: TRUE=el dato representa el error, FALSE=el dato representa h.
`expresion`	expresion a la que se le calculará la derivada

Value

el valor de la derivada de la expresion en el punto dado.

Examples

```
expresion=expression((x^2)+x)
punto=0
dato=1*10^(-5)
cincoPuntos(expresion,punto,dato,FALSE)
```

`derivadaRichardson` *derivadaRichardson(x,y)*

Description

Calcula la derivada respecto a x de los valores X y Y, exceptuando el primer y último valor de x, depende de h

Usage

```
derivadaRichardson(x, y, h = 0.01, n = 3)
```

Arguments

x	Lista de valores x de una funcion
y	Lista de valores y de una funcion
h	Valor a incrementar
n	Numero de iteraciones

Value

Derivada de los valores de x y calculo del error

Examples

```
x = seq(0, 1, by = 0.1)
y = seq(0, 1, by = 0.1)
datos = derivada(x, y, h = 0.01, n = 3)
datos$resultados
```

dosPuntos *Halla el valor de la derivada en un punto*

Description

Halla el valor de la derivada por el método de dos puntos de una expresion en un punto dado, con la restricción de un error o un h específico(según sea el caso).

Usage

```
dosPuntos(expresion, punto, dato, tipo)
```

Arguments

expresion	expresion a la que se le calculará la derivada
punto	punto en el que se evaluará la derivada
dato	puede representar el valor de la variación x(h) o el error con el que se quiere calcular el valor de la derivada.
tipo	especifíca si el anterior parametro(dato) corresponde a cualquiera de los dos datos que puede representar: TRUE=el dato representa el error, FALSE=el dato representa h.

Value

el valor de la derivada de la expresion en el punto dado.

Examples

```
expresion=expression((x^2)+x)
punto=0
dato=1*10^(-5)
dosPuntos(expresion,punto,dato,FALSE)
```

dy *Funcion dy*

Description

Funcion que evalua una expresion dada, utilizando los valores de "x" y "y" que recibe.

Usage

```
dy(edo, x, y)
```

Arguments

edo	La expresion que sera evaluada. DEBE escribirse con la funcion expression de R.
x	El valor de x que sera computado en la funcion
y	El valor de y que sera computado en la funcion

Value

Ninguno

`error` *Error Función*

Description

Esta función encuentra el error de la solución comparando la solución analítica del sistema que ya se conoce, con el valor obtenido en la solución.

Usage

```
error(solucion)
```

Arguments

`solucion`	La solución tanto en el eje x como en el eje y así como el valor t que se evaluó para encontrar dicha solución.

Details

La solución analítica que se encuentra en la función hace referencia al sistema de "-x+y+1" y "x-y".

Value

El valor retornado es un arreglo de tamaño 2, que contiene el error tanto en el eje x como el error en el eje y.

Author(s)

Pablo Millan y Oscar Fonseca

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (solucion)
{
    t = solucion[3]
    xanalitico = (1/4) * (2 * t - 9 * exp(-2 * t) - 3)
    yanalitico = (1/4) * (2 * t + 9 * exp(-2 * t) - 5)
    xerror = abs(xanalitico - solucion[1])
    yerror = abs(yanalitico - solucion[2])
    errores <- c(xerror, yerror)
    return(errores)
  }
```

errorEuler *Funcion errorEuler*

Description

Funcion que calcula los resultados exactos de una ecuacion diferencial ordinaria de primer orden por medio del metodo de Euler disponible en R, hayando el error absoluto a comparacion de los resultaods obtenidos por medio de la funcion metodoEuler

Usage

```
errorEuler(en)
```

Arguments

en	dataframe que contiene los resultados de la funcion metodoEuler

Value

Ninguno

Field *Field Función*

Description

Esta función representa el campo que se va a graficar.

Usage

```
Field(t, y_in, parameters)
```

Arguments

t	Variable independiente.
y_in	Variables
parameters	Parámetros

Value

Retorna una lista que conforma al campo que se va a graficar.

Author(s)

Pablo Millan y Oscar Fonseca

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (t, y_in, parameters)
{
    dy = numeric(length(y_in))
    for (i in 1:length(dy)) {
        for (j in 1:length(dy)) {
            assign(vars[j], y_in[j])
        }
        dy[i] = eval(parse(text = parameters[i]))
    }
    return(list(dy))
  }
```

`graficarEuler` *Funcion graficarEuler*

Description

Funcion que grafica la solucion de una ecuacion diferencial ordinaria de primer orden, el campo vectorial en que esta inmersa, graficando los puntos necesarios, y utiliza la funcion errorEuler para mostrar los errores.Hace uso de una funcion embedida para graficar el campo vectorial.

Usage

```
graficarEuler(en, edo)
```

Arguments

en	dataframe que contiene los resultados de la funcion metodoEuler
edo	la expresion a evaluar por medio de la funcion dy

Value

Ninguno

`graficarsimpson` *graficarsimpson*

Description

Genera la grafica de la funcion integrada por el metodo de Simpson

Usage

```
graficarsimpson(func, n, a, b, titulo = NULL, ejex = NULL,
  ejey = NULL)
```

Arguments

func	Funcion a graficar por medio del metodo de Simpson
n	Numero de particiones para graficar
a	Limite inferior de la integral
b	Limite superior de la integral
titulo	Opcional. Título de la gráfica
ejex	Opcional. Eje horizontal de la grafica
ejey	opcional. Eje vertical de la grafica

Value

Imprime una grafica con las aproximaciones hechas mediante el metodo de Simpson

---

graficartrap *graficartrap*

Description

Calcula la integral de acuerdo a los lineamientos del metodo de Simpson (Parabolas)

Usage

```
graficartrap(func, n, a, b, titulo = NULL, ejex = NULL, ejey = NULL)
```

Arguments

func	Funcion a graficar por medio del metodo de Simpson
n	Numero de particiones para graficar
a	Limite inferior de la integral
b	Limite superior de la integral
titulo	Opcional. Título de la gráfica
ejex	Opcional. Eje horizontal de la grafica
ejey	opcional. Eje vertical de la grafica

Value

Genera la grafica de los trapecios correspondientes a la integral

hallarErrorCincoPuntos

Halla el error del método de cinco puntos

Description

Halla el error de truncamiento al calcular el valor de la derivada de una expresion en un punto dado teniendo en cuenta la magnitud de la variación en x que determina la precisión del cálculo (h)

Usage

```
hallarErrorCincoPuntos(h, expresion, punto)
```

Arguments

h	la magnitud de la variación en x
expresion	expresion a la que se le calculará la derivada
punto	punto en el que se evaluará la derivada

Value

error de truncamiento del cálculo de la derivada con el método de cinco puntos

Examples

```
expresion=expression((x^2)+x)
h=1*10^(-5)
punto=0
hallarErrorCincoPuntos(h,expresion,punto)
```

hallarErrorDosPuntos *Halla el error del método de dos puntos*

Description

Halla el error de truncamiento al calcular el valor de la derivada de una expresion en un punto dado teniendo en cuenta la magnitud de la variación en x que determina la precisión del cálculo (h)

Usage

```
hallarErrorDosPuntos(h, expresion, punto)
```

Arguments

h	la magnitud de la variación en x
expresion	expresion a la que se le calculará la derivada
punto	punto en el que se evaluará la derivada

Value

error de truncamiento del cálculo de la derivada con el método de dos puntos

Examples

```
expresion=expression((x^2)+x)
h=1*10^(-5)
punto=0
hallarErrorDosPuntos(h,expresion,punto)
```

---

hallarErrorTresPuntos *Halla el error del método de tres puntos*

Description

Halla el error de truncamiento al calcular el valor de la derivada de una expresion en un punto dado teniendo en cuenta la magnitud de la variación en x que determina la precisión del cálculo (h)

Usage

```
hallarErrorTresPuntos(h, expresion, punto)
```

Arguments

h	la magnitud de la variación en x
expresion	expresion a la que se le calculará la derivada
punto	punto en el que se evaluará la derivada

Value

error de truncamiento del cálculo de la derivada con el método de tres puntos

Examples

```
expresion=expression((x^2)+x)
h=1*10^(-5)
punto=0
hallarErrorTresPuntos(h,expresion,punto)
```

hallarHCincoPuntos *Halla la variación de x del método de cinco puntos*

Description

Halla la variación de x necesaria (h), para que se obtenga el error dado como parametro al calcular el valor de la derivada de una expresion en un punto dado.

Usage

```
hallarHCincoPuntos(error, expresion, punto)
```

Arguments

error	la magnitud del error que determinará el valor de h
expresion	expresion a la que se le calculará la derivada
punto	punto en el que se evaluará la derivada

Value

la variación de x (h) para el que la derivada tiene el error de truncamiento dado

Examples

```
expresion=expression((x^2)+x)
error=1*10^(-5)
punto=0
hallarHCincoPuntos(error,expresion,punto)
```

hallarHDosPuntos *Halla la variación de x del método de dos puntos*

Description

Halla la variación de x necesaria (h), para que se obtenga el error dado como parametro al calcular el valor de la derivada de una expresion en un punto dado.

Usage

```
hallarHDosPuntos(error, expresion, punto)
```

Arguments

error	la magnitud del error que determinará el valor de h
expresion	expresion a la que se le calculará la derivada
punto	punto en el que se evaluará la derivada

Value

la variación de x (h) para el que la derivada tiene el error de truncamiento dado

Examples

```
expresion=expression((x^2)+x)
error=1*10^(-5)
punto=0
hallarHDosPuntos(error,expresion,punto)
```

---

hallarHTresPuntos *Halla la variación de x del método de tres puntos*

Description

Halla la variación de x necesaria (h), para que se obtenga el error dado como parametro al calcular el valor de la derivada de una expresion en un punto dado.

Usage

```
hallarHTresPuntos(error, expresion, punto)
```

Arguments

error	la magnitud del error que determinará el valor de h
expresion	expresion a la que se le calculará la derivada
punto	punto en el que se evaluará la derivada

Value

la variación de x (h) para el que la derivada tiene el error de truncamiento dado

Examples

```
expresion=expression((x^2)+x)
error=1*10^(-5)
punto=0
hallarHTresPuntos(error,expresion,punto)
```

---

integralRichardson *integralRichardson(x,y)*

Description

Integral definica calculada en terminos de X y Y

Usage

```
integralRichardson(x, y, maxParticiones = TRUE, particiones = 3)
```

Arguments

x	Lista de valores x de una funcion
y	Lista de valores y de una funcion
maxParticiones	Calcular el número máximo de particiones?
particiones	Numero de particiones para calcular si no se quieren hacer las máximas

Value

integral definida entre x [1] and el valor x [length(x)]

Examples

```
integral(x, y, maxParticiones = TRUE, Particiones = 3)
```

InterpSpline *InterpSpline*

Description

calcula la interpolación en 3D utilizando el metodo de Spline Cubico

Usage

```
InterpSpline(x,y,n, xlim = NULL, ylim = NULL, zlim = NULL,
                 xlab = NULL, ylab = NULL, zlab = NULL, add = FALSE, aspect = !add,
                       forceClipregion = FALSE, ...)
```

Arguments

x,y	son vectores de los puntos por donde deve pasar la interpolación
n	La longitud de la interpolación deseada.
xlim,ylim,zlim	Límites X, Y y Z. Si está presente, la trama se recorta a esta región.
xlab,ylab,zlab	Títulos para los ejes. nótese bien Estas deben ser cadenas de caracteres; No se aceptan expresiones. Los números serán obligados a cadenas de caracteres.
add	si se desea añadir puntos a una trama existente
aspect	ya sea una indicación lógica de si se debe ajustar la relación de aspecto, o una nueva relación.
forceClippregion	forzar la utilización de una región de recorte, se den o no límites.
...	Parámetros de material adicionales para pasar a surface3d y decorate3d

Value

retorna una matriz de la función generada.

Author(s)

Javier Marin, Andres Mariño

Examples

```
x=rnorm(10)
y=rnorm(10)
z=InterpSpline(x,y,10,col="green",xlab="X")
print(z)
z[3,2]
{
  }
```

---

interpSplineC *interpSplineC(x, y=NULL, z, xo=seq(min(x), max(x), length = nx), yo=seq(min(y), max(y), length = ny), linear = TRUE, extrap=FALSE, duplicate = "error", dupfun = NULL, nx = 40, ny = 40, jitter = 10^-12, jitter.iter = 6, jitter.random = FALSE)*

Description

Estas funciones implementan la interpolación bivariada en una grilla. para datos de entrada espaciados irregularmente. Estriado bilineal o bicúbico La interpolación se aplica utilizando diferentes versiones de algoritmos de Akima. se ha modificado de la función original

Usage

```
interpSplineC(x, y=NULL, z, xo=seq(min(x), max(x), length = nx),
       yo=seq(min(y), max(y), length = ny),
       linear = TRUE, extrap=FALSE, duplicate = "error", dupfun = NULL,
       nx = 40, ny = 40,
       jitter = 10^-12, jitter.iter = 6, jitter.random = FALSE)
```

Arguments

x	Vector de las coordenadas en x de los puntos o SpatialPointsDataFrame objeto. Valores faltantes no son aceptados.
y	Vector de las coordenadas en y de los puntos. Valores faltantes no son aceptados. Si se deja como NULL indica que x deberia ser un SpatialPointsDataFrame y z nombra la variable de interés es el banco de datos.
z	Vector de las coordenadas en z de los puntos o un carácter variable que nombra la variable de interés en SpatialPointsDataFrame x. Valores faltantes no son aceptados. x, y, y z deben tener la misma longitud (excepto si x es un SpatialPointsDataFrame) y puede contener no menos de 4 puntos. Los puntos de x y y no deben ser colineales, por ejemplo, no deben caer en la misma linea (dos vectores x y y tales que y = ax + b para algún a, b no produciran resultados relevantes). Algunas heuristicas se construyen para evitar este caso añadiendo un pequeño de jitter de x y y cuando el numero de valores NA en el resultado excede el 10%. interp es para casos en los que se tenga n los valores x, y esparcidos sobre un plano y un valor z para cada uno. Si, en cambio, estas tratando de evaluar una funcion matemática, o tener una interpretación gráfica de relaciones descritas por un polinomio, intenta outer().

xo	Vector de coordenadas de x en una grilla de salida. El predeterminado es 40 puntos, espaciados regularmente sobre el rango de x. si la extrapolación no se está usando (extrap=FALSE, el predeterminado), xo debería tener un rango que es cercano o está dentro del rango de x para resultados significantes.
yo	Vector de coordenadas yen la grilla de salida; análogo a xo, ver arriba.
linear	logico – indicando si es lineal o spline interpolacion que debe ser usada.
extrap	bandera logica: La extrapolacion debe ser usada afuera de el casco convexo determinado por los puntos?
duplicate	caracter string indicando como manejar duplicados en los puntos. los valores posibles son: "error" Produce mensaje de error, "strip" Elimina los valores z duplicados, "mean","median","user" calcula mean , median or funcion definida por el usuario (dupfun) de los valores z duplicados.
dupfun	Una funcion aplicada a los puntos duplicados si, duplicate= "user".
nx	Dimension de la grilla de salida en direccion de x
ny	Dimension de la grilla de salida en direccion de x
jitter	Jitter de cantidad de diff(range(XX))*jitter (XX=x or y) sera añadido a las coordenadas si se detectan puntos colineares. Despues la interpolación se vuelve a intentar. Notese que jitter no se genera aleatoriamente a menos que jitter.random se ponga en TRUE. esto asegura un resultado reproducible. tri.mesh de paquete tripack esa el mismo mecanismo jitter. Eso significa que puedes gráficar la triangulación en la cima de la interpolación y ver la misma triangulación como la usada para interpolar, ver ejemplos abajo.
jitter.iter	número de reintentos de iteración con jitter, cantidad que será multiplicada en cada iteración por iter^1.5
jitter.random	lógico, ver jitter, predeterminado en FALSE

Author(s)

akima modificado por Javier marin, Andres Mariño, traducido, Leonardo Gómez

References

Akima, H. (1978). A Method of Bivariate Interpolation and Smooth Surface Fitting for Irregularly Distributed Data Points. ACM Transactions on Mathematical Software 4, 148-164.

Akima, H. (1996). Algorithm 761: scattered-data surface fitting that has the accuracy of a cubic polynomial. ACM Transactions on Mathematical Software 22, 362–371.

R. J. Renka (1996). Algorithm 751: TRIPACK: a constrained two-dimensional Delaunay triangulation package. ACM Transactions on Mathematical Software. 22, 1-8.

`lagrange.poly` *lagrange.poly*

Description

Calcula el tercer polinomio de lagrange

Usage

```
lagrange.poly(x, y)
```

Arguments

`x`	Valor en x para el polinomio
`y`	Valor en y para el polinomio

Value

Función con el tercer polinomio

`lagrange_evaluator` *lagrange_evaluator*

Description

Esta funcion devuelve el valor de las ecuaciones `x(t)` y `y(t)` para una curva parametrica para un paramero t.

Usage

```
lagrange_evaluator(t, formulas)
```

Arguments

`t`	Vector formulas Objeto.

Details

Esta funcion es parte del paquete de analisis numerico creado por los estudiantes de la Pontificia Universidad Javeriana para este curso en el semestre 1830.

Examples

```
t = c(0,0.25,0.5,0.75,1)
x = c(-1,0,1,0,1)
y = c(0,1,0.5,0,-1)
formulas = lagrange_parametric(t,x,y)
lagrange_evaluator(t, formulas)
```

lagrange_parametric *lagrange_parametric*

Description

Esta funcion devuelve las ecuaciones `x(t)` y `y(t)` para una curva parametrica.

Usage

```
lagrange_parametric(t, x, y)
```

Arguments

`x`	Vector y Vector t Vector. Los tres vectores deben tener el mismo numero de elementos.

Details

Esta funcion es parte del paquete de analisis numerico creado por los estudiantes de la Pontificia Universidad Javeriana para este curso en el semestre 1830.

Examples

```
t = c(0,0.25,0.5,0.75,1)
x = c(-1,0,1,0,1)
y = c(0,1,0.5,0,-1)
formulas = lagrange_parametric(t,x,y)
```

metodoEuler *Funcion metodoEuler*

Description

Funcion que resuelve ecuaciones diferenciales ordinarias de primer orden, por medio del metodo numerico de Euler. Esta funcion en particular UNICAMENTE puede utilizarse para problemas de valor inicial.

Usage

```
metodoEuler(edo, h, x0, y0, xf)
```

Arguments

`edo`	El nombre de la expresion a evaluar.
`h`	El tamanio del paso entre los valores de x. NO MENORES A 1e-6
`x0`	El valor inicial de x
`y0`	El valor inicial de y al ser evaluada en x0
`xf`	El valor final de x para el ejercicio

Value

dataframe con los valores calculados de x, acorde a h, y y

ODE_Sys *Función ODE_Sys*

Description

Esta es la principal función del paquete y la que debe ser llamada por el usuario, ya que a partir de esta función se llama a las demás funciones con el fin de cumplir el objetivo del paquete que es resolver el sistema de ecuaciones diferenciales.

Usage

```
ODE_Sys(d, vars, init, h, lims, n, method, point)
```

Arguments

d	Representa al vector de las ecuaciones diferenciales que componen el sistema a solucionar. Ejemplo -> ["Eqn1","Eqn2","Eqn3", ...]
vars	Representa al vector con los nombre de la variables incluyendo la variable independiente (t) al final del vector. Ejemplo -> ["x1","x2","x3", ...,"t"]
init	Vector con los valores iniciales incluyendo la variable indepediente (t) al final del vector. Ejemplo -> [x1_0,x2_0,x3_0, ...,t_0]
h	Constante.
lims	Vector con los límites desados de la solución. Ejemlo -> [x_liminf,x_limsup,y_liminf,y_limsup]
n	Número de puntos de la solución que se requieren.
method	Método para encontrar la solución. Se encoge entre las siguientes opciones -> ["euler","midpoint",rk4"]
point	Punto deseado para evaluar la solución.

Details

Si el usuario no ingresa la constante h, esta será definida por default como 0.1. Así mismo, si el usuario no ingresa el número de puntos para hallar la solución, este será definido por default como 250.

Value

La función retorna una lista de valores correspondientes a la solución numérica para cada valor de la variable independiente, así como la solución aproximada para un valor de la variable independiente. Así mismo también se encuentra como salida la gráfica de la solución, así como el campo de pendientes, únicamente si el sistema ingresado por el usuario es de segundo orden.

Author(s)

Pablo Millan y Oscar Fonseca

Examples

```
# Ejemplos:
sol = ODE_Sys(c("-x+y+1","x-y"),c("x","y","t"),c(-3,1,0),0.1,c(-4,4,-3,3),500,"rk4",0.62)

## The function is currently defined as
function (d, vars, init, h, lims, n, method, point)
{
    if (missing(h)) {
        h = 0.1
    }
    if (missing(n)) {
        n = 250
    }
    values = Solucion(d, vars, init, h, n, method, point)
    aux = point - values[length(d) + 1, ]
    index = which.min(abs(aux))
    solution_point = values[, index]
    solution <- list(trajectory = values, answer = solution_point)
    plotfield(field, lims, d, vars)
    return(solution)
  }
```

`plotfield` *plotfield función*

Description

Función que granfica el campo de pendientes, así como la solución del sistema de ecuaciones diferenciales.

Usage

```
plotfield(values, lims, d, vars)
```

Arguments

`values`	Valores que conforman la solución del sistema.
`lims`	Vector con los límites desados de la solución. Ejemlo -> [x_liminf,x_limsup,y_liminf,y_limsup]
`d`	Representa al vector de las ecuaciones diferenciales que componen el sistema a solucionar. Ejemplo -> ["Eqn1","Eqn2","Eqn3", ...]
`vars`	Representa al vector con los nombre de la variables incluyendo la variable independiente (t) al final del vector. Ejemplo -> ["x1","x2","x3", ...,"t"]

Details

Esta función únicamente se llevará a cabo solo si el sistema de ecuaciones diferenciales es de segundo orden.

Value

La función tiene como salida la grafica de la solución del sistema junto con el campo de pendientes. Similar a la herramienta enviada por la profesora.

Author(s)

Pablo Millan y Oscar Fonseca

Examples

```
# Ejemplos:
#plotfield(c(-4,4,-3,3),c("x*y","x*y-1"),c("x","y","t"))
#plotfield(c(-4,4,-3,3),c("x*y","x*y-1"),c("x","y","t"))
#plotfield(c(-4,4,-3,3),c("x*y","x*y-1"),c("x","y","t"))
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as

plotfield <- function(values, lims, d, vars){
  if (length(d)==2){

    Field <- function(t, y_in, parameters){
      dy = numeric(length(y_in))

      for (i in 1:length(dy)){
        for (j in 1:length(dy)){
          assign(vars[j],y_in[j])
        }
        dy[i]=eval(parse(text=parameters[i]))
      }
      return(list(dy))
    }
    Field.flowField <- flowField(Field, xlim=c(lims[1],lims[2]),
                                 ylim = c(lims[3],lims[4]), parameters = d,
                                 points = 20,
                                 add = FALSE, xlab = vars[1], ylab = vars[2] ,
                                 main = "Solution")

    grid()
   lines(values[1,],values[2,], col='red', type = 'l',xlim=c(lims[1],lims[2]),ylim=c(lims[3],lims[4]))
  }
}


{ ~kwd1 }
{ ~kwd2 }
```

`posicionFalsa` *Posicion falsa*

Description

Para una ecuacion no lineal que cumpla la condicion de f(x0)*f(x1)<0 recordando que este metodo es un derivado del metodo de secante.

Usage

```
posicionFalsa(funcion, x0, x1)
```

Arguments

funcion	Ecuacion no lineal definida anteriormente como: nombre_funcion <- function(x)(Ecuacion no lineal en terminos de x)
x0	Limite inferior
x1	Limite superior

Value

Una tabla con las aproximaciones de la raiz de la ecuacion no lineal, con los diferentes errores desde 10^-1 a 10^-6

---

puntofijo *Punto fijo*

Description

Para ecuaciones no lineales donde el punto fijo enviado por el usuario existe en el dominio de la derivada de la funcion, aproximando este valor hasta llegar a una raiz con un error hasta de 10^-6

Usage

```
puntofijo(funcion, x0)
```

Arguments

funcion	Ecuacion no lineal definida anteriormente como: nombre_funcion <- function(x)(Ecuacion no lineal en terminos de x)
x0	Punto enviado por el usuario el cual pertenece a la segunda derivada de la funcion donde posiblemente se encuentre la raiz

Value

Una tabla con las aproximaciones de la raiz de la ecuacion no lineal, con los diferentes errores desde 10^-1 a 10^-6

---

raizPuFijPosFSec *Raices con metodo de punto fijo, posicion falsa, secante*

Description

Desarrollado para ecuaciones no lineales, por medio de una funcion , y ya sea un punto o dos se generara raices para esa ecuacion, teniendo en cuenta las restricciones de cada funcion descrita anteriormente, debe escribir la ecuacion en forma de funcion y demas parametros y se calculara evaluando las restricciones de los metodos para esa ecuacion.

Usage

```
raizPuFijPosFSec(funcion, x0 = NULL, x1 = NULL)
```

Arguments

funcion	Ecuacion no lineal definida anteriormente como: nombre_funcion <- function(x)(Ecuacion no lineal en terminos de x)
x0	Limite menor del intervalo para Secante y PosicionFalsa, para PuntoFijo es el valor inicial del usuario
x1	Limite mayor del intervalo para Secante y PosicionFalsa

Value

Una tabla con las aproximaciones de la raiz de la ecuacion no lineal, con los diferentes errores desde 10^-1 a 10^-6

---

secante *Secante*

Description

Para ecuciones no lineales donde la funcion es doblemente diferenciable, es decir que su segunda derivada existe y es continua, ademas debe tener una raiz unica para generar la raiz.

Usage

```
secante(funcion, x0, x1)
```

Arguments

funcion	Ecuacion no lineal definida anteriormente como: nombre_funcion <- function(x)(Ecuacion no lineal en terminos de x)
x0	Limite inferior
x1	Limite superior

Value

Una tabla con las aproximaciones de la raiz de la ecuacion no lineal, con los diferentes errores desde 10^-1 a 10^-6

---

simpson *simpson*

Description

Calcula la integral de acuerdo a los lineamientos del metodo de Simpson (Parabolas)

Usage

```
simpson(fun, a, b, n)
```

Arguments

fun	Funcion a integrar por el metodo de Simpson
a	Limite inferior de la integral
b	Limite superior de la integral
n	Numero de particiones para graficar

Value

Retorna un valor de tipo entero que representa la suma de como resultado de la integral

---

SMRoot *SMRoot*

Description

Obtener las raices de la función dada

Usage

```
SMRoot(func = NULL, xo = NULL, E = 10^-6, maxiter = 100, method = "steffensen")
```

Arguments

func	Función matemática dada
xo	Valor inicial dado
E	Tolerancia
maxiter	Número máximo de iteraciones
method	Steffenson o Muller

Value

Lista de resultados con: Valor de la raíz, Error and número de iteraciones necesitado

Root	Valor de la raíz
Error	Error del método númerico
Iterations	número de iteraciones necesitado

Author(s)

Maria Fernanda Garces, Juan Sebastian Leon

solucion *Función Solución*

Description

Esta función resuelve el sistema de ecuaciones diferenciales, es llamada dentro de la función principal ODE_Sys con el fin de hallar los valores que conforman la solución.

Usage

```
solucion (d, vars, init, h, n, method, point)
```

Arguments

d	Representa al vector de las ecuaciones diferenciales que componen el sistema a solucionar. Ejemplo -> ["Eqn1","Eqn2","Eqn3", ...]
vars	Representa al vector con los nombre de la variables incluyendo la variable independiente (t) al final del vector. Ejemplo -> ["x1","x2","x3", ...,"t"]
init	Vector con los valores iniciales incluyendo la variable indepediente (t) al final del vector. Ejemplo -> [x1_0,x2_0,x3_0, ...,t_0]
h	Constante.
n	Número de puntos de la solución que se requieren.
method	Método para encontrar la solución. Se encoge entre las siguientes opciones -> ["euler","midpoint",rk4"]
point	Punto deseado para evaluar la solución.

Details

La solución claramente dependerá principalmente del método a implantar para resolver el sistema de ecuaciones diferenciales.

Value

La función retorna una lista de valores correspondientes a la solución numérica para cada valor de la variable independiente.

Author(s)

Pablo Millan y Oscar Fonseca

Examples

```
# Ejemplos:
#sol=solucion(c("x*y","x*y-1"),c("x","y","t"),c(0.5,0.5,0),0.1,500,"euler",0.62)
#sol=solucion(c("x*y","x*y-1"),c("x","y","t"),c(-2,1,0),0.01,400,"midpoint",0.5)
#sol=solucion(c("x*y","x*y-1"),c("x","y","t"),c(-2,2,0),0.01,700,"rk4",0.674)
#sol=solucion(c("x-y+z","x*y-1","z+3*x"),c("x","y","z","t"),c(0,1,1,0),0.01,450,"euler",0.5)
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.
```

```
## The function is currently defined as
Solucion <- function(d, vars, init, h, n, method, point){
  if (method=="euler"){
    values=matrix(init,ncol = 1)

    for (i in 1:length(d)){
      assign(vars[i],init[i])
    }

    npoints=1
    values=cbind(values,numeric(length(d)+1))

    while (npoints<n) {
      for (i in 1:length(d)){
        values[i,npoints+1]=values[i,npoints]+h*eval(parse(text=d[i]))
      }

      for (i in 1:length(d)){
        assign(vars[i],values[i,npoints+1])
      }

      values[length(d)+1,npoints+1]=values[length(d)+1,npoints]+h

      assign(vars[length(d)+1],values[length(d)+1,npoints+1])

      values=cbind(values,numeric(length(d)+1))
      npoints=npoints+1

    }

    values=values[,-npoints-1]
  }

  if (method=="midpoint"){
    values=matrix(init,ncol = 1)

    for (i in 1:length(d)){
      assign(vars[i],init[i]+h/2)
    }

    npoints=1
    values=cbind(values,numeric(length(d)+1))

    while (npoints<n) {
      for (i in 1:length(d)){
        values[i,npoints+1]=values[i,npoints]+h*eval(parse(text=d[i]))
      }

      for (i in 1:length(d)){
        assign(vars[i],values[i,npoints+1]+h/2)
      }

      values[length(d)+1,npoints+1]=values[length(d)+1,npoints]+h

      assign(vars[length(d)+1],values[length(d)+1,npoints+1])

      values=cbind(values,numeric(length(d)+1))
```

```
      npoints=npoints+1

    }

    values=values[,-npoints-1]
  }

  if (method=="rk4"){
    values=matrix(init,ncol = 1)

    for (i in 1:length(d)){
      assign(vars[i],init[i]+h/2)
    }

    npoints=1
    values=cbind(values,numeric(length(d)+1))

    while (npoints<n) {
      k1=numeric(length(d))
      k2=numeric(length(d))
      k3=numeric(length(d))
      k4=numeric(length(d))

      for (i in 1:length(d)){
        k1[i]=h*eval(parse(text=d[i]))
        assign(vars[i],values[i,npoints]+k1[i]/2)
      }
      for (i in 1:length(d)){
        k2[i]=h*eval(parse(text=d[i]))
        assign(vars[i],values[i,npoints]+k2[i]/2)
      }
      for (i in 1:length(d)){
        k3[i]=h*eval(parse(text=d[i]))
        assign(vars[i],values[i,npoints]+k3[i])
      }
      for (i in 1:length(d)){
        k4[i]=h*eval(parse(text=d[i]))
      }
      for (i in 1:length(d)){
        values[i,npoints+1]=values[i,npoints]+ k1[i]/6 + k2[i]/3 + k3[i]/3 + k4[i]/6
      }
      for (i in 1:length(d)){
        assign(vars[i],values[i,npoints+1])
      }
      values[length(d)+1,npoints+1]=values[length(d)+1,npoints]+h
      assign(vars[length(d)+1],values[length(d)+1,npoints+1])

      values=cbind(values,numeric(length(d)+1))
      npoints=npoints+1
    }
    values=values[,-npoints-1]
  }
  return (values)
}
```

```
{ ~kwd1 }
{ ~kwd2 }
```

`tablaCincoPuntos` *Tabla de derivadas por cinco puntos*

Description

Imprime una tabla de la derivada evaluada f'(a) a partir del error o de h y la formula de derivada por cinco puntos

Usage

```
tablaCincoPuntos(funcion, punto, dato, tipo, cantidadIteraciones)
```

Arguments

`funcion`	Expresion a la que se le calculará la derivada
`punto`	Punto en el que se evaluará la derivada
`dato`	Aqui se contiene h o el error dependiendo del parametro tipo
`tipo`	Este es un valor booleano que sirve para saber a partir de que calcular la derivada:(True es a partir del error)(False a partir de h)
`cantidadIteraciones`	Cuantas veces dividiremos entre dos el valor del dato. Con mas iteraciones, el dato que se encuentra en la ultima tupla es mas exacto, o mas cercano a la respuesta verdadera.
`h`	La magnitud de la variacion en x

Value

No retorna nada, imprime la matriz del error, el h y el valor de la funcion derivada en un punto

Examples

```
expresion=expression(x^5)
punto=2
dato=90
tipo=FALSE
iteraciones=20
tablaCincoPuntos(expresion,punto,dato,tipo,iteraciones)
punto=2
dato=90
tipo=TRUE
iteraciones=20
tablaCincoPuntos(expresion,punto,dato,tipo,iteraciones)
```

tablaDosPuntos *Tabla de derivadas por dos puntos*

Description

Imprime una tabla de la derivada evaluada f'(a) a partir del error o de h

Usage

```
tablaDosPuntos(funcion, punto, dato, tipo, cantidadIteraciones)
```

Arguments

funcion	Expresion a la que se le calculará la derivada
punto	Punto en el que se evaluará la derivada
dato	Aqui se contiene h o el error dependiendo del parametro tipo
tipo	Este es un valor booleano que sirve para saber a partir de que calcular la derivada:(True es a partir del error)(False a partir de h)
cantidadIteraciones	Cuantas veces dividiremos entre dos el valor del dato. Con mas iteraciones, el dato que se encuentra en la ultima tupla es mas exacto, o mas cercano a la respuesta verdadera.
h	La magnitud de la variación en x

Value

No retorna nada, imprime la matriz del error, el h y el valor de la funcion derivada en un punto

tablaTresPuntos *Tabla de derivadas por tres puntos*

Description

Imprime una tabla de la derivada evaluada f'(a) a partir del error o de h y la formula de derivada por tres puntos

Usage

```
tablaTresPuntos(funcion, punto, dato, tipo, cantidadIteraciones)
```

Arguments

funcion	Expresion a la que se le calculará la derivada
punto	Punto en el que se evaluará la derivada
dato	Aqui se contiene h o el error dependiendo del parametro tipo
tipo	Este es un valor booleano que sirve para saber a partir de que calcular la derivada:(True es a partir del error)(False a partir de h)
cantidadIteraciones	Cuantas veces dividiremos entre dos el valor del dato. Con mas iteraciones, el dato que se encuentra en la ultima tupla es mas exacto, o mas cercano a la respuesta verdadera.
h	La magnitud de la variacion en x

Value

No retorna nada, imprime la matriz del error, el h y el valor de la funcion derivada en un punto

Examples

```
expresion=expression(x^5)
punto=2
dato=90
tipo=FALSE
iteraciones=20
tablaTresPuntos(expresion,punto,dato,tipo,iteraciones)
expresion=expression(x^5)
punto=2
dato=90
tipo=TRUE
iteraciones=20
tablaTresPuntos(expresion,punto,dato,tipo,iteraciones)
```

---

trapezoid *trapezoid*

Description

Calcula la integral de acuerdo a los lineamientos del metodo del trapecio

Usage

```
trapezoid(func, a, b, n)
```

Arguments

func	Funcion a graficar por medio del metodo de Simpson
a	Limite inferior de la integral
b	Limite superior de la integral
n	Numero de particiones para graficar

Value

Retorna un valor de tipo entero que representa la suma de los trapecios como resultado de la integral

---

`tresPuntos` *Halla el valor de la derivada en un punto*

Description

Halla el valor de la derivada por el método de tres puntos de una expresion en un punto dado, con la restricción de un error o un h específico(según sea el caso).

Usage

```
tresPuntos(expresion, punto, dato, tipo)
```

Arguments

`expresion`	expresion a la que se le calculará la derivada
`punto`	punto en el que se evaluará la derivada
`dato`	puede representar el valor de la variación x(h) o el error con el que se quiere calcular el valor de la derivada.
`tipo`	especifíca si el anterior parametro(dato) corresponde a cualquiera de los dos datos que puede representar: TRUE=el dato representa el error, FALSE=el dato representa h.

Value

el valor de la derivada de la expresion en el punto dado.

Examples

```
expresion=expression((x^2)+x)
punto=0
dato=1*10^(-5)
tresPuntos(expresion,punto,dato,FALSE)
```

Index