UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIA E TECNOLOGIA – CCT
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
DCC703 - COMPUTAÇÃO GRÁFICA
PROF. LUCIANO FERREIRA SILVA

**Aluna: Wandressa Reis**

**Matrícula: 2020014698**

# RELATÓRIO DESENHOS DA CURVA DE BÉZIER ATRAVÉS DOS ALGORITMOS EQUAÇÃO PARAMÉTRICA E CASTELJAU

## RESUMO

O relatório apresenta a implementação dos algoritmos da Equação Paramétrica e de Casteljau para o desenho da Curva de Bézier, além da análise dos resultados obtidos, incluindo a comparação entre as curvas geradas por cada método.

## 1. INTRODUÇÃO

A Curva de Bézier é um conceito matemático desenvolvido pelo engenheiro francês Pierre Bézier, que a utilizou no design de automóveis enquanto trabalhava na Renault. Sua formulação foi baseada no algoritmo criado em 1957 por Paul Casteljau, engenheiro da Citroën.

Atualmente, as curvas de Bézier são amplamente utilizadas em computação gráfica para a geração de curvas suaves, animações, modelagem 3D e diversas aplicações em design digital.

O princípio por trás dessas curvas envolve a definição de um conjunto de pontos de controle. A curva inicia em um ponto e termina em outro, sendo influenciada pelos pontos de controle, que determinam sua forma ao "atraí-la" ou "afastá-la" em determinadas direções.

## 2. ALGORITMOS

Neste tópico, abordaremos os métodos implementados para o desenho da Curva de Bézier, destacando suas especificidades e explicando os detalhes de sua implementação.

### 2.1. Equação Paramétrica

A equação paramétrica da Curva de Bézier é baseada na combinação linear dos pontos de controle, ponderados pelo polinômio de Bernstein. Esse método permite a geração suave da curva a partir dos pontos de controle, garantindo transições suaves entre os segmentos.

**Polinômio de Bernstein:**

$$B_{i,\,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

O código apresentado implementa a equação paramétrica da Curva de Bézier em Python. Os principais componentes são:

- Função binomio_newton(n, i): Calcula o coeficiente binomial necessário para os polinômios de Bernstein.
- Função polinomio_bernstein(n, i, t): Define os polinômios de Bernstein, que são usados para calcular os pontos intermediários da curva.
- Função curva_bezier(pontos): Implementa a equação paramétrica, iterando valores de ttt entre 0 e 1 para calcular os pontos da curva.
- Função main(): Define um conjunto de pontos de controle, chama a função curva_bezier() e utiliza Matplotlib para exibir a curva junto com os pontos de controle.

O código gera uma curva suave que passa próxima aos pontos de controle, mas sem necessariamente tocá-los (exceto no primeiro e último ponto).

```
from math import factorial as fac
import matplotlib.pyplot as plt


def binomio_newton(n, i): # Definindo a
    return fac(n) / (fac(i) * fac(n - i))
def polinomio_bernstein(n, i, t):
    return binomio_newton(n, i) * (t**i) * ((1 - t)**(n - i))
def curva_bezier(pontos):
    n = len(pontos)
    curva = []
    for c in range(0, 100 + 1):
        t = c / 100  # definindo t no range (0, 1)
         x_soma = sum(polinomio_bernstein(n, i, t) * pontos[i][0] for i
in range(n + 1))
         y_soma = sum(polinomio_bernstein(n, i, t) * pontos[i][1] for i
in range(n + 1))
        curva.append((x_soma, y_soma))
    return curva
```

```
def main():
    pontos = [(0, 0), (1, 2), (3, 3), (4, 0)]
    curva = curva_bezier(pontos)
    curva_x, curva_y = zip(*curva)
       plt.plot(curva_x, curva_y, 'b-', label='Curva de Bézier - Eq
Paramétrica')
    pontos_x, pontos_y = zip(*pontos)
    plt.plot(pontos_x, pontos_y, 'ro--', label='Pontos de Controle')
    plt.legend()
    plt.grid(True)
    plt.show()
if __name__ == "__main__":
    main()
```

A execução do código gera um gráfico que ilustra a Curva de Bézier construída a partir dos pontos de controle fornecidos. Os principais resultados observados são:

- A curva se aproxima dos pontos de controle, mas não necessariamente passa por eles, exceto pelo primeiro e último.
- A suavidade da curva depende da distribuição dos pontos de controle e da quantidade de amostras de ttt utilizadas (100 pontos no código).
- O método baseado na equação paramétrica gera curvas suaves e bem definidas, sendo adequado para aplicações em gráficos vetoriais, animações e modelagem 3D.

Abaixo são apresentados alguns resultados obtidos para curvas lineares, quadráticas e cúbicas.

**Figura 1 - Curva Linear Eq. Paramétrica**

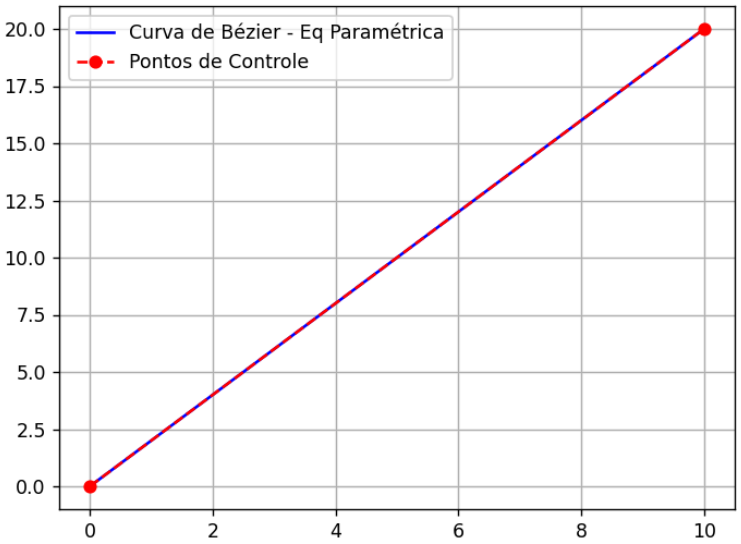



**Figura 2 - Curva Quadrática Eq. Paramétrica**

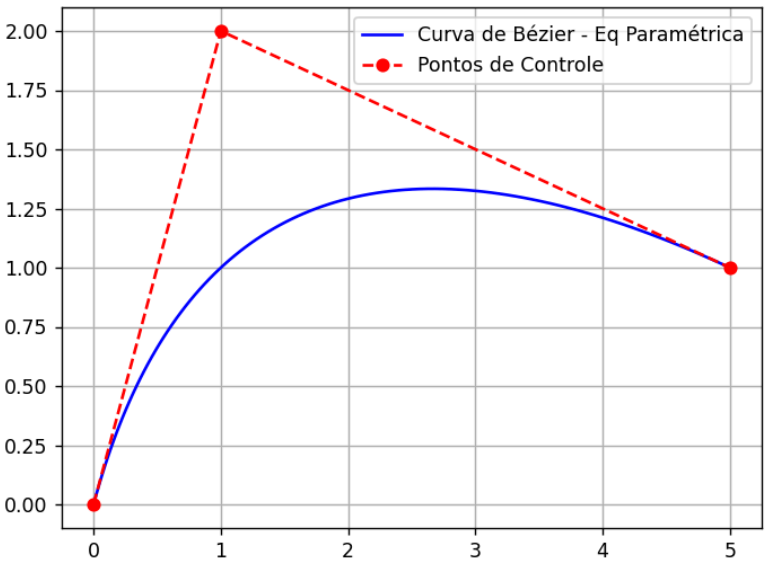
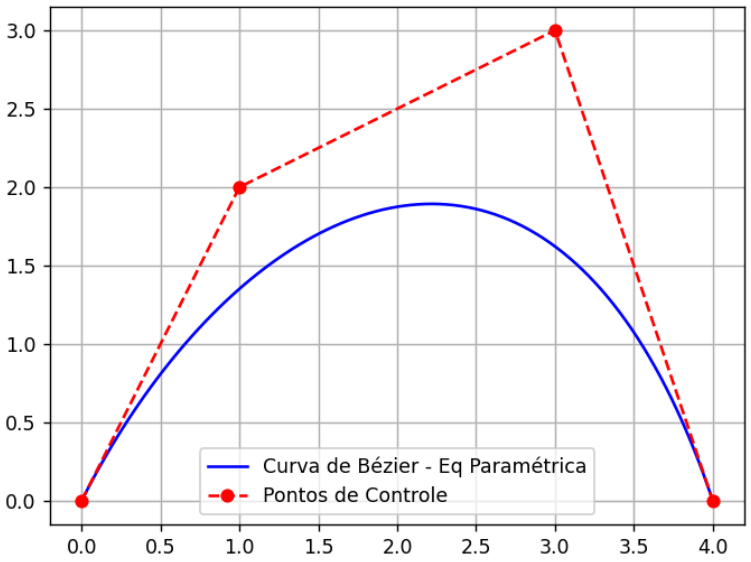


**Figura 3 - Curva Cúbica Eq. Paramétrica**




## 2.2. Casteljau

O algoritmo de De Casteljau é um método recursivo utilizado para avaliar e subdividir curvas de Bézier. Ele permite calcular pontos intermediários da curva com alta precisão, além de ser numericamente mais estável do que a equação paramétrica baseada nos polinômios de Bernstein.

**Código python implementado:**

```
import numpy as np
import matplotlib.pyplot as plt


def de_casteljau(P0, P1, P2, P3, t):


    # Algoritmo de De Casteljau para subdividir a curva de Bézier recursivamente
    if t > 0.05:
        e = t / 2
        P0N1, P1N1, P2N1, P3N1, P0N2, P1N2, P2N2, P3N2 = ponto_medio_curva(P0, P1, P2, P3)
        de_casteljau(P0N1, P1N1, P2N1, P3N1, e)
```

```
        de_casteljau(P0N2, P1N2, P2N2, P3N2, e)
    else:
        plt.plot(*zip(*[P0, P1, P2, P3]), 'ro--')

def ponto_medio_curva(P0, P1, P2, P3):
    # Calcula os pontos médios para subdivisão da curva
    M01 = (np.array(P0) + np.array(P1)) / 2
    M12 = (np.array(P1) + np.array(P2)) / 2
    M23 = (np.array(P2) + np.array(P3)) / 2

    M012 = (M01 + M12) / 2
    M123 = (M12 + M23) / 2

    M0123 = (M012 + M123) / 2

    plt.plot(M0123[0], M0123[1], 'b-')  # Desenhar pontos da curva

    return P0, M01, M012, M0123, M0123, M123, M23, P3

def plot_bezier():
    # Função para configurar a curva Bézier e iniciar a subdivisão
    P0, P1, P2, P3 = (0, 0), (1, 2), (3, 3), (4, 0)

    plt.figure(figsize=(6, 6))
    plt.plot(*zip(*[P0, P1, P2, P3]), 'go--', label='Pontos de
Controle')  # Desenha os pontos de controle
    de_casteljau(P0, P1, P2, P3, 1)
```

```
    plt.legend()
    plt.show()


if __name__ == "__main__":
    plot_bezier()
```

**Função de_casteljau(P0, P1, P2, P3, t)**

- Essa função executa o algoritmo de Casteljau recursivamente para subdividir a curva.
- Se o parâmetro t for maior que 0.05, o código divide a curva ao meio e chama a função novamente para cada metade.
- Caso contrário, a função apenas plota os pontos de controle conectados por segmentos.

**Função ponto_medio_curva(P0, P1, P2, P3)**

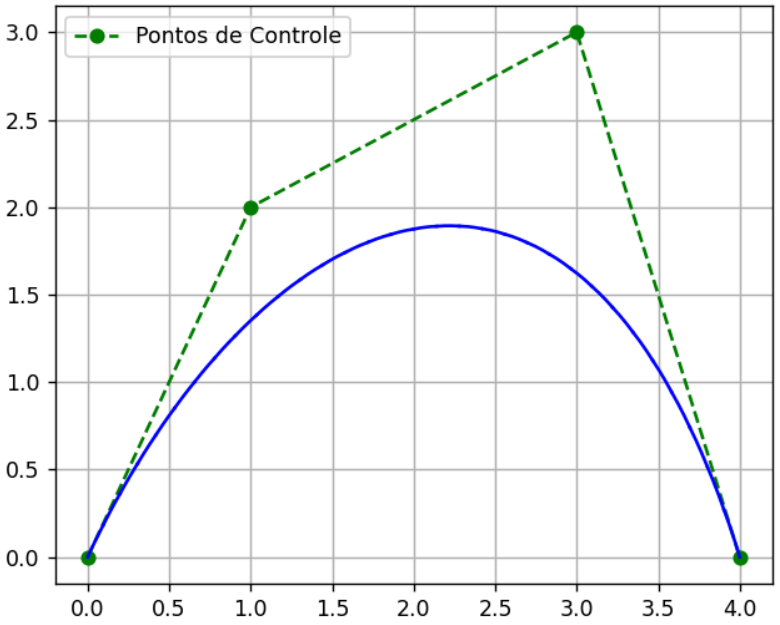
- Calcula os pontos médios entre os pontos de controle para gerar novos segmentos.
- Plota um ponto intermediário da curva (M0123), que é calculado pela interpolação sucessiva dos pontos médios.

**Função plot_bezier()**

- Define um conjunto de pontos de controle (P0, P1, P2, P3).
- Inicializa o gráfico e chama de_casteljau() para desenhar a curva por subdivisão.

Abaixo é apresentado um dos resultados gráficos obtidos.

**Figura 4 - Curva Cúbica Casteljau**




## 3. RESULTADOS

A implementação dos dois algoritmos permitiu a geração de curvas de Bézier precisas, conforme ilustrado nas figuras. Ambos os métodos produziram curvas idênticas para o mesmo conjunto de pontos de controle. No entanto, diferenças podem ser consideradas em relação à eficiência computacional e à abordagem de construção:

**Equação Paramétrica**:

- **Precisão**: A curva é gerada diretamente pela avaliação contínua do polinômio de Bernstein para valores de $t$ entre 0 e 1, resultando em uma curva suave com 100 pontos amostrados.
- **Complexidade**: Para $n$ pontos de controle, cada ponto da curva requer O(n) operações devido ao cálculo dos coeficientes de Bernstein. Isso pode se tornar computacionalmente custoso para curvas com muitos pontos de controle.
- **Resultado Visual**: A curva não passa pelos pontos intermediários de controle, mas sua forma é fortemente influenciada por eles (Figuras 1-3).

**Algoritmo de Casteljau**:

- **Abordagem Recursiva**: A subdivisão sucessiva da curva até um limite ($t \leq 0.05$) gera uma aproximação linear segmentada da curva. Embora menos direto, esse método é numericamente estável e evita instabilidades associadas a polinômios de alto grau.

- **Eficiência**: Cada subdivisão requer operações de interpolação linear O(n2) para *n* pontos), mas é mais eficiente para ajustes locais e subdivisões parciais.

## 4. CONCLUSÃO

Este trabalho demonstrou a implementação prática de dois algoritmos fundamentais para a geração de curvas de Bézier: a equação paramétrica (baseada em polinômios de Bernstein) e o algoritmo de Casteljau. Ambos produzem curvas idênticas, porém com trade-offs distintos.

A escolha entre os métodos depende do contexto de aplicação. Para cenários que demandam suavidade e precisão absoluta, a equação paramétrica é ideal. Já em casos que envolvem ajustes frequentes ou curvas de alto grau, o algoritmo de Casteljau oferece vantagens.