Uso di Runge-Kutta al 4 ordine e un metodo multistep

Enrico Bertolazzi

Carica le librerie

```
> restart ;
  with(plots):
Warning, the name changecoords has been redefined
```

pDefinisce la procedura RK4 ovvero Runge-Kutta al 4 ordine

```
> RK4 := proc(f, x0::float, y0::float, h::float, n::integer)
    local i::integer,
          xi::float,
          yi::float,
          K1::float,
          K2::float,
          K3::float,
          K4::float,
          res::float ;
    xi := x0 ;
    yi := y0 ;
    # inizializza la lista che conterra` la soluzione
    res := [ [xi, yi] ] ;
    for i from 1 to n do
      K1 := h*f(xi,     yi) ;
      K2 := h*f(xi+h/2, yi+K1/2) ;
      K3 := h*f(xi+h/2, yi+K2/2) ;
      K4 := h*f(xi+h,   yi+K3) ;
      yi := evalf(yi + (1/6)*(K1+2*K2+2*K3+K4) ) ;
      xi := evalf(xi + h) ;
      # aggiunge il punto calcolato alla lista
      res := [ op(res), [xi, yi] ] ;
    end do;
    return res ;
  end proc ;
          RK4 := proc(f, x0::float, y0::float, h::float, n::integer)
          local i::integer, xi::float, yi::float, K1::float, K2::float, K3::float, K4::float, res::float;
                xi := x0;
                yi := y0;
                res := [[xi, yi]];
                for i to n do K1 := h*f(xi, yi);
                      K2 := h*f(xi + 1/2*h, yi + 1/2*K1);
                      K3 := h*f(xi + 1/2*h, yi + 1/2*K2);
```

```
            K4 := h*f(xi+ h, yi + K3);
            yi := evalf(yi + 1/6*K1 + 1/3*K2 + 1/3*K3 + 1/6*K4);
            xi := evalf(xi+ h);
            res := [op(res), [xi, yi]];
        end do;
        return res;
    end proc;
```

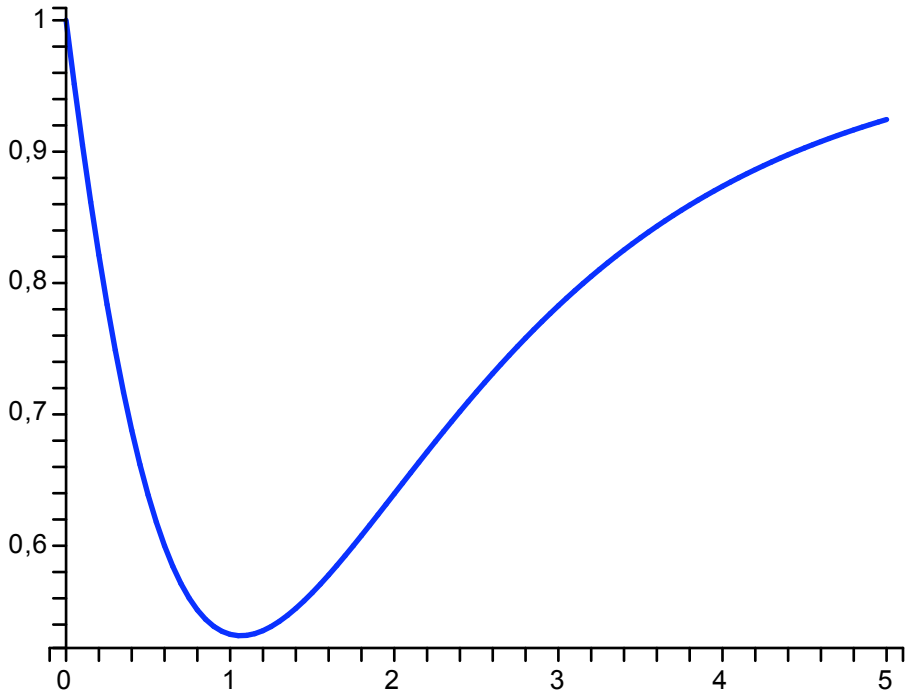
Esempio d'uso

```
> # Definisce la funzione da approssimare
  f := (x,y) -> x^2/(1+x^2)-y ;
```

$$f := (x, y) \rightarrow \frac{x^2}{1 + x^2} - y$$

```
> # risolve il problema
  pts := RK4(f,0.0,1.0,0.05,100) :

> # disegna la soluzione
  plot(pts,style=line,thickness=2,color=blue);
```



Procedura Multistep

```
> MSRK4 := proc(f, x0::float, y0::float, h::float, n::integer)
    local i::integer,
          j::integer,
```

```
        xi::float,
        yi::float,
        ff::Vector,
        w::Vector,
        res::float ;
  ff := Vector(4) ;
  # inizializza il vettore dei pesi per il metodo multistep
  w  := <-3/8,37/24,-59/24,55/24> ;
  # inizializza la lista che conterra` la soluzione
  res := RK4(f, x0, y0, h, 3) ;
  ff := evalf(<seq(f(res[i][1], res[i][2]),i=1..4)>) ;
  xi := res[4][1] ;
  yi := res[4][2] ;
  for i from 5 to n do
    yi := evalf(yi + h*add(w[j]*ff[j],j=1..4)) ;
    xi := evalf(xi + h) ;
    # aggiunge il punto calcolato alla lista
    res := [ op(res), [xi, yi] ] ;
    # aggiorna ff
    ff[1] := ff[2] ;
    ff[2] := ff[3] ;
    ff[3] := ff[4] ;
    ff[4] := evalf(f(xi,yi)) ;
  end do;
  return res ;
end proc ;
```

```
MSRK4 := proc(f, x0::float, y0::float, h::float, n::integer)
local i::integer, j::integer, xi::float, yi::float, ff::Vector, w::Vector, res::float;
    ff := Vector(4);
    w := <(-3)/8, 37/24, (-59)/24, 55/24>;
    res := RK4(f, x0, y0, h, 3);
    ff := evalf(<seq(f(res[i][1], res[i][2]), i = 1 .. 4)>);
    xi := res[4][1];
    yi := res[4][2];
    for i from 5 to n do yi := evalf(yi + h*add(w[j]*ff[j], j = 1 .. 4));
        xi := evalf(xi + h);
        res := [op(res), [xi, yi]];
        ff[1] := ff[2];
        ff[2] := ff[3];
        ff[3] := ff[4];
        ff[4] := evalf(f(xi, yi));
    end do;
    return res;
end proc;
```

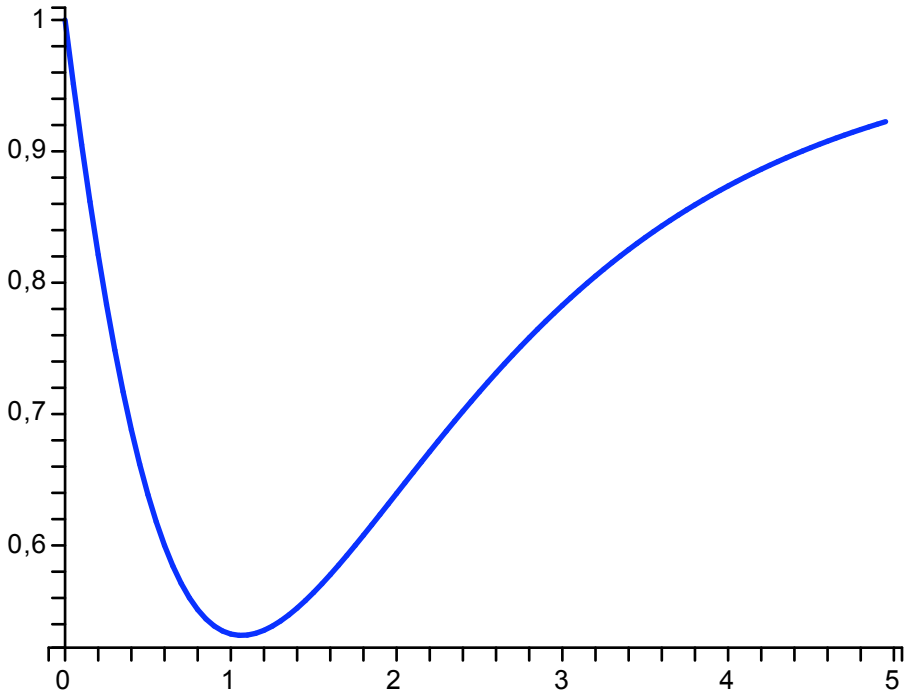
Esempio d'uso

```
> # Definisce la funzione da approssimare
  f := (x,y) -> x^2/(1+x^2)-y ;
```

$$f := (x, y) \rightarrow \frac{x^2}{1 + x^2} - y$$

```
> # risolve il problema
  pts := MSRK4(f,0.0,1.0,0.05,100) :

> # disegna la soluzione
  plot(pts,style=line,thickness=2,color=blue);
```




```
>
```