Ambiente de programação científica orientada a objetos NeoPZ: Uma introdução para novos alunos

Francisco T. Orlandini
LabMeC
UNICAMP

Supervisionado por Nathan Shauer e Philippe Devloo
francisco.orlandini@gmail.com

Outubro 2014

Resumo

Neste tutorial buscamos apresentar de forma simples os conceitos fundamentais para um estudante ingressante no LabMeC, assim como algumas das ferramentas que virão a ser utilizadas no futuro. No decorrer de três meses é esperado que o aluno ambiente-se com a linguagem de programação C++, com o software Mathematica e com as ideias centrais em torno das quais se baseia o Método dos Elementos Finitos, além de uma visualização e experimentação inicial com o pacote NeoPZ.

Conteúdo

1 Introdução

1.1 Motivação

Devido à grade dos cursos de graduação em Engenharia, é comum que o aluno ingressante no LabMeC não tenha conhecimento prévio na área em que virá a trabalhar. Este tutorial busca introduzir o aluno à ideia da implementação computacional do Método dos Elementos Finitos (FEM) distinguindo e esclarecendo os conceitos necessários para tal.

1.2 NeoPZ

Atualmente, existem diversos de pacotes que implementam o FEM disponíveis no mercado, dentre eles podem ser citados *Phoenix*, *Ansys*, *Sap2000*, *FlexPDE*, *Consol* e *Hermes*, dentre outros. O uso do pacote NeoPZ faz-se interessante pois proporciona uma maior liberdade ao usuário, por exemplo, quanto à liberdade quanto à representação de geometrias arbitrárias - a aproximação de um círculo por um conjunto de segmentos de retas pode, por exemplo, influenciar bastante na simulação de problemas eletromagnéticos. Também permite a resolução de qualquer equação diferencial descrevendo um fenômeno físico, podendo então ser aplicado nas mais diversas áreas do conhecimento. Os diferenciais mais importantes do ambiente NeoPZ em relação aos outros pacotes são:

- Diversos espaços de aproximação H^1, L^2, H^{div}, H^{curl}, podendo também o usuário definir o seu próprio espaço de aproximação.
- Diversos métodos de resolução de Sistemas Lineares - Decomposição LU, LDL^t, Cholesky. Métodos iterativos como Gradiente conjugado, *GMRES*, com ou sem pré-condicionador, dentre outros.
- Compila em qualquer plataforma - Windows, Linux, OSX
- HP adaptatividade em 1, 2 e 3D - Permite refinamento da malha somente onde este se faz necessário para obter uma melhor aproximação.

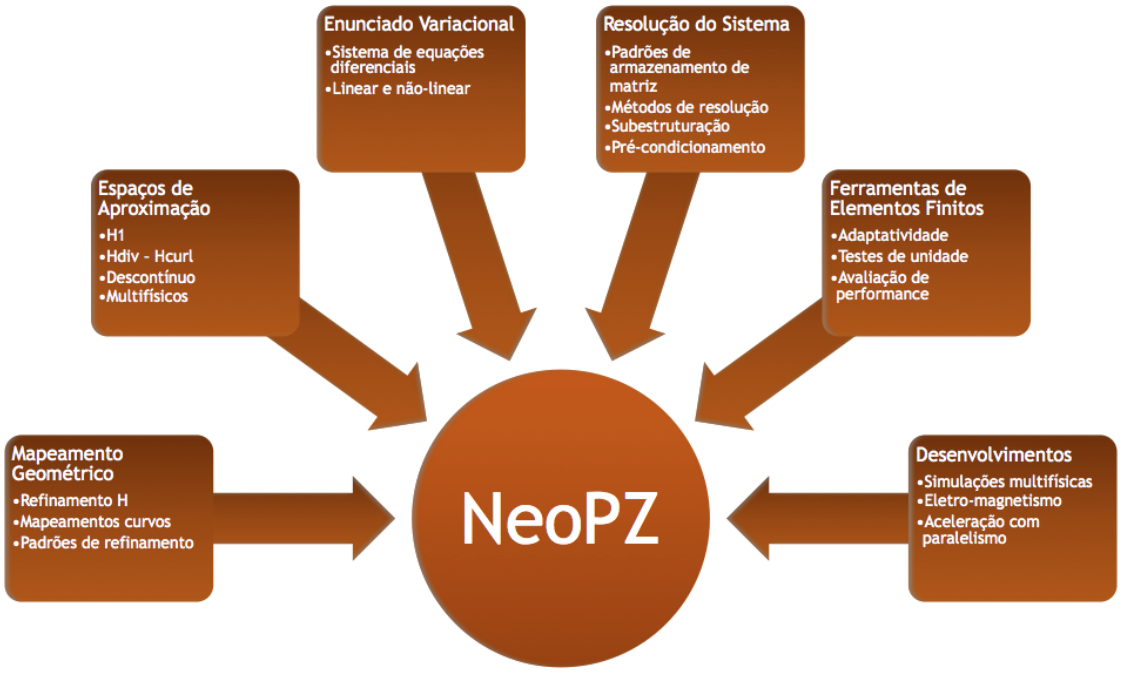

Figura 1: Visão geral da arquitetura NeoPZ

1.3 Organização do tutorial e cronograma proposto

As atividades propostas são divididas em quatro partes, e é esperado que o estudante leve três semanas para cada uma delas. Em caso de dificuldades, alguém do LabMeC deve ser avisado. A estrutura das partes é:

1. Introdução a C++
2. Introdução ao FEM
3. Introdução ao *Mathematica*
4. Introdução ao pacote NeoPZ

Na primeira parte é discutida a questão da programação orientada a objeto e a justificativa do emprego desta filosofia no pacote NeoPZ. Em seguida, são propostos dois pequenos exercícios para familiarizar o estudante com conceitos essenciais de C++.

Já na segunda parte, através da leitura de capítulos do livro Finite Elements: An Introduction Volume I (Oden et al., 1981), é esperado que o estudante adquira certa familiaridade com o FEM através da análise de problemas unidimensionais. Na terceira parte

o estudante deve realizar dois exercícios utilizando o software Mathematica e aplicar os conceitos vistos na leitura dos capítulos do livro. Finalmente, na quarta parte o aluno tem uma primeira experiência com pacote NeoPZ, observando a simulação da mesma equação diferencial analisada nos capítulos lidos de Oden et al. (1981) e podendo comparar os resultados com os obtidos no Mathematica.

2 Programação orientada a objeto e introdução a C++

2.1 Programação orientada a objeto

Conforme explicado em Deitel and Deitel (2012): *"Objects, or more precisely[...], the classes objects come from, are essentially reusable software components. There are date objects, time objects, audio objects, video objects, automobile objects, people objects, etc. Almost any noun can be reasonably represented as a software object in terms of attributes (e.g., name, color and size) and behaviours (e.g., calculating, moving and communicating). Software developers are discovering that using a modular, object−oriented design and implementation approach can make software−development groups much more productive than was possible with earlier popular techniques like "structured programming"−object−oriented programs are often easier to understand, correct and modify."*. Para uma maior compreensão dos paradigmas envolvidos na filosofia de programação orientada a objeto, é sugerida a leitura de Eckel (2000). O conceito de herança entre classes também é extensivamente utilizado no pacote NeoPZ.

2.2 A linguagem de programação C++

2.3 Exercício

2.3.1 Ambientação

- Criar um projeto novo em XCode
- Rodar o programa principal que ele mesmo criou
- Acrescentar uma função que realize a operação $f(x) = Sin(3\pi x)$
- Imprimir o resultado para $0 \leq x \leq \pi$
- Executar o programa passo a passo utilizando o *debugger*
- Documentar um código utilizando Doxygen

2.3.2 Estudo prático da linguagem C++

Classe Vetor

Primeiramente, o aluno deve criar uma classe que implementa um vetor de números reais. No *header* da classe, os objetivos da classe, assim como os de quaisquer métodos que venham a ser criados, devem estar bem explicados. A classe criada deve conter, pelo menos:

- Construtor e destrutor
- Método que calcula a norma do vetor
- Métodos do tipo GetVal, SetVal
- Construtor de cópia
- Seleção de elemento através de operador [ ]
- Operador =
- Operador + e operador -
- Operador ==

Obs: O livro Deitel and Deitel (2012) é fortemente recomendado para eventuais dúvidas, tanto relativas à sintaxe da linguagem quanto à ideia da classe em si

Classe Matriz

Em seguida, o aluno deve criar uma classe que implementa uma matriz. Obviamente, alguns dos métodos criados no exercício anterior podem vir a ser aproveitados (diretamente ou não). Os atributos e métodos necessários para a classe devem ser pensados pelo aluno, sendo livre a forma da implementação e as estruturas de dado envolvidas. Entretanto, assim como na classe de vetor, alguns objetivos devem ser cumpridos:

- A matriz deve ter elementos do tipo $REAL$ (Ver $typedef$ em Deitel and Deitel (2012))
- Construtor vazio;
- Construtor que cria uma matriz com m linhas e n colunas
- Construtor de cópia
- Destrutor
- Método para redimensionamento da matriz ($Resize$)
- Método que transpõe a matriz ($Transpose$)
- Métodos do tipo $GetVal$ e $PutVal$
- Operador () para seleção de elemento da matriz
- Operador = para atribuição
- Operador == para comparação
- Operações de adição, subtração e multiplicação de matrizes (assim como seus respectivos operadores)
- Operação do produto entre matriz e vetor
- Uma função do tipo $debugStop()$

O aluno é incentivado a criar quaisquer métodos que venham a ser necessários para cumprir o objetivo. A classe deve ser testada com o auxílio de uma rotina que teste operações básicas possíveis de ser realizadas com ela. Ao final do exercício, o aluno deve rtesponder as seguintes questões:

1. Qual o significado de *const* após a assinatura de um método?
2. Qual o significado de passar uma variável *const* por referência?
3. Quando são chamados os construtores e destrutores (*Dica: utilizando a função std::cout, faça que os construtores e o destrutor de sua classe imprimam uma mensagem quando forem chamados e o endereço do objeto responsável pela sua chamada*)?
4. Existe utilidade dos métodos $GetVal$ e $PutVal$ se há o operador parênteses para seleção de elemento?
5. Por que a utilização do $typedef\ double\ REAL$;?

3 O Método dos Elementos Finitos (FEM): Uma introdução

Introdução

Nesta atividade, o aluno deve estudar o primeiro capítulo de Oden et al. (1981) e realizar os exercícios propostos no decorrer do capítulo. Caso haja interesse em se aprofundar no assunto, a subseção 3.2 sugere a leitura do segundo capítulo do livro e realça os pontos importantes.

3.1 Oden - Capítulo I

No primeiro capítulo de Oden et al. (1981), o FEM é apresentado através da análise de um problema chamado problema modelo. O problema modelo se trata de uma equação diferencial com solução analítica conhecida, para fins de análise e validação. Claramente, métodos numéricos não justificam o esforço computacional em casos assim, entretanto o exemplo é posto para fins didáticos. A Formulação fraca, ou forma variacional, é apresentada como uma forma de relaxamento das condições necessárias para uma solução satisfatória. Ao invés de nos contentarmos úica e somente com uma função $u(x)$. que satisfaça a equação diferencial em todos os pontos do domínio, buscamos agora uma função que satisfaça a nossa formulação fraca, ou seja, que satisfaça a nossa equação na forma integral. A nossa condição fraca também, naturalmente, diminui nossos critérios de suavidade quanto à solução desejada (na formulação variacional não aparece a segunda derivada da função $u(x)$). É recomendada uma leitura atenta quanto aos espaços das funções $u(x)$ e $v(x)$, dado que a discussão de qual é o espaço adequado para cada problema é de fundamental importância.

Neste capítulo também é abordada a técnica de aproximação de Galerkin, e como ela nos permite restringir o espaço de funções no qual buscamos nossa aproximação através da criação de um subespaço N-dimensional do espaço de funções analisado, caracterizado por um conjunto infinito das chamadas funções de base. Finalmente, o FEM é apresentado como a aplicação da técnica de aproximação de Galerkin sobre a forma fraca do nosso problema aliado a um método sistemático para a escolha das funções de base, sendo o domínio dividido em regiões - chamadas de elementos - e as funções de base definidas individualmente sobre cada elemento. Uma aproximação para a solução analítica é então apresentada, com o uso de funções chapéu e a divisão do domínio em quatro elementos. Os exercícios que acompanham o texto são altamente recomendados. Na seção 4 deste tutorial o aluno deve realizar um programa no software Mathematica que permite a visualização dos procedimentos realizados neste capítulo.

3.2 Opcional: Oden - Capítulo II

Caso, neste momento, haja um interesse grande em minúcias das aplicações do FEM em problemas unidimensionais é recomendada a leitura do segundo capítulo do livro. Neste capítulo temos uma abordagem mais geral de problemas unidimensionais, dando-nos a capacidade para utilização do método em qualquer equação diferencial linear de segunda ordem, assim como maior detalhamento na implementação de um algoritmo utilizando o FEM. Após a apresentação da chamada forma clássica de uma equação diferencial de segunda ordem, o texto segue para uma análise cuidadosa dos tipos de condições de contorno, e como estas condições aparecerão na formulação fraca é essencial para a elaboração de um bom algoritmo - Além, naturalmente, de uma maior compreensão do método. Também é abordada neste capítulo a questão da escolha das funções de base, i.e., quais as restrições para que um determinado conjunto de funções seja um conjunto adequado para a representação de um espaço. Com o uso de funções Lagrangianas, neste capítulo é desenvolvido um raciocínio que permite o uso de funções de base de maior ordem, ao invés de somente as funções chapéu.

Quanto à parte computacional, é apresentado em maiores detalhes o conceito de assemblagem, que permite que os cálculos sejam feitos individualmente em cada elemento e depois posicionados adequadamente em uma matriz global. Este conceito permite, por exemplo, a paralelização de tarefas em um determinado estágio de um algoritmo que apresenta uma implementação do FEM. Assim como no Capítulo 1, os exercícios deste capítulo também são recomendados. Na subseção 4.4 são sugeridas modificações no programa desenvolvido no Mathematica para fazer uso das técnicas apresentadas no Capítulo 2.

4 Introdução ao software Mathematica: Criando um primeiro código utilizando o FEM

Implementação do Problema Modelo

Conforme abordado na subseção 3.1, neste tutorial do Mathematica a proposta é de criar um algoritmo para a resolução numérica do problema modelo. Para isto, é necessário ter em mente tanto a equação diferencial que origina a forma fraca como as condições de contorno do problema em questão. Assim como no Capítulo 1 de Oden et al. (1981), as funções de base escolhida serão lineares, as funções chapéu. Para melhor visualização da técnica, o algoritmo será feito de modo que o número de elementos em que o domínio será dividido seja variável, assim pode ser observada esta alteração no resultado final, quando comparado com a solução analítica.

4.1 Implementação das funções chapéu

As funções chapéu são elaboradas conforme o procedimento seguido no Capítulo 1 de Oden et al. (1981), entretanto, desta vez criaremos um método geral para a divisão do domínio em qualquer número de elementos - todos com o mesmo comprimento. Assim sendo, como nosso domínio é a reta $0 \leq x \leq 1$, se tivermos nel elementos, cada um terá comprimento $h = \frac{1}{nel}$. Tendo em vista um nó interno x_i ao nosso domínio - i.e., excluindo os nós de fronteira, podemos dizer que a função de base correspondente será definida por $\frac{x-(x_i-h)}{h}, x_i - h \leq x \leq x_i$ e $\frac{x_i+h-x}{h}, x_i \leq x \leq x_i + h$, tendo valor nulo para qualquer outro valor de x. Na Figura 2, vemos as funções de formas geradas ao dividirmos o domínio em 4 elementos (pode-se verificar que o resultado é idêntico ao apresentado em Oden et al. (1981)). O Mathematica apresenta também a função *D[f,x]*, que a realiza a derivada de f em relação a x, e esta pode ser útil mais adiante no cálculo dos elementos da matrix K e do vetor F.

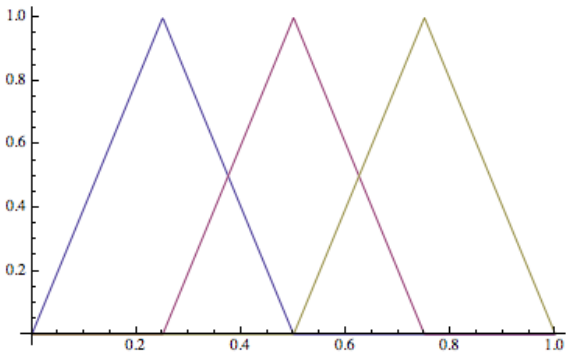

Figura 2: Funções de forma geradas pelo *software* Mathematica

4.2 Implementação das Estruturas de Dados

Nesta versão do algoritmo, tanto a Matriz de Rigidez quanto o Vetor de Carga serão computados de uma só vez, ao invés da soma das matrizes (ou vetores, no caso do vetor de carga) locais de cada elemento, apenas para fins de simplificação do algoritmo. Assim sendo, a função Table do Mathematica pode ser utilizada para criá-los. A função MatrixForm permite a visualização da matriz ou vetor passada como parâmetro na forma matricial, como pode ser visto na Figura 3.

```
K:  ( 8.16667  -3.95833     0.
     -3.95833   8.16667  -3.95833
      0.       -3.95833   8.16667 )

F:  ( 0.0625 )
    ( 0.125  )
    ( 0.1875 )
```

Figura 3: Matriz de Rigidez (K) e Vetor de Carga (F) gerados pelo algoritmo e visualizadas através do comando *MatrixForm*.

4.3 Resolução do Problema

Após obter a matriz K e o vetor F, o comando *LinearSolve* pode ser utilizado para resolver o sistema e encontrar os coeficientes (no caso de 4 elementos, confira-os com os coeficientes obtidos em Oden et al. (1981)). Lembrando que a aproximação é calculada a partir da somatória das multplicações de $\alpha_j \phi_j$, o comando *Plot* pode ser utilizado para visualizar, por exemplo, a comparação da aproximação com a solução analítica do problema, como pode ser visto na figura 4.

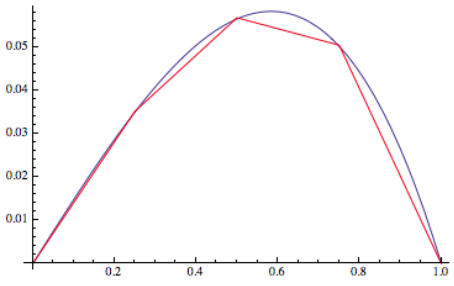

Figura 4: Comparação da aproximação obtida com a solução analítica.

4.4 Opcional: Implementação de Funções Lagrangianas

Nesta seção o aluno deve implementar um algoritmo mais generalizado da resolução do problema modelo através do FEM. Baseado fortemente no algoritmo anterior, desta vez

alterações devem ser feitas de modo que a computação da matriz de rigidez e do vetor de carga seja feita separadamente para cada elemento - e depois seja implementado o processo de *assemblagem*, no qual cada matriz local é alocada na matriz global. Além disso, também é interessante a opção de que as funções de base a serem utilizadas desta vez sejam funções lagrangianas de ordem a ser especificada pelo usuário. Por mais que rotineiramente o uso de funções lagrangianas seja bastante diminuto comparado ao uso de funções hierárquicas, essas permitem ao aluno uma visão bastante clara da influência da ordem dos polinômios das funções de base.

4.4.1 Elemento mestre

Conforme explicado no capítulo 2 do livro de Oden et al. (1981), todas as computações serão realizadas no chamado Elemento Mestre. Com o uso da coordenada generalizada ξ, cada elemento será mapeado de modo que, se o elemento i é delimitado pelos nós x_i e x_{i+1}, estes pontos serão mapeados, respectivamente, nos pontos $\xi = -1$ e $\xi = +1$. As funções de forma serão definidas no domínio de ξ, e através da matriz de dependência, serão relacionadas às funções de base globais correspondentes (este processo será descrito em detalhes na subseção 4.4.3). Como deve estar claro neste ponto, as integrais também serão realizadas no domínio $\xi = [-1, 1]$, e portanto obteremos, por exemplo, para a matriz de rigidez local:

$$\begin{aligned} Kel_{j,k} &= \int_{x[i]}^{x[i+1]} [\phi_j(x)\phi_k(x) + \frac{d\phi_j(x)}{dx}\frac{d\phi_k(x)}{dx}]dx \\ &= \int_{-1}^{1} [\psi_j(\xi)\psi_k(\xi) + \frac{d\psi_j(\xi)}{d\xi}\frac{d\xi}{dx}\frac{d\psi_k(\xi)}{d\xi}\frac{d\xi}{dx}]\frac{d\xi}{dx}^{-1} d\xi \end{aligned}$$

4.4.2 Funções de Forma Lagrangianas

As funções de forma utilizadas no algoritmo devem ser lagrangianas. Conforme definidas no Capítulo 2 de Oden et al. (1981), se deseja-se utilizar funções de ordem p e da subdivisão do elemento obtém-se $p+1$ pontos equidistantes $x_i, i = 1, 2, ..., p+1$(que são mapeados no domínio ξ) as funções de forma são, portanto:

$$\psi_i(\xi) = \prod_{j=1, j\neq i}^{p+1} \frac{\xi - \xi_j}{\xi_i - \xi_j}$$

Com a propriedade de:

$$\psi_i(\xi_j) = \begin{cases} 1, & j = i \\ 0, & j \neq i \end{cases}$$

4.4.3 Assemblagem

Assemblagem é o processo de posicionar corretamente as matrizes locais geradas na computação individual de cada elemento na matriz global. Isso equivale a associar cada função de forma (definidas no elemento mestre, portanto, em ξ) à função de base equivalente(definida no domínio do problema, portanto, em x). Assim, se temos *nel* elementos e $p+1$ funções de forma, nossa matriz de dependência terá *nel* linhas e $p+1$ colunas, sendo o elemento $[dependencyMatrix]_{i,j}$ a função de base associada à j-ésima função de forma do

elemento i. No caso de funções lagrangianas, é bom lembrar que a primeira função de forma do elemento $i+1$ corresponde à mesma função de base do que a última função de forma do elemento i, portanto, elas devem ter o mesmo coeficiente α_j associado a elas, *i.e.*, o mesmo grau de liberdade.

Matriz de dependencia:

$$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 5 & 6 & 7 \\ 7 & 8 & 9 \\ 9 & 10 & 11 \\ 11 & 12 & 13 \\ 13 & 14 & 15 \\ 15 & 16 & 17 \\ 17 & 18 & 19 \\ 19 & 20 & 21 \end{pmatrix}$$

Figura 5: Matriz de dependência gerada no caso da divisão do domínio em dez elementos e funções de forma de segundo grau.

4.4.4 Condições de Contorno

No caso de condições de contorno de Dirichlet (Ver seção 2.7.4 de Oden et al. (1981)), queremos forçar, por exemplo, $\alpha_1 = v_1$. Temos $\alpha_1 K_{1,1} + \alpha_2 K_{1,2} + ... + \alpha_N K_{1,N} = F_1$ (onde $N = nel * p + 1$), portanto, fazendo $F_1 = v_1 * BIG$ e $K_{1,1} = BIG$, onde $BIG \gg 1$ forçamos $\alpha_1 = \frac{F_1 - (\alpha_2 K_{1,2} + ... + \alpha_N K_{1,N})}{K_{1,1}} = v_1$. O mesmo raciocínio pode ser ser utilizado para impor um valor para α_N.

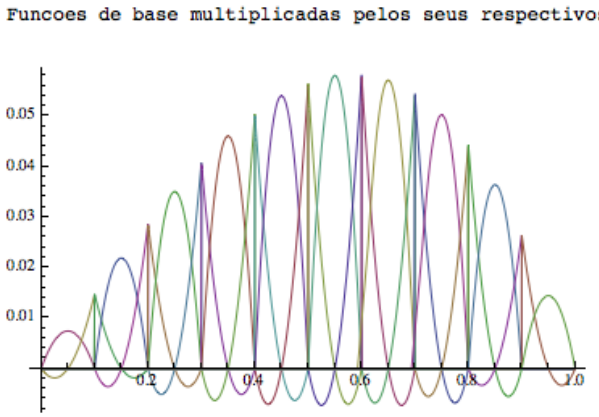

Figura 6: Funções de base multiplicadas pelos seus respectivos coeficientes, gerados no caso da divisão do domínio em dez elementos e funções de forma de segundo grau. A aproximação obtida corresponde à soma destas funções.

5 O pacote NeoPZ e a implementação do Problema Modelo

Nesta seção o aluno deve instalar o Pacote NeoPZ, rodar uma simulação já implementada e analisar seu algoritmo, visando a ambientação com o pacote.

5.1 Dependências

Para a instalação do pacote NeoPZ, siga o tutorial encontrado no site do LabMeC em `http://www.labmec.org.br/wiki/howto/start`.

5.2 Exemplo Simples

Após a instalação do pacote NeoPZ o aluno deve executar o programa *ExemploSimples.cpp*, no qual é implementado o Problema Modelo do primeiro capítulo de Oden et al. (1981) com número variável de elementos. O aluno deve então ler o código, entendendo cada linha deste, para se ambientar com o ambiente NeoPZ - inclusive o material criado para este exemplo, *TPZMatModelProblem*, que implementa a formulação fraca do problema modelo. Entretanto, um aviso: Não é esperado que o aluno compreenda todas as estruturas de dados utilizadas no NeoPZ para a resolução de um problema, somente a estrutura de dados de forma geral, apresentada nos arquivos *ExemploSimples.cpp*, *TPZMatModelProblem.h* e *TPZMatModelProblem.cpp*. Após a execução do programa, será gerado um arquivo *.vtk* que poderá ser visualizado utilizando o software *Paraview*, e também será gerada no próprio console de execução a saída na formatação compatível com o Mathematica, para fins de comparação. É esperado que o aluno verifique se foram obtidos os mesmos resultados obtidos na seção 4.

Referências

Paul Deitel and Harvey Deitel. *C++ How to Program*. Prentice-Hall, 8 edition, 2012.

Bruce Eckel. *Thinking in C++: Introduction to Standard C++*, volume 1. Prentice Hall, 2, 2000.

LabMeC. Laboratório de Mecânica Computacional. URL `http://www.labmec.org.br`.

J. T. Oden, E. B. Becker, and G. F. Carey. *Finite Elements: An Introduction Volume I*. Prentice-Hall, 1981.