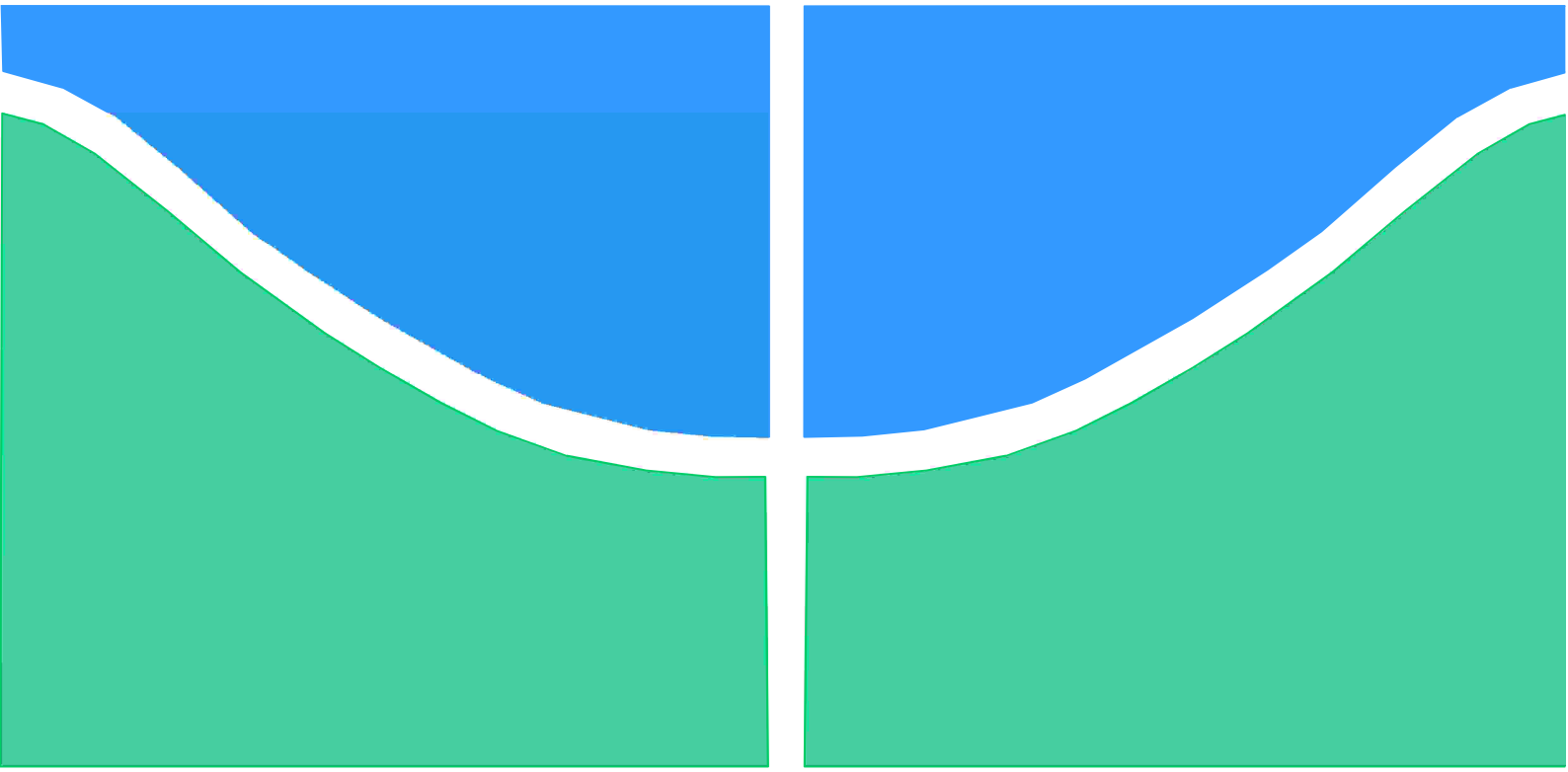

PROJETO DE GRADUAÇÃO

ANÁLISE BIDIMENSIONAL ISOGEOMÉTRICA DO MÉTODO DOS ELEMENTOS DE CONTORNO

Por,
Mateus Vinícius Honório de Sena

Brasília, 23 de novembro de 2017

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia
Departamento de Engenharia Mecânica

PROJETO DE GRADUAÇÃO

ANÁLISE BIDIMENSIONAL ISOGEOMÉTRICA DO MÉTODO DOS ELEMENTOS DE CONTORNO

Por,
Mateus Vinícius Honório de Sena

Relatório submetido como requisito parcial para obtenção
do grau de Engenheiro Mecânico.

BANCA EXAMINADORA

Prof. Éder Lima de Albuquerque (ENM-UnB)(Orientador) _____

Prof. Marcus Vinícius Girão de Morais (ENM-UnB)(Examinador) _____

MSc. Álvaro Campos Ferreira (ENM-UnB)(Examinador) _____

Brasília/DF, Novembro de 2017.

RESUMO

Este trabalho propõe uma formulação isogeométrica dos métodos dos elementos de contorno para problemas bidimensionais potenciais. Na formulação isogeométrica dos métodos dos elementos de contorno, as funções de forma polinomiais são substituídas por funções B-Splines racionais não-uniformes conhecidas como NURBS. Como as NURBS são funções de base mais utilizadas em programas CAD, com o intuito de representar as propriedades geométricas de figuras planas ou sólidas, a discretização do modelo geométrico é dispensável. No entanto, devido a complexidade matemática das NURBS quando comparadas às funções de forma polinomiais tradicionais, seu custo computacional passa a ser demasiado elevado. Diferente do MEC tradicional, as condições de contorno na formulação isogeometrica não podem ser aplicadas diretamente ao problema, uma vez que os pontos de controle estão tipicamente fora do contorno. Para superar este problema, uma matriz de transformação **E** para B-Splines é capaz de relacionar os valores entre os pontos de controle e os pontos de colocação. Soluções numéricas para problemas isogeométricos bidimensionais foram obtidas e comparadas às soluções analíticas e resultados satisfatórios para erros foram encontrados.

Palavras-chave: Método dos elementos de contorno, NURBS, matriz de transformação **E**, formulação isogeométrica.

ABSTRACT

This work proposes an isogeometric boundary element method for potential two-dimensional problems. In the isogeometric boundary element method, the polynomial shape functions are replaced by non-uniform rational B-Splines functions known as NURBS. Since NURBS are the most used basys functions in CAD, in order to represent the geometric properties of solid or plane figures, the discretization of the geometric model is dispensable. However, due to the mathematical complexity of NURBS when compared to traditional polynomial shape functions, their computational cost becomes too high. Unlike the traditional MEC, the boundary conditions in the isogeometric formulation can not be applied directly to the problem, since the control points are typically outside the boundary. To overcome this problem, a transformation matrix **E** to B-Splines is able to relate the values between the control points and the collocation points. Numerical solutions for two-dimensional isogeometric problems were obtained and compared to analytical solutions and satisfactory results for errors were reached.
Keywords: Boundary element method, NURBS, transformation matrix **E**, isogeometric.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

Símbolos Latinos

$B_{i,n}$	Polinômio de Bernstein
$N_{i,n}$	Função Base
$R_{i,n}$	Função Base Racional
n	Grau da Curva
P_i	Ponto de Controle
n	Grau do polinômio de Bernstein
U	Vetor Nós
u_i	Nó
K	Ordem da Derivada
k	Multiplicidade dos Nós
m	Número de Nós menos um
C	Vetor com os Pontos da Curva
w	Peso
$\vec{n}$	Vetor Unitário Normal ao Contorno
$\vec{r}$	Vetor Unitário Radial ao Contorno
$\vec{s}$	Vetor Unitário Tangente ao Contorno
t	Parâmetro da curva paramétrica
V	Vetor Nós
v_j	Nó

L	Ordem da derivada
f	Função dada qualquer
S	Contorno original
S”	Contorno adaptado
A	Área do contorno
T	Temperatura
T^*	Temperatura no contorno
q	Fluxo de calor
q^*	Fluxo de calor através do contorno
x_d	Coordenada x do ponto fonte
y_d	Coordenada y do ponto fonte
c	Contorno
H	Matriz H
G	Matriz G
E	Matriz E
q_e	Fluxo de calor no contorno externo

Símbolos Gregos

Γ	Contorno da área Ω
Ω	Área
ξ	Ponto de Gauss
η	Ponto de Gauss
θ	Ângulo de contorno
ϵ	Parâmetro radial
θ_{int}	Ângulo interno de contorno
π	Numero irracional "pi"

1 INTRODUÇÃO

Atualmente, projetos complexos em engenharia requerem elevados níveis de conhecimento em mecânica e em matemática. A computação tem um papel fundamental nas resoluções destes problemas complexos, que muitas vezes demandam muito tempo e dinheiro para serem solucionados. Por exemplo, problemas complexos já eram resolvidos muito antes do surgimento dos computadores. De acordo com [1], registros históricos mostram que uma das primeiras vezes em que utilizou-se a ferramenta de análise estrutural foi em meados de 1745. Naquela época, o Papa Bento XIV estava preocupado com a estabilidade da cúpula central da Basílica de São Pedro, no Vaticano. Então, foi pedido um estudo a respeito da sua condição estrutural. Trincas haviam se desenvolvido e foi questionado se os arcos metálicos instalados nela eram adequados à estrutura. Os 3 matemáticos incumbidos de resolver tal problema o simplificaram da seguinte forma:

- Entender a mecânica do problema;
- Identificar mecanismos importantes;
- Desenvolver um modelo mecânico adequado.

Os matemáticos concluíram que os esforços não seriam resistidos pelos arcos metálicos já instalados. Ou seja, a estrutura colapsaria. No entanto, até os dias atuais a estrutura mantém-se intacta. A causa da análise errônea deve-se às excessivas simplificações feitas, como considerações em 2D e restrições geométricas, o que geraram um superdimensionamento da estrutura. Um posterior estudo realizado com o Método dos Elementos Finitos (MEF) mostrou isto. Diversos outros problemas de análise estrutural foram resolvidos graças ao desenvolvimento dos computadores a partir da segunda metade do século XX. Porém, antes disso foram desenvolvidos vários métodos numéricos para resolverem equações diferenciais. Dentre eles, destacam-se o método de Ritz [2], o método de Trefftz [3] e o método das diferenças finitas [4]. A principal ideia de Ritz foi de aproximar as soluções das equações diferenciais por funções multiplicadas por parâmetros desconhecidos. Já para Trefftz, sua principal ideia consistia na utilização de soluções fundamentais das equações diferenciais e aproximar somente os valores de contorno. Por fim, a ideia do método das diferenças finitas é de aproximar numericamente a diferenciação por uma equação algébrica simples. Todos os métodos lidam com equações simultâneas, em que suas soluções são inviáveis, ou até mesmo impossíveis, de serem resolvidas sem o auxílio de um computador digital.

1.1 Métodos Numéricos: MEF e MEC

Com o advento da computação, os métodos anteriormente citados passaram a ser muito mais utilizados e também houve um grande desenvolvimento dos mesmos. As soluções analíticas, que são exatas, passaram a ser substituídas por métodos numéricos, uma vez que aproximações precisam ser feitas e, devido as complexidades das equações dos problemas, modelos gerados a

partir dessas soluções poderiam produzir modelos próximos dos reais. Em um primeiro momento, o Método das Diferenças Finitas (MDF) surgiu, depois o Método dos Elementos Finitos (MEF), a partir das ideias de Ritz, e, por último, o Método dos Elementos de Contorno (MEC) a partir da ideia de Trefftz. Neste trabalho será dada uma ênfase maior a este último método. Dessa forma, a seguir são mostradas breves explicações sobre o MEF e o MEC.

1.2 Método dos Elementos Finitos (MEF)

O Método dos Elementos Finitos (MEF) é uma forma de solução numérica de equações diferenciais parciais. Sua formulação requer a existência de uma equação integral que de certa forma possibilita substituir a integral sobre um domínio complexo por um somatório de integrais sobre domínios de geometria simples, conforme mostrado na Figura 1.1 [5]. Essa formulação pode ser baseada no método dos deslocamentos, em métodos híbridos e mistos e em modelos de equilíbrio. Cada elemento é formulado através de equações algébricas obtidas por considerações matemáticas, físicas ou experimentais. Por fim, a solução final do sistema é aproximada, uma vez que um modelo discreto é construído pelo arranjo total de todos os elementos [6].

1.3 Método dos Elementos de Contorno (MEC)

De acordo com [7], o Método dos Elementos de Contorno (MEC) é um método integral que consiste em obter as equações integrais somente com informações do contorno de um problema. Pode-se definir uma equação integral como uma equação que possua uma função operada por uma integral.

O MEC dá soluções mais precisas, tendo em vista que utiliza soluções de equações integrais sem aproximação no domínio. As informações do domínio são obtidas a partir das variáveis do contorno, pois os valores das variáveis nos pontos internos podem ser escritos como funções das variáveis no contorno.

Embora possua diversas vantagens quando comparado a outros métodos, o MEC não é bastante difundido comercialmente pois o mesmo gera matrizes cheias e não simétricas em suas soluções. Desta forma, tanto o uso da memoria quanto o tempo computacional tornam-se elevados. Soluções recentes tentam amenizar estes problemas. O método dos Elementos de Contorno Rápido [8] torna o MEC competitivo em problemas de larga escala.

A Figura 1.1 ilustra uma comparação feita na discretização entre o MEF e o MEC. Percebe-se que no MEF todo o domínio deve ser discretizado, enquanto que no MEC apenas o contorno deve ser discretizado. Este fator é uma das grandes vantagens do MEC em detrimento ao MEF.

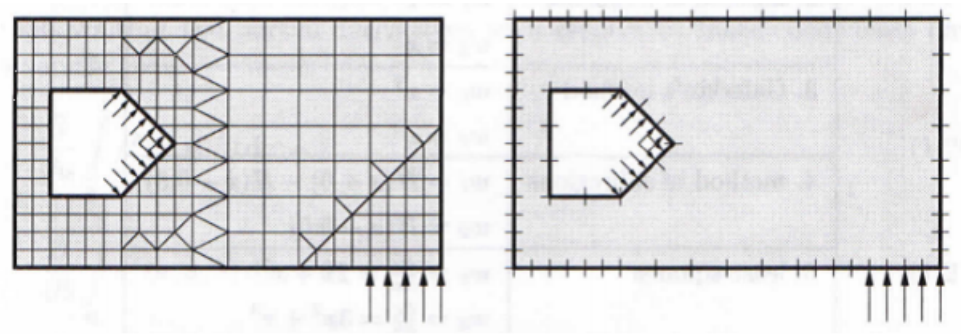

Figura 1.1: *Discretização com MEF (à esquerda) e com MEC (à direita)* [9].

1.4 Necessidade da Análise Isogeométrica

Projetos de engenharia demandam elevados trabalhos de projetistas com relação a produção de desenhos técnicos e de análises de esforços estruturais. Essa combinação de fatores, desde os primeiros projetos, tiveram que ser realizadas de maneiras simultâneas e integradas entre as equipes de desenhistas e de analistas estruturais, a fim de que o projeto final chegasse a um resultado satisfatório. Com o advento da computação, em meados da metade do seculo XX, as relações entre equipes de projeto passaram a ser diferentes. A começar pela utilização do Desenho Assistido por Computador (CAD), de responsabilidade de desenhistas/projetistas industriais, e pela Engenharia Assistida por Computador (CAE), de responsabilidade de analistas estruturais. Arquivos de CAD, depois de concluídos, eram passados às equipe de CAE, sendo assim remodelados de acordo com geometrias adequadas de análise, criadas malhas e introduzidos códigos em larga escala em análise de elementos finitos. Todo esse processo de engenharia complexa de remodelagem custa caro para a indústria em geral. Estima-se que cerca de 80% do tempo total de análise é tomado nesse processo. Sabe-se também que a medida que projetos de engenharia mais complexos são propostos, maiores são as quantidades de componentes presentes neles. Na Figura 1.2 a seguir é feita uma análise relativa do aumento da complexidade de projeto com relação ao aumento da quantidade de componentes de sistemas mecânicos (ordenada) e com o tempo de manufatura de produção (abcissa). Projetos de engenharia e análises estruturais não são esforços contrários. Todos os projetos comunicam-se de maneira simultânea com análises estruturais, e vice-e-versa.

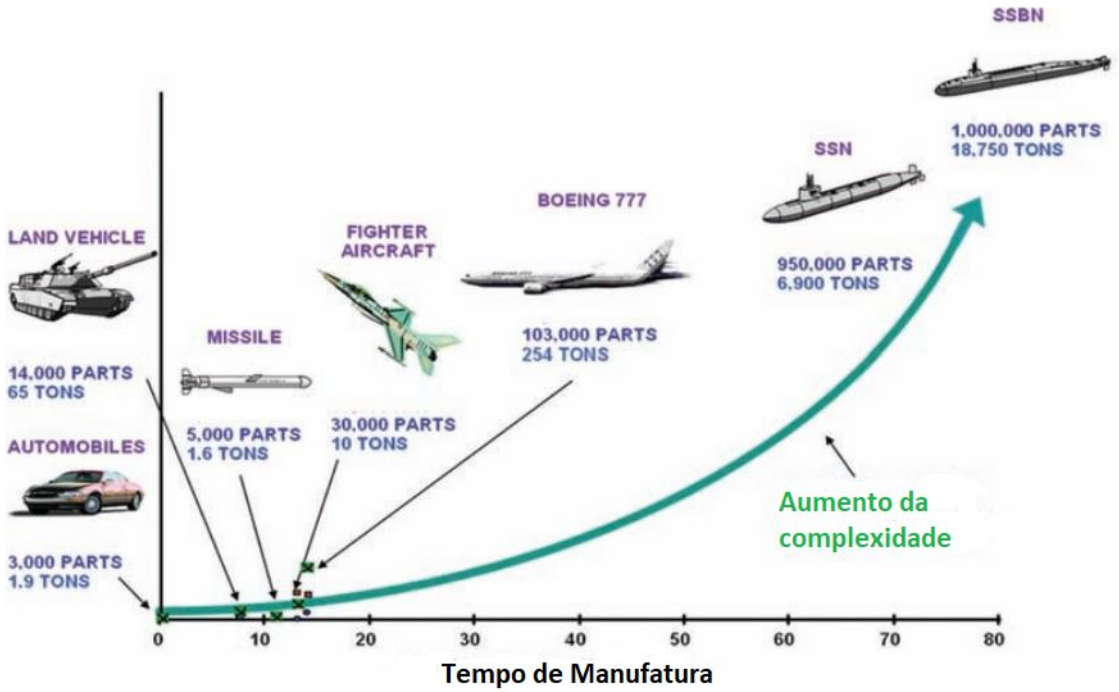

Figura 1.2: *Crescimento da complexidade dos projetos em engenharia, assim como o tempo de manufatura dos equipamentos* [10].

De acordo com estudos da Sandia National Laboratories, a geração de malhas requer cerca de 20% do tempo total de análise, enquanto que a criação de uma geometria adequada de análise requer cerca de 60%. Desta forma, apenas 20% do tempo restante é destinado à analise estrutural em si. Ou seja, a razão entre modelagem e análise é de 80% para 20% e há um grande interesse em melhorar estes números. Na Figura 1.3 é explicitada a estimativa, em percentagem, de tempo gasto para cada componente da geração de modelo e análise estrutural na Sandia National Laboratories. A integração entre CAD e CAE tem sido fundamental para que essa melhora ocorra.

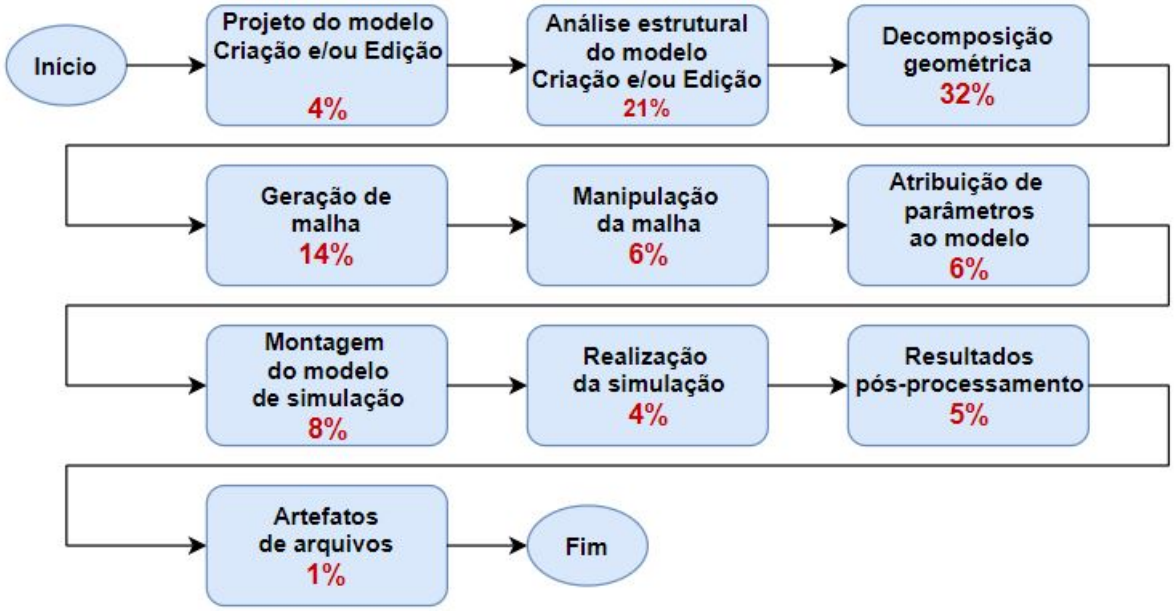

Figura 1.3: *Estimativa de tempo gasto para cada componente da geração de modelo e análise estrutural na Sandia National Laboratories* [10].

Sabe-se que uma malha em elementos finitos é apenas uma aproximação da geometria CAD. Essa aproximação existente pode criar erros em relação aos resultados analíticos. Sem aperfeiçoar a geometria e a malha adaptativa, a convergência dos resultados com alta precisão são inviáveis de serem realizados.

Aparentemente é preciso que sejam quebradas barreiras entre projeto em engenharia e análise estrutural a fim de remodelar todo o processo. Embora seja preciso manter, ao mesmo tempo, compatibilidade com práticas já existentes. Dessa forma, a ideia de focar em apenas um modelo geométrico para que possa ser também um modelo de análise estrutural. Isto requer uma mudança da clássica análise em elementos finitos para um procedimento de análise baseado em representações em CAD. Este conceito é melhor compreendido como Análise Isogeométrica, introduzido pela primeira vez por [10]. Na Figura 1.4 é comparada a análise em elementos finitos e a análise isogeométrica, que utiliza geometrias NURBS que trazem maior suavidade às curvas dos elementos em questão.

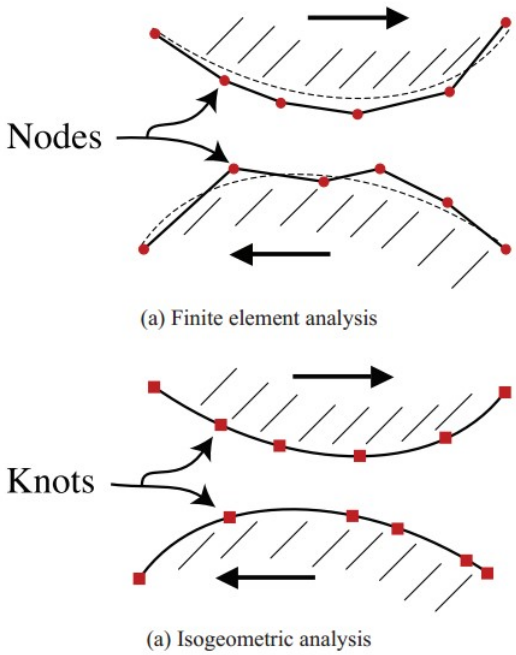

Figura 1.4: *Contato deslizante. (a) Elementos finitos com funções de forma polinomiais geram problemas de contato deslizante. (b) Geometrias NURBS atingem suavidade de corpos reais.*[10]

1.5 Motivação

Segue-se uma sequência de outros trabalhos anteriores sobre modelagem isogeométrica. O primeiro desta linha de pesquisa a abordar este tema foi a tese de doutorado [8], em que uma formulação rápida do método dos elementos de contorno isogeométrica foi desenvolvida para a análise de problemas potenciais e elásticos, bidimensionais e tridimensionais.

Nesta mesma linha de pesquisa, foram desenvolvidos os Projetos de Graduação de [11], [15], [16] e [20], além de trabalhos conjuntos com os grupos de pesquisa GFFM e GDS do Departamento de Engenharia Mecânica. No entanto, todos estes trabalhos anteriores focaram no cálculo de propriedades geométricas (área, volume e centroide) de figuras planas e de sólidos.

1.6 Objetivo

Este trabalho tem como objetivo o desenvolvimento de um programa em linguagem Julia para análises térmicas de modelos planos usando a formulação isogeométrica do método dos elementos de contorno.

1.7 Organização do trabalho

Em um primeiro momento, no Capítulo 1, é feita uma abordagem histórica inicial das simulações numéricas e métodos numéricos. No Capítulo 2 é feita uma revisão bibliográfica das funções de base utilizadas no MEC isogeométrico 2D. Já no Capítulo 3 discute-se o método dos

elementos de contorno isogeométrico, em que não são utilizadas funções de base polinomiais, mas sim as NURBS. Já no Capítulo 4 é explicada e detalhada a implementação computacional aplicada a este trabalho. No Capítulo 5 são mostrados os resultados obtidos para problemas potencias de condução de calor em diferentes geometrias em 2D. Por fim, no Capítulo 6, faz-se uma abordagem final do trabalho com comentários a cerca dos resultados obtidos e trabalhos futuros.

2 NURBS

Neste capítulo são revisadas as formulações matemáticas e modelagens geométricas utilizadas. Para descrever e modelar desenhos de curvas em desenhos assistidos por computador (CAD), assim como para aproximar as variáveis de campo, faz-se o uso das NURBS (B-Splines racionais não uniformes). Desta forma, elabora-se uma breve fundamentação teórica de conceitos de curvas, curvas de Bézier, curvas de Bézier racionais, B-Splines e NURBS.

2.1 Curvas

De acordo com [15], curvas são representadas matematicamente de maneira explícita, implícita ou paramétrica. Para este trabalho, são exploradas as curvas do tipo paramétricas.

A forma explícita é representada da seguinte maneira:

$$y = f(x), \tag{2.1}$$

em que a variável da ordenada y é uma função da variável da abcissa x. Embora esta forma seja útil em diversas aplicações, ela não pode representar adequadamente funções de múltiplos valores. Esse tipo de representação possui aplicações limitadas em gráficos computacionais ou em desenho assistido por computador.

A forma implícita é representada das seguintes maneiras:

$$f(x, y) = 0 \tag{2.2}$$

ou

$$f(x, y, z) = 0, \tag{2.3}$$

em que a expressão (2.2) é utilizada em problemas planos (curvas) e a expressão (2.3) é utilizada em problemas tridimensionais (superfícies). Ela é capaz de representar funções de múltiplos valores. As aplicações são diversas, como em gráficos computacionais e em CAD. Um exemplo é de um circulo de raio unitário com centro na origem representado pela Figura 2.1:

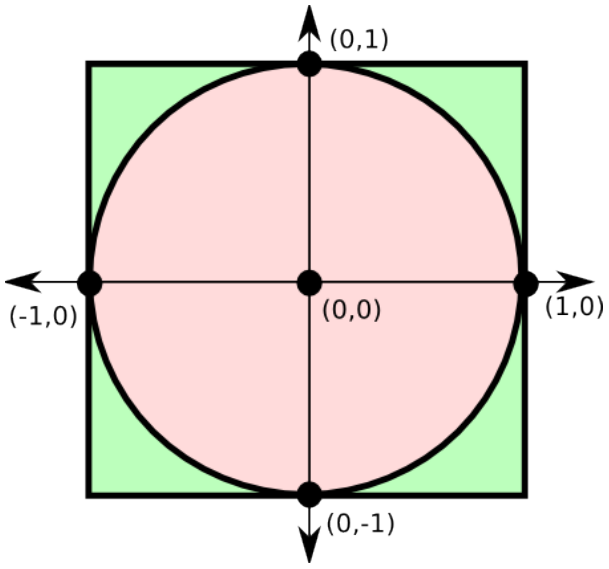

Figura 2.1: *Circulo de raio unitário centrado na origem* [12].

A expressão que representa a forma implícita do círculo acima é a seguinte:

$$f(x, y) = x^2 + y^2 - 1 = 0 \tag{2.4}$$

Por fim, a forma paramétrica é representada da seguinte maneira:

$$x = f(t); \quad y = g(t); \quad z = h(t); \tag{2.5}$$

em que t é o parâmetro. Ela não depende das coordenadas dos eixos, representa facilmente funções de múltiplos valores, derivadas infinitas e possuí mais graus de liberdade quando comparado às formas explícitas e implícitas. Cada coordenada do ponto da curva é representada separadamente como uma função explícita de um parâmetro independente. A curva é representada pela expressão a seguir:

$$C(u) = (x(u), y(u)), \qquad a \leq u \leq b, \tag{2.6}$$

em que u é a variável independente ou parâmetro e o intervalo $[a, b]$ é arbitrário, mas geralmente normalizado entre $[1, 0]$. Para o primeiro quadrante do círculo da Figura 2.1 anterior é definido pelas funções paramétricas a seguir:

$$x(u) = cos(u) \qquad 0 \leq u \leq \pi/2 \tag{2.7}$$

$$y(u) = sin(u) \qquad 0 \leq u \leq \pi/2 \tag{2.8}$$

2.2 Curvas de Bézier

A abordagem das curvas NURBS necessita alguns conceitos antecedentes, especialmente as curvas paramétricas polinomiais (curvas de Bézier), um caso especial das NURBS. Utilizadas em diversas aplicações em softwares de desenhos gráficos, foi introduzida pela primeira vez pelo francês Pierre Bézier em 1962, quando funcionário da Renault.

As curvas de Bézier são determinadas por um polígono de controle, como mostrado na Figura 2.2 a seguir:

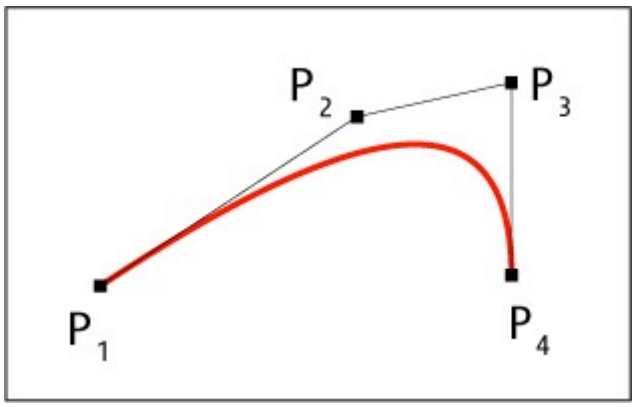

Figura 2.2: *Polígono de controle com 4 pontos* [13].

As principais propriedades das curvas de Bézier são:

- As funções de base são reais;
- O grau n do polinômio que define o segmento de curva é um a menos que o número de pontos do polígono de controle;
- A curva geralmente é orientada pela forma do polígono de controle;
- O primeiro e o último pontos da curva são coincidentes com o primeiro e último pontos do polígono de controle.

Uma curva de Bézier de grau n é expressa pela seguinte expressão:

$$C(u) = \sum_{i=0}^{n} B_{i,n}(u) P_i, \qquad 0 \leq u \leq 1, \tag{2.9}$$

em que $B_{i,n}$ é a função de base e também um polinômio de Bernstein de grau n, dado por:

$$B_{i,n}(u) = \frac{n!}{i!(n-i)!} u^i (1-u)^{n-1}, \tag{2.10}$$

em que os coeficientes geométricos P_i são os pontos de controle do polígono que define a malha de controle, que interpolam os pontos.

As funções de base possuem algumas características [18], tais como:

- São sempre não negativas;
- $\sum_{i=0}^{n} B_{i,n}(u) = 1$;
- $B_{i,n}(u)$ é simétrico em relação a $u = 1/2$;
- $B_{i,n}(u)$ tem máximo no intervalo $0 \leq u \leq 1$;
- $B_{0,n}(0) = B_{1,n}(1) = 1$;
- São funções recursivas pois, $B_{i,n}(u) = (1-u)B_{i,n}(u) + uB_{i-1,n-1}(u)$, sendo que $B_{i,n}(u) = 0$ se $i < 0$ ou $i > n$;

Sua derivada é dada pela expressão seguinte:

$$\frac{dB_{i,n}}{du} = n(B_{i-1,n-1}(u) - B_{i,n-1}(u)), \tag{2.11}$$

em que $B_{-1,n-1}(u) = B_{n,n-1}(u) = 0$.

Para que as propriedades discutidas acima sejam válidas, toma-se como pressuposto que o parâmetro seja $0 \leq u \leq 1$.

Dessa forma, a partir da equação (2.11), é deduzida a expressão da derivada da curva de Bézier mostrada abaixo:

$$C'(u) = \sum_{i=0}^{n} B'_{i,n}(u)P_i = \sum_{i=0}^{n} n(B_{i-1,n-1}(u) - B_{i,n-1}(u))P_i = n\sum_{i=0}^{n-1} B_{i,n-1}(u)(P_{i+1} - P_i) \tag{2.12}$$

sendo a derivada de uma curva de Bézier de grau n uma curva de Bézier de grau $n-1$. Para que seja possível a representação exata de curvas cônicas, é preciso realizar uma divisão de polinômios. Logo, uma função racional da curva de Bézier é definida.

2.2.1 Curvas de Bézier Racionais

De acordo com [15], todas as curvas cônicas (incluindo o círculo), podem ser representadas por funções racionais, que são definidas pela razão de dois polinômios. Dessa forma, eles são representados pelas funções racionais na forma:

$$x(u) = \frac{X(u)}{W(u)} \qquad y(u) = \frac{Y(u)}{W(u)}, \tag{2.13}$$

em que *X(u), Y(u) e W(u)* são polinômios, sendo que cada função coordenada possui o mesmo denominador.

A definição de uma curva de Bézier racional de grau n é representada a seguir:

$$C(u) = \frac{\sum_{i=0}^{n} B_{i,n}(u) w_i P_i}{\sum_{i=0}^{n} B_{i,n}(u) w_i}, \qquad 0 \leq u \leq 1, \tag{2.14}$$

em que P_i e de $B_{i,n}$ são os mesmos de antes e w_i são valores escalares chamados de pesos. Logo, $W(u) = \sum_{i=0}^{n} B_{i,n}(u) w_i$ é a função denominador comum. Assume-se que $w_i > 0$ para todo i. Isso assegura que $W(u) > 0 \ \forall \ u \in [0, 1]$.

Reescreve-se a expressão (2.14) da seguinte maneira:

$$C(u) = \sum_{i=0}^{n} R_{i,n}(u) P_i, \qquad 0 \leq u \leq 1, \tag{2.15}$$

em que

$$R_{i,n}(u) = \frac{B_{i,n}(u) w_i}{\sum_{j=0}^{n} B_{j,n}(u) w_j}. \tag{2.16}$$

A expressão (2.16) representa as funções base de Bernstein nas formas racionais para esta curva. A Figura 2.3a representa uma função base cúbica, enquanto que a Figura 2.3b corresponde a uma curva de Bézier racional cúbica. Algumas propriedades desta função de base são mostradas a seguir:

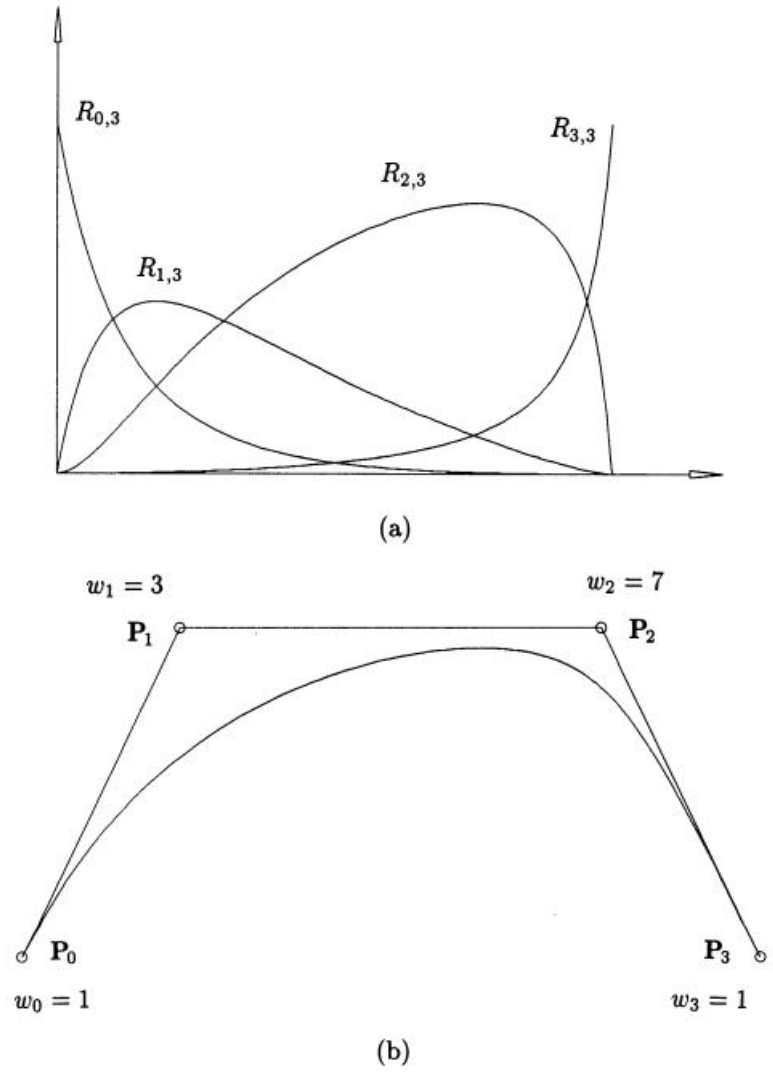

Figura 2.3: *Cúbico racional. (a) Funções de base; (b) Curva de Bézier Racional* [18].

- São sempre não negativas;
- $\sum_{i=0}^{n} R_{i,n}(u) = 1$;
- $R_{0,n}(0) = R_{1,n}(1) = 1$;
- $R_{i,n}(u)$ obtém um máximo intervalo $0 \leq u \leq 1$;
- Se $w_i = 1$ para todo i, então $R_{i,n}(u) = B_{i,n}(u)$ para todo i, isto é, $B_{i,n}(u)$ são casos especiais de $R_{i,n}(u)$.

Conclui-se que esta função base possui as mesmas características que a função base anterior, divergindo apenas a simetria com relação ao valor intermediário do parâmetro. Uma forma peculiar de representar os pontos das curvas racionais é a partir de coordenadas homogêneas. Estas permitem fácil aplicação de transformadas geométricas e são, basicamente, a representação da curva racional de dimensão n em uma curva polinomial de dimensão $n + 1$ coordenadas.

2.3 B-Spline

Uma curva B-spline, em que o "B" é proveniente da palavra base, é formada por várias curvas de Bézier que possuam segmentos com certo grau de continuidade nos pontos de união, independente da função dos segmentos. Ela pode ser definida através da fórmula recorrente cuja função base é mostrada na Figura 2.4, sendo que:

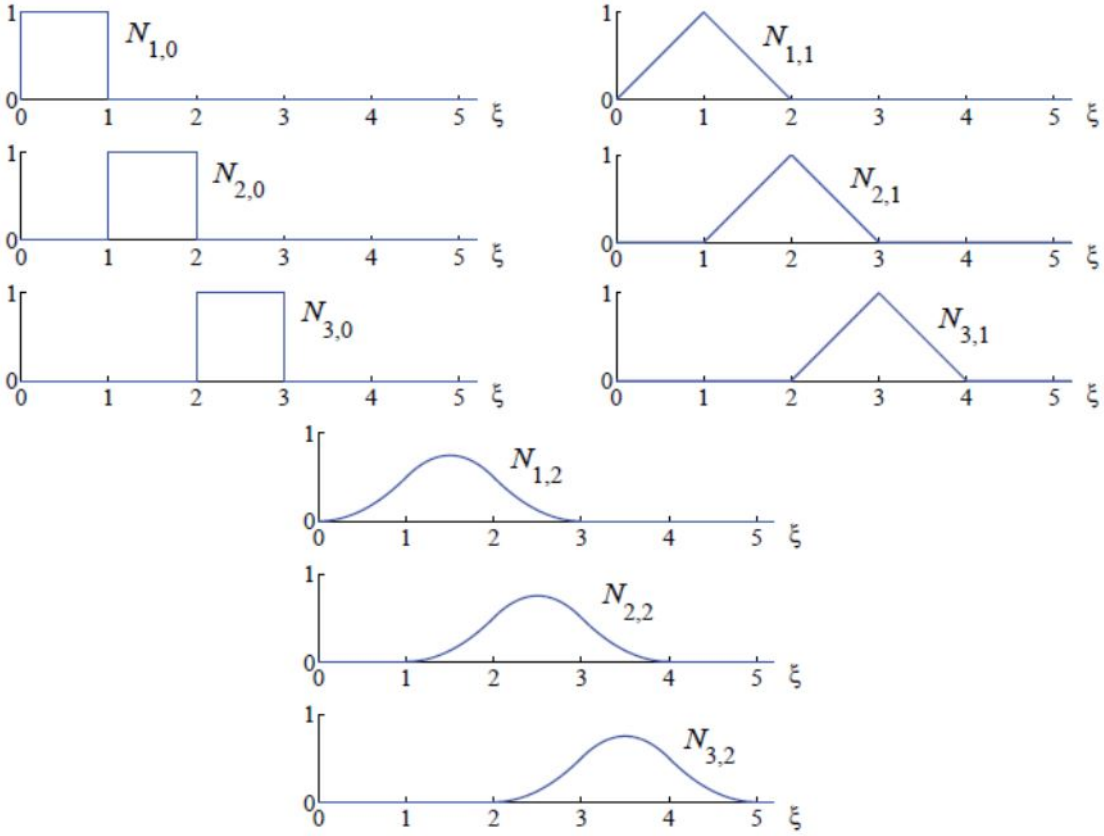

Figura 2.4: *Função base de ordem 0, 1 e 2 para o vetor nós uniforme* $u = 0, 1, 2, 3, ...$[10]

$$N_{i,0} = \begin{cases} 1, & \text{se} \quad u_i \leq u \leq u_{i+1} \\ 0, & \text{demais casos} \end{cases}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_i + 1} N_{i+1,p-1}(u) \tag{2.17}$$

Os valores de u_i são chamados nós e formam o vetor nós **U**. O vetor **U** deve ser composto de valores reais não decrescentes, sendo assim, $u_i \leq u_{i+1}$. Para que a função de base seja calculada, é necessário que o vetor nós e o grau p da curva sejam dados.

Algumas propriedades das funções de base são:

- $N_{i,p} = 0$, se $u \notin [u_i, u_{i+p+1})$;

- $N_{i,p} \geq 0$ em todos os casos;
- $N_{i,p}$ atinge apenas um máximo, exceto se $p = 0$;
- Para um dado intervalo de nós $[u_i, u_{i+p+1})$ ao qual pertence u, $\sum_{j=i-p}^{i} N_{j,p} = 1$;
- Em um nó, $N_{i,p}$ é diferenciável $p - k$ vezes, sendo k a multiplicidade do nó. Desta forma, aumentando-se o grau da curva ou diminuindo a multiplicidade dos nós, aumenta-se o nível de continuidade;
- A derivada da função base é:

$$N_{i,p}^{(K)}(u) = p(\frac{N_{i,p-1}^{(K-1)}}{u_{i+p} - u_i} - \frac{N_{i+1,p-1}^{(K-1)}}{u_{i+p+1} - u_{i+1}}) \tag{2.18}$$

em que $N_{i,p}^{(K)}$ refere-se à K-ésima derivada.

Sabe-se também que:

- As bases são sempre maiores ou iguais a zero;
- A curva não é afetada por uma transformação afim;
- A curva está sempre contida na envoltória convexa dos pontos de controle;
- Cada ponto da curva é calculado como uma soma ponderada de parte dos pontos de controle. Logo, uma mudança de um ponto de controle afetará a curva apenas localmente.

Ao utilizar esta função de base, encontra-se a expressão da B-Spline não racional a seguir:

$$C(u) = \sum_{i=0}^{n} N_{i,p}(u) P_i, \quad a \leq u \leq b \tag{2.19}$$

sendo o seu vetor nó representado da seguinte maneira:

$$\mathbf{U} = [\underbrace{a, \ldots, a}_{p+1}, u_{p+1}, \ldots, u_{m-p-1}, \underbrace{b, \ldots, b}_{p+1}] \tag{2.20}$$

em que $m + 1$ é o número de nós. Essas curvas possuem as seguintes características:

- $m = n + p + 1$;
- Se $n = p$, a B-Spline é uma curva de Bézier;
- $C(0) = P_0$ e $C(u_{m+1}) = P_n$;
- O controle da geometria da curva pode ser efetuado localmente, isto é, mover um ponto de controle P_i afeta apenas o intervalo $[u_i, u_{i+p+1})$;

- É possível fazer com que vários pontos de controle sejam coincidentes. Isto pode ser usado para formar ângulos com a curva, por exemplo;
- Um vetor nó de forma $\mathbf{U} = [\underbrace{0, \ldots, 0}_{p+1}, \underbrace{1, \ldots, 1}_{p+1}]$ leva a um polinômio de Bernstein de grau p;
- Sua derivada é dada pela expressão seguinte:

$$C^{(K)}(u) = \sum_{i=0}^{n} N_{i,p}^{(K)}(u) P_i = p \sum_{i=0}^{n-K} N_{i,p-K}(u) P_i^{(K)}, \tag{2.21}$$

em que tem-se:

$$P_i^{(K)} = \begin{cases} P_i, & \text{se} \quad K = 0 \\ \frac{p-1+K}{u_{i+p+1} - u_{i+K}} (P_{i+1}^{K-1} - P_i^{K-1}), & \text{se} \quad K > 0. \end{cases}$$

Dessa forma, o novo vetor nós $\mathbf{U}$ é definido excluindo os valores iniciais e finais do original de forma que:

$$\mathbf{U} = [\underbrace{a, \ldots, a}_{p-K+1}, u_{p+1}, \ldots, u_{m-p-1}, \underbrace{b, \ldots, b}_{p-K+1}]. \tag{2.22}$$

2.4 NURBS

As curvas NURBS (B-Splines racionais não uniformes) consistem numa união de temas já discutidos: curvas Bézier racionais, B-Splines e coordenadas homogêneas. As curvas são controladas utilizando pesos, como mostra a equação (2.23) a seguir:

$$C(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i P_i}{\sum_{i=0}^{n} N_{i,p}(u) w_i}, \quad a \leq u \leq b. \tag{2.23}$$

Para um dado vetor nós na forma:

$$\mathbf{U} = [\underbrace{a, \ldots, a}_{p+1}, u_{p+1}, \ldots, u_{m-p-1}, \underbrace{b, \ldots, b}_{p+1}], \tag{2.24}$$

a representação em coordenadas homogêneas seria uma B-spline dada por:

$$C(u)^w = \sum_{i=0}^{n} N_{i,p}(u) P_i^w. \tag{2.25}$$

Como dito, as NURBS são controladas por pesos e o efeito da variação de pesos é explícito na Figura 2.5 a seguir:

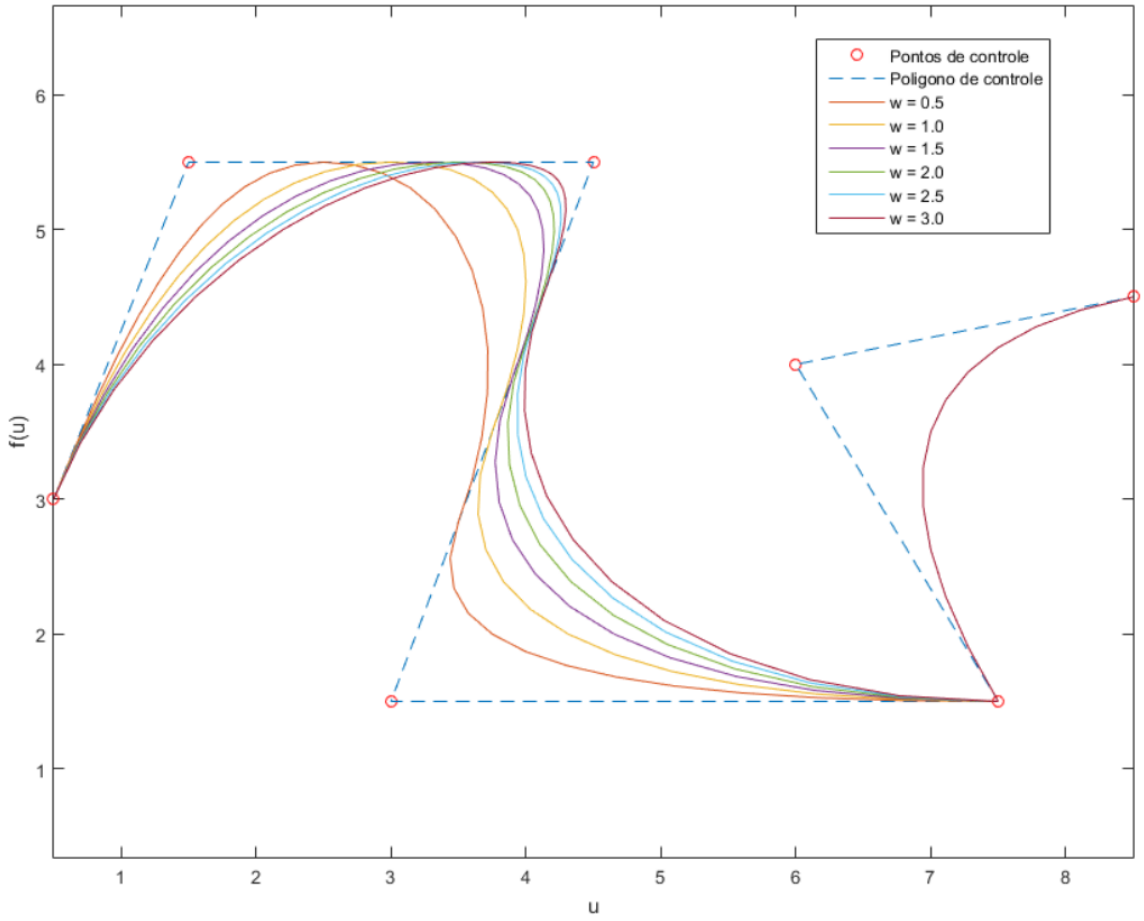

Figura 2.5: *Função aproximada com NURBS e diferentes valores de peso para o terceiro ponto de controle.*

Neste exemplo é mostrado um caso de curva interpolada utilizando NURBS, em que há uma variação nos valores dos pesos, sendo possível concluir que quanto maior for os valores designados para os mesmos (no nó em questão), mais próxima a curva estará daquele ponto de controle. Isso será melhor detalhado mais a frente neste capítulo.

As NURBS possuem algumas características, tais como:

- A função base racional é sempre não negativa:

$$R_{i,u}(u) = \frac{N_{i,p}(u)w_i}{\sum_{i=0}^{n} N_{i,p}(u)w_i} \geq 0 \tag{2.26}$$

- $\sum_{i=0}^{n} R_{i,p}(u) = 1$;
- $R_{0,p}(0) = R_{1,p}(1) = 1$;
- $R_{i,p}(u)$ obtém um máximo no intervalo $0 \leq u \leq$ para $p > 0$;
- $R_{i,p} = 0$, se $u \notin [u_i, u_{i-p+1})$;
- $R_{i,p} = N_{i,p}$, se $w_i \forall i$;

- A curva de Bézier racional é um caso especial de NURBS. Isso ocorre quando não há nós internos;
- O controle da geometria da curva pode ser efetuado localmente, ou seja, mover um ponto de controle P_i afeta apenas o intervalo $[u_i, u_{i+p+1})$;
- É possível fazer com que vários pontos de controle sejam coincidentes;
- Sua derivada pode ser escrita de acordo com a relação:

$$C^{(K)}(u) = \frac{A^{(K)}(u) - \sum_{i=1}^{K} \binom{K}{i} w^{(i)}(u) C^{(K-1)}(u)}{w(u)}, \tag{2.27}$$

em que **A** é o vetor formado pelas primeiras três coordenadas de $C(u)^w$ e $w(u)$ é a coordenada referente aos pesos nessa relação.

2.4.1 O efeito dos pesos nas NURBS

Com o auxilio da biblioteca *nurbs_toolbox* [19], uma rotina foi feita para mostrar o efeito dos pesos nas NURBS de forma detalhada. Dessa forma, fez-se um exemplo para que seja possível realizar tal detalhamento, sendo realizadas as seguintes etapas:

- Escolha das coordenadas dos pontos de controle e do polígono de controle, sendo $\{P_0, ..., P_4\} = \{(0,0), (1,1), (3,2), (4,1), (5,-1)\}$:

```
pntsC = [0 1 3 4 5;    % Coordenadas em x
         0 1 2 1 -1]; % Coordenadas em y
```

Figura 2.6: *Matriz com as coordenadas dos pontos de controle.*

- Em seguida, é feita a interpolação utilizando as NURBS, em que são mantidos fixos os valores dos pesos (iguais a 1) em todos os pontos $\{P_0, ..., P_4\}$, menos no ponto P_1, em que é feita uma variação dos pesos de 0 até 4 (com um passo de uma unidade). Dessa forma, tem-se o peso como sendo $\{w_0, ..., w_4\} = \{1, w_1, 1, 1, 1\}$. A figura ilustra o comportamento da curva interpolada com a variação do valor do peso no ponto P_1:

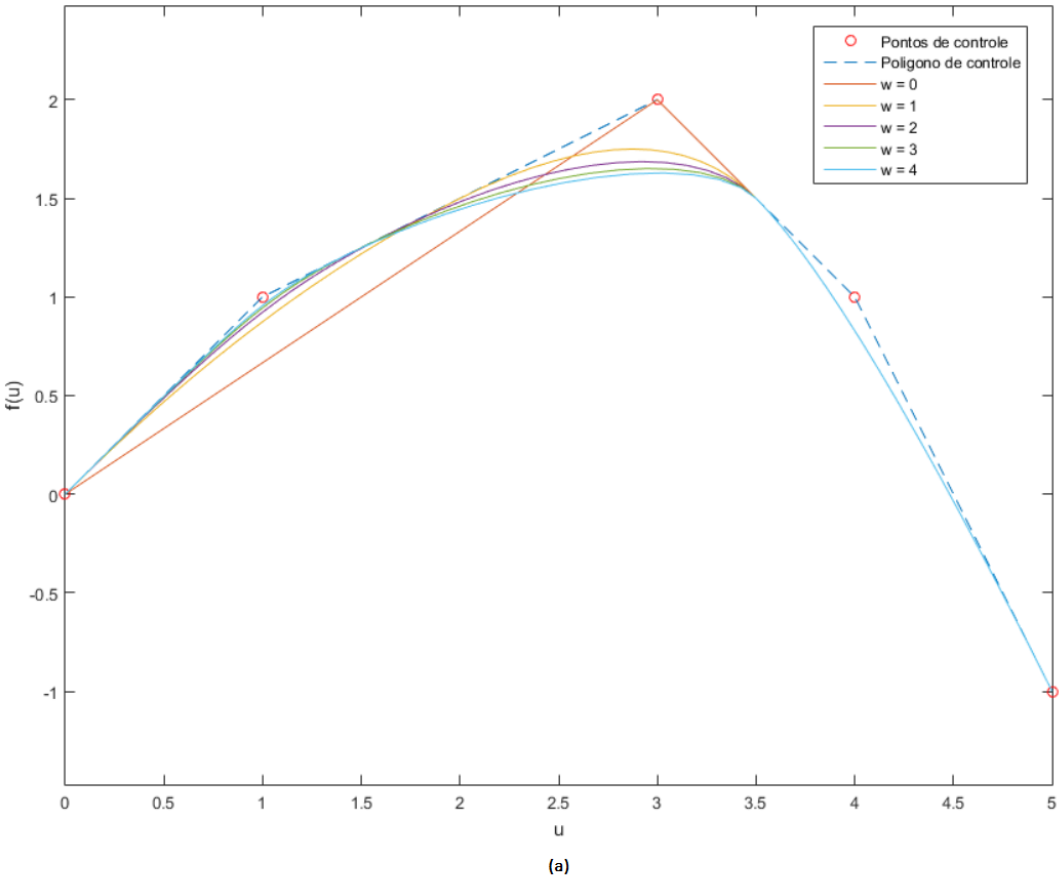

Figura 2.7: *(a) Interpolação usando NURBS sendo variados os pesos no ponto P_1. (b) Imagem aproximada no ponto P_1.*

Sendo assim, a partir das expressões em (2.17) e calculando o ponto na curva racional B-Spline em $u = 1$, tem-se que:

$$
\begin{aligned}
N_{3,0}(1) &= 1 \\
N_{2,1}(1) &= \frac{2-1}{2-1} N_{3,0}(1) = 1 \\
N_{3,1}(1) &= \frac{1-1}{2-1} N_{3,0}(1) = 0 \\
N_{1,2}(1) &= \frac{2-1}{2-0} N_{2,1}(1) = 1/2 \\
N_{2,2}(1) &= \frac{1-0}{2-0} N_{2,1}(1) = 1/2 \\
N_{3,2}(1) &= 0
\end{aligned}
\tag{2.28}
$$

Já as representações gráficas das funções de base quadráticas das curvas presentes na Figura 2.7 são mostradas a seguir:

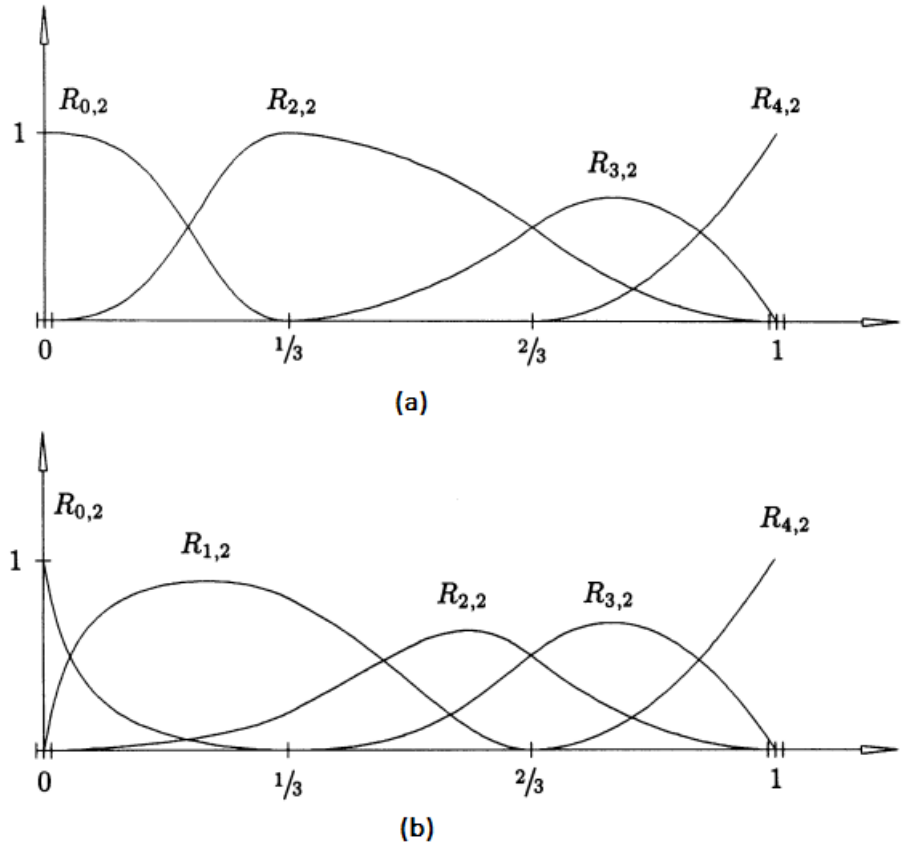

Figura 2.8: *Funções de base quadráticas para as curvas. (a)* $w_1 = 0$. *(b)* $w_1 = 4$ [18].

3 O MÉTODO DOS ELEMENTOS DE CONTORNO ISOGEOMÉTRICO

Neste capitulo é feita uma análise da formulação do MEC para a obtenção equação integral de contorno, sendo utilizada a formulação isogeométrica do método dos elementos de contorno.

3.1 Equação Integral de Contorno

Neste trabalho a formulação desenvolvida do MEC será para problemas descritos pela seguinte equação diferencial:

$$\nabla^2 u = f, \tag{3.1}$$

conhecida como a equação governante da teoria potencial. Para $f = 0$, a equação (3.1) é conhecida como equação de Laplace. Para $f \neq 0$, ela é conhecida como equação de Poisson. Os problemas que envolvem este tipo de equação são diversos, como a condução térmica em transferência de calor, a torção de barras não circulares, deflexão de membranas elásticas, flexão de placas apoiadas e dentre outros.

Uma vez que as aplicações neste trabalho serão em problemas de condução de calor sem geração, a formulação do MEC será desenvolvida para a equação de Laplace (particularização da equação de Poisson) escrita da seguinte forma:

$$\nabla^2 T = 0. \tag{3.2}$$

Ao multiplicá-la por um peso $w(x, y)$ e integrando-a sobre o domínio A, assume-se que o resultado da integral seja zero. Desta forma, tem-se que:

$$\int\int_A (\nabla^2 T) w dA = 0, \tag{3.3}$$

$$\int\int_A (\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}) w dA = 0, \tag{3.4}$$

$$\int\int_A \frac{\partial^2 T}{\partial x^2} w dA + \int_A \frac{\partial^2 T}{\partial y^2} w dA = 0. \tag{3.5}$$

Através do teorema de Gauss-Green, tem-se que:

$$\int_s f(x, y) n_x ds = \int_A \frac{\partial f}{\partial x} dA, \tag{3.6}$$

em que f é uma função dada qualquer, n_x é a componente na direção x do vetor $\mathbf{n}$ normal ao

contorno s da área A. Ao aplicar o teorema de Gauss-Green à equação (3.3), tem-se que:

$$\int_s \frac{\partial T}{\partial x} w n_x ds = \int_A \frac{\partial}{\partial x}(\frac{\partial T}{\partial x} w) dA. \tag{3.7}$$

Ao aplicar a regra da derivada do produto, tem-se:

$$\int_s \frac{\partial T}{\partial x} w n_x ds = \int_A \frac{\partial^2 T}{\partial x^2} w dA + \int_A \frac{\partial T}{\partial x}\frac{\partial w}{\partial x} dA. \tag{3.8}$$

Rearranjando a equação anterior de forma conveniente, tem-se que:

$$\int_A \frac{\partial^2 T}{\partial x^2} w dA = \int_s \frac{\partial T}{\partial x} w n_x ds - \int_A \frac{\partial T}{\partial x}\frac{\partial w}{\partial x} dA. \tag{3.9}$$

De maneira análoga, para n_y como componente na direção y do vetor normal $\mathbf{n}$ ao contorno s de área A, tem-se que:

$$\int_A \frac{\partial^2 T}{\partial y^2} w dA = \int_s \frac{\partial T}{\partial y} w n_y ds - \int_A \frac{\partial T}{\partial y}\frac{\partial w}{\partial y} dA. \tag{3.10}$$

Ao substituir as equações (3.9) e (3.10) na equação (3.5), tem-se que:

$$\int_s (\frac{\partial T}{\partial x} w n_x + \frac{\partial T}{\partial y} w n_y) ds - \int_A (\frac{\partial T}{\partial x}\frac{\partial w}{\partial x} + \frac{\partial T}{\partial y}\frac{\partial w}{\partial y}) dA = 0. \tag{3.11}$$

Podendo ser simplificada da seguinte forma:

$$\int_s \frac{\partial T}{\partial n} w ds - \int_A (\frac{\partial T}{\partial y}\frac{\partial w}{\partial y}) dA = 0. \tag{3.12}$$

Ao considerar algumas igualdades no lado direito da equação (3.12), tem-se que:

$$\int_s \frac{\partial T}{\partial n} w ds - \int_s T\frac{\partial w}{\partial n} ds + \int_A T\nabla^2 w dA = 0. \tag{3.13}$$

Com o intuito de obter uma equação integral que não possua integrais de domínio (de área), a função peso w deve ser escolhida de forma que a integral de domínio desapareça. Em uma função na qual o Laplaciano é igual a zero, esta exigência é respeitada. Porém a escolha mais adequada é uma função cujo Laplaciano é o delta de Dirac:

$$\nabla^2 w = -\frac{\delta(x - x_d, y - y_d)}{k}. \tag{3.14}$$

Neste caso, tem-se que:

$$w = T^* = \frac{-1 \ln r}{2\pi k}, \tag{3.15}$$

em que T^* é conhecida como a solução fundamental. Logo, tem-se que:

$$\int_s \frac{\partial T}{\partial n} T^* ds - \int_s T \frac{\partial T^*}{\partial n} ds + \int_A T \frac{[-\delta(x - x_d)]}{k} dA = 0, \tag{3.16}$$

em que (x_d, y_d) é a coordenada do ponto fonte (colocação). O ponto fonte pode pertencer ao domínio, pertencer ao contorno ou não pertencer nem ao domínio e nem ao contorno.

Com o ponto fonte no interior do domínio A, tem-se que:

$$T(x_d, y_d) = \int_s T q^* ds - \int_s T^* q ds, \tag{3.17}$$

em que a equação (3.17) é a equação integral de contorno quando o ponto fonte encontra-se no interior do domínio. O termo q^* refere-se a solução fundamental de fluxo e é dada por:

$$q^* = \frac{1}{2\pi r^2}(r_x n_x + r_y n_y) \tag{3.18}$$

Com o ponto fonte (x_d, y_d) no contorno, é feita a seguinte alteração (Figura 3.1):

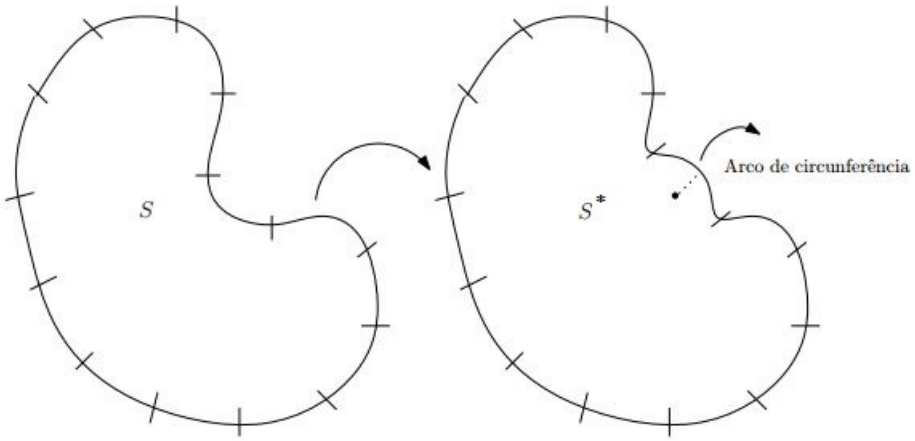

Figura 3.1: *Contornos original S e adaptado S^** [5].

Logo, faz-se a seguinte alteração:

$$T(x_d, y_d) = \int_{s-s^*} T q^* ds - \int_{s-s^*} T^* q ds + \int_{s^*} T q^* ds - \int_{s^*} T^* q ds. \tag{3.19}$$

Para um fluxo de calor do contorno, a solução fundamental de fluxo de calor é:

$$q^* = \frac{1}{2\pi r^2}[(x - x_d)n_x + (y - y_d)n_y], \tag{3.20}$$

sendo $r = [(x - x_d)^2 + (y - y_d)^2]^{1/2}$. Desta forma, tem-se que:

$$\int_{s^*} Tq^* ds = \int_{\theta_1}^{\theta_2} T \frac{1}{2\pi r^2}[(x - x_d)n_x + (y - y_d)n_y]ds$$

.

Em s^*, tem-se que:

$$\mathbf{r} = (x - x_d)\mathbf{i} + (y - y_d)\mathbf{j}$$
$$|\mathbf{r}| = \mathbf{r} = \sqrt{(x - x_d)^2 + (y - y_d)^2}$$
$$\mathbf{n} = \frac{(x - x_d)\mathbf{i} + (y - y_d)\mathbf{j}}{r},$$

em que $\mathbf{n}$ é um vetor unitário. Se $r_x = (x - x_d)$ e $r_y = (y - y_d)$, tem-se que:

$$\mathbf{n} = \frac{r_x\mathbf{i} + r_y\mathbf{j}}{r}$$

e

$$n_x = \frac{r_x}{r} \quad \text{e} \quad n_y = \frac{r_y}{r}.$$

Logo,

$$\int_{s^*}^{s} Tq^* ds = \int_{\theta_1}^{\theta_2} \frac{T}{2\pi r^2}(r_x \frac{r_x}{r} + r_y \frac{r_y}{r})\varepsilon d\theta.$$

Para $r = \varepsilon$ e para qualquer θ, tem-se:

$$\int_{s^*} Tq^* ds = \int_{\theta_1}^{\theta_2} \frac{T}{2\pi} d\theta.$$

Ao fazer $\varepsilon \to 0$, T assume o valor de $T(d)$. Logo,

$$\int_{s^*} Tq^* ds = \frac{T(d)(\theta_2 - \theta_1)}{2\pi}.$$

Faz-se a mesma análise para

$$\int_{s^*} Tq^* ds = \int_{\theta_1}^{\theta_2} \frac{-1}{2\pi r} \ln r q r d\theta.$$

Como $r = \varepsilon$, que é uma constante, tem-se que:

$$\int_{s^*} Tq^* ds = \frac{-1}{2\pi k} \varepsilon \ln \varepsilon \int_{\theta_1}^{\theta_2} q d\theta.$$

Ao fazer $\varepsilon \to 0$, tem-se que:

$$\int_{s^*} Tq^* ds = 0.$$

Ao retornar a equação (3.19), tem-se que:

$$T(x_d, y_d) = \int_s Tq^* ds - \int_s T^* q ds + \frac{T(x_d, y_d)(\theta_2 - \theta_1)}{2\pi} - 0$$

$$T(x_d, y_d) = [1 - \frac{(\theta_2 - \theta_1)}{2\pi}] = \int_s Tq^* ds - \int_s T^* q ds$$

$$T(x_d, y_d) = [\frac{2\pi - (\theta_2 - \theta_1)}{2\pi}] = \int_s Tq^* ds - \int_{T^* q ds}^{max}.$$

Na Figura 3.2 a seguir é possível compreender como é o ângulo interno de contorno.

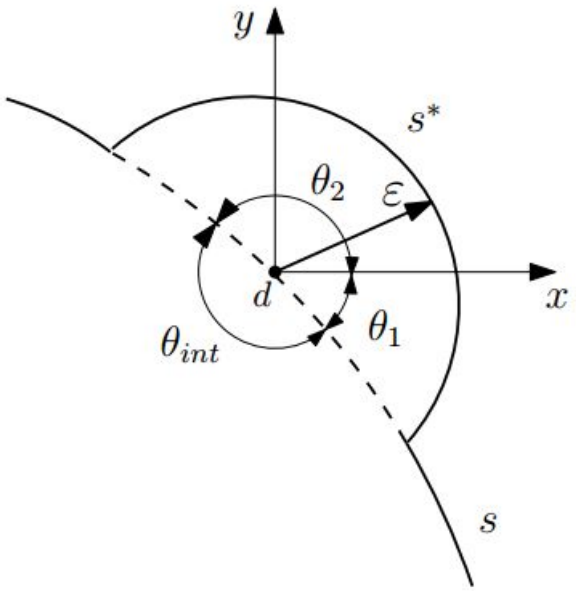

Figura 3.2: *Ângulo interno do contorno* [7].

Com os valores encontrados pra cada elemento, tem-se a equação integral de contorno a seguir:

$$\frac{\theta_{int}}{2\pi} T(x_d, y_d) = \int_s Tq^* ds - \int_s T^* q ds, \tag{3.21}$$

em que θ_{int} é o angulo interno de contorno.

Por fim, quando o ponto fonte não pertence ao contorno e nem ao domínio, tem-se que:

$$\int_s Tq^* ds - \int_s T^* q ds = 0. \tag{3.22}$$

Desta forma, uma generalização pode ser feita e a equação integral de contorno pode ser

escrita como:

$$cT(x_d, y_d) = \int_s Tq^* ds - \int_s T^* q ds, \tag{3.23}$$

em que

$$c = \begin{cases} 1, & \text{se} \quad (x_d, y_d) \in \text{ao dominio} \\ \frac{\theta_{int}}{2\pi}, & \text{se} \quad (x_d, y_d) \in \text{ao contorno} \\ 0, & \text{se} \quad (x_d, y_d) \notin \text{ao dominio ou ao contorno} \end{cases}$$

Quando o ponto de colocação/fonte encontra-se em um ponto suave do contorno, ou seja, não é um canto, tem-se que:

$$c = \frac{\theta_{int}}{2\pi} = \frac{\pi}{2\pi} = \frac{1}{2}. \tag{3.24}$$

3.2 Cálculo da temperatura e do fluxo de calor em pontos internos

A temperatura e o fluxo de calor em pontos internos ao domínio S podem ser calculados de maneira simples, uma vez que os valores do contorno já foram calculados em um primeiro momento [7]. Basta considerar o ponto de colocação esteja localizado no interior do domínio e utilizar a equação integral (3.17). Como todas as integrais são calculadas no contorno, a única variável desconhecida é a temperatura no ponto interno. Com relação ao fluxo de calor, é preciso derivar a equação (3.17) em relação às coordenadas do ponto de colocação. Desta maneira, tem-se que:

$$\frac{\partial T(x_d, y_d)}{\partial x_d} = \frac{\partial}{\partial x_d}\left[\int_s Tq^* ds - \int_s qT^* ds\right] \tag{3.25}$$

e

$$\frac{\partial T(x_d, y_d)}{\partial x_d} = \int_s T\frac{\partial q^*}{\partial x_d} ds - \int_s q\frac{\partial T^*}{\partial x_d} ds, \tag{3.26}$$

em que:

$$\frac{\partial T^*}{\partial x_d} = \frac{r_x}{2\pi k r^2}, \tag{3.27}$$

$$\frac{\partial T^*}{\partial y_d} = \frac{r_y}{2\pi k r^2} \tag{3.28}$$

e que:

$$\frac{\partial q^*}{\partial x_d} = \frac{n_x(r_x^2 - r_y^2) + 2n_y r_x r_y}{2\pi r^4} \tag{3.29}$$

e

$$\frac{\partial q^*}{\partial y_d} = \frac{n_y(-r_x^2 + r_y^2) + 2n_y r_x r_y}{2\pi r^4}. \tag{3.30}$$

3.3 Formulação Integral de Contorno Discretizada

Na expressão (3.23), a equação integral de contorno do MEC para problemas potenciais pode ser escrita da seguinte maneira:

$$cT(x_d, y_d) = \int_s Tq^* ds - \int_s T^* q ds \tag{3.31}$$

sendo aplicada às coordenadas do ponto fonte x_d e y_d, T referente à temperatura e q referente ao fluxo de calor no contorno. Como neste trabalho utiliza-se o método isogeométrico, então as variáveis são aproximadas utilizando as mesmas funções das NURBS. A expressão (3.31) pode ser reescrita da seguinte forma, sabendo que $R_{i,k}(t)$ é a função base racional:

$$cT(x_d, y_d) = \int_\Gamma \sum_{i=1}^{n+1} \left(T_i^c R_{i,k}(t)\right) q^*(x_d, y_d) ds - \int_\Gamma T^*(x_d, y_d) \sum_{i=1}^{n+1} \left(q_i^c R_{i,k}(t)\right) ds, \tag{3.32}$$

em que T_i^c e q_i^c são a temperatura e o fluxo de calor, respectivamente, no ponto de controle i. Como estes valores são pontuais (não variam ao longo do contorno), pode-se organizá-los da seguinte forma:

$$cT(x_d, y_d) = \sum_{i=1}^{n+1} \left(T_i^c \int_\Gamma R_{i,k}(t) q^*(x_d, y_d) d\Gamma\right) - \sum_{i=1}^{n+1} \left(q_i^c \int_\Gamma T^*(x_d, y_d) R_{i,k}(t) d\Gamma\right) \tag{3.33}$$

Na equação (3.33) acima não existe aproximações na geometria, sendo estas apenas nas variáveis T_i^c e q_i^c. Seu contorno é parametrizado por t como mostrado a seguir:

$$cT(x_d, y_d) = \sum_{i=1}^{n+1} \left(T_i^c \int_{t_{min}}^{t_{max}} R_{i,k}(t) q^*(x_d, y_d) \frac{d\Gamma}{dt} dt\right) - \sum_{i=1}^{n+1} \left(q_i^c \int_{t_{min}}^{t_{max}} T^*(x_d, y_d) R_{i,k}(t) \frac{d\Gamma}{dt} dt\right) \tag{3.34}$$

em que

$$\frac{d\Gamma}{dt} = \sqrt{\left(\frac{dx(t)}{dt}\right)^2 + \left(\frac{dy(t)}{dt}\right)^2} \tag{3.35}$$

Cada função de base é não nula apenas em um intervalo único, porém intersectante. Esse intervalo pode ser entendido como o domínio de influência daquele ponto de controle. Esse domínio começa em t_i e termina em t_{i+k}. Nesta formulação isogeométrica, não se define elementos,

já que eles não seriam independentes: as integrais são regidas apenas pelo domínio de influência. Elas podem ser reduzidas para somente os intervalos não-nulos como a seguir:

$$cT(x_d, y_d) = \sum_{i=1}^{n+1} \left(T_i^c \int_{t_i}^{t_{i+k}} R_{i,k}(t) q^*(x_d, y_d) \frac{d\Gamma}{dt} dt \right) - \sum_{i=1}^{n+1} \left(q_i^c \int_{t_i}^{t_{i+k}} T^*(x_d, y_d) R_{i,k}(t) \frac{d\Gamma}{dt} dt \right) \tag{3.36}$$

Desta forma, torna-se simples calcular somente uma vez as funções de base e propriedades geométricas dos pontos de Gauss para cada coluna. É importante salientar que é mais caro computacionalmente calcular as funções de base das NURBS do que as funções polinomiais regulares, tornando-se interessante tomar estas precauções. Para o cálculo numérico das integrais, mais uma mudança de variáveis é necessária, sendo regularizado o intervalo de integração, o que possibilita utilizar a quadratura de Gauss.

$$cT(x_d, y_d) = \sum_{i=1}^{n+1} \left(T_i^c \int_{-1}^{1} R_{i,k}(t) q^*(x_d, y_d) \frac{d\Gamma}{dt} \frac{dt}{d\xi} d\xi \right) - \sum_{i=1}^{n+1} \left(q_i^c \int_{-1}^{1} T^*(x_d, y_d) R_{i,k}(t) \frac{d\Gamma}{dt} \frac{dt}{d\xi} d\xi \right), \tag{3.37}$$

em que

$$\frac{dt}{d\xi} = \frac{t_{i+k} - t_i}{2}. \tag{3.38}$$

Desta forma, a equação (3.37) pode ser reorganizada em termos de duas matrizes, **H** e **G**:

$$\sum_{i=1}^{n+1} \left(T_i^c \left(\int_{-1}^{1} R_{i,k}(t) q^*(x_d, yd) \frac{d\Gamma}{dt} \frac{dt}{d\xi} d\xi - cR_{i,k}(x_d, y_d) \right) \right) = \sum_{i=1}^{n+1} \left(q_i^c \int_{-1}^{1} T^*(x_d, y_d) R_{i,k}(t) \frac{d\Gamma}{dt} \frac{dt}{d\xi} d\xi \right) \tag{3.39}$$

$$\mathbf{HT} = \mathbf{Gq}$$

em que a matriz **H** representa o termo à esquerda de (3.39). Percebe-se que o termo c tem influencia em k termos desta matriz, o que não ocorre no MEC tradicional. Já a matriz **G** é representada pelo termo à direita de (3.39).

3.4 Integrais regulares, quase-singulares, fracamente singulares e fortemente singulares

As integrais são divididas da seguinte forma quanto ao tipo de singularidade que surge durante a integração [1]:

- **Integrais Regulares:** caso em que o ponto de colocação não encontra-se dentro do elemento ou, caso contrário, o integrando envolve a solução fundamental T^* e a função de forma tende a zero, cancelando a singularidade.

- **Integrais quase-singulares:** mesmo caso de integrais regulares, exceto que o ponto de colocação é próximo ao elemento que está sendo integrado.

- **Integrais fracamente singulares:** caso em que o ponto de colocação pertence ao intervalo de integração, a função de forma não tende a zero enquanto o integrando envolve a solução fundamental T^*.

  Uma técnica utilizada no MEC para calcular indiretamente as integrais singulares é a consideração de um corpo de temperatura constante, no caso de problemas potenciais, e o deslocamento de corpo rígido, no caso de problemas elásticos. Essa técnica consiste em calcular primeiro todas as integrais não-singulares para um ponto de colocação em específico, e então o deslocamento de corpo rígido é aplicado para calcular a integral singular. Em funções polinomiais o número de integrais singulares é igual ao número de translações de corpo rígido. Isso ocorre porque, no MEC convencional, para cada ponto de colocação, o elemento singular correspondente tem somente uma função de base associada àquele elemento. Portanto só existe uma integral singular em cada elemento.

  Nas NURBS, as funções de base não tem essa característica, várias delas são não nulas nos pontos de colocação. Para cada ponto de colocação se tem n integrais singulares que precisam de tratamento especial.

  As integrais de natureza fracamente singular são calculadas pela transformada proposta por [18]. O método aplica uma transformação de coordenadas cúbicas de forma que as abscissas se concentram em torno da singularidade, e faz com que o jacobiano dessa transformação se anule no ponto singular. Esta transformação é dada por:

$$\xi = \frac{(\gamma - \gamma')^3 + \gamma'(\gamma'^2 + 3)}{1 + 3\gamma'^2}, \tag{3.39}$$

  em que

$$\xi' = \sqrt[3]{\xi(\xi^2 - 1) + |\xi^2 - 1|} + \sqrt[3]{\xi(\xi^2 - 1) + |\xi^2 - 1| + \xi} \tag{3.40}$$

  e ξ' é a localização da singularidade no espaço original e γ representa a nova variável de integração. Desta forma, o jacobiano dessa transformação é:

$$d\xi = \frac{3(\gamma - \gamma')^2}{1 + 3\gamma'^2} d\gamma. \tag{3.41}$$

  A formulação mostrada é unidimensional. No entanto, a extensão para duas dimensões é direta, uma vez que cada conjunto de abcissas é transformado separadamente e os pesos multiplicados pelo jacobiano.

A transformada de Telles apresenta um método eficiente para o cálculo de integrais fracamente singulares e quase-singulares. Essa transformação polinomial de terceiro grau melhora a aproximação da quadratura de Gauss dentro da faixa perto da singularidade [20]. Dessa forma, essa técnica pode ser aplicada de maneira simples no método dos elementos de contorno isogeométrico, apresentando uma característica auto-adaptável em que é produzida uma variável que depende da mínima distância do ponto de colocação ao elemento. No caso de longas distâncias do ponto de colocação, o método torna-se inativo.

Essa transformação pode ser para calcular integrais com uma singularidade logarítmica em uma de suas extremidades. Uma grande vantagem é que a integração de Gauss tradicional pode ser aplicada sem a necessidade de separação da parte regular da singular, uma vez que o Jacobiano da transformada é nulo na posição da singularidade.

A escolha dos pontos de integração é baseada numa distância mínima relativa do elemento ao ponto de colocação. Como esse método, a posição dos pontos de Gauss é alterada para aproximá-los da singularidade, para que o comportamento singular da solução fundamental seja bem descrito. A parcela da solução fundamental que não seria incluída na integração caso a transformação de Telles não fosse utilizada é representada pela área hachurada. Isso pode ser melhor compreendido na Figura 3.3:

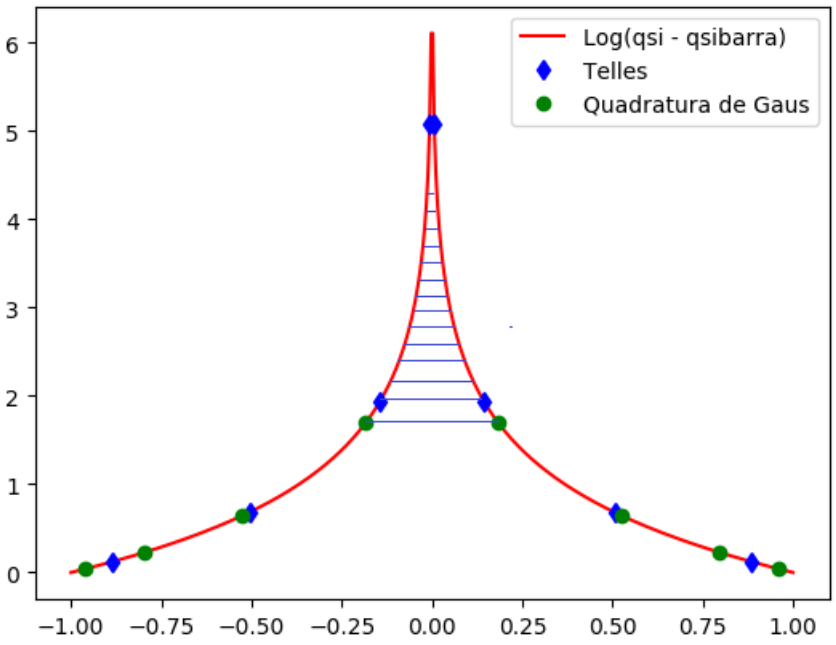

Figura 3.3: *Subdivisão de um subelemento.*

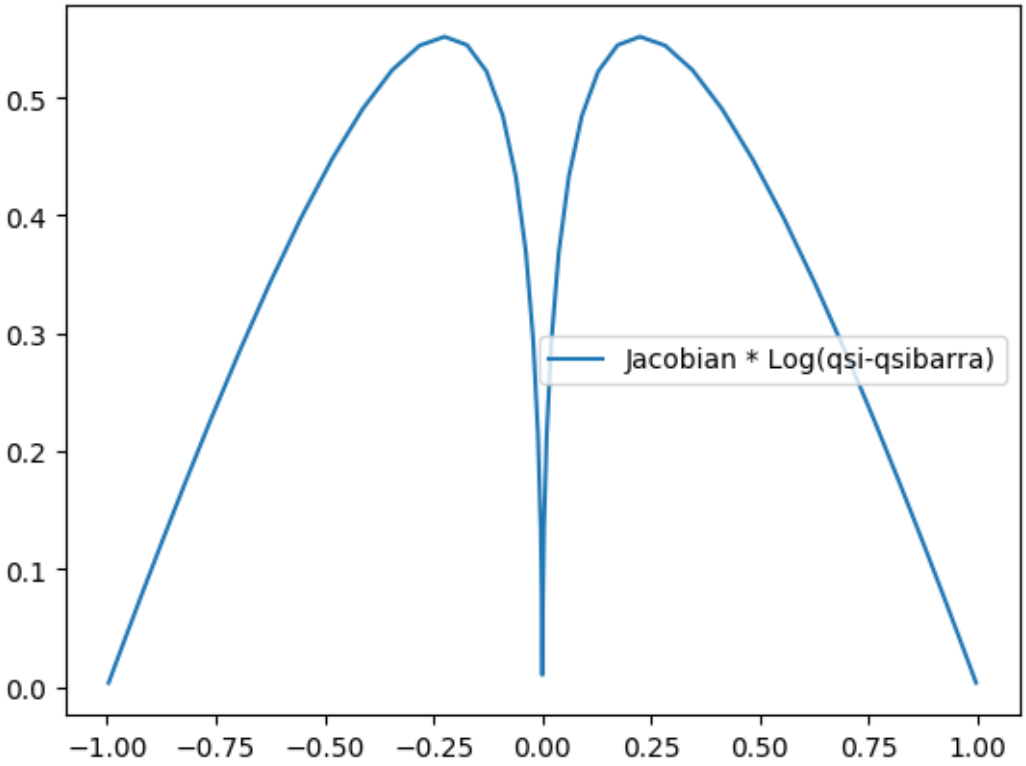

Figura 3.4: *Resultado da convergência no ponto de singularidade.*

Percebe-se que, com o auxilio da Transformada de Telles, integrais com certo grau de singularidades podem ser melhor calculadas do que quando comparadas com resultados obtidos utilizando integração de Gauss padrão.

- **Integrais fortemente singulares:** caso em que o ponto de colocação encontra-se dentro do elemento e o integrando envolve a solução fundamental q^*. Não existem integrais fortemente singulares na formulação de Laplace [8].

3.5 Pontos de colocação e condições de contorno

Os nós utilizados para definir as NURBS e os pontos de controle nem sempre são únicos, podendo ter uma multiplicidade de até o grau da curva. Além do mais, os pontos de controle estão no polígono de controle, e não na curva. Portanto, essas características intrínsecas fazem com que estas entidades não sejam as melhores escolhas para os pontos de colocação no MEC.

Como os pontos de controle estão tipicamente fora do contorno, as condições de contorno não podem ser aplicadas diretamente. Para corrigir este problema uma matriz de transformação **E** para B-splines é proposta [21]. Essa matriz usa as funções de base para relacionar os valores nos pontos de controle com os valores nos pontos de colocação.

A variação da função estudada pode ser representada por B-splines e, desta forma, preservar a continuidade das derivadas de primeira e segunda ordem entre elementos adjacentes.

Os valores do potencial (temperatura) e das derivadas do potencial (fluxo de calor) usando B-splines como funções de interpolação são, respectivamente, dados como:

$$T(t) = E_0(t)a_{1-1} + E_1(t)a_1 + ... + E_n(t)a_{i+(n-1)}, \tag{3.42}$$

$$q(t) = E_0(t)b_{1-1} + E_1(t)b_1 + ... + E_n(t)b_{i+(n-1)} \tag{3.43}$$

em que os nós correspondem a um intervalo entre i e $i-1$. a_i (pontos de controle) são coeficientes da representação do potencial e b_i são coeficientes da representação das derivadas do potencial. Já os elementos de $E_n(t)$ correspondem à função de transformação.

Podem ser escritos na forma matricial da seguinte maneira:

$$T(t) = \begin{bmatrix} E_0(t) & E_1(t) & ... & E_n(t) \end{bmatrix} \begin{bmatrix} a_{i-1} \\ a_i \\ ... \\ a_{i+(n-1)} \end{bmatrix}$$

$$q(t) = \begin{bmatrix} E_0(t) & E_1(t) & ... & E_n(t) \end{bmatrix} \begin{bmatrix} b_{i-1} \\ b_i \\ ... \\ b_{i+(n-1)} \end{bmatrix}.$$

Os coeficientes da matriz $\mathbf{H}$ são obtidos da seguinte maneira:

$$h_t^{ij} = -\frac{1}{2\pi} \int_{t=0}^{t=1} E_t(t) \frac{1}{r} \frac{\partial r}{\partial n} |G| dt \tag{3.44}$$

e os coeficientes da matriz $\mathbf{G}$ são obtidos da seguinte forma:

$$g_t^{ij} = -\frac{1}{2\pi} \int_{t=0}^{t=1} E_t(t) \ln(\frac{1}{r}) |G| dt. \tag{3.45}$$

As condições de contorno do problema especificam a temperatura ou o fluxo de calor do contorno, embora em geral os coeficientes a_i e b_i não são conhecidos.

Ao assumir que

$$\mathbf{a} = \mathbf{E}^{-1}\mathbf{T} \tag{3.46}$$

e

$$\mathbf{b} = \mathbf{E}^{-1}\mathbf{q} \tag{3.47}$$

aplicam-se as equações (3.46) e (3.47) ao sistemas de equações dos elementos de contorno, tem-se

que:

$$\mathbf{H}\mathbf{E}^{-1}\mathbf{T} = \mathbf{G}\mathbf{E}^{-1}\mathbf{q} = \mathbf{G}\mathbf{E}^{-1}\frac{\partial \mathbf{T}}{\partial \mathbf{n}}. \tag{3.48}$$

Esse é o resultado apos a aplicação da matriz de transformação, capaz de corrigir as condições de contorno previamente obtidas.

Sendo assim, a partir da expressão (3.39), tem-se que:

$$H_{i,k} = T_i^c \left(\int_{-1}^{1} R_{i,k}(t) q^*(x_d, yd) \frac{d\Gamma}{dt} \frac{dt}{d\xi} d\xi - cR_{i,k}(x_d, y_d) \right) \tag{3.49}$$

e

$$G_{i,k} = q_i^c \int_{-1}^{1} T^*(x_d, y_d) R_{i,k}(t) \frac{d\Gamma}{dt} \frac{dt}{d\xi} d\xi. \tag{3.50}$$

A equação (3.39) pode ser escrita, individualmente, como:

$$\sum_{i=1}^{n+1} H_{i,k} T_i^c = \sum_{i=1}^{n+1} G_{i,k} q_i^c. \tag{3.51}$$

Matricialmente, pode-se escrevê-la como:

$$\mathbf{H}\mathbf{T}^c = \mathbf{G}\mathbf{q}^c, \tag{3.52}$$

em que:

$$\mathbf{T} = \mathbf{E}\mathbf{T}^c \longrightarrow \mathbf{T}^c = \mathbf{E}^{-1}\mathbf{T} \tag{3.53}$$

$$\mathbf{q} = \mathbf{E}\mathbf{q}^c \longrightarrow \mathbf{q}^c = \mathbf{E}^{-1}\mathbf{q}. \tag{3.54}$$

4 IMPLEMENTAÇÃO COMPUTACIONAL

4.1 Introdução

De acordo com os conceitos abordados nos Capítulos 2 e 3, como o de curvas e funções de base (B-Splines e NURBS) e do MEC isogeométrico, foi possível desenvolver o programa *ProGeo*, implementado em um primeiro momento por [8] em linguagem MATLAB como $BEM2Diso$ e, posteriormente, em linguagem Julia, em que são calculadas as propriedades geométricas de figuras planas com abordagem isogeométrica. São utilizados elementos de contorno quadráticos contínuos e descontínuos para tal.

4.2 Descrição do programa *PropGeo*

O programa *PropGeo* utiliza a $NURBStoolbox$, fornecido por [10], para gerar os pontos das curvas e calcular as suas respectivas derivadas, sendo possível, desta forma, gerar as geometrias desejadas.

As seguintes etapas correspondem o passa a passo realizado pelo programa:

- Definição dos pontos e segmentos da geometria desejada, conforme arquivo *dad* de entrada, que formam as curvas e os contornos desejado;
- Conversão dos pontos e segmentos para as NURBS, em que são definidos os pontos de controle e pesos utilizando coordenadas homogêneas.
- Cálculo da primeira derivada das NURBS;
- Refinamento p, em que é aumentada a ordem das NURBS.
- Refinamento h, em que é aumentado o número de inserção de nós nas NURBS.
- Criação dos pontos de colocação;
- Criação da matriz **E** responsável pela transferência das condições de contorno dos pontos de controle para os pontos de colocação;
- Plotagem da geometria utilizando NURBS, dos pontos de controle, polígono de controle e pontos de colocação;
- Escolha do número de pontos de Gauss usado na integração (Quadratura Gaussiana);
- Criação das matrizes **H** e **G**;
- Aplicação das condições de contorno;
- Separação da temperatura e do fluxo de calor;
- Cálculo das variáveis desconhecidas;

- Transferência da temperatura e do fluxo de calor dos pontos de controle para os pontos de colocação;
- Criação do mapa de cores.

Para melhor ilustrar o funcionamento dos programas, assim como a sequência hierárquica do processo, faz-se o uso de um fluxograma das funções utilizadas para cada arquivo de entrada *dad* como mostrado pela Figura 5.3 a seguir:

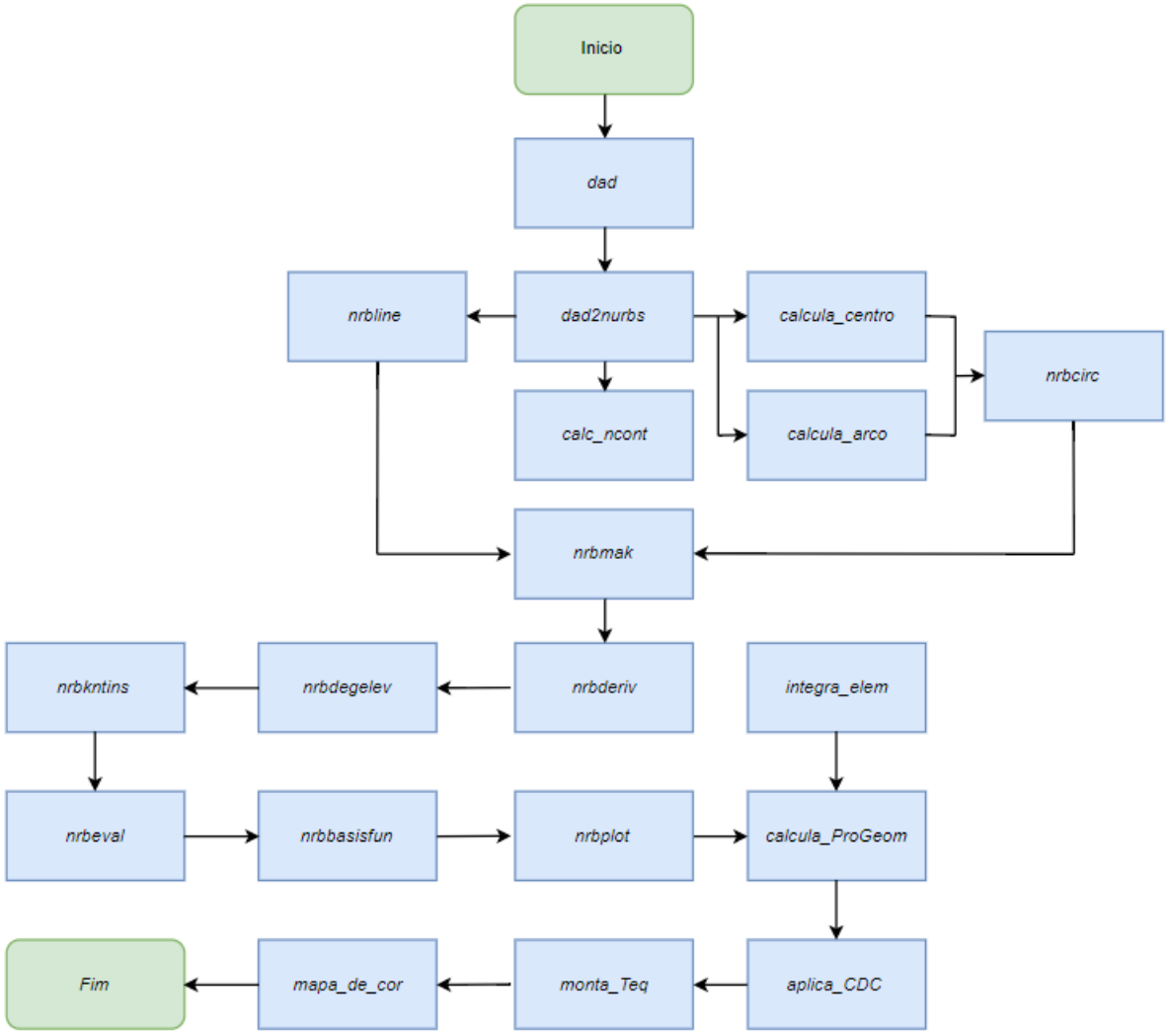

Figura 4.1: Fluxograma hierárquico do funcionamento do programa PropGeo.

Os programas e funções auxiliares ao principal programa *PropGeo* são mostrados a seguir, sendo feitas breves explicações de suas funcionalidades.

dad: arquivo de entrada de dados no qual são definidos os pontos e segmentos da geometria desejada, além dos valores analíticos das propriedades geométricas.

dad2nurbs: é a função em que são convertidos os dados do *dad* para a estrutura NURBS. Em um primeiro momento, calcula-se o número de geometrias fechadas com o auxilio da função

calc_ncont e, posteriormente, importa-se os pontos inicial e final de cada curva, assim como seu raio. Caso o valor do raio não seja nulo, usa-se a função *nrbcirc* importada da biblioteca *nurbs_toolbox* para criar a estrutura da curva. Caso o valor do raio seja nulo, usa-se a função *nrbline* importada novamente da biblioteca *nurbs_toolbox* para criar a estrutura de uma linha. Na função *nrbcirc*, o valor do raio, os pontos inicial e final da curva são usados para definir o centro da mesma, o qual é um dos parâmetros de entrada para a função *nrbcirc*, assim como com os ângulos inicial e final do arco e o raio. O centro é calculado pela função *calcula_centro* e os ângulos do arco são calculados pela função *calcula_arco*.

calc_ncont: função responsável por calcular o número de contornos para que o programa consiga diferenciar cada geometria e não ligue o último ponto de controle do contorno atual com o primeiro ponto de controle do novo contorno.

nrbline: função responsável por definir as características da linha a partir dos pontos inicial e final. Ela define também os pontos de controle e o vetor nós necessários para a função *nrbmak*, presente na biblioteca *nurbs_toolbox*, criar a estrutura da linha.

nrbmak: função responsável por criar a estrutura NURBS da curva a partir dos pontos de controle e do vetor nós calculados através das*nrbcirc* e *nrbline*.

calcula_centro: função responsável pelo cálculo do centro de um arco de círculo, onde entram-se com os pontos inicial e final do arco juntamente com o raio.

calcula_arco: função responsável por calcular o arco da circunferência, sendo assim calculados os ângulos inicial e final do arco. São utilizados os mesmo parâmetros de entrada da função *calcula_centro* descrita acima.

nrbcirc: função responsável por definir as características da curva a partir do raio, centro da curva, ângulos inicial e final do arco calculados pelas funções anteriormente comentadas. A partir dos dados de entrada, a função *nrbcirc* calcula o peso (w), define os pontos de controle e o vetor nós necessários para função *nrbmak* criar a estrutura da curva.

nrbderiv: função originada da biblioteca *nurbs_toolbox*, calcula os valores da primeira derivada de uma curva NURBS. Para isso, ela utiliza os valores gerados pela função *nrbmak*.

nrbdegelev: função originada da biblioteca *nurbs_toolbox*, responsável pelo refinamento p, em que a ordem das curvas e superfícies NURBS é elevada.

nrbkntins: função originada da biblioteca *nurbs_toolbox*, responsável pelo refinamento h, sendo inseridos um único ou múltiplos nós às superfícies ou curvas NURBS.

collocPts: vetor onde são alocados os pontos de colocação criados.

nrbeval: função originada da biblioteca *nurbs_toolbox* que avalia a NURBS nos pontos parametrizados.

nrbbasisfun: função originada da biblioteca *nurbs_toolbox*, responsável pelas funções de base da NURBS. São utilizadas tanto a função *nrbval* como a função $nrbbasisfun$ para a criação da matriz **E**, responsável pela transferência das condições de contorno dos pontos de controle

para os pontos de colocação.

nrbplot: função originada da biblioteca *nurbs_toolbox*, responsável pela representação gráfica da figura, formada pela NURBS, utilizando os valores gerados por *nrbmark*. Desta forma, são plotados os pontos de controle, o polígono de controlo, os pontos de colocação e o segmento.

integra_elem: função que calcula a integral de cada elemento do contorno que, posteriormente, serão somadas. Para esse cálculo, ela recebe como entrada os pontos da curva NURBS, sua derivada, os pesos e os pontos de Gauss, gerados pela função *calcula_PropGeom*, além dos valores consecutivos de u necessários para o cálculo da derivada $\frac{du}{d\xi}$. O valor de saída são os elementos das matrizes **H** e **G** referentes ao elemento que está sendo integrado. Tem-se também que, para esta função, são encontrados os vetores tangente e normal à curva.

calcula_PropGeom: função responsável por receber, como dados de entrada, os pontos das curvas NURBS, suas derivadas, o número de pontos de Gauss usados em cada integração e o número de pontos de controle. E, utilizando a função *integra_elem*, calcula as matrizes **H** e **G**.

aplica_CDC: função que aplica as condições de contorno trocando as colunas das matrizes **H** e **G**.

monta_Teq: função que separa os valores da temperatura e do fluxo de calor do problema.

mostra_heatmap: função responsável pela geração do mapa de cor de acordo com a geométrica da estrutura, valores de temperaturas e fluxos de calor.

Por fim, são feitos os cálculos dos erros percentuais referentes aos resultados analíticos e numéricos de valores de temperatura e fluxo de calor da seguinte maneira:

$$Erro\ Percentual = \left| \frac{Resultado\ Numérico - Resultado\ Analítico}{Resultado\ Analítico} \right| .100\% \tag{4.1}$$

Eles são apresentados juntamente com os valores numéricos e analíticos obtidos para cada propriedade.

4.3 LINGUAGEM JULIA

De acordo com [22], Julia é uma linguagem de programação dinâmica de alto nível e desempenho para computação numérica. Ela fornece um compilador sofisticado, execução paralela distribuída, precisão numérica e uma extensa biblioteca de funções matemáticas. A biblioteca da base Julia, em grande parte escrita na própria linguagem Julia, também integra bibliotecas sofisticadas em linguagens C e Fortran, de código aberto, para aplicações em álgebra linear, geração de números aleatórios, processamento de sinais e processamento de sequências. Além disso, a comunidade de desenvolvedores de Julia está contribuindo com uma série de pacotes externos através do gerenciador de pacotes embutidos.

A seguir são listadas algumas características importantes que concernem a linguagem Julia:

- Excelente desempenho, próximo ao encontrado em linguagens compiladas estaticamente, como em linguagem C.
- Gerenciador de pacotes embutido.
- Possui funções de chamada para Python e para C.

O compilador $just-in-time$, baseado em LLVM ($LowLevelVirtualMachine$), combinado com a estrutura simples da linguagem (sintaxe) permitem aproximar-se, muitas vezes, da performance encontrada na linguagem C. Para melhor ilustrar esses resultados, em [22] faz-se uma comparação entre o desempenho da linguagem Julia e de demais linguagens de programação que podem ser usadas na computação numérica e cientifica. Dessa forma, as linguagens utilizadas no teste de desempenho são: C, Fortran, Julia, Python, MATLAB/Octave, JavaScript, Java, Lua e Mathematica.

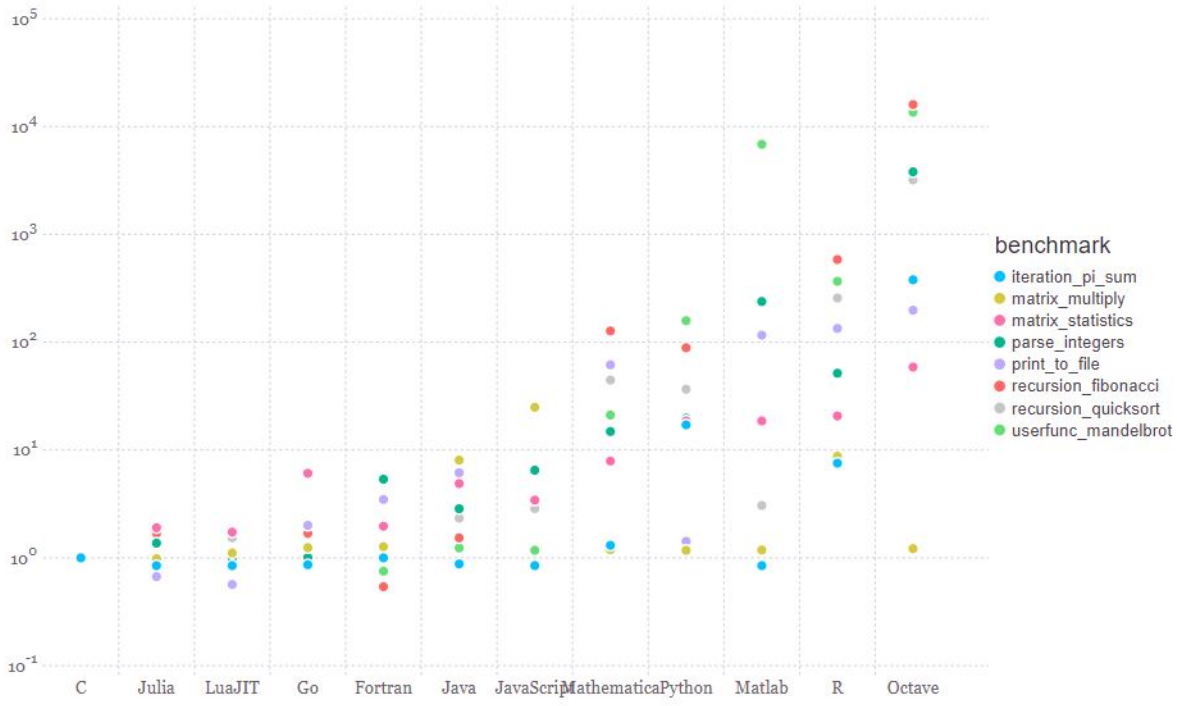

Figura 4.2: *Teste de desempenho relativo a linguagem C (quanto menor, melhor. Desempenho de C = 1)* [22].

Para o teste de desempenho foram realizadas algumas operações, como de soma iterativa de π, operações com matrizes, operações de recursão, dentre outros. Sendo assim, conclui-se que a linguagem Julia possui desempenho muito bom quando comparada às demais linguagens.

A linguagem Julia é uma linguagem de código aberto que oferece grande parte dos recursos do $MATLAB$. Seu núcleo de implementação é licenciado pelo MIT (Massachusetts Institute of

Tecnology), enquanto que algumas de suas bibliotecas possuem licença própria, enquanto outras podem ser compartilhadas. Hà a possibilidade de programar tanto em terminais (utilizando o JuliaPro e o Atom, por exemplo) ou na nuvem (utilizando o JuliaBox). O uso deste último é de extrema praticidade e pode ser operado em qualquer computador conectado à internet ou mesmo em um *smartphone*.

Sendo assim, optou-se por uma reimplementação do programa $BEM2DIso$, apresentado por [6], e rearranjá-lo de acordo com a sintaxe e as funções presentes nas bibliotecas em Julia. Esse novo programa foi chamado de $PropGeo$ e encontra-se no Anexo 1 deste trabalho, assim como suas demais funções.

5 RESULTADOS

Após uma exposição sucinta do papel de cada função do programa *PropGeo* no capítulo anterior, nesta etapa são realizadas, de maneira detalhada, explicação do arquivo *dad*, em que há a presença de outros arquivos de entrada. Os arquivos de dados utilizados neste trabalho são os dad_2, dad_3 e dad_4. Em seguida, são apresentados os resultados obtidos na compilação de todos os programas já descritos e discutidas as soluções presentes e utilizadas.

5.1 Arquivo de entrada dad

5.1.1 Arquivo dad_2

A estrutura do arquivo de entrada de entrada *dad_2* é mostrada nas Figuras 5.1 e 5.2 a seguir:

```
% Entrada de dados para análise de temperatura pelo
% método dos elementos de contorno

% Matriz para definição de pontos que definem a geometria
% PONTO = [número do ponto, coord. x do ponto, coord. y do ponto]

L=2;

PONTOS = [1 0 0   % Ponto 1
          2 2*L 0 % Ponto 2
          3 2*L L % Ponto 3
          4 0 L]; % Ponto 4

% Segmentos que definem a geometria
% SEGMENTOS=[No do segmento, No do ponto inicial, No do ponto final,
%                                                      Raio]
% Raio do segmento: > 0 -> O centro é à esquerda do segmento (do ponto
%                          inicial para o ponto final)
%                   < 0 -> O centro é à direita do segmento (do ponto
%                          inicial para o ponto final)
%                   = 0 -> O segmento é uma linha reta

SEGMENTOS = [1 1 2 0;  % Segmento 1 do primeiro contorno
             2 2 3 0;  % Segmento 1 do segundo contorno
             3 3 4 0;  % Segmento 1 do terceiro contorno
             4 4 1 0]; % Segmento 1 do quarto contorno
```

Figura 5.1: *Estrutura do arquivo dad_2 referente às definições de pontos e segmentos.*

```
% Condições de contorno nos segmentos
% CCSeg=[no do segmento, tipo da CDC, valor da CDC]
% tipo da CDC = 0 => a temperatura é conhecida
% tipo da CDC = 1 => O fluxo é conhecido

CCSeg=[1 1 0   % CC no segmento 1 do primeiro contorno
       2 0 100 % CC no segmento 1 do segundo contorno
       3 1 0   % CC no segmento 1 do terceiro contorno
       4 0 0]; % CC no segmento 1 do quarto contorno

kmat=1; % Condutividade térmica do material
```

Figura 5.2: *Estrutura do arquivo dad_2 referente às condições de contorno do problema.*

Primeiramente, a matriz **PONTOS** é criada, sendo que a primeira coluna refere-se ao número do ponto, a segunda coluno refere-se à coordenada x do ponto enquanto que a terceira e última coluna refere-se à coordenada y do ponto.

Em um segundo momento, a matriz **SEGMENTOS** é criada, a primeira coluna referente ao número do segmento, a segunda coluna referente ao número do ponto inicial, a terceira coluna referente ao número do ponto final e a última caluna referente ao raio do cilindro.

Por fim, é criada a matriz **CCSeg**, para a definição da malha, em que são encontradas as condições de contorno para cada segmento. A primeira coluna refere-se ao número do segmento, enquanto que a segunda coluna refere-se ao tipo de condição de contorno conhecida e a terceira coluna refere-se ao valor da temperatura ou fluxo e calor.

5.1.2 Arquivo dad_3

A estrutura do arquivo de entrada de entrada *dad_3* é mostrada nas Figuras 5.3 e 5.4 a seguir:

```
% Entrada de dados para análise de temperatura pelo
% método dos elementos de contorno do programa BEM2Diso

% Matriz para definição de pontos que definem a geometria
% PONTO = [número do ponto, coord. x do ponto, coord. y do ponto]

a=1;
b=2;

PONTOS = [1 -b 0  % Ponto 1
          2 b 0   % Ponto 2
          3 -a 0  % Ponto 3
          4 a 0]; % Ponto 4

% Segmentos que definem a geometria
% SEGMENTOS=[No do segmento, No do ponto inicial, No do ponto final,
%                                                      Raio]
% Raio do segmento: > 0 -> O centro é à esquerda do segmento (do ponto
%                          inicial para o ponto final)
%                   < 0 -> O centro é à direita do segmento (do ponto
%                          inicial para o ponto final)
%                   = 0 -> O segmento é uma linha reta

SEGMENTOS = [1 1 2 b    % Segmento 1 do primeiro contorno
             2 2 1 b    % Segmento 2 do primeiro contorno
             3 3 4 -a   % Segmento 1 do segundo contorno
             4 4 3 -a]; % Segmento 2 do segundo contorno
```

Figura 5.3: *Estrutura do arquivo dad_3 referente às definições de pontos e segmentos para o dad_3.*

```
% Condições de contorno nos segmentos
% CCSeg=[no do segmento, tipo da CDC, valor da CDC]
% tipo da CDC = 0 => a temperatura é conhecida
% tipo da CDC = 1 => O fluxo é conhecido

CCSeg = [1 1 -200  % CC no segmento 1 do primeiro contorno
         2 1 -200  % CC no segmento 2 do primeiro contorno
         3 0 100   % CC no segmento 1 do segundo contorno
         4 0 100]; % CC no segmento 2 do segundo contorno

kmat=1; % condutividade térmica do material [W/m.K]

% Geração automática de pontos internos
npi=17;
NPX=npi; % Número de pontos internos na direção X
NPY=npi; % Número de pontos internos na direção Y

a=1; % Raio interno do primeiro cilindro [m]
b=2; % Raio externo do segundo cilindro [m]
Ti=100; % Temperatura no contorno [K]
qo=-200; % Fluxo de calor no contorno externo [W/m^2]
qi=-qo*b/a; % Fluxo de calor analitico no contorno interno [W/m^2]
To=Ti-qo*b*log(b/a); % Temperatura analitica no contorno externo [K]
```

Figura 5.4: *Estrutura do arquivo dad_3 referente às condições de contorno para o dad_3.*

Primeiramente, a matriz **PONTOS** é criada, sendo que a primeira coluna refere-se ao número do ponto, a segunda coluno refere-se à coordenada x do ponto enquanto que a terceira e última coluna refere-se à coordenada y do ponto.

Em um segundo momento, a matriz **SEGMENTOS** é criada, a primeira coluna referente ao número do segmento, a segunda coluna referente ao número do ponto inicial, a terceira coluna referente ao número do ponto final e a última caluna referente ao raio do cilindro.

Por fim, é criada a matriz **CCSeg**, para a definição da malha, em que são encontradas as condições de contorno para cada segmento. A primeira coluna refere-se ao número do segmento, enquanto que a segunda coluna refere-se ao tipo de condição de contorno conhecida e a terceira coluna refere-se ao valor da temperatura ou fluxo e calor.

5.1.3 Arquivo dad_4

A estrutura do arquivo de entrada de entrada *dad_4* é mostrada nas Figuras 5.5 e 5.6 a seguir:

```
% Entrada de dados para análise de temperatura pelo
% método dos elementos de contorno do programa BEM2Diso

% Matriz para definição de pontos que definem a geometria
% PONTO = [número do ponto, coord. x do ponto, coord. y do ponto]

PONTOS = [1 0 0      % Ponto 1
          2 2 0      % Ponto 2
          3 2 1      % Ponto 3
          4 1 2      % Ponto 4
          5 0 2      % Ponto 5
          6 .25 1    % Ponto 6
          7 1.25 1]; % Ponto 7

% Segmentos que definem a geometria
% SEGMENTOS=[No do segmento, No do ponto inicial, No do ponto final,
%                                                      Raio]
% Raio do segmento: > 0 -> O centro é à esquerda do segmento (do ponto
%                          inicial para o ponto final)
%                   < 0 -> O centro é à direita do segmento (do ponto
%                          inicial para o ponto final)
%                   = 0 -> O segmento é uma linha reta

SEGMENTOS = [1 1 2 0     % Segmento 1 do primeiro contorno
             2 2 3 0     % Segmento 2 do primeiro contorno
             3 3 4 1     % Segmento 3 do primeiro contorno
             4 4 5 0     % Segmento 4 do primeiro contorno
             5 5 1 0     % Segmento 5 do primeiro contorno
             6 6 7 -.5   % Segmento 1 do segundo contorno
             7 7 6 -.5]; % Segmento 2 do segundo contorno
```

Figura 5.5: *Estrutura do arquivo dad_4 referente às definições de pontos e segmentos.*

```
% Condições de contorno nos segmentos
% CCSeg=[no do segmento, tipo da CDC, valor da CDC]
% tipo da CDC = 0 => a temperatura é conhecida
% tipo da CDC = 1 => O fluxo é conhecido

CCSeg = [1 0 0   % CC no segmento 1 do primeiro contorno
         2 1 30   % CC no segmento 2 do primeiro contorno
         3 1 30   % CC no segmento 3 do primeiro contorno
         4 1 30   % CC no segmento 4 do primeiro contorno
         5 0 100   % CC no segmento 5 do primeiro contorno
         6 1 0   % CC no segmento 1 do segundo contorno
         7 1 0]; % CC no segmento 2 do segundo contorno

kmat=1; % condutividade térmica do material [W/m.K]
```

Figura 5.6: *Estrutura do arquivo dad_4 referente às condições de contorno para o dad_4.*

Primeiramente, a matriz **PONTOS** é criada, sendo que a primeira coluna refere-se ao número do ponto, a segunda coluno refere-se à coordenada x do ponto enquanto que a terceira e última coluna refere-se à coordenada y do ponto.

Em um segundo momento, a matriz **SEGMENTOS** é criada, a primeira coluna referente ao número do segmento, a segunda coluna referente ao número do ponto inicial, a terceira coluna referente ao número do ponto final e a última caluna referente ao raio do cilindro.

Por fim, é criada a matriz **CCSeg**, para a definição da malha, em que são encontradas as condições de contorno para cada segmento. A primeira coluna refere-se ao número do segmento, enquanto que a segunda coluna refere-se ao tipo de condição de contorno conhecida e a terceira coluna refere-se ao valor da temperatura ou fluxo e calor.

5.1.4 Resultados

5.1.5 Arquivo dad_2

Este exemplo é referente a um problema de condução de calor em um retângulo, sendo modelado como um problema 2D. Sabe-se que a geometria possui 4 segmentos. As temperaturas nos segmentos 2 e 4 são de $100°C$ e $0°C$, respectivamente. Já os fluxos de calor nos segmentos 1 e 3 são de $0W/m^2$.

Na Figura 5.7 a seguir é possível verificar os pontos fonte, curva resultante, polígono de controle e pontos de controle, conforme a legenda. Vale ressaltar que para estes resultados, foram considerados 22 pontos de Gauss na integração (na quadratura Gaussiana), além dos refinamentos $p = 1$ e $h = 10$.

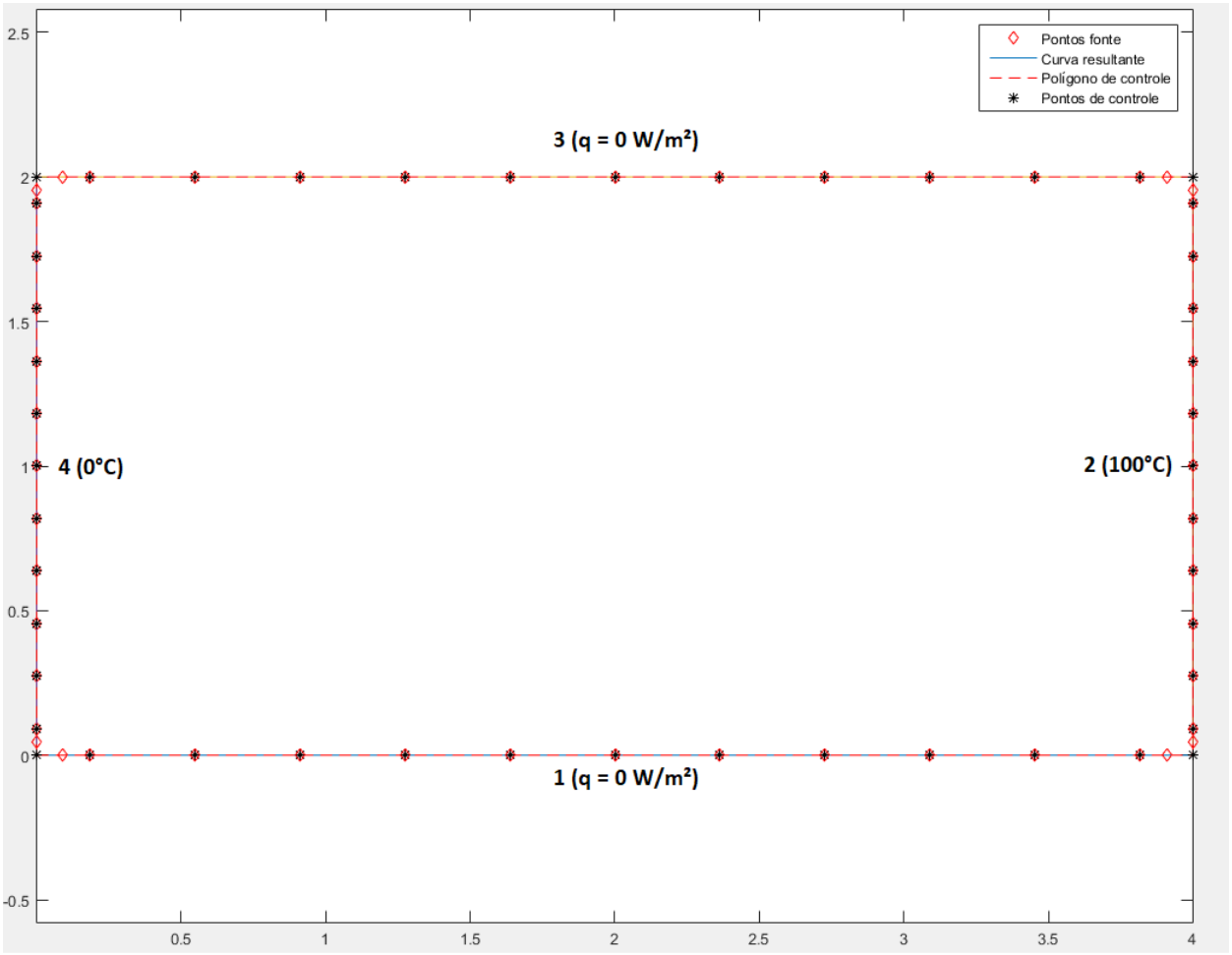

Figura 5.7: *Resultado obtido para condução de calor em um retângulo.*

Para melhor ilustrar o problema da condução de calor em um retângulo, é plotado uma representação em mapa de cor na Figura 5.8 a seguir, em que é possível perceber o comportamento da temperatura e do fluxo de calor.

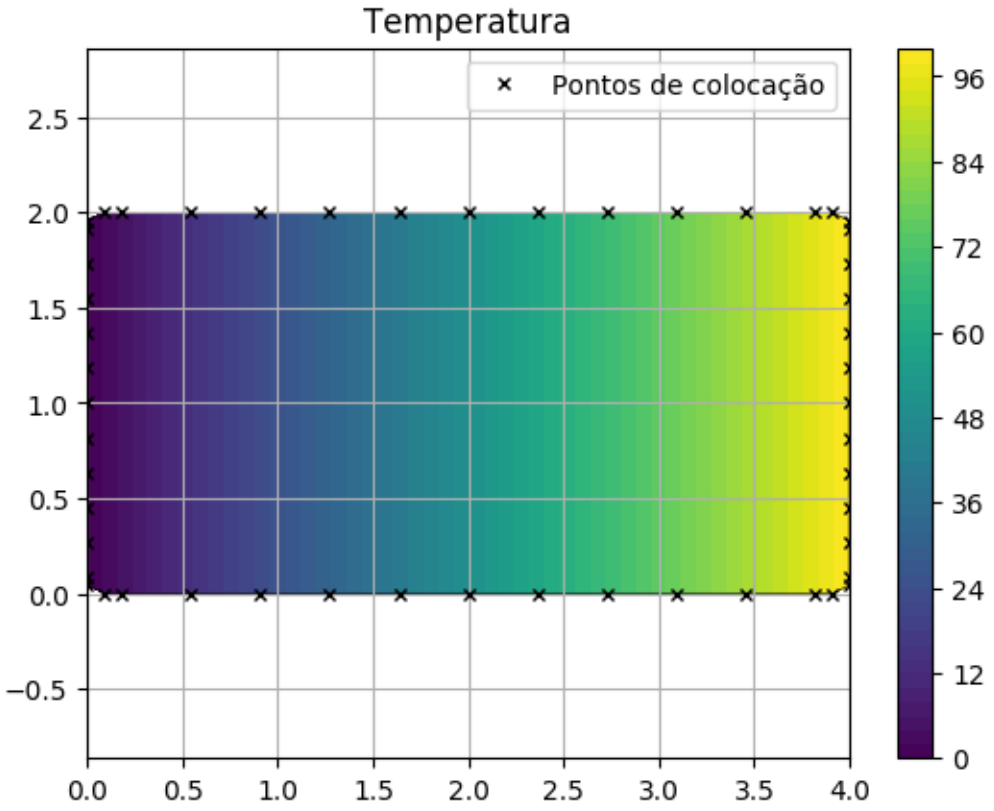

Figura 5.8: *Mapa de cor obtido para condução de calor em um retângulo.*

Percebe-se que há uma gradação de cor mais quente para cor mais fria a medida que desloca-se para a esquerda.

5.1.6 Arquivo dad_3

Este exemplo é referente a um problema de condução de calor em um cilindro sendo modelado como um problema 2D. Sabe-se tanto a temperatura no contorno interno, que é $100°C$, como o fluxo de calor no contorno externo, que é de $-200\frac{W}{m^2}$. A condutividade térmica do material é de $1\frac{W}{m°C}$. Os raios interno e externo do cilindro possuem 1 m e 2 m, respectivamente. A Figura 5.9 a seguir ilustra melhor o problema, sendo S_e o contorno externo e S_i o contorno interno. A solução analítica para a temperatura deste problema é dada por:

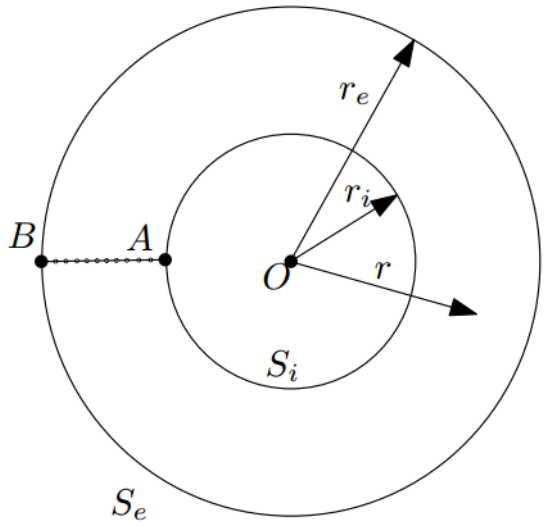

Figura 5.9: *Problema potencial em uma região anelar*[8].

$$T(r) = T_i + q_e r_e \log\left(\frac{r}{r_i}\right), \tag{5.1}$$

sendo T_i é a temperatura no contorno interno, q_e é o fluxo de calor no contorno externo, r_e é o raio externo do cilindro e r_i é o raio interno do cilindro. A solução analítica para o fluxo de calor é dada por:

$$q(r) = q_e \frac{r_e}{r}. \tag{5.2}$$

Na Figura 5.10 a seguir é possível verificar os pontos fonte, curva resultante, polígono de controle e pontos de controle, conforme a legenda. Vale ressaltar que para estes resultados, foram considerados 22 pontos de Gauss na integração (na quadratura Gaussiana), além dos refinamentos $p = 1$ e $h = 10$.

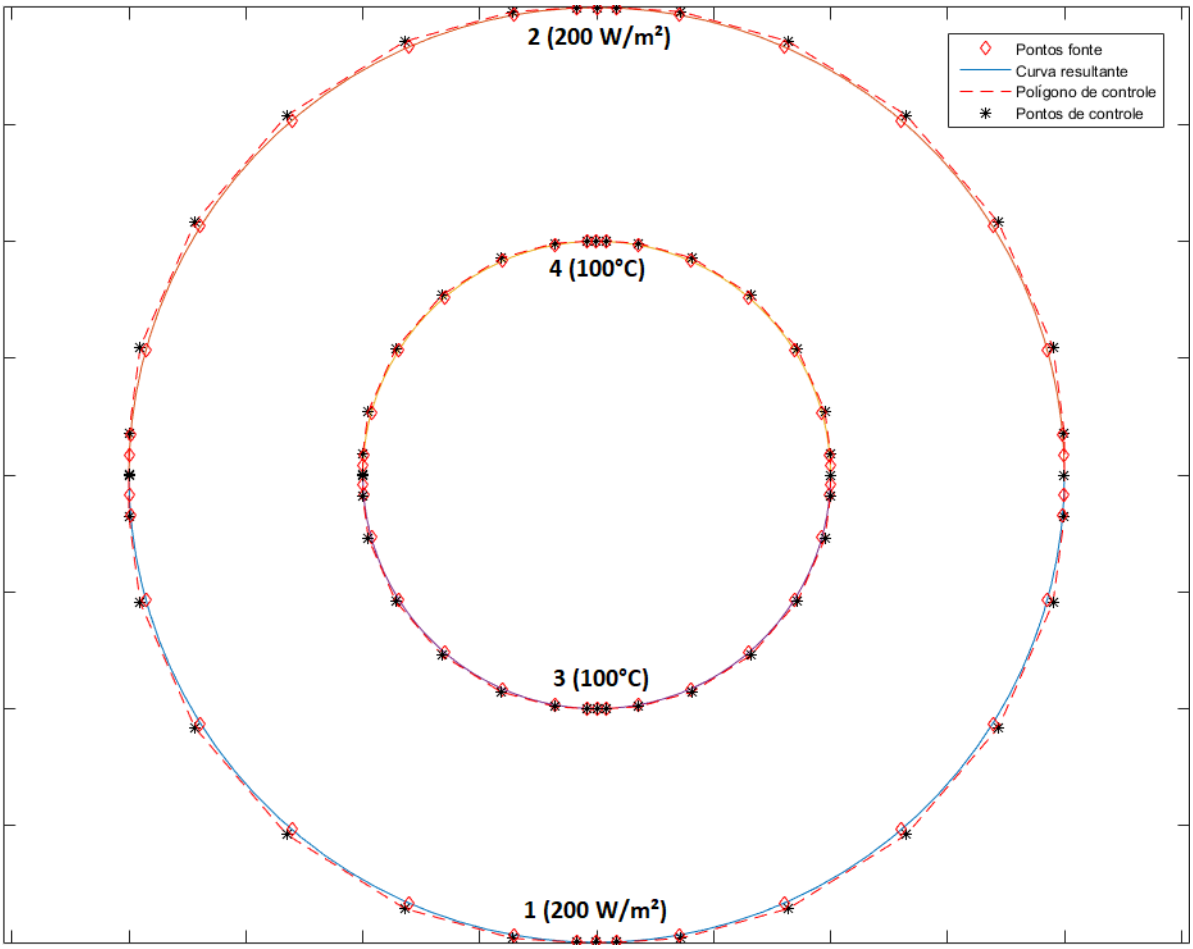

Figura 5.10: *Resultado obtido para condução de calor em um cilindro.*

Para melhor ilustrar o problema da condução de calor em um cilindro, é plotado uma representação em mapa de cor na Figura 5.11 a seguir, em que é possível perceber o comportamento físico tanto da temperatura quanto do fluxo de calor com a variação do tamanho do raio. Desta forma, as temperaturas e fluxos de calor são passadas dos pontos de controle para os pontos de colocação, com o auxilio da matriz **E**.

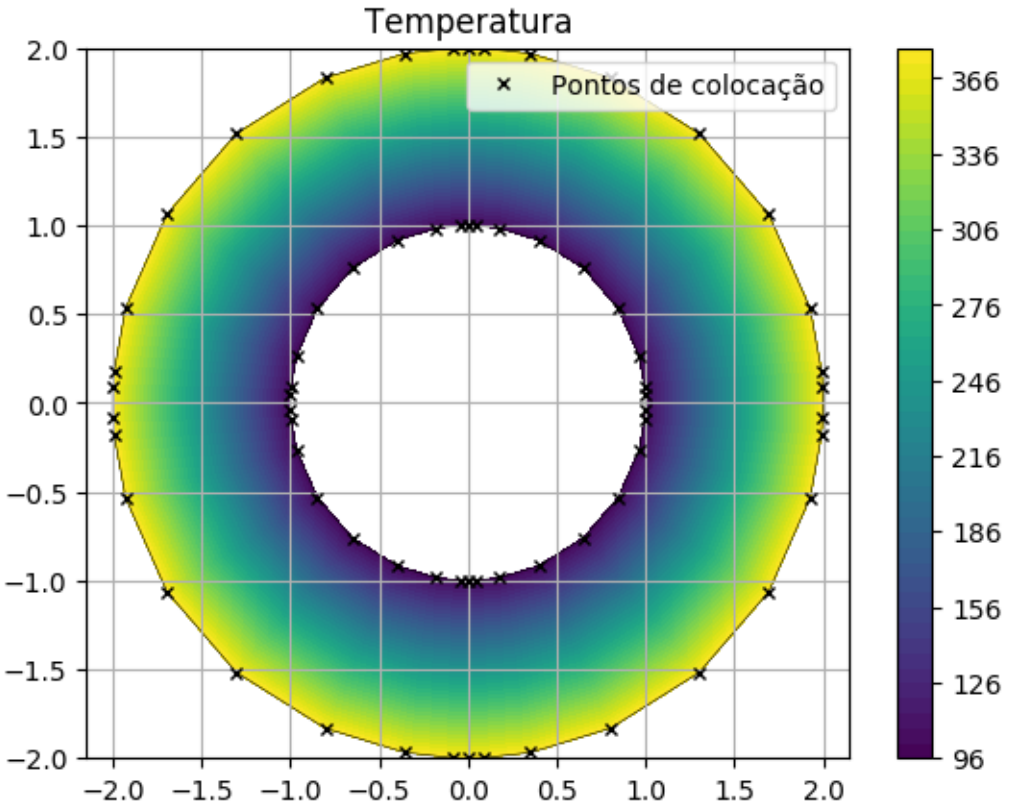

Figura 5.11: *Mapa de cor obtido para condução de calor em um cilindro.*

Percebe-se que ha uma gradação de cor mais fria para cor mais quente a medida que a temperatura e o raio do cilindro aumentam. Para que seja feita uma comparação das soluções analíticas e numéricas, utiliza-se a seguinte relação:

$$Erro\ Percentual = \left| \frac{Resultado\ Numérico - Resultado\ Analítico}{Resultado\ Analítico} \right| .100\%$$

Tabela 5.1: Resultados das soluções analíticas e numéricas para a temperatura.

Resultados	**Valor**
Solução numérica	377.2700 $°C$
Solução analítica	377.2589 $°C$
Erro percentual de temperatura	0.0029
Erro absoluto de temperatura	2.9437e-05

Tabela 5.2: Resultados das soluções analíticas e numéricas para o fluxo de calor.

Resultados	**Valor**
Solução numérica	-399.9251 W/m^2
Solução analítica	-400 W/m^2
Erro percentual de temperatura	0.0598
Erro absoluto de temperatura	5.9844e-04

Desta forma, visto que tanto os erros percentuais como os erros absolutos para os resultados de temperatura e fluxo de calor foram muito baixos, conclui-se que os valores foram satisfatórios, o que valida o programa *PropGeo* para o uso em análise isogeométrica bidimensional do método dos elementos de contorno.

5.1.7 Arquivo dad_4

Este exemplo é referente a um problema de condução de calor em uma geometria mais complexa que a da subseção anterior, sendo modelado como um problema 2D. Sabe-se que a geometria possui 7 segmentos, conforme as Figuras 5.5 e 5.6. As temperaturas nos segmentos 1 e 5 são de $0°C$ e $100°C$, respectivamente. Já os segmentos 2, 3 e 4 têm fluxo de calor de $30W/m^2$, enquanto que os segmentos 6 e 7 têm fluxo de calor de $0W/m^2$. O raio interno do segmento 3 é de $1m$.

Na Figura 5.12 a seguir é possível verificar os pontos fonte, curva resultante, polígono de controle e pontos de controle, conforme a legenda. Vale ressaltar que para estes resultados, foram considerados 22 pontos de Gauss na integração (na quadratura Gaussiana), além dos refinamentos $p = 1$ e $h = 10$.

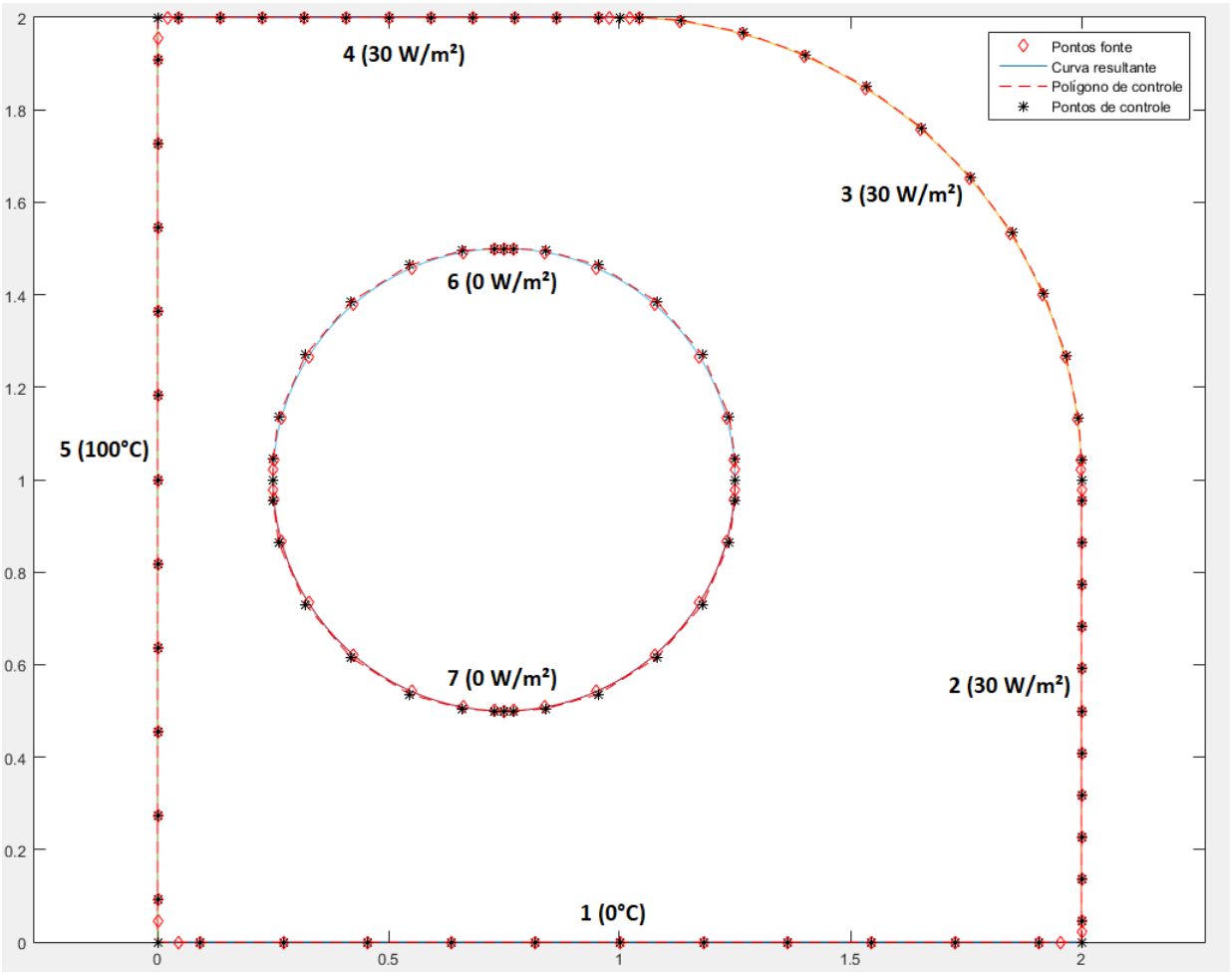

Figura 5.12: *Resultado obtido para condução de calor em tal geometria.*

Para melhor ilustrar o problema da condução de calor na geometria mais complexa, é plotada uma representação em mapa de cor na Figura 5.13 a seguir, em que é possível perceber o comportamento físico tanto da temperatura quanto do fluxo de calor.

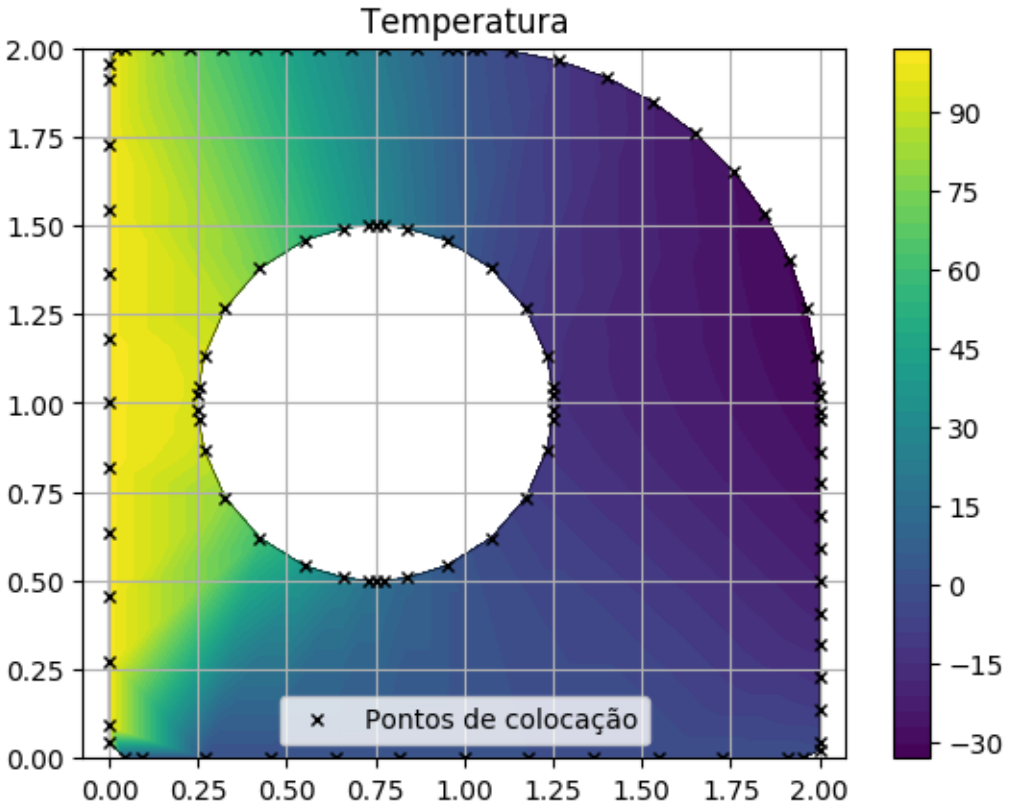

Figura 5.13: *Mapa de cor obtido para condução de calor em tal geometria.*

6 CONCLUSÃO E TRABALHOS FUTUROS

6.1 Conclusão

Neste trabalho foi apresentada uma formulação isogeométrica dos métodos dos elementos de contorno para problemas bidimensionais. Sabendo que as NURBS são as funções de base mais utilizadas em programas CAD, a sua utilização na análise isogeométrica é imprescindível. Sendo assim, em um primeiro momento foi iniciada a implementação de uma abordagem simplificada do método dos elementos de contorno isogeométricos para problemas bidimensionais, com o auxílio do programa $BEM2Diso$ em linguagem $MATLAB$ [8]. Na segunda etapa do Projeto de Graduação, reimplementou-se o programa anterior para linguagem Julia, sendo esse o *PropGeo*.

Sabendo que os pontos de colocação não estão necessariamente contidos nos polígonos de controle, percebeu-se a necessidade de tratar de maneira diferenciada as condições de contorno no método isogeométrico, sendo utilizada a matriz de transformação **E**. Com o auxílio da Transformada de Telles foram calculadas integrais que possuam certo grau de singularidade. Resultados satisfatórios foram obtidos com relação aos valores de temperatura e fluxo de calor em problemas potenciais bidimensionais, em que valores de erros entre soluções numéricas e analíticas foram comparadas. Diminuições dos erros podem ocorrem com aumento do número de ponto de Gauss na integração, assim como pelos refinamentos h e p.

6.2 Trabalhos futuros

Como sugestão para trabalhos futuros que abranjam o método dos elementos de contorno isogeométrico, tem-se:

- Implementação da transformação de NURBS em curvas de Bézier (extração de Bezier). As NURBS têm um alto custo computacional, devido a sua forma de cálculo recursiva. Alguns trabalhos da literatura, como em vistos em [1], transformam as NURBS em curvas de Bézier para diminuir o custo computacional.
- Estender a formulação para outros problemas, como problemas elásticos e acústicos.
- Estender a formulação para problemas tridimensionais.

REFERÊNCIAS BIBLIOGRÁFICAS

Referências

[1] G. Beer. *Advanced numerical simulation methods: from CAD data directly to simulation results*. CRC Press/Balkema, 2015.

[2]

[3]

[4]

[5] A. F. M. Azevedo. *Método dos Elementos Finitos*. Faculdade de Engenharia da Universidade do Porto, Porto, 2003.

[6] C. A. Felippa. *Finite Element Discretization and the Direct Stiffness Method*, 2011.

[7] E. L. de Albuquerque. *Introdução ao Método dos Elementos de Contorno*. Universidade de Brasília, Brasília, 2014.

[8] L. S. Campos. *Método dos elementos de contorno isogeométricos rápido*. PhD thesis, Universidade de Brasília, Brasília, 2016.

[9] F. M. Loyola. *Modelagem tridimensional de problemas potenciais usando o método dos elementos de Contorno*, MSc thesis, Universidade de Brasília, Brasília, 2017.

[10] T.J.R. Hughes, J.A. Cottrell, Y. Bazilevs *Isogeometric Analysis: CAD, finite elements, NURBS, axact geometry, and mesh refinement.* Computer Methods in Applied Mechanics and Engineering. Willey, 2009.

[11] C. A. S. P. Sant'Ana. *Criação de modelos planos e sólidos usando NURBS*, Projeto de Graduação, Universidade de Brasília, Brasília, 2017.

[12] B. B. Correa. *Cálculo de Propriedades de Sólidos Modelados em IGES*, Projeto de Graduação, Universidade de Brasília, Brasília, 2017.

[13] J. E. S. Gonçalves. *Leitura e interpretaçao de arquivos IGES para uso em programas de Elementos de Contorno*, Projeto de Graduação, Universidade de Brasília, Brasília, 2016.

[16] M. A. Correia. *Cálculo de Propriedades Geométricas Usando Modelagem Isogeométrica*, Projeto de Graduação, Universidade de Brasília, Brasília, 2016.

[15] D. F. Rogers. *An Introduction to NURBS*. Academic Press, 2001.

[16] http://www.cs.cmu.edu/ tcortina/15110f11/lab9/index.html, acessado dia 05/05/2017 às 18h50, Brasília, 2017.

[17] http://www.ecartouche.ch/contentreg/cartouche/graphics/en/html/CurveslearningObject3.html, acessado dia 05/05/2017 às 18h50, Brasília, 2017.

[18] L. Piegl and W. Tiller. *The NURBS Book*, Springer, second edition, 1997.

[19] M. Spink. Nurbs toolbox. Disponível em:<http://www.aria.uklinux.net/nurbs.php3>. Acesso em: 05/06/2017.

[20] J.C.F. Telles *A self adaptative co-ordinate transformation for efficient numerical evaluation of general boundary element integrals.* International Journal for Numerical Methods in Engineering, 1987.

[19] J.J.S.P. Cabral, L.C. Wrobel, C.A. Brebbia *A BEM formulation using B-splines: I-uniform blending functions* Computational Mechanics Institute, Wessex Institute os Technology, Ashurst Lodge, Ashurst, Southampton, England, 1990.

[22] https://julialang.org/, acessado dia 05/10/2017 às 14h45, Brasília, 2017.

ANEXOS

Programa PropGeo

```
include("dad.jl")
include("nurbs.jl")
include("calcula_PropGeom.jl")
include("formatiso.jl")
include("telles.jl")
include("inpoly.jl")
include("mostra_problema.jl")
using FastGaussQuadrature
using PyCall
using PyPlot
plt=PyPlot
@pyimport matplotlib.tri as tri

# plotlyjs()

PyPlot.close("all")   # close all plot windows

# O numero de nos (knots), m, o numero de pontos de controle, k, e a ordem da
# curva, n, estao relacionados por:
#                           m = k + n + 1

PONTOS,SEGMENTOS,CCSeg,kmat=dad_3() #Arquivo de entrada de dados
# PONTOS,SEGMENTOS,MALHA,CCSeg,kmat=dad_2() #Arquivo de entrada de dados

# NOS,ELEM=format_dad(PONTOS,SEGMENTOS,MALHA)# formata os dados (cria as
crv,contorno=format_dad_iso(PONTOS,SEGMENTOS)# formata os dados

# display(mostra_geo(crv))

dcrv=map(x->nrbderiv(x),crv)
n = length(crv);    # Numero total de elementos

p=1;#refinamento p

for i=1:n
degree=crv[i].order-1
coefs,knots = bspdegelev(degree,crv[i].coefs,crv[i].knots,p)
crv[i] = nrbmak(coefs,knots)
end

h=10;#refinamento h
for i=1:n
novosnos=linspace(0,1,h+2)
degree=crv[i].order-1
coefs,knots = bspkntins(degree,crv[i].coefs,crv[i].knots,novosnos[2:end-1])
```

```
crv[i] = nrbmak(coefs,knots)
end

z=0;
for k=1:n
for i=1:crv[k].number
z=z+1
end
end
numcurva=zeros(Integer,z)
collocPts=zeros(z)
CDC=zeros(z,3)
collocCoord=zeros(z,3)
z=0;
nnos=zeros(Integer,n)
for k=1:n
p=crv[k].order-1;
nnos[k]=crv[k].number;
valorCDC=CCSeg[k,3];
tipoCDC=CCSeg[k,2];
for i=1:crv[k].number
z=z+1;
numcurva[z]=k;
collocPts[z]=sum(crv[k].knots[(i+1):(i+p)])/p;
if(i==2)
collocPts[z-1]=(collocPts[z]+collocPts[z-1])/2;
end
if(i==nnos[k])
collocPts[z]=(collocPts[z]+collocPts[z-1])/2;
end

CDC[z,:] = [z,tipoCDC,valorCDC];
end
end
nnos2=cumsum([0 nnos'],2);

E=zeros(length(collocPts),length(collocPts));
for i=1:length(collocPts)
collocCoord[i,:]=nrbeval(crv[numcurva[i]], collocPts[i]);
B, id = nrbbasisfun(crv[numcurva[i]],collocPts[i])
E[i,id+nnos2[numcurva[i]]]=B
end
# legend('Curva resultante','Poligono de controle','Pontos de controle','Pontos fonte')
H,G=calcula_PropGeom(collocCoord,nnos2,crv,dcrv,kmat)

H=H-E/2

A,b= aplica_CDC(G,H,CDC,E)
x=A\b # Calcula o vetor x

Tc,qc=monta_Teq(CDC,x) # Separa temperatura e fluxo
```

```
97
98  T=E*Tc
99  q=E*qc
100
101
102 Ti=100
103 qo=-200
104 ri=1
105 ro=2
106 Toa=Ti-qo*ro*log(ro/ri) # Temperatura analitica no circulo externo
107
108 ncont=size(contorno,1)
109 ncollocpoints=size(collocPts,1)
110 ELEM=zeros(Integer,ncollocpoints,2)
111 noini=1
112 for ii=1:ncont
113 icont=contorno[ii,1]
114 ncont=contorno[ii,2]
115 nofim=nnos2[icont+ncont]
116 indnos=collect(noini:nofim)
117 ELEM[noini:nofim,1]=indnos
118 ELEM[noini:nofim,2]=indnos+1
119 ELEM[nofim,2]=noini
120 noini=nofim+1
121 end
122
123 # mostra_problema(ELEM,NOS_GEO,NOS,tipoCDC,valorCDC,normal)
124 mostra_heatmap(collocCoord[:,1:2],T,ELEM)
```

Arquivo de entrada dad

```
# Entrada de dados para analise de temperatura pelo
# metodo dos elementos de contorno

# Matriz para definicao de pontos que definem a geometria
# PONTOS = [numero do ponto, coord. x do ponto, coord. y do ponto]
function dad_2()
# Entrada de dados

# Matriz para definicao de pontos
# PONTO = [numero do ponto, coordenada x do ponto, coordenada y do ponto];
L=2;
PONTOS  = [1   0   0
2   2*L   0
3   2*L  L
4   0  L];

# Segmentos que definem a geometria
#  SEGMENTOS=[No do segmento, No do ponto inicial, No do ponto final,
#                                                  Radio]
# Raio do segmento: > 0 -> O centro eh a esquerda do segmento (do ponto
#                          inicial para o ponto final)
#                   < 0 -> O centro eh a direita do segmento (do ponto
#                          inicial para o ponto final)
#                   = 0 -> O segmento eh uma linha reta

SEGMENTOS = [1 1 2 0;
2 2 3 0;
3 3 4 0;
4 4 1 0];

# Condicoes de contorno nos segmentos
# CCSeg=[Segmento,tipo da CDC em x, valor da CDC em x , ...
#                            tipo da CDC em y, valor da CDC em y]
# tipo da CDC = 0 => o deslocamento eh conhecido
# tipo da CDC = 1 => a forca de superficie eh conhecida
# Para condicoes de contorno de forca normal conhecida proceder:
# tipo da CDC em x = 2, valor da CDC em x = valor da forca normal
# Neste caso pode-se atribuir quaisquer valores para tipo da CDC em y e
# para valor da CDC em y

CCSeg=[1 1 0
2 0 1
3 1 0
4 0 0];

k=1;
return PONTOS,SEGMENTOS,CCSeg,k

```

```
end
function dad_1(ne=10,et=1)
PONTOS  = [1   0  0
2    1  0]

# Segmentos que definem a geometria
#  SEGMENTOS=[No do segmento, No do ponto inicial, No do ponto final
#                                                  Raio, tipo do elemento]
# Raio do segmento: > 0 -> O centro eh a esquerda do segmento (do ponto
#                          inicial para o ponto final)
#                   < 0 -> O centro eh a direita do segmento (do ponto
#                          inicial para o ponto final)
#                   = 0 -> O segmento eh uma linha reta
# Tipo do elemento = 1 -> Elemento quadratico continuo
#                  = 2 -> Elemento quadratico descontinuo
#                  = 3 -> Elemento linear continuo

SEGMENTOS = [1 1 2  .5 et
2 2 1  .5 et]

CCSeg=[1 1 0
2 0 1];

k=1;

return PONTOS,SEGMENTOS,CCSeg,k
end
function dad_0(ne=10)
PONTOS  = [1 0 0
2 1 0
3 1 1
4 0 1 ]

# Segmentos que definem a geometria
#  SEGMENTOS=[No do segmento, No do ponto inicial, No do ponto final
#                                                  Raio, tipo do elemento]
# Raio do segmento: > 0 -> O centro eh a esquerda do segmento (do ponto
#                          inicial para o ponto final)
#                   < 0 -> O centro eh a direita do segmento (do ponto
#                          inicial para o ponto final)
#                   = 0 -> O segmento eh uma linha reta
# Tipo do elemento = 1 -> Elemento quadratico continuo
#                  = 2 -> Elemento quadratico descontinuo

SEGMENTOS = [1 1 2 0
2 2 3 0
3 3 4 0
4 4 1 0]
CCSeg=[1 1 0
2 0 1
3 1 0
```

```
4 0 0];

k=1;

return PONTOS,SEGMENTOS,CCSeg,k
end

function dad_3()

a=1;
b=2;

PONTOS  = [1 -b 0 ;
2 b 0 ;
3 -a 0 ;
4 a 0];

# Segmentos que definem a geometria
#  SEGMENTOS=[No do segmento, No do ponto inicial, No do ponto final,
#                                                  Raio]
# Raio do segmento: > 0 -> O centro eh a esquerda do segmento (do ponto
#                          inicial para o ponto final)
#                   < 0 -> O centro eh a direita do segmento (do ponto
#                          inicial para o ponto final)
#                   = 0 -> O segmento eh uma linha reta
SEGMENTOS = [1 1 2 b;
2 2 1 b;
3 3 4 -a
4 4 3 -a];

CCSeg=[1 1 -200
2 1 -200
3 0 100
4 0 100];

kmat=1;

# Geracao automatica de pontos internos
npi=17;
NPX=npi; # Numero de pontos internos na direcao X
NPY=npi; # Numero de pontos internos na direcao Y

# disp('Erros em relacao a solucao analitica')
# nnos=size(NOS,1);
# npint=size(PONTOS_int,1);
# r=sqrt(PONTOS_int(:,2).^2+PONTOS_int(:,3).^2);
a=1;
b=2;
Ti=100;
qo=-200;
qi=-qo*b/a;
```

```
151 To=Ti-qo*b*log(b/a);
152
153 return PONTOS,SEGMENTOS,CCSeg,kmat
154 end
155
156 function dad_4()
157
158 PONTOS  = [1   0   0
159 2   2   0
160 3   2   1
161 4   1   2
162 5   0   2
163 6   .25  1
164 7   1.25 1];
165
166 # Segmentos que definem a geometria
167 #  SEGMENTOS=[No do segmento, No do ponto inicial, No do ponto final,
168 #                                                  Raio]
169 # Raio do segmento: > 0 -> O centro eh a esquerda do segmento (do ponto
170 #                          inicial para o ponto final)
171 #                   < 0 -> O centro eh a direita do segmento (do ponto
172 #                          inicial para o ponto final)
173 #                   = 0 -> O segmento eh uma linha reta
174 SEGMENTOS = [1 1 2  0
175 2 2 3  0
176 3 3 4  1
177 4 4 5  0
178 5 5 1  0
179 6 6 7  -.5
180 7 7 6  -.5];
181
182 CCSeg=[1 0 0
183 2 1 1
184 3 1 1
185 4 1 1
186 5 0 0
187 6 1 0
188 7 1 0 ];
189
190 kmat=1;
191
192 return PONTOS,SEGMENTOS,CCSeg,kmat
193 end
```

Função calcula_PropGeom

```
function calcula_PropGeom(collocCoord,nnos,crv,dcrv,kmat)
dcrvs=map(x->nrbderiv(x),crv)
n = length(crv);    # Number of curves
ncollocpoints=size(collocCoord,1)

H=zeros(ncollocpoints,ncollocpoints);
G=zeros(ncollocpoints,ncollocpoints);
npgauss=12;
qsi,w=gausslegendre(npgauss) # Calcula pesos e pontos de Gauss
for i = 1 : n
uk=unique(crv[i].knots)

ne=length(uk)-1;
for j=1:ne
range=[uk[j],uk[j+1]]
mid=(uk[j+1]+uk[j])/2
for k=1:ncollocpoints
xfonte=collocCoord[k,1]
yfonte=collocCoord[k,2]


# Integrais de dominio
g,h,id=integra_elem(xfonte,yfonte,crv[i],dcrv[i],range,mid,qsi,w,kmat,collocPts[k])

# Integracao sobre o elemento (I = int F n.r/r dGama)
H[k,id+nnos[i]]=H[k,id+nnos[i]]+h;
G[k,id+nnos[i]]=G[k,id+nnos[i]]+g;

end
end
end
return H,G
end
function  integra_elem(xfonte,yfonte,crv,dcrv,range,mid,qsi,w,k,cpt)

# Integracao sobre os elementos (integral I)

# Numero de pontos de Gauss usados na integracao de I
npg = length(qsi);

dudqsi=(range[2]-range[1])/2
g = zeros(crv.order)
h = zeros(crv.order)
id = zeros(crv.order)
eet = (range[1]+range[2]-2*cpt)/(range[1]-range[2]);
eta,Jt=telles(qsi,eet);

# id=zeros(Int,crv.order,1)
```

```
for i = 1 : npg # Percorre os pontos de integracao
p,dp=nrbdeval(crv,dcrv, eta[i]/2*(range[2]-range[1])+mid)
B, id = nrbbasisfun(crv,eta[i]/2*(range[2]-range[1])+mid)
dgamadu=norm(dp)
x=p[1]-xfonte # Calcula a coordenada x do ponto de integracao
y=p[2]-yfonte # Calcula a coordenada y do ponto de integracao
dxdqsi=dp[1];
dydqsi=dp[2];
dgamadqsi=sqrt(dxdqsi^2+dydqsi^2);

sx=dxdqsi/dgamadqsi; # Componente x do vetor tangente
sy=dydqsi/dgamadqsi; # Componente y do vetor tangente

nx=sy; # Componente x do vetor normal unitario
ny=-sx; # Componente y do vetor normal unitario

r=sqrt(x^2+y^2); # raio
rx=x/r; # Componente x do vetor unitario r
ry=y/r; # Componente y do vetor unitario r
nr=nx*rx+ny*ry; # Produto escalar dos vetores unitarios r e n
Tast=-1/(2*pi*k)*log(r)
qast=1/(2*pi)*(rx*nx+ry*ny)/r

h = h + B*qast*dgamadu *dudqsi* w[i]* Jt[i]
g = g + B*Tast*dgamadu *dudqsi* w[i]* Jt[i]

end
return g,h,id
end

function aplica_CDC(G,H,CDC,E)
# Aplica as condicoes de contorno trocando as colunas das matrizes H e G

ncdc = size(CDC,1); # numero de linhas da matriz CDC
A=H;
B=G;
for i=1:ncdc # Laco sobre as condicoes de contorno
tipoCDC = CDC[i,2]; # Tipo da condicao de contorno
if tipoCDC == 0 # A temperatura eh conhecida
colunaA=-A[:,i]; # Coluna da matriz H que sera trocada
A[:,i]=-B[:,i]; # A matriz H recebe a coluna da matriz G
B[:,i]=colunaA; # A mstriz G recebe a coluna da matriz H
end
end
valoresconhecidos=E\CDC[:,3] # Valores das condicoes de contorno
b=B*valoresconhecidos; # vetor b
return A,b
end

function monta_Teq(CDC,x)
# Separa fluxo e temperatura
```

```
100
101 # ncdc = numero de linhas da matriz CDC
102 # T = vetor que contem as temperaturas nos nos
103 # q = vetor que contem o fluxo nos nos
104
105 ncdc = size(CDC,1);
106 T=zeros(ncdc)
107 q=zeros(ncdc)
108 for i=1:ncdc # Laco sobre as condicoes de contorno
109 tipoCDC=CDC[i,2] # Tipo da condicao de contorno
110 valorCDC=CDC[i,3] # Valor da condicao de contorno
111 valorcalculado=x[i] # Valor que antes era desconhecido
112 if tipoCDC == 1 # Fluxo eh conhecido
113 T[i] = valorcalculado; # A temperatura eh o valor calculado
114 q[i] = valorCDC; # O fluxo eh a condicao de contorno
115 else # A temperatura eh conhecida
116 T[i] = valorCDC; # A temperatura eh a condicao de contorno
117 q[i] = valorcalculado; # O fluxo eh o valor calculado
118 end
119 end
120 return T,q
121 end
122 end{lstlisting}
123
124 \newpage
125 \subsection*{Programa formatiso}
126 \label{Anexo 4}
127
128 \begin{lstlisting}
129 function calcula_arco(x1,y1,x2,y2,xc,yc,raio)
130 # Function to compute the tet1 angle between the line from point (x1,y1) to (xc,yc) and the
131 # horizontal direction and the the tet2 angle between the line from point (x2,y2) to (xc,yc)
132 # and the horizontal direction
133
134 dx1 = x1 - xc; dy1 = y1 - yc
135 dx2 = x2 - xc; dy2 = y2 - yc
136 tet1=atan2(dy1,dx1);
137 a=[dx1,dy1,0];
138 b=[dx2,dy2,0];
139 angle = atan2(norm(cross(a,b)),dot(a,b));
140 if(raio>0)
141 tet2=tet1+angle;
142 else
143 tet2=tet1-angle;
144 end
145 [tet1,tet2]
146 end
147
148 function calcula_centro(x1,y1,x2,y2,raio)
149 # Compute the center of an arc given two points and the radius
150
```

```
151 xm=(x1+x2)/2
152 ym=(y1+y2)/2
153 b=sqrt((x2-x1)^2+(y2-y1)^2)
154 t1=(x2-x1)/b
155 t2=(y2-y1)/b
156 n1=t2
157 n2=-t1
158 h=sqrt(abs(raio^2-(b/2)^2))
159 if(raio>0)
160 if(n1==0)
161 xc=xm
162 yc=ym-n2/abs(n2)*h
163 else
164 xc=-n1/abs(n1)*sqrt(h^2*n1^2/(n1^2+n2^2))+xm;
165 yc=n2/n1*(xc-xm)+ym
166 end
167 else
168 if(n1==0)
169 xc=xm
170 yc=ym+n2/abs(n2)*h
171 else
172 xc=n1/abs(n1)*sqrt(h^2*n1^2/(n1^2+n2^2))+xm;
173 yc=n2/n1*(xc-xm)+ym
174 end
175 end
176 [xc,yc]
177 end
178
179 function calc_ncont(SEGMENTOS)
180 num_seg=size(SEGMENTOS,1);
181 t=1;
182 p2=0;
183 p_ini = SEGMENTOS[1,2];
184 icont=1; # indice de contornos
185 contorno=zeros(Int32,1,2)
186 contorno[1,1]=p_ini;
187 while(t<num_seg)    # While over all lines
188 while(p2!=p_ini)
189 p1  = SEGMENTOS[t,2]
190 p2  = SEGMENTOS[t,3]
191 if p2 == p_ini
192 if t < num_seg
193 p_ini = SEGMENTOS[t+1,2]
194 icont=icont+1;
195 contorno=[contorno; zeros(Int32,1,2)]
196 contorno[icont,1]=p_ini;
197 contorno[icont-1,2]=p_ini-contorno[icont-1,1]
198 end;
199 end;
200 t=t+1
201 end                              #end of while p2
```

```
202 end
203 if(icont>1)
204 contorno[icont,2]=num_seg-sum(contorno[1:icont-1,2]);
205 else
206 contorno[1,2]=num_seg;
207 end
208 return contorno
209 end
210
211 function format_dad_iso(PONTOS,SEGMENTOS)
212 contorno=calc_ncont(SEGMENTOS);
213 ncont=size(contorno,1);
214 icrv=0;
215 ncurves=sum(contorno[:,2])
216 crv=Array{Curve}(ncurves)
217 jj=0;
218 for j=1:ncont
219 pini=contorno[j,1]; # Ponto onde comeca o contorno j
220 nseg=contorno[j,2];
221 for i=1:nseg
222 icrv=icrv+1;
223 raio=SEGMENTOS[i+pini-1,4] #define valores para o raio
224 np1=Int32(SEGMENTOS[i+pini-1,2]); # Define as coordenas x
225 np2=Int32(SEGMENTOS[i+pini-1,3]); #Define as coordenasdas y
226 p1=[PONTOS[np1,2], PONTOS[np1,3],0]; #Define o primeiro ponto da curva
227 p2=[PONTOS[np2,2], PONTOS[np2,3],0]; #Define o segundo ponto da curva
228 if(raio==0)
229 crv[icrv]=nrbline(p1,p2);
230 else
231 xc,yc=calcula_centro(p1[1],p1[2],p2[1],p2[2],raio);
232 sang,eang = calcula_arco(p1[1],p1[2],p2[1],p2[2],xc,yc,raio);
233 centro=[xc,yc,0];
234 crv[icrv]=nrbcirc(raio,centro,sang,eang);
235 end
236 end
237 end
238 return crv,contorno
239 end
240
241 function monta_Teq(tipoCDC,valorCDC,x)
242 # Separa fluxo e temperatura
243
244 # ncdc = numero de linhas da matriz CDC
245 # T = vetor que contem as temperaturas nos nos
246 # q = vetor que contem o fluxo nos nos
247
248 ncdc = length(tipoCDC)
249 T=zeros(ncdc)
250 q=zeros(ncdc)
251 for i=1:ncdc # Laco sobre as condicoes de contorno
252 if tipoCDC[i] == 1 # Fluxo eh conhecido
```

```
253 T[i] = x[i]; # A temperatura eh o valor calculado
254 q[i] = valorCDC[i]; # O fluxo eh a condicao de contorno
255 else # A temperatura eh conhecida
256 T[i] = valorCDC[i]; # A temperatura eh a condicao de contorno
257 q[i] = x[i]; # O fluxo eh o valor calculado
258 end
259 end
260 return T,q
261 end
262 function mostra_geo(crvs)
263 p=plot(legend=:none,aspect_ratio=:equal)
264 for i in crvs
265 p=nrbplot(i)
266 end
267 p
268 end
```

Função formatiso

```
function calcula_arco(x1,y1,x2,y2,xc,yc,raio)
# Function to compute the tet1 angle between the line from point (x1,y1) to (xc,yc) and the
# horizontal direction and the the tet2 angle between the line from point (x2,y2) to (xc,yc)
# and the horizontal direction

dx1 = x1 - xc; dy1 = y1 - yc
dx2 = x2 - xc; dy2 = y2 - yc
tet1=atan2(dy1,dx1);
a=[dx1,dy1,0];
b=[dx2,dy2,0];
angle = atan2(norm(cross(a,b)),dot(a,b));
if(raio>0)
tet2=tet1+angle;
else
tet2=tet1-angle;
end
[tet1,tet2]
end

function calcula_centro(x1,y1,x2,y2,raio)
# Compute the center of an arc given two points and the radius

xm=(x1+x2)/2
ym=(y1+y2)/2
b=sqrt((x2-x1)^2+(y2-y1)^2)
t1=(x2-x1)/b
t2=(y2-y1)/b
n1=t2
n2=-t1
h=sqrt(abs(raio^2-(b/2)^2))
if(raio>0)
if(n1==0)
xc=xm
yc=ym-n2/abs(n2)*h
else
xc=-n1/abs(n1)*sqrt(h^2*n1^2/(n1^2+n2^2))+xm;
yc=n2/n1*(xc-xm)+ym
end
else
if(n1==0)
xc=xm
yc=ym+n2/abs(n2)*h
else
xc=n1/abs(n1)*sqrt(h^2*n1^2/(n1^2+n2^2))+xm;
yc=n2/n1*(xc-xm)+ym
end
end
[xc,yc]
```

```
49 end
50
51 function calc_ncont(SEGMENTOS)
52 num_seg=size(SEGMENTOS,1);
53 t=1;
54 p2=0;
55 p_ini = SEGMENTOS[1,2];
56 icont=1; # indice de contornos
57 contorno=zeros(Int32,1,2)
58 contorno[1,1]=p_ini;
59 while(t<num_seg)    # While over all lines
60 while(p2!=p_ini)
61 p1  = SEGMENTOS[t,2]
62 p2  = SEGMENTOS[t,3]
63 if p2 == p_ini
64 if t < num_seg
65 p_ini = SEGMENTOS[t+1,2]
66 icont=icont+1;
67 contorno=[contorno; zeros(Int32,1,2)]
68 contorno[icont,1]=p_ini;
69 contorno[icont-1,2]=p_ini-contorno[icont-1,1]
70 end;
71 end;
72 t=t+1
73 end                              #end of while p2
74 end
75 if(icont>1)
76 contorno[icont,2]=num_seg-sum(contorno[1:icont-1,2]);
77 else
78 contorno[1,2]=num_seg;
79 end
80 return contorno
81 end
82
83 function format_dad_iso(PONTOS,SEGMENTOS)
84 contorno=calc_ncont(SEGMENTOS);
85 ncont=size(contorno,1);
86 icrv=0;
87 ncurves=sum(contorno[:,2])
88 crv=Array{Curve}(ncurves)
89 jj=0;
90 for j=1:ncont
91 pini=contorno[j,1]; # Ponto onde comeca o contorno j
92 nseg=contorno[j,2];
93 for i=1:nseg
94 icrv=icrv+1;
95 raio=SEGMENTOS[i+pini-1,4] #define valores para o raio
96 np1=Int32(SEGMENTOS[i+pini-1,2]); # Define as coordenas x
97 np2=Int32(SEGMENTOS[i+pini-1,3]); #Define as coordenasdas y
98 p1=[PONTOS[np1,2], PONTOS[np1,3],0]; #Define o primeiro ponto da curva
99 p2=[PONTOS[np2,2], PONTOS[np2,3],0]; #Define o segundo ponto da curva
```

```
100 if(raio==0)
101 crv[icrv]=nrbline(p1,p2);
102 else
103 xc,yc=calcula_centro(p1[1],p1[2],p2[1],p2[2],raio);
104 sang,eang = calcula_arco(p1[1],p1[2],p2[1],p2[2],xc,yc,raio);
105 centro=[xc,yc,0];
106 crv[icrv]=nrbcirc(raio,centro,sang,eang);
107 end
108 end
109 end
110 return crv,contorno
111 end
112
113 function monta_Teq(tipoCDC,valorCDC,x)
114 # Separa fluxo e temperatura
115
116 # ncdc = numero de linhas da matriz CDC
117 # T = vetor que contem as temperaturas nos nos
118 # q = vetor que contem o fluxo nos nos
119
120 ncdc = length(tipoCDC)
121 T=zeros(ncdc)
122 q=zeros(ncdc)
123 for i=1:ncdc # Laco sobre as condicoes de contorno
124 if tipoCDC[i] == 1 # Fluxo eh conhecido
125 T[i] = x[i]; # A temperatura eh o valor calculado
126 q[i] = valorCDC[i]; # O fluxo eh a condicao de contorno
127 else # A temperatura eh conhecida
128 T[i] = valorCDC[i]; # A temperatura eh a condicao de contorno
129 q[i] = x[i]; # O fluxo eh o valor calculado
130 end
131 end
132 return T,q
133 end
134 function mostra_geo(crvs)
135 p=plot(legend=:none,aspect_ratio=:equal)
136 for i in crvs
137 p=nrbplot(i)
138 end
139 p
140 end
```

Função telles

```
function telles(gamm,eet)

eest = eet^2 - 1;
term1 = eet*eest + abs(eest);
if term1 < 0
term1 = (-term1)^(1/3);
term1 = -term1;
else
term1 = term1^(1/3);
end

term2 = eet*eest - abs(eest);
if term2 < 0
term2 = (-term2)^(1/3);
term2 = -term2;
else
term2 = term2^(1/3);
end
GAMM = term1 + term2 + eet;


Q = 1 + 3*GAMM^2;
A = 1/Q;
B = -3*GAMM/Q;
C = 3*GAMM^2/Q;
D = -B;

eta = A*gamm.^3 + B*gamm.^2 + C*gamm + D;
Jt = 3*A*gamm.^2 + 2*B*gamm + C;
return  eta,Jt
end
```

```
function inpoly(p, node, edge, reltol = 1.0e-12)

#   INPOLY: Point-in-polygon testing.
#
#  Determine whether a series of points lie within the bounds of a polygon
#  in the 2D plane. General non-convex, multiply-connected polygonal
#  regions can be handled.
#
#  SHORT SYNTAX:
#
#    in = inpoly(p, node);
#
#    p    : The points to be tested as an Nx2 array [x1 y1; x2 y2; etc].
#    node: The vertices of the polygon as an Mx2 array [X1 Y1; X2 Y2; etc].
#          The standard syntax assumes that the vertices are specified in
#          consecutive order.
#
#    in   : An Nx1 logical array with IN(i) = TRUE if P(i,:) lies within the
#          region.
#
#  LONG SYNTAX:
#
#   [in, on] = inpoly(p, node, edge, tol);
#
#   edge: An Mx2 array of polygon edges, specified as connections between
#         the vertices in NODE: [n1 n2; n3 n4; etc]. The vertices in NODE
#         do not need to be specified in connsecutive order when using the
#         extended syntax.
#
#   on   : An Nx1 logical array with ON(i) = TRUE if P(i,:) lies on a
#         polygon edge. (A tolerance is used to deal with numerical
#         precision, so that points within a distance of
#         reltol*min(bbox(node)) from a polygon edge are considered "on" the
#         edge.
#
#  EXAMPLE:
#
#    polydemo;       #  Will run a few examples
#
#  See also INPOLYGON

#  The algorithm is based on the crossing number test, which counts the
#  number of times a line that extends from each point past the right-most
#  region of the polygon intersects with a polygon edge. Points with odd
#  counts are inside. A simple implementation of this method requires each
#  wall intersection be checked for each point, resulting in an O(N*M)
#  operation count.
#
```

```
#  This implementation does better in 2 ways:
#
#    1. The test points are sorted by y-value and a binary search is used to
#       find the first point in the list that has a chance of intersecting
#       with a given wall. The sorted list is also used to determine when we
#       have reached the last point in the list that has a chance of
#       intersection. This means that in general only a small portion of
#       points are checked for each wall, rather than the whole set.
#
#    2. The intersection test is simplified by first checking against the
#       bounding box for a given wall segment. Checking against the bbox is
#       an inexpensive alternative to the full intersection test and allows
#       us to take a number of shortcuts, minimising the number of times the
#       full test needs to be done.
#
#    Darren Engwirda: 2005-2009
#    Email          : d_engwirda@hotmail.com
#    Last updated   : 28/03/2009 with MATLAB 7.0
#
#  Problems or suggestions? Email me.
nnode = size(node,1);

# #  PRE-PROCESSING
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
n  = size(p,1);
nc = size(edge,1);

#  Choose the direction with the biggest range as the "y-coordinate" for the
#  test. This should ensure that the sorting is done along the best
#  direction for long and skinny problems wrt either the x or y axes.
dxy = maximum(p,1)-minimum(p,1);
if dxy[1]>dxy[2]
#  Flip co-ords if x range is bigger
p = p[:,[2,1]];
node = node[:,[2,1]];
end

#  Polygon bounding-box
dxy = maximum(node,1)-minimum(node,1);
tol = reltol*minimum(dxy);
if tol==0.0
tol = reltol;
end

#  Sort test points by y-value
i = sortperm(p[:,2]);
y= p[i,2];
x = p[i,1];
# #  MAIN LOOP
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
cn = falses(n);     #  Because we're dealing with mod(cn,2) we don't have
```

```
#  to actually increment the crossing number, we can
#  just flip a logical at each intersection (faster!)
on = cn[:];
for k = 1:nc          #  Loop through edges
#  Nodes in current edge
n1 = edge[k,1];
n2 = edge[k,2];

#  Endpoints - sorted so that [x1,y1] & [x2,y2] has y1<=y2
#            - also get xmin = min(x1,x2), xmax = max(x1,x2)
y1 = node[n1,2];
y2 = node[n2,2];
if y1<y2
x1 = node[n1,1];
x2 = node[n2,1];
else
yt = y1;
y1 = y2;
y2 = yt;
x1 = node[n2,1];
x2 = node[n1,1];
end
if x1>x2
xmin = x2;
xmax = x1;
else
xmin = x1;
xmax = x2;
end
#  Binary search to find first point with y<=y1 for current edge
if y[1]>=y1
start = 1;
elseif y[n]<y1
start = n+1;
else
lower = 1;
upper = n;
for j = 1:n
start = convert(Int32,round(1/2*(lower+upper)));
if y[start]<y1
lower = start+1;
elseif y[start-1]<y1
break;
else
upper = start-1;
end
end
end
#  Loop through points
for j = start:n
#  Check the bounding-box for the edge before doing the intersection
```

```
151 #  test. Take shortcuts wherever possible!
152 Y = y[j];   #  Do the array look-up once & make a temp scalar
153 if Y<=y2
154 X = x[j];   #  Do the array look-up once & make a temp scalar
155 if X>=xmin
156 if X<=xmax
157
158 #  Check if we're "on" the edge
159 on[j] = on[j] || (abs((y2-Y)*(x1-X)-(y1-Y)*(x2-X))<=tol);
160
161 #  Do the actual intersection test
162 if (Y<y2) && ((y2-y1)*(X-x1)<(Y-y1)*(x2-x1))
163 cn[j] = ~cn[j];
164 end
165
166 end
167 elseif Y<y2   #  Deal with points exactly at vertices
168 #  Has to cross edge
169 cn[j] = ~cn[j];
170 end
171 else
172 #  Due to the sorting, no points with >y
173 #  value need to be checked
174 break
175 end
176 end
177 end
178 #  Re-index to undo the sorting
179 cn[i] = cn|on;
180 on[i] = on;
181
182 return cn;
183
184 end      #  inpoly()
```

Função mostra_problema

```
function mostra_heatmap(NOS,T,ELEM)
nelem=size(ELEM,1);
figure();

XY=NOS;
triang = tri.Triangulation(XY[:,1], XY[:,2])
cor=T
t=triang[:triangles]+1
centroid=(XY[t[:,1],:]+XY[t[:,2],:]+XY[t[:,3],:])/3.0;
i = inpoly(centroid,NOS,ELEM);
ind=collect(1:size(t,1));
ind=ind[i];
#  Take triangles with internal centroids
t = t[ind,:];

nelem=size(ELEM,1);
plt.plot(NOS[:,1],NOS[:,2],"kx",markersize=4,linewidth=1,label="Pontos de colocacao");

# Plot the node of the elements
plt.grid("on")
plt.legend()

ELEM2=[ELEM ELEM[:,2]];

ncont=50;
# plt.triplot(XY[:,1], XY[:,2], t-1, color=(0.0,0.,0.),linewidth=0.4)
plt.triplot(NOS[:,1], NOS[:,2], ELEM2-1, color=(0.0,0.,0.),linewidth=0.4)

plt.tricontourf(XY[:,1], XY[:,2], t-1, cor, ncont)
plt.colorbar()
plt.axis("equal")
plt.title("Temperatura")
ax = plt.gca() # get current axes
end
```