Universidade Estadual de Santa Cruz – UESC

Relatórios de Implementações de Métodos da Disciplina Análise Numérica

Relatório de implementações realizadas por
Swami de Paula Lima

Disciplina Análise Numérica

Curso Ciência da Computação

Semestre 2021.1

Professor Gesil Sampaio Amarante II

Ilhéus – BA
2021

ÍNDICE

1. Linguagens e Padrões

1.1 Python: a Linguagem escolhida

A linguagem escolhida para a implementação de todos os métodos foi Python. Todas as implementações foram feitas na versão 3.8.8 do interpretador, no ambiente Windows.

No início, por mera conveniência, a linguagem escolhida havia sido C++. A escolha primária do C++ se dava ao fato da familiaridade com a linguagem, que é amplamente explorada no curso. A manipulação simples de arranjos das mais variadas dimensões nas linguagens da família C também foram um fator inicial determinante. Entretanto, apesar da facilidade e aparente rapidez na produção dos códigos para C, ainda no início uma alternativa teve de ser considerada: Não havia forma simplificada, simples ou direta de analisar funções matemáticas passadas pelo usuário. Os métodos, inicialmente, foram implementados usando uma função estática, não definida pelo usuário, mas enviar essa função se mostrou trabalhoso e pouco eficiente.

Dessa forma, simplesmente por causa de uma função nativa do Python, todos os códigos foram migrados para o mesmo. Todo o aprendizado da linguagem teve de acontecer do 0, visto que até então, não havia familiaridade nenhuma com a linguagem Python. Alguns dias tiveram de ser dedicados a aprendizado básico e implementação dos métodos transcrevendo-os de C, procurando similaridades e fazendo o máximo de associação simples. A função que causou a migração, chamada *exec( )*, executa uma string passada como parâmetro como se fosse uma linha de código. Com isso, a função, previamente formatada conforme as especificações, pode ser passada para o código e ser executada. Uma facilidade encontrada é o fato dessa função executada não ser encapsulada. Assim, variáveis definidas dentro dessa função podem ser capturadas e utilizadas fora de seu escopo.

Dado determinado momento, as ferramentas de manipulação de *arrays* na linguagem python começaram a se mostrar ineficientes e exaustivas. A forma de captura das matrizes no arquivo foi mantida, por já ter sido implementada, mas uma ferramenta foi adicionada: a biblioteca *NumPy*, que se torna dependência a partir de então e deve ser instalada. O que ela faz é facilitar a manipulação numérica de arranjos e matrizes, fazendo com que possam ser mais simples de manipular, como acontece naturalmente na linguagem C. Alguns métodos nativos da biblioteca também facilitam a utilização de numérica de alguns arranjos e manipulações simples que se tornam necessárias as vezes, como matrizes já preenchidas com um valor inicial e métodos de tamanho e grandeza, além de nativamente conter aritmética de matrizes.

1.2 Definições de entrada e saída

Por conta da escolha da linguagem, e para facilitar a implementação, algumas coisas são padrões entre os métodos de entrada, além de sua estrutura. Entre elas, podemos listar:

- No caso das funções de entrada, devem respeitar a sintaxe natural da linguagem Python. A aritmética utilizada é simples (+, -, *, /), porém a exponenciação deve ser observada. Caso queiramos escrever a função:

$$x^2 + 5x + 2$$

Ela deverá ser escrita da seguinte maneira:

$$x ** 2 + (5 * x) + 2$$

com a fatoração sendo escrita na forma ***base ** expoente***. Observe também que *5x* foi escrito entre parênteses, para garantir que vai ser devidamente calculado, e que é necessário colocar o símbolo da multiplicação entre *5* e *x*.

- Caso tenhamos matrizes, elas devem ser escritas linha por linha, separadas por vírgulas. Na resolução de sistemas, os sistemas devem ser convertidos previamente para a matriz. Então, o sistema:

$$2x_1 + 8x_2 - 9x_3$$
$$x_1 - 5x_2 - 8x_3$$
$$6x_1 + x_2 - 4x_3$$

  deveria ser escrito na entrada como:

```
2, 8, -9
1, -5, -8
6, 1, -4
```

- Funções trigonométricas podem ser escritas português, da seguinte maneira:
  1. Seno: sen(x);
  2. Cosseno: cos(x);
  3. Tangente: tan(x).

- As divisões/frações devem ser feitas usando parênteses. Caso seja um elemento, ele inteiro deve estar entre parênteses:

  $7⁄8\ x_{1,\ldots}$

  deve ser escrita na forma:

```
(7/8)
```

  caso seja parte de uma função/equação, o numerador deve estar entre parênteses:

$$\frac{x^2 + 2x}{3}$$

  deve ser escrito na forma:

```
(x**2 + (2 * x))/3
```

- É extremamente importante que todas as funções de entrada estejam em função de X.
- Todos os arquivos de entrada deverão ter o nome tNOMEDOMETODO.in e a saída será sempre tNOMEDOMETODO.out.

As definições das entradas e a forma como devem estar dispostas no arquivo serão ditas em cada um dos métodos.

2. Implementações de Métodos

2.1. Derivadas Numéricas de Primeira e segunda ordem

Este método de derivação é praticamente um parser. Ele transforma um texto dado para a sua forma derivada. O gráfico também é plotado no final do código (implementado de forma experimental).

2.1.1. Estrutura dos arquivos de entrada e saída

Os arquivos de entrada seguem o seguinte padrão:

- Expressão **f(x)**, formatada com o padrão no item 1.2;
- O valor de **x**, inteiro;
- O valor de **delta**, real;
- A ordem de derivação (1ª ou 2ª).

Após algumas análises de símbolos, o **";"** foi escolhido para separar as informações, ao invés de quebras de linhas, como nas implementações anteriores, para dar mais dinâmica para os arquivos. O seu uso foi por não haver uso comum para este símbolo em Python.

O arquivo de saída gerará uma lista, em linhas, de todas as expressões derivadas segundo as especificações dos dados de entrada.

Como bônus, usei por sugestão o Mathplot, para plotar a função. O código implementado não é meu, mas queria ver como a função se comportaria.

2.1.2. Dificuldades enfrentadas

Essa dificuldade foi a mesma enfrentada em todas as outras implementações, e será explicada aqui: tempo. Devido a problemas de saúde conhecidos, tive apenas 4 dias para realizar a implementação dos métodos, além de fazer estudos complementares. Precisei estudar alguns códigos e algoritmos que encontrei na internet e ver mais alguns vídeos de como fazer isso em Python, já que sou Júnior ainda na linguagem e não tive tempo de me aprimorar. A maioria das dificuldades, foi tempo e desconhecimento de usar plenamente a linguagem, como o uso de funções.

2.2. Integração por Trapézio Simples e Multiplo

Este método foi implementado de forma bem simples: dado os valores iniciais e finais de **x**, calculamos a área de cada trapézio dado. A quantidade de trapézios é fornecida na entrada, e a base to todos os trapézios são iguais, sendo equidistantemente divididas.

2.2.1. Estrutura dos arquivos de entrada e saída

Os arquivos de entrada seguem o seguinte padrão:

- Expressão **f(x)**, formatada com o padrão no item 1.2;
- Dois valores para **x**, inteiros, sendo um final e um inicial, separados por vírgulas;
- O número de trapézios no qual a área da função deve ser dividida.

Após algumas análises de símbolos, o **";"** foi escolhido para separar as informações, ao invés de quebras de linhas, como nas implementações anteriores, para dar mais dinâmica para os arquivos. O seu uso foi por não haver uso comum para este símbolo em Python.

O arquivo de saída gerará uma linha com o valor do cálculo da integral.

2.1.2. Dificuldades enfrentadas

As dificuldades principais foram as mesmas do item acima. As funções foram conferidas usando bibliotecas conhecidas de Python.

2.3. Regra de Simpson (⅓ e ⅜)

A forma apresentada de solução exclui a forma dinâmica de análise da melhor forma de implementação do segmento. Por falta de tempo para o polimento, não consegui criar uma forma de análise que fizesse com que o programa analisasse se o uso de ⅓ ou ⅜ seria melhor. Dessa forma, o método deve ser inserido pelo usuário, e será usado para o cálculo. Para meus estudos posteriores em Python, essa implementação deverá ser feita.
O método então recebe os valor inicial de x e o final, do segmento, e aplicando a fórmula usa trios ou quartetos de pontos para criar trapézios que podem ser facilmente calculados.

2.3.1. Estrutura dos arquivos de entrada e saída

Os arquivos de entrada seguem o seguinte padrão:

- Expressão **f(x)**, formatada com o padrão no item 1.2;
- Dois valores para **x**, inteiros, sendo um final e um inicial, separados por vírgulas;
- O número de trapézios no qual a área da função deve ser dividida;
- O tipo de operação no qual se deseja usar (⅓ ou ⅜**).**

Após algumas análises de símbolos, o **";"** foi escolhido para separar as informações, ao invés de quebras de linhas, como nas implementações anteriores, para dar mais dinâmica para os arquivos. O seu uso foi por não haver uso comum para este símbolo em Python.

O arquivo de saída gerará uma lista, em linhas, de todas soluções pedidas na entrada. O valor dado é apenas do valor total da integral.

2.3.2. Dificuldades enfrentadas

Além das já descritas, a única dificuldade foi na criação de uma forma dinâmica de identificação, para escolher a melhor forma de cálculo.

2.4. Polinômios de Lagrange

Esse método recebe os pontos dados e calcula através do uso da interpolação pelos polinômios na base de Lagrange.

2.4.1. Estrutura dos arquivos de entrada e saída

Os arquivos de entrada seguem o seguinte padrão:

- Uma sequencia de pontos, onde os valores de **x** e **y** de cada ponto estão entre parênteses e separados por vírgula. Cada tupla de pontos deve estar separada por um **“;”**.

O arquivo de saída gerará um polinômio de lagrange.

Como bônus, usei por sugestão o Mathplot, para plotar a função. O código implementado não é meu, mas queria ver como a função se comportaria.

2.4.2. Dificuldades enfrentadas

Nenhuma além das já descritas.

2.5. Regressão Linear

A regressão Linear aqui implementada foi feita da forma bem simplificada: diferente de separar os pontos em (x,y), como a média é importante e para facilitar os cálculos, os valores de **x** e **y** foram colocados em arranjos separados.

2.5.1. Estrutura dos arquivos de entrada e saída

Os arquivos de entrada seguem o seguinte padrão:

- Valores de **x** e **y,** entre parênteses e separados por **;.** Note a diferença na forma, já que não são tuplas. A entrada deve ser na forma:

$$(x_1, x_2, x_3, \dots, x_n); (y_1, y_2, y_3, \dots, y_n)$$

Após algumas análises de símbolos, o **“;”** foi escolhido para separar as informações, ao invés de quebras de linhas, como nas implementações anteriores, para dar mais dinâmica para os arquivos. O seu uso foi por não haver uso comum para este símbolo em Python.

O arquivo de saída gerará um valor, de crescimento ou queda.

Para demonstrar melhor o método, a biblioteca Mathplot foi usada, de forma experimental.

2.5.2. Dificuldades enfrentadas

A dificuldade aqui foi em fazer a biblioteca MthPLot funcionar como esperado.

2.6. Polinômios por diferenças divididas de Newton

Como uma forma quase igual à de Lagrange, não há muito o que explicar sobre a implementação, além das modificações feitas no método de cálculo para a adaptação do método.

2.6.1. Estrutura dos arquivos de entrada e saída

Os arquivos de entrada seguem o seguinte padrão:

- Uma sequencia de **n** tuplas (x,y), entre parênteses, separadas por **;**, que constituem uma sequência de pontos.

O arquivo de saída gerará um polinômio.

Para realizar a impressão do gráfico, novamente foi usada a biblioteca Mathplot.

2.6.2. Dificuldades enfrentadas

Por ser uma melhoria de um método já aplicado, não houveram grandes dificuldades.

3. Considerações Finais

Diferente do outro relatório, no qual faltaram duas implementações por conta de não conseguir implementar, dessa vez os faltantes se dão pela falta de tempo. Devido a graves problemas de saúde, meus e de minha mãe, tive apenas o período de 4 dias para estudar, fazer os códigos e este relatório. O tempo, ainda que eu tenha passado praticamente o dia inteiro nos estudos, se mostrou insuficiente para criar todo o necessário. Entretanto os métodos criados funcionam e estarão disponíveis para acesso no repositório criado no GitHub.

A forma com que os métodos foram implementados não melhoraram, já que não tive tempo de estudar a linguagem. Assim, ainda faltam pequenas melhorias nas implementações, como melhor uso de bibliotecas e aplicação de funções.

As implementações, completas e incompletas, casos de teste e demais arquivos, podem ser encontrados no repositório do GITHUB: **https://github.com/LordTesseract/CET086.** Minha intenção é completar este mesmo com o fim da matéria. Acredito que seja importante para meu crescimento essas implementações, visto que elas são difíceis e as soluções podem melhorar meu entendimento, tanto da linguagem, como dos recursos disponíveis para suas implementações.