Aproximação Numérica: Cálculo de Raiz Quadrada e Seno em Assembly MIPS
Organização de Computadores I

Juliana Miranda Bosio, Vinícios Rosa Buzzi

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
Caixa Postal 5094 – 88035-972 – Florianópolis – SC – Brazil

Cálculo do Erro Absoluto

Para avaliar a precisão do método iterativo de Newton no cálculo da raiz quadrada e do cálculo do seno utilizando a série de Taylor, utilizamos o erro absoluto. Este erro é uma medida de quão próximo o valor calculado está do valor exato.

O erro absoluto é definido como a diferença entre o valor exato e o valor calculado pelo método iterativo, em termos absolutos. A fórmula utilizada para calcular o erro absoluto é a seguinte:

$$\text{Erro Absoluto} = |\text{Valor Exato} - \text{Valor Calculado}|$$

- Valor Exato é o resultado obtido utilizando a instrução `sqrt.d` para calcular a raiz quadrada ou a função exata para calcular o seno.
- Valor Calculado é o resultado obtido utilizando o método iterativo de Newton após um número n de iterações para a raiz quadrada, ou a série de Taylor com um determinado número de termos para o seno.

Ao calcular o erro absoluto para diferentes valores de x e diferentes números de iterações, podemos avaliar a convergência e a precisão dos métodos iterativos utilizados. Esta medida é fundamental para entender a eficácia dos algoritmos e ajustar o número de iterações ou termos necessários para alcançar a precisão desejada.

1. Projeto 1: Cálculo da Raiz Quadrada Usando o Método de Newton em Assembly MIPS

1.1. Objetivo

- **Cálculo da Raiz Quadrada:** Implementar um procedimento que utilize o método iterativo de Newton para calcular a raiz quadrada de um número, com uma estimativa inicial de 1.
- **Iterações:** Permitir que o usuário informe o número de iterações para refinar a estimativa da raiz quadrada, variando entre 5 e 20 iterações.
- **Precisão dos Cálculos:** Garantir que todos os cálculos sejam realizados utilizando registradores de ponto flutuante de precisão dupla, assegurando a precisão dos resultados.
- **Comparação de Resultados:** Comparar o resultado do procedimento de raiz quadrada com o resultado da instrução `sqrt.d`, avaliando o erro absoluto entre os dois valores.

1.2. Implementação

1.2.1. Armazenamento dos Dados

O código define uma estrutura principal para armazenar os dados:

- `prompt`: Uma string que armazena a mensagem solicitando ao usuário que informe um número para descobrir sua raiz.
- `iteracoes_prompt`: Uma string que armazena a mensagem solicitando ao usuário que informe o número de iterações.
- `out`: Uma string que armazena a mensagem exibindo a raiz do número informado.
- `x`: Uma variável de precisão dupla (`double`) que armazena o número fornecido pelo usuário.
- `e`: Uma variável de precisão dupla (`double`) que armazena a estimativa inicial.
- `raiz`: Uma variável de precisão dupla (`double`) que armazena a raiz calculada.
- `dois`: Uma variável de precisão dupla (`double`) que armazena o valor 2.0.

1.2.2. Área do Programa

1. **Inicializações de Endereços**:
    - `la $a0, prompt`: Carregar o endereço da mensagem de entrada.
    - `li $v0, 4`: Código de serviço para impressão de string.
    - `li $v0, 5`: Código de serviço para leitura de número inteiro.
2. **Leitura do Valor de x**:
    - `syscall`: Imprimir a mensagem solicitando o valor de `x`.
    - `syscall`: Ler o valor do usuário e armazenar em `$f0`.
    - Conversão de inteiro para ponto flutuante de precisão dupla (`cvt.d.w`).
3. **Leitura do Número de Iterações**:
    - `syscall`: Imprimir a mensagem solicitando o número de iterações.
    - Armazenar o número de iterações em `$t0`.
4. **Chamada da Função Raiz**:
    - `jal raiz_quadrada`: Chamar a função que calcula a raiz quadrada utilizando o método de Newton.
5. **Armazenamento e Exibição do Resultado**:
    - `la $a0, out`: Carregar o endereço da mensagem de resultado.
    - `li $v0, 4`: Código de serviço para impressão de string.
    - `syscall`: Imprimir a mensagem de resultado.
    - `li $v0, 3`: Código de serviço para impressão de ponto flutuante.
    - `syscall`: Mostrar o valor calculado da raiz de `x` na tela.
6. **Finalização do Programa**:
    - `li $v0, 10`: Código de serviço para encerrar o programa.
    - `syscall`: Chamar o serviço do sistema para encerrar.

1.3. Execução do Programa

A execução do programa foi realizada conforme o esperado, sem problemas significativos. Durante o processo, foram observadas várias características e dados nos registradores, demonstrando o funcionamento correto do código. A seguir, analisaremos tecnicamente como cada registrador é utilizado para calcular a raiz quadrada de um número x.

Registrador	Nome	Valor (Hexadecimal)
$f0	Valor de x	0x4048F5C3 (32.0)
$f2	Estimativa inicial e	0x3FF00000 (1.0)
$f4	Constante dois (2.0)	0x40000000 (2.0)
$f6	Quociente	0x00000000
$t0	Contador de iterações	0x0000000A (10)

Table 1. Valores dos registradores durante a execução do cálculo da raiz quadrada de x.

De acordo com a Tabela 1, podemos resumir o processo de cálculo da raiz quadrada da seguinte forma:

- Inicializa-se o registrador `$f0` com o valor de x (neste caso, 32.0), que é o número do qual se deseja calcular a raiz quadrada.
- O registrador `$f2` é carregado com uma estimativa inicial e, que neste exemplo é 2.0. Essa estimativa será refinada ao longo das iterações.
- O registrador `$f4` é utilizado para armazenar o valor constante de 2.0, que é utilizado no cálculo da nova estimativa.
- O loop de iteração começa com a verificação se o contador de iterações (armazenado em `$t0`) é igual a zero. Se for, o loop termina.
- Para cada iteração:
  - O valor de x é dividido pela estimativa atual e (armazenado em `$f2`) e o resultado é armazenado em `$f6`.
  - O valor de `$f6` é então somado à estimativa atual e para formar a nova soma, que é armazenada novamente em `$f6`.
  - A nova estimativa e é calculada dividindo a soma anterior por 2.0 (armazenado em `$f4`), atualizando assim o valor de e.
- Após o cálculo da nova estimativa, o contador de iterações é decrementado em 1.
- O loop se repete até que o contador de iterações atinja zero.
- Ao final do loop, o valor final da estimativa de raiz quadrada é armazenado na variável ‘raiz‘ para ser utilizado posteriormente.

O processo detalhado de execução do programa e a análise dos registradores demonstram que o cálculo da raiz quadrada foi implementado com sucesso. A tabela dos valores dos registradores ilustra o fluxo de dados e a correta atualização das variáveis envolvidas. Cada iteração refina a estimativa, garantindo que a função converja para a raiz quadrada correta do número x.

1.4. Convergência dos Resultados

1.4.1. Cálculo da Convergência

1. **Escolha de valores de x**:
    - Utilizamos os seguintes valores de x: 5, 25, 50 , para observar a convergência do método iterativo de Newton.
2. **Comparação dos resultados**:
    - Calculamos a raiz quadrada desses valores utilizando o método iterativo de Newton com diferentes números de iterações (5, 10, 15, 20 iterações).
    - Comparamos esses resultados com o valor exato da função de raiz quadrada utilizando a instrução `sqrt.d`.

1.4.2. Exemplo de Comparação

A Tabela 2 expõe os resultados obtidos com as iterações e diferentes valores de x para o cálculo da raiz quadrada utilizando o método iterativo de Newton.

x	n **Iterações**	**Valor Calculado**	**Valor Exato**	**Erro Absoluto**
5	5	2,238095238095238	2,236067977499789	0,002027260595449
5	10	2,23606797749979	2,236067977499789	0,000000000000001
5	15	2,23606797749979	2,236067977499789	0,000000000000001
5	20	2,23606797749979	2,236067977499789	0,000000000000001
25	5	5,000023178253949	5,000000000000000	0,000023178253949
25	10	5,000000000000000	5,000000000000000	0,000000000000000
25	15	5,000000000000000	5,000000000000000	0,000000000000000
25	20	5,000000000000000	5,000000000000000	0,000000000000000
50	5	7,072628275743689	7,071067811865475	0,0015604638782140
50	10	7,071067811865475	7,071067811865475	0,0000000000000000
50	15	7,071067811865475	7,071067811865475	0,0000000000000000
50	20	7,071067811865475	7,071067811865475	0,0000000000000000

Table 2. Comparação dos resultados utilizando o método iterativo de Newton com diferentes números de iterações para vários valores de x.

Os valores calculados foram arredondados de acordo com a quantidade de números significativos informados pelo usuário, garantindo a precisão adequada para a comparação. Esta tabela fornece uma visão abrangente da convergência do método iterativo de Newton para diferentes valores de x e diferentes números de iterações, demonstrando a precisão crescente à medida que aumentamos o número de iterações. Os resultados mostram que já nas primeiras iterações, a precisão alcançada é bastante alta, evidenciando a eficácia do método.

2. Projeto 2: Aproximação do Seno via Série de Taylor em Assembly MIPS

2.1. Objetivo

- **Implementação da Série de Taylor:** Aproximar a função seno utilizando os primeiros 20 termos da série de Taylor.
- **Precisão dos Cálculos:** Garantir a precisão dos cálculos ao implementar a série, utilizando registradores de ponto flutuante de precisão dupla.
- **Manipulação de Ponto Flutuante:** Utilizar instruções de carregamento, armazenamento e operações aritméticas de ponto flutuante em Assembly MIPS.
- **Verificação de Convergência:** Avaliar a convergência dos resultados obtidos pelo procedimento, verificando se aproximam-se do valor esperado.

2.2. Implementação

2.2.1. Armazenamento dos Dados

O código define uma estrutura principal para armazenar os dados:

- `x`: Variável de precisão dupla que armazena o valor de entrada fornecido pelo usuário.
- `resultado`: Variável de precisão dupla que armazena o valor calculado do seno após a série de Taylor ser aplicada.
- `zero_point_zero`: Constante de precisão dupla usada para inicializar valores a 0.0.
- `one_point_zero`: Constante de precisão dupla usada para inicializar valores a 1.0.

2.2.2. Área do Programa

A seção de texto do código (`.text`) é responsável pela execução do programa e inclui as seguintes etapas:

1. Inicializações de Endereços:
    - `la $a0, prompt`: Carregar o endereço da mensagem de entrada.
    - `li $v0, 4`: Código de serviço para impressão de string.
    - `li $v0, 7`: Código de serviço para leitura de ponto flutuante.
2. Leitura do Valor de x:
    - `syscall`: Imprimir a mensagem solicitando o valor de x.
    - `syscall`: Ler o valor do usuário e armazenar em `$f0`.
3. Chamada da Função Seno:
    - `jal seno`: Chamar a função `seno` para calcular o seno de x.
4. Armazenamento e Exibição do Resultado:
    - `la $a0, resultado_msg`: Carregar o endereço da mensagem de resultado.
    - `li $v0, 4`: Código de serviço para impressão de string.
    - `syscall`: Imprimir a mensagem de resultado.
    - `li $v0, 3`: Código de serviço para impressão de ponto flutuante.
    - `syscall`: Mostrar o valor calculado de $seno(x)$ na tela.
5. Finalização do Programa:
    - `li $v0, 10`: Código de serviço para encerrar o programa.
    - `syscall`: Chamar o serviço do sistema para encerrar.

2.3. Execução do Programa

A execução do programa foi de acordo com o esperado, sem encontrar problemas significativos. Durante o processo, foram observadas várias características e dados nos registradores, demonstrando o funcionamento correto do código. A seguir, analisaremos tecnicamente como cada registrador é utilizado para calcular o seno de 57 utilizando a série de Taylor.

Registrador	Nome	Valor (Hexadecimal)
$f0	Valor de x	0x4048F5C3 (57.0)
$f2	Sinal	0xBFF00000 (-1.0)
$f4	Fatorial	0x3FF00000 (1.0)
$f6	Termo Atual	0x00000000
$f8	Potência	0x00000000
$f12	Seno	0x00000000
$a0	Parâmetro	0x00000000
$t0	Contador de Termos	0x00000000
$t1	Limite de Termos	0x00000014 (20)
$t2	Fatorial	0x00000000

Table 3. Valores dos registradores durante a execução do cálculo do seno.

De acordo com a Tabela 3, podemos resumir o processo de cálculo do seno utilizando a série de Taylor da seguinte forma:

- Inicializa-se o registrador `$f12` com 0.0, que será utilizado para armazenar o resultado do cálculo do seno.
- Carrega-se o registrador `$f2` com o valor -1.0, que será usado para alternar os sinais nos termos da série de Taylor.
- Para cada iteração do loop, o termo atual da série é calculado, dividindo o valor de $(-1)^n$ pelo fatorial $(2n + 1)!$, com o resultado armazenado em `$f4`.
- O valor de `x` (por exemplo, 57.0) é elevado à potência $(2n + 1)$, com o resultado guardado em `$f8`.
- O fator calculado em `$f4` é multiplicado pelo valor de potência em `$f8` para obter o termo atual da série, armazenado em `$f6`.
- O termo atual é adicionado ao resultado do seno, resultando na atualização de `$f12` com `$f12` = `$f12` + `$f6`.
- O numerador é atualizado ao multiplicar por x, preparando para a próxima iteração.
- Este processo se repete até que o contador de termos atinja 20, conforme definido no registrador `$t1`.
- Durante cada iteração do loop, o contador de termos `$t0` é incrementado, garantindo que a série de Taylor utilize 20 termos para a aproximação do seno.

O processo detalhado de execução do programa e a análise dos registradores mostram que o cálculo do seno utilizando a série de Taylor foi implementado com sucesso. A tabela dos valores dos registradores demonstra o fluxo de dados e a correta atualização das variáveis envolvidas. A cada iteração, o termo atual é calculado e adicionado ao resultado final, garantindo uma aproximação precisa do seno de 57. A utilização correta dos

registradores permite que o programa alcance a precisão desejada, utilizando 20 termos da série de Taylor. Essa implementação em Assembly exemplifica a eficácia e a precisão dos cálculos matemáticos complexos quando realizados de maneira estruturada e detalhada.

2.4. Convergência dos Resultados

Para demonstrar que o resultado obtido com o procedimento está convergindo para o valor esperado, comparamos os valores calculados pelo algoritmo com os valores exatos da função seno para diferentes valores de x. A convergência pode ser observada através da redução do erro absoluto entre o valor calculado e o valor exato conforme mais termos da série de Taylor são adicionados.

2.4.1. Cálculo da Convergência

1. **Escolha de valores de x**:
    - Utilizamos valores de x em graus comuns como 30, 45 e 60 graus, convertidos para radianos.
2. **Comparação dos resultados**:
    - Calculamos o seno desses valores utilizando a série de Taylor com diferentes números de termos (5, 10, 15, 20 termos).
    - Comparamos esses resultados com o valor exato da função seno.

2.4.2. Exemplo de Comparação

A Tabela 4 expõe os resultados obtidos com as iterações e diferentes valores de x para o cálculo do seno utilizando a série de Taylor.

Ângulo	n Termos	Valor Calculado	Valor Exato	Erro Absoluto
30	5	0.5000000000202792	0.500000000000000	0.0000000000202792
30	10	0.4999999999999993	0.500000000000000	0.0000000000000007
30	15	0.4999999999999993	0.500000000000000	0.0000000000000007
30	20	0.4999999999999993	0.500000000000000	0.0000000000000007
45	5	0.7071067829368669	0.707106781186547	0.0000000017503199
45	10	0.7071067811865472	0.707106781186547	0.0000000000000002
45	15	0.7071067811865472	0.707106781186547	0.0000000000000002
45	20	0.7071067811865472	0.707106781186547	0.0000000000000002
60	5	0.8660254450997807	0.866025403784438	0.0000000413153427
60	10	0.8660254037844382	0.866025403784438	0.0000000000000002
60	15	0.8660254037844382	0.866025403784438	0.0000000000000002
60	20	0.8660254037844382	0.866025403784438	0.0000000000000002

Table 4. Comparação dos resultados utilizando a série de Taylor com diferentes números de termos para vários valores de x em radianos

Os valores calculados foram arredondados de acordo com a quantidade de números significativos informados pelo usuário, garantindo a precisão adequada para a

comparação. Esta tabela fornece uma visão abrangente da convergência da série de Taylor para diferentes valores de x e diferentes números de termos. Observa-se que, mesmo com um número relativamente pequeno de termos ($n = 5$), os valores calculados já apresentam uma alta precisão em relação aos valores exatos. À medida que aumentamos o número de termos na série, a precisão continua a aumentar, demonstrando uma convergência rápida e eficiente do método.

3. Considerações Finais

Em ambos os problemas, a principal dificuldade nas implementações foi a cautela no uso de variáveis de ponto flutuante, com devido olhar de usar sempre os registradores de números pares do coprocessador para o armazenamento de doubles, e fazendo também as conversões corretas. Analogamente, a implementação destes métodos ajudou a elucidar o funcionamento dos métodos matemáticos utilizados, ao passo que os simplifica em instruções sequenciais. Além disso, o cálculo da convergência evidencia a eficácia do uso de valores de iterações cada vez maiores no aperfeiçoamento da precisão dos dois resultados. Conclui-se, portanto, que a implementação destes programas foi fundamental no aprendizado do uso de procedimentos, float e instruções relacionadas, melhorando o entendimento do conteúdo visto em sala de aula.

4. Repositório da Atividade

Confira os códigos implementados acessando o $QRCode$ abaixo ou pelo link:

`https://github.com/buzziologia/UFSC/tree/main/OrganizacaoDeComputadores/Laboratorio03`

Figure 1. Repositório Github