UNIVERSIDADE FEDERAL DE ALAGOAS

Instituto Federal de Alagoas - IC

Engenharia de Computação

Sistemas de Controle 2

Implementação de um controlador em espaço de estados para o jogo “Can U Control?”

Élisson França de Souza

José Renilson Almeida da Silva

Monique Moreira Moraes

Tiago Cabral Carvalho da Silva

Maceió - 2023

ÍNDICE

Resumo

Em controle de espaço de estados, normalmente é trabalhado com equações com várias equações diferenciais com vários parâmetros. Tem-se um desafio muito grande quando precisamos realizar o controle deles quando temos várias entradas e uma apenas saída (MISO), onde uma leve alteração na saída pode mudar todo o sistema. Com tal assunto complexo, o jogo “Can U Control?” tem uma abordagem simplificada de mostrar visualmente como um sistema se comporta quando alteramos a saída do sistema. Métodos de controle complexos ou métodos matemáticos mais simples podem ser usados para resolver tais sistemas de forma rápida, porém com abordagens e erros diferentes. O método de Euler é uma forma de resolver esses sistemas, sendo priorizado para sistemas lineares, porém com algumas mudanças no método calculado, pode ser usado para outros sistemas e obtendo bons resultados, mesmo sendo simples, porém rápido para sistemas em que o erro pode ser tolerado.
Palavras-chaves: sistema de espaço de estados, método de Euler para espaço de estados, método de Euler para jogos.

Abstract

In state-space control, it is usually worked with equations with several differential equations with several parameters. There is a very big challenge when we need to control them when we have several inputs and only one output (MISO), where a slight change in the output can change the entire system. With such a complex subject, the game “Can U Control?” takes a simplified approach to visually showing how a system behaves when we change the output of the system. Complex control methods or simpler mathematical methods can be used to solve such systems quickly, but with different approaches and errors. Euler's method is a way to solve these systems, being prioritized for linear systems, but with some changes in the calculated method, it can be used for other systems and obtaining good results, even being simple, but fast for systems in which the error can be tolerated.
Keywords: state space system, Euler method for state space, Euler method for games.

1. Introdução

O objetivo deste documento é relatar o desempenho obtido no jogo Can U Control nos quatro mapas disponíveis no jogo. Foram implementadas as funções de espaço de estados para a movimentação do personagem no mapa, que é um campo de vetores que direciona o personagem. O conjunto destas duas funções, por sua vez, altera o campo de vetores a fim de que o personagem alcance o alvo, fazendo com que o personagem se movimente tendendo a uma outra trajetória.

O código desenvolvido para este trabalho foi elaborado em linguagem Python, utilizando conceitos matemáticos como o método de Euler para calcular possíveis posições do personagem e jogar na saída com base nesse método, junto com uma busca exaustiva do termo de saída para o controle do personagem.

2. O jogo

“Can U Control” é um jogo com propósito educativo que visa facilitar a compreensão dos conceitos de Espaço de Estados Dinâmico. É baseado em sistemas dinâmicos e a representação do sistema é a de espaço de estados, que nada mais é do que uma equação diferencial de primeira ordem onde as variáveis de estado estão associadas a uma representação vetorial única. O valor de cada um desses vetores em um instante no tempo representa a direção que o estado irá seguir e assim para cada ponto no espaço terá uma direção diferente. O gráfico cartesiano com todos esses vetores é chamado de Campo Vetorial, que é justamente o mapa do jogo. Os elementos presentes no mapa são o personagem, o alvo, caixas de pontuação e marcas de morte. Se o personagem chegar no alvo, a fase é concluída. A movimentação do jogo é feita pela alteração do campo vetorial e não do controle do personagem.

O jogo transfere dados relacionados a vitórias, mapa, nível, posições X e Y do jogador, posições X e Y do alvo, caixas bônus e possíveis caveiras que indicam a zona que o personagem não pode passar. Com esses parâmetros X e Y do personagem, temos que controlar e mover o personagem até um objetivo, podendo ser as caixas bônus ou o objetivo do personagem, controlando apenas um parâmetro de saída que interfere nas duas funções de entrada definidas para cada um dos mapas.

3. Método de Euler

O método de Euler é um método de integração baseado em uma aproximação linear de uma função desconhecida y(x) que tem como finalidade resolver uma equação diferencial ordinária (EDO) de primeira ordem. Ele segue a ideia de que, em um intervalo pequeno o suficiente, a taxa de variação de y é aproximadamente constante. Usado para descrever ou simular processos e sistemas modelados por taxa de variação, que podem possuir até dezenas ou centenas de variáveis. Esse método também é conhecido como a série de Taylor truncada na primeira derivada. É um método de fácil implementação e compreensão, mas não é tão preciso quanto outros métodos de integração, como o Runge Kutta de 4º ordem, por exemplo.

Com o método de Euler, podemos obter relações de equações diferenciais, onde podemos buscar uma certa trajetória ou tendência da trajetória para uma determinada função em um espaço definido de passo (indicado por k).

Para explicar o método de integração de Euler, vamos considerar a seguinte EDO de primeira ordem: $\frac{dy}{dx} = f(x, y)$.

O objetivo é encontrar uma solução aproximada para y em um intervalo de x, dado um valor inicial y_0 em x_0. A ideia básica do método de Euler é aproximar a derivada dy/dx por meio de uma diferença finita e usar essa aproximação para calcular os valores subsequentes de y.

Primeiramente para a utilização deste método é necessário definir o tamanho do passo k, que quanto menor for, melhor será a precisão da integração, mas em contrapartida resultará em mais cálculos. Seguidamente define-se o valor inicial y_0 em x_0, e a partir de x_0 calcula-se o valor do próximo ponto y para cada x_i, segundo a fórmula:

$$y_{i+1} = y_i + k * f(x_i, y_i)$$

$$x_{i+1} = x_i + k$$

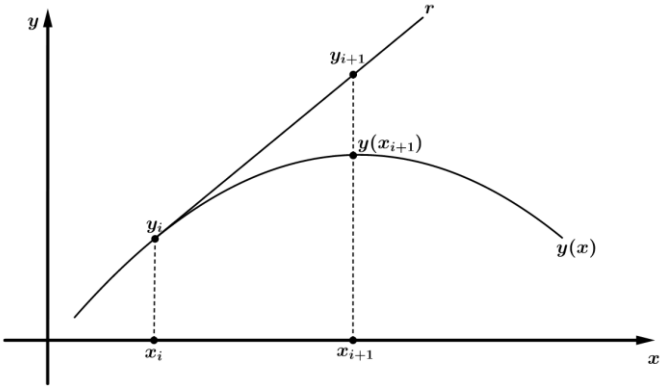

Figura 1: exemplo de função real e aproximação por método de Euler.

onde x_i é valor atual do x, y_i é o valor atual de y e $f(x_i, y_i)$ é o valor da função $f(x, y)$ no ponto (x_i, y_i).

Segue abaixo um exemplo do método de Euler para aproximação de uma função aparentemente senoidal, onde a curva em azul representa a função de fato e a em vermelho representa a aproximação de Euler para essa mesma função.

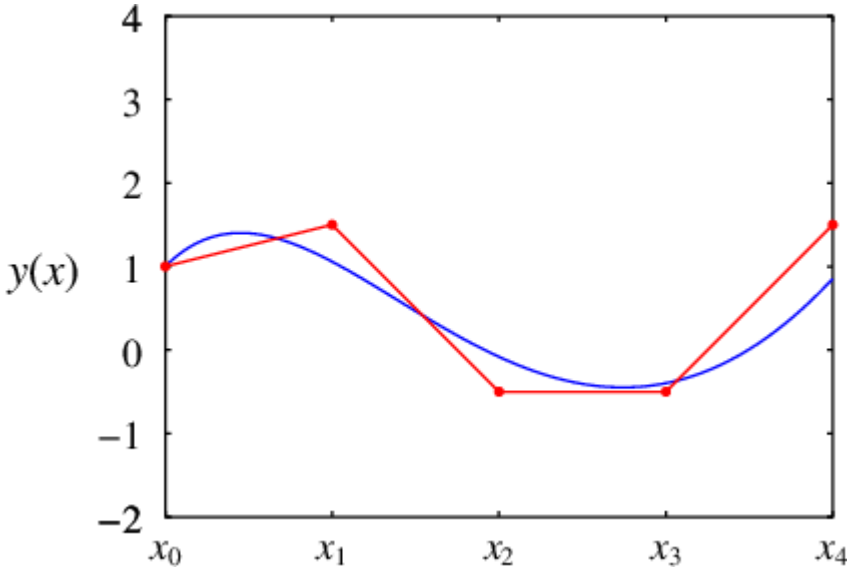

Figura 2.

É importante mencionar que o método de Euler possui algumas limitações. Ele pode introduzir erros acumulativos à medida que avança no intervalo, especialmente se o tamanho do passo k for grande.

4. Código Implementado

O código implementado foi feito usando os conceitos apresentados na descrição do jogo, onde se é usado o conceito do método de Euler, junto com técnicas de distância euclidiana (do alvo para com a posição relativa do personagem, atual e futuras). Foi usado também uma técnica de setar nível de saída, pois há um limite máximo e mínimo (+1 e -1, respectivamente) da saída do código. No jogo, isso remete a uma linha preta criada no local do personagem, indicando em locais prováveis onde o mesmo tenderá a se mover:

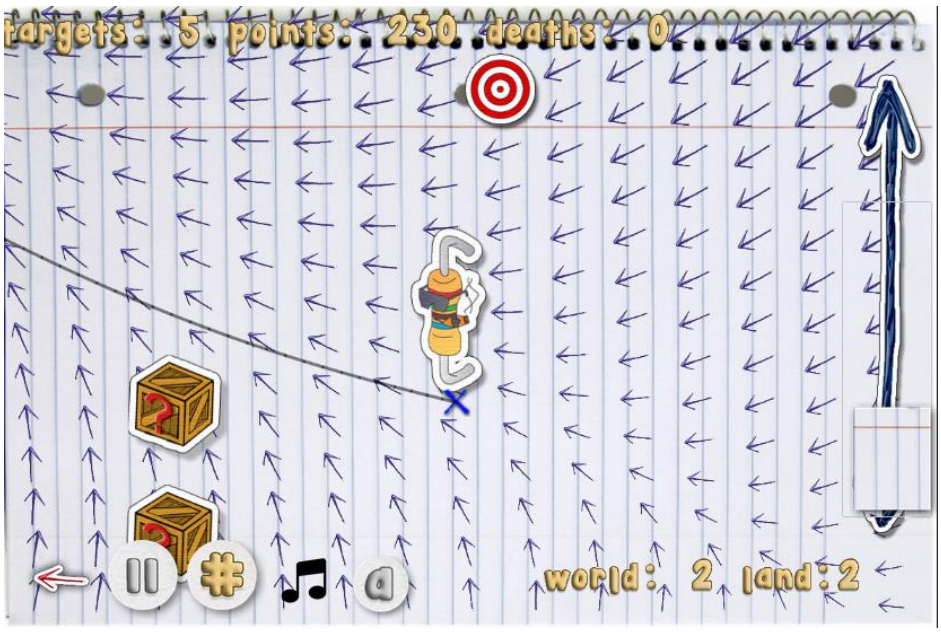

Figura 3: imagem do jogo com tendência de trajetória.

O fluxograma de eventos do código pode ser visto na figura 4.

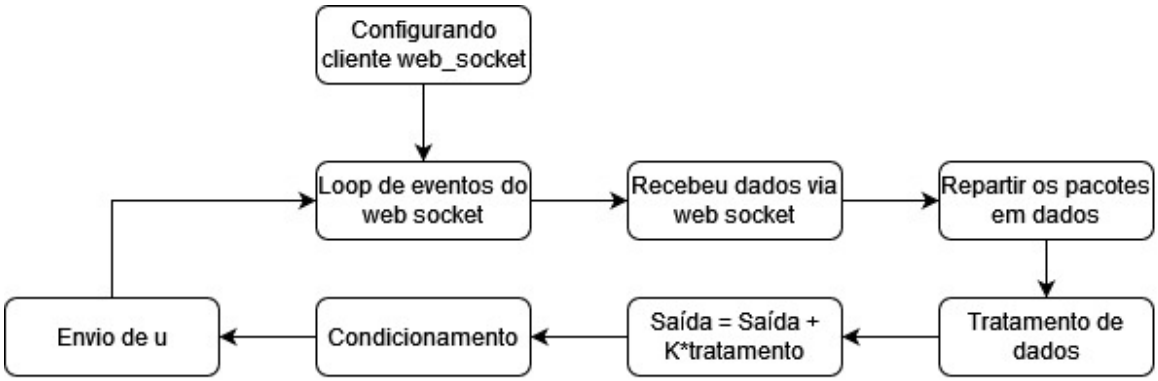

Figura 4: fluxograma do algoritmo desenvolvido.

Ao receber um pacote de dados, é chamado um call-back para evento de mensagem, o código é iniciado e então reparte os dados do pacote em fragmentos que possam ser tratados e analisados, pois eles chegam como strings.

Figura 5: fluxograma de separação de dados.

```
def repartir_mensagens(pacote):
    vitorias = int(pacote[0])
    mapa = int(pacote[1])
    nivel = int(pacote[2])
    jogador = pacote[3], pacote[4]
    alvo = pacote[5], pacote[6]
    caixas, caveiras = [], []
    if(len(pacote) > 8):
        caixas.append((pacote[7], pacote[8]))
    if(len(pacote) > 10):
        caixas.append((pacote[9], pacote[10]))
    if(len(pacote) > 12):
        caixas.append((pacote[11], pacote[12]))
        for i in range(13, len(pacote), 2):
            caveiras.append((pacote[i], pacote[i+1]))
    return vitorias, mapa, nivel, jogador, alvo, caixas, caveiras

def receber_mensagem(cliente, servidor, pacote):
    global saida
    global contador
    global level_anterior
    global passou

    values = [float(x) for x in pacote[:-1].split(",") if x.strip()]
    vitorias, mapa, nivel, jogador, alvo, caixas, caveiras = \
    repartir_mensagens(values)
```

Código 1: código de repartir pacote.

Com o pacote repartido, ainda na função de call-back, temos então as definições do que é realmente possíveis objetivos, tais como alvo e caixas bônus como objetivos, definição de quantos pontos de análise pode ser calculado, e o passo de tempo para calcular os retornos das funções de espaço de estados. O passo e a

definição de quanto pontos analisados interferem muito no sistema, pois se for muitos pontos analisados e retornar uma saída, o sistema mudará após a saída for colocada no jogo, e isso mudará o campo vetorial, onde se o personagem não for rápido, ele não chegará ao seu objetivo, e se ele for pouco, o sistema não conseguirá analisar objetos a uma distância considerada média para o jogo.

Cada mapa tem uma constante (k) que define um termo multiplicador que servirá como o passo na saída, e o passo na análise, será feito por uma variável chamada dt. O código mostrado no código 2, mostra como fazemos isso.

```
if level_anterior != nivel:
   print("Quantidade de fases passadas:", passou)
   passou += 1
   level_anterior = nivel
   saida = 0

k = mapas.define_equacoes(mapa, nivel, 0, 0, 0, False)
CaveirasX = []
CaveirasY = []
ObjetivosX = []
ObjetivosY = []
ObjetivosX.append(alvo[0])
ObjetivosY.append(alvo[1])
quant = 25
dt = 0.01
if(len(caixas) >= 1 and nivel != 3):
   ObjetivosX.append(caixas[0][0])
   ObjetivosY.append(caixas[0][1])
   if(len(caixas) >= 2):
      ObjetivosX.append(caixas[1][0])
      ObjetivosY.append(caixas[1][1])
      if(len(caixas) >= 3):
         ObjetivosX.append(caixas[2][0])
         ObjetivosY.append(caixas[2][1])
saida = saida + (k)*computacaoInicial(jogador[0], \
jogador[1], mapa, nivel, quant, dt, ObjetivosX, ObjetivosY, \
CaveirasX, CaveirasY)
```

Código 2: código de parametrização de dados e a chamada de saída.

Cada fase possui certos parâmetros para serem descobertos, definidos em conjunto com as equações, visando que o personagem não saia do campo de visão, esses parâmetros são definidos com base em observações e testes manuais. Dados dos ganhos de K (obtidos como parâmetro “`False`” na função “`mapas.define_equacoes`”)

para a saída foram feitos em conjunto com as funções de mapa, seus valores podem ser substituídos pelo código 3.

```
#Função equivalente a
#define_equacoes() com parâmetro
#False
def define_const(mapa, nivel):
   const = 1
   if(mapa == 1):
      if(nivel == 1):
         const = 0.25
      elif(nivel == 2):
         const = 0.5
      elif(nivel == 3):
         const = 0.25
      elif(nivel == 4):
         const = 0.25
   elif(mapa == 2):
      const = 0.25
      if(nivel == 3):
         const = 0.2
   elif(mapa == 3):
      const = 0.3
   elif(mapa == 4):
      if(nivel == 1):
         const = 0.8
      elif(nivel == 2):
         const = 0.3
      elif(nivel == 3):
         const = 0.7
      elif(nivel == 4):
         const = 0.6
   return const
```

Código 3: constantes K para a saída, equivalente ao retorno da função "`define_equacoes()`" com último parâmetro "`False`".

```
def maximo(mapa, nivel):
    uMax = 10
    if mapa == 1:
        if nivel == 3:
            uMax = 8
        elif nivel == 4:
            uMax = 8

    if mapa == 2:
        if nivel == 2:
            uMax = 10
        elif nivel == 3:
            uMax = 9
        elif nivel == 4:
            uMax = 10

    if mapa == 3:
        if nivel == 2:
            uMax = 8
        if nivel == 4:
            uMax = 8

    if mapa == 4:
        if nivel == 1:
            uMax = 8
        elif nivel == 3:
            uMax = 8
        elif nivel == 4:
            uMax = 10
    return uMax
```

Código 4: constantes para U em cada analise da computação para evitar prováveis "fugas" do mapa.

Considerando o método de Euler, temos que tratar e checar os próximos passos do personagem por força bruta, onde no algoritmo, estamos fazendo com 25 passos simulados duas vezes, ou seja, podemos checar até 50 passos à frente, checando se aquele resultado é realmente bom. Cada mapa tem um limite de checagem de U a ser considerado para reduzir a chance do personagem a sair do campo de visão do indivíduo, porém permitindo que ele saia rapidamente para retornar para um provável caminho mais curto, como mostra o fragmento de código do bloco

2. O código deve parametrizar e organizar os valores a serem enviados para a computação dos valores e seus possíveis tratamentos, como indicado no código 5.

```
def computacaoInicial(x, y, mapa, nivel, quant, dt, ObjetivosX, \
ObjetivosY, CaveirasX, CaveirasY):
   melhorDist = np.inf
   melhorI = 0
   uMax = maximo(mapa, nivel)
   for i in range(-uMax, uMax + 1):
      valorUI = i / uMax
      distMinima = np.inf
      distancia = np.inf
      x1 = x
      y1 = y
      u = valorUI
      for k in range(quant):
         for q in range(len(ObjetivosX)):
            f = 1 if q == 0 else 2
            distancia = f*((x1 - ObjetivosX[q])*(x1 - ObjetivosX[q]) + \
(y1 - ObjetivosY[q])*(y1 - ObjetivosY[q]))
            if (distancia < distMinima):
                distMinima = distancia
         l12 = mapas.define_equacoes(mapa, nivel, x1, y1, u)
         x1 += l12[0]*dt
         y1 += l12[1]*dt
         if(distMinima < melhorDist):
            melhorDist = distMinima
            melhorI = valorUI
      for j in range(-uMax, uMax + 1):
         valorUJ = j / uMax
         x11 = x1
         y11 = y1
         u = valorUJ
         for k in range(quant):
            for q in range(len(ObjetivosX)):
               f = 1 if q == 0 else 2
               distancia = f*(x11 - ObjetivosX[q])*(x11 - ObjetivosX[q]) + \
(y11 - ObjetivosY[q])*(y11 - ObjetivosY[q])
               if (distancia < distMinima):
                  distMinima = distancia
            l12 = mapas.define_equacoes(mapa, nivel, x11, y11, u)
            x11 += l12[0]*dt
            y11 += l12[1]*dt
```

```
            x11 += l12[0]*dt
            y11 += l12[1]*dt
            if(distMinima < melhorDist):
               melhorDist = distMinima
               melhorI = valorUI
    return melhorI
```

Código 5: código de computação de dados para a distância e melhor parâmetro u.

Vendo o código 5, no momento de salvar a menor distância, é analisada o conjunto de prováveis objetivos, tais como alvo e caixas. Quando estamos checando a distância do personagem para com o alvo, temos que o peso (f) é mínimo e sendo igual a 1, e quando não, o seu peso é 2, priorizando assim o alvo como o melhor caminho a ser considerado. Essa distância é calculada com base na distância euclidiana de um sistema em duas dimensões, porém resumida para evitar gasto de tempo desnecessário para com os cálculos, como descrita abaixo:

$$distância = f * ((x - objx)^2 + (y - objy)^2)$$

Checando as melhores distâncias para o objetivo mais próximo, temos que considerar então o cálculo das funções de espaço de estados em conjunto com as variáveis adotadas, tais como a posição do personagem, sendo eles a origem, e por uma busca exaustiva de U, buscando sempre o caminho com menor distância, não sendo necessário usar a derivada, pois a substituição dos valores prováveis de x, y e u, gera os prováveis pontos onde o personagem poderá passar e sendo atualizadas as novas coordenadas para atualização, sendo multiplicado por uma constante de baixo valor que é o passo do personagem nesse método de análise:

$$x = x + f(x, y) * dt$$

$$y = y + f(x, y) * dt$$

O valor considerado da melhor distância é sempre o valor do loop para a primeira equação por deixar sempre mais próximo do objetivo e mais preciso o movimento.

A figura 5 mostra bem como usar o método de Euler modificado para encaixar aos requisitos do projeto de controlar somente uma saída.

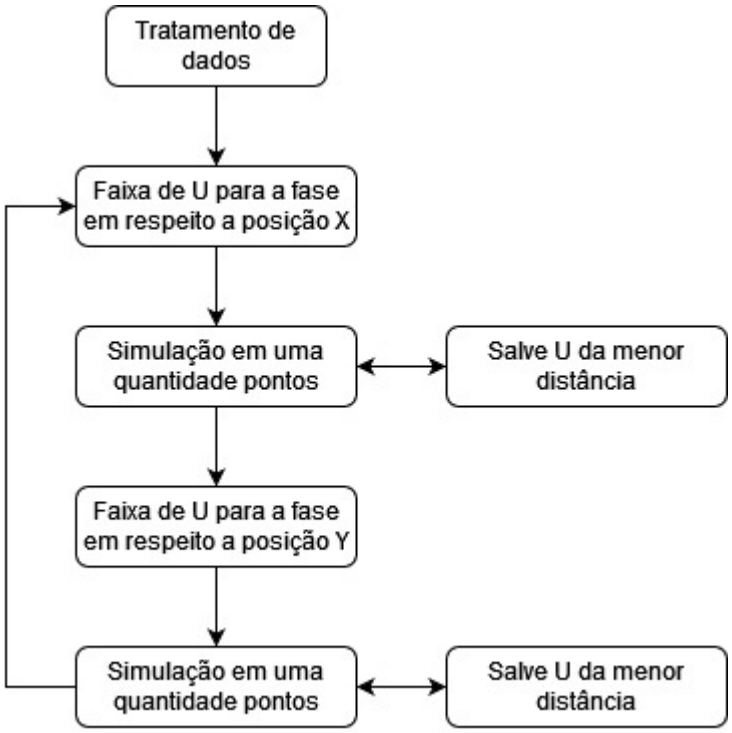

Figura 5: fluxograma de computação das coordenadas.

O código, por fim, condiciona a saída para que não escreva valores indevidos na saída, limitando a saída entre o intervalo de -1 a 1.

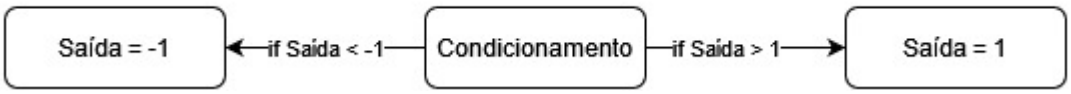

Figura 6: fluxograma do condicionamento da saída (u que será inscrito no jogo).

```
if abs(saida) >= 1:
    saida = (saida/abs(saida))
```

Código 6: código de condicionamento da saída para ficar no intervalo válido sem alterar seu sinal.

Com isso, podemos enviar o valor de u, que controla o personagem, levando para mais próximo do objetivo que ele teve como mais próximo.

5. Conclusão

Concluímos que o algoritmo de Euler junto com outras técnicas é eficiente para controlar a trajetória do personagem no jogo “Can U Control?”, mesmo se perdendo em alguns casos, possibilitando que o algoritmo seja rápido e de certa forma confiável.

A série de Taylor ou o usando o método de Runge Kutta, funcionaram também, expandindo as equações, usando até a ordem 4, porém suas habilidades não foram bem desenvolvidas por deixar o sistema lento e gerar muito mais erros acumulativos.

Por fim, notamos que técnicas simples como Euler, busca exaustiva com passos curtos e limitação da saída, tornaram o algoritmo mais eficiente e funcional na maioria das vezes testadas.

6. Github

https://github.com/RenilsonA/SC2

7. Referências

Método de Euler, Wikipédia. Disponível: <https://en.wikipedia.org/wiki/Euler_method>. Acesso em 10/05/23;

Método de Euler, UFJF. Disponível em: <https://www.fisica.ufjf.br/~sjfsato/fiscomp1/node45.html>. Acesso em 10/05/23;

Distância Euclidiana, Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/Dist%C3% A2ncia_euclidiana>. Acesso em 10/05/23;

Método de Euler Explícito para Resolver EDOs, Artigo de Victoria Tanaka. Disponível em: <https://edisciplinas.usp.br/pluginfile.php/4530296/mod_resource/content/1/modelo3.pdf>. Acesso em 11/05/23;

Método de Euler, Disponível em: <https://www.ufrgs.br/reamat/CalculoNumerico/livro-py/ pdvi-metodo_de_euler.html>. Acesso em 11/05/23;