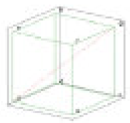


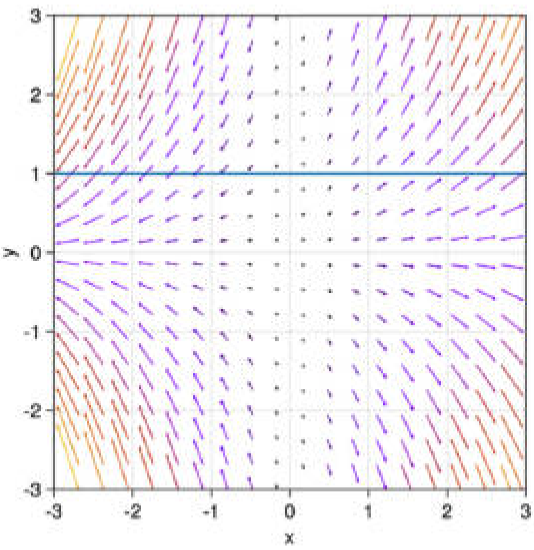
ch11 sec11.3 欧拉方法

Table of Contents

ch11 sec11.3 欧拉方法

现在有关微分方程的解法有很多, 在实际应用中不会使用欧拉方法, 但是理解欧拉方法是理解其他微分方程方法的基础. 我们先来看看大致的步骤(也是设计算法的步骤), 然后再看看欧拉方法的不足之处.

上一节我们经过绘制下面这样的斜率场



假设我们身处于一条曲线上, 要在坐标系里遨游. 每个点的斜率为我们指出了下一步该往哪个方向走.

假设出发点的坐标为:

$$\{x_0, y_0\}$$

当我们在一个点沿着斜率指明的方向前进一小步Δx, 那么在x方向移动了

$$x_0 + \Delta x$$

根据该点的斜率,可以计算出在y 方向的位移, 假设斜率公式(导数) 为$dy/dx = y$

则在x_0处的斜率为:

$$dy/dx = y_0$$

由此我们可以计算出y方向值:

$$y_0(\Delta x) + y_0$$

于是当我们在在x方向移动Δx 后, 到达的坐标为:

$$\{x_1 = x_0 + \Delta x, y_1 = y_0(\Delta x) + y_0\}$$

y_1 既是函数在x_1 的取值.

我们在看看第二步的情况, 仍然只在x 方向移动 Δx,由此的x 方向的新坐标为:

$$x_1 + \Delta x$$

则在x_1处的斜率为:

$$dy/dx = y_1$$

因此在y方向的变化为:

$$y = y_1(\Delta x)$$

于是当我们在在x方向移动第二次移动Δx 后, 到达的坐标为:

$$\{x_2 = x_1 + \Delta x, y_2 = y_1(\Delta x) + y_1\}$$

于是可以得到通项表达式: 当我们沿着x方向移动$n \cdot \Delta x$ 时 坐标为:

$$\{x_n = x_{n-1} + \Delta x, y_n = y_{n-1}(\Delta x) + y_{n-1}\}$$

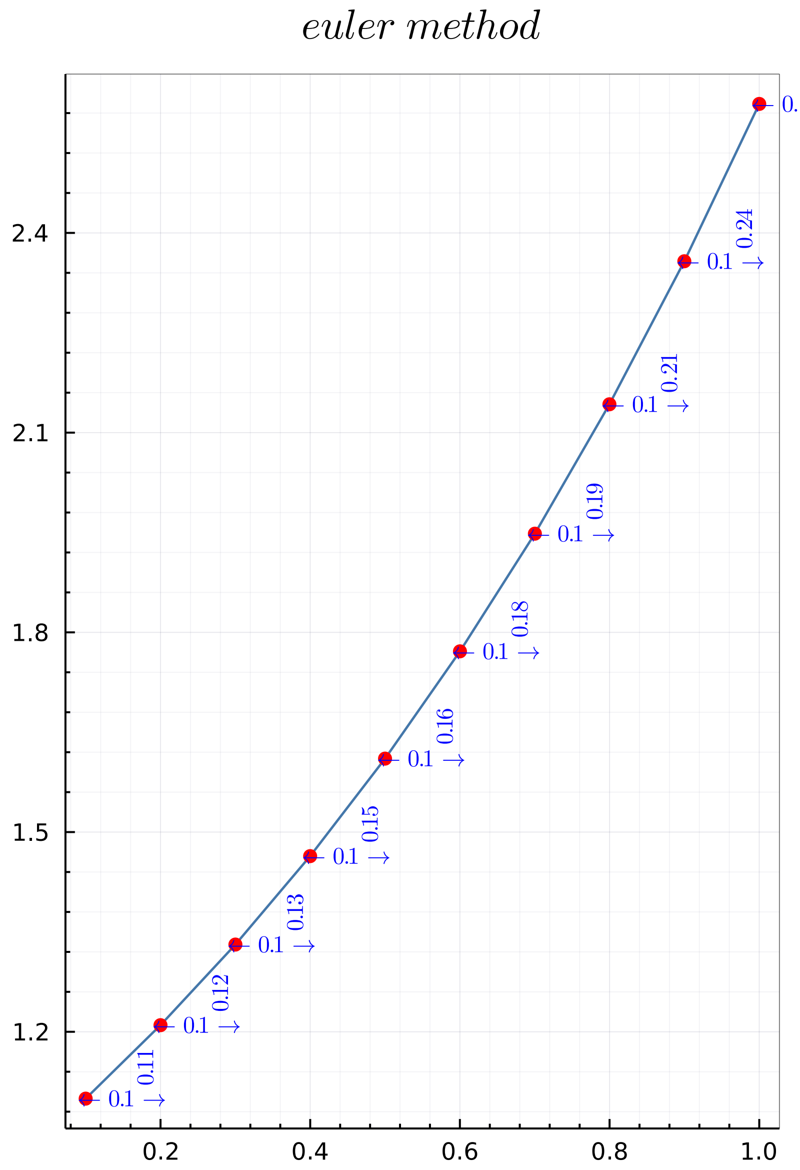
行走十步的结果

表格

10 rows × 4 columns

	title Any	x Any	y Any	Δy Any
1	P1	0.1	1.1	0.11=(1.1)(0.1)
2	P2	0.2	1.21	0.121=(1.21)(0.1)
3	P3	0.3	1.331	0.1331=(1.331)(0.1)
4	P4	0.4	1.464	0.1464=(1.464)(0.1)
5	P5	0.5	1.61	0.161=(1.61)(0.1)
6	P6	0.6	1.771	0.1771=(1.771)(0.1)
7	P7	0.7	1.948	0.1948=(1.948)(0.1)
8	P8	0.8	2.143	0.2143=(2.143)(0.1)
9	P9	0.9	2.357	0.2357=(2.357)(0.1)
10	P10	1.0	2.594	0.2594=(2.594)(0.1)

图像



如果你仔细观察通过欧拉方法获取的解曲线, 会发现曲线并不是那么平滑, 因为我们是在Δx的范围内用直线来近似曲线, 我们的解曲线实际是由一些小的直线拼接而成.

如果要让结果更为精确, 我们把Δx 的值再取小一点, 比如$0.1 \to 0.05$

来看看当$\Delta x = 0.05$ 的值和图像

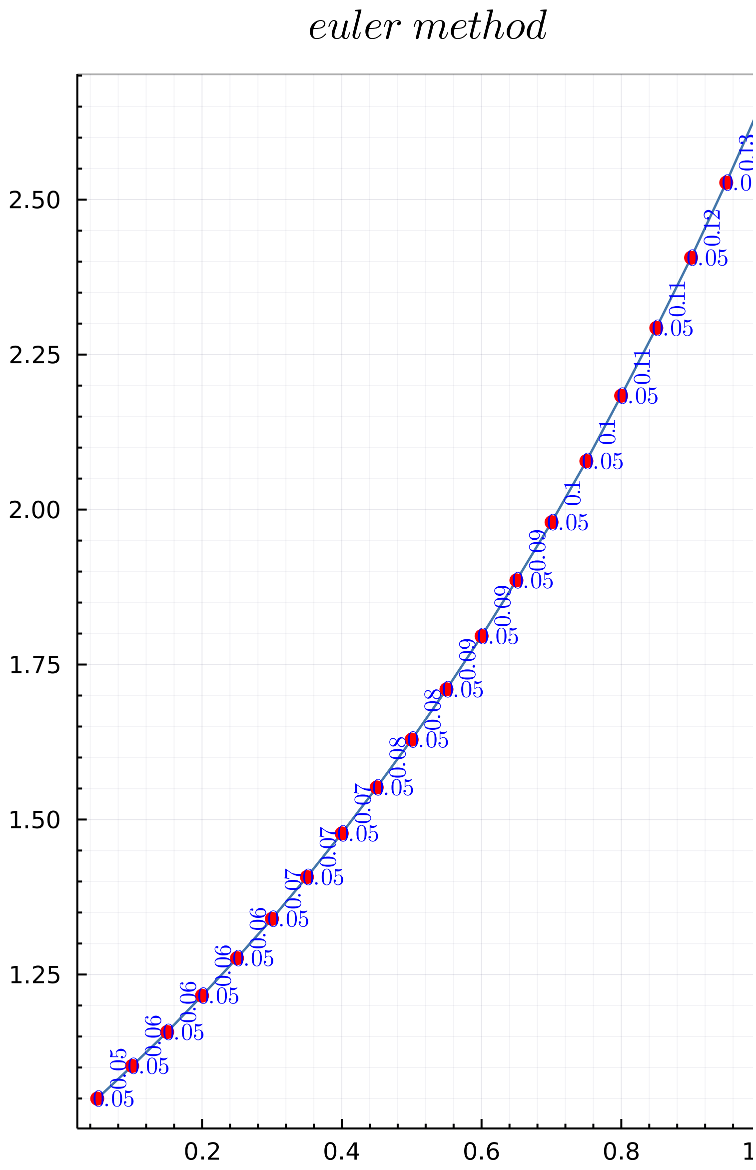
表格

20 rows × 4 columns

	title	x	y	Δy
	Any	Any	Any	Any
1	P1	0.05	1.05	0.0525=(1.05)(0.05)
2	P2	0.1	1.103	0.0551=(1.103)(0.05)
3	P3	0.15	1.157	0.0579=(1.157)(0.05)

图像

	title Any	x Any	y Any	Δy Any
4	P4	0.2	1.216	0.0608=(1.216)(0.05)
5	P5	0.25	1.276	0.0638=(1.276)(0.05)
6	P6	0.3	1.34	0.067=(1.34)(0.05)
7	P7	0.35	1.407	0.0704=(1.407)(0.05)
8	P8	0.4	1.478	0.0739=(1.478)(0.05)
9	P9	0.45	1.552	0.0776=(1.552)(0.05)
10	P10	0.5	1.629	0.0814=(1.629)(0.05)
11	P11	0.55	1.71	0.0855=(1.71)(0.05)
12	P12	0.6	1.796	0.0898=(1.796)(0.05)
13	P13	0.65	1.886	0.0943=(1.886)(0.05)
14	P14	0.7	1.9795	0.099=(1.9795)(0.05)
15	P15	0.75	2.078	0.1039=(2.078)(0.05)
16	P16	0.8	2.184	0.1092=(2.184)(0.05)
17	P17	0.85	2.293	0.1146=(2.293)(0.05)
18	P18	0.9	2.406	0.1203=(2.406)(0.05)
⋮	⋮	⋮	⋮	⋮




```
md"""
现在有关微分方程的解法有很多, 在实际应用中不会使用欧拉方法, 但是理解欧拉方法是理解其他微分方程方法的基础.  我们先来看看大致的步骤(也是设计算法的步骤), 然后再看看欧拉方法的不足之处.

上一节我们经过绘制下面这样的斜率场

![](https://tva1.sinaimg.cn/orj360/e6c9d24egy1h3b5dko2scj20pm0pejuw.jpg)

```

- 假设我们身处于一条曲线上，要在坐标系里遨游. 每个点的斜率为我们指出了下一步该往哪个方向走.
- 假设出发点的坐标为:
- $\{x_0,y_0 \}$
- 当我们在一个点沿着斜率指明的方向前进一小步Δx, 那么在x方向移动了
- $x_0 + \Delta x$
- 根据该点的斜率,可以计算出在y 方向的位移, 假设斜率公式(导数) 为$dy/dx=y$
- 则在x_0处的斜率为:
- $dy/dx=y_0$
- 由此我们可以计算出y方向值:
- $y_0(\Delta x)+y_0$
- 于是当我们在在x方向移动Δx 后, 到达的坐标为:
- $\{x_1=x_0 + \Delta x , y_1=y_0(\Delta x)+y_0\}$
- y_1 既是函数在x_1 的取值.
- 我们在看看第二步的情况, 仍然只在x 方向移动 Δx,由此的x 方向的新坐标为:
- $x_1 + \Delta x$
- 则在x_1处的斜率为:
- $dy/dx=y_1$
- 因此在y方向的变化为:
- $y=y_1(\Delta x)$
- 于是当我们在在x方向移动第二次移动Δx 后, 到达的坐标为:
- $\{x_2=x_1 + \Delta x , y_2=y_1(\Delta x)+y_1\}$
- 于是可以得到通项表达式: 当我们沿着x方向移动$n \cdot \Delta x$ 时 坐标为:
- $\{x_n=x_{n-1}+\Delta x , y_n=y_{n-1}(\Delta x)+y_{n-1} \}$
- 行走十步的结果
- 表格 | 图像 |
- :-------- | :---------------: |
- $(store["p10"]) | $(store["plot1"]) |

```
如果你仔细观察通过欧拉方法获取的解曲线，会发现曲线并不是那么平滑，因为我们是在$\Delta x$的范围内用直线来近似曲线，我们的解曲线实际是由一些小的直线拼接而成.

如果要让结果更为精确，我们把$\Delta x$ 的值再取小一点，比如$0.1 \to  0.05$


来看看当$\Delta x=0.05$ 的值和图像


表格     | 图像 |
:------- | :---------------: |
$(store["p102"])    | $(store["plot2"])  |

"""
```

"store success!"

```
let

    Δx=0.1
    titlearr,xarr,yarr,Δyarr=[],[],[],[]
    df(p0::point)= p0.y        #导函数, 输入一个当前坐标, 返回该处的变化率
    p0=point(0,1)  #从 P0 开始
    getval=eulerfunc(p0,df)
    res=[getval(n) for n in 1:10]

    for (i,n) in enumerate(res)
         push!(titlearr,"P$(i)")
         push!(xarr,n.x)
         push!(yarr,n.y)
         push!(Δyarr,"$(round(n.y*Δx,digits=4))=($(n.y))($(Δx))")
    end

    df=DataFrame(;title=titlearr,x=xarr,y=yarr,Δy=Δyarr)

    save("p10",df)
    save("res1",res)

end
```

```
"store success!" = "store success!"
"store success!" = "store success!"
```

"store success!"

```
let

    Δx=0.05
    titlearr,xarr,yarr,Δyarr=[],[],[],[]
    df(p0::point)= p0.y        #导函数, 输入一个当前坐标, 返回该处的变化率
    p0=point(0,1)  #从 P0 开始
    getval=eulerfunc(p0,df,0.05)
    res=[getval(n) for n in 1:20]

    for (i,n) in enumerate(res)
         push!(titlearr,"P$(i)")
         push!(xarr,n.x)
         push!(yarr,n.y)
         push!(Δyarr,"$(round(n.y*Δx,digits=4))=($(n.y))($(Δx))")
    end

    df=DataFrame(;title=titlearr,x=xarr,y=yarr,Δy=Δyarr)

    save("p102",df)
    save("res2",res)

end
```

```
"store success!" = "store success!"
"store success!" = "store success!"
```

```
"store success!"
```

```
let
    Δx=0.1
    offset1=(1/2)*Δx
    offset2=Δx-0.02
    data,plotdata=store["res1"],store["p10"]
    function getann1(p)
         return (
          p.x+offset1,p.y,text(L"\leftarrow %$(Δx)
        \rightarrow",pointsize=8,color=:blue)
         )

    end

    function getann2(idx,p)

              str=round(p.y*Δx,digits=2)
         return (
          p.x+offset2,p.y+0.05,text(L"%$(str)",pointsize=8,color=:blue,rotation=90)
         )

    end

    annarr1=[getann1(p) for p in data]
    annarr2=[getann2(idx,p) for (idx, p) in enumerate(data)]

    p1=plot(plotdata.x,plotdata.y,frame=:semi,label=false,ann=
    [annarr1...,annarr2...],size=(400,600),title=L"euler \ method")
    p2=scatter!(plotdata.x,plotdata.y,ms=4,color=:red,label=false)

    save("plot1",p2)

end
```

```
"store success!" = "store success!"
```

```
"store success!"
```

```
let
    Δx=0.05
    offset1=(1/2)*Δx
    offset2=Δx-0.02
    data,plotdata=store["res2"],store["p102"]
    function getann1(p)
         return (
          p.x+offset1,p.y,text(L" %$(Δx)",pointsize=8,color=:blue)
         )

    end

    function getann2(idx,p)

            str=round(p.y*Δx,digits=2)
         return (
          p.x+offset2,p.y+0.05,text(L"%$(str)",pointsize=8,color=:blue,rotation=90)
         )

    end

    annarr1=[getann1(p) for p in data]
    annarr2=[getann2(idx,p) for (idx, p) in enumerate(data)]

    p1=plot(plotdata.x,plotdata.y,frame=:semi,label=false,ann=
    [annarr1...,annarr2...],size=(400,600),title=L"euler \ method")
    p2=scatter!(plotdata.x,plotdata.y,ms=4,color=:red,label=false)

    save("plot2",p2)

end
```

```
"store success!" = "store success!"
```

Example

example 3

根据微分方程 $dy/dx = -x/y$ 使用$\Delta x = 0.1$ 来近似点$[0,1]$ 附近的函数取值

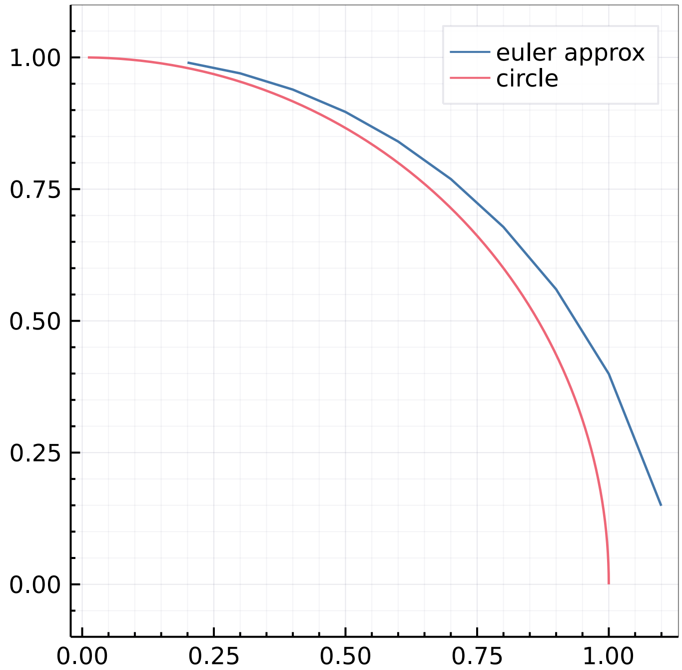
之前看到这个微分方程的原函数是圆的方程, 我们用已知条件近似后,与原曲线比较一下, 看看近似的效果

表格

10 rows × 4 columns

	title	x	y	Δy
	Any	Any	Any	Any
1	P1	0.2	0.99	0.099=(0.99)(0.1)
2	P2	0.3	0.9697	0.097=(0.9697)(0.1)
3	P3	0.4	0.939	0.0939=(0.939)(0.1)
4	P4	0.5	0.8965	0.0896=(0.8965)(0.1)
5	P5	0.6	0.8403	0.084=(0.8403)(0.1)
6	P6	0.7	0.769	0.0769=(0.769)(0.1)
7	P7	0.8	0.678	0.0678=(0.678)(0.1)
8	P8	0.9	0.56	0.056=(0.56)(0.1)
9	P9	1.0	0.3994	0.0399=(0.3994)(0.1)
10	P10	1.1	0.149	0.0149=(0.149)(0.1)

图像



欧拉方法近似的值和原函数的取值不太吻合.

```
md"""

!!! example
    example 3

    根据微分方程 $dy/dx=-x/y$ 使用$\Delta x =0.1$ 来近似点$[0,1]$ 附近的函数取值

之前看到这个微分方程的原函数是圆的方程, 我们用已知条件近似后,与原曲线比较一下, 看看近似的效果

```

```
表格      | 图像 |
:------- | :---------------: |
$(store["pcircle"])     | $(store["plot3"])  |


欧拉方法近似的值和原函数的取值不太吻合.

"""
```

"store success!"

```
let

    Δx=0.1
    titlearr,xarr,yarr,Δyarr=[],[],[],[]
    df(p0::point)=-(p0.x/p0.y)      #导函数, 输入一个当前坐标, 返回该处的变化率
    p0=point(0.1,1)  #从 P0 开始
    getval=eulerfunc(p0,df)
    res=[getval(n) for n in 1:10]

    for (i,n) in enumerate(res)
         push!(titlearr,"P$(i)")
         push!(xarr,n.x)
         push!(yarr,n.y)
         push!(Δyarr,"$(round(n.y*Δx,digits=4))=($(n.y))($(Δx))")
    end

    df=DataFrame(;title=titlearr,x=xarr,y=yarr,Δy=Δyarr)

    save("pcircle",df)
    save("res3",res)

end
```

```
"store success!" = "store success!"
"store success!" = "store success!"
```

"store success!"

```
let
  tspan=0:0.02:0.5*pi
  xs,ys=[cos(t) for t in tspan],[sin(t) for t in tspan]

  plotdata=store["pcircle"]

  plot(plotdata.x,plotdata.y,ratio=:equal,frame=:semi,size=(360,360),label="euler approx")
  p3= plot!(xs,ys, label="circle")
  save("plot3",p3)

end
```

```
"store success!" = "store success!"
```

```
eulerfunc (generic function with 2 methods)
```

```julia
begin
    store=Dict()

    function save(key::String, dict)
        if haskey(store,key)==true
           @show  "already has key:$(key), try another key"

        else
          store[key]=dict
          if haskey(store,key)==true
              @show  "store success!"
          else
              @show  "has error, try later"
          end
        end
    end

"""
 read(key::String)

        if  haskey(store, key)==true
            return  store[key]
        else
            @show "there is no key in store!"

        end
    end

"""

    function read(key::String)

        if  haskey(store, key)==true
            return  store[key]
        else
            @show "there is no key in store!"

        end
    end


"""
  eulerfunc  欧拉方法

   struct  point
        x::Float16
        y::Float16
    end

"""
  struct  point
        x::Float16
        y::Float16
  end

  function eulerfunc(p0::point,df,Δx=0.1)  #定义欧拉方法
         return  function   (nstep)
```

```
                slope=df(p0)
                x,y=p0.x,p0.y
                for n in 1:nstep

                    x=x+Δx
                    y=y+slope*Δx
                    slope=df(point(x,y))

                end

               return point(x,y)

          end

     end



end
```

```
@htl("""<script src="https://cdn.bootcdn.net/ajax/libs/mathjax/3.2.0/es5/tex-svg-full.min.js"></script>
""")
```