DISSERTAÇÃO DE MESTRADO Nº 672

DESENVOLVIMENTO DE UM FRAMEWORK PARA MÉTODOS SEM MALHA

Naísses Zoia Lima

DATA DA DEFESA: 18/03/2011

Universidade Federal de Minas Gerais

Escola de Engenharia

Programa de Pós-Graduação em Engenharia Elétrica

DESENVOLVIMENTO DE UM FRAMEWORK PARA MÉTODOS SEM MALHA

Naísses Zoia Lima

Dissertação de Mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do Título de Mestre em Engenharia Elétrica.

Orientador: Prof. Renato Cardoso Mesquita

Belo Horizonte - MG

Março de 2011

"Desenvolvimento de Um Framework Para Métodos Sem Malha"

Naísses Zoia Lima

Dissertação de Mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do grau de Mestre em Engenharia Elétrica.

Aprovada em 18 de março de 2011.

Por:

Prof. Dr. Renato Cardoso Mesquita
DEE (UFMG) - Orientador

Prof. Dr. Elson José da Silva
DEE (UFMG)

Prof. Dr. Jaime Arturo Ramírez
DEE (UFMG)

Prof. Dr. Eduardo Henrique da Rocha Coppoli
Eng. Elétrica (CEFET/MG)

Resumo

Métodos sem malha vêm ganhando espaço na área de métodos numéricos como uma alternativa aos métodos tradicionais, como o Método dos Elementos Finitos (FEM) e o Método das Diferenças Finitas (FDM). A principal característica dos métodos sem malha é a ausência de uma malha para construção de suas funções de forma, funções usadas para gerar as aproximações da solução. Dessa maneira, o domínio é representado por um conjunto de nós espalhados ao invés de uma malha ou um *grid*, como ocorre no FEM e no FDM. Este trabalho apresenta um *framework* para a solucão de problemas eletromagnéticos utilizando métodos sem malha, chamado *MFree Framework*. O *framework* é desenvolvido na linguagem de programação C++ sob o paradigma de programação genérica usando o mecanismo de *templates*. A programação genérica tem como objetivo representar algoritmos e estruturas de dados da forma mais genérica possível, com as abstrações descritas por *conceitos*. No MFree Framework, as estruturas de dados são parametrizadas, permitindo que métodos, funções de forma, procedimentos de integração, formulações e outros componentes sejam combinados de várias maneiras para solucionar um determinado problema. Como os tipos de dados são implementados de modo a satisfazer um ou um conjunto de conceitos, novos tipos podem ser desenvolvidos e usados com as estruturas de dados previamente disponíveis de forma direta, o que torna o *framework* bastante extensível. Na versão atual, o *framework* instancia aplicações capazes de solucionar problemas utilizando o *Element-Free Galerkin Method* (EFG), o *Meshless Local Petrov-Galerkin Method* (MLPG) e o *Point Interpolation Method* (PIM) e suas variações. Todavia, outros métodos podem ser facilmente implementados devido ao caráter genérico com que é construído. Para testar essas instanciações do *framework*, alguns problemas eletrostáticos e magnetostáticos são resolvidos e o MFree Framework demonstra ser uma ferramenta de grande flexibilidade, utilidade e importância para futuros desenvolvimentos na área de métodos sem malha.

Abstract

Meshless methods are an alternative to traditional numerical methods such as the Finite Element Method (FEM) and the Finite Difference Method (FDM). The main feature of meshless methods is the absence of a mesh to construct their shape functions, which are used to generate approximations for the solution. Thus, the domain is represented by a set of scattered nodes instead of a mesh or a grid, as in FEM and FDM. This work presents a framework for solving electromagnetic problems using meshless methods, called *MFree Framework*. The framework is developed in C++ programming language under the generic programming paradigm using the *template* mechanism. Generic programming aims to represent algorithms and data structures in the most generic form as possible, with the abstractions described by *concepts*. In MFree Framework, the data structures are parameterized, allowing methods, shape functions, integration procedures, formulations and other components to be combined in various ways to solve a particular problem. As data types are implemented to meet one or a set of concepts, new types can be developed and used with the previously available data structures in a direct manner, which makes the framework very extensible. In the current version, the framework instantiates applications able to solve problems using the *Element-Free Galerkin Method* (EFG), the *Meshless Local Petrov-Galerkin Method* (MLPG) and *Point Interpolation Method* (PIM) and its variations. However, other methods can be easily implemented due to the generic characteristic with which it is built. To test the framework instantiations, electrostatic and magnetostatic problems are solved and the MFree Framework shows to be a tool with great flexibility, utility and importance for future developments in the field of meshless methods.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

α	Fator de penalidade para imposição das condições de contorno essenciais
α_I	Fator de escalonamento para domínio de influência
α_Q	Fator de escalonamento para domínio de quadratura
α_S	Fator de escalonamento para domínio de suporte
β	Fator de penalidade para imposição das condições de interface entre meios
$\mathbf{\Phi}$	Vetor contendo as funções de forma nodais
ϵ	Permissividade elétrica
Γ	Fronteira do domínio Ω
Γ_g	Fronteira do domínio Ω com condição de contorno de Dirichlet
Γ_h	Fronteira do domínio Ω com condição de contorno de Neumann
Γ_I	Interface entre meios
Γ_q^i	Fronteira do domínio de quadratura Ω_q do nó i
Γ_{qg}^i	Fronteira do domínio de quadratura Ω_q do nó i, com condição de contorno de Dirichlet
Γ_{qh}^i	Fronteira do domínio de quadratura Ω_q do nó i, com condição de contorno de Neumann
Γ_{qi}^i	Fronteira do domínio de quadratura Ω_q do nó i que não tem interseção com a fronteira global Γ
μ	Permeabilidade magnética

Ω	Domínio
Ω_q^i	Domínio de quadratura do nó i
$\overrightarrow{B}$	Indução magnética ou densidade de fluxo magnético
$\overrightarrow{D}$	Indução elétrica ou densidade de fluxo elétrico
$\overrightarrow{E}$	Campo elétrico
$\overrightarrow{f}$	Densidade volumar da força de Lorentz
$\overrightarrow{H}$	Campo magnético
$\overrightarrow{J_l}$	Densidade linear de corrente elétrica
$\overrightarrow{J}$	Densidade superficial de corrente elétrica
ϕ_i	Função de forma do nó i
$\phi_{,k}$	Derivada parcial de ϕ em relação à k-ésima dimensão
ρ	Densidade volumar de carga elétrica
ρ_s	Densidade superficial de carga elétrica
σ	Condutividade elétrica
$\mathbf{A}$	Matriz de momentos do MLS
$\mathbf{a}$	Vetor de coeficientes a_j para funções de forma
$\mathbf{A}^{-1}$	Inversa da matriz de momentos do MLS
$\mathbf{A}_{,k}^{-1}$	Derivada parcial de $\mathbf{A}^{-1}$ em relação à k-ésima dimensão
$\mathbf{B}$	Matriz utilizada no MLS
$\mathbf{b}$	Vetor de coeficientes b_j para RPIMp
$\mathbf{B}_{,k}$	Derivada parcial de $\mathbf{B}$ em relação à k-ésima dimensão
$\mathbf{F}$	Vetor $n \times 1$ do sistema linear discretizado
$\mathbf{K}$	Matrix $n \times n$ do sistema linear discretizado
$\mathbf{p}(\mathbf{x})$	Polinômio utilizado para funções de forma

$\mathbf{p}^T_{,k}(\mathbf{x})$	Derivada de $\mathbf{p}^T(\mathbf{x})$ em relação à k-ésima dimensão
$\mathbf{P}_m$	Matrix de momentos de polinômios para o RPIMp
$\mathbf{P}_Q$	Matriz de momentos do PIM
$\mathbf{P}_Q^{-1}$	Inversa da matriz de momentos do PIM
$\mathbf{R}(\mathbf{x})$	Vetor contendo as RBFs dos nós no domínio de suporte
$\mathbf{R}_Q$	Matriz de momentos do RPIM
$\mathbf{R}_Q^{-1}$	Inversa da matriz de momentos do RPIM
$\mathbf{R}_{,k}(\mathbf{x})$	Derivada de $\mathbf{R}(\mathbf{x})$ em relação à k-ésima dimensão
$\mathbf{U}$	Vetor $n \times n$ do sistema linear discretizado contendo os parâmetros nodais a serem determinados
$\mathbf{U}_S$	Vetor contendo os parâmetros nodais
$a_j(\mathbf{x})$	j-ésimo coeficiente para funções de forma
b_j	j-ésimo coeficiente de termos polinomials para o RPIMp
$d_m(\mathbf{x})$	Espaçamento nodal médio na vizinhança do ponto $\mathbf{x}$
d_m^i	Espaçamento nodal médio na vizinhança do nó i
$d_{km}(\mathbf{x})$	Espaçamento nodal médio na direção k na vizinhança do ponto $\mathbf{x}$
d_{km}^i	Espaçamento nodal médio na direção k na vizinhança do nó i
g	Valor da condição de contorno de Dirichlet
h	Valor da condição de contorno de Neumann
J	Funcional utilizado no MLS
$p_j(\mathbf{x})$	j-ésimo termo do polinômio $p(\mathbf{x})$
R	Função de base radial
r_I^i	Raio de influência do nó i
r_Q^i	Raio de quadratura do nó i

$r_S(\mathbf{x})$	Raio de suporte do ponto $\mathbf{x}$
$R_{,k}$	Derivada de R em relação à k-ésima dimensão
r^i_{kI}	Raio de influência na direção k do nó i
r^i_{kQ}	Raio de quadratura na direção k do nó i
$r_{kS}(\mathbf{x})$	Raio de suporte na direção k do ponto $\mathbf{x}$
Ri	RBF centrada no nó i
u^h	Aproximação da função u
u_i	Parâmetro nodal do nó i
w	Função de teste
W_I	Função peso para o nó I utilizada no MLS
w_i	Função de teste para o nó i
$W_{I,k}$	Deriavada parcial de W_I em relação à k-ésima dimensão

Capítulo 1

Introdução

Fenômenos físicos ligados a sistemas de engenharia são frequentemente descritos, em um certo domínio, por *equações diferenciais parciais* e por um conjunto de restrições adicionais, chamadas de *condições de contorno* ou *condições de fronteira*. Ao conjunto equação diferencial parcial (EDP) mais condições de contorno dá-se o nome *problema de valor de contorno* (PVC). A solução para um problema de valor de contorno é aquela que satisfaz, simultaneamente, a equação diferencial parcial em todo o domínio e as condições de contorno [9].

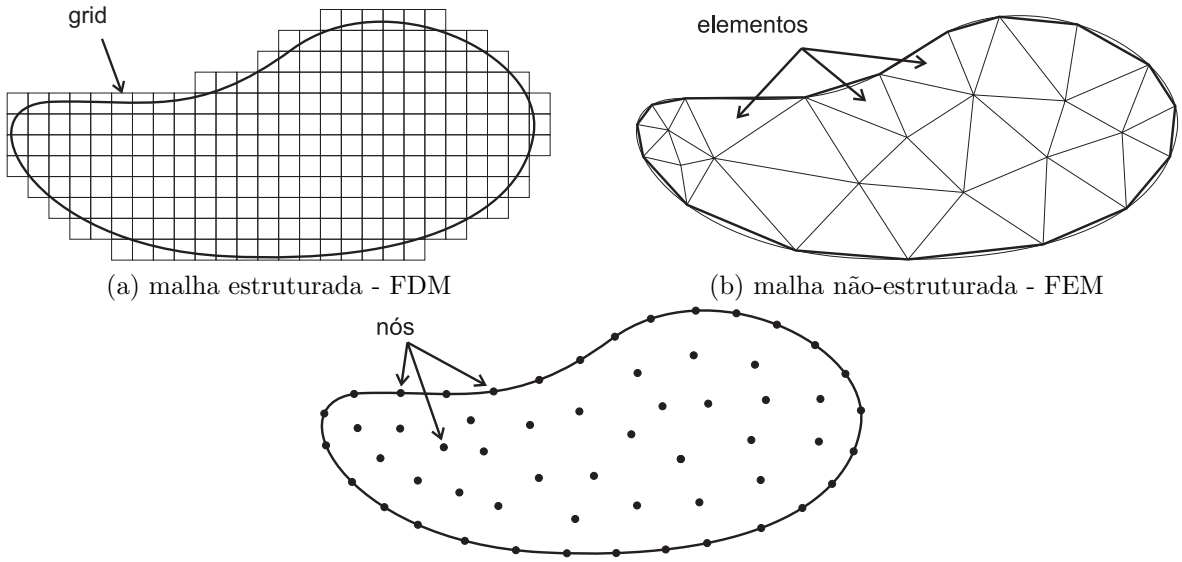
Alguns problemas de valor de contorno possuem solução analítica. A existência de tal solução depende do tipo de equação diferencial presente e da geometria do domínio do problema. Problemas cujos domínios possuem geometrias complexas não têm, geralmente, solução analítica.

No eletromagnetismo, citam-se como exemplos recorrentes de sistemas e fenômenos descritos por PVCs, as máquinas elétricas [11] onde investiga-se como estão distribuídos em seus componentes os campos elétrico e magnético, fenômenos físicos relacionados à propagação de ondas eletromagnéticas [29, 13] tais como refração, reflexão e ressonância, o aquecimento de corpos externos por meio de microondas [53], a análise de correntes parasitas [56, 58], dentre outros.

Sistemas reais normalmente são representados por domínios com geometrias complexas e a solução dos PVCs associados é obtida através de métodos computacionais. Na computação, problemas de valor de contorno são resolvidos utilizando-se métodos numéricos. Dentre os diversos existentes, o método dos elementos finitos (FEM) [26] e o método das diferenças finitas (FDM) [52] são considerados tradicionais, com muitos trabalhos já desenvolvidos e em estágio de maturidade avançado. Por isso, são os mais utilizados atualmente em problemas de valor de contorno. Já os métodos sem malha (MFree) [33] são métodos numéricos em rápido desenvolvimento, com estágio de maturidade bem inferior aos tradicionais, todavia já mostram resultados promissores nos trabalhos publicados na comunidade científica [33].

Os métodos de elementos finitos e de diferenças finitas possuem como característica comum a necessidade de que uma malha representando o domínio do problema seja gerada. No FDM, a malha é estruturada (também chamada de *grid*) e de fácil geração (veja Figura 1.1a). Entretanto, um *grid* não se adapta bem a determinados tipos de geometria como as geometrias curvas, gerando erros de aproximação na solução numérica. No FEM, a malha é geralmente não estruturada e composta de elementos, os quais usualmente são triângulos ou quadriláteros em duas dimensões, e tetraedros ou hexaedros em três dimensões, que se adaptam melhor à geometria do problema, inclusive às geometrias complexas. A Figura 1.1b ilustra um caso em que os elementos são triangulares.

Uma malha de boa qualidade impõe alguns requisitos no formato de seus elementos. Em duas dimensões, por exemplo, elementos triangulares devem ser o mais próximo possível de triângulos equiláteros. Se a malha contiver elementos mal formados, a precisão dos resultados numéricos é afetada. Desta forma, o método dos elementos finitos necessita, antes de mais nada, que uma malha de boa qualidade esteja disponível. O processo de geração de malha pode ser completamente automatizado sem necessidade de intervenção humana. Contudo, quando uma malha de qualidade é requerida, gerá-la de forma automática, torna-se uma tarefa difícil e, fatalmente, alguma intervenção humana é necessária [33]. Atualmente, em duas dimensões isso não é um problema: diversos geradores de malha estão disponíveis. Entretanto, a geração automática de malhas de boa qualidade em três dimensões ainda está em aberto, necessitando da intervenção humana na maior parte dos casos. Em problemas cuja geometria varia com o tempo, é preciso gerar, para cada instante de tempo, uma nova malha, agravando as dificuldades.



(a) malha estruturada - FDM

(b) malha não-estruturada - FEM

(c) nós - *Mfree*

Figura 1.1: Representação do domínio em diferentes métodos numéricos. (a) Malha estruturada (*grid*) usada no FDM. (b) Malha não-estruturada usada no FEM. (c) Nos métodos sem malha, nós são distribuídos sobre o domínio e sua fronteira. Figura retirada de [18].

Ao contrário dos métodos tradicionais, os métodos sem malha não necessitam de uma malha para representar o domínio. O conceito de método sem malha implica que o domínio do problema seja representado, idealmente, apenas por um conjunto de nós distribuídos arbitrariamente [33]. A aproximação é obtida através dos nós distribuídos ao longo do domínio e de sua fronteira (Figura 1.1c). Assim, as dificuldades associadas à geração de uma malha de boa qualidade, como ocorre com o FEM, são eliminadas. Além disso, a geração de nós pode ser feita de forma automática e sem dificuldades por meio de pré-processadores.

Diversos métodos sem malha foram desenvolvidos até hoje. Dentre eles, destacam-se o Smoothed Particle Hydrodynamics Method (SPH) [24], o Reproducing Kernel Particle Method (RKPM) [36], o Element-Free Galerkin Method (EFG) [7], o Meshless Local Petrov-Galerkin Method (MLPG) [6, 5], o Point Interpolation Method (PIM) [33]. Alguns trabalhos foram desenvolvidos combinando métodos sem malha com outros métodos como EFG ou MLPG acoplado ao FEM [32]. Também existem trabalhos onde diferentes tipos de funções de aproximação local, chamadas de *funções de forma*, são combinadas, gerando métodos mistos [19, 20]. Análise adaptativa e refinamento da solução numérica em métodos MFree também são outros focos de estudos que podem ser vistos em [22, 33].

Por ser uma área de pesquisa relativamente nova e em rápido desenvolvimento, muitos trabalhos envolvendo métodos sem malha são publicados na comunidade científica todos os anos. Entretanto, praticamente não existem disponíveis *frameworks* ou mesmo *softwares* que utilizem métodos sem malha para solução de problemas de valor de contorno, ao contrário do que ocorre com os métodos tradicionais. De forma sucinta, *framework* é um conjunto de classes cooperativas que formam um projeto reutilizável, destinado a uma classe específica de aplicações [23]. Um *framework* de *software* pode ser visto como um projeto capaz de construir outros projetos de *software* e aplicações, através da especialização de suas funcionalidades e da combinação dos blocos genéricos que ele oferece.

O objetivo deste trabalho é desenvolver um *framework* com o qual seja possível construir aplicações capazes de solucionar problemas de valor de contorno na área de eletromagnetismo, utilizando métodos sem malha. O *framework*, denominado *MFree Framework*, centraliza e disponibiliza os métodos MFree mais importantes e presentes na literatura, assim como as ferramentas com as quais novos métodos podem ser desenvolvidos e pesquisados no futuro.

O *framework* proposto é implementado na linguagem de programação C++ e segue uma estrutura genérica que permite instanciar qualquer tipo de método sem malha. Foi construído utilizando, primariamente, o paradigma da *programação genérica*, que é possível graças à existência do mecanismo *templates* presente na linguagem C++. A programação genérica foca em fornecer implementações genéricas de algoritmos e estruturas de dados que sejam tão eficientes quanto as implementações especializadas. Projetos desenvolvidos sob o paradigma de programação genérica, frequentemente, executam mais rápido que projetos desenvolvidos sob o paradigma convencional de orientação a objetos [23]. Como a resolução de problemas via métodos numéricos, em geral, envolve muito processamento de dados, operações matriciais e solução de grandes sistemas lineares, a eficiência é um importante requisito de projeto de *software*.

O MFree Framework, em sua versão atual, suporta a solução de problemas em uma (1D) e duas (2D) dimensões. O *framework* permite que os problemas possuam domínio composto de múltiplos materiais e com geometria formada por retas e curvas. Os métodos disponíveis são o EFG, o MLPG, o PIM e o método misto proposto em [19], mas outros são facilmente instanciáveis.

O presente texto está organizado da seguinte forma: o Capítulo 2 faz uma revisão geral sobre os conceitos básicos para este trabalho, incluindo *frameworks*, programação genérica, as equações fundamentais do eletromagnetismo e as formulações de problemas estáticos em uma e duas dimensões. O Capítulo 3 introduz os principais conceitos a respeito de métodos sem malha e descreve os métodos disponíveis na versão atual do *framework*. O Capítulo 4 detalha a estrutura do MFree Framework e suas funcionalidades. No Capítulo 5, exemplos são resolvidos e os resultados mostrados. Por fim, o Capítulo 6 traz as conclusões e propostas futuras.

Capítulo 2

Conceitos Básicos

Os *Frameworks* desempenham um papel importante no desenvolvimento de *software*, pois fornecem formas de reuso não apenas de código, mas também de análise e projeto. É graças à existência dos *frameworks* que várias aplicações de grande porte podem ser construídas rapidamente. Neste trabalho, é construído um *framework* destinado a solucionar problemas na área de eletromagnetismo utilizando métodos sem malha. O presente capítulo faz uma revisão geral sobre *frameworks* e suas aplicações. Aborda, também, os fundamentos da programação genérica, o paradigma de programação utilizado na construção do MFree Framework. Por fim, um resumo sobre a teoria eletromagnética, base para a solução dos problemas eletromagnéticos tratados neste texto, é apresentado.

2.1 Frameworks

Um *framework* de *software* é um conjunto de classes que incorpora um projeto abstrato de soluções para uma família de problemas relacionados [17]. É constituído de blocos de código com funcionalidade genérica, os quais podem ser especializados ou sobrescritos pelo programador, a fim de proporcionar uma funcionalidade específica. Um *framework* é reusável e seu código é envolvido por uma interface muito bem definida, através da qual as aplicações são construídas. Pode-se dizer que um *framework* é uma aplicação quase

completa, mas com algumas partes faltando. Tais partes são fornecidas pela aplicação para atingir o propósito desejado [17].

O *framework* determina a arquitetura da aplicação. Ele define a estrutura geral, a divisão em classes e objetos, as principais responsabilidades destes, como as classes e objetos colaboram entre si, e o fluxo de controle. Um *framework* predefine os parâmetros de projeto citados para que o projetista ou programador da aplicação possa se concentrar nas particularidades de sua aplicação. Assim, dão maior ênfase à reutilização de projeto que a de código, apesar de muitas vezes disponibilizar classes concretas com as quais é possível trabalhar diretamente [23].

Ao contrário das bibliotecas de *software*, os *frameworks* determinam o fluxo de controle da aplicação. Quando se usa uma biblioteca, o programador escreve o bloco principal da aplicação e se encarrega de efetuar as chamadas aos métodos disponíveis na biblioteca e de criar as colaborações entre os objetos. Quando se usa um *framework*, o programador reutiliza o bloco principal da aplicação e escreve o código chamado pelo *framework*. Neste caso, a comunicação entre objetos já está definida e o programador não necessita saber quando uma chamada de método é feita, pois é o *framework* quem a faz (veja Figura 2.1). Esta característica, presente nos *frameworks*, é denominada *inversão de controle*.

Consequentemente, através de *frameworks*, as aplicações podem ser construídas mais rapidamente. Elas terão estruturas similares, por isso, tornam-se mais fáceis para manutenção e mais consistentes para seus usuários. Por outro lado, como as decisões de projeto já foram tomadas pelo *framework*, os desenvolvedores perdem um pouco de liberdade [23], sendo obrigados a seguir os padrões definidos por sua arquitetura.

Atualmente, muitos *frameworks* estão disponíveis para auxiliar a construção de aplicações de *software*. A seguir, são apresentados alguns exemplos e os tipos de aplicações para os quais foram desenvolvidos.

- .NET Framework

  Também conhecido como Microsoft .NET, é um *framework* desenvolvido pela Microsoft, com o objetivo de criar uma plataforma única para desenvolvimento e execução de sistemas e aplicações. Através dele é possível construir aplicações de *console*,

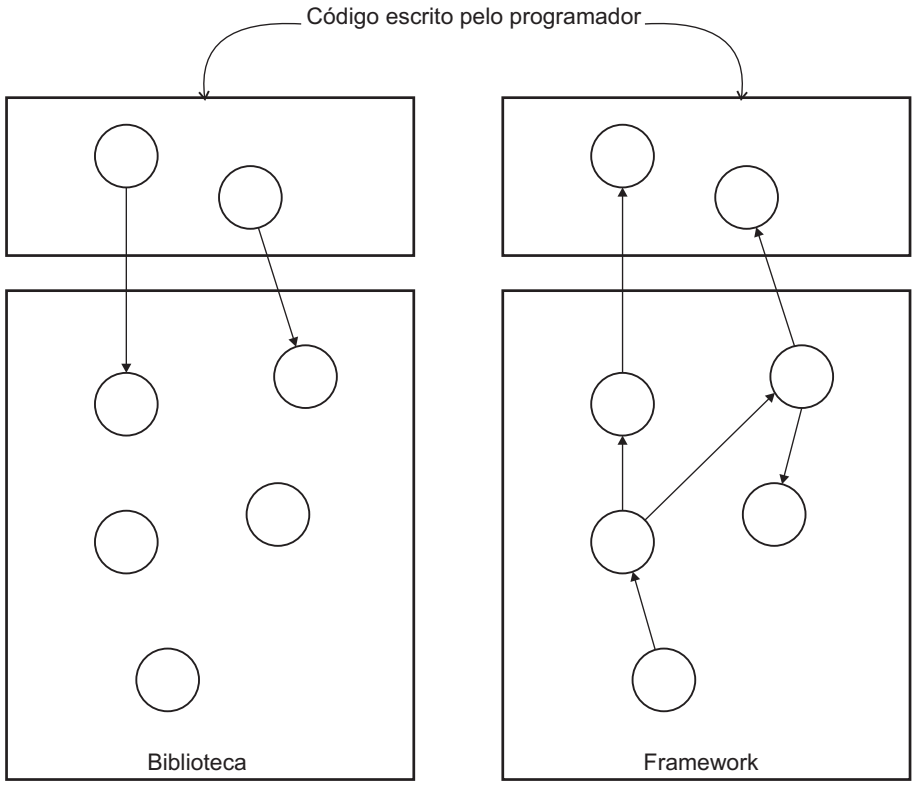


Figura 2.1: Fluxo de controle em bibliotecas e *frameworks*. Círculos representam objetos e setas representam colaborações entre os objetos. Observe que os objetos criados pelo programador fazem chamadas aos objetos da biblioteca, enquanto os objetos do *framework* fazem chamadas aos objetos criados pelo programador, ilustrando a inversão de controle.

aplicações que envolvem interfaces gráficas de usuário, aplicações para *Web*, dentre outros. Maiores detalhes em [40].

- Apache Struts

  É um *framework open-source*, desenvolvido pela Apache, destinado ao desenvolvimento de aplicações Java para *Web*. Utiliza a arquitetura *Model-View-Controller* (MVC) [23] e suporta as tecnologias AJAX e SOAP. Maiores detalhes em [4].

- Qt

  É um *framework* multi-plataforma, em C++, usado para constuir aplicações de interfaces gráficas de usuário e com capacidade de integração com a *Web*. Oferece ainda comunicação com banco de dados, integração com OpenGL e *multithreading*. Desenvolvido pela Nokia, o Qt está disponível em Linux, Mac OS X, Windows, Symbian e MeeGo. Maiores detalhes em [44].

- Brasil Finite Element Method – BRFEM

É um framework desenvolvido por um projeto cooperativo envolvendo a Universidade Federal de Minas Gerais, a Universidade de São Paulo e a Universidade Federal de Santa Catarina. Escrito na linguagem C++, utilizando o paradigma de programação orientada a objetos, o *framework* destina-se à construção de aplicações para cálculo de campo eletromagnético, através do método de elementos finitos. Suporta resolução de problemas em duas (2D) e três (3D) dimensões. Maiores detalhes em [54].

Enquanto o método dos elementos finitos possui uma grande quantidade de *softwares* e alguns *frameworks* disponíveis, praticamente ambos não existem para métodos sem malha e, quando existem, não englobam a solução de problemas de eletromagnetismo. O MFree Framework foi desenvolvido justamente para cobrir essa lacuna.

Um *framework* para métodos sem malha deve possibilitar o desenvolvimento de aplicações destinadas a solucionar problemas de valor de contorno utilizando métodos sem malha. Assim como o BRFEM, o MFree Framework é escrito em C++ e destina-se à solução de problemas de eletromagnetismo, mas é extensível a problemas de outras áreas. Uma das grandes diferenças entre os dois *frameworks* reside no paradigma de programação: enquanto o BRFEM utiliza a orientação a objetos, o MFree Framework é construído com base em programação genérica. A seção seguinte aborda os principais conceitos e características envolvidos na programação genérica.

2.2 Programação Genérica

Programação genérica é uma metodologia de desenvolvimento de *software* que se concentra no desenvolvimento de algoritmos e estruturas de dados genéricos em um certo nível de abstração que os torna capazes de serem aplicados a problemas diversos. O nível de abstração deve ser suficientemente adequado de forma a maximizar a reutilização de código sem sacrificar o desempenho, aspecto também importante da metodologia.

Enquanto o paradigma clássico de programação orientada a objetos (OO) busca representar o sistema de *software* através de objetos que se comunicam entre si, relacionados o

mais próximo possível da sua representação no mundo real [15], a programação genérica busca representar o sistema de *software* por estruturas de dados e algoritmos da forma mais genérica possível. Entretanto, os dois paradigmas não são mutuamente exclusivos: em muitos casos há combinação de ambos, mas sempre com predominância de um deles.

Em C++, a programação genérica é implementada por meio do mecanismo de *templates*, o qual permite que classes e funções sejam parametrizadas. A programação com *templates* é uma técnica muito eficiente uma vez que os tipos parametrizados são resolvidos em tempo de compilação [55]. Já a programação orientada a objetos é implementada através dos mecanismos de herança e funções virtuais. Especificamente, na OO a resolução de tipos é dinâmica (os tipos são resolvidos em tempo de execução), sendo menos eficiente que a programação genérica nesse aspecto.

A primeira etapa no desenvolvimento de um algoritmo genérico é a identificação do que é comum entre as implementações similares deste algoritmo. Suponha, por exemplo, um método que some dois inteiros e retorne o resultado. Suponha, também, um outro método com o mesmo propósito, mas que some dois números do tipo ponto flutuante. Os dois métodos são mostrados abaixo.

```
int soma(int x1, int x2)
{
  return x1 + x2;
}

float soma(float x1, float x2)
{
  return x1 + x2;
}
```

Observe que os dois métodos são quase idênticos, diferindo apenas no tipo dos parâmetros de entrada e de retorno. É possível criar um único algoritmo (mostrado a seguir), substituindo `int` e `float` pelo parâmetro *template* `T`. Com isso, tem-se um algoritmo genérico que consegue trabalhar com inteiros, pontos flutuantes e, em princípio, qualquer tipo numérico. Em vez de escrever uma versão do algoritmo `soma` para cada tipo numérico para o qual se deseja efetuar a soma entre dois elementos, escreve-se apenas um

algoritmo genérico *template*. Isso reduz o código da aplicação e também contribui para sua manutenção, uma vez que qualquer alteração de que o algoritmo necessite é realizada apenas em um único local. Do contrário, seria necessário efetuar as modificações em todos os métodos `soma` existentes na aplicação. Além disso, como o processo de substituição do parâmetro *template* `T` pelo tipo concreto é feito em tempo de compilação, o algoritmo genérico continua com a mesma eficiência que suas versões especializadas.

```
template <typename T>
T soma(T x1, T x2)
{
  return x1 + x2;
}
```

Na programação genérica, as abstrações são representadas pelos requisitos existentes sobre os parâmetros de algoritmos e estruturas genéricas. Um conjunto de requisitos forma *conceitos*. Um algoritmo genérico é descrito por um ou mais conceitos que devem ser satisfeitos em sua implementação concreta [55]. Tome, novamente como exemplo, a função *template* `soma`. Observe que a passagem de parâmetros é feita por cópia. Para que isso seja possível, é necessário que o tipo `T` tenha um construtor de cópia presente em sua implementação. Além disso, para que a soma `x1 + x2` possa ser realizada, `T` deve fornecer um operador de adição `+` em sua interface. Existem, portanto, dois requisitos impostos sobre o tipo `T`: (1) possuir construtor de cópia e (2) possuir operador de adição `+`. Os conceitos relacionados a estes requisitos são, respectivamente, (1) `CopyConstructible<T>` e (2) `Addable<T>`. Qualquer tipo que não satisfaça estes dois conceitos não pode ser usado como parâmetro *template* no método `soma`. Na tentativa de uso de um tipo inadequado, um erro de compilação é gerado. Note que, através dos conceitos, os requisitos sobre os parâmetros *template* de métodos ficam clara e totalmente descritos.

Na programação genérica, freqüentemente *traits* são usadas. Uma classe *traits* fornece mapeamentos de tipos, funções ou constantes. *Traits* são importantes porque permitem que decisões sejam tomadas fora do contexto imediato em que são utilizadas e porque as decisões com base em tipos são tomadas em tempo de compilação [3]. Resoluções de tipo e decisões quando tomadas em tempo de compilação resultam em algoritmos mais eficientes. Considere um exemplo simples para melhor compreensão do papel das *traits*.

Imagine uma função *template* `foo`, de propósito irrelevante, que receba um parâmetro *template* que represente um *array*, e que declare um objeto do tipo `X` do mesmo tipo dos objetos armazenados no *array*. A função é mostrada abaixo.

```
template <typename Array>
void foo(const Array& v, int n)
{
  X x;
  ...
}
```

A questão presente é como determinar o tipo `X`. É possível acessá-lo através do próprio tipo `Array` via operador de escopo `::`. Imagine que a classe `Array` defina o tipo dos elementos armazenados através do atributo `value_type`. Ao trocar `X` por `typename Array::value_type` na função `foo` definida acima, o problema é solucionado. A deficiência dessa estratégia é que ela só funciona caso a classe `Array` defina o tipo interno `value_type`. Para classes criadas pelo próprio programador, isso não é um problema. No entanto, se for desejável usar uma classe criada por outro programador que não define `value_type`, ou até mesmo um ponteiro para inteiros (`int*`), por exemplo, essa estratégia não funciona. A solução para esse problema reside no uso de *traits*. Cria-se uma classe `array_traits` capaz de efetuar o mapeamento através de `Array` para o tipo correspondente aos elementos armazenados. Cria-se, também, especializações de `array_traits` para tipos específicos, a saber, para `int*`. Agora, a função *template* `foo` obtém o tipo `X` através da classe *traits* `array_traits`. As classes traits e a nova função `foo` são mostradas abaixo.

```
//traits base
template <typename Array>
struct array_traits
{
  typedef typename Array::value_type    value_type;
}

//traits especializada para int*
template <>
```

```
struct array_traits<int*>
{
  typedef int    value_type;
}

template <typename Array>
void foo(const Array& v, int n)
{
  typename array_traits<Array>::value_type x;
  ...
}
```

Observe que o tipo dos elementos armazenados por `Array` é obtido de forma indireta através de `array_traits`. Isso traz a vantagem de se poder utilizar qualquer tipo para `Array` desde que se forneça uma versão de `array_traits` correspondente, isto é, os tipos existentes não precisam ser modificados para funcionar com a função `foo`, basta escrever uma classe `traits` especializada.

Talvez a contribuição mais significante na área de programação genérica é a *Standard Template Library* (STL) [2], que foi adaptada e incorporada à bilbioteca padrão do C++. A STL é uma biblioteca que fornece um número grande de operações úteis, chamadas de algoritmos, e estruturas de dados que funcionam como coleções de objetos, denominadas *containers*. Tanto os algoritmos quanto os *containers* são *templates*. Entretanto, os algoritmos não são funções membro dos *containers*, como normalmente acontece no paradigma clássico de orientação a objetos. Ao invés disso, os algoritmos são escritos de um modo genérico que permite que sejam usados por qualquer *container*. Para isso, os projetistas da STL introduziram o conceito abstrato de *iterators* [55]. Um *iterator* é uma estrutura responsável por percorrer os elementos armazenados nas coleções. Todo algoritmo na STL trabalha com *iterators* e cada *container* fornece seus próprios *iterators*. Desse modo, os algoritmos são escritos uma única vez e usados por todas as coleções. Isso reduz drasticamente o número de definições de operações contidas na STL.

Outro exemplo de aplicação da programação genérica é a *Computational Geometry Algorithms Library* (CGAL) [10]. A CGAL é uma biblioteca de geometria computacional, cujo propósito principal é fornecer acesso fácil a algoritmos geométricos eficientes e confiáveis. A biblioteca, escrita em C++, é usada em várias áreas como computação gráfica, sistemas de informações geográficas, biologia molecular, imagens médicas, robótica, geração

de malhas, métodos numéricos, dentre outras. O MFree Framework utiliza a CGAL para as questões de geometria computacional envolvidas nos métodos sem malha.

A *Matrix Template Library 4* (MTL4) [25] também é baseada em programação genérica e escrita em C++. A MTL4 é uma biblioteca para representação matricial e álgebra linear, construída visando trabalhar em alto desempenho. Ela é usada pelo MFree Framework para representação e operação matricial.

Um *framework* construído sob o paradigma da programação genérica é uma poderosa ferramenta de desenvolvimento de *software*. Ela combina a facilidade de construir aplicações usando um *framework* e as características da programação genérica tais como flexibilidade, generalidade e eficiência. De acordo com [23], projetos desenvolvidos sob o paradigma de programação genérica frequentemente executam mais rápido que projetos desenvolvidos sob o paradigma convencional de orientação a objetos. Esta é a proposta do MFree Framework.

2.3 Problemas Estáticos em Eletromagnetismo

Problemas eletromagnéticos são descritos pela teoria eletromagnética cujos postulados são matematicamente expressos por um conjunto de equações fundamentais: as *equações de Maxwell*, as *relações constitutivas* e a *equação da força de Lorentz* [37]. Duas áreas do eletromagnetismo em que diversos problemas se enquadram são a *eletrostática* e a *magnetostática*. Nessas áreas, as condições são estáticas, isto é, as grandezas físicas são constantes no tempo e as equações de Maxwell tornam-se mais simples.

Um problema de valor de contorno é descrito por uma formulação denominada *forma forte*, que é composta por conjuntos de equações diferenciais e de condições de contorno. As mais comuns destas são a condição de contorno essencial ou de Dirichlet e a condição de contorno natural ou de Neumann. A primeira envolve a imposição de valores sobre a solução da equação diferencial governante, enquanto a segunda, a imposição de valores sobre a derivada da solução da equação diferencial governante. O MFree Framework, atualmente, implementa formulações para problemas eletrostáticos e magnetostáticos com condições

de contorno essenciais e naturais, mas outras formulações e condições de contorno podem ser escritas pelo desenvolvedor.

Esta seção apresenta, resumidamente, as equações fundamentais do eletromagnetismo, da eletrostática e da magnetostática. As formas fortes para problemas eletrostáticos e magnetostáticos em duas dimensões são desenvolvidas e, por fim, apresentadas as formulações generalizadas para problemas estáticos em uma e duas dimensões.

2.3.1 Equações Fundamentais do Eletromagnetismo

Um campo eletromagnético se caracteriza por quatro funções vetoriais da posição e do tempo: o *campo elétrico* $\overrightarrow{E}$ $[V/m]$, a *indução elétrica* ou *densidade de fluxo elétrico* $\overrightarrow{D}$ $[C/m^2]$, o *campo magnético* $\overrightarrow{H}$ $[A/m]$ e a *indução magnética* ou *densidade de fluxo magnético* $\overrightarrow{B}$ $[T$, ou $Wb/m^2]$ [37]. As leis básicas do campo eletromagnético são as equações de Maxwell, que envolvem esses quatro vetores e são

$$\nabla \cdot \overrightarrow{D} = \rho \qquad \text{(Lei de Gauss da eletricidade)} \tag{2.1}$$

$$\nabla \times \overrightarrow{H} = \overrightarrow{J} + \frac{\partial \overrightarrow{D}}{\partial t} \qquad \text{(Lei de Ampère generalizada)} \tag{2.2}$$

$$\nabla \cdot \overrightarrow{B} = 0 \qquad \text{(Lei de Gauss do magnetismo)} \tag{2.3}$$

$$\nabla \times \overrightarrow{E} = -\frac{\partial \overrightarrow{B}}{\partial t} \qquad \text{(Lei de Faraday da indução)} \tag{2.4}$$

onde $\overrightarrow{J}$ é a densidade superficial de corrente elétrica $[A/m^2]$ e ρ é a densidade volumar de carga elétrica $[C/m^3]$.

As equações de Maxwell não são as únicas relações existentes entre os vetores de campo. As relações adicionais variam de meio para meio conforme a constituição do material de cada um e são chamadas de relações constitutivas:

$$\overrightarrow{D} = \overrightarrow{\overrightarrow{\epsilon}} \cdot \overrightarrow{E} \tag{2.5}$$

$$\overrightarrow{B} = \overrightarrow{\overrightarrow{\mu}} \cdot \overrightarrow{H} \tag{2.6}$$

$$\overrightarrow{J} = \overrightarrow{\overrightarrow{\sigma}} \cdot \overrightarrow{E} \tag{2.7}$$

onde o tensor $\overrightarrow{\overrightarrow{\epsilon}}$ é a permissividade elétrica do meio $[F/m]$, $\overrightarrow{\overrightarrow{\mu}}$ sua permeabilidade magnética $[H/m]$ e $\overrightarrow{\overrightarrow{\sigma}}$ sua condutividade elétrica $[(\Omega m)^{-1}]$. Em conjunto, estes três tensores se denominam *propriedades constitutivas do meio*.

Um meio é dito linear se suas propriedades constitutivas não dependem dos vetores $\overrightarrow{E}$, $\overrightarrow{D}$, $\overrightarrow{H}$, $\overrightarrow{B}$ e $\overrightarrow{J}$. É isotrópico quando tiver, em torno de cada ponto, as mesmas propriedades em diferentes direções. Nesses meios, a resposta a um estímulo aplicado em torno de um ponto não pode depender da direção de onde veio esse estímulo. Consequentemente, num meio isotrópico, a permissividade, a permeabilidade e a condutividade devem ser escalares [37]. Neles, as relações constitutivas simplificam-se para

$$\overrightarrow{D} = \epsilon \overrightarrow{E} \tag{2.8}$$

$$\overrightarrow{B} = \mu \overrightarrow{H} \tag{2.9}$$

$$\overrightarrow{J} = \sigma \overrightarrow{E} \tag{2.10}$$

A isotropia não impede que os escalares ϵ, μ e σ dependam do ponto. Um meio é homogêneo ou uniforme quando suas propriedades constitutivas são uniformes, ou seja, quando não variam no espaço. Em qualquer meio linear e isotrópico, homogêneo ou não, as propriedades constitutivas são positivas, sendo que a condutividade pode ser nula. Um meio cuja condutividade elétrica é não-nula é chamado de condutor, caso contrário, dielétrico perfeito ou isolante perfeito.

Ainda é necessária uma equação que descreva o comportamento de cargas na presença de um campo eletromagnético. Essa equação, denominada equação da força de Lorentz, é

$$\overrightarrow{f} = \rho \overrightarrow{E} + \overrightarrow{J} \times \overrightarrow{B} \tag{2.11}$$

onde $\overrightarrow{f}$ é a densidade volumar da força de Lorentz $[N/m^3]$, que representa a força por unidade de volume que o campo eletromagnético exerce sobre uma densidade volumar de carga ρ e uma densidade superficial de corrente $\overrightarrow{J}$.

As equações de Maxwell são equações diferenciais e suas derivadas devem existir na região em que estejam sendo aplicadas. Há, no entanto, um tipo importante de situação em que isso não acontece e, nesse caso, é preciso saber como abordá-lo. Trata-se do comportamento do campo eletromagnético na travessia de uma superfície que separa dois meios de

propriedades constitutivas diferentes. Nessa superfície, chamada *fronteira* ou *interface*, pode haver ou não uma descontinuidade, um salto, nos valores do campo medidos por um observador de cada lado da fronteira [37]. As equações que especificam as descontinuidades na fronteira entre meios, denominadas *condições de interface*, são deduzidas a partir das equações de Maxwell na forma integral e das relações constitutivas. A dedução das condições de interface não é tratada neste trabalho e as equações são apenas apresentadas. Sua obtenção é apresentada em [37].

Seja um domínio Ω composto de dois meios arbitrariamente numerados 1 e 2, conforme mostra a Figura 2.2. O vetor unitário $\hat{\mathbf{n}}$ normal à interface Γ_I entre os meios, por convenção, é dirigido do meio 1 para o 2. As condições de interface são dadas por

$$\hat{\mathbf{n}} \cdot (\overrightarrow{D_2} - \overrightarrow{D_1}) = \rho_s \tag{2.12}$$

$$\hat{\mathbf{n}} \cdot (\overrightarrow{B_2} - \overrightarrow{B_1}) = 0 \tag{2.13}$$

$$\hat{\mathbf{n}} \times (\overrightarrow{H_2} - \overrightarrow{H_1}) = \overrightarrow{J_l} \tag{2.14}$$

$$\hat{\mathbf{n}} \times (\overrightarrow{E_2} - \overrightarrow{E_1}) = \overrightarrow{0} \tag{2.15}$$

$$\hat{\mathbf{n}} \cdot (\overrightarrow{J_2} - \overrightarrow{J_1}) = -\frac{\partial \rho_s}{\partial t} \tag{2.16}$$

onde ρ_s é a densidade superficial de carga $[C/m^2]$ e $\overrightarrow{J_l}$ é a densidade linear de corrente $[A/m]$, ambas na interface.

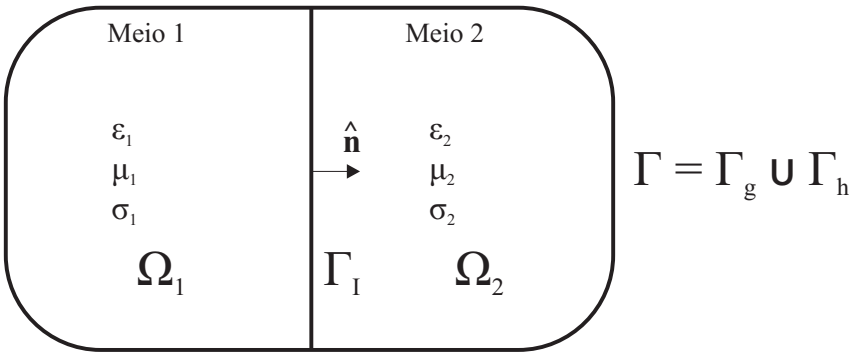


Figura 2.2: Representação de um domínio em 2D. O domínio Ω é composto de dois meios diferentes Ω_1 e Ω_2 ($\Omega = \Omega_1 \cup \Omega_2$). Γ é o contorno externo de Ω ($\Gamma = \partial\Omega$) e é formado pela união entre Γ_g (contorno onde se aplicam as condições de contorno de Dirichlet) e Γ_h (contorno onde se aplicam as condições de contorno de Neumann), com $\Gamma_g \cap \Gamma_h = \emptyset$. Γ_I é a interface entre os meios.

2.3.2 Problemas Eletrostáticos

A eletrostática é uma área do eletromagnetismo onde apenas o campo elétrico está presente e este é estático, ou seja, constante no tempo. Não há presença de campo magnético nem de correntes elétricas, e todas as grandezas eletromagnéticas são constantes no tempo. Considerando um meio isotrópico e desprezando as variações temporais, as equações do eletromagnetismo são simplificadas para

$$\nabla \cdot \overrightarrow{D} = \rho \tag{2.17}$$

$$\nabla \times \overrightarrow{E} = 0 \tag{2.18}$$

$$\overrightarrow{D} = \epsilon \overrightarrow{E} \tag{2.19}$$

$$\overrightarrow{f} = \rho \overrightarrow{E} \tag{2.20}$$

$$\hat{\mathbf{n}} \cdot (\overrightarrow{D_2} - \overrightarrow{D_1}) = \rho_s \tag{2.21}$$

$$\hat{\mathbf{n}} \times (\overrightarrow{E_2} - \overrightarrow{E_1}) = \overrightarrow{0} \tag{2.22}$$

A Equação 2.18 mostra que o campo elétrico é irrotacional, por isso é conservativo. Logo, pode-se expressá-lo como o gradiente de uma função escalar V, denominada *potencial escalar elétrico*:

$$\nabla \times \overrightarrow{E} = 0 \quad \Longrightarrow \quad \overrightarrow{E} = -\nabla V \tag{2.23}$$

Substituindo 2.19 em 2.17

$$\nabla \cdot \overrightarrow{D} = \rho \quad \Longrightarrow \quad \nabla \cdot (\epsilon \overrightarrow{E}) = \rho$$

e usando a Equação 2.23, chega-se a

$$\nabla \cdot (\epsilon \nabla V) = -\rho \tag{2.24}$$

que é uma equação de Poisson em V e ρ. Através das Equações 2.17, 2.19, 2.21 e 2.23 pode-se mostrar que

$$\epsilon_1 \frac{\partial V_1}{\partial n} - \epsilon_2 \frac{\partial V_2}{\partial n} = \rho_s \tag{2.25}$$

É apresentada, abaixo, a forma forte para problemas eletrostáticos, estabelecidos da seguinte maneira:

Forma forte

Dados ϵ, ρ e ρ_s, determinar a função $V : \Omega \to \Re$ que satisfaça

$$\begin{aligned} \nabla \cdot (\epsilon \nabla V) &= -\rho && em \quad \Omega \\ V &= g && em \quad \Gamma_g \\ -\epsilon \frac{\partial V}{\partial n} &= h && em \quad \Gamma_h \\ \epsilon_1 \frac{\partial V_1}{\partial n} - \epsilon_2 \frac{\partial V_2}{\partial n} &= \rho_s && em \quad \Gamma_I \end{aligned} \tag{2.26}$$

onde g e h são os valores impostos pelas condições de Dirichlet e Neumann, respectivamente.

2.3.3 Problemas Magnetostáticos

Assim como a eletrostática, a magnetostática é uma área do eletromagnetismo em que as grandezas eletromagnéticas são constantes no tempo. Entretanto, aqui há presença de um campo magnético ao invés de um campo elétrico. Na magnetostática pode ou não haver correntes elétricas, mas, são constantes no tempo quando estas existem. Considerando um meio isotrópico e desprezando as variações temporais, as equações do eletromagnetismo são simplificadas para

$$\nabla \cdot \overrightarrow{B} = 0 \tag{2.27}$$

$$\nabla \times \overrightarrow{H} = \overrightarrow{J} \tag{2.28}$$

$$\overrightarrow{B} = \mu \overrightarrow{H} \tag{2.29}$$

$$\overrightarrow{f} = \overrightarrow{J} \times \overrightarrow{B} \tag{2.30}$$

$$\hat{\mathbf{n}} \cdot (\overrightarrow{B_2} - \overrightarrow{B_1}) = 0 \tag{2.31}$$

$$\hat{\mathbf{n}} \times (\overrightarrow{H_2} - \overrightarrow{H_1}) = \overrightarrow{J_l} \tag{2.32}$$

Através da Equação 2.27, vê-se que a indução magnética possui divergente nulo, por isso é solenoidal. É possível, portanto, expressá-la como o rotational de uma função vetorial $\overrightarrow{A}$, denominada *potencial vetor magnético*:

$$\nabla \cdot \overrightarrow{B} = 0 \quad \Longrightarrow \quad \overrightarrow{B} = \nabla \times \overrightarrow{A} \tag{2.33}$$

Substituindo 2.29 em 2.28

$$\nabla \times \overrightarrow{H} = \overrightarrow{J} \quad \Longrightarrow \quad \nabla \times \frac{1}{\mu}\overrightarrow{B} = \overrightarrow{J}$$

e usando a Equação 2.33, chega-se a

$$\nabla \times \frac{1}{\mu}\nabla \times \overrightarrow{A} = \overrightarrow{J} \tag{2.34}$$

Para problemas em duas dimensões, o campo magnético tem componentes apenas nas direções x e y. Pela lei de Ampère (Equação 2.28), a densidade de corrente elétrica deve ter componentes apenas em z, ou seja, $\overrightarrow{J} = J_z\hat{\mathbf{z}}$. Pela definição 2.33, o potencial vetor magnético $\overrightarrow{A}$ também deve possuir componentes apenas na direção z, isto é, $\overrightarrow{A} = A_z\hat{\mathbf{z}}$, e da definição de rotacional, vem

$$\overrightarrow{B} = \begin{vmatrix} \hat{\mathbf{x}} & \hat{\mathbf{y}} & \hat{\mathbf{z}} \\ \dfrac{\partial}{\partial x} & \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial z} \\ 0 & 0 & A_z \end{vmatrix} = \frac{\partial A_z}{\partial y}\hat{\mathbf{x}} - \frac{\partial A_z}{\partial x}\hat{\mathbf{y}}$$

Substituindo a equação acima em 2.28

$$\nabla \times \overrightarrow{H} = \begin{vmatrix} \hat{\mathbf{x}} & \hat{\mathbf{y}} & \hat{\mathbf{z}} \\ \dfrac{\partial}{\partial x} & \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial z} \\ \dfrac{1}{\mu}\dfrac{\partial A_z}{\partial y} & -\dfrac{1}{\mu}\dfrac{\partial A_z}{\partial x} & 0 \end{vmatrix}$$

$$= \frac{\partial}{\partial z}\left(\frac{1}{\mu}\frac{\partial A_z}{\partial x}\right)\hat{\mathbf{x}} + \frac{\partial}{\partial z}\left(\frac{1}{\mu}\frac{\partial A_z}{\partial y}\right)\hat{\mathbf{y}} - \left[\frac{\partial}{\partial x}\left(\frac{1}{\mu}\frac{\partial A_z}{\partial x}\right) + \frac{\partial}{\partial y}\left(\frac{1}{\mu}\frac{\partial A_z}{\partial y}\right)\right]\hat{\mathbf{z}}$$
$$= J_z\hat{\mathbf{z}}$$

Como $\overrightarrow{A}$ possui componente apenas na direção z:

$$\frac{\partial}{\partial x}\left(\frac{1}{\mu}\frac{\partial A_z}{\partial x}\right) + \frac{\partial}{\partial y}\left(\frac{1}{\mu}\frac{\partial A_z}{\partial y}\right) = -J_z$$

ou

$$\nabla \cdot (\frac{1}{\mu}\nabla A_z) = -J_z \tag{2.35}$$

que é uma equação de Poisson em A_z e J_z. Através das Equações 2.29, 2.32 e 2.33 pode-se mostrar que

$$\frac{1}{\mu_1}\frac{\partial A_{z1}}{\partial n} - \frac{1}{\mu_2}\frac{\partial A_{z2}}{\partial n} = J_l \tag{2.36}$$

onde J_l é a componente z da densidade linear de corrente elétrica $\overrightarrow{J_l}$.

É apresentada, abaixo, a forma forte para problemas magnetostáticos, estabelecidos da seguinte maneira:

Forma forte

Dados μ, J_z e J_l, determinar a função $A_z : \Omega \subset \Re^2 \rightarrow \Re$ que satisfaça

$$\begin{aligned}
\nabla \cdot (\frac{1}{\mu}\nabla A_z) &= -J_z && em \quad \Omega \\
A_z &= g && em \quad \Gamma_g \\
-\frac{1}{\mu}\frac{\partial A_z}{\partial n} &= h && em \quad \Gamma_h \\
\frac{1}{\mu_1}\frac{\partial A_{z1}}{\partial n} - \frac{1}{\mu_2}\frac{\partial A_{z2}}{\partial n} &= J_l && em \quad \Gamma_I
\end{aligned} \tag{2.37}$$

onde g e h são os valores impostos pelas condições de Dirichlet e Neumann, respectivamente.

2.3.4 Generalização de Problemas Estáticos em 2D

Como se vê nas Equações 2.26 e 2.37, as formas fortes para problemas eletrostáticos e magnetostáticos em duas dimensões são semelhantes e podem ser generalizadas a uma formulação comum. Com efeito, vários problemas de valor de contorno são governados por equações diferenciais do tipo equação de Poisson. Seja o domínio Ω mostrado na Figura 2.2. De forma geral, um problema estático 2D é estabelecido da seguinte maneira:

Dados k, f e p, determinar a função $u : \Omega \subset \Re^2 \rightarrow \Re$ que satisfaça

$$\begin{aligned} \nabla \cdot (k\nabla u) &= f && em \quad \Omega \\ u &= g && em \quad \Gamma_g \\ -k\frac{\partial u}{\partial n} &= h && em \quad \Gamma_h \\ k_1\frac{\partial u_1}{\partial n} - k_2\frac{\partial u_2}{\partial n} &= p && em \quad \Gamma_I \end{aligned} \tag{2.38}$$

onde k está ligado à característica de material, f ao termo fonte e p ao termo fonte localizado na interface de materiais; g e h são os valores impostos pelas condições de Dirichlet e Neumann, respectivamente.

Se o problema tratado for eletrostático, então os parâmetros de 2.38 possuem a correspondência:

$$u = V \qquad k = \epsilon \qquad f = -\rho \qquad p = \rho_s$$

e se for um magnetostático, então:

$$u = A_z \qquad k = \frac{1}{\mu} \qquad f = -J_z \qquad p = J_l$$

Cabe ressaltar que as formas fortes apresentadas possuem apenas condições de contorno de Dirichlet e de Neumann. Entretanto, existem outros tipos de condições de contorno que podem ser aplicadas, como as condições de contorno mistas, que combinam as de Dirichlet e de Neumann simultaneamente.

2.3.5 Generalização de Problemas Estáticos em 1D

Da mesma forma que pode-se generalizar problemas estáticos em duas dimensões, é possível desenvolver uma formulação geral para problemas estáticos em uma dimensão. O processo de obtenção da forma forte é o mesmo feito nas seções anteriores, por isso a Equação 2.38 (em 2D) é particularizada para 1D. Por simplicidade, são considerados problemas com apenas um material.

Seja o domínio Ω, em uma dimensão, constituído por um meio, como mostra a Figura 2.3. Observe que a fronteira Γ se reduz a pontos e Ω a um segmento. Em uma dimensão,

existe apenas a componente x. Portanto, gradiente e divergente se reduzem a derivadas em relação a x. Além disso, a derivada $\frac{\partial u}{\partial n}$ também se torna $\frac{du}{dx}$. Consequentemente, a forma forte para um problema estático 1D é estabelecida da seguinte maneira:

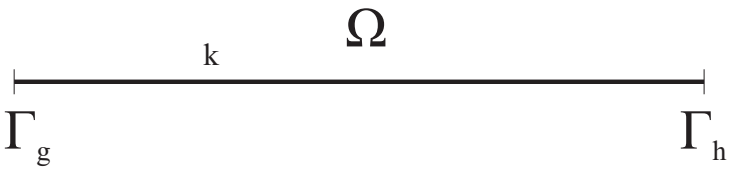


Figura 2.3: Representação de um domínio em 1D. O domínio Ω é composto de um meio de propriedade k. Γ é o contorno de Ω ($\Gamma = \partial\Omega$) e é formado pela união entre Γ_g (contorno onde se aplicam as condições de contorno de Dirichlet) e Γ_h (contorno onde se aplicam as condições de contorno de Neumann), com $\Gamma_g \cap \Gamma_h = \emptyset$.

Dados k e f, determinar a função $u : \Omega \subset \Re \rightarrow \Re$ que satisfaça

$$\begin{aligned} \frac{du}{dx}\left(k\frac{du}{dx}\right) &= f && em \quad \Omega \\ u &= g && em \quad \Gamma_g \\ -k\frac{du}{dx} &= h && em \quad \Gamma_h \end{aligned} \tag{2.39}$$

onde k está ligado à característica de material e f ao termo fonte; g e h são os valores impostos pelas condições de Dirichlet e Neumann, respectivamente.

Capítulo 3

Métodos sem Malha

Como já discutido previamente, métodos sem malha são métodos numéricos destinados a solucionar problemas de valor de contorno. Eles foram criados com o objetivo de eliminar a necessidade de geração de uma malha, por isso trabalham apenas com nós sem uma conectividade pré-estabelecida entre eles [33].

O processo de solução de um PVC através de métodos sem malha consiste, basicamente, em distribuir nós sobre o domínio do problema e suas fronteiras (veja Figura 1.1c) e construir funções de forma para cada um dos nós (nesse passo um sistema de equações é montado e depois resolvido). Ao final, tem-se uma solução aproximada do problema em todo o domínio.

Existem duas grandes categorias de métodos numéricos para resolver problemas de valor de contorno: métodos diretos e indiretos. Métodos diretos, conhecidos como métodos de *forma forte* (por exemplo, o FDM e métodos de colocação), discretizam e resolvem o problema diretamente. Métodos indiretos, também chamados de métodos de *forma fraca* (por exemplo, o FEM), estabelecem primeiro um sistema de equações alternativo, composto de uma forma fraca que governa o mesmo fenômeno físico, e depois o soluciona. A ideia essencial de um método de forma fraca é estimar o comportamento global do sistema como um todo e, assim, encontrar a melhor solução dentre as possíveis, de modo a levar o sistema ao estado de equilíbrio [33].

A forma fraca é derivada da forma forte, geralmente, de duas maneiras: através de *princípios de energia* (abordagem física) ou pelo método dos *resíduos ponderados* (abordagem matemática), sendo o último mais geral. A forma fraca exige condições de diferenciabilidade *mais fracas* que a forma forte. Isso é conseguido, matematicamente, introduzindo uma *função de teste*, na *forma integral* de uma formulação residual, para "absorver" uma derivada da forma forte, convertendo esta em uma forma fraca. Portanto, as equações da forma fraca são expressas de modo *integral*, enquanto as da forma forte de modo *diferencial*.

Nos métodos sem malha, tanto formas fortes quanto fracas são usadas [33]. O Finite Point Method [45] é um método MFree que utiliza a forma forte, enquanto o EFG, MLPG e PIM utilizam as fracas. Este trabalho foca em métodos baseados em forma fraca, geralmente mais robustos, estáveis, precisos, confiáveis, eficientes, por isso de maior importância prática.

As principais diferenças entre os métodos sem malha que utilizam formulações fracas se encontram (1) na maneira em que são calculadas as funções de forma, (2) se a formulação utilizada é uma forma fraca global ou local e (3) como é realizada a integração da formulação. Funções de forma são funções utilizadas para gerar aproximações locais para uma variável de interesse. Podem ser geradas por meio de diferentes métodos, sendo os mais utilizados o Moving Least Squares (MLS), funções de Shepard, Point Interpolation Methods (PIMs) - com funções de base polinomiais, radiais, e radiais em conjunto com polinomiais -, Partition of Unity (PU), Smoothed Particle Hydrodynamics (SPH) e Reproducing Kernel Particle Method (RKPM). Os três primeiros são implementados pelo *framework*.

A forma fraca pode ser *global* ou *local*. O termo *global* implica que a forma fraca deve ser satisfeita globalmente, ou seja, em todo o domínio do problema. Já o termo *local* requer que a forma fraca seja satisfeita localmente em subregiões do domínio, e a união de todas as subregiões deve cobrir todo o domínio do problema. Dois métodos são utilizados para discretizar a forma fraca: o de Galerkin (também conhecido como Bubnov-Galerkin) e o de Petrov-Galerkin. A diferença entre eles está na escolha das funções de base utilizadas para construção das funções de forma e de teste. No primeiro, as funções de base são as mesmas para as de forma e de teste; no segundo, as funções de base são diferentes.

Outra diferença entre os métodos sem malha é como efetuar a integração da forma fraca durante o processo de montagem do sistema de equações. Os dois principais tipos usados em MFree são a integração por *malha de fundo* (ou *malha de integração*) e por *subdomínios locais*. No primeiro caso, uma malha formada por células é gerada cobrindo todo o domínio, e a integração é realizada em cada célula. Normalmente, as células são triângulos ou retângulos em 2D e tetraedros ou hexaedros em 3D, por simplicidade. Deve ser pontuado que a malha de fundo não é igual à malha utilizada em elementos finitos (a segunda necessita de muito mais requisitos de qualidade que a primeira). No MFree, a malha é usada apenas para integração. No FEM, é usada tanto para integração quanto para cálculo das funções de forma e, consequentemente, deve ter seus elementos o mínimo distorcidos. A geração de uma malha de integração para métodos sem malha é mais fácil e flexível que a geração de uma malha para o FEM.

No caso da integração por subdomínios locais, para cada nó é criado um subdomínio, chamado *domínio de integração* ou *domínio de quadratura*. A união de todos deve cobrir completamente o domínio do problema, podendo haver ou não superposição entre eles. A integração é feita em cada um dos domínios de quadratura de forma independente. Estes são, geralmente, círculos ou retângulos em 2D e esferas ou hexaedros em 3D, por simplicidade. O tipo de integração empregado em um método sem malha está diretamente ligado à forma fraca utilizada: se for fraca global, a integração por malha de fundo é empregada; se fraca local, a integração é por subdomínios locais.

Este capítulo descreve os principais conceitos envolvidos em métodos sem malha e detalha os diferentes processos de geração de funções de forma, assim como os métodos sem malha existentes no *framework*.

3.1 Aproximação Local, Domínio de Suporte e Domínio de Influência

Suponha que se queira determinar a aproximação u^h de uma determinada função em um ponto arbitrário $\mathbf{x} = (x, y, z)$, pertencente ao domínio do problema, e que se conheça

os valores da função em alguns pontos do domínio, chamados *nós*. A aproximação (ou interpolação) $u^h(\mathbf{x})$ é dada por

$$u^h(\mathbf{x}) = \sum_{i \in S_n} \phi_i(\mathbf{x}) u_i = \mathbf{\Phi}(\mathbf{x}) \mathbf{U}_S \tag{3.1}$$

onde S_n é o conjunto de nós pertencentes a um domínio local compacto referente ao ponto $\mathbf{x}$ (o domínio local é chamado de *domínio de suporte* e os nós de *nós de suporte*), u_i é o *parâmetro nodal* do i-ésimo nó do domínio de suporte, $\mathbf{U}_S$ é o vetor que contém todos os parâmetros nodais dos nós de suporte, $\phi_i(\mathbf{x})$ é a função de forma do i-ésimo nó criada usando todos os nós de suporte e é chamada de *função de forma nodal* e $\mathbf{\Phi}(\mathbf{x})$ é o vetor contendo todas as funções de forma nodais.

O domínio de suporte de um ponto $\mathbf{x}$ determina o número de nós usados para aproximar o valor da função em $\mathbf{x}$. Um domínio de suporte pode ser *ponderado* usando funções que se anulam em sua fronteira, como mostra a Figura 3.1b. Dessa forma, cada nó de suporte recebe um grau de importância na aproximação em $\mathbf{x}$. O domínio de suporte pode ter diferentes formas, sendo que o tamanho e formato podem variar entre diferentes pontos de interesse $\mathbf{x}$, como mostrado na Figura 3.1a. Os formatos mais usados são circular ou retangular. [33]

Geralmente, os domínios de suporte são determinados de acordo com a densidade nodal local. A razão para isso é que os domínios de suporte devem ajustar sua dimensão de acordo com o número de nós contidos na vizinhança do ponto, de modo que o número de nós de suporte não seja muito discrepante entre diferentes domínios de suporte. Domínios circulares podem ser calculados por:

$$r_S(\mathbf{x}) = \alpha_S d_m(\mathbf{x}) \tag{3.2}$$

onde $r_S(\mathbf{x})$ é o raio de suporte do ponto $\mathbf{x}$ a ser determinado, α_S é o fator de escalonamento para domínios de suporte e $d_m(\mathbf{x})$ é o espaçamento nodal médio na vizinhança de $\mathbf{x}$. Para domínios de suporte retangulares, utiliza-se a definição

$$r_{kS}(\mathbf{x}) = \alpha_S d_{km}(\mathbf{x}) \tag{3.3}$$

onde $r_{kS}(\mathbf{x})$ é o raio de suporte do ponto $\mathbf{x}$ a ser determinado na direção k e $d_{km}(\mathbf{x})$ o espaçamento nodal médio na direção k, na vizinhança de $\mathbf{x}$. Em duas dimensões, por exemplo, $k = x, y$, $r_{xS}(\mathbf{x})$ e $r_{yS}(\mathbf{x})$ são os raios de suporte de $\mathbf{x}$ nas direções x e y,

respectivamente, e $d_{xm}(\mathbf{x})$ e $d_{ym}(\mathbf{x})$ os espaçamentos nodais médios nas direções horizontal e vertical em torno de $\mathbf{x}$.

O conceito de domínio de suporte funciona bem para uma distribuição regular de nós ou quando a densidade nodal não varia muito ao longo do domínio. Todavia, em problemas reais, frequentemente aparecem singularidades nas funções a serem aproximadas, com isso a densidade de nós sofre mudanças abruptas. Em tais casos, podem ocorrer erros de aproximação devido à escolha ruim dos nós de suporte, por exemplo, quando estes se localizam apenas em um lado do domínio de suporte. A fim de se evitar problemas desse tipo, o conceito de *domínio de influência* deve ser usado.

Domínio de influência é definido como o domínio sobre o qual um nó exerce influência [33]. Observe que domínio de influência refere-se a um nó e está ligado à região local, enquanto domínio de suporte refere-se a um ponto e está ligado a nós locais. O domínio de suporte pode ser determinado através dos domínios de influência e é uma forma alternativa de selecionar nós de suporte. Essa estratégia funciona bem para problemas nos quais a distribuição dos nós é irregular ou quando a densidade nodal varia muito no domínio.

Os domínios de influência são definidos para todos os nós do problema e podem ter diferentes formatos e tamanhos de um nó para outro, sendo que a união de todos os domínios de influência deve cobrir completamente o domínio do problema de modo a permitir que aproximações sejam construídas em qualquer ponto pertencente ao domínio. Considere como exemplo a Figura 3.1c. O nó 1 possui raio de influência r_1, o nó 2 r_2, e assim por diante. Para determinar os nós usados no cálculo das funções de forma em um ponto $\mathbf{x}_Q$, verifica-se quais nós possuem domínio de influência que cubra o ponto $\mathbf{x}_Q$. 1 e 2 serão nós de suporte para $\mathbf{x}_Q$, mas 3 não.

Assim como os domínios de suporte, os domínios de influência são determinados de acordo com a densidade nodal local. Um nó localizado em uma região de maior densidade normalmente tem um domínio de influência menor que um localizado em uma região com menor concentração de nós. Domínios de influência circulares podem ser calculados através de

$$r_I^i = \alpha_I d_m^i \tag{3.4}$$

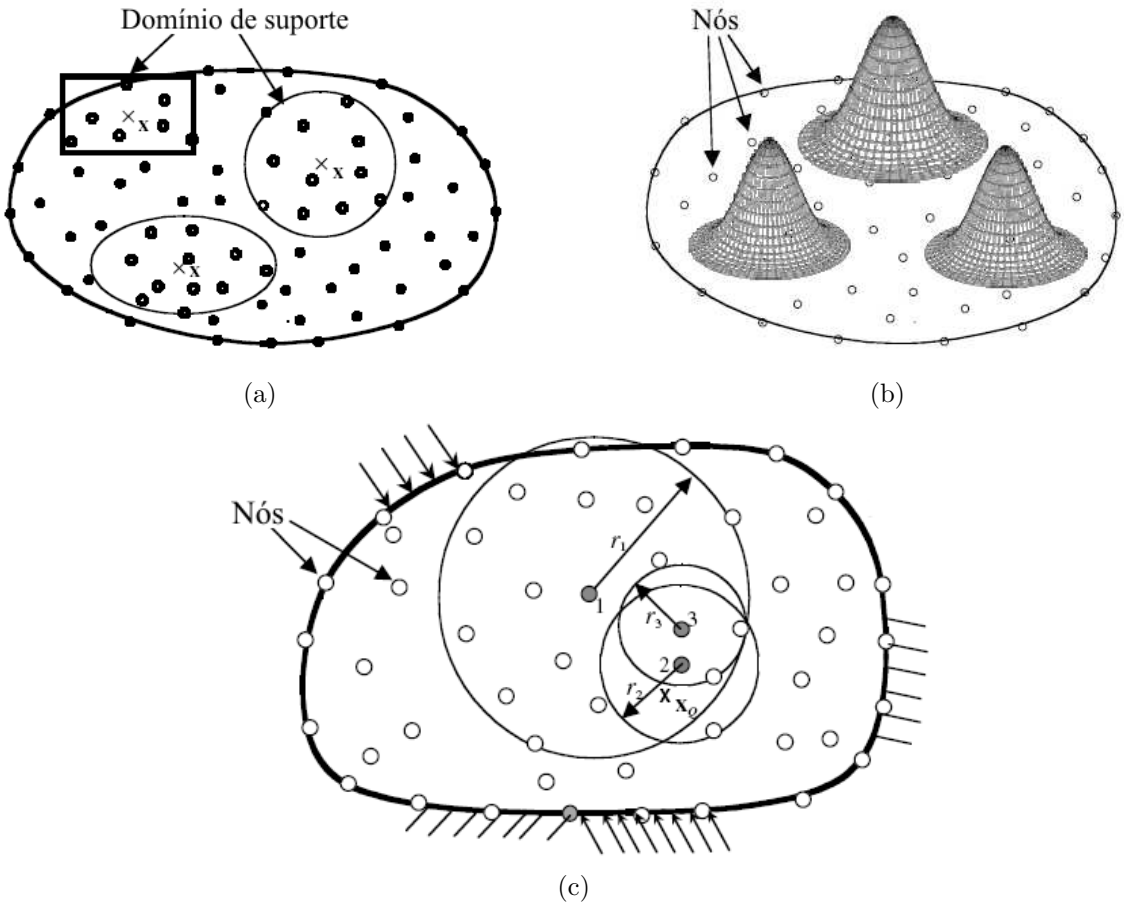


Figura 3.1: Domínio de suporte e domínio de influência. (a) Domínio de suporte no ponto $\mathbf{x}$ - determina os nós que são utilizados para a aproximação em $\mathbf{x}$. O domínio de suporte pode ter diferentes formatos e tamanhos de um ponto para outro. Os formatos mais usados são circular ou retangular. (b) Domínios de suporte ponderados. (c) Domínios de influência dos nós 1, 2 e 3. O domínio de suporte do ponto $\mathbf{x}_Q$ é composto pelos nós cujos domínios de influência cobrem $\mathbf{x}_Q$ - os nós 1 e 2 cobrem e o nó 3 não. Figura retirada de [32].

onde r_I^i é o raio de influência do nó i a ser determinado, α_I é o fator de escalonamento para domínios de influência e d_m^i é o espaçamento nodal médio na vizinhança de i. Para domínios de influência retangulares, utiliza-se a definição

$$r_{kI}^i = \alpha_I d_{km}^i \tag{3.5}$$

onde r_{kI}^i é o raio de influência do nó i a ser determinado na direção k e d_{km}^i o espaçamento nodal médio na direção k, na vizinhança de i. Em duas dimensões, por exemplo, $k = x, y$, r_{xI}^i e r_{yI}^i são os raios de influência de i nas direções x e y, respectivamente, e d_{xm}^i e d_{ym}^i os espaçamentos nodais médios nas direções horizontal e vertical.

3.2 Funções de Forma

Como visto na Equação 3.1, *funções de forma* são funções especiais utilizadas para aproximar uma determinada função através de uma combinação linear. A criação dessas funções para métodos sem malha é um dos seus pontos mais centrais e importantes. O desenvolvimento de métodos para construir funções de forma tem sido uma das áreas de maior pesquisa dentro de métodos sem malha. O desafio é como criá-las, de forma eficiente e de boa qualidade, usando nós distribuídos arbitrariamente no domínio do problema [33]. Um bom método de construção de funções de forma deve satisfazer certas propriedades ou requisitos que são listados a seguir.

1. **Distribuição Nodal Arbitrária**
   A distribuição nodal deve ser arbitrária com razão e, mais flexível que no método dos elementos finitos. Por uma distribuição "arbitrária com racionalidade" quer-se dizer que a distribuição de nós pode ser irregular desde que seja possível construir funções de forma. Se uma distribuição nodal arbitrária sem sentido é utilizada não é possível construir as funções de forma.

2. **Estabilidade**
   O algoritmo deve ser estável com respeito à irregularidade da distribuição nodal, isto é, deve funcionar para distribuições arbitrárias (regulares ou irregulares) de nós.

3. **Suporte Compacto**
   O domínio de suporte deve ser pequeno (compacto) de modo a incluir apenas um

pequeno número de nós localizados na vizinhança do ponto onde é realizada a aproximação. O suporte compacto faz com que o sistema de equações seja esparso, o que aumenta a eficiência computacional em termos de armazenamento e tempo de processamento [33].

4. **Eficiência**
   Como as funções de forma são construídas durante o processamento, o algoritmo deve ser computacionalmente eficiente. A construção das funções de forma pode se tornar inviável na prática caso o procedimento seja muito custoso, independentemente de as funções geradas serem de boa qualidade.

5. **Consistência**
   As funções de forma construídas devem possuir uma certa ordem de consistência, isto é, devem ser capazes de reproduzir localmente de forma exata os polinômios daquela ordem. Para um método sem malha convergir, suas funções de forma devem satisfazer um certo grau de consistência. O grau de consistência é medido como o grau do polinômio de mais alta ordem que a aproximação é capaz de reproduzir de forma exata [22]. Sendo capaz de reproduzir exatamente uma função constante, diz-se que a aproximação tem consistência de ordem zero ou C^0. Caso reproduza exatamente uma função linear, tem consistência de ordem um ou C^1.

6. **Partição da Unidade**
   As funções de forma devem satistazer o critério da partição da unidade (PU):

$$\sum_{i \in S_n} \phi_i(\mathbf{x}) = 1 \tag{3.6}$$

   Este critério indica que as funções de forma possuem consistência C^0, isto é, são capazes de reproduzir uma função constante (nesse caso é igual a um). Não sendo atendido o critério, as funções de forma não serão capazes de reproduzir um termo constante da solução.

7. **Propriedade do Delta de Kronecker**
   A propriedade delta de Kronecker é dada por

$$\phi_i(\mathbf{x}_j) = \begin{cases} 1 & \text{se} \quad i = j \\ 0 & \text{se} \quad i \neq j \end{cases} \tag{3.7}$$

   É desejável que as funções de forma satisfaçam a propriedade delta de Kronecker, permitindo que as condições de contorno essenciais sejam tratadas de forma fácil.

Caso contrário, procedimentos adicionais devem ser usados, como multiplicadores de Lagrange e método das penalidades, implicando o aumento do custo computacional. Funções de forma que satisfazem o critério do delta de Kronecker são *interpolantes*, isto é, a aproximação gerada passa exatamente sobre os pontos onde a função a ser aproximada é conhecida.

8. **Compatibilidade**
   Compatibilidade está associada à continuidade da função de aproximação ao longo do domínio do problema [33]. É desejável que as funções de forma sejam compatíveis em todo o domínio. Senão técnicas para tratamento das incompatibilidades nas regiões em que existem podem ser usadas, mas aumentam o custo computacional.

9. **Independência Linear**
   As funções de forma construídas para todos os nós do problema devem ser linearmente independentes de modo que formem uma base para a construção das funções de aproximação, gerando, assim, um sistema de equações estável. Funções de forma linearmente independentes podem ser obtidas (1) escolhendo funções de base adequadas e (2) determinando de forma correta os domínios de suporte em todo domínio [33].

Existem várias maneiras de construir funções de forma e cada uma gera aproximações que satisfazem um certo subconjunto das propriedades listadas anteriormente. De fato, não há um procedimento que gere aproximações que reúnam todas essas propriedades ao mesmo tempo. Tem-se, portanto, um "*trade off*" entre utilizar um procedimento ou outro, e a escolha é baseada no problema ou no método sem malha adotado. A seguir, apresentam-se os métodos de construção de funções de forma mais utilizados e que estão disponíveis no *framework*.

3.2.1 Moving Least Squares

O *Moving Least Squares* (MLS), ou método dos mínimos quadrados móveis, é um método originalmente desenvolvido por matemáticos para ajuste de superfícies e regressão de dados [30]. O MLS é um dos métodos mais utilizados atualmente para construção de funções de forma de métodos sem malha. A duas maiores características que o tornam

popular são: (1) a função de aproximação é contínua e suave em todo o domínio quando um número suficiente de nós é usado; e (2) o método é capaz de gerar uma aproximação com a ordem de consistência que se desejar [33].

Seja $u(\mathbf{x})$ uma função definida em um domínio Ω. A aproximação de $u(\mathbf{x})$ em um ponto $\mathbf{x}$ é denotada por $u^h(\mathbf{x})$. O MLS aproxima a função da seguinte forma:

$$u^h(\mathbf{x}) = \sum_j^m p_j(\mathbf{x}) a_j(\mathbf{x}) = \mathbf{p}^T(\mathbf{x})\mathbf{a}(\mathbf{x}) \tag{3.8}$$

onde m é o número de monômios da base polinomial, $\mathbf{a}(\mathbf{x})$ é um vetor de coeficientes (que dependem de $\mathbf{x}$) dado por

$$\mathbf{a}(\mathbf{x}) = [a_0(\mathbf{x}),\ a_1(\mathbf{x}),\ \cdots,\ a_m(\mathbf{x})]^T \tag{3.9}$$

e $\mathbf{p}(\mathbf{x})$ é um vetor de funções de base que contém, normalmente, monômios de baixa ordem para garantir completeza da aproximação. Funções enriquecidas podem ser inseridas na base para melhorar a eficiência ou acrescentar características especiais na aproximação. Em 1D, uma base polinomial completa de ordem m dada por

$$\mathbf{p}(\mathbf{x}) = \mathbf{p}(x) = [p_0(x),\ p_1(x),\ \cdots,\ p_m(x)]^T = [1,\ x,\ x^2,\ \cdots,\ x^m]^T \tag{3.10}$$

Em 2D e 3D, as bases polinomiais são construídas a partir do triângulo e da pirâmide de Pascal, respectivamente (ver [33]). Bases completas de ordem m para os dois espaços são mostradas abaixo.

$$\mathbf{p}(\mathbf{x}) = \mathbf{p}(x, y) = [1,\ x,\ y,\ xy,\ x^2,\ y^2,\ \cdots,\ x^m,\ y^m]^T \tag{3.11}$$

e

$$\mathbf{p}(\mathbf{x}) = \mathbf{p}(x, y, z) = [1,\ x,\ y,\ z,\ xy,\ yz,\ zx,\ x^2,\ y^2,\ z^2,\ \cdots,\ x^m,\ y^m,\ z^m]^T \tag{3.12}$$

O vetor de coeficientes $\mathbf{a}(\mathbf{x})$ é determinado usando os valores de $\mathbf{u}(\mathbf{x})$ em um conjunto de nós $(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n)$ no domínio de suporte de $\mathbf{x}$. Um funcional de resíduos quadráticos ponderado é construído:

$$\begin{aligned} J &= \sum_{I=1}^n W(\mathbf{x} - \mathbf{x}_I)[u^h(\mathbf{x}, \mathbf{x}_I) - u(\mathbf{x}_I)]^2 \\ &= \sum_{I=1}^n W_I(\mathbf{x})[\mathbf{p}^T(\mathbf{x}_I)\mathbf{a}(\mathbf{x}) - u_I]^2 \end{aligned} \tag{3.13}$$

onde W_I é uma função de peso para o nó I centrada em $\mathbf{x}$, e u_I é o valor da função u em $\mathbf{x}_I$, chamado parâmetro nodal. Normalmente, a função de peso W_I é uma função de base radial, como as mostradas no Apêndice B. Os coeficientes de $\mathbf{a}(\mathbf{x})$ são determinados minimizando o funcional J, isto é, fazendo

$$\frac{\partial J}{\partial \mathbf{a}} = 0 \tag{3.14}$$

que resulta no sistema de equações lineares:

$$\mathbf{A}(\mathbf{x})\mathbf{a}(\mathbf{x}) = \mathbf{B}(\mathbf{x})\mathbf{U}_S \tag{3.15}$$

ou

$$\mathbf{a}(\mathbf{x}) = \mathbf{A}^{-1}(\mathbf{x})\mathbf{B}(\mathbf{x})\mathbf{U}_S \tag{3.16}$$

onde

$$\mathbf{A}(\mathbf{x}) = \sum_{I=1}^{n} W_I(\mathbf{x})\mathbf{p}(\mathbf{x}_I)\mathbf{p}^T(\mathbf{x}_I) \tag{3.17}$$

$$\mathbf{B}(\mathbf{x}) = [\mathbf{B}_1,\ \mathbf{B}_2,\ \cdots,\ \mathbf{B}_n] \tag{3.18}$$

$$\mathbf{B}_I = W_I(\mathbf{x})\mathbf{p}(\mathbf{x}_I) \tag{3.19}$$

$$\mathbf{U}_S = [u_1,\ u_2,\ \cdots,\ u_n]^T \tag{3.20}$$

em que a matriz $\mathbf{A}$ é chamada *matriz de momentos* e $\mathbf{U}_S$ é o vetor contendo os parâmetros nodais dos nós do domínio de suporte. Note que a Equação 3.16 existirá somente se a inversa $\mathbf{A}^{-1}$ existir. Isto é alcançado fazendo com que o número de nós no domínio de suporte seja muito maior que o número de termos da base polinomial, isto é, $n \gg m$.

Substituindo a Equação 3.16 em 3.8, chega-se à aproximação por MLS:

$$u^h(\mathbf{x}) = \sum_{I=1}^{n} \phi_I(\mathbf{x})u_I = \mathbf{\Phi}(\mathbf{x})\mathbf{U}_S \tag{3.21}$$

com

$$\mathbf{\Phi}(\mathbf{x}) = [\phi_1(\mathbf{x}),\ \phi_2(\mathbf{x}),\ \cdots,\ \phi_n(\mathbf{x})] \tag{3.22}$$

$$\phi_I(\mathbf{x}) = \mathbf{p}^T\mathbf{A}^{-1}\mathbf{B}_I \tag{3.23}$$

onde $\mathbf{\Phi}(\mathbf{x})$ é o vetor com as funções de forma dos nós contidos no domínio de suporte de $\mathbf{x}$. Para obter as derivadas das funções de forma, primeiramente escreve-se

$$\boldsymbol{\gamma} = \mathbf{p}^T(\mathbf{x})\mathbf{A}^{-1} \tag{3.24}$$

cujas derivadas parciais em relação a k-ésima dimensão são

$$\boldsymbol{\gamma}_{,k} = \mathbf{p}_{,k}^T(\mathbf{x})\mathbf{A}^{-1} + \mathbf{p}^T(\mathbf{x})\mathbf{A}_{,k}^{-1} \tag{3.25}$$

onde $\mathbf{A}_{,k}^{-1}$ é dada por

$$\mathbf{A}_{,k}^{-1} = -\mathbf{A}^{-1}\mathbf{A}_{,k}\mathbf{A}^{-1} \tag{3.26}$$

Finalmente, obtém-se as derivadas parciais de $\boldsymbol{\Phi}$ em relação à k-ésima dimensão através de

$$\boldsymbol{\Phi}_{,k} = \boldsymbol{\gamma}_{,k}\mathbf{B} + \boldsymbol{\gamma}\mathbf{B}_{,k} \tag{3.27}$$

onde

$$\mathbf{B}_{,k} = [\mathbf{B}_{1,k},\ \mathbf{B}_{2,k},\ \cdots,\ \mathbf{B}_{n,k}] \tag{3.28}$$

$$\mathbf{B}_{I,k} = W_{I,k}(\mathbf{x})\mathbf{p}(\mathbf{x}_I) \tag{3.29}$$

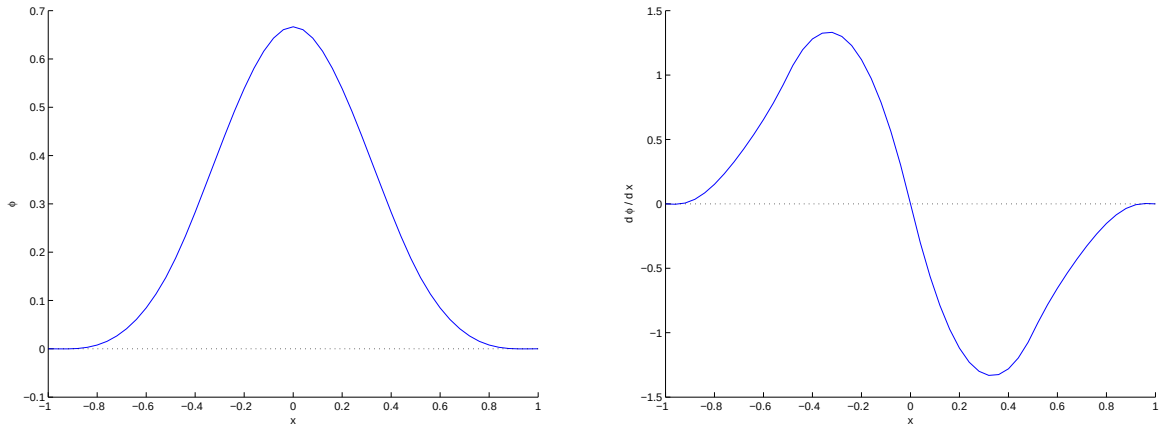
Vale observar que as funções de forma geradas através do MLS não satisfazem o critério do delta de Kronecker $\phi_I(\mathbf{x}_J) \neq \delta_{IJ}$, o que resulta em $u^h(\mathbf{x}_I) \neq u_I$, isto é, os parâmetros nodais u_I não são iguais aos valores da função aproximada $u^h(\mathbf{x}_I)$ nos nós [33]. Portanto, as funções do MLS não são interpolantes e métodos para impor as condições de contorno de Dirichlet devem ser utilizados. Observe, também, que a função de peso W desempenha dois papéis importantes para o MLS: (1) fornecer pesos diferentes para os resíduos nos nós do domínio de suporte (nós mais próximos de $\mathbf{x}$ recebem maior peso que os distantes); e (2) fazer com que os nós entrem e saiam do domínio de suporte de maneira gradual. Por causa deste segundo papel de W, a aproximação gerada pelo MLS é contínua em todo o domínio e, consequentemente, satisfaz o critério de compatibilidade.

Uma característica interessante do MLS é que a consistência da aproximação depende da ordem do polinômio completo utilizado na base $\mathbf{p}(\mathbf{x})$. Normalmente, polinômios de primeira ordem são usados e a consistência C^1 é garantida. Por esta razão, o critério de partição da unidade também é atendido. A prova da consistência para as funções geradas pelo MLS pode ser vista em [33].

A Figura 3.2 mostra a função de forma gerada pelo MLS e sua derivada em 1D. A função de forma em $x = 0$ foi obtida usando cinco nós igualmente distribuídos no domínio $[-1, 1]$, e a *Spline* cúbica dada pela Equação B.4 como função de peso, com raio de suporte igual

a 2. Note que o valor de ϕ em $x = 0$ é menor que 1, e a função não satisfaz o delta de Kronecker. Além disso, a dimensão do domínio de suporte da aproximação por MLS é determinada pelo raio r da função de peso W.

A Figura 3.3 mostra a função de forma e suas derivadas em 2D para o nó $(0, 0)$. Foram usados 5×5 nós igualmente distribuídos no domínio $[-2, 2] \times [-2, 2]$.



(a) (b)

Figura 3.2: Funções de forma MLS em 1D para o nó $x = 0$ usando 5 nós igualmente distribuídos em $x \in [-1, 1]$ e polinômios de primeira ordem na base. *Spline* cúbica foi usada como função de peso, com $r = 2$. (a) Mostra $\phi(x)$ e (b) a derivada de $\phi(x)$ em relação a x. Note que as funções de forma construídas com o MLS não satisfazem a propriedade do delta de Kronecker.

3.2.2 Funções de Shepard

As *funções de Shepard* foram propostas em 1968 [22] e são um caso particular do Moving Least Squares, com a presença de polinômios de grau zero na base. Tais funções possuem baixo custo computacional para construção, em detrimento da consistência da aproximação gerada: têm consistência C^0 visto que somente polinômios de grau zero são utilizados na base.

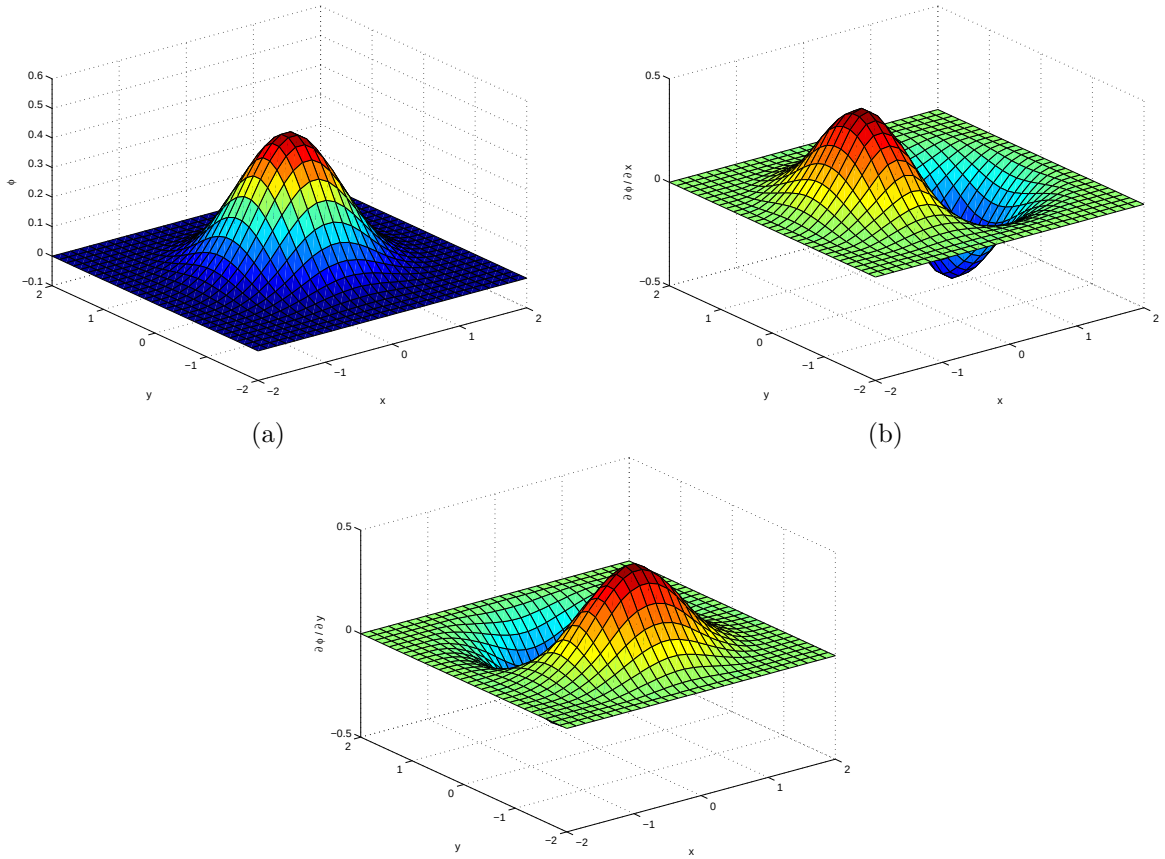


Figura 3.3: Funções de forma MLS em 2D para o nó $\mathbf{x} = (0, 0)$ usando 5×5 nós igualmente distribuídos em $(x, y) \in [-2, 2] \times [-2, 2]$ e polinômios de primeira ordem na base. *Spline* cúbica foi usada como função de peso, com $r = 2$. (a) Mostra $\phi(x, y)$, (b) a derivada parcial de $\phi(x, y)$ em relação a x e (c) a derivada parcial de $\phi(x, y)$ em relação a y.

As funções de Shepard são definidas da mesma forma que o MLS, mas fazendo $\mathbf{p}^T = [1]$. Com isso, as matrizes $\mathbf{A}(\mathbf{x})$ e $\mathbf{B}(\mathbf{x})$ definidas na Equação 3.15 se reduzem a

$$\mathbf{A}(\mathbf{x}) = \sum_{I=1}^{n} W_I(\mathbf{x}) \tag{3.30}$$

$$\mathbf{B}(\mathbf{x}) = [W_1(\mathbf{x}),\ W_2(\mathbf{x}),\ \cdots,\ W_n(\mathbf{x})] \tag{3.31}$$

Dessa forma, a aproximação utilizando funções de Shepard fica:

$$u^h(\mathbf{x}) = \sum_{I=1}^{n} \phi_I(\mathbf{x}) u_I \tag{3.32}$$

com

$$\phi_I(\mathbf{x}) = \frac{W_I(\mathbf{x})}{\sum_{I=1}^{n} W_I(\mathbf{x})} \tag{3.33}$$

As derivadas das funções de forma são obtidas de forma direta da equação acima:

$$\phi_{I,k}(\mathbf{x}) = \frac{W_{I,k}(\mathbf{x}) \sum_{I=1}^{n} W_I(\mathbf{x}) - W_I(\mathbf{x}) \sum_{I=1}^{n} W_{I,k}(\mathbf{x})}{\left[\sum_{I=1}^{n} W_I(\mathbf{x})\right]^2} \tag{3.34}$$

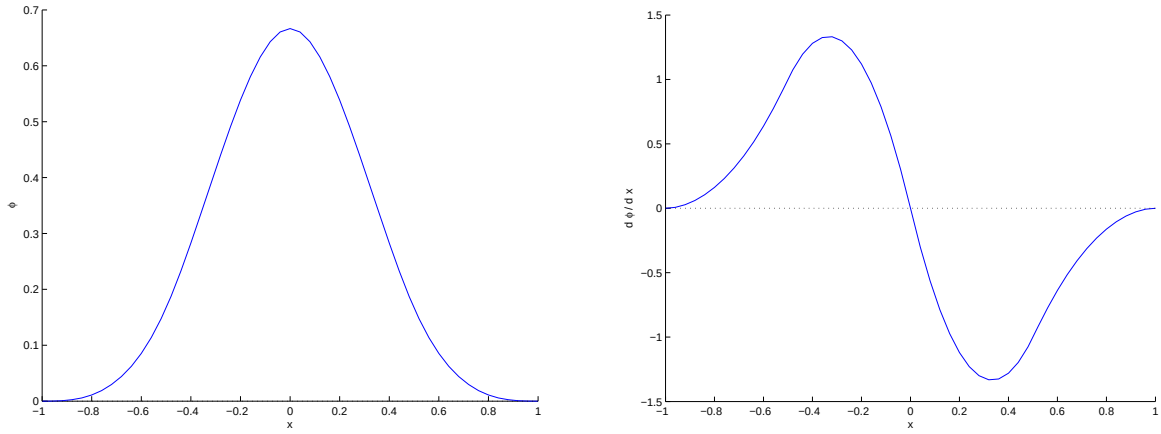
As funções de Shepard possuem as mesmas características das funções geradas pelo MLS: não satisfazem o delta de Kronecker, são consistentes (ordem zero de consistência), satisfazem a partição da unidade (pois possuem consistência C^0) e são contínuas no domínio, isto é, são compatíveis.

A Figura 3.4 mostra a função de forma gerada pelo método de Shepard e sua derivada em 1D. A função de forma em $x = 0$ foi obtida usando cinco nós igualmente distribuídos no domínio $[-1, 1]$, e a *Spline* cúbica dada pela Equação B.4 como função de peso, com raio de suporte igual a 2. Assim como no MLS, as funções não satisfazem o delta de Kronecker e a dimensão do domínio de suporte é determinada pelo raio da função de peso W.

A Figura 3.5 mostra a função de forma e suas derivadas em 2D no nó $(0, 0)$. Foram usados 5×5 nós igualmente distribuídos no domínio $[-2, 2] \times [-2, 2]$.

3.2.3 Point Interpolation Method

O *Point Interpolation Method* (PIM), ou método de interpolação de pontos, foi originalmente desenvolvido em [34] e, como o nome sugere, obtém uma aproximação fazendo com que a função de interpolação passe pelos valores da função em cada nó localizado no



(a) (b)

Figura 3.4: Funções de forma Shepard em 1D para o nó $x = 0$ usando 5 nós igualmente distribuídos em $x \in [-1, 1]$. *Spline* cúbica foi usada como função de peso, com $r = 2$. (a) Mostra $\phi(x)$ e (b) a derivada de $\phi(x)$ em relação a x. Note que as funções de forma construídas com o método de Shepard não satisfazem a propriedade do delta de Kronecker.

domínio de suporte. Seja uma função $u(\mathbf{x})$ definida no domínio Ω. A aproximação $u^h(\mathbf{x})$ em um ponto $\mathbf{x}$ na região de suporte de um ponto $\mathbf{x}_Q$ é dada por:

$$u^h(\mathbf{x}, \mathbf{x}_Q) = \sum_{i=1}^{n} B_i(\mathbf{x}) a_i(\mathbf{x}_Q) \tag{3.35}$$

onde $B_i(\mathbf{x})$ são as funções de base, n é o número de nós no domínio de suporte de $\mathbf{x}_Q$, e $a_i(\mathbf{x}_Q)$ é o coeficiente para a função de base $B_i(\mathbf{x})$, para um dado ponto $\mathbf{x}_Q$.

Inicialmente, funções polinomiais foram usadas como funções de base:

$$u^h(\mathbf{x}) = \sum_{i=1}^{n} p_i(\mathbf{x}) a_i(\mathbf{x}_Q) = \mathbf{p}^T(\mathbf{x}) \mathbf{a}(\mathbf{x}_Q) \tag{3.36}$$

onde $\mathbf{p}^T$ tem a mesma definição que nas Equações 3.10, 3.11 e 3.12, e $\mathbf{a}(\mathbf{x}_Q)$ é um vetor da forma

$$\mathbf{a}(\mathbf{x}_Q) = [a_1,\ a_2,\ \cdots,\ a_n]^T \tag{3.37}$$

Note que os coeficientes a_i são constantes na vizinhança do ponto de interesse x_Q, e são atualizados somente se o domínio de suporte associado a x_Q sofrer mudança. Por isso, em um domínio de suporte, a aproximação gerada pelo PIM é consistente, admitindo que não haja nós duplicados no domínio. A ordem de consistência da aproximação depende

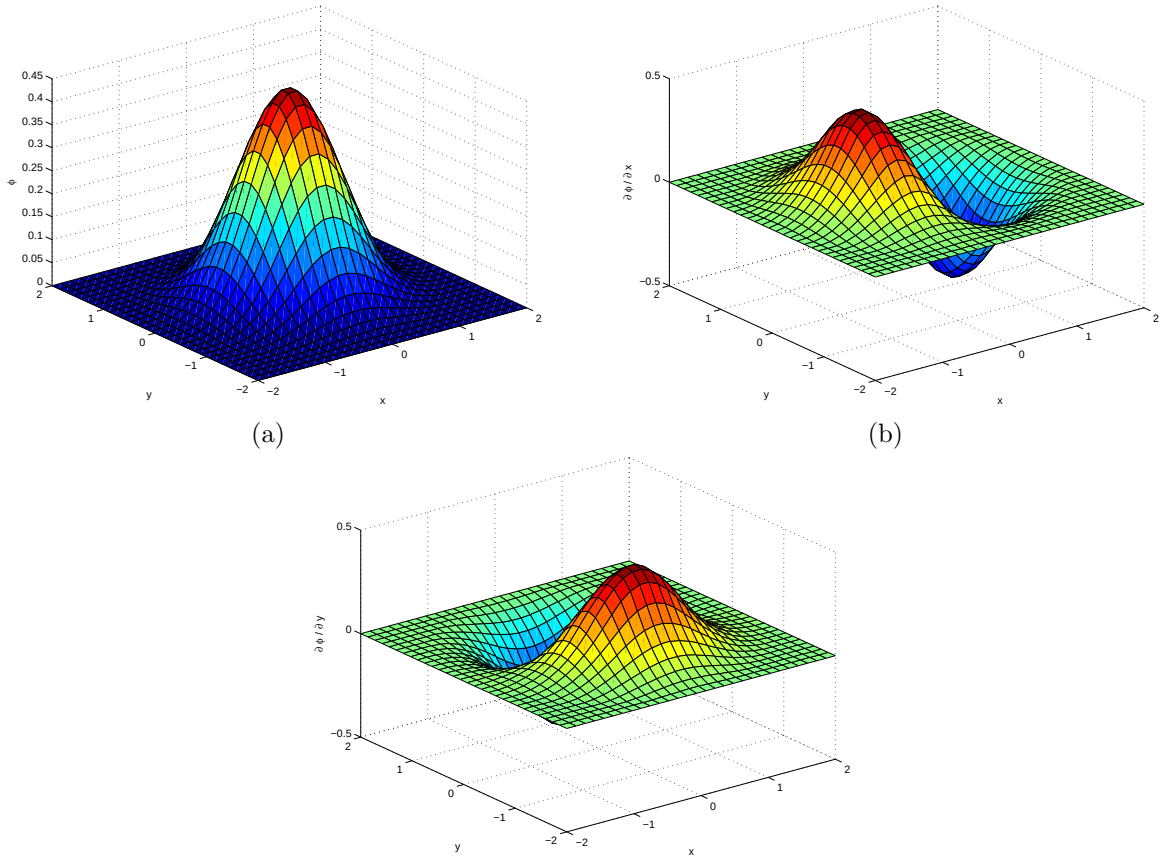


Figura 3.5: Funções de forma Shepard em 2D para o nó $\mathbf{x} = (0, 0)$ usando 5×5 nós igualmente distribuídos em $(x, y) \in [-2, 2] \times [-2, 2]$. *Spline* cúbica foi usada como função de peso, com $r = 2$. (a) Mostra $\phi(x, y)$, (b) a derivada parcial de $\phi(x, y)$ em relação a x e (c) a derivada parcial de $\phi(x, y)$ em relação a y.

da base polinomial usada. Os coeficientes a_i são calculado forçando que a Equação 3.36 seja satisfeita nos n nós de suporte. Para cada nó i, tem-se que

$$u_i = \mathbf{p}^T(\mathbf{x}_i)\mathbf{a}(\mathbf{x}_Q), \quad i = 1, 2, \cdots, n \tag{3.38}$$

onde u_i é o valor da função no nó i. A Equação 3.38 pode ser reescrita na forma matricial:

$$\mathbf{U}_S = \mathbf{P}_Q\mathbf{a}(\mathbf{x}_Q) \tag{3.39}$$

em que $\mathbf{U}_S$ é um vetor contendo todos os valores u_i da função nos n nós de suporte:

$$\mathbf{U}_S = [u_1,\ u_2,\ \cdots,\ u_n]^T \tag{3.40}$$

e a matriz $\mathbf{P}_q$ é chamada *matriz de momentos*, dada por

$$\mathbf{P}_Q = \begin{bmatrix} \mathbf{p}^T(\mathbf{x}_1) \\ \mathbf{p}^T(\mathbf{x}_2) \\ \vdots \\ \mathbf{p}^T(\mathbf{x}_n) \end{bmatrix} \tag{3.41}$$

Assumindo que a inversa de $\mathbf{P}_Q$ exista, da Equação 3.39 vem

$$\mathbf{a}(\mathbf{x}_Q) = \mathbf{P}_Q^{-1}\mathbf{U}_S \tag{3.42}$$

Substituindo a Equação 3.42 na 3.36:

$$u^h(\mathbf{x}) = \mathbf{\Phi}(\mathbf{x})\mathbf{U}_S = \mathbf{p}^T(\mathbf{x})\mathbf{P}_Q^{-1}\mathbf{U}_S \tag{3.43}$$

Logo, as funções de forma são dadas por

$$\mathbf{\Phi}(\mathbf{x}) = \mathbf{p}^T(\mathbf{x})\mathbf{P}_Q^{-1} = [\phi_1(\mathbf{x}),\ \phi_2(\mathbf{x}),\ \cdots,\ \phi_n(\mathbf{x})] \tag{3.44}$$

As derivadas das funções de forma são obtidas da Equação 3.44 de forma fácil, pois todas as funções envolvidas são polinômios. A derivada de k-ésima dimensão $\mathbf{\Phi}_{,k}$ é dada por

$$\mathbf{\Phi}_{,k}(\mathbf{x}) = \mathbf{p}_{,k}^T(\mathbf{x})\mathbf{P}_Q^{-1} \tag{3.45}$$

onde $\mathbf{p}_{,k}^T(\mathbf{x})$ é a k-ésima derivada de $\mathbf{p}^T(\mathbf{x})$.

As funções de forma geradas pelo PIM possuem a propriedade do delta de Kronecker pela própria definição, que força a função aproximada a passar pelos valores funcionais nos nós do domínio de suporte. Por isso, elas são interpolantes e a imposição das condições de contorno essenciais é feita de forma direta nos métodos sem malha. As funções de forma do PIM não são compatíveis, pois nenhuma função peso é usada, e sofrem saltos de um ponto para outro quando há mudanças no domínio de suporte. Tal característica diminui a convergência dos métodos sem malha e é desejável que procedimentos para tratar as incompatibilidades ao longo do domínio sejam aplicados.

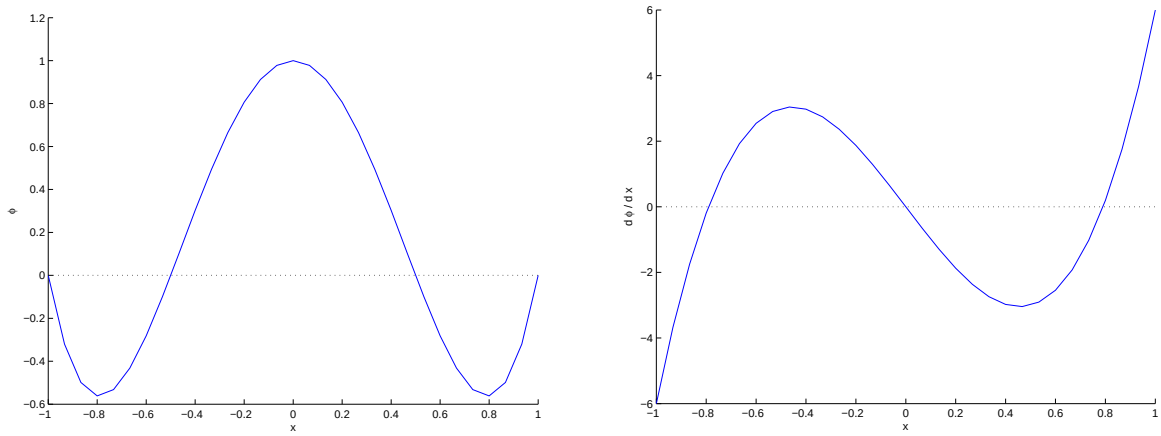
Além disso, as funções do PIM são consistentes de acordo com a base polinomial usada: se um polinômio completo de ordem k é usado na base, então as funções de forma têm consistência C^k [33]. Da Equação 3.36, observa-se que o número de monômios na base é igual ao número de nós contidos no domínio de suporte, por isso os nós de suporte devem ser escolhidos de tal maneira que se garanta a consistência desejada. O PIM satisfaz o critério de suporte compacto desde que as funções de forma sejam construídas usando nós em uma vizinhança compacta. A partição da unidade é respeitada se, na base polinomial, os termos de ordem zero forem incluídos.

Deve-se notar que é possível a matriz de momentos $\mathbf{P}_Q$ ser singular dependendo da configuração dos nós dentro do domínio de suporte, o que impede que as funções de forma sejam construídas pelo PIM. Técnicas para evitar a singularidade da matriz de momentos foram desenvolvidas, e algumas delas são: aplicar uma pequena perturbação aleatória nos nós de suporte; utilizar o algoritmo de triangularização de matriz em $\mathbf{P}_Q$ [35]; e utilizar funções de base radial como funções de base. Esta última estratégia é a mais interessante e leva ao surgimento de mais duas novas vertentes de geração de funções de forma, variantes do PIM, apresentadas nas seções seguintes.

A Figura 3.6 mostra a função de forma gerada pelo PIM e sua derivada em 1D. A função de forma em $x = 0$ foi obtida usando cinco nós igualmente distribuídos no domínio $[-1, 1]$. Note que a função satisfaz o delta de Kronecker, isto é, $\phi(0.0) = 1$ e $\phi(-1.0) = \phi(-0.5) = \phi(0.5) = \phi(0.5) = 0$.

3.2.4 Radial Point Interpolation Method

Como visto na seção 3.2.3, as funções de forma geradas através do PIM utilizando uma base polinomial possuem características interessantes como consistência e, principalmente, satisfação do delta de Knronecker, o que facilita o tratamento da imposição das condições de contorno de Dirichlet. A grande desvantagem do PIM polinomial é que problemas de singularidade da matriz de momentos podem ocorrer, comprometendo, nesse caso, o processo de construção das funções de forma. Para criar uma matriz de momentos não singular, introduz-se funções de base radial (apresentadas no Apêndice B) na formulação



(a) (b)

Figura 3.6: Funções de forma PIM em 1D para o nó $x = 0$ usando 5 nós igualmente distribuídos em $x \in [-1, 1]$. (a) Mostra $\phi(x)$ e (b) a derivada de $\phi(x)$ em relação a x. Note que as funções de forma construídas com o PIM satisfazem a propriedade do delta de Kronecker.

do PIM, mais especificamente, nas funções de base. O PIM utilizando RBFs é denominado *Radial Point Interpolation Method* (RPIM), ou método de interpolação de pontos radial.

No RPIM, a aproximação $u^h(\mathbf{x})$ da Equação 3.35 é feita escolhendo RBFs como funções de base $B_i(\mathbf{x})$:

$$u^h(\mathbf{x}) = \sum_{i=1}^{n} R_i(\mathbf{x}) a_i(\mathbf{x}_Q) = \mathbf{R}^T(\mathbf{x})\mathbf{a}(\mathbf{x}_Q) \tag{3.46}$$

onde $R_i(\mathbf{x})$ é uma RBF centrada no nó i calculada no ponto $\mathbf{x}$, e $\mathbf{R}(\mathbf{x})$ é um vetor contendo todas as RBFs $R_i(\mathbf{x})$ relativas aos nós de suporte de $\mathbf{x}_Q$, dado por

$$\mathbf{R}(\mathbf{x}) = [R_1(\mathbf{x}),\ R_2(\mathbf{x}),\ \cdots,\ R_n(\mathbf{x})]^T \tag{3.47}$$

Os coeficientes a_i são calculados fazendo com que a Equação 3.46 seja satisfeita para todos os n nós do domínio de suporte. Para cada nó j, tem-se que

$$u_j = \mathbf{R}^T(\mathbf{x}_j)\mathbf{a}(\mathbf{x}_Q), \quad j = 1, 2, \cdots, n \tag{3.48}$$

onde u_j é o valor da função no nó j. Reescrevendo a Equação 3.48 na forma matricial, vem

$$\mathbf{U}_S = \mathbf{R}_Q \mathbf{a}(\mathbf{x}_Q) \tag{3.49}$$

onde $\mathbf{U}_S$ é um vetor como definido em 3.40, e $\mathbf{R}_Q$ é a *matriz de momentos* dada por:

$$\mathbf{R}_Q = \begin{bmatrix} R_1(\mathbf{x}_1) & R_2(\mathbf{x}_1) & \cdots & R_n(\mathbf{x}_1) \\ R_1(\mathbf{x}_2) & R_2(\mathbf{x}_2) & \cdots & R_n(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ R_1(\mathbf{x}_n) & R_2(\mathbf{x}_n) & \cdots & R_n(\mathbf{x}_n) \end{bmatrix} \tag{3.50}$$

Como $R_i(\mathbf{x}_j) = R_j(\mathbf{x}_i)$, a matriz de momentos $\mathbf{R}_Q$ é simétrica. Dessa maneira, $\mathbf{R}_Q$ é definida positiva e, portanto, tem inversa [33]. Da Equação 3.49, vem

$$\mathbf{a}(\mathbf{x}_Q) = \mathbf{R}_Q^{-1}\mathbf{U}_S \tag{3.51}$$

Substituindo a Equação 3.51 na 3.46, chega-se a

$$u^h(\mathbf{x}) = \mathbf{\Phi}(\mathbf{x})\mathbf{U}_S = \mathbf{R}^T(\mathbf{x})\mathbf{R}_Q^{-1}\mathbf{U}_S \tag{3.52}$$

Logo, as funções de forma são dadas por

$$\mathbf{\Phi}(\mathbf{x}) = \mathbf{R}^T(\mathbf{x})\mathbf{R}_Q^{-1} = [\phi_1(\mathbf{x}),\ \phi_2(\mathbf{x}),\ \cdots,\ \phi_n(\mathbf{x})] \tag{3.53}$$

As derivadas das funções de forma são obtidas da Equação 3.53, em que é preciso tomar as derivadas apenas de $\mathbf{R}^T$, pois $\mathbf{R}_Q^{-1}$ é constante para um dado ponto $\mathbf{x}_Q$. A k-ésima derivada $\mathbf{\Phi}_{,k}$ é dada por

$$\mathbf{\Phi}_{,k} = \mathbf{R}_{,k}^T(\mathbf{x})\mathbf{R}_Q^{-1} \tag{3.54}$$

onde $\mathbf{R}_{,k}^T(\mathbf{x})$ é a k-ésima derivada de $\mathbf{R}^T(\mathbf{x})$, sendo:

$$\mathbf{R}_{,k}(\mathbf{x}) = [R_{1,k}(\mathbf{x}),\ R_{2,k}(\mathbf{x}),\ \cdots,\ R_{n,k}(\mathbf{x})]^T \tag{3.55}$$

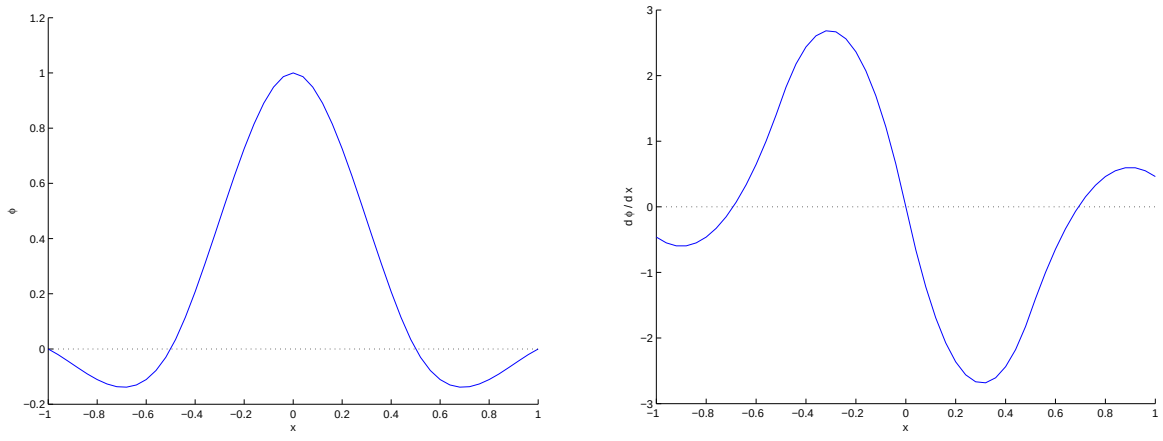
Da mesma forma que o PIM, o RPIM gera funções que satisfazem o critério do delta de Kronecker. A única diferença entre eles é o uso de funções de base radial ao invés de polinomiais. Como $\mathbf{R}_Q$ é sempre inversível [33], $\mathbf{R}_Q^{-1}$ existirá e essa é a maior vantagem de se usar RBFs ao invés de polinômios como funções de base. Entretanto, as funções de forma geradas pelo RPIM não são consistentes, isto é, não conseguem reproduzir de

maneira exata polinômios de nenhum grau. A razão por trás disso encontra-se no fato de não haver polinômios nas funções de base do RPIM, apenas RBFs são usadas, e estas não conseguem reproduzir polinômios. Apesar disso, o RPIM consegue gerar aproximações para polinômios com a precisão desejada quando se refinam os nós [33]. Note que como as funções de forma do RPIM não são consistentes, também não formam uma partição da unidade.

Uma forma de se alcançar consistência nas funções de forma do RPIM é adicionar, em sua base, termos polinomais às RBFs. Com isso, cria-se uma variação do RPIM, que é apresentada na próxima seção.

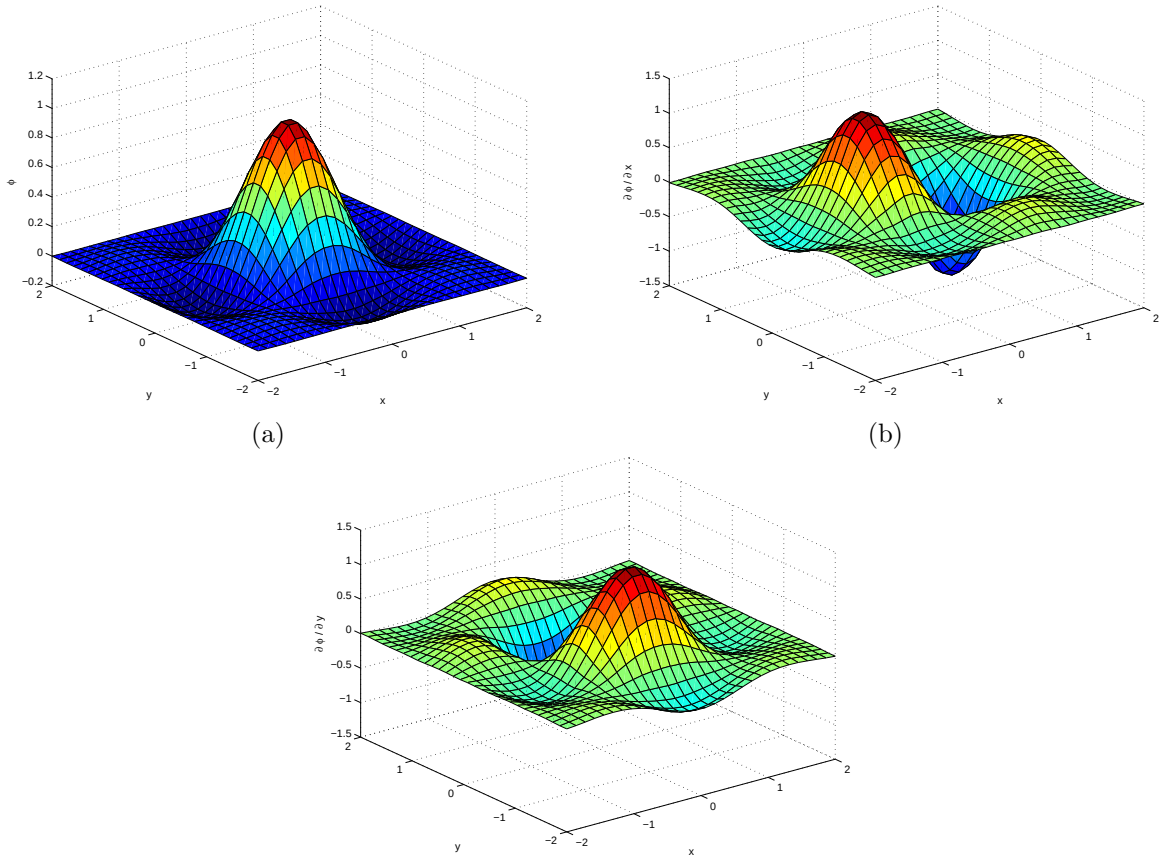
A Figura 3.7 mostra a função de forma gerada pelo RPIM e sua derivada em 1D. A função de forma em $x = 0$ foi obtida usando cinco nós igualmente distribuídos no domínio $[-1, 1]$, e a *Spline* cúbica dada pela Equação B.4 como RBF, com raio de suporte igual a 2. Assim como no PIM, as funções possuem a propriedade do delta de Kronecker.

A Figura 3.8 mostra a função de forma e suas derivadas em 2D no nó $(0, 0)$. Foram usados 5×5 nós igualmente distribuídos no domínio $[-2, 2] \times [-2, 2]$.



(a) (b)

Figura 3.7: Funções de forma RPIM em 1D para o nó $x = 0$ usando 5 nós igualmente distribuídos em $x \in [-1, 1]$. *Spline* cúbica foi usada como RBF, com $r = 2$. (a) Mostra $\phi(x)$ e (b) a derivada de $\phi(x)$ em relação a x. Note que as funções de forma construídas com o RPIM satisfazem a propriedade do delta de Kronecker.



(a) (b) (c)

Figura 3.8: Funções de forma RPIM em 2D para o nó $\mathbf{x} = (0, 0)$ usando 5×5 nós igualmente distribuídos em $(x, y) \in [-2, 2] \times [-2, 2]$. *Spline* cúbica foi usada como RBF, com $r = 2$. (a) Mostra $\phi(x, y)$, (b) a derivada parcial de $\phi(x, y)$ em relação a x e (c) a derivada parcial de $\phi(x, y)$ em relação a y.

3.2.5 Radial Point Interpolation Method with Polynomials

Na seção 3.2.4 viu-se que o RPIM não é consistente e falha na construção de aproximações exatas de funções lineares (na verdade, de qualquer função polinomial). A solução para essa deficiência encontra-se na adição de termos polinomiais às funções de base do RPIM. Em geral, isso também aumenta a precisão dos resultados [33]. Adicionar termos polinomiais à RBFs foi proposto por [49] para aproximar funções. O uso de RBFs e polinômios de primeira ordem no RPIM foi sugerido por [57] para que as funções de forma tivessem consistência C^1. O RPIM incluindo termos polinomiais em suas funções de base é denomiado *Radial Point Interpolation Method with Polynomials* (RPIMp), ou método de interpolação de pontos radial com polinômios.

O RPIMp aproxima funções para um ponto de interesse $\mathbf{x}_Q$ da seguinte forma:

$$u^h(\mathbf{x}) = \sum_{i=1}^{n} R_i(\mathbf{x})a_i + \sum_{j=1}^{m} p_j(\mathbf{x})b_j = \mathbf{R}^T(\mathbf{x})\mathbf{a} + \mathbf{p}^T(\mathbf{x})\mathbf{b} \tag{3.56}$$

onde a_i são os coeficientes para as RBFs $R_i(\mathbf{x})$, b_i são os coeficientes para os termos polinomiais $p_j(\mathbf{x})$ (como definidos para o PIM em 3.37), n é o número de nós no domínio de suporte de $\mathbf{x}_Q$, m é o número de termos polinomiais adicionados na base, $\mathbf{R}^T(\mathbf{x})$ e $\mathbf{p}^T(\mathbf{x})$ têm a mesma definição que nas Equações 3.47 e 3.10, 3.11, 3.12, respectivamente, e $\mathbf{a}$ e $\mathbf{b}$ são vetores definidos como

$$\mathbf{a} = [a_1,\ a_2,\ \cdots,\ a_n]^T \tag{3.57}$$

$$\mathbf{b} = [b_1,\ b_2,\ \cdots,\ b_m]^T \tag{3.58}$$

Os coeficientes a_i e b_j são calculados forçando que a Equação 3.56 seja satisfeita nos n nós do domínio de suporte. Para cada nó de suporte l, tem-se:

$$u_l = \sum_{i=1}^{n} R_i(\mathbf{x}_l)a_i + \sum_{j=1}^{m} p_j(\mathbf{x}_l)b_j = \mathbf{R}^T(\mathbf{x}_l)\mathbf{a} + \mathbf{p}^T(\mathbf{x}_l)\mathbf{b}, \quad l = 1, 2, \cdots, n \tag{3.59}$$

onde u_l é o valor da função no nó l. A Equação 3.59 escrita na forma matricial torna-se:

$$\mathbf{U}_S = \mathbf{R}_Q\mathbf{a} + \mathbf{P}_m\mathbf{b} \tag{3.60}$$

onde $\mathbf{U}_S$ é definido como na Equação 3.40, $\mathbf{R}_Q$ é a matriz de momentos de RBFs definida na Equação 3.50, e $\mathbf{P}_m$ é a matriz de momentos de polinômios dada por:

$$\mathbf{P}_m = \begin{bmatrix} p_1(\mathbf{x}_1) & p_2(\mathbf{x}_1) & \cdots & p_m(\mathbf{x}_1) \\ p_1(\mathbf{x}_2) & p_2(\mathbf{x}_2) & \cdots & p_m(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ p_1(\mathbf{x}_n) & p_2(\mathbf{x}_n) & \cdots & p_m(\mathbf{x}_n) \end{bmatrix} \tag{3.61}$$

Os termos polinomiais devem satisfazer um critério extra para garantir uma aproximação única para a função, e as seguintes restrições são impostas [33]:

$$\sum_{i=1}^{n} p_j(\mathbf{x}_i)a_i = 0, \quad j = 1, 2, \cdots, m \tag{3.62}$$

ou na forma matricial:

$$\mathbf{P}_m^T\mathbf{a} = \mathbf{0} \tag{3.63}$$

Combinando as Equações 3.60 e 3.63, chega-se ao sistema matricial

$$\begin{bmatrix} \mathbf{R}_Q & \mathbf{P}_m \\ \mathbf{P}_m^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} = \begin{bmatrix} \mathbf{U}_S \\ \mathbf{0} \end{bmatrix} \tag{3.64}$$

ou

$$\mathbf{G} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} = \begin{bmatrix} \mathbf{U}_S \\ \mathbf{0} \end{bmatrix} \tag{3.65}$$

Como $\mathbf{R}_Q$ é simétrica, a matriz $\mathbf{G}$ também simétrica. Se $\mathbf{G}$ for inversível, então uma solução única para os vetores de coeficientes $\mathbf{a}$ e $\mathbf{b}$ é obtida como

$$\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} = \mathbf{G}^{-1} \begin{bmatrix} \mathbf{U}_S \\ \mathbf{0} \end{bmatrix} \tag{3.66}$$

Em vez de tentar resolver o sistema desta maneira, parte-se da Equação 3.60 para escrever $\mathbf{a}$ como:

$$\mathbf{a} = \mathbf{R}_Q^{-1}\mathbf{U}_S - \mathbf{R}_Q^{-1}\mathbf{P}_m\mathbf{b} \tag{3.67}$$

Substituindo a Equação 3.67 na 3.63, tem-se que

$$\mathbf{b} = \mathbf{S}_b\mathbf{U}_S \tag{3.68}$$

onde

$$\mathbf{S}_b = [\mathbf{P}_m^T\mathbf{R}_Q^{-1}\mathbf{P}_m]^{-1}\mathbf{P}_m^T\mathbf{R}_Q^{-1} \tag{3.69}$$

Substituindo a Equação 3.68 em 3.67, obtém-se

$$\mathbf{a} = \mathbf{S}_a\mathbf{U}_S \tag{3.70}$$

onde

$$\mathbf{S}_a = \mathbf{R}_Q^{-1}[1 - \mathbf{P}_m\mathbf{S}_b] = \mathbf{R}_Q^{-1} - \mathbf{R}_Q^{-1}\mathbf{P}_m\mathbf{S}_b \tag{3.71}$$

Finalmente, a Equação 3.56 é escrita como

$$u^h(\mathbf{x}) = \mathbf{\Phi}(\mathbf{x})\mathbf{U}_S = [\mathbf{R}^T(\mathbf{x})\mathbf{S}_a + \mathbf{p}^T(\mathbf{x})\mathbf{S}_b]\mathbf{U}_S \tag{3.72}$$

Logo, as funções de forma são dadas por

$$\mathbf{\Phi}(\mathbf{x}) = \mathbf{R}^T(\mathbf{x})\mathbf{S}_a + \mathbf{p}^T(\mathbf{x})\mathbf{S}_b = [\phi_1(\mathbf{x}),\ \phi_2(\mathbf{x}),\ \cdots,\ \phi_n(\mathbf{x})] \tag{3.73}$$

As derivadas das funções de forma são obtidas da Equação 3.73 derivando $\mathbf{R}^T$ e $\mathbf{p}^T$:

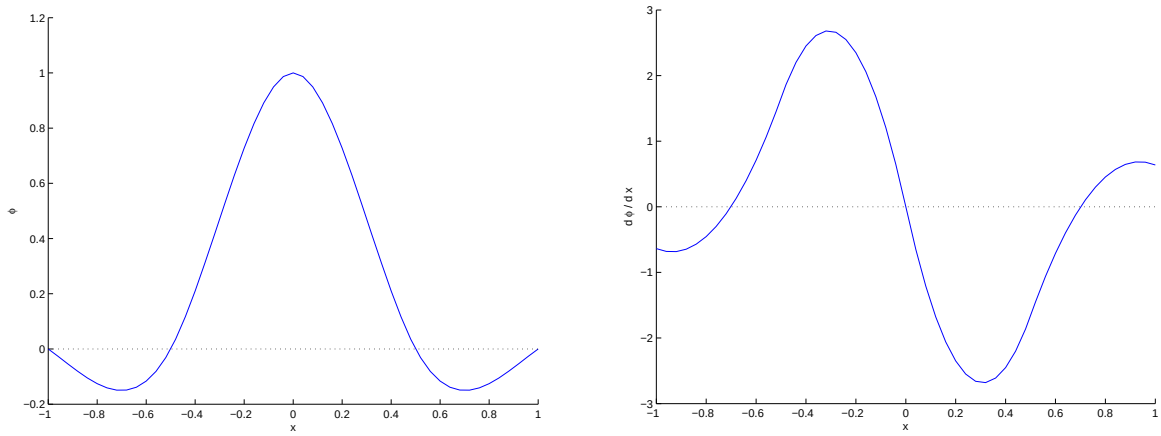
$$\mathbf{\Phi}_{,k}(\mathbf{x}) = \mathbf{R}^T_{,k}(\mathbf{x})\mathbf{S}_a + \mathbf{p}^T_{,k}(\mathbf{x})\mathbf{S}_b \tag{3.74}$$

Para que a matriz $\mathbf{P}_m^T\mathbf{R}_Q^{-1}\mathbf{P}_m$, definida na Equação 3.69, tenha inversa, é necessário que $n \gg m$, isto é, o número de nós no domínio de suporte deve ser muito maior que o número de termos polinomiais na base do RPIMp [33]. Normalmente são usados polinômios de primeira ordem, o que corresponde a um valor pequeno para m.

Com o RPIMp consegue-se gerar funções de forma consistentes (de acordo com a ordem polinomial usada nas funções de base). Como geralmente usa-se polinômios de grau um, a aproximação possui consistência C^1. As outras propriedades do PIM são preservadas: satisfação do delta de Kronecker, partição da unidade (se termos polinomiais de ordem zero estiverem na base) e suporte compacto. As funções do RPIMp também são incompatíveis devido aos saltos da aproximação que ocorrem quando o domínio de suporte sofre mudanças em regiões vizinhas.

A Figura 3.9 mostra a função de forma gerada pelo RPIMp e sua derivada em 1D. A função de forma em $x = 0$ foi obtida usando cinco nós igualmente distribuídos no domínio $[-1, 1]$, e a *Spline* cúbica dada pela Equação B.4 como RBF, com raio de suporte igual a 2. Assim como no PIM e no RPIM, as funções possuem a propriedade do delta de Kronecker.

A Figura 3.10 mostra a função de forma e suas derivadas em 2D no nó $(0, 0)$. Foram usados 5×5 nós igualmente distribuídos no domínio $[-2, 2] \times [-2, 2]$.



(a) (b)

Figura 3.9: Funções de forma RPIMp em 1D para o nó $x = 0$ usando 5 nós igualmente distribuídos em $x \in [-1, 1]$ e polinômios de primeira ordem na base. *Spline* cúbica foi usada como RBF, com $r = 2$. (a) Mostra $\phi(x)$ e (b) a derivada de $\phi(x)$ em relação a x. Note que as funções de forma construídas com o RPIMp satisfazem a propriedade do delta de Kronecker.

3.2.6 Resumo Comparativo dos Métodos

Após a apresentação dos diferentes métodos de construção de funções de forma, realiza-se uma comparação entre eles de acordo com as principais características discutidas no início da seção 3.2: consistência, partição da unidade, compatibilidade e critério do delta de Kronecker. O suporte compacto é atendido por todos os métodos através da escolha adequada dos nós de suporte na vizinhança do ponto de aproximação em que as funções de forma são construídas. A Tabela 3.1 mostra os comparativos entre os métodos.

3.3 Métodos sem malha

Esta seção apresenta os métodos sem malha presentes no *framework* proposto. O enfoque é dado à aplicação de problemas em duas dimensões. Adotou-se como base o problema estático geral dado pela Equação 2.38. Como ressaltado anteriormente, todos os métodos abordados neste trabalho utilizam formas fracas. Para cada um deles desenvolve-se a

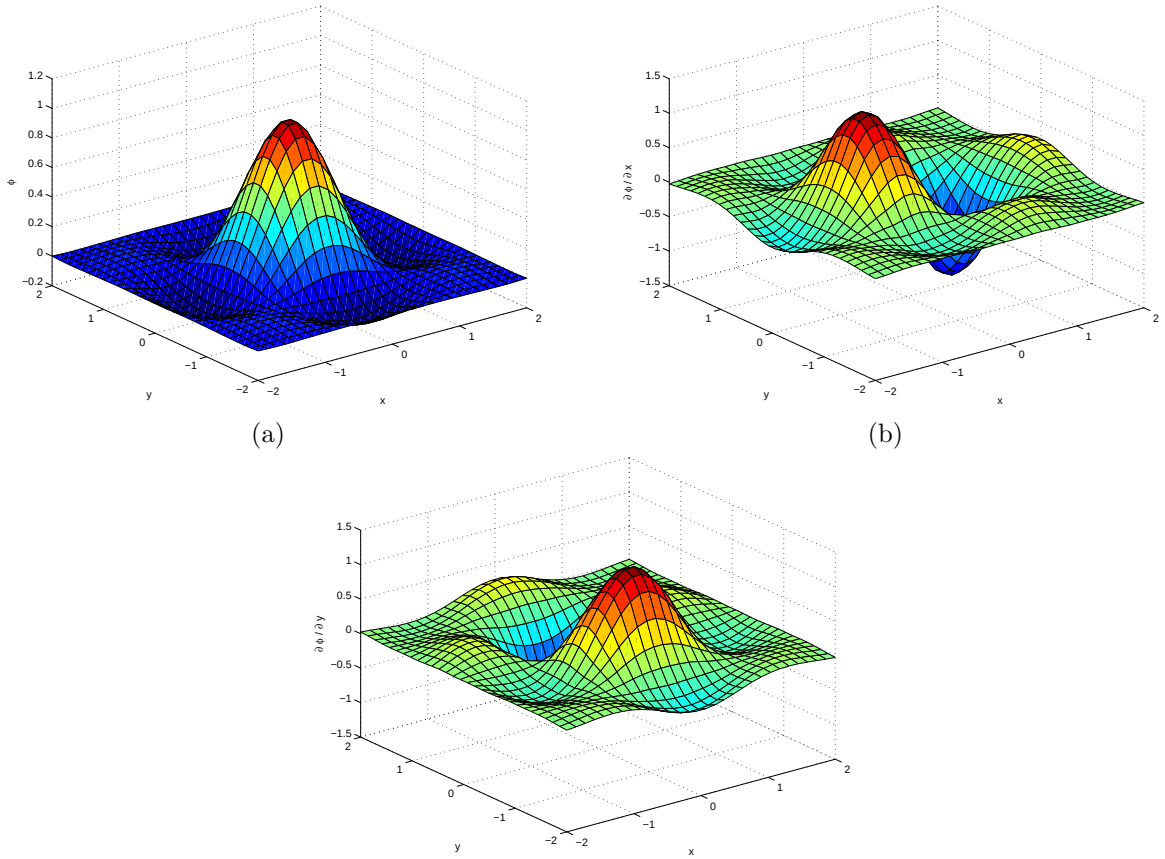


Figura 3.10: Funções de forma RPIMp em 2D para o nó $\mathbf{x} = (0, 0)$ usando 5×5 nós igualmente distribuídos em $(x, y) \in [-2, 2] \times [-2, 2]$ e polinômios de primeira ordem na base. *Spline* cúbica foi usada como RBF, com $r = 2$. (a) Mostra $\phi(x, y)$, (b) a derivada parcial de $\phi(x, y)$ em relação a x e (c) a derivada parcial de $\phi(x, y)$ em relação a y.

Tabela 3.1: Comparação entre os tipos de funções de forma.

Funções de Forma	**Consistência**[a]	**PU**[b]	**Compatibilidade**	**Delta de Kronecker**
MLS	C^k	Sim	Sim	Não
Shepard	C^0	Sim	Sim	Não
PIM	C^k	Sim	Não	Sim
RPIM	Não	Não	Não	Sim
RPIMp	C^k	Sim	Não	Sim

[a] k é grau do polinômio completo de mais alta ordem contido na base.
[b] Critério da partição da unidade.

formulação, discute-se como impor as condições de contorno essenciais, como realizar a integração da forma fraca e como tratar as descontinuidades de materiais.

3.3.1 Element-Free Galerkin Method

O *Element-Free Galerkin* (EFG) foi um dos primeiros métodos sem malha baseado no método de Galerkin a surgir. Desenvolvido em 1994 por Belytschko e outros [7], inicialmente, destinava-se a resolver problemas mecânicos e, com o tempo, passou a ser aplicado a problemas eletromagnéticos [47]. O EFG utiliza o Moving Least Squares para construir as funções de forma, o método de Galerkin para desenvolver o sistema de equações discretizado e uma malha de fundo para realizar a integração. Como as funções de forma MLS não possuem a propriedade do delta de Kronecker, técnicas adicionais são necessárias para impor as condições de contorno de Dirichlet. Para isto, duas abordagens são frequentemente utilizadas: multiplicadores de Lagrange [47] e método das penalidades [33].

O primeiro deles faz com que as condições de contorno essenciais sejam satisfeitas adicionando à formulação residual um termo referente aos multiplicadores de Lagrange. O sistema de equações discretizado torna-se maior e sua matriz deixa de ser definida positiva, aumentando o esforço computacional necessário para solucioná-lo. O método das penalidades adiciona um termo de penalidade à formulação do problema, as dimensões do sistema não sofrem alterações e sua matriz continua definida positiva. É um método mais simples, todavia, as condições de contorno essenciais nunca são impostas de forma precisa, apenas aproximada. Neste trabalho apresenta-se o EFG com o método das penalidades, mas o *framework* permite que outros modos de impor as condições de contorno essenciais sejam usados por meio de formulações implementadas pelo programador.

3.3.1.1 Formulação com o Método das Penalidades

Seja um domínio arbitrário Ω com um conjunto de nós distribuídos em seu interior e contorno Γ, como mostra a Figura 3.11. Repete-se a forma forte 2.38 para problemas

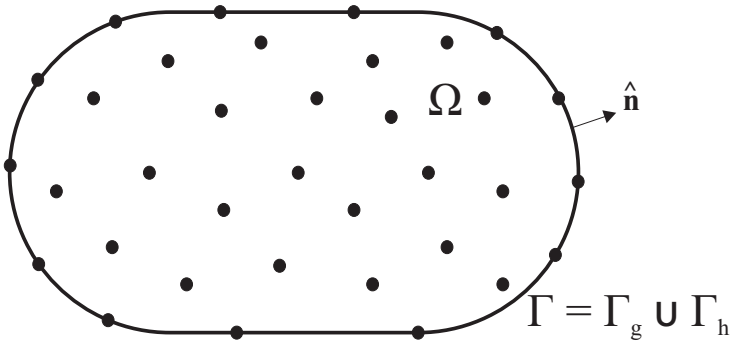


Figura 3.11: Representação de um domínio para o EFG. Γ é o contorno externo de Ω ($\Gamma = \partial\Omega$) e é formado pela união entre Γ_g (contorno onde se aplicam as condições de contorno de Dirichlet) e Γ_h (contorno onde se aplicam as condições de contorno de Neumann), com $\Gamma_g \cap \Gamma_h = \emptyset$.

estáticos por conveniência: dados k e f, tem-se por objetivo determinar a função $u : \Omega \rightarrow \Re$ que satisfaça

$$\begin{aligned} \nabla \cdot (k\nabla u) &= f && em \quad \Omega \\ u &= g && em \quad \Gamma_g \\ -k\frac{\partial u}{\partial n} &= h && em \quad \Gamma_h \end{aligned} \tag{3.75}$$

onde k é a característica de material do domínio Ω e f o termo fonte presente; g e h são os valores impostos pelas condições de Dirichlet e Neumann, respectivamente.

Seja S o espaço das *funções admissíveis* e U o espaço das *funções de teste*:

$$S = \{u \mid u \in H^1(\Omega)\} \tag{3.76}$$

$$U = \{w \mid w \in H^1(\Omega)\} \tag{3.77}$$

onde $H^1(\Omega)$ é o espaço de funções com derivada de primeira ordem de quadrado integrável. Através do método dos resíduos ponderados, escrevemos:

$$\int_\Omega [\nabla \cdot (k\nabla u) - f] w \; d\Omega = 0 \quad \forall w \in U \tag{3.78}$$

Como as funções de forma MLS não satisfazem o delta de Kronecker, utiliza-se o método das penalidades para impor as condições de contorno essenciais. A equação acima transforma-se em:

$$\int_\Omega [\nabla \cdot (k\nabla u) - f] w \; d\Omega - \alpha \int_{\Gamma_g} (u - g) w \; d\Gamma = 0 \quad \forall w \in U \tag{3.79}$$

onde α é uma constante, chamada *fator de penalidade*, que força a solução u ser igual a g no contorno de Dirichlet. Um dos problemas do método das penalidades é como escolher o valor adequado para α. Idealmente, este deve ser infinito para que as condições de contorno sejam completamente satisfeitas. Entretanto, na prática isso é inviável e as condições são atendidas de forma aproximada. Geralmente, usa-se um valor elevado para o fator de penalidade. O problema é que, se α for muito pequeno, as condições de contorno não são satisfeitas corretamente e, se α for muito grande, surgem problemas numéricos. Um estudo da influência do fator de penalidade sobre a precisão dos resultados é feito em [27], onde conclui-se que os melhores valores para α estão entre 10^4 e 10^8.

Reescrevendo a Equação 3.79, tem-se que

$$\int_\Omega [\nabla \cdot (k\nabla u)]w \; d\Omega - \alpha \int_{\Gamma_g} (u-g)w \; d\Gamma = \int_\Omega [f]w \; d\Omega \quad \forall w \in U \tag{3.80}$$

Seja a identidade vetorial

$$\nabla \cdot (g\overrightarrow{v}) = \nabla g \cdot \overrightarrow{v} + g\nabla \cdot \overrightarrow{v} \tag{3.81}$$

$$g\nabla \cdot \overrightarrow{v} = \nabla \cdot (g\overrightarrow{v}) - \nabla g \cdot \overrightarrow{v} \tag{3.82}$$

Reescrevendo a Equação 3.82 com a correspondência $g = w$ e $\overrightarrow{v} = k\nabla u$, tem-se que:

$$w\nabla \cdot (k\nabla u) = \nabla \cdot (wk\nabla u) - \nabla w \cdot (k\nabla u) \tag{3.83}$$

Substituindo a Equação 3.83 na Equação 3.80:

$$\int_\Omega [\nabla \cdot (wk\nabla u)] \; d\Omega - \int_\Omega [k\nabla w \cdot \nabla u] \; d\Omega - \alpha \int_{\Gamma_g} (u-g)w \; d\Gamma = \int_\Omega [f]w \; d\Omega \quad \forall w \in U \tag{3.84}$$

Aplicando o teorema da divergência no primeiro termo da Equação 3.84, tem-se que

$$\int_\Gamma [wk\nabla u \cdot \hat{\mathbf{n}}] \; d\Gamma - \int_\Omega [k\nabla w \cdot \nabla u] \; d\Omega - \alpha \int_{\Gamma_g} (u-g)w \; d\Gamma = \int_\Omega [f]w \; d\Omega \quad \forall w \in U \tag{3.85}$$

Considerando que $\nabla u \cdot \hat{\mathbf{n}} = \frac{\partial u}{\partial n}$, a Equação 3.85 é reescrita como:

$$-\int_\Omega [k\nabla w \cdot \nabla u] \; d\Omega - \alpha \int_{\Gamma_g} (u-g)w \; d\Gamma = \int_\Omega [f]w \; d\Omega - \int_\Gamma \left[wk\frac{\partial u}{\partial n}\right] \; d\Gamma \quad \forall w \in U \tag{3.86}$$

Como $\Gamma = \Gamma_g \cup \Gamma_h$ e pela condição de contorno de Neumann da Equação 3.75, a 3.86 pode ser escrita como

$$\int_\Omega [k\nabla w \cdot \nabla u]\ d\Omega + \alpha \int_{\Gamma_g} (u-g)w\ d\Gamma - \int_{\Gamma_g} \left[wk\frac{\partial u}{\partial n}\right]\ d\Gamma = -\int_\Omega [f]w\ d\Omega - \int_{\Gamma_h} [h]w\ d\Gamma \quad \forall w \in U \tag{3.87}$$

O valor que se utiliza para o fator de penalidade é, normalmente, muito alto e, com isso, o termo integral referente à penalidade é muito maior que o termo integral referente à derivada na direção normal em Γ_g da Equação 3.87, isto é,

$$\alpha \int_{\Gamma_g} (u-g)w\ d\Gamma \gg \int_{\Gamma_g} \left[wk\frac{\partial u}{\partial n}\right]\ d\Gamma \tag{3.88}$$

e o terceiro termo da Equação 3.87 pode ser desprezado. Com isso, a Equação 3.87 torna-se:

$$\int_\Omega [k\nabla w \cdot \nabla u]\ d\Omega + \alpha \int_{\Gamma_g} (u-g)w\ d\Gamma = -\int_\Omega [f]w\ d\Omega - \int_{\Gamma_h} [h]w\ d\Gamma \quad \forall w \in U \tag{3.89}$$

Pode-se, então, escrever a *forma fraca* como: dados f, g e h, determinar $u \in S$, tal que:

$$\int_\Omega k\nabla w \cdot \nabla u\ d\Omega + \alpha \int_{\Gamma_g} (u-g)w\ d\Gamma = -\int_\Omega wf\ d\Omega - \int_{\Gamma_h} wh\ d\Gamma \quad \forall w \in U \tag{3.90}$$

Observe que a solução da Equação 3.90 deve possuir derivadas de primeira ordem, enquanto a solução de 3.75 deve possuir derivadas de segunda ordem. Isso ocorre porque, durante o processo de construção da forma fraca, as derivadas das funções admissíveis são transferidas para as funções de teste. Ou seja, na formulação fraca buscam-se soluções com requisitos *mais fracos* de diferenciabilidade, justificando seu nome.

O EFG utiliza o método de Galerkin para obter as equações discretas da forma fraca por meio das aproximações:

$$\begin{aligned} u(\mathbf{x}) \approx u^h(\mathbf{x}) &= \sum_{i \in S_n} \phi_i(\mathbf{x}) u_i \\ w(\mathbf{x}) \approx w^h(\mathbf{x}) &= \sum_{i \in S_n} \phi_i(\mathbf{x}) w_i \end{aligned} \tag{3.91}$$

onde S_n é o conjunto de nós, pertencente ao domínio de suporte do ponto $\mathbf{x}$, usado para construir as funções de forma MLS $\phi_i(\mathbf{x})$. Note que as mesmas funções são usadas para gerar as aproximações das funções admissíveis e de teste.

Substituindo as expressões da Equação 3.91 na forma fraca 3.90, chega-se, após desenvolvimentos, ao sistema linear:

$$[\mathbf{K} + \mathbf{K}^{\alpha}]\mathbf{U} = \mathbf{F} + \mathbf{F}^{\alpha} \tag{3.92}$$

onde

$$\begin{aligned} K_{ij} &= \int_{\Omega} k \nabla \phi_i \cdot \nabla \phi_j \, d\Omega \\ K_{ij}^{\alpha} &= \alpha \int_{\Gamma_g} \phi_i \phi_j \, d\Gamma \\ F_i &= -\int_{\Omega} \phi_i f \, d\Omega - \int_{\Gamma_h} \phi_i h \, d\Gamma \\ F_i^{\alpha} &= \alpha \int_{\Gamma_g} \phi_i g \, d\Gamma \end{aligned} \tag{3.93}$$

Os coeficientes u_i de todos os nós do domínio são obtidos resolvendo o sistema linear 3.92. Depois disso, é possível calcular a aproximação para qualquer ponto $\mathbf{x}$ no domínio do problema.

3.3.1.2 Integração

A partir das Equações 3.93 fica evidente a necessidade de realizar integrações sobre o domínio do problema e sobre as fronteiras de Dirichlet e Neumann. Essas integrações devem ser feitas numericamente uma vez que se desconhecem expressões analíticas para as funções ϕ_i e ϕ_j. A quadratura de Gauss é a mais usada em métodos numéricos [33] e é adotada pelo MFree Framework. Técnicas de quadratura aproximam a integral de uma função por um somatório ponderado de alguns valores da função dentro do domínio. Para isso, exigem uma *malha de integração*. O procedimento de integração por quadratura gaussiana pode ser visto em [50]. No EFG, diferentemente do FEM, essa malha é usada apenas para integrar as equações discretizadas, e, em princípio, é totalmente independente da distribuição de nós.

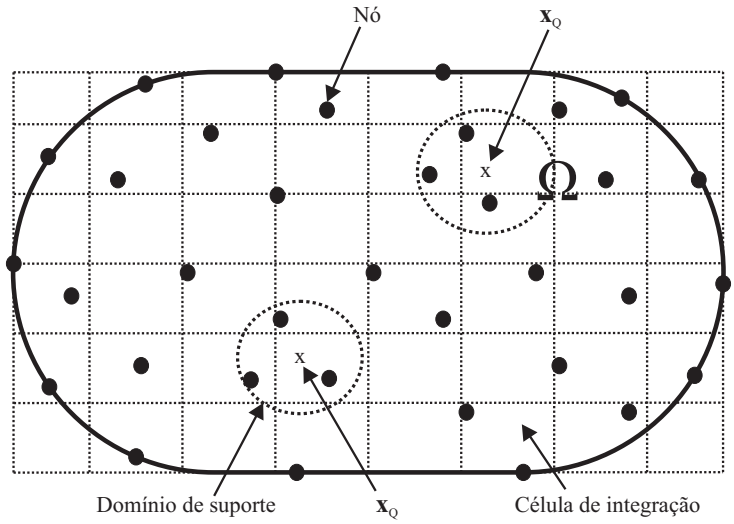


Figura 3.12: Malha de integração para o EFG. A malha é composta por células de integração. Cada célula contém pontos de quadratura $\mathbf{x}_Q$. Os domínios de suporte são determinados para cada ponto de quadratura.

A Figura 3.12 mostra uma malha de fundo para o domínio Ω da Figura 3.11, composta de células sem superposição. Atribui-se a cada célula *pontos de quadratura* $\mathbf{x}_Q$, também chamados *pontos de integração*. Para que o processo de integração seja realizado de forma correta, deve-se ficar atento à densidade da malha de fundo. De acordo com [33], é necessário que o número total de pontos de quadratura n_Q seja superior a dois terços do número de nós n_t no domínio do problema, isto é:

$$n_Q > \frac{2}{3} n_t \tag{3.94}$$

3.3.1.3 Procedimento de Solução

O procedimento de solução de um problema utilizando o EFG é similar ao do FEM [33]. Inicialmente, a geometria do problema é modelada, geram-se os nós para discretizar o domínio do problema, e a malha de integração é criada. O sistema linear de equações é montado por meio de dois *loops*: o externo para todas células da malha de fundo e o interno para os pontos de integração $\mathbf{x}_Q$ em um célula.

Para cada ponto de quadratura $\mathbf{x}_Q$, obtém-se os nós do domínio de suporte e, com eles, calculam-se as funções de forma por MLS. As equações discretizadas locais (Equação 3.93)

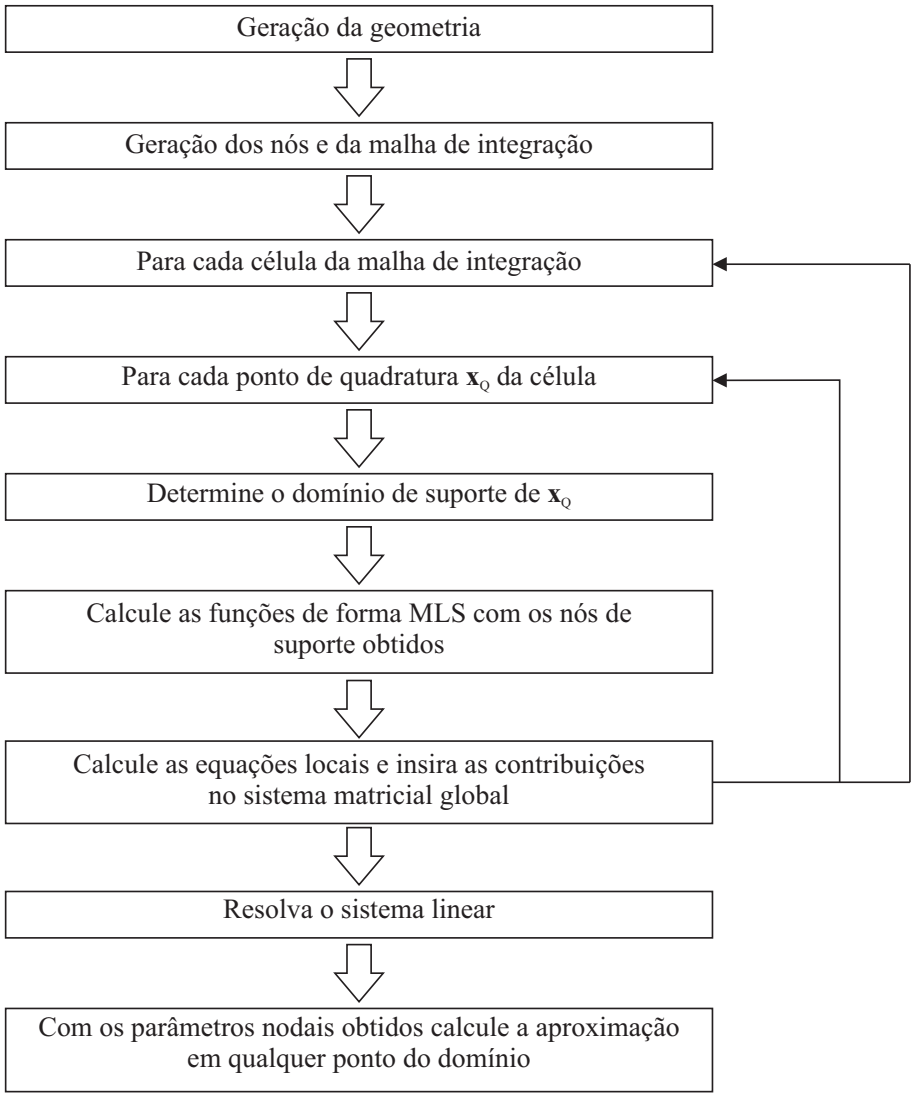


Figura 3.13: Diagrama de fluxo para o EFG.

são criadas e as contribuições efetuadas no sistema linear global da Equação 3.92. Depois de montado, o sistema é resolvido e, em seguida, obtém-se os parâmetros u_i de todos os nós do domínio. Ao final, as aproximações $u^h(\mathbf{x})$, em qualquer ponto $\mathbf{x}$ localizado no domínio do problema, podem ser efetuadas. A Figura 3.13 mostra o diagrama de fluxo para o processo de solução do EFG.

3.3.1.4 Tratamento das Descontinuidades de Materiais

O domínio de um problema pode ser composto de regiões com diferentes materiais. No FEM, o tratamento da descontinuidade de materiais é direto, pois as arestas dos elementos coincidem com as interfaces entre meios, e as funções de forma do FEM satisfazem a propriedade do delta de Kronecker. No entanto, no EFG não há como definir interfaces com base em elementos. Além disso, as funções de forma MLS não satisfazem o delta de Kronecker, e tratamentos especiais são necessários para garantir-se a continuidade da aproximação na fronteira entre materiais. Cordes e Moran [12] utilizam o método dos multiplicadores de Lagrange e tratam a interface de materiais como uma condição de contorno especial. Krongauz e Belytschko [28] propõem a adição de funções com descontinuidades em suas derivadas às funções de forma. Nesta seção apresenta-se o método das penalidades para o tratamento das condições de continuidade, como apresentado em [33].

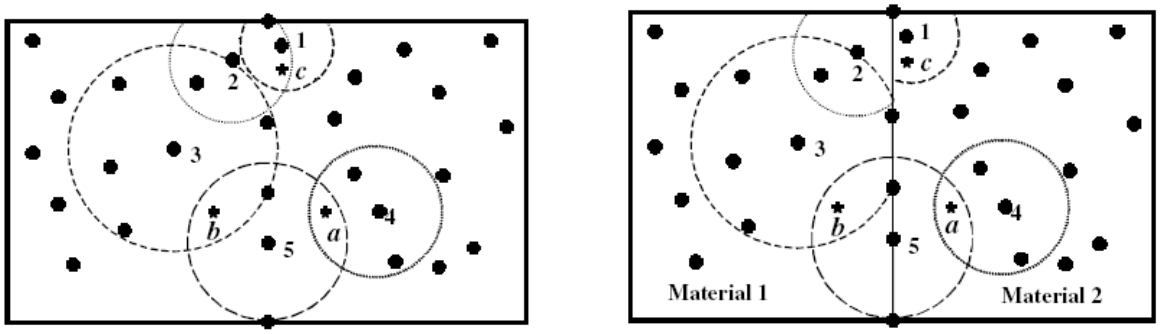
Considere um domínio composto de dois meios, como mostra a Figura 3.14b. Na interface, distribui-se um conjunto de nós que pertence aos dois meios ao mesmo tempo. Impõe-se a *regra de não penetração* dos domínios de influência: pontos contidos no material 1 são influenciados apenas por nós do material 1 e da interface; pontos contidos no material 2 são influenciados apenas por nós do material 2 e da interface; e pontos contidos na interface são influenciados por nós dos materiais 1 e 2 e da interface. Tome como exemplo o ponto c. Em um domínio com apenas um meio (Figura 3.14a), c possui os nós 1 e 2 em seu domínio de suporte. Entretanto, na Figura 3.14b, o nó 2 não está presente no domínio de suporte de c devido à regra de não penetração.

Em problemas de eletromagnetismo, as condições de continuidade entre materiais são expressas com duas restrições. Uma atua sobre a variável de aproximação e a outra sobre sua componente normal:

$$u_1 = u_2 \qquad em \quad \Gamma_I \tag{3.95}$$

$$k_1 \frac{\partial u_1}{\partial n} = k_2 \frac{\partial u_2}{\partial n} \qquad em \quad \Gamma_I \tag{3.96}$$

De acordo com o procedimento apresentado em [33], apenas uma condição de continuidade é forçada pelo método das penalidades. Adotando a restrição da Equação 3.95, pode-se



(a) (b)

Figura 3.14: Tratamento de interface de materiais no EFG. (a) Mostra os domínios de influência em um domínio com um material e (b) em um domínio com dois materiais. Figura retirada de [33].

reescrever a forma fraca (Equação 3.90) acrescentando um termo de penalidade para a restrição, o que resulta em:

$$\int_\Omega k\nabla w \cdot \nabla u \; d\Omega + \alpha \int_{\Gamma_g} (u-g)w \; d\Gamma + \beta \int_{\Gamma_I} (w_1 - w_2)(u_1 - u_2) \; d\Gamma$$
$$= -\int_\Omega wf \; d\Omega - \int_{\Gamma_h} wh \; d\Gamma \quad \forall w \in U \tag{3.97}$$

onde β é o fator de penalidade para a condição de continuidade.

Escrevendo as aproximações para u_1, u_2, w_1 e w_2 utilizando o MLS, tem-se que

$$\begin{aligned} u_1^h(\mathbf{x}) &= \sum_{i \in S_{n1}} \Phi_{i1}(\mathbf{x}) u_{i1} \\ u_2^h(\mathbf{x}) &= \sum_{i \in S_{n2}} \Phi_{i2}(\mathbf{x}) u_{i2} \\ w_1^h(\mathbf{x}) &= \sum_{i \in S_{n1}} \Phi_{i1}(\mathbf{x}) w_{i1} \\ w_2^h(\mathbf{x}) &= \sum_{i \in S_{n2}} \Phi_{i2}(\mathbf{x}) w_{i2} \end{aligned} \tag{3.98}$$

onde S_{n1} e S_{n2} são os conjuntos de nós de suporte que influenciam $\mathbf{x}$ nos meios 1 e 2, respectivamente, e Φ_{i1} e Φ_{i2} são as funções de forma MLS criadas usando esses nós.

A substituição das equações dadas por 3.98 na Equação 3.97 leva ao sistema discreto:

$$[\mathbf{K} + \mathbf{K}^\alpha + \mathbf{K}^\beta]\mathbf{U} = \mathbf{F} + \mathbf{F}^\alpha \tag{3.99}$$

onde $\mathbf{K}$, $\mathbf{K}^{\alpha}$, $\mathbf{F}$ e $\mathbf{F}^{\alpha}$ são dados pelas equações 3.93, e $\mathbf{K}^{\beta}$ é da forma:

$$K_{ij}^{\beta} = \beta \int_{\Gamma_I} [\phi_{i1} - \phi_{i2}][\phi_{j1} - \phi_{j2}] \, d\Gamma \tag{3.100}$$

Note que a integração é feita ao longo da interface dos materiais, por isso a matriz $\mathbf{K}^{\beta}$ possui elementos referentes aos nós localizados tanto na fronteira como nos meios 1 e 2. Apenas uma condição de continuidade foi imposta. É possível que as duas restrições sejam satisfeitas utilizando o método das penalidades. Para isso, duplicam-se os nós da fronteira entre os meios. Cada um destes nós gera uma equação no sistema linear global. Portanto, têm-se duas equações para cada nó de interface que corresponderão às restrições 3.95 e 3.96. Assim como as condições de contorno essenciais, as de continuidade na interface de materiais são impostas no *framework* por meio de formulações, permitindo que outros modos de tratar descontinuidades entre meios sejam utilizados.

3.3.2 Meshless Local Petrov-Galerkin Method

O EFG, visto na seção 3.3.1, apesar de ser um método MFree ainda requer uma malha para efetuar a integração da forma fraca. Por essa razão, não é um método totalmente livre de malha. O *Meshless Local Petrov-Galerkin Method* (MLPG) é um método sem malha desenvolvido em 1998 por Atluri e Zhu [6] que difere do EFG por utilizar a abordagem de Petrov-Galerkin na discretização da forma fraca e uma *formulação local*. Desse modo, o MLPG não necessita de uma malha para integração, pois esta é feita por *subdomínios locais*, sendo chamados métodos verdadeiramente sem malha.

O MLPG tem sido aperfeiçoado e estendido ao longo dos anos [33]. Atualmente, é um dos métodos sem malha de maior importância. Originalmente, as funções de forma podiam ser geradas utilizando diferentes abordagens, a saber, o Reproducing Kernel Particle Method, o Moving Least Squares, funções de base radial, funções de Shepard, dentre outras, e as condições de contorno essenciais eram impostas por meio do método das penalidades. Na verdade, o termo MLPG refere-se ao método sem malha utilizando uma formulação local de Petrov-Galerkin, e qualquer método de construção de funções de forma pode ser utilizado. Entretanto, na literatura, o MLPG recebe outras denominações dependendo do tipo de método usado para gerar as função de forma. Por exemplo, caso utilize-se o PIM,

o método é chamado LPIM (Seção 3.3.3.2). Neste trabalho, implicitamente, considera-se o MLPG com funções de aproximação geradas pelo MLS, mas quaisquer outros tipos são passíveis de ser utilizados no *framework*. Apresentam-se os métodos das penalidades e colocação [33] para que as condições de contorno de Dirichlet sejam satisfeitas.

Como afirmado anteriormente, o MLPG utiliza a formulação de Petrov-Galerkin, que permite as funções de forma e de teste serem escolhidas de maneira independente. A grande ideia por trás disso é que a forma integral do método dos resíduos ponderados fica confinada a um subdomínio pequeno em torno de um nó. Isso implica a forma fraca ser satisfeita em cada nó do problema em um sentido *integral local*. Portanto, a integração da forma fraca é realizada em um *domínio de quadratura local*, sendo independente de um nó para o outro. [33]. Além disso, esse domínios podem assumir qualquer forma, mas, normalmente, utilizam-se círculos e retângulos por serem mais simples. O único requisito imposto sobre os domínios de quadratura é que a união destes deve cobrir completamente o domínio do problema.

De acordo com a função de teste usada, o MLPG pode ser classificado da seguinte forma [5]:

- MLPG1: A função de teste é a função peso W usada para construir as funções de forma MLS.
- MLPG2: A função de teste é a função Delta de Dirac, resultando em um método de colocação.
- MLPG3: A função de teste é o resíduo da equação diferencial da forma forte, resultando em um problema de mínimos quadrados.
- MLPG4: A função de teste é a solução fundamental modificada da equação diferencial. Esta abordagem é conhecida como *Local Boundary Integration Equation Method* (LBIE).
- MLPG5: A função de teste é a função de Heaviside.
- MLPG6: A função de teste é idêntica à função de forma, resultando em uma abordagem equivalente ao método de Galerkin.

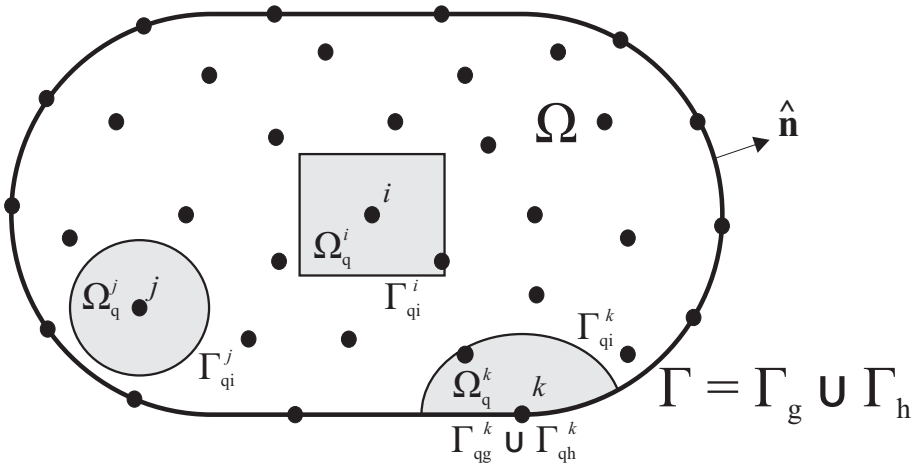


Figura 3.15: Representação de um domínio para o MLPG. Em destaque, os domínios de quadratura locais relativos aos nós i, j e k.

As versões do MLPG mais utilizadas são a MLPG1, MLPG4 e MLPG5. Atualmente, o MFree Framework disponibiliza as versões 1 e 5, mas outras podem ser implementadas.

3.3.2.1 Formulação com o Método das Penalidades

Seja um domínio arbitrário Ω com um conjunto de nós distribuídos em seu interior e contorno Γ, como mostra a Figura 3.15. Cada nó possui um domínio de quadratura local Ω_q cuja fronteira é Γ_q. Γ_q é composta da união de Γ_{qi}, Γ_{qg} e Γ_{qh}. Γ_{qi} é o contorno interno ao domínio global, Γ_{qg} a interseção entre Γ_q e a fronteira global de Dirichlet Γ_g e Γ_{qh} a interseção entre Γ_q e a fronteira global de Neumann Γ_h.

Considere o problema estático geral dado pela Equação 3.75. Como o MLPG usa uma formulação local, os espaçoes das funções admissíveis e de teste são definidos localmente, isto é, para cada subdomínio de quadratura Ω_q. Seja S o espaço das funções admissíveis e U o espaço das funções de teste:

$$S = \{u \mid u \in H^1(\Omega_q)\} \tag{3.101}$$

$$U = \{w \mid w \in H^1(\Omega_q)\} \tag{3.102}$$

onde $H^1(\Omega_q)$ é o espaço de funções com derivada de primeira ordem de quadrado integrável. Como se utiliza a formulação de Petrov-Galerkin, para cada nó i o método dos resíduos ponderados estabelece que:

$$\int_{\Omega_q^i} [\nabla \cdot (k\nabla u) - f] w_i \; d\Omega + \alpha \int_{\Gamma_{qg}^i} (u - g) w_i \; d\Gamma = 0 \quad \forall w_i \in U \tag{3.103}$$

onde se usou o método das penalidades para impor as condições de contorno essenciais. Seguindo um procedimento análogo ao realizado no EFG (desenvolvendo a expressão, aplicando identidades vetoriais e o teorema da divergência) chega-se a:

$$\int_{\Gamma_q^i} w_i k \frac{\partial u}{\partial n} \; d\Gamma - \int_{\Omega_q^i} k \nabla w_i \cdot \nabla u \; d\Omega + \alpha \int_{\Gamma_{qg}^i} (u - g) w_i \; d\Gamma = \int_{\Omega_q^i} w_i f \; d\Omega \quad \forall w_i \in U \tag{3.104}$$

que é a *forma fraca local* para o MLPG. Observe a diferença entre a equação acima e a forma fraca obtida para o EFG, dada pela Equação 3.90. Enquanto no EFG a equação deve ser satisfeita em todo o domínio Ω, no MLPG deve ser satisfeita apenas no domínio de quadratura local Ω_q de cada nó. Como $\Gamma_q^i = \Gamma_{qi}^i \cup \Gamma_{qg}^i \cup \Gamma_{qh}^i$, a Equação 3.104 pode ser reescrita como:

$$\begin{aligned} \int_{\Gamma_{qi}^i} w_i k \frac{\partial u}{\partial n} \; d\Gamma + \int_{\Gamma_{qg}^i} w_i k \frac{\partial u}{\partial n} \; d\Gamma + \int_{\Gamma_{qh}^i} w_i k \frac{\partial u}{\partial n} \; d\Gamma - \int_{\Omega_q^i} k \nabla w_i \cdot \nabla u \; d\Omega \\ + \; \alpha \int_{\Gamma_{qg}^i} (u - g) w_i \; d\Gamma = \int_{\Omega_q^i} w_i f \; d\Omega \quad \forall w_i \in U \end{aligned} \tag{3.105}$$

A seguir, a forma fraca é particularizada para as versões 1 e 5 do MLPG e apresentam-se os respectivos sistemas lineares discretizados.

MLPG1

No MLPG1 a função de teste w_i é a função de peso W utilizada para construir as funções de forma no MLS. Os suportes de w_i e W são independentes. Entretanto, escolhe-se w_i nula em Γ_{qi}^i para simplificar a formulação, isto é, $w_i = 0$ em Γ_{qi}^i. Assim, a primeira integral

da Equação 3.105 se anula. Desenvolvendo o termo referente à penalidade e tomando a condição de contorno de Neumann, a Equação 3.105 se torna:

$$-\int_{\Omega_q^i} k\nabla w_i \cdot \nabla u \; d\Omega + \; \alpha \int_{\Gamma_{qg}^i} w_i u \; d\Gamma + \int_{\Gamma_{qg}^i} w_i k \frac{\partial u}{\partial n} \; d\Gamma$$
$$= \int_{\Omega_q^i} w_i f \; d\Omega + \int_{\Gamma_{qh}^i} w_i h \; d\Gamma + \alpha \int_{\Gamma_{qg}^i} w_i g \; d\Gamma \quad \forall w_i \in U \tag{3.106}$$

que é a *forma fraca local* para o MLPG1.

As funções u são aproximadas por u^h da seguinte forma:

$$u(\mathbf{x}) \approx u^h(\mathbf{x}) = \sum_{i \in S_n} \phi_i(\mathbf{x}) u_i \tag{3.107}$$

onde S_n é o conjunto de nós, pertencente ao domínio de suporte do ponto $\mathbf{x}$, usado para construir as funções de forma MLS $\phi_i(\mathbf{x})$. Substituindo a aproximação dada pela Equação 3.107 na forma fraca local (Equação 3.106) chega-se, após desenvolvimentos, ao sistema discretizado:

$$[\mathbf{K} + \mathbf{K}^\alpha]\mathbf{U} = \mathbf{F} + \mathbf{F}^\alpha \tag{3.108}$$

onde

$$\begin{aligned} K_{ij} &= -\int_{\Omega_q^i} k\nabla w_i \cdot \nabla \phi_j \; d\Omega + \int_{\Gamma_{qg}^i} w_i k \frac{\partial \phi_j}{\partial n} \; d\Gamma \\ K_{ij}^\alpha &= \alpha \int_{\Gamma_{qg}^i} w_i \phi_j \; d\Gamma \\ F_i &= \int_{\Omega_q^i} w_i f \; d\Omega + \int_{\Gamma_{qh}^i} w_i h \; d\Gamma \\ F_i^\alpha &= \alpha \int_{\Gamma_{qg}^i} w_i g \; d\Gamma \end{aligned} \tag{3.109}$$

MLPG5

No MLPG5 a função de teste é a função de Heaviside, definida da seguinte maneira:

$$w_i(\mathbf{x}) = \begin{cases} 1 & \text{se} \quad \mathbf{x} \in \Omega_q^i \\ 0 & \text{se} \quad \mathbf{x} \notin \Omega_q^i \end{cases} \tag{3.110}$$

Dessa forma, $\nabla w_i = 0$. Desenvolvendo o termo referente à penalidade e tomando a condição de contorno de Neumann, a Equação 3.105 se torna:

$$\int_{\Gamma^i_{qi}} k\frac{\partial u}{\partial n}\, d\Gamma + \int_{\Gamma^i_{qg}} k\frac{\partial u}{\partial n}\, d\Gamma + \alpha \int_{\Gamma^i_{qg}} u\, d\Gamma = \int_{\Omega^i_q} f\, d\Omega + \int_{\Gamma^i_{qh}} h\, d\Gamma + \alpha \int_{\Gamma^i_{qg}} g\, d\Gamma \tag{3.111}$$

que é a *forma fraca local* para o MLPG5. Observe que o emprego da função de Heaviside faz com que a integral das funções de forma sobre o domínio seja cancelada, o que reduz muito o custo computacional do método.

Utilizando a aproximação dada pela Equação 3.107 e substituindo-a na Equação 3.111, chega-se, após desenvolvimentos, ao sistema discretizado:

$$[\mathbf{K} + \mathbf{K}^\alpha]\mathbf{U} = \mathbf{F} + \mathbf{F}^\alpha \tag{3.112}$$

onde

$$\begin{aligned} K_{ij} &= \int_{\Gamma^i_{qi}} k\frac{\partial \phi_j}{\partial n}\, d\Gamma + \int_{\Gamma^i_{qg}} k\frac{\partial \phi_j}{\partial n}\, d\Gamma \\ K^\alpha_{ij} &= \alpha \int_{\Gamma^i_{qg}} \phi_j\, d\Gamma \\ F_i &= \int_{\Omega^i_q} f\, d\Omega + \int_{\Gamma^i_{qh}} h\, d\Gamma \\ F^\alpha_i &= \alpha \int_{\Gamma^i_{qg}} g\, d\Gamma \end{aligned} \tag{3.113}$$

3.3.2.2 Integração

Como visto anteriormente, a integração da forma fraca no MLPG é realizada por subdomínios de quadratura e nenhuma malha é necessária. Assim como no EFG, as integrações devem ser feitas numericamente uma vez que se desconhecem as expressões analíticas para as funções ϕ_j, e a quadratura de Gauss é utilizada. Cada nó possui um domínio de integração Ω_q e, a cada um destes, são atribuídos pontos de quadratura $\mathbf{x}_Q$, como mostra a Figura 3.16.

Os domínios de quadratura, geralmente, são determinados de acordo com a densidade local de nós, de forma bem similar aos domínios de influência (ver Seção 3.1). Domínios circulares podem ser calculados por:

$$r_Q^i = \alpha_Q d_m^i \tag{3.114}$$

onde r_Q^i é o raio de quadratura do nó i a ser determinado, α_Q o fator de escalonamento para domínios de quadratura e d_m^i o espaçamento nodal médio na vizinhança de i. Para domínios de quadratura retangulares, utiliza-se a definição

$$r_{kQ}^i = \alpha_Q d_{km}^i \tag{3.115}$$

onde r_{kQ}^i é o raio de quadratura do nó i a ser determinado na direção k e d_{km}^i o espaçamento nodal médio do nó i na direção k. Em duas dimensões, por exemplo, $k = x, y$, r_{xQ}^i e r_{yQ}^i são os raios de quadratura do nó i nas direções x e y, respectivamente, e d_{xm}^i e d_{ym}^i os espaçamentos nodais médios nas direções horizontal e vertical.

A dimensão do domínio de quadratura é muito importante para a precisão e estabilidade do MLPG. Domínios pequenos geram resultados errados, enquanto domínios grandes acarretam dificuldades para o processo de integração. De acordo com [33], normalmente, valores de α_Q entre $1,5$ e $2,5$ geram os melhores resultados na prática, sendo $\alpha_Q = 1,5$ o mais econômico.

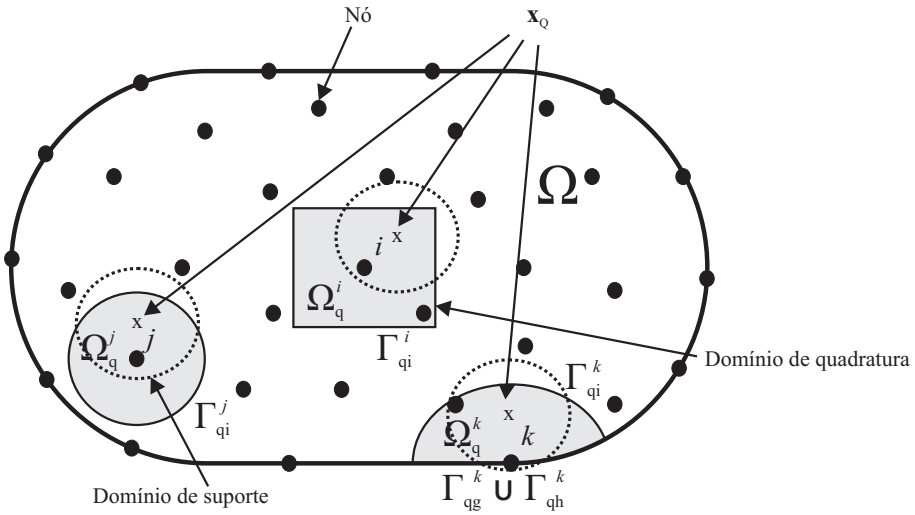


Figura 3.16: Domínios de quadratura para o MLPG. Cada domínio contém pontos de quadratura $\mathbf{x}_Q$. Os domínios de suporte são determinados para cada ponto de quadratura.

3.3.2.3 Procedimento de Solução

O procedimento de solução de um problema utilizando o MLPG é similar ao do EFG (3.13) de uma maneira. A grande diferença é que no MLPG não existe uma malha de fundo, e os pontos de quadratura $\mathbf{x}_Q$ referem-se ao domínio de integração Ω_q. Primeiramente, criam-se a geometria do problema, os nós e os respectivos domínios de quadratura Ω_q. Para cada nó do domínio, obtém-se o Ω_q correspondente. Para cada ponto de integração $\mathbf{x}_Q$ em Ω_q, determinam-se os nós de suporte e as funções de forma utilizando o MLS. Computam-se as equações locais, inserindo-as no sistema global. Após montado o sistema, este é resolvido, obtendo-se os parâmetros nodais u_i. As aproximações $u^h(\mathbf{x})$ podem, enfim, ser calculadas em todo o domínio do problema. A Figura 3.17 mostra o diagrama de fluxo para o MLPG.

No MLPG, a integração de um determinado subdomínio de quadratura Ω_q gera uma equação no sistema global e corresponde, assim, a uma linha deste. Portanto, a integração de diferentes nós gera contribuições em linhas distintas do sistema, e cada subdomínio de quadratura pode ser integrado de forma independente e em qualquer ordem, sem necessitar de mecanismos de sincronização. A Figura 3.18 ilustra as contribuições oriundas das integrações dos subdomínios referentes aos nós i e j. Observe que a integração de Ω_q^i corresponde à equação associada à linha i do sistema, enquanto a integração de Ω_q^j corresponde à linha j.

Essa é uma característica interessante do MLPG, pois permite que o processo de montagem do sistema de equações seja paralelizado de maneira natural, distribuindo, para cada processador, a tarefa de realizar a integração de um subdomínio Ω_q e contribuir na correspondente linha do sistema. Outra vantagem é que as condições de contorno essenciais podem ser impostas de forma alternativa, como é visto na próxima seção.

3.3.2.4 Método da Colocação

A imposição das condições de contorno essenciais pelo método das penalidades envolve a escolha do fator de penalidade α. Se α for escolhido de modo inadequado, instabilidade e resultados errôneos podem ocorrer. Devido às características do MLPG, existe uma

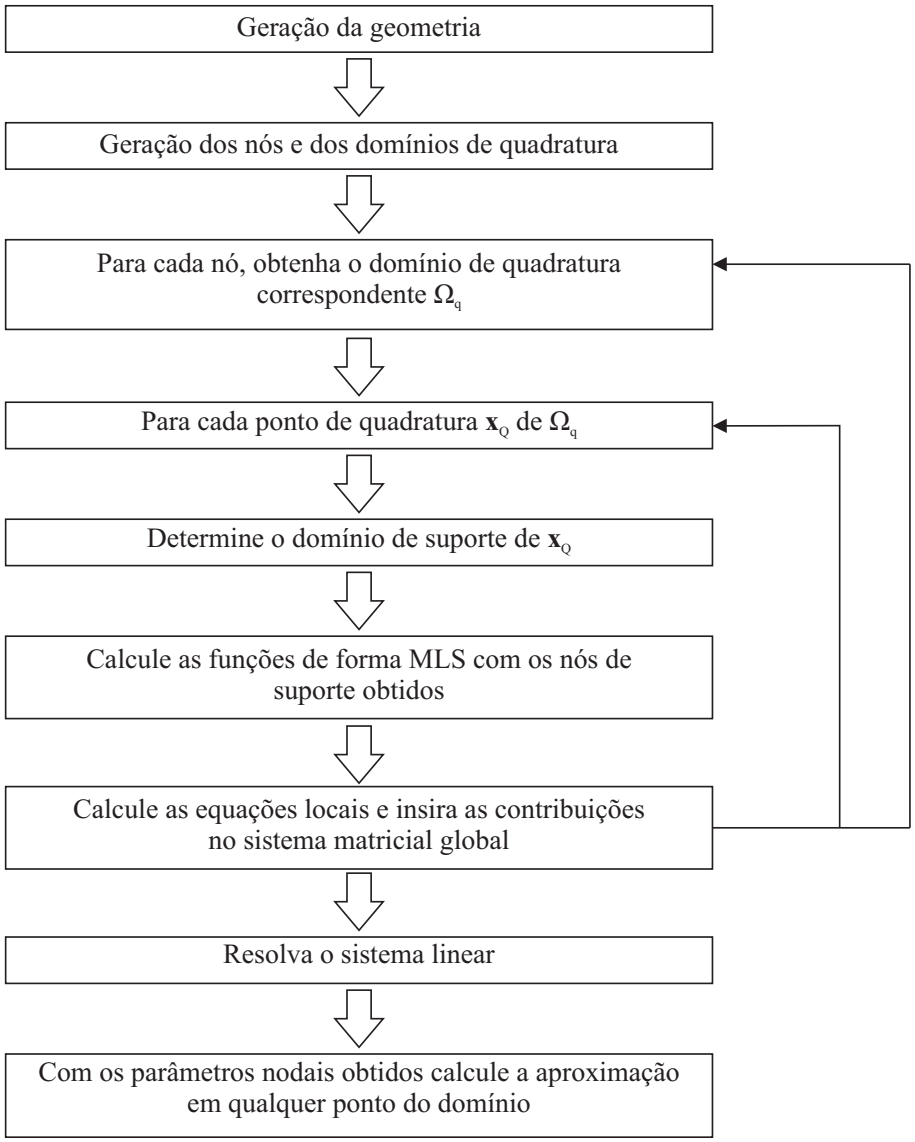


Figura 3.17: Diagrama de fluxo para o MLPG.

forma alternativa e simples de impor as condições de contorno essenciais. Esta forma é chamada *método da colocação* (ou *método da interpolação direta*) [33] e foi proposta com a intenção de simplificar a formulação do MLPG.

Como discutido na Seção 3.3.2.3, o MLPG monta as equações nó a nó, permitindo que se use diferentes equações para os nós no interior do domínio e na fronteira. Para cada nó j,

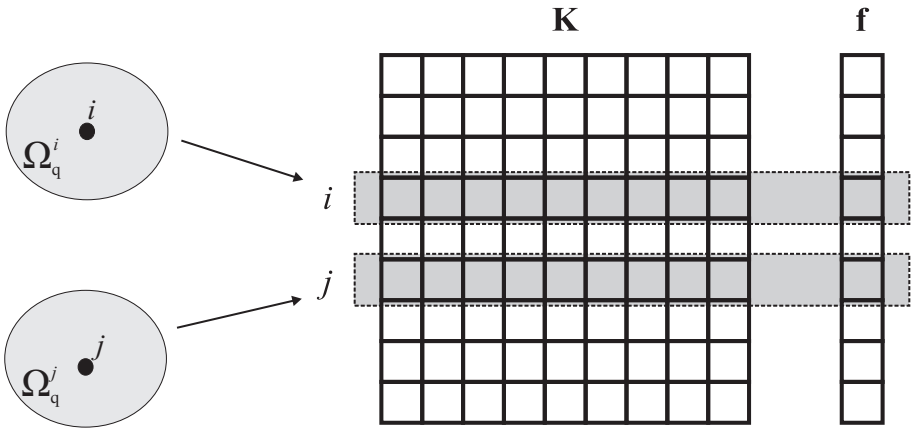


Figura 3.18: Contribuição dos nós no sistema de equações do MLPG. A integração do subdomínio Ω_q^i, referente ao nó i, contribui apenas na linha i do sistema, enquanto a integração do subdomínio Ω_q^j, referente ao nó j, apenas na linha j. Figura adaptada de [18].

localizado no contorno global de Dirichlet Γ_g, a condição de contorno essencial é forçada utilizando a aproximação MLS por colocação, isto é:

$$u_j^h(\mathbf{x}) = \sum_{i \in S_n} \phi_i(\mathbf{x}) u_i = g_j \tag{3.116}$$

onde g_j é o valor da condição de contorno de Dirichlet no nó j. A equação acima é uma equação algébrica linear para j. Portanto, não há necessidade de se calcular a equação da forma fraca local para os nós localizados em Γ_g e as condições essenciais são diretamente impostas no sistema de equações global.

Essa maneira de tratar as condições de contorno de Dirichlet é direta e eficaz, simplifica, significantemente, o processo de imposição das condições de contorno, agora satisfeitas exatamente. Além disso, a computação para os nós da fronteira de Dirichlet é simplificada [33], reduzindo o custo computacional.

Uma variação do método da interpolação direta é proposta em [31] e aplica o procedimento de colocação tanto nos nós da fronteira de Dirichlet quanto de Neumann. O método estabelece que os domínios de quadratura que possuem interseção com as fronteiras Γ_g e Γ_h devem ser reduzidos de modo a tangenciar estas fronteiras. Após isso, aplica-se a Equação 3.116 para os nós em Γ_g e a equação 3.117 para os nós em Γ_h:

$$k \frac{\partial u_j^h}{\partial n}\bigg|_{\mathbf{x}} = \sum_{i \in S_n} k \frac{\partial \phi_i}{\partial n}\bigg|_{\mathbf{x}} u_i = h_j \tag{3.117}$$

onde h_j é o valor da condição de contorno natural no nó j

3.3.2.5 Tratamento de Descontinuidades de Materiais

Quando o domínio do problema é composto de mais de um material, é necessário garantir a continuidade da aproximação na fronteira entre os meios, de acordo com as restrições dadas pelas Equações 3.95 e 3.96, como discutido na Seção 3.3.1.4. Apresenta-se, aqui, a abordagem proposta por [31] que aplica o método de colocação nos nós localizados na interface de materiais.

Cada meio do domínio é considerado separadamente, como mostra a Figura 3.19. Adota-se a regra de não penetração dos domínios de influência, como no EFG. Dois conjuntos de nós de colocação são distribuídos na fronteira entre os meios, cada conjunto pertencendo a um meio. Por exemplo, o nó i_1 pertence ao meio 1 e i_2 ao meio 2, mas ambos têm a mesma localização espacial no domínio do problema. Além disso, os domínios de quadratura Ω_q relativos aos nós que não estão na interface, mas, que a cruzam, são ajustados de modo a evitar o cruzamento. O procedimento do MLPG é aplicado, normalmente, para os nós que não estão em Γ_I. A cada par de nós i localizados na interface, aplica-se a aproximação MLS por colocação, de acordo com as equações de continuidade 3.95 e 3.96, resultando em:

$$\sum_{j \in S_{n1}} \phi_j(\mathbf{x}_{i1}) u_j - \sum_{k \in S_{n2}} \phi_k(\mathbf{x}_{i2}) u_k = 0 \tag{3.118}$$

$$\sum_{j \in S_{n1}} k_1 \frac{\partial \phi_j}{\partial n}\bigg|_{\mathbf{x}_{i_1}} u_j - \sum_{k \in S_{n2}} k_2 \frac{\partial \phi_k}{\partial n}\bigg|_{\mathbf{x}_{i_2}} u_k = 0 \tag{3.119}$$

onde S_{n1} é o conjunto de nós de suporte do nó i_1, usados para calcular as funções de forma ϕ_j e suas derivadas $\partial\phi_j/\partial n$, e S_{n2} o conjunto de nós de suporte de i_2, usados para calular ϕ_k e $\partial\phi_k/\partial n$.

Note que não é necessário calcular as equações da forma fraca local para os nós de interface e que não há domínios de quadratura que interceptam Γ_I. Caso interceptassem, domínios complicados para integração poderiam surgir. Dessa maneira, o método da colocação é considerado simples e elegante.

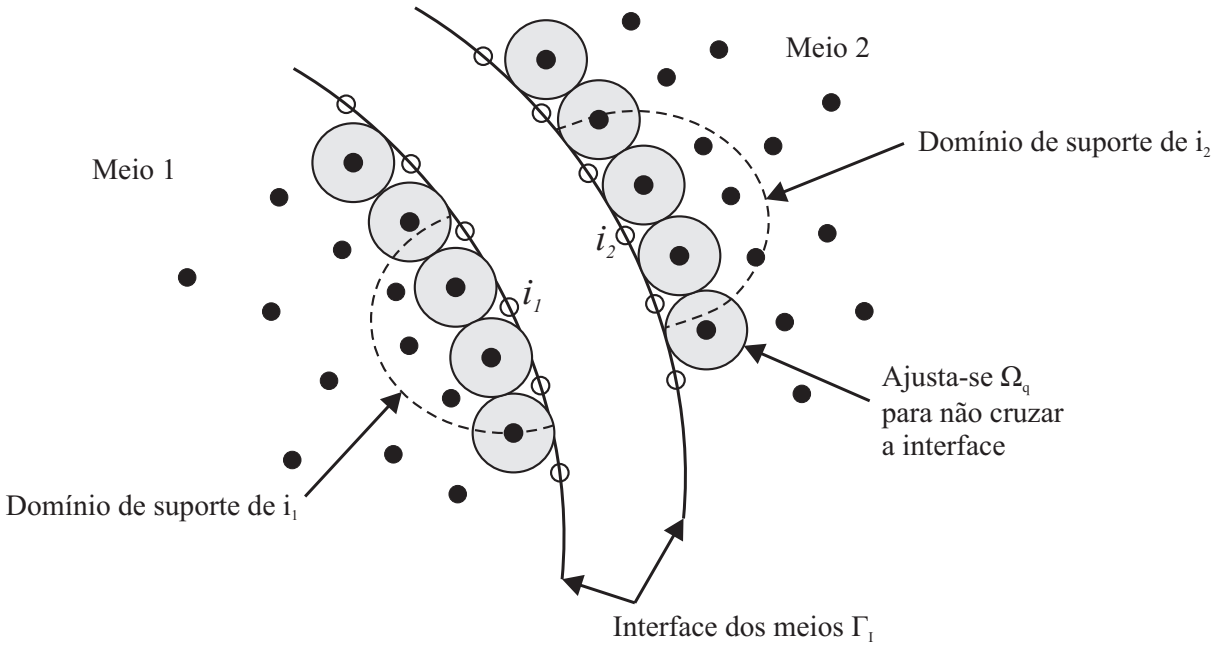


Figura 3.19: Tratamento de interface de materiais no MLPG. Nós de colocação i_1 e i_2 são dispostos em cada lado da interface.

3.3.3 Point Interpolation Methods

O *Point Interpolation Method* (PIM) foi desenvolvido por Liu e Gu em 1999 [34] com o objetivo de substituir as aproximações por MLS na construção das funções de forma. As grandes vantagens do PIM são a boa precisão nas aproximações e a satisfação do delta de Kronecker por suas funções de forma, permitindo que a imposição das condições de contorno essenciais seja facilmente realizada, assim como no FEM [32].

O PIM é classificado de acordo com o tipo de formulação e de função de forma utilizados. Se o método de Galerkin com uma forma fraca global é usado, chamamos o PIM de *Non-conforming Point Interpolation Method* (NPIM). Caso a formulação de Petrov-Galerkin com uma forma fraca local seja usada, é denominado *Local Point Interpolation Method* (LPIM). Cada um deles permite que uma das funções de forma PIM, RPIM ou RPIMp seja aplicada. A Tabela 3.2 resume os métodos existentes e suas características.

Tabela 3.2: Família do Point Interpolation Methods.

Sigla	**Nome**	**Formulação**	**Função de Forma**	**Compatibilidade**[a]
NPIM	Nonconforming Point Interpolation Method	Galerkin Global	PIM	Não
NRPIM	Nonconforming Radial Point Interpolation Method	Galerkin Global	RPIM	Não
NRPIMp	Nonconforming Radial Interpolation Method with polynomials	Galerkin Global	RPIMp	Não
LPIM	Local Point Interpolation Method	Petrov-Galerkin Local	PIM	Sim[b]
LRPIM	Local Radial Point Interpolation Method	Petrov-Galerkin Local	RPIM	Sim[b]
LRPIMp	Local Radial Point Interpolation Method with polynomials	Petrov-Galerkin Local	RPIMp	Sim[b]

[a]Ver Seção 3.2 para detalhes.
[b]Os problemas de incompatibilidade são amenizados por usar uma formulação local.

3.3.3.1 Nonconforming Point Interpolation Method

O *Nonconforming Point Interpolation Method* (NPIM) é um método da família do PIM que usa a formulação de Galerkin global. O NPIM é muito semelhante ao EFG, exceto pelas funções de forma usadas. Enquanto o EFG usa o MLS para gerar suas aproximações, o NPIM utiliza alguma variação do PIM. Isso simplifica a formulação, pois nenhum procedimento para impor as condições de contorno de Dirichlet é necessário, visto que as funções de forma satisfazem a propriedade do delta de Kronecker.

O método, como o nome diz, é não conforme por causa da característica de incompatibilidade de suas funções de aproximação, o que pode gerar descontinuidades na solução. Entretanto, [32] mostra que o NPIM converge para a solução exata mesmo com a presença de problemas de compatibilidade. As taxas de convergência são ligeiramente inferiores às obtidas pelo FEM, o que é justificado pelas não conformidades.

Os procedimentos de obtenção da forma fraca, do sistema de equações discretizado e de integração são idênticos aos do EFG, exceto pela definição dos espaços das funções admissíveis e de teste, pois as funções de forma do NPIM satisfazem a propriedade do delta de Kronecker enquanto as do EFG não. Sejam S e U os espaços das funções admissíveis e de testes, respectivamente, definidos da seguinte maneira:

$$S = \{u \mid u \in H^1(\Omega),\ u(x,y) = g, \quad \forall (x,y) \in \Gamma_g\} \tag{3.120}$$

$$U = \{w \mid w \in H^1(\Omega),\ w(x,y) = 0, \quad \forall (x,y) \in \Gamma_g\} \tag{3.121}$$

onde $H^1(\Omega)$ é o espaço de funções com derivada de primeira ordem de quadrado integrável.

Utilizando os mesmos procedimentos da Seção 3.3.1 para derivar a forma fraca e excluindo o termo da penalidade, chega-se a:

$$\int_\Omega k \nabla w \cdot \nabla u \ d\Omega = -\int_\Omega wf \ d\Omega - \int_{\Gamma_h} wh \ d\Gamma \quad \forall w \in U \tag{3.122}$$

que é a *forma fraca* para o NPIM. Com as aproximações dadas pela Equação 3.91, onde as funções ϕ_i agora são obtidas por algum método PIM, chega-se ao sistema linear discretizado:

$$\mathbf{KU} = \mathbf{F} \tag{3.123}$$

onde

$$\begin{aligned} K_{ij} &= \int_\Omega k \nabla \phi_i \cdot \nabla \phi_j \ d\Omega \\ F_i &= -\int_\Omega \phi_i f \ d\Omega - \int_{\Gamma_h} \phi_i h \ d\Gamma \end{aligned} \tag{3.124}$$

3.3.3.2 Local Point Interpolation Method

O *Local Point Interpolation Method* (LPIM) é um método da família do PIM que utiliza a formulação de Petrov-Galerkin local. Do mesmo modo que o MLPG, utiliza uma forma fraca local. A grande diferença entre os dois métodos é que o MLPG usa o MLS para construir suas funções de forma, enquanto o LPIM utiliza alguma variação do PIM. A formulação do LPIM é mais simples, pois não há necessidade de procedimentos adicionais para impor as condições de contorno essenciais, visto que as funções de forma satisfazem o delta de Kronecker.

Uma vez que o LPIM utiliza uma forma fraca local, os problemas de incompatibilidade das funções de forma PIM são amenizados [32]. De acordo com estudos, o LPIM possui taxas de convergência maiores que o FEM, EFG e MLPG [32].

Da mesma forma que utilizou-se o EFG para descrever o NPIM, o MLPG é usado como base para desenvolver a forma fraca e o sistema de equações discretizado do LPIM. Ambos possuem o mesmo procedimento de integração. Uma vez que as funções de forma do LPIM possuem a propriedade do delta de Kronecker e que a formulação utilizada é local, definem-se os espaços das funções admissíveis e de teste S e U, respectivamente, da seguinte forma:

$$S = \{u \mid u \in H^1(\Omega_q),\ u(x,y) = g, \quad \forall (x,y) \in \Gamma_{qg}\} \tag{3.125}$$

$$U = \{w \mid w \in H^1(\Omega_q)\} \tag{3.126}$$

onde $H^1(\Omega_q)$ é o espaço de funções com derivada de primeira ordem de quadrado integrável e Γ_{qg} é a parte da fronteira Γ_q do subdomínio local de integração que faz interseção com a fronteira global de Dirichlet. A partir dos desenvolvimentos da Seção 3.3.2 para constuir a formulação fraca, obtém-se:

$$\int_{\Gamma_q^i} w_i k \frac{\partial u}{\partial n}\, d\Gamma - \int_{\Omega_q^i} k \nabla w_i \cdot \nabla u\, d\Omega = \int_{\Omega_q^i} w_i f\, d\Omega \quad \forall w_i \in U \tag{3.127}$$

que é a *forma fraca local* para o LPIM. Assim como o MLPG, o LPIM possui as variações 1 a 6, descritas na Seção 3.3.2. As mais usadas são as versões 1 e 5, que são mostradas a seguir.

LPIM1

O LPIM1 utiliza como função de teste uma função de base radial que se anula na fronteira dos sudomínios locais de integração. Com isso, a *forma fraca local* para o LPIM1 resulta em:

$$-\int_{\Omega_q^i} k \nabla w_i \cdot \nabla u\, d\Omega + \int_{\Gamma_{qg}^i} w_i k \frac{\partial u}{\partial n}\, d\Gamma = \int_{\Omega_q^i} w_i f\, d\Omega + \int_{\Gamma_{qh}^i} w_i h\, d\Gamma \quad \forall w_i \in U \tag{3.128}$$

Através da aproximação dada pela Equação 3.107 e da expressão acima, obtém-se o sistema linear de equações:

$$\mathbf{KU} = \mathbf{F} \tag{3.129}$$

onde

$$K_{ij} = -\int_{\Omega_q^i} k\nabla w_i \cdot \nabla\phi_j \; d\Omega + \int_{\Gamma_{qg}^i} w_i k\frac{\partial\phi_j}{\partial n} \; d\Gamma$$
$$F_i = \int_{\Omega_q^i} w_i f \; d\Omega + \int_{\Gamma_{qh}^i} w_i h \; d\Gamma \tag{3.130}$$

LPIM5

O LPIM5 utiliza a função de Heaviside (Equação 3.110) como função de teste. Com isso, a *forma fraca local* para o LPIM5 resulta em:

$$\int_{\Gamma_{qi}^i} k\frac{\partial u}{\partial n} \; d\Gamma + \int_{\Gamma_{qg}^i} k\frac{\partial u}{\partial n} \; d\Gamma = \int_{\Omega_q^i} f \; d\Omega + \int_{\Gamma_{qh}^i} h \; d\Gamma \tag{3.131}$$

Através da aproximação dada pela Equação 3.107 e da expressão acima, obtém-se o sistema linear de equações:

$$\mathbf{KU} = \mathbf{F} \tag{3.132}$$

onde

$$K_{ij} = \int_{\Gamma_{qi}^i} k\frac{\partial\phi_j}{\partial n} \; d\Gamma + \int_{\Gamma_{qg}^i} k\frac{\partial\phi_j}{\partial n} \; d\Gamma$$
$$F_i = \int_{\Omega_q^i} f \; d\Omega + \int_{\Gamma_{qh}^i} h \; d\Gamma \tag{3.133}$$

3.3.3.3 Tratamento de Descontinuidades de Materiais com o LPIM

Se o domínio do problema é composto de mais de um meio com materiais diferentes, é necessário garantir que a aproximação seja contínua na interface entre os meios, de acordo com as Equações 3.95 e 3.96. Pode-se utilizar o mesmo procedimento do MLPG, descrito na Seção 3.3.2.5. Contudo, como as funções de forma do LPIM satisfazem o delta de Kronecker, um método alternativo utilizando o *critério da visibilidade generalizado* pode ser empregado.

Seja um domínio Ω formado por dois meios Ω_1 e Ω_2, como mostra a Figura 3.20. Os nós são divididos em três conjuntos: S_1, S_2 e S_I. S_1 contém os nós que pertencem exclusivamente à região 1 (nós j, m, k), S_2 os nós exclusivos da região 2 (nós n, p) e S_I os nós que estão sobre a interface Γ_I (nó i). O critério da visibilidade é aplicado aos domínios de influência. Desse modo, o domínio de suporte de um ponto em Ω_1 contém nós de S_1 e S_I. Similarmente, o domínio de suporte de um ponto em Ω_2 contém nós de S_2 e S_I. Por fim, um ponto em Γ_I conterá nós de suporte de S_1, S_2 e S_I.

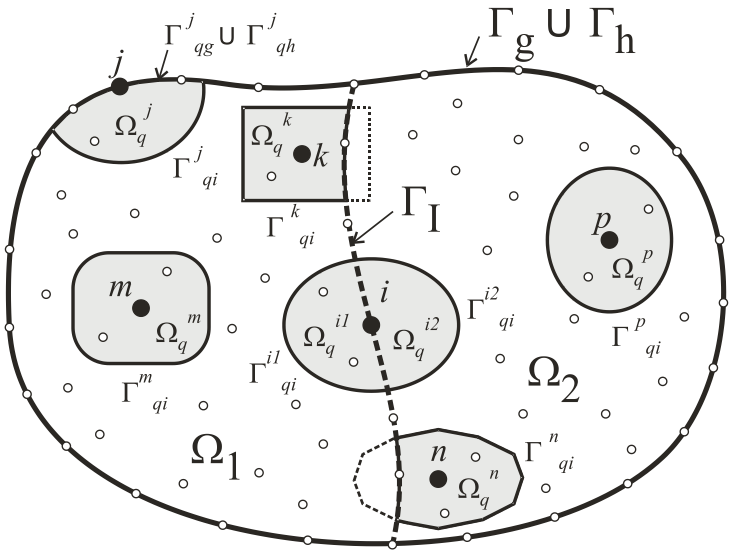


Figura 3.20: Tratamento de interface de materiais no LPIM. Representação do domínio Ω com destaque para os domínios de quadratura localizados nas regiões 1 e 2 e na interface Γ_I.

Aplica-se o critério da visibilidade também para os domínios locais de quadratura. Consideram-se as regiões separadamente. Para a integração do domínio local de um nó de S_1, apenas a geometria correspondente à região 1 é utilizada, Γ_I é considerada uma fronteira interna e a equação da forma fraca local é usada. Para o nó k, por exemplo, o domínio local de quadratura resultante é mostrado na Figura 3.20 e todas as fronteiras são consideradas internas. A mesma ideia se aplica à região 2.

Um nó i localizado em Γ_I possui o domínio de integração local tanto nas regiões 1 e 2, como mostra a Figura 3.21, e a fronteira Γ_I é considerada separadamente. A forma fraca

local, dada pela Equação 3.127, é aplicada a i. Para a porção de Ω_q^i na região 1 (Ω_q^{i1}), temos que:

$$\int_{\Gamma^{i1}_{qi\cup qg\cup qh}} w_i k_1 \frac{\partial u}{\partial n}\, d\Gamma + \int_{\Gamma^{i1}_{qI}} w_i k_1 \frac{\partial u}{\partial n}\, d\Gamma - \int_{\Omega_q^{i1}} k_1 \nabla w_i \cdot \nabla u\, d\Omega = \int_{\Omega_q^{i1}} w_i f\, d\Omega \quad \forall w_i \in U \tag{3.134}$$

e para Ω_q^{i2}:

$$\int_{\Gamma^{i2}_{qi\cup qg\cup qh}} w_i k_2 \frac{\partial u}{\partial n}\, d\Gamma + \int_{\Gamma^{i2}_{qI}} w_i k_2 \frac{\partial u}{\partial n}\, d\Gamma - \int_{\Omega_q^{i2}} k_2 \nabla w_i \cdot \nabla u\, d\Omega = \int_{\Omega_q^{i2}} w_i f\, d\Omega \quad \forall w_i \in U \tag{3.135}$$

A integração é feita para todo o domínio local, por isso somam-se as Equações 3.134 e 3.135, que resulta em:

$$\begin{aligned} &\int_{\Gamma^{i1}_{qi\cup qg\cup qh}} w_i k_1 \frac{\partial u}{\partial n}\, d\Gamma + \int_{\Gamma^{i1}_{qI}} w_i k_1 \frac{\partial u}{\partial n}\, d\Gamma - \int_{\Omega_q^{i1}} k_1 \nabla w_i \cdot \nabla u\, d\Omega \\ +&\int_{\Gamma^{i2}_{qi\cup qg\cup qh}} w_i k_2 \frac{\partial u}{\partial n}\, d\Gamma + \int_{\Gamma^{i2}_{qI}} w_i k_2 \frac{\partial u}{\partial n}\, d\Gamma - \int_{\Omega_q^{i2}} k_2 \nabla w_i \cdot \nabla u\, d\Omega \\ &= \int_{\Omega_q^{i1}} w_i f\, d\Omega + \int_{\Omega_q^{i2}} w_i f\, d\Omega \quad \forall w_i \in U \end{aligned} \tag{3.136}$$

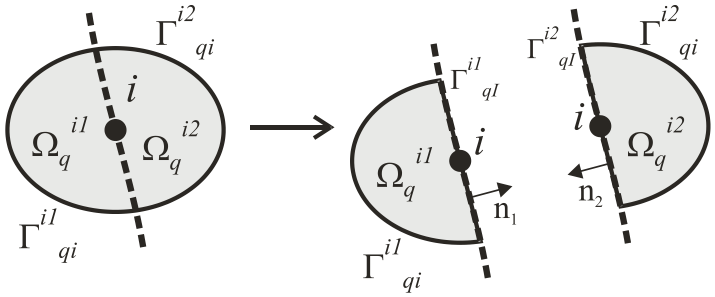


Figura 3.21: Domínio de integração local para um nó i na interface. O critério da visibilidade é aplicado, dividindo Ω_q^i em dois: um pertencente à região 1 (Ω_q^{i1}) e outro à região 2 (Ω_q^{i2}).

Como a interface é a mesma para as duas porções do subdomínio ($\Gamma^{i1}_{qI} = \Gamma^{i2}_{qI}$), $\hat{\mathbf{n}}_1 = -\hat{\mathbf{n}}_2$ e as funções de forma possuem a propriedade do delta de Kronecker, pela condição de

continuidade na interface 3.96 as integrais relativas a Γ_{qI}^{i1} e Γ_{qI}^{i2} são canceladas na Equação 3.136. A equação final para um nó i localizado na interface é dada por:

$$\int_{\Gamma_{qi\cup qg\cup qh}^{i1}} w_i k_1 \frac{\partial u}{\partial n}\, d\Gamma + \int_{\Gamma_{qi\cup qg\cup qh}^{i2}} w_i k_2 \frac{\partial u}{\partial n}\, d\Gamma - \int_{\Omega_q^{i1}} k_1 \nabla w_i \cdot \nabla u\, d\Omega - \int_{\Omega_q^{i2}} k_2 \nabla w_i \cdot \nabla u\, d\Omega = \int_{\Omega_q^{i1}} w_i f\, d\Omega + \int_{\Omega_q^{i2}} w_i f\, d\Omega \quad \forall w_i \in U \tag{3.137}$$

Note que, ao contrário do método de colocação, os nós da interface não são duplicados e o sistema global permamece com o mesmo número de equações. As equações para os nós em Γ_I envolvem integrais, isto é, são satisfeitas nas regiões vizinhas a cada nó gerando, assim, uma solução melhor que o método da colocação, em que as equações são algébricas e satisfetas pontualmente.

3.3.4 MLPG Misto

O *MLPG misto* foi proposto em 2008 por Fonseca e outros [19] como uma técnica alternativa de impor as condições de contorno essenciais no MLPG tradicional. Como discutido na Seção 3.3.2, o MLPG constrói suas funções de forma através do MLS e técnicas adicionais, como o método da penalidade, são usadas para impor as condições de contorno essenciais devido ao fato das aproximações por MLS não possuírem a propriedade do delta de Kronecker. Também foi visto que no LPIM (Seção 3.3.3.2) as condições de contorno essenciais são diretamente satisfeitas porque as funções de forma PIM são interpolantes.

O método misto combina os métodos MLPG e LPIM através do uso alternado das funções de forma MLS e PIM. De maneira sucinta, o método misto utiliza o MLS para construir as funções de forma no interior do domínio e o PIM para as funções de forma próximas ou no contorno de Dirichlet. Desse modo, as condições essenciais são satisfeitas diretamente. Essa abordagem é interessante pois une o baixo custo computacional do MLS [19] e a imposição direta das condições de contorno de Dirichlet via PIM. A combinação das funções de forma só é possível pois satisfazem o critério da partição da unidade. Originalmente, o método misto utiliza a função de Heaviside (Equação 3.110) como função de teste, e o MLS e o RPIMp para construir as funções de forma.

A determinação das funções de forma a serem utilizadas pelos nós é feita com base em uma classificação de acordo com a proximidade ao contorno de Dirichlet Γ_g. Um nó é classificado como *externo* caso seu domínio de suporte contenha algum nó localizado em Γ_g. Do contrário, o nó é classificado como *interno*. A Figura 3.22 ilustra o procedimento de classificação.

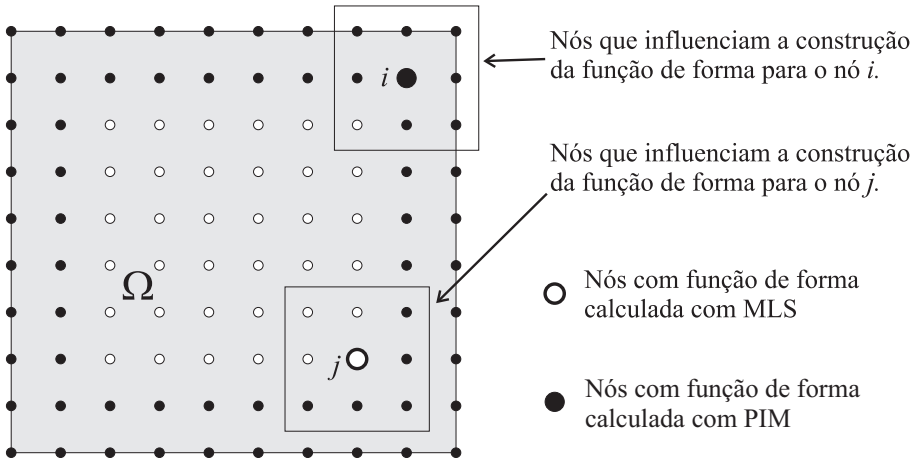


Figura 3.22: Classificação dos nós no MLPG misto. A classificação é feita de acordo com a proximidade à fronteira de Dirichlet. Nós cujos domínios de suporte contenham nós localizados no contorno de Dirichlet são classificados como externos (nó i). Caso contrário, são classificados como externos (nó j). Figura retirada de [19].

A função de forma usada é definida de acordo com a classificação do nó que define o domínio de integração. Na Figura 3.22, quando se estiver construindo a aproximação para o nó i, as funções de forma utilizadas por ele e seus vizinhos (nós de suporte) é o PIM, pois i é um nó externo. Por outro lado, a aproximação para o nó j é feita utilizando o MLS, pois j é um nó interno. Note que um nó pode ter duas funções de forma diferentes associadas, dependendo do subdomínio sendo integrado.

A formulação do MLPG misto é a mesma do LPIM, dada pela Equação 3.127, tomando-se o cuidado de utilizar as funções de forma de acordo a metodologia descrita anteriormente. Os resultados em [19] mostram que o MLPG misto gera aproximações com boa precisão e com custo computacional inferior ao LPIM, principalmente quando aumenta-se o número de nós no domínio do problema.

3.4 Solução do Sistema Linear de Equações

Uma das últimas etapas no processo de solução de um problema por métodos sem malha é resolver o sistema linear de equações. Existem diversos métodos, diretos e iterativos, para esta finalidade. Aqueles que trabalham com matrizes esparsas são preferíveis uma vez que o sistema discretizado é, normalmente, esparso, consequência do suporte compacto das funções de forma dos métodos MFree. Dentre as técnicas existentes, podemos citar:

- Decomposição LU
- Decomposição de Cholesky
- Método do Gradiente Conjugado (GC)
- Método do Gradiente Bi-conjugado (BICG)
- Método do Resíduo Mínimo Generalizado (GMRES)
- Método Multifrontal Assimétrico aplicado à Decomposição LU

Todos os métodos, com exceção do último, são descritos em [50]. O Método Multifrontal Assimétrico [14] é a solução atualmente utilizada pelo MFree Framework por meio da biblioteca *Unsymmetric Multifrontal Sparse LU Factorization Package* (UMFPACK) [1]. Apesar do método ser destinado a sistemas não simétricos, gerados com formulações Petrov-Galerkin, pode ser aplicado também a sistemas simétricos, geralmente obtidos com formulações Galerkin. Pretende-se, em versões futuras do *framework*, adicionar implementações de métodos dedicados exclusivamente a sistemas simétricos, por exemplo, o do Gradiente Conjugado.

Capítulo 4

MFree Framework

Este capítulo descreve a arquitetura e as funcionalidades do MFree Framework, bem como os conceitos e classes presentes. Como visto anteriormente, o *framework* é desenvolvido na linguagem de programação C++ utilizando o paradigma da programação genérica*. Inicialmente, apresenta-se uma visão geral de sua arquitetura e, posteriormente, as partes mais importantes que o compõem. Para mostrar a relação entre conceitos e classes, e destes entre si, são usados diagramas UML†. Nos diagramas de classes, elementos amarelos e cinza correspondem, respectivamente, a conceitos e a itens concretos presentes no *framework*, como classes e *function objects* (Na versão preto e branco do texto, conceitos estão em cinza claro e itens concretos em cinza escuro).

Deve-se ressaltar que este texto apresenta apenas as principais partes do *framework*, suficientes para descrever sua arquitetura. O detalhamento completo de todos os componentes é apresentado como documentação junto ao código fonte.

* Para maior detalhes, ver Seção 2.2
† Detalhes sobre diagramas UML podem ser vistos em [21]

4.1 Arquitetura do Framework

O código-fonte do MFree Framework está completamente contido em um *namespace* chamado `MFREE`, criado para efetuar a organização lógica do *framework* e separá-lo do escopo global. Os arquivos estão separados por pastas de acordo com as funcionalidades das classes ou funções que contêm. Todos os nomes de tipos e funções possuem a terminação referente à dimensão a que são destinados. Por exemplo, `Domain_1` e `Domain_2` representam domínios em 1D e 2D, respectivamente. Se a terminação de dimensão é suprimida, então a aplicabilidade é a qualquer dimensão.

Como o *framework* é baseado em programação genérica, diversos conceitos são identificados e a implementação de cada tipo é feita de forma a modelar um determinado conceito ou um conjunto deles. Dessa maneira, novos tipos podem ser implementados pelo desenvolvedor desde que satisfaçam os requisitos impostos pelos conceitos existentes, encontrando-se aí os os pontos de extensão do *framework*.

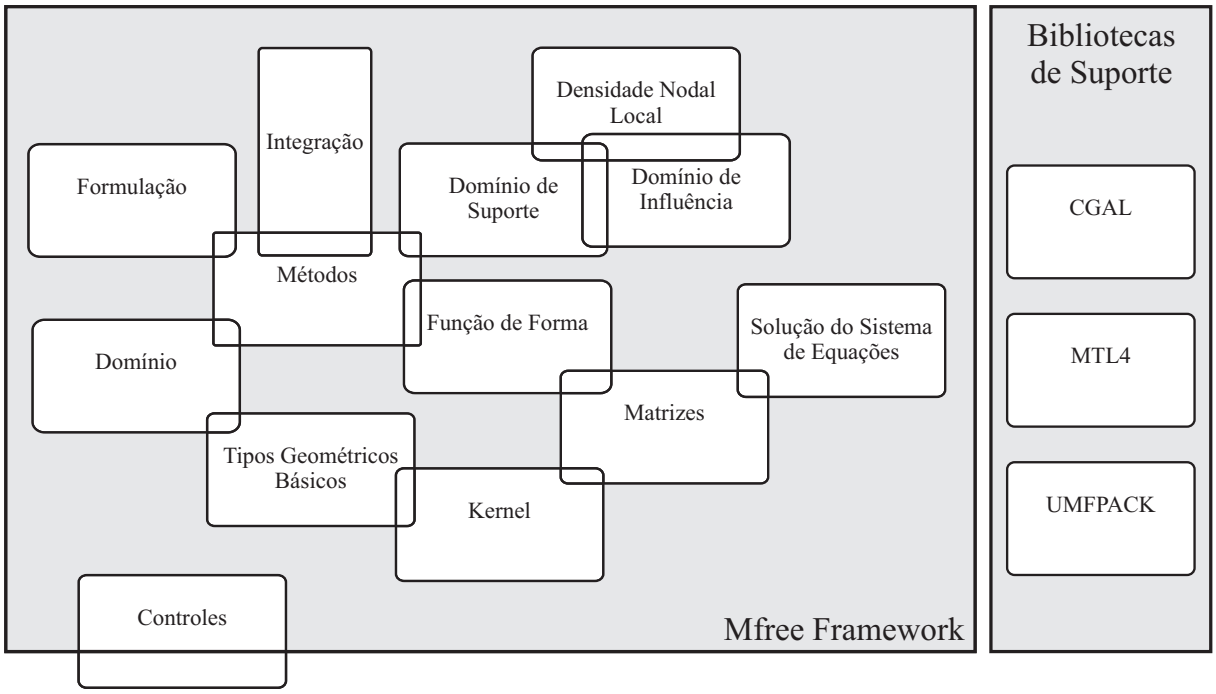


Figura 4.1: Estrutura do MFree Framework. Divisão em módulos de acordo com as funcionalidades.

De maneira geral, pode-se decompor o MFree Framework em alguns módulos com base em suas funcionalidades, como mostra a Figura 4.1. Os módulos não são completamente

independentes e trabalham em conjunto muitas vezes. O *Kernel* é a parte mais básica do *framework* e possui o papel principal de fornecer os tipos de representação numérica para coordenadas no espaço, para os valores das funções de forma e do sistema global de equação. Também fornece os *tipos geométricos básicos* como pontos, círculos, vetores, segmentos, arcos, dentre outros. Através do Kernel, por exemplo, é informado se a solução para o problema a ser resolvido envolve números reais ou complexos.

O módulo *métodos* representa os métodos sem malha e tem acesso ao *domínio* do problema, ao *domínio de suporte* utilizado, às *funções de forma* dos nós, ao procedimento de *integração* empregado e à *formulação* física do problema. O módulo *densidade nodal local* é responsável por determinar a concentração de nós em uma determinada região do domínio, calculando, para isso, as distâncias nodais locais segundo critérios apresentados na Seção 4.7.

Ao final do processo de montagem do sistema global de equações pelo método sem malha, é necessário solucioná-lo. O módulo *solução do sistema de equações* faz este papel. Já *matrizes* oferece os tipos para as representações matriciais, usadas tanto pelo sistema global de equações quanto para cálculo das funções de forma.

Os controles têm acesso a todos os módulos do *framework* e determinam o fluxo de execução principal do *framework*, isto é, determinam os passos a serem executados desde o ínicio até o fim do processo de solução de um problema. Os controles coordenam a criação e configuração dos objetos e se encarregam de efetuar as chamadas de funções necessárias. Constituem a interface entre o *framework* e aplicação escrita pelo desenvolvedor. Desse modo, a aplicação envolve-se apenas com a *configuração* do problema e do método sem malha utilizado, enquanto o *framework* faz todo o resto do trabalho. A Seção 4.15 apresenta, em detalhes, os tipos de controles implementados e suas funcionalidades.

Basicamente, para construir uma aplicação utilizando o MFree Framework, o desenvolvedor deve fornecer os tipos a serem utilizados pelo método sem malha, fazendo isso através de uma *traits* para o controle. Esses tipos estão relacionados aos definidos por cada um dos módulos da Figura 4.1, os quais são apresentados com maiores detalhes nas seções seguintes. Além disso, o desenvolvedor deve implementar um dos controles disponíveis e configurar as estruturas de dados de acordo com o problema e com o método sem malha.

A Figura 4.2 mostra o fluxo de execução do MFree Framework de forma simplificada. Observe que o controle é implementado em partes pela aplicação, que faz uma chamada ao método `execute()`, sendo este o ponto de entrada para o *framework*. A partir daí, o fluxo de controle é determinado exclusivamente pelo *framework* que, após o pós-processamento, retorna o fluxo para a aplicação. Observe, também, que a classe base para o controle é parametrizada pela *traits* implementada na aplicação e que fornece os tipos usados pelo método sem malha, a qual deve ser um modelo de `ControllerTraits`. Os passos da Figura 4.2 estão diretamente relacionados aos módulos da Figura 4.1, isto é, em cada passo um dos módulos é criado para que possa ser usado quando preciso, como mostra-se a seguir.

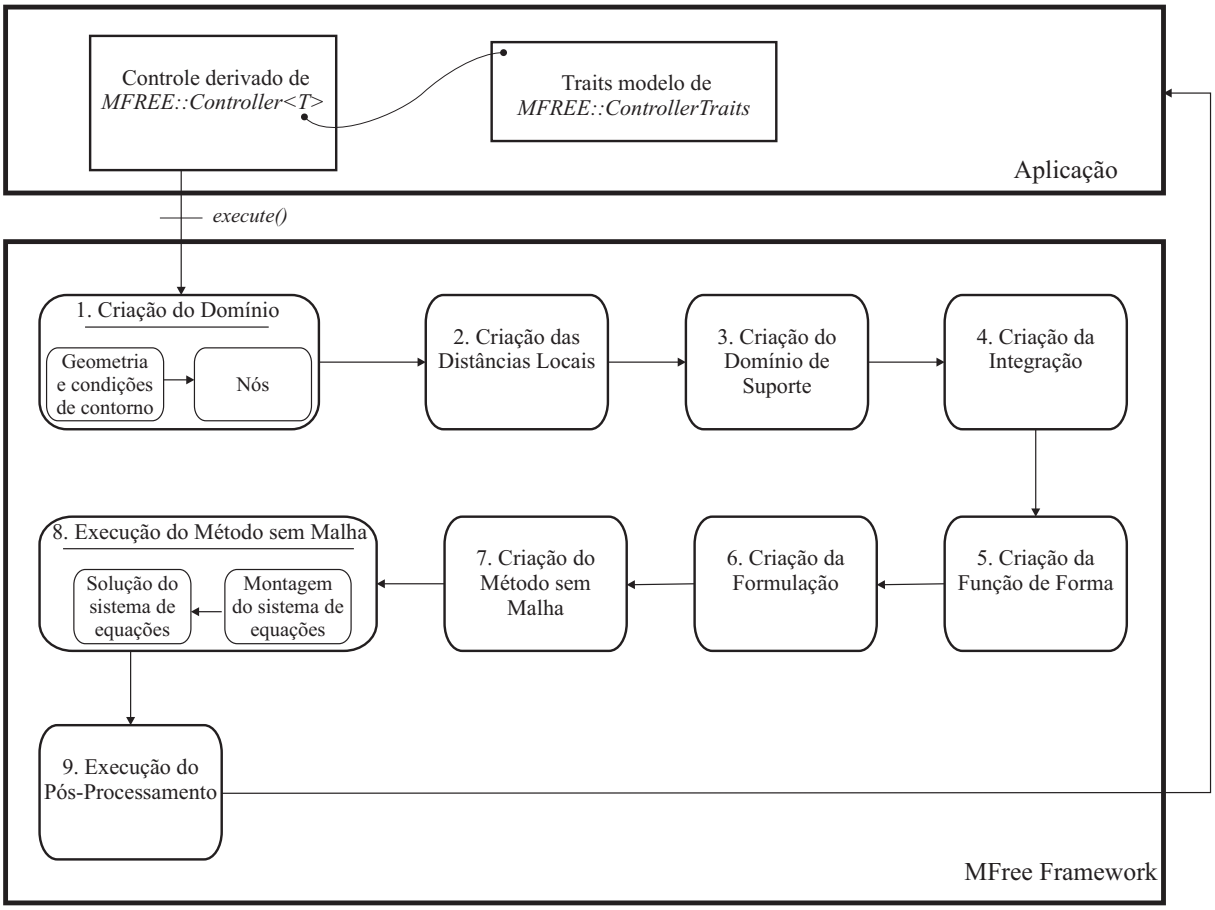


Figura 4.2:
Fluxo de controle do MFree Framework. `MFREE::Controller<T>` é a classe template base para todos os controles e é parametrizada pela *traits*, modelo de `MFREE::ControllerTraits`, que fornece os tipos para o método sem malha a ser utilizado. Note que a chamada ao método `execute()` do controle pela aplicação transfere o fluxo de controle para o *framework*, que o devolve após a execução do pós-processamento.

O primeiro passo executado é a criação do *domínio* do problema, em que o cliente entra com os dados da geometria, com as condições de contorno e com os nós. A geometria pode ser composta de segmentos e curvas e por múltiplos materiais. Além disso, qualquer tipo de condição de contorno pode ser utilizada desde que sua implementação esteja de acordo com o conceito `PointFunction`, descrito na Seção 4.4. De forma resumida, `PointFunction` representa uma função que recebe um ponto e retorna um valor por meio da função membro `eval(...)`. É usada para representar condições de contorno e propriedades físicas de materiais. As estruturas usadas para constituir o domínio do problema são detalhadas na Seção 4.6.

No segundo passo, o *framework* encarrega-se de determinar as distâncias locais entre nós, ligadas à *densidade nodal local* no domínio, de acordo com a Seção 4.7. Em seguida, as estruturas para *domínio de suporte*, *integração* e *função de forma* são criadas, sendo o cliente responsável por informar as configurações de cada uma delas, como ordens de integração numérica, ordem da base polinomial para a construção das funções de forma (quando aplicável), fatores de escalonamento para o domínio de suporte, dentre outros.

O domínio de suporte pode ser calculado de três maneiras: estipulando um formato específico para ele (normalmente, retangular ou circular), baseando-se em domínios de influência ou em um certo número de vizinhos mais próximos. O programador interessado em criar um novo tipo de domínio de suporte deve modelar o conceito `SupportDomain`, isto é, deve disponibilizar, na interface do tipo criado, as funções membro requisitadas por `SupportDomain`. A Seção 4.9 detalha os conceitos envolvidos com domínio de suporte, os tipos existentes na versão atual do *framework* e os procedimentos de busca utilizados.

Os conceitos relativos a funções de forma no MFree Framework são mostrados na Seção 4.10. Atualmente, é possível construí-las usando MLS, funções de Shepard, PIM, RPIM e RPIMp (descritos na Seção 3.2). Outros tipos de funções de forma podem ser aplicadas. Para isso, a implementação deve modelar o conceito `ShapeFunction` que, basicamente, fornece os métodos para calcular as funções e suas derivadas em um determinado ponto.

A integração da forma fraca é feita numericamente devido a presença de funções cujas integrais não possuem expressão analítica, e a quadratura gaussiana[‡] é empregada. Duas

[‡] O procedimento de integração por quadratura de Gauss é mostrado em [50]

formas de integração são implementadas na versão atual do *framework*: por malha de fundo e por subdomínios locais. Caso o desenvolvedor deseje utilizar um novo procedimento de integração, deve criar um tipo que modele o conceito `Integration`, que fornece *iterators* para os pontos de integração no domínio, nos contornos e na interfaces entre regiões do problema. Detalhes sobre integrações no MFree Framework são apresentados na Seção 4.11.

Retornando às etapas do fluxo de controle do *framework* da Figura 4.2, no sexto passo, a *formulação* para o problema é criada. Seu papel principal é efetuar a montagem do sistema global de equações de acordo com a formulação física do problema. O *framework* disponibiliza formulações cuja forma forte é governada por uma equação diferencial parcial de *Poisson*. Entretanto, qualquer formulação pode ser implementada desde que satisfaça os requisitos impostos pelo conceito `Formulation`, mostrado na Seção 4.13.

Os passos sete e oito são executados completamente pelo *framework*. A partir das estruturas de dados criadas e configuradas nos passos anteriores, constrói-se o objeto para o *método sem malha*, o qual é executado após isso: o sistema global de equações é montado e posteriormente resolvido. A versão atual do *framework* disponibiliza os métodos EFG, MLPG, NPIM, LPIM e MLPG Misto (descritos na Seção 3.3). Outros métodos podem ser implementados modelando o conceito `Method`. Detalhes sobre métodos sem malha no *framework* são apresentados na Seção 4.12.

Por fim, executa-se o pós-processamento. A implementação padrão do MFree Framework calcula a solução aproximada nos pontos informados pelo cliente através de um arquivo de entrada (com formato descrito no Apêndice A.3), gravando-a em um arquivo de saída (de acordo com formato descrito no Apêndice A.5). O desenvolvedor interessado em outras operações de pós-processamento pode fazê-las reimplementando o método `post_processing(...)` pertencente ao controle.

As seções seguintes apresentam, em maiores detalhes, os módulos do MFree Framework destacados na Figura 4.1 com os respectivos conceitos e tipos implementados.

4.2 Bibliotecas de Suporte

Nos métodos sem malha estão presentes problemas de geometria computacional como buscas espaciais, interseção de polígonos, dentre outros. Utiliza-se, para isso, a biblioteca *Computational Geometry Algorithms Library* (CGAL) [10]. A CGAL é construída utilizando programação genérica e possui uma coleção vasta de algoritmos e estruturas de dados muito eficientes, além de ser uma biblioteca de código fonte aberto.

Os cálculos de funções de forma envolvem operações de matrizes, e o sistema de equações global é um sistema esparso. No MFree Framework, as representações e operações matriciais são todas feitas por meio da *Matrix Template Library 4* (MTL4) [25]. A MTL4 é construída, também, com base em programação genérica com objetivo de ser uma biblioteca de alto desempenho.

Para resolver o sistema equações lineares, adota-se a *Unsymmetric Multifrontal Sparse LU Factorization Package* (UMFPACK) [1], capaz de resolver sistemas esparsos de forma bastante eficiente. Em testes realizados pelo autor deste trabalho, sistemas de ordem $10^4 \times 10^4$ foram resolvidos em poucos segundos (aproximadamente 2 a 3 segundos) em um Laptop Intel Core 2 Duo com 3GB de RAM.

4.3 Matrizes

No *framework*, matrizes são usadas no processo de construção das funções de forma e no sistema de equações global. Dois tipos de matrizes devem estar disponíveis: densa e esparsa. A matriz esparsa é utilizada, basicamente, no sistema de equações global do método sem malhas. Os tipos matriciais são declarados através de uma *traits* que modela o conceito `MatrixTraits` (Figura 4.3). `Matrix` representa uma matriz densa, `Sparse_matrix` uma matriz esparsa e `Vector` um vetor (matriz linha ou coluna).

O MFree Framework utiliza a biblioteca MTL4 para representar e operar com matrizes. Para isso, disponibiliza a classe `Mtl_matrix_traits`, que implementa o conceito

`MatrixTraits`. `Mtl_matrix_traits` é parametrizada por `Number`, que representa o tipo numérico dos elementos armazenados nas matrizes e no vetor.

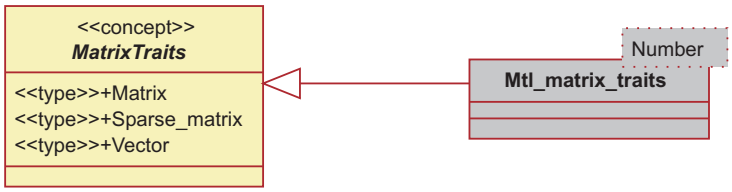


Figura 4.3: Diagrama UML: *Traits* para matrizes.

4.4 Kernel

O Kernel é a parte central e mais básica do *framework*. Contém a definição dos tipos numéricos usados pelas coordenadas espaciais, funções de forma e sistema de equações global. Define, também, o Kernel usado pela CGAL, os tipos geométricos básicos e as *traits* para matrizes, como mostra a Figura 4.4. É através do Kernel que as informações destes tipos são obtidas por todas as estruturas de dados do *framework*. Esse mecanismo é possível pois as estruturas de dados são parametrizadas pelo Kernel ou por *traits*. Por sua vez, todas as *traits* são parametrizadas pelo Kernel.

`FT`, `NT` e `ST` representam os tipos numéricos para as coordenadas geométricas do espaço, para as funções de forma e para o sistema de equações, respectivamente. Dessa forma, o *framework* permite resolver problemas cuja solução é complexa. Basta, para tal, fazer `ST` representar um número complexo. Tipos complexos em C++ são implementados pela classe template `complex<T>`.

`CGAL_Kernel` é o Kernel usado pela CGAL em suas estruturas de dados e algoritmos. `Matrix_traits` e `System_matrix_traits` são as *traits* de matrizes para as funções de forma e para o sistema de equações, nessa ordem. `Point_function` é o tipo para funções $f(\mathbf{x})$ que recebem um ponto e retornam um valor do tipo `ST` por meio da função `eval(...)`, como pode ser visto na Figura 4.7. São utilizadas para impor condições de contorno e descrever propriedades físicas de materiais.

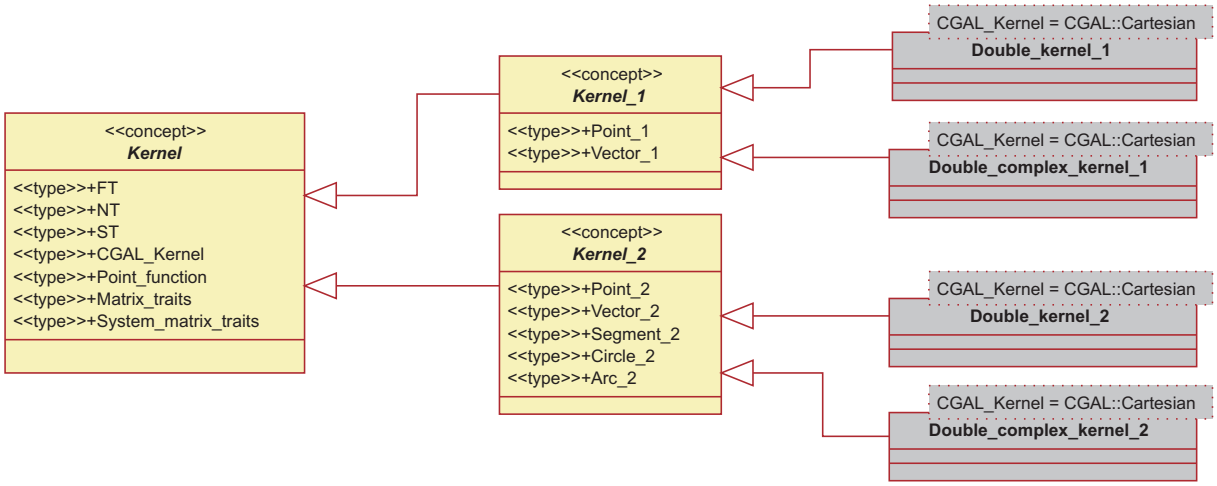


Figura 4.4:
Diagrama UML: Kernel. `Kernel_1` e `Kernel_2` refinam `Kernel`.

A Figura 4.4 ilustra o conceito `Kernel` e as classes que o implementam. Observe que `Kernel_1` e `Kernel_2` refinam `Kernel`, ou seja, devem satisfazer os requisistos deste. Além disso, definem os tipos geométricos básicos presentes nos respectivos espaços. As classes `Double_kernel_1` e `Double_complex_kernel_1` correspondem a Kernels no espaço unidimensional destinados a solucionar problemas cuja solução é real ou complexa, respectivamente, com `FT` e `NT` iguais a `double`. De forma análoga em duas dimensões, existem as classes `Double_kernel_2` e `Double_complex_kernel_2`.

4.5 Tipos Geométricos Básicos

Os tipos geométricos básicos são utilizados por praticamente todas as estruturas de dados do *framework*, mas principalmente para descrever a geometria do problema. São eles: pontos, vetores, segmentos, círculos e arcos. A Figura 4.5 mostra o diagrama UML com os tipos básicos existentes.

O conceito `Point` representa um ponto no espaço. Observe que ele deve satisfazer quatro conceitos: `IdItem`, `EqualComparable`, `LessComparable` e `RandomAccessCoordinate`. `IdItem` implica que os objetos devem possuir um identificador associado a cada um. Um tipo satisfaz `EqualComparable` se é possível efetuar o teste de igualdade entre dois obje-

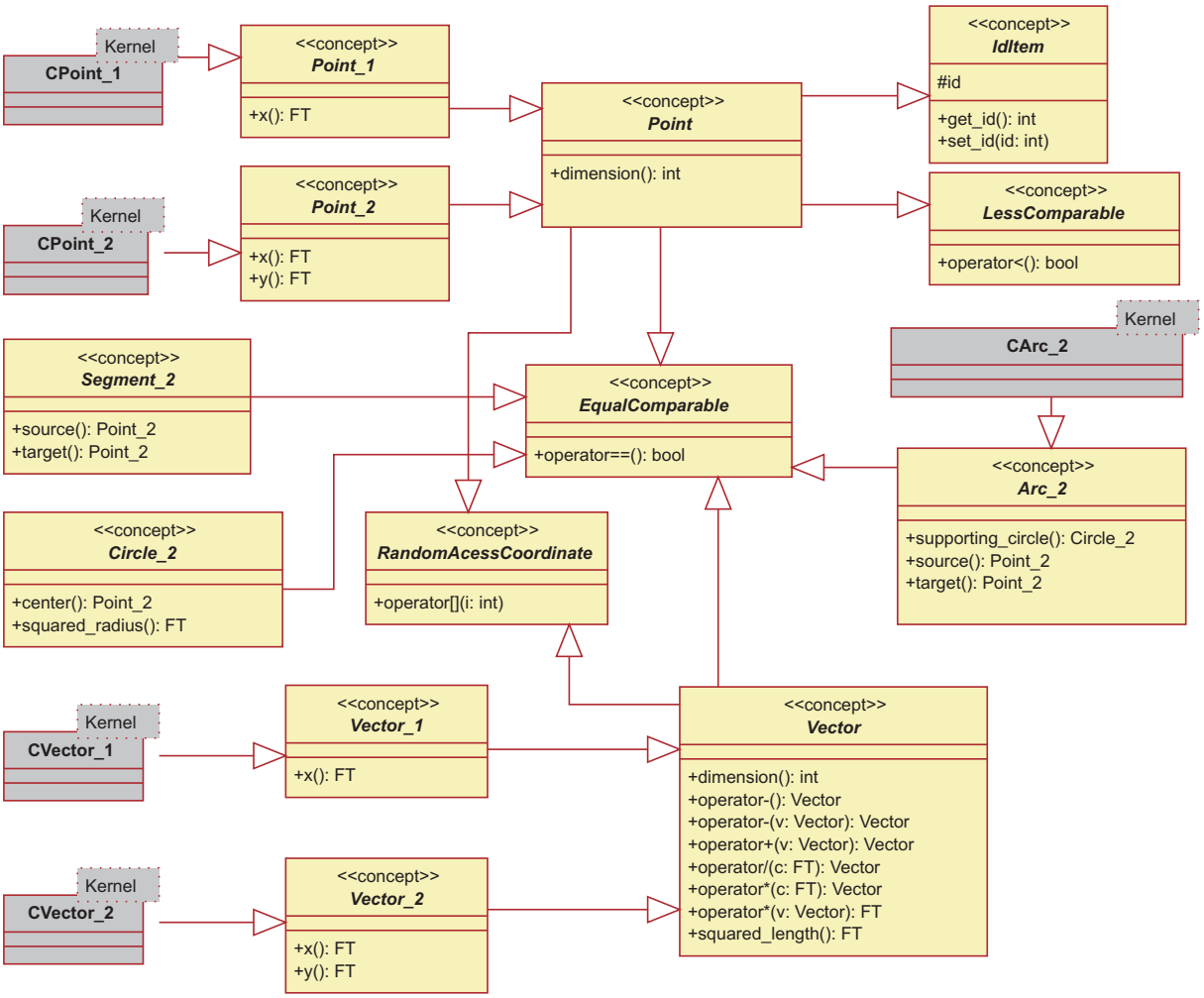


Figura 4.5: Diagrama UML: Tipos geométricos básicos.

tos desse tipo. De modo semelhante, `LessComparable` exige que seja possível efetuar o teste de comparação *menor que* entre os dois objetos. Por fim, `RandomAccessCoordinate` garante acesso às coordenadas cartesianas de um objeto através do operador colchetes, da forma como é feita com *arrays*. Observe que `Point_1` e `Point_2` refinam `Point` e representam, respectivamente, pontos nos espaços uni e bidimensional. As implementações para pontos são feitas pelas classes `CPoint_1` e `CPoint_2`.

Vetores no espaço são descritos pelo conceito `Vector` de forma similar a pontos e são implementados por `CVector_1` e `CVector_2`. `Circle_2` representa um círculo em 2D e usa-se a implementação da CGAL através da classe `CGAL::Circle_2`. `Segment_2` representa um segmento em 2D e usa-se a implementação disponível na CGAL por meio de `CGAL::Segment_2`. Arcos em 2D são descritos por `Arc_2` e implementados pela classe

`CArc_2`. Note que todos os tipos geométricos básicos são parametrizados por `Kernel` para obterem as informações que necessitam.

4.6 Representação do Domínio

O domínio do problema é representado por sua geometria, condições de contorno impostas e distribuição nodal. A geometria é dividida em regiões e cada região está associada a um material que descreve as propriedades físicas do meio. Em uma dimensão, a geometria é formada por um segmento no eixo x. Em duas dimensões, pode ser formada por segmentos, círculos e arcos no plano xy. A Figura 4.6 mostra os elementos relacionados à composição do domínio.

As condições de contorno são implementadas pelas classes `Boundary_condition_1` e `Boundary_condition_2`. Note que os valores das condições de contorno são dados pelo conceito `Point_function`, que representa uma função de ponto (Figura 4.7). Emprega-se o polimorfismo dinâmico através da classe base `Point_base_function` para que qualquer função criada pelo desenvolvedor da aplicação possa ser utilizada, permitindo condições de contorno de qualquer tipo nos problemas. Apesar dessa estratégia estar ligada ao paradigma de orientação à objetos e seu emprego exigir o uso de funções virtuais (o que normalmente reduz o desempenho computacional [55]), isto não constitui um problema pois o número de condições de contorno definidas no problema é relativamente pequeno, e o desempenho não é prejudicado. De maneira semelhante às condições de contorno, as propriedades de materiais são definidas usando, também, funções de ponto.

As classes `Domain_1` e `Domain_2` representam domínios em 1D e 2D, respectivamente. As estruturas de dados foram construídas para ser possível, a partir do domínio do problema, iterar sobre as regiões, sobre a distribuição nodal, sobre as curvas da geometria, interfaces de materiais e acessar as condições de contorno. Através das regiões é possível acessar as interfaces com regiões vizinhas, e das interfaces obtêm-se as regiões envolventes. Todos os acessos são feitos por meio de `iterators` no estilo STL, conferindo flexibilidade à implementação.

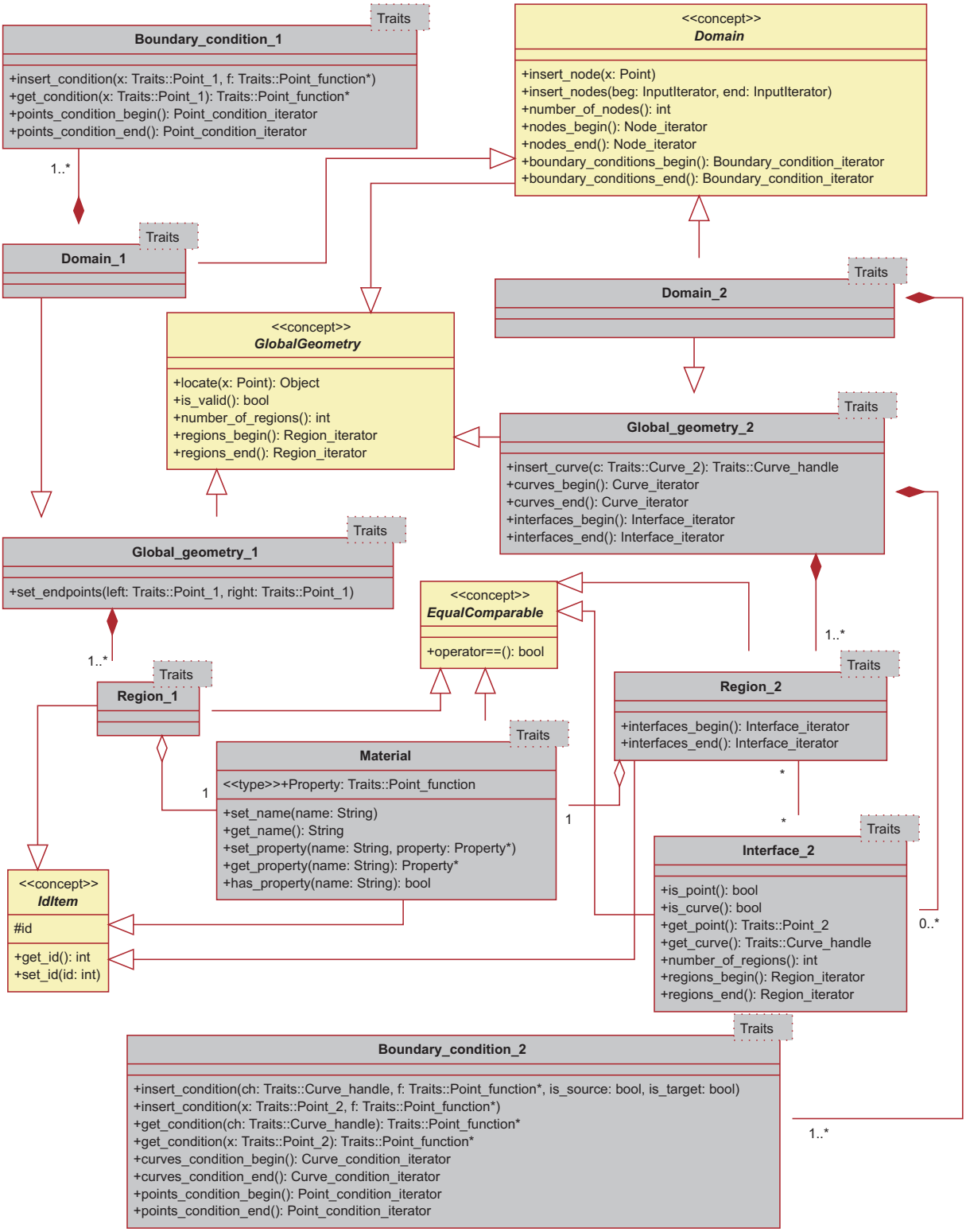


Figura 4.6: Diagrama UML: Domínio.

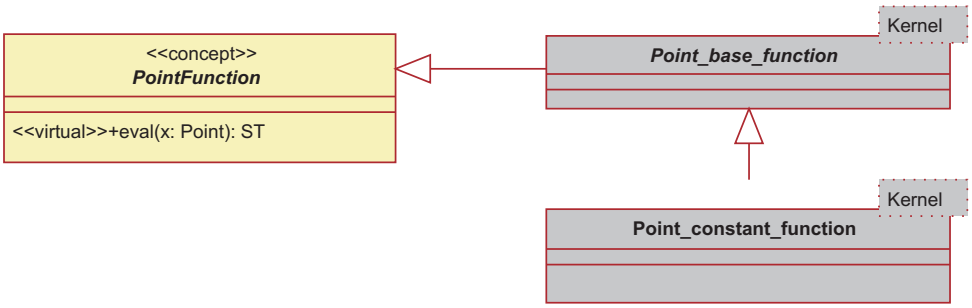


Figura 4.7: Diagrama UML: Função de ponto.

A Figura 4.8 mostra as *traits* para domínios. `Domain_traits_1` fornece os tipos para `Domain_1`. Já em duas dimensões, `Domain_segment_traits_2` associa-se a um domínio com geometria composta exclusivamente de segmentos de reta, enquanto `Domain_curve_traits_2` permite que haja segmentos de reta, círculos e arcos de círculo. Dessa forma, o programador utiliza apenas a classe `Domain_2` e a parametriza com a *traits* de acordo com a geometria do problema.

É possível, também, realizar pesquisas de localização de pontos no domínio. Para a busca, usa-se uma estrutura de dados da geometria computacional chamada *Mapa Trapezoidal* [8] cuja implementação está presente na CGAL, por ser mais eficiente que a busca por força bruta. A localização de pontos em Mapas Trapezoidais executa em tempo $O(\log n)$, onde n é o número de arestas da geometria. Todavia, é permitido utilizar outras estratégias de busca, informando seu tipo no campo `Point_location_strategy` da *traits* do domínio.

4.7 Densidade Nodal Local

Em métodos sem malha, a densidade nodal local está ligada à distância entre os nós numa determinada vizinhança. É representada pelo conceito `LocalDistance` (Figura 4.9) e usada para calcular domínios de suporte, de influência e de quadratura.

A distância local pode ser fixa ou estimada pelo *framework*. A primeira aplica-se a distribuições uniformes de nós e é representada pela classe `Fixed_local_distance`, sendo o desenvolvedor responsável por informar a distância predeterminada. A segunda é mais

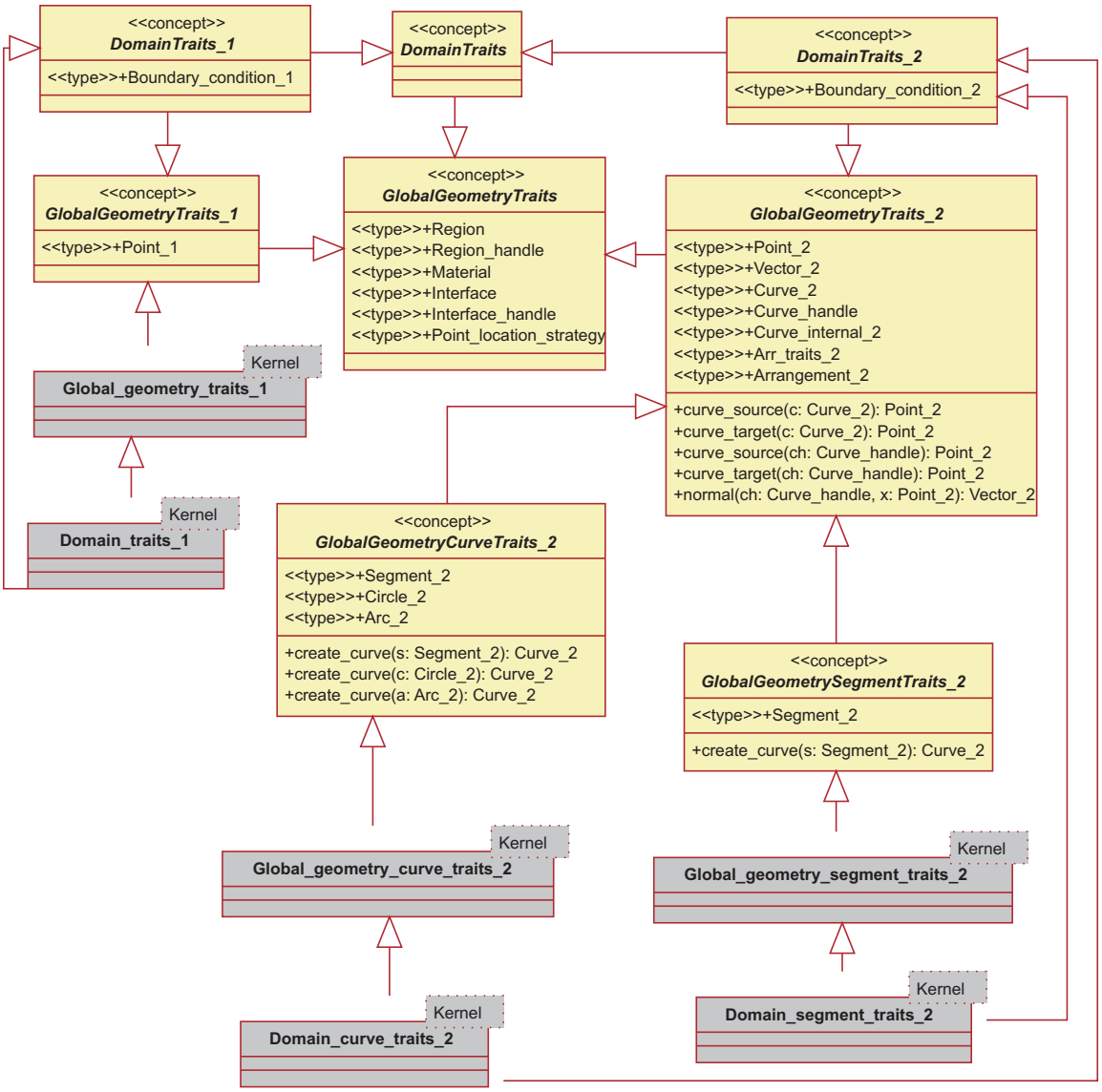


Figura 4.8: Diagrama UML: *Traits* para domínio.

geral e pode ser usada para distribuições nodais uniformes ou não. É representada pelo conceito `EstimatedLocalDistance`. Atualmente, o *framework* disponibiliza uma implementação para `EstimatedLocalDistance` através da classe `Knn_local_distance`. A estimativa da distância local é feita através do algoritmo *K-Nearest-Neighbour* (KNN) [16] de forma semelhante ao procedimento realizado em [47]. O KNN é um algoritmo que objetiva pesquisar, dado um ponto $\mathbf{x}$ e um conjunto de pontos P no espaço, os k pontos de P mais próximos de $\mathbf{x}$. Portanto, `Knn_local_distance` estima a distância local a partir dos k nós mais próximos em uma vizinhança. A CGAL oferece uma implementação do

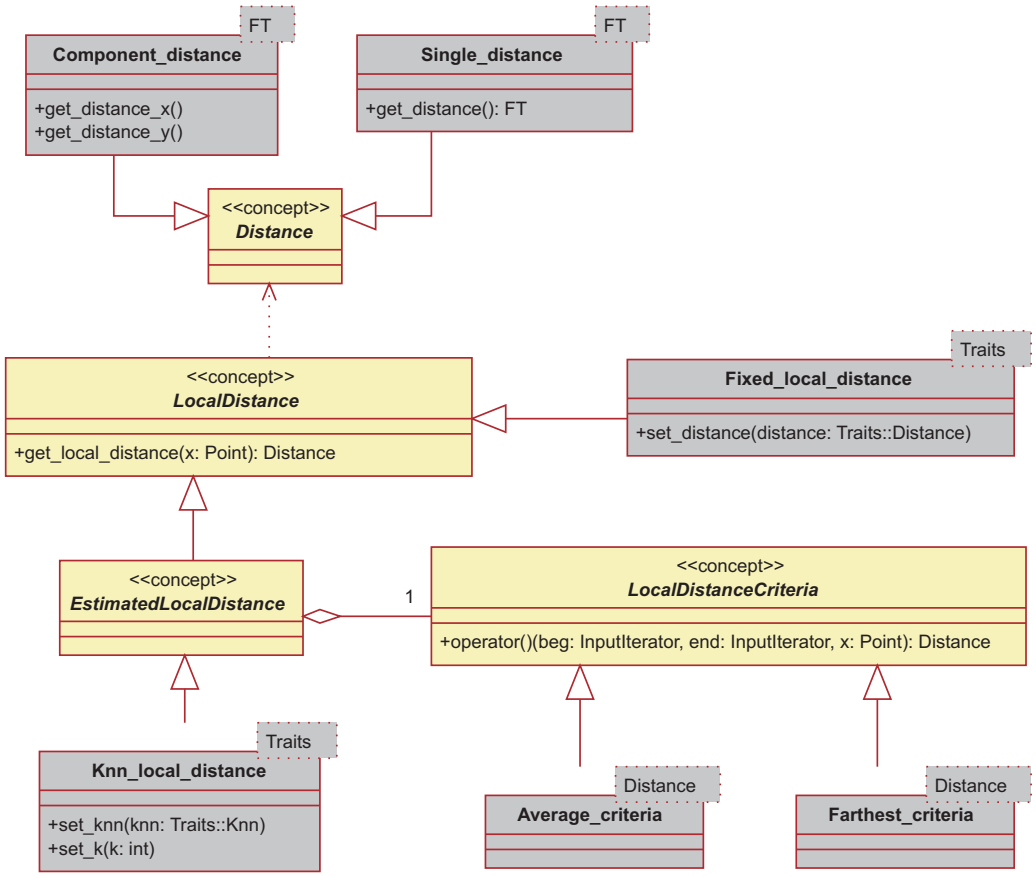


Figura 4.9: Diagrama UML: Densidade nodal local.

KNN por meio de *Kd-trees* [8], que possuem tempo de construção $O(n \log n)$, espaço de armazenamento $O(n)$ e tempo de busca $O(\sqrt{n} + k)$, onde n é o número total de pontos.

Note, na Figura 4.9, que há um critério para estimar distâncias locais, representado pelo conceito `LocalDistanceCriteria`. O *framework* disponibiliza dois critérios: (1) distância local média e (2) distância local máxima, correspondentes aos *functions objects* `Average_criteria` e `Farthest_criteria`, respectivamente. No primeiro caso, toma-se a média das distâncias entre os nós locais selecionados. No segundo, toma-se a distância máxima.

Dois tipos de distância estão disponíveis. A mais comum é representada por `Single_distance` e definida como a distância euclidiana entre dois pontos. É usada, geralmente, com subdomínios (influência, suporte e quadratura) circulares (Equações 3.2, 3.4 e 3.114). O outro tipo, dado por `Component_distance`, representa as distâncias

euclidianas entre as componentes de dois pontos, e usa-se mais frequentemente com subdomínios retangulares (Equações 3.3, 3.5 e 3.115).

As classes *traits* que modelam `LocalDistanceTraits` são mostradas na Figura 4.10. Distâncias locais fixas e estimadas são passíveis de trabalhar tanto com `Single_distance` quanto com `Component_distance`. Por exemplo, `Fixed_local_single_distance_traits` é a traits que combina `Fixed_local_distance` e `Single_distance`. Por outro lado, `Knn_local_component_distance_traits_2` combina `Knn_local_distance` e `Component_distance`. Observe que as *traits* para distâncias locais estimadas são parametrizadas por `Criteria` que deve ser um modelo de `LocalDistanceCriteria`.

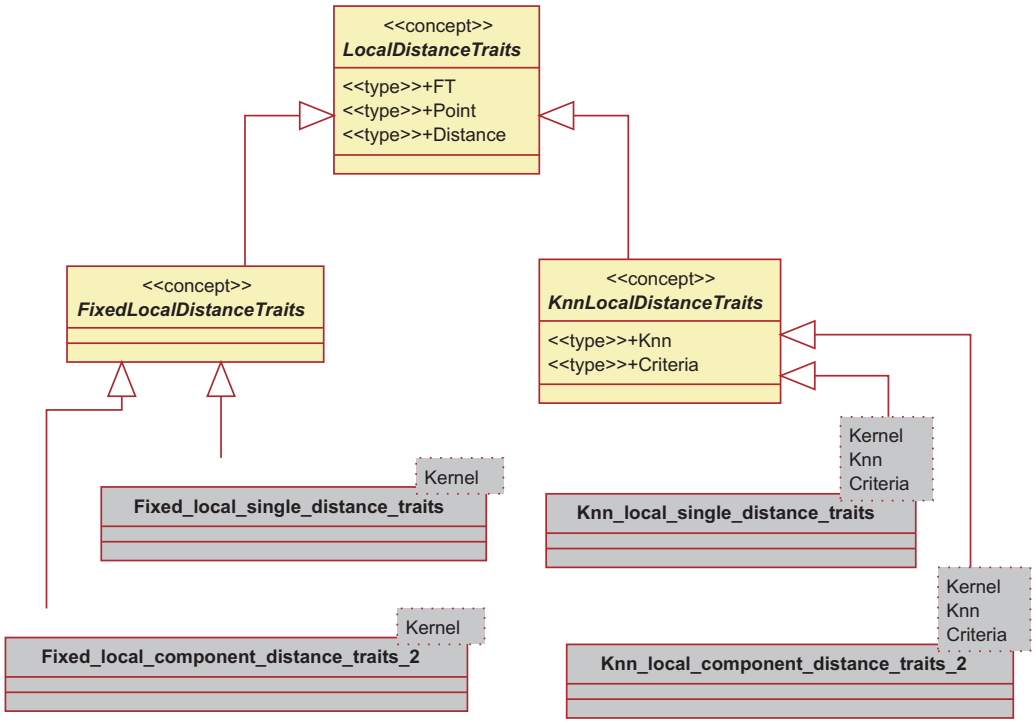


Figura 4.10: Diagrama UML: *Traits* para densidade nodal local.

4.8 Domínio de Influência

Domínios de influência são representados por dois conceitos: `Radius` e `LocalDomain` (Figura 4.11). O primeiro está ligado à dimensão do domínio de influência, o segundo ao formato e auxilia as buscas por domínio de suporte, como é visto em detalhes na Seção 4.9.

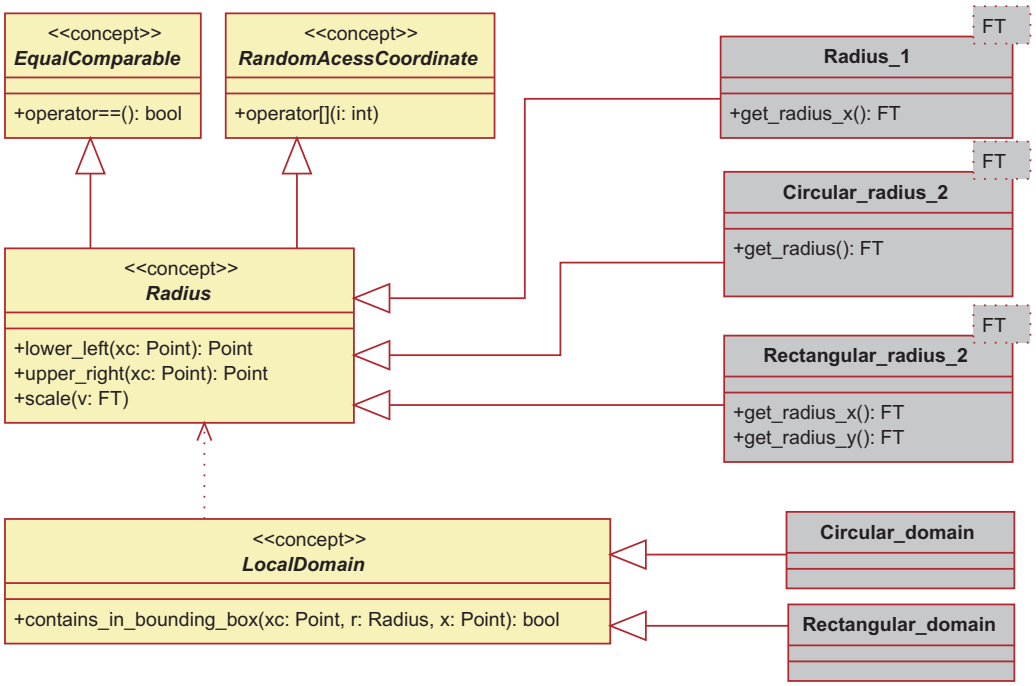


Figura 4.11: Diagrama UML: Domínio de influência.

`Radius` representa o raio de influência. Em uma dimensão é implementado pela classe `Radius_1`. Em duas dimensões, pode ser pode ser circular (classe `Circular_radius_2`) ou retangular (classe `Rectangular_radius_2`), de acordo com as Equações 3.4 e 3.5. Um raio de influência retangular possui componentes independentes nas direções x e y, enquanto um circular possui apenas a componente radial.

As buscas espaciais realizadas pelo domínio de suporte baseado em domínios de influência, utilizando as estruturas de dados de geometria computacional escolhidas (Seção 4.9), são buscas que utilizam regiões ortogonais. Para que domínios de influência de qualquer forma possam ser usados, `LocalDomain` define o método `contains_in_bounding_box(...)` que

testa se um ponto `xc` realmente pertence ao domínio de influência, dado que `xc` está contido na caixa delimitadora do subdomínio, a qual corresponde justamente à uma região ortogonal da busca. No *framework*, domínios de influência circulares são representados por `Circular_domain`, e retangulares por `Rectangular_domain`.

4.9 Domínio de Suporte

Domínio de suporte é definido pelo conceito `SupportDomain`, mostrado na Figura 4.12. Observe que o conceito exige a implementação de dois métodos, que retornam os nós pertencentes ao domínio de suporte do ponto `x`. Na primeira versão do método, `x` pertence à região de *handle* `rh`. Na segunda versão, `x` pertence à interface dada por `ih`. Assim, é possível efetuar buscas pelos nós de suporte em domínios com múltiplas regiões e interfaces.

Todas as implementações de domínio de suporte no MFree Framework atendem o *critério de não penetração*, isto é, pontos pertencentes a uma região possuem nós de suporte apenas desta região e das interfaces que a envolvem. Pontos sobre uma interface possuem nós de suporte que estão nesta interface e em suas regiões vizinhas. Para que as buscas sejam efetuadas de forma eficiente em todo o domínio do problema, cria-se, para cada região, uma estrutura de busca contendo os nós pertencentes à região associada.

Três tipos de domínio de suporte estão disponíveis, atualmente, no *framework*: (1) com formatos específicos (`ShapeSupportDomain`), (2) baseado em domínios de influência (`InfluenceSupportDomain`) e (3) baseados nos k vizinhos mais próximos (`KNodesSupportDomain`). Todos os tipos possuem versões em uma e duas dimensões.

`ShapeSupportDomain` representa um domínio de suporte com um formato específico. Os nós de suporte de um ponto $\mathbf{x}$ são aqueles contidos dentro do subdomínio correspondente a $\mathbf{x}$, como mostra a Figura 4.13a. Note que $\mathbf{x}_1$ possui um domínio de suporte retangular que contém os nós 1, 2, 3 e 4. Já $\mathbf{x}_2$ possui domínio de suporte circular e seus nós de suporte são 5, 6 e 7. A Figura 4.15 mostra conceito para a *traits* usada

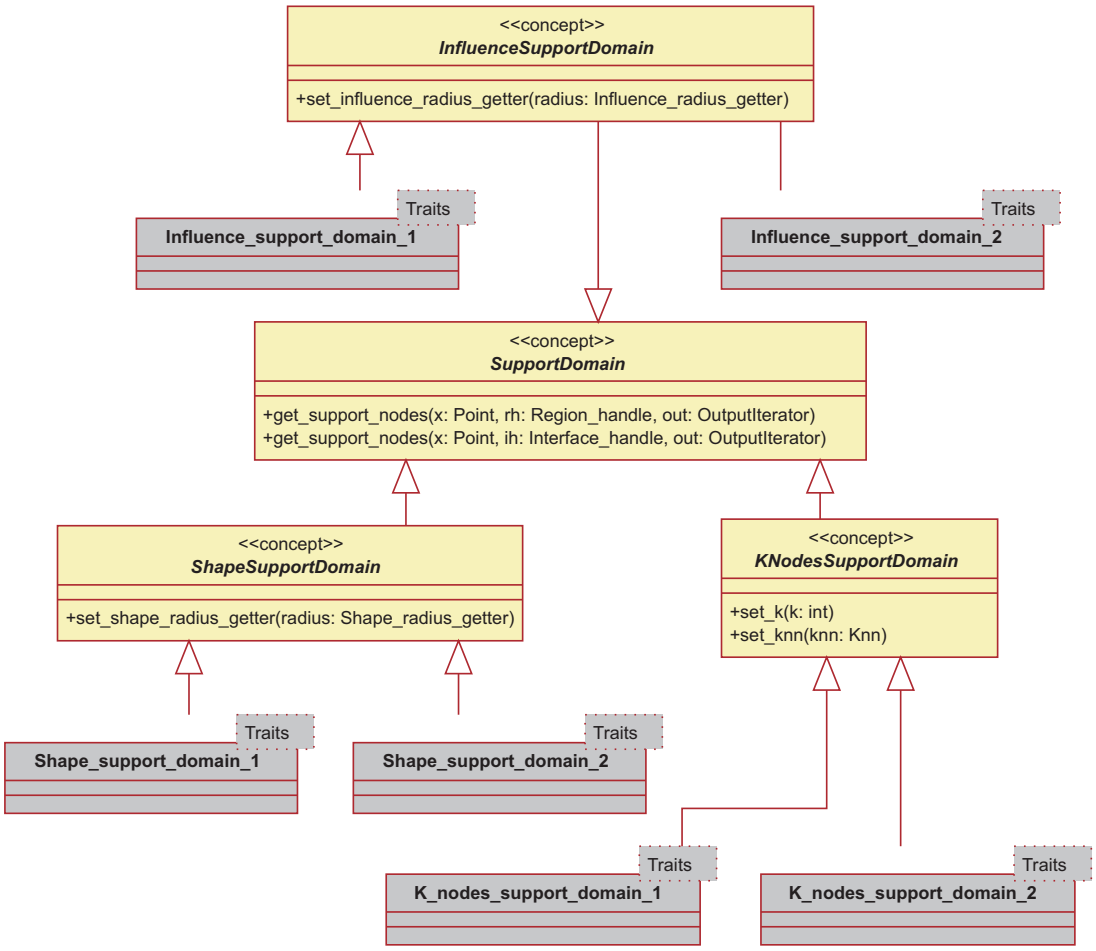


Figura 4.12: Diagrama UML: Domínio de suporte.

por `ShapeSupportDomain`, chamada `ShapeSupportDomainTraits`. Note que esta deve fornecer uma estrutura de busca, dada pelo campo `Search_structure`, capaz de efetuar *buscas intervalares ortogonais*. Buscas intervalares ortogonais são definidas da seguinte maneira: dado um conjunto de pontos P e uma região ortogonal r no espaço, retorne todos os pontos de P que pertençam à r. A estrutura de busca utilizada no *framework* é chamada *Range-tree* [8]. *Range-trees* estão disponíveis na CGAL e são construídas em tempo $O(n \log^{d-1} n)$, usam espaço $O(n \log^{d-1} n)$ e efetuam buscas em tempo $O(\log^d n + k)$, onde d é a dimensão do espaço, n o número total de pontos e k o número de pontos reportados pela busca. Uma estrutura de busca alternativa é a *Kd-tree* (Seção 4.7), a qual é utilizada para obter os domínios de suporte em [46]. Um domínio de suporte R_s é determinado em dois passos, mostrados na Figura 4.14. No primeiro, realiza-se uma busca intervalar ortogonal relativa à caixa delimitadora C de R_s, obtendo os nós contidos em C (Figura 4.14a). Após isso, testam-se quais nós retornados no passo an-

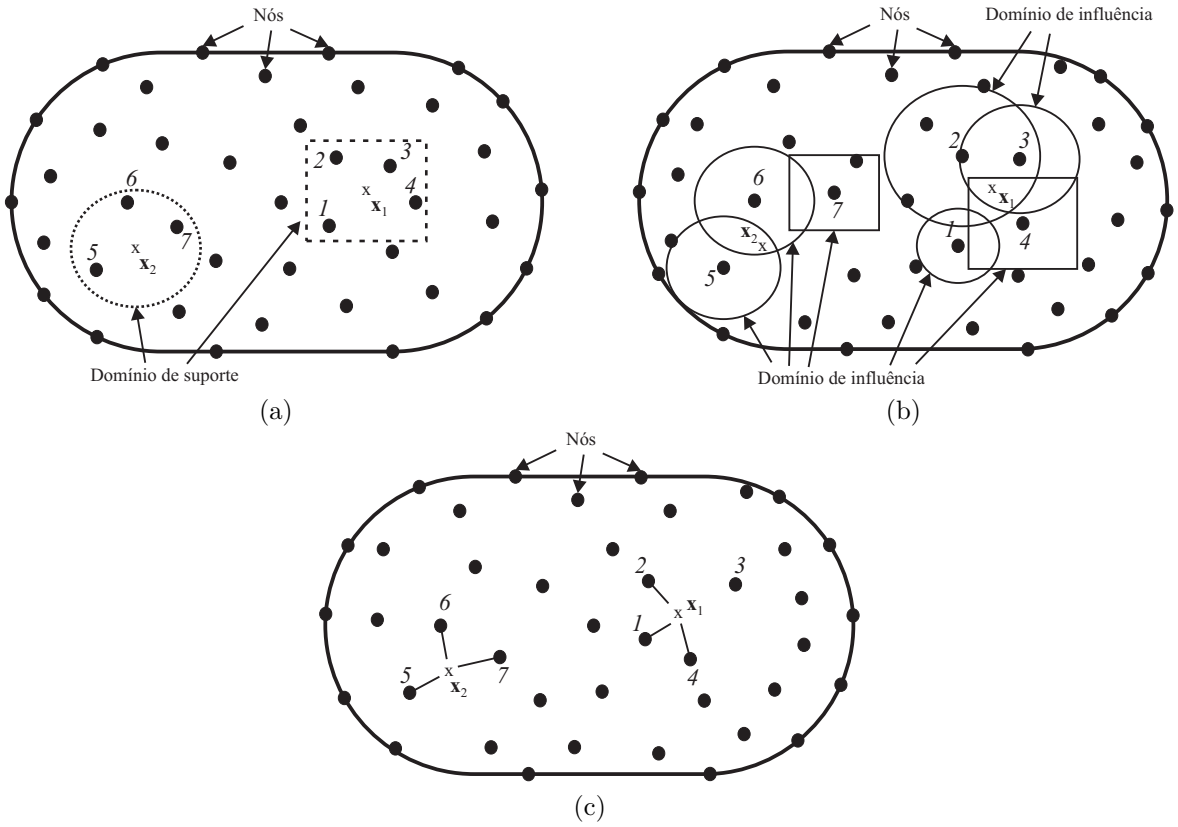


Figura 4.13: Tipos de domínio de suporte no MFree Framework. (a) Domínio de suporte com formato específico. Os nós 1, 2, 3 e 4 são suporte para o ponto $\mathbf{x}_1$; os nós 5, 6 e 7 são suporte para $\mathbf{x}_2$. (b) Domínio de suporte baseado em domínio de influência. Os nós 2, 3 e 4 são suporte para o ponto $\mathbf{x}_1$; os nós 5 e 6 são suporte para $\mathbf{x}_2$. (c) Domínio de suporte baseado no k nós vizinhos mais próximos, com $k = 3$. Os nós 1, 2 e 4 são suporte para o ponto $\mathbf{x}_1$; os nós 5, 6 e 7 são suporte para $\mathbf{x}_2$.

terior realmente pertencem à R_s (Figura 4.14b). Qualquer formato para domínios de suporte pode ser usado, desde que sejam fornecidos a caixa delimitadora e o teste se a forma contém um ponto. O *framework* implementa formatos circulares e retangulares para o subdomínio, através das *traits* `Circular_shape_support_domain_traits_2` e `Rectangular_shape_support_domain_traits_2` (Figura 4.15). As dimensões dos domínios de suporte são obtidas usando as Equações 3.2 e 3.3.

Domínio de suporte baseado em domínio de influência é representado pelo conceito `InfluenceSupportDomain` e é o mais comumente usado. Os nós de suporte de um ponto $\mathbf{x}$ são aqueles cujos domínios de influência contêm $\mathbf{x}$, como mostra a Figura 4.13b. Observe que os domínios de influência podem ter diferentes formatos e tamanhos. O domínio de suporte do ponto $\mathbf{x}_1$ é composto pelos nós 2, 3 e 4, enquanto 5 e 6 são su-

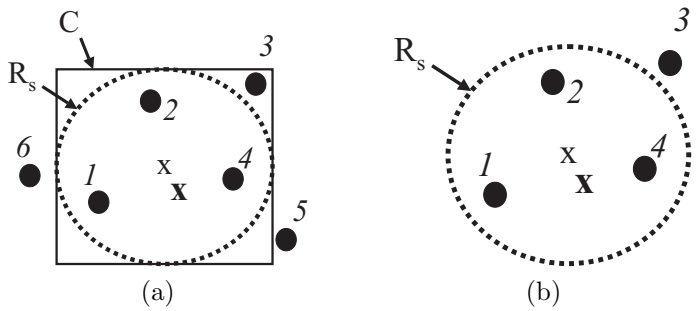


Figura 4.14: Determinação do domínio de suporte com formato específico. (a) Primeiro passo: efetua-se uma busca intervalar ortogonal com a caixa C delimitadora do domínio de suporte R_s. A busca retorna os nós 1, 2, 3 e 4. (b) Segundo passo: testa-se quais nós pertencem à R_s. Os nós de suporte são 1, 2 e 4.

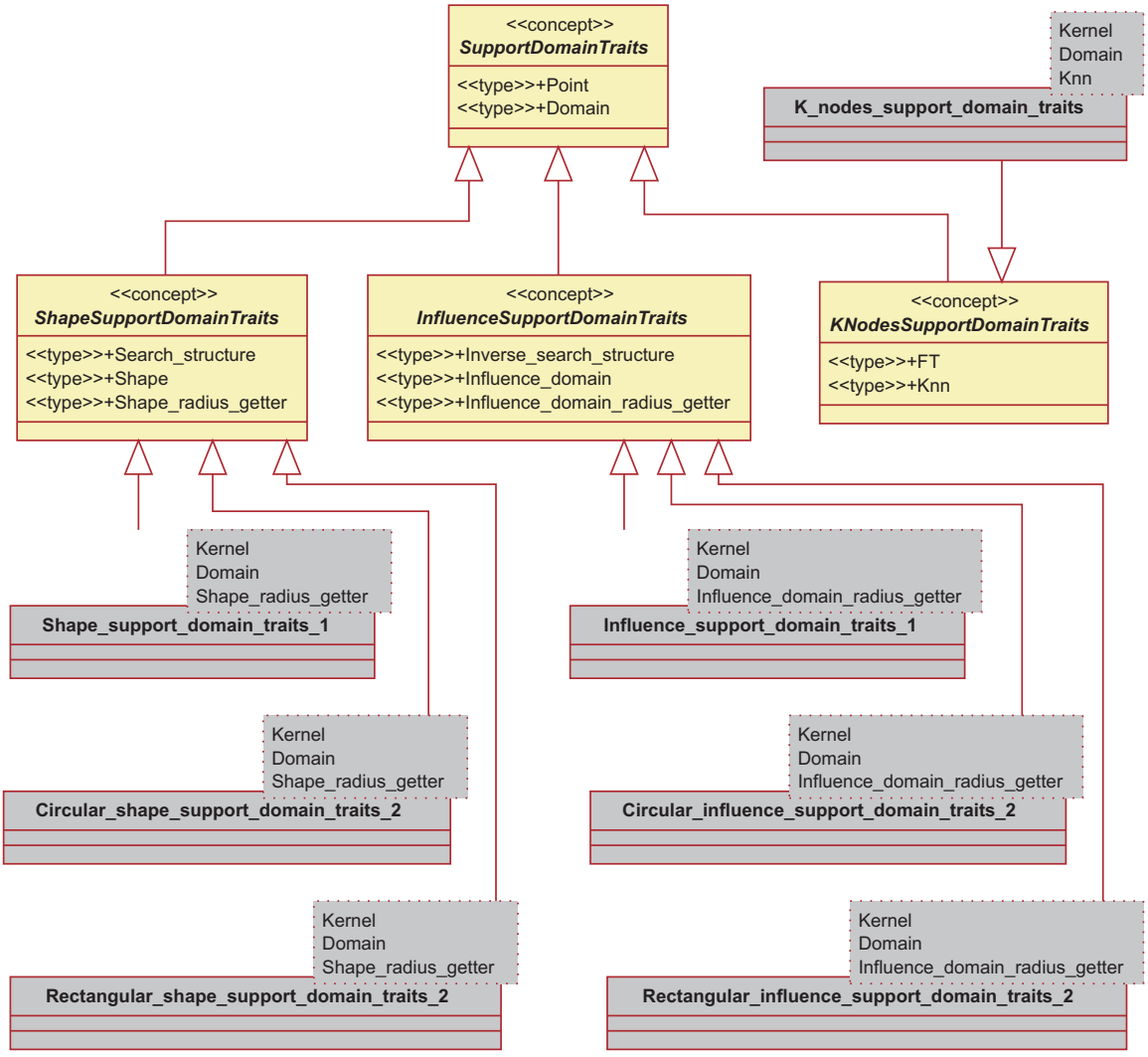


Figura 4.15: Diagrama UML: *Traits* para domínio de suporte.

porte para $\mathbf{x}_2$. A Figura 4.15 mostra o conceito `InfluenceSupportDomainTraits`, a *traits* usada por `InfluenceSupportDomain`, que deve fornecer uma estrutura de busca inversa (por meio do campo `Inverse_search_structure`), capaz de efetuar *buscas intervalares ortogonais inversas*. Estas buscas são definidas da seguinte forma: dado um conjunto de intervalos ortogonais S e um ponto p no espaço, retorne todos os intervalos de S que contenham p. No presente contexto, p refere-se ao ponto a se determinar o domínio de suporte e os intervalos de S correspondem aos domínios de influência dos nós, representados por suas caixas delimitadoras pois os intervalos são ortogonais. A estrutura de busca inversa utilizada é a *Segment-tree* [8]. *Segment-trees* são implementadas pela CGAL e possuem tempo de construção $O(n \log^d n)$, espaço de armazenamento $O(n \log^d n)$ e tempo de busca inversa $O(\log^d n + k)$, onde d é a dimensão do espaço, n o número total de pontos e k o número de intervalos reportados pela busca. De forma semelhante a `ShapeSupportDomain`, os nós de suporte de um ponto $\mathbf{x}$ são determinados em duas etapas, mostradas na Figura 4.16. Na primeira, efetua-se a busca intervalar ortogonal inversa, e os nós cujas caixas delimitadoras C dos domínios de influência R_i contêm $\mathbf{x}$ são retornados (Figura 4.16a). Após isso, verificam-se quais domínios de influência retornados no primeiro passo cobrem, realmente, o ponto $\mathbf{x}$ (Figura 4.16b). Com este procedimento, é possível utilizar domínios de influência de qualquer formato, desde que sejam fornecidos a caixa delimitadora e o teste se um ponto pertence ou não ao domínio local. Domínios de influência circulares e retangulares são implementados pelo *framework* por meio das *traits* `Circular_influence_support_domain_traits_2` e `Rectangular_influence_support_domain_traits_2`. Os raios de influência são calculados pelas Equações 3.4 e 3.5.

Por fim, o último tipo de domínio de suporte disponível é baseado em um certo número de nós vizinhos, representado pelo conceito `KNodesSupportDomain` (Figura 4.12). O domínio de suporte de um ponto $\mathbf{x}$ é determinado buscando os k nós mais próximos de $\mathbf{x}$, onde k é escolhido pelo cliente. Constitui um tipo simples de determinar os nós de suporte de um ponto. É útil quando deseja-se que todas as funções de forma sejam construídas exatamente com o mesmo número de nós ou quando as funções de forma não utilizam funções de peso ou RBFs, como no caso do PIM polinomial. A estrutura de busca utilizada pelo *framework* para esse tipo de domínio de suporte é o KNN, descrito na Seção 4.7. A Figura 4.13c mostra exemplos desse tipo de domínio de suporte com $k = 3$. Os nós 1, 2 e 4 são suporte para o ponto $\mathbf{x}_1$, enquanto 5, 6 e 7 são suporte para $\mathbf{x}_2$.

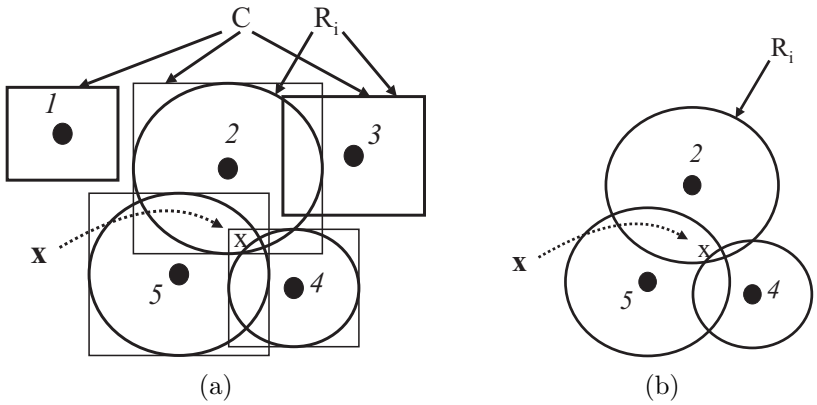


Figura 4.16: Determinação do domínio de suporte baseado em domínio de influência. (a) Primeiro passo: efetua-se uma busca intervalar ortogonal inversa para o ponto $\mathbf{x}$, com os intervalos dados por cada caixa C delimitadora dos domínios de influência R_i's. A busca retorna os nós 2, 4 e 5 com os respectivos domínios de influência. (b) Segundo passo: testa-se quais domínios de influência R_i contêm $\mathbf{x}$. Os nós de suporte são 2 e 5.

4.10 Funções de Forma

Funções de forma são representadas pelo conceito `ShapeFunction` (Figura 4.17), e devem implementar os métodos para configurar os nós de suporte a serem utilizados (`set_support_nodes(...)`) e para calcular as funções nodais (`eval(...)`) e suas derivadas (`deval(...)`) em um ponto `x`. Além disso, a função de forma informa se satisfaz a propriedade do delta de Kronecker (Equação 3.7) através da *tag* `Phi_kronecker_delta_tag`.

Os métodos de construção de funções de forma disponíveis no MFree Framework em sua versão atual são o *Moving Least Squares*, funções de *Shepard*, *Point Interpolation Method*, *Radial Point Interpolation Method* e *Radial Point Interpolation Method with Polynomials*, implementados, respectivamente, pelas classes `Phi_mls`, `Phi_shepard`, `Phi_pim`, `Phi_rpim` e `Phi_rpimp`.

Existe, ainda, a classe `Phi_hybrid` que representa funções de forma mistas e que é utilizada pelo método MLPG misto (Seção 3.3.4). Esse tipo híbrido utiliza ora funções de forma `Phi_K`, ora `Phi_T`, definidas pela *traits* `PhiHybridTraits` (Figura 4.18). `Phi_K` deve satisfazer o delta de Kronecker e é utilizada em regiões próximas ao contorno de Dirichlet. `Phi_T` é aplicada em regiões distantes da fronteira de Dirichlet.

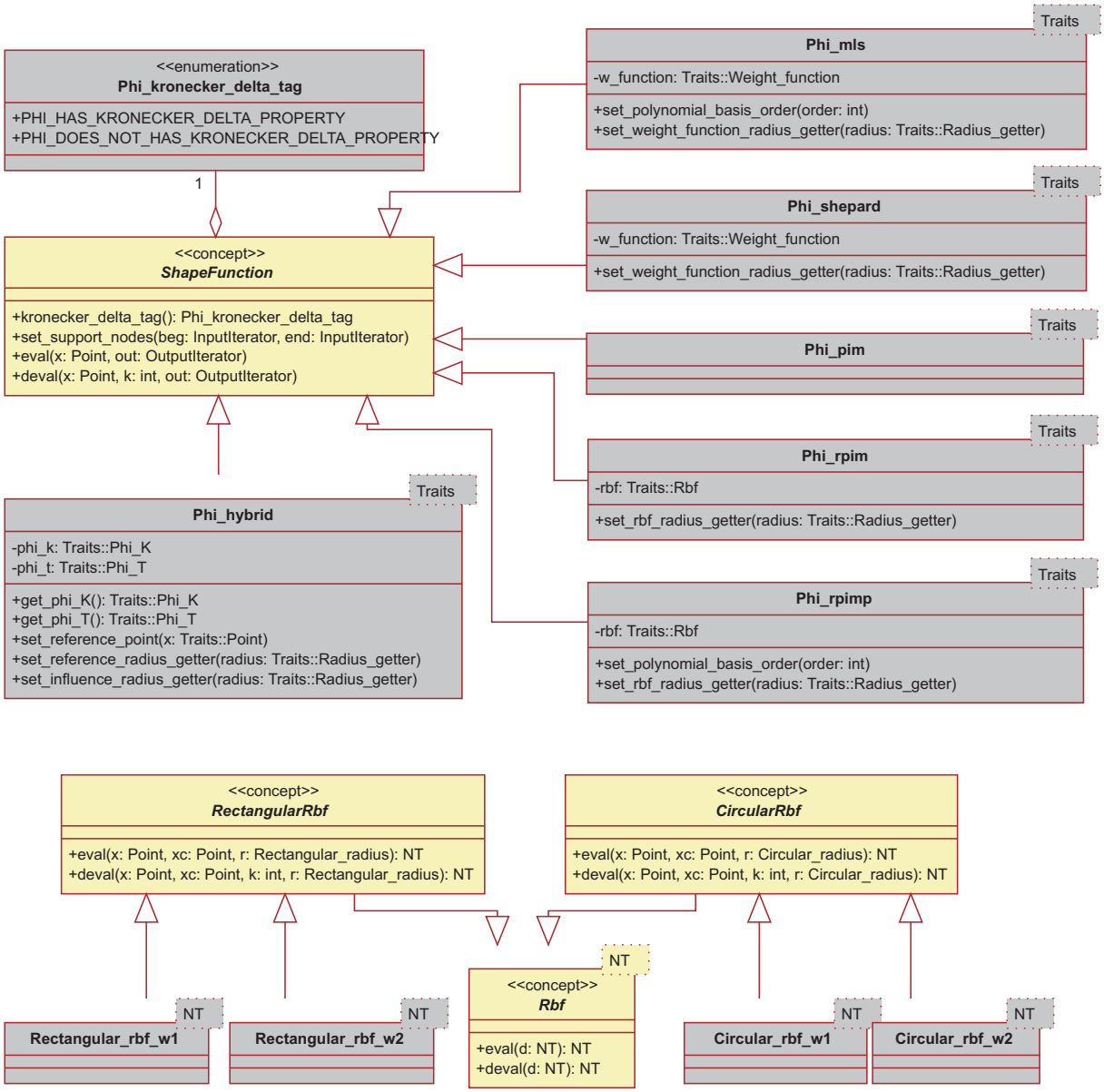


Figura 4.17: Diagrama UML: Função de forma.

Algumas funções de forma utilizam RBFs para gerar as aproximações locais. Uma RBF é modelada pelo conceito `Rbf`, que exige a implementação dos métodos para calcular o valor e a derivada da função de base radial a uma distância `d` em relação ao seu centro normalizada pelo raio de suporte, de acordo com o Apêndice B. Dois tipos de RBFs estão disponíveis: circulares (`CircularRbf`) e retangulares (`RectangularRbf`). No primeiro caso, `d` é escolhido como a distância entre o centro da RBF e o ponto de interesse, sendo o raio de suporte radial. No segundo caso, toma-se o produto tensorial entre os valores da RBF avaliados separadamente em cada direção do espaço, `d` é computado como a distância

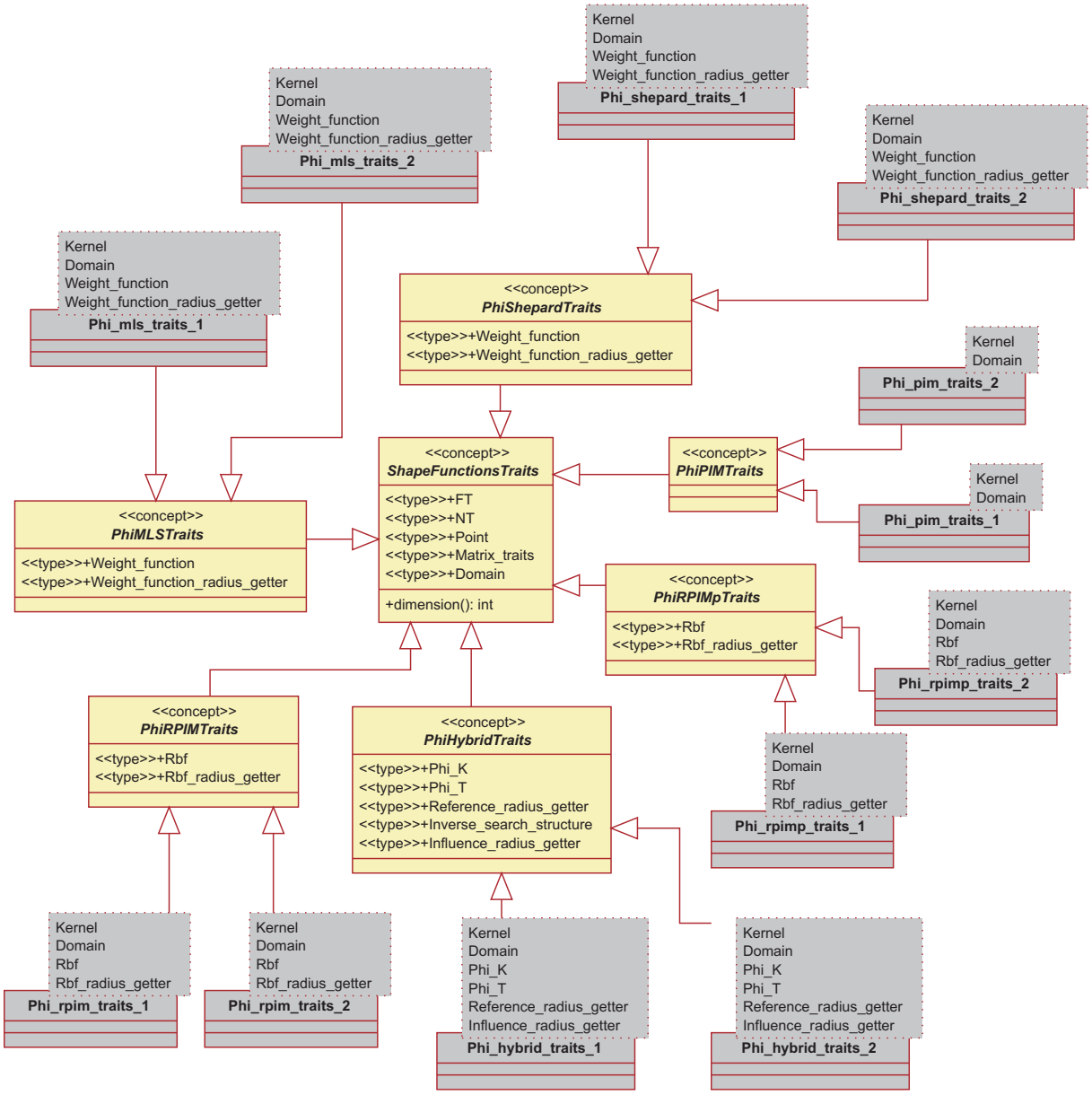


Figura 4.18: Diagrama UML: *Traits* para função de forma.

entre o centro e o ponto de interesse para cada direção, existindo um raio de suporte para cada componente. Na versão atual do *framework*, as RBFs *spline* cúbica (Equação B.4) e quártica (Equação B.6) são implementadas pelas classes de nomes pós-fixados em `w1` e `w2`, respectivamente. Por exemplo, `Circular_rbf_w1` é a RBF circular *spline* cúbica.

4.11 Integração

O processo de montagem do sistema de equações envolve a integração de funções sem expressão analítica, sendo necessário utilizar integração numérica. O MFree Framework emprega a quadratura de Gauss.

O conceito de integração é representado por `Integration`, como mostra a Figura 4.19. Observe que `Integration` fornece *iterators* para os pontos de integração localizados dentro do domínio e em suas fronteiras. Ponto de integração é implementado pela classe `Integration_point`, sendo constituído pela representação espacial e pelo peso e jacobiano utilizados na quadratura. Caso esteja localizado sobre um contorno, possui ainda um vetor normal unitário que aponta para fora do domínio.

Existem, atualmente, dois tipos de integração no *framework*. O primeiro deles é a integração por malha de fundo (utilizada no EFG), ligada ao conceito `CellIntegration`, mostrado na Figura 4.19. Dois formatos de células da malha estão disponíveis: triangular e retangular, implementados pelas *traits* `Triangular_cell_integration_traits_2` e `Rectangular_cell_integration_traits_2` (Figura 4.20). Em uma dimensão, as células são sempre segmentos, representadas por `Cell_integration_traits_1`. A malha de integração é lida de um arquivo externo, de acordo com o formato descrito no Apêndice A.2.

O outro tipo de integração é por domínio local (utilizada no MLPG), representado pelo conceito `LocalDomainIntegration` (Figura 4.19). Geralmente utiliza-se a implementação `Local_domain_integration_polygon`, que possibilita computar a interseção entre os subdomínios de quadratura e o domínio global. Isso é realizado utilizando a representação de polígonos da CGAL `CGAL::General_polygon_2`, que permite realizar operações de interseção entre polígonos. Os subdomínios de quadratura podem ter qualquer formato, sendo circular e retangular disponibilizados pelo *framework* através das *traits* `Rectangular_local_domain_integration_polygon_traits_2` e `Circular_local_domain_integration_polygon_traits_2` (Figura 4.20). Para outros formatos de subdomínio, o programador deve criar uma *traits* que forneceça a forma inicial do polígono por meio do campo `Local_domain_initial_polygon`. Quando o domínio do problema é formado por múltiplas regiões, a regra de não penetração é utilizada e os

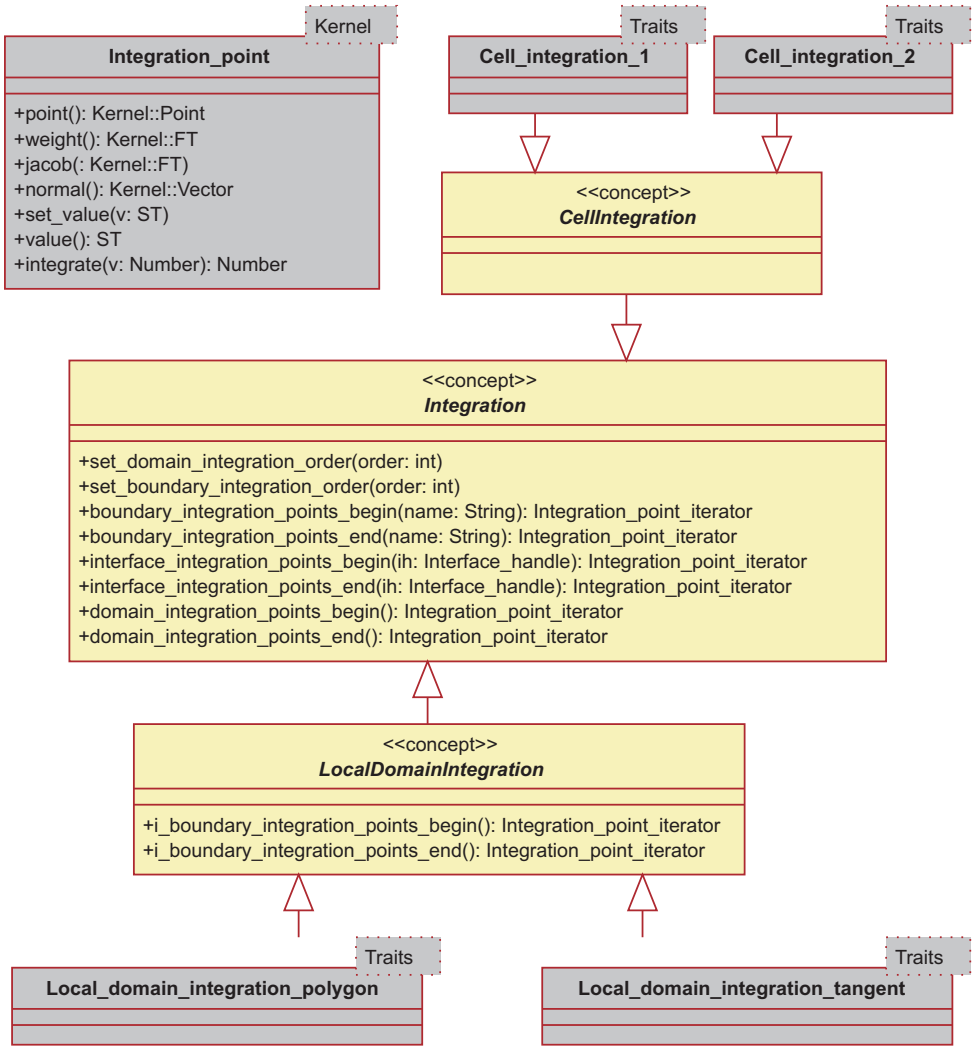


Figura 4.19: Diagrama UML: Integração.

subdomínios de integração não ultrapassam as fronteiras da região a que pertencem. A regra de não penetração para subdomínios de quadratura pode ser vista com detalhes na Seção 3.3.3.3.

Outra implementação de subdomínios de quadratura é dada por `Local_domain_integration_tangent` (Figura 4.19). Nesse caso, os subdomínios são redimensionados de modo a tangenciar os contornos do domínio do problema, inclusive fronteiras entre regiões. Essa implementação é útil em métodos de colocação para impor condições de contorno essenciais e naturais (Seção 3.3.2.4) e para forçar a continuidade da aproximação nas fronteiras entre diferentes materiais (Seção 3.3.2.5). O *framework* disponibiliza uma implementação para subdomínios tangentes circulares através da

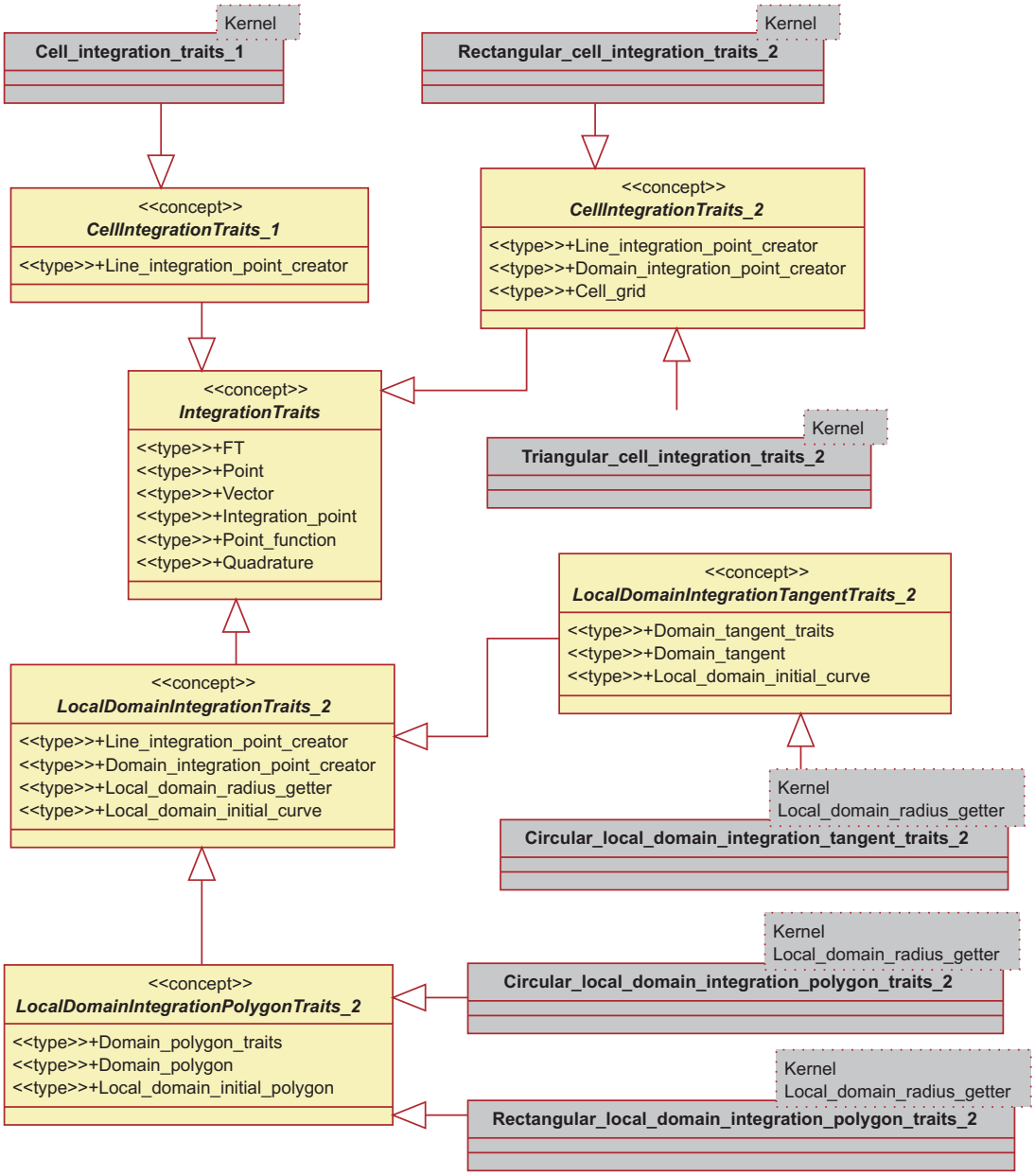


Figura 4.20: Diagrama UML: *Traits* para integração.

traits `Circular_local_domain_integration_tangent_traits_2` (Figura 4.20), mas é possível utilizar subdomínios com outros formatos. Para isso, deve-se criar uma classe *traits* que forneça o formato inicial da curva (campo `Local_domain_initial_curve`) e o tipo que a modifica de modo a tornar-se tangente ao contorno do domínio global (campo `Domain_tangent`), caso necessário.

4.12 Métodos sem Malha

No *framework*, métodos sem malha são implementados de acordo com o conceito `Method` (Figura 4.21). Todos os métodos MFree devem ter em sua interface as funções `assemble()`, `solve()`, `calculate_solution(...)` e `calculate_derivative_solution(...)`. `assemble()` é responsável por montar o sistema de equações e utiliza, para isso, uma formulação fornecida pelo parâmetro template `Formulation`. `solve()` resolve o sistema de equações. As funções `calculate_solution(...)` e `calculate_derivative_solution(...)` calculam a aproximação da solução e suas derivadas em um ponto especificado como parâmetro. Basicamente, todos essas funções de `Method` delegam à formulação a execução de seus papéis. O conceito de formulação é descrito na Seção 4.13.

A Figura 4.21 mostra o diagrama UML para os métodos sem malha. Observe que a `Method` estão associados um domínio de suporte (`SupportDomain`), uma função de forma (`ShapeFunction`), uma integração (`Integration`), uma formulação (`Formulation`) e um domínio (`Domain`), cujos tipos são fornecidos pela *traits* `MethodTraits` (Figura 4.22). Dessa forma, é possível resolver um problema utilizando qualquer método sem malha que possua uma combinação possível desses tipos, conferindo grande flexibilidade ao *framework*.

A Figura 4.22 mostra as *traits* disponíveis para os métodos sem malha. O conceito `MethodTraits` descreve os requisitos mínimos de tipos que um método necessita. Além dos já citados anteriormente, existem `System_matrix_traits` e `Equation_system_solver`, que correspondem, respectivamente, à *traits* usada para obter as representações matriciais do sistema de equações global e o tipo responsável por solucioná-lo. `System_matrix_traits` é obtida diretamente do Kernel, e `Equation_system_solver` deve ser um modelo de `EquationSystemSolver`, descrito na Seção 4.14. Todos os métodos utilizam a classe `Method_traits`, com exceção de Mlpg1 e Lpim1, que usam `Method_LPG1_traits`. Isto se dá pelo fato destes métodos necessitarem da especificação de uma função de teste, que ocorre através do parâmetro `Test_function`, normalmente uma RBF.

A seguir, descreve-se as configurações para os métodos disponíveis no MFree Framework:

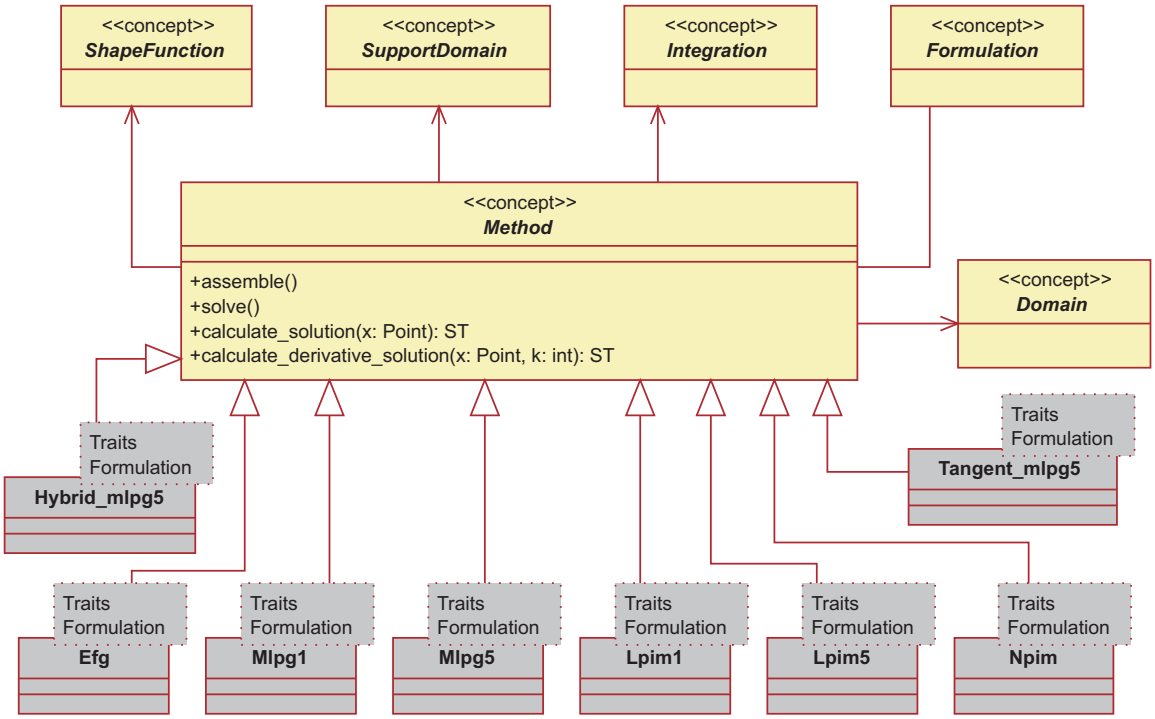


Figura 4.21: Diagrama UML: Métodos sem malha.

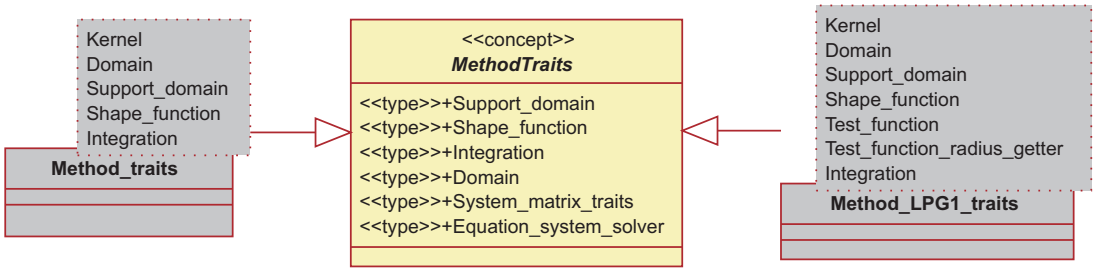


Figura 4.22: Diagrama UML: *Traits* para métodos sem malha.

- Efg : Element-Free Galerkin Method
  - **Função de forma:** `Phi_mls` ou `Phi_shepard`
  - **Integração:** deve ser um modelo de `CellIntegration`
  - **Formulação:** deve ser um modelo de `GlobalGalerkinFormulation`
  - **Traits:** `Method_traits`
- Mlpg1 : Meshless Local Petrov-Galerkin Method, versão 1
  - **Função de forma:** `Phi_mls` ou `Phi_shepard`
  - **Integração:** deve ser um modelo de `LocalDomainIntegration`
  - **Formulação:** deve ser um modelo de `LocalPetrovGalerkinFormulation`
  - **Traits:** `Method_LPG1_traits`

- Mlpg5 : Meshless Local Petrov-Galerkin Method, versão 5
  - **Função de forma:** `Phi_mls` ou `Phi_shepard`
  - **Integração:** deve ser um modelo de `LocalDomainIntegration`
  - **Formulação:** deve ser um modelo de `LocalPetrovGalerkinFormulation`
  - **Traits:** `Method_traits`
- Npim : Nonconforming Point Interpolation Method
  - **Função de forma:** `Phi_pim` ou `Phi_rpim` ou `Phi_rpimp`
  - **Integração:** deve ser um modelo de `CellIntegration`
  - **Formulação:** deve ser um modelo de `GlobalGalerkinFormulation`
  - **Traits:** `Method_traits`
- Lpim1 : Local Point Interpolation Method, versão 1
  - **Função de forma:** `Phi_pim` ou `Phi_rpim` ou `Phi_rpimp`
  - **Integração:** deve ser um modelo de `LocalDomainIntegration`
  - **Formulação:** deve ser um modelo de `LocalPetrovGalerkinFormulation`
  - **Traits:** `Method_LPG1_traits`
- Lpim5 : Local Point Interpolation Method, versão 5
  - **Função de forma:** `Phi_pim` ou `Phi_rpim` ou `Phi_rpimp`
  - **Integração:** deve ser um modelo de `LocalDomainIntegration`
  - **Formulação:** deve ser um modelo de `LocalPetrovGalerkinFormulation`
  - **Traits:** `Method_traits`
- Tangent_mlpg5 : MLPG, versão 5, em que os domínios de quadratura são sempre tangentes aos contornos do domínio. As condições de contorno e de continuidade na interface de materiais são impostas pelo método de colocação.
  - **Função de forma:** `Phi_mls` ou `Phi_shepard`
  - **Integração:** `Local_domain_integration_tangent`
  - **Formulação:** deve ser um modelo de `LocalPetrovGalerkinFormulation`
  - **Traits:** `Method_traits`
- Hybrid_mlpg5 : Método misto baseado no MLPG, versão 5, descrito na Seção 3.3.4.
  - **Função de forma:** `Phi_hybrid`
  - **Integração:** deve ser um modelo de `LocalDomainIntegration`
  - **Formulação:** deve ser um modelo de `LocalPetrovGalerkinFormulation`
  - **Traits:** `Method_traits`

4.13 Formulação

O papel principal da formulação no *framework* é executar a montagem do sistema de equações de acordo com a formulação física do problema. É dividida em duas grandes categorias, conforme o método usado para discretizar a forma fraca. A Figura 4.23 mostra o diagrama UML com os tipos de formulações existentes.

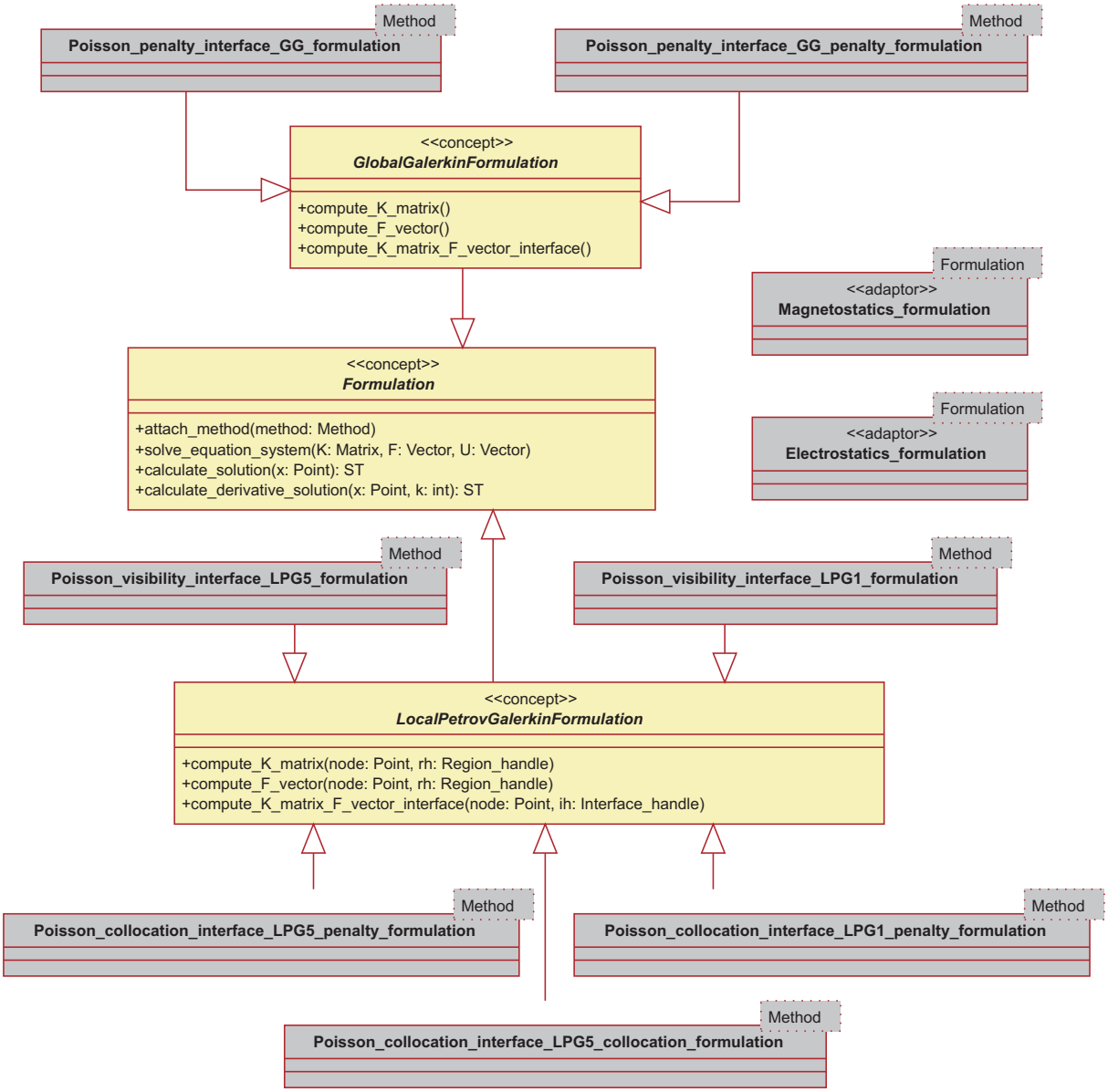


Figura 4.23: Diagrama UML: Formulação.

O conceito básico representante de uma formulação é `Formulation`, que fornece funções para resolver o sistema de equações e calcular a solução aproximada e suas derivadas. Essas funções são chamadas pelo método sem malha que faz uso da formulação. Observe que existem duas grandes linhas: uma baseada no método de Galerkin global (`GlobalGalerkinFormulation`) e outra no de Petrov-Galerkin local (`LocalPetrovGalerkinFormulation`). Os dois conceitos exigem a implementação de três métodos cujos objetivos são: (1) montar a matriz K do sistema (`compute_K_matrix()`), (2) montar o vetor F do sistema (`compute_F_vector()`), e (3) fazer as contribuições em K e F oriundas do tratamento da descontinuidade nas interfaces entre diferentes materiais.

As formulações disponíveis no *framework* correspondem às que derivam de formas fortes descritas por uma equação diferencial de Poisson, de acordo com a Equações 2.39 e 2.38 em uma e duas dimensões, respectivamente. Entretanto, formulações para qualquer problema podem ser implementadas modelando os conceitos apropriados.

`Poisson_penalty_interface_GG_formulation` corresponde a uma formulação utilizando o método de Galerkin global na forma fraca e o da penalidade para tratamento das descontinuidades de materiais. Nenhum procedimento é usado para impor as condições de contorno essenciais, portanto, as funções de forma do método sem malha devem satisfazer o delta de Kronekcer. Essa é a formulação usada pelo método `Npim` e pode ser vista na Seção 3.3.3.1.

`Poisson_penalty_interface_GG_penalty_formulation` é semelhante à formulação anterior (`Poisson_penalty_interface_GG_formulation`), porém utiliza o método da penalidade para impor as condições de contorno essenciais. Por isso, o método sem malha deve utilizar funções de forma que não possuem a propriedade do delta de Kronecker. Essa formulação é usada pelo método `Efg` e descrita na Seção 3.3.1.

`Poisson_visibility_interface_LPG5_formulation` utiliza o método de Petrov-Galerkin local na forma fraca e o critério da visibilidade generalizado para tratar as descontinuidades de materiais. Nenhum método é utilizado para impor as condições de contorno essenciais. Considera-se que, implicitamente, a função de teste utilizada é a função de Heaviside. Essa formulação é utilizada pelos métodos `Lpim5` e `Hybrid_mlpg5`. De forma semelhante, porém utilizando explicitamente funções de teste do tipo RBF, a for-

mulação `Poisson_visibility_interface_LPG1_formulation` é empregada pelo método `Lpim1`. Ambas formulações são descritas na Seção 3.3.3.2.

As formulações `Poisson_collocation_interface_LPG5_penalty_formulation` e `Poisson_collocation_interface_LPG1_penalty_formulation` também aplicam o método de Petrov-Galerkin local para discretizar a forma fraca. Utilizam o método de colocação para tratar descontinuidades de materiais e penalidade para impor condições de contorno essenciais. A primeira formulação usa, implicitamente, a função de Heaviside como função de teste, e é utilizada pelo método `Mlpg5`. A segunda usa, explicitamente, uma função de base radial como função de teste, e é utilizada pelo método `Mlpg1`. A descrição das duas formulações é mostrada na Seção 3.3.2.

Por fim, `Poisson_collocation_interface_LPG5_collocation_formulation` utiliza o método da colocação tanto para tratar as descontinuidades de materiais quanto para impor as condições de contorno essenciais e naturais. É utilizada por `Tangent_mlpg5`. O procedimento de imposição das condições de contorno é mostrado na Seção 3.3.2.4.

Note que na Figura 4.23 existem os tipos `Electrostatics_formulation` e `Magnetostatics_formulation`. Estes são classes *adaptors* que transformam as formulações gerais do tipo Poisson para formulações de problemas eletrostáticos e magnetostáticos, respectivamente. Na verdade, o que os *adaptors* fazem é o mapeamento dos parâmetros da formulação estática genérica para a eletrostática ou magnetostática. Observe que as duas classes são parametrizadas por `Formulation`, que é a formulação a ser adaptada. As formas forte para problemas eletrostáticos e magnetostáticos são dadas pelas Equações 2.26 e 2.37.

4.14 Solução do Sistema de Equações

A solução do sistema global de equações deve ser feito por um *solver* que seja um modelo de `EquationSystemSolver`, como mostra a Figura 4.24. Observe que o conceito `EquationSystemSolver` implica que o método `solve(...)` esteja presente na interface. Este método recebe a matriz `A` e o vetor `b` do sistema, calcula a solução, e retorna-

a por meio do vetor `x`, também passado como parâmetro. Existem dois tipos de *solvers*. Um é destinado a resolver sistemas de equações lineares, representado pelo conceito `LinearEquationSystemSolver`, e outro para sistemas de equações não lineares, representado por `NonLinearEquationSystemSolver`.

A versão atual do MFree Framework fornece um *solver* para sistemas de equações lineares, implementado pela classe `Umfpack_linear_system_solver`, que utiliza a biblioteca UMFPACK. A UMFPACK usa um algoritmo destinado a solucionar sistemas não simétricos, mas que pode ser aplicado também a sistemas simétricos. Sistemas não simétricos, normalmente, são produzidos por métodos que utilizam Perov-Galerkin para discretizar a forma fraca, como o MLPG. Sistemas simétricos são produzidos, geralmente, quando se usa Galerkin para discretizar a forma fraca, como faz o EFG. Existem métodos dedicados, exclusivamente, a solucionar sistemas de equações simétricos e que o fazem de forma bastante eficiente, como o Método do Gradiente Conjugado (Seção 3.4). Pretende-se implementar métodos para sistemas simétricos em versões futuras do *framework*.

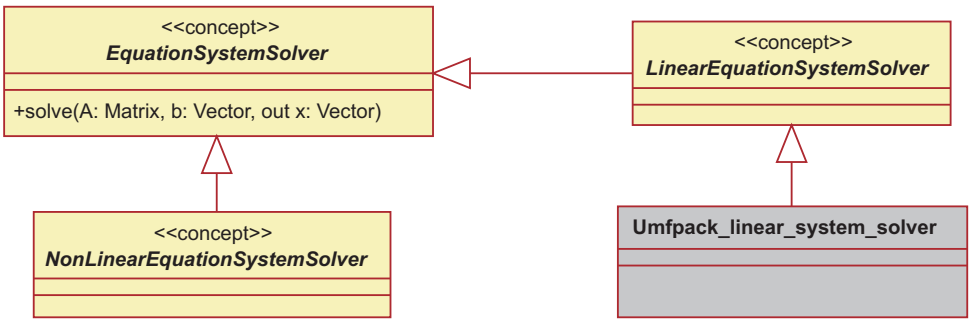


Figura 4.24: Diagrama UML: Solução do sistema linear.

4.15 Controles

Como visto na Seção 4.1, os controles são responsáveis pelo fluxo de execução principal do MFree Framework. Eles são, na verdade, a interface entre o *framework* e a aplicação escrita pelo desenvolvedor. As Figuras 4.25 e 4.26 mostram os controles e os conceitos das *traits* que devem ser utilizadas.

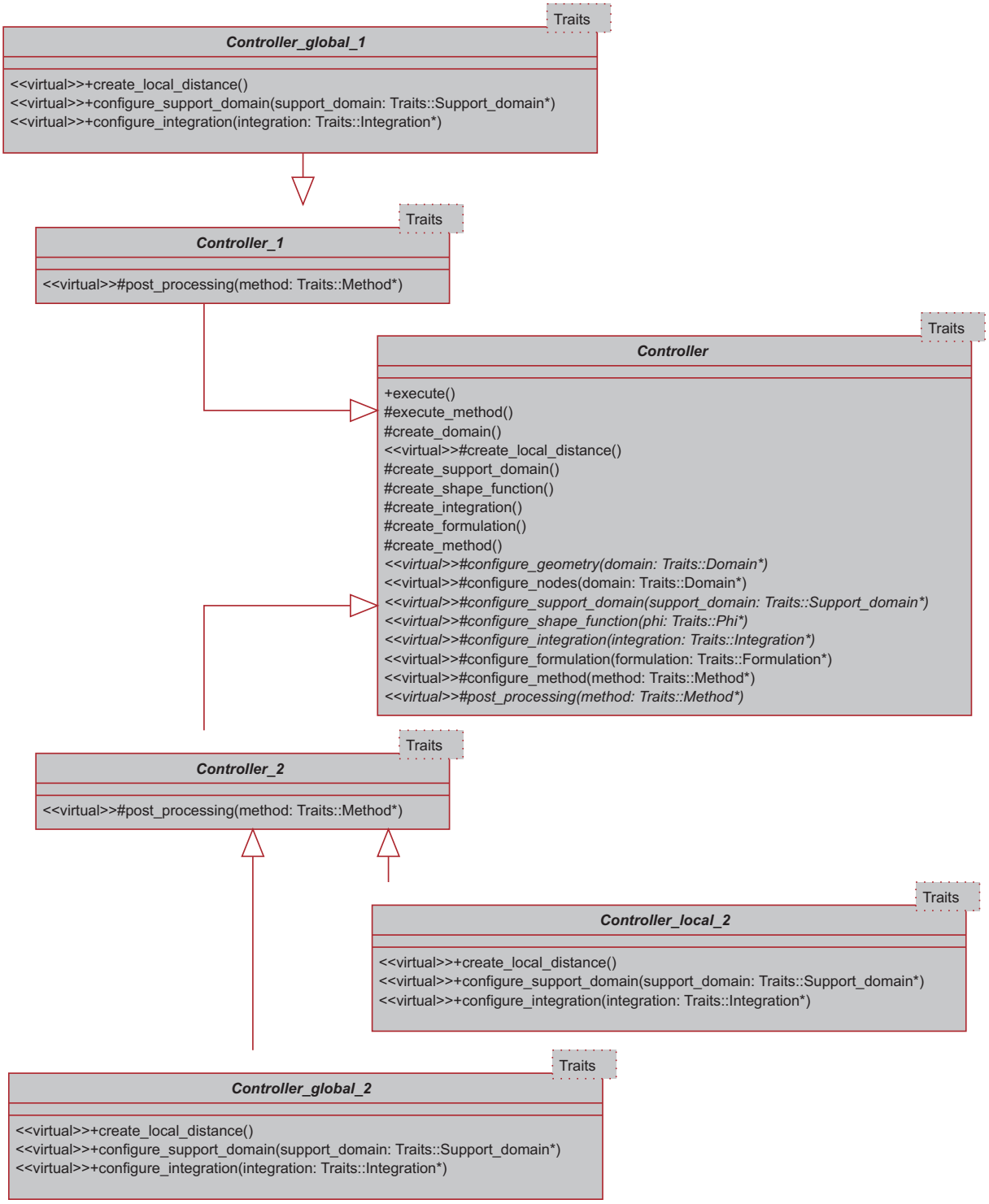


Figura 4.25: Diagrama UML: Controles.

Para construir uma aplicação, o programador deve (1) criar uma classe *traits* que forneça os tipos requeridos pelo controle, (2) criar uma classe derivada de um dos controles

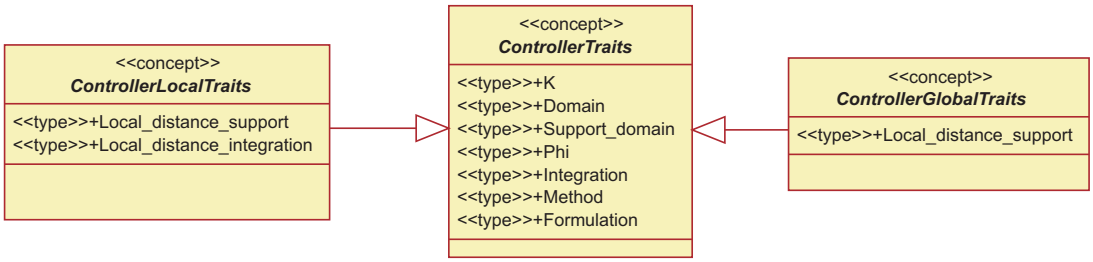


Figura 4.26: Diagrama UML: *Traits* para controles.

existentes, (3) implementar os métodos *abstratos*[§], configurando as características do método sem malha de acordo com suas necessidades, e (4) efetuar uma chamada ao método `execute()` do controle criado. Observe que `execute()` é o ponto de entrada para o fluxo de controle determinado pelo *framework*.

A classe base para controles é `Controller` (Figura 4.25). Os métodos de `Controller` têm função de criar as estruturas de dados, configurá-las, resolver o problema e executar o pós-processamento. Existem dois tipos de controles: um destinado aos métodos de forma fraca global (`Controller_global_1` e `Controller_global_2`) e outro aos de forma fraca local (`Controller_local_2`). No segundo caso, além de existir uma estrutura relacionada às distâncias locais utilizada pelo domínio de suporte, existe outra para os domínios de integração, visto que esta é realizada por subdomínios locais.

A seguir, descrevem-se os passos do fluxo de controle do MFree Framework mostrados na Figura 4.2.

1. **Criação do Domínio**
   Cria-se o domínio do problema. Inicialmente, a geometria e as condições de contorno são construídas. O processo de determinação da geometria e condições de contorno ainda é feito por meio de código dentro da função `configure_geometry(...)`. A ideia era que esse procedimento fosse usado apenas na fase inicial do desenvolvimento do *framework* para que se pudesse testar o correto funcionamento de problemas com diferentes tipos de geometria e condições de contorno. Entrar com essas informações via código não é adequado pois exige que o cliente codifique o problema e que a aplicação seja sempre recompilada. A melhor forma para isso é fornecer tais

[§] Métodos abstratos são métodos apenas declarados mas não implementados. Em C++, são denominados *virtuais puros*. No diagrama de classes, métodos abstratos são representados pelo estilo itálico.

informações através de um arquivo externo, o que se pretende fazer em versões futuras do *framework*. Após lidas geometria e condições de contorno, criam-se os nós por meio da função `configure_nodes(...)`. Por padrão, os nós são lidos de um arquivo externo, conforme formato descrito em A.1.

2. **Criação das Distâncias Locais**
   Criam-se as distâncias locais relacionadas às densidades de nós em todo o domínio. O *framework* faz os cálculos, por meio da função `create_local_distance()`, a partir do tipo especificado em `Traits`.

3. **Criação do Domínio de Suporte**
   Cria-se a estrutura para domínio de suporte. Cabe ressaltar que os domínios são calculados de fato durante o processo de montagem do sistema de equações. O *framework* faz a criação da estrutura de dados automaticamente. Caso o programador deseje configurar algum parâmetro extra, deve fazê-lo por meio da função `configure_support_domain(...)`.

4. **Criação da Integração**
   Cria-se a estrutura de dados para integração. O *framework* oferece uma implementação padrão. Caso o desenvolvedor deseje configurar a integração de modo alternativo, deve redefinir a função `configure_integration(...)`.

5. **Criação da Função de Forma**
   Nesse passo é criada a estrutura para construção das funções de forma. O programador deve, obrigatoriamente, definir a função `configure_shape_function(...)`, onde é configurada a estrutura de dados.

6. **Criação da Formulação**
   Cria-se a formulação. Caso necessário, a função `configure_formulation(...)` deve ser implementada para configurar algum parâmetro extra da formulação.

7. **Criação do Método sem Malha**
   Cria-se o método sem malha. O procedimento é feito automaticamente pelo *framework*. Uma configuração alternativa pode ser feita redefinindo o método `configure_method(...)`.

8. **Execução do Método sem Malha**
   Nesse item, o problema é solucionado. Primeiramente, monta-se o sistema de equações que, posteriormente, é resolvido. Essa etapa é totalmente automatizada.

9. **Execução do Pós-Processamento**
   Executa-se o pós-processamento. Por padrão, nesse passo é calculada a aproximação da solução nos pontos informados no arquivo externo descrito em A.3, e a solução é gravada em um arquivo de formato A.5. O desenvolvedor pode, por ventura, redefinir a função `post_processing(...)` para executar os procedimentos de pós-processamento que desejar.

Capítulo 5

Resultados

No capítulo 4 foi descrita a arquitetura do MFree Framework. Este capítulo traz algumas aplicações do *framework* a problemas eletrostáticos e magnetostáticos comuns em uma e duas dimensões. Investiga-se os problemas da calha, do capacitor, do eletroímã e do campo magnético uniforme. Em alguns deles, há variações de geometria, das condições de contorno ou da constituição dos materiais. Os resultados são comparados com a solução analítica, quando disponível, ou com a obtida utilizando o Método dos Elementos Finitos com elementos triangulares de primeira ordem. Para visualização dos resultados, usou-se o software MATLAB [38] versão R2007b.

O intuito do capítulo não é discutir com profundidade a qualidade das soluções ou mesmo qual método aplicar a determinado problema de eletromagnetismo, mas sim ilustrar as opções e ferramentas disponíveis no *framework* e verificar o seu correto fucionamento. Quando julgado apropriado, algumas considerações com respeito às características do método ou da solução são feitas, todavia de forma superficial, como ressaltado anteriormente.

5.1 Capacitor 1D

A primeira aplicação desenvolvida destina-se à solução de um problema em uma dimensão, para a qual o *framework* disponibiliza o *Element-Free Galerkin Method*. O problema escolhido é o do capacitor 1D. A aproximação obtida pelo método é verificada junto à solução analítica. A Figura 5.1 mostra o problema do capacitor em uma dimensão.

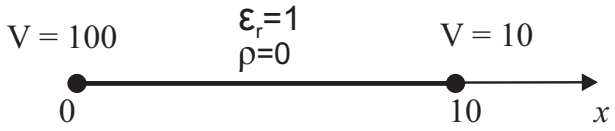


Figura 5.1: Capacitor em 1D. Não há densidade de carga e a permissividade elétrica relativa do meio é igual a um. O potencial elétrico é dado em $Volts$. As condições de contorno essenciais são $V(0) = 100V$ e $V(10) = 10V$.

Note que há apenas condições de contorno essenciais, localizadas nos extremos do domínio ($x = 0$ e $x = 10$). Não há densidade de carga elétrica. A forma forte do problema é dada pela Equação 2.39, com $k = \epsilon_r = 1$ e $f = \rho = 0$. A solução analítica para o potencial elétrico V^a e campo elétrico E^a é:

$$\begin{aligned} V^a(x) &= -9x + 100 \quad [V] \\ E^a(x) &= 9 \quad [V/m] \end{aligned} \tag{5.1}$$

O método sem malha EFG é utilizado com uma distribuição uniforme de 11 nós. As funções de forma são geradas por MLS com polinômios de primeira ordem e função de peso *spline* quártica (Equação B.6). O fator de penalidade é $\alpha = 10^6$ e a integração é feita com 1 ponto de quadratura por célula, com 10 células ao todo. Os domínios de influência são gerados com $\alpha_I = 1.5$. A Figura 5.2 mostra a solução numérica obtida. O erro absoluto percentual é calculado por:

$$erro = 100 \times \left| \frac{U^a - U^h}{U^a} \right| \quad [\%] \tag{5.2}$$

onde U^h é a solução numérica e U^a a analítica.

Note que a solução gerada pelo EFG é semelhante à analítica mas com erros percentuais da ordem de 10^{-5}. A princípio, como a solução do problema é linear e as funções de

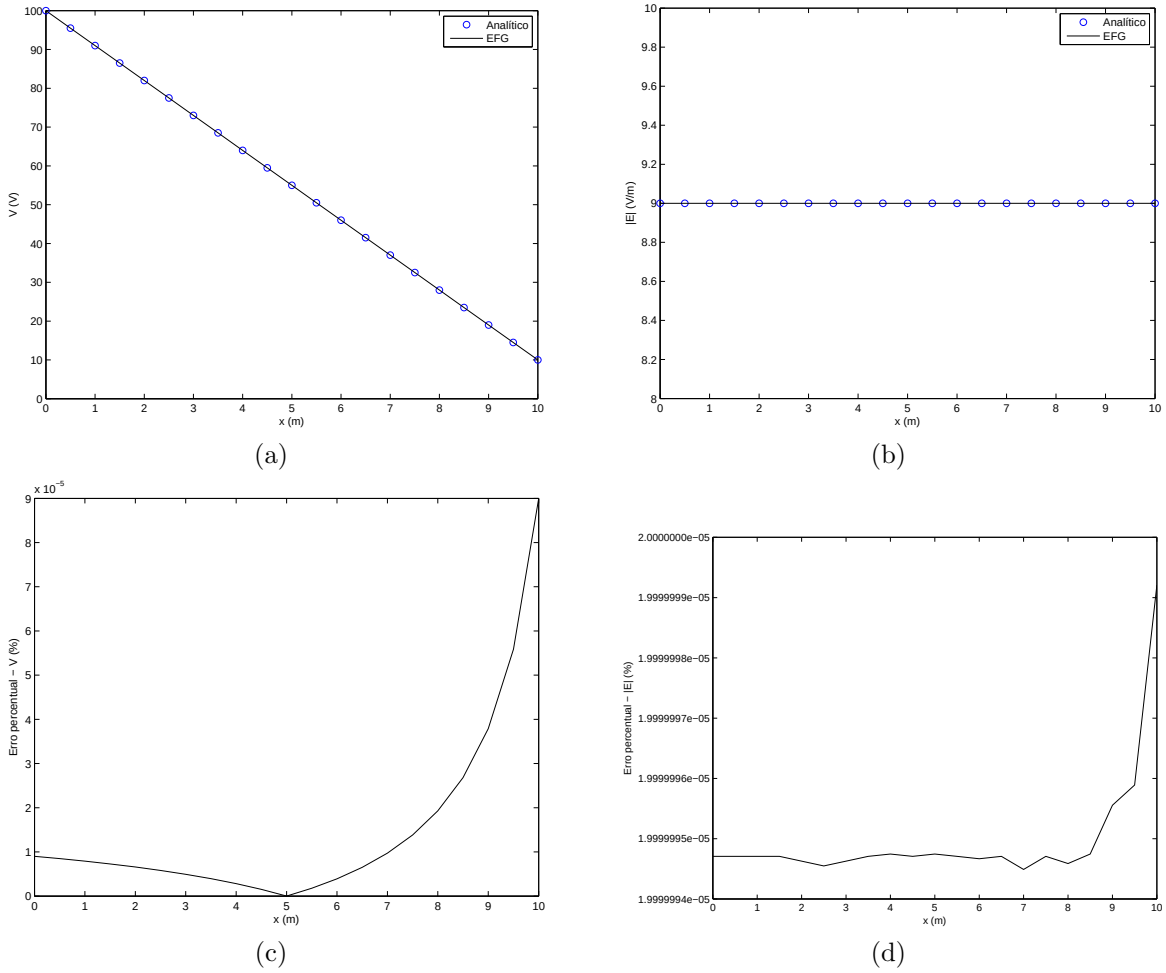


Figura 5.2: Solução para o problema do capacitor em 1D. (a) Compara a solução analítica com a numérica obtida pelo EFG para o potencial elétrico e (b) para o campo elétrico. (c) e (d) mostram os erros para potencial e campo elétricos, respectivamente. O erro é calculado pela Equação 5.2. Usou-se 11 nós distribuídos uniformemente e 10 células para integração.

forma utilizadas possuem uma base polinomial linear, a solução pelo método EFG deveria ser exata. Entretanto, o método da penalidade, utilizado para impor as condições de contorno essenciais, o faz de forma aproximada e provoca erros na aproximação (note que os erros são máximos no contorno). Resolvendo novamente o problema, mas com fator de penalidade $\alpha = 10^{10}$, os erros percentuais diminuem na mesma proporção da variação de α e ficam da ordem de 10^{-9}, o que comprova que são, de fato, advindos do procedimento de imposição das condições de contorno essenciais.

5.2 Calha 2D

O problema da calha é um problema muito comum na eletrostática. Pode-se ver a solução deste utilizando o FEM em [48], e métodos sem malha em [19], [20] e [47]. Esta seção aborda o problema com duas variações localizadas nas condições de contorno essenciais: a primeira em que são todas constantes e a segunda onde uma delas é senoidal. No primeiro caso utiliza-se o *Meshless Local Petrov-Galerkin Method*, versão 1, para obter a solução aproximada. No segundo caso, usa-se o *Nonconforming Point Interpolation Method*. Em todas as situações constrastam-se as aproximações com as soluções analíticas dos problemas. A taxa de convergência é investigada para o NPIM, dada a característica de incompatibilidade de suas funções de forma.

5.2.1 Condição de Contorno de Dirichlet Constante

A Figura 5.3 mostra o problema da calha em duas dimensões. As condições de contorno essenciais são todas constantes. Observe que há um salto no valor do potencial elétrico nas regiões próximas aos vértices $(0, 4)$ e $(4, 4)$, de $10V$ para $100V$. Este salto torna o problema de difícil solução para os métodos numéricos, que não preveem essa descontinuidade.

A forma forte para o problema da calha é dada pela Equação 2.26, com $\rho = 0$ e $\epsilon_r = 1$. A solução analítica, obtida de [48], é:

$$V^a(x, y) = \frac{400}{\pi} \sum_{k=1}^{\infty} \frac{\sin \frac{(2k-1)\pi x}{4} \sinh \frac{(2k-1)\pi y}{4}}{(2k-1) \sinh (2k-1)\pi} \tag{5.3}$$

O método sem malha utilizado é o MLPG1 com uma distribuição uniforme de 11×11 nós. As funções de forma são construídas pelo MLS de ordem polinomial um. As funções de teste usadas pela forma fraca são as funções de peso do MLS: a *spline* cúbica, dada pela Equação B.4. O fator de penalidade para imposição das condições de contorno de Dirichlet é $\alpha = 10^5$. Domínios de influência e quadratura retangulares são usados. A integração é realizada com 1 ponto de quadratura por segmento do subdomínio de integração em integrais de contorno, e 2×2 pontos por subdomínio em integrais de

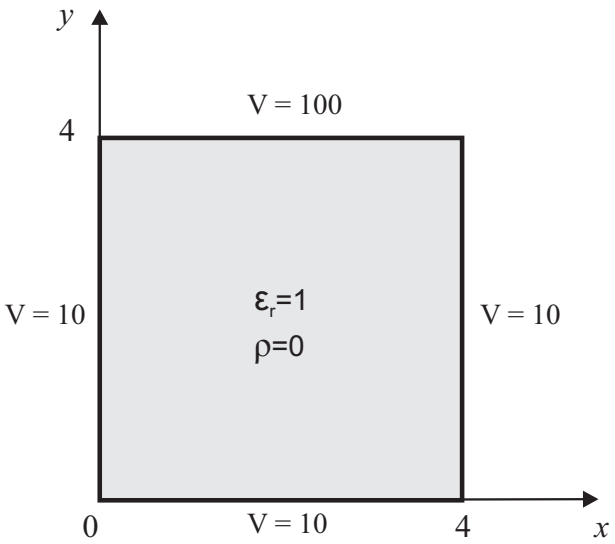


Figura 5.3: Calha em 2D. Não há densidade de carga e a permissividade elétrica relativa do meio é igual a um. As condições de contorno essenciais são todas constantes. O potencial elétrico é dado em $Volts$.

domínio. Os parâmetros para os raios de influência e quadratura adotados são $\alpha_I = 2.0$ e $\alpha_Q = 0.5$. A Figura 5.4 mostra a solução e o erro absoluto percentual para o problema. Note que as condições de contorno essenciais não são satisfeitas de forma exata pois as funções de forma geradas pelo MLS não possuem a propriedade do delta de Kronecker. Observe, também, que o erro é alto nas regiões próximas aos vértices $(0, 4)$ e $(4, 4)$ por causa da descontinuidade do potencial nestes pontos. Na verdade, isso ocorre com métodos numéricos em geral pois o potencial é definido com dois valores diferentes em cada um desses vértices. Todavia, a aproximação no restante do domínio é coerente com a solução analítica e possui erros percentuais inferiores a 0.4%.

5.2.2 Condição de Contorno de Dirichlet Senoidal

Uma variação do problema da calha é obtida usando-se, na parte superior da geometria, uma condição de contorno de Dirichlet do tipo senoidal ao invés de constante. O domínio do problema é $[0, 10] \times [0, 10]$. Aproveita-se a simetria do problema, e a geometria é dividida em duas, com eixo de simetria em $x = 5$. Resolve-se apenas a metade do problema (em $0 \leq x \leq 5$). Para isso, é necessário inserir uma condição de contorno natural homogênea na região de simetria. A Figura 5.5 mostra a variação do problema

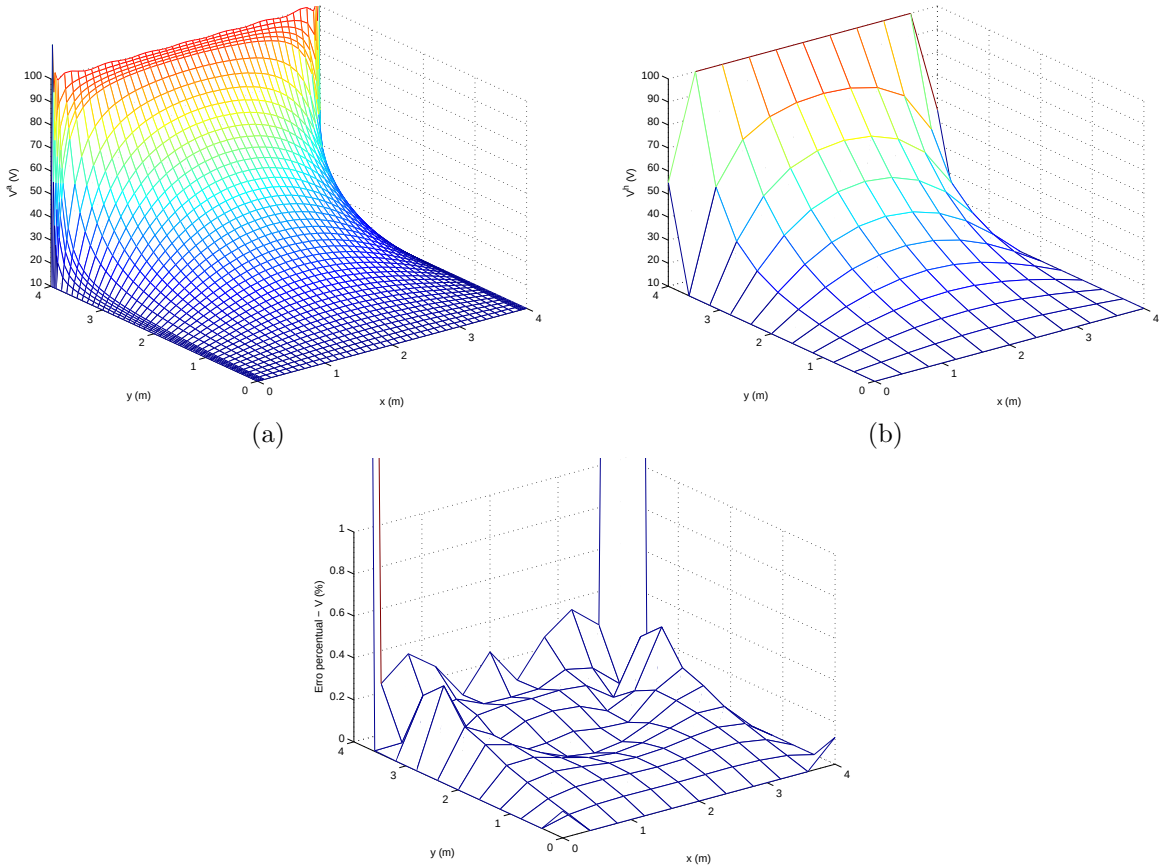


Figura 5.4: Solução para o problema da calha em 2D. (a) Solução analítica, (b) solução utilizando o MLPG1 e (c) erro absoluto percentual, calculado pela Equação 5.2. É usada uma distribuição uniforme de 11×11 nós.

da calha em duas dimensões. Observe que, com essa configuração, elimina-se o problema da descontinuidade do potencial nos vértices superiores pois a variação deste é gradual.

A forma forte para o problema é dada pela Equação 2.26, com $\rho = 0$, $\epsilon_r = 1$ e $h = 0$. A solução analítica é:

$$V^a(x, y) = 90 \frac{\sin(\frac{\pi x}{10}) \sinh(\frac{\pi y}{10})}{\sinh(\pi)} + 10 \tag{5.4}$$

O método sem malha utilizado é o NPIM com uma distribuição de 21×41 nós igualmente espaçados. As funções de forma são geradas utilizando o RPIMp com base polinomial de primeira ordem e função de base radial *spline* cúbica (Equação B.4). Domínios de influência retangulares são usados, com $\alpha_I = 1.5$. A malha de integração é constituída de

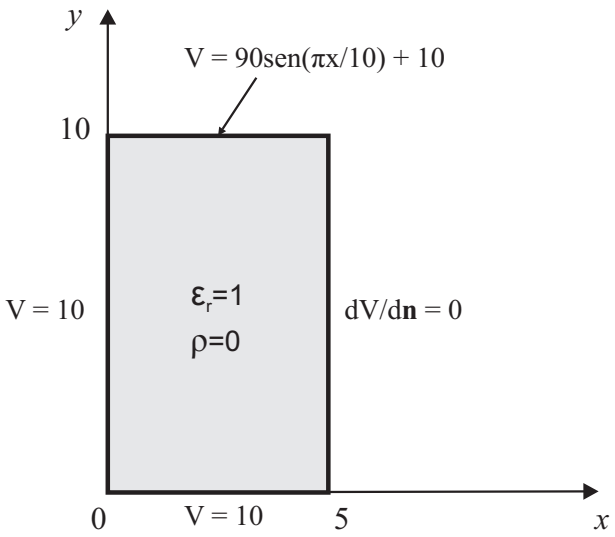


Figura 5.5: Variação da calha em 2D. Não há densidade de carga e a permissividade elétrica relativa do meio é igual a um. A condição de contorno essencial em $y = 10$ é do tipo senoidal, as outras são constantes. O potencial elétrico é dado em $Volts$.

20×40 células retangulares. 5 pontos de quadratura por segmento de célula retangular são usados para as integrais de contorno e 4×4 por célula nas integrais de domínio. A Figura 5.6 mostra a solução e o erro percentual para o problema. Note que as condições de contorno essenciais são satisfeitas exatamente uma vez que as funções de forma do RPIMp possuem a propriedade do delta de Kronecker. A aproximação gerada tem erro absoluto percentual da ordem de 10^{-3}.

Sabe-se que o NPIM é não conforme devido à incompatibilidade de suas funções de forma (Seção 3.3.3). Portanto, é investigada a taxa de convergência do NPIM e comparada com a obtida pelo FEM. Para isso, varia-se a distribuição nodal de 66 a 51681 nós igualmente espaçados, obtendo as soluções para o NPIM e FEM simultaneamente. A despeito do número de nós e células de integração, mantêm-se os mesmos parâmetros usados anteriormente para o NPIM. O número de células de integração variam conforme a distribuição nodal, partindo de 50 chegando a 51200. O erro calculado para as taxas de convergência é dado por:

$$erro = \frac{\sqrt{\int_\Omega (U^h - U^a)^2 d\Omega}}{\sqrt{\int_\Omega (U^a)^2 d\Omega}} \tag{5.5}$$

onde U^h é a solução numérica e U^a a analítica.

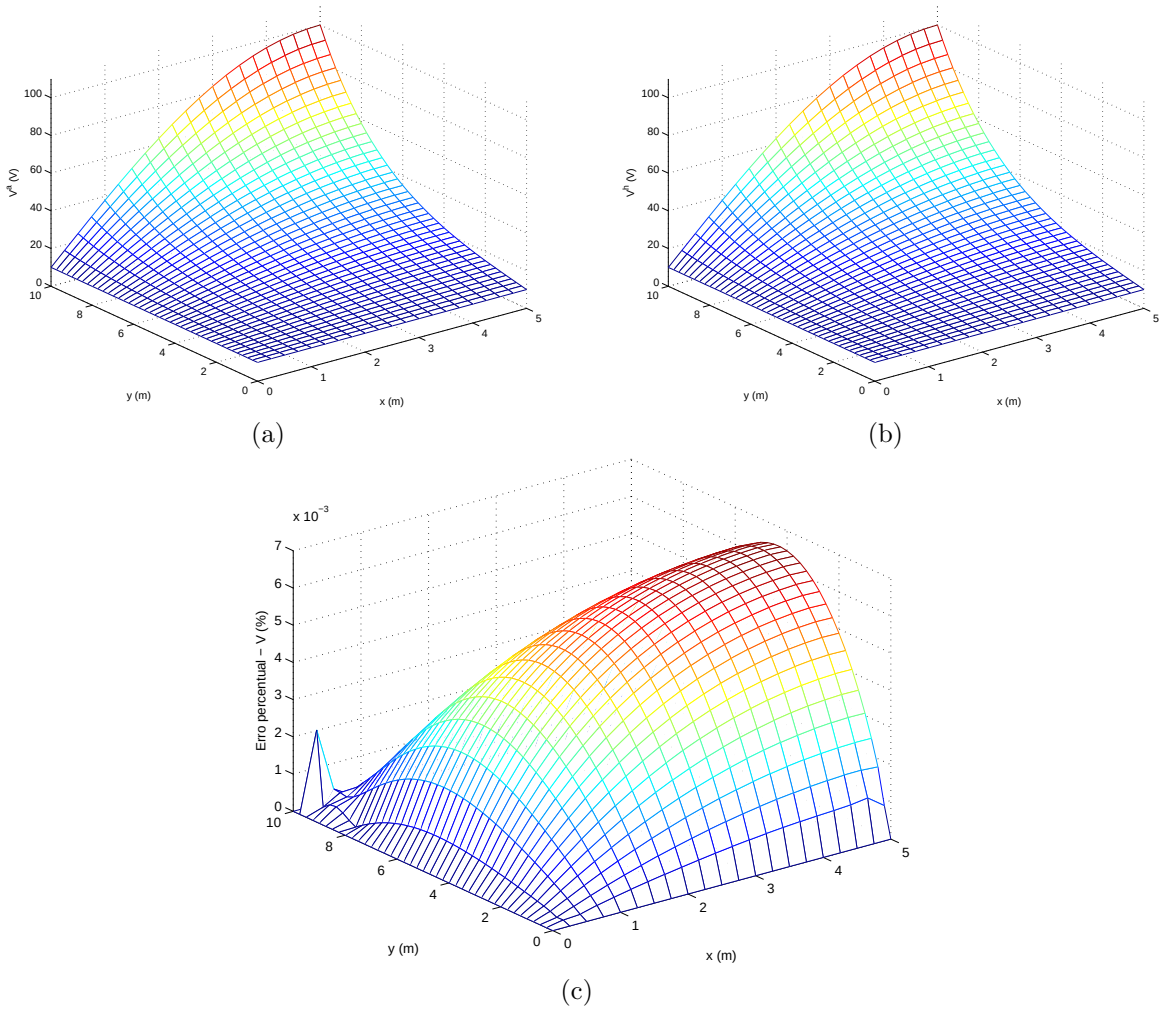


Figura 5.6: Solução para o problema da variação da calha em 2D. (a) Solução analítica, (b) solução utilizando o NPIM e (c) erro absoluto percentual, calculado pela Equação 5.2. É usada uma distribuição de 21×41 nós. A malha de integração é composta de 20×40 células retangulares.

A Figura 5.7 mostra as convergências do NPIM e FEM. As taxas de convergência são obtidas utilizando os últimos quatro pontos do gráfico, de modo a obter o comportamento assintótico. Observe que as taxas são praticamente iguais, aproximadamente 2.0, sendo ligeiramente menor para o NPIM (1.97 para o NPIM e 2.0 para o FEM). Esse resultado é esperado devido à incompatibilidade das funções de forma RPIMp, como discutido anteriormente, e está de acordo com os testes realizados em [32], onde o NPIM alcança taxas de convergência sempre inferiores às do FEM. Todavia, o NPIM produz soluções mais precisas (com menor erro) para todas as distribuições usadas.

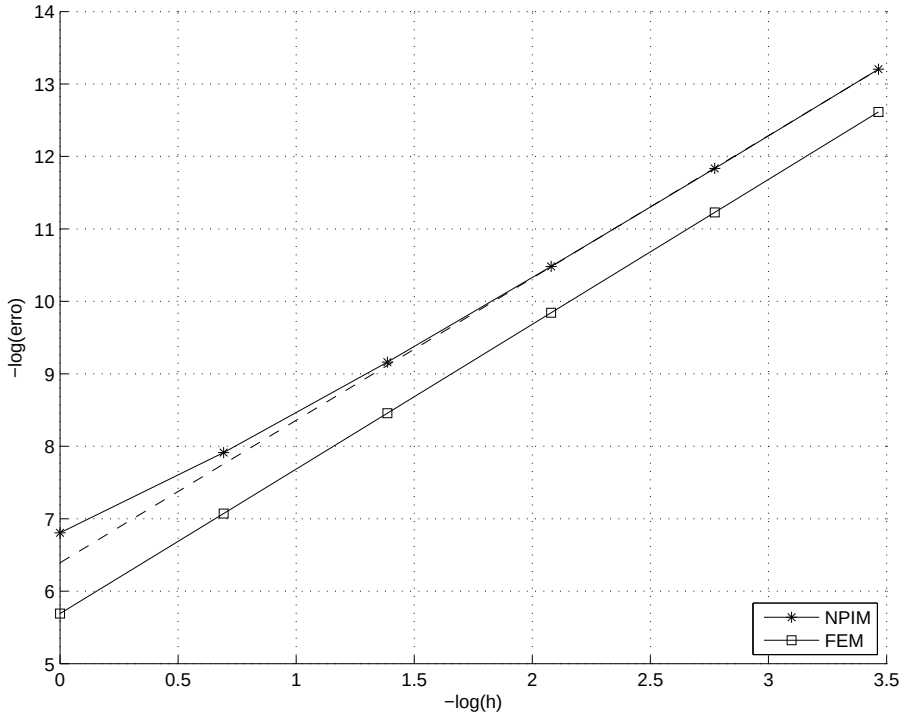


Figura 5.7: Convergência do NPIM para o problema da variação da calha em 2D. h é o espaçamento entre os nós. As taxas de convergência são calculadas usando os últimos 4 pontos. A curva tracejada corresponde à reta ajustada aos últimos 4 pontos da curva de erro do NPIM e é utilizada para obter a taxa de convergência do método sem malha. O erro é dado pela Equação 5.5.

5.3 Capacitor Quadrado 2D

A seção atual traz a aplicação do *Element-Free Galerkin Method* ao problema do capacitor quadrado em duas dimensões. O intuito é mostrar a capacidade do *framework* de tratar geometrias com buracos. Os resultados são comparados com a solução produzida pelo método dos elementos finitos.

A Figura 5.8 mostra o problema do capacitor quadrado em duas dimensões. O condutor externo está a um potencial elétrico de $100V$ e o interno a $10V$. Pretende-se determinar a distribuição do potencial em todo o domínio.

A forma forte para o problema é dada pela Equação 2.26, com $\rho = 0$ e $\epsilon_r = 1$. A solução é obtida usando o método EFG com uma distribuição de 392 nós igualmente espaçados.

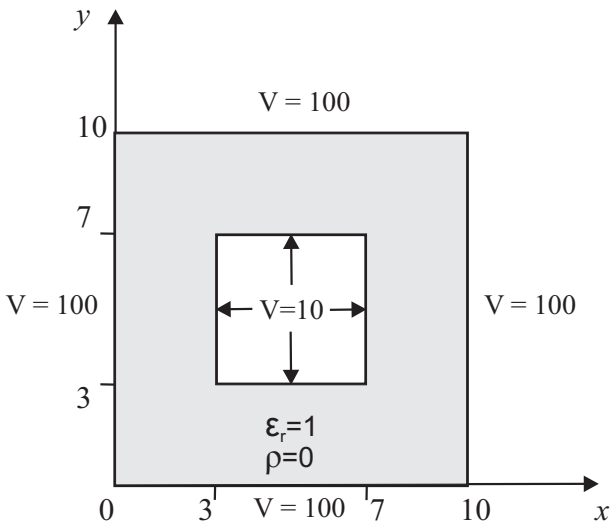


Figura 5.8: Capacitor quadrado em 2D. Não há densidade de carga e a permissividade elétrica relativa do meio é igual a um. O potencial elétrico é dado em $Volts$.

As funções de forma são geradas por meio das funções de Shepard com função de peso *spline* cúbica (Equação B.4), e as condições de contorno de Dirichlet impostas por meio do método das penalidades, com $\alpha = 10^3$. Usam-se domínios de influência circulares e $\alpha_I = 1.5$. A malha de integração é composta de 800 células triangulares. São usados 5 pontos de quadratura por segmento de célula nas integrais de contorno e 3 pontos por célula nas integrais de domínio. Obteve-se, também, a solução utilizando o método dos elementos finitos, com 391 nós e 674 elementos triangulares. A Figura 5.9 mostra as soluções para os dois métodos. Observe que a aproximação criada pelo EFG se assemelha muito à criada pelo FEM.

5.4 Capacitor Cilíndrico 2D

Nesta seção trata-se o problema do capacitor cilíndrico em duas dimensões (Figura 5.10). O problema é semelhante ao do capacitor quadrado, porém as placas agora são circulares. Testa-se, portanto, a capacidade do *framework* de trabalhar com problemas que envolvem geometrias curvas. O condutor externo está a um potencial de $100V$ e o interno a $10V$. Pretende-se determinar a distribuição, no domínio, do potencial elétrico cuja solução analítica é:

$$V^a(r) = \frac{90}{\ln 5} \ln r + 10 \quad [V] \tag{5.6}$$

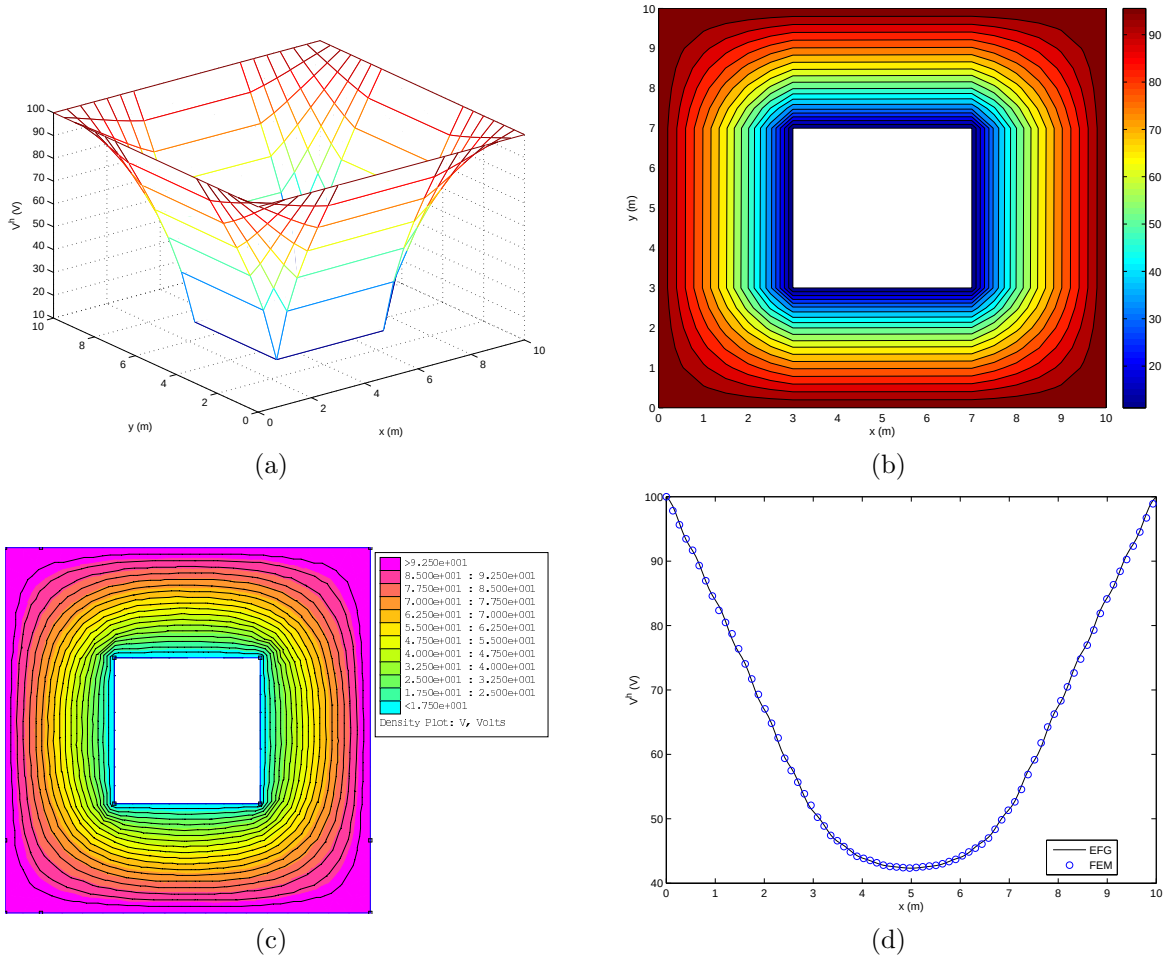


Figura 5.9: Solução para o problema do capacitor quadrado em 2D. Solução utilizando o EFG: (a) Superfície e (b) curvas de nível. (c) Solução utilizando o FEM. Em (d) comparam-se as soluções dos dois métodos no trecho $y = 2$. O EFG utiliza 392 nós igualmente espaçados e uma malha de integração com 800 células triangulares. O FEM usa 391 nós e 674 elementos triangulares.

onde r é a distância radial ao centro $(5, 5)$ do capacitor, com $1 \leq r \leq 5$.

A forma forte do problema é dada pela Equação 2.26, com $\rho = 0$ e $\epsilon_r = 1$. A solução numérica é obtida com o método sem malha MLPG misto (`Hybrid_Mlpg5`), visto na Seção 3.3.4. As funções de forma são geradas utilizando o MLS internamente e o RPIMp nas regiões próximas ao contorno de Dirichlet. Em ambos métodos, usam-se polinômios de primeira ordem e a RBF *spline* cúbica (Equação B.4) na base. Domínios de influência e de quadratura circulares são adotados, com $\alpha_I = 2.0$ e $\alpha_Q = 0.5$. Os nós são obtidos a partir da malha gerada com 466 nós distribuídos de forma irregular. Por isso, calculam-se

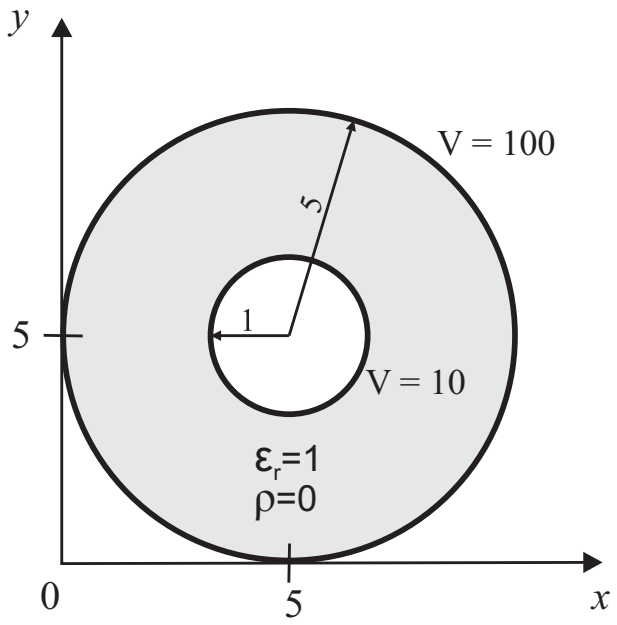
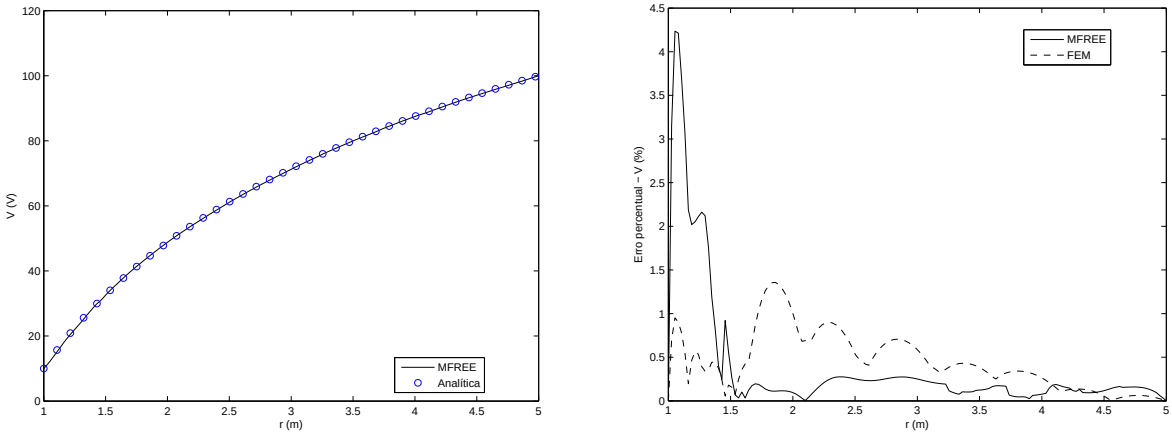


Figura 5.10: Capacitor cilíndrico em 2D. Não há densidade de carga e a permissividade elétrica relativa do meio é igual a um. O potencial elétrico é dado em $Volts$.

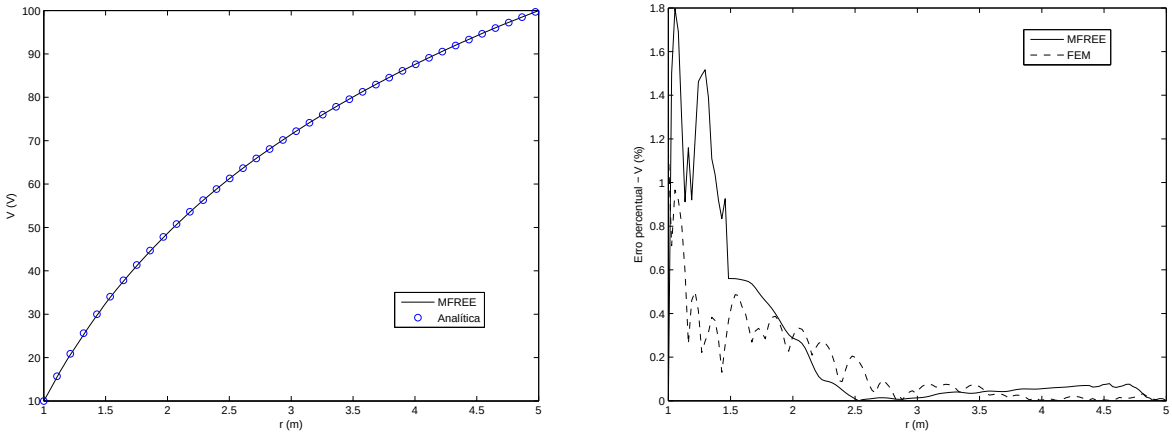
as distâncias locais a partir do estimador `Knn_local_distance`. Para os domínios de influência e quadratura, são utilizados 3 e 5 nós vizinhos, respectivamente. A integração é feita com 4 pontos de quadratura por trecho de quadrante do subdomínio de integração nas integrais de contorno. Integrais de domínio não estão presentes devido ao uso da função de Heaviside como função de teste (versão 5 do MLPG) e porque não há densidades de carga elétrica. Resolve-se, também, o problema com o método dos elementos finitos. A malha é a mesma gerada para obter os 466 nós, resultando em 788 elementos triangulares. A Figura 5.11 mostra a solução utilizando o método misto e os erros absolutos percentuais obtidos com os dois métodos para o potencial elétrico na linha radial $1 \leq r \leq 5$.

O RPIMp é usado para construir as funções de forma onde há condição de contorno essencial, portanto estas são satisfeitas de forma exata. Note que o erro da solução do método sem malha é maior que o da solução do FEM apenas nas regiões próximas às placas, onde as funções de forma são calculadas com o RPIMp no método misto. Os erros dos dois métodos possuem a mesma ordem de grandeza. Resolvendo o problema com os mesmos parâmetros mas com 1997 nós, o que gera um total de 3778 elementos para o FEM, chega-se aos resultados mostrados na Figura 5.12. Observe que os erros novamente têm a mesma ordem de grandeza, sendo que os maiores ocorrem na região próxima à extremidade $r = 1$ para os dois métodos.



(a) (b)

Figura 5.11: Solução para o problema do capacitor cilíndrico em 2D com 466 nós. (a) Solução para o potencial elétrico utilizando método misto (MLS + RPIMp) na linha radial com $1 \leq r \leq 5$, onde r é a distância radial ao centro $(5, 5)$ do capacitor. (b) Erro absoluto percentual (Equação 5.2) para o potencial elétrico obtido com o método misto e com o FEM na mesma linha radial de (a).

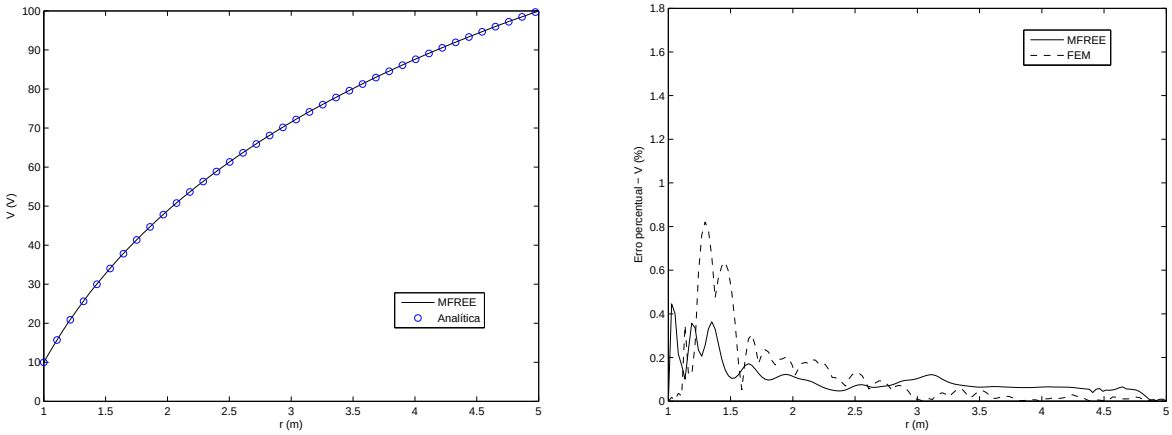


(a) (b)

Figura 5.12: Solução para o problema do capacitor de placas circulares em 2D com 1997 nós. (a) Solução para o potencial elétrico utilizando método misto (MLS + RPIMp) na linha radial com $1 \leq r \leq 5$, onde r é a distância radial ao centro $(5, 5)$ do capacitor. (b) Erro absoluto percentual (Equação 5.2) para o potencial elétrico obtido com o método misto e com o FEM na mesma linha radial de (a).

Como discutido anteriormente, os maiores erros ocorrem próximo à placa interna ($r = 1$). Isso pode ser explicado pelo fato da placa circular ser discretizada por segmentos de reta, o que gera erros de aproximação na solução numérica. Uma maneira de amenizar esse problema é aumentar a discretização da placa, isto é, representá-la por um número maior

de segmentos, o que corresponde a elevar o número de nós (método sem malha) e de elementos (método dos elementos finitos) ligados à placa interna. Efetuando tal procedimento, resolveu-se novamente o problema com 2559 nós e um total de 4614 elementos para o FEM. Os resultados são mostrados na Figura 5.12. Observe que os erros percentuais na região próxima à placa interna diminuíram por um fator de 4 para o método sem malha e 1.5 para o método dos elementos finitos, comprovando que a origem dos altos erros nessa região é devido à discretrização da placa circular interna.



(a) (b)

Figura 5.13: Solução para o problema do capacitor de placas circulares em 2D utilizando 2559 nós, com discretização maior da placa circular interna em relação à configuração cujos resultados são mostrados na Figura 5.13. (a) Solução para o potencial elétrico utilizando método misto (MLS + RPIMp) na linha radial com $1 \leq r \leq 5$, onde r é a distância radial ao centro $(5, 5)$ do capacitor. (b) Erro absoluto percentual (Equação 5.2) para o potencial elétrico obtido com o método misto e com o FEM na mesma linha radial de (a).

5.5 Capacitor de Placas Paralelas 2D

Outra aplicação muito comum em eletrostática é o capacitor de placas paralelas. Até o momento resolveu-se problemas cujo domínio é formado por apenas um tipo de material. Nesta seção investiga-se problemas com dois materiais de propriedade físicas distintas e a aplicação das técnicas para garantir as condições de continuidade na interface dos meios, como discutido no Capítulo 3. Para isso, duas configurações são usadas: o interior do capacitor sendo constituído de dois dielétricos (1) dispostos lado a lado e (2) posicionados de forma concêntrica.

5.5.1 2 Materiais Dispostos Lado a Lado

A Figura 5.14 mostra o capacitor de placas paralelas com dois dielétricos dispostos lada a lado, sendo que o meio 2 possui permissividade elétrica relativa cinco vezes maior que o meio 1. A solução analítica do problema é:

$$V^a(x,y) = \begin{cases} -\frac{75}{2}x + 100 & \text{se} \quad 0 \leq x \leq 2 \\ -\frac{15}{2}x + 40 & \text{se} \quad 2 \leq x \leq 4 \end{cases} \tag{5.7}$$

$$E^a(x,y) = \begin{cases} \frac{75}{2} & \text{se} \quad 0 \leq x < 2 \\ \frac{15}{2} & \text{se} \quad 2 < x \leq 4 \end{cases} \tag{5.8}$$

em que V^a é dados em $Volts$ e E^a em V/m. Novamente, a forma forte para o problema é dada pela Equação 2.26, com $\rho = 0$, $\epsilon_{r1} = 1$ e $\epsilon_{r2} = 5$.

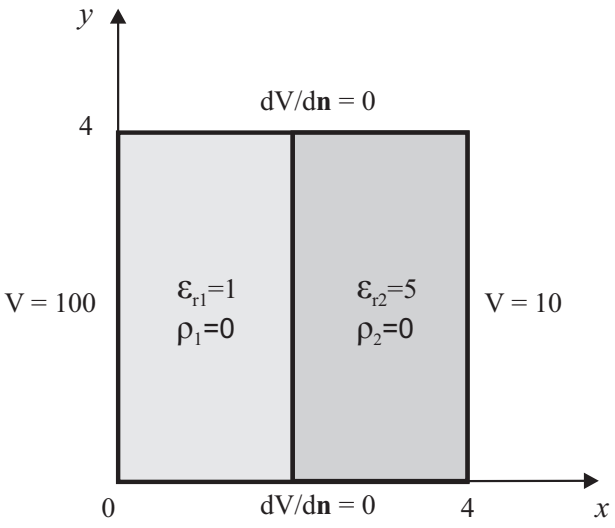


Figura 5.14: Capacitor de placas paralelas com dois dielétricos lado a lado em 2D. Não há densidade de carga e a permissividade elétrica relativa do meio 2 é cinco vezes maior que a do meio 1. O potencial elétrico é dado em $Volts$.

A solução é obtida utilizando o método MLPG5 com subdomínios de integração tangenciando as fronteiras de materiais, de Dirichlet e de Neumann, como visto nas Seções 3.3.2.4 e 3.3.2.5. No MFree Framework, o método é representado pela classe `Tangent_mlpg5`. Domínios de influência e quadratura circulares são utilizados, com $\alpha_I = \alpha_Q = 1.5$. As funções de forma são construídas via MLS com polinômios lineares e função de peso *spline* cúbica (Equação B.4). Efetua-se a integração com 1 ponto de quadratura por trecho de

quadrante do subdomínio de integração circular nas integrais de contorno. A formulação não exige integração de domínio. 11×11 nós são distribuídos uniformemente. As condições de contorno essenciais e naturais, assim como as condições de continuidade, são impostas pelo método de colocação. A Figura 5.15 mostra a solução numérica obtida pelo MLPG5 e a comparação com a solução analítica em $y = 2$. O erro percentual é da ordem de 10^{-11} (Figura 5.15d), isto é, numericamente nulo. Além disso, o campo elétrico sofre uma descontinuidade na interface de materiais como esperado (Figura 5.15c), confirmando a eficácia da metodologia de imposição das condições de continuidade entre diferentes materiais.

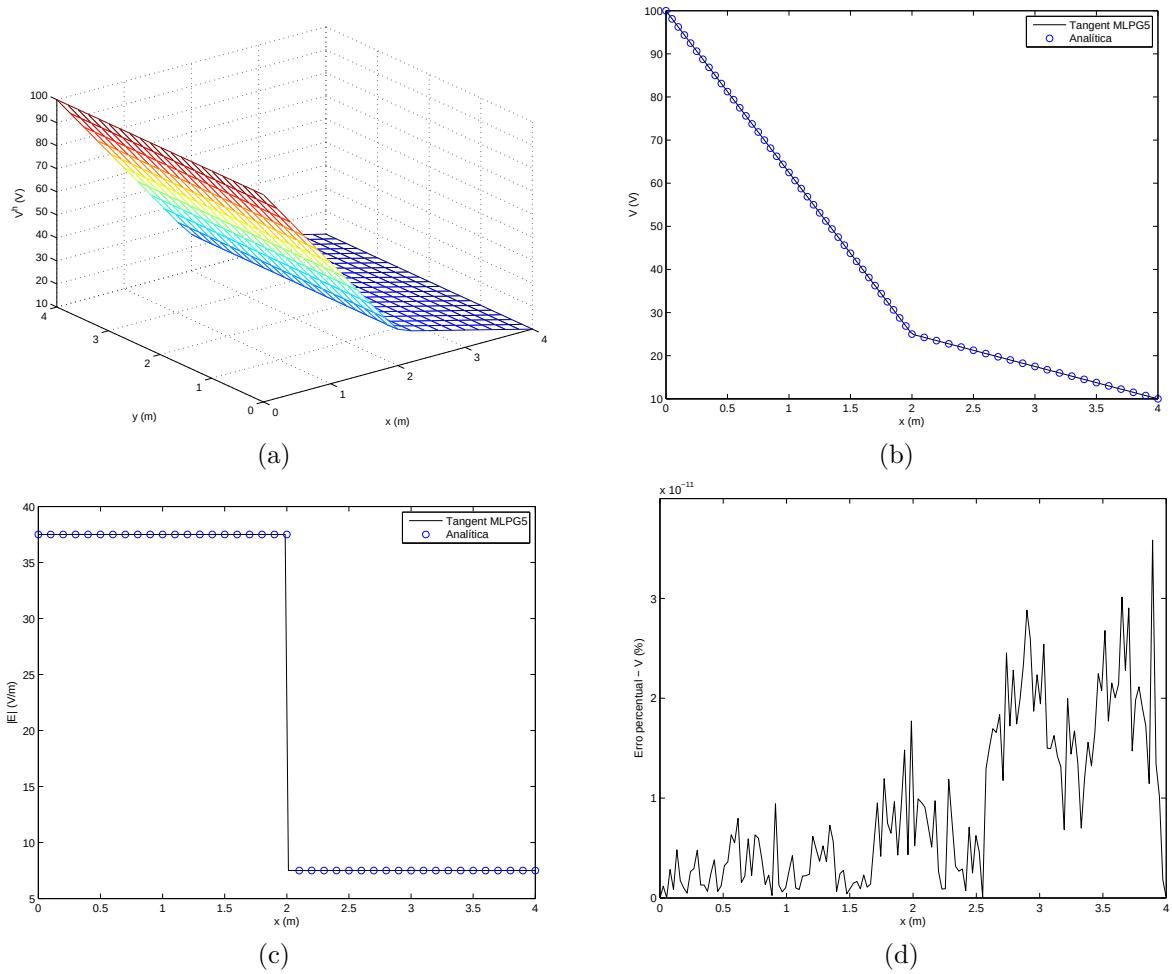


Figura 5.15: Solução para o problema do capacitor de placas paralelas com dois materiais lado a lado em 2D. (a) Distribuição do potencial elétrico calculado pelo MLPG5. Comparação entre a solução numérica e a analítica para (b) o potencial elétrico e (c) o campo elétrico no trecho $y = 2$. (d) Erro absoluto percentual, dado pela Equação 5.2, para o potencial elétrico. O MLPG5 utiliza 121 nós igualmente espaçados. As condições de contorno essenciais são impostas pelo método de colocação.

5.5.2 2 Materiais Concêntricos

Aborda-se, nesta seção, o problema do capacitor de placas paralelas com dois dielétricos posicionados de forma concêntrica, isto é, um no centro e outro em sua volta, como mostra a Figura 5.16. O meio 2 possui permissividade elétrica relativa dez vezes maior que o meio 1. Não há densidades de carga no problema. A forma forte é dada pela Equação 2.26, com $\rho = 0$, $\epsilon_{r1} = 1$ e $\epsilon_{r2} = 10$.

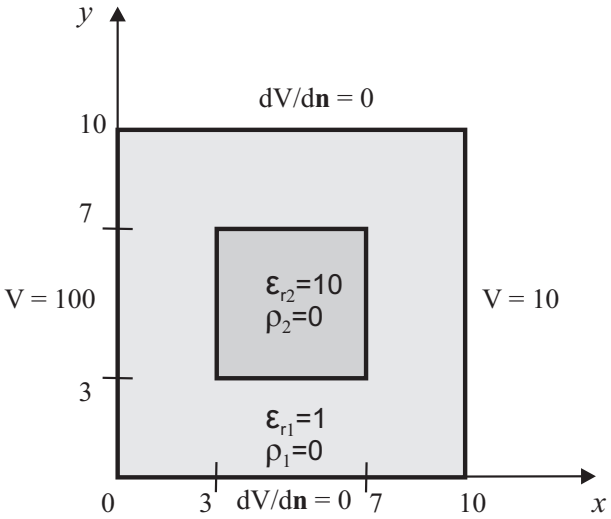


Figura 5.16: Capacitor de placas paralelas com dois dielétricos concêntricos em 2D. Não há densidade de carga e a permissividade elétrica relativa do meio 2 é dez vezes maior que a do meio 1. O potencial elétrico é dado em $Volts$.

O método sem malha empregado é o LPIM5 com tratamento das condições de continuidade de materiais utilizando o critério da visibilidade generalizado (Seção 3.3.3.3). Constroem-se as funções de forma por meio do RPIMp com polinômios de primeira ordem e RBF *spline* cúbica (Equação B.4). Os domínio de influência e quadratura são retangulares, sendo $\alpha_I = 1.5$ e $\alpha_Q = 0.5$. A integração é executada com 1 ponto de quadratura por segmento do subdomínio de integração nas integrais de contorno, e integrais de domínio não existem. Distribuem-se, uniformemente, 21×21 nós. O problema também é resolvido utilizando o FEM com duas malhas: uma de 438 nós e 795 elementos triangulares e outra muita mais densa de 39947 nós e 79043 elementos. A solução do FEM com a malha mais densa é adotada como referência. A Figura 5.17 mostra os resultados. Observe que as soluções de ambos métodos se assemelham, inclusive em $y = 5$. Note, também, que a continuidade da aproximação é atendida para o potencial elétrico (Figura 5.17c) e o campo

elétrico sofre descontinuidade (Figura 5.17d) nas interfaces entre materiais. Entretanto, o módulo do campo elétrico obtido pelo FEM é mais condizente com a solução referência que o obtido pelo LPIM5 com o método da visibilidade generalizado.

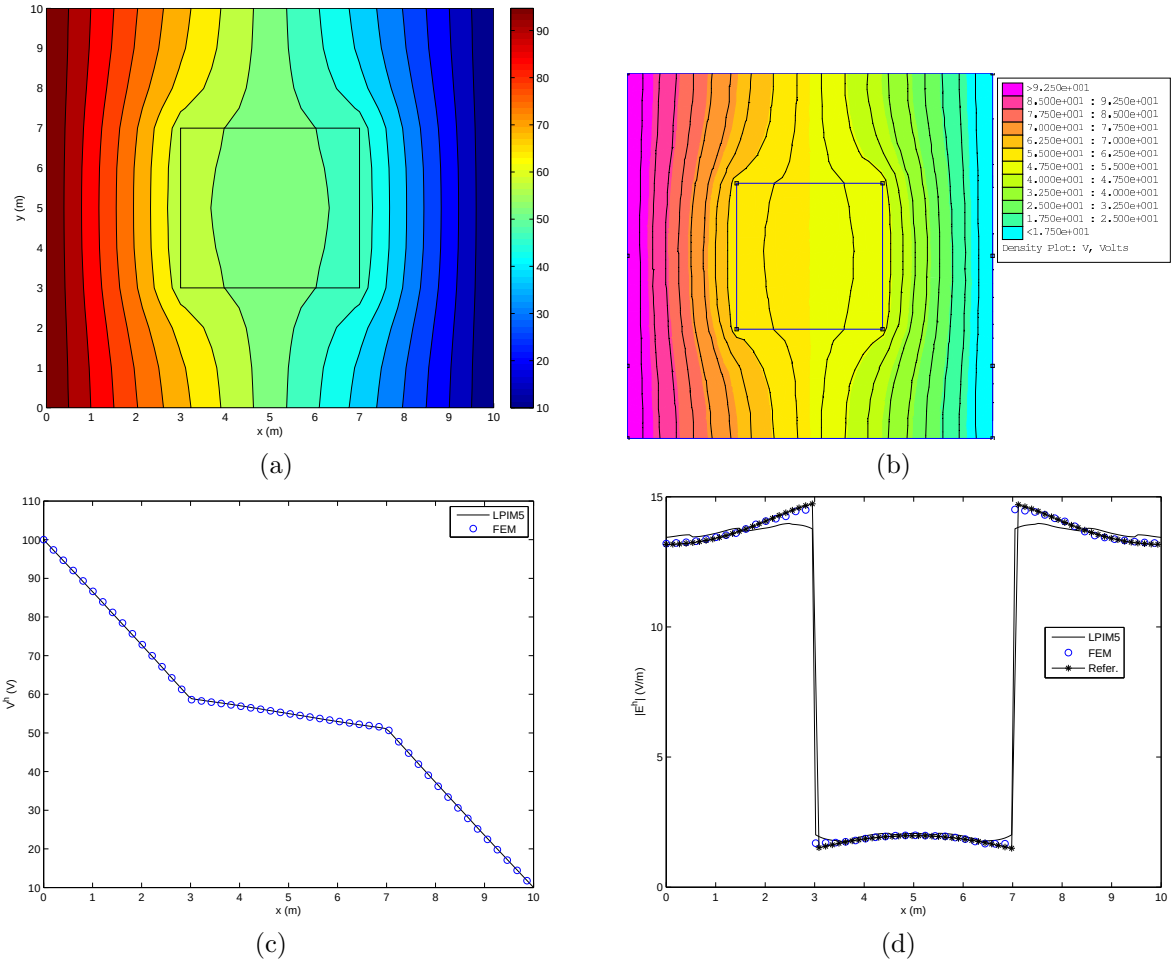


Figura 5.17: Solução para o problema do capacitor de placas paralelas com dois materiais concêntricos em 2D. Curvas de nível para o potencial elétrico calculado (a) pelo LPIM5 e (b) pelo FEM. Comparam-se as soluções dos dois métodos no trecho $y = 5$ para (c) o potencial elétrico e (d) o campo elétrico. O LPIM5 utiliza 441 nós igualmente espaçados. O FEM usa 438 nós e 795 elementos triangulares.

5.6 Campo Magnético Uniforme 2D

O objetivo da presente seção é aproximar um campo magnético uniforme de indução unitária em uma região utilizando o NPIM com dois tipos de funções de forma, RPIM e

RPIMp. Como o RPIM gera aproximações inconsistentes (Seção 3.2.4), não é capaz de produzir termos constantes na solução. Dessa maneira, espera-se que não seja capaz de reproduzir o campo uniforme, ao contrário do RPIMp, que é consistente (Seção 3.2.5).

O problema é mostrado na Figura 5.18. Não há correntes elétricas presentes e a permeabilidade magnética relativa do meio é 1. A forma forte é dada pela Equação 2.37, com $\mu_r = 1$ e $J_z = 0$. O campo a ser aproximado possui indução magnética $\overrightarrow{B} = 1\hat{\mathbf{x}} \quad [T]$.

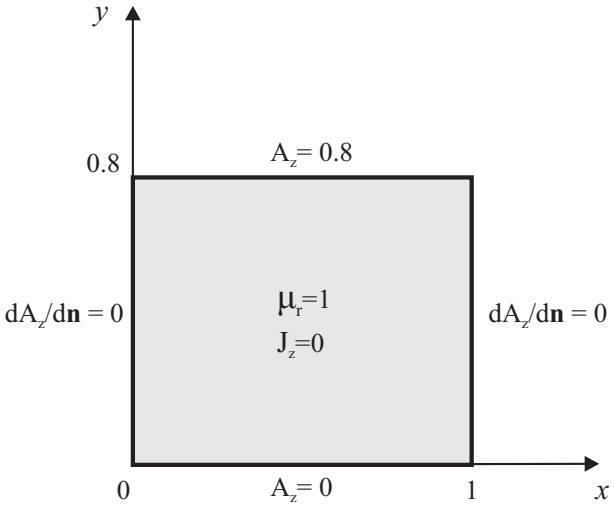


Figura 5.18: Domínio para um campo magnético uniforme em 2D. Não há densidade de corrente elétrica e a permeabilidade magnética relativa do meio é igual a 1. O potencial vetor magnético na direção z é dado em Wb/m.

O problema é resolvido utilizando uma distribuição de 6×5 nós igualmente espaçados e uma malha de integração de 5×4 células retangulares. Os domínios de influência são circulares e $\alpha_I = 1.5$. A integração é realizada com 5 pontos de quadratura por segmento de célula nas integrais de contorno e 2×2 por célula nas integrais de domínio. Constróem-se as funções de forma com o RPIM e RPIMp. Em ambos, é usada a RBF *spline* cúbica, dada pela Equação B.4. Polinômios de primeira ordem são usados pelo RPIMp. A Figura 5.19 mostra o campo magnético aproximado com os dois tipos de função de forma e o erro absoluto percentual. Como já previsto, a aproximação usando RPIM não consegue reproduzir o campo de maneira adequada pois não satisfaz a partição da unidade, isto é, não reproduz termos constantes da solução, gerando um campo desalinhado e erros muito altos (superiores a 100%). Já a solução com o RPIMp produz corretamente o campo

uniforme (com erros inferiores a 0.1%) pois tem ordem de consistência C^1, sendo capaz de reproduzir termos de lineares e constantes da solução.

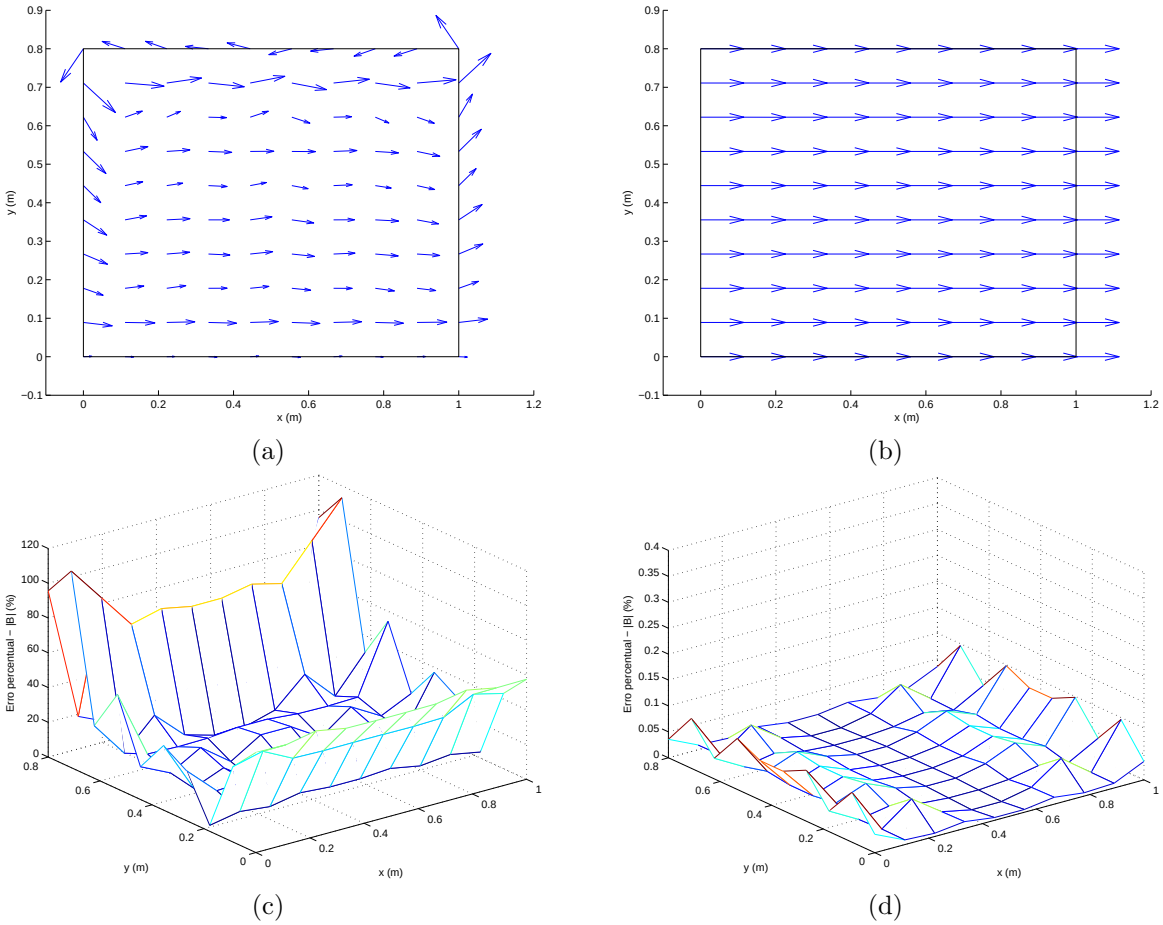


Figura 5.19: Solução para o problema do campo magnético uniforme em 2D. A solução é obtida utilizando o NPIM. Campo magnético aproximado com funções de forma construídas pelo (a) RPIM e (b) RPIMp. Erro absoluto percentual (Equação 5.2) para o módulo da indução magnética aproximada com funções de forma (c) RPIM e (d) RPIMp. O NPIM utiliza 30 nós igualmente espaçados e uma malha de integração com 20 células retangulares.

5.7 Eletroímã 2D

Por fim, resolve-se o problema do eletroímã em duas dimensões. Trata-se de um problema mais complicado que os anteriores pois envolve três materiais distintos, interfaces entre

dois e três meios e fontes de correntes elétricas. Utiliza-se o LPIM5 com o método da visibilidade generalizado para tratar as descontinuidades de materiais (Seção 3.3.3.3). A taxa de convergência do LPIM5 também é obtida.

O problema é mostrado na Figura 5.20. A forma forte é dada pela Equação 2.37, com $\mu_{r1} = \mu_{r3} = 1$, $\mu_{r2} = 1000$, $J_{z1} = J_{z2} = 0$ e $J_{z3} = 10^5$, com J_z dado em A/m^2.

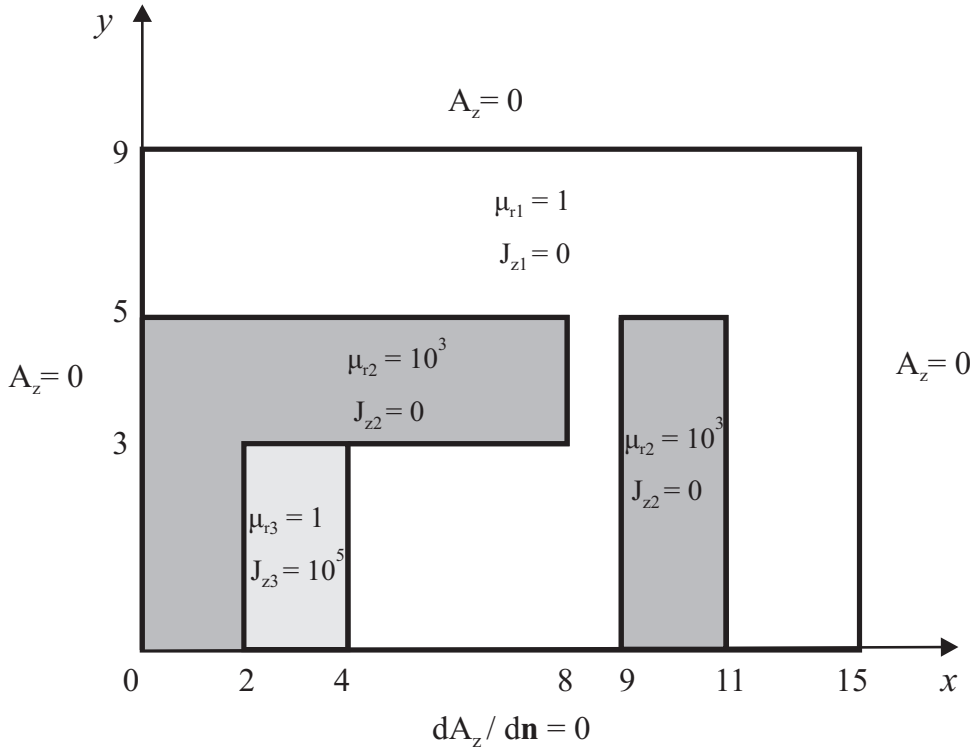
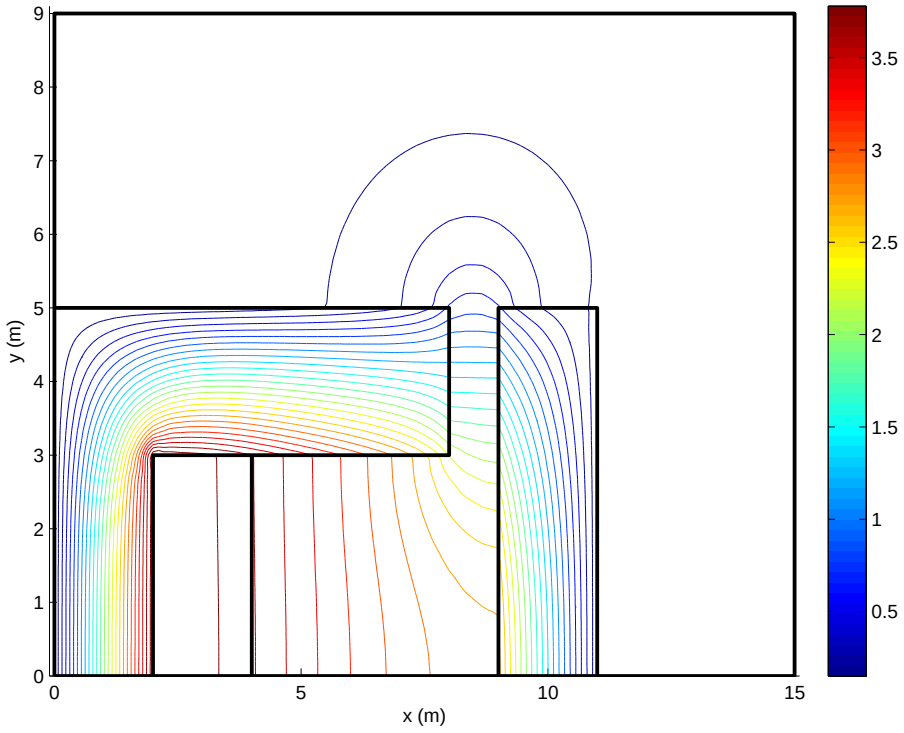


Figura 5.20: Eletroímã em 2D. A densidade de corrente elétrica e o potencial vetor magnético, ambos na direção z, são dados em A/m^2 e Wb/m, respectivamente. O material ferromagnético possui permeabilidade magnética relativa $\mu_{r2} = 10^3$.

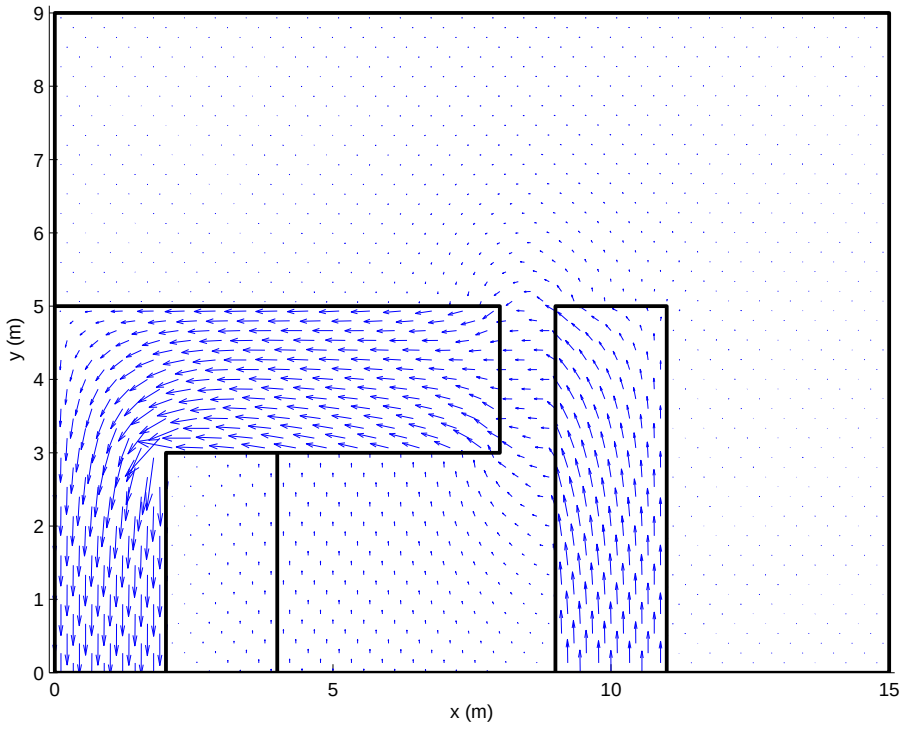
Utilizam-se 1288 nós distribuídos com mesmo espaçamento. As funções de forma são geradas pelo RPIMp com base polinomial de primeira ordem e RBF *spline* cúbica (Equação B.4). Domínios de influência e quadratura retangulares são empregados, com $\alpha_I = 1.5$ e $\alpha_Q = 0.5$. A integração é realizada com 4 pontos de quadratura em cada segmento do subdomínio de integração para integrais de contorno e 1 ponto por subdomínio de quadratura nas integrais de domínio. A Figura 5.21 mostra a solução para as linhas de fluxo magnético e indução magnética $\overrightarrow{B}$ distribuídas no espaço. Observe que as linhas se concentram no material ferromagnético e saem deste perpendicularmente, como deveria

ser. A indução magnética é circulante no núcleo do eletroímã com sentido dado por uma corrente elétrica na direção $+\hat{\mathbf{z}}$, o que está de acordo com a densidade positiva J_{z3}.

Investiga-se a taxa de convergência para o método. A distribuição nodal varia entre 589 e 34945 nós igualmente espaçados. Para cada distribuição, calcula-se o erro, dado pela Equação 5.5, da aproximação usando o LPIM5 e o FEM. A solução referência ("analítica" aproximada U^a) é tomada como a gerada pelo método dos elementos finitos em uma malha contendo 10^6 nós. A Figura 5.22 mostra a convergência do LPIM5 e FEM para o problema do eletroímã. O LPIM5 tem taxa de convergência um pouco superior à do FEM (1.39 contra 1.34). Os resultados indicam que o método sem malha é mais preciso que o de elementos finitos, para o mesmo número de nós.



(a)



(b)

Figura 5.21: Solução para o problema do eletroímã em 2D. (a) Linhas de fluxo magnético e (b) indução magnética $\overrightarrow{B}$. O LPIM5 utiliza 1288 nós igualmente espaçados e subdomínios de integração retangulares. As descontinuidades nas interfaces entre materiais são tratadas com o método da visibilidade generalizado.

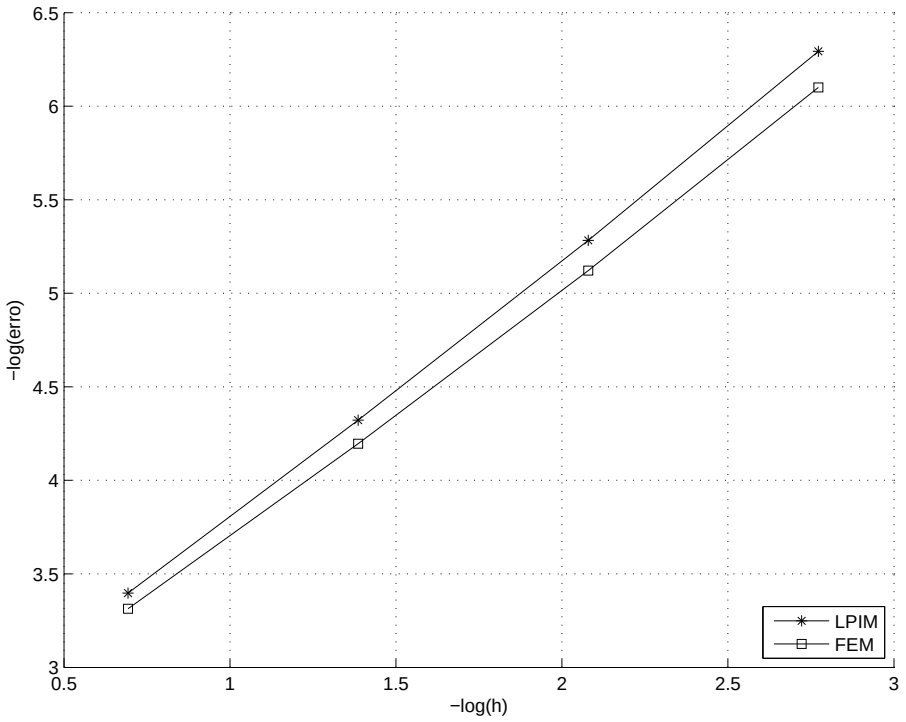


Figura 5.22: Convergência do LPIM5 para o problema do eletroímã em 2D. h é o espaçamento entre os nós. O erro é dado pela Equação 5.5.

Capítulo 6

Conclusão

No presente trabalho desenvolveu-se um *framework*, chamado MFree Framework, destinado a construção de aplicações de *software* para solucionar problemas eletromagnéticos através de métodos sem malha. O *framework* foi construído usando a linguagem de programação C++ com base no paradigma da programação genérica, resultando em uma ferramenta com as características flexibilidade e eficiência computacional.

Inicialmente, uma revisão dos principais conceitos a respeito de *frameworks* e programação genérica foi apresentada. Além disso, mostrou-se um resumo da teoria eletromagnética aplicada a problemas de eletromagnetismo, com ênfase nos eletrostáticos e magnetostáticos. Suas formas fortes foram apresentadas assim como uma generalização para problemas estáticos em uma e duas dimensões.

O MFree Framework, em sua versão atual, implementa os métodos sem malha considerados mais importantes e de expressão na literatura, a saber o *Element-Free Galerkin Method*, *Meshless Local Petrov-Galerkin Method*, *Nonconforming Point Interpolation Method* e *Local Point Interpolation Method*. Também disponibiliza diferentes técnicas para construção de funções de forma tais como *Moving Least Squares*, funções de *Shepard* e *Point Interpolation Method* e suas variações. Uma revisão mais detalhada de métodos sem malha foi mostrada, incluindo os tipos de funções de forma e os métodos citados anteriormente, assim como os diferentes tipos de domínios de suporte, de processos de

integração, as formas de imposição das condições de contorno e o tratamento das descontinuidades da aproximação entre diferentes materiais.

Uma visão geral da arquitetura do *framework* foi apresentada. Viu-se que um grande número de combinações entre métodos, funções de forma, processos de integração e formulações é permitido devido ao caráter genérico do MFree Framework, que possui parametrização para, praticamente, todas as estruturas de dados através de *traits*. Para utilizar um método com um determinado tipo de função de forma, de integração e de formulação, estes devem ser compatíveis entre si. Esta compatibilidade é estabelecida por meio do atendimento dos requisitos impostos pelos *conceitos* existentes no *framework*.

Os pontos de extensão do MFree Framework se encontram justamente nos *conceitos*, isto é, através destes novos tipos podem ser desenvolvidos pelo programador, incluídos e usados com os já disponíveis. Ou seja, outros métodos sem malha e procedimentos de contrução de funções de forma, por exemplo, podem ser facilmente implementados utilizando as ferramentas já disponíveis pelo *framework*. Todas essas características conferem ao MFree Framework um alto grau de flexibilidade e o tornam uma ferramenta importante para o desenvolvimento de novas metodologias em métodos sem malha.

No *framework*, estruturas de dados e algoritmos da biblioteca CGAL são utilizados para solucionar problemas de geometria computacional que surgem durante o processo de solução de um problema com métodos sem malha. Além disso, representações e operações de matrizes são feitas utilizando a biblioteca MTL4. As duas bibliotecas citadas anteriormente são construídas, assim como o MFree framework, com base em programação genérica e possuem estruturas de dados e algoritmos eficientes, o que confere ao *framework* uma outra característica importante: eficiência computacional.

Para validar as implementações e ilustrar as capacidades do *framework* desenvolvido, resolveram-se alguns problemas comuns em eletromagnetismo, como o do capacitor cilíndrico, quadrado e de placas paralelas, o problema da calha e do eletroímã. Em tais problemas estiveram presentes as seguintes características: domínios compostos de um, dois e três materiais distintos, geometrias formadas por segmentos e curvas, com e sem buracos presentes, e distribuições regulares e irregulares de nós. Todos os métodos sem malha disponíveis foram utilizados, assim como diferentes tipos de funções de forma, integração por subdomínio local e por malha de fundo e diferentes maneiras de impor condições de

contorno essenciais e de tratar as descontinuidades de materiais. Os resultados mostraram concordância e boa precisão entre os obtidos por métodos sem malha e as soluções analíticas, quando disponíveis, e os gerados pelo tradicional método dos elementos finitos.

Trabalhos Futuros

Atualmente, para criação do domínio do problema no MFree Framework, geometria e condições de contorno devem ser codificadas na aplicação. Como visto, essa forma não é adequada, e tais informações deveriam ser lidas de um arquivo externo. Para isso, é necessário estabelecer uma especificação (um formato) de como devem ser apresentadas em arquivo para que, posteriormente, possam ser lidas pelo *framework*. Essa é uma das prioridades para as versões futuras.

Somente problemas estáticos foram solucionados com o *framework*. Uma possiblidade interessante é aplicá-lo, também, em problemas com variação temporal, como os problemas quase estáticos e os de propagação de ondas eletromagnéticas. Exemplos de aplicação incluem [56, 42].

Um aspecto interessante que este trabalho não trata é a possibilidade de que os meios sejam não lineares, isto é, que suas propriedades físicas dependam dos valores do campo eletromagnético. Um exemplo é o que ocorre com a permeabilidade magnética em determinados materiais, que depende do valor do campo magnético a que estão sujeitos. Para resolver problemas desse tipo, deve-se implementar *solvers* de sistemas de equações não lineares. Um exemplo de como solucionar tais problemas no contexto do método de elementos finitos pode ser vista em [39].

O *framework* é construído em C++ com programação genérica visando ser flexível e eficiente. Um caminho para aumentar o desempenho computacional é utilizar algoritmos paralelizados, visto que máquinas multinúcleos são cada vez mais acessíveis. Em [18] são apresentadas maneiras de paralelizar o MLPG e é mostrado que as versões paralelas executam até 1,96 vezes mais rapidamente em processadores com dois núcleos e até 3,78 vezes em computadores com quatro núcleos quando comparados às versões sequênciais.

Outra possiblidade é a utilização de unidades de processamento gráfico (GPU) de placas de vídeo. As GPUs se destinavam, inicialmente, a processamento gráfico e jogos, e vêm ganhando espaço na área de computação de alto desempenho. Um exemplo de paralelização do método sem malha *Radial Point Interpolation Method* utilizando GPUs pode ser visto em [41].

A versão atual do MFree Framework soluciona problemas em até duas dimensões. Uma perspectiva interessante para futuros desenvolvimentos é expandir a aplicabilidade a problemas em três dimensões, o que tornaria o *framework* mais completo. Exemplos da utilização do EFG e do MLPG em problemas eletrostáticos 3D são mostrados em [47] e [43], respectivamente. Alguns aspectos dos procedimentos para aplicação de métodos sem malha em três dimensões podem ser vistos em [51].

Publicações

Durante o período do mestrado, uma publicação diretamente associada ao trabalho de dissertação foi gerada:

- Lima, N. Z., Mesquita, R. C., Junior, M. L. A. A framework for meshless methods using generic programming. In: 14th Biennial IEEE Conference on Electromagnetic Field Computation (CEFC), 2010.

Além desta, outras duas foram aceitas para publicação:

- Lima, N. Z., Fonseca, A. R., Mesquita, R. C. Application of Local Point Interpolation Method to Electromagnetic Problems with Material Discontinuities using a new Visibility Criterion. In: Compumag, 2011.
- Lima, N. Z., Mesquita, R. C., Facco, W. G., Moura, A. S., Silva, E. J. The Non-conforming Point Interpolation Method applied to Electromagnetic Problems. In: Compumag, 2011.

Dois outros trabalhos não ligados diretamente ao tema de dissertação de mestrado foram desenvolvidos e aceitos para publicação como consequência da colaboração com outros alunos pertencentes ao Grupo de Otimização e Projeto Assistido por Computador (GOPAC):

- Facco, W. G., Silva, E. J., Moura, A. S., Lima, N. Z., Saldanha, R. R. Handling Material Discontinuities in the Generalized Finite Element Method to Solve Wave Propagation Problems. In: Compumag, 2011.
- Moura, A. S., Saldanha, R. R., Silva, E. J., Lisboa, A. C., Facco, W. G., Lima, N. Z. A Recursive Sparsification of the Inverse Hodge Matrix. In: Compumag, 2011.

Referências Bibliográficas

[1] The UMFPACK home page. `http://www.cise.ufl.edu/research/sparse/umfpack/`, 2011. Acesso em Janeiro de 2011. [citado na(s) páginas(s) 81, 88]

[2] Standard Template Library Programmer's Guide. `http://www.sgi.com/tech/stl/`, Acesso em janeiro de 2011. [citado na(s) páginas(s) 13]

[3] Andrei Alexandrescu. Traits: The else-if-then of Types (C++ Report). Technical report, April 2000. [citado na(s) páginas(s) 11]

[4] Apache. Apache Struts home page. `http://struts.apache.org/`, Acesso em janeiro de 2011. [citado na(s) páginas(s) 8]

[5] S N Atluri and S Shen. The meshless local Petrov-Galerkin (MLPG) method - A simple and less-costly alternative to the finite element and boundary element methods . CMES - Computer Modeling in Engineering and Sciences, 3(1):11–51, 2002. [citado na(s) páginas(s) 3, 62]

[6] S. N. Atluri and T. Zhu. A new Meshless Local Petrov-Galerkin (MLPG) approach in computational mechanics. Computational Mechanics, 22(2):117 – 127, 1998. [citado na(s) páginas(s) 3, 61]

[7] T. Belytschko, Y. Y. Lu, and L. Gu. Element-free Galerkin methods. International Journal for Numerical Methods in Engineering, 37(2):229 – 256, 1994. [citado na(s) páginas(s) 3, 52]

[8] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars. Computational Geometry - Algorithms and Applications. Springer, 3rd edition, 2008. [citado na(s) páginas(s) 94, 96, 100, 103]

[9] William E. Boyce and Richard C. DiPrima. Elementary Differential Equations and Boundary Value Problems. John Wiley & Sons, 7th edition, 2001. [citado na(s) páginas(s) 1]

[10] The CGAL home page. `http://www.cgal.org`, 2009. Acesso em Julho de 2009. [citado na(s) páginas(s) 13, 88]

[11] E.H.R. Coppoli, R.C. Mesquita, and R.S. Silva. Field-circuit coupling with Element-Free Galerkin Method. page 1, May 2010. [citado na(s) páginas(s) 1]

[12] L. W. Cordes and B. Moran. Treatment of material discontinuity in element-free Galerkin method. Computer Methods in Applied Mechanics and Engineering, 139:75–89, 1996. [citado na(s) páginas(s) 59]

[13] B.C. Correa, E.J. Silva, A.R. Fonseca, D.B. Oliveira, and R.C. Mesquita. Meshless Local Petrov-Galerkin in solving microwave guide problems. pages 1 –1, May 2010. [citado na(s) páginas(s) 1]

[14] Timothy A Davis. Algorithm 832: UMFPACK V4.3 - an unsymmetric-pattern multifrontal method. ACM Transactions on Mathematical Software (TOMS), 30(2):196–199, 2004. [citado na(s) páginas(s) 81]

[15] H. M. Deitel. C++ How to Program. Prentice Hall, 5th edition, 2005. [citado na(s) páginas(s) 10]

[16] Richard O. Duda, Peter E. Hart, and David G. Stork. Pattern Classification. Wiley-Interscience, 2nd edition, 2000. [citado na(s) páginas(s) 95]

[17] M. E. Fayad, D. C. Schmidt, and R. E. Johnson. Building application frameworks: object-oriented foundations of framework design. John Wiley & Sons, 1999. [citado na(s) páginas(s) 6, 7]

[18] A. R. Fonseca. Algoritmos Eficientes em Métodos sem Malha. Exame de qualificação, Universidade Federal de Minas Gerais, 2009. [citado na(s) páginas(s) 3, 70, 147]

[19] A. R. Fonseca, S. A. Viana, E. J. Silva, and R. C. Mesquita. Imposing Boundary Conditions in the Meshless Local Petrov-Galerkin Method. IET Science, Measurement & Technology, 2(6):387–394, 2008. [citado na(s) páginas(s) 3, 4, 79, 80, 124]

[20] A.R. Fonseca, B.C. Correa, E.J. Silva, and R.C. Mesquita. Improving the Mixed Formulation for Meshless Local Petrov Galerkin Method. IEEE Transactions on Magnetics, 46(8):2907 –2910, 2010. [citado na(s) páginas(s) 3, 124]

[21] Martin Fowler. UML Distilled - A Brief Guide to the Standard Object Modeling Language. Addison-Wesley Professional, 2003. [citado na(s) páginas(s) 82]

[22] Thomas-Peter Fries and Hermann-Georg Matthies. Classification and Overview of Meshfree Methods. Technical report, Technical University Braunschweig, Department of Mathematics and Computer Science, 2004. [citado na(s) páginas(s) 3, 31, 36]

[23] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1st edition, 1995. [citado na(s) páginas(s) 4, 7, 8, 14]

[24] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics - Theory and application to non-spherical stars. Royal Astronomical Society, Monthly Notices, 181:375–389, 1977. [citado na(s) páginas(s) 3]

[25] Peter Gottschling, Cornelius Steinhardt, Luis Beroiza, and Andrew Lumsdaine. The Matrix Template Library 4. `http://www.mtl4.org`, Acesso em janeiro de 2011. [citado na(s) páginas(s) 14, 88]

[26] Thomas J. R. Hughes. The Finite Element Method: Linear Static and Dynamic Finite Element Analysis. Dover Publications, 2000. [citado na(s) páginas(s) 2]

[27] S. Ikuno, K. Takakura, and A. Kamitani. Influence of Method for Imposing Essential Boundary Condition on Meshless Galerkin/Petrov-Galerkin Approaches. IEEE Transactions on Magnetics, 43(4):1501–1504, 2007. [citado na(s) páginas(s) 54]

[28] Y. Krongauz and T. Belytschko. EFG approximation with discontinuous derivatives. International Journal for Numerical Methods in Engineering, 41(7):1215–1233, 1998. [citado na(s) páginas(s) 59]

[29] O. Laghrouche, P. Bettess, E. Perrey-Debain, and J. Trevelyan. Wave interpolation finite elements for Helmholtz problems with jumps in the wave speed. Computer Methods in Applied Mechanics and Engineering, 194(2-5):367 – 381, 2005. Selected papers from the 11th Conference on The Mathematics of Finite Elements and Applications. [citado na(s) páginas(s) 1]

[30] P. Lancaster and K. Salkauskas. Surfaces generated by moving least squares methods. Mathematics of Computation, 37:141–158, 1981. [citado na(s) páginas(s) 32]

[31] Q. Li, S. Shen, Z. D. Han, and S. N. Atluri. Application of Meshless Local Petrov-Galerkin (MLPG) to Problems with Singularities, and Material Discontinuities, in

3-D Elasticity. CMES - Computer Modeling in Engineering and Sciences, 4:571–585, 2003. [citado na(s) páginas(s) 70, 71]

[32] G. R. Liu. Mesh Free Methods: Moving Beyond the Finite Element Method. CRC Press, 1st edition, 2002. [citado na(s) páginas(s) 3, 29, 72, 73, 75, 128]

[33] G. R. Liu. Mesh Free Methods: Moving Beyond the Finite Element Method. CRC Press, 2nd edition, 2009. [citado na(s) páginas(s) 2, 3, 24, 25, 27, 28, 30, 31, 32, 33, 35, 42, 44, 45, 46, 47, 49, 52, 56, 57, 59, 60, 61, 62, 67, 69, 70, 165]

[34] G. R. Liu and Y. T. Gu. A point interpolation method. In Proceedings of 4th Asia-Pacific Conference on Computational Mechanics, pages 1009–1014, Singapore, December 1999. [citado na(s) páginas(s) 38, 72]

[35] G. R. Liu and Y. T. Gu. A matrix triangularization algorithm for the polynomial point interpolation method. Computer Methods in Applied Mechanics and Engineering, 192(19):2269 – 2295, 2003. [citado na(s) páginas(s) 42]

[36] W. K. Liu, J. Adee, and S. Jun. Reproducing Kernel and Wavelet Particle Methods for elastic and plastic problems. Advanced Computational Methods for Material Modeling, pages 175–190, 1993. [citado na(s) páginas(s) 3]

[37] Annita Macedo. Eletromagnetismo. Editora Guanabara, 1988. [citado na(s) páginas(s) 14, 15, 16, 17]

[38] MathWorks. The MATLAB home page. `http://www.mathworks.com/products/matlab/`, Acesso em fevereiro de 2011. [citado na(s) páginas(s) 121]

[39] Renato Cardoso Mesquita. Cálculo de Campos Eletromagnéticos Tridimensionais utilizando Ele PhD thesis, Universidade Federal de Santa Catarina, 1990. [citado na(s) páginas(s) 147]

[40] Microsoft. .NET Framework Developer Center. `http://msdn.microsoft.com/pt-br/netframework/`, Acesso em janeiro de 2011. [citado na(s) páginas(s) 8]

[41] S. Nakata, Y. Takeda, N. Fujita, and S. Ikuno. Parallel Algorithm for Meshfree Radial Point Interpolation Method on Graphics Hardware. IEEE Transactions on Magnetics, PP(99):1 –1, 2010. [citado na(s) páginas(s) 148]

[42] W.L. Nicomedes, R.C. Mesquita, and F.J. da Silva Moreira. 2-d scattering integral field equation solution through an imls meshless-based approach. Magnetics, IEEE Transactions on, 46(8):2783 –2786, 2010. [citado na(s) páginas(s) 147]

[43] W.L. Nicomedes, R.C. Mesquita, and F.J.S. Moreira. A Meshless Local Boundary Integral Equation method for three dimensional scalar problems. pages 1 –1, May 2010. [citado na(s) páginas(s) 148]

[44] Nokia. Qt - Cross-plataform application and UI framework. `http://qt.nokia.com/`, Acesso em janeiro de 2011. [citado na(s) páginas(s) 8]

[45] E. Onate, S. Idelsohn, O. C. Zienkiewicz, and R. L. Taylor. A finite point method in computational mechanics. Applications to convective transport and fluid flow. International Journal for Numerical Methods in Engineering, 39:3839–3866, 1996. [citado na(s) páginas(s) 25]

[46] G. F. Parreira, A. R. Fonseca, A. C. Lisboa, E. J. Silva, and R. C. Mesquita. Efficient Algorithms and Data Structures for Element-free Galerkin Method. IEEE Transactions on Magnetics, 42(4):659–662, 2006. [citado na(s) páginas(s) 100]

[47] G. F. Parreira, E. J. Silva, A. R. Fonseca, and R. C. Mesquita. The element-free Galerkin method in threedimensional electromagnetic problems. IEEE Transactions on Magnetics, 42(4):711–714, 2006. [citado na(s) páginas(s) 52, 95, 124, 148]

[48] Anastasis C. Polycarpou. Introduction to the Finite Element Method in Electromagnetics. Morgan & Claypool, 2006. [citado na(s) páginas(s) 124]

[49] M. J. D. Powell. The theory of radial basis function approximation in 1990. Advances in Numerical Analysis, 2:303–322, 1992. [citado na(s) páginas(s) 46]

[50] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. Numerical Recipes - The Art of Scientific Computing. Cambridge University Press, 3rd edition, 2007. [citado na(s) páginas(s) 56, 81, 86]

[51] P. Schembri, D. L. Crane, and J. N. Reddy. A three-dimensional computational procedure for reproducing meshless methods and the finite element method. International Journal for Numerical Methods In Engineering, 61:896–927, 2004. [citado na(s) páginas(s) 148]

[52] A. Taflove. Computational Electrodynamics - The Finite-Difference Time-Domain Method. Artech House, Norwood, MA, 2000. [citado na(s) páginas(s) 2]

[53] F. Torres and B. Jecko. Complete FDTD analysis of microwave heating processes in frequency-dependent and temperature-dependent media. IEEE

Transactions on Microwave Theory and Techniques, 45(1):108 –117, January 1997. [citado na(s) páginas(s) 1]

[54] UFMG, USP, and UFSC. BRFEM home page. `http://www.cpdee.ufmg.br/~renato/brfem/index.html`, Acesso em janeiro de 2011. [citado na(s) páginas(s) 9]

[55] David Vandevoorde and Nicolai M. Josuttis. C++ Templates: The Complete Guide. Addison-Wesley, 2003. [citado na(s) páginas(s) 10, 11, 13, 92]

[56] S.A. Viana, D. Rodger, and H.C. Lai. Application of the local radial point interpolation method to solve eddy-current problems. IEEE Transactions on Magnetics, 42(4):591 –594, 2006. [citado na(s) páginas(s) 1, 147]

[57] J. G. Wang and G. R. Liu. A point interpolation meshless method based on radial basis functions. International Journal for Numerical Methods in Engineering, 54:1623–1648, 2002. [citado na(s) páginas(s) 46]

[58] Yong Zhang, K.R. Shao, Youguang Guo, Jianguo Zhu, D.X. Xie, and J.D. Lavers. A Comparison of Point Interpolative Boundary Meshless Method Based on PBF and RBF for Transient Eddy-Current Analysis. IEEE Transactions on Magnetics, 43(4):1497 –1500, 2007. [citado na(s) páginas(s) 1]

Apêndices

Apêndice A

Arquivos de Entrada e Saída

O MFree Framework efetua a leitura e escrita em arquivos através das classes `Data_reader` e `Data_writer`, respectivamente.

A.1 Arquivo de Entrada - Nós

Nós são lidos a partir de um arquivo no formato da Figura A.1. Os identificadores (id) dos nós devem começar a serem numerados por 0 (zero).

```
*NODES_NUM
n
*NODES_COORD
id_1 x1_1 x2_1 ... xd_1
id_2 x1_2 x2_2 ... xd_2
...
id_n x1_n x2_n ... xd_n
*NODES_END
```

Figura A.1: Arquivo de entrada para nós

onde

n : número de nós a serem lidos
d : dimensão do espaço
id_i : id do i-ésimo nó, com $i = 1 \ldots n$
xj_i : j-ésima coordenada do i-ésimo nó, com $i = 1 \ldots n$ e $j = 1 \ldots d$

A.2 Arquivo de Entrada - Malha de Integração

A malha de integração é lida de um arquivo no formato da Figura A.2. São suportadas malhas com células triangulares e retangulares. Os vértices de uma célula devem ser informados no sentido anti-horário.

```
*CELL_GRID_BEGIN
*VERTICES_NUM
n
*VERTICES_COORD
idv_1 x1_1 x2_1 ... xd_1
idv_2 x1_2 x2_2 ... xd_2
...
idv_n x1_n x2_n ... xd_n
*CELLS_NUM
m
*CELLS_VERTICES
idc_1 v1_1 v2_1 ... vp_1
idc_2 v1_2 v2_2 ... vp_2
...
idc_m v1_m v2_m ... vp_m
*CELL_GRID_END
```

Figura A.2: Arquivo de entrada para malha de integração

onde

n : número de vértices a serem lidos
d : dimensão do espaço
idv_i : id do i-ésimo vértice, com $i = 1 \ldots n$
xj_i : j-ésima coordenada do i-ésimo vértice, com $i = 1 \ldots n$ e $j = 1 \ldots d$
m : número de células a serem lidas
p : número de vértices por célula
idc_i : id da i-ésima célula, com $i = 1 \ldots m$
vj_i : j-ésimo vértice da i-ésima célula, com $i = 1 \ldots m$ e $j = 1 \ldots p$

A.3 Arquivo de Entrada - Pontos

A solução do problema é calculada em pontos lidos de um arquivo no formato da Figura A.3.

```
*SOLUTION_POINTS_NUM
n
*SOLUTION_POINTS_COORD
x1_1 x2_1 ... xd_1
x1_2 x2_2 ... xd_2
...
x1_n x2_n ... xd_n
*SOLUTION_POINTS_END
```

Figura A.3: Arquivo de entrada para pontos

onde

n : número de pontos a serem lidos
d : dimensão do espaços
xj_i : j-ésima coordenada do i-ésimo ponto, com $i = 1 \ldots n$ e $j = 1 \ldots d$

A.4 Arquivo de Entrada - Parâmetros

Alguns parâmetros devem ser fornecidos ao *framework* para que este possa ser executado corretamente. Os parâmetros são lidos de um arquivo no formato da Figura A.4 e armazenados por meio da classe `Parameters`.

```
*PARAMETERS_BEGIN

ALPHA_S = alpha_s
ALPHA_Q = alpha_q
DX = dx
DY = dy
DS = ds
PHI_POLYNOMIAL_BASIS_ORDER = phi_order
SUPPORT_K_NEIGHBORS = support_k
INTEGRATION_K_NEIGHBORS = integration_k
INTEGRATION_BOUNDARY_ORDER = integration_boundary_order
INTEGRATION_DOMAIN_ORDER = integration_domain_order
INTEGRATION_BOUNDARY_DIVISIONS = integration_boundary_div
DIRICHLET_PENALTY_FACTOR = dirichlet_penalty_factor
INTERFACE_PENALTY_FACTOR = interface_penalty_factor
PATH_NODES = path_nodes
PATH_CELL_GRID = path_cell_grid
PATH_SOLUTION_POINTS = path_solution_points
PATH_SAVE_SOLUTION = path_save_solution

*PARAMETERS_END
```

Figura A.4: Arquivo de entrada para parâmetros

onde

alpha_s : fator de escalonamento para domínio de suporte
alpha_q : fator de escalonamento para domínio de quadratura
dx : distância local na direção x
dy : distância local na direção y
ds : distância local radial
phi_order : ordem do polinômio utilizado na base das funções de forma
support_k : número de vizinhos utilizado pelo KNN para domínio de suporte

integration_k : número de vizinhos utilizado pelo KNN para domínio de quadratura

integration_boundary_order : ordem de integração em contornos

integration_domain_order : ordem de integração em domínios

integration_boundary_div : número de divisões dos contornos para integração

dirichlet_penalty_factor : fator de penalidade para imposição das condições de contorno de Dirichlet

interface_penalty_factor : fator de penalidade para tratamento das descontinuidades de materiais nas interfaces

path_nodes : caminho para o arquivo de entrada de nós

path_cell_grid : caminho para o arquivo de entrada de malha de integração

path_solution_points : caminho para o arquivo de entrada de pontos para calcular a solução

path_save_solution : caminho para o diretório onde é gravado o arquivo contendo a solução

A.5 Arquivo de Saída - Solução

A solução u^h para os pontos informados é gravada em um arquivo no formato da Figura A.5.

```
uh_1
uh_2
...
uh_n
```

Figura A.5: Arquivo de saída para solução

onde

n : número de soluções para os n pontos fornecidos
uh_i : solução para o i-ésimo ponto fornecido, com $i = 1 \ldots n$

Apêndice B

Funções de Base Radial

Funções de base radial (RBF) são frequentemente utilizadas junto aos métodos de construção de funções de forma, por isso algumas delas são apresentadas aqui. Uma função de base radial $R(d)$ é uma função de *simetria radial*, isto é, seus valores dependem apenas da distância ao centro x_c da função. Define-se r como o *raio de suporte* da RBF e d como a distância ao centro x_c normalizada por r, a saber

$$d = \frac{\|\mathbf{x} - \mathbf{x}_c\|}{r} \tag{B.1}$$

As derivadas de $R(d)$ são obtidas através da regra da cadeia, sendo

$$R_{,k} = \frac{\partial R}{\partial x_k} = \frac{\partial R}{\partial d}\frac{\partial d}{\partial x_k} \tag{B.2}$$

com

$$\frac{\partial d}{\partial x_k} = \frac{1}{r}\frac{x_k - x_{ck}}{\|\mathbf{x} - \mathbf{x}_c\|} \tag{B.3}$$

onde $R_{,k}$ é a derivada parcial de R em relação à k-ésima dimensão, x_k é k-ésima componente de $\mathbf{x}$, e x_{ck} a k-ésima componente de $\mathbf{x}_c$.

As RBFs apresentadas nesta seção encontram-se em [33]. A RBF *spline* cúbica é dada por:

$$R(d) = \begin{cases} \frac{2}{3} - 4d^2 + 4d^3 & \text{se } d \leq \frac{1}{2} \\ \frac{4}{3} - 4d + 4d^2 - \frac{4}{3}d^3 & \text{se } \frac{1}{2} < d \leq 1 \\ 0 & \text{se } d > 1 \end{cases} \tag{B.4}$$

e sua derivada:

$$\frac{R(d)}{d} = \begin{cases} -8d + 12d^2 & \text{se } d \leq \frac{1}{2} \\ -4 + 8d - 4d^2 & \text{se } \frac{1}{2} < d \leq 1 \\ 0 & \text{se } d > 1 \end{cases} \tag{B.5}$$

A RBF *spline* quártica é dada por:

$$R(d) = \begin{cases} 1 - 6d^2 + 8d^3 - 3d^4 & \text{se } d \leq 1 \\ 0 & \text{se } d > 1 \end{cases} \tag{B.6}$$

e sua derivada:

$$\frac{R(d)}{d} = \begin{cases} -12d + 24d^2 - 12d^3 & \text{se } d \leq 1 \\ 0 & \text{se } d > 1 \end{cases} \tag{B.7}$$

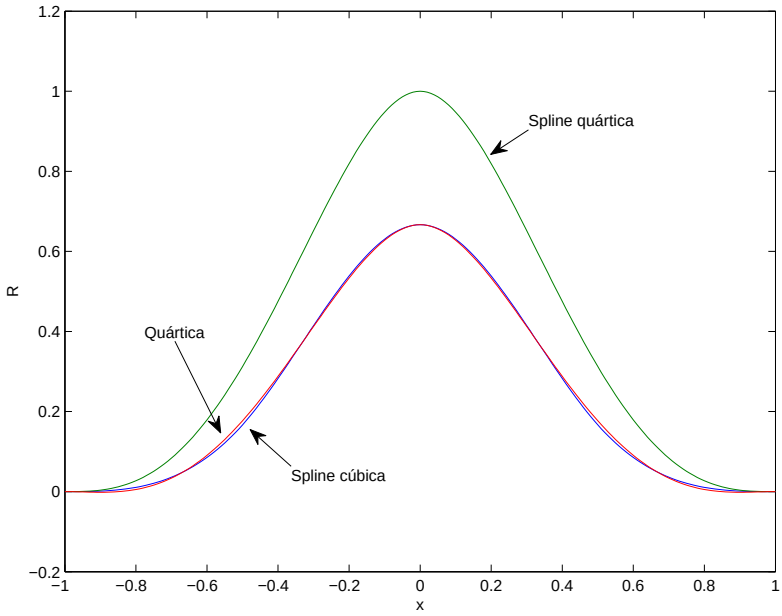
A RBF quártica é dada por:

$$R(d) = \begin{cases} \frac{2}{3} - \frac{9}{2}d^2 + \frac{19}{3}d^3 - \frac{5}{2}d^4 & \text{se } d \leq 1 \\ 0 & \text{se } d > 1 \end{cases} \tag{B.8}$$

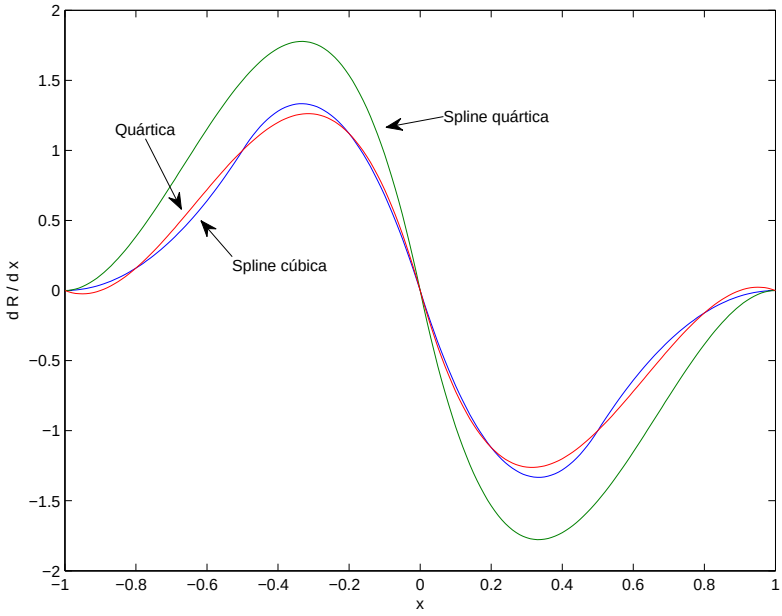
e sua derivada:

$$\frac{R(d)}{d} = \begin{cases} -9d + 19d^2 - 10d^3 & \text{se } d \leq 1 \\ 0 & \text{se } d > 1 \end{cases} \tag{B.9}$$

A Figura B.1 mostra os gráficos para as RBFs das Equações B.4, B.6 e B.8 e suas derivadas em uma dimensão. Como as funções são semelhantes, apenas o gráfico da *spline* cúbica é apresentado em duas dimensões (Figura B.2).

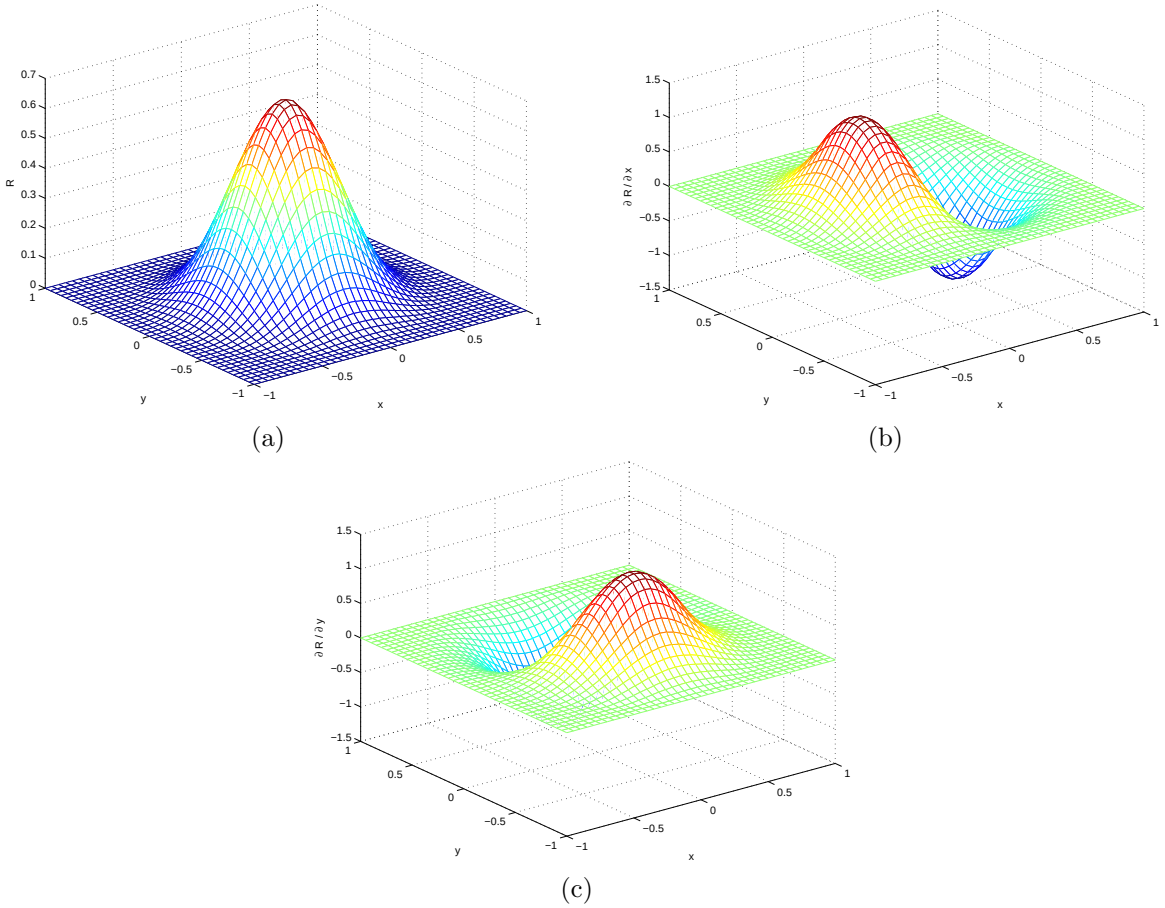


(a)



(b)

Figura B.1: Funções de base radial em 1D. Estão presentes *spline* cúbica, *spline* quártica e quártica, com $x_c = 0$. (a) Mostra $R(d)$ e (b) a derivada $R_{,x}(d)$.



(a) (b) (c)

Figura B.2: RBF *spline* cúbica em 2D, com $\mathbf{x}_c = (0, 0)$. (a) Mostra $R(d)$, (b) a derivada $R_{,x}(d)$ e (c) a derivada $R_{,y}(d)$.