Relatório do EP de MAC0209

João Pedro Feitosa
Marcelo Nascimento
Renan Ryu Kajihara

8 de abril de 2024

Resumo

O presente relatório descreve as atividades realizadas no segundo Exercício Programa da disciplina "Modelagem e Simulação (MAC0209)", que consiste em realizar a modelagem e simulação de três processos distintos: Movimento 1D, queda livre e sistemas dinâmicos caóticos, analisando os resultados de cada um desses modelos determinísticos, observando suas particularidades.

Conteúdo

1 Introdução

Com o intuito de estudar aplicações de modelagem e simulação determinísticas, realizou-se 3 experimentos simples. Cada experimento consistiu da simulação seguida da visualização do movimento de partículas inseridas em diferentes contextos de movimento e de comportamento caótico:

1. Movimento Uniforme
2. Queda Livre
3. Sistemas Dinâmicos Caóticos

2 Objetivos

O exercício teve como objetivo a implementação e visualização dos modelos fornecidos, com o intuito de compreender suas distinções, assim como o uso de diferentes técnicas de simulação.

3 Cronograma

Divisão das tarefas e implementação dos algoritmos: 25/03 - 28/03.
Redação do relatório: 29/03 - 01/04.

4 Dados e métodos

A implementação de cada modelo resulta em um conjunto de dados, que pode ser visualizado por meio de bibliotecas como o Matplotlib, aqui utilizada na formação de gráficos e o Pandas, aqui utilizado para a formação de tabelas. Constam, a seguir, os métodos experimentais utilizados em cada um dos experimentos realizados.

4.1 Movimento 1D Uniforme

Primeiramente, utilizou-se a biblioteca NumPy do Python para gerar um vetor correspondente a um período de tempo equispaçado de 10 segundos, medido de 0,1 em 0,1 segundo.

Criou-se, em seguida, uma função "visualizacao". Essa função tem como objetivo, a partir de valores fornecidos de posição inicial e de velocidade de uma partícula em movimento uniforme, calcular sua trajetória a partir do método analítico. A função realiza a operação representada abaixo, para cada tempo armazenado em um dado vetor de tempos, na qual S(t) representa a posição de uma partícula em função de um tempo t, v representa a velocidade da partícula e S_0 representa sua posição inicial.

$$S(t) = S_0 + v.t$$

Criado o vetor correspondente à trajetória da partícula através do intervalo de tempo atribuído, a função "visualizacao", com o auxílio da biblioteca Matplotlib, gera a visualização desta trajetória em função do tempo.

A partir do vetor de tempo previamente criado foram simuladas as trajetórias de partículas para múlitplas velocidades. Os resultados obtidos podem ser visualizados na seção 5.1 do presente relatório.

4.2 Queda Livre - Modelagem de Sistema Dinâmico

A fim de se simular o movimento de partículas no contexto de queda livre, utilizou-se o método de Euler para se calcular a varição da velocidade e da posição de ditas partículas em função do tempo.

Utilizou-se, mais especificamente, a técnica conhecida como vetor de estados, que representa o estado de uma partícula em um determinado momemento do tempo. Este vetor, cuja estrutura é retrata abaixo, é dado, neste contexto, pelo quarteto: aceleração da gravidade (g), velocidade da partícula (v), posição (altura) da partícula (s) e o tempo (t).

$$\vec{V} = [g, v, s, t]$$

Para cada momento no tempo, o vetor de estados adota uma composição distinta. Uma vez que as medidas de aceleração, velocidade e posição de uma partícula estão correlacionadas, entretanto, um vetor de estados de um determinado tempo pode ser calculado a partir do vetor de estados imediatamente anterior.

Utilizando-se a função desenvolvida "proximo-estado", calcula-se a variação do vetor de estados de uma partícula ao longo do tempo, de acordo com a seguinte equação, na qual $\vec{V_t}$ representa o vetor de estados no momento t e $\vec{R_t}$ representa o vetor de variação no mesmo momento:

$$\vec{V}_{t+1} = \vec{V}_t + dt.\vec{R}_t$$
$$\vec{V}_{t+1} = [g, v, s, t] + dt.[0, g, v, 1]$$

Repare que:

- A aceleração da gravidade não varia com o tempo
- A velocidade varia em função da aceleração
- A posição varia em função da velocidade

Utilizando-se o método retratado acima, realizou-se a simulação e a visualização da queda livre de uma partícula situada a 20 metros do solo exposta a uma aceleração de 10 m/s^2. Os resultados obtidos podem ser visualizados na seção 5.2 do presente relatório.

4.3 Sistemas Dinâmicos Caóticos

Os modelos forma definidos em funções que recebem os parâmetros e retornam o calculo desses valores, ao relizar diversas chamadas iterativamente, um conjunto de dados é armazenado e é destinado a biblioteca Mathplot para que sejam vizualizados.

No modelo logístico, para que seja possível a visualização, foi necessário descartar uma quantidade inicial de dados e configurar o Mathplot para mostrar pontos pequenos e que não sejam conectados entre sí. O exemplo utilizado foi obtido no próprio slide da disciplina.

Para simular as equações de Lorenz e gerar os mapas de Lorenz, foi necessário a utilização do método de Euler. Nesse sentido, o método de Euler permite que, à partir da derivada de uma função, utiliza-se a aproximação de Taylor ignorando os termos maiores que segunda ordem, para calcular uma aproximação para o valor de f(t+dt) a partir do valor de f(t). Para a primeira equação de Lorenz, por exemplo, temos que:

$$\frac{dx}{dt} = -ax + ay$$

$$\rightarrow \frac{x(t+dt) - x(t)}{dt} \approx -ax + ay$$

$$\rightarrow x(t+dt) \approx (-ax(t) + ay).dt + x(t)$$

Além disso, para gerar os mapas de Lorenz os parâmetros iniciais constantes devem ser inicializados com valores específicos para que os gráficos possuam seus formatos característicos. Nesse sentido, foi escolhido 10 para o valor de a, 28 para o valor de r e 8/3 para o valor de b.

Ademais, para gerar os pontos do gráfico, foi escolhido 0,01 para o valor de dt, já que o valor de dt deveria ser pequeno o suficente para gerar um número considerável de pontos que, ao serem juntados, formassem os gráficos característicos de Lorenz.

5 Resultados experimentais

Constam, a seguir, os resultados experimentaris em cada um dos experimentos realizados.

5.1 Movimento 1D Uniforme

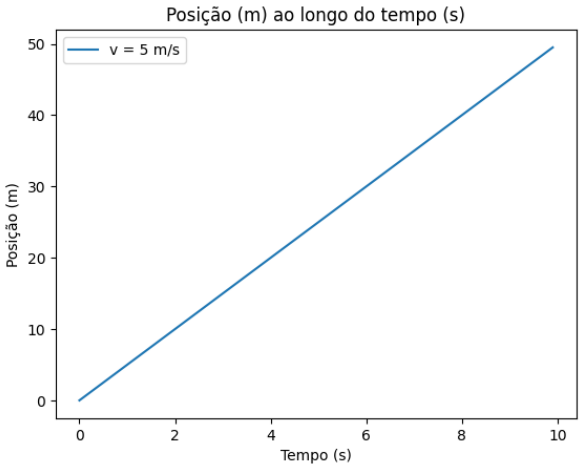


Figura 1: Trajetória Movimento Uniforme - v = 5 m/s

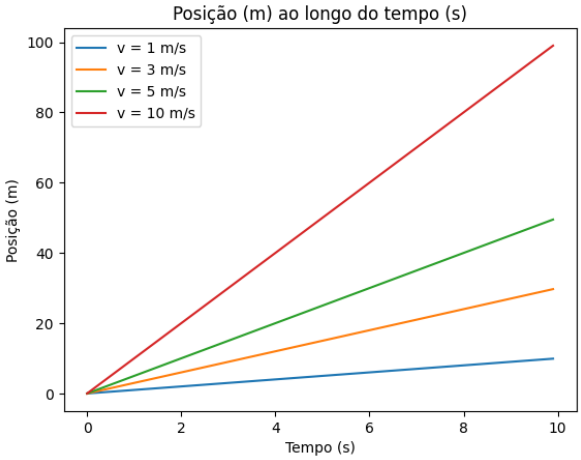


Figura 2: Trajetórias Movimento Uniforme - Comparação de velocidades

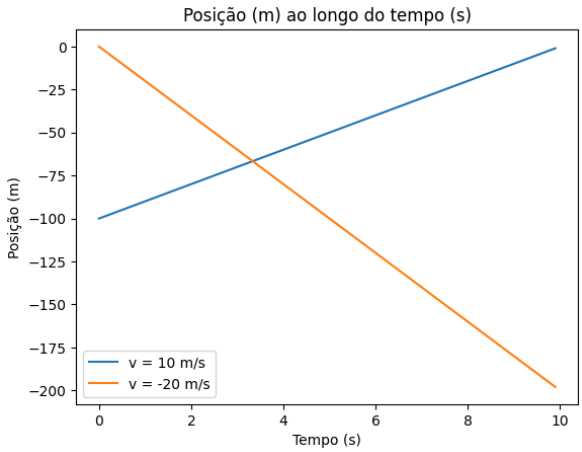


Figura 3: Trajetórias Movimento Uniforme - Velocidades opostas

5.2 Queda Livre - Modelagem de Sistema Dinâmico

Constam, na figura 4, os dados obtidos através da simulação do movimento de uma partícula em queda livre, utilizando-se o método de Euler e dados os seguintes parâmetros iniciais: velocidade inicial (v_0) de 0 m/s (queda livre), aceleração da gravidade (g) de -10 m/s^2 e altura inicial (h) de 10 m. O intervalo de tempo foi definido de $t_0 = 0$ s até t = 2 s, medido a cada 0,1 segundo (dt = 0,1 s).

	g (m/s^2)	v (m/s)	s (m)	t (s)
0	-10	0	20	0
1	-10.0	-1.0	20.0	0.1
2	-10.0	-2.0	19.9	0.2
3	-10.0	-3.0	19.7	0.3
4	-10.0	-4.0	19.4	0.4
5	-10.0	-5.0	19.0	0.5
6	-10.0	-6.0	18.5	0.6
7	-10.0	-7.0	17.9	0.7
8	-10.0	-8.0	17.2	0.8
9	-10.0	-9.0	16.4	0.9
10	-10.0	-10.0	15.5	1.0
11	-10.0	-11.0	14.5	1.1
12	-10.0	-12.0	13.4	1.2
13	-10.0	-13.0	12.2	1.3
14	-10.0	-14.0	10.9	1.4
15	-10.0	-15.0	9.5	1.5
16	-10.0	-16.0	8.0	1.6
17	-10.0	-17.0	6.4	1.7
18	-10.0	-18.0	4.7	1.8
19	-10.0	-19.0	2.9	1.9
20	-10.0	-20.0	1.0	2.0

Figura 4: Tabela de Vetores de Estado

Analisando esses dados pôde-se observar que, enquanto a velocidade cresce de forma linear, a distância varia seguindo uma tendência quadrática, o que é esperado.

Pôde-se reparar também que, após os 2 segundos contabilizados, a altura final da párticula, obtida através do método de Euler, é de 1 metro, isto é, 1 metro acima do solo.

Sabe-se que, utilizando-se o método analítico, pode-se determinar a posição de uma partícula com precisão infinita, isto é, sem erros. A fórmula para a obtenção da posição de uma partícula em função do tempo, dado um movimento uniformemente variado (queda livre), está representada abaixo:

$$S(t) = S_0 + v_0.t + \frac{g}{2}.t^2$$

Utilizando-a, pôde-se calcular a verdadeira posição da partícula no estante t = 2 s, dada as condições previamente explicitadas: S(2) = 0 m. Comparando-se o valor obtido ao medido previamente, utilizando-se o método de Euler, pôde-se perceber uma defasagem de 1 metro. Isso indica que o método de Euler possui imprecisão no seu cálculo.

Com o objetivo de se averiguar o constatado, traçou-se, o gráfico da posição da partícula calculada em cada instante de tempo t, calculada tanto pelo método de euler quanto pelo método analítico (Figura 5). Analisando o gráfico, constatou-se que, a diferença calculada entre os métodos (imprecisão) não só existe de fato mas que ela cresce com o passar do tempo.

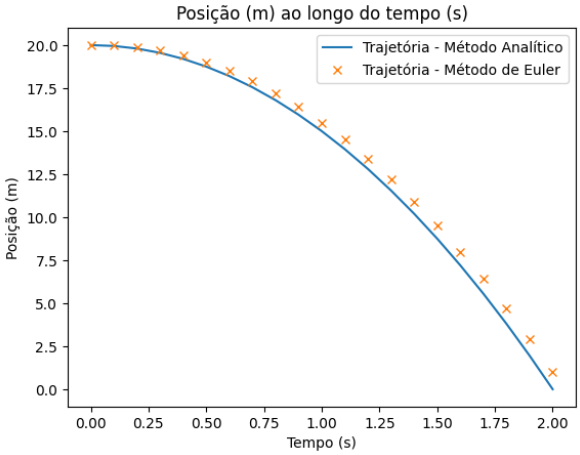


Figura 5: Comparação de trajetórias entre métodos analítico e de Euler - 20 iterações

A fim de investigar como essa imprecisão se comporta, simulou-se o mesmo cenário, isto é, velocidade inicial nula, aceleração da gravidade de -10 m/s^2 e altura inicial de 20 metros, mas dessa vez medido pelo método de Euler em 40 iterações de 0,05 segundo ao invés de 20 iterações de 0,1 segundo. A figura 6 representa o resultado obtido. Analisando o novo gráfico, pôde-se perceber que o erro entre os valores calculados pelo método de Euler e os valores reais correspondentes diminuiu.

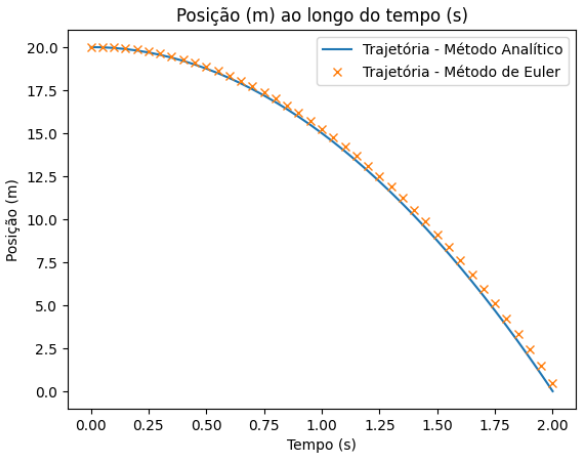


Figura 6: Comparação de trajetórias entre métodos analítico e de Euler - 40 iterações

Com o intuito de entender mais a fundo o comportamento do erro medido pelo método de Euler em função do número de iterações compreendidas por período de tempo, simulou-se o mesmo cenário múltiplas outras vezes, cada vez alterando-se o número de iterações - e o intervalo dt adotado. Em cada uma dessas simulações, anotou-se a posição final da partícula no instante t = 2 s. Os resultados obtidos são esclarecedores e constam na Figura 7.

s (m)	n_iteracoes
1.000.000	20
0.500000	40
0.333333	60
0.250000	80
0.200000	100
0.100000	200
0.020000	1000
0.002000	10000

Figura 7: Tabela de Posições Finais por Número de Iterações

Analisando-se a tabela, pôde-se compreender que, aumentado o número de iterações por período de tempo compreendido e, consequentemente diminuído o intervalo dt adotado, os resultados obtidos se aproximam cada vez mais dos calculados pelo método analítico (S(2) = 0 m). Isso se deve porque o método analítico tem caráter diferencial, isto é, considera parcelas infinitesimais de tempo nos cálculos. Diminuindo o intervalo de tempo dt, aproxima-se cada vez mais de um intervalo de tempo que tende a 0, aumentando-se a precisão dos cálculos.

5.3 Sistemas Dinâmicos Caóticos

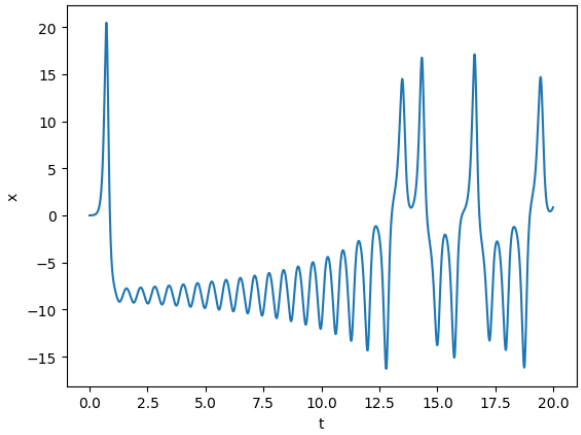


Figura 8: Comportamento dos valores de x ao longo do tempo nas equações de Lorenz

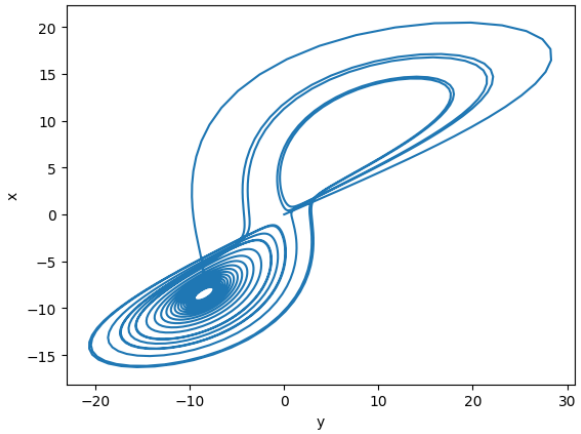


Figura 9: Gráfico entre os valores de x e y ao longo do tempo nas equações de Lorenz

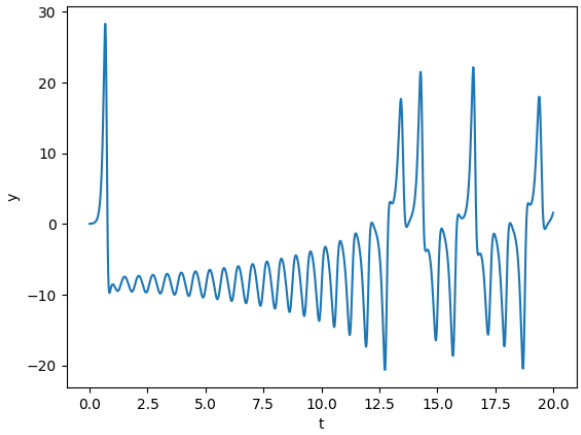


Figura 10: Comportamento dos valores de y ao longo do tempo nas equações de Lorenz

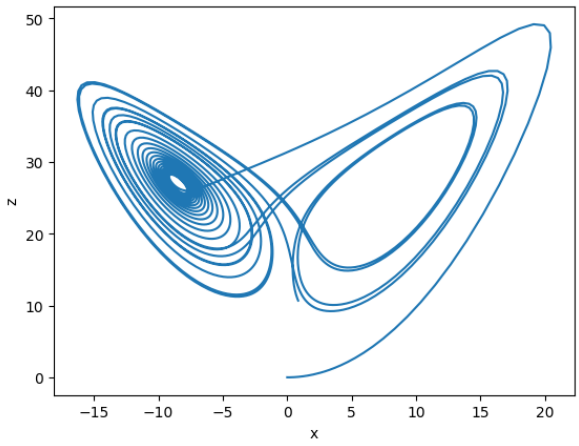


Figura 11: Gráfico entre os valores de z e x ao longo do tempo nas equações de Lorenz

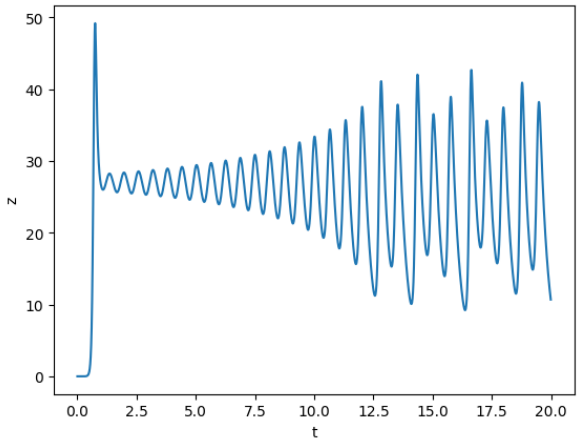


Figura 12: Comportamento dos valores de z ao longo do tempo nas equações de Lorenz

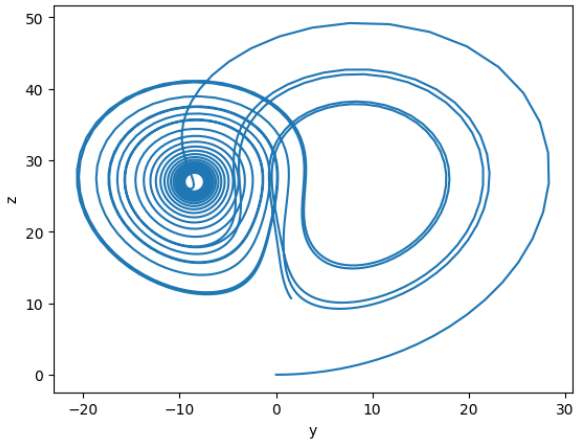


Figura 13: Gráfico entre os valores de z e y ao longo do tempo nas equações de Lorenz

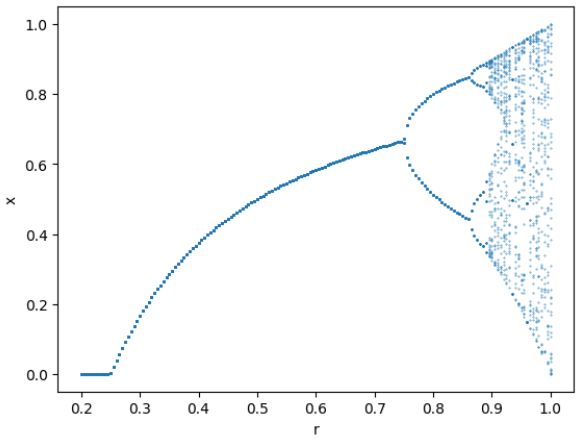


Figura 14: Mapa logístico

Os gráficos de Lorenz mostram que x, y e z possuem um comportamento complexo ao longo do tempo, na qual seus valores aumentam e diminuem varias vezes no intervalo t, fazendo com que o comportamento dessas funções pareçam aleatórias, apesar de não serem. Além disso, é notório que os gráficos que não possuem o parâmetro t possuem um formato semelhante a uma borboleta, sendo esse formato o responsável pela origem da expressão "efeito borboleta", criada, também, por Edward Lorenz.

O gráfico do mapa logístico é registrado a partir de 200 iterações. Seu comportamento se mantem constante até $r = 0.75$, a partir disso, é a primeira bifurcação, seguida de mais duas, até o comportamento caótico do sistema.

6 Conclusão

Após a realização dos experimentos, é possível concluir que o método de Euler é um ferramental útil e poderoso para a simulação de sistemas determinísticos, sobretudo quando se adota um intervalo dt da magnitude correta. Além disso, no contexto dos mapas de Lorenz, foi perceptível a grande influência que os valores iniciais têm em sistemas caóticos - algo que não se observa em tamanha intensidade em sistemas regulares -, em que uma pequena mudança pode alterar completamente o comportamento geral do sistema, tornando a previsão das coordenadas inviável passado algum período de tempo.

7 Notebook

As saídas dos comandos apt-get e pip foram reduzidas, pois, na versão orignal, o conteúdo total preencheu 10 páginas de logs de instalação.

EP2_MAC0209

April 8, 2024

Exercício de MAC0209 - Modelagem e Simulação

João Pedro Feitosa - 10741569 (IME-USP)

Marcelo Nascimento - 11222012 (IME-USP)

Renan Ryu Kajihara - 14605762 (IME-USP)

EP2 - Modelagem Determinística ***

1 Setup

[11]:
```
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
```

2 Lib

[12]:
```
# implementa a posicao de uma particula em um movimento uniforme 1D, dada uma␣
 ↪velocidade v, uma posição inicial s0 e um tempo t
def posicao_uniforme(t, s0, v):
  s = s0 + v*t
  return s

# dados uma dupla (ou mais) de posição inicial e velocidade, gera a trajetória␣
 ↪de uma particula (ou mais) ao longo do tempo

def visualizacao(posicoes_iniciais, velocidades, tempos):
  for s0, v in zip(posicoes_iniciais, velocidades):
    posicoes = []
    for t in tempos:
      posicoes.append(posicao_uniforme(t, s0, v))

    plt.plot(tempos, posicoes, label = f"v = {v} m/s")

  plt.legend()
  plt.xlabel("Tempo (s)")
```

```
  plt.ylabel("Posição (m)")
  plt.title("Posição (m) ao longo do tempo (s)")
  plt.show()
```

[13]:
```
# calcula a variação do vetor de estados, através do método de euler, expressa␣
 ↪uma variação de tempo entre os estados (dt):
def proximo_estado(estado_atual, dt):
  return estado_atual + dt*np.array([0, estado_atual[0], estado_atual[1], 1])


# calcula a posição de uma partícula em um movimento uniformemente variado,␣
 ↪através do método analítico:

def posicao_analitica(s0, v0, a, t):
  s = s0 + v0*t + (a/2)*t**2
  return s
```

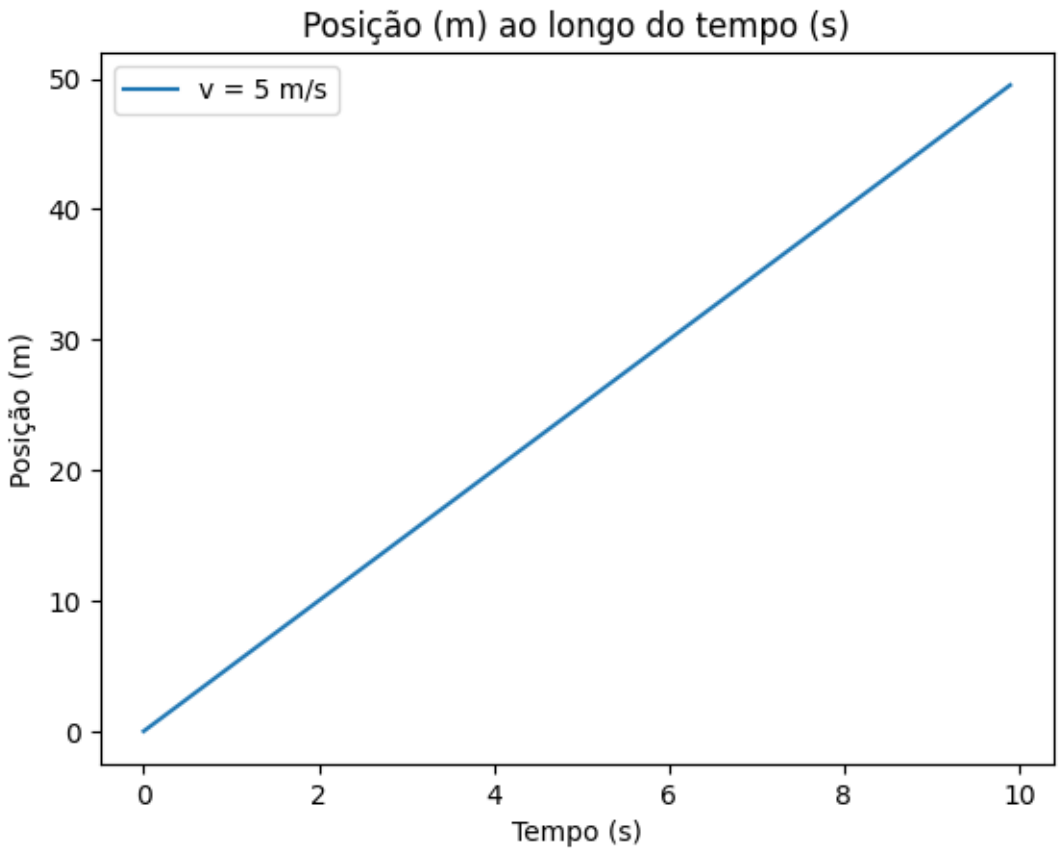
[14]:
```
#implementa a primeira equação dos mapas de Lorenz
def firstequation (a, x, y, t, dt):
  x_1= (-(a*x)+a*y) *dt + x
  return x_1

#implementa a segunda equação dos mapas de Lorenz
def secondequation(r, x, y, z, t, dt):
  y_1= (r*x-x*z-y)*dt+y
  return y_1

#implementa a terceira equação dos mapas de Lorenz
def thirdequation (b, x, y, z, t, dt):
  z_1 = dt*(x*y-b*z)+z
  return z_1

# Implementa o logistic map
def logisticMap(x, r):
  return 4*r*x*(1-x)
```

3 Main 1 - Movimento 1D Uniforme

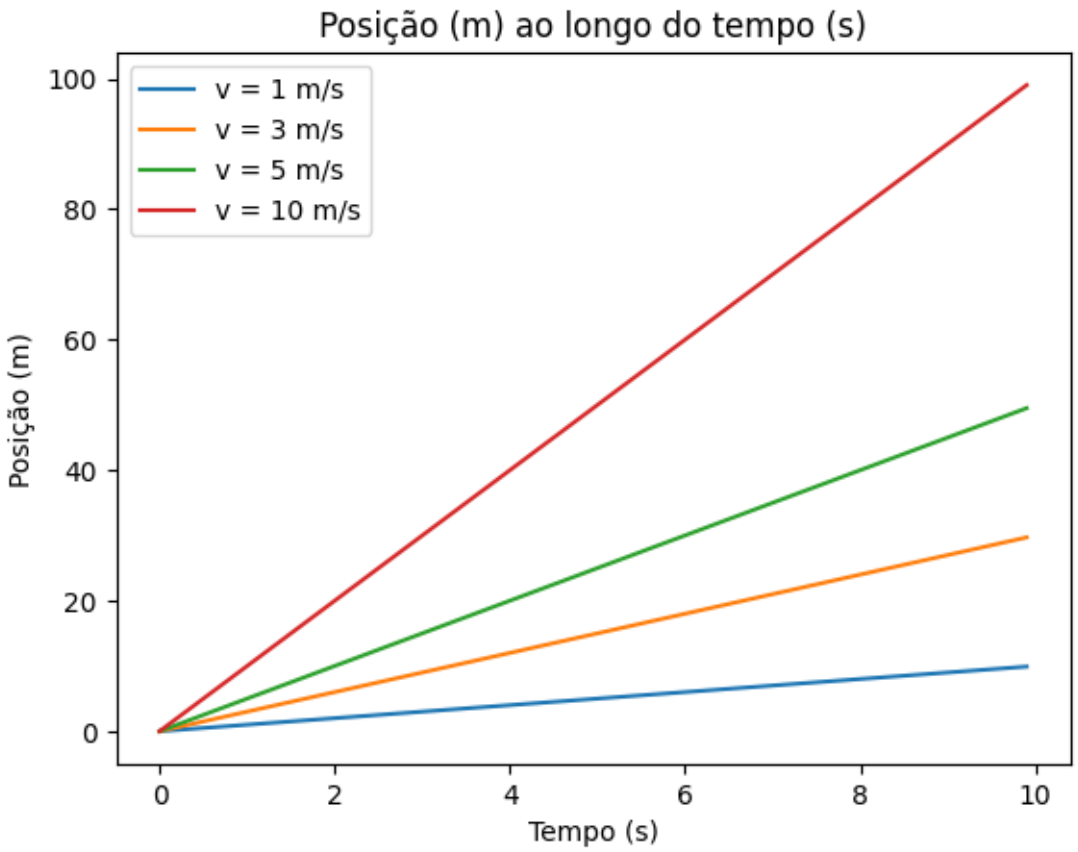
[15]:
```
def main_1():

  # criamos o vetor tempos, que representa o tempo de 0 a 10 segundos medidod␣
 ↪de 0,1 a 0,1 s:
  tempos = np.arange (0, 10, 0.1)
```

```
 # utilizando a função visualizacao criada, geraremos (utilizando o método␣
↪analítico) e visualizaremos a trajetoria de uma particula de velocidade 5 m/
↪s:
 v = 5
 s0 = 0

 print("Trajetória de uma partícula de velocidade 5 m/s e posição inicial 0 m:
↪")
 print()
 visualizacao([s0], [v], tempos)

 # Podemos analisar a trajetórias de múltiplas partículas em movimento␣
↪uniforme 1D com diferentes velocidades:
 velocidades = [1,3,5,10]
 pos_iniciais = [0,0,0,0]

 print()
 print("Trajetórias de partículas de velocidades 1, 3, 5 e 10 m/s e posições␣
↪iniciais 0 m:")
 print()
 visualizacao(pos_iniciais, velocidades, tempos)

 # Podemos, por fim, analisar a trajetória de 2 partículas com velocidades␣
↪(direções) opostas:
 velocidades = [10, -20]
 pos_iniciais = [-100, 0]

 print()
 print("Trajetória de partículas de velocidades opostas:")
 visualizacao(pos_iniciais, velocidades, tempos)

main_1()
```

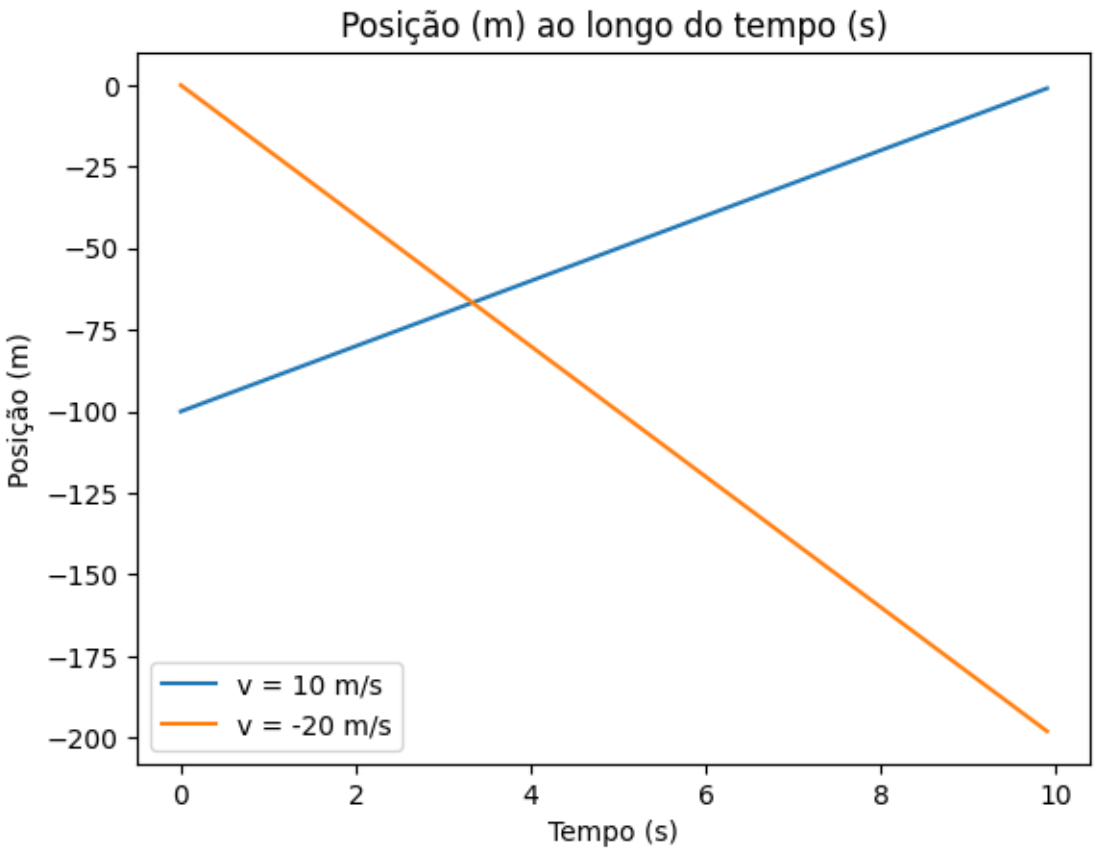
```
Trajetória de uma partícula de velocidade 5 m/s e posição inicial 0 m:
```



Trajetórias de partículas de velocidades 1, 3, 5 e 10 m/s e posições iniciais 0 m:



Trajetória de partículas de velocidades opostas:



4 Main 2 - Queda Livre com Modelagem Dinâmica

[16]:
```
def main_2():

  # Desejamos, primeiramente, simular uma queda-livre simples, calculada por␣
↪meio da abordagem de modelagem dinâmica (vetor de estados)
  # Adotaremos a aceleração da gravidade como 10 m/s2, a posição inicial␣
↪(altura) de 20 m e uma velocidade inicial de 0 m/s (queda livre):

  g = -10
  v0 = 0
  h = 20
  t0 = 0

  # Vetor de estados inicial:
  vetor = np.array([g, v0, h, t0])

  # Calcularemos e expressaremos a variação desse vetor através de 20 iterações␣
↪(desconsiderando a inicial), marcadas por 0,1 segundo:
```

```
print("variação desse vetor através de 20 iterações (desconsiderando a inicial), marcadas por 0,1 segundo:")
print()

estados = pd.DataFrame(columns = ["g (m/s^2)", "v (m/s)", "s (m)", "t (s)"])

for i in range(21):
  estados = pd.concat([estados, pd.DataFrame([dict(zip(estados.columns, vetor))])]).reset_index(drop = True)
  vetor = proximo_estado(vetor, 0.1)

display(estados)

# Utilizaremos o método analítico para averiguar os resultados obtidos pela simulação:

print()
print(f"Posição real da partícula no tempo t = 2, calculado pelo método analítico: {posicao_analitica(20, 0, -10, 2)}m.")

# Vamos analisar melhor essa divergência:

vetor_inicial = vetor = np.array([g, v0, h, t0])

pos_analiticas = []
pos_eulers = []
tempos = []

for i in range(21):
  t = vetor[3]
  s = vetor[2]
  tempos.append(t)
  pos_analiticas.append(posicao_analitica(20, 0, -10, t))
  pos_eulers.append(s)
  vetor = proximo_estado(vetor, 0.1)

print()
print("Divergência entre trajetórias calculadas pelo método analítico e o método de Euler:")
print()

plt.plot(tempos, pos_analiticas, label = "Trajetória - Método Analítico")
plt.plot(tempos, pos_eulers, marker='x', linestyle='None', label = "Trajetória - Método de Euler")
plt.legend()
```

```
plt.xlabel("Tempo (s)")
plt.ylabel("Posição (m)")
plt.title("Posição (m) ao longo do tempo (s)")
plt.show()

# Agora, iremos observar o que ocorre caso calculemos o mesmo período de 2s␣
↪através de 40 iterações, ao invés de 20:

vetor_inicial = vetor = np.array([g, v0, h, t0])

pos_analiticas = []
pos_eulers = []
tempos = []

for i in range(41):
  t = vetor[3]
  s = vetor[2]
  tempos.append(t)
  pos_analiticas.append(posicao_analitica(20, 0, -10, t))
  pos_eulers.append(s)
  vetor = proximo_estado(vetor, 0.05)

print()
print("Divergência entre trajetórias calculadas pelo método analítico e o␣
↪método de Euler - 40 iterações:")
print()

plt.plot(tempos, pos_analiticas, label = "Trajetória - Método Analítico")
plt.plot(tempos, pos_eulers, marker='x', linestyle='None', label =␣
↪"Trajetória - Método de Euler")
plt.legend()
plt.xlabel("Tempo (s)")
plt.ylabel("Posição (m)")
plt.title("Posição (m) ao longo do tempo (s)")
plt.show()

# Agora, vamos observar o mesmo para a velocidade:

vetor_inicial = vetor = np.array([g, v0, h, t0])

v_analiticas = []
v_eulers = []
tempos = []

for i in range(41):
  t = vetor[3]
  v = vetor[1]
```

```
    tempos.append(t)
    v_analiticas.append(v0 + g*t)
    v_eulers.append(v)
    vetor = proximo_estado(vetor, 0.05)

  print()
  print("Divergência entre velocidades calculadas pelo método analítico e o método de Euler - 40 iterações:")
  print()

  plt.plot(tempos, v_analiticas, label = "Velocidade - Método Analítico")
  plt.plot(tempos, v_eulers, marker='x', linestyle='None', label = "Velocidade - Método de Euler")
  plt.legend()
  plt.xlabel("Tempo (s)")
  plt.ylabel("Velocidade (m/s)")
  plt.title("Velocidade (m/s) ao longo do tempo (s)")
  plt.show()

  # Podemos observar como o a altura final calculada varia em função do número de iterações realizada

  print()
  print("Posição final da partícula calculada para diversas quantidades de iterações:")
  print()

  pos_2s = pd.DataFrame(columns = ["s (m)", "n_iteracoes"])

  for n_iteracoes in [20, 40, 60, 80, 100, 200, 1000, 10000]:
    vetor = vetor_inicial = vetor = np.array([g, v0, h, t0])

    for i in range(n_iteracoes):
      vetor = proximo_estado(vetor, 2/n_iteracoes)

    pos_2s = pd.concat([pos_2s, pd.DataFrame([dict(zip(pos_2s.columns, [vetor[2], n_iteracoes]))])]).reset_index(drop = True)

  display(pos_2s)

main_2()
```

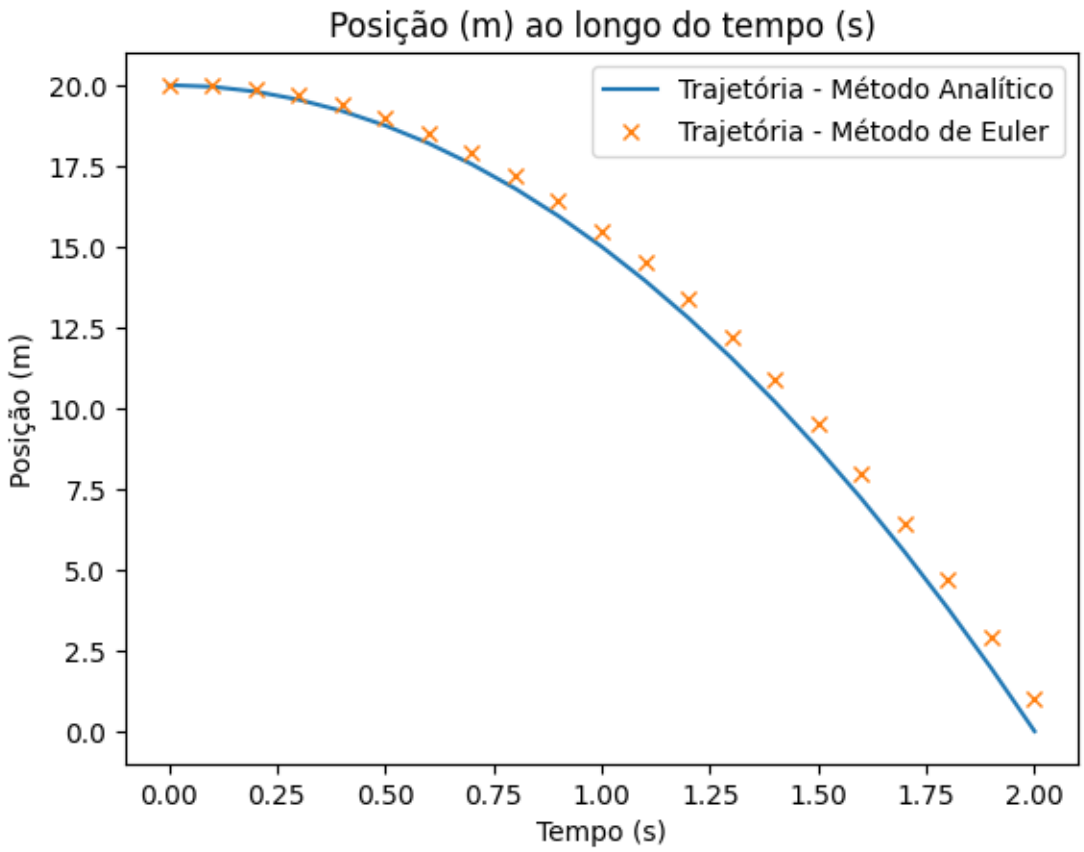
variação desse vetor através de 20 iterações (desconsiderando a inicial), marcadas por 0,1 segundo:

```
   g (m/s^2) v (m/s) s (m) t (s)
```

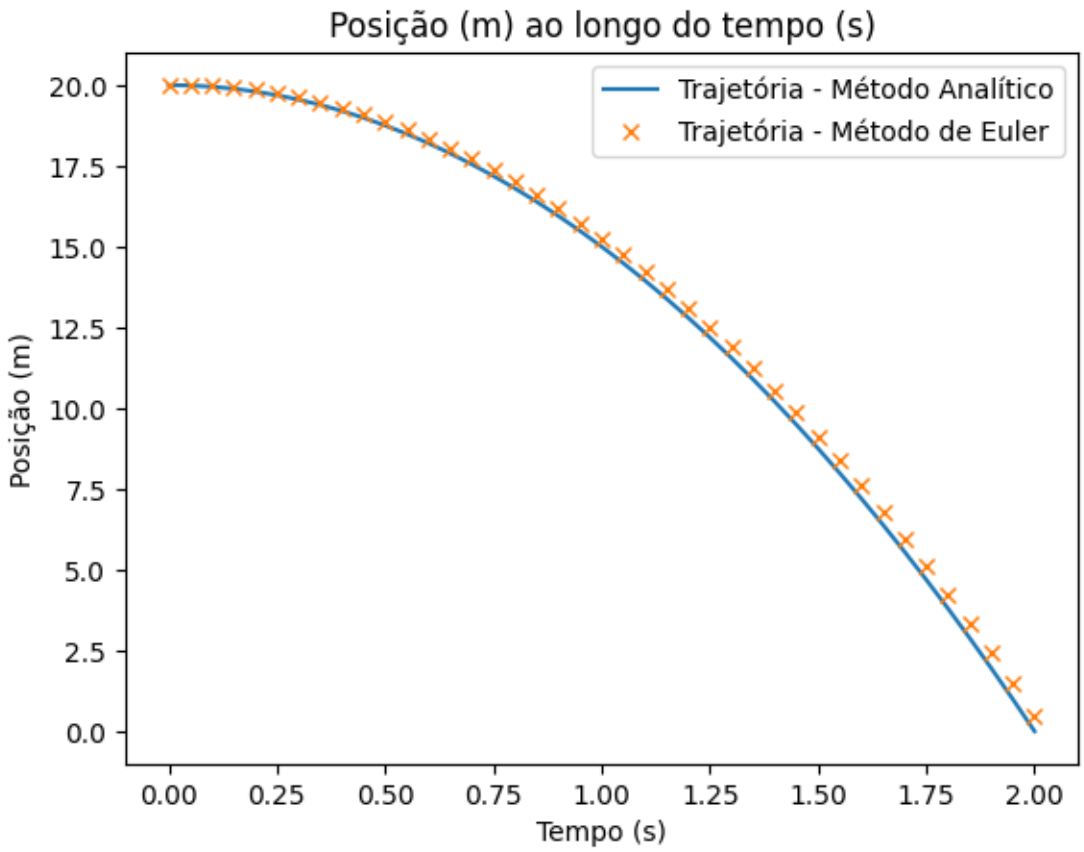
```
0        -10       0    20     0
1      -10.0    -1.0  20.0   0.1
2      -10.0    -2.0  19.9   0.2
3      -10.0    -3.0  19.7   0.3
4      -10.0    -4.0  19.4   0.4
5      -10.0    -5.0  19.0   0.5
6      -10.0    -6.0  18.5   0.6
7      -10.0    -7.0  17.9   0.7
8      -10.0    -8.0  17.2   0.8
9      -10.0    -9.0  16.4   0.9
10     -10.0   -10.0  15.5   1.0
11     -10.0   -11.0  14.5   1.1
12     -10.0   -12.0  13.4   1.2
13     -10.0   -13.0  12.2   1.3
14     -10.0   -14.0  10.9   1.4
15     -10.0   -15.0   9.5   1.5
16     -10.0   -16.0   8.0   1.6
17     -10.0   -17.0   6.4   1.7
18     -10.0   -18.0   4.7   1.8
19     -10.0   -19.0   2.9   1.9
20     -10.0   -20.0   1.0   2.0
```

Posição real da partícula no tempo t = 2, calculado pelo método analítico: 0.0m.

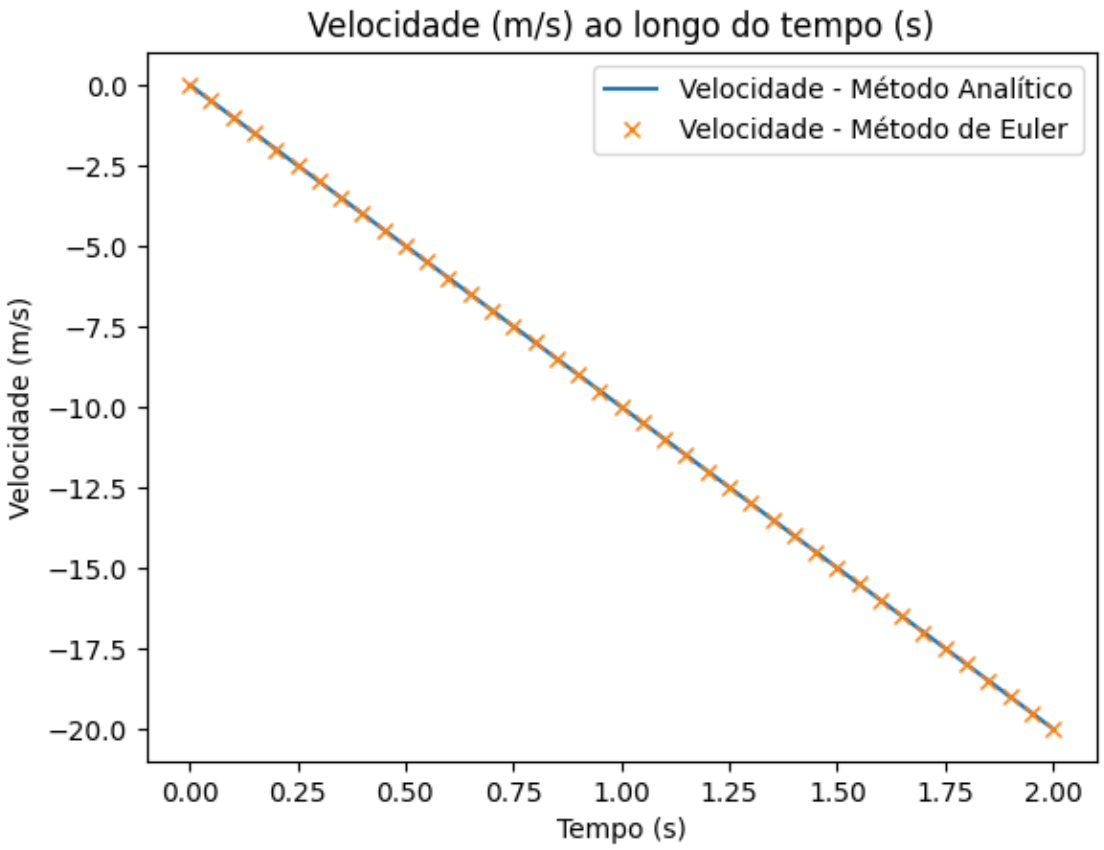
Divergência entre trajetórias calculadas pelo método analítico e o método de Euler:



Divergência entre trajetórias calculadas pelo método analítico e o método de Euler - 40 iterações:



Divergência entre velocidades calculadas pelo método analítico e o método de Euler - 40 iterações:



Posição final da partícula calculada para diversas quantidades de iterações:

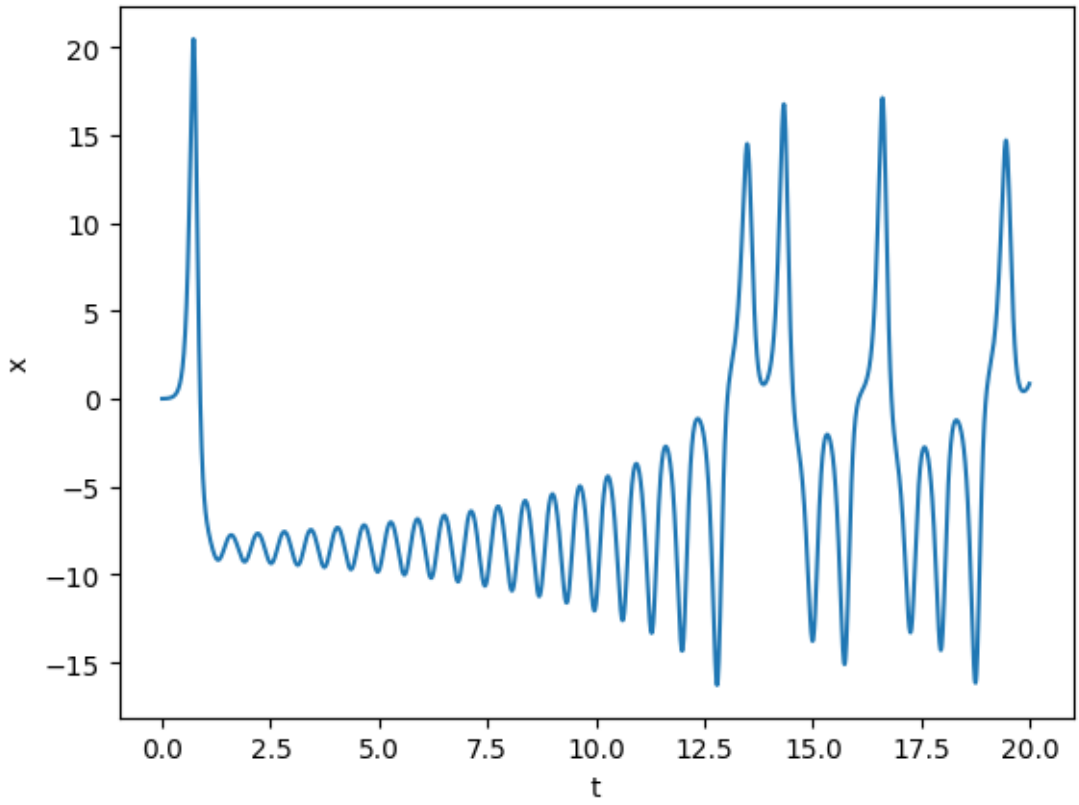
	s (m)	n_iteracoes
0	1.000000	20
1	0.500000	40
2	0.333333	60
3	0.250000	80
4	0.200000	100
5	0.100000	200
6	0.020000	1000
7	0.002000	10000

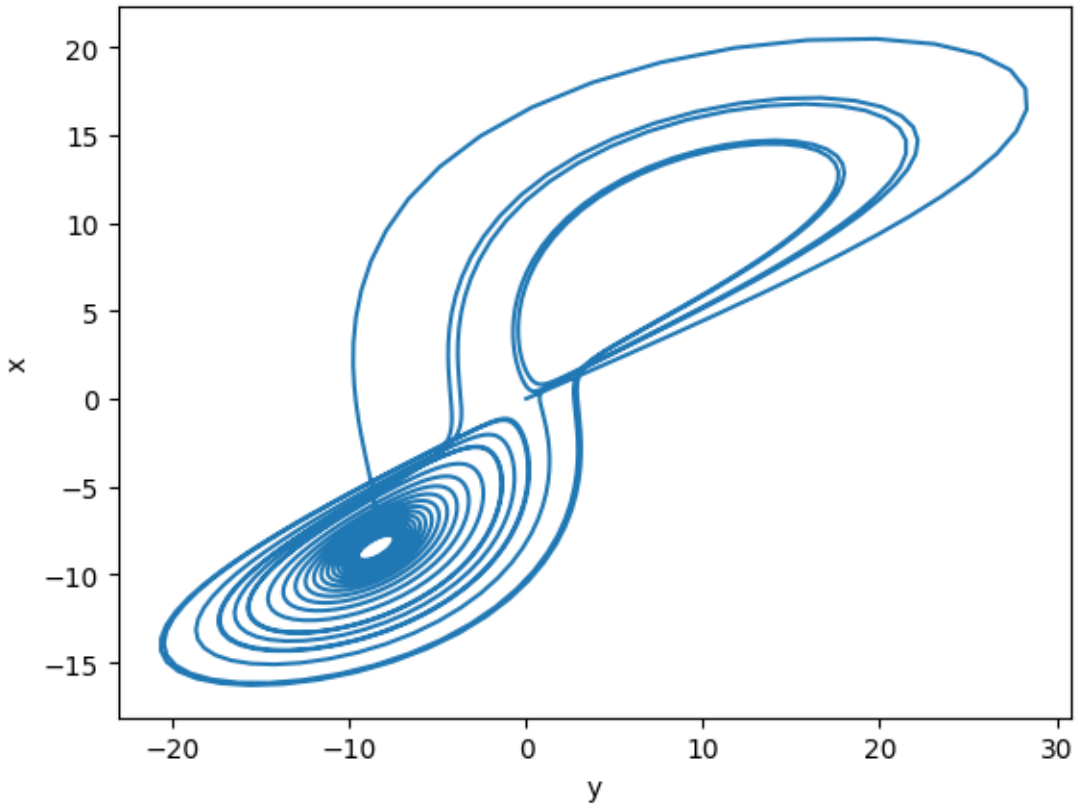
5 Main 3 - Visualização de Sistemas Dinâmicos Caóticos

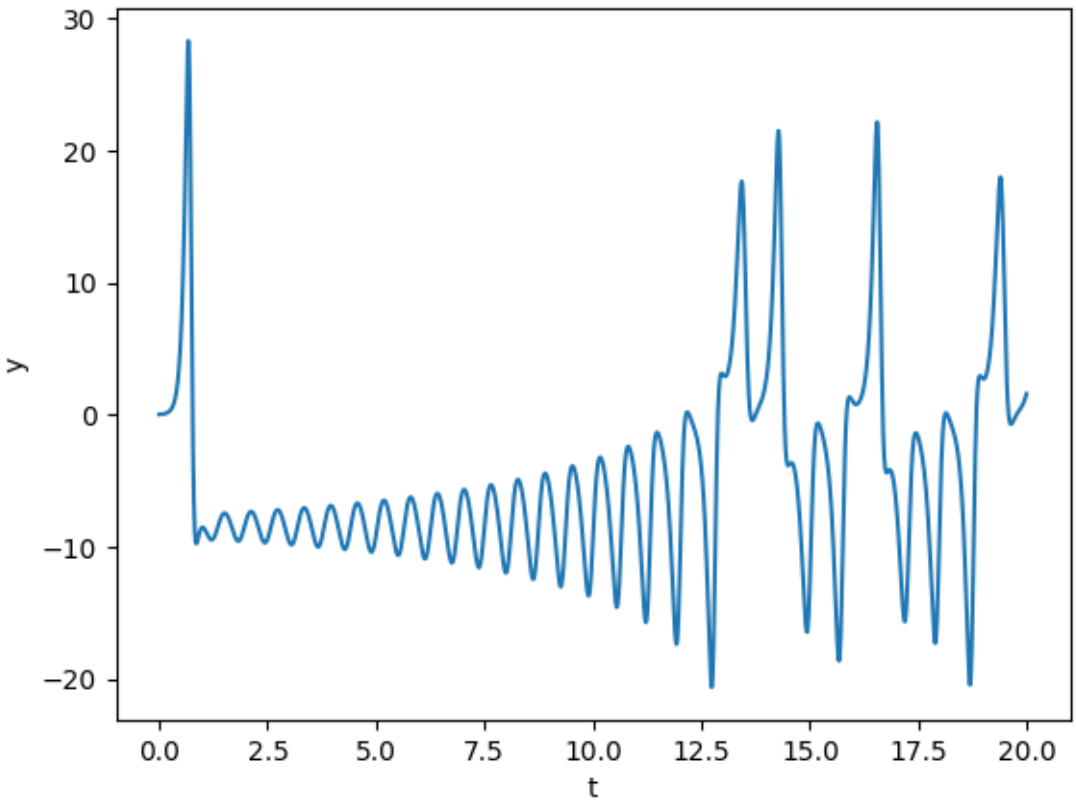
[17]:
```
def main():
  a=10.0
  r=28.0
  b=8.0/3.0
```

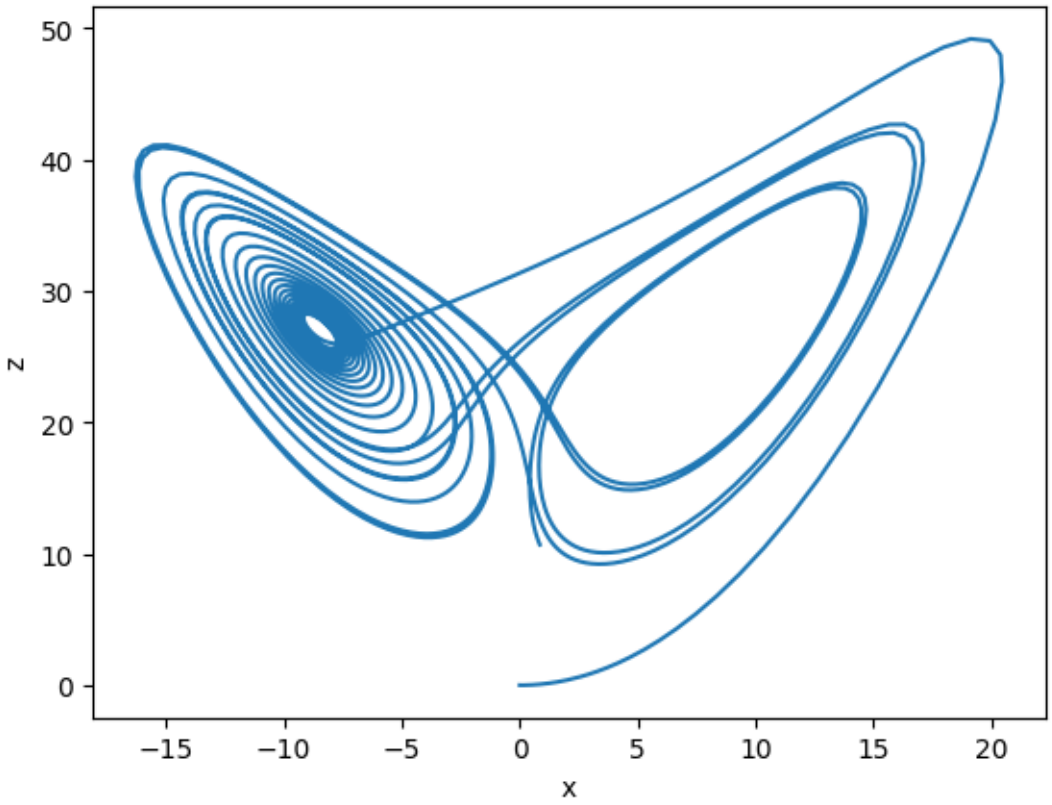
```
x= 0.01
y=0.01
z=0.01

t=0
dt=0.01

vet_t=[]
vet_x=[]
vet_y=[]
vet_z=[]

while(t<=20):
  vet_t.append(t)
  vet_x.append(x)
  vet_y.append(y)
  vet_z.append(z)

  x= firstequation(a,x,y,t,dt)
  y= secondequation (r,x,y,z,t,dt)
  z= thirdequation(b,x,y,z,t,dt)

  t+=dt

#plota os gráficos das funções de Lorenz
plt.plot (vet_t, vet_x)
plt.xlabel('t')
plt.ylabel('x')
plt.show()

plt.plot(vet_y, vet_x)
plt.xlabel('y')
plt.ylabel('x')
plt.show()

plt.plot(vet_t, vet_y)
plt.xlabel('t')
plt.ylabel('y')
plt.show()

plt.plot(vet_x, vet_z)
plt.xlabel('x')
plt.ylabel('z')
plt.show()

plt.plot(vet_t, vet_z)
```

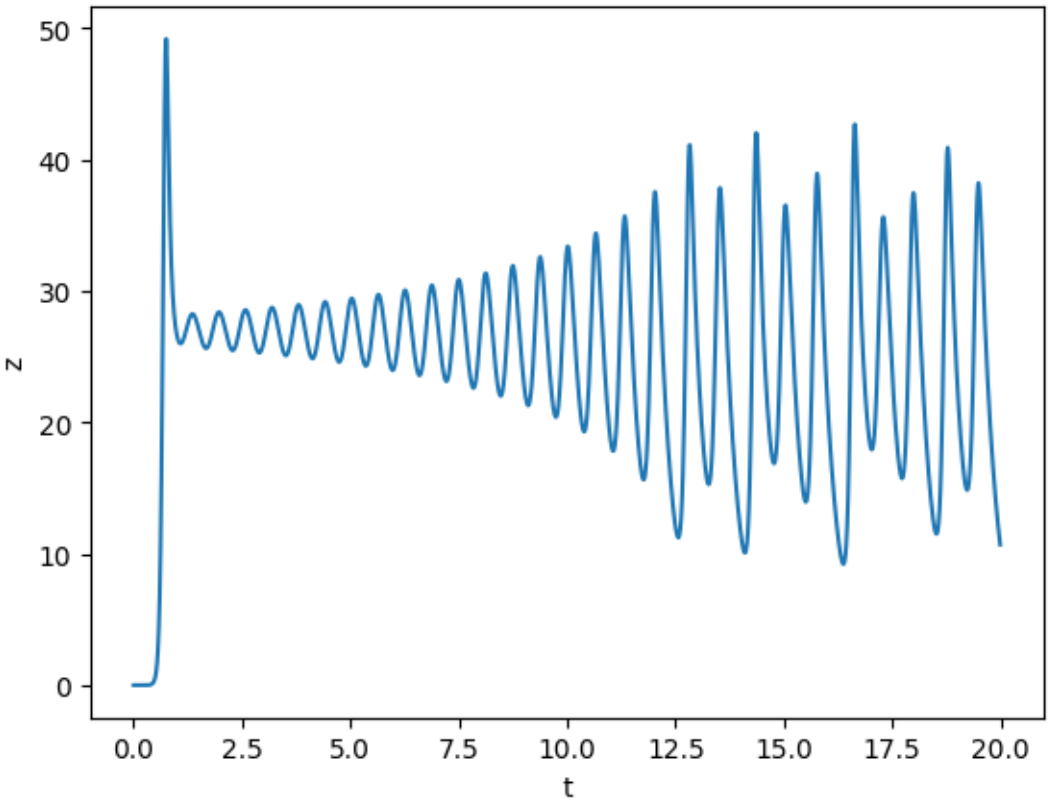
```
  plt.xlabel('t')
  plt.ylabel('z')
  plt.show()

  plt.plot(vet_y, vet_z)
  plt.xlabel('y')
  plt.ylabel('z')
  plt.show()


  # Mapa logistico
  dx = 0.01
  x = dx
  r = 0.2
  dr = 0.005
  v_r = []
  v_x = []

  while r < 4.0:
      # Discarta os valores até as 200 iterações
      for i in range(0, 200):
          x = logisticMap(x, r)
      for i in range(0, 50):
          x = logisticMap(x, r)
          v_r.append(r)
          v_x.append(x)
      r = r + dr
      x = x + dx

  plt.scatter (v_r, v_x, s=0.1)
  plt.xlabel('r')
  plt.ylabel('x')
  plt.show()
main()
```

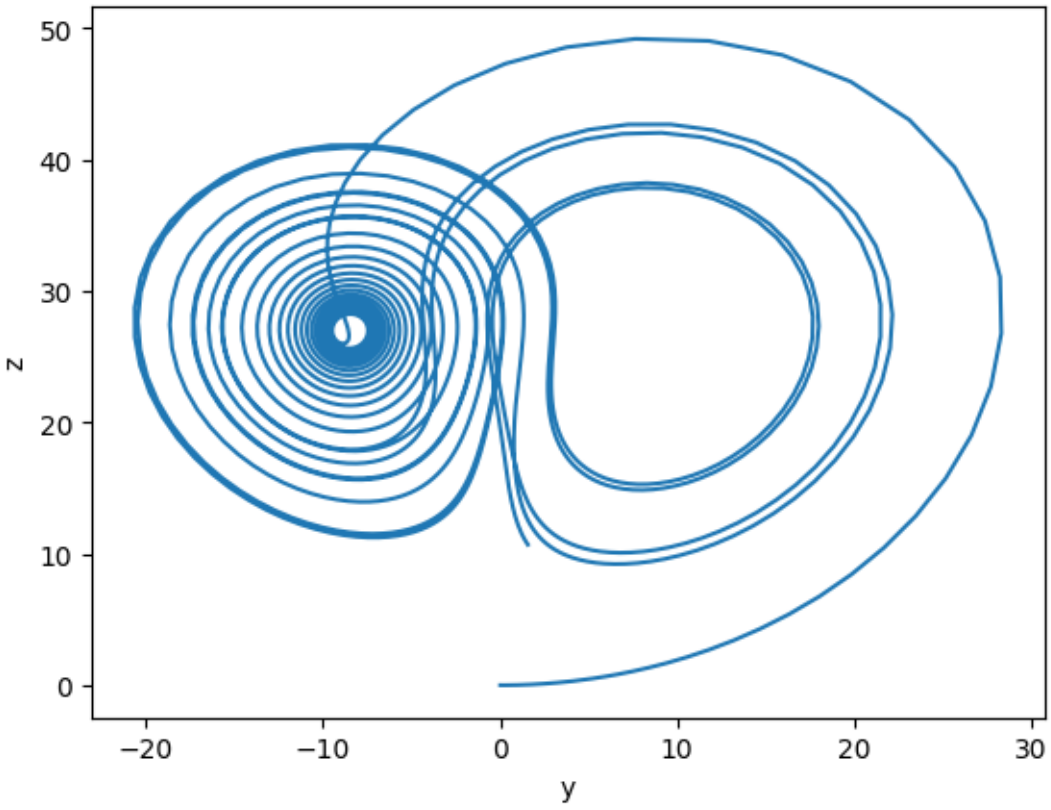

20
15
10
5
0
−5
−10
−15
x
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
t

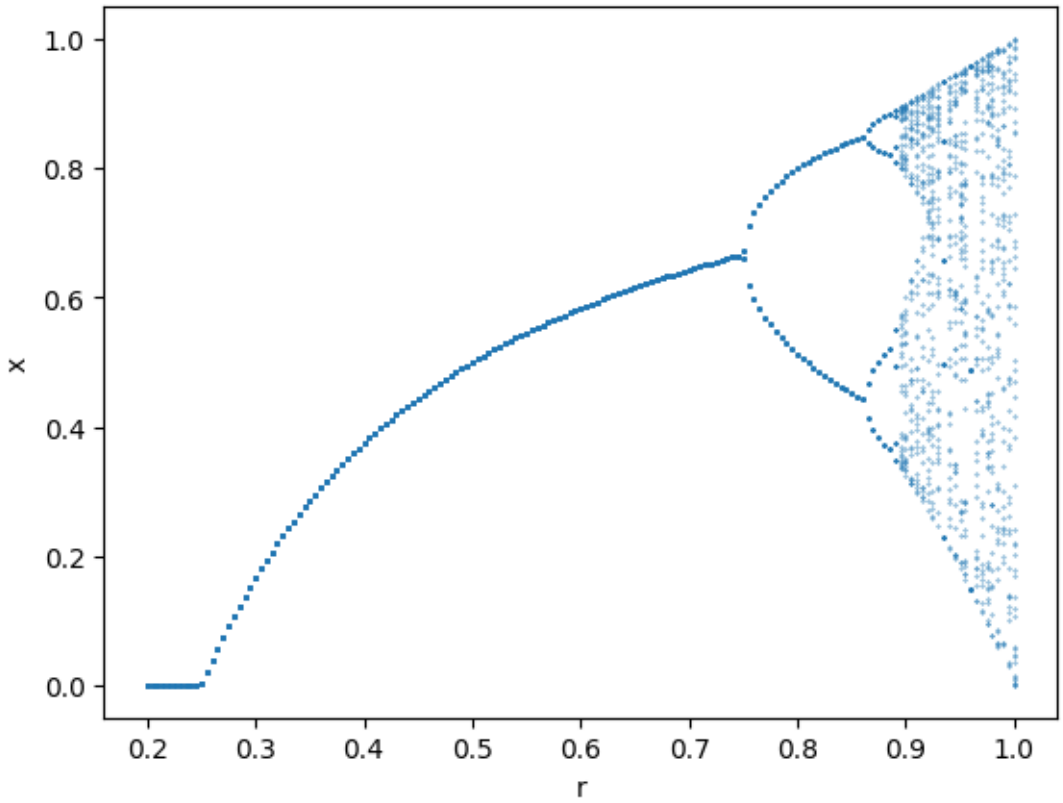

20
15
10
5
0
−5
−10
−15
x
−20
−10
0
10
20
30
y


30
20
10
0
−10
−20
y
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
t


50
40
30
20
10
0
z
−15
−10
−5
0
5
10
15
20
x


50
40
30
20
10
0
z
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
t


50
40
30
20
10
0
z
−20
−10
0
10
20
30
y



[18]:
```
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!pip install pypandoc
```

```
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
pandoc is already the newest version (2.9.2.1-3ubuntu2).
texlive is already the newest version (2021.20220204-1).
texlive-latex-extra is already the newest version (2021.20220204-1).
texlive-xetex is already the newest version (2021.20220204-1).
0 upgraded, 0 newly installed, 0 to remove and 45 not upgraded.
Requirement already satisfied: pypandoc in /usr/local/lib/python3.10/dist-
packages (1.13)
```

[21]:
```
from google.colab import drive
drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[25]:
```
!cp "drive/My Drive/Colab Notebooks/EP2_MAC0209.ipynb" ./
!jupyter nbconvert --to PDF "EP2_MAC0209.ipynb"
!cp "EP2_MAC0209.pdf" "drive/My Drive/Colab Notebooks/"
```

```
[NbConvertApp] Converting notebook EP2_MAC0209.ipynb to PDF
[NbConvertApp] Support files will be in EP2_MAC0209_files/
[NbConvertApp] Making directory ./EP2_MAC0209_files
[NbConvertApp] Making directory ./EP2_MAC0209_files
[NbConvertApp] Making directory ./EP2_MAC0209_files
[NbConvertApp] Making directory ./EP2_MAC0209_files
[NbConvertApp] Making directory ./EP2_MAC0209_files
[NbConvertApp] Making directory ./EP2_MAC0209_files
[NbConvertApp] Making directory ./EP2_MAC0209_files
[NbConvertApp] Making directory ./EP2_MAC0209_files
[NbConvertApp] Making directory ./EP2_MAC0209_files
[NbConvertApp] Making directory ./EP2_MAC0209_files
[NbConvertApp] Making directory ./EP2_MAC0209_files
[NbConvertApp] Making directory ./EP2_MAC0209_files
[NbConvertApp] Making directory ./EP2_MAC0209_files
[NbConvertApp] Making directory ./EP2_MAC0209_files
[NbConvertApp] Writing 54708 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 582434 bytes to EP2_MAC0209.pdf
```