

# Métodos Numéricos

Sistema de Partículas

Carol, Daniel, Mariana e Heitor

# Introdução

# Objetivos

- Aplicar os conhecimentos obtidos na sala de aula sobre resolução de equações para resolver um problema
  - No caso, o deslocamento de partículas em um sistema

# Ferramentas Utilizadas

- C++11
- Sublime Text
- Github



# Atividade no Github



Jul Sep

Oct

31 26

28 29

2

3

7

19

24

25

26

trabalho 1

# Metodologia

# Framework de Funções

- Conjunto de classes que herdam da interface “FunçãoReal”
- Implementação a noção de Funções Elementares
  - Conjunto de funções indutivo
- Ajudou muito na implementação dos métodos

## *FunçãoReal*

double eval(x)

double evalDerivada(x)

FunçãoReal derivada()

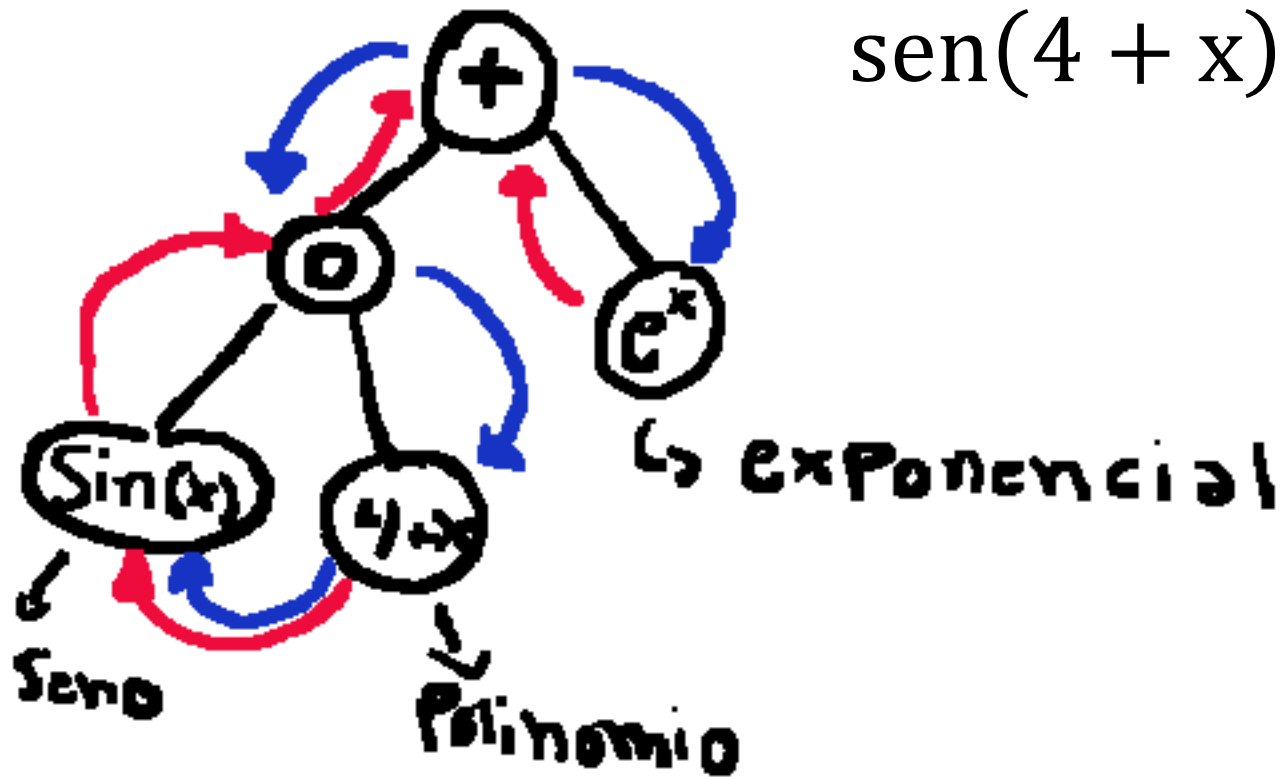
double integral(a,b)

# Funções Representáveis

- É o conjunto das funções que podem ser construídas por uma quantidade finita de composições e combinações das quatro operações elementares
- Com as bases sendo função identidade, exponencial, polinômios nos números reais de qualquer grau e qualquer função definida no C++ de double para double
- Sabendo a derivada e os valores das funções bases, podemos saber as mesmas coisas sobre as compostas usando regras de derivação e as operações usuais



# Como funciona a avaliação



# Uso pelo usuário

```
auto f = compose(newFun(FuncaoExistente(std::sin,"seno"),
                        newFun(Polinomio({4,1})) + newFun(Exponencial()));

double a = f->eval(4);
double a = f->evalDerivada(4);
FuncaoRealP g = f->derivada();
```

# Detalhes de Implementação

- Para funções definidas no C++ temos que calcular a derivada numericamente
  - Afinal, não temos nenhum jeito de saber a forma de função
- Integração teve que ser numérica porque integrais de funções elementares nem sempre são elementares.
  - $e^{-x^2}$ ,  $\frac{\sin(x)}{x}$  e  $x^x$
- Muitas vezes a derivada analítica é feita por falta de *constant folding* e outras simplificações
- É possível fazer os métodos numéricos serem aplicadas a funções dadas pelo usuário usando um *parser* bem simples
  - “ $x+3*5$ ”
  - `newFun(Identidade()) + Constante(3)* Constante(5)`

# Como achar um chute inicial

- Foram criados dois métodos
  - Por ser um semi-algoritmo temos um limite de tempo de 2 segundos
- Determinista
  - Uma busca linear entre um intervalo  $[-a, a]$  tal que cada passo é dado por um  $\epsilon$  arbitrário, estamos procurando valores positivos e negativos em  $f$ 
    - No caso  $\epsilon = 1$  e os limites do intervalo vão aumentando exponencialmente se não for encontrado.
- Aleatório
  - Vários valores aleatórios de um intervalo  $[-a, a]$  vão sendo amostrados até que ache um valor positivo e negativo em  $f$

# Comparações

| Função               | Raiz           | Determinista                 | Aleatório                        | Razão do Tempo |
|----------------------|----------------|------------------------------|----------------------------------|----------------|
| $x^2 - 3$            | $\pm\sqrt{3}$  | 1<br>9,999                   | $-922,337 * 10^{16}$<br>0.000000 | 0.47           |
| $e^x$                | Nenhuma        | -10,000<br>-10,000           | -55,272.870326<br>-55,272.870326 | 0.26           |
| $x - 30,000,000,000$ | 30,000,000,000 | Não Encontrado               | -9,416,739,371<br>72,118,240,017 | 36,371         |
| $x - 0.000,000,01$   | 0.000,000,01   | 0<br>9,999                   | -65,649.619553<br>80,842.798726  | 30.40          |
| $\text{sen}(x)$      | $n * \pi$      | 9,996.000000<br>9,999.000000 | -73239.612944<br>-70376.461651   | 58             |

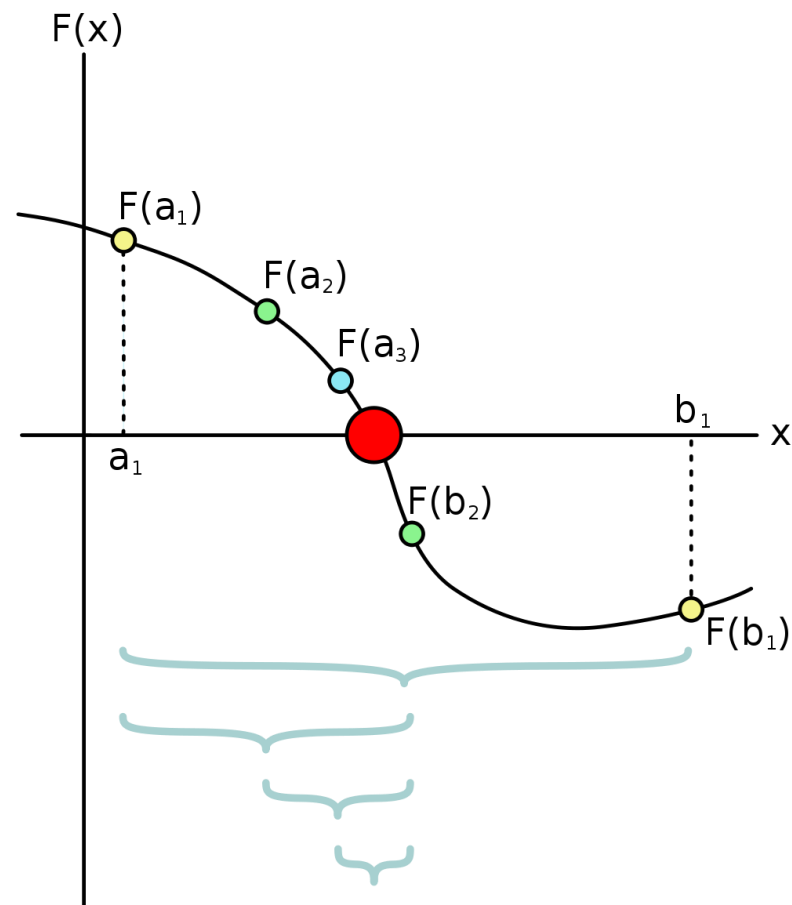
# Métodos Numéricos

- Bisseccção
  - Para ajudar a diminuir o intervalo
- Ponto Fixo
- Newton
  - Usando o Ponto Fixo
- Newton modificado
  - Também usa Ponto Fixo

```
typedef std::tuple<double,double> intervalo;  
  
tnw::intervalo bissec(tnw::intervalo a_b, FuncaoRealP f, double epsilon);  
double pontoFixo(double incial, FuncaoRealP phi, double epsilon);  
double newton(double incial, FuncaoRealP f, double epsilon);
```

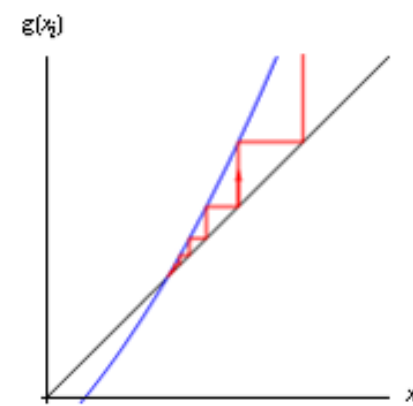
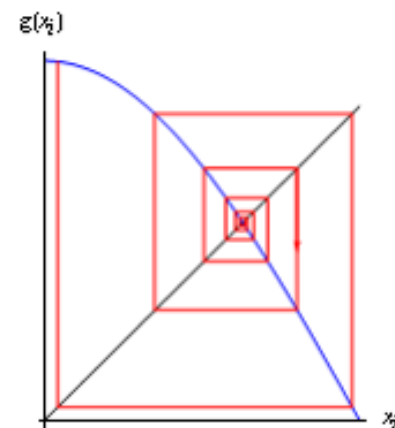
# Bisseccção

- Recebe um intervalo  $(a,b)$ , uma função e um  $\epsilon$
- Até que a diferença entre  $a$  e  $b$  seja menor que o  $\epsilon$ , calcula o ponto médio do intervalo e atualiza o valor de  $a$  ou  $b$  baseado no sinal da função.
- Ao final, retorna o intervalo  $(a,b)$  obtido



# Ponto Fixo

- Recebe uma estimativa inicial para a raiz, uma função de iteração  $f$  e o  $\epsilon$
- Calcula os valores  $x_{n+1} = f(x_n)$  até que a diferença entre  $x_{n+1}$  e  $x_n$  seja menor que o  $\epsilon$ .
- Ao final, retorna  $x_{n+1}$





# Newton e Newton modificado

- Ambas utilizam o método do Ponto Fixo com a função de iteração apropriada

$$x_{n+1} = x_n - \frac{f'(x_n)}{f(x_n)}$$

```
double newton(double incial, FuncaoRealP f, double epsilon) {  
    return pontoFixo(incial,  
        newFun(identidade())-(f/f->derivada()),  
        epsilon);  
}
```

# Interface para o usuário

O usuário entra os dados necessários e é dada uma tabela com as respostas

É necessário uma largura de mais de 80 caracteres para ver o quadro-resposta

```
Calculadora de Raízes

+-----+
| As funções que terão as raízes calculadas serão da forma p0*(e^d) - 4d². |
| Insira os valores de p0 seguidos de enter. Para prosseguir, digite 0. |
+-----+

0.5
1.6
2
0.8
1.2
0

+-----+
| Insira agora uma precisão (épsilon). |
| Menor é melhor. |
+-----+

0.000001

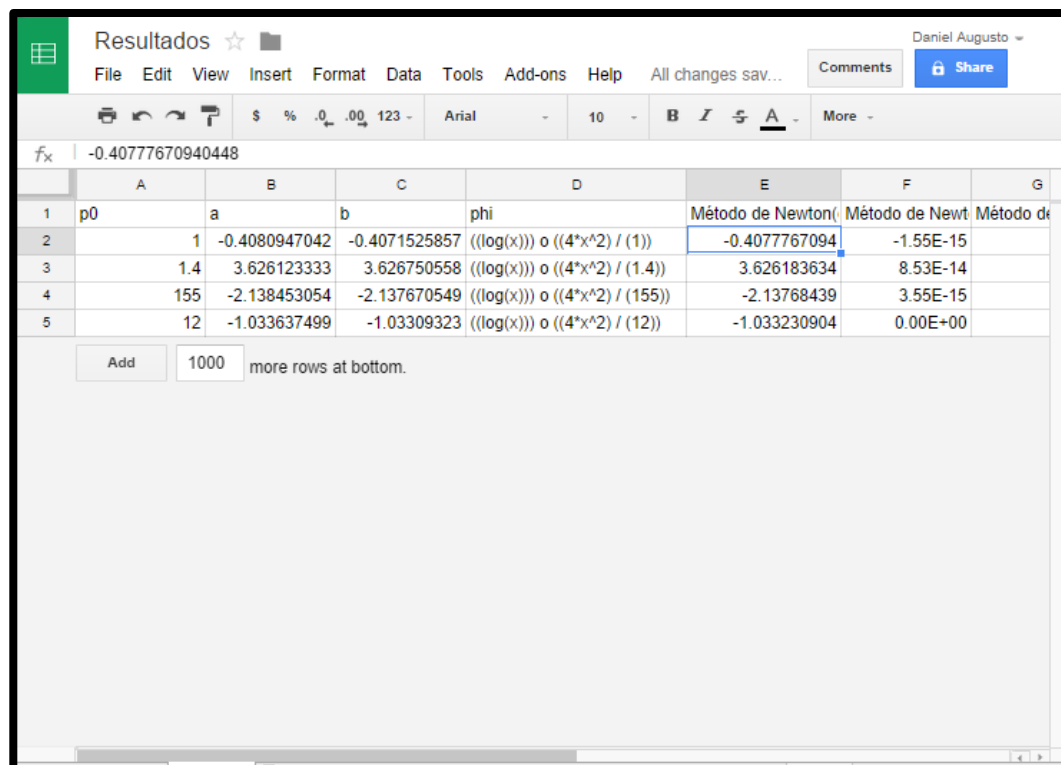
+-----+
| p0 | Ponto Fixo | Newton-Raphson | Newton Modificado |
|    | d          | d              | d                  |
+-----+-----+-----+-----+
| 0.5000 | 5.4826 | 5.4826 | 5.4826 |
| 1.6000 | 3.3105 | -0.4940 | -0.4940 |
| 2.0000 | 2.6179 | 2.6179 | 2.6179 |
| 0.8000 | 4.7079 | 4.7079 | 4.7080 |
| 1.2000 | 3.9528 | 3.9528 | 3.9529 |
+-----+-----+-----+-----+

+-----+
| Digite o nome de um arquivo .csv para guardar. |
| o quadro comparativo completo. |
+-----+

arquivo.csv
```

# Interface para o usuário

Depois é pedido um nome para salvar um arquivo .csv com os dados mais específicos de cada método e a quantidade de iterações para comparação



|   | A   | B             | C             | D                                     | E                | F                | G                |
|---|-----|---------------|---------------|---------------------------------------|------------------|------------------|------------------|
| 1 | p0  | a             | b             | phi                                   | Método de Newton | Método de Newton | Método de Newton |
| 2 | 1   | -0.4080947042 | -0.4071525857 | $((\log(x))) \circ ((4*x^2) / (1))$   | -0.4077767094    | -1.55E-15        |                  |
| 3 | 1.4 | 3.626123333   | 3.626750558   | $((\log(x))) \circ ((4*x^2) / (1.4))$ | 3.626183634      | 8.53E-14         |                  |
| 4 | 155 | -2.138453054  | -2.137670549  | $((\log(x))) \circ ((4*x^2) / (155))$ | -2.13768439      | 3.55E-15         |                  |
| 5 | 12  | -1.033637499  | -1.03309323   | $((\log(x))) \circ ((4*x^2) / (12))$  | -1.033230904     | 0.00E+00         |                  |

# Conclusão

- O trabalho foi uma ótima maneira de fixar os conceitos aprendidos em sala de aula
- A maior dificuldade foi com programação em C++