Frederico José Ribeiro Pelogia

Lista 3 - Controle de Sistemas Dinâmicos
Derivada e Integral Numérica

São José dos Campos - Brasil

Dezembro de 2020

Frederico José Ribeiro Pelogia

Lista 3 - Controle de Sistemas Dinâmicos
Derivada e Integral Numérica

Relatório apresentado à Universidade Federal de São Paulo como parte dos requisitos para aprovação na disciplina Controle de Sistemas Dinâmicos

Docente: Prof. Dr. Henrique Mohallem Paiva

Universidade Federal de São Paulo - UNIFESP

Instituto de Ciência e Tecnologia - Campus São José dos Campos

São José dos Campos - Brasil

Dezembro de 2020

Sumário

1 Introdução

Sistemas dinâmicos podem ser descritos por uma ou mais equações diferenciais, que podem ser complexas demais para serem solucionadas analiticamente. Dessa forma, para criar modelos e simulações para esse tipo de sistema, são recomendadas técnicas de integração numérica.

O propósito desse relatório é investigar a preferência pela integração numérica em relação à derivação numérica, quando tratando-se de situações com ruído, que sempre estarão presentes em problemas reais. Os testes realizados utilizarão o Método de Euler, que apresenta uma forma simples, mas razoável, de tratar equações diferenciais computacionalmente (1).

O método e as simulações serão implementados na linguagem de programação C e os dados, salvos em um arquivo de texto. Para a criação dos gráficos a partir dos dados será utilizada a liguagem de programação Python, com a biblioteca gráfica `matplotlib`.

2 Desenvolvimento

2.1 Código com os cálculos

Primeiramente será apresentado o código na íntegra e depois serão comentadas as etapas para sua construção e será detalhado seu funcionamento.

2.1.1 Código na íntegra

O código que realiza os cálculos das derivadas e integrais está apresentado no Listing 2.1.

Code Listing 2.1 – Código em C para cálculo das derivadas e integrais

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<math.h>
4
5  // Funcao que gera uma distribuicao normal
6  float * normal_dist(int n, float step, float mean, float std){
7      // numero de termos: n / step
8      // media: mean
9      // desvio padrao: std
10     int i;
11     float *nvec;
12     nvec = malloc((n/step) * sizeof(float));
13     for(i = 0; i < n/step ; i++){
14         nvec[i] = exp(-0.5*pow((i*step - mean)/std, 2))/(std*sqrt(2*M_PI));
15     }
16     return nvec;
17 }
18
19 int main(int argc, char* argv[]){
20     // Declarando variaveis importantes
21     int i,t_max = 5;
22     float *t, *y,*yd_ref, *yi_ref, *yd, *z, *zd, *zi,*noise,  step = 0.01,
       offset = 0;
23
24     // Abrindo arquivo de texto para escrita
25     FILE* fdata = fopen("data.txt","w");
26
27     // Recebendo argumentos pela linha de comando
28     if(argc > 1){
29         t_max = atoi(argv[1]);
```

```
30          step = atof(argv[2]);
31          offset = atof(argv[3]);
32      }
33
34      // Alocando memoria para as variaveis
35      t = malloc((t_max/step) * sizeof(float));
36      y = malloc((t_max/step) * sizeof(float));
37      yd_ref = malloc((t_max/step) * sizeof(float));
38      yi_ref = malloc((t_max/step) * sizeof(float));
39      yd = malloc((t_max/step) * sizeof(float));
40      z = malloc((t_max/step) * sizeof(float));
41      zd = malloc((t_max/step) * sizeof(float));
42      zi = malloc((t_max/step) * sizeof(float));
43
44      // Vetor com distribuicao normal de media 0 e desvio padrao 0.2
45      noise = normal_dist(t_max, step, 0, 0.2);
46
47      //Valores iniciais para y, sua derivada e sua integral
48      y[0] = 0;
49      zd[0] = 0;
50      zi[0] = 0;
51
52
53      // loop principal pelos instantes de tempo
54      for(i = 1; i < t_max/step; i++){
55          //t: vetor com os instantes de tempo
56          t[i] = i*step - offset;
57
58          //y: y(t) = t^2
59          y[i] = pow(t[i],2);
60
61          //yd_ref: derivada teorica de y
62          yd_ref[i] = 2*t[i];
63
64          //yi_ref: integral teorica de y
65          yi_ref[i] = pow(t[i],3)/3;
66
67          //yd: derivada numerica de y (Metodo de Euler)
68          yd[i] = (y[i] - y[i-1])/step;
69
70          //z: z(t) = y(t) + ruido branco gaussiano
71          z[i] = y[i] + noise[rand()% (int) (t_max/step)]; // amostra
        aleatoria da distr. normal
72
73          //zd: derivada numerica de y com ruido (Metodo de Euler)
74          zd[i] = (z[i] - z[i-1])/step;
75
```

```
76          //zd: derivada numerica de y com ruido (Metodo de Euler)
77          zi[i] = zi[i-1] + z[i]*step;
78
79          // Gravando os dados no arquivo de texto
80          fprintf(fdata, "%f %f %f %f %f %f %f %f\n", t[i], y[i], yd[i],
    yd_ref[i], yi_ref[i], z[i], zd[i], zi[i]);
81      }
82
83      // fechando arquivo de texto
84      fclose(fdata);
85
86      // desalocando as variaveis
87      free(t);
88      free(y);
89      free(yd_ref);
90      free(yi_ref);
91      free(yd);
92      free(z);
93      free(zd);
94      free(zi);
95      free(noise);
96
97      return 0;
98  }
```

Pode-se perceber que o código inicia com a declaração e a alocação das variáveis necessárias, além da definição de alguns valores iniciais. Em seguida, é criado um loop principal que itera pelos instantes de tempo, populando os vetores e fazendo a escrita em um arquivo de texto de saída. Após isso, o arquivo de saída é fechado e as variáveis são desalocadas.

Uma maneira muito mais eficiente, em termos de memória, de se escrever o código seria armazenar apenas o valor das variáveis no tempo presente e no imediatamente anterior. Entretanto, armazenar o histórico completo das variáveis, como foi feito no código, poderia ser interessante caso processamentos mais elaborados fossem necessários.

2.1.2 Variáveis básicas

As variáveis *t, y, yd_ref, yi_ref* são essenciais para o presente relatório.

- A variável t é um vetor com os instantes de tempo de 0 a 5 segundos com passo 0.01, representando uma amostragem a cada 10 ms. Note, na linha 56 do Listing 2.1, que é definido um *offset* opcional, inicializado como 0 por padrão.

- A variável y é um vetor em que cada elemento na posição i recebe o valor de $(t[i])^2$, representando a função $y(t) = t^2$.

- As variáveis *yd_ref* e *yi_ref*, são vetores que possuem na posição i, respectivamente, $2t[i]$ e $\frac{(t[i])^3}{3}$, representando a derivada e a integral da função $y(t)$. É interessante notar que essas duas variáveis assumem conhecimento da derivada e da integral teóricas da função $y(t)$. Isso ocorre pois elas serão utilizadas como referência para avaliar o desempenho das aproximações numéricas.

2.1.3 Derivada numérica sem ruído

A variável *yd* é um vetor que recebe, na posição i,

$$y_d[i] = \frac{y[i] - y[i-1]}{\Delta t},$$

que é o valor da derivada numérica da função $y(t)$ no instante $t = t[i]$, utilizando a aproximação do Método de Euler.

2.1.4 Adição de ruído

Para a adição de um ruído branco gaussiano à variável y, foram realizadas as seguintes etapas:

- Criação de uma função que retorna um vetor com uma distribuição normal com média μ e desvio padrão σ. A função criada foi a *normal_dist*, cuja definição está entre as linhas 6 a 17 do Listing 2.1. A função de densidade de probabilidade utilizada como base (2) foi
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

- Chamada da função para criação de um vetor *noise* com uma distribuição normal de média 0 e desvio padrão 0.2.

- Criação da variável z que possui, na posição i, o valor de $y[i]$ somado com uma **amostra aleatória** do vetor *noise*. A amostra aleatória é retirada calculando-se um índice através do resto da divisão de um número aleatório pelo número de elementos do vetor, como pode se observar na linha 71 do Listing 2.1.

2.1.5 Derivada numérica com ruído

A variável *zd* é um vetor que recebe, na posição i,

$$z_d[i] = \frac{z[i] - z[i-1]}{\Delta t},$$

que é o valor da derivada numérica da função $z(t)$ no instante $t = t[i]$, utilizando a aproximação do Método de Euler.

2.1.6 Integral numérica com ruído

A variável *yd* é um vetor que recebe, na posição i,

$$z_i[i+1] = z_i[i] + \Delta t z[i],$$

que é o valor da integral numérica da função $z(t)$ no instante $t = t[i]$, utilizando a aproximação do Método de Euler.

2.2 Código para geração dos gráficos

Para a criação dos gráficos a partir dos dados salvos no arquivo de texto, foi escrito um código em Python utilizando a bilbioteca gráfica `matplotlib`. O código está apresentado no Listing 2.2.

Code Listing 2.2 – Código Python para geração dos gráficos.

```
1  import matplotlib.pyplot as plt
2
3  t = []
4  y = []
5  yd = []
6  yd_ref = []
7  yi_ref = []
8  z = []
9  zd = []
10 zi = []
11
12 # Abertura do arquivo com os dados
13 with open('data.txt','r') as f:
14     # Leitura linha a linha do arquivo
15     for line in f.readlines():
16         values = line[:-1].split() # separacao das colunas
17         # guardando os valores nas listas
18         t.append(float(values[0]))
19         y.append(float(values[1]))
20         yd.append(float(values[2]))
21         yd_ref.append(float(values[3]))
22         yi_ref.append(float(values[4]))
23         z.append(float(values[5]))
24         zd.append(float(values[6]))
25         zi.append(float(values[7]))
26
27 # Criacao dos graficos
28 fig, axs = plt.subplots(1, 2, figsize=(14,4))
29 fig.suptitle('Derivada sem ruido')
30 axs[0].plot(t, y)
```

```
axs[0].set(xlabel='t(s)', ylabel='Funcao y = x^2')
axs[0].grid()
axs[1].plot(t,yd, label='Numerica')
axs[1].set(xlabel='t(s)', ylabel='Derivada de y(t)')
axs[1].plot(t,yd_ref, label='Teorica')
axs[1].grid()
axs[1].legend()

fig, axs = plt.subplots(figsize=(10,5))
fig.suptitle('Adicao de ruido')
axs.plot(t, y, label='y(t) = x^2',zorder=2)
axs.set(xlabel='t(s)', ylabel='Funcao y(t) e z(t)')
axs.plot(t,z, label='z(t) = y(t) + ruido',zorder=1)
axs.grid()
axs.legend()

fig, axs = plt.subplots(1, 2, figsize=(14,4))
fig.suptitle('Derivada com ruido')
axs[0].plot(t, z)
axs[0].set(xlabel='t(s)', ylabel='Funcao z = x^2 + ruido')
axs[0].grid()
axs[1].plot(t,zd, label='Numerica com ruido (dz/dt Metodo de Euler)')
axs[1].set(xlabel='t(s)', ylabel='Derivada')
axs[1].plot(t,yd_ref, label='Teorica sem ruido (dy/dt)')
axs[1].grid()
axs[1].legend()

fig, axs = plt.subplots(1,2, figsize=(14,4))
fig.suptitle('Integral com ruido')
axs[0].plot(t, z)
axs[0].set(xlabel='t(s)', ylabel='Funcao z = x^2 + ruido')
axs[0].grid()
axs[1].plot(t,zi, label='Integral Numerica com ruido (Metodo de Euler)')
axs[1].set(xlabel='t(s)', ylabel='Integral')
axs[1].plot(t,yi_ref, label='Integral Teorica sem ruido')
axs[1].grid()
axs[1].legend()

plt.show()
```

Para a execução dos dois códigos em conjunto, foi criado o script *run.sh*, apresentado no Listing 2.3.

Code Listing 2.3 – Script para execução dos códigos produzidos.

```
#!/bin/bash
gcc deriv_int.c -o di -lm
./di 5 0.01 0
```

```
4 python plot.py
```

3 Resultados e Discussões

3.1 Derivada sem ruído

A Figura 1 apresenta o gráfico de $y(t)$ e um gráfico que compara yd com yd_ref, isto é, a aproximação numérica com o valor teórico da derivada.

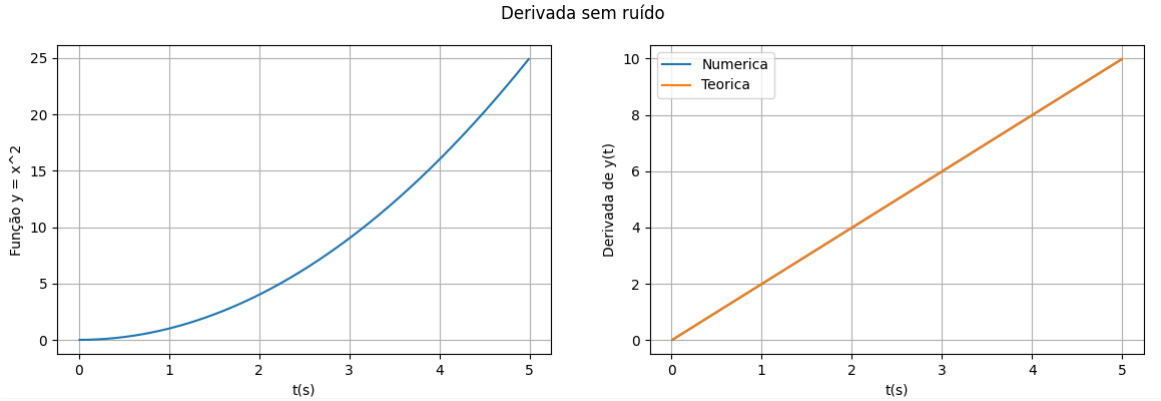

Figura 1 – Gráfico da função $y(t)$ e de suas derivadas teórica e numérica. Fonte: Autor

Percebe-se que, sem a presença de ruído, as curvas da derivada teórica e da numérica estão sobrepostas, indicando que a aproximação foi bem sucedida e praticamente exata.

3.2 Adição de ruído

A Figura 2 apresenta um gráfico que compara a função $y(t)$ e sua versão $z(t)$, que apresenta a adição de ruído.

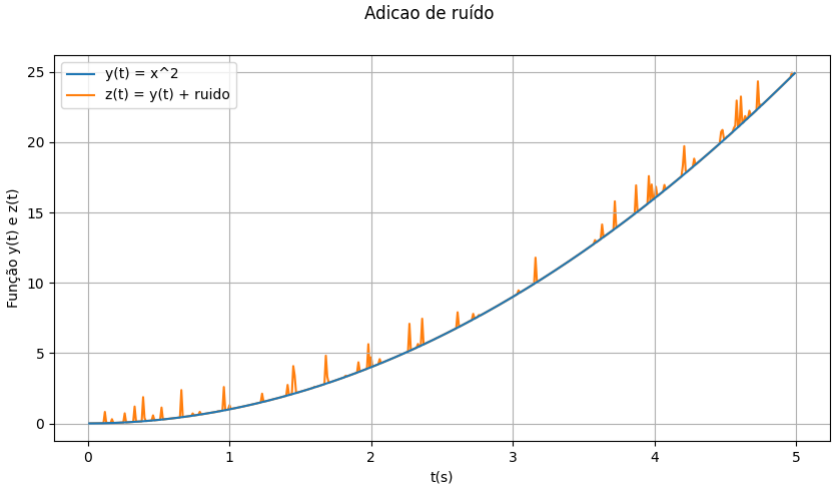

Figura 2 – Gráficos das funções $y(t) = t^2$ e $z(t) = y(t) +$ ruído. Fonte: Autor

Nota-se que o ruído adicionado gera alguns pequenos picos na curva original de $y(t)$. O procedimento para adição do ruído foi abordado no Capítulo de Desenvolvimento.

3.3 Derivada com ruído

A Figura 3 apresenta o gráfico da função $z(t)$ e um gráfico que compara a derivada teórica de $y(t)$ com a aproximação pelo método de Euler da derivada de $z(t)$.

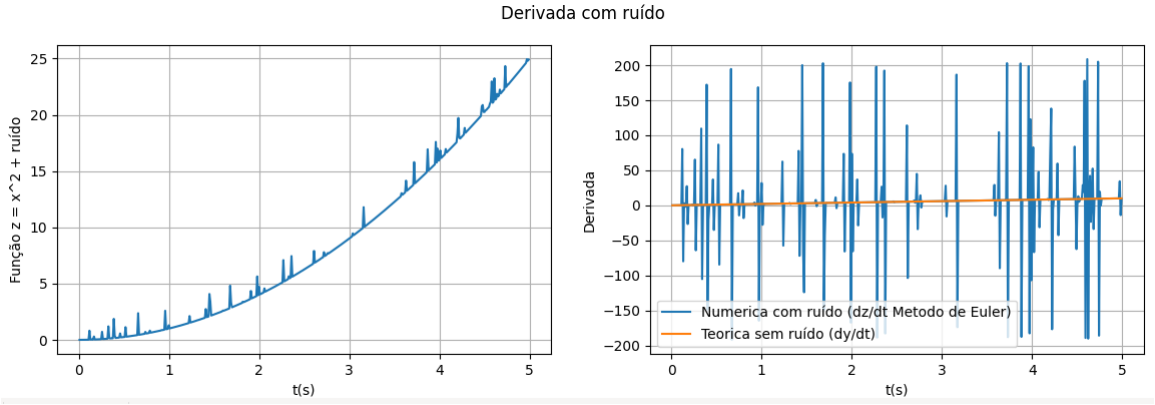

Figura 3 – Gráfico da função $z(t)$ e uma comparação de sua derivada numérica com a teórica de $y(t)$. Fonte: Autor

Nota-se que, mesmo que pequeno, o ruído fez com que a derivda numérica ficasse extremamente imprecisa, atingindo valores muito diferentes dos esperados pela derivada teórica.

3.4 Integral com ruído

A Figura 4 apresenta o gráfico da função $z(t)$ e um gráfico que compara a integral teórica da função $y(t)$ com a integral numérica, pelo Método de Euler, de $z(t)$.

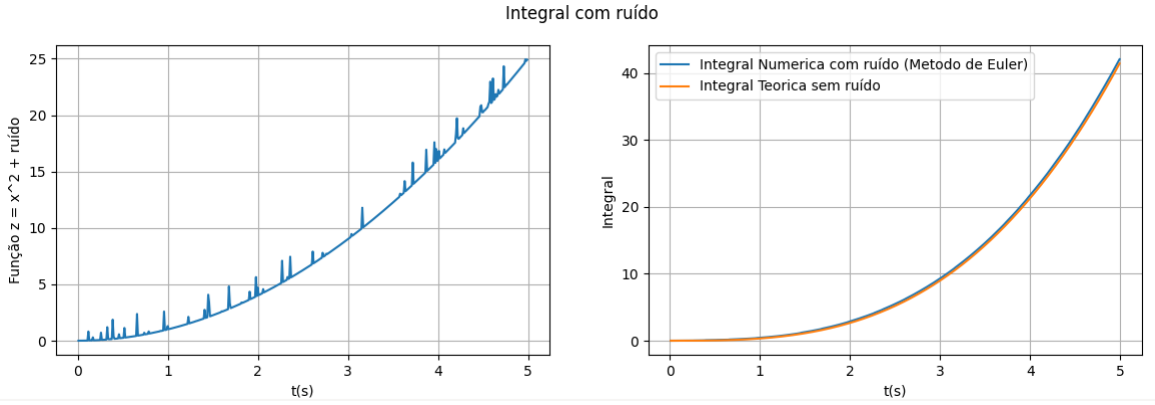

Figura 4 – Gráfico da função $z(t)$ e uma comparação de sua integral numérica com a teórica de $y(t)$. Fonte: Autor

Percebe-se que, diferentemente da derivada numérica, a integral numérica apresentou uma boa aproximação para seu valor teórico, dado que as curvas estão quase sobrepostas na Figura 4. Essa robustez para lidar com ruído é o principal motivo da utilização de integradores em simulações numéricas de EDOs, em detrimento de derivadores.

4 Considerações Finais

Assim, os códigos escritos e os testes realizados foram sufientes para notar que a derivação numérica é muito mais sensível e menos robusta à presença de ruído do que a integração numérica. Isso motiva a preferência pela utilização de integradores à derivadores para simulações computacionais de sistemas dinâmicos e equações diferenciais.

Referências

1 DAWKINS, P. *Differential Equations - Eulers Method*. 2003. Acessado em 24 de Dezembro de 2020. Disponível em: <https://tutorial.math.lamar.edu/Classes/DE/EulersMethod.aspx>. Citado na página 3.

2 WEISSTEIN, E. W. *Normal distribution*. MathWorld–A Wolfram Web Resource, 2020. Acessado em 24 de Dezembro de 2020. Disponível em: <https://mathworld.wolfram.com/NormalDistribution.html>. Citado na página 8.