Cálculo Numérico: Projeto da Calculadora de Soma e Subtração

Cleber Alberto Cabral Ferreira da Silva, Heloísa Bezerra Neves
Alunos da disciplina de Cálculo Numérico, Turma T7.

Resumo: Este projeto tem como objetivo realizar operações básicas de soma e subtração para uma máquina com precisão pré-estabelecida. Para isso, foi elaborado um programa em linguagem C que execute os comandos do usuário.
Palavras-chave: Soma, Subtração, Calculadora, Máquina, Linguagem C.

1. Introdução:

A maioria dos problemas que ocorrem em matemática numérica é causada pelo fato de que na aritmética de máquina não se consegue realizar as operações como na matemática real, uma vez que as máquinas trabalham em um subconjunto finito de números racionais (SANTOS; SILVA, 2010, p26).

Neste projeto, implementaremos uma calculadora com operações básicas de adição e subtração, a partir de uma parametrização de Máquina fornecida por um arquivo de texto, que também conterá os dados que devem ser calculados.

O usuário informará o nome do arquivo contendo os dados ao programa, onde será feito o processamento e cálculo dos resultados.

Primeiramente, devemos estabelecer as regras que este arquivo deve obedecer, além do limites inferiores e superiores dos parâmetros da máquina. O arquivo esperado segue o modelo:

$$t;\ e_{min};\ e_{max};\ N$$
$$x_1^1;x_2^1;operação_1$$
$$x_1^2;x_2^2;operação_2$$
$$\dots$$
$$x_1^N;x_2^N;operação_N$$

O arquivo deve conter em sua primeira linha os dados da Máquina a qual devemos operar, onde N significa o número de operações a ser realizada e x_j^i é um número com mantissa normalizada com 7 dígitos significativos. Além disso, restringiremos os demais parâmetros através dos seguintes limites para a especificação da máquina:

Parâmetro	Descrição	Limites
t	Número de dígitos significativos.	$2 <= t <= 7$
e_{min}	Expoente mínimo.	$-9 <= e_{min} <= -4$
e_{max}	Expoente máximo.	$4 <= e_{max} <= 9$

Sendo assim, temos a seguinte estrutura de um arquivo de texto válido:

```
4;-6;+6;5
+4.345670E+01;+2.125000E+00;+
+1.000000E-02;-4.342107E+03;-
+6.590472E-01;+2.771043E+07;+
-5.003500E-02;-8.679890E-02;-
+2.083090E-05;+2.081400E-05;-
```

A partir da segunda linha em diante, temos a primeira parcela (sinal, mantissa normalizada com seis casas decimais seguida do expoente), segunda parcela e operação de soma ou subtração. Todas as três informações estarão separadas por ";".

Nosso software ficará incubido de validar as informações passadas, tanto da mantissa, quanto das parcelas informadas, de forma que acuse qualquer parcela inválida ou se o resultado obtido esteja fora da região de operação da máquina, identificando-o como *underflow* ou *overflow*.

2. Desenvolvimento:

Sendo estabelecidas as regras descritas na introdução, agora discorreremos sobre as técnicas e o fluxos utilizados para o cálculo dos resultados obtidos.

2.1. Elaboração do Código:

Nosso programa foi desenvolvido na linguagem C utilizando as bibliotecas <stdio.h> e <stdlib.h>. Sendo a primeira contendo as funções básicas `printf`, `scanf` e `fscanf,` e na segunda a função `abs`.

Todas variáveis do projeto são `int`, `char` ou variantes das duas.

Definimos dois `struct` que representam a Máquina e um Número qualquer, conforme exemplo:

```
/**
 * Struct representando a Máquina
 */
typedef struct {
    int nSignificativos;
    int expoenteMinimo;
    int expoenteMaximo;
    int qtdOperacoes;
} Maquina;
```

```
/**
 * Struct representando um Número
 */
typedef struct {
    int sinal;
    int expoente;
    int mantissa[15];
} Numero;
```

Tomamos uma atenção especial com o struct do Número que possui algumas peculiaridades. O seu atributo de sinal terá o valor "-1" quando se tratar de um número negativo e "1" quando for um número positivo. Seu atributo de mantissa receberá os algarismos lidos do arquivo, descartando o ponto que separa o primeiro numeral, das casas decimais. Também tomamos a precaução de atribuir zero para todas as posições da mantissa antes de preenchê-la com os valores do arquivo.

Por exemplo, o número +6.875408E-03 seria representado por:

```
int sinal = +1;
int expoente = -3;
int mantissa[15] = {6, 8, 7, 5, 4, 0, 8, 0, 0, 0, 0, 0, 0, 0, 0};
```

Os limites da mantissa estão separados em constantes definidas após os struct.

```
const int BASE_PADRAO = 10; /* Base padrao da maquina */
const int MIN_DIGITOS = 2; /* Limite inferior dos digitos significativos */
const int MAX_DIGITOS = 7; /* Limite superior dos digitos significativos */
const int MIN_EXPOENTE_MINIMO = -9; /* Limite inferior do expoente minimo */
const int MAX_EXPOENTE_MINIMO = -4; /* Limite superior do expoente minimo */
const int MIN_EXPOENTE_MAXIMO = 4; /* Limite inferior do expoente maximo */
const int MAX_EXPOENTE_MAXIMO = 9; /* Limite superior do expoente maximo */
```

Ao executar o projeto, fornecemos ao usuário um menu com duas opções. A primeira opção (número 1) diz respeito que o usuário deve informar o nome do arquivo de texto contendo os dados. Este

mesmo arquivo deve estar presente na mesma pasta onde encontra-se o executável do projeto. A segunda opção (número 2) apenas finaliza o programa. Segue o exemplo do menu do sistema:

```
-----------------------------------------------
-- UFPE - Calculo Numerico - Calculadora v1.0 --
-----------------------------------------------
-- Prof: Andre Tiba                          --
-- Alunos: CLEBER ALBERTO CABRAL F DA SILVA   --
--         HELOISA BEZERRA NEVES              --
-----------------------------------------------
Digite o numero da opcao desejada:
1 - Digitar o nome do arquivo;
2 - Sair do programa.
```

Sobre a opção menu, o software perguntará ao usuário infinitas vezes caso ele informe um número diferente de 1 ou 2. Após selecionar a primeira opção, será requerido o nome do arquivo que contém os dados, no formato ([nome].[extensão]). Caso o arquivo não exista ou ocorra algum erro de leitura, o programa informará ao usuário que não conseguiu ler o arquivo.

Ao ler o arquivo, preencheremos nosso `struct Maquina` com as informações provenientes do arquivo, e outros dois `struct Numero` com as informações dos algarismos para cada linha lida. Estes structs serão validados de acordo com os valores preestabelecidos na introdução do projeto. Caso eles estejam inválidos, exibimos uma mensagem de erro para o usuário informando quais parametrizações estão incorretas, do contrário, partimos para o cálculo das operações.

O resultado da operação de uma linha será calculado imediatamente após sua validação. Sendo assim, não será necessário ter um vetor contendo todos os números de todas as linhas. O preenchimento dos structs foi feito através da função `fscanf`.

2.2. Fluxo de processamento:

Agora descreveremos o fluxo utilizado pelo programa para realizar as operações.

Seguindo as recomendações do fluxo da operação de soma descrito no livro-texto visto em sala de aula, pudemos elaborar um diagrama de atividades adaptado à nossa realidade na linguagem. Ele abrangerá as dificuldades que presenciamos ao realizar operações com as mantissas e expoentes dos números, além de outros pontos que valem ser ressaltados.

Logo abaixo vamos apresentar o pseudocódigo proposto pelo livro-texto, e logo abaixo nossa adaptação do mesmo em forma de um diagrama de atividades:

1. Se $c_1 \neq c_2$; *(igualar os expoentes)*
faça
Se $c_1 < c_2$;
faça
$m_1 = m_1 \times b^{c_1 - c_2}$;
$c = c_2$;
fim ;
Senão;
faça
$m_2 = m_2 \times b^{c_2 - c_1}$;
$c = c_1$;

2. $m = m_1 + m_2$;

3. Se $|m| \geq b$; *(normalizar)*
faça
$m = m \times b^{-1}$;
$c = c + 1$;
Se $c > e_2$
"overflow";
Pare.
fim

4. Se $|m| < 1$; *(normalizar)*
enquanto $|m| < 1$;
faça
$m = m \times b$;
$c = c - 1$;
Se $c < e_1$
"underflow";
Pare.
fim;

5. $m = O(m)$; *(arredondar)*
$x = m \times b^c$;
Pare.

6. Fim.

Imagem 1: Pseudocódigo do algoritmo da adição

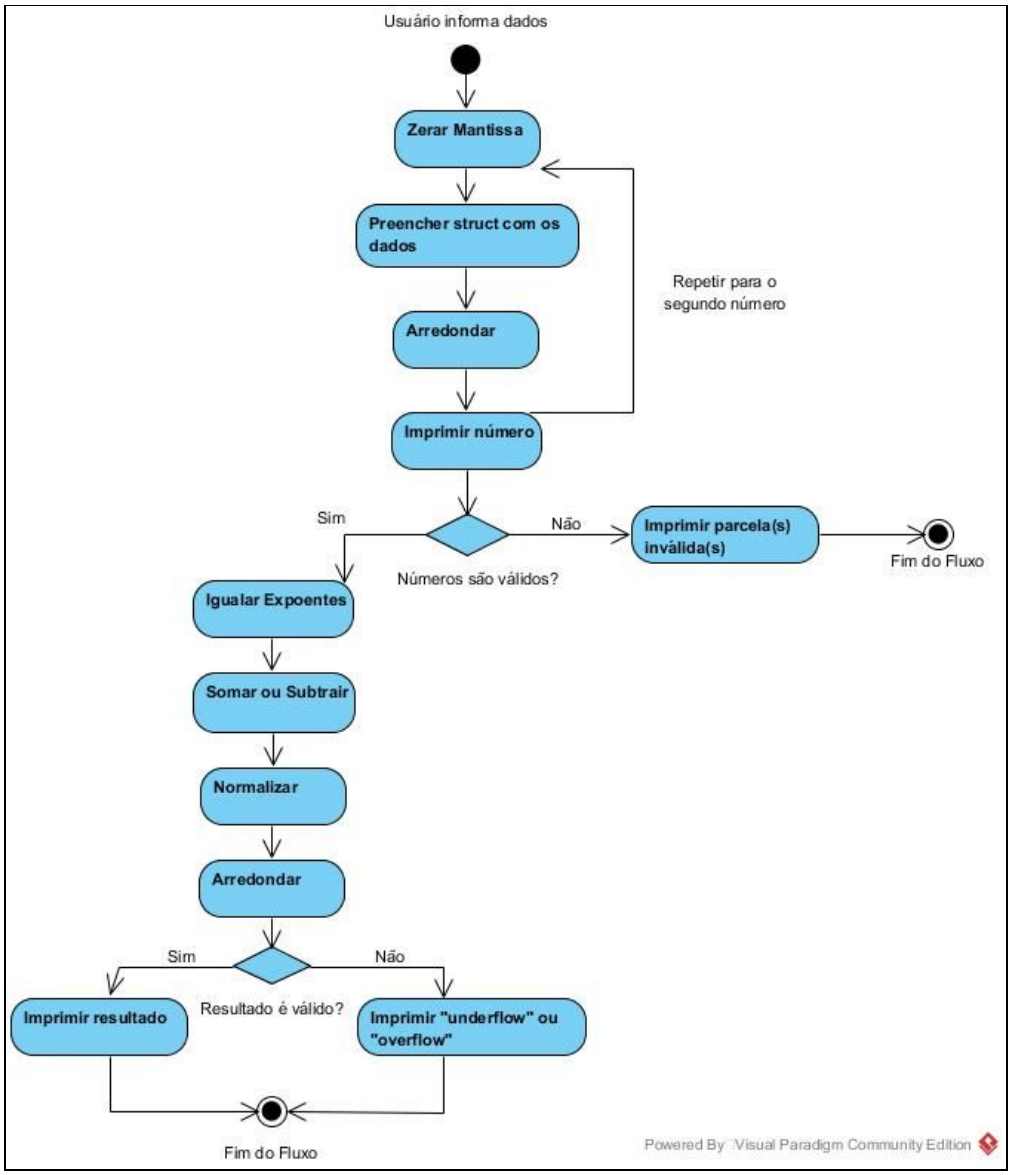

Imagem 2: Diagrama de atividade baseado no algoritmo

Percebemos que ao desenvolver a solução escrita em C, acrescentamos algumas atividades auxiliares para realizar os cálculos, como por exemplo a de *Zerar a Mantissa*.

Como sabemos, nossos números são formados por um vetor de inteiros que representa a mantissa. Para efeito de facilidade em nossos cálculos, antes de preencher os dados de um número nesta mantissa, atribuímos o valor zero para todas as posições do vetor da mantissa, a fim de evitar qualquer erro de referência.

2.2.1. Arredondamento:

O livro-texto define a função arredondamento como da seguinte maneira:

Definição 1.3: Seja $\mathbf{F}(b, t, e_1, e_2)$ um sistema de ponto flutuante. Uma função

$\mathbf{O} : \mathcal{R} \rightarrow \mathbf{F}$

$x \rightarrow \mathbf{O}x = \bar{x}$, é chamada um arredondamento se:

1) $\mathbf{O}x = \bar{x} = x$, se $x \in \mathbf{F}$;

2) Se $x \leq y$ então $\mathbf{O}x = \bar{x} \leq \mathbf{O}y = \bar{y}$;

3) Se $x \notin \mathbf{F}$ então entre x e $\mathbf{O}x = \bar{x}$ não existem elementos de **F.**

Imagem 3: Definição de arredondamento

Através desta definição, e da teoria vista em sala de aula, elaboramos uma função de arredondamento que observa os números da mantissa de acordo com o número de algarismos significativos parametrizado na Máquina. A normalização está sendo aplicada após a leitura e validação dos números e mais uma vez no cálculo do resultado final.

Em nossa rotina, lido um número qualquer, observamos o algarismo *x* da mantissa correspondente na posição do módulo da quantidade de algarismos significativos, e aplicamos as seguintes regras:

Regra	Decisão
$x > 5$	Incremento +1 no algarismo anterior ao atual.
$x < 5$	Atribuo $x = 0$.
$x = 5$	Verifico se há após *x* algum número diferente de zero. Caso exista, incremento +1 no algarismo anterior ao atual, do contrário, nada será alterado.

Após o arredondamento, os algarismos desde a posição do módulo da quantidade de números significativos até o final da mantissa serão atribuídos para zero.

2.2.2. Ajustes dos Expoentes:

Após o arredondamento e validação das parcelas dos números, agora chegamos à prévia do cálculo em si. Antes de realizar o cálculo, assim como o pseudocódigo do livro-texto nos diz, precisamos igualar os expoentes dos números lidos, segundo a regra descrita:

1. Se $c_1 \neq c_2$; *(igualar os expoentes)*
   faça
      Se $c_1 < c_2$;
         faça
            $m_1 = m_1 \times b^{c_1 - c_2}$;
            $c = c_2$;
         fim ;
      Senão;
         faça
            $m_2 = m_2 \times b^{c_2 - c_1}$;
            $c = c_1$;
2. $m = m_1 + m_2$;

Assim como podemos perceber, à medida que o expoente de um dos números é adequada à do outro, sua mantissa também é modificada. Neste caso, N zeros serão adicionados ao início da mantissa, onde $N = c_1 - c_2$ quando $c_1 < c_2$ ou $N = c_2 - c_1$ quando $c_1 > c_2$. Sendo assim, tivemos que implementar uma função de *padding* da mantissa, que pode ser conferida abaixo, onde `p` equivale a N, `m` à mantissa e `n` à quantidade de posições que há na mantissa.

```
void padding(int p, int* m, int n) {

    int i;

    for (i = n - 1; i >= 0; i--) {
        if (i + p <= n - 1) {
            m[i + p] = m[i];
        }
        m[i] = 0;
    }
}
```

2.2.3. Soma ou Subtração:

Já com os números ajustados à mesa potência, agora chegamos na parte fundamental da calculadora: a soma ou subtração. De acordo com o operador lido no arquivo (+ ou -), devemos decidir qual fluxo seguir.

2.2.3.1. Soma:

No caso da Soma, efetuamos operações simples entres as mantissas, somando item a item do vetor de inteiros, partindo do final ao início do vetor.

	Posição	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	Número 1:	1	2	3	5	6	3	4	0	0	0	0	0	0	0	0
+	Número 2:	4	7	6	4	3	0	1	0	0	0	0	0	0	0	0
	Resultado:	**5**	**9**	**9**	**9**	**9**	**3**	**5**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Lembrando que no algoritmo, quando a soma de dois números for maior que nove, devemos somar +1 no algarismo anterior. Sendo assim, devemos estar atentos quanto a esta verificação no último algarismo das iterações, neste caso, o primeiro elemento da mantissa. Caso ele seja maior que nove, devemos aplicar um *padding* de uma posição para a direita, deixar o resto da divisão por 10 do algarismo maior que nove, e somar +1 no novo item adicionado, conforme exemplo:

	Posição	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	Número 1:	9	2	3	5	6	3	4	0	0	0	0	0	0	0	0
+	Número 2:	4	7	6	4	3	0	1	0	0	0	0	0	0	0	0
	Resultado inicial:	**13**	**9**	**9**	**9**	**9**	**3**	**5**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	Deslocando:	**0**	**13**	**9**	**9**	**9**	**9**	**3**	**5**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	Resultado final:	**1**	**3**	**9**	**9**	**9**	**9**	**3**	**5**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Após o *padding,* também será somado +1 no expoente do resultado.

2.2.3.2. Subtração:

No caso da subtração, temos que utilizar uma metodologia diferente ao subtrair os dois números. Primeiramente, devemos identificar qual dos dois números informados é o maior em módulo, que será o minuendo da operação.

Semelhantemente à adição, devemos subtrair os valores do final ao início das mantissas, com o diferencial de que, caso o valor do minuendo naquela posição seja menor que o do subtraendo, devemos incrementar uma dezena ao minuendo e somar +1 ao algarismo de uma casa à esquerda da mantissa. No demais, seguimos o fluxo normal de uma subtração.

	Posição	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	Número 1:	9	2	3	5	6	3	4	0	0	0	0	0	0	0	(
-	Número 2:	4	7	6	4	3	0	1	0	0	0	0	0	0	0	(
	Resultado:	**4**	**4**	**7**	**1**	**3**	**3**	**3**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**(**

Podemos perceber que no caso da subtração não utilizar a técnica do *padding* para corrigir os valores da mantissa.

2.3. Testes de Execução:

Agora iremos focar nos testes das possíveis situações que o programa pode enfrentar, tanto relacionado a Máquina quanto aos números e seu resultado.

2.3.1. Máquina inválida:

Neste teste iremos verificar se o programa está validando a Máquina segundo os limites estabelecidos na introdução do relatório.

Arquivo de Entrada	**Saída**
1;-10;+10;1 +4.345670E+01;+2.125000E+00;+	Processando o arquivo... O num. digitos significativos deve obedecer 2 <= N <= 7. Valor atual: 1 O expoente minimo deve obedecer -9 <= N <= -4. Valor atual: -10 O expoente maximo deve obedecer 4 <= N <= 9. Valor atual: 10 Parametros incorretos. Corrija os erros listados acima.

Como podemos perceber, os limites foram verificados.

2.3.2. Máquina válida:

Arquivo de Entrada	**Saída**
4;-6;+6;1 +4.345670E+01;+2.125000E+00;+	Processando o arquivo... F(10,4,-6,6) numero de oper.: 1 +4.346E1;+2.125E0;+; resultado: +4.558E1 Fim do processamento.

Com a parametrização da máquina ajustada, tudo ocorreu bem.

2.3.3. Parcelas Inválidas:

Vamos informar um arquivo com três operações. A primeira possui as duas parcelas dos números inválidas, a segunda apenas o primeiro número inválido e finalmente na última apenas o segundo número inválido. Verifiquemos se o programa identificou os erros de validação:

Arquivo de Entrada	**Saída**
4;-6;+6;3 +4.345670E+11;+2.125000E-07;+ +1.000000E-09;-4.342107E+03;- +6.590472E-01;+2.771043E+07;+	Processando o arquivo... F(10,4,-6,6) numero de oper.: 3 +4.346E11;+2.125E-7;+; resultado: parcela 1 invalida, overflow. parcela 2 invalida, underflow. +1.000E-9;-4.342E3;-; resultado: parcela 1 invalida, underflow +6.590E-1;+2.771E7;+; resultado: parcela 2 invalida, overflow. Fim do processamento.

Como pudemos verificar, a rotina identificou e exibiu os erros de validação.

2.3.4. Soma ou Subtração de números distantes:

Neste caso especial, iremos testar uma das funções da técnica de arredondamento. Aplicaremos a soma ou subtração de números muitos distantes entre si, buscando verificar se o resultado final será igual ao número de maior módulo.

Arquivo de Entrada	Saída
4;-6;+6;2 +4.345670E+01;+2.125000E-06;+ +1.000000E-05;+4.342107E+03;-	Processando o arquivo... F(10,4,-6,6) numero de oper.: 2 +4.346E1;+2.125E-6;+; resultado: +4.346E1 +1.000E-5;+4.342E3;-; resultado: -4.342E3 Fim do processamento.

Como podemos ver, os resultados tenderam a ser iguais aos números de maior módulo.

2.3.5. Quantidade de números significativos:

Verificaremos se o programa se adequa a uma nova quantidade de dígitos significativos informada.

Arquivo de Entrada	Saída
7;-6;+6;2 +4.345670E+01;+2.125000E-06;+ +1.000000E-05;+4.342107E+03;-	Processando o arquivo... F(10,7,-6,6) numero de oper.: 2 +4.345670E1;+2.125000E-6;+; resultado: +4.345670E1 +1.000000E-5;+4.342107E3;-; resultado: -4.342107E3 Fim do processamento.
2;-6;+6;2 +4.345670E+01;+2.125000E-06;+ +1.000000E-05;+4.342107E+03;-	Processando o arquivo... F(10,2,-6,6) numero de oper.: 2 +4.3E1;+2.1E-6;+; resultado: +4.3E1 +1.0E-5;+4.3E3;-; resultado: -4.3E3 Fim do processamento.

Podemos ver que o programa adequou-se bem aos novos dígitos significativos.

2.3.6. Resultados inválidos:

Neste teste verificaremos se o programa identifica números que ultrapassam os limites da Máquina.

Arquivo de Entrada	Saída
4;-6;+6;2 +4.345670E+06;+6.125000E+06;+ +2.083090E-05;+2.081400E-05;-	Processando o arquivo... F(10,4,-6,6) numero de oper.: 2 +4.346E6;+6.125E6;+; resultado: overflow +2.083E-5;+2.081E-5;-; resultado: underflow Fim do processamento.

Como visto, o programa identificou bem os resultados que ultrapassam os limites da Máquina.

2.3.7. Regra de Sinais:

Como o próprio teste já diz, verificaremos se o programa aplica as regras matemáticas do sinais:

Arquivo de Entrada	Saída
4;-6;+6;3 +4.345670E+01;+2.125000E-06;- +1.000000E-05;-4.342107E+03;- -6.590472E-01;+2.771043E+06;+	F(10,4,-6,6) numero de oper.: 3 +4.346E1;+2.125E-6;-; resultado: +4.346E1 +1.000E-5;-4.342E3;-; resultado: +4.342E3 -6.590E-1;+2.771E6;+; resultado: +2.771E6 Fim do processamento.

Como vemos, os números de maior módulo tem preferência sobre o sinal, além de que o operador matemático também influenciou no sinal do resultado.

2.3.8. Subtração de Números Iguais:

Veremos se o programa traz como resultado o número zero.

Arquivo de Entrada	Saída
4;-6;+6;2 +4.345670E+01;+4.345670E+01;- +4.342107E+03;-4.342107E+03;+	Processando o arquivo... F(10,4,-6,6) numero de oper.: 2 +4.346E1;+4.346E1;-; resultado: 0.000E0 +4.342E3;-4.342E3;+; resultado: 0.000E0 Fim do processamento.

O resultado esperado foi almejado, como vemos.

3. Conclusões:

Podemos concluir a partir destes testes que a calculadora implementada respondeu bem às funcionalidades esperadas, tanto no tratamento das mantissas quando nas regras de sinais e expoentes dos números.

Além disso, podemos verificar que os algoritmos de ordenação e arredondamento funcionaram como esperado e de formas eficazes.

4. Referências Bibliográficas:

- SANTOS & SILVA, Métodos Numéricos 3ª Edição, Editora Universitária UFPE, 2010.