Julia como CAS

Usando Reduce e Plots.jl

Osmar Cardoso Lopes Filho

GitHub

Sumário

---

1 Introdução

1.1 REDUCE

Como descrito no site oficial, REDUCE é um sistema algébrico computacional (CAS em inglês) portátil e de uso geral. Seu desenvolvimento teve início na década de 1960, com sua primeira versão sendo lançada ao público em 1968 (10 anos antes da primeira versão do antigo Macsyma).

Ainda que por grande parte de sua existência tenha sido um programa comercial e pago, em 2008 o código fonte for aberto e o REDUCE passou a ser grátis e livre.

1.2 Reduce.jl

O pacote Reduce.jl é uma interface entre a linguagem Julia e o Reduce. Seu funcionamento pode ser dividido em duas partes: Primeiramente, variáveis do tipo `Symbol`, `Expr` são traduzidas para expressões algébricas e envidas para o Reduce. Em seguida, o resultado é analizado sintaticamente para reconstruir ASTs (*Abstract Syntax Trees*) em Julia.

Reduce.jl é capaz de estender localmente as funções nativas de Julia para os tipos `Symbol` e `Expr`, permitindo escrevermos expressões simbólicas sem nos preocupar com novas sintaxes.

2 Instalação

2.1 Julia

Para instalar a linguagem Julia, acesse o site oficial e siga as instruções na aba “Download”. A partir disso, você terá em seu sistema dois executaveis em CLI (*command line interface*): `julia` e `juliaup`. Seus usos são:

- `julia`: Inicializar o REPL (Read-Eval-Print-Loop) interativo da linguagem ou compilar código em Julia caso um arquivo seja passado junto com o comando;
- `juliaup`: Atualizar e gerenciar versões da linguagem Julia no sistema,

2.2 Kernel IJulia

Para utilizar Julia através do Jupyter Notebook ou Lab, instale primeiro a linguagem e depois siga as instruções de instalação do pacote IJulia.jl. Feito isso, o Jupyter deve ser capaz de identificar o *kernel* de Julia instalado pelo pacote em seu sistema.

2.3 Reduce.jl

Para instalar o Reduce.jl, siga as instruções na seção "Setup" da documentação. Esse processo também instalará um executavel do Reduce para uso com o pacote. Portanto, não precisamos nos preocupar em instalar o Reduce manualmente.

3 Curvas em Duas Dimensões

3.1 Parametrização

Em duas dimensões, curvas são parametrizadas por aplicações da seguinte forma:

$$\alpha : I \longrightarrow \mathbb{R}^2$$

Podendo serem separadas em componentes para cada coordenada:

$$\alpha_i : I \longrightarrow \mathbb{R}, i = 1, 2$$
$$\alpha(t) = (\alpha_1(t), \alpha_2(t)), t \in I$$

Caso α seja diferenciável, o seguinte vetor, chamado de vetor tangente, é bem definido:

$$\alpha'(t) = (\alpha_1'(t), \alpha_2'(t)), t \in I$$

Para renderizar essas curvas, usaremos o pacote Plots.jl. Contudo, precisamos antes coletar alguns pontos ao longo de seus traços para então desenhá-las. Fazemos isso através da seguinte função:

```
function curvepoints(expressions, exprvar, trange)
    rangelen = length(trange)
    domaindim = length(expressions)
    points = zeros(Float64, (rangelen, domaindim))

    for i in 1:rangelen
        eval(:($exprvar = $(trange[i])))
        points[i, :] = eval.(expressions)
    end

    return points
end
```

Nela, uma array de expressões, `expressions`, a variável parâmetro, `exprvar` e os valores dessa variável, `trange` são passados. Então, a função avalia o parâmetro em cada valor e depois a expressão com esse parâmetro. O resultado obtido é guardado na array `points`.

Feito isso, desenharemos a Figura 1 como exemplo:

$$\alpha(t) = (\cos(t)\log(t+1), \sin(t)\log(t+1))$$

```
using Plots

curve = [:(cos(t) * log(t+1)), :(sin(t) * log(t+1))]
```

```
cpnts = curvepoints(curve, :t, range(0, 30*π, length=1500))

plot(
  (cpnts[:,1], cpnts[:,2]);

  linewidth=2,
  size=(600,600),
  label="",
  aspect_ratio=1
)
```

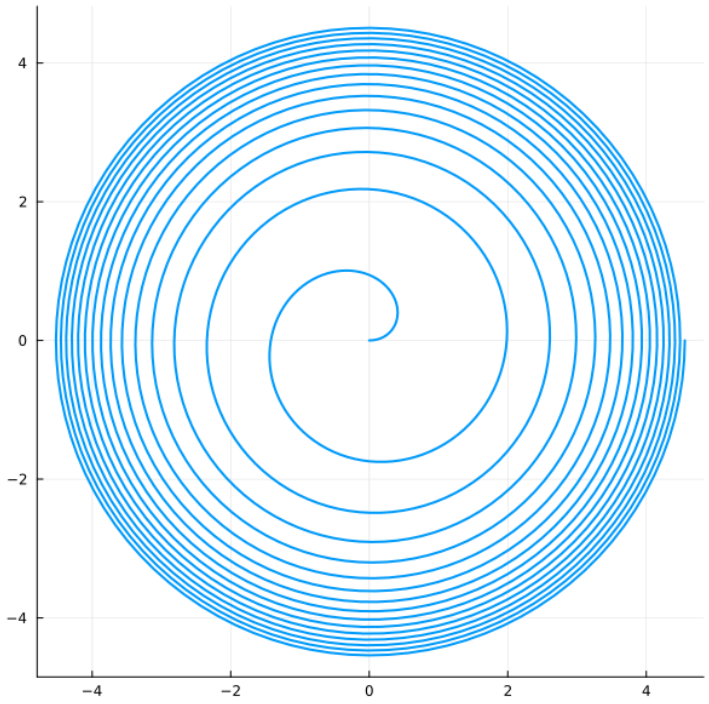

Figura 1: Curva espiral.

Criaremos também uma função para calcular os vetores velocidade em certos pontos na curva:

```
df(:x, :x)
function curvevectors(expressions, exprvar, tvalues)
    valueslen = length(tvalues)
    domaindim = length(expressions)
    vectors = zeros(Float64, (valueslen*3, domaindim))

    derivatives = df.(expressions, exprvar)

    for i in 1:valueslen
        eval(:($exprvar = $(tvalues[i])))

        e_expr = eval.(expressions)
        e_derv = eval.(derivatives)

        vectors[(i-1)*3 + 1, :] = e_expr
        vectors[(i-1)*3 + 2, :] = e_expr + e_derv
        vectors[(i-1)*3 + 3, :] = fill(NaN, (1, domaindim))
```

```
        end

        return vectors
    end
```

Com isso, desenhamos a Figura 2:

```
plot!(
    (cvecs[:,1], cvecs[:,2]);

    linewidth=2,
    arrow=true,
    label=""
)
```

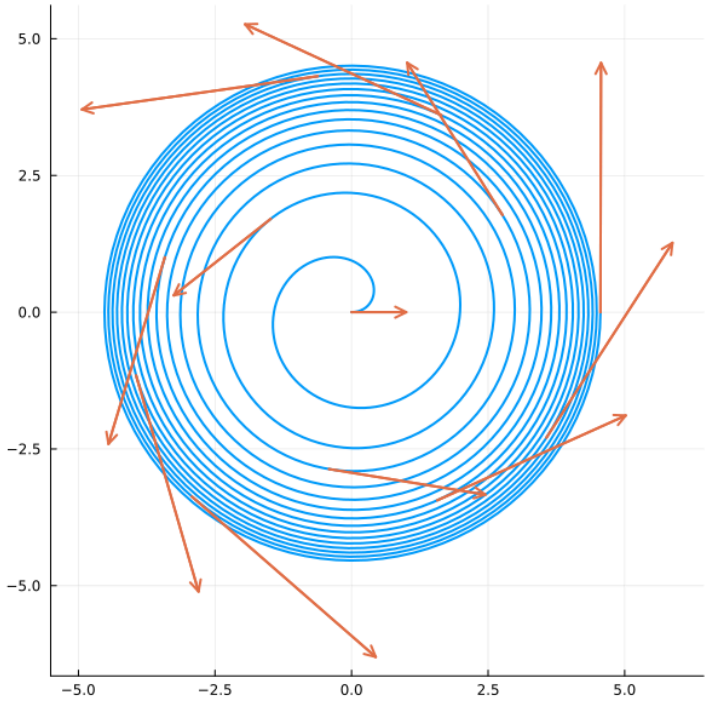

Figura 2: Curva espiral com vetores velocidade.

3.2 Regularidade

Define-se como curva regular qualquer curva $\alpha : I \longrightarrow \mathbb{R}^n$ que satifaz o seguinte:

$$\forall t \in I,\, \alpha'(t) \neq 0$$

3.3 Reparametrização

Uma reparametrização de uma curva α é uma aplicação $\beta : I_0 \longrightarrow \mathbb{R}^2$ tal que $\forall t \in I_0,\, \beta(t) = \alpha(\varphi(t))$, onde $\varphi : I_0 \longrightarrow I$ é um difeomorfismo.

Dado dois conjuntos abertos, $U \in \mathbb{R}^n$ e $V \in \mathbb{R}^m$, definimos como difeomorfismo qualquer bijeção $f : U \longrightarrow V$ tal que ambas f e f^{-1} sejam diferenciáveis.

Definimos uma reparametrização $\beta = \alpha \circ \varphi$ como positiva caso $\varphi' > 0$. Caso $\varphi' < 0$, definimos β como uma reparametrização negativa.

3.4 Comprimento de Arco

Definimos o comprimento de arco de uma curva regular α da seguinte forma:

$$L(\alpha) = \int_a^b \|\alpha'(t)\|\, dt = \int_a^b \sqrt{\langle \alpha', \alpha' \rangle} dt$$

Como essa função é um difeomorfismo, podemos usá-la para reparametrizar curvas. Seja $\varphi(s) = L^{-1}(t)$ sua invérsa, definimos $\beta(s) = \alpha \circ \varphi(s)$ como a reparametrização por comprimento de arco, ou unit-speed, de α.

Para calcular esse comprimento, escrevemos primeiro sua expressão simbolicamente e tentamos simplificá-la computacionalmente. Caso isso seja possível, Reduce retornará uma expressão fechada. Caso contrário, o comando inicial ou expressões com outras integrais podem ser retornadas. A seguir, escrevemos a expressão para o comprimento da curva já apresentada:

```
curved1t = df.(curve, :t)
curvelen = int(sqrt(curved1t[1]^2 + curved1t[2]^2), :t)
```

O resultado obtido é:

```
:(int((sqrt((t ^ 2 + 2t + 1) * log(t + 1) ^ 2 + 1) * sqrt(cos(t) ^ 2 + sin(t) ^
2)) / abs(t + 1), t))
```

$$\int \left(\frac{\sqrt{(t^2 + 2t + 1)\log(t+1)^2 + 1}\ \sqrt{\cos(t)^2 + \sin(t)^2}}{|t+1|} \right) dt$$

Ou seja, ainda temos uma integral, mas com uma expressão expandida. Em seguida, utilizamos o pacote `TRIGSIMP` do Reduce para simplificar expressões trigonométricas:

```
curvelen = :(trigsimp($curvelen)) |> rcall
```

$$\int \left(\frac{\sqrt{(t^2 + 2t + 1)\, \log(t+1)^2 + 1}}{|t+1|} \right)$$

A partir desse ponto, utilizamos ferramentas computacionais para resolver o problema.

3.5 Curvatura

Primeiramente, dado o círculo unitário S^1 e a curva diferenciável $\gamma : I \longrightarrow S^1$, definimos a função ângulo como $\theta(s)$ tal que $\gamma(s) = (\cos(\theta(s)), \sin(\theta(s)))$.

Feito isso, tomamos uma curva $\alpha(t)$ com $\|\alpha'(t)\| = 1\ \forall t \in I$ (unit-speed). Concluímos que $\alpha' : I \longrightarrow S^1$ e tomamos sua função ângulo θ. Definimos então sua curvatura $\kappa(t)$ da seguinte forma:

$$\kappa(t) = \theta'(t) = \det(\alpha'(t), \alpha''(t))$$

Como $\alpha'(t)$ é um vetor unitário, concluímos que $|\kappa(t)| = \|\alpha''(t)\|$.

Por exemplo, consideremos o círculo de raio 2, centrado na origem. Calculamos a seguir a primeira derivada da sua expressão:

```
circexpr = [:(r*cos(t)), :(r*sin(t))]
r = 2
circexprd1t = df.(circexpr, :t)
```

Prosseguimos calculando o comprimento de arco:

```
:(trigsimp(sqrt($(circexprd1t[1])^2 + $(circexprd1t[2])^2))) |> rcall
```

Com isso, obtemos $|r| = r$. Concluímos, portanto, que a invérsa comprimento de arco é dada por $\varphi(s) = \frac{s}{r}$.

Reparametrizamos, então, o círculo e calculamos suas derivadas:

```
t = :(s/r)
circexpr = [:(r*cos($t)), :(r*sin($t))]
circexprd1t = df.(circexpr, :s)
circexprd2t = df.(circexpr, :s, 2)
```

Verificamos que a primeira derivada é unitária e prosseguimos calculando o absoluto da curvatura a partir da segunda derivada:

```
:(trigsimp(sqrt($(circexprd2t[1])^2 + $(circexprd2t[2])^2))) |> rcall
```

Concluímos que a curvatura é dada por $\frac{1}{r}$ em todos os pontos.

4 Curvas em Três Dimensões

Para desenhar curvas em 3 dimensões, utilizamos as mesmas funções que já escrevemos:

```
curve3d = [:(cos(t) * log(t+1)), :(sin(t) * log(t+1)), :(t)]

curve3dpnts = curvepoints(curve3d, :t, range(0, 30*π, length=1500))
curve3dvecs = curvevectors(curve3d, :t, range(0, 30*π, length=1500))

plot(
  (curve3dpnts[:,1], curve3dpnts[:,2], curve3dpnts[:,3]);

  linewidth=2,
  size=(600,600),
  label="",
  aspect_ratio=1,
  seriescolor=:inferno,
  line_z=f(x,y,z) = z,
  colorbar_entry=false
)
```

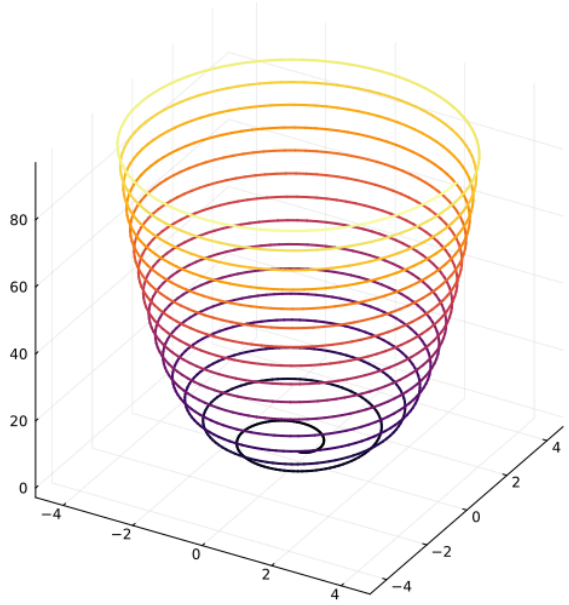

Figura 3: Espiral tridimensional.

4.1 Equações de Frenet

Seja $\alpha(t) : I \longrightarrow \mathbb{R}^3$, $\alpha(t) = (\alpha_1(t), \alpha_2(t), \alpha_3(t))$, tal que:

- $\|\alpha''(t)\| \neq 0 \ \forall t$;
- $\|\alpha'(t)\| = 1 \ \forall t$

Com isso, definimos a seguinte base ortonormal para o $\mathbb{R}^3$:

$$T := \frac{dr}{ds}; \quad N := \frac{\frac{dT}{ds}}{\|\frac{dT}{ds}\|}; \quad B := T \times N$$

Concluimos que:

$$\begin{bmatrix} T' \\ N' \\ B' \end{bmatrix} = \begin{bmatrix} 0 & \kappa & 0 \\ -\kappa & 0 & \tau \\ 0 & -\tau & 0 \end{bmatrix} \begin{bmatrix} T \\ N \\ B \end{bmatrix}$$

Onde κ é a curvatura e τ a torsão de α.