UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

CENTRO DE TECNOLOGIA

ESCOLA POLITÉCNICA

Problema de Valor de Contorno (PVC) Usando o Método dos Elementos Finitos

COC473 - Álgebra Linear Computacional
2024.1
Rio de Janeiro, 2024

Beatriz Farias do Nascimento DRE 122053127
Carlos Henrique Ferreira Brito Filho DRE 120081409
Mikaela Rikberg Alves DRE 119106107
Nycolas Silva Felix DRE 121054887
Thiago Moutinho de Carvalho Maksoud DRE 119048139

Visão geral

Este relatório detalha o funcionamento e a implementação de um código para resolver um Problema de Valor de Contorno (PVC) utilizando o Método dos Elementos Finitos (FEM) em um domínio bidimensional. O problema é resolvido no quadrado unitário com f(x,y)= 1f(x,y) e u(x,y)=0 no contorno.

Objetivos

- Gerar uma malha estruturada de pontos e triângulos.
- Calcular a matriz de rigidez local e global.
- Aplicar as condições de contorno.
- Resolver o sistema linear resultante para obter a solução aproximada.

Funções e Suas Descrições

1. `gerar_pontos(nx, ny)`

Esta função gera os pontos da malha em um grid estruturado.

- **Parâmetros**:
  - `nx`: Número de pontos na direção x.
  - `ny`: Número de pontos na direção y.
- **Saída**:
  - `points`: Array com as coordenadas dos pontos
- **Método**:
  - Usamos a função linspace do numpy para criar nx valores igualmente espaçados entre 0 e 1 para o eixo x e ny valores igualmente espaçados entre 0 e 1 para o eixo y. Depois disso, utilizamos a função meshgrid (também do numpy) para formar uma grade de pontos em um espaço 2D com as coordenadas replicadas de x e y. Finalmente, as matrizes são transformadas em vetores novamente usando a função ravel do numpy e combinadas numa única matriz de duas colunas usando a função column_stack do numpy.

.

2. `gerar_triangulos(points, nx, ny)`

Esta função gera a triangulação da malha.

- **Parâmetros**:
  - `points`: Array de pontos.
  - `nx`: Número de pontos na direção x.
  - `ny`: Número de pontos na direção y.
- **Saída**:
  - `triangles`: Array de triângulos, cada triângulo representado pelos índices dos pontos.

- **Método**:
    - Para cada posição na grade, os índices dos pontos que formam os vértices dos triângulos são calculados e anexados num array que é o retorno da função.

3. `gerar_malha_estruturada(nx, ny)`

Combina `gerar_pontos` e `gerar_triangulos` para criar a malha.

- **Parâmetros**:
    - `nx`: Número de pontos na direção x.
    - `ny`: Número de pontos na direção y.
- **Saída**:
    - `points`: Array com as coordenadas dos pontos.
    - `triangles`: Array de triângulos.

4. `plottar_malha_estruturada(points, triangles)`

Plota a malha estruturada gerada.

- **Parâmetros**:
    - `points`: Array de pontos.
    - `triangles`: Array de triângulos.
- **Método**:
    - Transformamos os triângulos em objetos Triangulation do matplotlib para facilitar a visualização. Depois disso, a numeração dos pontos e dos vértices são adicionados na figura.

5. `gerar_matriz_adjacencia(points, triangles)`

Gera a matriz de adjacência para os pontos na malha.

- **Parâmetros**:
    - `points`: Array de pontos.
    - `triangles`: Array de triângulos.
- **Saída**:
    - `adjacency_matrix`: Matriz de adjacência.
- **Método**:
    - Criamos laços e percorremos os triângulos e seus pontos anexando à matriz de adjacência os pontos onde as arestas estão conectadas. Por fim, adicionamos à essa matriz a auto-conexão.

6. `plottar_matriz_adjacencia(matriz)`

Plota a matriz de adjacência.

- **Parâmetros**:
    - `matriz`: Matriz de adjacência.

7. `gerar_matriz_de_rigidez_local(points, triangle, k)`

Calcula a matriz de rigidez local para um triângulo.

- **Parâmetros**:
  - `points`: Array de pontos.
  - `triangle`: Array com os índices dos pontos do triângulo.
  - `k:` Coeficiente de condutividade térmica
- **Saída**:
  - `K_local`: Matriz de rigidez local.
  - `F_local`: Vetor de força local.
- **Método**:
  - A matriz de rigidez local K_e é calculada usando os vetores b e c (calculados a partir das coordenadas dos pontos do triângulo), a constante de rigidez k e a área do triângulo (calculada na função `area_triangulo`, definida a seguir). O vetor de força local F_e é calculado como um vetor constante multiplicado pela área do triângulo. Finalmente, K_e e F_e são adicionados em K_local e F_local, que são o retorno da função.

9. `area_triangulo(p1, p2, p3)`

Calcula a área de um triângulo a partir dos seus vértices.

- **Parâmetros**:
  - `p1, p2, p3`: Vértices do triângulo
- **Saída**:
  - `area`: Área do triângulo
- **Método**:
  - Criamos uma matriz 3x3 onde cada linha contém os valores `[1, x, y]` correspondentes a cada vértice do triângulo (`p1`, `p2`, `p3`). O determinante dessa matriz (calculada na função `determinant_3x3`, definida a seguir) é, em seguida, dividido por 2 e o retorno da função é o valor absoluto do resultado.

10. `determinant_3x3(matrix)`

Calcula a área de um triângulo a partir dos seus vértices.

- **Parâmetros**:
  - `matrix`: Matrix 3x3
- **Saída**:
  - `determinante`: Determinante da matriz
- **Método**:
  - Aplicamos a fórmula clássica do determinante nos pontos de uma matriz 3x3.

11. `gerar_matriz_de_rigidez_global(points, triangles, K_local, F_local, loads)`

Monta a matriz de rigidez global agregando todas as matrizes de rigidez locais.

- **Parâmetros**:
  - `points`: Array de pontos.
  - `triangles`: Array de triângulos.
  - `K_local: Matriz de rigidez local.`
  - `F_local: Vetor de força local.`
  - `loads: Vetor com cargas (opcional).`
- **Saída**:
  - `K`: Matriz de rigidez global.
  - `F`: Vetor de força global.
- **Método**:
  - Criamos a variável K, que é uma matriz esparsa em formato lista de listas usando a função lil_matrix do numpy, e a variável F, inicializada como um vetor de zeros. Um laço então percorre cada combinação de vértices do triângulo para somar os valores das matrizes de rigidez locais à matriz de rigidez global enquanto o vetor de força local é somado ao vetor de força global. No caso de loads não ser nulo, ao invés de calcularmos o valor da carga do nó, o valor enviado é utilizado.

12. `resolver_problema(K, F, points)`

Resolve o sistema de equações de um problema de elementos finitos usando a eliminação gaussiana.

Parâmetros:

- `K`: Matriz de rigidez global.
- `F`: Vetor de força global.
- `points`: Array de pontos.

Saída:

- `U`: Vetor de deslocamentos ou temperaturas nos pontos da malha.

Método:

- Primeiro, a função aplica as condições de contorno ao sistema. Isso é feito pela função auxiliar `aplicar_condicoes_de_contorno`, que modifica a matriz de rigidez global `K` e o vetor de força global `F` para incluir as condições de contorno impostas pelo problema.
- Depois de ajustar o sistema para as condições de contorno, a função resolve o sistema de equações lineares `K * U = F` utilizando eliminação gaussiana. A função `eliminacao_gaussiana` realiza este passo, retornando o vetor `U` que contém os deslocamentos ou temperaturas nos pontos da malha.
- Finalmente, a função retorna `U`, que representa a solução do problema de elementos finitos.

13. `aplicar_condicoes_de_contorno(K, F, points, boundary_conditions)`

Aplicar as condições de contorno na matriz de rigidez global e no vetor de forças.

- **Parâmetros**:
  - `K`: Matriz de rigidez global.
  - `F`: Vetor de forças.
  - `points`: Array de pontos.
  - `boundary_conditions (opcional)`: Array de pontos com o valor de contorno.
- **Saída**:
  - `K`: Matriz de rigidez global modificada.
  - `F`: Vetor de forças modificado.
- **Método**:
  - Altera o valor de pontos na borda do domínio para 0 por padrão no caso de `boundary_conditions` ser nulo. Caso contrário, utiliza o valor dos nós enviados.

14. `eliminacao_gaussiana(A, b)`

Implementa o método da eliminação gaussiana para resolver um sistema de equações lineares da forma $A \cdot x = b$

- **Parâmetros**:
  - `A`: Matriz de coeficientes do sistema de equações.
  - `b`: Vetor de termos independentes.
- **Saída**:
  - `x`: Vetor solução do sistema de equações
- **Método**:
  - Chamamos a função eliminacao_para_frente, que realiza a primeira fase do método do escalonamento. Finalmente retornamos o resultado da funcão substituicao_reversa.

15. `eliminacao_para_frente(A, b)`

Implementa a primeira fase de eliminação da eliminação gaussiana, transformando a matriz A em uma matriz triangular superior e ajustando o vetor b de acordo com as operações realizadas em A.

- **Parâmetros**:
  - `A`: Matriz de coeficientes
  - `b`: Vetor de termos independentes.
- **Saída**:
  - `A`: Matriz de coeficientes modificado
  - `b`: Vetor de termos independentes modificado
- **Método**:

- Chamamos a função pivotamento que irá realizar o pivotamento parcial da coluna i. Para cada linha abaixo dessa linha é calculado o fator de multiplicação com o intuito de transformar A em uma matriz triangular superior, preparando-a para a fase de substituição reversa.

16. `pivotamento(A, b, i)`

Realiza o pivotamento parcial em uma matriz `A` e no vetor `b`, garantindo que o elemento pivô seja o maior em valor absoluto na coluna atual.

- **Parâmetros**:
  - `A`: Matriz de coeficientes
  - `b`: Vetor de termos independentes.
  - `i`: Índice da coluna atual onde o pivoteamento é realizado.
- **Saída**:
  - `A`: Matriz de coeficientes modificado
  - `b`: Vetor de termos independentes modificado
- **Método**:
  - Iteramos sobre os valores da linha procurando o valor absoluto do elemento na coluna `i` é máximo.

17. `substituicao_reversa(A, b)`

Resolve o sistema triangular superior resultante da eliminação gaussiana, encontrando as soluções para as variáveis desconhecidas.

- **Parâmetros**:
  - `A`: Matriz triangular superior
  - `b`: Vetor de termos independentes.
- **Saída**:
  - `x`: resultado
- **Método**:
  - Iteramos sobre todas as linhas i, aplicando a fórmula da substituição reversa. Então, usamos a função `dot` do numpy para calcular o produto interno dos elementos. Finalmente calculamos o resultado ao subtrair em b[i] e dividir pelo elemento da diagonal principal da matriz triangular superior.

18. `plottar_matriz_de_rigidez_global(K, F, points, boundary_conditions):`

Plota a matriz de rigidez global.

- **Parâmetros**:
  - `K`: Matriz de rigidez global.
  - `F`: Vetor de forças.
  - `points`: Array de pontos.
  - `boundary_conditions (opcional)`: Array de pontos com o valor de contorno.

- **Método**:
    - Plotta a matriz de rigidez global aplicando as `boundary_conditions` utilizando a função `aplicar_condicoes_de_contorno.`

19. `plotar_solucao_3d(U):`

Plotta a matriz de solução em 3D.

- **Parâmetros**:
    - `U:` Matriz de resultado.
- **Método**:
    - Plotta a solução em 3D.

20. `ler_arquivo(nome_arquivo):`

Lê um arquivo .json no seguinte formato:

```
{
  "points": [[x1, y1], [x2, y2], ...],
  "triangles": [[p1, p2, p3], ...],
  "loads": {x1, load}, ...},
  "boundary_conditions": {x1, value}, ...},
  "k": 1
}
```

- **Parâmetros**:
    - `nome_arquivo:` Nome do arquivo (aceita path).
- **Saída**:
    - `points`: Array com as coordenadas dos pontos
    - `triangles`: Array de triângulos, cada triângulo representado pelos índices dos pontos.
    - `loads:` Dicionário de nó => carga
    - `boundary_conditions`: Dicionário de nó => valor no contorno
    - `k:` Coeficiente de condutividade térmica
- **Método**:
    - Carrega o arquivo json e retorna às variáveis transformadas.

Execução

1. **Geração da Malha**:
    - Gera uma malha estruturada com NxN pontos.
2. **Plotagem da Malha e da Matriz de Adjacência**:
    - Plota a malha estruturada.
    - Plota a matriz de adjacência.
3. **Montagem e Resolução do Sistema**:
    - Calcula a matriz de rigidez local.
    - Monta a matriz de rigidez global.
    - Impõe as condições de contorno.
    - Resolve o sistema linear.
4. **Visualização da Solução**:
    - Plota a solução aproximada.

Resultados

- A malha estruturada e a matriz de adjacência foram geradas e plotadas corretamente.
- A matriz de rigidez global foi montada e as condições de contorno foram impostas.
- A solução do sistema foi obtida e visualizada em um gráfico 3D.

Conclusão

O código implementa com sucesso a resolução de um PVC usando o método dos elementos finitos. Ele gera a malha, calcula as matrizes necessárias, impõe as condições de contorno e resolve o sistema linear para obter a solução aproximada. A visualização da malha e da solução proporciona uma compreensão clara do comportamento da solução no domínio considerado.

Recomendações

- Para malhas maiores, pode ser necessário otimizar o código para melhorar o desempenho.
- Adicionar tratamento de erros para garantir a robustez do código em diferentes situações.
- Expandir o código para lidar com diferentes tipos de condições de contorno e funções f(x,y).