INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO
CAMPUS SÃO PAULO

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

KETHELYN ALVES

RAFAEL TEIXEIRA

VINÍCIUS TIAGO

Entrega projeto para SPOMVAL - Vetores, Geometria Analítica e Álgebra Linear

SÃO PAULO

2025

Resumo

TEMA 1 – Curvas e Superfícies por Pontos Especificados Use como base:

Você deverá desenvolver um programa que:

1. Receba um conjunto de pontos no plano (2D) ou no espaço (3D).
2. Determine a equação da curva (2D) ou da superfície (3D) que exatamente passa por todos os pontos fornecidos, assumindo um modelo predeterminado (polinomial de grau n para 2D ou quadrática para 3D).
3. Verifique se a matriz de coeficientes é invertível (ou seja, o sistema tem solução única); em caso contrário, aviso de “pontos degenerados”.
4. Gere um gráfico da curva/superfície e mostre os pontos dados.
5. Gere relatório explicando o processo.

O presente projeto consiste no desenvolvimento de uma ferramenta computacional, implementada em Python, destinada à interpolação exata de pontos em espaços bidimensionais (2D) e tridimensionais (3D). O *software* calcula a equação matemática precisa (polinomial para 2D ou quadrática para 3D) que conecta um conjunto de coordenadas fornecidas. Além do cálculo algébrico, realizado mediante a resolução de sistemas lineares, o sistema executa verificações de consistência (invertibilidade da matriz) para a detecção de pontos degenerados, gera visualizações gráficas das curvas ou superfícies resultantes e exporta um relatório técnico que detalha o processo.

Descrição do Projeto

Este repositório armazena uma solução algorítmica para problemas de interpolação em Geometria Analítica e Álgebra Linear Computacional. O programa é capaz de receber um conjunto arbitrário de coordenadas e determinar a função matemática exata que descreve a curva ou superfície que passa por todos os pontos especificados.

O sistema opera diferenciando automaticamente a modelagem baseada na dimensionalidade dos dados de entrada:

- Caso 2D: Ajuste de um polinômio de grau **n** (onde **n** é derivado do número de pontos menos 1).
- Caso 3D: Ajuste de uma superfície baseada em um modelo quadrático geral (cônicas/quádricas).

Funcionalidades

O software foi estruturado para cumprir rigorosamente as seguintes etapas de processamento:

1. Entrada de Dados: Interface para inserção de coordenadas de pontos no plano cartesiano (x, y) ou no espaço euclidiano (x, y, z).
2. Modelagem Matemática: Construção dinâmica das equações baseada nos modelos predeterminados.
3. Validação de Solvabilidade: Verificação da invertibilidade da matriz de coeficientes. O sistema alerta sobre "pontos degenerados" caso o determinante da matriz seja nulo ou a matriz seja singular, indicando que não há solução única para o conjunto de pontos dado.
4. Visualização Gráfica: Geração de gráficos interativos utilizando `matplotlib`, plotando simultaneamente os pontos originais (discretos) e a curva/superfície calculada (contínua).
5. Relatório Técnico: Geração automática de um arquivo de texto explicando o processo, exibindo a matriz montada e a equação final.

Fundamentação Teórica

O núcleo do projeto baseia-se na resolução de um sistema de equações lineares da forma A \cdot x = B, onde A é a matriz construída a partir das coordenadas dos pontos (variáveis independentes) e x é o vetor de coeficientes da equação alvo.

Modelo 2D (Polinomial)

Para n+1 pontos, buscamos um polinômio de grau n:
P(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n

Isso resulta em um sistema linear onde a matriz A assume a forma de uma Matriz de Vandermonde.

Modelo 3D (Superfície Quadrática)

Para pontos no espaço, o modelo geral de segunda ordem utilizado é:
z = Ax^2 + By^2 + Cxy + Dx + Ey + F
Necessitando de um mínimo de 6 pontos para determinar unicamente os 6 coeficientes, assumindo não haver degeneração (como pontos coplanares em configurações específicas).

Pré-requisitos e Instalação

Para executar este projeto, é necessário ter o Python 3 instalado, juntamente com as bibliotecas de computação numérica e visualização.

Clone o repositório:

No bash execute:
“git clone https://github.com/KethelynAlves/-Projeto-Curvas-e-Superficies-por-Pontos-Especificados”

2. Instale as dependências:

No bash execute:
“pip install numpy matplotlib”

Como Usar

Execute o arquivo principal do projeto:

No bash execute:
“python SPOMVAL-MAISCOMPLEXO.py”

O Código

Bibliotecas usadas:

Bibliotecas usadas para auxílio com os gráficos
Matplotlib (pyplot e mplot3d): Responsável pela renderização gráfica.
NumPy: Utilizado para manipulação eficiente de arrays e vetores, além de auxiliar na criação de malhas de coordenadas (meshgrid) e espaços lineares (linspace) para a plotagem.

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
```

Função para a resolução com método Gauss - Determinante

Esta função implementa manualmente o método de Eliminação de Gauss com Pivotamento Parcial.

```python
def gauss_solve(A, b):
    n = len(b)

    import copy
    A = copy.deepcopy(A)
    b = copy.deepcopy(b)

    # Matriz aumentada
    for i in range(n):
        A[i].append(b[i])

    det = 1  # Determinante acumulado

    for i in range(n):

        # Pivotamento parcial
        max_row = i
        for k in range(i + 1, n):
            if abs(A[k][i]) > abs(A[max_row][i]):
                max_row = k

        # Troca de linha muda sinal do determinante
        if max_row != i:
            A[i], A[max_row] = A[max_row], A[i]
            det *= -1

        if abs(A[i][i]) < 1e-12:
            return None, 0

        pivot = A[i][i]
        det *= pivot
```

Função para a resolução com método Gauss - Determinante

```
        # Normaliza linha
        for j in range(i, n + 1):
            A[i][j] /= pivot

        # Zera abaixo
        for k in range(i + 1, n):
            fator = A[k][i]
            for j in range(i, n + 1):
                A[k][j] -= fator * A[i][j]

    # Retrossubstituição
    x = [0] * n
    for i in range(n - 1, -1, -1):
        soma = sum(A[i][j] * x[j] for j in range(i + 1, n))
        x[i] = A[i][n] - soma

    return x, det
```

Método de Potência

Implementa um algoritmo iterativo para encontrar o autovalor dominante (o de maior valor absoluto) da matriz de coeficientes A.

```
def metodo_potencia(A, iteracoes=50):
    import math
    n = len(A)

    v = [1] * n

    for _ in range(iteracoes):
        w = [0] * n
```

Método de Potência

```
        # produto Av
        for i in range(n):
            for j in range(n):
                w[i] += A[i][j] * v[j]

        norma = math.sqrt(sum(w[i] ** 2 for i in range(n)))
        if norma == 0:
            return 0, v

        v = [w[i] / norma for i in range(n)]

    # autovalor
    Av = [sum(A[i][j] * v[j] for j in range(n)) for i in
range(n)]
    lambda_max = sum(v[i] * Av[i] for i in range(n))

    return lambda_max, v
```

Função Gerar relatório

Responsável pela persistência dos dados. Utiliza o gerenciador de contexto with open(...) para criar ou sobrescrever um arquivo de texto (.txt) de forma segura. O relatório compila:

```
def gerar_relatorio(titulo, A, det, coeficientes,
lambda_max):
    with open("relatorio_equacao_da_curva.txt", "w") as arq:
        arq.write("RELATÓRIO DO PROCESSO DE EQUACAO DA
CURVA\n")

arq.write("====================================\n\n")
```

Função Gerar relatório

```
        arq.write(f"Título: {titulo}\n\n")

        arq.write("Matriz do sistema (A):\n")
        for linha in A:
            arq.write(str(linha) + "\n")

        arq.write(f"\nDeterminante da matriz A: {det}\n")
        if det == 0:
            arq.write("A matriz é singular — não existe
solução única.\n\n")

        arq.write("\nCoeficientes encontrados:\n")
        arq.write(str(coeficientes) + "\n\n")

        arq.write("Maior autovalor da matriz A (método da
potência):\n")
        arq.write(str(lambda_max) + "\n\n")

        arq.write("Observações:\n")
        arq.write("• O sistema foi resolvido pelo método de
eliminação de Gauss.\n")
        arq.write("• O determinante foi obtido multiplicando
os pivôs.\n")
        arq.write("• O método da potência estimou o maior
autovalor.\n")
        arq.write("• Aplicação prática em SI: Previsão de
tendências em bancos de dados (2D) e calibração de sensores
biométricos (3D).\n")

    print("\nRelatório gerado:
relatorio_equacao_da_curva.txt\n")
```

Equação da Curva 2D

Gerencia o fluxo para problemas no plano cartesiano:

1. **Entrada:** Solicita n pontos ao usuário.
2. **Modelagem:** Constrói dinamicamente uma **Matriz de Vandermonde**, onde cada linha representa um ponto e as colunas representam as potências de *x (x^0, x^1, …, x^{n-1}).*
3. **Visualização:** Gera um gráfico de dispersão (*scatter*) para os pontos originais e traça uma linha contínua para o polinômio calculado, permitindo a verificação visual do ajuste.
4. **Formatação:** Converte os coeficientes encontrados em uma string formatada (ex: *y = ax^2 + bx + c*) compatível com softwares matemáticos como o Desmos.

```
def equacao_2d():
    print("\n=== EQUAÇÃO DA CURVA 2D ===")
    n = int(input("Quantos pontos? "))
    #vetor que armazenará os pontos seguido pelo loop de
entrada
    pontos = []
    for i in range(n):
        print(f"Ponto {i+1}:")
        x = float(input("x: "))
        y = float(input("y: "))
        pontos.append((x, y))
        print()

    # Monta A e b
    A = []
    b = []
    for (x, y) in pontos:
        linha = [x ** k for k in range(n)]
        A.append(linha)
        b.append(y)
```

Equação da Curva 2D

```
# Resolve sistema com Gauss
    coef, det = gauss_solve(A, b)
    if coef is None:
        print("Sistema sem solução única.")
        return

    print("\nCoeficientes:")
    print(coef)

    equacao_str = "y = "
    for i in range(n-1, -1, -1):
        c = coef[i]
        if c == 0: continue
        sinal = "+" if c >= 0 and i != n-1 else ""
        grau = f"x^{i}" if i > 1 else ("x" if i == 1 else "")
        equacao_str += f" {sinal} {c:.4f}{grau}"
    print("Equação da curva: " + equacao_str)

    # GRÁFICO
    xs = [p[0] for p in pontos]
    ys = [p[1] for p in pontos]

    plot_x = np.linspace(min(xs), max(xs), 500)
    plot_y = [sum(coef[i] * (x ** i) for i in range(n)) for x
in plot_x]

    plt.scatter(xs, ys, color="red")
    plt.plot(plot_x, plot_y)
    plt.title("Equação da Curva 2D")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.grid()
    plt.show()
```

Equação da Curva 2D

```
    lambda_max, _ = metodo_potencia(A)
    gerar_relatorio("Equação da Curva 2D", A, det, coef,
lambda_max)
```

Gráfico

```
# GRÁFICO
xs = [p[0] for p in pontos]
    ys = [p[1] for p in pontos]

    plot_x = np.linspace(min(xs), max(xs), 500)
    plot_y = [sum(coef[i] * (x ** i) for i in range(n)) for x
in plot_x]

    plt.scatter(xs, ys, color="red")
    plt.plot(plot_x, plot_y)
    plt.title("Equação da Curva 2D")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.grid()
    plt.show()

    lambda_max, _ = metodo_potencia(A)
    gerar_relatorio("Equação da Curva 2D", A, det, coef,
lambda_max)
```

Equação da Curva 3D (6 Pontos)

Gerencia o fluxo para problemas no espaço tridimensional:

Equação da Curva 3D (6 Pontos)

1. **Entrada:** Solicita estritamente 6 pontos, quantidade necessária para definir uma quádrica sem ambiguidade (assumindo não degeneração).
2. **Modelagem:** Monta o sistema linear baseada na equação geral de segunda ordem: *z = ax^2 + by^2 + cxy + dx + ey + f*. As colunas da matriz correspondem aos termos *[x^2, y^2, xy, x, y, 1]*.
3. **Visualização:** Utiliza *np.meshgrid* para criar uma malha de coordenadas e *ax.plot_surface* para renderizar a superfície resultante. Os pontos originais são plotados em 3D para comparação.
4. **Formatação:** Gera a equação textual formatada para visualização.

```
def equacao_3d():
    print("\n=== EQUAÇÃO DA CURVA 3D ===")
    print("Serão necessários 6 pontos para determinar a
superfície.")

    pontos = []
    for i in range(6):
        print(f"Ponto {i+1}:")
        x = float(input("x: "))
        y = float(input("y: "))
        z = float(input("z: "))
        pontos.append((x, y, z))
        print()

    # Monta sistema
    A = []
    b = []
    for (x, y, z) in pontos:
        A.append([x*x, y*y, x*y, x, y, 1])
        b.append(z)

    coef, det = gauss_solve(A, b)
    if coef is None:
        print("Sistema degenerado — não existe superfície
única.")
```

Equação da Curva 3D (6 Pontos)

```
        return

    a, b2, c, d, e, f = coef

    print("\nCoeficientes:")
    print(f"a={a:.2f}, b={b2:.2f}, c={c:.2f}, d={d:.2f},
e={e:.2f}, f={f:.2f}")

    # Formata para mascara z = ax^2 + by^2 + ...
    geo_str = f"z = {a:.4f}x^2 + {b2:.4f}y^2 + {c:.4f}xy +
{d:.4f}x + {e:.4f}y + {f:.4f}"
    # Remover sinais de +-
    print(f"\nCopie para o GeoGebra: {geo_str}")
```

Gráfico 3D

```
# GRÁFICO 3D
    fig = plt.figure()
    ax = fig.add_subplot(111, projection="3d")

    xs = np.linspace(-2, 2, 50)
    ys = np.linspace(-2, 2, 50)
    X, Y = np.meshgrid(xs, ys)
    Z = a*X*X + b2*Y*Y + c*X*Y + d*X + e*Y + f

    ax.plot_surface(X, Y, Z, alpha=0.5)
    ax.scatter([p[0] for p in pontos],
               [p[1] for p in pontos],
               [p[2] for p in pontos],
               color="red")
```

Equação da Curva 3D (6 Pontos)

```
    ax.set_title("Superfície Quadrática")
    plt.show()

    lambda_max, _ = metodo_potencia(A)
    gerar_relatorio("Equação da Curva 3D", A, det, coef,
lambda_max)
```

Menu Principal

Um loop infinito (*`while True`*) que fornece a interface de usuário (CLI), permitindo selecionar entre as operações 2D, 3D ou encerrar o programa, garantindo que a aplicação continue rodando até ordem expressa do usuário.

```
while True:
    print("\n=== MENU ===")
    print("1 - Equação da Curva 2D")
    print("2 - Equação da Curva 3D")
    print("0 - Sair")

    op = input("Escolha: ")

    if op == "1":
        equacao_2d()
    elif op == "2":
        equacao_3d()
    elif op == "0":
        break
    else:
        print("Opção inválida!")
```

Funcionamento do Código

Siga as instruções no terminal para escolher a dimensão (2 ou 3) e inserir a quantidade e as coordenadas dos pontos.

Exemplo de Entrada (2D)

Para gerar uma parábola simples y = x^2:

1. Ponto 1: (-1, 1)
2. Ponto 2: (0, 0)
3. Ponto 3: (1, 1)

Exemplo de Entrada (3D)

Para gerar um paraboloide circular z = x^2 + y^2:

1. Pontos: (0,0,0),
2. (1,0,1),
3. (-1,0,1),
4. (0,1,1),
5. (0,-1,1),
6. (1,1,2)

Tecnologias Utilizadas

- Python 3: Linguagem base.
- NumPy: Para manipulação de arrays multidimensionais e resolução de sistemas lineares (`numpy.linalg.solve`).
- Matplotlib: Para plotagem de gráficos 2D e renderização de superfícies 3D (`mpl_toolkits.mplot3d`).

Referências externas

Algebra Linear e Aplicações. Disponível em: <https://www.ime.unicamp.br/~marcia/AlgebraLinear/aplicacao_curvas_superficies.html>. Acesso em: 29 nov. 2025.