PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

João Carlos Leão Peixoto

Análise isogeométrica com modelagem interativa de múltiplas regiões NURBS e T-Splines

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-Graduação em Engenharia Civil da PUC-Rio.

Orientador: Prof. Luiz Fernando Campos Ramos Martha

Rio de Janeiro

Fevereiro de 2024

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

João Carlos Leão Peixoto

Análise isogeométrica com modelagem interativa de múltiplas regiões NURBS e T-Splines

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-Graduação em Engenharia Civil da PUC-Rio. Aprovada pela Comissão Examinadora abaixo.

Prof. Luiz Fernando Campos Ramos Martha
Orientador
Departamento de Engenharia Civil – PUC-Rio

Prof. Evandro Parente Junior
Departamento de Engenharia Estrutural e Construção Civil – UFC

Prof. André Maués Brabo Pereira
Departamento de Ciência da Computação – UFF

Rio de Janeiro, 26 de Fevereiro de 2024

João Carlos Leão Peixoto

Graduou-se em Engenharia Civil na Universidade Federal de Alagoas (UFAL) em 2020. Iniciou o curso de mestrado em Engenharia Civil (área de estruturas) na Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) em 2021. Atua na linha de pesquisa de Computação Gráfica aplicada ao desenvolvimento de ferramentas para Mecânica Computacional.

Ficha catalográfica

Peixoto, João Carlos Leão

Análise isogeométrica com modelagem interativa de múltiplas regiões NURBS e T-Splines / João Carlos Leão Peixoto; orientador: Luiz Fernando Campos Ramos Martha. – 2024.
182 f. : il. color. ; 30 cm

Dissertação (mestrado)–Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Civil e Ambiental, 2024.
Inclui bibliografia

1. Engenharia Civil e Ambiental - Teses. 2. Análise isogeométrica. 3. NURBS. 4. T-Splines. 5. Mecânica computacional. 6. Interface gráfica de usuário. I. Martha, Luiz Fernando Campos Ramos. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Civil e Ambiental. III. Título.

COD 624

Agradecimentos

Gostaria de começar agradecendo a Deus por ter me dado força e sabedoria durante todo o processo de realização deste trabalho.

Também expresso minha gratidão à PUC-Rio, pela oportunidade e confiança que me foi dada para ingressar no programa de Pós-graduação em Engenharia Civil.

Quero agradecer ao meu orientador, Luiz Fernando Martha, pela paciência e apoio durante toda a minha pesquisa. Sua dedicação e conhecimento foram fundamentais para o desenvolvimento deste projeto.

Agradeço à minha mãe, Márcia, por seu amor incondicional e incentivo em todos os momentos da minha vida. À minha noiva, Rhayane, por ser meu porto seguro, minha fonte de inspiração e motivação.

Agradeço a todos os professores, por tornarem possível o meu mestrado, mesmo em um período de restrições e desafios. Aos meus colegas e amigos pela valiosa ajuda que me ofereceram durante o curso.

O presente trabalho foi realizado com o apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Este trabalho também foi realizado com o apoio da Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro - (FAPERJ).

Resumo

Peixoto, J. C.; Martha, L. F. **Análise isogeométrica com modelagem interativa de múltiplas regiões NURBS e T-Splines**. Rio de Janeiro, 2024. 182p. Dissertação de Mestrado - Departamento de Engenharia Civil, Pontifícia Universidade Católica do Rio de Janeiro.

A Análise Isogeométrica (IGA) é um método de análise numérica de estruturas que surge com a proposta de unificação entre projeto e simulação, permitindo a criação de modelos computacionais que preservam a geometria exata do problema. Essa abordagem é possível por meio de uma classe de funções matemáticas denominadas NURBS (Non-Uniform Rational B-Splines), amplamente utilizadas em sistemas CAD (Computer-Aided Design) para modelagem de curvas e superfícies. Na análise isogeométrica, as mesmas funções que representam a geometria aproximam as variáveis de campo. Neste contexto, foi desenvolvido este trabalho que tem como objetivo fornecer uma ferramenta no âmbito da mecânica computacional para análise isogeométrica bidimensional de problemas de elasticidade linear, incluindo as etapas de modelagem, análise e visualização de resultados. O sistema é composto por dois softwares: FEMEP (Finite Element Method Educational Computer Program), desenvolvido em Python e responsável pela etapa de modelagem geométrica, e FEMOOLab (Finite Element Method Object-Oriented Laboratory), software MATLAB para análise e exibição de resultados. A ferramenta proposta apresenta uma interface gráfica de usuário (GUI) que permite a visualização e manipulação intuitiva de curvas NURBS com recursos avançados de modelagem, como interseção de curvas e recursos de reconhecimento de região que agilizam e simplificam o processo. Uma contribuição significativa deste trabalho reside na capacidade de gerar malhas isogeométricas não estruturadas, utilizando T-Splines baseadas em um algoritmo de decomposição de domínio. O sistema de código aberto permite a colaboração e o desenvolvimento contínuo pela comunidade de usuários e desenvolvedores.

Palavras-chave

Análise Isogeométrica; NURBS; T-Splines; Mecânica Computacional; Interface Gráfica de Usuário.

Abstract

Peixoto, J. C.; Martha, L. F. **Isogeometric analysis with interactive modeling of multiple NURBS and T-Splines patches**. Rio de Janeiro, 2024. 182p. Dissertação de Mestrado - Departamento de Engenharia Civil, Pontifícia Universidade Católica do Rio de Janeiro.

Isogeometric Analysis (IGA) is a numerical analysis method for structures that arises with the proposal of unification between design and simulation, allowing the creation of computational models that preserve the exact geometry of the problem. This approach is possible by a class of mathematical functions called NURBS (Non-Uniform Rational B-Splines), widely used in CAD (Computer-Aided Design) systems for modeling curves and surfaces. In isogeometric analysis, the same functions representing the geometry approximate the field variables. In this context, this work was developed to provide a tool within the scope of computational mechanics for two-dimensional isogeometric analysis of linear elasticity problems, including the steps of modeling, analysis, and visualization of results. The system consists of two software programs: FEMEP (Finite Element Method Educational Computer Program), developed in Python and responsible for the geometric modeling stage, and FEMOOLab (Finite Element Method Object-Oriented Laboratory), a MATLAB software for analysis and display of results. The proposed tool features a graphical user interface (GUI) that allows intuitive visualization and manipulation of NURBS curves with advanced modeling features such as curve intersection and region recognition features that streamline and simplify the process. A significant contribution of this work lies in the ability to generate non-structured isogeometric meshes, using T-Splines based on a domain decomposition algorithm. The open-source system allows collaboration and continuous development by the community of users and developers.

Keywords

Isogeometric Analysis; NURBS; T-Splines; Computational Mechanics; Graphical User Interface.

Sumário

Lista de Figuras

Lista de Símbolos

$\widetilde{\Omega}$	Domínio no espaço *parent*
$\widehat{\Omega}$	Domínio no espaço paramétrico
Ω	Domínio no espaço físico
d_p	Número de dimensões no espaço paramétrico
d_s	Número de dimensões no espaço físico
Ξ	Vetor de *knots* na direção ξ
$\mathcal{H}$	Vetor de *knots* na direção η
$\widehat{\Xi}$	Vetor de *knots* únicos na direção ξ
$\widehat{\mathcal{H}}$	Vetor de *knots* únicos na direção η
$\Delta\Xi$	Vetor de intervalos de *knots* na direção ξ
$\Delta\mathcal{H}$	Vetor de intervalos de *knots* na direção η
$\tilde{\xi}, \tilde{\eta}$	Direções no espaço *parent* / Referência para coordenadas no espaço *parent*
ξ, η	Direções paramétricas / Referência para coordenadas paramétricas
x, y	Coordenadas cartesianas
x^w, y^w, z^w	Coordenadas homogêneas
ξ_i	*Knot* de índice i na direção ξ
η_i	*Knot* de índice i na direção η
p	Ordem polinomial na direção ξ
q	Ordem polinomial na direção η
n	Número de funções de base / Número de pontos de controle na direção ξ
m	Número de funções de base / Número de pontos de controle na direção η / Novo número de funções base e pontos de controle após refinamento
m_i	Multiplicidade do *knot* de índice i
$N_{i,p}(\xi)$	Função de base B-Spline de índice i na direção ξ, ordem polinomial p
$M_{j,q}$	Função de base B-Spline de índice j na direção η, ordem polinomial q

$B_{i,p}(\xi)$	Polinômios de Bernstein de índice i na direção ξ, ordem polinomial p
$\boldsymbol{N}$	Vetor de funções de base B-Splines
$\boldsymbol{C}(\xi)$	Curva B-Spline ou NURBS
$\boldsymbol{C}^w(\xi)$	Curva B-Spline projetiva
$\boldsymbol{C}^b$	Curva Bézier
$\boldsymbol{C}$	Operador de extração
$\boldsymbol{S}(\xi,\eta)$	Superfície B-Spline ou NURBS
$\boldsymbol{P}$	Ponto de controle
$\boldsymbol{P}^w$	Ponto de controle projetivo
$\boldsymbol{P}^b$	Ponto de controle Bézier
w	Peso associado ao ponto de controle
$\boldsymbol{W}$	Matriz diagonal de pesos
$\boldsymbol{w}$	Vetor de pesos dos pontos de controle
$\boldsymbol{w}$	Vetor de pesos dos pontos de controle Bézier
A	Índice global
s	*Anchor*
α	Coeficiente de inserção de *knots*
$R_{i,p}(\xi)$	Função de base NURBS univariada de índice i na direção ξ, ordem polinomial p
$\boldsymbol{R}$	Vetor de funções de base NURBS
$W(\xi)$	Função peso
$R_{i,j}^{p,q}(\xi,\eta)$	Função de base NURBS bivariada. Na direção ξ: índice i e ordem polinomial p. Na direção η: índice j e ordem polinomial q
$\boldsymbol{S_{Coons}}(\xi,\eta)$	Superfície Coons
$\boldsymbol{R_1}(\xi,\eta)$	Superfície *ruled* em ξ
$\boldsymbol{R_2}(\xi,\eta)$	Superfície *ruled* em η
$\boldsymbol{T}(\xi,\eta)$	Superfície bilinear
e	Elemento
n_{en}	Número de funções de base não nulas no elemento / Número de pontos de controle de conectividade

a	Índice de funções de base do elemento / Índice de pontos de controle em conectividade com o elemento
$\boldsymbol{d}$	Variáveis de controle
$\boldsymbol{\phi}^e(\tilde{\xi}, \tilde{\eta})$	Mapeamento do espaço *parent* ao espaço paramétrico de um elemento
$\boldsymbol{S}^e(\xi, \eta)$	Mapeamento do espaço paramétrico ao espaço físico de um elemento
$\boldsymbol{x}^e(\tilde{\xi}, \tilde{\eta})$	Mapeamento do espaço *parent* ao espaço físico de um elemento
$\boldsymbol{J}_{\tilde{\xi}}$	Jacobiano associado ao mapeamento $\boldsymbol{\phi}^e$
$\boldsymbol{J}_{\xi}$	Jacobiano associado ao mapeamento $\boldsymbol{S}^e$
$\boldsymbol{\sigma}$	Vetor de tensões
σ_{xx}	Componente de tensão normal em x
σ_{yy}	Componente de tensão normal em y
τ_{xy}	Componente de tensão de cisalhamento
$\boldsymbol{E}$	Matriz constitutiva
E	Módulo de elasticidade
ν	Coeficiente de Poisson
ε	Vetor de deformações
ε_{xx}	Componente de deformação normal em x
ε_{yy}	Componente de deformação normal em y
γ_{xy}	Componente de deformação de cisalhamento
$\boldsymbol{u}$	Vetor de deslocamentos
$\boldsymbol{D}$	Operador diferencial
$\boldsymbol{R}$	Matriz de funções de base
$\boldsymbol{B}$	Matriz deformação-deslocamento / Vetor de polinômios de Bernstein
$\boldsymbol{K}$	Matriz de rigidez
$\boldsymbol{f}$	Vetor de forças externas

1 Introdução

O Método dos Elementos Finitos (MEF) consiste em uma técnica de análise numérica amplamente difundida e utilizada na resolução de diversos problemas de engenharia como: análise de tensões, transferência de calor, mecânica dos fluidos e eletromagnetismo [1]. Apesar de ser uma ferramenta poderosa para modelar e analisar estruturas, observa-se uma limitação significativa: a dificuldade e o tempo consumido na geração de modelos de análise a partir de um modelo de projeto.

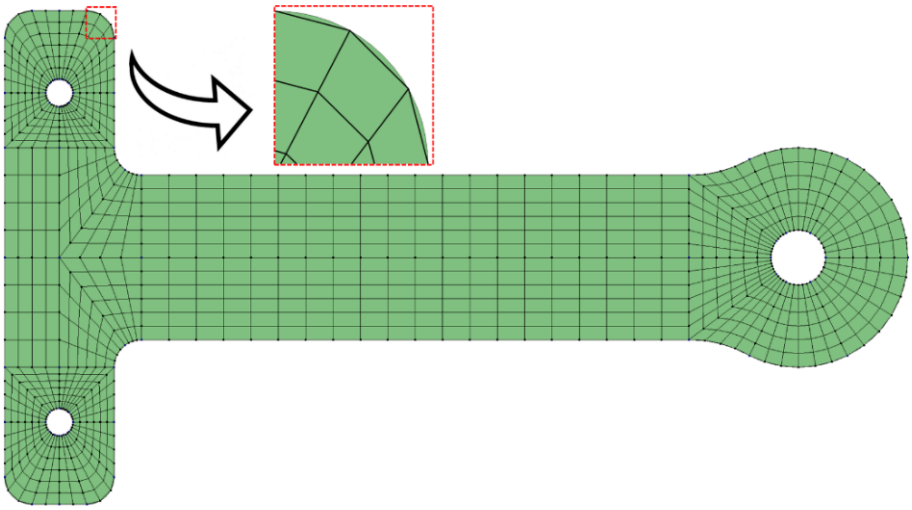

Figura 1: Malha de elementos finitos comparada à geometria original de uma peça mecânica.

Para realizar a análise por meio do MEF, é necessário efetuar previamente a subdivisão do domínio do problema em elementos, como triângulos ou quadriláteros, processo conhecido como geração de malha. Cada elemento é caracterizado por um número finito de graus de liberdade, que representam a resposta mecânica do elemento [2]. As equações diferenciais parciais que governam o comportamento do sistema são reformuladas em uma forma integral chamada formulação fraca [1], que é aproximada por meio de funções de interpolação definidas em termos dos graus de liberdade. A partir daí, um sistema de equações algébricas é formado, que é então resolvido numericamente para obter a solução do problema [3]. Verifica-se, no entanto, que a geração de malhas precisas e adequadas à geometria envolve um processo cuidadoso e demorado, que pode refletir na

precisão dos resultados obtidos pelo Método dos Elementos Finitos. Estima-se que 80% do tempo total de análise em problemas mais complexos é consumido apenas na geração de malha [4]. Com o avanço da tecnologia, projetos estão se tornando cada vez mais desafiadores, o que pode aumentar ainda mais o tempo necessário para gerar uma malha adequada.

Como pode ser observado na Figura 1, que representa uma peça metálica baseada em um exemplo de Bathe [5], ao comparar a malha de elementos finitos com a geometria original do modelo, é preciso estar ciente de que a malha apresenta aproximações em relação à forma real da peça. Nesse sentido, uma técnica de análise numérica avançada conhecida como Análise Isogeométrica (AIG), tem despertado crescente interesse nos últimos tempos e sido cada vez mais utilizada em diversas áreas, como engenharia aeroespacial, engenharia mecânica, engenharia civil, biologia computacional, indústria automobilística, entre outras [4, 6]. Esse método de análise numérica foi inicialmente desenvolvido por Hughes et al. [6] e surge com a proposta de oferecer uma alternativa ao Método dos Elementos Finitos tradicional. A análise isogeométrica oferece uma abordagem diferenciada, usando funções matemáticas chamadas NURBS (*Non-Uniform Rational B-Splines*) para representar a geometria do problema. A mesma classe de funções é utilizada para aproximar as variáveis de campo da análise. Isso minora erros de aproximação de malha, uma vez que a geometria é preservada de maneira exata durante a análise. Os modelos de geometria e análise não são mais entidades separadas, mas sim uma entidade única. Na formulação de elementos finitos, as aproximações geométricas estão comumente presentes devido às funções polinomiais, que são empregadas para interpolar a forma original do problema. Dessa maneira, a AIG acaba resultando em um processo com maior precisão e convergência mais rápida.

A grande vantagem da análise isogeométrica em relação ao Método dos Elementos Finitos está na possibilidade de unificação entre os sistemas CAD (*Computer-Aided Design*) e análise. Isso significa que o processo de modelagem e análise de estruturas é integrado e bidirecional [4], possibilitando que as alterações feitas em um sistema sejam automaticamente refletidas no outro. O procedimento em questão é ilustrado na Figura 2, onde é exibido um modelo NURBS de um automóvel, no qual ocorre alteração no design. Essa estratégia proporciona uma nova dimensão para o projeto e análise de estruturas, permitindo que os engenheiros

tenham um melhor controle sobre todo o processo. A AIG é especialmente útil em problemas de análise que envolvem geometrias complexas, onde a representação precisa da geometria é essencial para a solução do problema.

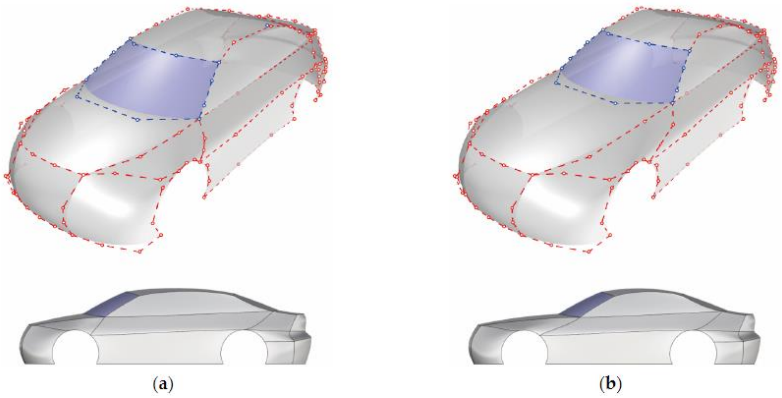

Figura 2: (a) Modelo NURBS simplificado de um veículo automotivo. (b) Modificação na geometria do modelo [7].

Devido à sua capacidade de lidar com geometrias das mais variadas formas, a análise isogeométrica tem se mostrado uma ferramenta valiosa em muitas áreas, desde a modelagem de estruturas de aviões [8, 9], aplicação na indústria automotiva [10] e até na simulação de fluxo sanguíneo em artérias [11, 12], como mostrado na Figura 3. Essa técnica de análise inovadora está transformando a forma como os engenheiros lidam com problemas de análise estrutural e abrindo novas possibilidades para a resolução de problemas cada vez mais exigentes.

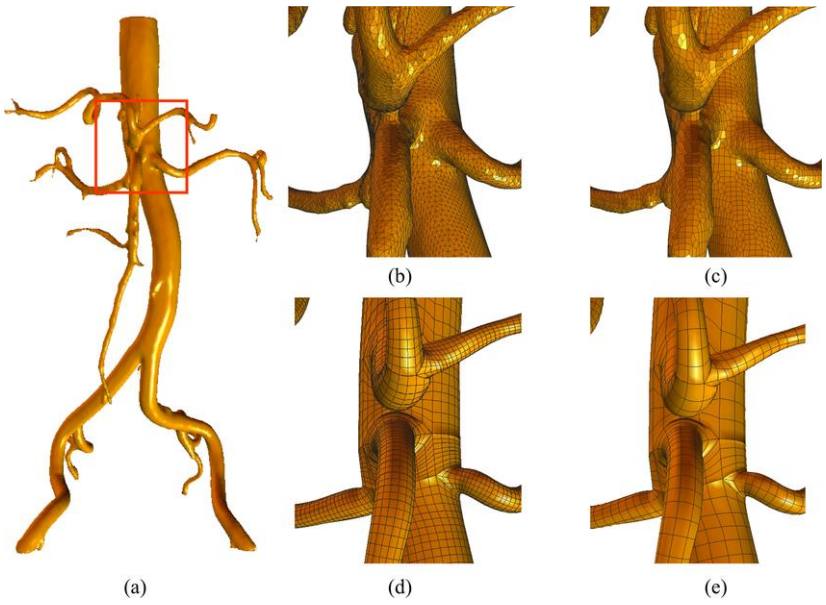

Figura 3: Representação de uma aorta. (a) Modelo geométrico de superfície. (b), (c), (d) Diferentes malhas de elementos finitos. (e) Malha NURBS [11].

As NURBS são padrões convencionais da indústria de *Computer-Aided Design*, sendo amplamente utilizadas em softwares de modelagem, além de serem base fundamental da análise isogeométrica. Apesar das NURBS serem indicadas para estabelecer a ponte inicial entre CAD e AIG, esse processo pode não ser tão simples e direto. Muitas vezes, modelos NURBS são compostos por múltiplos *patches*, isto é, regiões distintas que se unem para formar uma única representação de um objeto. Essa composição pode levar a espaçamentos vazios entre os *patches* durante e execução da análise [13]. Ademais, uma limitação das NURBS reside nos refinamentos, uma vez que a estrutura NURBS permite apenas refinamento global, abrangendo todo o domínio, e não localizados em regiões específicas [13]. T-Splines [14], que são uma generalização de NURBS, oferecem uma solução para essas questões através da introdução de T-Junctions e *extraordinary points.*

A utilização de T-Splines na análise isogeométrica é discutida por Scott et al. [15], através de uma técnica chamada extração Bézier. Essa técnica se fundamenta na definição de um operador que mapeia funções Bézier para funções NURBS [16] ou T-Splines [15], apresentando a vantagem de facilitar a adaptação de algoritmos de MEF para incorporar a AIG [15, 16]. Uma contribuição significativa das T-Splines é a sua capacidade de viabilizar a criação de malhas isogeométricas não estruturadas por meio de *extraordinary points*, conforme formulação desenvolvida na tese de Scott [17].

1.1 Motivação e Objetivo

Com o objetivo de auxiliar a comunidade acadêmica, estudantes, professores e pesquisadores, foi desenvolvido esse trabalho que visa fornecer um software interativo para modelagem geométrica bidimensional de múltiplos *patches* NURBS e T-Splines, em conjunto com uma ferramenta adicional para análise isogeométrica de modelos gerados no modelador. As regiões do tipo NURBS permitem a formação de malhas estruturadas, enquanto T-Splines com *extraordinary points*, permitem a geração de malhas não estruturadas. O sistema completo, disponível em código aberto, foi concebido com a intenção de oferecer uma plataforma didática para explorar os conceitos e princípios da AIG. Nesse sentido, o sistema desenvolvido pode ser considerado um software educacional.

O modelador de sólidos 2D é denominado FEMEP, sendo desenvolvido originalmente, dentro da mesma linha de pesquisa, por Bomfim [18] para modelagem de modelos isoparamétricos de elementos finitos. A implementação foi realizada em Python com o paradigma de programação orientada a objeto (POO). O aplicativo conta com uma interface gráfica de usuário, desenvolvida com o *framework* Qt. A representação visual do modelo é realizada pela biblioteca OpenGL, responsável pela renderização de elementos gráficos na tela. A análise dos modelos gerados pelo FEMEP é realizada pela ferramenta intitulada FEMOOLab, implementada em MATLAB sob a arquitetura de POO.

Uma das motivações para este trabalho reside na complexidade inerente à compreensão de NURBS. Essa abordagem matemática, embora seja poderosa em termos de representação de formas complexas, pode ser intimidadora para iniciantes. O software educacional busca desmistificar as NURBS, fornecendo explicações claras e demonstrações visuais interativas que facilitam o entendimento dos conceitos subjacentes. Com uma interface intuitiva e funcionalidades que permitem manipular e visualizar as curvas e superfícies NURBS em tempo real, a comunidade poderá explorar e experimentar os princípios da modelagem, aprimorando sua compreensão prática.

A possibilidade de explorar as potencialidades da análise isogeométrica, um método numérico que tem ganhado destaque recentemente, é o ponto de relevância da ferramenta. Com a crescente importância desse método, o software educacional surge como uma forma de oferecer uma abordagem inicial à AIG, fornecendo aos usuários a oportunidade de investigar a interação entre modelagem e análise em um contexto didático, facilitando o aprendizado e o uso prático dessa nova técnica.

Uma contribuição deste trabalho reside na capacidade de gerar malhas isogeométricas não estruturadas por meio da aplicação de T-Splines, tomando como partida um algoritmo de decomposição de domínio desenvolvido por Miranda e Martha [19]. Esta abordagem permite uma representação mais flexível e adaptável da geometria, superando as limitações das malhas estruturadas convencionais e, ao mesmo tempo, incorporando o conceito de análise isogeométrica.

Adicionalmente, o sistema promove a colaboração e a disseminação de conhecimento. Por ser um projeto de código aberto, permite que a comunidade

contribua para seu aprimoramento contínuo, além de possibilitar a criação de recursos complementares e a troca de informações entre usuários e desenvolvedores.

1.2 Trabalhos Correlatos

A estratégia computacional desenvolvida nesta dissertação é uma combinação de esforços de diversos trabalhos prévios. O desejo de criação de um software de modelagem surgiu em 1991 após trabalhos realizados por Campos [20, 21] sobre técnicas de geração de malhas de elementos finitos com estruturas de dados topológicas, e por Cavalcanti [22] sobre subdivisões planares. A ideia inicial era desenvolver um ambiente de modelagem para elementos finitos, que fosse capaz de representar subdivisões planares. Em 1992, Cavalcanti [23] realizou grandes avanços, originando uma metodologia para construir subdivisões espaciais coerente com a topologia e geometria.

Novos trabalhos foram produzidos posteriormente [24-26], buscando um sistema para simulações da mecânica computacional extensível, configurável e independente do problema a ser investigado. Mais recentemente, a biblioteca HETOOL [27-28] surgiu como uma convergência dos progressos anteriores. Implementado em Python e contando com uma formatação JSON, o HETOOL possibilitou o desenvolvimento de um ambiente de modelagem bidimensional baseado em subdivisões planares. O gerenciamento de atributos é um dos aspectos fundamentais da biblioteca HETOOL, que proporciona versatilidade para a representação de problemas mecânicos. Um atributo de modelagem pode ser, por exemplo, uma carga distribuída ao longo de uma aresta, propriedades de um material, ou uma condição de suporte.

Demonstrando a aplicabilidade dessa biblioteca, Bomfim [18] criou a ferramenta chamada FEMEP, que permitiu a criação de modelos planos utilizando curvas poligonais e a geração de malhas de elementos finitos. O programa conta com alguns recursos de modelagem como a interseção automática de curvas e o reconhecimento automático de regiões fechadas. É possível salvar os modelos através de um arquivo de extensão JSON, que pode ser lido novamente sempre que

necessário. Múltiplas abas também são suportadas, o que permite a modelagem de mais de um problema de maneira simultânea.

No desenvolvimento da versão inicial do FEMEP, decidiu-se incluir curvas predefinidas para a modelagem dos tipos reta e linha poligonal. Houve a iniciativa de expandir a ferramenta para incorporar novos tipos de curvas paramétricas. A adição de novas curvas poderia ser feita de maneira simples, bastando apenas realizar a devido tratamento na forma de poligonal equivalente. Entretanto, características como a continuidade das curvas seriam perdidas. Em última instância, o programa continuaria a lidar com linhas poligonais, mas com uma maior quantidade de segmentos retilíneos.

Nesse cenário, este trabalho buscou introduzir no FEMEP classes de curvas paramétricas do tipo NURBS, sem a necessidade de simplificações por poligonais equivalentes. As curvas NURBS do modelador são utilizadas para gerar superfícies do tipo Coons em formato NURBS, conforme formulação apontada por Lin [29] e Piegl [30]. Essas superfícies são criadas a partir de quatro curvas de contorno, formando uma região NURBS de malhas estruturadas para AIG. Este trabalho também introduziu no FEMEP a possibilidade de geração de regiões com malhas não estruturadas, utilizando T-Splines com *extraordinary points* baseadas na tese de Scott [17] e no algoritmo de decomposição de domínio de Miranda e Martha [19].

Uma abordagem alternativa para a geração automática de malhas não estruturadas na análise isogeométrica, consiste no algoritmo de Barroso [31, 32], desenvolvido na Universidade Federal do Ceará e na University of Colorado at Boulder, baseado em triângulos racionais Bézier. O algoritmo em questão tem se mostrado robusto em casos de geometrias complexas com múltiplos furos e regiões estreitas, além de apresentar um bom desempenho mesmo em modelos com pouca discretização e elevada curvatura. Embora essa abordagem não tenha sido incorporada neste trabalho, vale sua menção pela contribuição significativa na área.

1.3 Estrutura da Texto

O texto é organizado em oito capítulos. O primeiro capítulo aqui se encerra. Foi apresentado uma introdução à análise isogeométrica e destacados os objetivos

e motivações para a elaboração deste trabalho. Trabalhos correlatos anteriores foram revisados para fornecer um contexto base ao estudo.

O Capítulo 2 traz uma revisão bibliográfica ampla sobre NURBS e B-Splines. São apontados conceitos básicos como vetores de *knots*, funções de base, construção de curvas e superfícies B-Splines e refinamentos. Esses conceitos são posteriormente expandidos para NURBS.

O terceiro capítulo aborda a formulação de análise isogeométrica voltada para problemas de análise linear elástica bidimensional. Definições importantes como os diferentes espaços e mapeamentos na AIG são levantadas. A conectividade também é um tópico relevante tratado no capítulo.

O quarto capítulo trata sobre T-Splines. Inicia-se com os conceitos de T-Mesh e vetores de *knots* locais. Em seguida, prossegue-se para uma abordagem de T-Splines cúbicas, comumente empregadas na análise isogeométrica.

O Capítulo 5 introduz a extração Bézier, uma técnica importante para AIG, que utiliza funções de Bézier e operadores de extração para a determinação das funções de base. A análise por T-Splines se baseia nessa técnica.

O sexto capítulo oferece as principais características e funcionalidades do software de modelagem FEMEP. A arquitetura do programa é exibida, com as principais classes apontadas. São apresentadas funcionalidades como modelagem de curvas NURBS, refinamentos, visualização de propriedades NURBS, interseções e exportação de modelos para análise.

O sétimo capítulo aborda sobre o software de análise FEMOOLab. Aqui são exibidos modelos elaborados no modelador e submetidos a análise isogeométrica. Aspectos computacionais de implementação da AIG sob o contexto de orientação a objetos também são abordados.

O Capítulo 8 encerra o trabalho com as devidas conclusões e considerações finais. Visando o aprimoramento contínuo das ferramentas, são realizadas sugestões de melhorias e extensões futuras.

A redação do texto foi pensada para tornar a leitura didática. Em determinados trechos, são realizadas analogias entre o Método dos Elementos Finitos e a Análise Isogeométrica. Recomenda-se apenas que o leitor tenha

conhecimento básico em elementos finitos. A leitura do segundo e do terceiro capítulos é suficiente para a compreensão de NURBS e análise isogeométrica bidimensional de NURBS. Os dois capítulos subsequentes (quarto e quinto) introduzem os conceitos de T-Splines e extração Bézier. Para leitores com experiência em análise isogeométrica, a estrutura pode não parecer adequada em um primeiro momento, pois a abordagem de T-Splines ocorre após a abordagem de análise. Entretanto, a organização proposta segue um padrão sequencial de acordo com a complexidade dos tópicos, inicialmente NURBS e posteriormente T-Splines.

2 Fundamentos de B-Splines e NURBS

B-Splines e NURBS são classes de funções matemáticas com uma vasta gama de aplicações, sendo utilizadas para representar objetos de maneira precisa. NURBS são uma extensão de B-Splines, que introduzem funções peso, abrindo caminho para a construção de formas mais complexas [33], há uma obra extensa dedicada a esse tópico [30].

Na análise isogeométrica, NURBS são utilizadas como entidades matemáticas de suporte à análise devido a uma série de razões. Elas apresentam propriedades importantes, como a forma de representação paramétrica, e estão integradas em sistemas CAD existentes, o que é particularmente útil na AIG. Isso permite a transferência direta de modelos para a análise numérica, sem a necessidade de uma conversão adicional [4]. Além disso, podem ser facilmente modificadas e refinadas para se adequarem a novas condições de análise ou para melhorar a precisão da geometria. A capacidade de alterar facilmente a geometria, sem a necessidade de gerar um novo modelo para simulação, é um benefício significativo da análise isogeométrica e torna a modelagem e análise de estruturas complexas uma tarefa mais eficiente.

Este capítulo tem como objetivo fornecer um panorama abrangente sobre B-Splines e NURBS no contexto bidimensional. A ideia é fornecer ao leitor conhecimento sobre essas funções matemáticas, compreendendo suas definições, propriedade e aplicações relevantes para análise isogeométrica, suficiente para o embasamento dos capítulos seguintes.

2.1 Representações de Curvas e Superfícies

Curvas e superfícies podem ser representadas pelas formas explícita, implícita e paramétrica, cada uma possui abordagens distintas para descrever entidades geométricas, apresentando vantagens e limitações específicas [34]. No caso particular deste trabalho, o foco é exclusivo na forma paramétrica, entretanto, as demais serão brevemente introduzidas apenas com propósito elucidativo.

A representação explícita se refere à descrição direta por meio de uma fórmula matemática, sendo especialmente útil quando se deseja obter uma expressão simples. Para o caso de curvas e superfícies, respectivamente, a forma explícita assume: $y = f(x)$ e $z = f(x, y)$. É importante destacar que nem todos os objetos geométricos podem ser representados dessa maneira, principalmente em casos mais complexos.

A forma implícita é uma representação mais geral, que permite uma maior flexibilidade na descrição de entidades geométricas em comparação com a representação explícita. Isso ocorre porque a formulação implícita não está limitada a uma função matemática direta. Respectivamente, curvas e superfícies apresentam a seguinte forma implícita: $f(x, y) = 0$ e $f(x, y, z) = 0$. A desvantagem desse tipo de representação está na dificuldade de obtenção de coordenadas de pontos. Enquanto que na forma explícita é relativamente simples determinar pontos de uma curva, na forma implícita esse processo pode ser complicado. A natureza dessa representação implica que podem haver múltiplas soluções ou que nem todas as coordenadas sejam facilmente obtidas. Isso requer o uso de métodos numéricos ou técnicas adicionais para resolver as equações e encontrar os pontos desejados. Em se tratando de superfícies, essa tarefa pode ser ainda mais desafiadora.

Nesse sentido, a forma paramétrica surge como uma boa alternativa. Ao mesmo tempo que permite representar entidades geométricas complexas, a forma paramétrica também possibilita a avaliação simples das coordenadas dos pontos. Nessa representação, as coordenadas são expressas como funções de parâmetros como t, u e v. Cada valor do parâmetro t corresponde a um ponto específico de curva; cada dupla de valores (u, v) a um ponto de superfície. Essa forma torna mais fácil a obtenção das coordenadas desejadas.

A construção de B-Splines e NURBS é baseada no conceito de parametrização. Dessa maneira, uma abordagem inicial sobre curvas e superfícies paramétricas é fundamental para o bom entendimento dessas técnicas de modelagem geométrica.

2.1.1 Curvas Paramétricas

Uma curva paramétrica é uma função vetorial que mapeia um parâmetro para um ponto no espaço n-dimensional [35]. A curva é definida como o conjunto de pontos gerados pela função ao variar o parâmetro. A forma paramétrica de uma curva no R^2 é dada por:

$$\boldsymbol{C}(t) = \big(x(t), y(t)\big) \tag{1}$$

onde t é o parâmetro independente definido em um dado intervalo fechado $[a, b]$, com $a < b$. Normalmente o intervalo é expresso na forma normalizada $[0, 1]$. A título de exemplo, um círculo de raio unitário centrado na origem, como exposto na Figura 4, é descrito pela forma implícita:

$$x^2 + y^2 = 1 \tag{2}$$

De maneira alternativa, o mesmo círculo pode ser expressado em termos da sua forma paramétrica:

$$\boldsymbol{C}(t) = (cos(t), sen(t)) \tag{3}$$

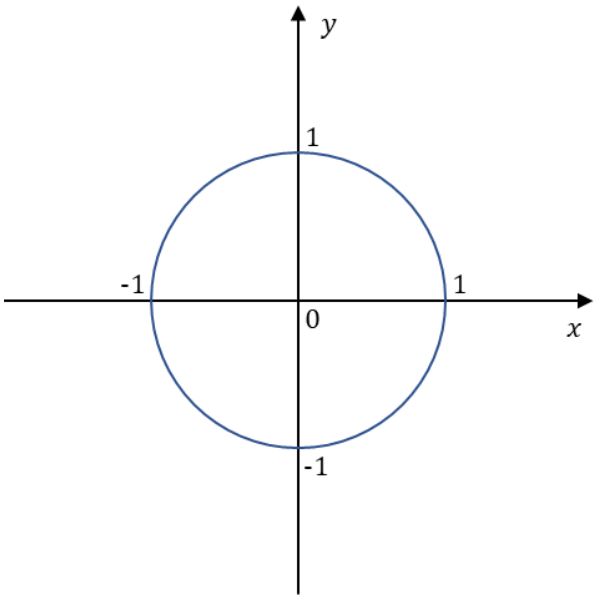

Figura 4: Representação de uma curva no R^2.

2.1.2 Superfícies Paramétricas

Uma superfície paramétrica é representada através do mapeamento de dois parâmetros para um ponto no espaço n-dimensional, assim, ela é dita como biparamétrica [35]. Para o caso bidimensional, a forma paramétrica de uma superfície é dada pela Equação (4).

$$\boldsymbol{S}(u, v) = (x(u, v), y(u, v)) \tag{4}$$

A Figura 5 apresenta uma superfície paramétrica genérica no R^2, definida por parâmetros u e v, cada um contido no intervalo [0, 1]. Pode-se notar que mantendo um dos parâmetros constante e variando o outro, obtém-se uma curva paramétrica. Essa curva representa a trajetória formada por pontos específicos da superfície correspondentes ao parâmetro fixado. A figura fornece exemplos dessas curvas paramétricas. Para ilustração, são consideradas as curvas $u = 0.75$ e $v = 0.75$ na Figura 5. Por outro lado, fixando ambos os parâmetros u e v, obtém-se um único ponto na superfície paramétrica, dado pela interseção das curvas paramétricas referentes à cada parâmetro constante.

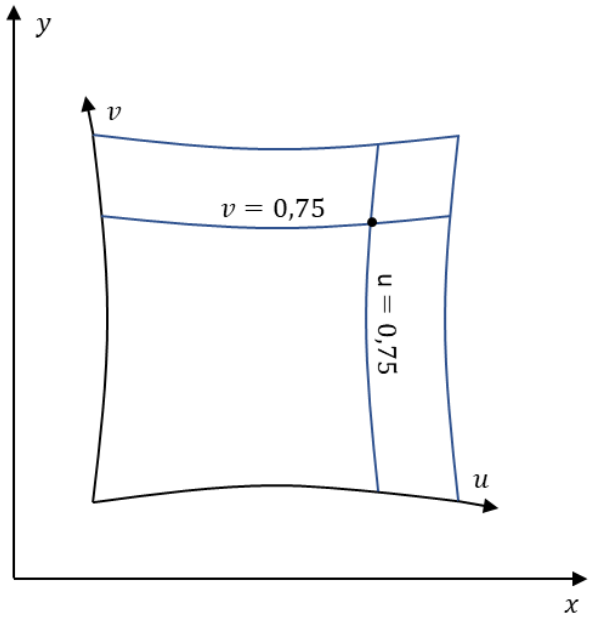

Figura 5: Representação de uma superfície paramétrica genérica no R^2.

Na área da computação gráfica, a representação de entidades por meio da forma paramétrica é extremamente vantajosa, pois é mais simples de ser implementada computacionalmente. Basta variar os parâmetros de uma curva ou

superfície paramétrica para renderizar as geometrias desejadas. Outra vantagem é a capacidade de gerar detalhes refinados em áreas específicas da geometria, ao controlar de forma precisa a variação dos parâmetros. Isso permite a criação de superfícies suaves, curvas complexas e efeitos visuais detalhados.

Tanto as B-splines quanto as NURBS são baseadas em funções paramétricas, que descrevem a posição e a forma das curvas e superfícies de maneira precisa e controlável. A manipulação dos parâmetros dessas funções permite ajustar e personalizar a geometria conforme necessário, proporcionando flexibilidade e versatilidade na modelagem de objetos na computação gráfica. Com sua capacidade de representar formas suaves e precisas, as B-Splines e as NURBS se tornaram fundamentais no campo da modelagem computacional.

2.2 B-Splines

B-Splines podem ser entendidas como as entidades que realizam o mapeamento entre o espaço paramétrico $\widehat{\Omega} \subset \mathbb{R}^{d_p}$ e espaço físico $\Omega \subset \mathbb{R}^{d_s}$. Ao contrário da análise clássica de elementos finitos, o espaço paramétrico das B-Splines abrange vários elementos dentro de um *patch* específico. No Método dos Elementos Finitos, o espaço paramétrico está vinculado a um único elemento, onde cada elemento no espaço paramétrico é mapeado para um elemento no espaço físico. Na análise isogeométrica, esse mapeamento engloba todos os elementos contidos em um *patch*. Como será detalhado mais adiante, o espaço paramétrico de uma B-Spline é formado a partir de um vetor de *knots*, que desempenha um papel crucial na definição dos elementos no contexto da AIG.

2.2.1 Vetor de *Knots*

Para construir uma B-Spline, é necessário inicialmente definir um vetor de *knots*, que consiste em um conjunto de coordenadas paramétricas (*knots*) de valores reais, não negativos e dispostos em ordem crescente. O vetor de *knots* pode ser representado na forma $\Xi = [\xi_1, \xi_2, \dots, \xi_{n+p+1}]$, de tamanho sempre igual a: $n + p + 1$, onde n é o número de funções base, p é o grau polinomial e ξ_i é a coordenada paramétrica de índice $i = 1, 2, \dots, n + p + 1$.

O vetor de *knots* é uniforme se todas as coordenadas paramétricas estiverem igualmente espaçadas, caso contrário, será dito como não uniforme. Os *knots* podem assumir valores repetidos, quando isso acontecer, afirma-se que existe multiplicidade. Um vetor de *knots* é aberto se seu primeiro e último valores possuírem multiplicidade $p + 1$. B-Splines geradas a partir de vetores de *knots* abertos são padrões em sistemas CAD e destacam-se por uma propriedade importante: são interpolatórias nas extremidades. Essa característica é uma grande facilitadora na imposição das condições de contorno. Além disso, essa propriedade é fundamental para problemas envolvendo múltiplos *patches*, uma vez que esses são unidos nas extremidades dos espaços paramétricos. Neste trabalho, será lidado apenas com vetores de *knots* abertos.

2.2.2 Funções de Base B-Splines

A partir de um dado vetor de *knots*, podem ser determinadas funções de base B-Splines associadas, utilizando um processo recursivo. A formulação dessas funções foi originalmente proposta por Cox [36] e de Boor [37]. Para $p = 0$, as funções de base são obtidas através de uma função definida por partes:

$$N_{i,0}(\xi) = \begin{cases} 1 & se\ \xi_i \leq \xi < \xi_{i+1} \\ 0 & caso\ contrário \end{cases} \tag{5}$$

Para ordens superiores $(p = 1, 2, 3, \dots)$, tem-se:

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi) \tag{6}$$

Na Equação (6), define-se que os termos na forma $0/0$ assumem valor nulo. Vale observar que um conjunto de funções base para uma dada ordem p, depende do conjunto de funções referentes a $p - 1$.

As primeiras derivadas das funções base B-Splines são dadas pela Equação (7). Conforme será abordado posteriormente, essas derivadas são essenciais para a montagem da matriz de deformação-deslocamento $\boldsymbol{B}^e$, que por sua vez é necessária para calcular a matriz de rigidez do elemento $\boldsymbol{K}^e$.

$$\frac{d}{d\xi}N_{i,p}(\xi) = \frac{p}{\xi_{i+p} - \xi_i}N_{i,p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}}N_{i+1,p-1}(\xi) \qquad (7)$$

2.2.3 Exemplo: Construindo as Funções de Base

Para demonstrar a construção das funções de base B-Splines pelas Equações (5) e (6), apresenta-se uma exemplificação simples nessa seção, baseada no exemplo mostrado por Cottrel et al. [4], com uma abordagem detalhada. Considere o vetor de *knots* $\Xi = [0, 0, 0, 0.5, 1, 1, 1] = [\xi_1, \xi_2, \xi_3, \xi_4, \xi_5, \xi_6, \xi_7]$ e uma ordem polinomial $p = 2$. É possível observar que se trata de um vetor de *knots* aberto, pois os primeiros e últimos termos se repetem $p + 1$ vezes. O número de termos do vetor de *knots* é 7, dessa forma, a quantidade de funções de base n são determinadas pela relação $7 = n + p + 1$ (ver Seção 2.2.1), obtém-se assim $n = 4$. Desse modo, será demonstrado como encontrar: $N_{1,2}(\xi), N_{2,2}(\xi), N_{3,2}(\xi)$ e $N_{4,2}(\xi)$.

O procedimento para encontrar as funções de base consiste em um processo recursivo. Nesse sentido, é necessário inicialmente calcular as funções de ordem polinomial $p = 0$. Aplicando a Equação (5) para $i = 1$:

$$N_{1,0}(\xi) = \begin{cases} 1 & se\ \xi_1 \leq \xi < \xi_2 \\ 0 & caso\ contrário \end{cases} \qquad (8a)$$

Ao observar o vetor de *knots*, nota-se que $\xi_1 = \xi_2$. Portanto, não há um valor de ξ que satisfaça o termo $\xi_1 \leq \xi < \xi_2$, assim:

$$N_{1,0}(\xi) = 0 \qquad (8a)$$

De forma análoga para $i = 2$:

$$N_{2,0}(\xi) = 0 \qquad (8b)$$

Em seguida, aplicando a Equação (5) para $i = 3$ e $i = 4$:

$$\begin{aligned} N_{3,0}(\xi) &= \begin{cases} 1 & se\ \xi_3 \leq \xi < \xi_4 \\ 0 & caso\ contrário \end{cases} \\ N_{3,0}(\xi) &= \begin{cases} 1 & se\ 0 \leq \xi < 0.5 \\ 0 & caso\ contrário \end{cases} \end{aligned} \qquad (8c)$$

$$N_{4,0}(\xi) = \begin{cases} 1 & se\ \xi_4 \leq \xi < \xi_5 \\ 0 & caso\ contrário \end{cases}$$
$$N_{4,0}(\xi) = \begin{cases} 1 & se\ 0.5 \leq \xi < 1 \\ 0 & caso\ contrário \end{cases} \quad (8d)$$

Para $i = 5$ e $i = 6$, o processo é análogo à $i = 1$ e $i = 2$:

$$N_{5,0}(\xi) = 0 \quad (8e)$$

$$N_{6,0}(\xi) = 0 \quad (8f)$$

Na Figura 6 estão representadas graficamente as funções de base B-Splines constantes ($p = 0$).

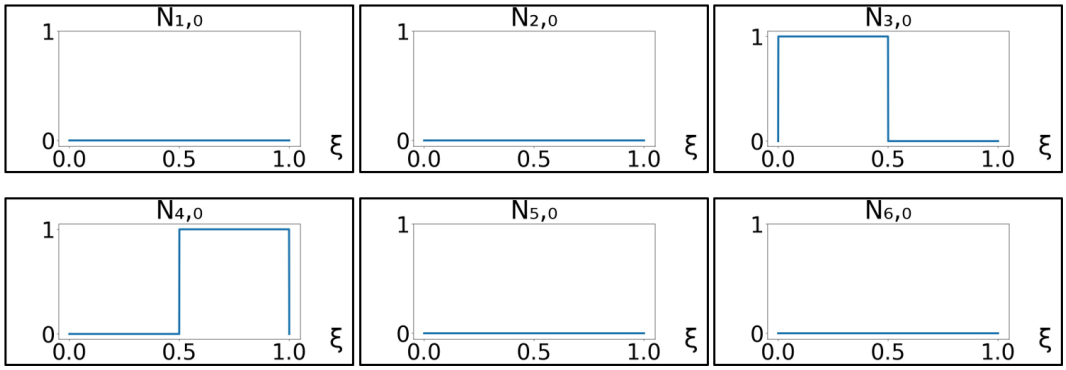

Figura 6: Funções de base constantes do vetor de *knots* $\Xi = [0, 0, 0, 0.5, 1, 1, 1]$.

Pode-se agora prosseguir com a determinação das funções de base para $p = 1$. Empregando a Equação (6):

$$N_{1,1}(\xi) = \frac{\xi - \xi_1}{\xi_2 - \xi_1} N_{1,0}(\xi) + \frac{\xi_3 - \xi}{\xi_3 - \xi_2} N_{2,0}(\xi)$$
$$N_{1,1}(\xi) = \frac{\xi - 0}{0 - 0} \cdot 0 + \frac{0 - \xi}{0 - 0} \cdot 0 = 0 \quad (9a)$$

$$N_{2,1}(\xi) = \frac{\xi - \xi_2}{\xi_3 - \xi_2} N_{2,0}(\xi) + \frac{\xi_4 - \xi}{\xi_4 - \xi_3} N_{3,0}(\xi)$$
$$N_{2,1}(\xi) = \frac{\xi - 0}{0 - 0} \cdot 0 + \frac{0.5 - \xi}{0.5 - 0} N_{3,0}(\xi) \quad (9b)$$
$$N_{2,1}(\xi) = \begin{cases} -2\xi + 1 & se\ 0 \leq \xi < 0.5 \\ 0 & caso\ contrário \end{cases}$$

$$N_{3,1}(\xi) = \frac{\xi - \xi_3}{\xi_4 - \xi_3} N_{3,0}(\xi) + \frac{\xi_5 - \xi}{\xi_5 - \xi_4} N_{4,0}(\xi)$$

$$N_{3,1}(\xi) = \frac{\xi - 0}{0.5 - 0} N_{3,0}(\xi) + \frac{1 - \xi}{1 - 0.5} N_{4,0}(\xi)$$

$$N_{3,1}(\xi) = \begin{cases} 2\xi & se\ 0 \le \xi < 0.5 \\ 0 & caso\ contrário \end{cases} + \begin{cases} -2\xi + 2 & se\ 0.5 \le \xi < 1 \\ 0 & caso\ contrário \end{cases} \quad (9c)$$

$$N_{3,1}(\xi) = \begin{cases} 2\xi & se\ 0 \le \xi < 0.5 \\ -2\xi + 2 & se\ 0.5 \le \xi < 1 \\ 0 & caso\ contrário \end{cases}$$

$$N_{4,1}(\xi) = \frac{\xi - \xi_4}{\xi_5 - \xi_4} N_{4,0}(\xi) + \frac{\xi_6 - \xi}{\xi_6 - \xi_5} N_{5,0}(\xi)$$

$$N_{4,1}(\xi) = \frac{\xi - 0.5}{1 - 0.5} N_{4,0}(\xi) + \frac{1 - \xi}{1 - 1} \cdot 0 \quad (9d)$$

$$N_{4,1}(\xi) = \begin{cases} 2\xi - 1 & se\ 0.5 \le \xi < 1 \\ 0 & caso\ contrário \end{cases}$$

$$N_{5,1}(\xi) = \frac{\xi - \xi_5}{\xi_6 - \xi_5} N_{5,0}(\xi) + \frac{\xi_7 - \xi}{\xi_7 - \xi_6} N_{6,0}(\xi)$$

$$N_{5,1}(\xi) = \frac{\xi - 1}{1 - 1} \cdot 0 + \frac{1 - \xi}{1 - 1} \cdot 0 = 0 \quad (9e)$$

Na Figura 7 são visualizados os gráficos das funções de base B-Splines lineares.

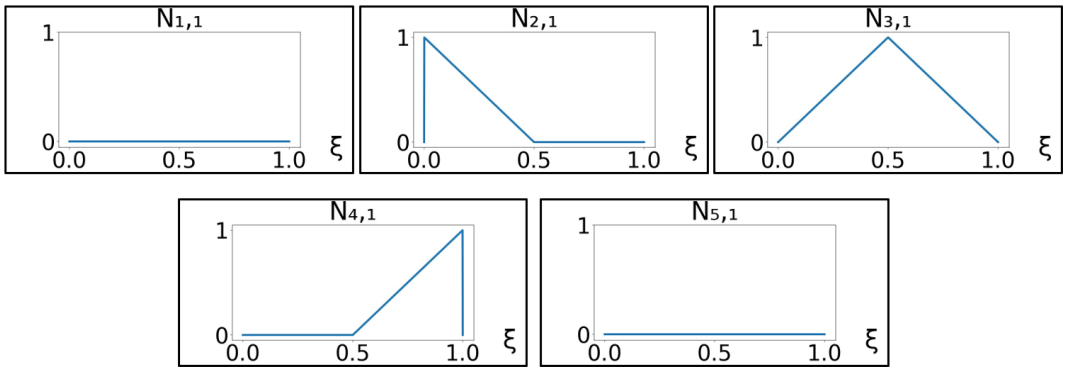

Figura 7: Funções de base lineares do vetor de *knots* $\Xi = [0, 0, 0, 0.5, 1, 1, 1]$.

Por fim, determina-se as funções de base B-Splines para $p = 2$, ilustradas na Figura 8.

$$N_{1,2}(\xi) = \frac{\xi - \xi_1}{\xi_3 - \xi_2} N_{1,1}(\xi) + \frac{\xi_4 - \xi}{\xi_4 - \xi_2} N_{2,1}(\xi)$$

$$N_{1,2}(\xi) = \frac{\xi - 0}{0 - 0} \cdot 0 + \frac{0.5 - \xi}{0.5 - 0} N_{2,1}(\xi) \tag{10a}$$

$$N_{1,2}(\xi) = \begin{cases} 4\xi^2 - 4\xi + 1 & se\ 0 \leq \xi < 0.5 \\ 0 & caso\ contrário \end{cases}$$

$$N_{2,2}(\xi) = \frac{\xi - \xi_2}{\xi_4 - \xi_2} N_{2,1}(\xi) + \frac{\xi_5 - \xi}{\xi_5 - \xi_3} N_{3,1}(\xi)$$

$$N_{2,2}(\xi) = \frac{\xi - 0}{0.5 - 0} N_{2,1}(\xi) + \frac{1 - \xi}{1 - 0} N_{3,1}(\xi) \tag{10b}$$

$$N_{2,2}(\xi) = \begin{cases} -6\xi^2 + 4\xi & se\ 0 \leq \xi < 0.5 \\ 2\xi^2 - 4\xi + 2 & se\ 0.5 \leq \xi < 1 \\ 0 & caso\ contrário \end{cases}$$

$$N_{3,2}(\xi) = \frac{\xi - \xi_3}{\xi_5 - \xi_3} N_{3,1}(\xi) + \frac{\xi_6 - \xi}{\xi_6 - \xi_4} N_{4,1}(\xi)$$

$$N_{3,2}(\xi) = \frac{\xi - 0}{1 - 0} N_{3,1}(\xi) + \frac{1 - \xi}{1 - 0.5} N_{4,1}(\xi) \tag{10c}$$

$$N_{3,2}(\xi) = \begin{cases} 2\xi^2 & se\ 0 \leq \xi < 0.5 \\ -6\xi^2 + 8\xi - 2 & se\ 0.5 \leq \xi < 1 \\ 0 & caso\ contrário \end{cases}$$

$$N_{4,2}(\xi) = \frac{\xi - \xi_4}{\xi_6 - \xi_4} N_{4,1}(\xi) + \frac{\xi_7 - \xi}{\xi_7 - \xi_5} N_{5,1}(\xi)$$

$$N_{4,2}(\xi) = \frac{\xi - 0.5}{1 - 0.5} N_{4,1}(\xi) + \frac{1 - \xi}{1 - 1} \cdot 0 \tag{10d}$$

$$N_{4,2}(\xi) = \begin{cases} 4\xi^2 - 4\xi + 1 & se\ 0.5 \leq \xi < 1 \\ 0 & caso\ contrário \end{cases}$$

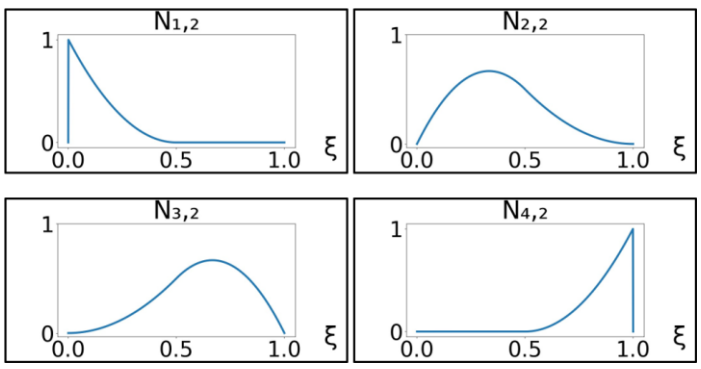

Figura 8: Funções de base quadráticas do vetor de *knots* $\Xi = [0, 0, 0, 0.5, 1, 1, 1]$.

2.2.4 Propriedades das Funções de Base

Algumas propriedades relevantes das funções base B-Splines podem ser destacadas:

i. Constituem uma partição de unidade: $\sum_{i=1}^{n} N_{i,p}(\xi) = 1$.

ii. São não negativas ao longo do espaço paramétrico.

iii. São linearmente independentes: $\sum_{i=1}^{n} \alpha_i N_{i,p}(\xi) = 0 \Leftrightarrow \alpha_i = 0$, $i = 1, 2, \dots, n$.

iv. Possuem continuidade C^{p-m_i} em um dado *knot* ξ_i, onde m_i é a multiplicidade no *knot* de índice i.

v. Uma dada função base $N_{i,p}$ tem o suporte de $p+1$ *knot spans*. Um *knot span* representa o intervalor paramétrico entre *knots*. Dessa forma, o suporte abrange $[\xi_i, \xi_{i+p+1}]$.

As propriedades iv. e v. podem parecer mais complexas à primeira vista. De forma a facilitar seu entendimento, essas propriedades serão abordadas de maneira didática no exemplo fornecido a seguir. Considere o vetor de *knots* aberto $\Xi = [0, 0, 0, 0.2, 0.4, 0.6, 0.6, 0.8, 1, 1, 1] = [\xi_1, \xi_2, \xi_3, \xi_4, \xi_5, \xi_6, \xi_7, \xi_8, \xi_9, \xi_{10}, \xi_{11}]$ e uma ordem polinomial $p = 2$. As funções de base B-Splines para esse caso estão mostradas na Figura 9.

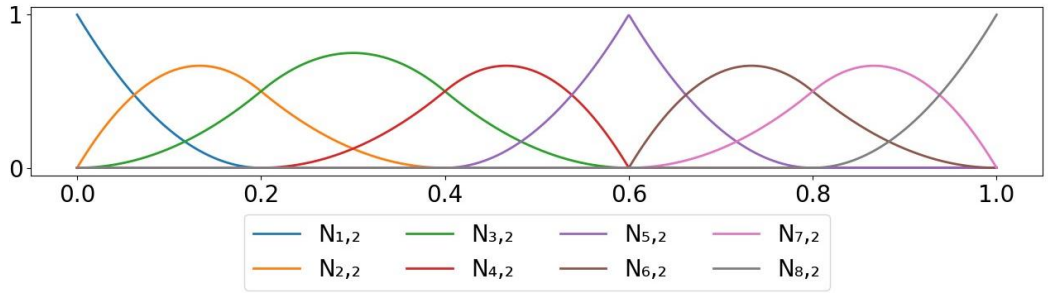

Figura 9: Funções de base B-Splines quadráticas do vetor de *knots* $\Xi = [0, 0, 0, 0.2, 0.4, 0.6, 0.6, 0.8, 1, 1, 1]$.

O efeito da propriedade iv., que trata da continuidade, torna-se mais evidente no *knot* $\xi = 0.6$ e nas extremidades do domínio paramétrico. Verifica-se nessas localidades a natureza interpolatória das funções. O *knot* $\xi = 0.6$ se repete duas vezes no vetor de *knots*, apresentando multiplicidade $m_i = 2$. Portanto, nesse ponto, a continuidade é dada por $C^{p-m_i} = C^{2-2} = C^0$. Já nas extremidades, devido

ao vetor de *knots* ser aberto, a continuidade é $C^{p-m_i} = C^{p-(p+1)} = C^{-1}$, indicando que as bases estão completamente descontínuas, marcando assim o término do domínio. Nos demais *knots* as funções são $C^{p-m_i} = C^{2-1} = C^{1}$.

A propriedade v., relacionada ao suporte, pode ser observada em cada função de base. Todas essas funções permanecem maior que zero em $p+1$ *knot spans*. Em um primeiro momento, pode parecer que uma maior multiplicidade implica numa diminuição do suporte das funções. Basta observar a função $N_{5,2}$, que possui suporte em $[0.4, 0.6]$, enquanto que $N_{3,2}$, por exemplo, tem suporte [0.0, 0.6]. No entanto, é importante notar que o suporte continua a iniciar em ξ_i e a terminar em ξ_{i+p+1}. O que ocorre é que podem existir *spans* com comprimento paramétrico nulo devido à repetição de certos *knots*.

Na análise isogeométrica, os elementos não são delimitados por todos os *spans,* pois isso acarretaria em casos com elementos definidos em *spans* nulos. Assim, acaba que elementos correspondem a *knot spans* não nulos.

2.2.5 Curvas B-Splines

Uma curva B-Spline é construída através de uma combinação linear das funções de base B-Splines $\{N_{i,p}(\xi)\}_{i=1}^{n}$, onde os coeficientes são definidos como os pontos de controle $\{\boldsymbol{P}_i\}_{i=1}^{n}$. Sendo $\boldsymbol{P}_i \in \mathbb{R}^2$ para uma curva contida no espaço bidimensional. Dado o fato de que uma curva B-Spline consiste em uma combinação linear, a quantidade de pontos de controle será a mesma quantidade de funções de base. A expressão da curva B-Spline é dada por:

$$\boldsymbol{C}(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi)\boldsymbol{P}_i \tag{11}$$

Na Figura 10, ilustra-se o exemplo de uma curva quadrática B-Spline formada a partir das funções de base da Figura 9. Destacados em vermelho estão os *knots*, que são os pontos na curva correspondentes às coordenadas paramétricas do vetor de *knots*. Representado em azul, tem-se o polígono de controle, definido como o segmento que conecta os pontos de controle da curva.

Pode-se notar como a propriedade de continuidade das funções de base (ver Seção 2.2.4) reflete na continuidade da curva da Figura 10. Os pontos de controle no início e no final da curva apresentam natureza interpolatória, pois o vetor de *knots* é aberto, que implica em continuidade C^{-1} nos *knots* $\xi = 0$ e $\xi = 1$. Essa B-Spline também apresenta um ponto de controle com efeito interpolatório no *knot* $\xi = 0.6$, que ocorre em decorrência da continuidade C^0 nessa localidade.

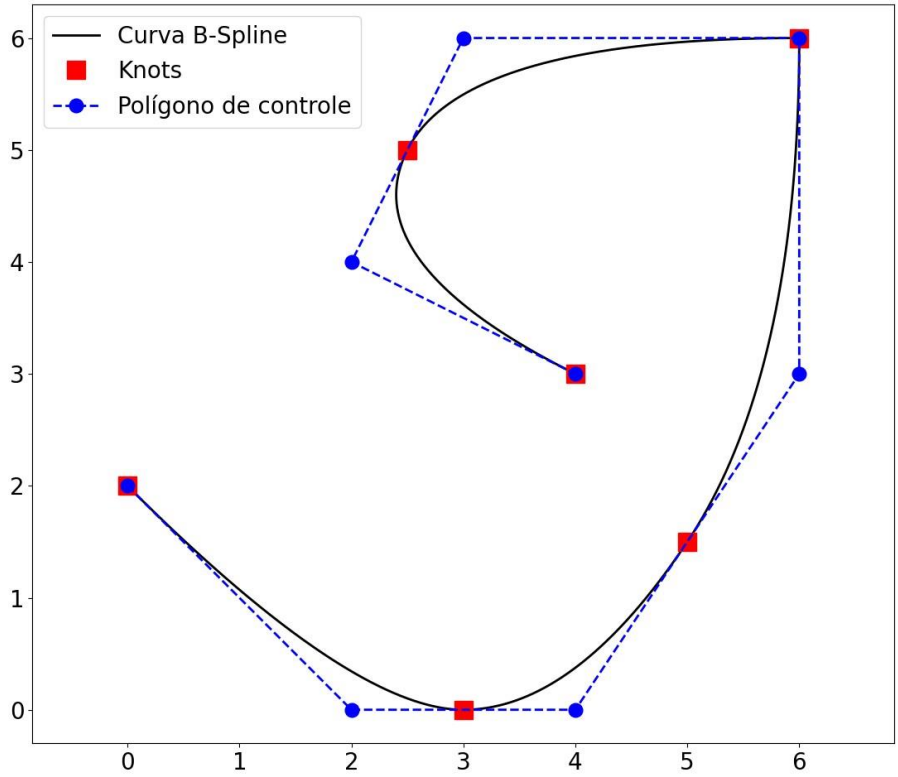

Figura 10: Curva quadrática B-Spline construída a partir das funções de base do vetor de *knots* $\Xi = [0, 0, 0, 0.2, 0.4, 0.6, 0.6, 0.8, 1, 1, 1]$.

2.2.6 Superfícies B-Splines

Uma superfície de produto tensorial B-Spline é formada a partir de dois conjuntos de funções de base: $\{N_{i,p}(\xi)\}_{i=1}^{n}$ para a direção paramétrica ξ, bem como $\{M_{j,q}(\eta)\}_{j=1}^{m}$ para a direção η; juntamente com uma série de pontos $\boldsymbol{P}_{i,j}$. A expressão para a superfície B-Spline é definida por:

$$\boldsymbol{S}(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m} N_{i,p}(\xi) M_{j,q}(\eta) \boldsymbol{P}_{i,j} \tag{12}$$

onde $\{N_{i,p}(\xi)\}_{i=1}^{n}$ e $\{M_{j,q}(\eta)\}_{j=1}^{m}$ representam as funções de base B-Spline univariadas de ordem p e q, correspondentes aos vetores de *knots* $\Xi = [\xi_1, \xi_2, \dots, \xi_{n+p+1}]$ e $\mathcal{H} = [\eta_1, \eta_2, \dots, \eta_{m+q+1}]$, respectivamente. Se a superfície for bidimensional, então $\boldsymbol{P}_{i,j} \in \mathbb{R}^2$.

Pode-se escrever a Equação (12) de forma mais compacta através de um índice global: $A = n(j-1) + i$ com $A = 1, 2, \dots, N$. Sendo $N = n\,.\,m$. Assim:

$$\boldsymbol{S}(\xi,\eta) = \sum_{A=1}^{N} N_A^{p,q}(\xi,\eta) \boldsymbol{P}_A \tag{13}$$

onde $\{N_A^{p,q}(\xi,\eta)\}_{A=1}^{N}$ são as funções de base B-Spline bivariadas dadas por: $N_A^{p,q}(\xi,\eta) = N_{i,p}(\xi) M_{j,q}(\eta)$.

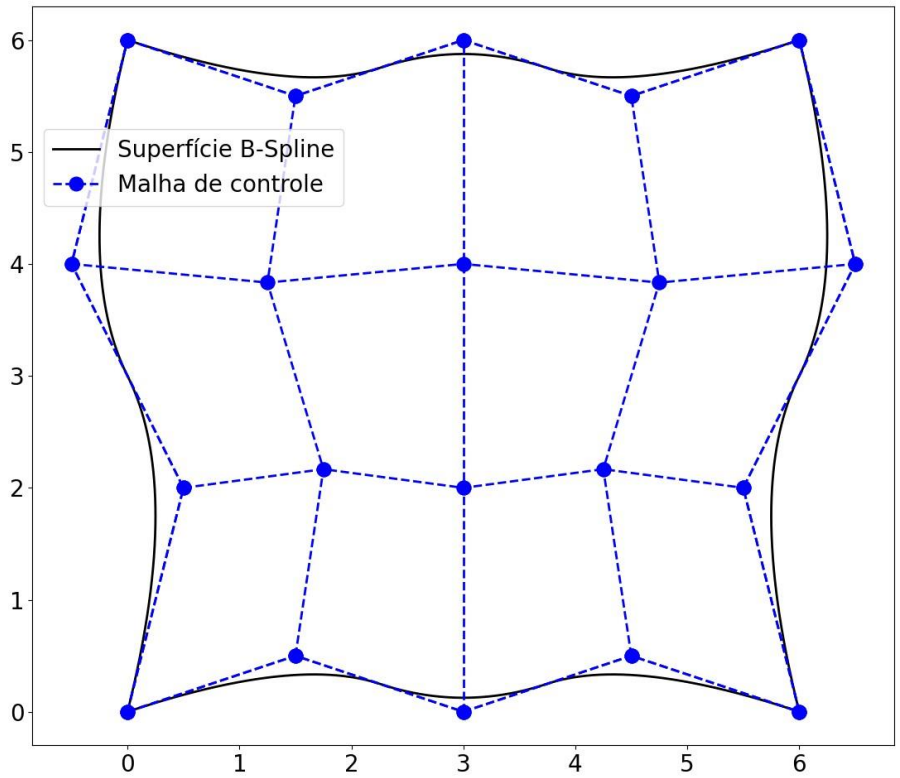

Figura 11: Superfície biquadrática B-Spline formada pelos vetores de *knots*: $\Xi = [0, 0, 0, 0.333, 0.667, 1, 1, 1]$ e $\mathcal{H} = [0, 0, 0, 0.5, 1, 1, 1]$.

O suporte de uma função bivariada $N_A^{p,q}(\xi,\eta)$ se estende pela área paramétrica $[\xi_i,\xi_{i+p+1}] \otimes [\eta_j,\eta_{j+q+1}]$, indicando que essa função será nula em qualquer região fora desse intervalo.

A Figura 11 mostra o exemplo de uma superfície biquadrática B-Spline originada dos vetores de *knots* $\Xi = [0,0,0,0.333,0.667,1,1,1]$ e $\mathcal{H} = [0,0,0,0.5,1,1,1]$. Em azul está representada a malha de controle, constituída pelos segmentos que ligam os pontos de controle da superfície.

2.2.7 Anchors

Anchor ou ponto de ancoragem é uma entidade abstrata, útil para facilitar o entendimento de T-Splines, que serão abordadas posteriormente neste trabalho. Para o caso de vetores de *knots* abertos, o número de *knots* é dado pela relação: $n + p + 1$, onde n é o número de funções de base (ver Seção 2.2.1). É interessante notar que os *knots* não possuem correspondência de um para um com as funções de base. Nesse sentido, é conveniente associar uma localização no espaço paramétrico (no intervalo do vetor de *knots*) para cada função de base. Essas localizações paramétricas recebem o nome de *anchors* $\{s_i\}_{i=1}^{n}$, que são definidos por:

$$s_i = \begin{cases} \xi_{i+(p+1)/2} & se\ p\ for\ ímpar \\ \frac{1}{2}\left(\xi_{i+(p/2)} + \xi_{i+(p/2)+1}\right) & se\ p\ for\ par \end{cases} \qquad (14)$$

De forma simplificada, se o grau polinomial p for ímpar, os *anchors* serão localizados nos *knots*; caso p seja par, os *anchors* estarão no centro dos *knot spans*. O número de *anchors* sempre será igual a quantidade de pontos de controle, o que torna mais intuitivo relacionar esses pontos com uma posição do espaço paramétrico.

2.2.8 Refinamentos

Uma funcionalidade desejada para modelagem geométrica e análise é a capacidade de refinamento, especialmente quando se busca obter resultados mais precisos. Nesse sentido, existem alguns tipos de algoritmos com a finalidade de

refinamento de B-Splines, que realizam modificações nos vetores de *knots* e pontos de controle, mantendo a geometria original. Piegl e Tiller [30] oferecem uma análise aprofundada sobre esse assunto na obra *The NURBS Book*. Para tornar mais acessível o entendimento dessas técnicas, são estabelecidas correlações com refinamentos empregados no método tradicional de elementos finitos, oferecendo uma visão mais abrangente dessas estratégias.

2.2.8.1 Elevação de Grau

Esse tipo de refinamento é realizado por meio da elevação de ordem polinomial, sendo análogo ao refinamento p em MEF. A título comparativo, em elementos finitos se for optado por substituir uma malha de elementos Q4 por uma malha de elementos Q8, por exemplo, o número de elementos do modelo permanece o mesmo, mas a quantidade de nós por elemento é aumentada. Nessa situação, as novas funções de forma não se mantêm na configuração linear (para Q4), adotando em vez disso a forma quadrática (para Q8). Em IGA, o refinamento por elevação de ordem mantém o número de elementos e eleva o número de pontos de controle.

Formulações foram desenvolvidas com o intuito de elevar o grau de curvas B-Splines [38, 39]. Piegl e Tiller [40] conduzem um estudo comparativo dessas formulações, juntamente com uma proposta apresentada por eles, que posteriormente foi incorporada na obra *The NURBS Book* [30]. É essa abordagem que será explorada neste trabalho. Basicamente ela pode ser dividida nas seguintes etapas:

- Decompor a curva B-Spline em curvas Bézier.
- Realizar a elevação de grau em cada Bézier.
- Remover os *knots* desnecessários.

Uma curva B-Spline pode ser representada como uma cadeia de curvas Bézier, unidas de ponta a ponta, onde cada Bézier corresponde a um elemento da B-Spline. Assim, para realizar a primeira etapa, de decomposição de uma B-Spline em várias Béziers, basta aplicar a inserção de *knots* (ver Seção 2.2.8.2) até que todos os *knots* internos tenham multiplicidade $m = p$, resultando em continuidade C^0 nessas localidades.

Posteriormente, os pontos de controle de cada Bézier são extraídos. Cada Bézier possui $p+1$ pontos de controle, de tal forma que o último ponto de uma determinada Bézier coincide com o primeiro ponto da próxima Bézier adjacente. A elevação de grau das curvas Bézier é então realizada. Esse procedimento consiste em um processo direto, que determina novos pontos de controle Bézier $\left\{\overline{\boldsymbol{P}}_i^{\boldsymbol{b}}\right\}_{i=1}^{m}$ a partir dos pontos de controle da Bézier inicial $\left\{\boldsymbol{P}_i^{\boldsymbol{b}}\right\}_{i=1}^{n}$, com $m = n+1$:

$$\overline{\boldsymbol{P}}_i^{\boldsymbol{b}} = \begin{Bmatrix} \boldsymbol{P}_1^{\boldsymbol{b}} & se\ i = 1 \\ \left(\frac{i-1}{p+1}\right)\boldsymbol{P}_{i-1} + \left(1 - \frac{i-1}{n+1}\right)\boldsymbol{P}_i & se\ 1 < i < n+1 \\ \boldsymbol{P}_n^{\boldsymbol{b}} & se\ i = n+1 \end{Bmatrix} \tag{15}$$

Com a elevação de grau executada, os novos pontos de controle das Béziers podem ser reagrupados para definir a B-Spline original, com o cuidado de eliminar os pontos repetidos. O novo vetor de *knots* da B-Spline apresenta todos os *knots* com multiplicidade elevada em uma unidade, devido a elevação de grau. O próximo passo é aplicar a remoção de *knots* nessa nova configuração. São removidos os *knots* inicialmente inseridos para a obtenção das Béziers. O processo de remover um *knot* ξ_k consiste em determinar uma nova série de pontos de controle conforme as Equações:

$$\overline{\boldsymbol{P}}_i = \begin{cases} \dfrac{\boldsymbol{P}_i - (1-\alpha_i)\overline{\boldsymbol{P}}_{i-1}}{\alpha_i} & k - p \le i \le \dfrac{2k - p - m_k - 1}{2} \\ \dfrac{\boldsymbol{P}_i - \alpha_i \overline{\boldsymbol{P}}_{i+1}}{1-\alpha_i} & \dfrac{2k - p - m_k + 2}{2} \le i \le k - m_k \end{cases} \tag{16}$$

$$\alpha_i = \frac{u - u_i}{u_{i+p+1} - u_i} \tag{17}$$

A Figura 12 demonstra um exemplo de refinamento por elevação de ordem. Na Figura 12a, são apresentadas as funções de base $\left\{N_{i,p}(\xi)\right\}_{i=1}^{n}$, relativas ao vetor de *knots* $\Xi = [0,0,0,1,1,1]$ e $p = 2$. O processo de elevação de ordem resulta nas funções de base $\left\{\overline{N}_{i,q}(\xi)\right\}_{i=1}^{m}$ com $q = 3$, referentes ao novo vetor de *knots* $\overline{\overline{\Xi}} = [0,0,0,0,1,1,1,1]$, que possui termos replicados de Ξ. Uma B-Spline construída

pelas funções $\{N_{i,p}(\xi)\}_{i=1}^{n}$ e por um conjunto de pontos de controle $\{\boldsymbol{P}_i\}_{i=1}^{n}$ é mostrada na Figura 12c. Aplicando o refinamento de elevação de grau, são obtidos os pontos de controle $\{\overline{\boldsymbol{P}}_i\}_{i=1}^{m}$, que, em combinação linear com as funções $\{\overline{N}_{i,q}(\xi)\}_{i=1}^{m}$, formam a curva B-Spline da Figura 12d.

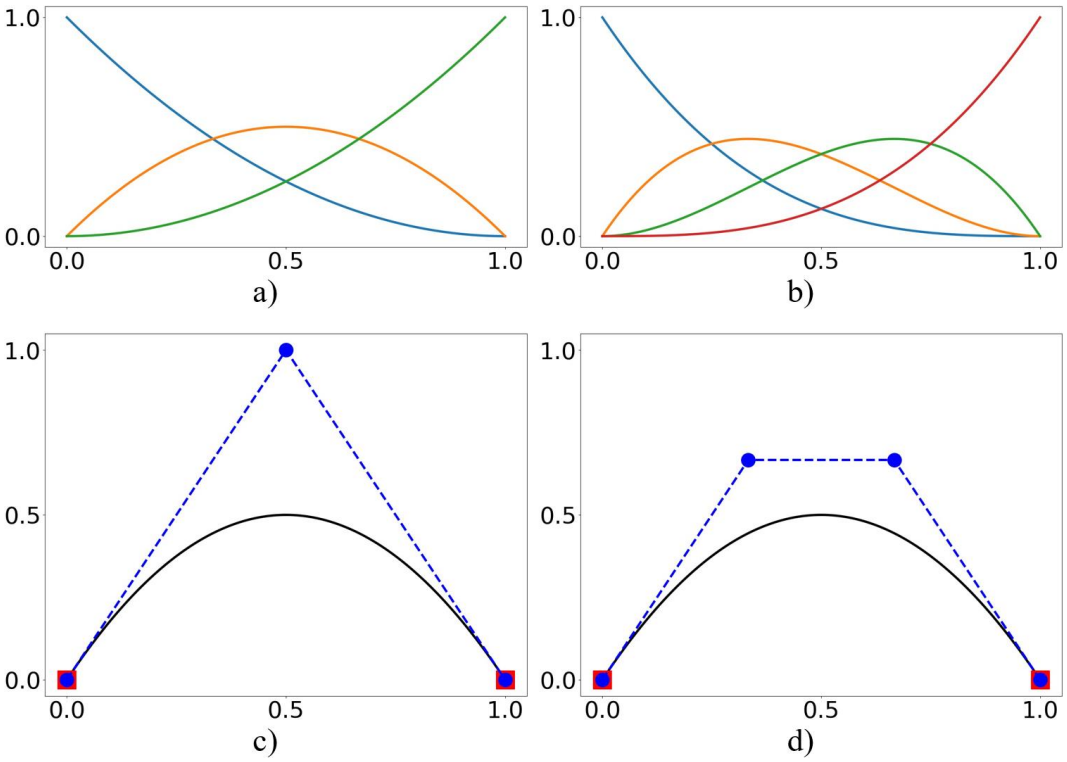

Figura 12: Refinamento por elevação de ordem. a) Funções de base quadráticas de $\Xi = [0, 0, 0, 1, 1, 1]$. b) Novas funções cúbicas referentes à $\overline{\Xi} = [0, 0, 0, 0, 1, 1, 1, 1]$. c) B-Spline formada a partir de Ξ. d) B-Spline após a elevação de ordem.

2.2.8.2 Inserção de *Knots*

Um outro tipo de refinamento consiste no processo de inserção de *knots*, que envolve a divisão de elementos existentes em elementos menores, apresentando similaridade com o refinamento h em MEF. Conforme abordado na Seção 2.2.4, cada elemento é definido entre *knots* de valores distintos e consecutivos, portanto, inserir novos valores de *knots* implica em aumentar a quantidade de elementos. Para realizar essa modificação sem alterar a geometria original de uma B-Spline, é necessário calcular um novo conjunto de pontos de controle.

Dado um vetor de *knots* $\Xi = [\xi_1, \dots, \xi_k, \xi_{k+1}, \dots \xi_{n+p+1}]$, deseja-se inserir um *knot* $\bar{\xi} \in [\xi_k, \xi_{k+1})$ de forma a obter um novo vetor de *knots* $\bar{\Xi} = [\bar{\xi}_1, \dots, \bar{\xi}_k, \bar{\xi}, \bar{\xi}_{k+2}, \dots \bar{\xi}_{m+p+1}]$ com $m = n + 1$. As funções de base $\{N_{i,p}(\xi)\}_{i=1}^{n}$ são redefinidas, mantendo a mesma ordem polinomial p. Um novo conjunto de funções $\{\bar{N}_{i,p}(\xi)\}_{i=1}^{m}$ é calculado através das Equações (5) e (6) utilizando $\bar{\Xi}$. Em seguida, determinam-se os novos pontos de controle $\{\bar{\boldsymbol{P}}_i\}_{i=1}^{m}$ em função dos pontos de controle antigos $\{\boldsymbol{P}_i\}_{i=1}^{n}$:

$$\bar{\boldsymbol{P}}_i = \begin{cases} \boldsymbol{P}_1 & i = 1 \\ \alpha_i \boldsymbol{P}_i + (1 - \alpha_i)\boldsymbol{P}_{i-1} & 1 < i < m \\ \boldsymbol{P}_n & i = m \end{cases} \tag{18}$$

onde os coeficientes α_i são dados por:

$$\alpha_i = \begin{cases} 1 & i \leq k - p \\ \dfrac{\bar{\xi} - \xi_i}{\xi_{i+p} - \xi_i} & k - p + 1 \leq i \leq k \\ 0 & i \geq k + 1 \end{cases} \tag{19}$$

Um exemplo de refinamento por inserção de *knots* é apresentado na Figura 13. Inicialmente são mostradas as funções de base $\{N_{i,p}(\xi)\}_{i=1}^{n}$ do vetor de *knots* $\Xi = [0, 0, 0, 0, 1, 1, 1, 1]$ com $p = 3$ na Figura 13a. Em seguida, é performado a inserção do *knot* $\bar{\xi} = 0.5$, resultando nas novas funções de base $\{\bar{N}_{i,p}(\xi)\}_{i=1}^{m}$ referentes ao novo vetor de *knots* $\bar{\Xi} = [0, 0, 0, 0, 0.5, 1, 1, 1, 1]$, ilustradas na Figura 13b. Simultaneamente, observa-se uma B-Spline com pontos de controle $\{\boldsymbol{P}_i\}_{i=1}^{n}$, construída a partir de Ξ, como apresentado na Figura 13c. Por fim, é apresentado a B-Spline da Figura 13d, que possui uma nova disposição de pontos de controle $\{\bar{\boldsymbol{P}}_i\}_{i=1}^{m}$, calculados pelas Equações (18) e (19). É interessante ressaltar que na análise isogeométrica, os elementos estão delimitados por *knots* de diferentes valores, representados pelos quadriláteros vermelhos. Percebe-se que na Figura 13c existe apenas um elemento limitado pelos *knots* $\xi = 0$ e $\xi = 1$. Na Figura 13d, um elemento adicional é criado devido ao novo *knot* $\bar{\xi} = 0.5$.

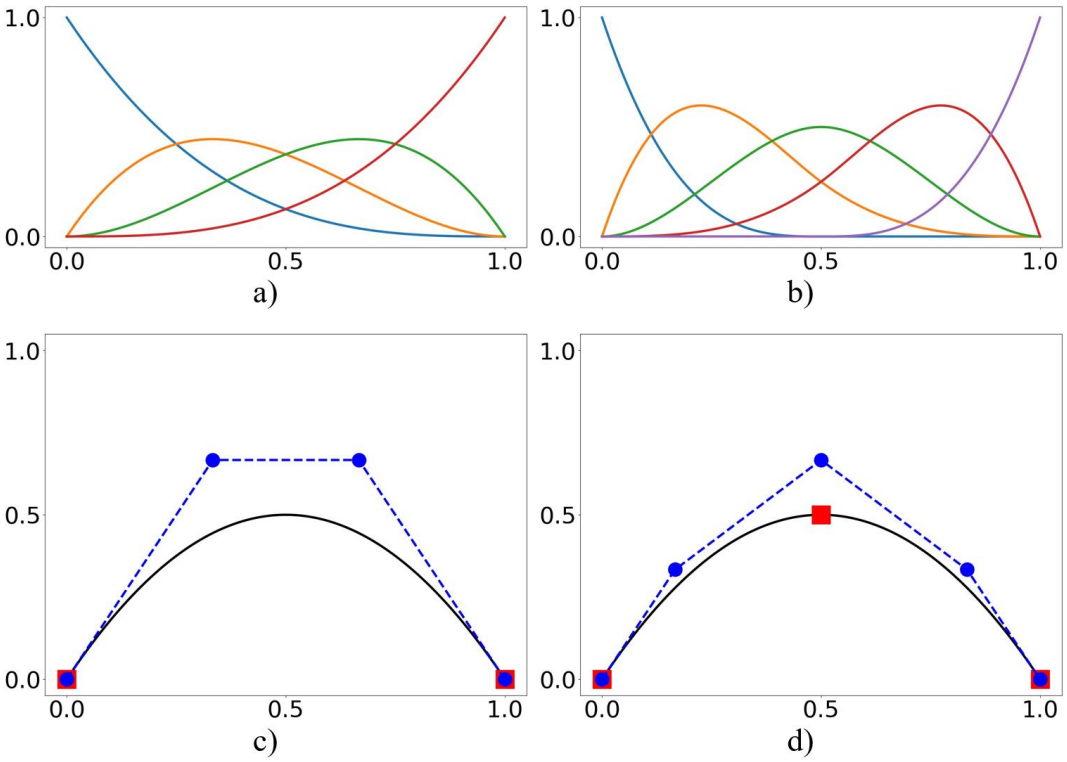

Figura 13: Refinamento por inserção de *knots*. a) Funções de base de $\Xi = [0, 0, 0, 0, 1, 1, 1, 1]$. b) Novas funções referentes à $\bar{\Xi} = [0, 0, 0, 0, 0.5, 1, 1, 1, 1]$. c) B-Spline formada a partir de Ξ; d) B-Spline após a inserção de $\bar{\xi} = 0.5$.

2.2.8.3 Refinamento k

O refinamento k é direcionado apenas para AIG, não apresentando analogia com refinamentos em MEF. A ideia é realizar uma combinação sequencial: inicialmente, a elevação de grau e, posteriormente, a inserção de *knots*. A manutenção dessa sequência é fundamental para o resultado final.

Se um determinado valor único de *knot* $\bar{\xi}$ é inserido em uma curva de ordem polinomial p, a continuidade nesse *knot* é dada por $C^{p-m} = C^{p-1}$. Em seguida, supondo que é realizado o refinamento de elevação de grau, a multiplicidade nos *knots* internos é aumentada em uma unidade. Com isso, a continuidade em $\bar{\xi}$ se mantém a mesma $C^{(p+1)-(1+1)} = C^{p-1}$.

Se a sequência do refinamento k for mantida, primeiro realizando a elevação de grau para depois realizar a inserção de *knots*, seria evitado aumento da multiplicidade de $\bar{\xi}$, resultando em uma maior continuidade. A implicação de manter a continuidade mais alta nos *knots* é a redução no número de pontos de

controle necessários para representar a curva. Isso, por sua vez, acarreta em menos graus de liberdade na análise, já que em AIG os graus de liberdade são atribuídos aos pontos de controle.

2.3 Non-Uniform Rational B-Splines (NURBS)

Embora B-Splines e NURBS sejam muitas vezes mencionadas em conjunto, é importante destacar que existe uma diferença significativa entre elas. As B-Splines são entidades matemáticas usadas para representar curvas, superfícies e sólidos, são definidas como uma combinação linear entre funções de base polinomiais e pontos de controle. As NURBS, por sua vez, são uma extensão das B-Splines, que permitem a associação de pesos em cada ponto de controle. Esses pesos são utilizados para definir funções de base racionais. As NURBS permitem a construção exata de formas cônicas, como círculos e elipses [4, 33], abrindo portas para uma maior possibilidade de representações geométricas na etapa de modelagem.

2.3.1 NURBS em uma Perspectiva Geométrica

Para compreender NURBS, a perspectiva geométrica de sua definição pode ser um bom ponto de partida, pois a visualização pode ajudar a elucidar conceitos complexos.

Uma entidade NURBS contida em um espaço $\mathbb{R}^{d_s}$ pode ser obtida pela projeção de uma B-Spline contida em $\mathbb{R}^{d_s+1}$. A Figura 14 ilustra a criação de uma curva bidimensional $\boldsymbol{C}(\xi)$, que assume a forma de um círculo NURBS, por meio de uma transformação projetiva de uma curva B-Spline tridimensional $\boldsymbol{C^w}(\xi)$. Essa transformação consiste em realizar uma projetação no plano $z = 1$ através de segmentos conectados à origem.

A curva $\boldsymbol{C^w}(\xi)$ é denominada curva projetiva e, de forma semelhante, seus pontos de controle recebem o nome de pontos de controle projetivos $\boldsymbol{P_i^w}$. Pode-se expressar os pontos de controle $\boldsymbol{P}_i$ da $\boldsymbol{C}(\xi)$ a partir de $\boldsymbol{P_i^w}$:

$$(\boldsymbol{P}_i)_j = \frac{(\boldsymbol{P_i^w})_j}{w_i} \qquad j = 1, \dots, d_s \tag{20}$$

onde w_i é a última coordenada de $\boldsymbol{P}_i^{\boldsymbol{w}}$**:**

$$w_i = (\boldsymbol{P}_i^{\boldsymbol{w}})_{d_s+1} \tag{21}$$

Do ponto de vista geométrico, w_i é interpretado como a altura dos pontos de controle projetivos.

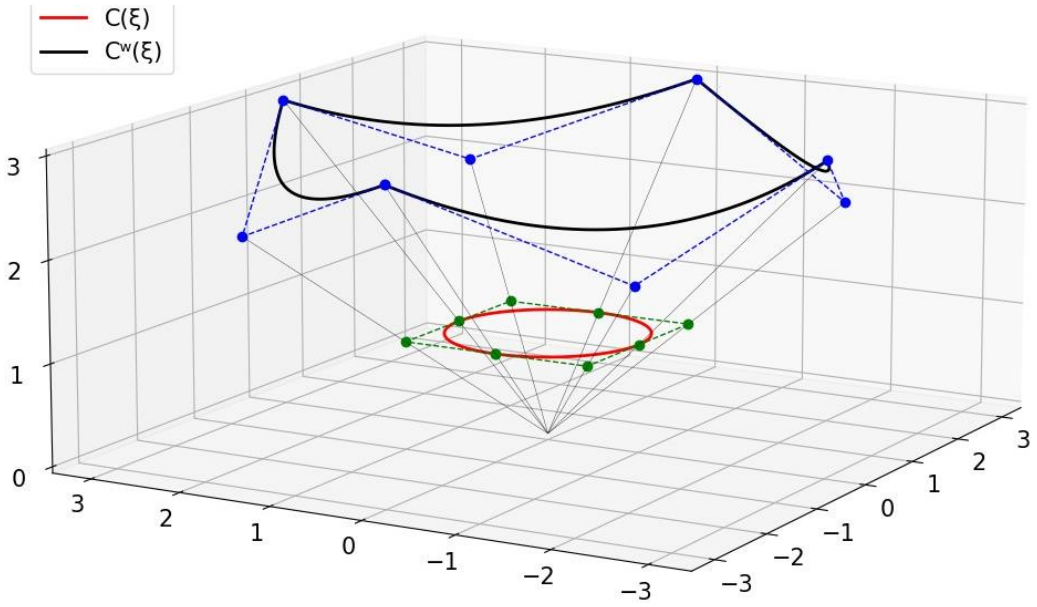

Figura 14: Projeção de uma curva B-Spline no $\mathbb{R}^3$ para a formação de um círculo NURBS no $\mathbb{R}^2$. Adaptado de Cottrell et al. [4].

A perspectiva geométrica de NURBS possui similaridade com o sistema de coordenadas homogêneo. Considerando um ponto $\boldsymbol{P}_i = (x_i, y_i)$, e definindo $w_i > 0$ como um peso associado a esse ponto, $\boldsymbol{P}_i$ pode ser reescrito em coordenadas homogêneas na forma $\boldsymbol{P}_i^{\boldsymbol{w}} = (x_i^w, y_i^w, z_i^w) = (x_i, y_i, 1)$. Multiplicar um valor não nulo às coordenadas homogêneas de um ponto não altera sua posição, assim, $\boldsymbol{P}_i^{\boldsymbol{w}} = (x_i, y_i, 1) = (x_i w_i, y_i w_i, w_i)$.

Se for desejado aplicar as técnicas de refinamento da Seção 2.2.8 em uma curva NURBS, deve-se realizar a transformação dos pontos de controle para $\boldsymbol{P}_i^{\boldsymbol{w}} = (x_i^w, y_i^w, z_i^w) = (x_i w_i, y_i w_i, w_i)$, executar as operações de refinamento e em seguida aplicar a transformação de volta $\boldsymbol{P}_i = (x_i, y_i) = (x_i^w / z_i^w, y_i^w / z_i^w)$ e $w_i = z_i^w$.

2.3.2 Funções de Base NURBS

O ponto chave das NURBS é a presença de pesos $\{w_i\}_{i=1}^{n}$ vinculados aos pontos de controle $\{\boldsymbol{P}_i\}_{i=1}^{n}$. A determinação das funções de base NURBS envolve a inclusão desses pesos, sendo definidas por:

$$R_{i,p}(\xi) = \frac{N_{i,p}(\xi)w_i}{W(\xi)} = \frac{N_{i,p}(\xi)w_i}{\sum_{\hat{\imath}=1}^{n} N_{\hat{\imath},p}(\xi)w_{\hat{\imath}}} \tag{22}$$

onde $N_{i,p}$ são as funções de base B-Splines de ordem p para um determinado vetor de *knots* Ξ. $W(\xi)$ é a função peso dada por uma combinação linear entre $\{N_{i,p}\}_{i=1}^{n}$ e $\{w_i\}_{i=1}^{n}$. Destaca-se que as funções de base NURBS herdam todas as propriedades das funções de base B-Splines (Seção 2.2.4): partição de unidade, não negatividade, independência linear, continuidade e suporte compacto.

É interessante observar que, para o caso especial em que todos os pesos são iguais, as funções de base NURBS se reduzem para funções de base B-Spline. Recordando da propriedade i. da Seção 2.2.4, as funções de base B-Spline constituem uma partição de unidade ($\sum_{\hat{\imath}}^{n} N_{\hat{\imath},p}(\xi) = 1$). Portanto, para o caso especial em questão, o denominador da Equação (22) se reduz ao peso $w_{\hat{\imath}}$, que é cancelado com o peso do numerador w_i, restando $R_{i,p}(\xi) = N_{i,p}(\xi)$.

A primeira derivada de uma função de base é obtida aplicando a regra do quociente na Equação (22):

$$\frac{d}{d\xi}R_{i,p}(\xi) = w_i \frac{W(\xi)N'_{i,p}(\xi) - W'(\xi)N_{i,p}(\xi)}{W^2(\xi)} \tag{23}$$

onde $N'_{i,p}(\xi) = \frac{d}{d\xi}N_{i,p}(\xi)$ e $W'(\xi) = \sum_{\hat{\imath}=1}^{n} N'_{\hat{\imath},p}(\xi)w_{\hat{\imath}}$.

2.3.3 Curvas e Superfícies NURBS

Uma vez que as funções de base NURBS são determinadas, define-se uma curva NURBS analogamente à curva B-Spline da Equação (11):

$$\boldsymbol{C}(\xi) = \sum_{i=1}^{n} R_{i,p}(\xi)\boldsymbol{P}_i \tag{24}$$

De forma similar à Equação (12), uma superfície NURBS é definida por:

$$\boldsymbol{S}(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m} R_{i,j}^{p,q}(\xi,\eta)\boldsymbol{P}_{i,j} \tag{25}$$

onde $R_{i,j}^{p,q}(\xi,\eta)$ são as funções de base NURBS bivariadas tomadas como:

$$R_{i,j}^{p,q}(\xi,\eta) = \frac{N_i(\xi)M_j(\eta)w_{i,j}}{\sum_{\hat{\imath}=1}^{n}\sum_{\hat{\jmath}=1}^{m} N_{\hat{\imath},p}(\xi)M_{\hat{\jmath},q}(\eta)w_{\hat{\imath},\hat{\jmath}}} \tag{26}$$

A Equação (25) pode ser escrita de forma mais compacta usando o índice global A, definido na Seção 2.2.6:

$$\boldsymbol{S}(\xi,\eta) = \sum_{A=1}^{N} R_A^{p,q}(\xi,\eta)\boldsymbol{P}_A \tag{27}$$

2.4 Formação de Superfícies Coons

Superfícies Coons constituem um tipo específico de superfície que pode ser expressa em termos de NURBS. Podem ser particularmente úteis, pois sua formação envolve a definição de quatro curvas NURBS de contorno. Algumas restrições para as curvas de contorno são necessárias para que seja possível a construção de uma superfície Coons [30]. Assumindo quatro curvas NURBS:

$$\boldsymbol{C}_{\boldsymbol{south}}(\xi) = \sum_{i=1}^{n} R_{i,p}(\xi)\boldsymbol{P}_i^{\boldsymbol{south}} \tag{28}$$

$$\boldsymbol{C}_{\boldsymbol{north}}(\xi) = \sum_{i=1}^{n} R_{i,p}(\xi)\boldsymbol{P}_i^{\boldsymbol{north}} \tag{29}$$

$$\boldsymbol{C}_{\boldsymbol{west}}(\eta) = \sum_{j=1}^{m} R_{j,q}(\eta)\boldsymbol{P}_j^{\boldsymbol{west}} \tag{30}$$

$$\boldsymbol{C_{east}}(\eta) = \sum_{j=1}^{m} R_{j,q}(\eta)\boldsymbol{P_j^{east}} \tag{31}$$

Assume-se $\boldsymbol{C_{south}}(\xi)$ e $\boldsymbol{C_{north}}(\xi)$ como curvas opostas na direção paramétrica ξ, definidas pelo mesmo vetor de *knots* $\Xi = [\xi_1, \xi_2, \dots, \xi_{n+p+1}]$ e com mesmo grau polinomial p. Analogamente, designa-se $\boldsymbol{C_{east}}(\eta)$ e $\boldsymbol{C_{west}}(\eta)$ como curvas também opostas, mas na direção η, formadas pelo vetor de *knots* $\mathcal{H} = [\eta_1, \eta_2, \dots, \eta_{m+q+1}]$ e de ordem q. Além disso, é necessário que as extremidades sejam coincidentes, formando uma região fechada: $\boldsymbol{C_{west}}(0) = \boldsymbol{C_{south}}(0)$; $\boldsymbol{C_{south}}(1) = \boldsymbol{C_{east}}(0)$; $\boldsymbol{C_{east}}(1) = \boldsymbol{C_{north}}(1)$; $\boldsymbol{C_{north}}(0) = \boldsymbol{C_{west}}(1)$.

A ideia por trás da formação de uma superfície Coons reside em uma interpolação bivariada, que pode ser construída por uma superposição de interpolações univariadas [29]. Uma superfície Coons é definida por:

$$\boldsymbol{S_{Coons}}(\xi, \eta) = \boldsymbol{R_1}(\xi, \eta) + \boldsymbol{R_2}(\xi, \eta) - \boldsymbol{T}(\xi, \eta) \tag{32}$$

onde $\boldsymbol{R_1}(\xi, \eta)$ é uma superfície *ruled* em ξ, $\boldsymbol{R_2}(\xi, \eta)$ é uma superfície *ruled* em η e $\boldsymbol{T}(\xi, \eta)$ é uma superfície bilinear.

Uma superfície *ruled* governada em uma certa direção é dada por uma interpolação linear de duas curvas definidas ao longo de direções paramétricas diferentes. Fixando um valor paramétrico ao longo dessas curvas diretrizes, obtém-se uma reta. Para o caso de $\boldsymbol{R_1}(\xi, \eta)$ e $\boldsymbol{R_2}(\xi, \eta)$:

$$\boldsymbol{R_1}(\xi, \eta) = (1 - \xi)\boldsymbol{C_{west}}(\eta) + \xi\boldsymbol{C_{east}}(\eta) \tag{33}$$

$$\boldsymbol{R_2}(\xi, \eta) = (1 - \eta)\boldsymbol{C_{south}}(\xi) + \eta\boldsymbol{C_{north}}(\xi) \tag{34}$$

Em termos de superfícies NURBS, as Equações (33) e (34) podem ser reescritas pelas Equações (35) e (36).

$$\boldsymbol{R_1}(\xi, \eta) = \sum_{i=1}^{2}\sum_{j=1}^{m} R_{i,j}^{1,q}(\xi, \eta)\boldsymbol{P_{i,j}^{1}} \tag{35}$$

Onde as funções de base $R_{i,j}^{1,q}(\xi,\eta)$ são associadas aos vetores de *knots* $\Xi_1 = [0,0,1,1]$ e $\mathcal{H}_1 = \mathcal{H} = [\eta_1, \eta_2, \dots, \eta_{m+q+1}]$. Os pontos de controle são dados por: $\boldsymbol{P}_{1,j}^{\boldsymbol{1}} = \boldsymbol{P}_j^{\boldsymbol{east}}$ e $\boldsymbol{P}_{2,j}^{\boldsymbol{1}} = \boldsymbol{P}_j^{\boldsymbol{west}}$.

$$\boldsymbol{R_2}(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{2} R_{i,j}^{p,1}(\xi,\eta)\boldsymbol{P}_{i,j}^{\boldsymbol{2}} \tag{36}$$

Onde as funções $R_{i,j}^{p,1}(\xi,\eta)$ são definidas pelos vetores de *knots* $\Xi_2 = \Xi = [\xi_1, \xi_2, \dots, \xi_{n+p+1}]$ e $\mathcal{H}_2 = [0,0,1,1]$. Nesse caso, os pontos de controle são: $\boldsymbol{P}_{i,1}^{\boldsymbol{2}} = \boldsymbol{P}_i^{\boldsymbol{south}}$ e $\boldsymbol{P}_{i,2}^{\boldsymbol{2}} = \boldsymbol{P}_i^{\boldsymbol{north}}$.

A superfície bilinear é obtida por uma interpolação linear envolvendo segmentos de reta ligando suas extremidades:

$$\begin{aligned}\boldsymbol{T}(\xi,\eta) = (1-\xi)(1-\eta)\boldsymbol{C_{south}}(0) + (1-\xi)\eta\boldsymbol{C_{north}}(0) + \\ \xi(1-\eta)\boldsymbol{C_{south}}(1) + \xi\eta\boldsymbol{C_{north}}(1)\end{aligned} \tag{37}$$

No contexto NURBS, a superfície bilinear pode ser expressada por:

$$\boldsymbol{T}(\xi,\eta) = \sum_{i=1}^{2}\sum_{j=1}^{2} R_{i,j}^{1,1}(\xi,\eta)\boldsymbol{P}_{i,j}^{\boldsymbol{3}} \tag{38}$$

As funções $R_{i,j}^{1,1}(\xi,\eta)$ são definidas por $\Xi_3 = [0,0,1,1]$ e $\mathcal{H}_3 = [0,0,1,1]$. Os pontos de controle correspondem às extremidades das curvas: $\boldsymbol{P}_{1,1}^{\boldsymbol{3}} = \boldsymbol{C_{south}}(0)$; $\boldsymbol{P}_{1,2}^{\boldsymbol{3}} = \boldsymbol{C_{south}}(1)$; $\boldsymbol{P}_{2,1}^{\boldsymbol{3}} = \boldsymbol{C_{north}}(0)$ e $\boldsymbol{P}_{2,2}^{\boldsymbol{3}} = \boldsymbol{C_{north}}(1)$.

Durante a determinação das superfícies $\boldsymbol{R_1}(\xi,\eta)$, $\boldsymbol{R_2}(\xi,\eta)$ e $\boldsymbol{T}(\xi,\eta)$ na forma de NURBS, são efetuadas algumas compatibilizações para a obtenção da superfície Coons $\boldsymbol{S_{Coons}}(\xi,\eta)$. As três superfícies devem apresentar o mesmo vetor de *knots* Ξ e ordem p na direção ξ. De forma similar, devem possuir vetor de *knots* $\mathcal{H}$ e ordem q na direção paramétrica η. Essas compatibilizações podem ser atingidas empregando a técnica de refinamento k (Seção 2.2.8.3), realizando as elevações de ordem e inserção de *knots* necessários [29]. Mais especificamente, o

refinamento k é aplicado nos *knot vectors* que possuírem a forma $[0, 0, 1, 1]$. Como resultado, obtém-se superfícies com o mesmo número de pontos de controle.

A Equação (32) é então definida na sua forma NURBS:

$$\boldsymbol{S_{Coons}}(\xi, \eta) = \sum_{i=1}^{n} \sum_{j=1}^{m} R_{i,j}^{p,q}(\xi, \eta) \boldsymbol{P_{i,j}^{Coons}} \tag{39}$$

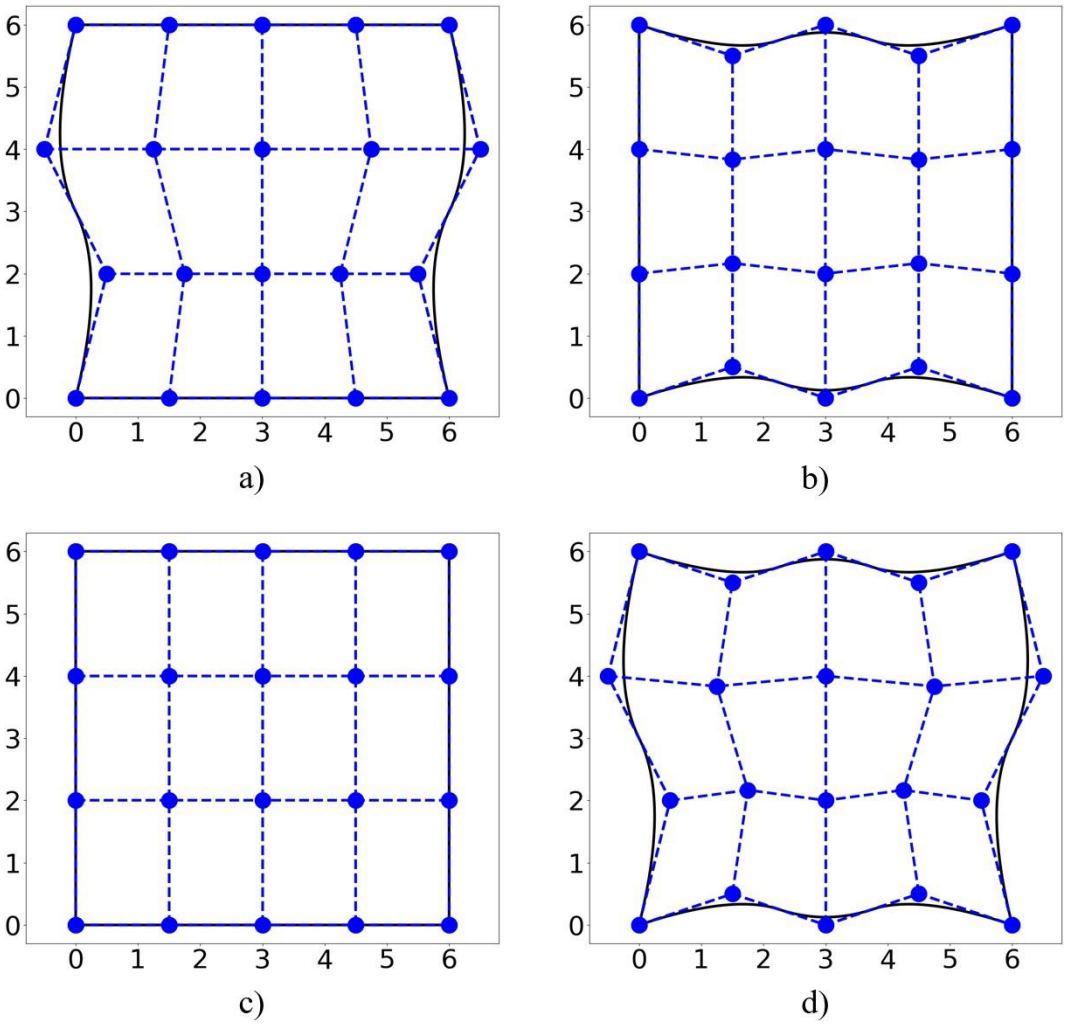

Figura 15: Formação de uma superfície Coons com NURBS. a) Superfície *ruled* na direção ξ. b) Superfície *ruled* em η. c) Superfície bilinear. d) Superfície Coons.

As funções $R_{i,j}^{p,q}(\xi, \eta)$ são obtidas com base nos vetores de *knots* $\Xi = [\xi_1, \xi_2, \dots, \xi_{n+p+1}]$ e $\mathcal{H} = [\eta_1, \eta_2, \dots, \eta_{m+q+1}]$. Os pontos de controle $\boldsymbol{P_{i,j}^{Coons}}$ são dados pela Equação (40), sendo adotado o procedimento de transformação em coordenadas homogêneas da Seção 2.3.1 para a determinação dos pesos.

$$\boldsymbol{P}_{i,j}^{\boldsymbol{Coons}} = \boldsymbol{P}_{i,j}^{\boldsymbol{1}} + \boldsymbol{P}_{i,j}^{\boldsymbol{2}} - \boldsymbol{P}_{i,j}^{\boldsymbol{3}} \quad (40)$$

Na Figura 15, é possível visualizar a sequência de superfícies que são necessárias para a construção de uma superfície do tipo Coons utilizando NURBS. Cada uma dessas superfícies é tratada como uma NURBS.

3 Análise Isogeométrica

Na formulação isoparamétrica de elementos finitos, as funções Lagrangianas são comumente usadas para discretizar a geometria e as variáveis de campo. Em muitos problemas, a geometria é aproximada, o que inevitavelmente leva a erros de natureza geométrica. Quando um processo de discretização é realizado em um modelo CAD, a informação da geometria inicial é perdida, formando um processo unidirecional entre modelagem e análise [41].

A análise isogeométrica segue o princípio da formulação isoparamétrica. O diferencial reside no fato de que as funções escolhidas para a análise são as mesmas que definem exatamente a geometria. Isso promove uma ligação com o modelo CAD inicial, tornando o processo de análise de natureza bidirecional em relação ao *design* [41].

Para o bom entendimento da formulação da análise isogeométrica, é necessário conhecer as novas implicações que essa abordagem resulta. Dessa forma, esse capítulo é iniciado pela definição de diferentes espaços de domínios, que são relevantes para a análise em questão. Em seguida, serão apresentadas expressões para o mapeamento entre esses espaços. Por fim, é desenvolvida a formulação isogeométrica de elementos finitos, que usam NURBS como base para análise. A formulação matemática será tratada para o caso bidimensional, pois essa foi a proposição adotada para a ferramenta computacional desenvolvida.

3.1 Espaços de Domínio

São quatro domínios relevantes no contexto de AIG com superfícies NURBS: espaço *parent*, espaço indicial, espaço paramétrico e espaço físico. Destaca-se a importância de se obter afinidade com esses domínios, antes de partir para a formulação isogeométrica.

3.1.1 Espaço Indicial e Espaço Paramétrico

Os *knot vectors* de superfícies NURBS são compostos por coordenadas paramétricas, a partir das quais surge o conceito de espaço indicial e espaço

paramétrico. A diferença entre esses dois domínios está essencialmente na presença de multiplicidade do vetor de *knots*: o espaço indicial é formado a partir da consideração de todas as coordenadas paramétricas, independentemente de haver valores repetidos ou não; já no espaço paramétrico, considera-se apenas os valores únicos, sem repetição.

A Figura 16 mostra os espaços indicial e paramétrico para uma superfície NURBS formada pelos vetores de *knots* $\Xi = [0, 0, 0, 0.333, 0.667, 1, 1, 1]$ e $\mathcal{H} = [0, 0, 0, 0.5, 1, 1, 1]$. As regiões em hachura formam o espaço paramétrico, pois são delimitadas por valores diferentes e crescentes de coordenadas paramétricas. Por outro lado, as regiões de cor branca apresentam área paramétrica nula, uma vez que são definidas entre os valores repetidos dos *knot vectors*. O espaço indicial engloba toda a região, incluindo elementos de área paramétrica nula e não nula.

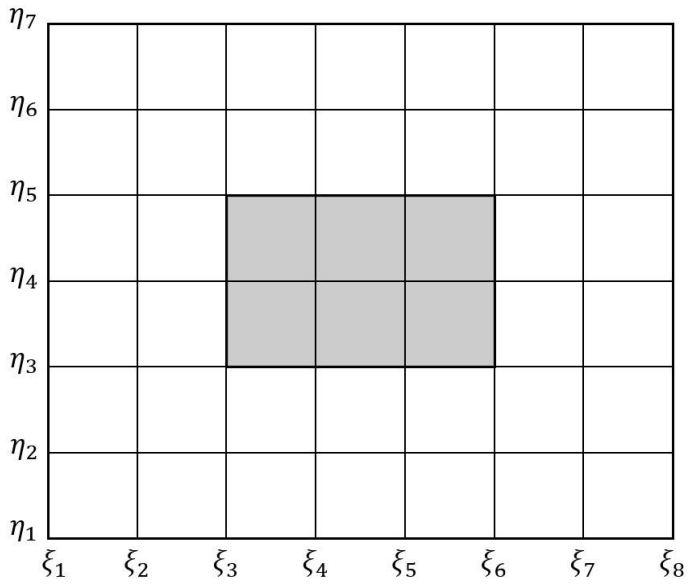

Figura 16: Espaços indicial e paramétrico definidos pelos vetores de *knots* $\Xi = [0, 0, 0, 0.333, 0.667, 1, 1, 1]$ e $\mathcal{H} = [0, 0, 0, 0.5, 1, 1, 1]$.

Na formulação da análise isogeométrica, os elementos devem apresentar áreas paramétrica diferentes de zero. Assim, é o espaço paramétrico que define a quantidade de elementos presentes em um modelo.

Uma presunção deste trabalho é que apenas vetores de *knots* normalizados são considerados, isto é, contidos no intervalo $[0, 1]$. Diante disso, para o caso bivariado ($d_p = 2$), o domínio paramétrico $\widehat{\Omega} = [0, 1]^{d_p}$ apresenta área total de

valor unitário. De maneira formal, associa-se domínio $\widehat{\Omega} = [0,1]^2$ ao conjunto de coordenadas paramétricas $\boldsymbol{\xi} = (\xi, \eta)$, contidas nos vetores de *knots* únicos $\widehat{\Xi}$ e $\widehat{\mathcal{H}}$. Os vetores de *knots* únicos possuem apenas coordenadas paramétricas não repetidas, onde $\widehat{\Xi} \subset \Xi$ e $\widehat{\mathcal{H}} \subset \mathcal{H}$. São definidos por:

$$\widehat{\Xi} = \left[\xi_1, \dots, \xi_{n_u}\right] \qquad \xi_i < \xi_{i+1} \qquad para \quad 1 \leq i \leq n_u - 1 \tag{41}$$

$$\widehat{\mathcal{H}} = \left[\eta_1, \dots, \eta_{n_v}\right] \qquad \eta_j < \eta_{j+1} \qquad para \quad 1 \leq j \leq n_v - 1 \tag{42}$$

Sendo n_u é o número de elementos únicos do vetor de *knots* Ξ e n_v é o número de elementos únicos vetor de *knots* $\mathcal{H}$. A partir dos *knot vectors* únicos, determina-se o domínio paramétricos de um elemento:

$$\widehat{\Omega}^e = [\xi_i, \xi_{i+1}] \otimes \left[\eta_j, \eta_{j+1}\right] \qquad para \quad \begin{matrix} 1 \leq i \leq n_u - 1 \\ 1 \leq j \leq n_v - 1 \\ \xi_i \in \widehat{\Xi}, \quad \eta_j \in \widehat{\mathcal{H}} \end{matrix} \tag{43}$$

3.1.2 Espaço Físico

O espaço físico contém a geometria em si. No caso bivariado, a geometria é formada por uma superfície NURBS, definida pela Equação (25) na Seção 2.3.3. No espaço bidimensional ($d_s = 2$), o domínio físico $\Omega \subset \mathbb{R}^{d_s}$ está associado ao sistema de coordenadas $\boldsymbol{x} = (x, y)$.

A Figura 17 demonstra o espaço físico bidimensional de uma superfície NURBS, construída com os *knot vectors* que determinam o espaço paramétrico da Figura 16. É interessante notar a divisão dos elementos no espaço físico. As divisões ocorrem ao longo das localidades correspondentes aos *knots* dos vetores de *knots* únicos $\widehat{\Xi}$ e $\widehat{\mathcal{H}}$. Observa-se que a quantidade de elementos no espaço físico é a mesma quantidade de elementos no espaço paramétrico.

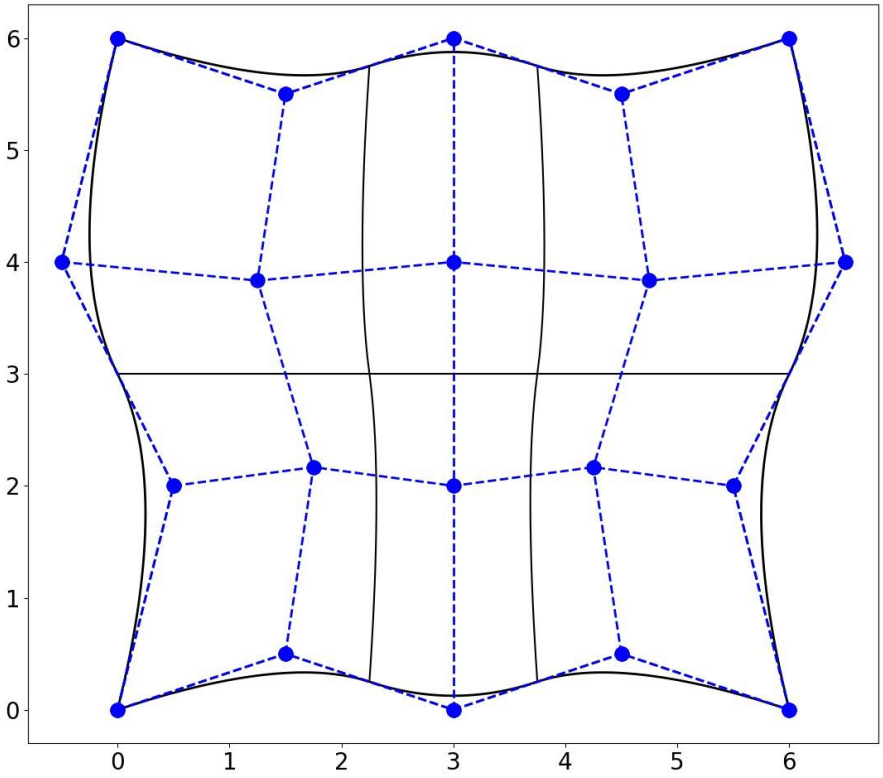

Figura 17: Espaço físico de uma superfície NURBS.

3.1.3 Espaço *Parent*

Ao contrário dos demais espaços mostrados previamente, o espaço *parent* ou espaço gaussiano não está relacionado à natureza NURBS, sendo usado apenas para a finalidade de permitir a integração numérica pelo método da quadratura de Gauss, que necessita de um domínio com intervalo fixo $\widetilde{\Omega} = [-1, 1]^{d_p}$. Para o caso bivariado, o domínio do espaço *parent* $\widetilde{\Omega} = [-1, 1]^2$ é associado ao conjunto de coordenadas locais $\tilde{\boldsymbol{\xi}} = (\tilde{\xi}, \tilde{\eta})$. Na Figura 3 é ilustrada a representação desse espaço.

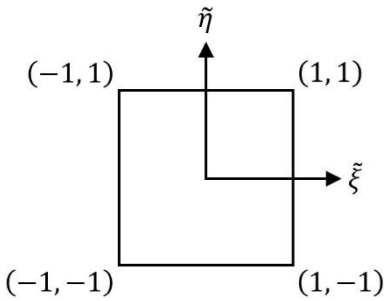

Figura 18: Espaço *parent*.

3.2 Geometria e Variáveis de Campo do Elemento

Conforme abordado no Capítulo 2, B-Splines e NURBS são expressas em função de coordenadas paramétricas. Para a análise isogeométrica, é necessário operar em termos de coordenadas locais do espaço *parent*, pois é nesse espaço em que é realizada a integração numérica para a determinação das matrizes de rigidez dos elementos. Para um elemento específico e, o objetivo é definir sua geometria pela Equação (44), em função das coordenadas locais $\tilde{\xi}$ e $\tilde{\eta}$.

$$\boldsymbol{x}^e(\tilde{\boldsymbol{\xi}}) = \boldsymbol{x}^e(\tilde{\xi}, \tilde{\eta}) = \sum_{a=1}^{n_{en}} R_a^e(\tilde{\xi}, \tilde{\eta}) \boldsymbol{P}_a^e \tag{44}$$

Onde a representa o índice das funções de base $R_a^e(\tilde{\xi}, \tilde{\eta})$ não nulas ao longo do elemento, e $\boldsymbol{P}_a^e$ são os pontos de controle associados. Os pontos de controle associados a um elemento definem sua conectividade. Enquanto em MEF a conectividade é dada nos nós do elemento, em AIG a conectividade é dada pelos pontos de controle. Uma abordagem sobre conectividade é dada na Seção 3.3.

As variáveis de campo da análise são aproximadas pela Equação (45), também definida em termos das coordenadas paramétricas locais.

$$\boldsymbol{u}^e(\tilde{\boldsymbol{\xi}}) = \boldsymbol{u}^e(\tilde{\xi}, \tilde{\eta}) = \sum_{a=1}^{n_{en}} R_a^e(\tilde{\xi}, \tilde{\eta}) \boldsymbol{d}_a^e \tag{45}$$

Onde $\boldsymbol{d}_a^e$ são as variáveis nodais associadas aos graus de liberdade (GL) de um elemento. Em elementos finitos os GL são atribuídos aos nós, na análise isogeométrica os GL são atribuídos aos pontos de controle. Vale destacar que na AIG, para problemas de elasticidade, os graus de liberdade não correspondem a deslocamentos do modelo, pois os pontos de controle não são interpolatórios à geometria. Nesse sentido, a Equação (45) assume uma forma semelhante ao Método de Rayleigh-Ritz.

Mapeamentos adequados entre os diferentes espaços devem ser realizados para possibilitar a atuação do espaço *parent* nas Equações (44) e (45).

3.3 Conectividade

Na análise isogeométrica, cada elemento possui pontos de controle específicos $\{\boldsymbol{P}_a^e\}_{a=1}^{n_{en}}$, que estabelecem conectividade com o mesmo. O número de pontos de controle de conectividade em um elemento é igual a quantidade de funções de base diferentes de zero n_{en} naquele elemento, sendo dependente das ordens polinomiais adotadas $n_{en} = (p+1)(q+1)$. Isso se deve a propriedade v. da Seção 2.2.4, que trata do suporte das funções de base. A definição de conectividade na análise isogeométrica é bem distinta da definição para análise isoparamétrica tradicional, pois nesta última a quantidade de nós de conectividade com um elemento depende do tipo de elemento adotado. Por exemplo, um elemento Q4 possui quatro nós de conectividade.

Assim, com base nos índices das funções de base diferentes de zero, determina-se os pontos de controle associados, os quais estabelecem conectividade no elemento avaliado.

Para o caso univariado, tem-se a curva quadrática da Figura 19, definida pelo *knot vector* $\Xi = [0, 0, 0, 0.333, 0.667, 1, 1, 1]$. O primeiro elemento está contido no domínio paramétrico $\widehat{\Omega}^1 = [0, 0.333]$. Observando as funções de base $\{N_{i,2}\}_{i=1}^{5}$, verifica-se que as funções diferentes de zero são: $N_{1,2}$, $N_{2,2}$ e $N_{3,2}$. Dessa forma, a conectividade é dada pelos pontos de controle $\boldsymbol{P}_i$ com $i = 1, 2, 3$.

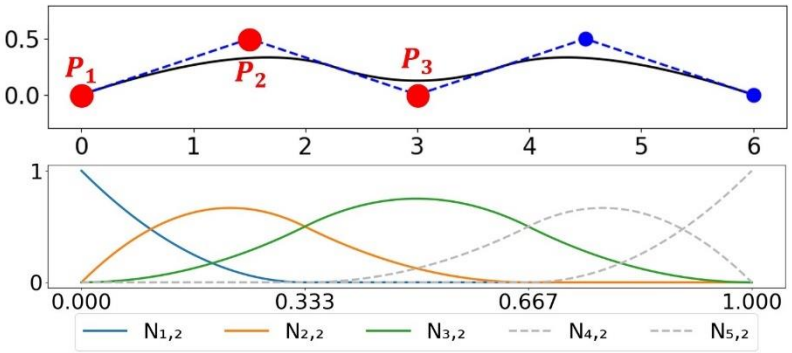

Figura 19: Pontos de controle de conectividade em uma curva NURBS.

Para o caso bivariado, toma-se como exemplo a Figura 20, baseada em figuras anteriores deste trabalho. O elemento destacado está contido no domínio físico Ω^1 e associado ao domínio paramétrico $\widehat{\Omega}^1$. Este último, está compreendido no intervalo: $\widehat{\Omega}^1 = [0, 0.333] \otimes [0, 0.5]$. Nesse caso, a determinação dos pontos de

controle de conectividade requer a análise de cada direção paramétrica individualmente.

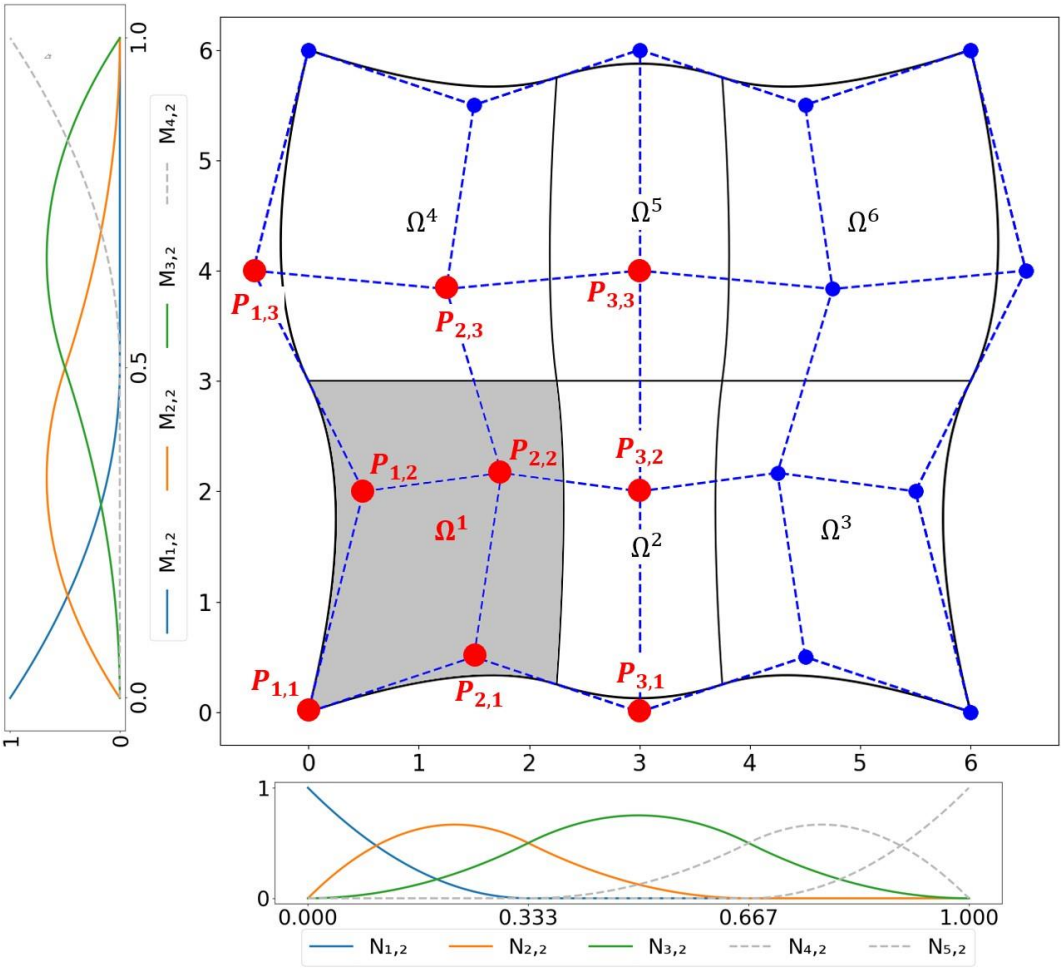

Figura 20: Pontos de controle de conectividade em uma superfície NURBS.

Os vetores de *knots* são dados por $\Xi = [0, 0, 0, 0.333, 0.667, 1, 1, 1]$ e $\mathcal{H} = [0, 0, 0, 0.5, 1, 1, 1]$. As funções base referentes a esses *knot vectors* são apresentadas na Figura 20 . Horizontalmente, estão dispostas as funções $\{N_{i,2}\}_{i=1}^{5}$ relacionadas a Ξ e, verticalmente, as funções $\{M_{j,2}\}_{j=1}^{4}$ de $\mathcal{H}$. Nota-se que, entre o intervalo paramétrico $[0, 0.333]$, na direção ξ, as funções não nulas são: $N_{1,2}$, $N_{2,2}$ e $N_{3,2}$. No intervalo $[0, 0.5]$, na direção η, as funções diferentes de zero são: $M_{1,2}$, $M_{2,2}$ e $M_{3,2}$. A conectividade desse elemento é dada pelos pontos de controle associados a essas funções: $\boldsymbol{P}_{i,j}$ com $i = 1, 2, 3$ e $j = 1, 2, 3$.

3.4 Mapeamentos

A utilização de NURBS para análise numérica incorpora o conceito de espaço paramétrico, que engloba todos os elementos de um dado *patch*. Assim, para realizar a transformação do espaço *parent* para o espaço físico, é necessário efetuar dois mapeamentos: o primeiro entre os espaços *parent* e paramétrico $\tilde{\phi}^e: \tilde{\Omega} \rightarrow \hat{\Omega}^e$; o segundo entre os espaços paramétrico e físico $\boldsymbol{S}^e: \hat{\Omega}^e \rightarrow \Omega^e$. A composição desses dois mapeamentos $\boldsymbol{S}^e \circ \tilde{\boldsymbol{\phi}}^e$ resulta em $\boldsymbol{x}^e: \tilde{\Omega} \rightarrow \Omega^e$. A Figura 21 ilustra todos esses mapeamentos.

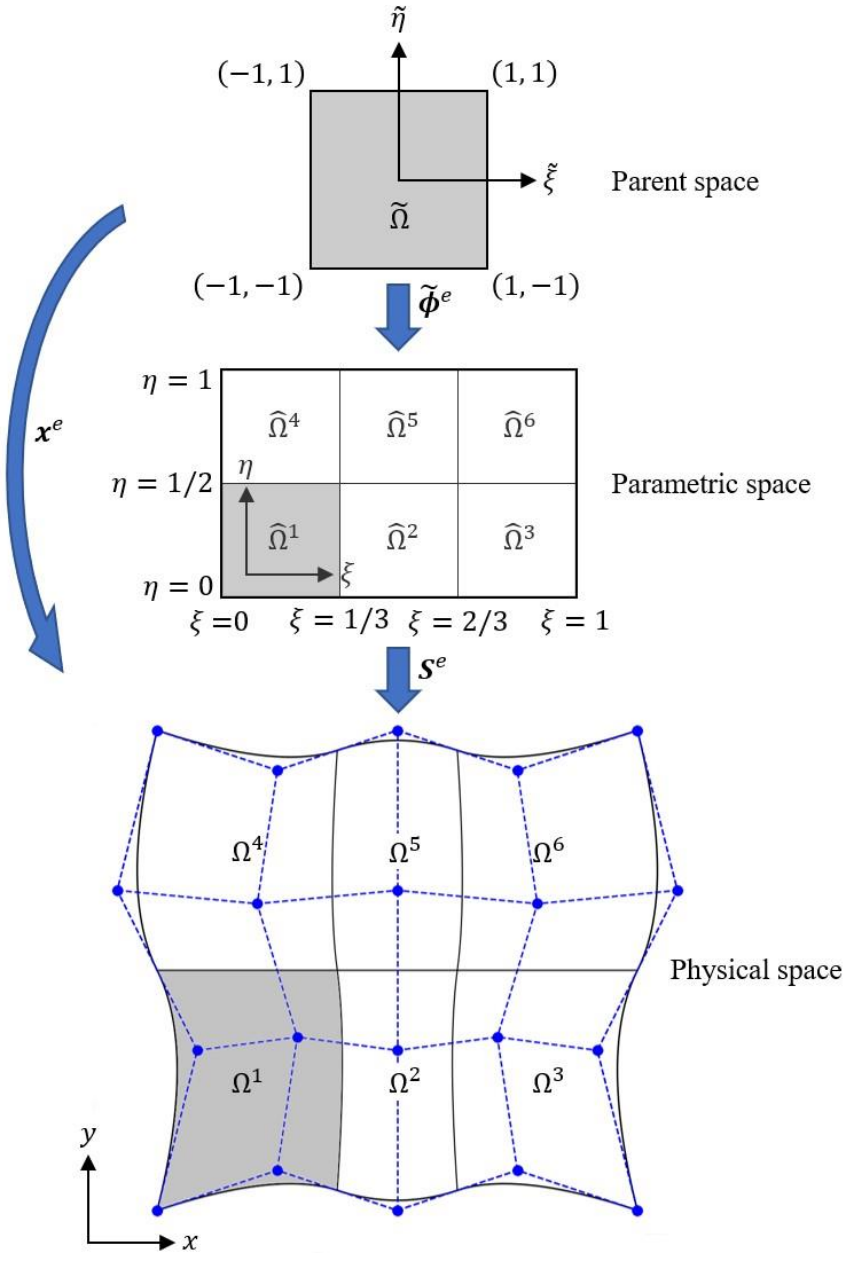

Figura 21: Mapeamentos entre os diferentes espaços. A cor cinza indica um elemento no contexto da análise isogeométrica.

O mapeamento $\tilde{\phi}^e$ pode ser obtido através de uma relação de proporcionalidade linear entre os espaços *parent* e paramétrico, conforme ilustrado na Figura 22. O domínio do espaço *parent* é definido no intervalo $[-1, 1]$ em cada direção. Assim, admitindo uma coordenada qualquer $\tilde{\xi}$, a distância até o início do intervalo, normalizada pelo intervalo total é dada por:

$$\frac{\tilde{\xi} + 1}{2} \tag{46}$$

A distância equivalente no espaço paramétrico pode ser obtida pela relação:

$$\frac{\tilde{\xi} + 1}{2}(\hat{\xi}_{i+1} - \hat{\xi}_i) \tag{47}$$

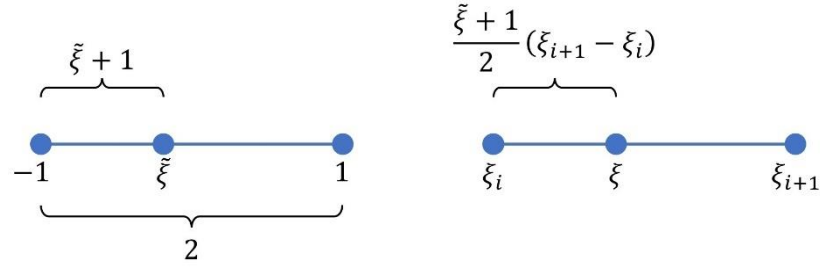

Figura 22: Relação entre espaços *parent* e paramétrico.

Assim, a relação obtida assume a forma:

$$\begin{aligned} \tilde{\phi}^e(\tilde{\xi}) &= \xi_i + \frac{\tilde{\xi} + 1}{2}(\xi_{i+1} - \xi_i) \\ &= \xi_i + \frac{\tilde{\xi}}{2}(\xi_{i+1} - \xi_i) + \frac{1}{2}(\xi_{i+1} - \xi_i) \\ &= \frac{1}{2}\left[(\xi_{i+1} - \xi_i)\tilde{\xi} + (\xi_{i+1} + \xi_i)\right] \end{aligned} \tag{48}$$

Para o caso bivariado, o mapeamento entre o espaço *parent* e o espaço paramétrico é:

$$\tilde{\boldsymbol{\phi}}^e(\tilde{\boldsymbol{\xi}}) = \tilde{\boldsymbol{\phi}}^e(\tilde{\xi}, \tilde{\eta}) = \begin{cases} \xi = \frac{1}{2}\left[(\hat{\xi}_{i+1} - \hat{\xi}_i)\tilde{\xi} + (\hat{\xi}_{i+1} + \hat{\xi}_i)\right] \\ \eta = \frac{1}{2}\left[(\hat{\eta}_{i+1} - \hat{\eta}_i)\tilde{\eta} + (\hat{\eta}_{i+1} + \hat{\eta}_i)\right] \end{cases} \tag{49}$$

E o Jacobiano associado é obtido por:

$$\begin{aligned} \boldsymbol{J}_{\tilde{\xi}} &= \begin{bmatrix} \frac{\partial \xi}{\partial \tilde{\xi}} & \frac{\partial \eta}{\partial \tilde{\xi}} \\ \frac{\partial \xi}{\partial \tilde{\eta}} & \frac{\partial \eta}{\partial \tilde{\eta}} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{2}(\xi_{i+1} - \xi_i) & 0 \\ 0 & \frac{1}{2}(\eta_{i+1} - \eta_i) \end{bmatrix} \end{aligned} \tag{50}$$

O determinante associado é dado pela expressão:

$$\left|\boldsymbol{J}_{\tilde{\xi}}\right| = \frac{1}{4}(\xi_{i+1} - \xi_i)(\eta_{i+1} - \eta_i) \tag{51}$$

A Equação (25) fornece uma relação entre as coordenadas do espaço paramétrico e a geometria do espaço físico de uma superfície NURBS. É possível restringi-la a um único elemento considerando apenas os termos com funções de base não nulas. Assim, o mapeamento $\boldsymbol{S}^e$ entre o espaço paramétrico e o espaço físico de um elemento é dado por:

$$\boldsymbol{S}^e(\boldsymbol{\xi}) = \boldsymbol{S}^e(\xi, \eta) = \sum_{a=1}^{n_{en}} R_a^e(\xi, \eta) \boldsymbol{P}_a^e \tag{52}$$

O Jacobiano dessa transformação assume a forma:

$$\begin{aligned} \boldsymbol{J}_{\boldsymbol{\xi}} &= \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \\ &= \begin{bmatrix} \sum_{a=1}^{n_{en}} \frac{\partial R_a^e}{\partial \xi} P_{x\,a}^{\ e} & \sum_{a=1}^{n_{en}} \frac{\partial R_a^e}{\partial \xi} P_{y\,a}^{\ e} \\ \sum_{a=1}^{n_{en}} \frac{\partial R_a^e}{\partial \eta} P_{x\,a}^{\ e} & \sum_{a=1}^{n_{en}} \frac{\partial R_a^e}{\partial \eta} P_{y\,a}^{\ e} \end{bmatrix} \end{aligned} \tag{53}$$

$$= \begin{bmatrix} \dfrac{\partial R_1^e}{\partial \xi} & \dfrac{\partial R_2^e}{\partial \xi} & \cdots & \dfrac{\partial R_{n_{en}}^e}{\partial \xi} \\ \dfrac{\partial R_1^e}{\partial \eta} & \dfrac{\partial R_2^e}{\partial \eta} & \cdots & \dfrac{\partial R_{n_{en}}^e}{\partial \eta} \end{bmatrix} \begin{bmatrix} P_{x\,1}^e & P_{y\,1}^e \\ P_{x\,2}^e & P_{y\,2}^e \\ \vdots & \vdots \\ P_{x\,n_{en}}^e & P_{y\,n_{en}}^e \end{bmatrix}$$

O determinante associado a esse mapeamento é indicado por $|\boldsymbol{J}_{\xi}|$.

Combinando os mapeamentos anteriores, obtém-se o mapeamento $\boldsymbol{x}^e$, que leva do espaço *parent* para o espaço físico:

$$\boldsymbol{x}^e(\tilde{\boldsymbol{\xi}}) = \boldsymbol{x}^e(\tilde{\xi}, \tilde{\eta}) = \sum_{a=1}^{n_{en}} R_a^e\left(\tilde{\phi}(\tilde{\xi}, \tilde{\eta})\right) \boldsymbol{P}_a^e \tag{54}$$

Pode-se designar o determinante associado a $\boldsymbol{x}^e$ pela expressão:

$$|\boldsymbol{J}| = |\boldsymbol{J}_{\xi}||\boldsymbol{J}_{\tilde{\xi}}| \tag{55}$$

É através desse mapeamento final e do determinante do Jacobiano associado que será possível realizar a transformação de uma integração no espaço físico para uma integração no espaço *parent*. Esse artifício é essencial para cálculo das matrizes de rigidez locais e do vetor de forças nodais, através da integração numérica pelo método da quadratura de Gauss.

3.5 Formulação Bidimensional de Elasticidade Linear para Análise Isogeométrica

Neste trabalho, a formulação é voltada para problemas bidimensionais de sólidos elásticos lineares sob estado plano de tensões. Inicialmente são apresentados conceitos fundamentais sobre a relação tensão-deformação e a relação deformação-deslocamento. Prossegue-se então para a formulação fraca de elasticidade linear estática, seguida da formulação discreta para análise isogeométrica.

3.5.1 Relação Constitutiva e Cinemática

A relação entre tensões e deformações é dada pela relação constitutiva. Para problemas lineares elásticos de uma dimensão, a relação constitutiva é dada pela lei de Hooke: $\sigma = E\varepsilon$, sendo E o módulo de elasticidade. No contexto bidimensional, a lei de Hooke pode ser escrita em uma forma mais genérica:

$$\boldsymbol{\sigma} = \boldsymbol{E}\varepsilon$$

$$\begin{bmatrix}\sigma_{xx}\\ \sigma_{yy}\\ \tau_{xy}\end{bmatrix} = \frac{E}{1-\nu^2}\begin{bmatrix}1 & \nu & 0\\ \nu & 1 & 0\\ 0 & 0 & \frac{1}{2}(1-\nu)\end{bmatrix}\begin{bmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ 2\gamma_{xy}\end{bmatrix} \tag{56}$$

onde $\boldsymbol{\sigma}$ representa as tensões, ε representa as deformações, $\boldsymbol{E}$ é matriz constitutiva para estado plano de tensões e ν é o coeficiente de Poisson.

A cinemática descreve a relação entre deformações e deslocamentos:

$$\varepsilon = \boldsymbol{D}\boldsymbol{u}$$

$$\begin{bmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ 2\gamma_{xy}\end{bmatrix} = \begin{bmatrix}\frac{\partial}{\partial x} & 0\\ 0 & \frac{\partial}{\partial y}\\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x}\end{bmatrix}\begin{bmatrix}u_x\\ u_y\end{bmatrix} \tag{57}$$

sendo $\boldsymbol{u}$ representa os deslocamentos e $\boldsymbol{D}$ o operador diferencial.

3.5.2 Formulação Fraca e Discretização para Análise Isogeométrica

A formulação fraca para a análise isogeométrica, juntamente com o processo de discretização e obtenção de uma formulação integral, segue um procedimento idêntico ao que é aplicado em elementos finitos. A distinção reside na seleção das funções de base. No caso do MEF, as funções de base são escolhidas como funções polinomiais, enquanto que na AIG opta-se por funções NURBS.

Inicialmente é considerado um determinado domínio Ω, cujo contorno é dado por Γ. São assumidos dois tipos de condições de contorno: essenciais e naturais. No contexto de problemas de elasticidade, essas condições correspondem

a imposição de deslocamentos $\overline{\boldsymbol{u}}$ e tesões $\bar{\boldsymbol{t}}$, respectivamente. Denota-se Γ_u como sendo a região onde as condições essenciais são impostas; e Γ_t como a região de aplicação das condições naturais. Tem-se que: $\Gamma = \Gamma_u \cup \Gamma_t$ e $\Gamma_u \cap \Gamma_t = \emptyset$.

Parte-se para a aplicação do princípio dos trabalhos virtuais. O fundamento do princípio dos trabalhos virtuais diz que o trabalho interno é igual ao trabalho realizado pelas forças externas, quando a estrutura é submetida a deslocamento virtuais:

$$\delta W_{int} = \delta W_{ext} \tag{58}$$

Para o caso de problemas contínuos, aplica-se uma integração sob o domínio. O objetivo é encontrar $\boldsymbol{u}$ de acordo com a formulação fraca:

$$\int_{\Omega^e} (\delta\varepsilon)^T \boldsymbol{\sigma}\, d\Omega = \int_{\Gamma^e} (\delta\boldsymbol{u})^T\, \bar{\boldsymbol{t}}\, d\Gamma + \int_{\Omega^e} (\delta\boldsymbol{u})^T \boldsymbol{b}\, d\Omega \tag{59}$$

onde $\boldsymbol{b}$ são as forças de corpo, $\delta\varepsilon$ são as deformações virtuais e $\delta\boldsymbol{u}$ os deslocamentos virtuais.

Os deslocamentos $\boldsymbol{u}$ podem ser obtidos a partir de um processo de interpolação entre as funções de base e coeficientes, definidos como vetores de deslocamento nodais $\boldsymbol{d}_a^e = \left[u_{x\,a}^e, u_{y\,a}^e\right]^T$, como mostrado na Equação (60). Emprega-se o mapeamento $\widetilde{\boldsymbol{\phi}}^e$ para determinar as coordenadas paramétricas $\boldsymbol{\xi} = (\xi, \eta)$, que vão nas funções de base.

$$\boldsymbol{u}(\boldsymbol{\xi}) = \boldsymbol{u}(\xi, \eta) = \sum_{a=1}^{n_{en}} R_a^e(\xi, \eta)\boldsymbol{d}_a^e \tag{60}$$

A Equação (60) pode ser reescrita na forma matricial:

$$\boldsymbol{u} = \boldsymbol{R}^e\boldsymbol{d}^e \tag{61}$$

$$\begin{bmatrix} u_x \\ u_y \end{bmatrix} = \begin{bmatrix} R_1^e & R_2^e & \cdots & R_{n_{en}}^e & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & R_1^e & R_2^e & \cdots & R_{n_{en}}^e \end{bmatrix} \begin{bmatrix} u_{x\,1} \\ u_{x\,2} \\ \vdots \\ u_{x\,n_{en}} \\ u_{y\,1} \\ u_{y\,2} \\ \vdots \\ u_{y\,n_{en}} \end{bmatrix}$$

onde $\boldsymbol{R}^e$ é a matriz de funções de base e $\boldsymbol{d}^e$ é o vetor de deslocamentos nodais do elemento.

Substituindo a Equação (61) na relação cinemática da Equação (57):

$$\varepsilon = \boldsymbol{B}^e \boldsymbol{d}^e$$

$$\begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ 2\gamma_{xy} \end{bmatrix} = \begin{bmatrix} \frac{\partial R_1^e}{\partial x} & \frac{\partial R_2^e}{\partial x} & \cdots & \frac{\partial R_{n_{en}}^e}{\partial x} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\partial R_1^e}{\partial y} & \frac{\partial R_2^e}{\partial y} & \cdots & \frac{\partial R_{n_{en}}^e}{\partial y} \\ \frac{\partial R_1^e}{\partial y} & \frac{\partial R_2^e}{\partial y} & \cdots & \frac{\partial R_{n_{en}}^e}{\partial y} & \frac{\partial R_1^e}{\partial x} & \frac{\partial R_2^e}{\partial x} & \cdots & \frac{\partial R_{n_{en}}^e}{\partial x} \end{bmatrix} \begin{bmatrix} u_{x\,1} \\ u_{x\,2} \\ \vdots \\ u_{x\,n_{en}} \\ u_{y\,1} \\ u_{y\,2} \\ \vdots \\ u_{y\,n_{en}} \end{bmatrix} \quad (62)$$

onde $\boldsymbol{B}^e$ é a matriz deformação-deslocamento do elemento. Observa-se que $\boldsymbol{B}^e$ apresenta apenas termos com derivadas das funções de base em relação às coordenadas x e y. No entanto, essas funções são definidas em coordenadas paramétricas ξ e η. Para resolver essa adversidade, recorre-se ao uso da matriz inversa jacobiana associada ao mapeamento $\boldsymbol{S}^e$. Aplicando a regra da cadeia para obter os termos diferenciais desejados:

$$\frac{\partial R_a^e}{\partial x} = \frac{\partial R_a^e}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial R_a^e}{\partial \eta}\frac{\partial \eta}{\partial x} \quad (63)$$

$$\frac{\partial R_a^e}{\partial y} = \frac{\partial R_a^e}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial R_a^e}{\partial \eta}\frac{\partial \eta}{\partial y} \quad (64)$$

A matriz inversa jacobiana fornece os termos $\frac{\partial \xi}{\partial x}$, $\frac{\partial \eta}{\partial x}$, $\frac{\partial \xi}{\partial y}$ e $\frac{\partial \eta}{\partial y}$ da Equação (63) e Equação (64):

$$\boldsymbol{J}_{\xi}^{-1} = \begin{bmatrix} \dfrac{\partial \xi}{\partial x} & \dfrac{\partial \eta}{\partial x} \\ \dfrac{\partial \xi}{\partial y} & \dfrac{\partial \eta}{\partial y} \end{bmatrix} \tag{65}$$

Após a determinação das Equações (63) e (64), é montada a matriz $\boldsymbol{B}^{e}$.

Pelas Equações (61) e (62), as deformações e deslocamentos virtuais podem ser expressadas por:

$$(\delta\varepsilon)^{T} = (\delta \boldsymbol{d}^{\boldsymbol{e}})^{T} \boldsymbol{B}^{eT} \qquad (\delta \boldsymbol{u})^{T} = (\delta \boldsymbol{d}^{\boldsymbol{e}})^{T} \boldsymbol{R}^{\boldsymbol{eT}} \tag{66}$$

Substituindo as Equações (66) e a relação constitutiva (56) na formulação fraca em (59):

$$(\delta \boldsymbol{d}^{\boldsymbol{e}}) \int_{\Omega^{e}} \boldsymbol{B}^{eT} \boldsymbol{E} \boldsymbol{B}^{e} \boldsymbol{d}_{e} \, d\Omega = (\delta \boldsymbol{d}^{\boldsymbol{e}}) \int_{\Gamma^{e}} \boldsymbol{R}^{\boldsymbol{eT}} \, \bar{\boldsymbol{t}} \, d\Gamma + (\delta \boldsymbol{d}^{\boldsymbol{e}}) \int_{\Omega^{e}} \boldsymbol{R}^{\boldsymbol{eT}} \boldsymbol{b} \, d\Omega \tag{67}$$

Para um deslocamento virtual $\delta \boldsymbol{d}^{\boldsymbol{e}}$ arbitrário, tem-se o sistema local:

$$\boldsymbol{K}_{e} \boldsymbol{d}_{e} = \boldsymbol{f}_{e} \tag{68}$$

onde $\boldsymbol{K}_{e}$ é a matriz de rigidez local:

$$\boldsymbol{K}_{e} = \int_{\Omega^{e}} \boldsymbol{B}^{eT} \boldsymbol{E} \boldsymbol{B}^{e} \, d\Omega \tag{69}$$

e $\boldsymbol{f}_{e}$ é o vetor de forças externas local:

$$\boldsymbol{f}_{e} = \int_{\Gamma^{e}} \boldsymbol{R}^{\boldsymbol{eT}} \, \bar{\boldsymbol{t}} \, d\Gamma + \int_{\Omega^{e}} \boldsymbol{R}^{\boldsymbol{eT}} \boldsymbol{b} \, d\Omega \tag{70}$$

A partir de um conjunto de elementos do problema, a montagem da matriz de rigidez global e do vetor de forças externas global são realizadas. Com isso, o sistema é resolvido para obtenção de um vetor de deslocamentos $\boldsymbol{d}$:

$$\boldsymbol{K} \boldsymbol{d} = \boldsymbol{f} \tag{71}$$

3.5.3 Integração Numérica

A determinação das matrizes de rigidez locais, bem como os vetores locais de forças externas envolvem o cálculo de integrais. De forma a possibilitar a implementação computacional, métodos de integração numérica são empregados. O método da quadratura de Gauss é bastante difundido para esse propósito, sendo ele adotado neste trabalho. A finalidade é alterar os limites de integração para o domínio compreendido pelo espaço *parent*. Os jacobianos dos mapeamentos abordados na Seção 3.4 são essenciais para essa finalidade. Assim, pode-se realizar a mudança dos domínios da matriz de rigidez e vetor de forças externas:

$$\boldsymbol{K}_e = t \int_{-1}^{1} \int_{-1}^{1} \boldsymbol{B}^{eT} \boldsymbol{E} \boldsymbol{B}^e \left|\boldsymbol{J}_{\boldsymbol{\xi}}\right| \left|\boldsymbol{J}_{\tilde{\boldsymbol{\xi}}}\right| d\tilde{\xi} d\tilde{\eta} \tag{72}$$

$$\boldsymbol{f}_e = \int_{-1}^{1} \boldsymbol{R}^{\boldsymbol{e}T} \, \bar{\boldsymbol{t}} \left|\boldsymbol{J}_{\boldsymbol{\xi}}\right| \left|\boldsymbol{J}_{\tilde{\boldsymbol{\xi}}}\right| d\tilde{\xi} + \int_{-1}^{1} \boldsymbol{R}^{\boldsymbol{e}T} \, \boldsymbol{b} \left|\boldsymbol{J}_{\boldsymbol{\xi}}\right| \left|\boldsymbol{J}_{\tilde{\boldsymbol{\xi}}}\right| d\tilde{\xi} \tag{73}$$

Aplicando a quadratura de Gauss para a resolução das integrais:

$$\boldsymbol{K}_e = t \sum_{i=1}^{n_g} \sum_{j=1}^{m_g} \boldsymbol{B}^{eT}(\xi_i, \eta_j) \boldsymbol{E} \boldsymbol{B}^e(\xi_i, \eta_j) \left|\boldsymbol{J}_{\boldsymbol{\xi}}\right| \left|\boldsymbol{J}_{\tilde{\boldsymbol{\xi}}}\right| W_i W_j \tag{74}$$

$$\boldsymbol{f}_e = \sum_{i=1}^{n_g} \boldsymbol{R}^{\boldsymbol{e}T}(\xi_i, \eta_j) \, \bar{\boldsymbol{t}} \left|\boldsymbol{J}_{\boldsymbol{\xi}}\right| \left|\boldsymbol{J}_{\tilde{\boldsymbol{\xi}}}\right| W_i + \sum_{i=1}^{n_g} \boldsymbol{R}^{\boldsymbol{e}T}(\xi_i, \eta_j) \, \boldsymbol{b} \left|\boldsymbol{J}_{\boldsymbol{\xi}}\right| \left|\boldsymbol{J}_{\tilde{\boldsymbol{\xi}}}\right| W_i \tag{75}$$

onde ξ_i e η_j são as coordenadas paramétricas dos pontos de integração, obtidas pelo mapeamento $\tilde{\boldsymbol{\phi}}^e$. Os valores m_g e n_g são as quantidades de pontos de integração para cada direção, W_i e W_j são pesos associados e t é a espessura do elemento.

4 T-Splines

As superfícies bivariadas NURBS apresentam uma estrutura de produto tensorial intrínseca à sua origem. Essa estrutura impõe uma certa regularidade na disposição dos pontos de controle e, consequentemente, na disposição dos elementos. T-Splines surgem como uma generalização de NURBS, oferecendo uma abordagem mais flexível ao permitir a representação geométrica por meio de uma configuração alternativa [14].

As NURBS seguem uma organização de pontos de controle na forma de uma grade retangular estruturada, semelhante à técnica de mapeamento bilinear transfinito [42], empregada para geração de malhas de elementos finitos. T-Splines, por outro lado, possibilitam a formação de uma malha de controle com linhas e colunas parciais, viabilizada devido à presença de T-Junctions. A Figura 23 ilustra bem essa distinção entre NURBS e T-Splines, destacando em vermelho as T-Junctions. Uma grande vantagem das T-Splines é a capacidade de realizar refinamento local [43] e permitir a formação de malhas não estruturadas [17], aspecto que será detalhado no Capítulo 5.

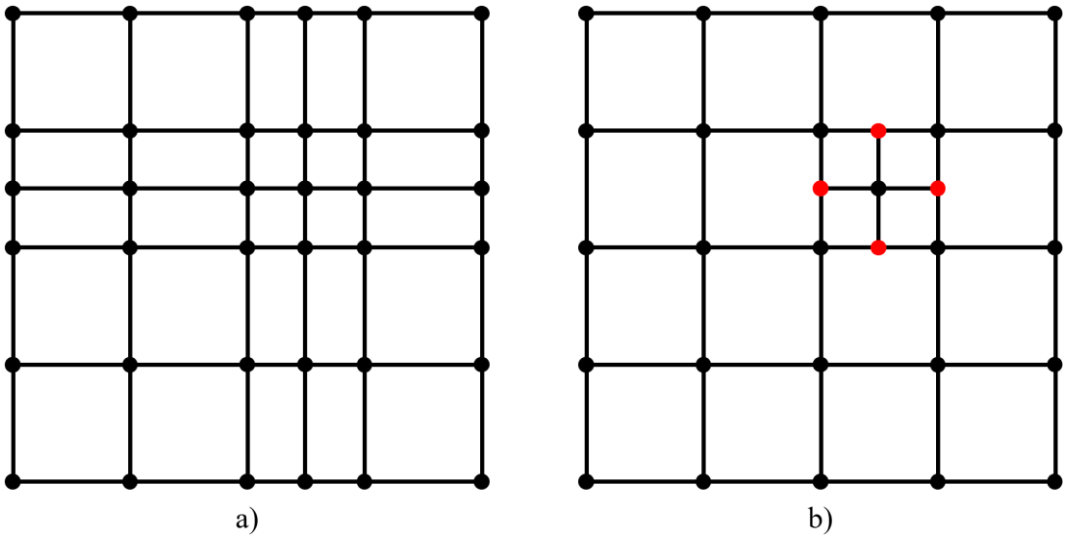

Figura 23: a) Malha de controle de uma superfície NURBS com estrutura de produto tensorial. b) Malha de controle (T-Mesh) de uma T-Spline, destacando em vermelho as T-Junctions.

Vale ressaltar que T-Splines carecem de significado físico em casos univariados (curvas). T-Splines surgem como uma alternativa à estrutura de produto

tensorial, que só é possível com ao menos duas dimensões paramétricas (superfícies). Nesse estudo, a menção à T-Spline está sempre relacionada a superfícies T-Spline.

4.1.1 T-Mesh

Uma T-Spline é formada a partir de uma T-Mesh, que consiste em uma malha de pontos de controle, semelhante à malha de controle de NURBS, mas permitindo a existência de vértices que conectam três arestas, formando T-Junctions. Fazendo uma analogia, T-Junctions são similares aos *hanging nodes* de elementos finitos [17].

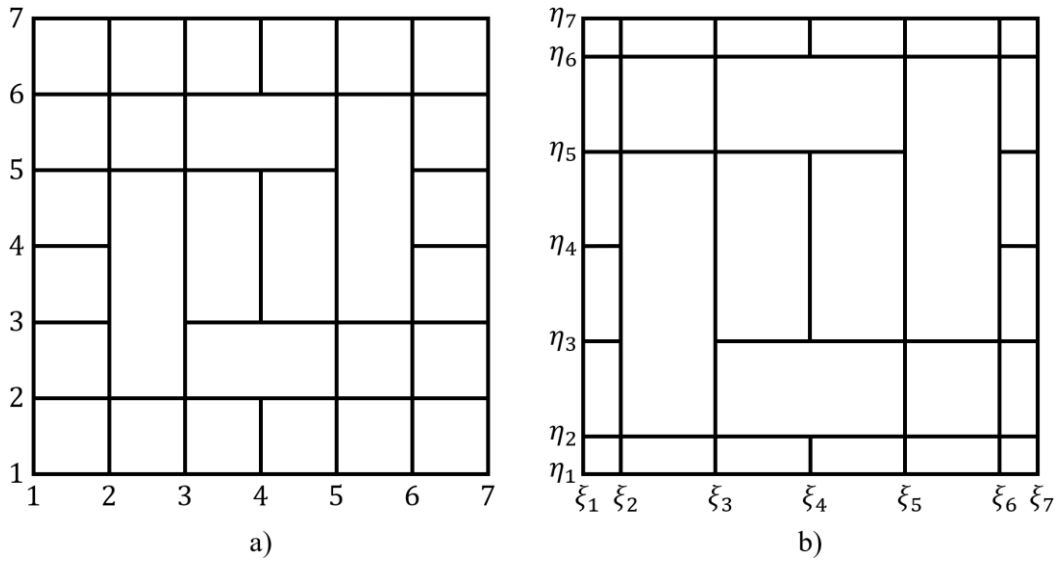

Figura 24: a) T-Mesh no espaço indicial. b) T-Mesh no espaço paramétrico.

A construção de uma T-Spline vai além da simples atribuição de uma T-Mesh. É necessário ainda fornecer informações paramétricas relativas à superfície. Isso é realizado através da configuração de uma T-Mesh no espaço indicial e no espaço paramétrico, como no exemplo mostrado na Figura 24. Uma T-Mesh no espaço indicial é marcada por números inteiros, que representam índices das chamadas linhas de *knots*. A T-Mesh no espaço paramétrico, por sua vez, é orientada nas coordenadas paramétricas ξ e η. Uma importante distinção reside no fato de que o espaço paramétrico desconsidera elementos internos de comprimento nulo, que surgem quando há valores de *knots* repetidos. Neste trabalho, o enfoque é em apenas T-Splines sem multiplicidade em *knots* internos. Assim, uma T-Mesh no

espaço paramétrico será correspondente a T-Mesh do espaço indicial, mantendo a mesma configuração de retângulos. Nesses espaços, é usual adotar um elemento nulo em cada extremidade, que desempenham um papel similar à propriedade interpolatória de um vetor de *knots* aberto.

No caso de superfícies bivariadas NURBS, as funções de base são dadas pelo produto tensorial de duas funções univariadas, cada uma definida em sua respectiva direção paramétrica. Dessa forma, dois vetores de *knots* são necessários. Em contrapartida, T-Splines têm suas funções de base (ou funções *blending*) determinadas por meio de um conjunto de vetores de *knots* locais à cada *anchor*. Esses vetores de *knots* locais são inferidos a partir da T-Mesh no espaço indicial. A expressão *blending*, usada para se referir às funções de T-Splines, se deve ao fato que nem toda T-Spline possui funções linearmente independentes, requisito essencial para a análise isogeométrica [44]. Uma discussão sobre esse tópico é abordada na Seção 4.1.9.

4.1.2 Vetor de *Knots* Local

Para a determinação dos vetores de *knots* locais, é necessário atentar inicialmente a ordem polinomial (p) da T-Spline. Isso porque os vetores de *knots* locais são determinados nos *anchor* $\{s_A\}_{A=1}^{N}$ (ver Seção 2.2.7), e a disposição desses *anchors* está relacionada com a ordem polinomial adotada. Para um p de valor par, *anchors* estão localizados nos interiores dos intervalos de *knots*; no caso de p ímpar, os *anchors* estão situados nos *knots*. Usualmente, em T-Splines, é adotado o mesmo grau polinomial para as duas direções paramétricas.

Considere o exemplo da Figura 25a, que apresenta ordem polinomial $p = 2$. Por ser um grau de valor par, o *anchor* s_α é situado no interior do elemento da T-Mesh no espaço indicial. Dessa forma, tem-se que $s_\alpha = [(0.25 + 0.75)/2\,, (0.75 + 1)/2] = [0.5, 0.875]$. Os vetores de *knots* locais de s_α são encontrados prolongando linhas verticais e horizontais nas quatro direções, até a interseção de $p/2 + 1$ linhas de *knots* ou vértices. Com isso, os intervalos entre essas interseções são tomados para a montagem desses vetores. Para s_α, os vetores de *knots* locais nas direções ξ e η são dados, respectivamente, por $\Xi_\alpha =$

$[0, 0.25, 0.75, 1]$ e $\mathcal{H}_\alpha = [0.25, 0.75, 1, 1]$. A quantidade de termos dos vetores de *knots* locais é sempre igual a $p + 2$.

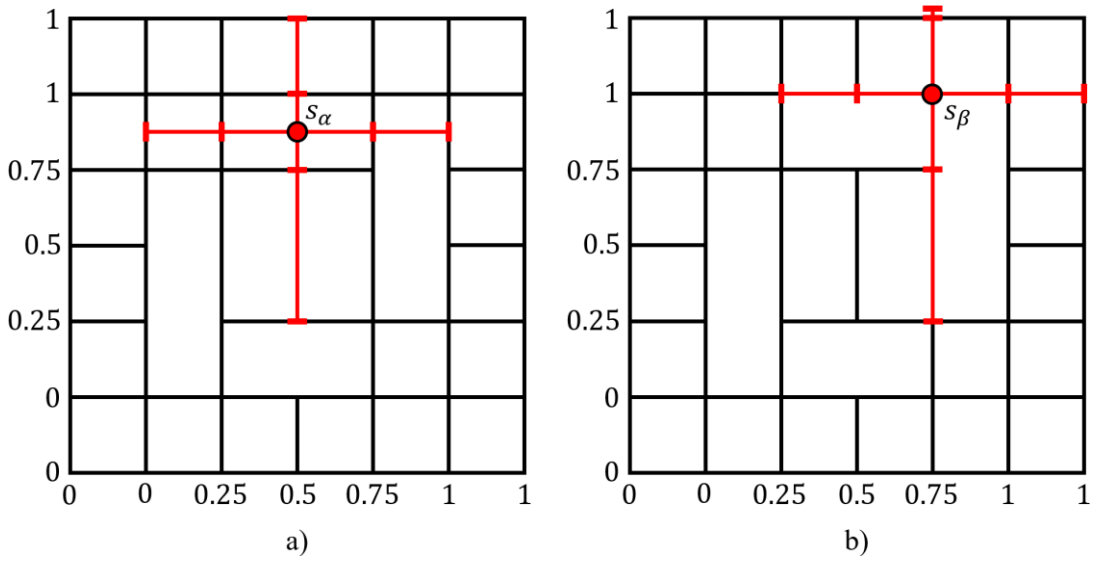

Figura 25: a) Vetores de *knots* locais do *anchor* s_α com $p = 2$. b) Vetores de *knots* de s_β para $p = 3$.

No exemplo da Figura 25b, foi assumido o valor ímpar de $p = 3$. Nesse caso, os *anchors* se localizam nos vértices dos elementos da T-Mesh do espaço indicial. Para o *anchor* $s_\beta = [0.75, 1]$, os vetores de *knots* locais são determinados a partir de $(p + 1)/2$ interseções de linhas auxiliares nas quatro direções de s_β. É relevante apontar que se as linhas auxiliares interceptarem e ultrapassarem as bordas durante esse processo, o valor do *knot* na extremidade é simplesmente replicado, como ocorreu na determinação de $\mathcal{H}_\beta$. Os vetores de *knots* locais desse exemplo são dados por $\Xi_\beta = [0.25, 0.5, 0.75, 1, 1]$ e $\mathcal{H}_\beta = [0.25, 0.75, 1, 1, 1]$. Vale notar que a quantidade de termos desses vetores de *knots* permanece $p + 2$.

4.1.3 Vetor de *Knots* Estendido

Os vetores de *knots* locais de T-Splines geralmente não são abertos. Com o objetivo de simplificar a obtenção das funções desses vetores, é utilizada uma técnica que converte o vetor de *knots* local em um vetor de *knots* estendido. O vetor de *knots* estendido é obtido realizando a repetição dos primeiros e últimos *knots* até que seja atingida multiplicidade $p + 1$, formando um vetor de *knots* aberto. Em seguida, a partir do vetor de *knots* estendido, é aplicada a formula recursiva de Cox-

de Boor [36, 37] que fornece um conjunto de funções. Apenas uma dessas funções é correspondente ao vetor de *knots* local de interesse, as demais são descartadas. Essa função é identificada por índice $n_t + 1$, onde n_t é o número de vezes que o primeiro *knot* foi adicionado para formar o vetor de *knots* estendido.

Considerando exemplo do vetor de *knots* local $\Xi_\beta = [0.25, 0.5, 0.75, 1, 1]$ e sua ordem $p = 3$, o vetor de *knots* estendido correspondente é dado por $[0.25, 0.25, 0.25, 0.25, 0.5, 0.75, 1, 1, 1, 1]$. Tem-se que $n_t = 4$, pois o *knot* inicial de valor 0.25 teve que ser repetido três vezes para a formação do vetor de *knots* estendido. Assim, a função referente ao vetor de *knots* local Ξ_β é a função de índice 4 do conjunto de funções obtidas a partir de vetor de *knots* estendido. A Figura 26 ilustra a função de base de Ξ_β, obtida pela técnica do vetor de *knots* estendido [17].

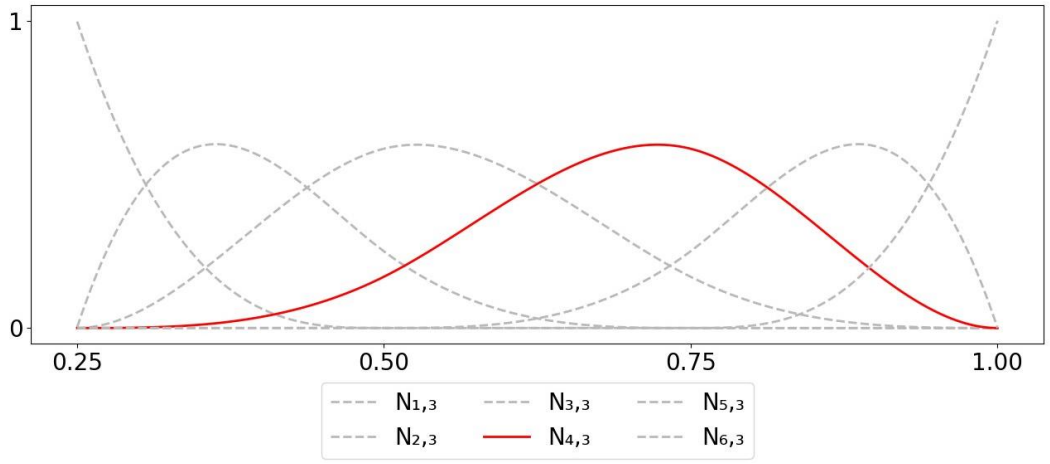

Figura 26: Funções do vetor de *knots* estendido $[0.25, 0.25, 0.25, 0.25, 0.5, 0.75, 1, 1, 1, 1]$. A função $N_{4,3}$ é correspondente ao vetor de *knots* local $[0.25, 0.5, 0.75, 1, 1]$.

4.1.4 Vetor de Intervalos de *Knots* Local

Uma vez determinados, em cada *anchor* s_A, os vetores de *knots* locais $\Xi_A = [\xi_{A,1}, \xi_{A,2}, \dots, \xi_{A,p+2}]$ e $\mathcal{H}_A = [\eta_{A,1}, \eta_{A,2}, \dots, \eta_{A,p+2}]$, são determinados os vetores de intervalos de *knots* locais $\Delta\Xi_A$ e $\Delta\mathcal{H}_A$:

$$\Delta\Xi_A = [\Delta\xi_{A,1}, \Delta\xi_{A,2}, \dots, \Delta\xi_{A,p+1}] \tag{76}$$

$$\Delta\mathcal{H}_A = [\Delta\xi_{A,1}, \Delta\xi_{A,2}, \dots, \Delta\xi_{A,p+1}] \tag{77}$$

obtidos a partir das sequências de intervalos $\Delta\xi_{A,i} = \xi_{A,i+1} - \xi_{A,i}$ e $\Delta\eta_{A,i} = \eta_{A,i+1} - \eta_{A,i}$. Para o exemplo da Figura 25a, os vetores de intervalos de *knots* locais são $\Delta\Xi_\alpha = [0.25, 0.5, 0.25]$ e $\Delta\mathcal{H}_\alpha = [0.5, 0.25, 0]$. Para a Figura 25b, são dados por $\Delta\Xi_\beta = [0.25, 0.25, 0.25, 0]$ e $\Delta\mathcal{H}_\beta = [0.5, 0.25, 0, 0]$. Os vetores de intervalos de *knots* locais desempenham papel na designação dos domínios das funções de base T-Splines, conforme explicado a seguir.

4.1.5 Funções de Base T-Spline

Para cada par de vetores de intervalos de *knots* locais $\Delta\Xi_A$ e $\Delta\mathcal{H}_A$, delimita-se na Equação (78) um domínio colocado na origem, a partir do qual as funções de base T-Spline são definidas.

$$\widehat{\Omega}_A = \left[0, \Delta\xi_{A,1} + \Delta\xi_{A,2} + \cdots + \Delta\xi_{A,p+1}\right] \otimes \left[0, \Delta\eta_{A,1} + \Delta\eta_{A,2} + \cdots + \Delta\eta_{A,p+1}\right] \quad (78)$$

Para os exemplos anteriores, os seguintes domínios são determinados: $\widehat{\Omega}_\alpha = [0, 1] \otimes [0, 0.75]$ e $\widehat{\Omega}_\beta = [0, 0.75] \otimes [0, 0.75]$.

A partir de cada domínio $\widehat{\Omega}_A$, define-se uma função de base T-Spline bivariada $R_A(\xi, \eta)$: $\widehat{\Omega}_A \to \mathbb{R}$, dada pela equação:

$$R_A(\xi, \eta) = \frac{N_A(\xi) M_A(\eta) w_A}{\sum_{\hat{A}=1}^{n} N_{\hat{A}}(\xi) M_{\hat{A}}(\eta) w_{\hat{A}}} \quad (79)$$

onde $N_A(\xi)$ e $M_A(\eta)$ são as funções univariadas, referentes respectivamente aos vetores de *knots* locais Ξ_A e $\mathcal{H}_A$, calculadas pela técnica do vetor de *knots* estendido (Seção 4.1.3). E w_A é o peso do ponto de controle P_A associado ao *anchor* s_A.

4.1.6 T-Spline Bicúbica e Configuração de Intervalos de *Knots*

Em T-Splines bicúbicas, os *anchors* estão sempre situados nos vértices da T-Mesh no espaço indicial. As funções de base, que são definidas nesses *anchors*, possuem uma relação direta com eles: para cada função de base, existe um *anchor* correspondente. Além disso, uma determinada função de base é associada a um ponto de controle específico, o que significa que para cada *anchor*, existe um ponto

de controle correlato. Portanto, na estrutura das T-Splines bicúbicas, a disposição dos vértices da T-Mesh no espaço indicial se assemelha à disposição dos pontos de controle da T-Mesh. Softwares de modelagem geométrica como Maya e Rhino operam com T-Splines bicúbicas, evidenciando o interesse comercial dessas entidades [13]. Neste trabalho, são consideradas apenas T-Splines bicúbicas, implementadas na própria ferramenta desenvolvida. Na Figura 27a, é mostrado o exemplo da T-Mesh de uma T-Spline bicúbica; os pontos destacados pela cor preta representam os pontos de controle regulares; de cor vermelha, estão os pontos de controle que formam T-Junctions. Na Figura 27b, é ilustrado a T-Mesh no espaço indicial referente a mesma T-Spline bicúbica.

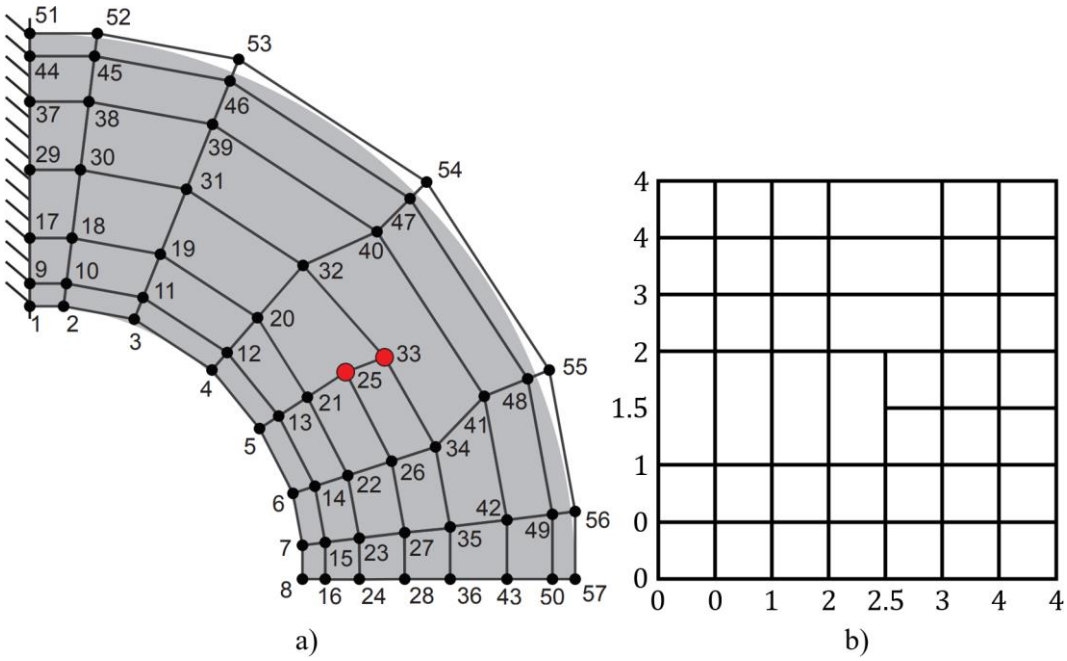

Figura 27: a) T-Mesh de uma T-Spline bicúbica [17]. b) T-Mesh no espaço indicial referente à T-Spline bicúbica.

A Figura 27a e a Figura 27b podem ser condensadas em uma representação mais compacta, conhecida como configuração de intervalos de *knots*, conforme a Figura 28. Essa representação é similar a malha de controle da T-Mesh, mas as arestas apresentam a indicação de valores de intervalo paramétricos, que definem os elementos da T-Mesh no espaço indicial. Nessa representação, pontos de controle e *anchors* são localizados no mesmo lugar. É relevante apontar que essa configuração não se aplica para T-Splines biquadráticas por exemplo. Nesse caso, a ordem polinomial de valor par acarretaria em *anchors* localizados no interior dos

elementos da T-Mesh no espaço indicial. Assim, a T-Mesh apresentaria arestas sem correspondência com as delimitações dos elementos do espaço indicial, tornando a representação de intervalos de *knots* na T-Mesh inadequada.

Na Figura 28, os símbolos triangulares situados nas extremidades representam um intervalo entre *knots* de 0, os quadrados representam valor de 0.5 e os pentágonos valor 1. Uma restrição importante é que os lados opostos de cada elemento devem apresentar a mesma soma de intervalos de *knots*, conforme mostrado no elemento destacado.

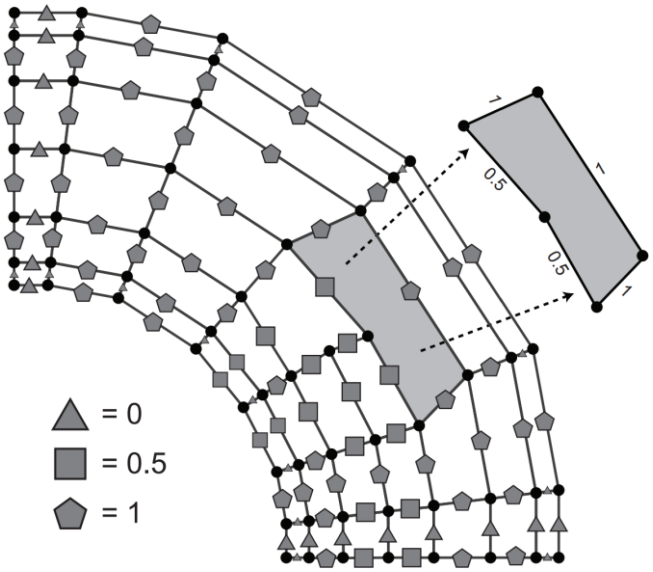

Figura 28: Configuração de intervalos de *knots* de uma T-Spline [17].

A partir de uma configuração de intervalos de *knots*, é possível prosseguir para a determinação das funções T-Splines. No exemplo exposto na Figura 29, é demonstrado esse procedimento para um determinado *anchor*. Inicialmente, o vetor de *knots* local é estipulado com base nos intervalos de *knots*, de forma semelhante ao processo da Seção 4.1.2. Em seguida, esse vetor de *knots* é inferido na T-Mesh no espaço indicial, a partir do qual é definido o domínio das funções. Com isso, parte-se para a obtenção das funções T-Splines pela Equação (79).

4.1.7 T-Mesh Estendida

Nas seções anteriores, o conceito de T-Mesh foi introduzido, sendo definida pela rede de pontos de controle. As T-Meshes no espaço indicial e no paramétrico

foram discutidas, neste estudo ambas são correspondentes e fornecem informações paramétricas para a T-Spline. Ressalta-se que os elementos da T-Mesh (rede de pontos de controle) são dados pelos quadriláteros da malha de controle, enquanto os elementos da T-Mesh no espaço indicial são os quadriláteros delimitados pelas linhas de *knots*. Na análise isogeométrica, o interesse principal reside nos elementos da T-Spline, pois são neles onde ocorre a integração numérica. É importante notar que, de modo geral, não existe uma correspondência de cada elemento da T-Spline com os elementos da T-Mesh ou elementos da T-Mesh no espaço indicial. Os elementos das T-Splines estão contidos em regiões onde a continuidade das funções de base é C^{∞}. Essas regiões são delimitadas por linhas de continuidade reduzida, que surgem através de extensões das T-Junctions e geralmente são representadas por linhas tracejadas. A T-Mesh que inclui todas as linhas de continuidade reduzida é conhecida como T-Mesh estendida.

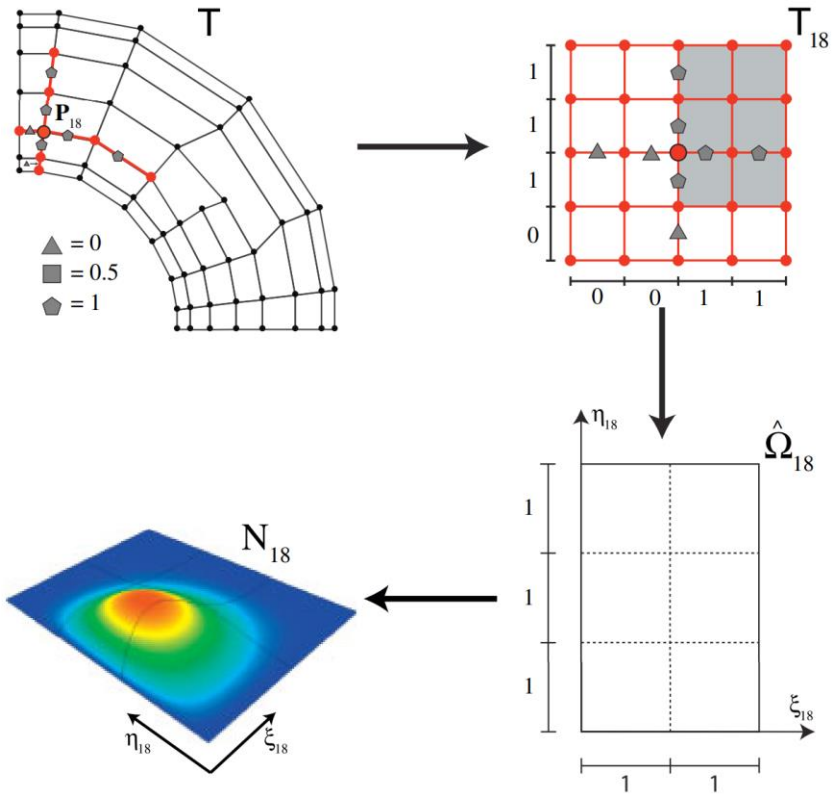

Figura 29: Obtenção da função T-Splines em um dado *anchor*, a partir de uma configuração de intervalos de *knots* [17].

Para encontrar as linhas de continuidade reduzida, utilizam-se os vetores de *knots* locais em cada *anchor*. Toma-se como exemplo a T-Mesh no espaço indicial da Figura 27b com $p = 3$. Na Figura 30 um determinado *anchor* é destacado e seu

vetor de *knots* local deduzido seguindo o procedimento da Seção 4.1.2. O suporte da função T-Spline referente a esse *anchor* consiste na região abrangida pelo vetor de *knots* local. Essa função apresenta uma continuidade reduzida nas linhas de *knots* referentes a cada componente do vetor de *knots* local, marcadas em vermelho na Figura 30. Se essas linhas de *knots* estiverem incompletas na área de suporte, são realizadas as suas respectivas extensões. A T-Mesh estendida no espaço indicial é obtida repetindo esse processo em todos os *anchors*. Em casos de ordem polinomial ímpar, a T-Mesh estendida pode ser determinada utilizando uma técnica mais simples, bastando apenas realizar as extensões partindo das T-Junctions, até que $(p+1)/2$ linhas de *knots* ortogonais sejam alcançadas.

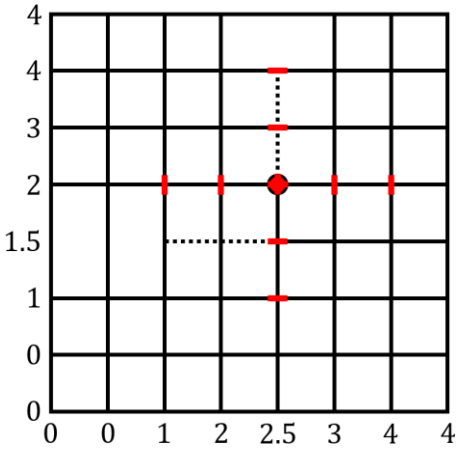

Figura 30: Determinação das linhas de continuidade reduzida na T-Mesh do espaço indicial.

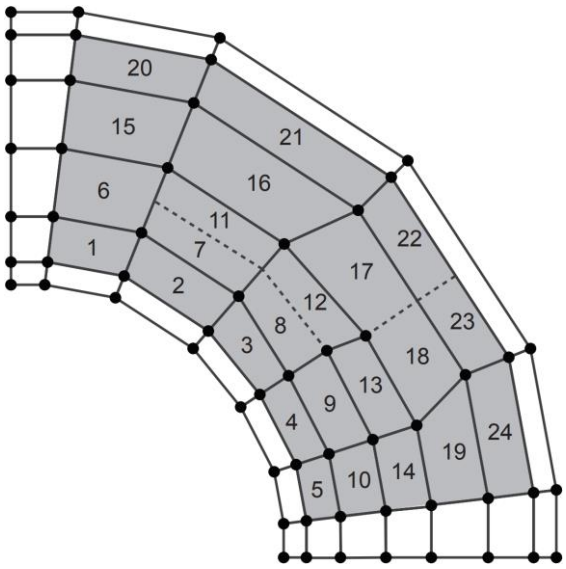

Figura 31: Elementos T-Spline em uma T-Mesh estendida [17].

Em T-Splines bicúbicas, linhas de continuidade reduzida podem ser representadas diretamente na T-Mesh. Na Figura 31 tem-se a T-Mesh estendida referente ao exemplo da Figura 30. Para cada elemento da T-Mesh estendida, excluindo os elementos de extremidade, há um correspondente elemento T-Spline.

4.1.8 Conectividade

A conectividade de elementos T-Spline é inferida da seguinte maneira: verifica-se quais elementos da T-Mesh estendida estão contidos no domínio de cada função de base. Recorda-se que para o caso cúbico, cada função de base é vinculada a um ponto de controle específico. Recorda-se também que o domínio da função é inferido estendendo o ponto de controle em $(p+1)/2 = 2$ interseções de segmentos da T-Mesh. Isso é ilustrado com mais clareza na Figura 32, onde o ponto de controle em destaque faz conectividade com os elementos em cor cinza da T-Mesh estendida.

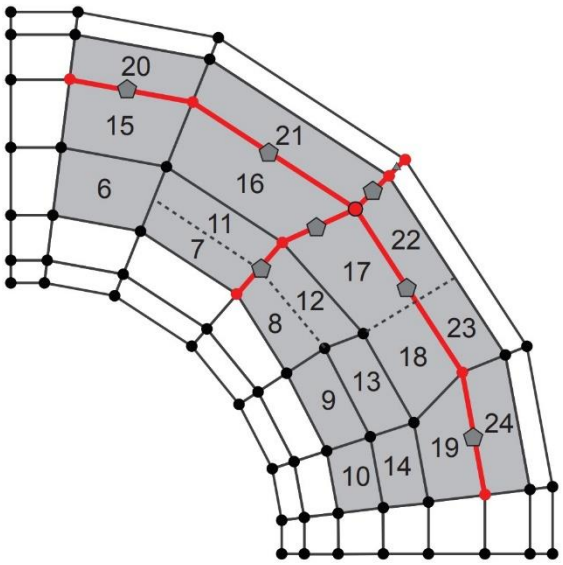

Figura 32: Elementos da T-Mesh estendida no suporte do ponto de controle destacado [17].

Esse processo precisa ser feito em todos os pontos de controle. Na Figura 33, observa-se os pontos de controle de conectividade com o elemento 10. Diferente de superfícies NURBS, onde a quantidade de pontos de controle que fazem conectividade é definida por $n_{en} = (p+1)(q+1)$, em T-Splines esse número pode variar de acordo com o elemento analisado.

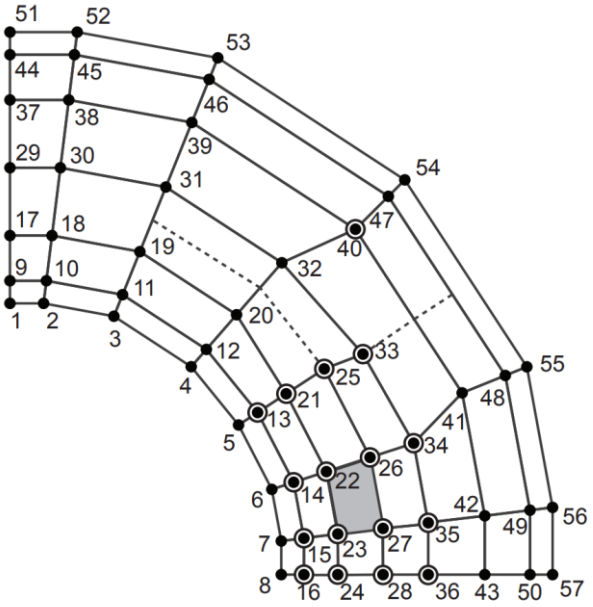

Figura 33: Conectividade em T-Splines [17].

4.1.9 T-Splines Adequadas para Análise

A simples definição de uma T-Mesh e uma configuração de intervalos de *knots* não torna a T-Spline adequada para análise sem uma investigação prévia. Uma T-Spline adequada para análise é definida como sendo aquela cujas funções são linearmente independentes. A independência linear das funções ocorre quando não há interseções das extensões das T-Junctions [44]. As extensões das T-Junctions são compostas por extensões de face e extensões de segmento. As extensões de face são as linhas de continuidade reduzida da Seção 4.1.7; as extensões de segmentos são linhas traçadas no sentido oposto até o encontro da primeira linha de *knots*.

Na Figura 34, ilustra-se as extensões das T-Junctions em dois exemplos. No primeiro, observa-se interseções entre essas extensões, tornando a T-Mesh inadequada para análise. No segundo exemplo, extensões das T-Junctions não se interceptam, o que indica que a T-Mesh é adequada para análise.

Nesse sentido, T-Splines podem parecer restritivas, mas sua inspeção é essencial para garantir a independência linear em qualquer escolha de *knots* [44]. O termo “funções *blending*” é comumente usado para se referir às funções de T-Splines. Isso ocorre porque uma T-Spline pode não ser adequada para a análise e

suas funções serem linearmente dependentes, o que leva a matrizes singulares. Uma vez que garantida que a T-Spline é adequada para análise, o termo “funções de base” é adotado.

Portanto, não é garantido que uma T-Mesh arbitrária seja sempre adequada para análise. Sem a devida inspeção, conforme discutido acima, é possível se deparar com situações em que a T-Mesh não é adequada para análise.

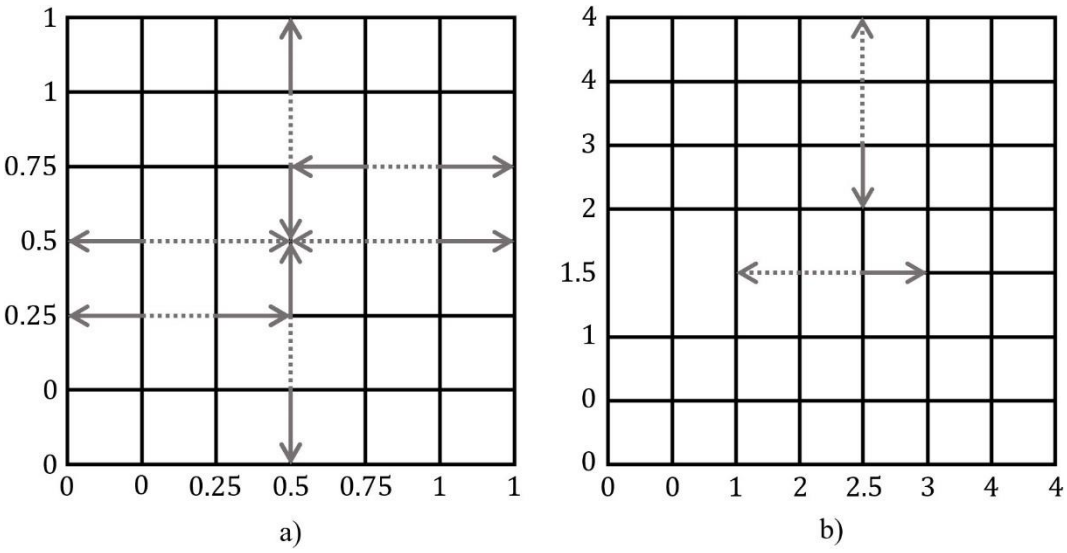

Figura 34: Extensões das T-Junctions. a) T-Mesh não adequada para análise. b) T-Mesh adequada para análise.

5 Extração Bézier de NURBS e T-Splines

A extração Bézier é uma técnica que permite a determinação de funções NURBS e T-Splines em termos de funções polinomiais de Bernstein. Operadores de extração Bézier são computados para realizar a transformação entre essas funções. Essa técnica envolve a decomposição de NURBS e T-Splines em elementos Bézier de continuidade C^0, que possuem uma representação local das funções de base. A extração Bézier proporciona uma estrutura que facilita a implementação da análise isogeométrica.

Os conceitos discutidos neste capítulo são baseados nos trabalhos de Borden et al. [16], que se concentraram na extração Bézier de NURBS, e Scott et al. [15], que exploraram a extração Bézier de T-Splines. Essas contribuições formam a base do entendimento atual da extração Bézier e seu papel na análise isogeométrica.

5.1 Extração Bézier de NURBS

Inicialmente, o foco será direcionado para a extração Bézier de NURBS. É apresentado conceito preliminar da determinação dos polinômios de Bernstein. Em seguida, será demonstrada a ideia do processo de decomposição de Bézier, bem como o método para calcular os operadores de extração, que são fundamentais nesse processo. Será discutido como NURBS e suas funções de base podem ser determinadas pela aplicação dos operadores de extração em elementos Bézier e polinômios de Bernstein. Por fim, esses conceitos serão expandidos para o caso bivariado.

5.1.1 Polinômios de Bernstein

Um elemento Bézier univariado é definido geralmente no domínio paramétrico [0,1], sendo formado a partir de um vetor de *knots* aberto, com $(p+1)$ componentes iniciais de valor zero e $(p+1)$ componentes de valor unitário. As funções de base referentes a esse elemento Bézier recebem o nome de polinômios de Bernstein. No contexto de extração Bézier, define-se um elemento Bézier contido no domínio paramétrico $[-1,1]$, que consiste no mesmo domínio das

funções de forma paramétricas de elementos finitos, como também, no mesmo domínio do espaço *parent*, eliminando a necessidade do mapeamento $\tilde{\boldsymbol{\phi}}^e$. Aplicando as Equações (5) e (6) no vetor de *knots* composto por $(p+1)$ valores -1, e $(p+1)$ valores 1, pode-se determinar os polinômios de Bernstein. De forma alternativa, realiza-se também essa determinação pela equação mais compacta abaixo:

$$B_{i,p}(\xi) = \frac{1}{2}(1-\xi)B_{i,p-1}(\xi) + \frac{1}{2}(1+\xi)B_{i-1,p-1}(\xi) \tag{80}$$

onde

$$B_{1,0}(\xi) \equiv 1 \quad \text{e} \quad B_{i,p}(\xi) \equiv 0 \ \text{ se } \ i < 1 \text{ ou } i > p+1 \tag{81}$$

Ilustrado na Figura 35, observa-se as funções polinomiais de Bernstein de ordem $p = 2$ e $p = 3$.

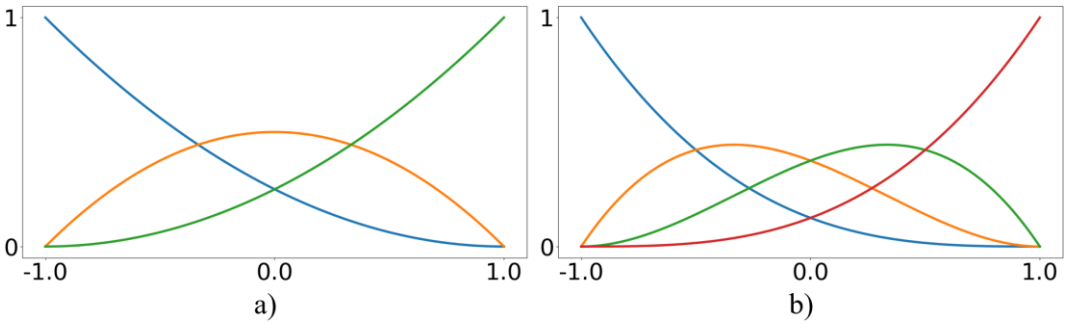

Figura 35: a) Polinômios de Bernstein quadráticos. b) Polinômios de Bernstein cúbicos.

A primeira derivada dos polinômios de Bernstein é dada por:

$$\frac{d}{d\xi}B_{i,p}(\xi) = \frac{p}{2}\Big(B_{i-1,p-1}(\xi) - B_{i,p-1}(\xi)\Big) \tag{82}$$

De forma análoga a B-Splines, uma curva Bézier pode ser definida por uma combinação linear entre os polinômios de Bernstein e uma série de pontos de controle:

$$\boldsymbol{C}^b(\xi) = \sum_{i=1}^{p+1} B_{i,p}(\xi)\boldsymbol{P}_i^b \tag{83}$$

5.1.2 Decomposição de Bézier

Uma NURBS pode ser expressa na forma de elementos Bézier, unidos de ponta a ponta. O método de decomposição de Bézier é utilizado para obter os elementos de Bézier que formam uma determinada NURBS. Esse processo consiste em aplicar a inserção de *knots* até que todos os *knots* internos do vetor de *knots* tenham multiplicidade p.

A decomposição de Bézier é melhor explicada através do exemplo disposto na Figura 36. A Figura 36a exibe as funções de base cúbicas do vetor de *knots* $\Xi_1 = [0, 0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1, 1]$. Na Figura 36b, é disposta uma curva B-Spline construída a partir das funções de base em questão. Para a decomposição Bézier ser concluída, é necessário inserir *knots* em Ξ_1, formando o vetor de *knots* $\Xi_2 = [0, 0, 0, 0, 0.25, 0.25, 0.25, 0.5, 0.5, 0.5, 0.75, 0.75, 0.75, 1, 1, 1, 1]$. Ilustrado na Figura 37, observa-se a sequência de novas funções de base para cada vez que um *knot* é inserido. Como resultado final, são obtidas funções de continuidade C^0 nos valores de *knots*. Percebe-se que em cada elemento, existem $p + 1$ funções que se assemelham com os polinômios de Bernstein cúbicos da Figura 35b. Na Figura 38, apresenta-se a sequência de curvas B-Spline. Sempre que é realizada uma operação de inserção de *knot*, novos pontos de controle são calculados pelas Equações (18) e (19). Embora um novo polígono de controle tenha sido formado, a geometria e a continuidade da curva original permanecem inalteradas. Um ponto importante deve ser ressaltado: diferente do que pode parecer intuitivo, a existência de continuidade C^0 nas funções de base, não implica necessariamente em continuidade C^0 em regiões da curva B-Spline.

5.1.3 Operadores de Extração

Conforme exemplificado anteriormente, a decomposição Bézier se baseia no refinamento por inserção de *knots* abordado na Seção 2.2.8.2, onde a finalidade é determinar um novo conjunto de pontos de controle $\overline{\boldsymbol{P}} = \{\overline{\boldsymbol{P}}_i\}_{i=1}^{n+m}$em função dos pontos de controle originais $\boldsymbol{P} = \{\boldsymbol{P}_i\}_{i=1}^{n}$. Nesse sentido, é conveniente expressar essa operação em forma matricial e, como resultado, chega-se na definição de um operador de extração.

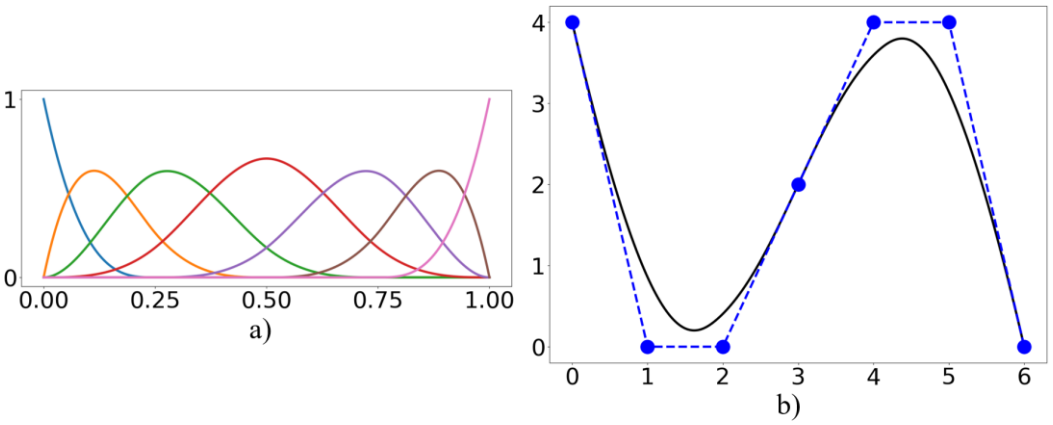

Figura 36: a) Funções de base do vetor de *knots* $\Xi = [0, 0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1, 1]$ de ordem $p = 3$. b) Curva B-Spline.

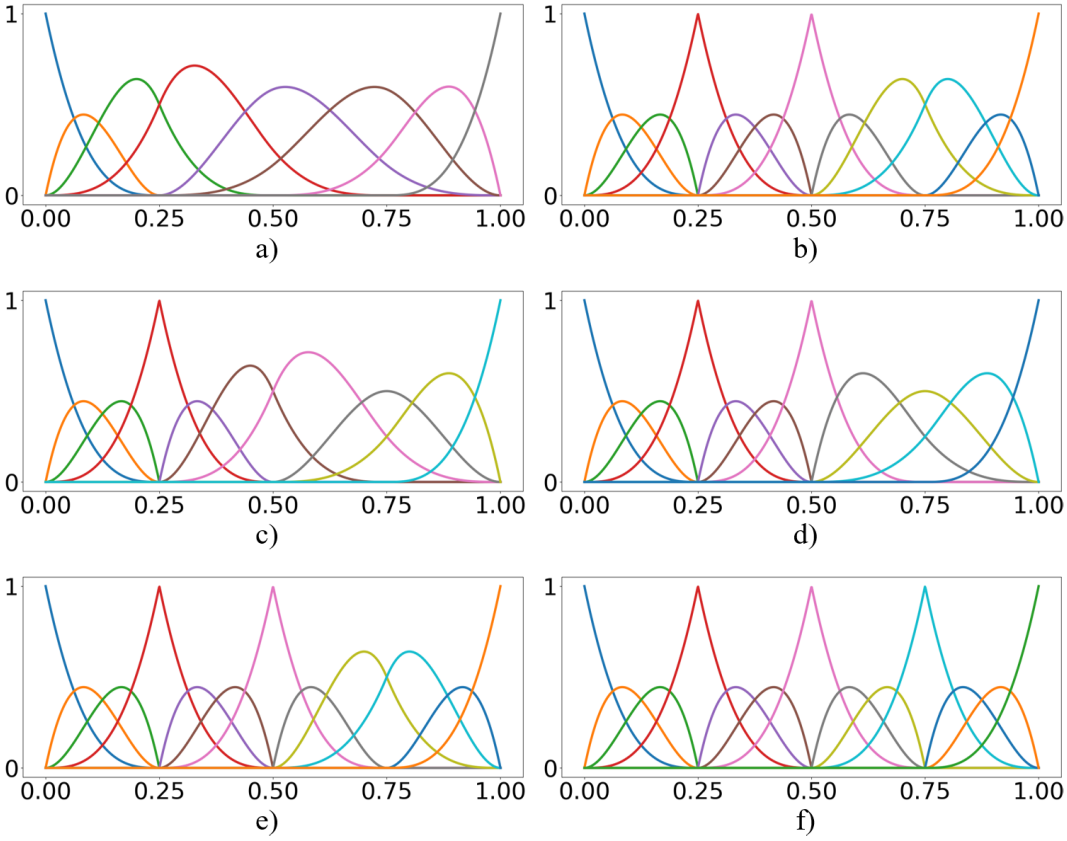

Figura 37: Sequência de funções de base referentes ao processo de decomposição Bézier envolvendo o vetor de *knots* $\Xi_1 = [0, 0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1, 1]$.

Dado um vetor de *knots* $\Xi = \{\xi_1, \xi_2, \dots, \xi_{n+p+1}\}$ e seja $\bar{\Xi} = \{\bar{\xi}_1, \bar{\xi}_2, \dots, \bar{\xi}_m\}$ uma série de *knots* a serem inseridos para realizar a decomposição Bézier de uma B-Spline, determina-se para cada $\bar{\xi}_j, j = 1, 2, \dots, m$ um conjunto de coeficientes α_i^j,

$i = 1, 2, \dots, n + j$ definidos pela Equação (19). Seguidamente, define-se uma matriz $\boldsymbol{C}^j$:

$$\boldsymbol{C}^j = \begin{bmatrix} \alpha_1 & 1-\alpha_2 & 0 & \cdots & & & 0 \\ 0 & \alpha_2 & 1-\alpha_3 & 0 & \cdots & & 0 \\ 0 & 0 & \alpha_3 & 1-\alpha_4 & 0 & \cdots & 0 \\ \vdots & & & & & & \\ 0 & \cdots & & & 0 & \alpha_{n+j-1} & 1-\alpha_{n+j} \end{bmatrix} \quad (84)$$

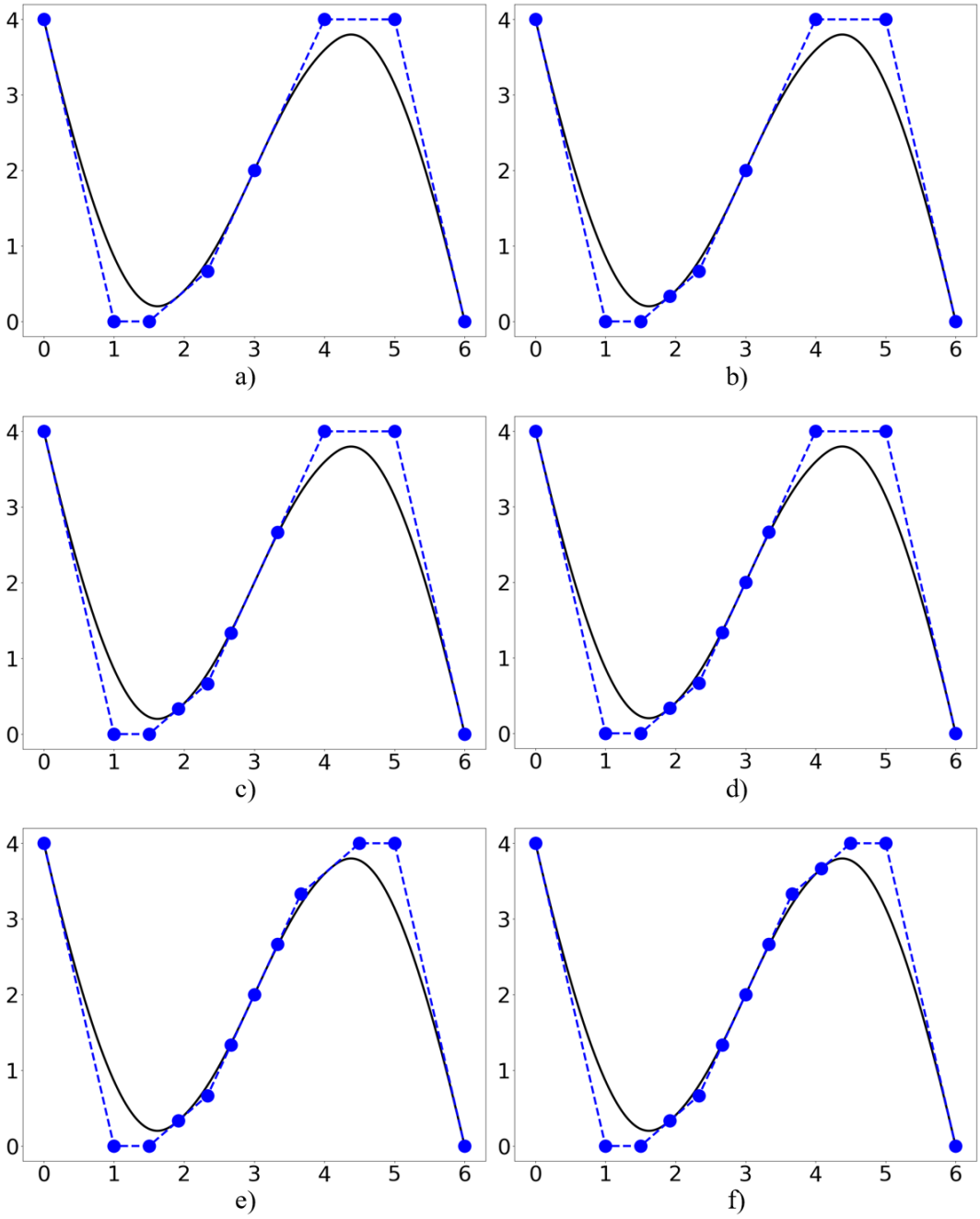

Figura 38: Sequência de curvas B-Spline e polígonos de controle referentes ao processo de decomposição Bézier da B-Spline formada a partir do vetor de *knots* $\Xi_1 = [0, 0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1, 1]$.

Com isso, a Equação (18) pode ser reescrita matricialmente:

$$\overline{\boldsymbol{P}}^{j+1} = \left(\boldsymbol{C}^{j}\right)^{T} \overline{\boldsymbol{P}}^{j} \tag{85}$$

sendo $\overline{\boldsymbol{P}}^{1}$ equivalente aos pontos de controle iniciais $\boldsymbol{P}$. O conjunto final de pontos de controle decorrentes da decomposição Bézier $\overline{\boldsymbol{P}}^{m+1} = \boldsymbol{P}^{b}$ é obtido mediante o produto de sucessivas matrizes $\left(\boldsymbol{C}^{j}\right)^{T}$:

$$\boldsymbol{C}^{T} = (\boldsymbol{C}^{m})^{T}(\boldsymbol{C}^{m-1})^{T} \dots (\boldsymbol{C}^{1})^{T} \tag{86}$$

assim,

$$\boldsymbol{P}^{b} = \boldsymbol{C}^{T}\boldsymbol{P} \tag{87}$$

Pode-se definir uma curva B-Spline em termos de funções de base Bernstein $\{B(\xi)\}_{i=1}^{n+m}$ e dos novos pontos de controle $\{\boldsymbol{P}^{b}\}_{i=1}^{n+m}$. Em notação matricial, tem-se que:

$$\boldsymbol{C}(\xi) = \boldsymbol{P^{b}}^{T}\boldsymbol{B}(\xi)$$

$$\boldsymbol{C}(\xi) = \begin{bmatrix} P_{x\,1} & P_{x\,2} & \cdots & P_{x\,n+m} \\ P_{y\,1} & P_{y\,2} & \cdots & P_{y\,n+m} \end{bmatrix} \begin{bmatrix} B_{1,p}(\xi) \\ B_{2,p}(\xi) \\ \vdots \\ B_{n+m,p}(\xi) \end{bmatrix} \tag{88}$$

A Equação (11), de curvas B-Splines, em forma matricial, é dada por:

$$\boldsymbol{C}(\xi) = \boldsymbol{P}^{T}\boldsymbol{N}(\xi)$$

$$\boldsymbol{C}(\xi) = \begin{bmatrix} P_{x\,1} & P_{x\,2} & \cdots & P_{x\,n} \\ P_{y\,1} & P_{y\,2} & \cdots & P_{y\,n} \end{bmatrix} \begin{bmatrix} N_{1,p}(\xi) \\ N_{2,p}(\xi) \\ \vdots \\ N_{n,p}(\xi) \end{bmatrix} \tag{89}$$

Ressalta-se que a decomposição Bézier, realizada por inserção de *knots*, não causa modificação na geometria da curva. Dessa forma, as Equações (89) e (88) são igualadas:

$$\boldsymbol{P}^{T}\boldsymbol{N}(\xi) = \boldsymbol{P^{b}}^{T}\boldsymbol{B}(\xi) \tag{90}$$

substituindo a Equação (87) em (90):

$$\begin{aligned} \boldsymbol{P}^T\boldsymbol{N}(\xi) &= (\boldsymbol{C}^T\boldsymbol{P})^T\boldsymbol{B}(\xi) \\ \boldsymbol{P}^T\boldsymbol{N}(\xi) &= \boldsymbol{P}^T\boldsymbol{C}\boldsymbol{B}(\xi) \end{aligned} \tag{91}$$

chega-se na relação entre as funções de base B-Spline e as funções Bernstein pela equação:

$$\boldsymbol{N}(\xi) = \boldsymbol{C}\boldsymbol{B}(\xi) \tag{92}$$

A matriz $\boldsymbol{C}$ é denominada operador de extração, sendo de grande valia para determinação das funções de base, uma vez que as funções Bernstein são facilmente calculadas. Observa-se que a única informação necessária para a construção desse operador é o vetor de *knots*.

Para obter e relação entre funções NURBS e funções Bernstein, a Equação (22) é escrita matricialmente:

$$\begin{aligned} \boldsymbol{R}(\xi) &= \frac{1}{W(\xi)}\boldsymbol{W}\boldsymbol{N}(\xi) \\ \boldsymbol{R}(\xi) &= \frac{1}{W(\xi)}\begin{bmatrix} w_1 & 0 & 0 & 0 \\ 0 & w_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & w_n \end{bmatrix}\begin{bmatrix} N_{1,p}(\xi) \\ N_{2,p}(\xi) \\ \vdots \\ N_{n,p}(\xi) \end{bmatrix} \end{aligned} \tag{93}$$

onde $W(\xi)$ é a função peso, dada por $W(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi)w_i$, e $\boldsymbol{W}$ uma matriz diagonal de pesos. A função peso pode ser reescrita pela relação apresentada pela Equação (94):

$$\begin{aligned} W(\xi) &= \boldsymbol{w}^T\boldsymbol{N}(\xi) \\ W(\xi) &= \boldsymbol{w}^T\boldsymbol{C}\boldsymbol{B}(\xi) \\ W(\xi) &= (\boldsymbol{C}^T\boldsymbol{w})^T\boldsymbol{B}(\xi) \\ W(\xi) &= \left(\boldsymbol{w}^{\boldsymbol{b}}\right)^T\boldsymbol{B}(\xi) \\ W(\xi) &= W^b(\xi) \end{aligned} \tag{94}$$

onde $\boldsymbol{w}$ é o vetor coluna de pesos associados aos pontos de controle $\boldsymbol{P}$, $\boldsymbol{w}^b = \boldsymbol{C}^T\boldsymbol{w}$ é o vetor coluna de pesos dos pontos de controle Bézier $\boldsymbol{P}^b$. A função $W^b(\xi)$ é função peso obtida pela combinação linear entre as funções Bernstein e os pesos associados $\boldsymbol{P}^b$. Conforme provado, $W^b(\xi)$ e $W(\xi)$ são equivalentes.

Empregando a função $W^b(\xi)$ e a Equação (92), tem-se a expressão que relaciona as funções Bernstein e as funções de base NURBS:

$$\boldsymbol{R}(\xi) = \frac{1}{W^b(\xi)}\boldsymbol{WCB}(\xi) \tag{95}$$

Os pontos de controle Bézier de curvas NURBS podem ser determinados seguindo o procedimento descrito na Seção 2.3.1. É realizada a transformação para os pontos de controle projetivos no $\mathbb{R}^{d_s+1}$, aplica-se o operador de extração e efetua-se a transformação de volta para o $\mathbb{R}^{d_s}$. Esse processo pode ser feito de maneira mais compacta pela expressão:

$$\boldsymbol{P}^b = (\boldsymbol{W}^b)^{-1}\boldsymbol{C}^T\boldsymbol{WP} \tag{96}$$

onde $\boldsymbol{W}^b$ é a matriz diagonal com os pesos dos pontos de controle Bézier. Multiplicando a Equação (96) por $\boldsymbol{W}^b$:

$$\boldsymbol{W}^b\boldsymbol{P}^b = \boldsymbol{C}^T\boldsymbol{WP} \tag{97}$$

Com base nas equações (95) e (97), define-se uma curva NURBS em termos de funções Bernstein e pontos de controle Bézier:

$$\begin{aligned}
\boldsymbol{C}(\xi) &= \boldsymbol{P}^T\boldsymbol{R}(\xi) \\
\boldsymbol{C}(\xi) &= \frac{1}{W^b(\xi)}\boldsymbol{P}^T\boldsymbol{WCB}(\xi) \\
\boldsymbol{C}(\xi) &= \frac{1}{W^b(\xi)}(\boldsymbol{C}^T\boldsymbol{WP})^T\boldsymbol{B}(\xi) \\
\boldsymbol{C}(\xi) &= \frac{1}{W^b(\xi)}(\boldsymbol{W}^b\boldsymbol{P}^b)^T\boldsymbol{B}(\xi)
\end{aligned} \tag{98}$$

5.1.4 Operadores de Extração Locais

No exemplo de decomposição Bézier da Figura 37, é possível determinar o operador de extração pelas Equações (84) e (86). Aplicando esse operador na Equação (92):

$$\begin{bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \\ N_5 \\ N_6 \\ N_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{7}{12} & \frac{2}{3} & \frac{1}{3} & \frac{1}{6} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{6} & \frac{1}{3} & \frac{2}{3} & \frac{2}{3} & \frac{2}{3} & \frac{1}{3} & \frac{1}{6} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{6} & \frac{1}{3} & \frac{2}{3} & \frac{7}{12} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \\ B_7 \\ B_8 \\ B_9 \\ B_{10} \\ B_{11} \\ B_{12} \\ B_{13} \end{bmatrix} \tag{99}$$

A decomposição Bézier tem como resultado a formação de elementos Bézier em cada intervalo de *knots*. Para o primeiro intervalo $[0, 0.25)$, as funções de base N_1, N_2, N_3 e N_4 podem ser representadas como uma combinação linear das funções Bernstein B_1, B_2, B_3 e B_4:

$$\begin{bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{bmatrix} = \begin{bmatrix} N_1^1 \\ N_2^1 \\ N_3^1 \\ N_4^1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \frac{1}{2} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{6} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \frac{1}{2} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{6} \end{bmatrix} \begin{bmatrix} B_1^1 \\ B_2^1 \\ B_3^1 \\ B_4^1 \end{bmatrix} \tag{100}$$

Os índices sobrescritos indicam o elemento em questão. Para os intervalos $[0.25, 0.5)$, $[0.5, 0.75)$ e $[0.75, 1)$:

$$\begin{bmatrix} N_2 \\ N_3 \\ N_4 \\ N_5 \end{bmatrix} = \begin{bmatrix} N_1^2 \\ N_2^2 \\ N_3^2 \\ N_4^2 \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & 0 & 0 & 0 \\ \frac{7}{12} & \frac{2}{3} & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} & \frac{2}{3} & \frac{2}{3} \\ 0 & 0 & 0 & \frac{1}{6} \end{bmatrix} \begin{bmatrix} B_4 \\ B_5 \\ B_6 \\ B_7 \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & 0 & 0 & 0 \\ \frac{7}{12} & \frac{2}{3} & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} & \frac{2}{3} & \frac{2}{3} \\ 0 & 0 & 0 & \frac{1}{6} \end{bmatrix} \begin{bmatrix} B_1^2 \\ B_2^2 \\ B_3^2 \\ B_4^2 \end{bmatrix} \tag{101}$$

$$\begin{bmatrix} N_3 \\ N_4 \\ N_5 \\ N_6 \end{bmatrix} = \begin{bmatrix} N_1^3 \\ N_2^3 \\ N_3^3 \\ N_4^3 \end{bmatrix} = \begin{bmatrix} \frac{1}{6} & 0 & 0 & 0 \\ \frac{2}{3} & \frac{2}{3} & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} & \frac{2}{3} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} B_7 \\ B_8 \\ B_9 \\ B_{10} \end{bmatrix} = \begin{bmatrix} \frac{1}{6} & 0 & 0 & 0 \\ \frac{2}{3} & \frac{2}{3} & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} & \frac{2}{3} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} B_1^3 \\ B_2^3 \\ B_3^3 \\ B_4^3 \end{bmatrix} \quad (102)$$

$$\begin{bmatrix} N_4 \\ N_5 \\ N_6 \\ N_7 \end{bmatrix} = \begin{bmatrix} N_1^4 \\ N_2^4 \\ N_3^4 \\ N_4^4 \end{bmatrix} = \begin{bmatrix} \frac{1}{6} & 0 & 0 & 0 \\ \frac{7}{12} & \frac{1}{2} & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} B_{10} \\ B_{11} \\ B_{12} \\ B_{13} \end{bmatrix} = \begin{bmatrix} \frac{1}{6} & 0 & 0 & 0 \\ \frac{7}{12} & \frac{1}{2} & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} B_1^4 \\ B_2^4 \\ B_3^4 \\ B_4^4 \end{bmatrix} \quad (103)$$

Dessa forma, é possível relacionar as funções Bernstein com as funções de base, definidas em cada intervalo de *knots*. Esse processo é ilustrado na Figura 39 com o vetor de *knots* $\Xi = [0, 0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1, 1]$. Nas Figura 39a, Figura 39b, Figura 39c e Figura 39d são apresentadas as funções de base em cada elemento, enquanto que nas Figura 39e, Figura 39f, Figura 39g e Figura 39h as respectivas funções Bernstein.

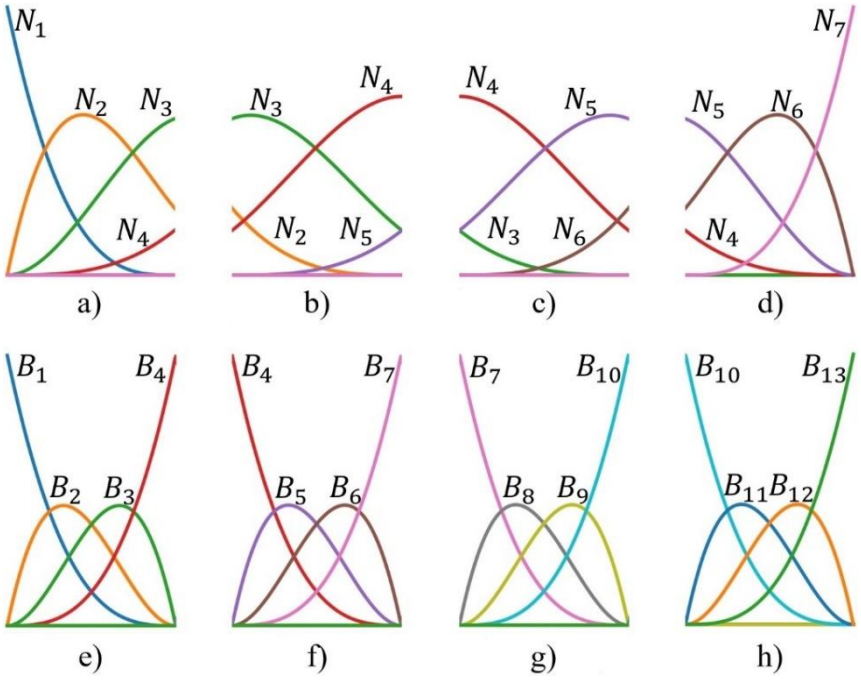

Figura 39: Decomposição Bézier nos intervalos de *knots* do *knot vector* $\Xi = [0, 0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1, 1]$.

Os termos dos operadores de extração que estabelecem relação entre essas diferentes funções, em um dado intervalo, é chamado de operador de extração local. A Equação (92) é escrita na sua forma local:

$$\boldsymbol{N}^e(\xi) = \boldsymbol{C}^e \boldsymbol{B}^e(\xi) \tag{104}$$

O operador de extração global $\boldsymbol{C}$ não precisa ser determinado na prática, apenas o operador local $\boldsymbol{C}^e$. Nota-se que a dimensão de $\boldsymbol{C}^e$ é dada por $(p+1)(p+1)$. Borden et al. [16] fornece em algoritmo para computar o operador de extração local. Novamente, destaca-se que somente o vetor de *knots* é necessário para essa tarefa, não havendo a necessidade de fornecer qualquer informação referente a geometria.

Reescrevendo a Equação (95) para determinar as funções de base NURBS em um determinado elemento:

$$\boldsymbol{R}^e(\xi) = \frac{1}{W^{b,e}(\xi)} \boldsymbol{W}^e \boldsymbol{C}^e \boldsymbol{B}^e(\xi) \tag{105}$$

onde $W^{b,e}(\xi)$ é a função peso do elemento:

$$W^{b,e}(\xi) = \sum_{i=1}^{p+1} B_{i,p}(\xi) w_i^{b,e} \tag{106}$$

Aplicando a regra da cadeia na equação acima, a primeira derivada das funções de base é dada por:

$$\begin{aligned} \frac{\partial \boldsymbol{R}^e(\xi)}{\partial \xi} &= \boldsymbol{W}^e \boldsymbol{C}^e \frac{\partial}{\partial \xi}\left(\frac{1}{W^{b,e}(\xi)} \boldsymbol{B}^e(\xi)\right) \\ \frac{\partial \boldsymbol{R}^e(\xi)}{\partial \xi} &= \boldsymbol{W}^e \boldsymbol{C}^e \left(\frac{1}{W^{b,e}(\xi)} \frac{\partial \boldsymbol{B}^e(\xi)}{\partial \xi} - \frac{\partial W^{b,e}(\xi)}{\partial \xi} \frac{\boldsymbol{B}^e(\xi)}{\left(W^{b,e}(\xi)\right)^2}\right) \end{aligned} \tag{107}$$

onde a derivadas da função peso do elemento é:

$$\frac{\partial W^{b,e}(\xi)}{\partial \xi} = \sum_{i=1}^{p+1} \frac{\partial B_{i,p}(\xi)}{\partial \xi} w_i^{b,e} \tag{108}$$

A relação entre os pontos de controle Bézier locais com os pontos de controle NURBS locais é dada de maneira similar à Equação (96):

$$\boldsymbol{P}^{b,e} = (\boldsymbol{W}^{b,e})^{-1}\boldsymbol{C}^{e^T}\boldsymbol{W}^{e}\boldsymbol{P}^{e} \tag{109}$$

onde $\boldsymbol{W}^{b,e}$ e $\boldsymbol{W}^{e}$ são, respectivamente, matrizes diagonais dos pesos associados aos pontos de controle Bézier e NURBS do elemento. Os pesos que formam a matriz $\boldsymbol{W}^{b,e}$ são obtidos pela expressão:

$$\boldsymbol{w}^{b} = \boldsymbol{C}^{T}\boldsymbol{w} \tag{110}$$

De forma análoga, a Equação (98) é escrita para um elemento Bézier:

$$\boldsymbol{C}(\xi) = \frac{1}{W^{b,e}(\xi)}(\boldsymbol{W}^{b,e}\boldsymbol{P}^{b,e})^{T}\boldsymbol{B}^{e}(\xi) \tag{111}$$

5.1.5 Operadores de Extração Bivariados

Em casos bivariados (superfícies NURBS), os operadores de extração locais são dados pelo produto tensorial entre os operadores univariados. Assim, tem-se:

$$\boldsymbol{C}^{e} = \boldsymbol{C}_{\xi}^{i} \otimes \boldsymbol{C}_{\eta}^{j} \tag{112}$$

onde $\boldsymbol{C}_{\xi}^{i}$ e $\boldsymbol{C}_{\eta}^{j}$ são os operadores de extração locais univariados de índices i e j, nas direções ξ e η, respectivamente.

Na Figura 40, observa-se uma superfície NURBS e seu domínio paramétrico formados a partir dos vetores de *knots* $\Xi = [0,0,0,0,0.25,0.5,0.75,1,1,1,1]$ e $\mathcal{H} = [0,0,0,0,0.25,0.5,0.75,1,1,1,1]$. Ambos idênticos ao *knot vector* do exemplo da Figura 39, com operadores de extração locais apresentados nas Equações (100) - (103). Para o elemento em destaque, realiza-se inicialmente a verificação dos seus operadores univariados em cada direção. Na direção ξ, seu operador univariado correspondente é o de índice 3, enquanto na direção η, é o de índice 2:

$$\boldsymbol{C}^7 = \boldsymbol{C}_\xi^3 \otimes \boldsymbol{C}_\eta^2$$

$$\boldsymbol{C}^7 = \begin{bmatrix} \frac{1}{6} & 0 & 0 & 0 \\ \frac{2}{3} & \frac{2}{3} & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} & \frac{2}{3} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{4} \end{bmatrix} \otimes \begin{bmatrix} \frac{1}{4} & 0 & 0 & 0 \\ \frac{7}{12} & \frac{2}{3} & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} & \frac{2}{3} & \frac{2}{3} \\ 0 & 0 & 0 & \frac{1}{6} \end{bmatrix} \tag{113}$$

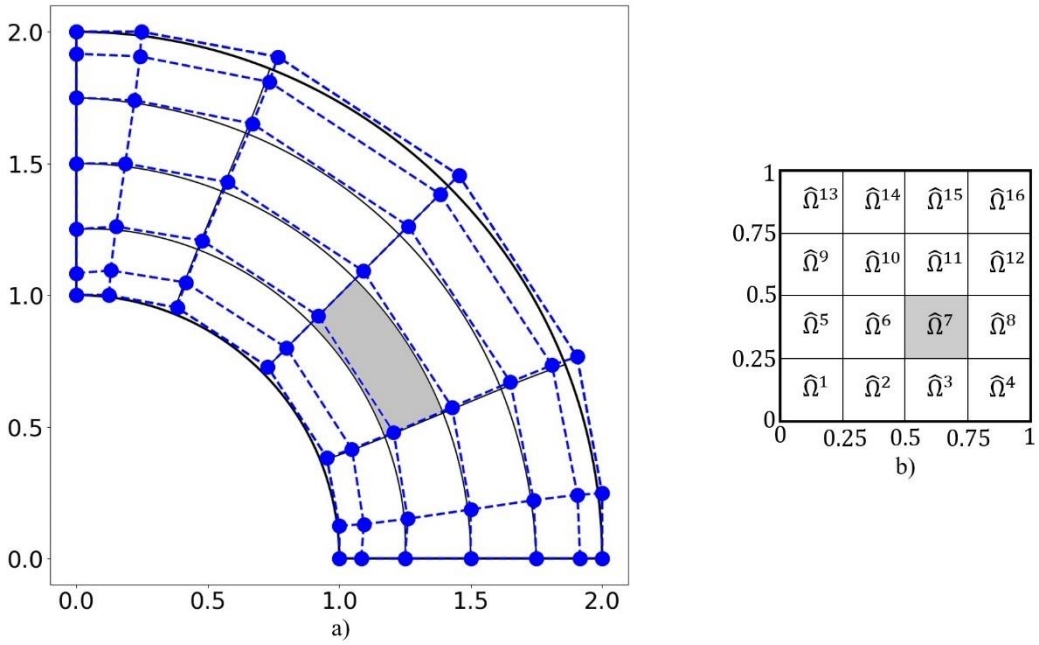

Figura 40: Determinação do operador de extração bivariado. a) Espaço físico. b) Espaço paramétrico.

5.2 Extração Bézier de T-Splines

Em T-Splines também é possível realizar a obtenção de operadores de extração e usá-los para determinar as funções de base. O trabalho de Scott et al. [15] oferece uma abordagem detalhada para encontrar esses operadores.

A obtenção de operadores de extração bivariados em T-Splines difere do processo realizado em NURBS. Como visto na Capítulo 4, em T-Splines, lida-se com vetores de *knots* locais a cada *anchor*, ou a cada ponto de controle para T-Splines cúbicas. A implicação disso é que surgem funções individuais e com

domínio próprio. Dessa maneira, para cada vetor de *knot* local, torna-se necessário computar uma linha de operador de extração para um elemento específico.

A explanação desse procedimento é feita de forma didática através de um exemplo. Na Figura 41, baseada na T-Spline cúbica da Figura 31, considera-se o domínio $\widehat{\Omega}_{18}$ referente ao *anchor* s_{18}, cujos vetores de *knots* locais são $\Xi_{18} = [0,0,0,1,2]$ e $\mathcal{H}_{18} = [0,0,1,2,3]$, nas direções ξ e η respectivamente. Determina-se as funções de base de Ξ_{18} e $\mathcal{H}_{18}$ utilizando a técnica do vetor de *knot* estendido, conforme a Seção 4.1.3. Em seguida, o processo de decomposição Bézier é aplicado nos próprios *knot vectors* estendidos. As Figura 42a e Figura 42b ilustram as funções de Ξ_{18} e $\mathcal{H}_{18}$, com as funções de interesse destacadas. Nas Figura 42c e Figura 42d, tem-se as funções de Bernstein resultantes.

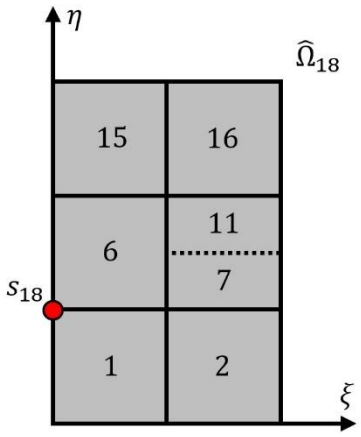

Figura 41: Domínio da função T-Spline N_{18}.

Os operadores de extração dos elementos são determinados. A partir deles, para cada intervalo de *knots*, as funções de base dos *knot vectors* estendidos são relacionadas com os polinômios de Bernstein, em um processo similar às Equações (100) - (103). Para a direção ξ:

$$\begin{bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{bmatrix} \tag{114}$$

$$\begin{bmatrix} N_2 \\ N_3 \\ N_4 \\ N_5 \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} B_4 \\ B_5 \\ B_6 \\ B_7 \end{bmatrix} \tag{115}$$

Na direção η:

$$\begin{bmatrix} M_1 \\ M_2 \\ M_3 \\ M_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \frac{1}{2} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{6} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{bmatrix} \tag{116}$$

$$\begin{bmatrix} M_2 \\ M_3 \\ M_4 \\ M_5 \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & 0 & 0 & 0 \\ \frac{7}{12} & \frac{2}{3} & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} & \frac{2}{3} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} B_4 \\ B_5 \\ B_6 \\ B_7 \end{bmatrix} \tag{117}$$

$$\begin{bmatrix} M_3 \\ M_4 \\ M_5 \\ M_6 \end{bmatrix} = \begin{bmatrix} \frac{1}{6} & 0 & 0 & 0 \\ \frac{7}{12} & \frac{1}{5} & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} B_7 \\ B_8 \\ B_9 \\ B_{10} \end{bmatrix} \tag{118}$$

As linhas marcadas nos operadores de extração representam a única função necessária para a determinação das funções T-Splines.

Inicia-se a determinação do operador de extração do elemento de índice global $A = 1$, disposto na Figura 41. Este elemento está localizado no primeiro intervalo de *knots* da função N_2 na direção ξ. Assim, é extraída a linha destacada referente ao primeiro elemento:

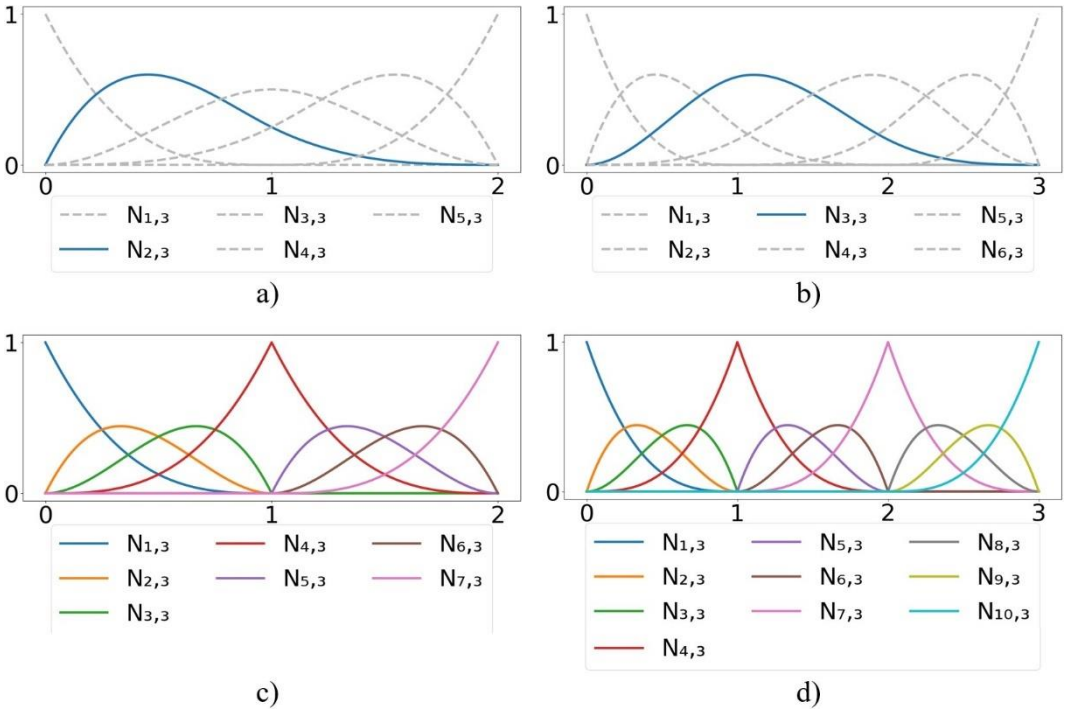

Figura 42: a) Função de base do vetor de *knots* local $\Xi_{18} = [0, 0, 0, 1, 2]$ pela técnica do vetor de *knots* estendido. b) Função de base de $\mathcal{H}_{18} = [0, 0,1, 2, 3]$. c) Decomposição Bézier aplicada a Ξ_{18}. e) Decomposição Bézier em $\mathcal{H}_{18}$.

$$c^{1,\xi} = \begin{bmatrix} 0 & 1 & \frac{1}{2} & \frac{1}{4} \end{bmatrix} \tag{119}$$

Na direção η, o elemento de índice global $A = 1$ está contido no primeiro intervalo de *knots* da função M_3. Com isso, obtém-se a linha relacionada ao primeiro elemento em η:

$$c^{1,\eta} = \begin{bmatrix} 0 & 0 & \frac{1}{2} & \frac{7}{12} \end{bmatrix} \tag{120}$$

Realizando o produto tensorial entre esses dois vetores, gera-se o vetor:

$$\begin{gathered} c_a^1 = c^{1,\eta} \otimes c^{1,\xi} \\ c_a^1 = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \\ 0 \quad \frac{1}{2} \quad \frac{1}{4} \quad \frac{1}{8} \quad 0 \quad \frac{7}{12} \quad \frac{7}{24} \quad \frac{7}{48}] \end{gathered} \tag{121}$$

onde a é o índice local do *anchor* 18 no elemento $A = 1$. Como resultado, obtém-se uma linha do operador de extração $\boldsymbol{C}^1$. Esse mesmo processo é repetido para

todos os elementos no domínio do *anchor* s_{18}, $(e = 2, 6, 7, 11, 15, 16)$. Para determinar o operador $\boldsymbol{C}^1$ por completo, deve-se seguir para os próximos *anchors* que possuem o elemento 1 nos seus domínios.

5.2.1 Operadores de Extração para Malhas não Estruturadas

Malhas não estruturadas oferecem uma maior flexibilidade e podem ser úteis na modelagem de domínios complexos e irregulares. No contexto da análise isogeométrica, um tratamento adequado é demandado para viabilizar a geração desse tipo de malha. Nesse sentido, Scott [17] desenvolveu uma estratégia baseada em extração Bézier, para permitir a formação de *patches* de T-Splines cúbicos não estruturados, por meio da utilização de *extraordinary points*.

Os chamados *extraordinary points* são os vértices de uma T-Mesh com valência diferente de quatro e que, ao mesmo tempo, não são T-Junctions [17]. A Figura 43 mostra um exemplo de uma região composta por *extraordinary points*, destacados por círculos em vermelho. Dizemos que os elementos da T-Mesh que se conectam com esses pontos possuem uma vizinhança *one-ring* com os *extraordinary points*. Os próximos elementos adjacentes formam uma vizinhança *two-ring*. Os *Extraordinary elements* são um conjunto formado por elementos *one-ring* e *two-ring*. Na Figura 43, observa-se esses elementos em destaque. Os demais elementos são chamados de regulares.

Em T-Splines, para cada vértice da T-Mesh, uma função de base é estabelecida e em seguida se inicia a determinação dos operadores de extração para os elementos no domínio da função. Para T-Splines com *extraordinary points*, são definidos operadores de extração padronizados nos *Extraordinary elements*. O processo descrito na Seção 5.2 é empregado apenas nos elementos regulares.

Para os *extraordinary elements*, a ideia é definir uma relação entre os pontos de controle Bézier P^b de um elemento com os pontos de controle originais P. Essa relação forma o operador de extração transposto $(\boldsymbol{C}^e)^T$.

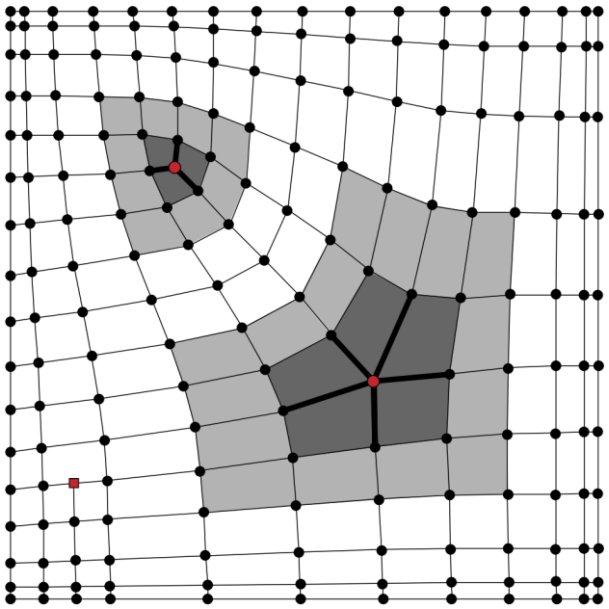

Figura 43: T-Mesh com *extraordinary points* [17].

Os pontos de controle Bézier de cada *extraordinary element* podem ser divididos em: 4 *face points* $P_6^{b,face}$, $P_7^{b,face}$, $P_{10}^{b,face}$ e $P_{11}^{b,face}$; 8 *edge points* $P_2^{b,edge}$, $P_3^{b,edge}$, $P_5^{b,edge}$, $P_8^{b,edge}$, $P_9^{b,edge}$, $P_{12}^{b,edge}$, $P_{14}^{b,edge}$ e $P_{15}^{b,edge}$; e 4 *vertex points* $P_1^{b,edge}$, $P_4^{b,edge}$, $P_{13}^{b,edge}$ e $P_{16}^{b,edge}$. A Figura 44 permite visualizar a disposição desses pontos de controle Bézier. Considerando intervalos de *knots* igualmente espaçados, tem-se as seguintes relações para os *face points*:

$$P_6^{b,face} = \frac{4}{9}\boldsymbol{P_A} + \frac{2}{9}\boldsymbol{P_B} + \frac{1}{9}\boldsymbol{P_C} + \frac{2}{9}\boldsymbol{P_D} \tag{122}$$

$$P_7^{b,face} = \frac{2}{9}\boldsymbol{P_A} + \frac{4}{9}\boldsymbol{P_B} + \frac{2}{9}\boldsymbol{P_C} + \frac{1}{9}\boldsymbol{P_D} \tag{123}$$

$$P_{10}^{b,face} = \frac{2}{9}\boldsymbol{P_A} + \frac{1}{9}\boldsymbol{P_B} + \frac{2}{9}\boldsymbol{P_C} + \frac{4}{9}\boldsymbol{P_D} \tag{124}$$

$$P_{11}^{b,face} = \frac{1}{9}\boldsymbol{P_A} + \frac{2}{9}\boldsymbol{P_B} + \frac{4}{9}\boldsymbol{P_C} + \frac{2}{9}\boldsymbol{P_D} \tag{125}$$

onde $\boldsymbol{P_A}$, $\boldsymbol{P_B}$, $\boldsymbol{P_C}$ e $\boldsymbol{P_D}$ são os pontos de controle que envolvem o elemento da T-Mesh.

Os *edge points* são definidos por:

$$P_i^{b,edge} = \frac{1}{2}P_a^{b,face} + \frac{1}{2}P_d^{b,face} \tag{126}$$

$$P_j^{b,edge} = \frac{1}{2} P_b^{b,face} + \frac{1}{2} P_c^{b,face} \tag{127}$$

Por fim, os *vertex points* são dados por:

$$P_j^{b,vertex} = \sum_{i=1}^{N} \frac{1}{4} P_i^{b,face} \tag{128}$$

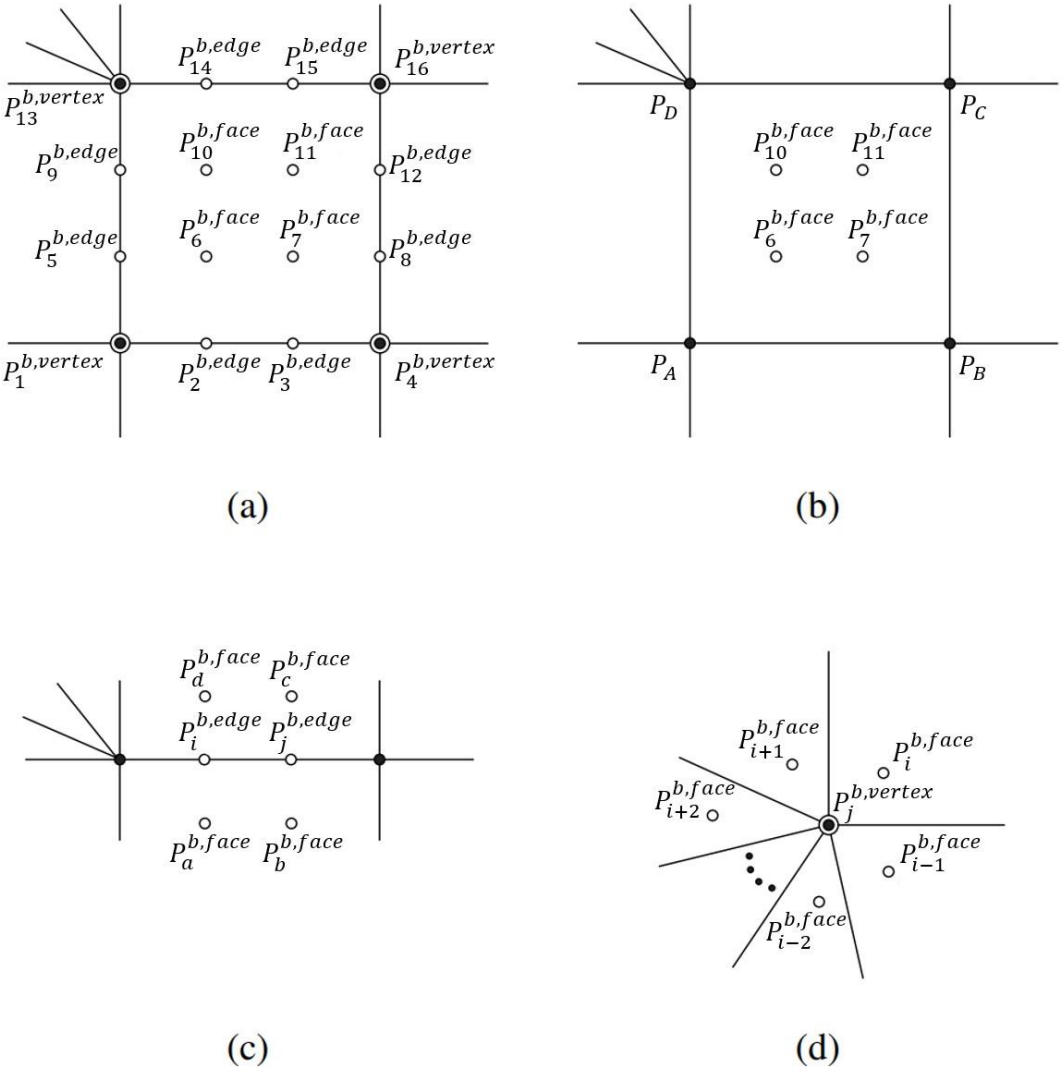

Figura 44: Pontos de controle Bézier de um *extraordinary element*. Adaptado de Scott [17].

6 Modelador Bidimensional FEMEP

Neste capítulo, apresenta-se a ferramenta computacional FEMEP (*Finite Element Method Educational Computer Program* - https://github.com/lfmartha/FEMEP), criada para a modelagem bidimensional de sólidos, com a finalidade de gerar modelos para a posterior análise, tanto no contexto de MEF, como também em AIG. Inicialmente, o programa foi concebido para ser uma ferramenta educacional voltada para análise isoparamétrica de elementos finitos. A expansão para incorporar uma modelagem mais complexa, no âmbito da análise isogeométrica, foi a contribuição deste trabalho.

A origem do FEMEP tem como pilar a biblioteca HETOOL, desenvolvida pelo grupo de pesquisa composto por colaboradores da Pontíficia Universidade Católica do Rio de Janeiro e da Universidade Federal Fluminense [28]. O HETOOL é uma estrutura de dados implementada em Python no paradigma de programação orientada a objetos (POO) para a modelagem geométrica bidimensional de forma iterativa. A aplicação tem como base a conhecida estrutura de dados Half-Edge para Sólidos 2-manifold, adaptada para modelos 2D de subdivisões planares. Bomfim [18] oferece uma análise detalhada sobre esse tópico. O tratamento dos elementos topológicos é realizado de maneira interna na biblioteca, de forma que o usuário não necessita de conhecimento prévio da estrutura Half-Edge para sua utilização. Além disso, a biblioteca disponibiliza o recurso de gerenciamento de atributos, realizado através de um arquivo de extensão JSON, que viabiliza a aplicabilidade em vários problemas da mecânica computacional.

6.1 Características Gerais

O modelador FEMEP é uma ferramenta poderosa e versátil, projetada tendo em vista várias características para atender as necessidades de estudantes e pesquisadores na área da mecânica computacional.

O software é desenvolvido em Python, uma linguagem de programação de alto nível e de tipagem dinâmica, com uma sintaxe simples e de fácil compreensão, além de permitir uma fácil integração com bibliotecas e frameworks disponíveis de

forma gratuita. Por essas razões, Python é uma escolha ideal para ambientes educacionais.

A implementação segue o padrão POO, que organiza o código na forma de classes, objetos, propriedades e métodos. As classes são estruturas que definem objetos, especificando propriedades pré-estabelecidas e apresentando métodos que podem ser executados. Um objeto é uma instância de uma classe, ou seja, um elemento de referência que possui as propriedades e métodos da classe. Propriedades são um conjunto de características e variáveis que um objeto pode apresentar. Os métodos são as operações que um dado objeto pode realizar.

Com uma interface gráfica interativa, a execução do código não segue um padrão sequencial. As ações do usuário são imprevisíveis e o programa precisa ser capaz de executar as rotinas adequadas de acordo com essas interações. Um paradigma adotado no FEMEP é a programação orientada a eventos, que permite que o fluxo do programa seja determinado por eventos. Um evento ocorre quando o usuário interage com um elemento gráfico, seja manipulando um botão do mouse ou selecionando um botão na interface.

O Qt (https://www.qt.io/product/framework) foi o *framework* utilizado para a criação da interface gráfica de usuário (GUI). Em Python, a comunicação com o Qt é realizada através da biblioteca PyQt5, que usa o conceito de POO para criar e manipular os elementos de interface (*Widgets*), conhecidos como QtWidgets no Qt. Cada QtWidget possui uma subclasse com propriedades e métodos específicos. Menus, botões, caixas de checagem e caixas de rolagem são exemplos desses elementos de interface.

No FEMEP, a visualização do modelo e de seus atributos é realizada pelo OpenGL por meio da biblioteca PyOpenGL. O OpenGL é uma API para renderização de objetos, amplamente difundida em áreas como computação gráfica e visualização científica.

O padrão de projeto é o MVC (*Model* - *View* - *Controller*). Essa arquitetura de software separa a aplicação em três camadas principais: a camada *model*, responsável pelo armazenamento de dados; o *controller*, que recebe as requisições do usuário e realiza os processamentos necessários no *model*; e a camada *view*, que coleta as informações do *model* para a exibição dos dados.

6.2 Principais Módulos

O FEMEP é um programa composto por diversos módulos, e permite que o usuário forneça informações iniciais através de sua interface. Essas informações de entrada são providas de duas maneiras: por eventos de mouse na área de modelagem, que são gerenciadas pela classe *Canvas*, ou pelo acionamento de elementos de interface, que são gerenciados pelo *AppController*.

O FEMEP é estruturado em torno de três classes principais, criadas com base no padrão MVC. Essas classes recebem o prefixo *He*, pois têm suas nomenclaturas derivadas da biblioteca HETOOL: *HeModel*, *HeView* e *HeController*. A classe *HeModel* armazena os dados do modelo no âmbito de entidades geométricas e topológicas. O *HeView* acessa o *HeModel* para a coleta de dados e os fornece ao *Canvas* para visualização na tela. O *HeController* é a classe central, que tem acesso a maioria dos módulos do programa e realiza as modificações no *HeModel*. A Figura 45 ilustra o diagrama de robustez formado pelas principais classes do programa.

Destaca-se também algumas classes fundamentais presentes no programa, que são provenientes da biblioteca HETOOL. Os operadores de Euler são um conjunto de classes que realizam alterações na estrutura de dados no nível de entidades topológicas, chamadas em resposta a cada iteração de modelagem executada pelo programa. Em particular, esses operadores criam superfícies fechadas e modificam superfícies existentes, adicionando ou excluindo faces, arestas e vértices. Além dos operadores de Euler, o programa também conta com operadores auxiliares, que não realizam modificações topológicas, mas desempenham outras funções específicas na estrutura de dados. Isso inclui a adição ou remoção de uma malha em uma determinada face, bem como a adição ou remoção de um atributo particular de um elemento geométrico. A classe *UndoRedo* armazena uma quantidade específica de comandos realizados pelo usuário, que podem ser desfeitos ou refeitos. Cada um desses comandos é representado por um conjunto de operações (operadores de Euler ou operadores auxiliares). A classe *AttribManager* é responsável pelo gerenciamento de atributos de modelagem, que são criados a partir de um arquivo JSON e transmitidos na forma de um dicionário em Python com a estrutura chave: valor.

A seguir, é feita uma apresentação concisa das classes principais, de forma a oferece uma visão geral útil. Para uma explicação mais detalhada, recomenda-se a leitura do trabalho de Bomfim [18].

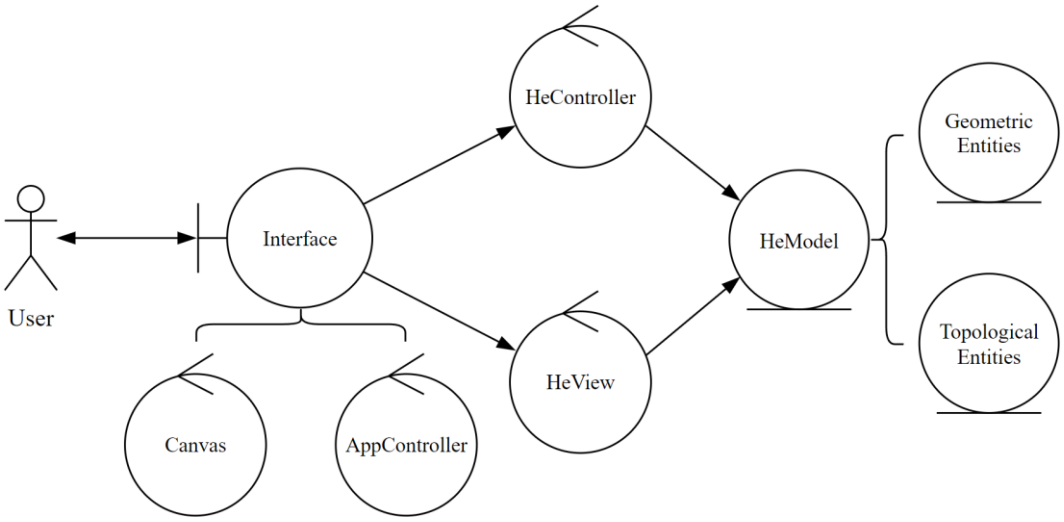

Figura 45: Diagrama de robustez com os módulos principais do FEMEP. Adaptado de Bomfim [18].

6.2.1 Classe *Canvas* e *AppController*

A classe *Canvas* recebe como herança a superclasse *QGLWidget* do Qt, que possui funcionalidades para exibir gráficos do OpenGL integrado com o Qt. O *QGLWidget* encapsula o sistema gráfico OpenGL e fornece funções de *callback* para a interface, como *initializeGL()*, usada para configuração inicial do OpenGL, a *resizeGL()*, que ajusta a visualização quando há alteração no tamanho do painel, e a *paintGL()*, responsável pela renderização de objetos. Além disso, o *QGLWidget* também fornece funções de *callback* para tratamento de interações com mouse, como o *mousePressEvent()*, *mouseMoveEvent()*, e *mouseReleaseEvent()*. Todas essas funções são reimplementadas na classe *Canvas* para atender às demandas específicas do FEMEP.

A classe *AppController* desempenha papel fundamental na gestão das interações do usuário por meio de botões e demais elementos de interface. O *AppController* associa cada um desses elementos de interface com métodos particulares e específicos. O *AppController* também é responsável por estabelecer conexão com outras classes do programa como *Canvas* e o *HeController*.

6.2.2 Classe *HeModel*

A classe *HeModel* tem a finalidade de armazenar as entidades topológicas e geométricas do modelo. As entidades topológicas são formadas por um conjunto de classes, cada uma correspondendo a um tipo de elemento da estrutura de dados *Half-Edge*: *Vertex*, *Half-Edge*, *Edge*, *Loop*, *Face* e *Shell*. A utilização da estrutura de dados Half-Edge facilita a realização de consultas de adjacência de maneira eficiente entre todos os elementos topológicos presentes no sólido (*shell*). Nessa estrutura, é introduzido o conceito da entidade abstrata semi-aresta (*half-edge*). Uma determinada aresta (*edge*) delimita duas faces (*faces*), permitindo que cada aresta seja separada em das semi-arestas, paralelas e orientadas em direções opostas. As semi-arestas proporcionam uma navegação eficiente no modelo. Os laços (loops) representam o percurso contido em uma face. O laço externo é referente a fronteira externa da face, enquanto os laços internos, se houverem, referem-se a fronteira de elementos desconexos internos [45].

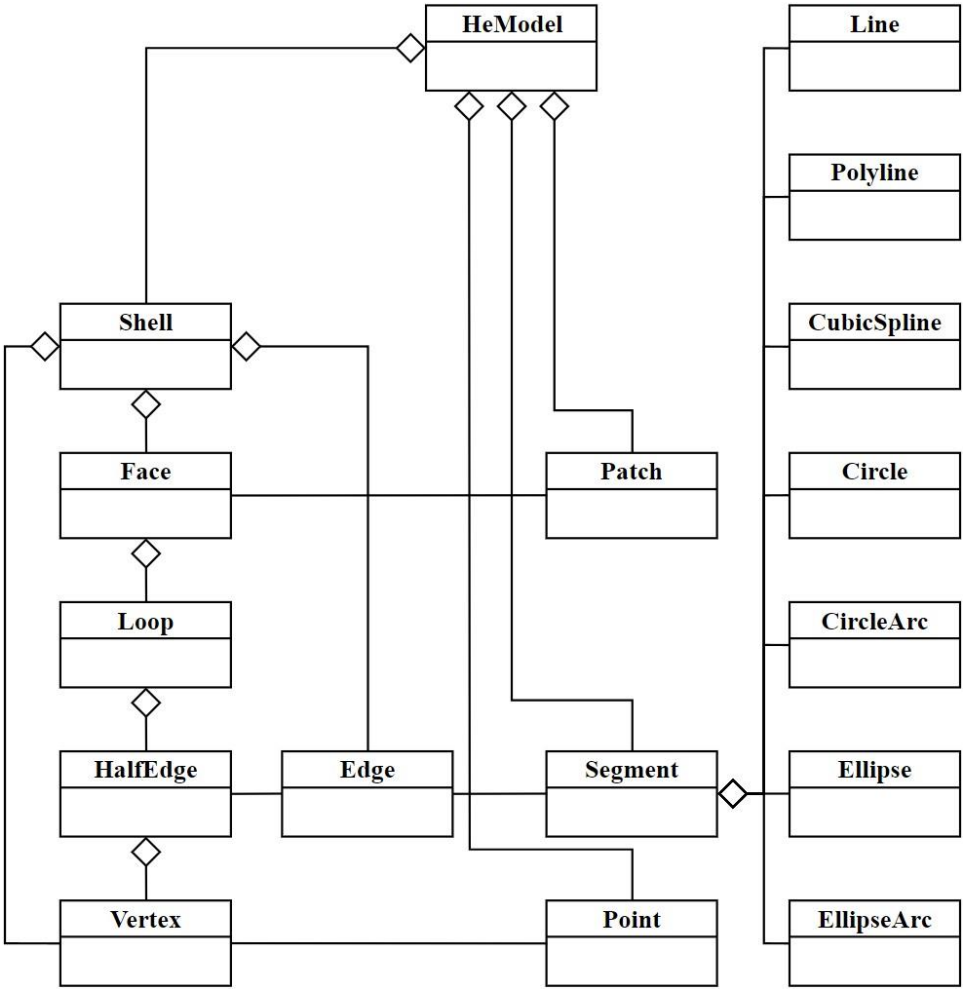

Figura 46: Diagrama de classes do *HeModel*. Adaptado de Bomfim [18].

As entidades geométricas usadas para representar a forma e posição de objetos no plano, são compostas por três classes: *Point*, *Segment* e *Patch*. A classe *Point* representa os pontos do modelo. A classe *Segment* é designada para representar as curvas. No FEMEP existem diferentes tipos de curvas, cada uma com sua classe própria (*Line*, *Polyline*, *CubicSpline*, *Circle*, *CircleArc*, *Ellipse* e *EllipseArc*). A classe *Segment* possui a propriedade *curve*, que é um objeto de uma das classes de curvas. A classe *Patch* representa as regiões fechadas no modelo.

As classes *Vertex*, *Edge* e *Face* retratam as entidades topológicas não abstratas e estão associadas de maneira bidirecional com as entidades geométricas. Isso significa que é possível obter as respectivas classes geométricas a partir das classes topológicas e vice-versa. O diagrama de classes do *HeModel* é apresentado na Figura 46, onde é possível observar a relação entres as entidades existentes.

6.2.3 Classe *HeView*

A classe *HeView* atua como um intermediário entre a classe *HeModel*, que guarda as informações o modelo, e a classe *Canvas*, que renderizar a cena de objetos na tela. Um objeto da classe *HeModel* é passado na forma de atributo para a classe *HeView*, que apresenta métodos específicos para a realização da leitura desse objeto, sem que haja alteração das informações. Alguns desses métodos incluem *getPoints()*, *getSegments()* e *getPatches()*, que retornam as respectivas entidades geométricas. O Método *getBoundBox()* retorna as coordenadas extremas da região formada pelo retângulo que abrange todas as entidades do modelo. Outros métodos como *snapToPoint()* e *snapToSegment()* recebem como argumento as coordenadas de um ponto e um valor de tolerância, e retornam um booleano indicando se há alguma entidade nas proximidades da coordenada, levando em consideração a tolerância especificada. Além disso, também retornam as coordenadas de um ponto ou segmento existente no modelo que está o mais próximo possível das coordenadas fornecidas.

6.2.4 Classe *HeController*

O *HeController* é a classe principal, com acesso a maioria dos módulos e encarregada de gerenciar todo o processamento do programa. As interações

realizadas pelo usuário que geram mudanças nas entidades geométricas e topológicas do modelo passam por métodos existentes no *HeController*. Por exemplo, o processo de inserção de curvas requer uma série de etapas e tratamentos, que são realizados no *HeController*. Primeiro, é necessário verificar se a nova curva realiza interseção com curvas existentes no modelo. Em caso positivo, os pontos de interseção são determinados e as curvas são divididas. Em seguida, é realizado o devido tratamento topológico dos segmentos e vértices resultantes.

O processo de exclusão de entidades pelo usuário também é gerenciado pelo *HeController*. A exclusão de um vértice entre duas curvas deve gerar a união dessas curvas. Se o vértice for comum a mais de duas curvas, a união não é possível, devendo ocorrer a exclusão tanto do vértice, como das curvas adjacentes. Após essas modificações, o tratamento topológico é realizado para as novas entidades existentes. A geração de malhas para análise por elementos finitos e análise isogeométrica também é processada por um método no *HeController*, que verifica o tipo de malha escolhida pelo usuário e realiza a chamada das funções adequadas da classe *Mesh*.

6.3 Interface e Funcionalidades

O FEMEP apresenta uma interface projetada para ser simples, que fornece ao usuário uma experiência intuitiva durante sua utilização. A interface é composta por barra principal, barra secundária, barra de modelagem, canvas e quadro dinâmico, conforme demonstrado na Figura 47.

A barra principal oferece algumas funcionalidades através de seus botões. Estes incluem a capacidade de criar uma nova aba, abrir e salvar um modelo, além de desfazer e refazer ações. A barra principal também oferece opções de ajuste da área de visualização como centralizar o modelo na tela, ampliar, reduzir e transladar.

A barra secundária conta com um botão para seleção de entidades (pontos, segmentos e regiões) e outro botão para deletar entidades. Possui o botão *grid*, que adiciona uma grade de pontos auxiliares para facilitar a modelagem no *canvas*. Apresenta um botão que ativa a interface do gerenciados de atributos, onde é possível associar atributos às entidades geométricas. Há também botões para a configuração de malha em regiões fechadas do modelo.

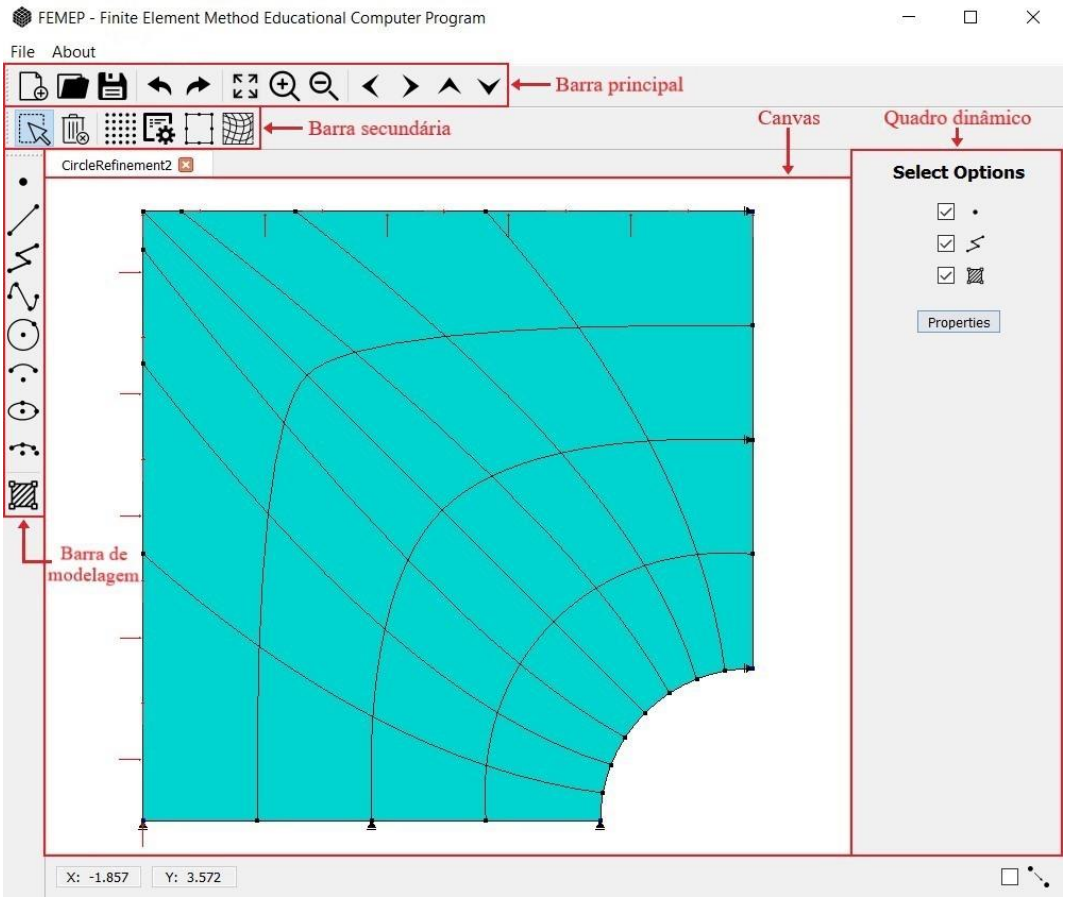

Figura 47: Interface do FEMEP.

6.3.1 Modelagem Interativa de Curvas

A barra de modelagem contém elementos de interface do tipo *ToggleButton*. Quando um desses botões é selecionado, é permitida a modelagem de uma curva específica predefinida, através do mouse no canvas ou inserindo informações com o teclado no quadro dinâmico. As curvas presentes no programa são um segmento de linha reta, linha poligonal, B-Spline cúbica, círculo, arco de círculo, elipse e arco de elipse. Todas são tratadas na forma de NURBS pelo FEMEP, apresentando propriedades de grau polinomial, vetor de *knots*, pontos de controle e pesos associados. Além das curvas mencionadas, também está presente na barra de modelagem um botão dedicado a criar pontos no modelo, bem como um botão para criar regiões.

O canvas é a área designada para a modelagem de curvas. Quando uma curva específica é selecionada, o usuário pode modelá-la clicando com o botão

esquerdo do mouse para definir pontos que a compõem. Esses são chamados de pontos de definição. Vale ressaltar que nem sempre esses pontos estão contidos nas curvas, ou são seus pontos de controle, embora isso possa ocorrer em certos tipos de geometria. Eles são, na verdade, referências a partir dos quais as curvas são desenhadas. Uma explicação breve é fornecida para cada tipo de curva.

No caso da linha, apenas um ponto inicial e um ponto final são necessários. A linha poligonal, composta por múltiplos segmentos de reta, pode ter tantos pontos quanto o usuário desejar. A B-Spline, similarmente, não tem um limite de pontos, que neste caso particular são seus pontos de controle. A curva do tipo círculo requer um ponto para definir seu centro e um ponto final para definir seu raio. Para o arco de círculo, além do centro, é preciso especificar um ponto de início e final do arco. A elipse é definida pelo primeiro ponto que determina seu centro, o próximo ponto determina o primeiro eixo, e o ponto seguinte determina o segundo eixo. Para o arco de elipse, precisa-se de cinco pontos: o primeiro define o centro, os dois seguintes definem os dois eixos da elipse, e os dois últimos definem o ponto inicial e final do arco de elipse. Ilustrado na Figura 48, tem-se as sequências de pontos necessários para a criação de cada uma dessas entidades. As linhas tracejadas indicam as atrações realizadas via projeção radial. No exemplo da Figura 48f, durante a determinação do final do arco, qualquer ponto escolhido ao longo da linha tracejada é atraído para um ponto contido na curva. Um procedimento semelhante é observado na Figura 48g, durante a definição do terceiro ponto que estabelece o segundo eixo da elipse. Na Figura 48h, o mesmo processo de atração é aplicado para determinar o segundo eixo da elipse, bem como o ponto inicial e final do arco.

O quadro dinâmico, localizado a direita, modifica sua interface de acordo com a entidade selecionada na barra de modelagem. Ele apresenta um modo teclado, que permite a inserção dos pontos que definem as curvas através de digitação de valores nos *LineEdits*. Uma vez que esses valores forem preenchidos, o usuário apenas deve pressionar um *PushButton* situado logo abaixo. A Figura 49 demonstra a interface do quadro dinâmico relativamente às curvas modeladas na Figura 48. Em se tratando das curvas do tipo círculo, arco de círculo, elipse e arco de elipse, nem sempre pode ser conveniente utilizar pontos coordenados para a criação dessas curvas. Por exemplo, pode-se desejar inserir um arco de círculo começando pelas coordenadas do centro, mas em seguida determinar um valor

específico para o raio e estabelecer um ângulo inicial e um ângulo final para o arco, conforme a Figura 49f. Dessa forma, o quadro dinâmico apresenta elementos do tipo *ComboBox*, que permite escolher se o usuário deseja fornecer pontos coordenados, ou determinar um comprimento e um ângulo.

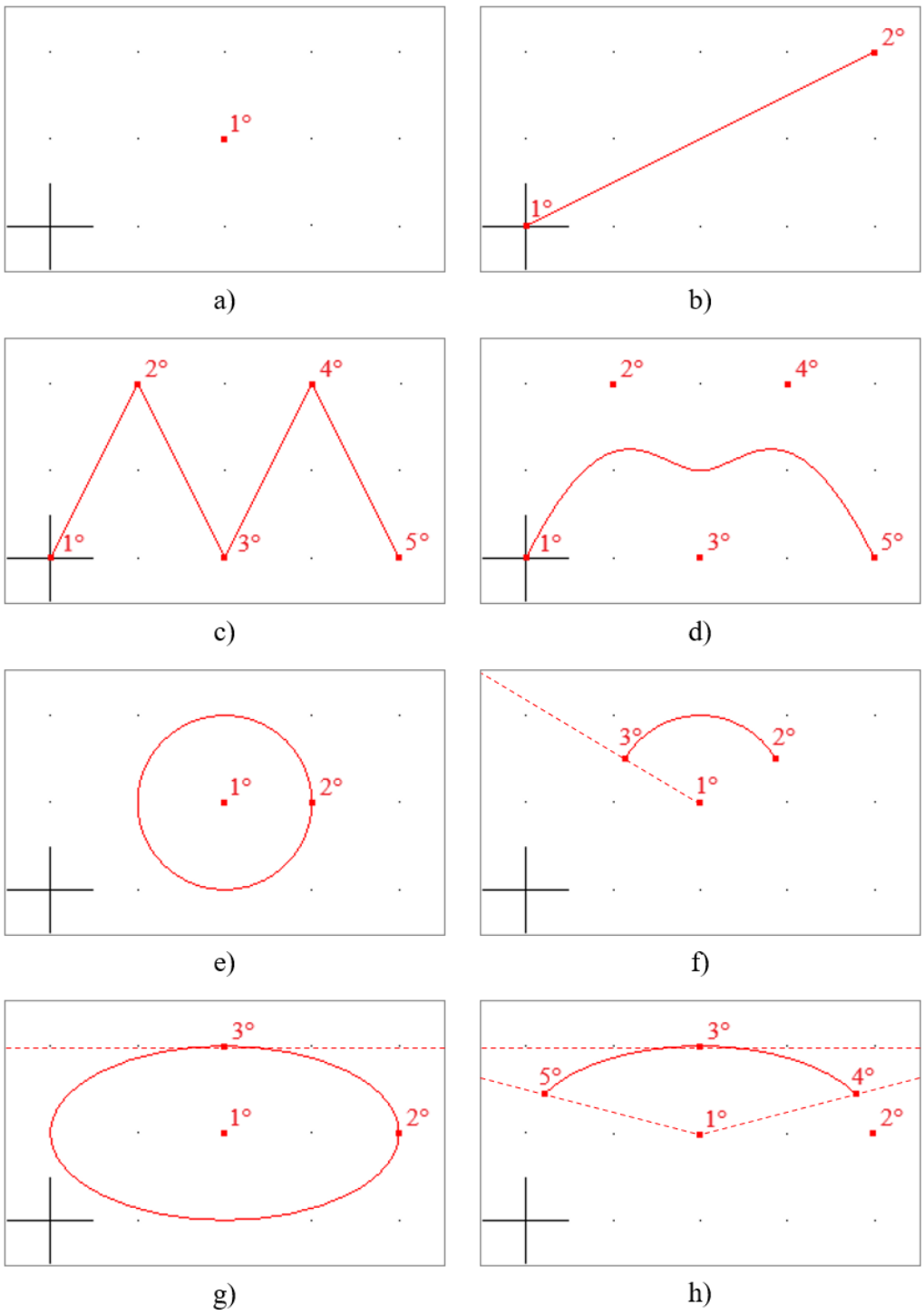

Figura 48: Modelagem no canvas. A numeração indica a sequência de pontos de definição necessários para a formação de cada entidade. a) Ponto. b) Reta. c) Linha poligonal. d) B-Spline cúbica. e) Círculo. f) Arco de círculo. g) Elipse. h) Arco de elipse.

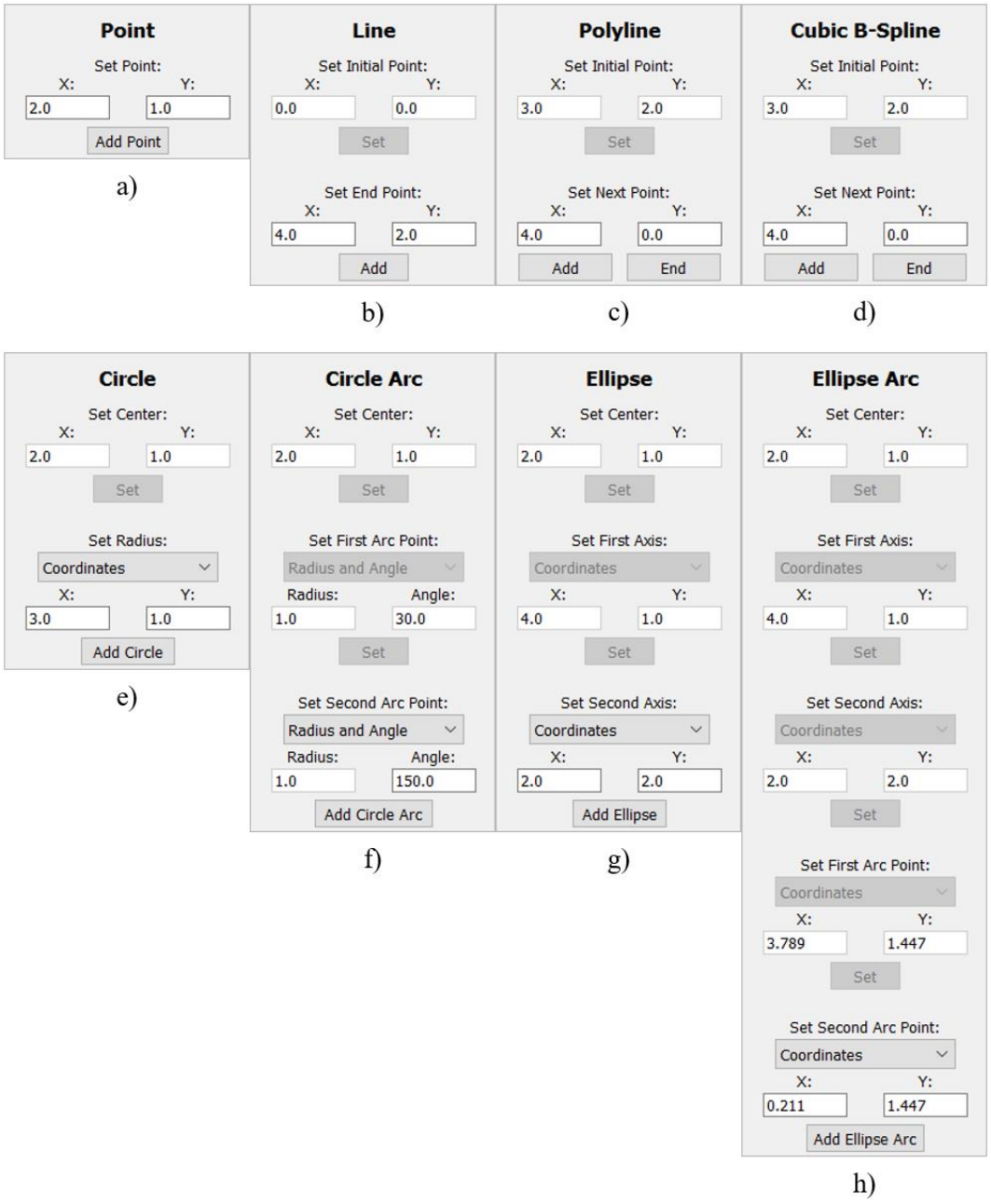

Figura 49: Quadro dinâmico em modo teclado para inserção de entidades. a) Ponto. b) Reta. c) Linha poligonal. d) B-Spline cúbica. e) Círculo. f) Arco de círculo. g) Elipse. h) Arco de elipse.

6.3.2 Coleta de Curvas

A coleta de uma curva é realizada através da classe *GeoCollector*, que se comunica com o *AppController* e com o *Canvas*, permitindo que os pontos de definição sejam inseridos tanto através do quadro dinâmico, como clicando com o mouse na área de modelagem, ou mesmo com uma combinação alternada das duas

formas de inserção. O *GeoCollector* apresenta métodos que auxiliam no controle da coleta dos pontos de definição, verificando se a curva apresenta uma quantidade limitada ou ilimitada de pontos, e se a coleta foi finalizada ou não. A classe *GeoCollector* conta com a propriedade *geoType*, uma variável *string* que especifica a curva em coleta. Além disso, possui a propriedade *geo*, que é uma instância de uma das classes de curvas: *Line*, *Polyline*, *CubicSpline*, *Circle*, *CircleArc*, *Ellipse* ou *EllipseArc*.

O início da coleta ocorre após o usuário pressionar um dos botões de curva na barra de modelagem, que gera uma chamada no *AppController*. Um dos seguintes métodos é acionado, a depender do tipo de curva escolhida: *on_actionLine()*, *on_actionPolyline()*, *on_actionCubicSpline()*, *on_actionCircle()*, *on_actionCircleArc()*, *on_actionEllipse()* ou *on_actionEllipseArc()*. Esses métodos são responsáveis por ajustar duas propriedades importantes, uma na classe *Canvas* e outra no *GeoCollector*, através das funções *setMouseAction()* e *setGeoType()*, respectivamente. O *setMouseAction()* é pertencente a classe *Canvas* e altera a propriedade *curMouseAction* para a *string* "COLLECTION", indicando que uma coleta é iniciada. O método *setGeoType()*, também pertencente a classe *Canvas*, chama o método homônimo na classe *GeoCollector*, que modifica a propriedade *geoType* para uma *string* correspondente ao tipo de curva selecionada.

A inserção dos pontos de definição da curva é realizada em seguida. O usuário pode optar por clicar com o botão esquerdo do mouse na região de modelagem, ou utilizar o quadro dinâmico. Quando um clique é detectado na área de modelagem, o método *mousePressEvent()* da classe *Canvas* é chamado. Este verifica se a propriedade *curMouseAction* está configurada para "COLLECTION". Caso positivo, é verificado se a coleta foi iniciada na classe *GeoCollector*. Para a inserção do primeiro ponto, a coleta ainda não está ativa, sendo necessário chamar o método *startGeoCollection()* do *GeoCollector*. O *startGeoCollection()* cria um objeto da classe de curva a ser coletada e o armazena na propriedade *geo*. Seguidamente, o método *insertPoint()* da classe *GeoCollector* é chamado para realizar a inserção do ponto de definição, que receberá o devido tratamento para permitir a formação da curva.

Caso seja escolhido o uso do quadro dinâmico, um processo análogo ocorre. No entanto, em vez de o método *mousePressEvent()* do *Canvas* ser chamado, o método correspondente ao botão pressionado na interface do quadro dinâmico é acionado no *AppController*. Nesse caso, não há a necessidade de realizar verificação se a coleta está ativa no *GeoCollector*, pois a inserção sistemática dos pontos de definição é uma característica do quadro dinâmico. Assim, de acordo com o botão pressionado, é possível determinar se está sendo inserido o primeiro ponto de definição, o último ou qualquer outro. O primeiro botão a ser pressionado, chama os métodos *startGeoCollection()* e *insertPoint()* do *GeoCollector*. Uma vez que um ponto de definição é inserido, o botão correspondente a esse ponto fica desabilitado na interface.

A inserção dos pontos de definição seguintes segue o mesmo processo: é chamado o método *mousePressEvent()* do *Canvas* ou o método do *AppController* associado ao *PushButton* correspondente. Entretanto, como a coleta já foi iniciada, o método *startGeoCollection()* não é mais acionado, sendo chamado apenas o método *insertPoint()*.

Para o último ponto de definição, um caminho distinto é percorrido para permitir a conclusão da coleta da curva. Sempre que um botão do mouse é pressionado, além da chamada ao método *mousePressEvent()*, o método *mouseReleaseEvent()* é acionado ao soltar o botão. O *mouseReleaseEvent()* não tem efeito na inserção dos pontos de definição anteriores, mas é crucial para o último ponto. No *mouseReleaseEvent()*, ocorre a verificação para determinar se a coleta foi concluída. A coleta termina caso o número de pontos da curva for limitado e a quantidade necessária de pontos de definição tiver sido atingida. Isso é verificado pelos métodos *isUnlimited()* e *hasFinished()* do *GeoCollector*. A coleta também termina caso o botão direito for pressionado e a curva for ilimitada, desde que o número mínimo de pontos tenha sido coletado. Após a conclusão, o método *getCollectedGeo()* do *GeoCollector* é acionado, retornando o objeto da curva recém coletada. Posteriormente, o método *insertSegment* do *Hecontroller* é chamado, sendo este responsável por inserir a curva no modelo e realizar verificações de interseção com curvas existentes. Finalmente, o método *endGeoCollection()* do *GeoCollector* é chamado, definindo a propriedade *geo* para *None*.

No caso de a inserção da uma curva ser realizada com o quadro dinâmico, a verificação do término da coleta não é necessária. Isso porque, ao pressionar o último botão da interface do quadro dinâmico, já se entende que a coleta será concluída. Este botão chama um método no *AppController* que aciona as funções *insertPoint()* e *getCollectedGeo()* do *GeoCollector*. A curva coletada é retornada e passada como argumento pelo *insertSegment()* do *Hecontroller*. Por fim, é chamado o *endGeoCollection()* do *GeoCollector*.

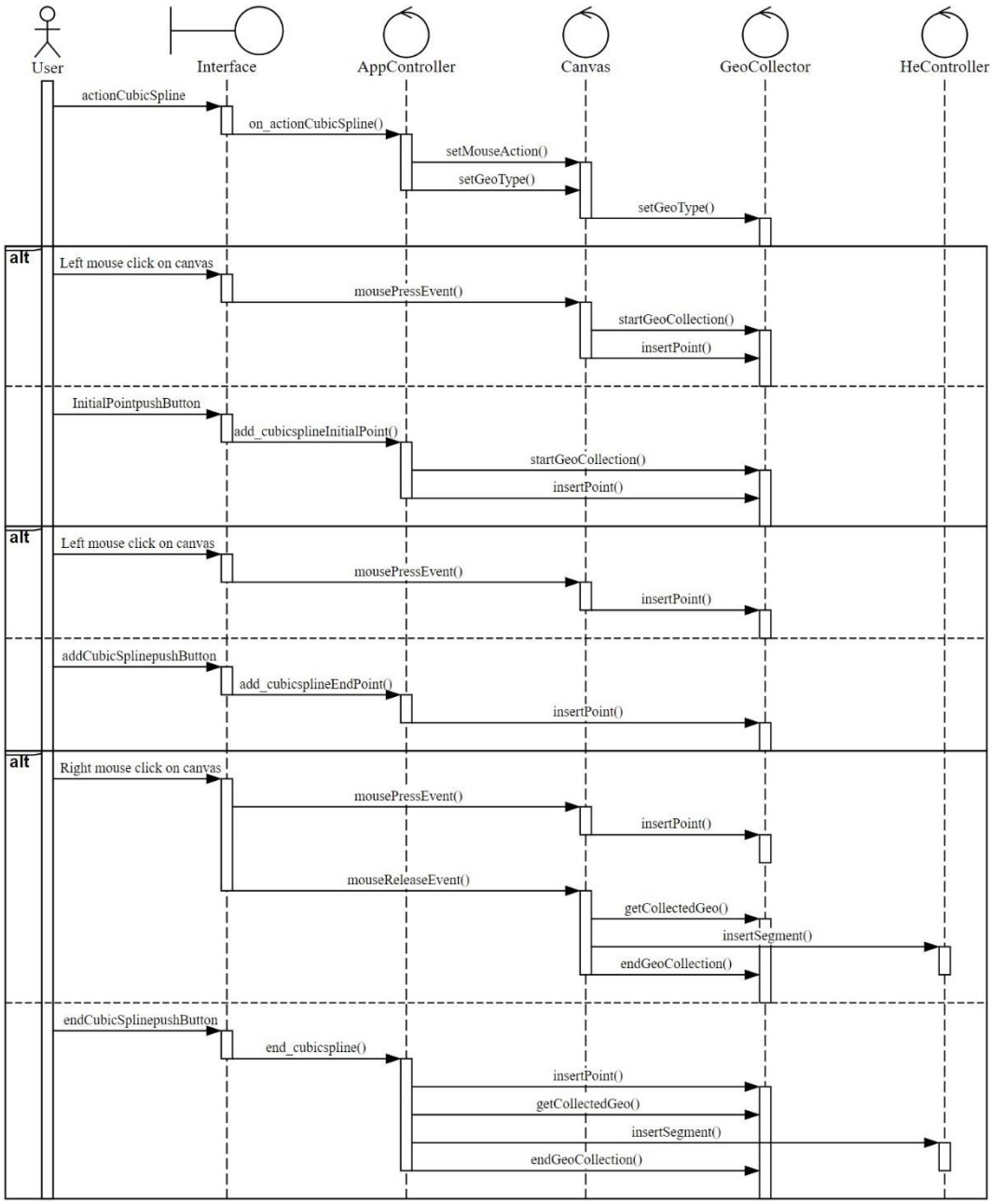

Figura 50: Diagrama de sequência para a coleta de uma curva da classe *CubicSpline*.

A Figura 50 apresenta o diagrama de sequência com os métodos acionados para a coleta de uma curva da classe *CubicSpline*, caracterizada por ter uma quantidade ilimitada de pontos de definição. Por outro lado, a Figura 51 exibe o diagrama de sequência para uma curva da classe *CircularArc*, que se distingue por possuir uma quantidade pré-determinada de pontos de definição.

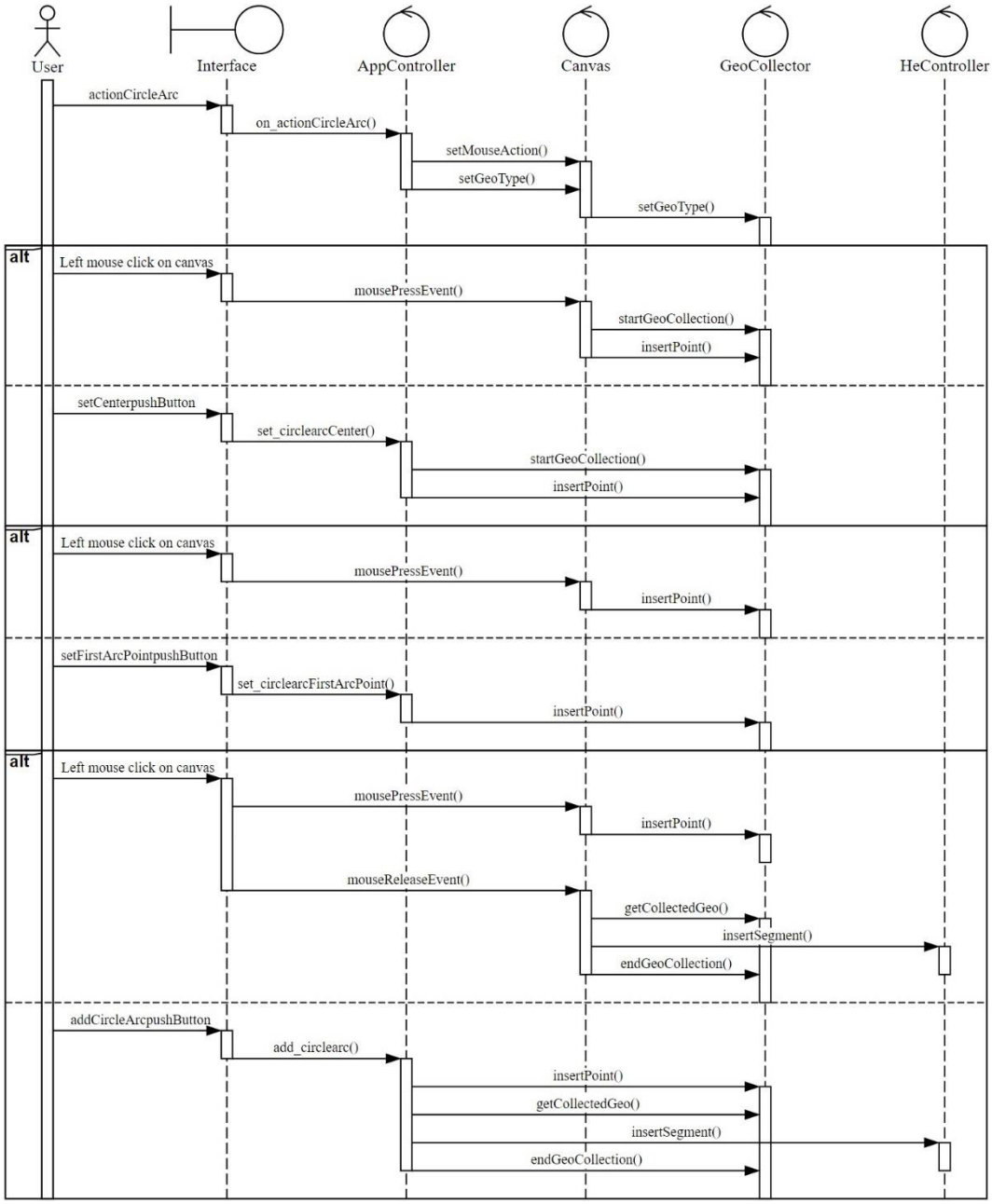

Figura 51: Diagrama de sequência para a coleta de uma curva da classe *CircleArc*.

6.3.3 Visualização de Entidades no *Canvas*

Após a coleta e inserção de uma curva, esta passa a ser uma entidade geométrica do modelo, especificamente do tipo segmento. O programa também inclui outros tipos de entidades geométricas, como pontos e regiões. Os pontos são localizados nos limites dos segmentos, enquanto que as regiões são automaticamente reconhecidas quando os segmentos formam uma área fechada. Todas essas entidades podem ser visualizadas na área de modelagem.

A classe Canvas possui várias funcionalidades, incluindo a renderização de objetos, que são as entidades mencionadas. O método *paintGL()* é encarregado de realizar essa tarefa. Sempre que alguma nova ação do usuário é realizada no FEMEP, a função *update()* da superclasse *QGLWidget* é acionada. Essa função atualiza a área de modelagem chamando o *paintGL()* para desenhar na tela novas entidades ou modificações que ocorreram nas entidades existentes. O *paintGL()* solicita as entidades ao *HeView*, através dos métodos *getPoints()*, *getSegments()* e *getPatches()*. O *HeView* acessa as entidades no *HeModel* e fornece ao *paintGL()*. A Figura 52 fornece o diagrama de sequência com os métodos executados para permitir a visualização de entidades.

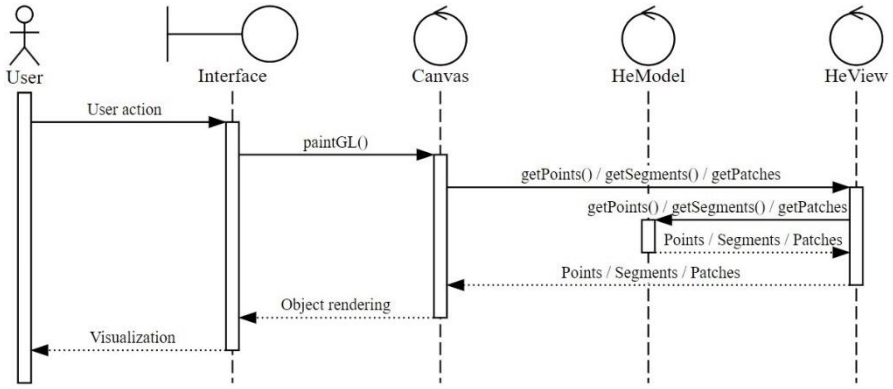

Figura 52: Diagrama de sequência para a visualização de entidades no modelo.

6.3.4 Propriedades NURBS das Curvas

Durante a modelagem, o método *insertPoint()* é chamado sempre que um ponto de definição é coletado. Esse método aciona a função *buildCurve()*, presente nas classes de curvas, que utiliza os pontos de definição como referências para a construção das curvas. Vale ressaltar que a implementação do método *buildCurve()*

é particular para cada tipo de curva, permitindo que o polimorfismo seja efetivamente aplicado. Nesse processo, são estabelecidas as propriedades NURBS, que incluem o grau polinomial, os pontos de controle, os pesos associados e o vetor de *knots*. O armazenamento dessas informações ocorre por meio de um objeto da biblioteca *geomdl* (https://pypi.org/project/geomdl/), passado na forma de um atributo denominado *nurbs*, presente nas classes de curvas. Essa biblioteca segue o padrão de POO e conta com métodos bastante úteis no âmbito de NURBS, que permitem, dentre outras coisas, a avaliação de coordenadas de pontos de uma curva, a avaliação de derivadas e refinamentos. A seguir será demonstrado a determinação das propriedades NURBS para cada uma das curvas presentes no programa.

Uma reta pode ser representada como uma NURBS de grau 1 e definida por dois pontos de controle, que possuem pesos associados de valores iguais. Dessa forma, a reta é na verdade uma B-Spline, que é um caso especial de uma NURBS. Para simplificar, adota-se valores unitários em todos os pesos. O vetor de *knots* para uma ordem polinomial 1 e dois pontos de controle é dado simplesmente por $\Xi = [0, 0, 1, 1]$. A Figura 53 demonstra a representação de uma reta NURBS.

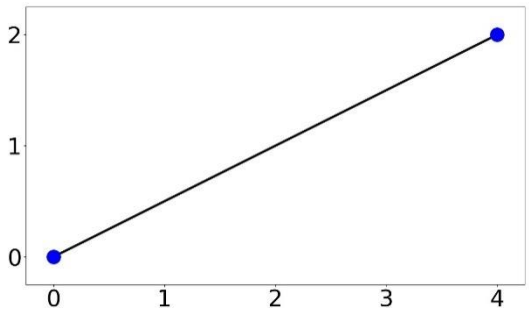

Figura 53: Reta representada na forma de uma curva NURBS linear.

Uma linha poligonal, na forma de uma NURBS, também apresenta grau de valor 1, mas pode possuir uma quantidade de pontos de controle maior do que dois. De forma análoga à reta, os pesos associados aos pontos de controle são todos iguais a 1. O vetor de *knots* é do tipo uniforme e, para sua determinação, é empregada a expressão da Seção 2.2.1, que relaciona a quantidade de termos desse vetor com a ordem polinomial (p) e o número de pontos de controle (n). A biblioteca *geomdl* disponibiliza o método *generate()*, que determina um vetor de *knots* uniforme recebendo como argumentos o grau da curva e a quantidade de pontos de controle. O método *generate()* é utilizado dentro da função *buildCurve()* da classe *Polyline*.

A Figura 54 ilustra o exemplo de uma linha poligonal NURBS de cinco pontos de controle, com vetor de *knots* $\Xi = \left[0, 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1, 1\right]$. Um detalhe interessante de observar é que os pontos de controle, destacados em azul, estão contidos na curva. Além disso, o polígono de controle coincide com a própria geometria. Essas são característica de uma NURBS linear, o mesmo pode ser notado para o caso da reta.

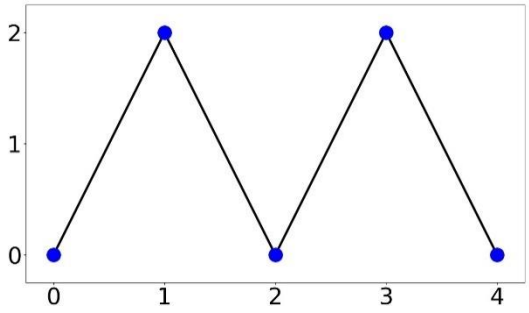

Figura 54: Linha poligonal representada na forma de uma curva NURBS linear.

Dentre as curvas presentes no FEMEP, encontra-se a B-Spline cúbica. Para que uma determinada B-Spline tenha ordem polinomial 3, é imprescindível a existência de ao menos quatro pontos de controle. No entanto, é possível que usuário deseje inserir uma quantidade menor de pontos de controle. Nesse caso, o programa gera curvas lineares ou quadráticas. Com dois pontos, a B-Spline é ajustada para grau 1; com três pontos, a B-Spline é definida com grau 2; e com quatro ou mais pontos, a B-Spline é sempre de grau 3. Em todos os casos, a B-Spline apresenta vetores de *knots* uniformes, determinados automaticamente pelo método *generate()* do *geomdl*. Na Figura 55, temos o exemplo de uma curva B-Spline cúbica de cinco pontos de controle, cujo vetor de *knots* é $\Xi = \left[0, 0, 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1, 1, 1\right]$.

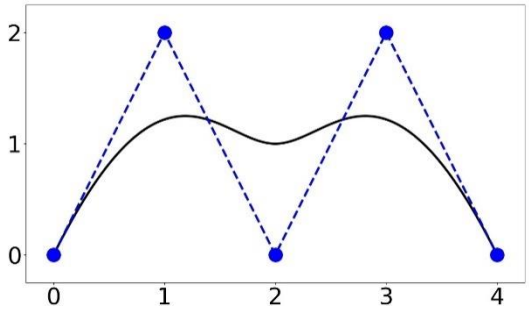

Figura 55: Representação de uma B-Spline cúbica.

Círculos podem ser construídos com curvas NURBS de diversas maneiras. A título de curiosidade, algumas técnicas mais complexas envolvem o uso de pesos de valor negativo, ou pontos de controle no infinito [4]. Neste trabalho, foi optado por representar círculos de acordo com o exemplo apresentado por Hughes et al. [4], que consiste na união de quatro arcos de 90°. Como resultado, tem-se a representação de um círculo através de uma NURBS quadrática de nove pontos de controle, de maneira que o polígono de controle forma um quadrado circunscrito, conforme o exemplo da Figura 56. O vetor de *knots* é dado por $\Xi = \left[0, 0, 0, \frac{1}{4}, \frac{1}{4}, \frac{1}{2}, \frac{1}{2}, \frac{3}{4}, \frac{3}{4}, 1, 1, 1\right]$, com quatro *spans* uniformes e multiplicidade 2 nos *knots* internos, indicando a união de quatro elementos de continuidade C^0. Os pesos associados são essenciais para permitir a formação de curvas cônicas, como no caso de um círculo. A Equação (129) fornece os pesos de acordo com o índice i do ponto de controle, basicamente há uma alternância de valores entre vértices e não vértices.

$$w_i = \begin{cases} 1 & para\ \ i\ \ ímpar \\ 1/\sqrt{2} & para\ \ i\ \ par \end{cases} \tag{129}$$

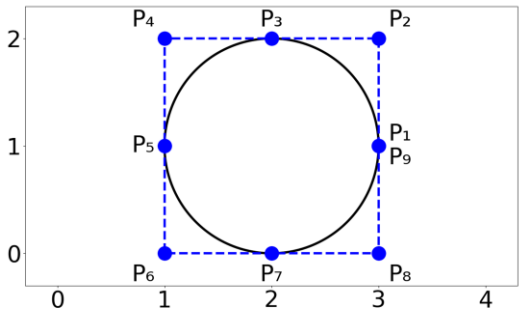

Figura 56: Representação de um círculo através de uma curva NURBS quadrática.

Na construção de arcos de círculo, adota-se um único elemento univariado NURBS para arcos com um ângulo central θ de até 90°. Arcos maiores requerem a combinação de múltiplos elementos. Um arco de círculo de até 90°, pode ser formado por uma curva NURBS quadrática com um vetor de *knots* $\Xi = [0, 0, 0, 1, 1, 1]$. Nesse caso, três pontos de controle são necessários: dois nas extremidades e o terceiro localizado na interseção das retas tangentes às extremidades, a uma distância d do centro do arco de círculo. A Figura 57a fornece uma representação visual dessa descrição. A determinação de d é realizada de maneira simples por trigonometria:

$$d = \frac{r}{cos(\theta_{loc}/2)} \tag{130}$$

Onde θ_{loc} é o ângulo de cada elemento de arco e r é o raio do arco de círculo. No caso particular de $0 < \theta \leq 90°$, temos que $\theta_{loc} = \theta$. Arcos com ângulo central superior a 90° são decompostos em arcos menores de ângulos congruentes. Para $90° < \theta \leq 180°$, realiza-se a decomposição do arco inicial em dois elementos de arco com $\theta_{loc} = \theta/2$. Este é representado por uma NURBS quadrática formada pela união de dois arcos menores que 90°. O vetor de *knots* apresenta dois *spans* e multiplicidade 2 nos *knots* internos: $\Xi = \left[0, 0, 0, \frac{1}{2}, \frac{1}{2}\ 1, 1, 1\right]$. Para um arco de círculo definido por $180° < \theta \leq 270°$, a decomposição é realizada em três elementos de arco com $\theta_{loc} = \theta/3$ e construído na forma de uma NURBS quadrática de vetor de *knots* $\Xi = \left[0, 0, 0, \frac{1}{3}, \frac{1}{3}, \frac{2}{3}, \frac{2}{3}, 1, 1, 1\right]$. Por fim, um arco de círculo com $270° < \theta \leq 360°$ apresenta $\theta_{loc} = \theta/4$, sendo representado por uma NURBS quadrática com $\Xi = \left[0, 0, 0, \frac{1}{4}, \frac{1}{4}, \frac{1}{2}, \frac{1}{2}, \frac{3}{4}, \frac{3}{4}, 1, 1, 1\right]$.

Nas Figura 57b, Figura 57c e Figura 57d observa-se exemplos de arcos de círculo que necessitam da decomposição em arcos menores para permitir uma representação adequada por meio de NURBS. Nota-se que nesses casos, os pontos de controle de elementos adjacentes são coincidentes. Uma vez determinado o ângulo θ_{loc} e a distância d, a montagem dos pontos de controle pode ser realizada facilmente, seguindo um processo similar à transformação de coordenadas polares em cartesianas. Quanto a determinação dos pesos associados, estes seguem um padrão de alternância similar ao caso do círculo completo:

$$w_i = \begin{cases} 1 & para\ i\ ímpar \\ cos(\theta_{loc}/2) & para\ i\ par \end{cases} \tag{131}$$

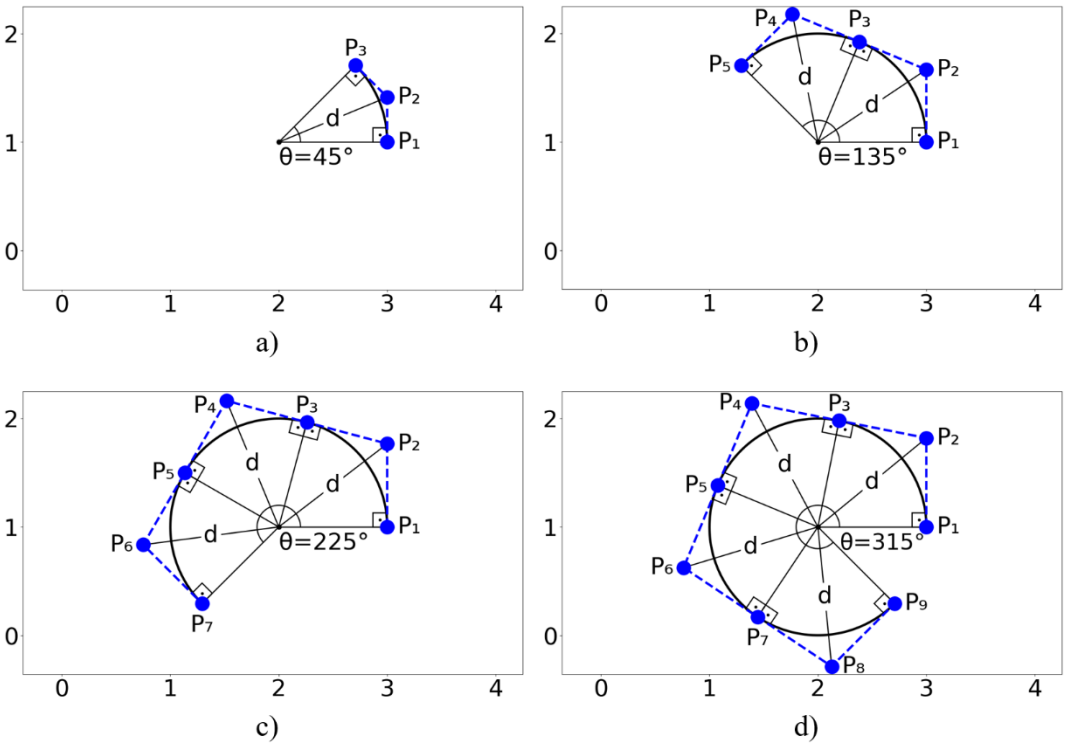

Figura 57: Arcos de círculos na forma de curvas NURBS. a) Arco de círculo com ângulo central $\theta = 45°$, formado por um único elemento NURBS. b) Arco de círculo com ângulo central $\theta = 135°$, construído por dois elementos de arco NURBS de $\theta_{loc} = 67.5°$. c) Arco de círculo com ângulo central $\theta = 225°$, representado por três elementos de arco NURBS de $\theta_{loc} = 75°$. d) Arco de círculo com ângulo central $\theta = 315°$, constituído de quatro elementos de arco NURBS de $\theta_{loc} = 78.75°$.

A representação de uma elipse por meio de NURBS se fundamenta na propriedade de covariância afim. Segundo essa propriedade, uma transformação afim aplicada a uma curva NURBS é obtida aplicando a transformação nos pontos de controle [4]. Uma elipse pode ser gerada através de uma transformação de escala, que é uma transformação afim, realizada em um círculo de raio unitário. Assim, para obter uma elipse NURBS, basta aplicar a transformação de escala nos pontos de controle de um círculo unitário. Essa transformação de escala deve ser executada na origem, portanto, se for desejada uma elipse rotacionada em torno do eixo z e deslocada na origem, será preciso realizar as transformações necessárias de rotação e translação, que também são transformações afins. A Figura 58 ilustra a sequência de transformações afins para a obtenção de uma elipse NURBS.

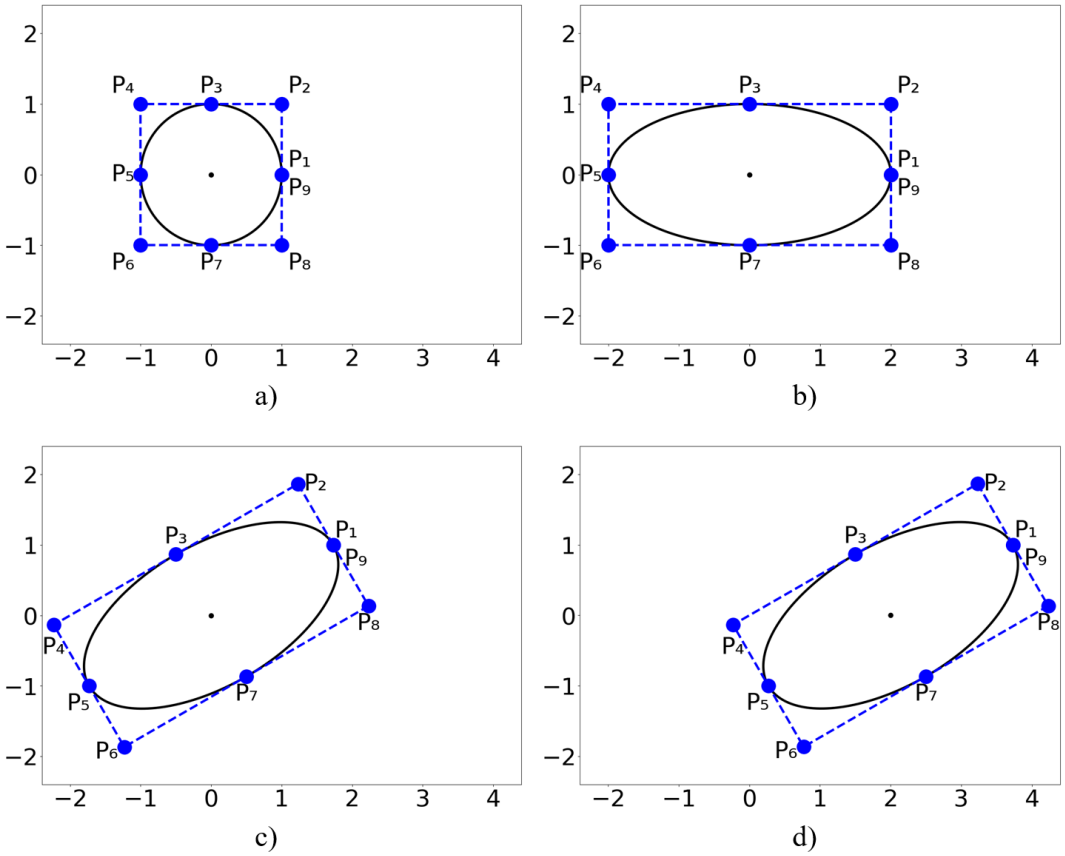

Figura 58: Transformações afins para criação de uma elipse a partir do círculo unitário.

A Equação (132) apresenta a matriz responsável por realizar a translação de um deslocamento Δx no eixo x e um deslocamento Δy no eixo y. Na Equação (133), é exposto a matriz de rotação de um ângulo ϕ em relação a origem em torno do eixo z. Já a equação (134), demonstra a matriz de transformação de escala, em que λ_x e λ_y representam os fatores de escala nos eixos x e y, respectivamente. Para o contexto da elipse, λ_x e λ_y são os comprimentos dos semieixos. Os pontos de controle de uma elipse na forma NURBS ($\boldsymbol{P}_i^{\boldsymbol{ellip}}$) são dados em função dos pontos de controle de um círculo unitário NURBS ($\boldsymbol{P}_i^{\boldsymbol{circ}}$), a partir de uma combinação das transformações afins abordadas, como mostrado na Equação (135).

$$\boldsymbol{T} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \tag{132}$$

$$\boldsymbol{R} = \begin{bmatrix} cos(\phi) & -sen(\phi) & 0 \\ sen(\phi) & cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{133}$$

$$\boldsymbol{S} = \begin{bmatrix} \lambda_x & 0 & 0 \\ 0 & \lambda_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{134}$$

$$\begin{bmatrix} x_i^{ellip} \\ y_i^{ellip} \end{bmatrix} = \boldsymbol{R}\,\boldsymbol{T}\,\boldsymbol{S} \begin{bmatrix} x_i^{circ} \\ y_i^{circ} \end{bmatrix} \tag{135}$$

A criação de arcos de elipse com curvas NURBS segue o mesmo princípio de aplicação das transformações afins descritas, nesse caso essas transformações são realizadas em arcos de círculos. A matriz de transformação de escala causa um efeito de distorção angular na elipse quando comparada ao círculo unitário. Esse efeito precisa ser corrigido para que um arco de elipse possa ser representado corretamente com NURBS. Deve-se encontrar um ângulo teórico θ_{theor} de forma que:

$$\begin{cases} r_{ellip} cos(\theta) = \lambda_x cos(\theta_{theor}) \\ r_{ellip} sen(\theta) = \lambda_y sen(\theta_{theor}) \end{cases} \tag{136}$$

Dividindo e simplificando as duas equações acima, obtém-se:

$$\theta_{theor} = arctan\left(\frac{sen(\theta)}{\lambda_y}, \frac{cos(\theta)}{\lambda_x}\right) \tag{137}$$

Após a determinação de θ_{theor}, constrói-se uma NURBS de arco de círculo com ângulo central de valor θ_{theor} e é realizado o procedimento de aplicação das transformações afins das Equações (132), (133) e (134). Observa-se na Figura 59 exemplos de arcos de elipse na forma de curvas NURBS.

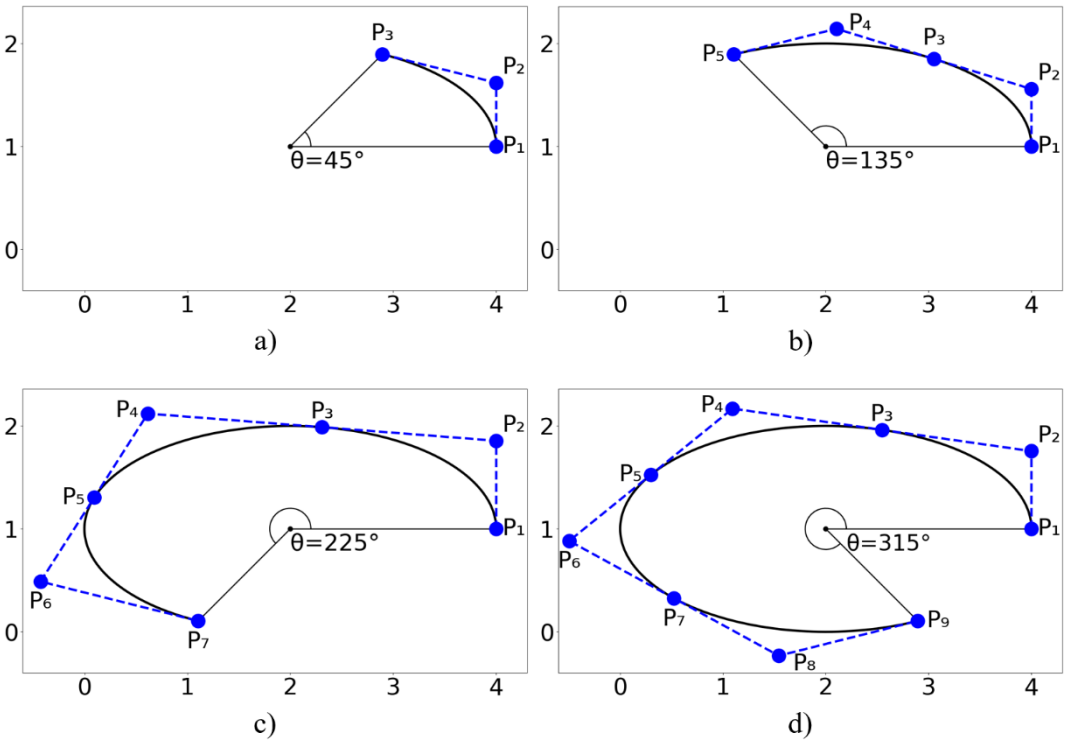

Figura 59: Arcos de elipse de diferentes ângulos centrais criados com NURBS.

6.3.5 Poligonais Equivalentes

Uma propriedade importante presente nas classes de curvas é a propriedade *eqPoly*, responsável por armazenar uma lista de pontos contidos em uma curva, que representam uma linha poligonal auxiliar. Essa linha poligonal é chamada de poligonal equivalente e é gerada de forma a garantir mais refinamento nas regiões de maior curvatura. A poligonal equivalente desempenha papel fundamental no programa, dentre suas atribuições, está a de prover informação para o desenho de sua curva no canvas. Além disso, poligonais equivalentes são usadas para calcular valores iniciais de interseções de curvas, que são posteriormente tratados para uma maior precisão.

O método estático *genEquivPolyline()* da classe *Curve* realiza a função de gerar uma poligonal equivalente. Esse método recebe como argumentos um objeto da curva em questão, uma lista e um valor de tolerância. Inicialmente, é realizado uma verificação condicional através do método *isStraight()* presente nas classes de curvas. O *isStraight()* averigua, com base na tolerância fornecida, se os pontos de controle da curva são colineares. Em caso positivo, é adicionado na lista o ponto

inicial da curva. Caso contrário, é realizado a divisão da curva no valor paramétrico 0.5, originando duas curvas metades. O método *genEquivPolyline()* é chamado para cada uma das duas curvas, resultando em um processo recursivo que finaliza com a obtenção da poligonal equivalente. Ao final, é necessário apenas acrescentar na lista o último ponto da curva inicial.

Localidades onde a curvatura é acentuada requerem mais iterações até que o método *isStraight()* retorne um valor booleano positivo. Por consequência, uma maior quantidade de pontos é adicionada nessas regiões. Essa característica torna mais precisa a representação de curvas por poligonal equivalente.

6.3.6 Visualização de Propriedades NURBS

O primeiro botão da barra secundária da interface é um *ToggleButton* que permite a seleção de entidades existentes no modelo. Quando esse elemento de interface é ativado, o *AppController* chama o método *on_actionSelect()*, que por sua vez aciona o método *setMouseAction()* da classe *Canvas* passando como argumento a *string* "SELECTION". A propriedade *curMouseAction* do *Canvas* é atualizada para armazenar a *string* fornecida. Realizado esse procedimento, a opção de opção de seleção é ativada no programa. O usuário pode clicar em uma entidade, ou clicar e arrastar o mouse na área de modelagem, criando um quadro de seleção. O método *mouseReleaseEvent()* do *Canvas* é chamado e aciona um dos métodos do *HeController*: *selectPick()*, se houve clique único; ou *selectFence()*, se houve quadro de seleção. Esses métodos verificam a entidade selecionada e configuram a seleção pela função *setSelected()*. Cada entidade geométrica apresenta a propriedade *selected* do tipo booleano, que armazena *True* se estiver selecionada, ou *False* caso não haja seleção.

O usuário pode clicar em uma entidade e pressionar o botão *Properties* no quadro dinâmico. Em seguida, os métodos *selectedVertices()*, *selectedEdges()* e *selectedFaces()* da classe *HeModel* verificam as seleções ativas. Com base nisso, o quadro dinâmico é atualizado para uma interface secundária de visualização, onde são exibidas propriedades geométricas e, para o caso de segmentos, propriedades NURBS (grau polinomial, vetor de *knots*, pontos de controle e pesos associados). A seleção de uma outra curva com o mouse realiza a atualização automática do

quadro dinâmico, que passa a mostrar as propriedades da nova curva. O programa requer que o usuário selecione uma entidade por vez para permitir a visualização de propriedades. Se mais de uma entidade estiver selecionada, há o devido tratamento para que nenhuma atualização do quadro dinâmico ocorra. A Figura 60 mostra a interface do quadro dinâmico de uma determinada curva inserida.

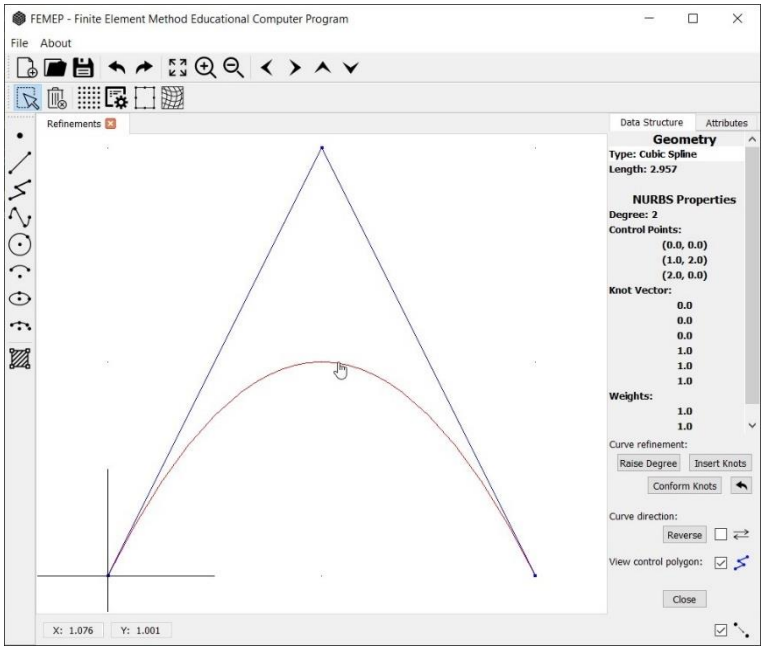

Figura 60: Quadro dinâmico com propriedades NURBS de uma curva genérica.

6.3.7 Refinamentos e Outros Recursos de Modelagem

Na região inferior do quadro dinâmico da Figura 60, observa-se alguns elementos de interface do tipo *PushButton* e *CheckBox*. Esses *widgets*, permitem a realização de refinamentos, no âmbito de análise isogeométrica, para uma dada curva selecionada. Também é possível inverter o sentido de uma curva, bem como visualizar um indicador do sentido dessa curva. Além disso, há a opção de visualização do polígono de controle.

O botão *Raise Degree* é responsável por aplicar a refinamento por elevação de ordem, exposto na seção 2.2.8.1. Ao ser pressionado pelo usuário, o método *degreeElevation()* é acionado na classe *AppController*. Em seguida, ocorre o direcionamento para o método de mesmo nome na classe *HeController*, que solicita ao *HeModel* o segmento em questão. Um outro método também denominado

degreeElevation(), pertencente a classe *Segment*, é chamado para realizar o refinamento na curva. A Figura 61 demonstra a aplicação da elevação de grau na curva apresentada na Figura 60.

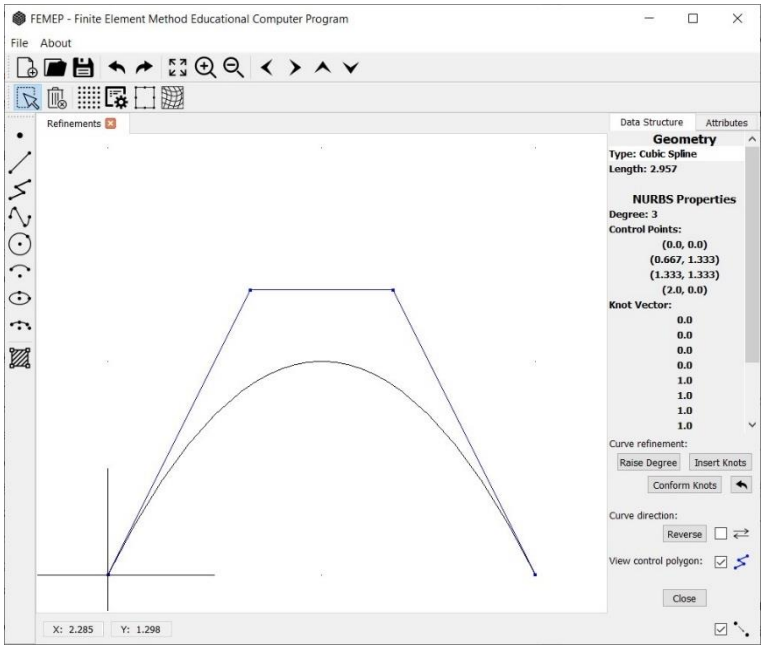

Figura 61: Refinamento por elevação de grau no FEMEP.

De forma similar, o botão *Insert Knots* pode ser pressionado pelo usuário para permitir a refinamento por inserção de *knots*, abordado na seção 2.2.8.2. O método *knotInsertion()* é acionado no *AppController*, que também direciona para o *HeController* e posteriormente para o *Segment*. Para que o usuário não necessite explicitar o *knot* a ser inserido, foi optado por realizar a inserção de *knots* localizados nos valores médios dos *spans*. A Figura 62 ilustra a realização desse refinamento na curva anterior (Figura 61).

O botão *Conform Knots* é útil quando se deseja que curvas de mesmo grau tenham o mesmo vetor de *knots*. O usuário deve realizar a seleção de múltiplas curvas e pressionar esse botão. O método *conformNurbsCurves()* da classe *Segment* é acionado, sendo este responsável por realizar o procedimento de inserir *knots* faltantes em relação à todas as curvas selecionadas.

Sempre que um refinamento é aplicado pela primeira vez em uma curva, a NURBS inicial fica armazenada na propriedade *originalNurbs* do *Segment*. Pode ser conveniente para o usuário querer resgatar a NURBS original. O botão *Rescue*

Curve oferece essa funcionalidade ao substituir a propriedade *nurbs* da curva pela propriedade *originalNurbs*. Isso possibilita que não seja necessário ter que apagar uma curva após uma série de refinamentos indesejados, por exemplo.

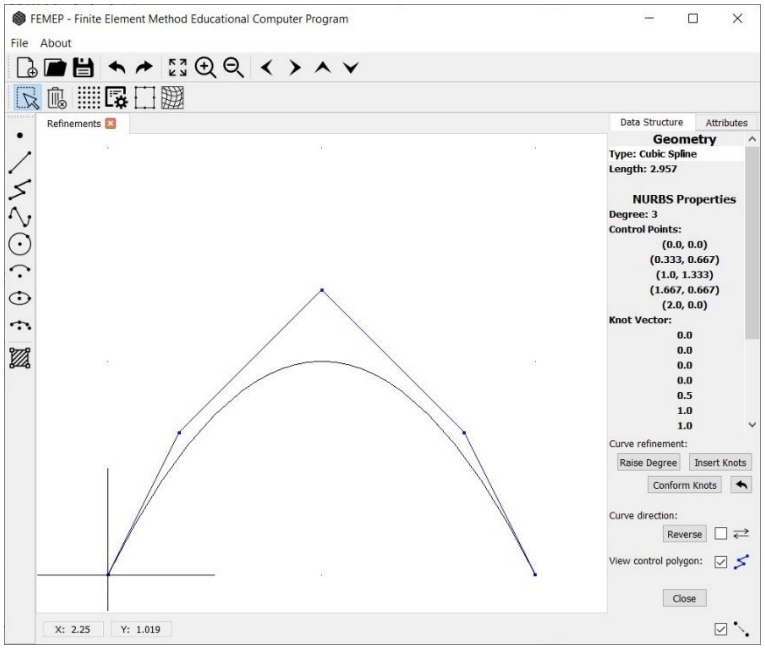

Figura 62: Refinamento por inserção de *knots* no FEMEP.

O botão *Reverse* realiza a inversão de sentido da NURBS de uma curva. O método *reverseNurbsCurve()* na classe *Segment* é chamado para inverter a ordem dos pontos de controle, de forma que não haja alteração na geometria. O *CheckBox* ao lado aciona o método *updateDirectionView()* da mesma classe *Segment*. Quando essa caixa de seleção é marcada ou desmarcada, ela fornece uma variável booleana *True* ou *False*, respectivamente. Essa variável é então passada para o *updateDirectionView()*, que atualiza a propriedade *directionView* do *Segment.* Durante a renderização das entidades na área de modelagem (seção 6.3.3), a propriedade *directionView* é verificada no método *paintGL()* da classe *Canvas*, para exibir ou não um indicador do sentido da curva.

O outro *CheckBox* com título "*View control polygon*", segue o mesmo princípio de funcionamento. Nesse caso, ele permite a visualização do polígono de controle de uma curva. De acordo com a marcação da caixa de seleção, o método *updateCtrlPolyView()* recebe um booleano que atualiza a propriedade *CtrlPolyView* do *Segment*. O *paintGL()*, por sua vez, verifica essa propriedade para poder traçar o polígono de controle.

6.3.8 Interseção de Curvas

A adição de uma curva no modelo pode resultar em interseções com curvas existentes. Nesse cenário, ocorre o devido tratamento para realizar a divisão das curvas nos pontos de interseção. Cada curva resultante é uma entidade própria, com suas devidas propriedades NURBS. Essa abordagem permite uma manipulação precisa e facilita a modelagem de superfícies a partir das curvas interseccionadas, como mostrado na Figura 63.

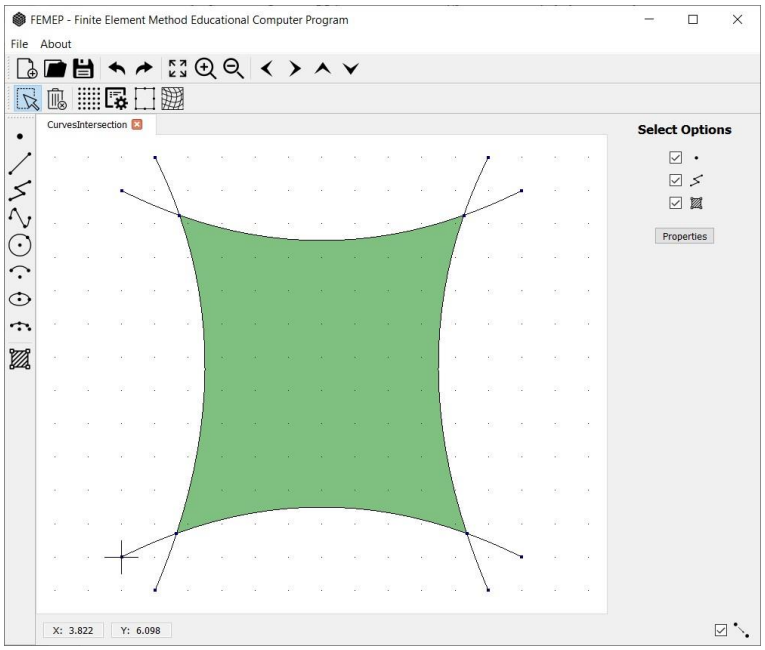

Figura 63: Interseções de curvas no FEMEP.

Após a coleta de uma curva, realizada por modelagem interativa (seção 6.3.2), é gerada uma instância de uma das classes de curvas presentes do FEMEP. Esse objeto é passado para o método *insertSegment()* do *HeController*, que cria uma entidade da classe *Segment* e adiciona o novo segmento no modelo. Nessa função, é verificado a existência de interseções com curvas no modelo através do método *intersectModel()* do *HeController*. Em seguida, prossegue-se para a divisão das curvas nos pontos de interseção por meio do método *splitExistingEdges()* da mesma classe *HeController*.

No *intersectModel()*, o *bounding box* da curva inserida é inicialmente definido. O modelo é examinado para identificar as curvas que estão contidas, ou que cortam esse *bounding box*. Assim, uma estrutura de repetição é criada para percorrer as curvas identificadas e determinar as interseções com a nova curva

adicionada. Para isso, as interseções são primeiramente estimadas pelas poligonais equivalentes através da função *computePolyPolyIntersection()* da classe *CompGeom*. Essa função recebe duas poligonais equivalentes e retorna os pontos de interseção, juntamente com as coordenadas paramétricas em cada uma das poligonais correspondentes a esses pontos. Posteriormente, a função *ParamCurveClosestPt()* da classe *Curve* é chamado para computar as coordenadas paramétricas pelo método de Newton-Raphson, tendo como base as coordenadas paramétricas das interseções nas poligonais.

O método *splitExistingEdges()* é acionado para realizar a divisão das curvas nos pontos de interseção. A função *split()*, pertencente a cada classe de curvas, é então chamada. Esta última é responsável por realizar a determinação das propriedades geométricas das novas curvas, que são passadas para o construtor da classe de curva correspondente, que retorna os objetos de curvas novas e independentes.

6.3.9 Geração Malhas Isogeométricas para NURBS

O reconhecimento de regiões fechadas é um dos recursos de modelagem do FEMEP. Após a detecção de uma região, a entidade topológica *patch* é adicionada no *HeModel*. A simples presença de um *patch* não o torna automaticamente uma superfície NURBS. Faz-se necessário que certas condições sejam satisfeitas e, em seguida, que seja executada uma rotina para determinar as propriedades NURBS da região. A construção de superfícies Coons (Seção 2.4) oferece uma excelente abordagem nesse sentido, onde as propriedades NURBS univariadas, das curvas de contorno, são utilizadas para determinar as propriedades NURBS bivariadas, da região em questão.

Para a geração de uma superfície Coons, é importante lembras das condições para sua formação. Primeiramente, são requeridas quatro curvas de contorno. Curvas opostas necessitam ter o mesmo grau polinomial e o mesmo vetor de *knots*. Os refinamentos da Seção 6.3.7 são ferramentas valiosas que o usuário pode empregar para manipular curvas e permitir a geração de superfícies NURBS com base em *patches* de Coons.

No FEMEP, esse processo é realizado através do botão de geração de malha, disponível na barra secundária. Quando acionado, o quadro dinâmico exibe um elemento de interface do tipo *ComboBox*, que disponibiliza opções para a geração de malha. Dentre as alternativas, encontram-se malhas de elementos finitos por mapeamento bilinear e trilinear transfinito [42], como também algoritmos de geração de malha via decomposição do domínio [19]. No caso em questão, a geração de uma superfície NURBS é realizada na opção *Isogeometric*.

A Figura 64 exibe a interface do programa para a geração de malhas. O usuário deve selecionar a região de interesse, escolher a opção *Isogeometric* e clicar no botão *Genetate Mesh*. As condições para a formação da superfície Coons devem ser previamente garantidas pelo usuário, caso contrário, exibe-se uma mensagem de erro. Relembra-se que essas condições consistem em curvas opostas com mesma ordem polinomial e mesmo vetor de *knots*. Casos a condição de mesmo *knot vector* em lados opostos não seja satisfeita, uma alternativa é a realização de uma decomposição do domínio, como é mostrado na Seção 6.3.10. Com isso, o método *generateMesh()* da classe *HeController* é acionado, que direciona para o método *generation()* da classe *MeshGeneration*, passando como argumento o tipo de malha escolhido.

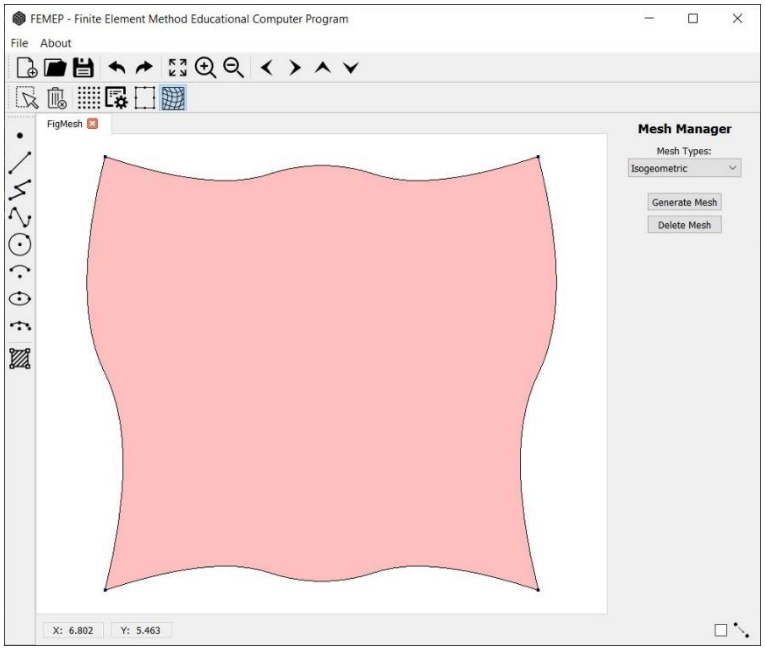

Figura 64: Interface de geração de malha do FEMEP.

Dentro da função *generation()*, são chamados os métodos correspondentes à opção de malha selecionada. No caso da opção *Isogeometric*, os métodos *getCoonsSurface()* e *isogeometricMesh()* são acionados. O *getCoonsSurface()* estabelece as propriedades NURBS do *patch* em questão, seguindo os procedimentos da Seção 2.4. Uma vez determinadas essas propriedades NURBS, gera-se um objeto da biblioteca *geomdl* a superfície NURBS. Esse objeto é armazenado na propriedade *nurbs* da classe *Patch*.

O método *isogeometricMesh()* é responsável por determinar as isocurvas, que são definidas por coordenadas paramétricas constantes em uma direção, ao longo da superfície. Os *knots* dos vetores de *knots* são selecionadas como essas coordenadas paramétricas constantes. Assim, essas curvas delimitam os elementos no espaço físico. Além disso, conectividade dos elementos também é estabelecida nesse método. A conectividade de um elemento é dada pelos pontos de controle, cujas funções de base associadas são diferentes de zero no domínio paramétrico do elemento. O algoritmo utilizado para estabelecer a conectividade é uma adaptação em Python do algoritmo de Nguyen et al. [41], presente no programa IGAFEM.

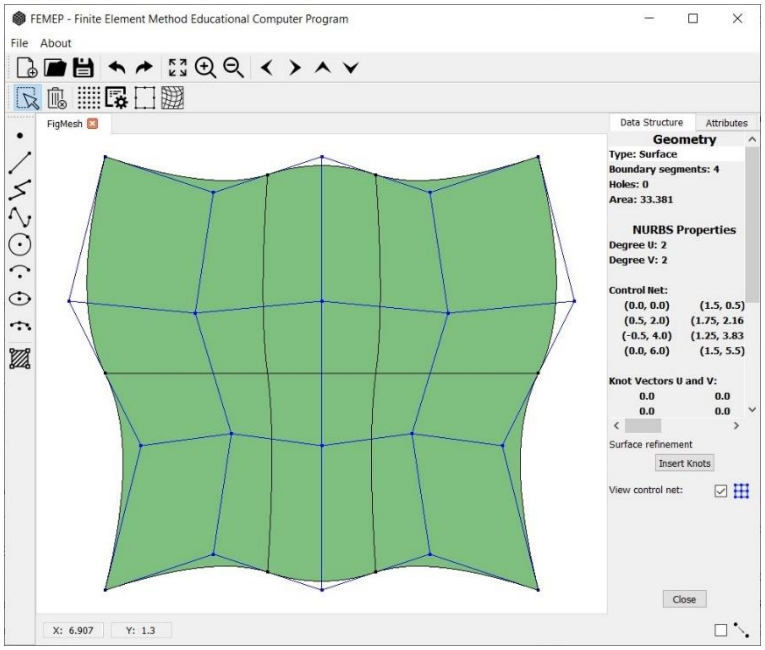

Figura 65: Superfície Coons gerada no FEMEP.

A Figura 65 ilustra uma região com uma superfície NURBS, gerada através do método Coons. Após a criação da superfície NURBS, as isocurvas são delineadas. Ademais, a interface de seleção passa a oferecer um botão que permite

realizar ao refinamento por inserção se *knots* na superfície. Adicionalmente, disponibiliza-se um *CheckBox* para visualizar a malha de controle, um recurso similar ao que ocorre em curvas.

6.3.10 Geração de Malhas Isogeométricas para T-Splines

Uma das possibilidades do FEMEP é a geração de malhas não estruturadas para análise isogeométrica, através de T-Splines com *extraordinary points*. Esse processo é voltado para casos em que a formação de *patches* NURBS por superfície Coons não é possível, devido a presença de *knot vectors* distintos em curvas opostas. Essa estratégia tem como partida o algoritmo de Miranda e Martha [19] de decomposição de regiões quadrilaterais para malhas não estruturadas de elementos finitos. A Figura 66 mostra os tipos de *templates* disponíveis e adotados nesse trabalho.

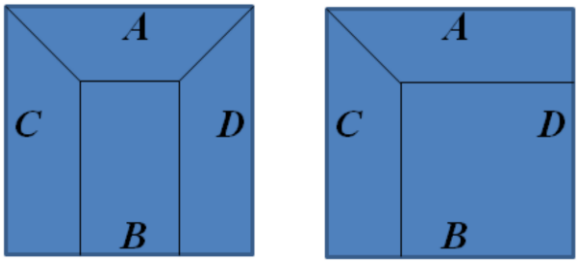

Figura 66: *Templates* de decomposição de regiões [19].

No contexto de elementos finitos, o algoritmo de templates de Miranda e Martha tem como dados de entrada um conjunto de pontos cartesianos ao longo das bordas de uma região quadrilateral. O *output* consiste em pontos cartesianos interiores, que formam a malha não estruturada. A Figura 67 ilustra esses pontos.

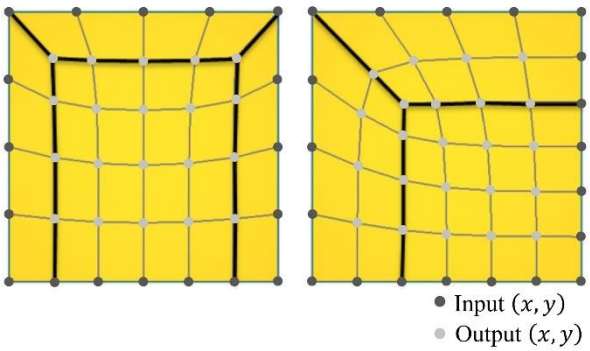

Figura 67: Dados de entrada e saída do algoritmo de Mirando e Martha [19].

No âmbito da análise isogeométrica, a ideia é formar um *patch* de T-Spline utilizando o algoritmo de templates para fornecer uma T-Mesh (malha de controle). A região deve ser definida por quatro curvas com grau polinomial cúbico, visto que a T-Spline deve ser de ordem cúbica, para permitir a presença dos *extraordinary points* descritos por Scott [17]. Para simplificação, requer-se também que as curvas tenham vetores de *knots* uniformes.

Pontos de controle das curvas de bordo são passados como *input* ao algoritmo de Miranda e Martha. Entretanto, emprega-se um pequeno artifício: alguns pontos de controle devem ser suprimidos durante a entrada de dados do algoritmo. Os pontos de controle suprimidos consistem naqueles que se encontram próximos à subdivisão dos subdomínios e contidos nas curvas de contorno, conforme indicado na Figura 68a.

A execução do algoritmo fornece como resultado os pontos de controle contidos no interior do *patch*, formando uma T-Mesh inicial, como apresentado na Figura 68b. Ocorre que, apesar dessa T-Mesh estar correta, a superfície T-Spline gerada teria bordas suavizadas na separação dos subdomínios, isto é, a T-Spline perderia as quinas, ficando com o aspecto semelhante ao da Figura 69a.

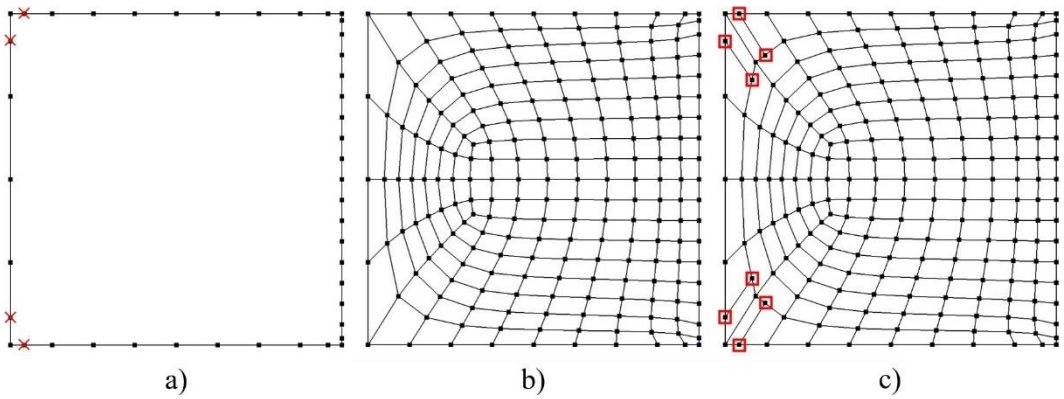

Figura 68: Processo utilizado para obtenção de uma T-Mesh com base no algoritmo de Miranda e Martha [19]. a) Pontos de controle passados como *input* ao algoritmo. Assinalados, encontram-se os pontos de controle suprimidos da entrada de dados. b) Pontos de controle obtidos após o *output*. c) Pontos de controle inseridos ao final do processo.

A solução encontrada foi introduzir determinadas T-Junctions, com base no trabalho de Casquero et al. [46], além de reintroduzir os pontos de controle incialmente suprimidos, formando uma T-Mesh final como ilustrado na Figura 68c. Assim, a superfície T-Spline, gerada a partir dessa T-Mesh, possui elementos nas

proximidades das quinas com redução de continuidade, tornando-se semelhante a T-Spline mostrada na Figura 69b, que preserva a geometria inicial da região.

É interessante notar que a T-Mesh obtida é sempre adequada para análise. Ao aplicar o procedimento de extensões das T-Junctions (Seção 4.1.9), observa-se que não ocorrem interseções, pois essas extensões serão sempre paralelas.

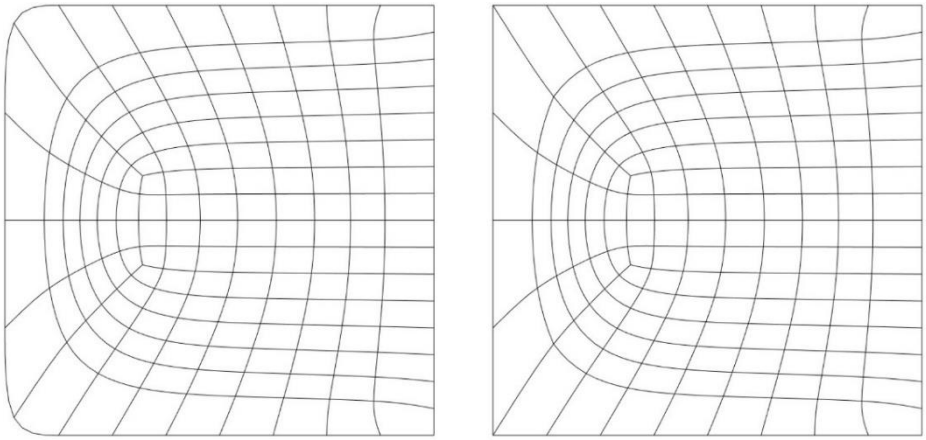

Figura 69: a) T-Spline com bordas suavidas. b) T-Spline corrigida.

Conforme discutido no Capítulo 4, uma T-Mesh por si só não fornece informação suficiente para a formação de uma T-Spline, sendo necessário o fornecimento adicional de informações paramétricas. Para T-Splines cúbicas (Seção 4.1.6), os intervalos paramétricos são atribuídos em cada aresta da T-Mesh (configuração de intervalos de *knots*). Ao longo das extremidades da T-Mesh, a distância paramétrica é nula, desempenhando um papel semelhante aos vetores de *knots* abertos em NURBS. Determinadas arestas que conectam as T-Junctions também são atribuídas com intervalo paramétrico nulo. Os demais segmentos da T-Mesh possuem intervalo paramétrico unitário, em virtude da presunção de curvas de contorno com *knot vectors* uniformes. A configuração de intervalos de *knots* é dada de acordo com a Figura 70. Nessa representação, os pontos em azul indicam as T-Junctions, enquanto os pontos em vermelho indicam os *extraordinary points*.

Vale lembrar que cada quadrilátero de arestas define os elementos da T-Mesh. No entanto, esses elementos não possuem correspondência com os elementos da T-Spline. Os elementos da T-Spline são delimitados por linhas de continuidade reduzida, dadas por extensões das T-Junctions, e devem possuir área paramétrica

diferente de zero. As extensões formam a chamada T-Mesh estendida (Seção 4.1.7) e são representadas pode linhas tracejadas. Na Figura 71, os elementos da T-Mesh estendida são destacados em cinza. As regiões com área paramétrica nula são indicadas em branco. Nesse caso em específico, as linhas de continuidade reduzida originaram novos elementos com área paramétrica zero.

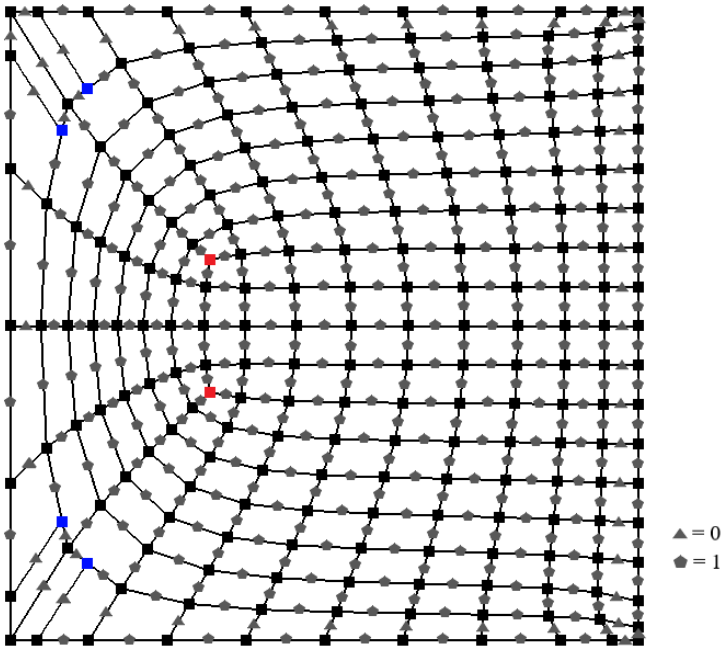

Figura 70: T-Mesh e configuração de intervalos de knots.

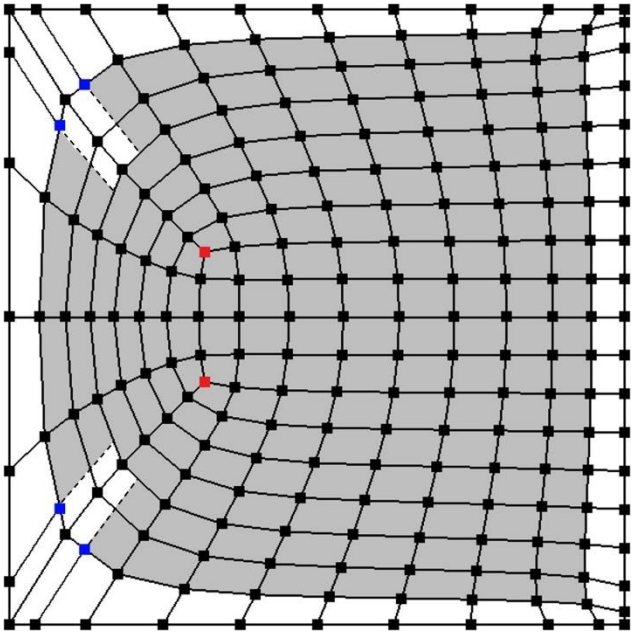

Figura 71: Elementos da T-Mesh estendida destacados. Regiões em branco indicam área paramétrica nula.

No FEMEP, a geração de *patchs* com T-Splines é realizada pela opção *Isogeometric* Template, disponível na interface de gerenciamento de malhas. A geração da T-Mesh é realizada pelo método *Msh2DTemplateIsogeometric()* da classe *MeshGeneration*. Essa função é responsável por determinar os segmentos da T-Mesh, como mostrado na Figura 72, bem como estabelecer a conectividade e os operadores de extração de cada elemento T-Spline.

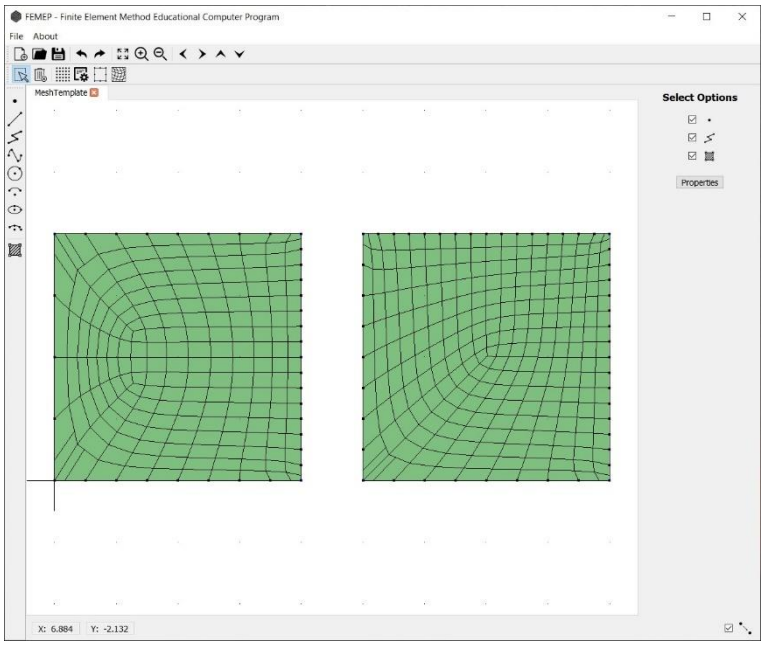

Figura 72: Patches com T-Meshes no FEMEP.

A determinação da conectividade em T-Splines não é algo trivial. O procedimento consiste em determinar os vetores de *knots* locais nos *anchors* (ou pontos de controle em T-Splines cúbicas) e verificar quais elementos estão contidos no domínio desses vetores. Para elementos afastados de T-Junctions e *extraordinary points*, a conectividade é obtida de forma simples. Uma vez que os vetores de *knots* locais possuem extensão em $(3 + 1)/2 = 2$ linhas de *knots*, para $p = 3$ (Seção 4.1.2), a conectividade é dada por pontos de controle no segundo entorno do elemento, conforme a Figura 73.

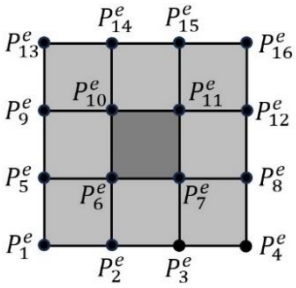

Figura 73: Conectividade de um elemento regular em T-Splines cúbicas.

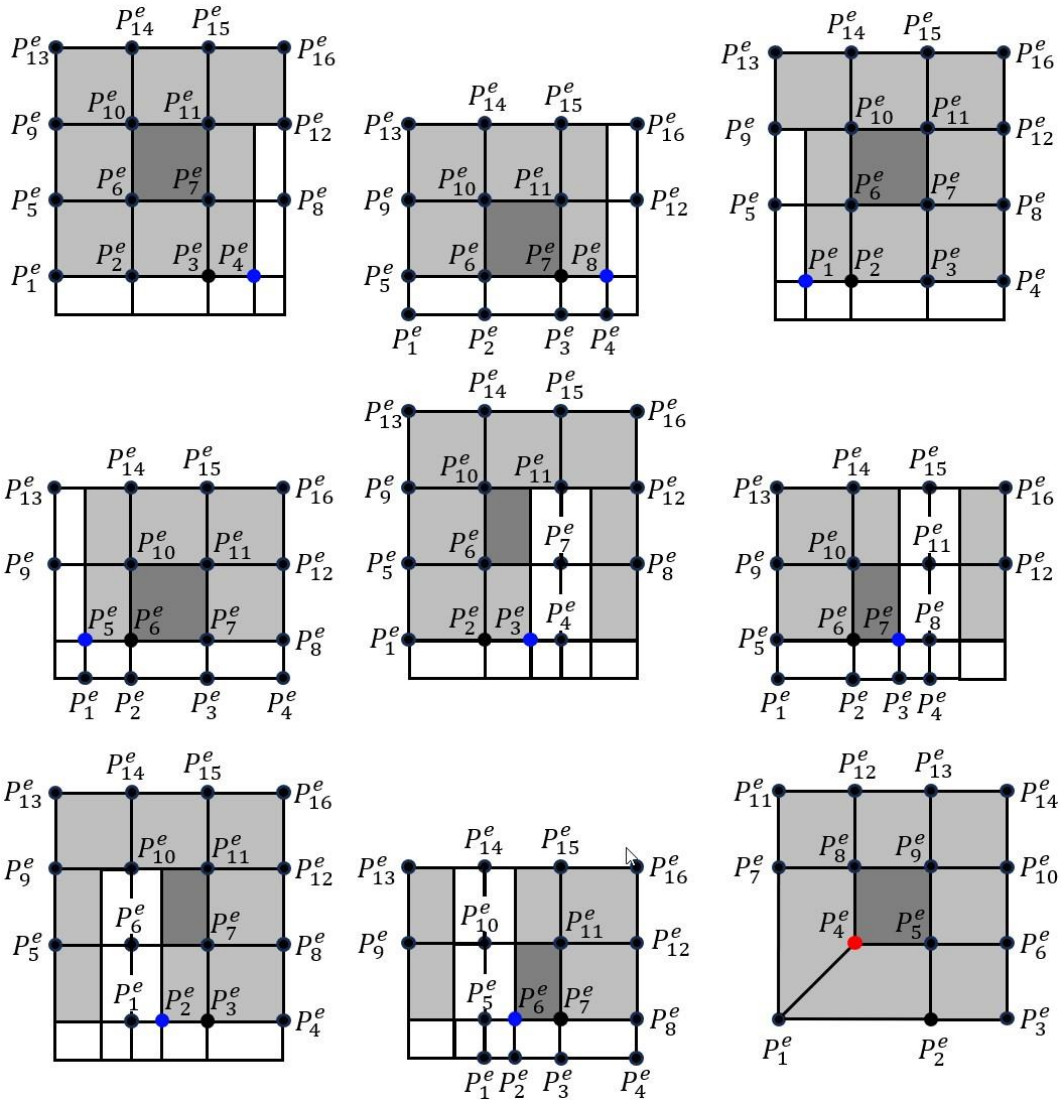

Figura 74: Diferentes casos de conectividade dos elementos T-Splines.

Para elementos próximos a T-Junctions, devem ser seguidos os procedimentos apresentados no Capítulo 4 para determinar a conectividade. Em se tratando de elementos próximo a *extraordinary points*, a conectividade também é dada por pontos de controle de vizinhança [17], embora a quantidade difira dos 16

usuais para T-Splines cúbicas. Particularmente aos padrões de T-Mesh desse trabalho, é possível estabelecer casos específicos de conectividade, como ilustrado na Figura 74.

Os operadores de extração, referentes a cada elemento da T-Spline, são então inferidos para cada um dos casos acima. No caso de elementos regulares, determina-se os operadores de acordo com o produto tensorial apresentado na Seção 5.1.5. Para elementos próximos das T-Junctions, segue-se o procedimento descrito na Seção 5.2.1. E para elementos próximos de *extraordinary points*, os operadores de extração são dados de acordo com a Seção 5.2.1.

6.4 Exportação para Análise

A análise dos modelos gerados pelo FEMEP é conduzida em um segundo software denominado FEMOOLab. A comunicação entre as duas ferramentas ocorre por meio de um arquivo de texto.

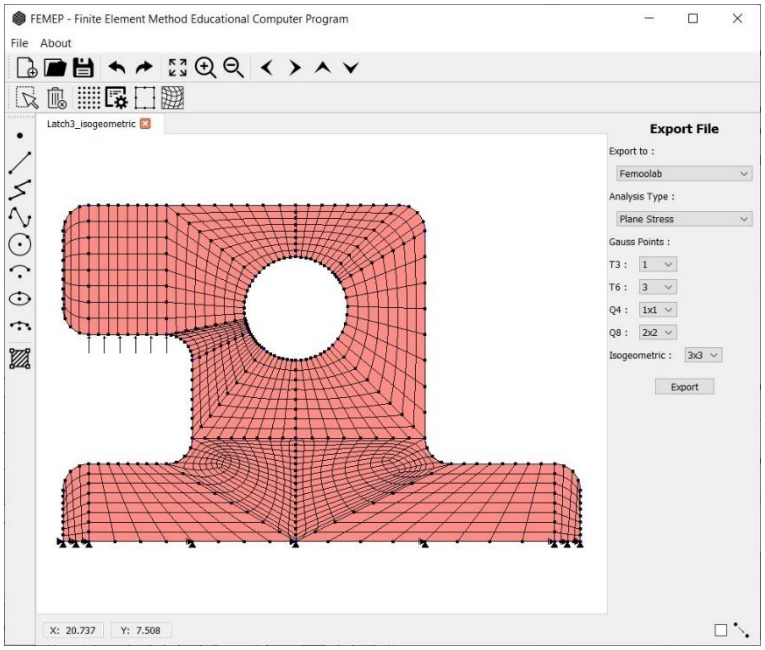

Figura 75: Interface de exportação no FEMEP.

Uma vez realizada a modelagem geométrica de um problema, o usuário deve acessar o gerenciamento de atributos para configurar as propriedades dos materiais da estrutura e definir as condições de contorno. Uma explicação mais

detalhada sobre o gerenciamento de atributos na versão inicial do FEMEP pode ser encontrada no trabalho de Bomfim [18].

O passo subsequente envolve o acesso à interface de exportação do arquivo, que pode ser feito ao selecionar a opção *Export* no menu *File*. A Figura 75 ilustra a interface em questão. Nesta etapa, o usuário tem a possibilidade de configurar a quantidade desejada de pontos de Gauss. Ao acionar o botão *Export*, um arquivo de texto é gerado, contendo todas as informações necessárias para a análise. Este arquivo inclui informações relativas aos pontos de controle e seus pesos associados, vetores de knots de cada superfície, ordens polinomiais, conectividade dos elementos e, no caso de T-Splines, os operadores de extração.

7 Software de Análise FEMOOLab

Neste capítulo, é apresentado a ferramenta FEMOOLab (*Finite Element Method Object-Oriented Laboratory* - https://github.com/rlrangel/FEMOOLab), que realiza a análise isogeométrica dos modelos previamente gerados no FEMEP. A ferramenta em questão é desenvolvida em MATLAB sob o paradigma de POO.

Da maneira semelhante ao FEMEP, o software FEMOOLab foi inicialmente projetado para análise isoparamétrica tradicional de elementos finitos. A iniciativa de incorporar a análise isogeométrica teve início nesse trabalho, com publicações realizadas ao longo de elaboração dessa pesquisa [47, 48]. Como será mostrado, a arquitetura de POO foi um grande facilitador para efetuar essa tarefa. A presença de classes e métodos permite uma melhor organização do código, com o armazenamento eficiente das informações geométricas, e a execução de etapas específicas da análise isogeométrica.

7.1 Módulos do Programa

O modelo a ser analisado é definido pela classe *Model*. Essa classe possui propriedades que armazenam objetos de outras classes presentes no programa, com informações essenciais para a análise isogeométrica. As propriedades gerais incluem um objeto da classe *Anm_PlaneStress*, que determina a análise por estado plano de tensões, e um objeto da classe *Results*, que contém os resultados obtidos.

Além disso, a classe *Model* conta com propriedades referentes ao modelo. Objetos de pontos de controle e elementos são armazenadas na classe *Model*. Cada ponto de controle possui uma propriedade que é uma referência para uma instância de uma das classes *Node_Isogeometric* ou *Node_Isogeometric_Bezier_Extraction*, e cada elemento apresenta uma propriedade que é uma referência para uma instância de uma das classes *Element_Isogeometric* ou *Element_Isogeometric_Bezier_Extraction*. As propriedades dos materiais ficam salvas na classe *Material*, que possui uma instância contida no *Model*. A classe *Surface* é empregada para armazenar superfícies do modelo e que também tem um objeto armazenado no *Model*. Similarmente, a classe *Gauss_Quad*, usada para a determinação da quadratura de Gauss, contém sua instância no *Model*. A leitura das informações do arquivo texto, exportado pelo FEMEP, é realizada pela classe *Read*.

As informações geométricas referentes à análise são essencialmente armazenadas nas classes de elementos e pontos de controle.

O programa conta ainda com as classes *Anl_LinearSteadyState* e *Plot_StructInPlane*. A *Anl_LinearSteadyState* é responsável por executar as etapas de processamento e pós-processamento. A *Plot_StructInPlane* tem a função de exibir na rela os resultados da classe *Results*. Isso inclui a configuração deformada, além dos deslocamentos e tensões, exibidos por um mapa de cores.

7.1.1 Classe *Element*

A classe *Element* atua como uma superclasse para os elementos. Existem três subclasses de elementos: *Element_isoparametric*, *Element_Isogeometric* e *Element_Isogeometric_Bezier_Extraction*. A *Element_isoparametric* é utilizada apenas em análise de elementos finitos isoparamétrica tradicional. As outras duas classes, *Element_Isogeometric* e *Element_Isogeometric_Bezier_Extraction*, são utilizadas na análise isogeométrica para diferentes tipos de problemas. A *Element_Isogeometric* é empregada para problemas modelados apenas com patches de NURBS, enquanto a *Element_Isogeometric_Bezier_Extraction* é usada em modelos que apresentam ao menos uma região de T-Splines. Neste último caso, o método da extração de Bézier é operado, e todas as regiões, incluindo as regiões NURBS, são analisadas com extração de Bézier.

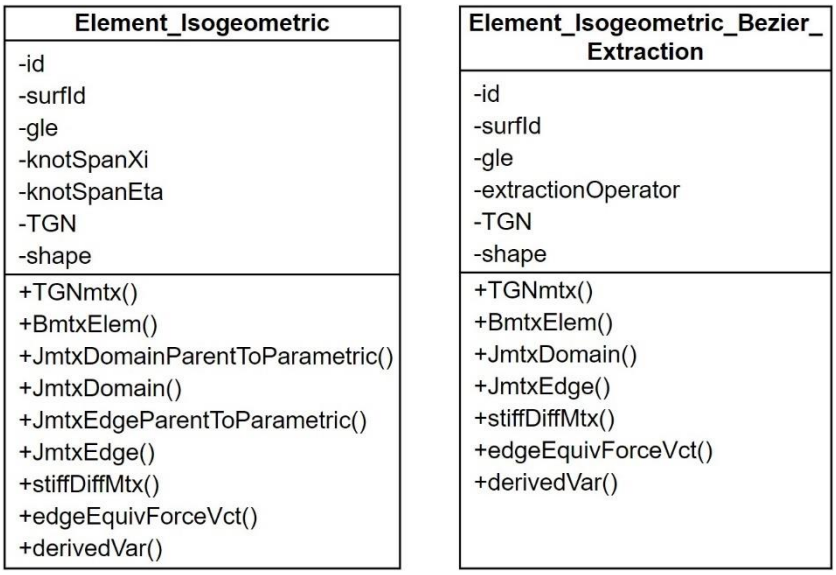

Figura 76: Classes *Element_Isogeometric* e *Element_Isogeometric_Bezier_Extraction*.

A classe *Element_Isogeometric* possui várias propriedades. A propriedade *id* armazena um número de identificação para o elemento, e *surfId* fornece o número id da superfície em que o elemento está contido. A propriedade *gle* armazena a numeração dos graus de liberdade do elemento. As propriedades *knotspanXi* e *knotspanEta* são vetores que definem os intervalos de *knots* do elemento nas direções ξ e η, respectivamente. A propriedade *TGN* armazena a matriz de extrapolação das tensões dos pontos de Gauss para a extremidade do elemento. Finalmente, a propriedade *shape* é um objeto de uma das subclasses da classe Shape (Seção 7.1.2). A classe *Element_Isogeometric_Bezier_Extraction* compartilha a maioria dessas propriedades, mas em vez de *knotspanXi* e *knotspanEta*, ela possui a propriedade *extractionOperator*, que contém o operador de extração do elemento. A Figura 76 mostra as propriedades e métodos presentes nessas duas classes. Os métodos são descritos na Seção 7.3, que trata da execução da análise.

7.1.2 Classe *Shape*

A classe *Shape*, de igual modo, possui subclasses associadas que especificam o tipo de elemento a ser analisado. No contexto de análise de elementos finitos isoparamétrica, as subclasses *Shape_Tria3*, *Shape_Tria6*, *Shape_Quad4* e *Shape_Quad8* representam, respectivamente, os elementos T3, T6, Q4 e Q8.

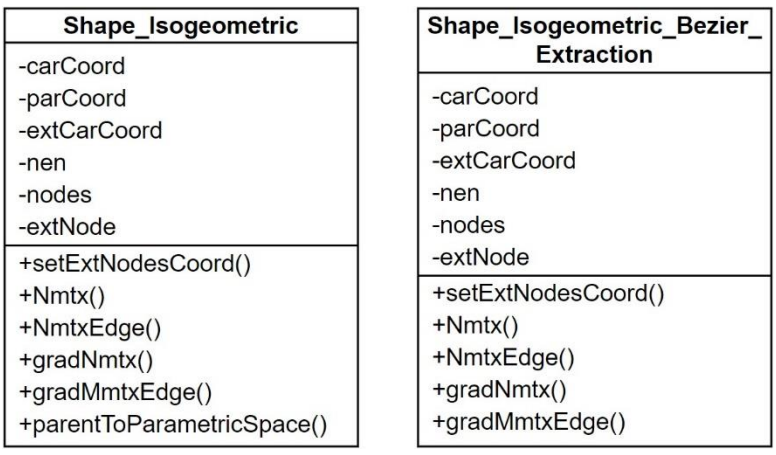

Figura 77: Classes *Shape_Isogeometric* e *Shape_Isogeometric_Bezier_Extraction*.

Para análise isogeométrica, as subclasses *Shape_Isogeometric* e *Shape_Isogeometric_Bezier_Extraction* são utilizadas. Estas classes armazenam, entre outras informações, as conectividades do elemento e possuem métodos para

avaliar as funções base e suas derivadas. A principal distinção entre elas é que a *Shape_Isogeometric_Bezier_Extraction* disponibiliza os polinômios de Bernstein, enquanto a *Shape_Isogeometric* fornece as funções de base NURBS em si.

A propriedade *carCoord* armazena as coordenadas dos pontos de controle de conectividade do elemento. A *parCoord* define as coordenadas no espaço *parent* das posições dos pontos de extrapolação. Os pontos de extrapolação são definidos como os pontos para onde as tensões são extrapoladas pela matriz TGN, a partir dos pontos de Gauss. A variável *nen* contém o número de pontos de controle de conectividade. *Nodes* é um vetor de objetos dos pontos de controle de conectividade, e *extNode* são instâncias dos pontos de extrapolação. Na Figura 77, observa-se os principais atributos e métodos dessas classes.

7.1.3 Classe Node

A *Node* é uma superclasse concebida para acomodar os nós, no caso isoparamétrico, ou pontos de controle, no caso isogeométrico. As informações dessas entidades são efetivamente armazenadas em uma das subclasses: *Node_isoparametric*, *Node_Isogeometric* ou *Node_Isogeometric_Bezier_Extraction*. Essas classes não possuem métodos implementados, pois sua função é de apenas realizar o armazenamento das informações.

As classes *Node_Isogeometric* e *Node_Isogeometric_Bezier_Extraction* apresentam as mesmas propriedades: a propriedade *id*, que contém um número de identificação para os pontos de controle; *coord*, que armazena as coordenadas cartesianas; *weight*, que dispõem do peso associado; *fixdDOF*, um vetor de booleanos que especifica se os graus de liberdade no ponto de controle estão fixos ou livres. Além disso, possuem a propriedade *elem*, que armazena um vetor de objetos de elementos que fazem conectividade com o ponto de controle em questão.

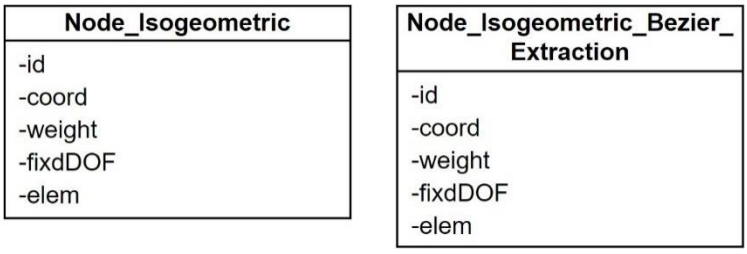

Figura 78: Classes *Node_Isogeometric* e *Node_Isogeometric_Bezier_Extraction*

7.2 Múltiplos *Patches*

Modelagem em várias regiões ou *patches* pode ser conveniente em muitos problemas, especialmente para modelos complexos. Em alguns casos, a estrutura de produto tensorial de NURBS pode não ser adequada para representar determinadas geometrias [4]. Nesse sentido, a divisão de um domínio em subdomínios se torna uma alternativa pertinente.

Neste estudo, assumimos compatibilidade geométrica entre *patches* adjacentes. Isso significa que os pontos de controle ao longo de uma determinada curva de contorno de uma região têm correspondência de um para um com os pontos de controle da curva de uma região adjacente. Essa correspondência resulta nos chamados *patches* conformes. A Figura 79 ilustra essa compatibilidade.

Dois ou mais pontos de controle em uma mesma localização devem ser interpretados como uma única entidade. Por consequência, durante a montagem da matriz de rigidez global e do vetor de forças de um problema, as variáveis de controle locais são consideradas entidades equivalentes no contexto global. O resultado é a formação de um único sistema, permitindo a junção dos *patches*.

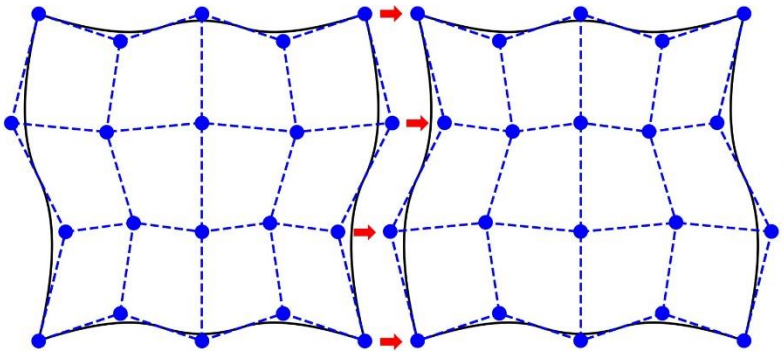

Figura 79: *Patches* conformes.

7.3 Processamento

Em se tratando de métodos numéricos para a resolução de equações diferenciais, a análise isogeométrica emprega uma estrutura computacional semelhante a análise por elementos finitos. O diferencial entre esses dois métodos reside nas rotinas das funções de base, como também nos mapeamentos empregadas na análise.

No contexto da ferramenta FEMOOLab, a POO é um recurso bastante explorado na implementação do programa. Esse paradigma elimina a necessidade de alguns procedimentos comumente adotados em código tradicionais de elementos finitos. Por exemplo, esses códigos normalmente utilizam uma matriz de conectividade IEN, onde cada linha corresponde a um elemento específico e as colunas determinam os nós de conectividade. Sempre que é necessário acessar a conectividade de um elemento, é realizada a consulta nessa matriz. Com a orientação a objetos, cada elemento pode ser tratado como um objeto com propriedades individuais, promovendo maior organização e eficiência no código. As diferentes etapas da análise são segmentadas em métodos, que consultam de forma simples as propriedades de um objeto.

Na etapa de processamento são determinadas a matriz de rigidez e o vetor de forças do problema, que são empregados para a resolução do sistema da Equação (71). O método *process()* da classe *Anl_LinearSteadyState* é responsável por executar o processamento. Inicialmente, esse método faz a chamada da função *gblStiffMtx()* presente na classe *Anm_PlaneStress*, para a montagem da matriz de rigidez global. Um laço é realizado para percorre os objetos de elementos. A cada iteração, aciona-se o método *stiffDiffMtx()* do elemento, que retorna a matriz de rigidez local correspondente. De semelhante modo, o vetor de forças é determinado pelo método *addEquivForce()* da classe *Model*, que aciona o método *edgeEquivForceVct()* dos elementos.

Nas Seções 7.3.1 e 7.3.2, apresentam-se os algoritmos em MATLAB utilizados para determinar as matrizes de rigidez na análise isogeométrica convencional e na análise isogeométrica por extração Bézier. Os procedimentos para a determinação das funções de base e os mapeamentos entre os domínios, necessários para a integração numérica, são detalhados de um ponto de vista computacional.

7.3.1 Análise Isogeométrica Convencional

A matriz de rigidez local é determinada via integração numérica pelo método da quadratura Gaussiana. Na análise isogeométrica convencional, é necessário efetuar a transformação dos pontos de Gauss, definidos no espaço

parent, para o espaço paramétrico da superfície NURBS. Isso implica na necessidade de incorporar um Jacobiano adicional, tal como explicado na Seção 3.5.3.

```
% Numerical integration of stiffness matrix part accounting for the
% diffusive term: [B]'[C][B]h
function K = stiffDiffMtx(this,surface)
    % Initialize element matrix
    ndof = this.anm.ndof;
    nen = this.shape.nen;
    K = zeros(nen*ndof,nen*ndof);

    % Material constitutive matrix
    C = this.anm.Cmtx(this);

    % Gauss points and weights
    [ngp,w,gp] = this.gauss.quadrature(this.gsystem_order);

    % Loop over Gauss integration points
    for i = 1:ngp
        % Parent coordinates
        r = gp(1,i);
        s = gp(2,i);

        % Element knot spans
        xiSpan  = this.knotSpanXi;
        etaSpan = this.knotSpanEta;

        % Element parametric coordinates
        xi  = this.shape.parentToParametricSpace(xiSpan,r);
        eta = this.shape.parentToParametricSpace(etaSpan,s);

        % Jacobian matrix parent to parametric space
        J1 = this.JmtxDomainParentToParametric;

        % Jacobian matrix parametric to physical space
        [J2,Rders] = this.JmtxDomain(surface,xi,eta);

        % Gradient matrix
        B = this.BmtxElem(J2,Rders,xi,eta);

        % Rigidity coefficient at integration point
        h = this.anm.rigidityCoeff(this,xi,eta);

        % Accumulate Gauss point contributions
        K = K + w(i) * B' * C * B * h * det(J1) * det(J2);
    end
end
```

Figura 80: Método de determinação das matrizes de rigidez locais.

A Figura 80 mostra o método *stiffDiffMtx()* da classe *Element_Isogeometric*, que determina a matriz de rigidez local de um dado elemento. O procedimento envolve uma sequência de passos, iniciando com uma estrutura de repetição em

torno no número de pontos de Gauss. Esses pontos, que consistem em coordenadas do espaço *parent*, são então transformados para o espaço paramétrico pela Equação (49). Essa transformação é realizada pelo método *parentToParametricSpace*, implementado na classe *Shape_Isogeometric* e apresentado na Figura 81. Para essa operação, é necessário fornecer como argumentos os intervalos de *knots* que definem o elemento, informações contidas nas propriedades *knotspanXi* e *knotspanEta*. O próximo passo é calcular a matriz Jacobiana associada a esse mapeamento, conforme a Equação (50), implementada no método *JmtxDomainParentToParametric*, disponível na Figura 82.

```
% Transformation of coordinates from the parent domain to the
% parametric domain.
function xi = parentToParametricSpace(~,range,pt)
    xi = 0.5 * ((range(2) - range(1)) * pt + range(2) + range(1));
end
```

Figura 81: Método de transformação entre os domínios *parent* e paramétrico.

```
% Compute Jacobian matrix associated with the mapping from the
% parent domain to the parametric domain.
function J1 = JmtxDomainParentToParametric(this)
    % Element knot spans
    xiSpan  = this.knotSpanXi;
    etaSpan = this.knotSpanEta;

    % Jacobian matrix
    J1 = [0.5*(xiSpan(2) - xiSpan(1)), 0.0;
          0.0, 0.5*(etaSpan(2) - etaSpan(1))];
end
```

Figura 82: Método para determinação da matriz Jacobiana associada ao mapeamento entre os domínios *parent* e paramétrico.

Em seguida, determina-se a segunda matriz Jacobiana, associada à transformação do espaço paramétrico para o espaço físico, dada pela Equação (53), implementada na função *JmtxDomain* e apresentada na Figura 83. Nessa etapa, realiza-se a determinação das primeiras derivadas das funções de base. Recorreu-se ao algoritmo do trabalho de Nguyen et al. [41] utilizado no programa IGAFEM. Trata-se de um algoritmo em C++ para maior eficiência, que se comunica com o MATLAB. As coordenadas dos pontos de controle de conectividade também são necessárias durante a execução desse método. A propriedade *carCoord* da classe *Shape_Isogeometric* contém essa informação, sendo acessada nesse momento.

```
% Compute Jacobian matrix associated with the mapping from the
% parametric domain to the physical domain in a position given by
% parametric coordinates.
function [J2,Rders] = JmtxDomain(this,surface,xi,eta)
    % Surface properties
    p = surface.degreeXi;
    q = surface.degreeEta;
    knotVectorXi = surface.knotVectorXi;
    knotVectorEta = surface.knotVectorEta;
    weights = surface.weights;

    % Matrix of basis functions derivatives
    Rders = this.shape.gradMmtx(xi,eta,p,q,knotVectorXi,...
        knotVectorEta,weights);

    % Cartesian coordinates matrix
    X = this.shape.carCoord;

    % Jacobian matrix
    J2 = Rders * X;
end
```

Figura 83: Método para o cálculo da matriz Jacobiana do mapeamento entre os domínios paramétrico e físico.

As derivadas e a matriz Jacobiana, determinadas na função *JmtxDomain*, são passadas como argumento para o método *BmtxElem*, mostrado na Figura 84, que determina a matriz deformação-deslocamento do elemento por meio da inversa da matriz Jacobiana. O método *Bmtx* da classe *Anm_PlaneStress* é chamado para reorganizar nas posições adequadas os elementos da matriz.

```
% Compute gradient matrix in a position of element domain given by
% parametric coordinates.
function B = BmtxElem(this,J2,Rders,xi,eta)
    % Matrix of d.o.f. shape functions derivatives w.r.t.
    % cartesian coordinates
    GradNcar = J2 \ Rders;

    % Gradient matrix
    B = this.anm.Bmtx(this,GradNcar,xi,eta);
end
```

Figura 84: Método para determinação da matriz deformação-deslocamento.

Finalmente, é possível prosseguir para o cálculo da matriz de rigidez local por integração numérica, conforme a Equação (74). A implementação é realizada ao final do código da Figura 80.

7.3.2 Análise Isogeométrica com Base em Extração Bézier

Neste trabalho, a técnica da extração Bézier é empregada na análise de modelos com *patches* de T-Splines. Na verdade, se o modelo for um misto de *patches* NURBS e T-Splines, todos os elementos são analisados por extração Bézier. Curiosamente, durante a determinação da matriz de rigidez local, em nenhum momento é indicado ao programa que determinado elemento é pertencente a uma região NURBS ou uma região T-Spline. Necessita-se apenas que as conectividades e os operadores de extração sejam fornecidos apropriadamente, sem nenhuma distinção.

```
% Numerical integration of stiffness matrix part accounting for the
% diffusive term: [B]'[C][B]h
function K = stiffDiffMtx(this,surface)
    % Initialize element matrix
    ndof = this.anm.ndof;
    nen = this.shape.nen;
    K = zeros(nen*ndof,nen*ndof);

    % Material constitutive matrix
    C = this.anm.Cmtx(this);

    % Polynomial orders
    p = surface.degreeXi;
    q = surface.degreeEta;

    % Gauss points and weights
    [ngp,w,gp] = this.gauss.quadrature(this.gsystem_order);

    % Loop over Gauss integration points
    for i = 1:ngp
        % Parent coordinates
        xi = gp(1,i);
        eta = gp(2,i);

        % Jacobian matrix
        [J,Rders] = this.JmtxDomain(p,q,xi,eta);

        % Gradient matrix
        B = this.BmtxElem(J,Rders,xi,eta);

        % Rigidity coefficient at integration point
        h = this.anm.rigidityCoeff(this,xi,eta);

        % Accumulate Gauss point contributions
        K = K + w(i) * B' * C * B * h * det(J);
    end
end
```

Figura 85: Método de determinação das matrizes de rigidez locais por extração Bézier.

O método *stiffDiffMtx()* da classe *Element_Isogeometric_Bezier_extraction* é empregado na determinação da matriz de rigidez local por extração Bézier, sendo apresentado na Figura 85.

```
        % Compute Jacobian matrix in a position of element domain given by
        % parametric coordinates.
        function [J,Rders] = JmtxDomain(this,p,q,xi,eta)
            % Weight vector
            ctrlpts = this.shape.nodes;
            weights = arrayfun(@(x) x.weight, ctrlpts);

            % Diagonal weight matrix
            W = diag(weights);

            % Bernstein polynomials
            Bernstein = this.shape.Mmtx(xi,eta,p,q);
            Bernstein = Bernstein';

            % Matrix of Bernstein polynomials derivatives
            Bernsteinders = this.shape.gradMmtx(xi,eta,p,q);
            BernsteinderXi = Bernsteinders(1,:)';
            BernsteinderEta = Bernsteinders(2,:)';

            % Extraction operator
            Ce = this.extractionOperator;

            % Weight function
            weightsb = Ce' * weights;
            Wfunc = weightsb' * Bernstein;

            % Weight function derivatives
            WfuncderXi = weightsb' * BernsteinderXi;
            WfuncderEta = weightsb' * BernsteinderEta;

            % Matrix of basis functions derivatives
            component1 = 1/Wfunc * BernsteinderXi - ...
                WfuncderXi * Bernstein / Wfunc^2;
            RderXi = W * Ce * component1;

            component2 = 1/Wfunc * BernsteinderEta - ...
                WfuncderEta * Bernstein / Wfunc^2;
            RderEta = W * Ce * component2;

            Rders = [RderXi'; RderEta'];

            % Cartesian coordinates
            X = this.shape.carCoord;

            % Jacobian matrix
            J = Rders * X;
        end
```

Figura 86: Método para o cálculo da matriz Jacobiana para análise com extração Bézier.

Comparando-se o algoritmo da Figura 85 com o algoritmo da Figura 80, nota-se a ausência do Jacobiano associado ao mapeamento entre os domínios *parent* e paramétrico. Isso ocorre, pois na extração Bézier as funções de base são determinadas com base nos polinômios de Bernstein, que são definidos no domínio $[-1, 1]$ em cada direção (Seção 5.1.1), o mesmo intervalo em que ocorre a integração numérica. Nesse caso, apenas o Jacobiano do mapeamento entre os domínios paramétrico e físico é calculado. O método *JmtxDomain()* mostrado na Figura 86 realiza essa etapa.

A diferença fundamental entre a análise isogeométrica convencional e a extração Bézier, reside na determinação das derivadas das funções de base. A extração Bézier utiliza as funções de Bernstein e operadores de extração, bem como os pesos associados aos pontos de controle de conectividade. A rotina da Figura 86 resume bem as formulações das funções de base da Seção 5.1.4.

Na extração Bézier, nota-se que a estrutura geral do cálculo de matriz de rigidez se mantém análoga a de elementos finitos. A única necessidade é a adaptação da rotina para as funções de forma. Isso representa uma vantagem em termos de implementação.

7.4 Pós-Processamento

Em uma análise estática linear elástica por elementos finitos, após a determinação da matriz de rigidez e do vetor de forças, um sistema linear é resolvido. A solução resulta em um vetor de deslocamentos nodais correspondentes aos graus de liberdade do problema. Vale ressaltar que os graus de liberdade estão intrinsecamente associados aos nós do modelo. Dependendo do tipo de elemento adotado, os nós podem estar localizados em diferentes posições, mas sempre contidos no interior da geometria de um elemento. Um elemento Q4, por exemplo, possui quatros nós localizados nos seus vértices. Um elemento Q8, por sua vez, apresenta quatros nós nos vértices e quatro nós nos pontos médios das arestas. O pós-processamento em elementos finitos, envolve a determinação das tensões no nós do elemento. As tensões são inicialmente calculadas nos pontos de Gauss do elemento e, posteriormente, extrapoladas para os nós.

Na análise isogeométrica, os graus de liberdade estão associados aos pontos de controle, que, em geral, não estão contidos na geometria do elemento. Dessa forma, um procedimento adicional é necessário para o pós-processamento. Determina-se os chamados pontos de extrapolação, situados em localidades do elemento que seguem o padrão dos nós de elementos isoparamétricos. Essa estratégia foi utilizada por Nguyen et al. [41], que adotaram em seu trabalho o padrão de um elemento Q4. Nesse estudo, optou-se por pontos de extrapolação que seguem a localização de nós de um elemento Q8, embora a adoção de qualquer outro padrão também fosse possível. A Figura 87 ilustra de maneira clara um exemplo de um elemento e seus pontos de controle, pontos de Gauss e pontos de extrapolação.

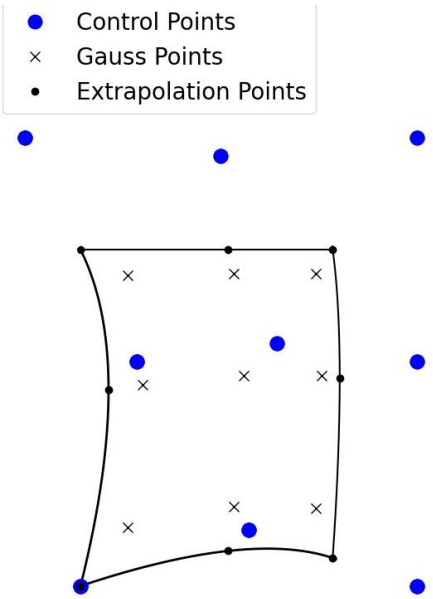

Figura 87: Pontos de controle, pontos de Gauss e pontos de extrapolação de um elemento.

Esses pontos de extrapolação são determinados de maneira simples. Recorre-se ao mapeamento da Equação (54), que transforma coordenadas do espaço *parent* para o espaço físico. Para o padrão de um elemento Q8, essas coordenadas são dadas por: $[-1,-1]$, $[1,-1]$, $[1,1]$, $[-1,1]$, $[0,-1]$, $[1,0]$, $[0,1]$ e $[-1,0]$.

A extrapolação é realizada pela definição de uma superfície de suavização de três coeficientes no espaço *parent* do elemento, como mostrado no Equação

(138). O objetivo é calculá-los pelo método de aproximação dos mínimos quadrados [49-51].

$$\boldsymbol{\tau}_s(\tilde{\xi},\tilde{\eta}) = c_1 + c_2\tilde{\xi} + c_3\tilde{\eta} \tag{138}$$

Define-se uma matriz $\boldsymbol{S}$ que determina os três coeficientes a partir de valores de tensões nos pontos de Gauss:

$$\boldsymbol{S} = \begin{bmatrix} 1 & \sum_{i=I}^{ngp}\tilde{\xi}_i & \sum_{i=I}^{ngp}\tilde{\eta}_i \\ \sum_{i=I}^{ngp}\tilde{\xi}_i & \sum_{i=I}^{ngp}\tilde{\xi}_i\tilde{\xi}_i & \sum_{i=I}^{ngp}\tilde{\xi}_i\tilde{\eta}_i \\ \sum_{i=I}^{ngp}\tilde{\eta}_i & \sum_{i=I}^{ngp}\tilde{\xi}_i\tilde{\eta}_i & \sum_{i=I}^{ngp}\tilde{\eta}_i\tilde{\eta}_i \end{bmatrix}^{-1} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \tilde{\xi}_I & \tilde{\xi}_{II} & \cdots & \tilde{\xi}_{ngp} \\ \tilde{\eta}_I & \tilde{\eta}_{II} & \cdots & \tilde{\eta}_{ngp} \end{bmatrix} \tag{139}$$

Onde o par $(\tilde{\xi}_i, \tilde{\eta}_i)$ determina as coordenadas de um ponto de Gauss e ngp é o número de pontos de Gauss adotados. Define-se também uma matriz de avaliação $\boldsymbol{E}$, que contém as coordenadas do no espaço *parent* dos pontos de extrapolação:

$$\boldsymbol{E} = \begin{bmatrix} 1 & -1 & -1 \\ 1 & 1 & -1 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix} \tag{140}$$

A extrapolação das tensões é efetuada pela relação:

$$\boldsymbol{\tau}_{extPts} = \boldsymbol{E}\,\boldsymbol{S}\,\boldsymbol{\tau}_{GausPts} \tag{141}$$

Onde $\boldsymbol{\tau}_{extPts}$ são as tensões nos pontos de extrapolação, que se deseja obter, e $\boldsymbol{\tau}_{GausPts}$ as tensões nos pontos de Gauss.

Para a determinação dos deslocamentos nos pontos de extrapolação, o procedimento difere da determinação das tensões. Os deslocamentos são obtidos

diretamente pela resolução do sistema linear, e são dados nos pontos de controle do modelo. Pode-se, portanto, obtê-los pela Equação (45), aplicando as mesmas coordenadas anteriores: $[-1,-1]$, $[1,-1]$, $[1,1]$, $[-1,1]$, $[0,-1]$, $[1,0]$, $[0,1]$ e $[-1,0]$.

O pós-processamento é realizado pelo método *posProcess()* da classe *Anl_LinearSteadyState*, onde são chamados métodos da classe *Anm_PlaneStress* para executar os próximos passos. Inicialmente, os pontos de extrapolação dos elementos são determinados pelo método *extPointsCoords()*. Em seguida, o método *gaussDerivedVar()* computa as tensões nos pontos de Gauss e os deslocamentos nos pontos de extrapolação. Posteriormente, os resultados das tensões nos pontos de Gauss são extrapolados para os pontos de extrapolação pela relação definida na Equação (141), realizada no método *elemDerivedVarExtrap()*. A matriz de extrapolação fica armazenada na propriedade TGN das classes de elementos. Finalmente, os resultados nos pontos de extrapolação dos elementos são suavizados no contexto global pelo método *nodeDerivedVarExtrap()*.

7.5 Exemplos de Modelos e Resultados

A ferramenta é demonstrada com alguns modelos de análise estática linear elástica, considerando materiais isotrópicos e estado plano de tensões. O estudo consiste em realizar refinamentos por inserção de *knots* nos modelos e verificar componentes de tensões máximas de interesse. Dessa forma, é possível averiguar a convergência dessas tensões de acordo com a quantidade de graus de liberdade. Uma comparação com a modelagem isoparamétrica de elementos finitos também é realizada para mostrar a maior precisão da análise isogeométrica. De forma simultânea, os mesmos problemas também são modelados com malhas não estruturadas de T-Splines com *extraordinary points*, que permitem maior refinamento nas regiões de interesse.

7.5.1 Placa Infinita com Furo Central

O problema consiste em uma placa infinita com um furo circular central submetida a uma tensão de tração uniaxial e constante, conforme a Figura 88. Devido à simetria, a modelagem é realizada considerando um quarto da placa. As

dimensões adotadas no modelo são denotadas por LxL, com $L = 4\ in$, e espessura $t = 1\ in$. O orifício circular possui um raio R de valor unitário. As propriedades do material são definidas pelo módulo de elasticidade $E = 10^5\ psi$ e pelo coeficiente de Poisson $\nu = 0.3$. A carga uniaxial T_x assume magnitude de 10 lbf/in.

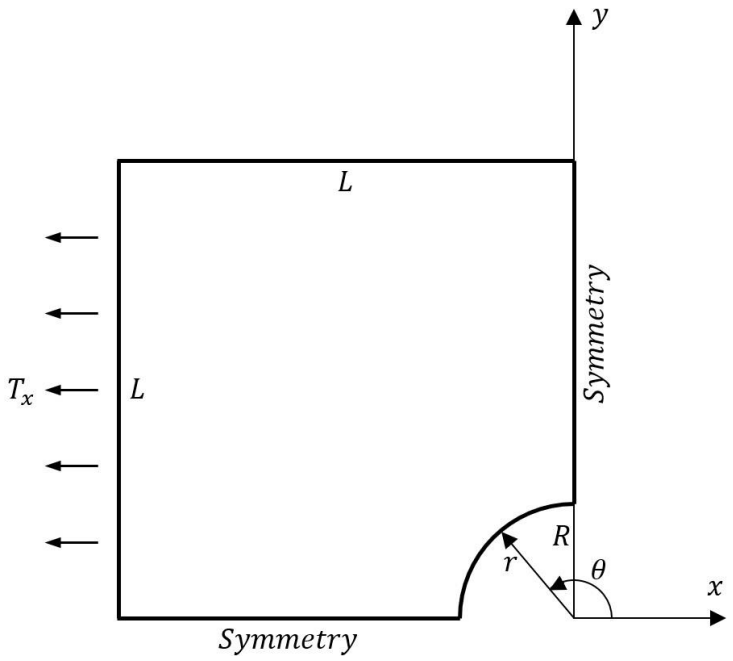

Figura 88: Placa com furo circular central.

A escolha desse problema específico decorre da sua simplicidade inerente, que permite a possibilidade de uma análise comparativa com a solução analítica [52, 41], dada por:

$$\sigma_{xx}(r,\theta) = T_x\left[1 - \frac{R^2}{r^2}\left(\frac{3}{2}cos2\theta + cos4\theta\right) + \frac{3}{2}\frac{R^4}{r^4}cos4\theta\right] \quad (142)$$

$$\sigma_{yy}(r,\theta) = -T_x\left[\frac{R^2}{r^2}\left(\frac{1}{2}cos2\theta - cos4\theta\right) + \frac{3}{2}\frac{R^4}{r^4}cos4\theta\right] \quad (143)$$

$$\sigma_{xy}(r,\theta) = T_x\left[-\frac{R^2}{r^2}\left(\frac{1}{2}sen2\theta + sen4\theta\right) + \frac{3}{2}\frac{R^4}{r^4}sen4\theta\right] \quad (144)$$

A solução é válida para uma tração no infinito, entretanto, o modelo é restrito a uma região finita no entorno do furo. Nesse sentido, para fins de verificação de convergência, é útil considerar as componentes de tensões analíticas nas condições de contorno naturais do problema. As condições de contorno são

dadas por: $\bar{\boldsymbol{t}} = \left[-\sigma_{xx}, -\sigma_{xy}\right]^T$ na face esquerda, $\bar{\boldsymbol{t}} = \left[\sigma_{xy}, \sigma_{yy}\right]^T$ na face superior, $u_x = 0$ na face direita e $u_y = 0$ na face inferior.

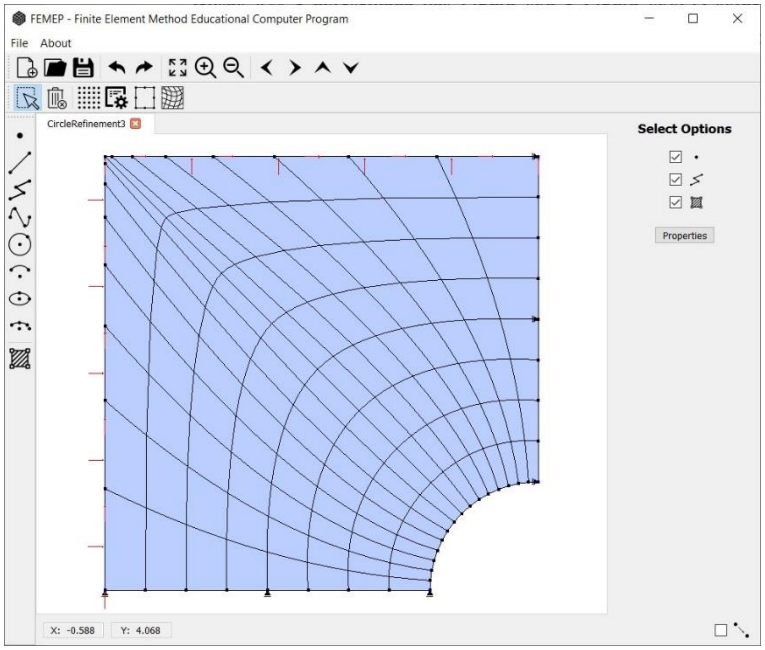

Figura 89: Placa com furo circular modelada no FEMEP com um *patch* de NURBS.

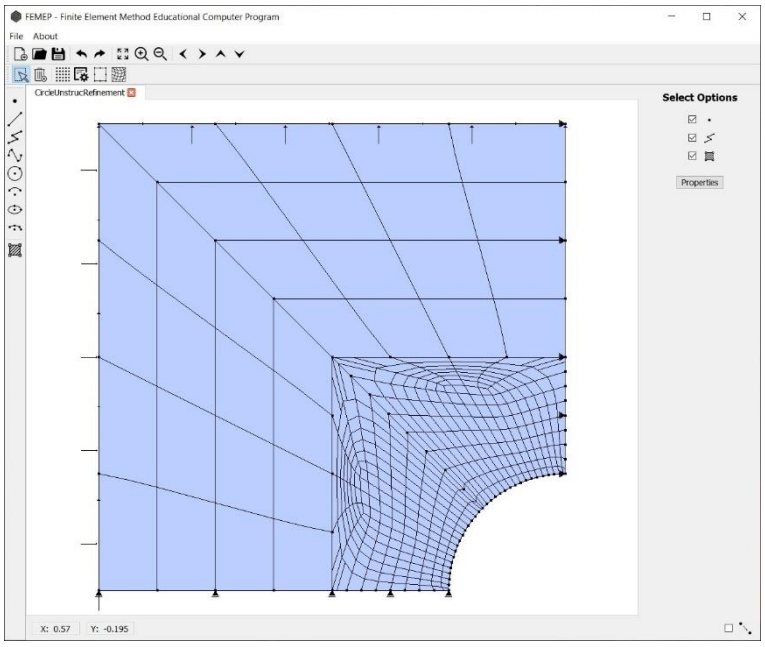

Figura 90: Placa com furo circular modelada no FEMEP com *patches* de NURBS e T-Splines.

Na Figura 89, o exemplo em questão é modelado pelo FEMEP através de um *patch* de NURBS biquadráticas. Um arco de círculo e uma B-Spline definem as

curvas de contorno na direção ξ; já as curvas da direção η são formadas por duas retas submetidas a elevação de grau (ver Seção 2.2.8.1).

Na Figura 90, a modelagem é realizada por meio de quatro regiões. Dois *patches* são de T-Splines bicúbicas com malhas não estruturadas nas proximidades do furo, enquanto dois *patches* são formados por superfícies NURBS cúbicas nas duas direções. As mesmas curvas que definem o modelo da Figura 89 são utilizadas, sendo as regiões subdividias internamente por curvas do tipo reta. Antes dos refinamentos por inserção de *knots*, são aplicadas as elevações de grau necessárias.

O FEMOOLab foi usado para realizar a análise, a distribuição da componente de tensão σ_{xx} é apresentada na Figura 91. No ponto superior ao orifício e contido na face, a solução analítica do problema fornece uma tensão máxima $\sigma_{xx}(r = 1, \theta = \pi/2) = 30\ psi$.

Para diferentes refinamentos por inserção de *knots*, efetuou-se uma comparação entre o número de graus de liberdade e o componente σ_{xx} na fronteira mencionada. Além dos modelos da Figura 89 e Figura 90, também foi considerado um modelo de elementos finitos com elementos Q8, gerado por mapeamento transfinito bilinear, como também o modelo de NURBS do software IGAFEM [41]. Em todos os casos, a análise foi efetuada com integração *full*. A Figura 92 ilustra um gráfico, para todos os cenários mencionados, com a tensão σ_{xx} máxima em função da quantidade de graus de liberdade empregados. O valor analítico de 30 psi foi convergido em todos os casos.

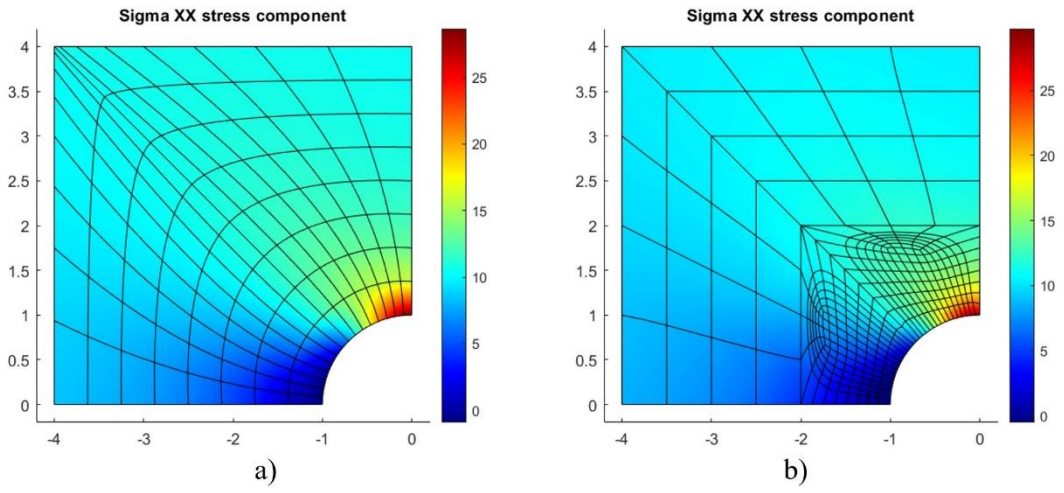

Figura 91: Distribuição da componente de tensão σ_{xx} para o problema da placa infinita com furo. a) NURBS. b) T-Splines com *extraordinary points*.

Notavelmente, observou-se que a formulação isoparamétrica com elementos Q8 obteve uma convergência mais lenta. As demais, com análise isogeométrica, mostraram-se mais eficientes e rápidas na convergência.

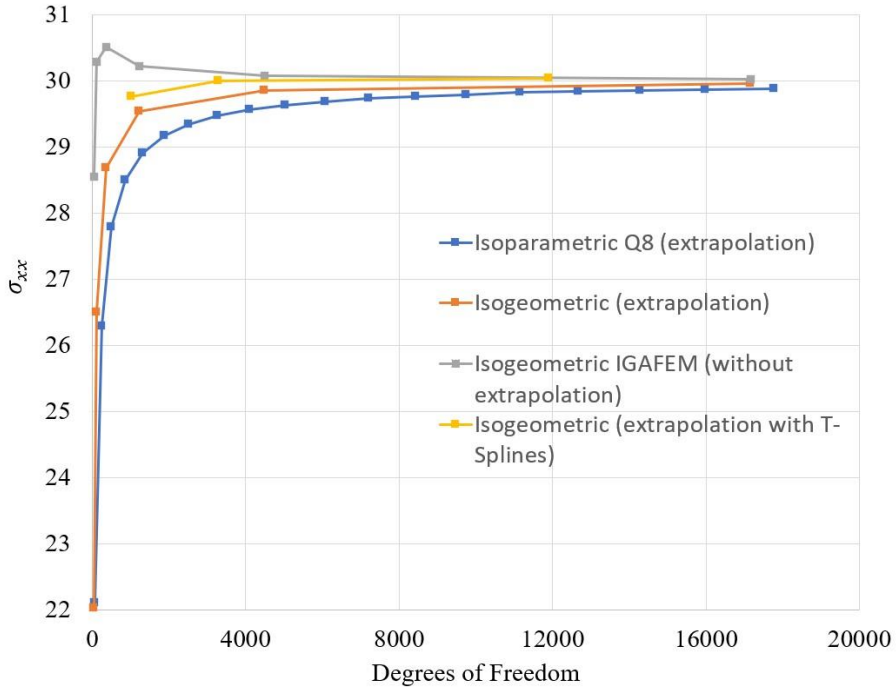

Figura 92: Tensão σ_{xx} máxima e quantidade de graus de liberdade para o problema da placa infinita com furo central.

Em relação ao modelo do software IGAFEM, uma distinção notável pertence a metodologia do pós-processamento. O FEMOOLab extrapola as tensões dos pontos de Gauss para os chamados pontos de extrapolação do elemento (Seção 7.4). Por outro lado, o IGAFEM calcula as tensões diretamente nas bordas dos elementos, sem realizar um processo de extrapolação. Nesse problema, o alvo da análise está na tensão σ_{xx} máxima, que se situa em um ponto de extremidade do elemento contido na região. Assim, nesse caso, a metodologia de determinação de tensões do IGAFEM se torna particularmente interessante. Por esse motivo, a convergência do IGAFEM se mostrou superior ao modelo de NURBS do FEMEP.

Por outro lado, o modelo com *patches* de T-Splines do FEMEP tem sua vantagem na flexibilidade de modelagem. Essa abordagem proporciona uma maior capacidade de refinamento em regiões específicas, graças a possibilidade de geração de malhas não estruturadas. No problema em questão, a concentração de tensões ocorre próxima a singularidade causada pelo furo. Nesse sentido, é

vantajoso desejar mais refinamento nessas localidades críticas e economizar recursos computacionais nas áreas onde a precisão exigida é menor. Dessa forma, o modelo que se destacou em termos de eficiência e precisão foi o que utilizou *patches* de T-Splines.

7.5.2 Cilindro de Parede Grossa

Este exemplo trata de um cilindro de parede grossa submetido a uma pressão interna p constante, de acordo com a Figura 93. De maneira análoga ao problema anterior, um quarto do problema é modelado devido à simetria. O raio interno r_1 assume valor de $1\ m$ e o raio externo r_2 assume valor de $3\ m$. Adota-se espessura $t = 1\ m$. As propriedades do material são dadas pelo módulo de elasticidade $E = 4\,.\,10^7\ \ kN/m^2$ e pelo coeficiente de Poisson $\nu = 0.25$. A carga p recebe valor de $10\ kN/m$.

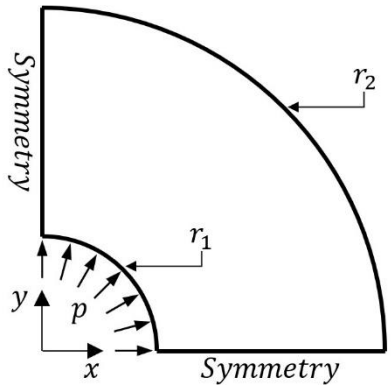

Figura 93: Cilindro de parede grossa.

A solução analítica (Solução de Lamé) [53] é dada pelas equações:

$$\sigma_{rr}(r) = -p\frac{(r_2^2 - r^2)r_1^2}{(r_2^2 - r_1^2)r^2} \tag{145}$$

$$\sigma_{\theta\theta}(r) = p\frac{(r_2^2 + r^2)r_1^2}{(r_2^2 - r_1^2)r^2} \tag{146}$$

sendo r o raio de interesse, determinado a partir da origem.

Ao longo da face interna, a condição de contorno natural consiste em $\bar{\boldsymbol{t}} = [p\ cos\theta, p\ sen\theta]^T$, onde θ é o ângulo com o eixo x. Para as condições de contorno essenciais, tem-se $u_x = 0$ na face esquerda e $u_y = 0$ na face inferior.

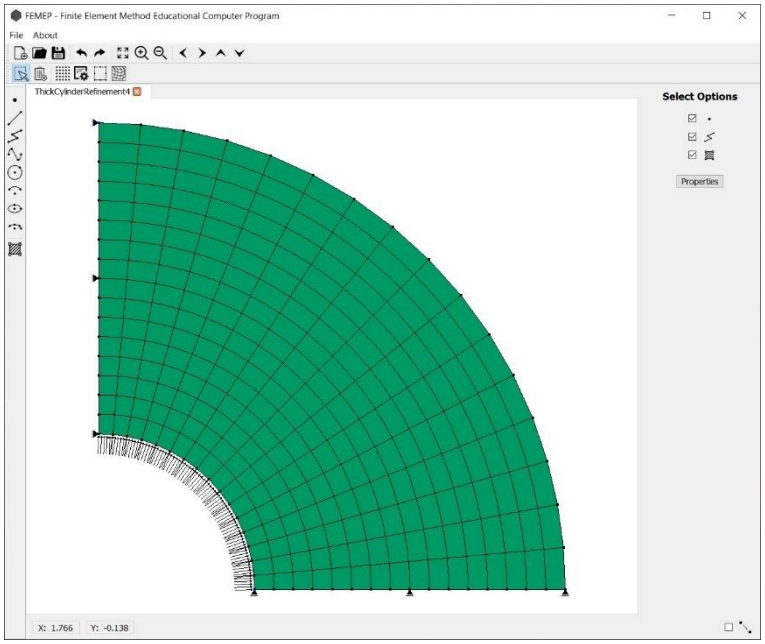

Figura 94: Cilindro de parede grossa modelado no FEMEP com um *patch* de NURBS.

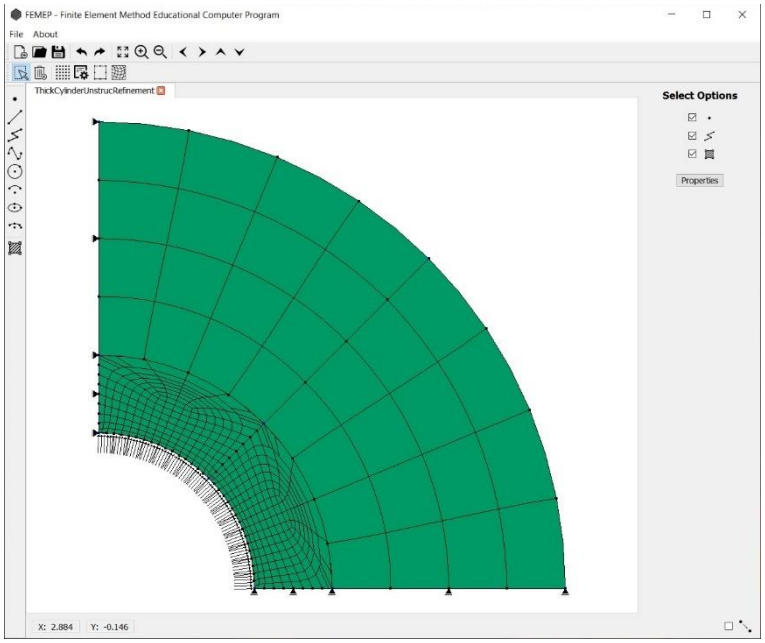

Figura 95: Cilindro de parede grossa modelado no FEMEP com *patches* de NURBS e T-Splines.

Na Figura 94 , o problema foi modelado no FEMEP utilizando um *patch* de superfície NURBS biquadrática. Curvas de contorno do tipo arco de círculo são definidas na direção ξ, e curvas do tipo reta definidas na direção η. Novamente, a elevação de grau é empregada para alcançar a ordem polinomial desejada, sempre antes de efetuar os refinamentos por inserção de *knots*, de forma a evitar perda de continuidade (ver Seção 2.2.8.3).

A Figura 95 demonstra o modelo no FEMEP modelado através de quatro *patches*, sendo duas regiões NURBS de grau 3, em ambas direções, e duas regiões T-Splines bicúbicas com *extraordinary points*, situadas nas localidades onde foram identificadas tensões mais críticas. A divisão interna dos *patches* se dá através de uma curva de arco de círculo e uma curva do tipo reta.

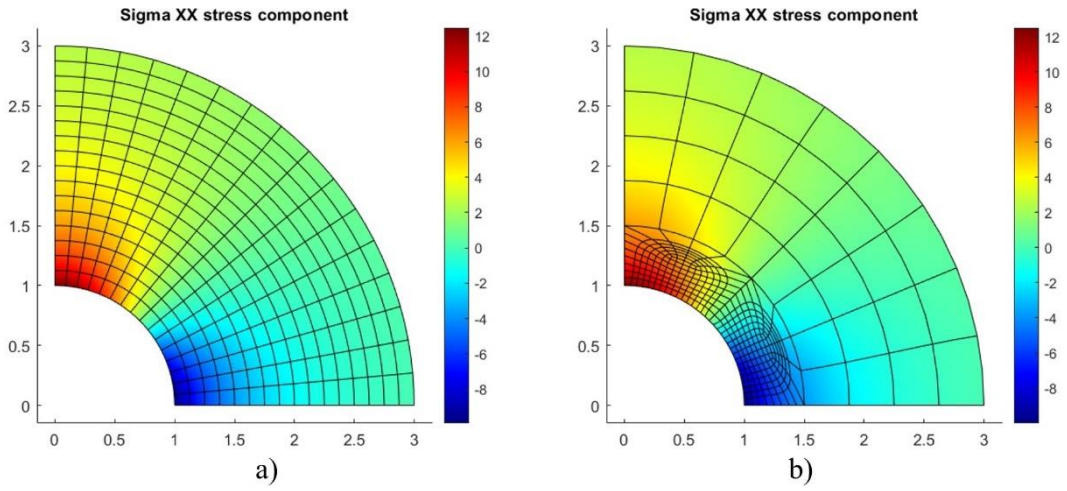

Figura 96: Distribuição da componente de tensão σ_{xx} para o problema do cilindro de parede grossa. a) NURBS. b) T-Splines com *extraordinary points*.

Com o auxílio do FEMOOLab, foi possível obter as distribuições da tensão σ_{xx} dos modelos, apresentadas na Figura 96. O valor máximo para essa componente ocorre no ponto $(r = 1,\ \theta = \pi/2)$. Nessa localidade, a solução analítica fornece a componente de tensão radial $\sigma_{rr}(r = 1) = -10\ kN/m^2$ e a componente tangencial $\sigma_{\theta\theta}(r = 1) = 12.5\ kN/m^2$. A tensão na direção x assume $\sigma_{xx} = \sigma_{rr}\ cos^2\theta + \sigma_{\theta\theta}\ sen^2\theta = 12.5\ kN/m^2$.

A relação entre a tensão σ_{xx} máxima e a quantidade de graus de liberdade, para modelos mais refinados por inserção de *knots*, é dada pelo gráfico da Figura 97. Um modelo de elementos finitos Q8 também é considerado para o problema.

Em todos os cenários, houve concordância com a solução analítica de Lamé [53]. Observa-se que nos modelos de elementos finitos a convergência foi mais lenta, enquanto que os modelos de análise isogeométrica apresentaram uma convergência mais eficiente.

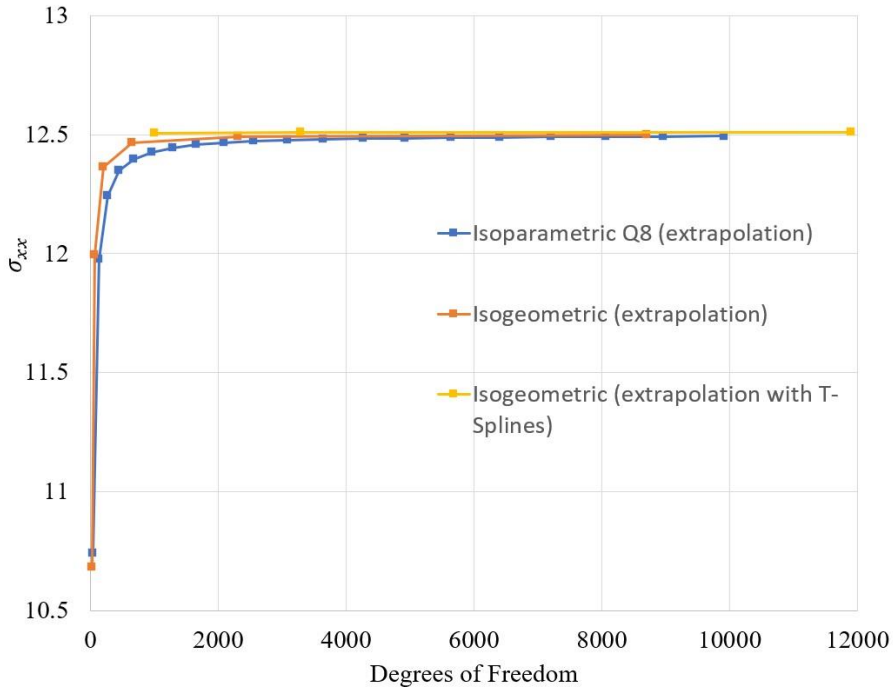

Figura 97: Tensão σ_{xx} máxima e quantidade de graus de liberdade para o problema do cilindro de paredes grossas.

Em elementos finitos, a conectividade é dada pelos nós que definem um determinado elemento. Nesse cenário, apenas os nós de fronteira apresentam conectividade em comum com elementos adjacentes. Na análise isogeométrica isso não ocorre. Nesse caso, a propriedade de suporte das funções de base determina os pontos de controle de conectividade. Devido a continuidade das funções NURBS, elementos adjacentes podem compartilhar uma maior quantidade de pontos de controle de conectividade em comum, não se restringindo apenas aos pontos de controle de fronteira. A consequência é que uma menor quantidade de graus de liberdade é requerida na AIG, conforme constatado nos resultados.

Em relação às T-Splines com *extraordinary points*, um resultado semelhante ao exemplo anterior foi observado. Devido a capacidade de maior refinamento na localidade onde a tensão σ_{xx} é máxima, esse caso exige uma menor quantidade de

graus de liberdade para obtenção de um resultado mais próximo da solução analítica.

7.5.3 Gancho Circular

A Figura 98 mostra um modelo de gancho bidimensional submetido a uma carga uniforme q, baseado no exemplo de Li [54]. O exemplo é formado por arcos de círculos de raios $r_1 = 0.1\,m$ e $r_2 = 0.2\,m$. Foi adotado um módulo de elasticidade de $E = 2\,.\,10^8\ kN/m^2$, um coeficiente de Poisson de $\nu = 0.3$ e espessura $t = 0.1\,m$. Admite-se um valor para a carga de $q = 1\ kN/m$.

O problema apresenta as condições de contorno de $\bar{\boldsymbol{t}} = [0, -q]^T$ na extremidade livre e $u = [u_x, u_y] = [0, 0]$ na extremidade suspensa.

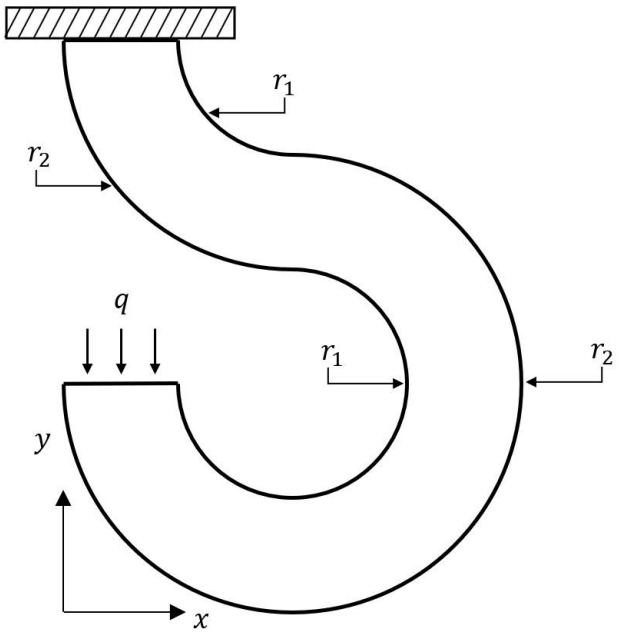

Figura 98: Gancho circular.

A modelagem do problema em NURBS no FEMEP é ilustrada na Figura 99. São considerados quatros *patches* de NURBS quadráticas nas duas direções, cada *patch* fica contido em arcos de 90°. Curvas do tipo arco de círculo formam o contorno na direção ξ e curvas do tipo reta compõe a direção η. A Figura 100 mostra um modelo com T-Splines com *extraordinary points* nos dois *patches* centrais.

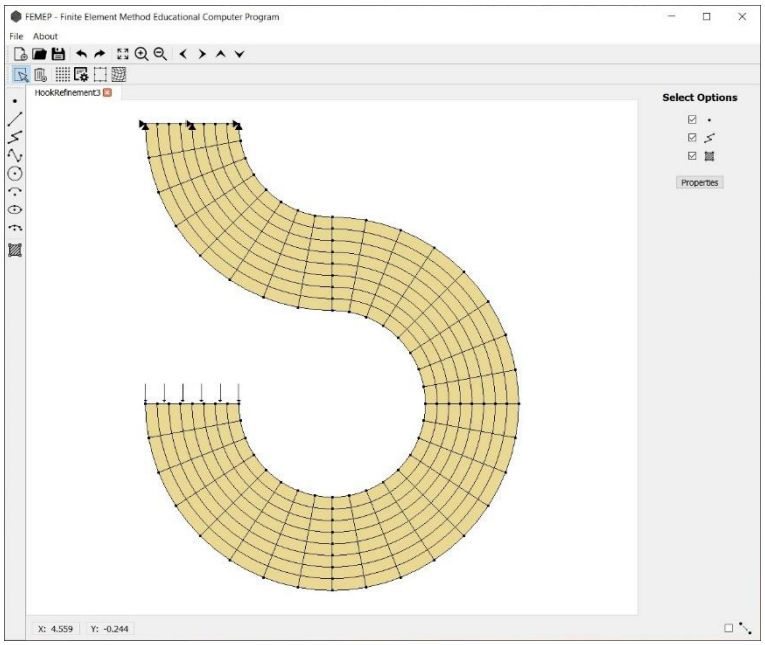

Figura 99: Gancho circular modelado no FEMEP com *patches* de NURBS.

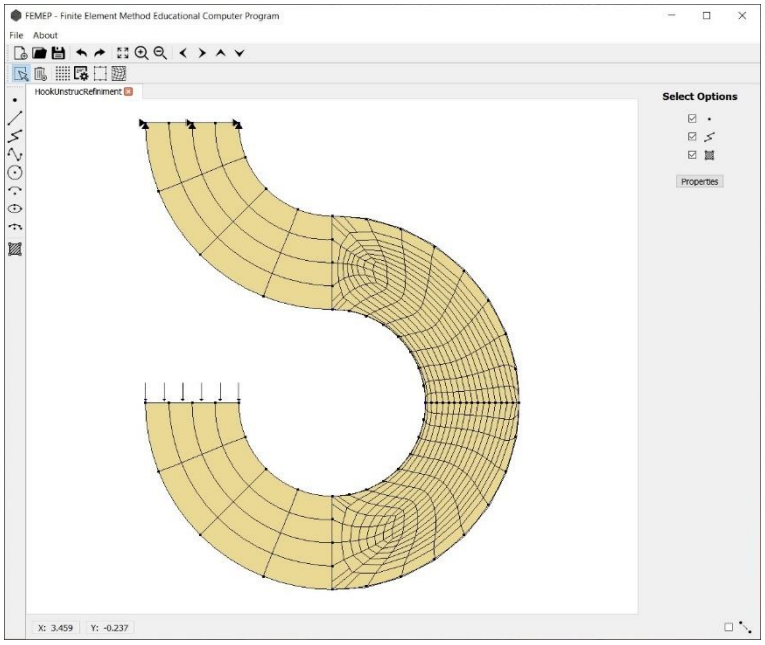

Figura 100: Gancho circular modelado no FEMEP com patches de NURBS e T-Splines.

A distribuição das componentes de tensão é mostrada na Figura 101. O gráfico da Figura 102 contém os resultados da componente de tensão σ_{yy} máxima verificadas nos modelos refinados por inserção de *knots*. Modelos com elementos Q8 também foram levados em conta. Os resultados seguem o padrão observado nos exemplos anteriores, com uma melhor convergência ocorrendo nos modelos de T-

Splines com *extraordinary points*. Modelos com *patches* de NURBS apresentam um desempenho superior aos modelos de elementos finitos.

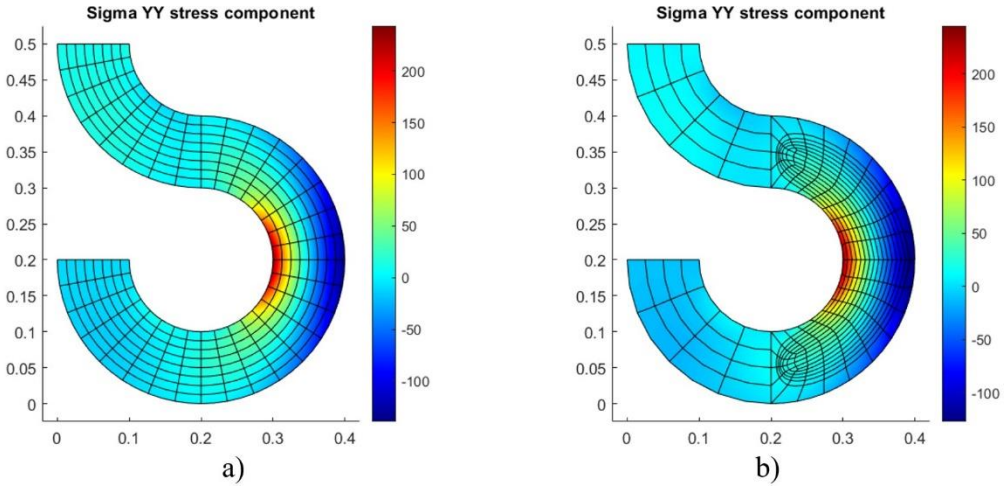

Figura 101: Distribuição da componente de tensão σ_{yy} para o problema do gancho circular. a) NURBS. b) T-Splines com *extraordinary points*.

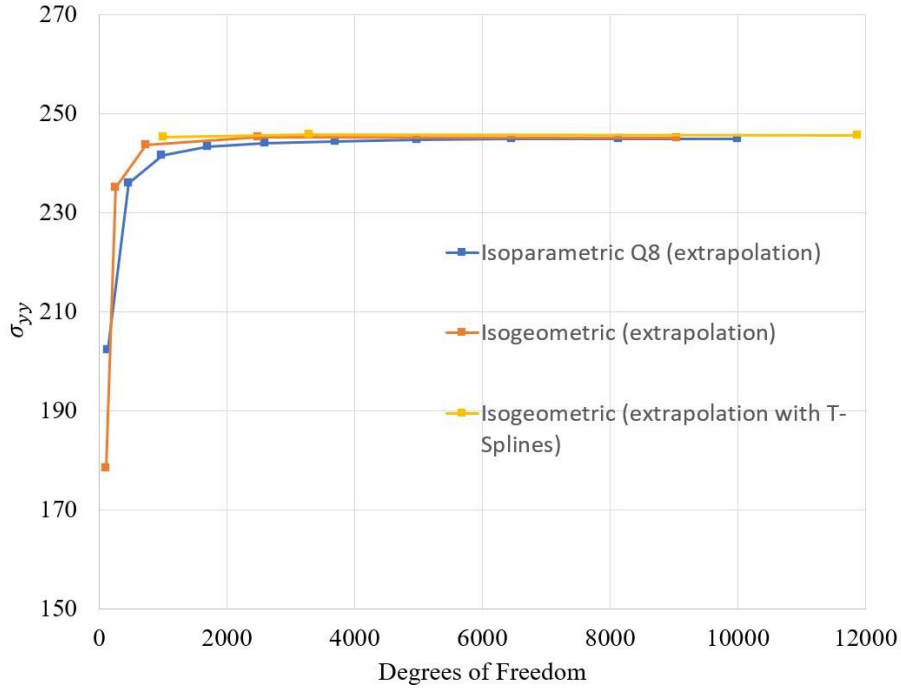

Figura 102: Tensão σ_{yy} máxima e quantidade de graus de liberdade para o problema do gancho circular.

7.6 Outros Modelos de Múltiplos *Patches*

Em sequência, apenas com o objetivo de demonstrar a competência dos programas em modelagem e análise, apresentam-se dois modelos de maior complexidade, que requerem um número amplo de *patches* para representação. Os resultados da configuração deformada, deslocamentos e tensões são então exibidos.

A Figura 103, inspirada na peça metálica de Bathe [5], ilustra a segmentação em *patches* NURBS. A peça apresenta uma carga concentrada aplicada no orifício da direita e é fixada nos orifícios da extremidade esquerda. A presença de vários orifícios no modelo exigiu a criação de múltiplas regiões, destacando o potencial da ferramenta em lidar com características geométricas variadas. Um total de 38 *patches* foram empregados para a criação desse modelo.

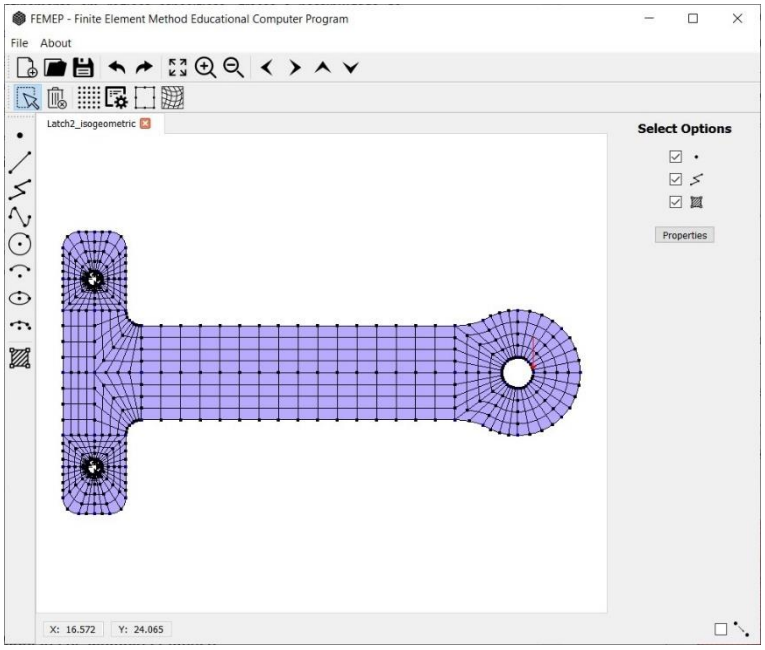

Figura 103: Peça metálica modelada com múltiplos *patches NURBS.* Inspirado em Bathe [5].

De forma semelhante, a Figura 104 apresenta uma estrutura modelada em vários *patches*. A peça possui condição de engaste em sua face inferior, e uma carga vertical uniformemente distribuída aplicada em seu braço. Os dois *patches* centrais são modelados com T-Splines para a formação de malhas não estruturadas, o que mostra a versatilidade do programa. No total, o modelo foi dividido em 17 *patches*.

A análise pelo FEMOOLab dos modelos da Figura 103 e Figura 104 fornece as configurações deformadas, mapas de deslocamentos e mapas tensões, apresentados na Figura 105 e na Figura 106, respectivamente.

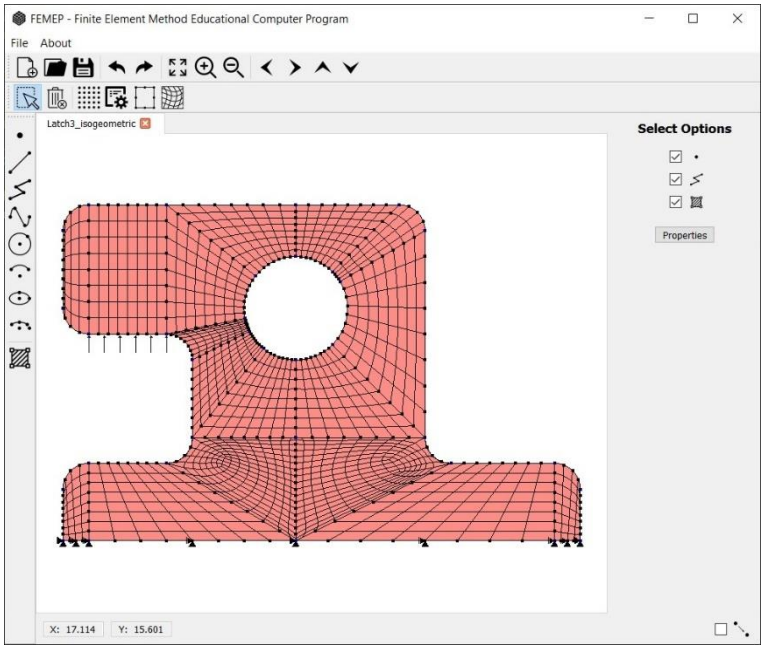

Figura 104: Estrutura modelada por *patches NURBS* e *T-Splines*.

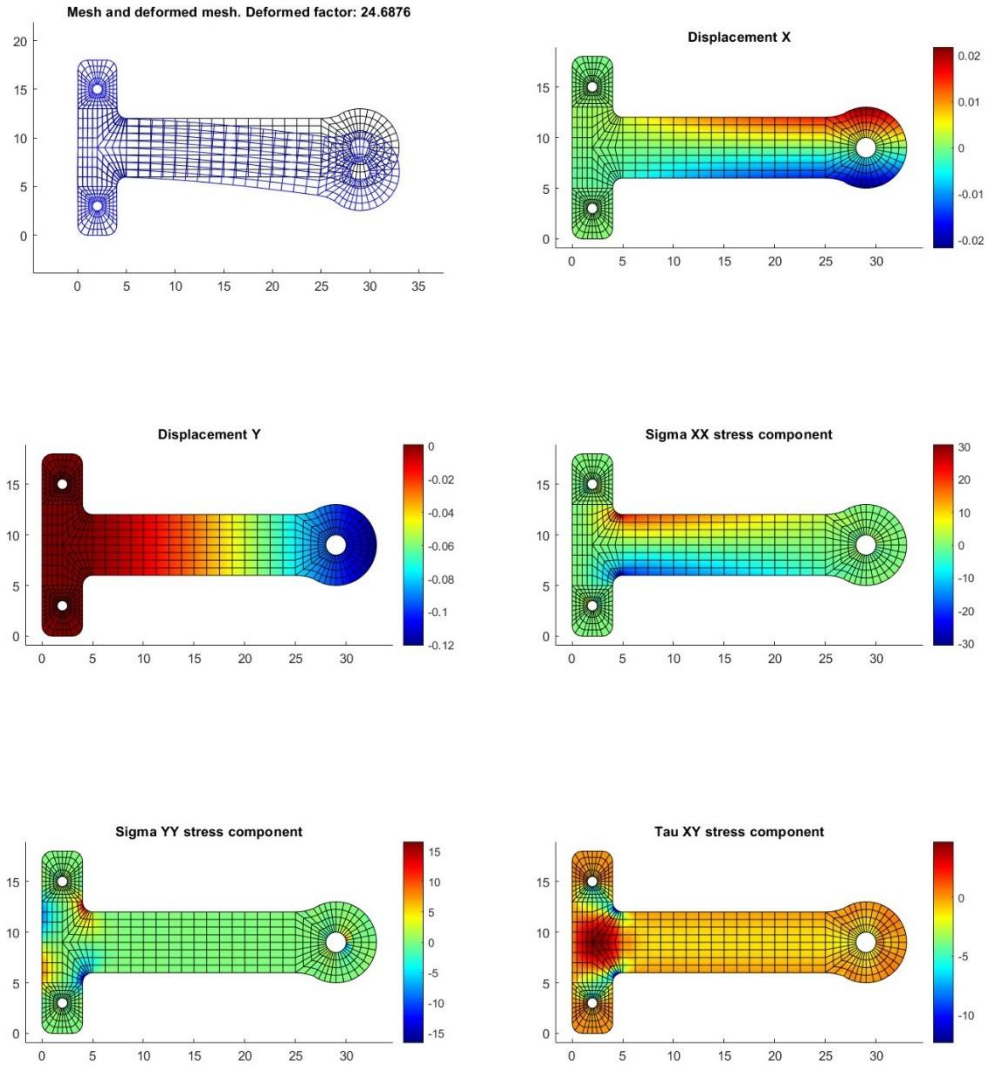

Figura 105: Deformada, deformações e tensões de uma peça metálica modelada com *patches* NURBS, inspirada em Bathe [5].

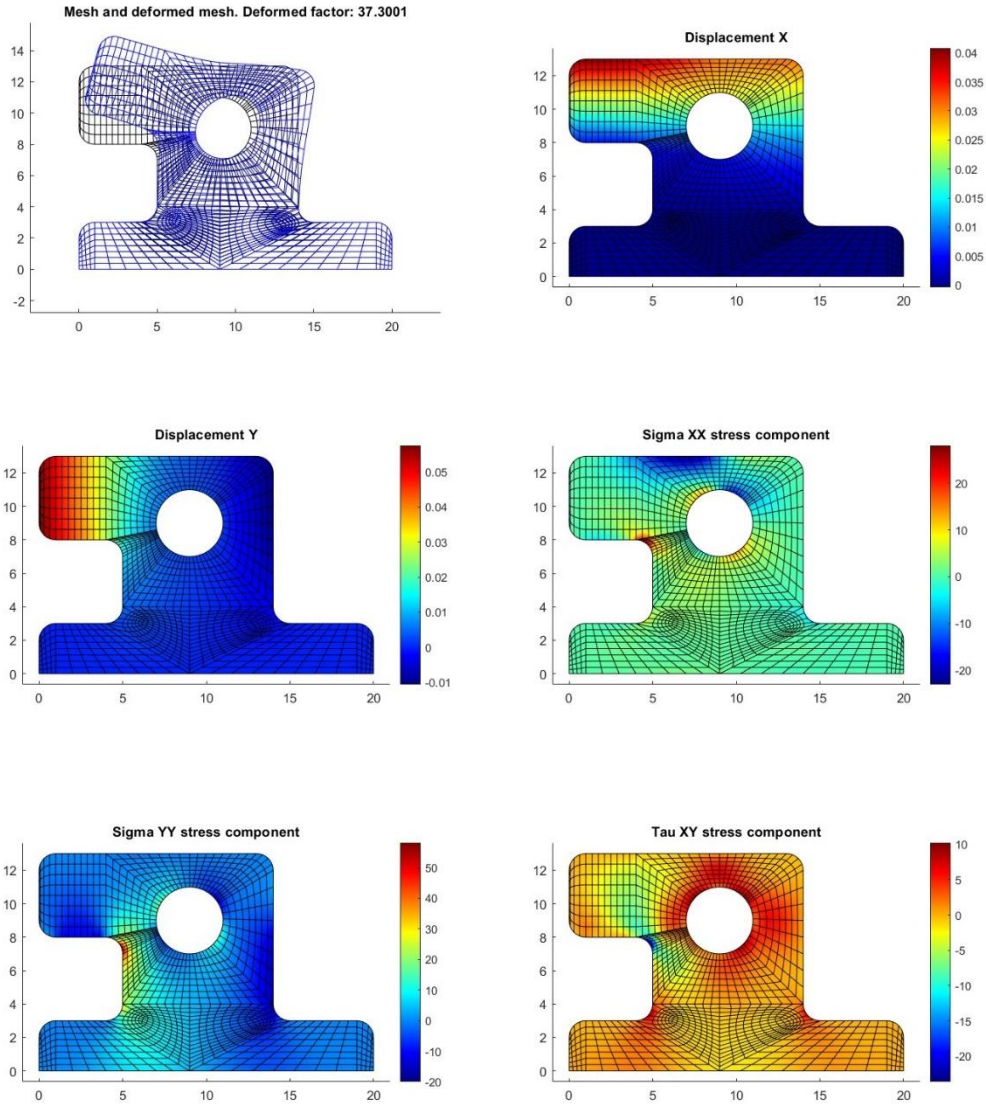

Figura 106: Configuração deformada, deslocamentos e tensão de uma estrutura modelada com múltiplos *patches* NURBS e T-Splines.

8 Conclusões e Considerações Finais

O trabalho desenvolveu duas ferramentas para modelagem e análise isogeométrica bidimensional de problemas de elasticidade linear estática. A interface gráfica de usuário do modelador, juntamente com a visualização dos resultados do software de análise, tornam as ferramentas ideais para estudantes ou pesquisadores que desejem ingressar nos conceitos da análise isogeométrica.

O modelador se destaca por sua capacidade de realizar a modelagem interativa de curvas, permitindo a visualização de propriedade NURBS, a execução de refinamentos e interseções automáticas entre curvas. A possibilidade de gerar *patches* de superfícies NURBS a partir de quatro curvas de contorno, utilizando o conceito de superfície Coons, é um recurso notável que viabiliza a análise bidimensional.

Além disso, um diferencial do modelador consiste na geração de *patches* de T-Splines de malhas não estruturadas, baseadas na presença de *extraordinary points*. A execução dessa funcionalidade foi possível graças a combinação de esforços de trabalhos anteriores de diferentes autores. O algoritmo de Miranda e Martha [19] desempenhou papel fundamental na geração de uma rede de pontos de controle. Scott [17] fornece uma formulação para análise de T-Splines com *extraordiary points* baseadas na técnica de extração Bézier. A contribuição do presente trabalho foi a automatização da geração de uma T-Spline.

Os programas foram verificados em diferentes problemas. Os resultados de componentes de tensões, obtidos da análise isogeométrica, não só convergiram, como se mostraram superior a um modelo de elemento finitos com elementos Q8. Além disso, verificou-se que um modelo com malhas não estruturadas de T-Splines exigiu uma menor quantidade de GL.

A presença de *patches* conformes permite a modelagem e análise de problemas com múltiplos *patches*. Um modelo misto com regiões NURBS e T-Splines é perfeitamente possível.

Destaca-se ainda que a arquitetura de POO resultou em um código organizado, que facilita a expansão futura das ferramentas.

8.1 Sugestões de Trabalhos Futuros

A natureza de código aberto das ferramentas apresentadas neste trabalho, permite que pesquisadores e desenvolvedores na área de mecânica computacional possam contribuir para o desenvolvimento e aprimoramento dos *softwares*.

Para futuras pesquisas e desenvolvimentos, é interessante considerar a implementação em uma única ferramenta integrada que englobe todas as etapas do processo de análise isogeométrica. Essa ferramenta unificada poderia incluir modelagem, análise, pós-processamento e visualização de resultados. Assim, proporcionaria ao usuário uma experiência mais coesa e eficiente, eliminando a necessidade de ter que alternar entre diferentes ambientes.

Incorporar novas classes de curvas ao modelador pode potencializar as opções de modelagem. Seria válido incluir uma NURBS genérica, permitindo ao usuário especificar diretamente coordenadas, pesos, *knot vector* e grau.

A geração de *patches* de superfícies NURBS exige a compatibilidade entre curvas opostas. O usuário deve explicitamente realizar as elevações de grau e as inserções de *knots* necessárias. Essa foi a linha adotada pensando em uma ferramenta didática. A execução automática desses refinamentos é uma sugestão para usuários que buscam resultados práticos, sem interesse no aspecto educacional.

Para T-Splines, o algoritmo de Miranda e Martha [19] possibilita a variação de parâmetros geométricos que influenciam a forma da malha. Isso abre a possibilidade de implementar uma janela iterativa para melhor ajuste das malhas.

A aplicação das ferramentas na análise de placas é uma outra sugestão. Teorias de placas requerem uma discretização geométrica que tenha continuidade C^1 entre os elementos. Normalmente, essa continuidade é definida por polinômios de ordem elevada, que são um problema para as funções de forma do MEF.

Por fim, a expansão para o caso de sólidos tridimensionais seria um avanço significativo. A evolução nessa direção permitiria modelagem e análise de estruturas mais complexas em diversas áreas, não apenas em engenharia civil. A formação de sólidos com NURBS é perfeitamente factível, através de uma estrutura de produto tensorial análoga a superfícies. T-Splines trivariadas com *extraordinary points* poderiam ser um desafio, com a tese de Scott [17] sendo um ponto de partida.

9 Referências Bibliográficas

1. FISH, Jacob; BELYTSCHKO, Ted. **A first course in finite elements**. New York: Wiley, 2007.
2. FELIPPA, Carlos A.; Introduction to finite element methods. **University of Colorado**, v. 885, 2004.
3. COOK, Robert D. et al. **Concepts and applications of finite element analysis**. John wiley & sons, 2007.
4. COTTRELL, J. Austin; HUGHES, Thomas J.R.; BAZILEVS, Yuri. **Isogeometric analysis: toward integration of CAD and FEA**. John Wiley & Sons, 2009.
5. BATHE, Klaus-Jürgen. **Finite element procedures**. Klaus-Jurgen Bathe, 2006.
6. HUGHES, Thomas J.R.; COTTRELL, John A.; BAZILEVS, Yuri. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. **Computer methods in applied mechanics and engineering**, v. 194, n. 39-41, p. 4135-4195, 2005.
7. LIU, Fan et al. A novel shape-adjustable surface and its applications in car design. **Applied Sciences**, v. 9, n. 11, p. 2339, 2019.
8. GROßMANN, David et al. Isogeometric simulation of turbine blades for aircraft engines. **Computer Aided Geometric Design**, v. 29, n. 7, p. 519-531, 2012.
9. GUERDER, Marie. **IsoGeometric analysis and shape optimisation of aircraft compressor blades**. Doctor of Philosophy (PhD) - Université de Lyon, 2022.
10. KURAISHI, Takashi et al. Space-time isogeometric analysis of car and tire aerodynamics with road contact and tire deformation and rotation. **Computational Mechanics**, v. 70, n. 1, p. 49-72, 2022.
11. BAZILEVS, Yuri et al. Isogeometric analysis of blood flow: a NURBS-based approach. In: **Computational Modelling of Objects Represented in Images. Fundamentals, Methods and Applications**. CRC Press, 2018. p. 91-96.

12. ZHANG, Yongjie et al. Patient-specific vascular NURBS modeling for isogeometric analysis of blood flow. **Computer methods in applied mechanics and engineering**, v. 196, n. 29-30, p. 2943-2959, 2007.
13. BAZILEVS, Yuri et al. Isogeometric analysis using T-splines. **Computer methods in applied mechanics and engineering**, v. 199, n. 5-8, p. 229-263, 2010.
14. SEDERBERG, Thomas W. et al. T-splines and T-NURCCs. **ACM transactions on graphics (TOG)**, v. 22, n. 3, p. 477-484, 2003.
15. SCOTT, Michael A. et al. Isogeometric finite element data structures based on Bézier extraction of T-splines. **International Journal for Numerical Methods in Engineering**, v. 88, n. 2, p. 126-156, 2011.
16. BORDEN, Michael J. et al. Isogeometric finite element data structures based on Bézier extraction of NURBS. **International Journal for Numerical Methods in Engineering**, v. 87, n. 1-5, p. 15-47, 2011.
17. SCOTT, Michael Andrew. **T-splines as a design-through-analysis technology**. Doctor of Philosophy (PhD) - University of Texas at Austin, 2011.
18. BOMFIM, D.S.; **Uma estratégia de modelagem aberta e extensível para criação de modelos de subdivisões planares para mecânica computacional**. Dissertação de Mestrado (Mestrado em Engenharia Civil) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2022.
19. DE OLIVEIRA MIRANDA, Antonio Carlos; MARTHA, Luiz Fernando. Hierarchical template-based quadrilateral mesh generation. **Engineering with Computers**, v. 33, p. 701-715, 2017.
20. CAMPOS, Jorge A.P.; **Geração de malhas de elementos finitos bidimensionais baseada em uma estrutura de dados topológica**. Dissertação de Mestrado (Mestrado em Engenharia Civil) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 1991.
21. CAMPOS, J. A. P.; MARTHA, L.F.; GATTASS, M.; Estrutura de Dados Topológica para Geração de Malhas Bidimensionais de Elementos Finitos, **Anais do XI Congresso Brasileiro de Engenharia Mecânica, ABCM**, São Paulo, vol. XIII, p. 137-140, 1991.
22. CAVALCANTI, P.R., CARVALHO, P.C.P.; MARTHA, L.F.; Criação e Manutenção de Subdivisões Planares, **Anais do IV Simpósio Brasileiro de**

Computação Gráfica e Processamento de Imagens, USP/SBC, São Paulo, 1991, p. 13-24, 1991.

23. CAVALCANTI, Paulo Roma. **Criação e Manutenção de subdivisões no espaço**. Tese (Doutorado em Informática: Ciência da Computação) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 1992.
24. SILVEIRA, E.S.S.S.; **Um sistema de modelagem bidimensional configurável para simulação adaptativa em mecânica computacional**. Dissertação de Mestrado (Mestrado em Engenharia Civil) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 1995.
25. CARVALHO, M.T.M.; **Uma estratégia para desenvolvimento de aplicações configuráveis em mecânica computacional**. Tese (Doutorado em Engenharia Civil) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 1995.
26. LIRA, W.W.M.; **Um sistema integrado configurável para simulações em mecânica computacional**. Dissertação de Mestrado (Mestrado em Engenharia Civil) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 1998.
27. BOMFIM, D. S. et al. Development of a Python Application Aiming at the Teaching-learning Process of the Half-Edge Data Structure. **Proceedings of the joint XLII Ibero-Latin-American Congress on Computational Methods in Engineering and III PanAmerican Congress on Computational Mechanics**, Rio de Janeiro, 2021.
28. BOMFIM, Danilo S. et al. HETOOL: A Half-Edge Topological Object-Oriented Library for generic 2-D geometric modeling. **SoftwareX**, v. 21, p. 101307, 2023.
29. LIN, Fenqiang; HEWITT, William T. Expressing Coons-Gordon surfaces as NURBS. **Computer-Aided Design**, v. 26, n. 2, p. 145-155, 1994.
30. PIEGL, Les; TILLER, Wayne. **The NURBS book**. Springer Science & Business Media, 1996.
31. BARROSO, Elias Saraiva et al. An efficient automatic mesh generation algorithm for planar isogeometric analysis using high-order rational Bézier triangles. **Engineering with Computers**, v. 38, n. 5, p. 4387-4408, 2022.

32. BARROSO, E.S.; **Geração de malhas de alta ordem para análise isogeométrica utilizando elementos de Bézier racionais**. Tese (Doutorado em Ciência da Computação) - Universidade Federal do Ceará, Fortaleza, 2022.
33. PIEGL, Les. On NURBS: a survey. **IEEE Computer Graphics and Applications**, v. 11, n. 01, p. 55-71, 1991.
34. GOLDMAN, Ron. **Pyramid algorithms: A dynamic programming approach to curves and surfaces for geometric modeling**. Elsevier, 2002.
35. ROGERS, David F.; **An introduction to NURBS: with historical perspective**. Morgan Kaufmann, 2001.
36. COX, Maurice G.; **The numerical evaluation of B-splines. IMA Journal of Applied mathematics**, v. 10, n. 2, p. 134-149, 1972.
37. DE BOOR, Carl. On calculating with B-splines. Journal of Approximation theory, v. 6, n. 1, p. 50-62, 1972.
38. COHEN, Elaine; LYCHE, Tom; SCHUMAKER, Larry L.; Algorithms for degree-raising of splines. **ACM Transactions on Graphics (TOG)**, v. 4, n. 3, p. 171-181, 1985.
39. PRAUTZSCH, Hartmut; PIPER, Bruce. A fast algorithm to raise the degree of spline curves. **Computer Aided Geometric Design**, v. 8, n. 4, p. 253-265, 1991.
40. PIEGL, Les; TILLER, Wayne. Software-engineering approach to degree elevation of B-spline curves. **Computer-Aided Design**, v. 26, n. 1, p. 17-28, 1994.
41. NGUYEN, Vinh Phu; BORDAS, Stéphane P. A.; RABCZUK, Timon. Isogeometric analysis: an overview and computer implementation aspects. **Mathematics and Computers in Simulation**, v. 117, p. 89-116, 2015.
42. HABER, Robert et al. A general two-dimensional, graphical finite element preprocessor utilizing discrete transfinite mappings. **International Journal for Numerical Methods in Engineering**, v. 17, n. 7, p. 1015-1044, 1981.
43. SEDERBERG, Thomas W. et al. T-spline simplification and local refinement. **ACM transactions on graphics (TOG)**, v. 23, n. 3, p. 276-283, 2004.
44. LI, Xin et al. On linear independence of T-spline blending functions. **Computer Aided Geometric Design**, v. 29, n. 1, p. 63-76, 2012.

45. MÄNTYLÄ, M.; **An Introduction to Solid Modeling Computer**. Science Press, Rockville, Maryland, 1988.
46. CASQUERO, Hugo et al. A hybrid variational-collocation immersed method for fluid-structure interaction using unstructured T-splines. **International Journal for Numerical Methods in Engineering**, v. 105, n. 11, p. 855-880, 2016.
47. PEIXOTO, J. C. L.; RANGEL, R. L.; MARTHA, L. F.; Integrated system for two-dimensional isoparametric and isogeometric analysis of finite elements. **11th International Conference on IsoGeometric Analysis**, Book of Abstracts of IGA 2023, Lyon, 2023.
48. PEIXOTO, J. C. L.; RANGEL, R. L.; MARTHA, L. F.; Isogeometric analysis with interactive modeling of multi-patches NURBS. **Proceedings of the XLIV Ibero-Latin-American Congress on Computational Methods in Engineering**, O Porto, 2023.
49. MARTHA, Luiz Fernando. Notas de aula do curso CIV 2118–Método dos elementos finitos. **Departamento de Engenharia Civil, Pontifícia Universidade Católica do Rio de Janeiro, PUC, Rio de Janeiro, 52p**, 1994.
50. HINTON, E.; CAMPBELL, JS411120. Local and global smoothing of discontinuous finite element functions using a least squares method. **International Journal for Numerical Methods in Engineering**, v. 8, n. 3, p. 461-480, 1974.
51. BURNETT, David S.; Finite element analysis: from concepts to applications. **Addison-Wesley**, 1987.
52. GOULD, Phillip L.; FENG, Yuan. Introduction to linear elasticity. **New York: Springer-Verlag**, 1994.
53. NEGI, L. S.; Strength of Materials. **McGraw Hill**, New Delhi, 2008.
54. LI, K.; Isogeometric Analysis Project. Kangli, 2010. Disponível em: < https://kangli.me/en/projects/IGA/>.