UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
INSTITUTO TÉRCIO PACITTI DE APLICAÇÕES E PESQUISAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

RODRIGO LOPES RANGEL MADUREIRA

ALGORITMOS DE INTERSEÇÕES DE CURVAS DE BÉZIER COM UMA APLICAÇÃO À LOCALIZAÇÃO DE RAÍZES DE EQUAÇÕES

Rio de Janeiro
2013

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
INSTITUTO TÉRCIO PACITTI DE APLICAÇÕES E PESQUISAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

RODRIGO LOPES RANGEL MADUREIRA

ALGORITMOS DE INTERSEÇÕES DE CURVAS DE BÉZIER COM UMA APLICAÇÃO À LOCALIZAÇÃO DE RAÍZES DE EQUAÇÕES

Dissertação de Mestrado submetida ao Corpo Docente do Departamento de Ciência da Computação do Instituto de Matemática, e Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários para obtenção do título de Mestre em Informática.

Orientador: Mauro Antônio Rincon

Co-orientador: Luis Menasché Schechter

Rio de Janeiro
2013

M183 Madureira, Rodrigo Lopes Rangel

Algoritmos de interseções de curvas de Bézier com uma aplicação à localização de raízes de equações / Rodrigo Lopes Rangel Madureira. – 2013.
125 f.: il.

Dissertação (Mestrado em Informática) – Universidade Federal do Rio de Janeiro, Instituto de Matemática, Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Programa de Pós-Graduação em Informática, Rio de Janeiro, 2013.

Orientador: Mauro Antônio Rincon.
Co-orientador: Luis Menasché Schechter.

1. Curvas de Bézier. 2. Métodos Numéricos. 3. Raízes de Funções Reais. 4. Computação Gráfica. – Teses. I. Rincon, Mauro Antônio (Orient.). II. Schechter, Luis Menasché (Co-orient.). III. Universidade Federal do Rio de Janeiro, Instituto de Matemática, Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Programa de Pós-Graduação em Informática. IV. Título

CDD

RODRIGO LOPES RANGEL MADUREIRA

Algoritmos de interseções de curvas de Bézier com uma aplicação à localização de raízes de equações

Dissertação de Mestrado submetida ao Corpo Docente do Departamento de Ciência da Computação do Instituto de Matemática, e Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários para obtenção do título de Mestre em Informática.

Aprovado em: Rio de Janeiro, _____ de ______________ de _________.

____

Prof. Dr. Mauro Antônio Rincon (Orientador)

____

Prof. Dr. Luis Menasché Schechter (Co-orientador)

____

Prof. Dr. Rodrigo Penteado Ribeiro de Toledo

____

Profa. Dra. Juliana Vianna Valério

____

Prof. Dr. Severino Collier Coutinho

Rio de Janeiro
2013

Aos meus pais e irmãos...

AGRADECIMENTOS

Tenho muito que agradecer a Deus, em primeiro lugar, por ter me ajudado a superar diversos obstáculos para conseguir chegar à conclusão desta dissertação.

Aos meus pais, Diva e Luciano, que me ajudaram nos momentos mais difíceis destes dois anos de dedicação e empenho neste trabalho. Espero que eles se sintam bastante orgulhosos por isso.

Aos meus orientadores, Mauro Rincon e Luis Menasché, que acrescentaram informações bastante valiosas que ajudaram a enriquecer ainda mais o meu trabalho, tornando-o mais consistente.

Aos colegas que fiz no mestrado ao longo de todos os períodos, com quem troquei experiências e informações muito valiosas para minha aprendizagem em todas as disciplinas da grade curricular.

Aos colegas de trabalho do Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, por terem me permitido conciliar com tranquilidade as aulas das disciplinas do meu mestrado com meu expediente.

À coordenação, à secretaria e ao corpo docente do Programa de Pós-Graduação em Informática (PPGI), especialmente aos professores da área de Algoritmos e Métodos Numéricos, mestres que me ensinaram a ser um eterno aprendiz.

RESUMO

Madureira, Rodrigo Lopes Rangel. **Algoritmos de interseções de curvas de Bézier com uma aplicação à localização de raízes de equações**. 2013. 112 f. Dissertação (Mestrado em Informática) - PPGI, Instituto de Matemática, Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2013.

As curvas de Bézier receberam essa denominação a partir do trabalho do engenheiro francês Pierre Bézier Etienne (1919-1999) com sistemas CADCAM na Renault, no início dos anos 60. Um pouco antes, Paul De Casteljau, um físico e matemático da Citröen, já havia desenvolvido métodos matemáticos equivalentes para essas curvas, muito importantes na área de Computação Gráfica, já que muitos softwares disponíveis no mercado utilizam este conceito. Os principais algoritmos para localizar interseções entre duas curvas de Bézier encontrados na literatura são: Subdivisão Recursiva de De Casteljau, Subdivisão Intervalar adotado por Koparkar e Mudur e Bézier Clipping, sendo este último bastante utilizado para Ray Tracing, um algoritmo usado em Computação Gráfica para geração de imagens. A proposta deste trabalho trata do estudo e análise comparativa dos três algoritmos para o cálculo de interseções entre duas curvas de Bézier, seguidos de um estudo de uma aplicação destes algoritmos para o cálculo de raízes reais de funções (não necessariamente polinomiais).

Palavras-chave: Curvas de Bézier, Métodos Numéricos, Raízes de Funções Reais, Computação Gráfica.

ABSTRACT

Madureira, Rodrigo Lopes Rangel. **Algoritmos de interseções de curvas de Bézier com uma aplicação à localização de raízes de equações**. 2013. 112 f. Dissertação (Mestrado em Informática) - PPGI, Instituto de Matemática, Instituto Tércio Pacitti, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2013.

Bézier curves received this name from the work of the French engineer Pierre Bézier Etienne (1919-1999) with CADCAM systems in Renault, at the beginning of 1960s. Earlier, Paul De Casteljau, a physicist and mathematician at Citroën, had already developed mathematical methods equivalent for these curves, which are very important in the area of Computer Graphics, since many of the available softwares utilize this concept. The main algorithms for finding intersections between two Bézier curves found in the literature are: De Casteljau's Recursive Subdivision, Interval Subdivision Method adopted by Koparkar and Mudur, and Bézier Clipping, which is used for Ray Tracing, an algorithm used in Computer Graphics to generate images. The proposal of this research is to deal with the study and comparative analysis of the three algorithms for the computation of intersections between two Bézier curves, followed by a study of one application of these algorithms to the calculation of real roots of functions (not necessarily polynomial).

Keywords: Bézier Curves, Numerical Methods, Roots of Real Functions, Computer Graphics.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

As curvas de Bézier receberam essa denominação a partir do trabalho do engenheiro francês Pierre Bézier Etienne (1919-1999) com sistemas CADCAM na Renault, no início dos anos 60, para o design de automóveis. Elas foram formalizadas como resultado do trabalho de Paul De Casteljau, um físico e matemático da Citröen que já havia desenvolvido no final da década de 50 métodos matemáticos equivalentes para geração dos pontos que compõem as curvas. (BUSS, 2003)

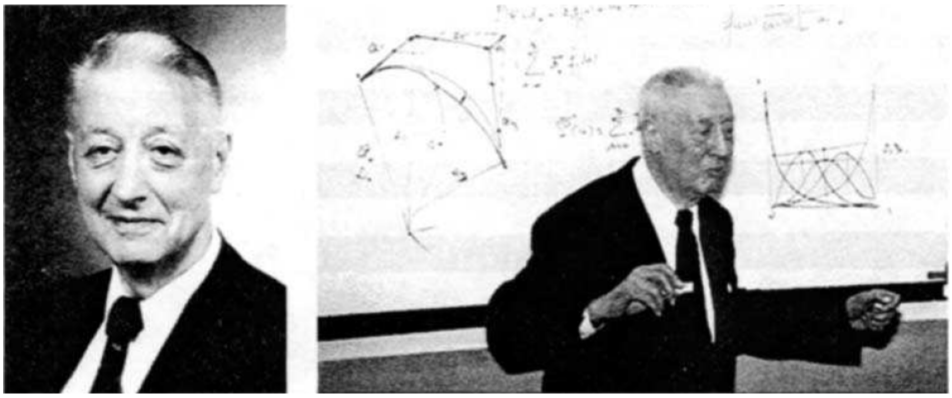

Figura 1.1: Pierre Bézier Etienne (ROGERS, 2001)

Hoje, transcorridas cinco décadas, pode-se afirmar que existem diversas aplicações para as curvas de Bézier, especialmente na área de Computação Gráfica, onde são trabalhadas nos campos de modelagem tridimensional e animações. São utilizadas em diversas aplicações gráficas disponíveis no mercado e representam uma ferramenta indispensável na era digital por permitir desenhos de contornos mais suaves de objetos, além de sua construção matemática favorecer inexoravelmente a sua manipulação através dos computadores.

A proposta deste trabalho é apresentar outra aplicação interessante das curvas de Bézier: localizar interseções de curvas algébricas planas, que são representadas através de funções de duas variáveis. Para isto, aproveita-se uma característica marcante das curvas de Bézier, que é a sua capacidade de se aproximar de outros tipos de curvas num dado intervalo. Assim, com a devida manipulação dos pontos de controle que vão definir a forma geral das curvas, é possível criar novas curvas que se aproximem de cônicas como parábolas, hipérboles, entre outras.

Os métodos mais tradicionais encontrados na literatura para localizar interseções de curvas algébricas planas são o cálculo analítico através da definição de resultante das curvas e o método numérico de Newton para sistemas não lineares. O cálculo da resultante será importante para o Teorema de Bézout, que utiliza este resultado para definir o número máximo de interseções de duas curvas em função de seus respectivos graus. Ao longo deste trabalho, serão discutidas as vantagens e desvantagens desses métodos tradicionais em comparação com os algoritmos que utilizam curvas de Bézier, os quais representam o escopo deste trabalho.

Os principais algoritmos para localizar interseções entre duas curvas de Bézier encontrados na literatura são: Subdivisão recursiva de De Casteljau, Subdivisão intervalar adotado por Koparkar e Mudur (SEDERBERG ; PARRY, 1986) e Bézier Clipping (NISHITA ; SEDERBERG, 1990). Estes algoritmos podem ser aplicados para o cálculo numérico das interseções de curvas algébricas planas e cada um possui sua particularidade e ordem de convergência.

Neste trabalho, será feito um estudo e uma análise comparativa dos três algoritmos para o cálculo de interseções entre duas curvas algébricas planas, aproximadas por curvas de Bézier.

1.1 Motivações

A principal motivação vem do trabalho desenvolvido no Projeto Final de Graduação em Ciência da Computação na Universidade Federal do Rio de Janeiro(UFRJ), onde foi mostrado um método de localização de raízes reais de polinômios de uma variável usando curvas de Bézier. Este método, baseado na subdivisão recursiva de De Casteljau e na forma paramétrica de Bernstein-Bézier para polinômios, trata de um problema equivalente a encontrar a interseção de uma curva de Bézier com uma reta, que no caso era a reta horizontal da origem das coordenadas (x, y) no $\mathbb{R}^2$. Agora, a ideia é generalizar este problema para encontrar interseções entre duas curvas de Bézier quaisquer, usando os algoritmos descritos anteriormente.

Outro fator que motiva o desenvolvimento deste trabalho está relacionado ao grau dos polinômios que definem duas curvas no $\mathbb{R}^2$. Se o grau for bastante elevado, o cálculo analítico pela resultante fica muito custoso, pois a elevação de grau é proporcional à elevação de número de linhas e colunas de uma matriz, cujo determinante é a resultante das curvas. Além disso, se o grau da resultante for alto, também é muito custoso encontrar as suas raízes para determinar os pontos de interseção. Essas razões tornaram interessante a aplicação dos métodos de curvas de Bézier ao invés do método analítico.

1.2 Roteiro

Capítulo 2: Contém as principais características e propriedades das curvas de Bézier. São mostradas algumas definições básicas que ajudam a compreender os passos para a construção de uma curva de graus 2 e 3 e a generalização para curvas de grau $n > 3$.

Capítulo 3: Mostra os principais algoritmos de interseção de duas curvas de Bézier encontrados na literatura, os quais são: Subdivisão recursiva de De Casteljau, Subdivisão intervalar adotado por Koparkar e Mudur (SEDERBERG ; PARRY, 1986) e Bézier Clipping (NISHITA ; SEDERBERG, 1990).

Capítulo 4: Neste capítulo, serão mostradas as definições do Teorema de Bézout, da Resultante de duas curvas algébricas planas e alguns exemplos de aplicação dos algoritmos vistos no capítulo 3 às interseções de curvas algébricas planas. Além disso, será mostrado um exemplo complementar de interseção de funções de uma variável.

Capítulo 5: Apresenta as conclusões deste trabalho, as vantagens e desvantagens de cada algoritmo estudado e uma análise comparativa com os demais métodos encontrados na literatura.

2 CURVAS DE BÉZIER

A base para compreender o processo de construção das curvas de Bézier nas seções deste capítulo começa a partir das curvas lineares. Em seguida, serão tratadas as curvas de grau arbitrário $n \geq 2$.

2.1 Segmentos de retas

São produzidos através do movimento de um ponto intermediário R_0 no segmento de reta que une os pontos P_0 e P_1 no $\mathbb{R}^2$. Os pontos P_0 e P_1 são denominados pontos de controle. Na Figura 2.1, são mostradas as etapas de construção de um segmento de reta de Bézier.

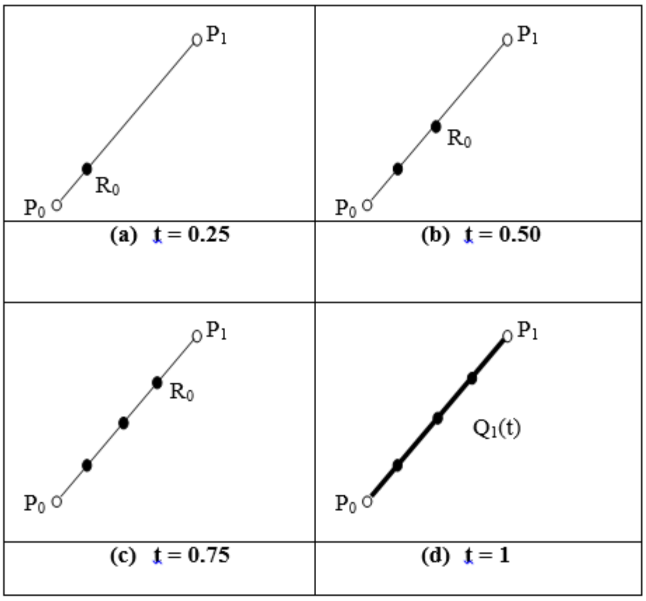

Figura 2.1: Etapas da construção do segmento de reta

Como pode ser observado, o ponto R_0 divide o segmento a cada instante por um parâmetro real $t \in [0,1]$. Então, conclui-se que:

$$R_0 - P_0 = t(P_1 - P_0), 0 \leq t \leq 1. \tag{2.1}$$

Desenvolvendo a equação 2.1, chega-se ao seguinte resultado:

$$R_0 = (1-t)P_0 + tP_1, 0 \leq t \leq 1. \tag{2.2}$$

2.2 Curvas de Bézier de grau 2

As curvas de Bézier de grau 2 são definidas por três pontos de controle P_0, P_1 e P_2 no $\mathbb{R}^2$. Elas são formadas pela concatenação de dois segmentos de retas, as quais são os segmentos P_0P_1 e P_1P_2. A Figura 2.2 mostra o processo de construção da curva em valores específicos desde $t = 0.25$ até chegar à curva quadrática $Q_2(t)$, quando $t = 1$. Através dos pontos de controle, construímos sequencialmente os segmentos de reta P_0P_1 e P_1P_2. Em seguida, teremos os pontos R_0 e R_1 dividindo estes segmentos pela mesma razão. Finalmente, através de R_0 e R_1, o segmento R_0R_1 será construído e teremos o ponto S_0 dividindo este último segmento pela mesma razão dos segmentos anteriores. A razão utilizada para dividir os segmentos será o parâmetro real $t \in [0,1]$.

Como podemos observar na Figura 2.2, o ponto S_0 se movimenta sobre uma curva de Bézier quadrática $Q_2(t)$ se R_0 se move sobre o segmento de reta P_0P_1 e R_1 sobre P_1P_2.

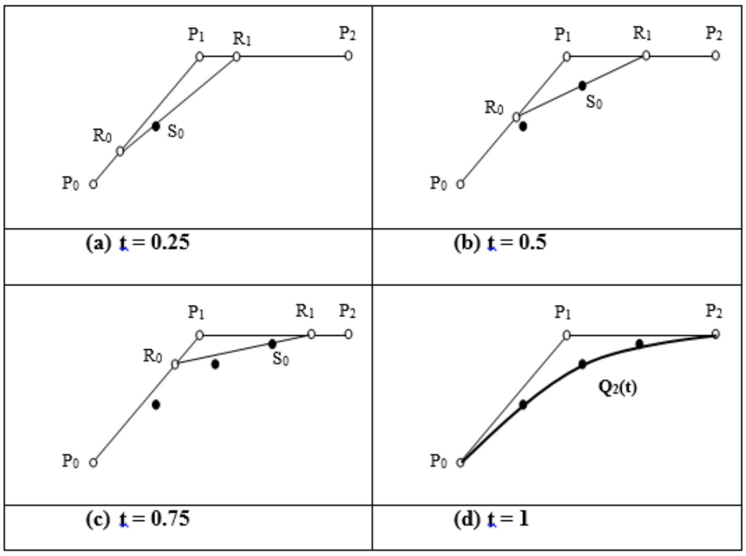

Figura 2.2: Etapas do processo de construção da curva de Bézier de grau 2

Assim, para algum parâmetro $t \in \mathbb{R}$:

$$R_0 - P_0 = t(P_1 - P_0) \Rightarrow R_0 = (1-t)P_0 + tP_1. \tag{2.3}$$

$$R_1 - P_1 = t(P_2 - P_1) \Rightarrow R_1 = (1-t)P_1 + tP_2. \tag{2.4}$$

$$S_0 - R_0 = t(R_1 - R_0) \Rightarrow S_0 = (1-t)R_0 + tR_1, 0 \leq t \leq 1. \tag{2.5}$$

Substituindo os valores de R_0 e R_1 das equações 2.3 e 2.4 na equação 2.5, chega-se ao resultado:

$$S_0 = Q_2(t) = (1-t)^2 P_0 + 2(1-t)tP_1 + t^2 P_2, 0 \leq t \leq 1. \tag{2.6}$$

Percebe-se então que o ponto inicial P_0 e o ponto final P_2 interceptam a curva quadrática $Q_2(t)$ e o ponto P_1 influencia o comportamento da curva. (BUSS, 2003)

2.3 Curvas de Bézier de grau 3

As curvas de Bézier mais comuns são as de grau 3, que são definidas por quatro pontos de controle P_0, P_1, P_2 e P_3 no $\mathbb{R}^2$. O método de construção, que é análogo ao que foi mostrado para o caso da curva quadrática, é ilustrado na Figura 2.3:

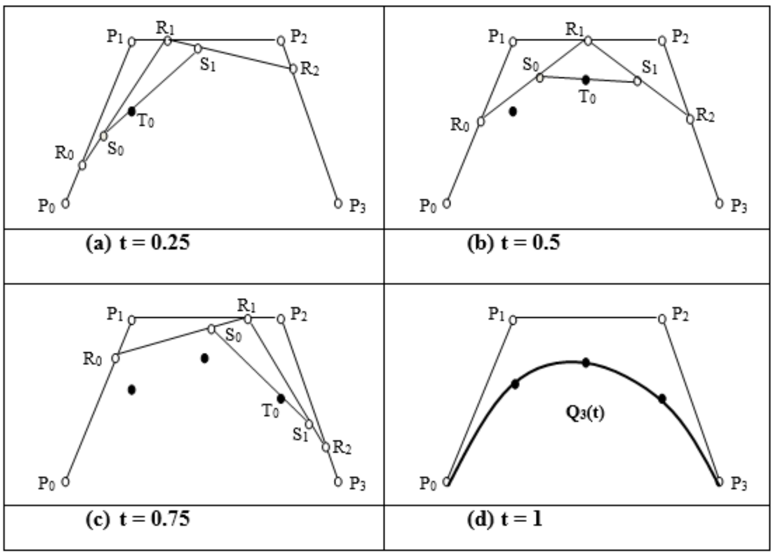

Figura 2.3: O processo de construção da curva de Bézier de grau 3

Como podemos observar na Figura 2.3, o ponto T_0 se movimenta sobre uma curva de Bézier cúbica $Q_3(t)$ se R_0 se move sobre o segmento de reta P_0P_1, R_1 sobre P_1P_2, R_2 sobre P_2P_3, S_0 sobre R_0R_1, S_1 sobre R_1R_2, T_0 sobre S_0S_1, e além disso, podemos ver que S_0 se movimenta sobre a curva quadrática definida pelos pontos P_0, P_1 e P_2, e S_1 se movimenta sobre a curva quadrática definida pelos pontos P_1, P_2 e P_3.

Assim, para algum parâmetro $t \in \mathbb{R}$:

$$R_0 - P_0 = t(P_1 - P_0) \Rightarrow R_0 = (1-t)P_0 + tP_1 \tag{2.7}$$

$$R_1 - P_1 = t(P_2 - P_1) \Rightarrow R_1 = (1-t)P_1 + tP_2 \tag{2.8}$$

$$R_2 - P_2 = t(P_3 - P_2) \Rightarrow R_2 = (1-t)P_2 + tP_3 \tag{2.9}$$

$$S_0 - R_0 = t(R_1 - R_0) \Rightarrow S_0 = (1-t)R_0 + tR_1 \tag{2.10}$$

$$S_1 - R_1 = t(R_2 - R_1) \Rightarrow S_1 = (1-t)R_1 + tR_2 \tag{2.11}$$

$$T_0 - S_0 = t(S_1 - S_0) \Rightarrow T_0 = (1-t)S_0 + tS_1, 0 \leq t \leq 1. \tag{2.12}$$

Substituindo os valores de R_0 e R_1 das equações 2.7 e 2.8 na equação 2.10, tem-se:

$$S_0 = (1-t)^2 P_0 + 2(1-t)tP_1 + t^2 P_2, 0 \leq t \leq 1. \tag{2.13}$$

A equação 2.13 já mostra que S_0 se move sobre uma curva quadrática de Bézier definida pelos pontos P_0, P_1 e P_2.

Substituindo os valores de R_1 e R_2 das equações 2.8 e 2.9 na equação 2.11, tem-se:

$$S_1 = (1-t)^2 P_1 + 2(1-t)tP_2 + t^2 P_3, 0 \leq t \leq 1. \tag{2.14}$$

Essa equação já mostra que S_1 se move sobre uma curva quadrática de Bézier definida pelos pontos P_1, P_2 e P_3.

Finalmente, substituindo os valores de S_0 e S_1 das equações 2.13 e 2.14 na equação 2.12, chega-se à equação da curva de Bézier cúbica:

$$T_0 = Q_3(t) = (1-t)^3 P_0 + 3(1-t)^2 tP_1 + 3(1-t)t^2 P_2 + t^3 P_3, 0 \leq t \leq 1. \tag{2.15}$$

2.4 Curvas de Bézier de grau arbitrário

Seja $n \geq 1$ um número inteiro. A curva de Bézier de grau n, $Q_n(t)$, determinada por $n+1$ pontos de controle $P_0, P_1, ..., P_n$ é uma curva definida parametricamente por:

$$Q_n(t) = \sum_{i=0}^{n} P_i B_i^n(t) \tag{2.16}$$

no domínio $t \in [0, 1]$, onde

$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i, 0 \leq i \leq n \tag{2.17}$$

são as funções-base denominadas *Polinômios de Bernstein*.

Na equação 2.15 da curva de Bézier de grau 3, os polinômios de Bernstein são: $B_0^3(t) = (1-t)^3; B_1^3(t) = 3(1-t)^2t; B_2^3(t) = 3(1-t)t^2; B_3^3(t) = t^3$.

Essas quatro funções estão representadas no gráfico da Figura 2.4.

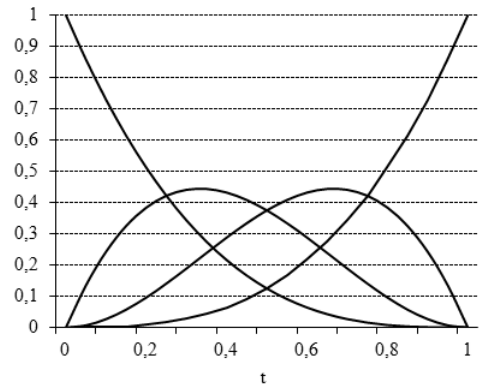

Figura 2.4: Polinômios de Bernstein de grau 3 (BUSS, 2003).

A soma dessas quatro funções é sempre igual a 1. Isso pode ser verificado pelo teorema binomial:

$$\sum_{i=0}^{3} B_3^i(t) = \sum_{i=0}^{3} \binom{3}{i} (1-t)^{3-i} t^i = [t + (1-t)]^3 = 1. \tag{2.18}$$

Temos também que $B_3^0(0) = 1$ e $B_3^3(1) = 1$. A partir deste resultado e do resultado de 2.18, conclui-se que $Q_3(t)$ é sempre calculada como uma média ponderada dos quatro pontos de controle e que $Q_3(t) = P_0$ e $Q_3(1) = P_3$, confirmando a observação de que $Q_3(t)$ começa em P_0 e termina em P_3. (BUSS, 2003); (LENGYEL, 2004).

Caso o domínio seja arbitrário $t \in [a, b]$ para algum $t \in \mathbb{R}$, as funções-base se tornam:

$$B_n^i(t) = \binom{n}{i} \left(\frac{b-t}{b-a}\right)^{n-i} \left(\frac{t-a}{b-a}\right)^i, 0 \leq i \leq n. \tag{2.19}$$

A Figura 2.5 mostra um exemplo de curva de Bézier de grau 3.

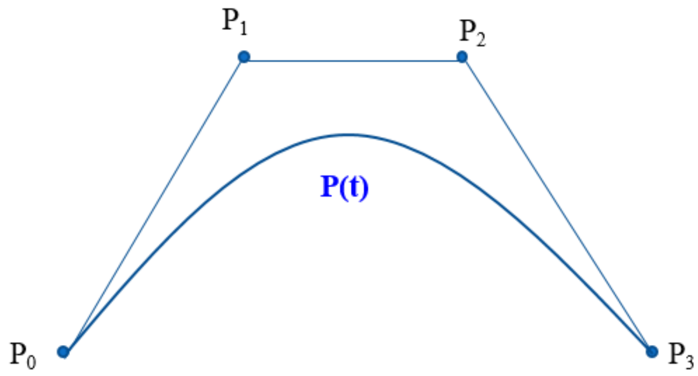

Figura 2.5: Curva de Bézier de grau 3.

2.5 Principais características

Algumas definições importantes das curvas de Bézier são (BUSS, 2003):

1. Para curvas de Bézier $P(t), 0 \leq t \leq 1$, de grau $n \geq 2$, o polígono formado pelos pontos de controle $P_0, P_1, ..., P_n$ formará o polígono de controle. A Figura 2.5 mostra uma curva cúbica e seu respectivo polígono de controle.

2. O ponto inicial P_0 e o ponto final P_n são denominados pontos extremos do polígono de controle, pois se localizam nas extremidades dele. Os pontos $P_1, P_2, ..., P_{n-1}$ são denominados pontos intermediários ou interiores do polígono de controle.

3. Os pontos extremos do polígono de controle também são extremidades da curva. Assim, $P_0 = Q_n(0)$ e $P_n = Q_n(1)$.

4. **Propriedade do fecho convexo:** a curva sempre estará contida no fecho convexo do polígono de controle.

5. **Propriedade de invariância afim:** Qualquer transformação nos pontos de controle implica em transformação na curva. A curva é uma aproximação mais suave do polígono de controle. Assim, a quantidade de vezes que os segmentos do polígono de controle interceptam a curva será no máximo a quantidade destes segmentos.

6. As derivadas da curva de Bézier de grau n nos pontos extremos:

$$\begin{aligned} P'(0) &= n(P_1 - P_0); \\ P'(1) &= n(P_n - P_{n-1}) \end{aligned} \tag{2.20}$$

   Caso o domínio seja arbitrário $t \in [a, b]$ para algum $t \in \mathbb{R}$, as derivadas nos pontos extremos se tornam:

$$\begin{aligned} P'(a) &= \left(\frac{n}{b-a}\right)(P_1 - P_0); \\ P'(b) &= \left(\frac{n}{b-a}\right)(P_n - P_{n-1}) \end{aligned} \tag{2.21}$$

2.6 Elevação de Grau

Dada uma curva de Bézier $P(t)$ de grau n definida pelos $n+1$ pontos de controle P_0, P_1, ... , P_n, podem ser encontrados $n+2$ pontos de controle $\widehat{P}_0$, $\widehat{P}_1$, ..., $\widehat{P}_{n+1}$ que definem uma curva de Bézier $\widehat{P}(t)$ de grau $n+1$ idêntica à curva $P(t)$. Para isso, deve ser usada a seguinte relação de recorrência (BUSS, 2003):

$$\begin{aligned}\widehat{P}_0 &= P_0;\\ \widehat{P}_{n+1} &= P_n;\\ \widehat{P}_i &= \left(\frac{i}{n+1}\right)P_{i-1} + \left(1 - \frac{i}{n+1}\right)P_i\end{aligned} \tag{2.22}$$

Na Figura 2.6, é mostrado um exemplo de elevação de uma curva de grau 2, cujos pontos de controle são P_0, P_1, P_2, para grau 3, onde os novos pontos de controle são $\widehat{P}_0, \widehat{P}_1, \widehat{P}_2, \widehat{P}_3$, definidos de acordo com a relação de recorrência vista anteriormente.

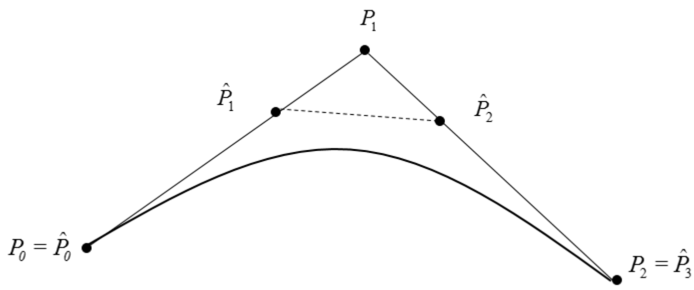

Figura 2.6: Elevação de grau de uma curva de grau 2 para grau 3 (BUSS, 2003)

3 PRINCIPAIS ALGORITMOS DE INTERSEÇÕES DE CURVAS DE BÉZIER

Neste capítulo, serão descritos os três algoritmos principais para cálculo das interseções entre duas curvas de Bézier, que neste trabalho serão utilizadas como aproximações para curvas algébricas planas num dado intervalo.

3.1 Subdivisão recursiva de De Casteljau

É o processo de divisão de uma curva de Bézier em duas sub-curvas usando o método de De Casteljau. É muito usado quando se quer aproximar uma curva de Bézier por segmentos de reta. (BUSS, 2003)

3.1.1 Área envolvente

O processo de subdivisão de De Casteljau utiliza como área envolvente para a curva o fecho convexo de seu polígono de controle. Suponha que uma curva de Bézier $Q_3\left(t\right)$ possui fecho convexo formado pelos pontos de controle P_0, P_1, P_2 e P_3 e é dividida em duas partes:

$$Q_{31}\left(t\right) = Q_3\left(t/2\right)$$

e

$$Q_{32}(t) = Q_3\left(\left(t+1\right)/2\right)$$

Então, $Q_{31}(t)$ e $Q_{32}(t)$ são também curvas cúbicas de Bézier. Como $Q_{31}(t)$ e $Q_{32}(t)$ ficam restritos ao domínio $[0,1]$, então $Q_{31}(t)$ é a primeira metade da curva e $Q_{32}(t)$ é a segunda metade da curva.

A Figura 3.1 mostra o resultado da subdivisão: $Q_{31}(t)$, cujo fecho convexo é formado pelos pontos de controle P_0, R_0, S_0, T_0, e $Q_{32}(t)$, cujo fecho convexo é formado pelos pontos de controle T_0, S_1, R_2, P_3

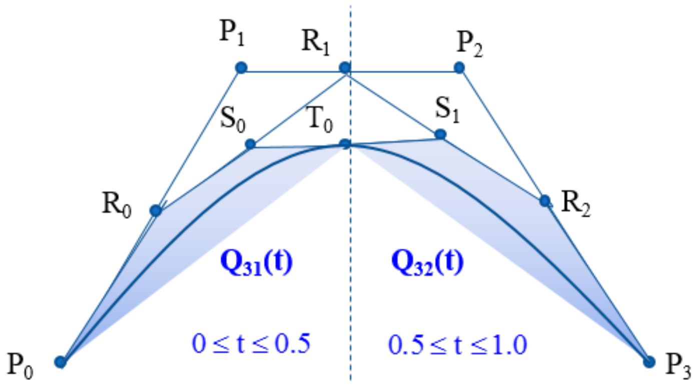

Figura 3.1: Na subdivisão recursiva, duas sub-curvas são obtidas

3.1.2 Funcionamento do algoritmo

No algoritmo, o fecho convexo do polígono de controle de cada curva é subdividido ao meio em dois fechos convexos menores que vão cobrir as duas sub-curvas resultantes. Depois, deve-se aplicar o procedimento de verificação de interseção nos fechos convexos menores que são gerados. Quando há interseção, o processo de subdivisão continua nos fechos convexos menores onde o teste de detecção foi positivo. Quando não há interseção, o processo de subdivisão é interrompido nos fechos convexos menores onde não há sobreposição.

A Figura 3.2 mostra um exemplo de três iterações para interseção de duas curvas de Bézier de grau 3. (SEDERBERG ; PARRY, 1986)

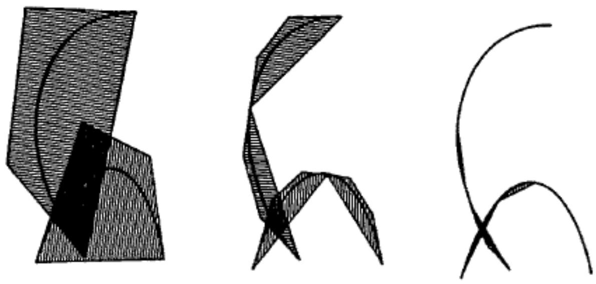

Figura 3.2: Três iterações da subdivisão recursiva de De Casteljau

A subdivisão recursiva associada ao método de De Casteljau é uma propriedade especial porque sempre resulta em polígonos de controle que convergem para a própria curva. (BUSS, 2003)

3.2 Subdivisão intervalar de Koparkar e Mudur

Trata-se de um algoritmo que funciona de forma análoga ao da Subdivisão recursiva de De Casteljau.

3.2.1 Área envolvente

Na subdivisão intervalar, ao invés da utilização do fecho convexo do polígono de controle, utiliza-se um *retângulo delimitador* para cobrir a curva nas coordenadas mínimas e máximas.

Cada curva é pré-processada para determinar suas tangentes horizontais e verticais nos pontos de coordenadas mínimas e máximas. As delimitações destas tangentes definem os intervalos que vão corresponder aos lados do retângulo.

A Figura 3.3 mostra um exemplo de uma curva $P(t)$ formada pelos pontos $P_i = (x_i, y_i), 0 \leq i \leq 3$ e seu retângulo delimitador onde $x_{min} = x_0$, $y_{min} = y_0$, $x_{max} = x_3$, $y_{max} = y_2$.

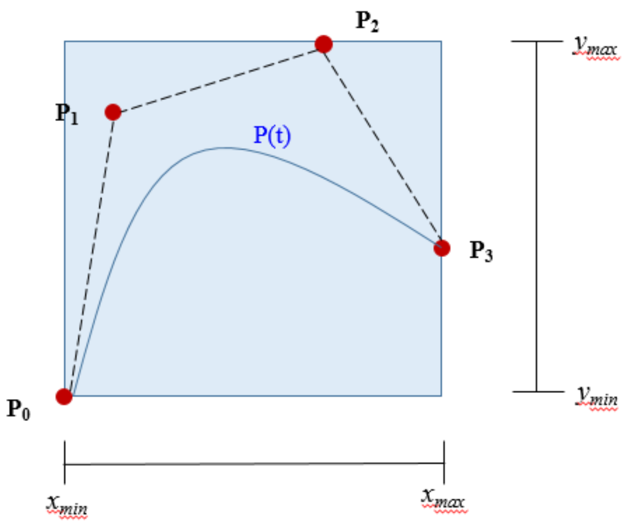

Figura 3.3: Uma curva de Bézier $P(t)$ e seu *retângulo delimitador*

3.2.2 Funcionamento do algoritmo

No algoritmo de Subdivisão Intervalar de Koparkar e Mudur, após a primeira iteração, de forma análoga ao que acontece na Subdivisão Recursiva de De Casteljau, o *retângulo delimitador* é dividido na coordenada correspondente à metade da curva, gerando dois novos *retângulos delimitadores*.

A Figura 3.4 mostra uma iteração do algoritmo para a curva da Figura 3.3, onde $x_{max1} = x_{min2} = x(t = 0.5)$ e $y_{max1} = y_{max2} = y(t = 0.5)$.

A Figura 3.5 mostra duas iterações do algoritmo para uma curva de Bézier cúbica. (SEDERBERG ; PARRY, 1986)

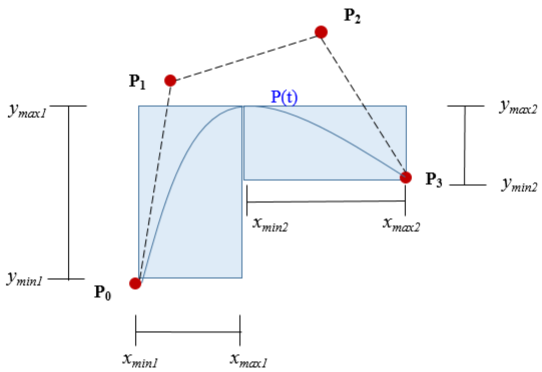

Figura 3.4: Uma iteração da subdivisão intervalar da curva da Figura 3.3

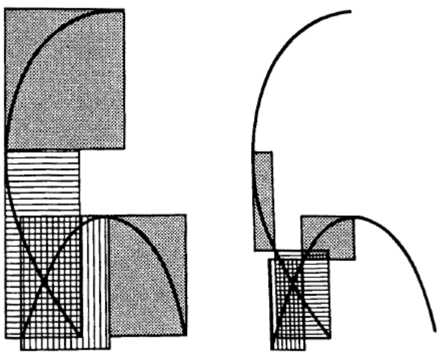

Figura 3.5: Duas iterações da subdivisão intervalar

3.3 Bézier Clipping

Trata-se de um algoritmo proposto em 1990 por Tomoyuki Nishita, professor do Departamento de Engenharia da Universidade de Tóquio, e Thomas W. Sederberg, professor do Departamento de Ciência da Computação da Universidade de Brigham Young, nos Estados Unidos, para o problema da interseção de curvas de Bézier. (NISHITA ; SEDERBERG, 1990)

3.3.1 Área envolvente

Fat Line. É a região de recorte que fica entre duas retas paralelas a uma reta L que intercepta os pontos de controle extremos da curva, sendo que uma das retas paralelas intercepta o ponto de controle que está à uma distância mínima de L e a outra intercepta o ponto de controle que está a uma distância máxima de L, como mostra a figura 3.6.

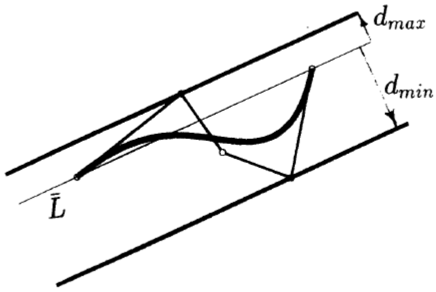

Figura 3.6: Fat Line limitando uma curva de Bézier de grau 4

Se a reta L possui equação implícita:

$$ax + by + c = 0 \tag{3.1}$$

Então, a distância de qualquer ponto (x, y) até L é dada por:

$$d(x, y) = \frac{ax + by + c}{\sqrt{a^2 + b^2}} \tag{3.2}$$

Assim, denotaremos por $d_i = d(x_i, y_i)$ a distância com sinal de um ponto de controle $P_i = (x_i, y_i)$ à reta L.

Logo, a região Fat Line cobrirá o seguinte conjunto de pontos:

$\{(x, y) | d_{min} \leq d(x, y) \leq d_{max}\}$

Onde

$d_{min} = min\{d_0, ..., d_n\}$

$d_{max} = max\{d_0, ..., d_n\}$

Para os casos das curvas quadráticas e cúbicas, que serão vistos a seguir, os cálculos de d_{min} e d_{max} são feitos de outra maneira para que a *Fat Line* se aproxime mais dos pontos da curva onde a derivada se anula.

Caso Quadrático. Se $d(t)$ é a distância de qualquer ponto na curva $P(t)$ até L, então, para curvas de Bézier quadráticas, temos:

$$d(t) = 2t(1-t)d_1 \tag{3.3}$$

cujos limites da Fat Line são dados por:

$$\begin{aligned} d_{min} &= min\left\{0, \frac{d_1}{2}\right\} \\ d_{max} &= max\left\{0, \frac{d_1}{2}\right\} \end{aligned} \tag{3.4}$$

O caso quadrático é mostrado na Figura 3.7 (NISHITA ; SEDERBERG, 1990).

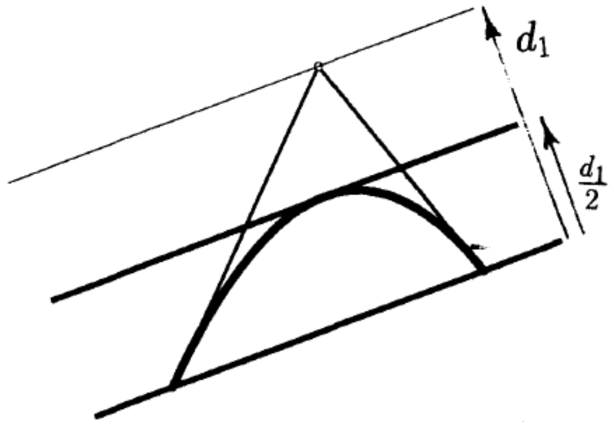

Figura 3.7: Fat Line para curva de Bézier quadrática

Caso cúbico. Para este caso, temos:

$$d(t) = 3t(1-t)[(1-t)d_1 + td_2] \tag{3.5}$$

Segundo (NISHITA ; SEDERBERG, 1990), no caso de curvas de Bézier cúbicas, devemos usar os seguintes casos para os valores de d_{min} e d_{max}:

Caso 1: $d_1 d_2 > 0$

Neste caso, os limites da Fat Line cobrem a seguinte região:

$$min\{0, d_1, d_2\}3t(1-t) \leq d(t) \leq max\{0, d_1, d_2\}3t(1-t) \tag{3.6}$$

Derivando a expressão $3t(1-t)$ de (3.6) e igualando a zero, é encontrado $t = \frac{1}{2}$. Assim, substituindo este valor de t em (3.6), podem ser usados os seguintes valores para d_{min} e d_{max}:

$$d_{min} = \frac{3}{4} min\{0, d_1, d_2\} d_{max} = \frac{3}{4} max\{0, d_1, d_2\} \tag{3.7}$$

Caso 2: $d_1 \leq 0$ e $d_2 \geq 0$ Neste caso, os limites da Fat Line cobrem a seguinte região:

$$3t(1-t)^2 d_1 \leq d(t) \leq 3t^2(1-t)d_2 \tag{3.8}$$

Neste caso, derivando as expressões $3t(1-t)^2$ e $3t^2(1-t)$ de (3.8) e igualando a zero, é encontrado $t = \frac{1}{3}$. Assim, substituindo este valor de t em (3.8), podem ser usados os seguintes valores para d_{min} e d_{max}:

$$\begin{aligned} d_{min} &= \frac{4}{9} min\{0, d_1, d_2\} \\ d_{max} &= \frac{4}{9} max\{0, d_1, d_2\} \end{aligned} \tag{3.9}$$

O caso cúbico é mostrado na Figura 3.8 (NISHITA ; SEDERBERG, 1990).

3.3.2 Funcionamento do algoritmo

Aqui, será mostrado o funcionamento para duas curvas cúbicas.

Sejam duas curvas de Bézier $P(t)$ e $Q(u)$ e uma *fat line* L que representa a fronteira de $Q(u)$. A Figura 3.9 mostra um exemplo.

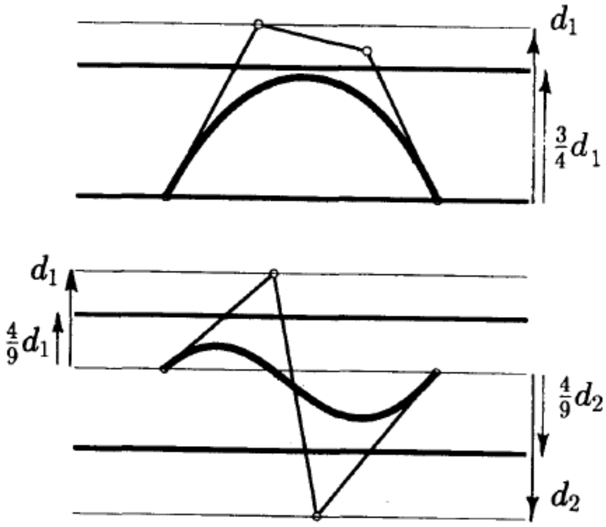

Figura 3.8: Fat Lines para curvas de Bézier cúbicas

Neste exemplo, dado por (NISHITA ; SEDERBERG, 1990), a curva de recorte inicial escolhida foi $P(t)$. As distâncias dos pontos de controle P_0, P_1, P_2, P_3 de $P(t)$ à reta dos extremos de $Q(u)$ dadas neste exemplo são respectivamente $d_0 = -5, d_1 = -1, d_2 = 2, d_3 = 3$.

Além disso, a distância mínima dos pontos de controle de $Q(u)$ à reta de seus extremos é dada por $d_{min} = -2$ e a distância máxima é dada por $d_{max} = 1$.

$P(t)$ é definida pela equação paramétrica

$P(t) = \sum_{i=0}^{n} P_i B_i^n(t)$

A função $d(t)$ é um polinômio na forma Bernstein-Bézier que faz a interpolação das distâncias dos pontos de controle da curva $P(t)$ à reta dos extremos da curva $Q(u)$ e pode ter a seguinte representação paramétrica:

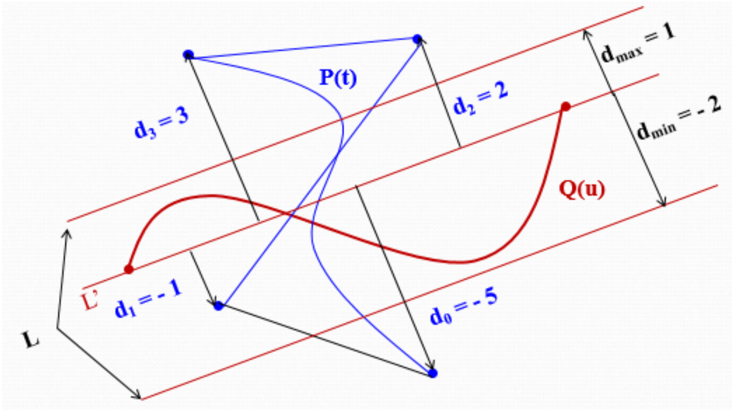

Figura 3.9: Fat Line para curva de Bézier cúbica Q(u) no processo de Bézier Clipping

$$D(t) = (t, d(t)) = \sum_{i=0}^{n} D_i B_i^n(t) \tag{3.10}$$

Nesta representação paramétrica, os pontos de controle $D_i = (t_i, d_i)$ são igualmente espaçados em $t(t_i = i/n)$ de forma que $\sum_{i=0}^{n}(i/n)B_i^n(t) = t[(1-t)+t]^n = t$. Assim, a coordenada horizontal de qualquer ponto $D(t)$ é de fato o parâmetro t.

A Figura 3.10 mostra a representação Bernstein-Bezier para interpolação das distâncias D_i (NISHITA ; SEDERBERG, 1990).

Agora, são feitos os seguintes cálculos para encontrar t_{min} e t_{max} no recorte da curva $P(t)$. Na Figura 3.10, t_{min} será a interseção do segmento D_0D_1 com a reta $d_{min} = -2$, enquanto t_{max} será a interseção do segmento D_0D_3 com a reta $d_{max} = 1$. Neste exemplo, $t_{min} = 0.25$ e $t_{max} = 0.75$.

Os valores de t para os quais $P(t)$ se encontra fora de L correspondem aos valores de t para os quais $D(t)$ se encontra abaixo de d_{min} ou acima de d_{max}. Na Figura 3.10, tem-se que $P(t)$ se encontra fora de L em $t < 0.25$ e $t > 0.75$. Dessa

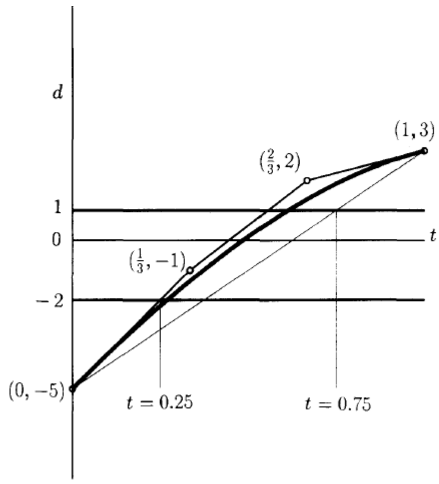

Figura 3.10: Curva de Bézier para representação Bernstein-Bézier de d(t)

forma, chega-se à uma das etapas do algoritmo de Bézier Clipping, que é o recorte das partes da curva $P(t)$ que ficarem nesses intervalos. Essas partes são descartadas de $P(t)$.

Após o recorte, uma fat line L será aplicada à curva $P(t)$ no intervalo $t \in [0.25, 0.75]$. O próximo passo é aplicar o recorte da curva $Q(u)$ através de $P(t)$, como mostra a Figura 3.11. Depois, $P(t)$ terá um novo recorte contra $Q(u)$ e assim alterna-se a curva de recorte sucessivamente.

3.4 Verificação de Interseções

3.4.1 Interseções múltiplas

No caso de múltiplas interseções, quando não há a possibilidade de se reduzir o intervalo de parâmetro da curva em que foi aplicado o recorte em pelo menos 20%, uma heurísitica sugerida por (NISHITA ; SEDERBERG, 1990) no algoritmo de

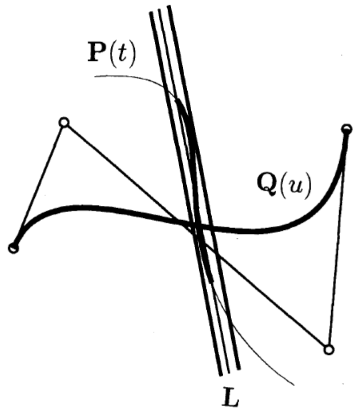

Figura 3.11: Segunda iteração do Bézier Clipping

Bézier Clipping é a seguinte: subdivide-se pela metade a curva com maior intervalo de parâmetro restante e interceptam-se suas sub-curvas resultantes com a curva de menor intervalo de parâmetro, como mostra a Figura 3.12.

Figura 3.12: (a) Duas interseções; (b) Resultado após a subdivisão de uma das curvas

3.4.2 Interseções simples

Para detectar interseções simples de duas curvas de Bézier, utiliza-se o critério do *retângulo delimitador* das curvas nos pontos máximo e mínimo. Este retângulo delimitará a curva nos lados que correspondem, respectivamente, ao intervalo dos pontos de coordenadas horizontais mínima e máxima e ao intervalo dos pontos de coordenadas verticais mínima e máxima. (MARSH, 2005)

A detecção neste caso funciona da seguinte forma: quando os respectivos retângulos delimitadores das curvas não se sobrepõem, não há interseção. Caso contrário, as curvas podem se interceptar ou não.

O procedimento para verificação de existência de interseções entre duas curvas de Bézier através de um *retângulo delimitador* é descrito da seguinte forma: (HECKBERT, 1994)

Algoritmo 1: VerificaIntersecao (P, Q)

% Procedimento que verifica se existem interseções simples entre duas curvas de Bézier P(t) e Q(u), cujos pontos de controle são respectivamente da forma (x_P, y_P) *e* (x_P, y_P).

% x_P*: coordenada horizontal dos pontos de controle de* $P(t)$.
% y_P*: coordenada vertical dos pontos de controle de* $P(t)$.
% x_Q*: coordenada horizontal dos pontos de controle de* $Q(u)$.
% y_Q*: coordenada vertical dos pontos de controle de* $Q(u)$.

se $(min(x_P) \geq max(x_Q))$ ***ou*** $(min(y_P) \geq max(y_Q))$ ***ou*** $(min(x_Q) \geq max(x_P))$ ***ou*** $(min(y_Q) \geq max(y_P))$ **então**
| Retorna 0;
senão
| Retorna 1;
fim se

4 CÁLCULO DE INTERSEÇÕES

Algumas definições que serão apresentadas a seguir são importantes para o cálculo das interseções de duas curvas algébricas planas.

4.1 Resultante

Uma curva é definida aqui como o conjunto de pontos em que uma função polinomial $f : \mathbb{R}^2 \to \mathbb{R}$ é igual a zero. Sejam duas curvas definidas por $f(x, y) = 0$ e $g(x, y) = 0$, representadas pelos seguintes polinômios na variável y e coeficientes no anel $\mathbb{R}[x]$:

$$f(x, y) = a_0(x)y^m + a_1(x)y^{m-1} + ... + a_m(x) = 0$$

$$g(x, y) = b_0(x)y^n + b_1(x)y^{n-1} + ... + b_n(x) = 0$$

A resultante de f, g em relação à variável x é definida pelo determinante da matriz de ordem $(m+n)$ com n linhas formadas pelos coeficientes de f seguidas por m linhas formadas pelos coeficientes de g. Subentende-se que os espaços em branco são preenchidos com zeros (VAINSENCHER, 2009).

$$R_{f,g} = \begin{vmatrix} a_0 & a_1 & a_2 & \ldots & a_m & & & \ldots & & \\ & a_0 & a_1 & \ldots & & a_m & & & \ldots & \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \\ & & & & \ldots & a_0 & a_1 & \ldots & & a_m \\ b_0 & b_1 & b_2 & \ldots & b_n & & & \ldots & & \\ & b_0 & b_1 & \ldots & & b_n & & & \ldots & \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \\ & & & & \ldots & b_0 & b_1 & \ldots & & b_n \end{vmatrix}$$

Uma observação importante é que a resultante de f, g pode ser calculada em relação à variável y, caso os polinômios estejam na variável x e os coeficientes no anel $\mathbb{R}[y]$.

Exemplo Calcular a resultante dos seguintes polinômios:

$$f(x, y) = x^2 + y^2 - 4$$

$$g(x, y) = xy - 1$$

Considerando os coeficientes no anel $\mathbb{R}[x]$, deve-se resolver o seguinte determinante:

$$R_{f,g} = \begin{vmatrix} 1 & 0 & x^2 - 4 \\ x & -1 & 0 \\ 0 & x & -1 \end{vmatrix}$$

Resolvendo-se esse determinante, encontra-se $R_{f,g} = x^4 - 4x^2 + 1$.

Para determinar os pontos de interseção de duas curvas algébricas planas f, g, basta realizar o cálculo $R_{f,g} = 0$, achar os valores de x e, em seguida, substituí-los em uma das equações para f ou g.

Neste exemplo, os pontos de interseção serão, com precisão de sete casas decimais:

$$(1.9318517, 0.5176381); (0.5176381, 1.9318517);$$

$$(-1.9318517, -0.5176381); (-0.5176381, -1.9318517).$$

4.2 Teorema de Bézout

Dados dois polinômios $f, g \in \mathbb{R}[x, y]$ e sejam os graus de f, g dados respectivamente por $grau(f)$ e $grau(g)$. Se f, g não possuem fatores comuns, ou seja, se $R_{f,g}$ não for igual a um polinômio identicamente nulo ($R_{f,g} \neq 0$), então o número de interseções entre as curvas f, g é finito e, no máximo, igual a $grau(f) \cdot grau(g)$ (COX ; LITTLE ; O'SHEA, 2007).

Observe que dois polinômios f, g não possuem fatores comuns se e somente se $mdc(f, g) = 1$. No exemplo visto na seção 4.1, como $R_{f,g} = x^4 - 4x^2 + 1$, ou seja, $R_{f,g}$ não é um polinômio identicamente nulo, então, de acordo com o Teorema de Bézout, o número de interseções deve ser, no máximo, igual a $grau(f) \cdot grau(g) = 2 \cdot 2 = 4$.

4.3 Aproximação de curvas algébricas planas por curvas de Bézier

Para encontrar os pontos de controle referentes à aproximação de uma curva algébrica plana por uma curva de Bézier, há duas alternativas:

4.3.1 Parametrização com Elevação de Grau

Os passos para esta alternativa são os seguintes:

1. Encontrar as parametrizações da curva algébrica plana em função de um ângulo θ, dadas por suas coordenadas $x(\theta)$ e $y(\theta)$. A curva fica localizada no intervalo $\theta \in [\theta_0, \theta_f]$, onde θ_0 e θ_f são notações para ângulos correspondentes aos pontos de controle inicial e final da curva, respectivamente.

2. Após a definição dos pontos extremos, definir um ponto intermediário na metade da curva, cujo ângulo é dado por $\theta_{1/2} = \frac{\theta_0+\theta_f}{2}$.

3. Em seguida, fazer a aproximação por uma curva de Bézier quadrática $P(t)$, cuja parametrização é $(x(t), y(t))$, $t \in [0, 1]$, $t \in \mathbb{R}$. Assim, os pontos extremos serão:

   $P_0 = P(t = 0) = (x(\theta_0), y(\theta_0))$;

   $P_2 = P(t = 1) = (x(\theta_f), y(\theta_f))$;

   O ponto intermediário da curva localizado em $t = 0.5$ será dado por:

   $P(t = 0.5) = (x(\theta_{1/2}), y(\theta_{1/2}))$;

   Assim, com os valores de $P(0.5)$, P_0 e P_1, o ponto de controle P_1 é encontrado através das substituições desses valores na seguinte equação:

   $P(t = 0.5) = (1 - 0.5)^2 P_0 + 2(1 - 0.5)(0.5)P_1 + (0.5)^2 P_2$

4. Ao final do passo anterior, deve-se encontrar a curva de grau 3 $\widehat{P(t)}$ através do processo de elevação de grau visto no capítulo 2. Aqui, são encontrados os pontos $\widehat{P_0}, \widehat{P_1}, \widehat{P_2}, \widehat{P_3}$.

4.3.2 Interpolação de curvas de Bézier

Com uma amostra de pontos $V_0, V_1, ..., V_n$ de uma curva algébrica plana num dado intervalo $[a, b]$, que possui $n+1$ subintervalos igualmente espaçados de tamanho $\frac{b-a}{n}$, é possível fazer uma interpolação destes pontos através de uma curva de Bézier de grau n para encontrar os pontos de controle $P_0, P_1, ..., P_n$ da curva.

Para um determinado ponto $V_i = (x_{V_i}, y_{V_i})$, podemos fazer as seguintes interpolações em suas respectivas coordenadas:

$$B_0^n(t_i)x_{P_0} + ... + B_n^n(t_i)x_{P_n} = x_{V_i}$$

$$B_0^n(t_i)y_{P_0} + ... + B_n^n(t_i)y_{P_n} = y_{V_i}$$

Suponha que o intervalo seja $[0, 1]$ e que n seja o número de subdivisões deste intervalo. Dessa forma, pode ser obtido o sistema linear $MP = V$, onde:

$$M = \begin{bmatrix} B_0^n(\frac{0}{n}) & B_1^n(\frac{0}{n}) & ... & B_n^n(\frac{0}{n}) \\ B_0^n(\frac{1}{n}) & B_1^n(\frac{1}{n}) & ... & B_n^n(\frac{1}{n}) \\ ... & ... & ... & ... \\ B_0^n(\frac{n}{n}) & B_1^n(\frac{n}{n}) & ... & B_n^n(\frac{n}{n}) \end{bmatrix}$$

$$P = \begin{bmatrix} P_0 \\ P_1 \\ ... \\ P_n \end{bmatrix}$$

$$V = \begin{bmatrix} V_0 \\ V_1 \\ ... \\ V_n \end{bmatrix}$$

Então, para achar os pontos de controle, basta resolver o sistema $P = M^{-1}V$.

4.4 Exemplos de cálculo de interseções

Nos exemplos a seguir, os cálculos foram feitos a partir de programas desenvolvidos em Scilab versão 5.4.0. Os programas estão nos apêndices A, B, C, D.

4.4.1 Exemplo 1

Encontrar as interseções entre as curvas algébricas planas:

$$f(x, y) = x^2 - 4x + 4y^2 = 0$$

$$g(x, y) = x^2 - 8x + y^2 + 12 = 0$$

nos intervalos $x \in [2, 4]$, usando o algoritmo de Bézier Clipping.

Considerando os coeficientes no anel $\mathbb{R}[x]$, deve-se resolver o seguinte determinante para achar a resultante de f e g:

$$R_{f,g} = \begin{vmatrix} 4 & 0 & x^2 - 4x & 0 \\ 0 & 4 & 0 & x^2 - 4x \\ 1 & 0 & x^2 - 8x + 12 & 0 \\ 0 & 1 & 0 & x^2 - 8x + 12 \end{vmatrix}$$

Esse determinante resolvido no Scilab tem como resultado $R_{f,g} = 2304 - 2688x + 1072x^2 - 168x^3 + 9x^4$, que não é um polinômio identicamente nulo. Logo,

$mdc(f, g) = 1$ e assim, de acordo com o Teorema de Bézout, f e g não possuem fatores comuns. Logo, f e g possuem no máximo $grau(f) \cdot grau(g) = 4$ interseções.

O número de interseções de f e g é igual a 2, como mostra a Figura 4.1, o que está de acordo com o Teorema de Bézout. Além disso, as interseções se localizam no intervalo $x \in [2, 4]$.

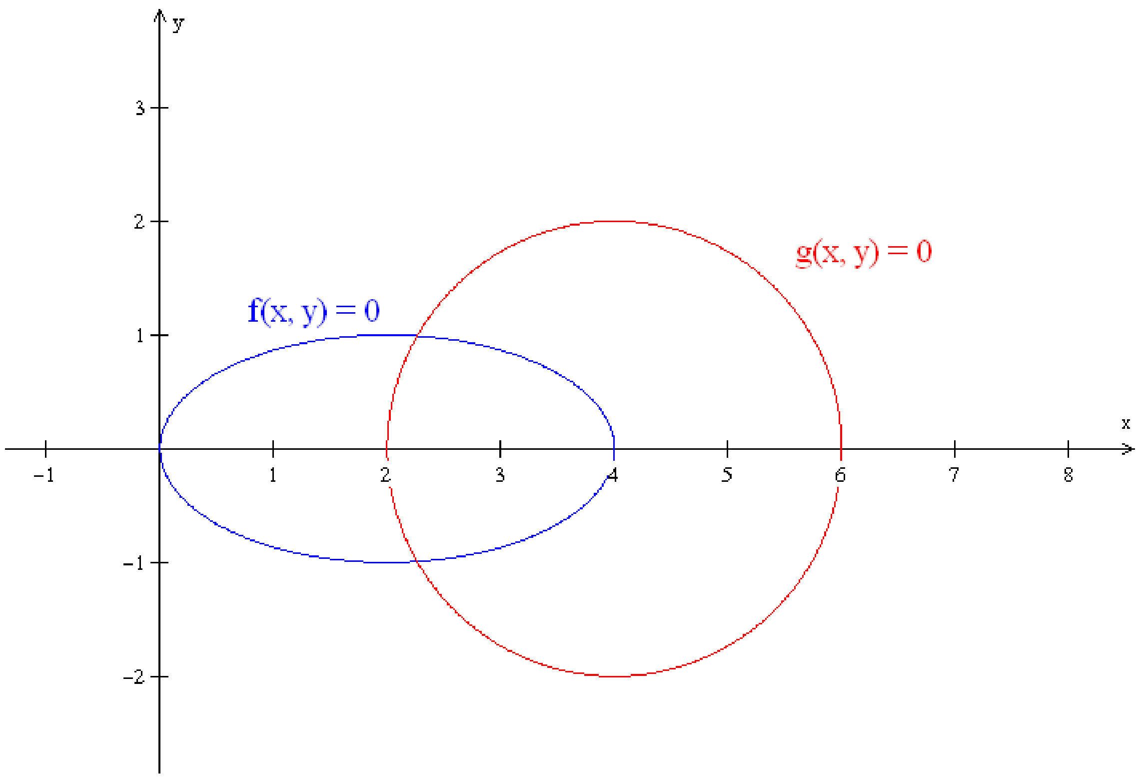

Figura 4.1: Gráfico do exemplo 1 com os pontos de controle no intervalo $x \in [2, 4]$

Para localizar as interseções das curvas associadas às funções f e g, são realizados os seguintes experimentos:

4.4.1.1 Experimento 1

a) Parametrização com elevação de grau:

Primeiro, a curva de equação $f(x, y) = 0$, representada por uma elipse de centro $(2, 0)$, será aproximada pela curva de Bézier $P(t)$ quadrática no intervalo $t \in [0, 1]$, e a curva de equação $g(x, y) = 0$, representada por uma circunferência de raio 2 e centro $(4, 0)$, será aproximada pela curva de Bézier $Q(u)$ no intervalo $u \in [0, 1]$.

Uma parametrização para $f(x, y) = 0$ é: $x(\theta) = 2 + 2cos(\theta)$, $y(\theta) = sen(\theta)$, $\theta \in [\frac{-\pi}{2}, \frac{\pi}{2}]$.

Os pontos extremos da curva $P(t)$, que será a aproximação de $f(x, y) = 0$ no intervalo $t \in [0, 1]$, ocorrem quando $\theta_0 = \frac{-\pi}{2}$ e $\theta_f = \frac{\pi}{2}$.

Neste caso, os pontos extremos são:

$P_0 = (x(\theta_0), y(\theta_0)) = (2, 1)$;

$P_2 = (x(\theta_f), y(\theta_f)) = (2, -1)$.

O ponto intermediário da curva ocorre quando $\theta_{1/2} = \frac{\theta_0+\theta_f}{2} = 0$.

Dessa forma, o ponto intermediário da curva ocorre quando:

$P(0.5) = (x(\theta_{1/2}), y(\theta_{1/2})) = (4, 0)$.

Assim,

$(4,0) = P(0.5) = 0.25P_0 + 0.5P_1 + 0.25P_2$

Substituindo os valores de P_0 e P_2, é encontrado:

$P_1 = (6,0)$.

Para achar os pontos de controle para a curva de Bézier $P(t)$, com grau 3, usa-se a técnica de elevação de grau. Os novos pontos de controle, neste caso, serão dados por:

$\widehat{P_0} = P_0 = (2,1)$;

$\widehat{P_1} = \frac{1}{3}P_0 + \frac{2}{3}P_1 = (4.6666667, 0.3333333)$;

$\widehat{P_2} = \frac{2}{3}P_1 + \frac{1}{3}P_2 = (4.6666667, -0.3333333)$;

$\widehat{P_3} = P_2 = (2,-1)$

Agora, é a vez da curva $Q(u)$, que será a aproximação de $g(x,y) = 0$ no intervalo $u \in [0,1]$. Uma parametrização para esta curva é: $x(\theta) = 4 + 2cos(\theta)$, $y(\theta) = 2sen(\theta)$, $\theta \in [\frac{\pi}{2}, \frac{3\pi}{2}]$.

Os pontos extremos da curva ocorrem quando $\theta_0 = \frac{\pi}{2}$ e $\theta_f = \frac{3\pi}{2}$.

Neste caso, os pontos extremos são:

$Q_0 = (x(\theta_0), y(\theta_0)) = (4,2)$;

$Q_2 = (x(\theta_f), y(\theta_f)) = (4,-2)$.

O ponto intermediário da curva ocorre quando $\theta_{1/2} = \frac{\theta_0+\theta_f}{2} = \pi$.

Dessa forma, o ponto intermediário da curva ocorre quando:

$Q(0.5) = (x(\theta_{1/2}), y(\theta_{1/2})) = (2, 0)$.

Assim,

$(2, 0) = Q(0.5) = 0.25Q_0 + 0.5Q_1 + 0.25Q_2$

Substituindo os valores de Q_0 e Q_2, é encontrado:

$Q_1 = (0, 0)$.

Para achar os pontos de controle para a curva de Bézier $Q(u)$, com grau 3, usa-se a técnica de elevação de grau. Os novos pontos de controle, neste caso, serão dados por:

$\widehat{Q_0} = Q_0 = (4, 2)$;

$\widehat{Q_1} = \frac{1}{3}Q_0 + \frac{2}{3}Q_1 = (1.3333333, 0.6666667)$;

$\widehat{Q_2} = \frac{2}{3}Q_1 + \frac{1}{3}Q_2 = (1.3333333, -0.6666667)$;

$\widehat{Q_3} = Q_2 = (4, -2)$

A Figura 4.2 mostra os gráficos das curvas originais e das curvas de Bézier aproximadas.

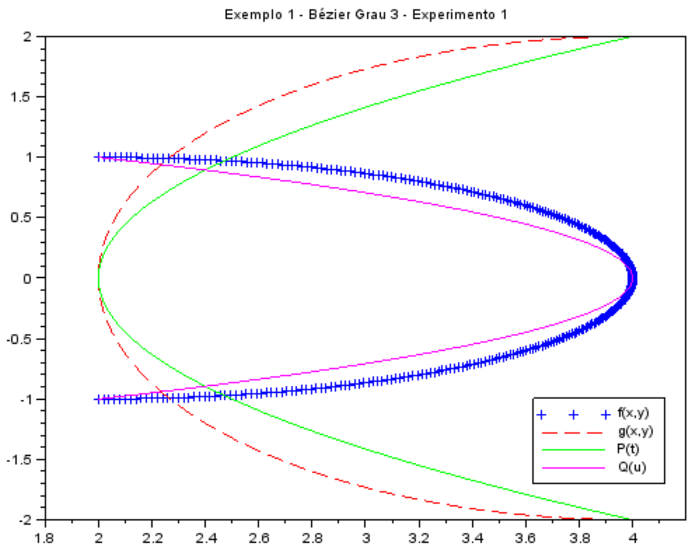

Figura 4.2: Curvas de Bézier $P(t)$ e $Q(u)$ no gráfico do exemplo 1

Portanto, já foram encontrados os pontos de controle para as curvas cúbicas $P(t)$ e $Q(u)$.

b) Aplicação do algoritmo de Bézier Clipping:

No algoritmo de Bézier Clipping, deve-se escolher uma das curvas para inicialização. Como a escolha é arbitrária, aqui o algoritmo será inicializado através de $Q(u)$, para fazer o recorte da curva $P(t)$. As etapas a serem seguidas para o recorte são:

1. A *Fat Line* é a região que se situa entre os segmentos de reta paralelos que correspondem ao segmento dos pontos extremos da curva $Q(u)$ e ao segmento por onde passa o ponto de controle intermediário com distância máxima (ou mínima) do segmento dos pontos extremos.

   Os coeficientes a, b, c da reta dos extremos $ax + by + c = 0$ será dada por:

$a = y_0 - y_3 = 4$

$b = x_3 - x_0 = 0$

$c = x_0 y_3 - x_3 y_0 = -16$

Portanto, a equação da reta que une os pontos extremos da curva será dada por:

$4x - 16 = 0$ ou $x - 4 = 0$.

2. As distâncias dos pontos de controle intermediários à reta que conecta pontos extremos são dadas por:

   $d_1 = -2.6666667;$

   $d_2 = -2.6666667;$

   No algoritmo de Bézier Clipping, conforme está descrito no capítulo 3, quando $d_1 d_2 > 0$, as distâncias máxima e mínima são dadas por:

   $d_{min} = \frac{3}{4} min(0, d_1, d_2) = \frac{3}{4}(-2.6666667) = -2$

   $d_{max} = \frac{3}{4} max(0, d_1, d_2) = 0$

   Logo, a Fat Line será a região localizada entre os segmentos de reta $x - 4 = 0$ e $x - 4 - d_{min} = x - 2 = 0$.

3. Agora, é necessário calcular as distâncias dos pontos de controle da curva que vai ser recortada, no caso a curva $P(t)$, a reta que une os pontos extremos de $Q(u)$.

   Neste caso, temos:

   $d_0 = -2; d_1 = 0.6666667; d_2 = 0.6666667; d_3 = -2.$

   Os valores dessas distâncias são interpolados através de uma curva de Bézier $D(t)$, de grau 3, cujos pontos de controle serão dados por:

   $(0, d_0), (1/3, d_1), (2/3, d_2), (1, d_3).$

   Assim,

$D(t) = (1-t)^3 d_0 + 3(1-t)^2 t d_1 + 3(1-t) t^2 d_2 + t^3 d_3.$

4. Ao final da primeira iteração, realizam-se as interseções de $D(t)$ com d_{min} e d_{max}.

   As raízes de $D(t) - d_{min} = 0$ são $t = 0$ e $t = 1$.

   As raízes de $D(t) - d_{max} = 0$ são $t = 0.4999441$ e $t = 0.5000559$.

   Logo, os valores mínimo e máximo encontrados de t no intervalo $[0, 1]$ são:

   $t_{min} = 0$ e $t_{max} = 1$.

Repare que o intervalo da curva $P(t)$ não foi alterado. De acordo com a heurística de (NISHITA ; SEDERBERG, 1990) vista no capítulo 3, isto significa que não houve recorte na curva e quando isso acontece, existem interseções múltiplas. Neste exemplo, são duas interseções. Portanto, será necessário uma subdivisão das curvas $P(t)$ e $Q(u)$.

As novas subdivisões de $P(t)$ serão dadas por $P_1(t)$ e $P_2(t)$, enquanto as novas subdivisões de $Q(u)$ serão dadas por $Q_1(u)$ e $Q_2(u)$.

A curva $P_1(t)$ é uma aproximação de $f(x, y) = 0$ quando as parametrizações da curva algébrica são dadas por: $x(\theta) = 2 + 2cos(\theta)$, $y(\theta) = sen(\theta)$, $\theta \in [0, \frac{\pi}{2}]$.

Os pontos extremos da curva $P_1(t)$ ocorrem quando $\theta_0 = \frac{\pi}{2}$ e $\theta_f = 0$. Já os pontos extremos da curva $P_2(t)$ ocorrem quando $\theta_0 = \frac{-\pi}{2}$ e $\theta_f = 0$.

Repetindo-se o processo visto anteriormente, os seguintes pontos de controle são encontrados:

Para $P_1(t)$:

$P_0 = (2, 1);$
$P_1 = (3.2189515, 0.9428091);$
$P_2 = (3.8856181, 0.6094757);$
$P_3 = (4, 0).$

Para $P_2(t)$:

$P_0 = (2, -1);$
$P_1 = (3.2189515, -0.9428091);$
$P_2 = (3.8856181, -0.6094757);$
$P_3 = (4, 0).$

Os pontos extremos da curva $Q_1(u)$ ocorrem quando $\theta_0 = \pi$ e $\theta_f = \frac{\pi}{2}$. Já os pontos extremos da curva $Q_2(u)$ ocorrem quando $\theta_0 = \pi$ e $\theta_f = \frac{3\pi}{2}$.

Analogamente,

Para $Q_1(u)$:

$Q_0 = (2, 0);$
$Q_1 = (2.1143819, 1.2189515);$
$Q_2 = (2.7810485, 1.8856181);$
$Q_3 = (4, 2).$

Para $Q_2(u)$:

$Q_0 = (2, 0);$
$Q_1 = (2.1143819, -1.2189515);$
$Q_2 = (2.7810485, -1.8856181);$

$Q_3 = (4, -2)$.

Agora, o algoritmo de Bézier Clipping é aplicado para $P_1(t)$ e $Q_1(u)$. O gráfico do programa em Scilab para essas curvas é mostrado na figura 4.3. O código está listado no apêndice B.

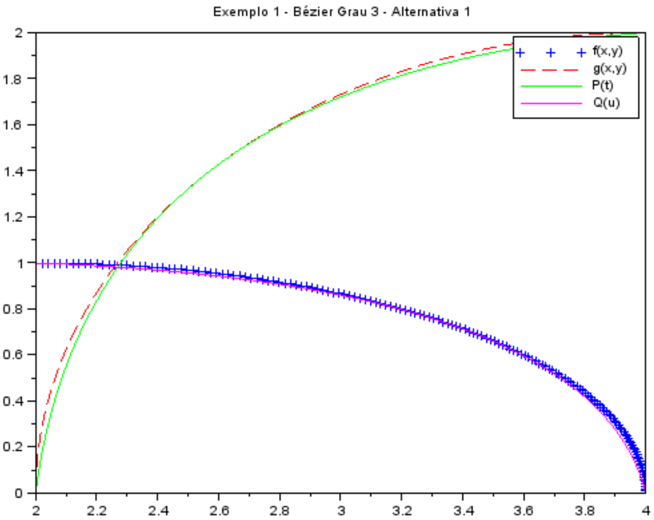

Figura 4.3: Gráfico do exemplo 1 nos intervalos $x \in [2, 4]$ e $y \geq 0$.

Começando o algoritmo pela curva $Q_1(u)$, tem-se:

1. Para a *Fat Line*:

   Os coeficientes a, b, c da reta dos extremos $ax + by + c = 0$ será dada por:

   $a = y_0 - y_3 = -2$

   $b = x_3 - x_0 = 2$

   $c = x_0 y_3 - x_3 y_0 = 4$

   Portanto, a equação da reta que une os pontos extremos da curva será dada por:

$-2x + 2y + 4 = 0$ ou $-x + y + 2 = 0$.

2. As distâncias dos pontos de controle intermediários são dadas por:

   $d_1 = 0.7810487$;

   $d_2 = 0.7810487$;

   No algoritmo de Bézier Clipping, quando $d_1 d_2 > 0$, o cálculo das distâncias máxima e mínima são dadas por:

   $d_{min} = \frac{3}{4} min(0, d_1, d_2) = 0$

   $d_{max} = \frac{3}{4} max(0, d_1, d_2) = \frac{3}{4}(0.7810487) = 0.5857865$

   Logo, a Fat Line será a região localizada entre os segmentos de reta $-x+y+2 = 0$ e $-x + y + 2 - d_{max} = -x + y - 1.4142135 = 0$.

3. Agora, é necessário calcular as distâncias dos pontos de controle da curva que vai ser recortada, no caso a curva $P_1(t)$, a reta que une os pontos extremos de $Q_1(u)$.

   Neste caso, temos:

   $d_0 = 0.7071068; d_1 = -0.1952622; d_2 = -0.9023689; d_3 = -1.4142136$.

   Os valores dessas distâncias são interpolados através de uma curva de Bézier $D_1(t)$, de grau 3, cujos pontos de controle serão dados por:

   $(0, d_0), (1/3, d_1), (2/3, d_2), (1, d_3)$.

   Assim,

   $D_1(t) = (1-t)^3 d_0 + 3(1-t)^2 t d_1 + 3(1-t) t^2 d_2 + t^3 d_3$.

4. Ao final da primeira iteração, realizam-se as interseções de $D_1(t)$ com d_{min} e d_{max}.

   A raiz de $D_1(t) - d_{min} = 0$ é $t = 0.2779173$.

   A raiz de $D_1(t) - d_{max} = 0$ é $t = 0.0452587$.

Logo, os valores mínimo e máximo encontrados de t no intervalo $[0, 1]$ são:

$t_{min} = 0.0452587$ e $t_{max} = 0.2779173$.

Substituindo estes valores na equação da curva $P_1(t)$, tem-se:

$P_1(t = 0.0452587) = (0.21621107, 0.9905379)$;

$P_1(t = 0.2779173) = (0.28883309, 0.8883309)$

Logo, a curva $Q_1(u)$ faz um recorte em $P(t)$ nos pontos $(0.21621107, 0.9905379)$ e $(0.28883309, 0.8883309)$, o que significa que uma das interseções, em $y \geq 0$, se encontra entre os valores máximo e mínimo para a coordenada horizontal nesses pontos.

Depois, repete-se o algoritmo aplicando o recorte de $Q_1(u)$ através de $P_1(t)$, em seguida de $P_1(t)$ através de $Q_1(u)$, e assim sucessivamente, alternando as curvas de recorte até que a execução do algoritmo pare segundo uma determinada tolerância.

A Tabela 4.1 mostra os resultados para quatro iterações. O tempo de execução verificado no Scilab foi de 10.054 segundos. A solução analítica, encontrada através do cálculo da resultante no Scilab, é $(2.2629657, 0.9913184)$. O código do programa em Scilab que gera os cálculos está no apêndice B.

Tabela 4.1: Exemplo 1 - Resultados para quatro iterações do Bézier Clipping

Iteração	Pontos extremos	Erro
1	(2.1621110,0.9905379);(2.888331,0.8883309)	0.625365200
2	(2.2663630,0.9758657);(2.274207,0.9907314)	0.015452740
3	(2.2696290,0.9820734);(2.269638,0.9820726)	0.009245844
4	(2.2696290,0.9820733);(2.269629,0.9820733)	0.009245062

Uma observação importante: para todos os exemplos, foi utilizada a norma do máximo para o cálculo do erro.

c) Aplicações da subdivisão recursiva de De Casteljau e da subdivisão intervalar:

A seguir, as tabelas 4.2 e 4.3 mostram os resultados para dez iterações dos pontos extremos obtidos para as sub-curvas de $P(t)$ e $Q(u)$ com os métodos de subdivisão recursiva de De Casteljau e subdivisão intervalar de Koparkar e Mudur, os quais coincidiram neste exemplo. O tempo de execução verificado no Scilab foi de 0.511 segundos. Os códigos para os programas que geraram os cálculos estão nos apêndices C e D.

Tabela 4.2: Exemplo 1 - Iterações para $P(t)$ - De Casteljau e Intervalar

Iteração	Pontos extremos de P(t)	Erro
1	(2.0000000,1.0000000);(3.414214,0.7071068)	1.1512483
2	(2.0000000,1.0000000);(2.810660,0.9053300)	0.5476943
3	(2.0000000,1.0000000);(2.431220,0.9656090)	0.2629657
4	(2.2220800,0.9860410);(2.431220,0.9656090)	0.1682543
5	(2.2220800,0.9860410);(2.275580,0.9815400)	0.0408857
6	(2.2489300,0.9838410);(2.275580,0.9815400)	0.0140357
7	(2.2622800,0.9827030);(2.275580,0.9815400)	0.0126143
8	(2.2622800,0.9827030);(2.268940,0.9821240)	0.0091944
9	(2.2622800,0.9827030);(2.265610,0.9824140)	0.0089044
10	(2.2622800,0.9827030);(2.263950,0.9825590)	0.0087594

Tabela 4.3: Exemplo 1 - Iterações para $Q(u)$ - De Casteljau e Intervalar

Iteração	Pontos extremos de Q(u)	Erro
1	(2.0000000,0.0000000);(2.5857860,1.4142140)	0.9913184
2	(2.1893400,0.8106602);(2.5857860,1.4142140)	0.4228956
3	(2.1893400,0.8106602);(2.3616750,1.1383250)	0.1806582
4	(2.1893400,0.8106602);(2.2690350,0.9809648)	0.1806582
5	(2.2275690,0.8974305);(2.2690350,0.9809648)	0.0938879
6	(2.2478980,0.9396022);(2.2690350,0.9809648)	0.0517162
7	(2.2583650,0.9603846);(2.2690350,0.9809648)	0.0309338
8	(2.2583650,0.9603846);(2.2636750,0.9707000)	0.0309338
9	(2.2610140,0.9655486);(2.2636750,0.9707000)	0.0257698
10	(2.2623430,0.9681259);(2.2636750,0.9707000)	0.0231925

4.4.1.2 Experimento 2

a) Interpolação das curvas de Bézier de grau 3:

Agora, as interseções entre as curvas $P(t)$ e $Q(u)$ serão calculadas através da segunda alternativa da seção 4.3, cujo objetivo é realizar a interpolação dos pontos das curvas $f(x,y) = 0$ e $g(x,y) = 0$ através das respectivas curvas de Bézier cúbicas $P(t)$ e $Q(u)$.

O intervalo na coordenada horizontal será dividido em número de subintervalos equidistantes, cuja quantidade corresponde ao grau da curva de Bézier aproximada. Neste exemplo, o intervalo $[2,4]$ será dividido em 3 partes.

Então, a matriz M das funções de base de Bernstein, os vetores VPX, VPY, correspondentes aos valores de x e y na equação $f(x,y) = 0$, e os vetores VQX, VQY, correspondentes aos valores de x e y na equação $g(x,y) = 0$, descritos na seção 4.3.2, neste exemplo serão dados por:

$$M = \begin{bmatrix} B_0^3(0) & B_1^3(0) & B_2^3(0) & B_3^3(0) \\ B_0^3(\frac{1}{3}) & B_1^3(\frac{1}{3}) & B_2^3(\frac{1}{3}) & B_3^3(\frac{1}{3}) \\ B_0^3(\frac{2}{3}) & B_1^3(\frac{2}{3}) & B_2^3(\frac{2}{3}) & B_3^3(\frac{2}{3}) \\ B_0^3(1) & B_1^3(1) & B_2^3(1) & B_3^3(1) \end{bmatrix} =$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.2962963 & 0.4444444 & 0.2222222 & 0.0370370 \\ 0.0370370 & 0.2222222 & 0.4444444 & 0.2962963 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$VPX = \begin{bmatrix} VPX_0 \\ VPX_1 \\ VPX_2 \\ VPX_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 8/3 \\ 10/3 \\ 4 \end{bmatrix} = \begin{bmatrix} 2 \\ 2.6666667 \\ 3.3333333 \\ 4 \end{bmatrix}$$

Ao substituir os valores de VPX na coordenada x da equação $f(x, y) = 0$, é obtido:

$$VPY = \begin{bmatrix} VPY_0 \\ VPY_1 \\ VPY_2 \\ VPY_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.9428090 \\ 0.7453560 \\ 0 \end{bmatrix}$$

$$VQX = \begin{bmatrix} VQX_0 \\ VQX_1 \\ VQX_2 \\ VQX_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 8/3 \\ 10/3 \\ 4 \end{bmatrix} = \begin{bmatrix} 2 \\ 2.6666667 \\ 3.3333333 \\ 4 \end{bmatrix}$$

Ao substituir os valores de VQX na coordenada x da equação $g(x, y) = 0$, é obtido:

$$VQY = \begin{bmatrix} VQY_0 \\ VQY_1 \\ VQY_2 \\ VQY_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1.4907120 \\ 1.8856181 \\ 2 \end{bmatrix}$$

Para os pontos de controle de $P(t)$, usaremos para as coordenadas horizontais e verticais os vetores PX e PY, cujos cálculos são feitos a partir dos seguintes sistemas lineares:

$$M(PX) = VPX \Rightarrow PX = M^{-1}(VPX) \Rightarrow PX = \begin{bmatrix} 2 \\ 2.6666667 \\ 3.3333333 \\ 4 \end{bmatrix}$$

$$M(PY) = VPY \Rightarrow PY = M^{-1}(VPY) \Rightarrow PY = \begin{bmatrix} 1 \\ 0.8770598 \\ 1.1551877 \\ 0 \end{bmatrix}$$

Analogamente, para os pontos de controle de $Q(u)$, usaremos para as coordenadas horizontais e verticais os vetores QX e QY, cujos cálculos são feitos a partir dos seguintes sistemas lineares:

$$M(QX) = VQX \Rightarrow QX = M^{-1}(VQX) \Rightarrow QX = \begin{bmatrix} 2 \\ 2.6666667 \\ 3.3333333 \\ 4 \end{bmatrix}$$

$$M(QY) = VQY \Rightarrow QY = M^{-1}(VQY) \Rightarrow QY = \begin{bmatrix} 0 \\ 2.3103755 \\ 1.7541196 \\ 2 \end{bmatrix}$$

Portanto, os pontos de controle usados neste caso serão:

Para a curva $P(t)$: $P_0 = (2, 1)$;
$P_1 = (2.6666667, 0.8770598)$;
$P_2 = (3.3333333, 1.1551877)$;
$P_3 = (4, 0)$.

Para a curva $Q(u)$: $Q_0 = (2, 0)$;
$Q_1 = (2.6666667, 2.3103755)$;
$Q_2 = (3.3333333, 1.7541196)$;
$Q_3 = (4, 2)$.

O gráfico para o comparativo entre as curvas originais e as curvas de Bézier de grau 3 geradas neste experimento são mostrados na Figura 4.4

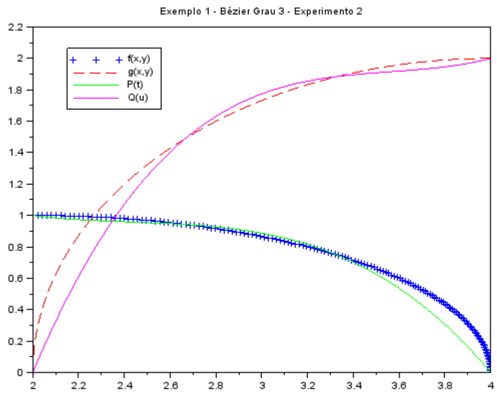

Figura 4.4: Gráfico para o experimento 2 do exemplo 1 nos intervalos $x \in [2, 4]$ e $y \geq 0$.

b) Aplicação do Bézier Clipping:

Aplicando o algoritmo de Bézier Clipping aos pontos anteriores, são encontradas as iterações que estão na Tabela 4.4. O tempo de execução verificado no Scilab foi de 9.543 segundos.

Deve-se observar que no caso em que a curva de Bézier possui grau 3, o algoritmo de Bézier Clipping executado através dos pontos de controle originais obtidos pela interpolação de Bézier apresentou erros absolutos máximos maiores que os vistos com parametrização e elevação de grau.

Tabela 4.4: Exemplo 1 - Iterações para Bézier Clipping usando Interpolação de Bézier da Seção 4.3 e Curva de Bézier de grau 3

Iteração	Pontos extremos	Curva	Erro
1	(2.2521710,0.7445511);(3.0775940,1.8117480)	$P(t)$	0.8204299
2	(2.2330210,0.9704604);(2.3936070,0.9600343)	$Q(u)$	0.1306411
3	(2.3533160,0.9428743);(2.4171010,1.0491940)	$P(t)$	0.1541355
4	(2.3642120,0.9616593);(2.3647500,0.9616289)	$Q(u)$	0.1017844
5	(2.3615240,0.9568654);(2.3722070,0.9748305)	$P(t)$	0.1092415
6	(2.3643630,0.9616508);(2.3643700,0.9616504)	$Q(u)$	0.1014042
7	(2.3638890,0.9608505);(2.3653070,0.9632363)	$P(t)$	0.1023415
8	(2.3643650,0.9616507);(2.3643650,0.9616507)	$Q(u)$	0.1013992
9	(2.3643140,0.9615651);(2.3644520,0.9617977)	$P(t)$	0.1014865
10	(2.3643650,0.9616507);(2.3643650,0.9616507)	$Q(u)$	0.1013991

4.4.1.3 Experimento 3

a) Interpolação de curvas de Bézier de grau 9

Se os pontos da curva original forem interpolados com uma curva de Bézier de grau 9, os resultados irão convergir para a solução analítica com erros absolutos máximos menores que os vistos para a curva de grau 3. Evidentemente, quanto maior o grau, maior será o número de pontos na interpolação e consequentemente melhor será a aproximação da curva.

Os pontos de controle usados neste caso serão:

Para a curva $P(t)$:

$P_0 = (2, 1);$
$P_1 = (2.2222222, 0.9815233);$
$P_2 = (2.4444444, 1.0622518);$
$P_3 = (2.6666667, 0.7713426);$

$P_4 = (2.8888889, 1.2489973);$
$P_5 = (3.1111111, 0.3946336);$
$P_6 = (3.3333333, 1.2881787);$
$P_7 = (3.5555556, 0.2043762);$
$P_8 = (3.7777778, 0.8701008);$
$P_9 = (4, 0).$

Para a curva $Q(u)$:

$Q_0 = (2, 0);$
$Q_1 = (2.2222222, 1.7402016);$
$Q_2 = (2.4444444, 0.4087525);$
$Q_3 = (2.6666667, 2.5763574);$
$Q_4 = (2.8888889, 0.7892673);$
$Q_5 = (3.1111111, 2.4979946);$
$Q_6 = (3.3333333, 1.5426852);$
$Q_7 = (3.5555556, 2.1245036);$
$Q_8 = (3.7777778, 1.9630467);$
$Q_9 = (4, 2).$

O gráfico para o comparativo entre as curvas originais e as curvas de Bézier de grau 9 nos intervalos dados são mostrados na Figura 4.5

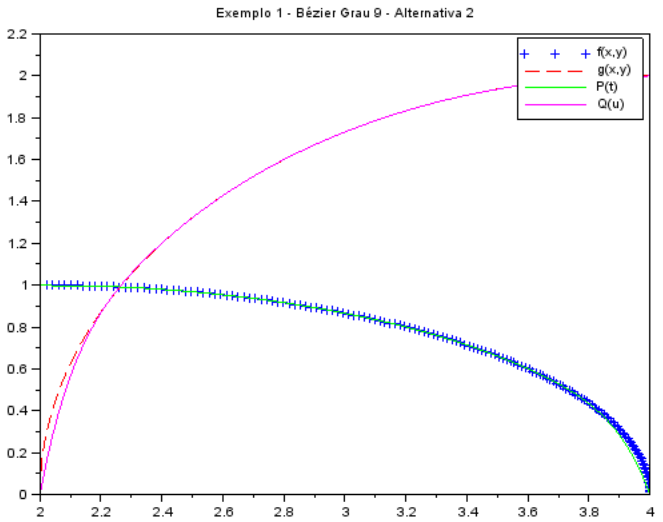

Figura 4.5: Gráfico do exemplo 1 nos intervalos $x \in [2, 4]$ e $y \geq 0$.

b) Aplicação do Bézier Clipping:

Os resultados do algoritmo de Bézier Clipping para interpolação através de uma curva de Bézier de grau 9 são mostrados na tabela 4.5. O tempo de execução verificado no Scilab foi de 37.305 segundos.

Tabela 4.5: Exemplo 1 - Iterações para Bézier Clipping usando Interpolação de Bézier da Seção 4.3 e Curva de Bézier de grau 9

Iteração	Pontos extremos	Curva	Erro
1	(2.1140560,0.6192522);(3.3032300,1.8747840)	$P(t)$	1.040265000
2	(2.1018140,0.9967448);(2.4842670,0.9701483)	$Q(u)$	0.221300800
3	(2.2153170,0.9016344);(2.4549860,1.2698230)	$P(t)$	0.278505000
4	(2.2512900,0.9923145);(2.2739960,0.9909161)	$Q(u)$	0.011675900
5	(2.2595660,0.9887561);(2.2734730,1.0133980)	$P(t)$	0.022079710
6	(2.2611440,0.9917321);(2.2612440,0.9917260)	$Q(u)$	0.001821836
7	(2.2612060,0.9917210);(2.2612420,0.9917863)	$P(t)$	0.001759686
8	(2.2612100,0.9917280);(2.2612100,0.9917280)	$Q(u)$	0.001755815
9	(2.2612100,0.9917280);(2.2612100,0.9917284)	$P(t)$	0.001755815

c) Aplicação da subdivisão recursiva de De Casteljau e da subdivisão intervalar:

Agora, os resultados para Subdivisão Recursiva de De Casteljau nas Tabelas 4.6 e 4.7, cujo tempo de execução verificado foi de 0.98 segundos:

Tabela 4.6: Exemplo 1 - Iterações para $P(t)$ - Subdivisão Recursiva de De Casteljau para curva de grau 9

Iteração	Pontos extremos de P(t)	Erro
1	(2.0000000,1.0000000);(3.0000000,0.8659385)	0.7370343
2	(2.0000000,1.0000000);(2.5000000,0.9681276)	0.2629657
3	(2.2500000,0.9923879);(2.5000000,0.9681276)	0.2370343
4	(2.2500000,0.9923879);(2.3750000,0.9825049)	0.1120343
5	(2.2500000,0.9923879);(2.3125000,0.9881072)	0.0495343
6	(2.2500000,0.9923879);(2.2812500,0.9904279)	0.0182843
7	(2.2500000,0.9923879);(2.2656250,0.9914547)	0.0129657
8	(2.2578130,0.9919332);(2.2656250,0.9914547)	0.0051532
9	(2.2617190,0.9916969);(2.2656250,0.9914547)	0.0026593
10	(2.2617190,0.9916969);(2.2636720,0.9915765)	0.0012470

Tabela 4.7: Exemplo 1 - Iterações para $Q(u)$ - Subdivisão Recursiva de De Casteljau para curva de grau 9

Iteração	Pontos extremos de Q(u)	Erro
1	(2.0000000,0.0000000);(3.0000000,1.7318770)	0.9913184
2	(2.0000000,0.0000000);(2.5000000,1.3222020)	0.9913184
3	(2.2500000,0.9711243);(2.5000000,1.3222020)	0.3308837
4	(2.2500000,0.9711243);(2.3750000,1.1678520)	0.1765339
5	(2.2500000,0.9711243);(2.3125000,1.0773050)	0.0859865
6	(2.2500000,0.9711243);(2.2812500,1.0267100)	0.0353919
7	(2.2500000,0.9711243);(2.2656250,0.9996296)	0.0201941
8	(2.2578130,0.9855676);(2.2656250,0.9996296)	0.0083112
9	(2.2617190,0.9926447);(2.2656250,0.9996296)	0.0083112
10	(2.2617190,0.9926447);(2.2636720,0.9961485)	0.0048301

Agora, os resultados para Subdivisão Intervalar para curva de Bézier de grau 9 nas Tabelas 4.8 e 4.9:

Tabela 4.8: Exemplo 1 - Iterações para $P(t)$ - Subdivisão Intervalar para curva de grau 9

Iteração	Pontos extremos de P(t)	Erro
1	(2.0000000,1.0000000);(2.3333333,0.9878335)	0.2629657
2	(2.1666670,0.9973865);(2.3333333,0.9878335)	0.0962990
3	(2.2500000,0.9962363);(2.3333333,0.9878335)	0.0703676
4	(2.2500000,0.9962363);(2.2916670,0.9932863)	0.0287010
5	(2.2604170,0.9956957);(2.2708330,0.9950311)	0.0078676
6	(2.2604170,0.9956957);(2.2656250,0.9953796)	0.0043773
7	(2.2604170,0.9956957);(2.2630210,0.9955416)	0.0043773
8	(2.2617190,0.9956197);(2.2630210,0.9955416)	0.0043013
9	(2.2623700,0.9955809);(2.2630210,0.9955416)	0.0042625
10	(2.2626950,0.9955613);(2.2630210,0.9955416)	0.0042429

Tabela 4.9: Exemplo 1 - Iterações para $Q(u)$ - Subdivisão Intervalar para curva de grau 9

Iteração	Pontos extremos de Q(u)	Erro
1	(2.0000000,0.0000000);(2.3333333,1.1279020)	0.9913184
2	(2.1666667,0.8318840);(2.3333333,1.1279020)	0.1594344
3	(2.2500000,1.0083760);(2.3333333,1.1279020)	0.1365841
4	(2.2500000,1.0083760);(2.2916670,1.0704480)	0.0791291
5	(2.2500000,1.0083760);(2.2708330,1.0405900)	0.0492721
6	(2.2604170,1.0248530);(2.2708330,1.0405900)	0.0492721
7	(2.2604170,1.0248530);(2.2656250,1.0328050)	0.0414865
8	(2.2604170,1.0248530);(2.2630210,1.0288510)	0.0375326
9	(2.2617190,1.0268580);(2.2630210,1.0288510)	0.0375326
10	(2.2623700,1.0278560);(2.2630210,1.0288510)	0.0375326

4.4.2 Exemplo 2

Encontrar as interseções entre as curvas algébricas planas:

$$f(x, y) = x^2 + 4y^2 - 4 = 0$$

$$g(x, y) = x^{15} + 3x^{10}y^2 + 3x^5y^4 + y^6 - 1 = 0$$

nos intervalos $x \in [-0.5, 0]$, $y \geq 0$.

No cálculo analítico, a resultante $R_{f,g}$ fica:

$$R_{f,g} = \begin{vmatrix} 1 & 0 & x^2+4x & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & x^2+4x & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & x^2+4x & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & x^2+4x & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & x^2+4x & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & x^2+4x \\ 1 & 0 & 3x^5 & 0 & 3x^{10} & 0 & x^{15}-1 & 0 \\ 0 & 1 & 0 & 3x^5 & 0 & 3x^{10} & 0 & x^{15}-1 \end{vmatrix}$$

Como $R_{f,g} = 1 + 128x^3 + 96x^4 + 24x^5 + 4098x^6 + 6048x^7 + 3792x^8 + 1274x^9 - 5904x^{10} - 7632x^{11} - 3833x^{12} - 960x^{13} + 3720x^{14} + 3832x^{15} + 1440x^{16} + 240x^{17} - 1265x^{18} - 960x^{19} - 240x^{20} - 20x^{21} + 240x^{22} + 120x^{23} + 15x^{24} - 24x^{26} - 6x^{27} + x^{30}$, ou seja, não é um polinômio identicamente nulo, então $mdc(f, g) = 1$ e assim, de acordo com o Teorema de Bézout, f e g não possuem fatores comuns. Logo, f e g possuem no máximo $grau(f) \cdot grau(g) = 2 \cdot 15 = 30$ interseções.

O número de interseções de f e g é igual a 4, como mostra a Figura 4.6, o que está de acordo com o Teorema de Bézout.

Resolvendo-se a equação $R_{f,g} = 0$, são encontradas as seguintes soluções analíticas:

$(1.1848949, 1.8263635); (1.1848949, -1.8263635);$

$(-0.2683510, 1.0006956); (-0.2683510, -1.0006956).$

Para as soluções numéricas através dos algoritmos do capítulo 3, foram realizados os seguintes experimentos:

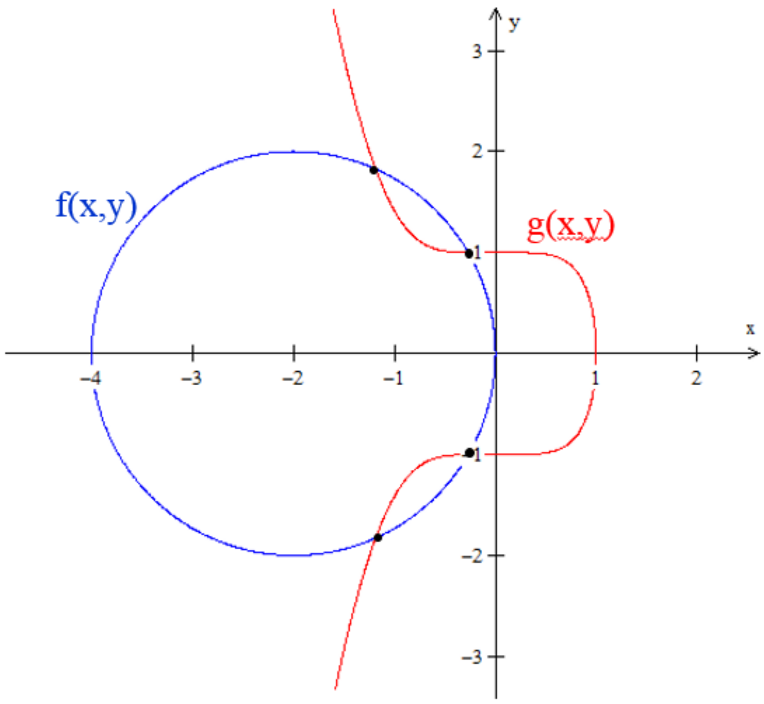

Figura 4.6: Gráfico do exemplo 2

4.4.2.1 Experimento

a) Parametrização com elevação de grau:

Deve-se aproximar primeiro a função f, que é uma circunferência, no intervalo dado por uma curva de Bézier $P(t), 0 \leq t \leq 1$, de grau 2, e em seguida, por uma curva de Bézier de grau 3.

Deve-se levar em consideração que a curva não altera sua concavidade no intervalo dado. Por isso, foi possível a aproximação inicial pela curva de Bézier de grau 2.

Uma parametrização para f é: $x(\theta) = 2cos(\theta) - 2$, $y(\theta) = 2sen(\theta)$.

Os pontos extremos da curva ocorrem quando $\theta_0 = 0.7227342$ e $\theta_1 \approx 0$.

Neste caso, os pontos extremos da curva de grau 2 são: $P_0 = (-0.5, 1.3228756)$ e $P_2 = (0, 0)$.

A metade da curva ocorre quando $\theta = \frac{1}{2}(\theta_0 + \theta_1) \approx 0.3613671$.

Neste caso, $x(\theta = 0.3613671) = -0.1291713$, $y(\theta = 0.3613671) = 0.7071067$.

Então, $P(t = 0.5) = (-0.1291713, 0.7071067)$.

Para encontrar P_1, basta resolver a seguinte equação para curva de Bézier de grau 2 quando $t = 0.5$:

$$P(t = 0.5) = 0.25(-0.5, 1.3228756) + 0.5P_1 + 0.25(0, 0)$$

Assim, os pontos encontrados para curva de Bézier de grau 2 são:

$$P_0 = (-0.5, 1.3228756); P_1 = (-0.0083426, 0.7527756); P_2 = (0, 0).$$

Para achar os pontos para curva de Bézier de grau 3, usa-se a técnica de elevação de grau, onde os novos pontos serão dados segundo a equação:

$$P_i = \left(\frac{i}{n+1}\right) P_{i-1} + \left(1 - \frac{i}{n+1}\right) P_i, 1 \leq i \leq n,$$

onde n é o grau anterior da curva.

Neste exemplo, n = 3.

Usando essa técnica, achamos os seguintes pontos de controle de grau 3 para $P(t)$:

$P_0 = (-0.5, 1.3228756); P_1 = (-0.1722284, 0.9428089);$

$P_2 = (-0.0055617, 0.5018504), P_3 = (0, 0).$

Para a função g, os pontos extremos de uma curva de Bézier $Q(u)$ associada de grau 2 ocorrem em $Q_0 = (-0.5, 1.0155048)$ e $Q_1 = (0, 1)$.

Na metade do intervalo $x \in [-0.5, 1]$, ou seja, quando escolhe-se como candidato $x = -0.25$, encontra-se o ponto central $Q_1 = (-0.25, 1.0004882)$.

Analogamente ao que foi feito na curva $P(t)$, os pontos de controle encontrados para $Q(u)$ por elevação para o grau 3 são:

$Q_0 = (-0.5, 1.0155048); Q_1 = (-0.3333333, 1.0054896);$

$Q_2 = (-0.1666667, 1.0003213), Q_3 = (0, 1).$

A Figura 4.7 mostra os gráficos plotados no Scilab para o exemplo:

b) Aplicação de Bézier Clipping de grau 3:

A Tabela 4.10 mostra os resultados para três iterações e eles convergem para a solução analítica $(-0.2683510, 1.0006956)$. O tempo de execução verificado foi de 10.313 segundos.

c) Aplicações da subdivisão recursiva de De Casteljau e da subdivisão inter-

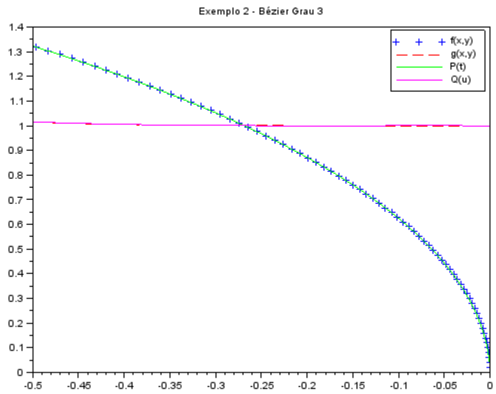

Figura 4.7: Gráfico do exemplo 2 no intervalo $x \in [-0.5, 0]$ e $y \geq 0$.

Tabela 4.10: Exemplo 2 - Resultados para três iterações do Bézier Clipping

Iteração	Pontos extremos	Erro
1	(-0.3795637,1.004231);(-0.239927,1.0001390)	0.1112127
2	(-0.2693283,1.001000);(-0.2687349,0.999981)	0.0009773
3	(-0.2690087,1.000451);(-0.2690086,1.000451)	0.0006577

valar:

A seguir, as Tabelas 4.11 e 4.12 mostram os resultados para dez iterações dos pontos extremos obtidos para as sub-curvas de $P(t)$ e $Q(u)$ com os métodos de subdivisão recursiva de De Casteljau e subdivisão intervalar de Koparkar e Mudur, os quais coincidiram neste exemplo.

Tabela 4.11: Exemplo 2 - Iterações para $P(t)$ - De Casteljau e Intervalar

Iteração	Pontos extremos de P(t)	Erro
1	(-0.500000,1.322880);(-0.129171,0.707107)	0.3221844
2	(-0.284378,1.026410);(-0.129171,0.707107)	0.2935886
3	(-0.284378,1.026410);(-0.199223,0.869612)	0.1310836
4	(-0.284378,1.026410);(-0.239913,0.948724)	0.0519716
5	(-0.284378,1.026410);(-0.261674,0.987744)	0.0257144
6	(-0.272908,1.007120);(-0.261674,0.987744)	0.0066770
7	(-0.272908,1.007120);(-0.267261,0.997444)	0.0064244
8	(-0.270077,1.002290);(-0.267261,0.997444)	0.0032516
9	(-0.268668,0.999865);(-0.267261,0.997444)	0.0032516
10	(-0.268668,0.999865);(-0.267964,0.998655)	0.0020406

Tabela 4.12: Exemplo 2 - Iterações para $Q(u)$ - De Casteljau e Intervalar

Iteração	Pontos extremos de Q(u)	Erro
1	(-0.500000,1.015500);(-0.250000,1.004120)	0.231649
2	(-0.375000,1.008900);(-0.250000,1.004120)	0.106649
3	(-0.312500,1.006280);(-0.250000,1.004120)	0.044149
4	(-0.281250,1.005140);(-0.250000,1.004120)	0.018351
5	(-0.281250,1.005140);(-0.265625,1.004620)	0.012899
6	(-0.273437,1.004880);(-0.265625,1.004620)	0.005086
7	(-0.269531,1.004750);(-0.265625,1.004620)	0.004054
8	(-0.269531,1.004750);(-0.267578,1.004680)	0.004054
9	(-0.268555,1.004710);(-0.267578,1.004680)	0.004014
10	(-0.268555,1.004710);(-0.268066,1.004700)	0.004014

Até aqui, foram tratadas as funções polinomiais $f : \mathbb{R}^2 \to \mathbb{R}$. O próximo exemplo mostra como estes algoritmos podem ser também utilizados para o cálculo de raízes reais de funções não necessariamente polinomiais $f : \mathbb{R} \to \mathbb{R}$.

4.4.3 Exemplo 3

Resolver a equação $sen(x) = e^{-x}$ no intervalo $[0, \pi/4]$.

4.4.3.1 Experimento

Aqui, foi usada a parametrização com elevação de grau e posterior aplicação dos algoritmos vistos no capítulo 3.

a) Parametrização com elevação de grau:

Aqui, as funções $f(x) = sen(x)$ e $g(x) = e^{-x}$ são aproximadas no intervalo $[0, \pi/4]$ por suas respectivas curvas de Bézier de grau 3, $P(t), 0 \leq t \leq 1$, e $Q(u), 0 \leq u \leq 1$. Depois, usando os mesmos procedimentos do Exemplo 1, os seguintes pontos de controle são encontrados:

Curva $P(t)$:

$P_0 = (0, 0); P_1 = (0.2617994, 0.2745423);$
$P_2 = (0.5235988, 0.5102445); P_3 = (0.7853982, 0.7071068).$

Curva $Q(u)$:

$Q_0 = (0, 1); Q_1 = (0.2617994, 0.7483298);$
$Q_2 = (0.5235988, 0.5669758); Q_3 = (0.7853982, 0.4559381).$

O gráfico deste exemplo desenhado com o Scilab 5.4.0 pode ser visto na Figura 4.8.

b) Aplicação dos algoritmos de interseção:

As Tabelas 4.13, 4.14 e 4.15 a seguir mostram os resultados para o Exemplo 3. O algoritmo de Bézier Clipping foi executado em 12.169 segundos. Há conver-

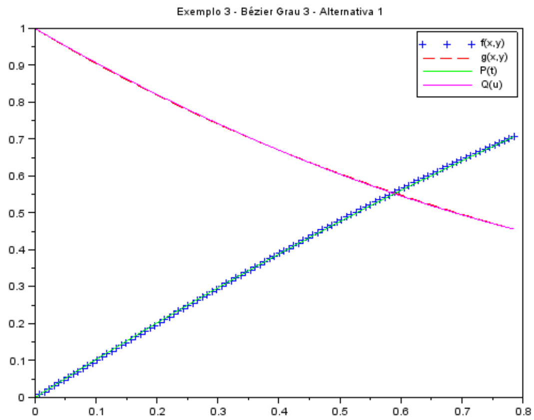

Figura 4.8: Gráfico do exemplo 3 no intervalo $x \in [0, 1]$.

gência para a solução analítica $(0.58853, 0.55514)$:

Tabela 4.13: Exemplo 3 - Resultados para duas iterações do Bézier Clipping

Iteração	Pontos extremos	Erro
1	(0.5842148,0.5551080);(0.6042325,0.5440004)	0.0200177
2	(0.5891215,0.5522378);(0.5891433,0.5522558)	0.0000218

Tabela 4.14: Exemplo 3 - Iterações para $P(t)$ - Subdivisão de De Casteljau e Intervalar

Iteração	Pontos extremos de P(t)	Erro
1	(0.392699,0.382683);(0.785398,0.707107)	0.196868
2	(0.392699,0.382683);(0.589049,0.552178)	0.195831
3	(0.490874,0.469251);(0.589049,0.552178)	0.097656
4	(0.539961,0.511170);(0.589049,0.552178)	0.048569
5	(0.564505,0.531787);(0.589049,0.552178)	0.024025
6	(0.576777,0.542011);(0.589049,0.552178)	0.013129
7	(0.582913,0.547101);(0.589049,0.552178)	0.008039
8	(0.585981,0.549641);(0.589049,0.552178)	0.005499
9	(0.587515,0.550910);(0.589049,0.552178)	0.004230
10	(0.588282,0.551544);(0.589049,0.552178)	0.003596

Tabela 4.15: Exemplo 3 - Iterações para $Q(u)$ - Subdivisão de De Casteljau e Intervalar

Iteração	Pontos extremos de Q(u)	Erro
1	(0.392699,0.675232);(0.785398,0.455938)	0.196868
2	(0.392699,0.675232);(0.589049,0.552401)	0.195831
3	(0.490874,0.610520);(0.589049,0.552401)	0.097656
4	(0.539961,0.580636);(0.589049,0.552401)	0.048569
5	(0.564505,0.566313);(0.589049,0.552401)	0.024025
6	(0.576777,0.559305);(0.589049,0.552401)	0.011753
7	(0.582913,0.555840);(0.589049,0.552401)	0.005617
8	(0.585981,0.554117);(0.589049,0.552401)	0.002739
9	(0.587515,0.553258);(0.589049,0.552401)	0.002739
10	(0.588282,0.552829);(0.589049,0.552401)	0.002739

Às vezes, ao resolver um sistema de equações não lineares pelo método de Newton-Raphson, quando a raiz está suficientemente próxima de pontos onde a derivada da curva se anula, o determinante da matriz jacobiana se anula e as iterações do método não convergem para a raiz.

O próximo exemplo vai mostrar que, neste caso, os algoritmos de interseções vistos nesse trabalho possuem maior eficiência que o método de Newton-Raphson.

4.4.4 Exemplo 4

Encontrar as interseções entre as curvas planas:

$f(x, y) = e^{x-1} - y - 1 = 0;$

$g(x, y) = x - y - 1 = 0;$

no intervalo $x \in [1, 2]$, $y \geq 0$.

No método de Newton-Raphson, a matriz jacobiana J é:

$$J = \begin{bmatrix} e^{x-1} & -1 \\ 1 & -1 \end{bmatrix}$$

A interseção entre as curvas na solução analítica é o ponto $(1, 0)$. Quando x se aproxima de 1, a determinante de J se aproxima de zero e o método de Newton-Raphson não converge para a solução.

4.4.4.1 Experimento

a) Interpolação de curvas de Bézier de grau 9:

Usando a Interpolação por curvas de Bézier de grau 9 da Interpolação de Bézier da Seção 4.3.2, os seguintes pontos de controle são encontrados:

Para a curva $P(t)$, aproximação de $f(x, y) = 0$:

$P_0 = (1.0000000, 0.0000000);$
$P_1 = (1.1111111, 0.1111111);$
$P_2 = (1.2222222, 0.2361111);$
$P_3 = (1.3333333, 0.3769841);$
$P_4 = (1.4444444, 0.5360450);$
$P_5 = (1.5555556, 0.7160053);$
$P_6 = (1.6666667, 0.9200562);$
$P_7 = (1.7777778, 1.1519731);$
$P_8 = (1.8888889, 1.4162505);$

$P_9 = (2.0000000, 1.7182818).$

Para a curva $Q(u)$, aproximação de $g(x, y) = 0$:

$Q_0 = (1.0000000, 0.0000000);$
$Q_1 = (1.1111111, 0.1111111);$
$Q_2 = (1.2222222, 0.2222222);$
$Q_3 = (1.3333333, 0.3333333);$
$Q_4 = (1.4444444, 0.4444444);$
$Q_5 = (1.5555556, 0.5555556);$
$Q_6 = (1.6666667, 0.6666667);$
$Q_7 = (1.7777778, 0.7777778);$
$Q_8 = (1.8888889, 0.8888889);$
$Q_9 = (2.0000000, 1.0000000).$

O gráfico deste exemplo desenhado com o Scilab 5.4.0 pode ser visto na Figura 4.9.

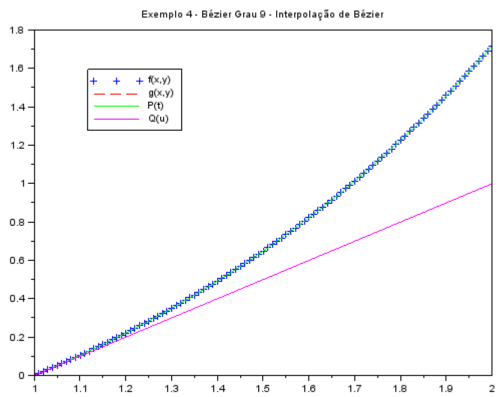

Figura 4.9: Gráfico do exemplo 4 no intervalo $x \in [1, 2]$.

b) Aplicação de Bézier Clipping:

Os resultados do algoritmo de Bézier Clipping para interpolação através de uma curva de Bézier de grau 9 são mostrados na tabela 4.16. O tempo de execução verificado foi de 42.584 segundos.

Tabela 4.16: Exemplo 4 - Iterações para Bézier Clipping usando Interpolação de Bézier da Seção 4.3 e Curva de Bézier de grau 9

Iteração	Pontos extremos	Curva	Erro
1	(1,0);(1.3321760,0.3321756000)	$P(t)$	0.3321756000
2	(1,0);(1.0003970,0.0003969348)	$Q(u)$	0.0003969348
3	(1,0);(1.0001100,0.0001102475)	$P(t)$	0.0001102475
4	(1,0);(1.0000440,4.409674e-005)	$Q(u)$	4.409674e-005
5	(1,0);(1.0000120,1.225385e-005)	$P(t)$	1.225385e-005
6	(1,0);(1.0000050,4.940473e-006)	$Q(u)$	4.940473e-006
7	(1,0);(1.0000020,1.522426e-006)	$P(t)$	1.522426e-006
8	(1,0);(1.0000010,7.612166e-007)	$Q(u)$	7.612166e-007
9	(1,0);(1.0000010,7.891473e-007)	$P(t)$	7.891473e-007

c) Aplicação da subdivisão recursiva de De Casteljau e da subdivisão intervalar:

O tempo de execução verificado foi de 0.786 segundos. Os resultados neste caso estão nas Tabelas 4.17 e 4.18:

Tabela 4.17: Exemplo 4 - Iterações para $P(t)$ - Subdivisão Recursiva de De Casteljau para curva de grau 9

Iteração	Pontos extremos de P(t)	Erro
1	(1,0);(1.5,0.6487213)	0.6487213
2	(1,0);(1.25,0.2840254)	0.2840254
3	(1,0);(1.125,0.1331485)	0.1331485
4	(1,0);(1.0625,0.06449446)	0.06449446
5	(1,0);(1.03125,0.03174341)	0.03174341
6	(1,0);(1.015625,0.01574771)	0.01574771
7	(1,0);(1.007812,0.007843097)	0.007843097
8	(1,0);(1.003906,0.003913889)	0.003913889
9	(1,0);(1.001953,0.001955034)	0.001955034
10	(1,0);(1.000977,0.0009770395)	0.0009770395

Tabela 4.18: Exemplo 4 - Iterações para $Q(u)$ - Subdivisão Recursiva de De Casteljau para curva de grau 9

Iteração	Pontos extremos de Q(u)	Erro
1	(1.0000000,0.0000000);(1.5000000,0.5000000)	0.5000000
2	(1.0000000,0.0000000);(1.2500000,0.2500000)	0.2500000
3	(1.0000000,0.0000000);(1.1250000,0.1250000)	0.1250000
4	(1.0000000,0.0000000);(1.0625000,0.0625000)	0.0625000
5	(1.0000000,0.0000000);(1.0312500,0.0312500)	0.0312500
6	(1.0000000,0.0000000);(1.0156250,0.0156250)	0.0156250
7	(1.0000000,0.0000000);(1.0078120,0.0078125)	0.0078125
8	(1.0000000,0.0000000);(1.0039060,0.00390625)	0.00390625
9	(1.0000000,0.0000000);(1.0019530,0.001953125)	0.001953125
10	(1.0000000,0.0000000);(1.0009770,0.0009770395)	0.0009770395

Agora, os resultados para Subdivisão Intervalar para curva de Bézier de grau 9 nas Tabelas 4.19 e 4.20:

Tabela 4.19: Exemplo 4 - Iterações para $P(t)$ - Subdivisão Intervalar para curva de grau 9

Iteração	Pontos extremos de P(t)	Erro
1	(1.0000000,0.0000000);(2.0000000,1.7182818)	1.7182818
2	(1.0000000,0.0000000);(1.1666667,0.1773313)	0.1773313
3	(1.0000000,0.0000000);(1.0833333,0.0859685)	0.0859685
4	(1.0000000,0.0000000);(1.0416667,0.04232158)	0.04232158
5	(1.0000000,0.0000000);(1.0208333,0.02099658)	0.02099658
6	(1.0000000,0.0000000);(1.0104170,0.01045742)	0.01045742
7	(1.0000000,0.0000000);(1.0052080,0.005218513)	0.005218513
8	(1.0000000,0.0000000);(1.0026040,0.002606711)	0.002606711
9	(1.0000000,0.0000000);(1.0013020,0.001302719)	0.001302719
10	(1.0000000,0.0000000);(1.0006510,0.0006512006)	0.0006512006

Tabela 4.20: Exemplo 4 - Iterações para $Q(u)$ - Subdivisão Intervalar para curva de grau 9

Iteração	Pontos extremos de Q(u)	Erro
1	(1.0000000,0.0000000);(2.0000000,1.0000000)	1.0000000
2	(1.0000000,0.0000000);(1.1666667,0.1666667)	0.1666667
3	(1.0000000,0.0000000);(1.0416667,0.04166667)	0.0416667
4	(1.0000000,0.0000000);(1.0208333,0.02083333)	0.0208333
5	(1.0000000,0.0000000);(1.0104170,0.01041667)	0.0104167
6	(1.0000000,0.0000000);(1.0052080,0.005208333)	0.0052083
7	(1.0000000,0.0000000);(1.0026040,0.002604167)	0.0026042
8	(1.0000000,0.0000000);(1.0013020,0.001302083)	0.0013021
9	(1.0000000,0.0000000);(1.0006510,0.0006510417)	0.0006510
10	(1.0000000,0.0000000);(1.0003260,0.0003255208)	0.0003260

5 CONCLUSÕES

Aqui, serão apresentadas as considerações sobre a convergência dos algoritmos vistos neste trabalho e as vantagens e desvantagens sobre outros métodos para localização de raízes de equações.

5.1 Convergência dos algoritmos

Através dos resultados encontrados nos experimentos realizados neste trabalho, nota-se que o algoritmo de Bézier Clipping, apesar de consumir menos iterações que os demais sob uma determinada precisão, apresentou um tempo de execução maior. Em contrapartida, os algoritmos de Subdivisão Recursiva de De Casteljau e Subdivisão Intervalar obtiveram um tempo de execução menor e custaram mais iterações para se aproximarem da solução analítica, já que a cada iteração a norma máxima do erro decai pela metade, o que confirma a convergência linear destes dois últimos métodos. Segundo (SCHULZ, 2009), foi comprovado que o método de Bézier Clipping de (NISHITA ; SEDERBERG, 1990) possui convergência quadrática.

Nas tabelas 5.1 e 5.2, para o exemplo 1 do capítulo 4, são mostrados os valores mínimo e máximo dos parâmetros de cada curva e seus respectivos erros absolutos durante as iterações do algoritmo de Bézier Clipping mostrando que a convergência é quadrática.

Tabela 5.1: Valores mínimo e máximo do parâmetro para Bézier Clipping usando Alternativa 2 da Seção 4.3 e Curva de Bézier de grau 9

Iteração	Curva de recorte	Parâmetro	Valor mínimo	Valor máximo
1	$Q(u)$	u	0.0452587	0.2779173
2	$P(t)$	t	0.3105570	0.3162347
3	$Q(u)$	u	0.1478380	0.1478508
4	$P(t)$	t	0.4344906	0.4344906

Tabela 5.2: Erros para os valores mínimo e máximo do parâmetro para Bézier Clipping usando Alternativa 2 da Seção 4.3 e Curva de Bézier de grau 9

Iteração	Curva de recorte	Erro absoluto
1	$Q(u)$	0.2326586
2	$P(t)$	0.0056775
3	$Q(u)$	1.2487298e-05
4	$P(t)$	1.7439412e-09

5.2 Principais contribuições

As principais contribuições deste trabalho são:

- O método numérico proposto aqui, através das curvas de Bézier, torna-se mais interessante do que o método analítico para localizar os pontos de interseção entre duas curvas algébricas planas se elas forem representadas por funções polinomiais de grau bastante elevado. Se os polinômios trabalhados tiverem um grau bastante elevado, o cálculo da resultante fica muito custoso. Além disso, também é muito custoso encontrar as suas raízes para determinar os pontos de interseção.

- Os algoritmos discutidos neste trabalho apresentam vantagem sobre o método de Newton-Raphson para sistemas não lineares quando a matriz jacobiana é singular neste método, de modo que não há convergência para a solução do problema, como foi visto no exemplo 4 do capítulo 4.

5.3 Trabalhos futuros

Uma melhoria que precisa ser feita é o procedimento de verificação de interseções simples, já que a utilização de *Bounding Box* ou de fechos convexos como regiões para cobrir as curvas de Bézier não funciona em todos os casos para detectar se duas curvas $f(x, y) = 0$ e $g(x, y) = 0$ se interceptam. É diferente do que acontece com as funções de uma variável que possuem forma explícita $y = f(x)$. Neste caso, a aplicação do Teorema do Valor Intermediário é capaz de detectar se duas curvas se interceptam para todas as funções de uma variável.

Outra melhoria para o algoritmo de Bézier Clipping foi encontrada no artigo (LOU ; LIU, 2012), onde ao invés da utilização de uma *fat line*, foi utilizada uma *fat curve*, que é uma região que se aproxima muito mais da curva. Essa técnica foi denominada *Hybrid Clipping*.

Além disso, um paralelismo aplicado aos algoritmos descritos neste trabalho pode reduzir seu tempo de execução, especialmente no Bézier Clipping, cujos estágios desde a formação da *fat line* até o final de cada iteração demandam bastante tempo de execução, conforme os experimentos mostrados.

REFERÊNCIAS

BUSS, S. **3-D Computer graphics** : a mathematical introduction with OpenGL. New York: Cambridge University Press, 2003.

COX, D. ; LITTLE, J. ; O'SHEA, D. **Ideals, varieties, and algorithms** : an introduction to computational algebraic geometry and commutative algebra. Estados Unidos: Springer, 2007.

HECKBERT, P. **Graphics gems IV**. Estados Unidos: Academic Press, 1994. (The Graphic Gems Series).

LENGYEL, E. **Mathematics for 3D game programming and computer graphics**. Massachusetts: Charles River Media, 2004.

LOU, Q. ; LIU, L. Curve intersection using hybrid clipping. **Computers & graphics**, Guidford, Eng., v. 36, p.309–320, 2012.

MARSH, D. **Applied geometry for computer graphics and CAD**. Estados Unidos: Springer, 2005. (Springer Undergraduate Mathematics Series).

NISHITA, T. ; SEDERBERG, T. Curve intersection using Bézier clipping. **Computer-Aided Design**, Guidford, Eng., v. 22, n.9, p.538–549, Nov. 1990.

ROGERS, D. **An introduction to NURBS with historical perspective**. San Francisco: Academic Press, 2001.

SCHULZ, C. Bézier clipping is quadratically convergent. **Computer-Aided Design**, Guidford, Eng., v. 26, n.1, p.61–74, 2009.

SEDERBERG, T. ; PARRY, S. Comparison of three curves intersection algorithms. **Computer-Aided Design**, Guidford, Eng., v. 18, n.1, p.58–63, Jan. 1986.

VAINSENCHER, I. **Introdução às curvas algébricas planas**. Rio de Janeiro - Brasil: IMPA, 2009. (Coleção Matemática Universitária).

APÊNDICE A

CÓDIGO FONTE: **BibliotecaFuncoesBezier.sce**

```



//Polinomio de Bernstein
//parametros de entrada: grau da curva (n),
//                       i (varia de 0 a n),
//                       parametro s (varia no intervalo [xinicial,xfinal]),
//                       xinicial, xfinal (extremos do intervalo da curva).
function [Bern] = B(n, i, s, xinicial, xfinal)

  Bern = factorial(n)/(factorial(n-i)*factorial(i))*((xfinal - s)/(xfinal - xinicial))^(n-i)*((s-xinicial)/(xfinal-xinicial))^i;

endfunction

//Derivada do polinomio de Bernstein
function [BernDer] = derB(n, i, s, xinicial, xfinal)

    if (s == xinicial) then
        t = toc(); // numero de segundos desde a ultima chamada a funcao tic()
        printf("Tempo de execucao: %g segundos\n", t);
        pause;
    end

    BernDer = factorial(n)/(factorial(n-i)*factorial(i)) * (((n-i)*((xfinal-s)/(xfinal-xinicial))^(n-i-1))*(-1/(xfinal-xinicial))*((s-xinicial)/(xfinal-xinicial))^(i)+((xfinal-s)/(xfinal-xinicial))^(n-i)*(i)*((s-xinicial)/(xfinal-xinicial))^(i-1)*(1/(xfinal-xinicial)));

endfunction
```

```

// Polinomio de Bernstein com incognita %s
function [Bernlog] = Blog(n, i, xinicial, xfinal)

  Bernlog = factorial(n)/(factorial(n-i)*factorial(i))*((xfinal - %s)/(xfinal - xinicial))^(n-i)*((%s-xinicial)/(xfinal-xinicial))^i;

endfunction

//Reta dos pontos extremos da curva
//Coordenadas horizontais dos extremos: XInicial, XFinal
//Coordenadas verticais dos extremos: YInicial, YFinal
function [a, b, c] = RetaExtremos(XInicial, XFinal, YInicial, YFinal)

    a = YInicial - YFinal; //printf("A == %.10g\n", a);
    b = XFinal - XInicial;
    c = XInicial * YFinal - XFinal * YInicial;

endfunction

//Distancia de um ponto (x,y) a reta ax + by + c = 0.
function [d] = Dist(x, y, a, b, c)

    d = (a * x + b * y + c) / sqrt (a^2 + b^2);

endfunction

// Novos pontos de controle a partir dos novos parametros smin e smax da curva a cada iteracao:
function [X,Y] = NovosPontosControleV2(XI,YI,smin,smax,n)


    sinicial = 0;
    sfinal = 1;

        X = zeros(1,n+1);

    xmin = 0; xmax = 0; ymin = 0; ymax = 0;
    der_xmin = 0; der_xmax = 0; der_ymin = 0; der_ymax = 0;


```

```
if (n == 3) then

    for i = 1:(n+1)


        xmin = xmin + XI(i) * B(n, i-1, smin, sinicial, sfinal);
        xmax = xmax + XI(i) * B(n, i-1, smax, sinicial, sfinal);
        ymin = ymin + YI(i) * B(n, i-1, smin, sinicial, sfinal);
        ymax = ymax + YI(i) * B(n, i-1, smax, sinicial, sfinal);

        der_xmin = der_xmin + XI(i) * derB(n, i-1, smin, sinicial, sfinal);
        der_xmax = der_xmax + XI(i) * derB(n, i-1, smax, sinicial, sfinal);
        der_ymin = der_ymin + YI(i) * derB(n, i-1, smin, sinicial, sfinal);
        der_ymax = der_ymax + YI(i) * derB(n, i-1, smax, sinicial, sfinal);

    end

        norma_min = norm([der_xmin der_ymin]);
        norma_max = norm([der_xmax der_ymax]);

        der_xmin = der_xmin/norma_min;
        der_ymin = der_ymin/norma_min;
        der_xmax = der_xmax/norma_max;
        der_ymax = der_ymax/norma_max;

         //Novos pontos de controle:

        //Primeiro, os intermediarios:
        X(2) = xmin + ((xmax - xmin)/3)*der_xmin;
        X(3) = xmax - ((xmax - xmin)/3)*der_xmax;

        Y(2) = ymin + ((xmax - xmin)/3)*der_ymin;
        Y(3) = ymax - ((xmax - xmin)/3)*der_ymax;

        //Depois, os extremos:
```

```
        X(1) = xmin;
        Y(1) = ymin;

        X(n+1) = xmax;
        Y(n+1) = ymax;

end

if (n > 3) then

        VX = zeros(n+1,1);
        VY = VX;
        M = zeros(n+1, n+1);

        for i = 1:(n+1)
                for j = 1:(n+1)
                        M(i,j) = B(n, j-1, smin + (smax - smin
                           )*(i-1)/n, smin, smax);
                        VX(i) = VX(i) + XI(j) * B(n, j-1, smin
                            + (smax - smin)*(i-1)/n, sinicial,
                            sfinal);
                        VY(i) = VY(i) + YI(j) * B(n, j-1, smin
                            + (smax - smin)*(i-1)/n, sinicial,
                            sfinal);
                end
        end

        X = inv(M) * VX;
        Y = inv(M) * VY;

end

endfunction

// Bezier Clipping Versao 1 (calcula as raizes de polinomios
   usando a incognita %s e a funcao roots do Scilab)
function[umin,umax,u1,u2,dmin,dmax,H1,H2,XQN,YQN] =
   BezierClippingV1(XP,YP,XQ,YQ,xinicial,xfinal,yinicial,
   yfinal,n)


        [a,b,c] = RetaExtremos(xinicial,xfinal,yinicial,yfinal
           );
```

```
        if (a == 0 | b == 0) then
            printf("Distancia proxima de zero!, a == %g\n", a);
            abort;
        end

        //Distancias dos pontos interiores da curva 1 a reta que une seus extremos:
        dint = zeros(1,n-1);

        for i = 2:n
            dint(i) = Dist(XP(i), YP(i), a, b, c);
        end

        //Calculo das distancias dmin e dmax:

        if (n == 3) then
            if (dint(2) * dint(3) > 0) then
                dmin = (3/4) * min(dint);
                dmax = (3/4) * max(dint);
            end
            if (dint(2) * dint(3) <= 0) then
                dmin = (4/9) * min(dint);
                dmax = (4/9) * max(dint);
            end
        end

        if (n > 3) then
            dmin = min(dint);
            dmax = max(dint);
        end

        //Distancias dos pontos de controle da curva 2 a reta que une os pontos extremos da curva 1:
        d2 = zeros(1,n+1);
        for i = 1:(n+1)
            d2(i) = Dist(XQ(i),YQ(i),a,b,c);
        end

        //Intersecoes de dmin e dmax com os segmentos do fecho convexo do poligono de controle da curva 2:

```

```
        u1 = zeros(1,n+1);
        u2 = u1;

        //    TIPO 1:

            polinomio = 0;

            for i = 1:n+1
                U(i) = (i-1)/n
                polinomio = polinomio + d2(i) * Blog(n,i
                   -1,0,1);
            end

            u1 = roots(polinomio - dmin);

            u2 = roots(polinomio - dmax);

            u1 = real(u1);
            u2 = real(u2);

            r1 = zeros(1,n);
            r2 = zeros(1,n);

              Ind1 = zeros(1,n+1);
              Ind2 = zeros(1,n+1);

            for i = 1:n
                if (u1(i) >= 0 & u1(i) <= 1) then
                    r1(i) = u1(i);
                    Ind1(j) = i; j = j+1;
                end
                if (u2(i) >= 0 & u2(i) <= 1) then
                    r2(i) = u2(i);
                    Ind2(k) = i; k = k+1;
                end
            end

        Ind1 = sparse(Ind1);
        Ind2 = sparse(Ind2);

        [ij_ind1,val_ind1,size_ind1] = spget(Ind1);
        [ij_ind2,val_ind2,size_ind2] = spget(Ind2);
```

```
        H1 = val_ind1;
        H2 = val_ind2;

        [l1,c1]=size(H1)
        [l2,c2]=size(H2)

        //Se H1 ou H2 for nula:
        if (l1 == 0 | c1 == 0) then
                        H1 = 1 // Pego o primeiro (ou qualquer
                           ) indice do vetor u1 que eh zero.
        end

       if (l2 == 0 | c2 == 0) then
                        H2 = 1 // Pego o primeiro (ou qualquer
                           ) indice do vetor u2 que eh zero.
        end

        // Valores dos parametros umin e umax:
            umin = min(  min(r1(H1)), min(r2(H2)) );
            umax = max(  max(r1(H1)), max(r2(H2)) );

////        //FIM TIPO 1

        if (umin == umax) then
            printf("Iteracoes encerradas. Motivo: Pontos
               extremos iguais\n");
            t = toc();
            printf("param_min = %.15g, param_max = %.15g\n",
               umin, umax);
            printf("Erro de parametro == %.15g\n\n", abs(umin
               - umax));
            printf("Tempo de execucao: %g segundos\n", t);
            pause;
        end

        //Novos pontos de controle:
        [XQN,YQN] = NovosPontosControleV2(XQ,YQ,umin,umax,n);

endfunction
```

```
// Bezier Clipping Versao 2 (faz calculo das raizes (t,D(t))
    atraves do poligono de controle das distancias)
function[umin,umax,u1,u2,dmin,dmax,H1,H2,XQN,YQN] =
    BezierClippingV2(XP,YP,XQ,YQ,xinicial,xfinal,yinicial,
    yfinal,n)


        [a,b,c] = RetaExtremos(xinicial,xfinal,yinicial,yfinal
            );

        if (a == 0 | b == 0) then
            printf("Distancia proxima de zero!, a == %g\n", a)
                ;
            abort;
        end

        //Distancias dos pontos interiores da curva 1 a reta
            que une seus extremos:
        dint = zeros(1,n-1);

        for i = 2:n
            dint(i) = Dist(XP(i), YP(i), a, b, c);
        end

        //Calculo das distancias dmin e dmax:

        if (n == 3) then
            if (dint(2) * dint(3) > 0) then
                dmin = (3/4) * min(dint);
                dmax = (3/4) * max(dint);
            end
            if (dint(2) * dint(3) <= 0) then
                dmin = (4/9) * min(dint);
                dmax = (4/9) * max(dint);
            end
        end

        if (n > 3) then
            dmin = min(dint);
            dmax = max(dint);
        end

```

```
        //Distancias dos pontos de controle da curva 2 a reta
           que une os pontos extremos da curva 1:
        d2 = zeros(1,n+1);
        for i = 1:(n+1)
            d2(i) = Dist(XQ(i),YQ(i),a,b,c);
        end

        //Intersecoes de dmin e dmax com os segmentos do fecho
            convexo do poligono de controle da curva 2:

        u1 = zeros(1,n+1);
        u2 = u1;

// TIPO 2


                Ind1 = zeros(1,n+1);
                Ind2 = zeros(1,n+1);

                j = 1; k = 1;


        for i = 1:n

            if ((d2(i) <= dmin & d2(i+1) >= dmin) | (d2(i+1)
               <= dmin & d2(i) >= dmin)) then
                [a,b,c] = RetaExtremos((i-1)/n, i/n, d2(i), d2
                   (i+1));
                u1(i) = (-b * dmin - c)/a;
                                Ind1(j) = i; j = j+1;
            end

            if ((d2(i) <= dmax & d2(i+1) >= dmax) | (d2(i+1)
               <= dmax & d2(i) >= dmax)) then
                [a,b,c] = RetaExtremos((i-1)/n, i/n, d2(i), d2
                   (i+1));
                u2(i) = (-b * dmax - c)/a;
                                Ind2(k) = i; k = k+1;
            end


       end

```

```
        if ((d2(1) <= dmin & d2(n+1) >= dmin) | (d2(n+1) <=
           dmin & d2(1) >= dmin)) then
            [a,b,c] = RetaExtremos(0, 1, d2(1), d2(n+1));
             u1(n+1) = (-b * dmin - c)/a;
                         Ind1(j) = n+1; j = j+1;
        end

        if ((d2(1) <= dmax & d2(n+1) >= dmax) | (d2(n+1) <=
           dmax & d2(1) >= dmax)) then
            [a,b,c] = RetaExtremos(0, 1, d2(1), d2(n+1));
             u2(n+1) = (-b * dmax - c)/a;
                         Ind2(k) = n+1; k = k+1;
        end

         r1 = sparse(Ind1);
         r2 = sparse(Ind2);

         [ij_r1,val_r1,size_r1] = spget(r1);
         [ij_r2,val_r2,size_r2] = spget(r2);

         H1 = val_r1;
         H2 = val_r2;


                 printf("\n");

         [l1,c1]=size(H1)
         [l2,c2]=size(H2)


         //Se H1 ou H2 for nula:
         if (l1 == 0 | c1 == 0) then
                         H1 = 1 // Pego o primeiro (ou qualquer
                            ) indice do vetor u1 que eh zero.
         end

        if (l2 == 0 | c2 == 0) then
                         H2 = 1 // Pego o primeiro (ou qualquer
                            ) indice do vetor u2 que eh zero.
        end

         umin = min(  min(u1(H1)), min(u2(H2)) );
```

```
        umax = max(  max(u1(H1)), max(u2(H2)) );

        if (umin == umax) then
            t = toc(); // numero de segundos desde a ultima chamada a funcao tic()
            printf("Iteracoes encerradas. Motivo: Pontos extremos iguais\n");
            printf("param_min = %.15g, param_max = %.15g\n", umin, umax);
            printf("Erro de parametro == %.15g\n\n", abs(umin - umax));
            printf("Tempo de execucao: %g segundos\n", t);
            pause;
        end

// FIM TIPO 2

        //Novos pontos de controle:
        [XQN,YQN] = NovosPontosControleV2(XQ,YQ,umin,umax,n);

endfunction

// Funcao para elevar o grau de uma curva de (grau-1) para grau:
function [XN, YN] = ElevateDegree(X, Y, grau)

    XN = zeros(grau+1,1);
    YN = zeros(grau+1,1);

    XN(1) = X(1);
    YN(1) = Y(1);
    XN(grau+1) = X(grau);
    YN(grau+1) = Y(grau);

    for i = 2:grau
        XN(i) = ((i-1)/(grau))*X(i-1)+(1-(i-1)/(grau))*X(i);
        YN(i) = ((i-1)/(grau))*Y(i-1)+(1-(i-1)/(grau))*Y(i);
    end

endfunction


function [Intersecao] = VerificaIntersecao(XP, YP, XQ, YQ, n)
```

```
    if(min(XP) >= max(XQ) | min(YP) >= max(YQ) | min(XQ) >=
       max(XP) | min(YQ) >= max(YP)) then
        Intersecao = 0;
    else
        Intersecao = 1;
    end

endfunction

// Subdivisao Recursiva: acha os poligonos de controle
   relativos as sub-curvas da curva original.
function [MPX, MPY, PDX, PDY, PEX, PEY] =
   SubdivisaoDeCasteljau(X,Y,n)

    MPX = zeros(n+1, n+1);
    MPY = zeros(n+1, n+1);
    PDX = zeros(n+1,1);
    PDY = zeros(n+1,1);
    PEX = PDX; PEY = PDY;

    k = n+1;

    for i = 1:n+1

        for j = 1:n+1
            if (i == 1) then
                MPX(i,j) = X(j);
                MPY(i,j) = Y(j);
            else
                if (j <= k) then
                    MPX(i,j) = (1/2)*(MPX(i-1,j) + MPX(i-1,j
                       +1));
                    MPY(i,j) = (1/2)*(MPY(i-1,j) + MPY(i-1,j
                       +1));
                end
            end
        end
        k = k-1;
    end

    m = n+1;
    // Sub-curvas de P(t):
```

```
    for k = 1:n+1
        PDX(k) = MPX(k,1);
        PDY(k) = MPY(k,1);
        PEX(k) = MPX(m,k);
        PEY(k) = MPY(m,k);
        m = m-1;
    end

endfunction

// Subdivisao Intervalar: acha as "Bounding Boxes" relativas
   as sub-curvas da curva original:
function [P1X, P1Y, P2X, P2Y] = SubdivisaoKoparkarMudur (P0X,
   P0Y, PX, PY, n, media)

    P1X = zeros(4,1);
    P2X = zeros(4,1);
    P1Y = zeros(4,1);
    P2Y = zeros(4,1);

    P1X(1) = PX(1); P1Y(1) = PY(1);
    P2X(4) = PX(n+1); P2Y(4) = PY(n+1);


    for i = 1:n+1
        P1X(4) = P1X(4) + B(n,i-1,media,0,1) * P0X(i);
        P1Y(4) = P1Y(4) + B(n,i-1,media,0,1) * P0Y(i);
    end

    P1X(2) = P1X(1);
    P1X(3) = P1X(4);
    P1Y(2) = P1Y(4);
    P1Y(3) = P1Y(1);

    P2X(1) = P1X(4);
    P2X(2) = P2X(1);
    P2X(3) = P2X(4);

    P2Y(1) = P1Y(4);
    P2Y(2) = P2Y(4);
    P2Y(3) = P2Y(1);

endfunction
```

```

// Funcao recursiva que acha os novos poligonos de controle e verifica as intersecoes a cada iteracao:
function [P1X,P1Y,P2X,P2Y,Q1X,Q1Y,Q2X,Q2Y,t_meio,u_meio,XPInt,YPInt,XQInt,YQInt] = IntersecaoSR(PX,PY,t0,t1,depth_p,QX,QY,u0,u1,depth_q,n,SolAn)

  printf("depth_p == %g\n", depth_p);
  printf("depth_q == %g\n\n", depth_q);

  while(depth_p > 0 | depth_q > 0)

    [MPX, MPY, P1X, P1Y, P2X, P2Y] = SubdivisaoDeCasteljau(PX,PY,n);

    printf("Extremos da Sub-curva P1:\n");
    printf("(%.7g,%.7g);(%.7g,%.7g)\n\n", P1X(1), P1Y(1), P1X(n+1), P1Y(n+1));

    printf("Extremos da Sub-curva P2:\n");
    printf("(%.7g,%.7g);(%.7g,%.7g)\n\n", P2X(1), P2Y(1), P2X(n+1), P2Y(n+1));

    printf("Erro absoluto P1 == %.7g\n", max(abs(P1X(1)-SolAn(1)), abs(P1X(n+1)-SolAn(1)), abs(P1Y(1)-SolAn(2)), abs(P1Y(n+1)-SolAn(2))));
    printf("Erro absoluto P2 == %.7g\n\n", max(abs(P2X(1)-SolAn(1)), abs(P2X(n+1)-SolAn(1)), abs(P2Y(1)-SolAn(2)), abs(P2Y(n+1)-SolAn(2))));

    depth_p = depth_p - 1;
    t_meio = (1/2)*(t0 + t1);

    [MQX, MQY, Q1X, Q1Y, Q2X, Q2Y] = SubdivisaoDeCasteljau(QX,QY,n);

    printf("Extremos da Sub-curva Q1:\n");
    printf("(%.7g,%.7g);(%.7g,%.7g)\n\n", Q1X(1), Q1Y(1), Q1X(n+1), Q1Y(n+1));

    printf("Extremos da Sub-curva Q2:\n");
    printf("(%.7g,%.7g);(%.7g,%.7g)\n\n", Q2X(1), Q2Y(1), Q2X(n+1), Q2Y(n+1));
```

```
        printf("Erro absoluto Q1 == %.7g\n", max(abs(Q1X(1)-
           SolAn(1)), abs(Q1X(n+1)-SolAn(1)), abs(Q1Y(1)-SolAn
           (2)), abs(Q1Y(n+1)-SolAn(2))));
        printf("Erro absoluto Q2 == %.7g\n\n", max(abs(Q2X(1)-
           SolAn(1)), abs(Q2X(n+1)-SolAn(1)), abs(Q2Y(1)-SolAn
           (2)), abs(Q2Y(n+1)-SolAn(2))));

        depth_q = depth_q - 1;
        u_meio = (1/2)*(u0 + u1);

        if ((depth_p == 0) & (depth_q == 0)) then
            t = toc(); // numero de segundos desde a ultima
               chamada a funcao tic()
            printf("Tempo de execucao: %g segundos\n", t);
            pause;
        end

        [Intersecao] = VerificaIntersecao(P1X, P1Y, Q1X, Q1Y,n
           );
        if (Intersecao <> 0) then
            XPInt = P1X;
            YPInt = P1Y;
            XQInt = Q1X;
            YQInt = Q1Y;
            printf("P1, Q1 se interceptam\n\n");
            IntersecaoSR(P1X,P1Y,t0,t_meio,depth_p,Q1X,Q1Y,u0,
               u_meio,depth_q,n,SolAn);
        end

        [Intersecao] = VerificaIntersecao(P2X, P2Y, Q1X, Q1Y,n
           );
        if (Intersecao <> 0) then
            XPInt = P2X;
            YPInt = P2Y;
            XQInt = Q1X;
            YQInt = Q1Y;
            printf("P2, Q1 se interceptam\n\n");
            IntersecaoSR(P2X,P2Y,t_meio,t1,depth_p,Q1X,Q1Y,u0,
               u_meio,depth_q,n,SolAn);
        end
```

```
        [Intersecao] = VerificaIntersecao(P1X, P1Y, Q2X, Q2Y,n
           );
        if (Intersecao <> 0) then
            XPInt = P1X;
            YPInt = P1Y;
            XQInt = Q2X;
            YQInt = Q2Y;
            printf("P1, Q2 se interceptam\n\n");
            IntersecaoSR(P1X,P1Y,t0,t_meio,depth_p,Q2X,Q2Y,
               u_meio,u1,depth_q,n,SolAn);
        end

        [Intersecao] = VerificaIntersecao(P2X, P2Y, Q2X, Q2Y,n
           );
        if (Intersecao <> 0) then
            XPInt = P2X;
            YPInt = P2Y;
            XQInt = Q2X;
            YQInt = Q2Y;
            printf("P2, Q2 se interceptam\n\n");
            IntersecaoSR(P2X,P2Y,t_meio,t1,depth_p,Q2X,Q2Y,
               u_meio,u1,depth_q,n,SolAn);
        end

    end

endfunction


// Funcao recursiva que acha novas "Bounding Boxes" e verifica
    as intersecoes a cada iteracao:
function [P1X,P1Y,P2X,P2Y,Q1X,Q1Y,Q2X,Q2Y,t_meio,u_meio,XPInt,
   YPInt,XQInt,YQInt] = IntersecaoSI(P0X, P0Y, Q0X, Q0Y, PX,PY
   ,t0,t1,depth_p,QX,QY,u0,u1,depth_q,n,SolAn)

    m = 3;

    if (n <> m) then
        [Intersecao] = VerificaIntersecao(PX, PY, QX, QY,n);
        n = m;
    else
        [Intersecao] = VerificaIntersecao(PX, PY, QX, QY,m);
    end
```

```


    printf("depth_p == %g\n", depth_p);
    printf("depth_q == %g\n\n", depth_q);

    while(depth_p > 0 | depth_q > 0)

        t_meio = (1/2)*(t0 + t1);
        [P1X, P1Y, P2X, P2Y] = SubdivisaoKoparkarMudur (P0X,
           P0Y, PX, PY, n, t_meio);

        printf("Sub-curva P1:\n");
        printf("(%.7g,%.7g);(%.7g,%.7g)\n\n", P1X(1), P1Y(1),
           P1X(n+1), P1Y(n+1));
        printf("Sub-curva P2:\n");
        printf("(%.7g,%.7g);(%.7g,%.7g)\n\n", P2X(1), P2Y(1),
           P2X(n+1), P2Y(n+1));

        printf("Erro absoluto P1 == %.7g\n", max(abs(P1X(1)-
           SolAn(1)), abs(P1X(n+1)-SolAn(1)), abs(P1Y(1)-SolAn
           (2)), abs(P1Y(n+1)-SolAn(2))));
        printf("Erro absoluto P2 == %.7g\n\n", max(abs(P2X(1)-
           SolAn(1)), abs(P2X(n+1)-SolAn(1)), abs(P2Y(1)-SolAn
           (2)), abs(P2Y(n+1)-SolAn(2))));


        depth_p = depth_p - 1;

        u_meio = (1/2)*(u0 + u1);
        [Q1X, Q1Y, Q2X, Q2Y] = SubdivisaoKoparkarMudur (Q0X,
           Q0Y, QX, QY, n, u_meio);


        printf("Sub-curva Q1:\n");
        printf("(%.7g,%.7g);(%.7g,%.7g)\n\n", Q1X(1), Q1Y(1),
           Q1X(n+1), Q1Y(n+1));

        printf("Sub-curva Q2:\n");
        printf("(%.7g,%.7g);(%.7g,%.7g)\n\n", Q2X(1), Q2Y(1),
           Q2X(n+1), Q2Y(n+1));

        printf("Erro absoluto Q1 == %.7g\n", max(abs(Q1X(1)-
           SolAn(1)), abs(Q1X(n+1)-SolAn(1)), abs(Q1Y(1)-SolAn
```

```
    (2)), abs(Q1Y(n+1)-SolAn(2))));
printf("Erro absoluto Q2 == %.7g\n\n", max(abs(Q2X(1)-
    SolAn(1)), abs(Q2X(n+1)-SolAn(1)), abs(Q2Y(1)-SolAn
    (2)), abs(Q2Y(n+1)-SolAn(2))));

depth_q = depth_q - 1;

if  (depth_p == 0 & depth_q == 0) then
     t = toc(); // numero de segundos desde a ultima
         chamada a funcao tic()
     printf("Tempo de execucao: %g segundos\n", t)
     pause;
end


[Intersecao] = VerificaIntersecao(P1X, P1Y, Q1X, Q1Y,m
    );
if (Intersecao <> 0) then
     XPInt = P1X;
     YPInt = P1Y;
     XQInt = Q1X;
     YQInt = Q1Y;
     printf("P1, Q1 se interceptam\n\n");
     IntersecaoSI(P0X, P0Y, Q0X, Q0Y,P1X,P1Y,t0,t_meio,
         depth_p,Q1X,Q1Y,u0,u_meio,depth_q,n,SolAn);
end

[Intersecao] = VerificaIntersecao(P2X, P2Y, Q1X, Q1Y,m
    );
if (Intersecao <> 0) then
     XPInt = P2X;
     YPInt = P2Y;
     XQInt = Q1X;
     YQInt = Q1Y;
     printf("P2, Q1 se interceptam\n\n");
     IntersecaoSI(P0X, P0Y, Q0X, Q0Y,P2X,P2Y,t_meio,t1,
         depth_p,Q1X,Q1Y,u0,u_meio,depth_q,n,SolAn);
end

[Intersecao] = VerificaIntersecao(P1X, P1Y, Q2X, Q2Y,m
    );
if (Intersecao <> 0) then
     XPInt = P1X;
```

```
            YPInt = P1Y;
            XQInt = Q2X;
            YQInt = Q2Y;
            printf("P1, Q2 se interceptam\n\n");
            IntersecaoSI(P0X, P0Y, Q0X, Q0Y,P1X,P1Y,t0,t_meio,
               depth_p,Q2X,Q2Y,u_meio,u1,depth_q,n,SolAn);
        end

        [Intersecao] = VerificaIntersecao(P2X, P2Y, Q2X, Q2Y,m
           );
        if (Intersecao <> 0) then
            XPInt = P2X;
            YPInt = P2Y;
            XQInt = Q2X;
            YQInt = Q2Y;
            printf("P2, Q2 se interceptam\n\n");
            IntersecaoSI(P0X, P0Y, Q0X, Q0Y,P2X,P2Y,t_meio,t1,
               depth_p,Q2X,Q2Y,u_meio,u1,depth_q,n,SolAn);
        end

    end

endfunction
```

APÊNDICE B

CÓDIGO FONTE: **BezierClipping.sce**.

```
clear;

clc;

tic(); //comeca a contagem de tempo de execucao do programa

exec ('BibliotecaFuncoesBezier.sce');

format('v',25);

n = 9;

flagCurva = 1;

tol = 10^-10;

numiter = 0;

tmin = 0; tmax = 1;
umin = 0; umax = 1;

// Extremos do intervalo do exemplo 1 (mudar estas variaveis
   para outros exemplos):
xinicial = 2;
xfinal = 4;


// Usando interpolacao de Bezier:

M = zeros(n+1, n+1);

for i = 1:(n+1)
```

```
    for j = 1:(n+1)
        M(i,j) = B(n, j-1, (i-1)/n, 0, 1);
    end
end

function [f] = f(x,y)
        f = x^2 - 4*x + 4*y^2; //elipse - Exemplo 1;
//    f = x^2 + 4*y^2 - 4; //circunferencia - Exemplo 2;
//        f = exp(x-1) - y - 1; //Exemplo 4
endfunction

function [g] = g(x,y)
    g = x^2 - 8*x + y^2 + 12; //circunferencia - Exemplo 1;
//    g = x^15 + 3*x^10*y^2 + 3*x^5*y^4 + y^6 - 1; //Exemplo 2;
//    g = x - y - 1; //Exemplo 4;
endfunction

// Para uso da Alternativa 2, com Interpolacao via curvas de Bezier:
VX1 = zeros(n+1,1);
VX2 = VX1;
VY1 = VX1;
VY2 = VX1;
X1 = VX1;
X2 = VX2;
Y1 = VY1;
Y2 = VY2;

for i = 1:(n+1)
    VX1(i) = xinicial + (xfinal - xinicial) * (i-1)/n;
    VX2(i) = VX1(i);
    Tf = roots(f(VX1(i),%s));
        Tg = roots(g(VX2(i),%s));
        [lf,cf] = size(Tf);
        [lg,cg] = size(Tg);
    for k = 1:lf //grau de f,g
        if real(Tf(k)) >= 0 then
            VY1(i) = Tf(k);
        end
        end
        for k = 1:lg
                if real(Tg(k)) >= 0 then
```

```
            VY2(i) = Tg(k);
        end
        end
end

// Pontos de controle:

X1 = inv(M)*VX1;
Y1 = inv(M)*VY1;

X2 = inv(M)*VX2;
Y2 = inv(M)*VY2;


X1 = real(X1);
Y1 = real(Y1);

X2 = real(X2);
Y2 = real(Y2);


// Pontos adquiridos com parametrizacao:

//Exemplo 1:

//elipse: (intervalo [2,4])
//X2 = [2 3.2189515 3.8856181 4];
//Y2 = [1 0.9428091 0.6094757 0];
//
////circunferencia:
//X1 = [2 2.1143819 2.7810485 4];
//Y1 = [0 1.2189515 1.8856181 2];

// Exemplo 2:

//X1 = [-0.5 -0.1722284 -0.0055617 0];
//Y1 = [1.3228756 0.9428089 0.5018504 0];
//
//X2 = [-0.5  -0.3333332  -0.1666668    0.];
//Y2 = [1.0155048 0.9931494 1.0022779 1.];

// Exemplo 3:
//X1 = [0 0.2617994 0.5235988 0.7853982]; //sen(x)
```

```
//Y1 = [0 0.2745423 0.5102445 0.7071068];
//
//X2 = [0 0.2617994 0.5235988 0.7853982]; //exp(-x)
//Y2 = [1 0.7483298 0.5669758 0.4559381];



//Grau para o qual se deseja elevar a curva:
m = 9;

while (n <> m)
    [XN1, YN1]= ElevateDegree(X1, Y1, n+1);
    [XN2, YN2]= ElevateDegree(X2, Y2, n+1);
    X1 = XN1;
    X2 = XN2;
    Y1 = YN1;
    Y2 = YN2;
    n = n+1;
end

// Guardo os pontos de controle originais:
XA = X1;
YA = Y1;
XB = X2;
YB = Y2;


//while ( (abs(tmax - tmin) > tol | abs(umin - umax) > tol) )
   //& numiter < 10 )

//Solucoes analiticas:

//Exemplo 1:
SolAn = [2.2629657 0.9913184]; //Elipse, circunferencia

//Exemplo 2:
//////SolAn = [-0.2683510 1.0006956];

//Exemplo 3:
////SolAn = [0.58853 0.55514]; //sin(x), exp(-x)

//Exemplo 4:
////SolAn = [1 0];
```

```

// Define titulo para a janela do grafico:
//xtitle("Exemplo 4 - Bezier Grau 9 - Alternativa 2");

////Exemplo 1 - Grafico:

//elipse:
teta1 = 0:1/100:%pi/2;
x1 = 2 + 2*cos(teta1);
y1 = sin(teta1);

plot(x1,y1,'+');

//circunferencia:

teta2 = %pi/2:1/100:%pi;
x2 = 4 + 2*cos(teta2);
y2 = 2*sin(teta2);

plot(x2,y2,'r--');


////Exemplo 2 - Grafico:

//teta = 0:1/100:0.7227342;
//x1 = 2*cos(teta) - 2;
//y1 = 2*sin(teta);
//
//plot(x1,y1,'+');
//
//xinicial = -0.5;
//xfinal = 0;
//
//x2 = xinicial:1/100:xfinal;
//[lx2,cx2] = size(x2);
//y2 = zeros(1,cx2);
////y2 = exp(x2 - 1) - 1;
//for k = 1:cx2
//    Tg2 = roots(g(x2(k),%s));
//      [lg2,cg2] = size(Tg2);
//    for l = 1:lg2 //grau de g
//        if (real(Tg2(l)) >= 1) then
//            y2(k) = Tg2(l);
```

```
//          end
//        end
//end

//plot(x2,y2,'r--');


////Exemplo 3 - Grafico:

////sin(x), exp(-x):

//xinicial = 0;
//xfinal = %pi/4;
//
//x1 = xinicial:(xfinal-xinicial)/100:xfinal;
//x2 = x1;
//
//y1 = sin(x1);
//y2 = exp(-x2);
//
//
//plot(x1,y1,'+');
//plot(x2,y2,'r--');


////Exemplo 4 - Grafico:

//xinicial = 1;
//xfinal = 2;
//
//x1 = xinicial:1/100:xfinal;
//y1 = exp(x1-1) - 1
//
//plot(x1,y1,'+');
//
//x2 = x1;
//y2 = x2 - 1;
//
//plot(x2,y2,'r--');
```

```
k = 100;

t = xinicial;

X1G = zeros(1,k+1);
Y1G = zeros(1,k+1);
X2G = zeros(1,k+1);
Y2G = zeros(1,k+1);
//
for i = 1:(k+1)
    for j = 1:(n+1)
                t = xinicial + (i-1)*((xfinal-xinicial)/k);
        X1G(i) = X1G(i) + X1(j)*B(n, j-1, t, xinicial, xfinal);
        Y1G(i) = Y1G(i) + Y1(j)*B(n, j-1, t, xinicial, xfinal);
        X2G(i) = X2G(i) + X2(j)*B(n, j-1, t, xinicial, xfinal);
        Y2G(i) = Y2G(i) + Y2(j)*B(n, j-1, t, xinicial, xfinal);
    end
end

plot(X1G,Y1G,'g');
plot(X2G,Y2G,'m');


legend(['f(x,y)';'g(x,y)';'P(t)';'Q(u)'],1);



xi1 = X1(1);
xf1 = X1(n+1);
yi1 = Y1(1);
yf1 = Y1(n+1);

//numiter = input("Numero de iteracoes: ");

numiter = 10;

iter = 0;

while(iter < numiter)
```

```

    iter = iter + 1;

    printf("Iteracao %g:\n", iter);

    printf("Pontos Minimo e Maximo da Curva 2:\n\n");
    printf("tmin = %.15g, tmax = %.15g\n", tmin, tmax);
    printf("Erro de parametro == %.15g\n\n", abs(tmin - tmax));
    [umin,umax,u1,u2,dmin1,dmax1,H1,H2,X2N,Y2N] = BezierClippingV2(X1,Y1,X2,Y2,xi1,xf1,yi1,yf1,n)
//    [umin,umax,u1,u2,dmin1,dmax1,H1,H2,X2N,Y2N] = BezierClippingV1(X1,Y1,X2,Y2,xi1,xf1,yi1,yf1,n)

        xi2 = X2N(1); yi2 = Y2N(1); xf2 = X2N(n+1); yf2 = Y2N(n+1);

    printf("umin = %g, umax = %.15g\n\n", umin, umax);
    printf("Pontos Min e Max:\n\n");
    printf("(xi,yi) = (%.15g,%.15g)\n",xi2,yi2);
    printf("(xf,yf) = (%.15g,%.15g)\n",xf2,yf2);
    printf("Erro absoluto == %.15g\n\n", max(abs(xi2 - SolAn(1)),abs(yi2 - SolAn(2)),abs(xf2 - SolAn(1)),abs(yf2 - SolAn(2))));


        X2 = X2N;
        Y2 = Y2N;


    printf("Pontos Minimo e Maximo da Curva 1:\n\n");
    printf("umin = %.15g, umax = %.15g\n", umin, umax);
    printf("Erro de parametro == %.15g\n\n", abs(umin - umax));
    [tmin,tmax,t1,t2,dmin2,dmax2,H1,H2,X1N,Y1N] = BezierClippingV2(X2,Y2,X1,Y1,xi2,xf2,yi2,yf2,n)
//    [tmin,tmax,t1,t2,dmin2,dmax2,H1,H2,X1N,Y1N] = BezierClippingV1(X2,Y2,X1,Y1,xi2,xf2,yi2,yf2,n)

        xi1 = X1N(1); yi1 = Y1N(1); xf1 = X1N(n+1); yf1 = Y1N(n+1);

   printf("tmin = %g, tmax = %.15g\n\n", tmin, tmax);
```

```
    printf("Pontos Min e Max:\n\n");
    printf("(xi,yi) = (%.15g,%.15g)\n", xi1, yi1);
    printf("(xf,yf) = (%.15g,%.15g)\n", xf1, yf1);
    printf("Erro absoluto == %.15g\n\n", max(abs(xi1 - SolAn(1)
        ),abs(yi1 - SolAn(2)),abs(xf1 - SolAn(1)),abs(yf1 -
        SolAn(2))));

            X1 = X1N;
            Y1 = Y1N;

end

t = toc(); // numero de segundos desde a ultima chamada a
    funcao tic()
printf("Tempo de execucao: %g segundos\n", t);
```

APÊNDICE C

CÓDIGO FONTE: **BézierSubdivisaoRecursivaCasteljau.sce**

```

clear;

clc;

tic(); //comeca a contagem de tempo de execucao do programa

exec ('BibliotecaFuncoesBezier.sce');

// Extremos do intervalo do Exemplo 1:
xinicial = 2;
xfinal = 4;

n = 9;

flagCurva = 1;

tol = 10^-5;

numiter = 0;

//// Exemplo 1 - Grau 3:
//
////elipse: (intervalo [2,4])
//PX = [2 3.2189515 3.8856181 4];
//PY = [1 0.9428091 0.6094757 0];
//
////circunferencia:
//QX = [2 2.1143819 2.7810485 4];
//QY = [0 1.2189515 1.8856181 2];
//
//
////Exemplo 2 - Grau 3:
//
//PX = [-0.5 -0.1722284 -0.0055617 0];
```

```
//PY = [1.3228756 0.9428089 0.5018504 0];
//
//QX = [-0.5  -0.3333332  -0.1666668    0.];
//QY = [1.0155048 0.9931494 1.0022779 1.];
//
//// Exemplo 3 - Grau 3:
//
//PX = [0 0.2617994 0.5235988 0.7853982]; //sen(x)
//PY = [0 0.2745423 0.5102445 0.7071068];
//
//QX = [0 0.2617994 0.5235988 0.7853982]; //exp(-x)
//QY = [1 0.7483298 0.5669758 0.4559381];
//


//Grau 9:

// Exemplo 1:

//elipse:
PX = [2.
    2.2222222222221716947388
    2.4444444444445743158667
    2.6666666666665364004984
    2.8888888888892125805796
    3.1111111111110872684549
    3.3333333333333996506553
    3.5555555555554860802658
    3.7777777777777750145560
    4.];

PY = [1.
    0.9815233367994240509802
    1.0622518068536983548711
    0.7713425854991458052723
    1.2489972781511422539324
    0.3946336381663453352076
    1.2881787186580666570990
    0.2043762382060556603847
    0.8701008088856934108435
    0.];

//circunferencia:
```

```

QX = [2.
     2.2222222222221716947388
     2.4444444444445743158667
     2.6666666666665364004984
     2.8888888888892125805796
     3.111111111110872684549
     3.3333333333333996506553
     3.5555555555554860802658
     3.7777777777777750145560
     4.];

QY = [0.
     1.7402016177713544031747
     0.4087524764122016929235
     2.5763574373159916497400
     0.7892672763330343954635
     2.4979945563019052556797
     1.5426851709985163196848
     2.1245036137072483839461
     1.9630466735988720827777
     2.0000000000000004440892];


// Exemplo 4:
//PX = [1.
//     1.1111111111110858473694
//     1.2222222222222871579334
//     1.3333333333332682002492
//     1.4444444444446062902898
//     1.5555555555554363422743
//     1.6666666666666998253277
//     1.7777777777777430401329
//     1.8888888888888875072780
//     2.];
//
//PY = [0.
//     0.1111111111565113718225
//     0.2361111109139252262068
//     0.3769841275188501317217
//     0.5360449724078226729773
//     0.7160052931575320300794
//     0.9200562127689542180065
```

```
//     1.1519731139785545082077
//     1.4162505142355308329627
//     1.7182818284590450907956];
//
//
//QX = [1.
//     1.1111111111110858473694
//     1.2222222222222871579334
//     1.3333333333332682002492
//     1.4444444444446062902898
//     1.5555555555554363422743
//     1.6666666666666998253277
//     1.7777777777777430401329
//     1.8888888888888875072780
//     2.];
//
//QY = [0.
//     0.1111111111111063864954
//     0.2222222222222427490124
//     0.3333333333332966219587
//     0.4444444444444783925974
//     0.5555555555555322655437
//     0.6666666666666571927635
//     0.7777777777777608037013
//     0.8888888888888901718133
//     1.];
//

tmin = 0; tmax = 1;
umin = 0; umax = 1;


depth_p = input("Escolha o numero de iteracoes: ");
printf("\n\n");

depth_q = depth_p;

// Exemplo 1:
SolAn = [2.2629657 0.9913184];

// Exemplo 4:
//SolAn = [1 0];

```

```
        printf("Erro absoluto P == %.7g\n", max(abs(PX(1)-
           SolAn(1)), abs(PX(n+1)-SolAn(1)), abs(PY(1)-SolAn
           (2)), abs(PY(n+1)-SolAn(2))));

        printf("Erro absoluto Q == %.7g\n\n", max(abs(QX(1)-
           SolAn(1)), abs(QX(n+1)-SolAn(1)), abs(QY(1)-SolAn
           (2)), abs(QY(n+1)-SolAn(2))));

// Curva Original:
POX = PX;
POY = PY;
QOX = QX;
QOY = QY;

[Intersecao] = VerificaIntersecao(PX, PY, QX, QY, n);

if (Intersecao == 1) then

    [P1X,P1Y,P2X,P2Y,Q1X,Q1Y,Q2X,Q2Y,t_meio,u_meio,XPInt,YPInt
       ,XQInt,YQInt] = IntersecaoSR(PX,PY,tmin,tmax,depth_p,QX
       ,QY,umin,umax,depth_q,n,SolAn);
else
    printf("As curvas nao se interceptam!!!\n");
end
```

APÊNDICE D

CÓDIGO FONTE: **BezierSubdivisaoIntervalar.sce**

```

clear;

clc;

tic(); //comeca a contagem de tempo de execucao do programa

exec ('BibliotecaFuncoesBezier.sce');

// Extremos do intervalo do Exemplo 1:
xinicial = 2;
xfinal = 4;

n = 9;

flagCurva = 1;

tol = 10^-5;

numiter = 0;

//// Exemplo 1 - Grau 3:
//
////elipse: (intervalo [2,4])
//PX = [2 3.2189515 3.8856181 4];
//PY = [1 0.9428091 0.6094757 0];
//
////circunferencia:
//QX = [2 2.1143819 2.7810485 4];
//QY = [0 1.2189515 1.8856181 2];
//
//
////Exemplo 2 - Grau 3:
//
//PX = [-0.5 -0.1722284 -0.0055617 0];
```

```
//PY = [1.3228756 0.9428089 0.5018504 0];
//
//QX = [-0.5  -0.3333332  -0.1666668    0.];
//QY = [1.0155048 0.9931494 1.0022779 1.];
//
//// Exemplo 3 - Grau 3:
//
//PX = [0 0.2617994 0.5235988 0.7853982]; //sen(x)
//PY = [0 0.2745423 0.5102445 0.7071068];
//
//QX = [0 0.2617994 0.5235988 0.7853982]; //exp(-x)
//QY = [1 0.7483298 0.5669758 0.4559381];
//


//Grau 9:

// Exemplo 1:

//elipse:
PX = [2.
    2.2222222222221716947388
    2.4444444444445743158667
    2.6666666666665364004984
    2.8888888888892125805796
    3.1111111111110872684549
    3.3333333333333996506553
    3.5555555555554860802658
    3.7777777777777750145560
    4.];

PY = [1.
    0.9815233367994240509802
    1.0622518068536983548711
    0.7713425854991458052723
    1.2489972781511422539324
    0.3946336381663453352076
    1.2881787186580666570990
    0.2043762382060556603847
    0.8701008088856934108435
    0.];

//circunferencia:
```

```

QX = [2.
     2.2222222222221716947388
     2.4444444444445743158667
     2.6666666666665364004984
     2.8888888888892125805796
     3.1111111111110872684549
     3.3333333333333996506553
     3.5555555555554860802658
     3.7777777777777750145560
     4.];

QY = [0.
     1.7402016177713544031747
     0.4087524764122016929235
     2.5763574373159916497400
     0.7892672763330343954635
     2.4979945563019052556797
     1.5426851709985163196848
     2.1245036137072483839461
     1.9630466735988720827777
     2.0000000000000004440892];


// Exemplo 4:
//PX = [1.
//     1.1111111111110858473694
//     1.2222222222222871579334
//     1.3333333333332682002492
//     1.4444444444446062902898
//     1.5555555555554363422743
//     1.6666666666666998253277
//     1.7777777777777430401329
//     1.8888888888888875072780
//     2.];
//
//PY = [0.
//     0.1111111111565113718225
//     0.2361111109139252262068
//     0.3769841275188501317217
//     0.5360449724078226729773
//     0.7160052931575320300794
//     0.9200562127689542180065
```

```
//     1.1519731139785545082077
//     1.4162505142355308329627
//     1.7182818284590450907956];
//
//
//QX = [1.
//     1.1111111111110858473694
//     1.2222222222222871579334
//     1.3333333333332682002492
//     1.4444444444446062902898
//     1.5555555555554363422743
//     1.6666666666666998253277
//     1.7777777777777430401329
//     1.8888888888888875072780
//     2.];
//
//QY = [0.
//     0.1111111111111063864954
//     0.2222222222222427490124
//     0.3333333333332966219587
//     0.4444444444444783925974
//     0.5555555555555322655437
//     0.6666666666666571927635
//     0.7777777777777608037013
//     0.8888888888888901718133
//     1.];
//

tmin = 0; tmax = 1;
umin = 0; umax = 1;


depth_p = input("Escolha o numero de iteracoes: ");
printf("\n\n");

depth_q = depth_p;

// Exemplo 1:
SolAn = [2.2629657 0.9913184];

// Exemplo 4:
//SolAn = [1 0];

```

```
        printf("Erro absoluto P == %.7g\n", max(abs(PX(1)-SolAn(1)), abs(PX(n+1)-SolAn(1)), abs(PY(1)-SolAn(2)), abs(PY(n+1)-SolAn(2))));

        printf("Erro absoluto Q == %.7g\n\n", max(abs(QX(1)-SolAn(1)), abs(QX(n+1)-SolAn(1)), abs(QY(1)-SolAn(2)), abs(QY(n+1)-SolAn(2))));


// Curva Original:
POX = PX;
POY = PY;
QOX = QX;
QOY = QY;

[Intersecao] = VerificaIntersecao(PX, PY, QX, QY, n);

if (Intersecao == 1) then
    [P1X,P1Y,P2X,P2Y,Q1X,Q1Y,Q2X,Q2Y,t_meio,u_meio,XPInt,YPInt,XQInt,YQInt] = IntersecaoSI(POX, POY, QOX, QOY, PX,PY,tmin,tmax,depth_p,QX,QY,umin,umax,depth_q,n,SolAn);
else
    printf("As curvas nao se interceptam!!!\n");
end
```