TRABALHO DE MÉTODOS NUMÉRICOS

RAÍZES DE EQUAÇÕES

INTRODUÇÃO

- Em vários problemas é possível achar sua solução resolvendo a equação $f(x) = 0$
- Ex: Em engenharia ambiental, a seguinte equação pode ser usada para calcular o nível local de descarga de poluentes: $c(x) = 10 - 20(e^{-0.2x} - e^{-0.75x})$.

RESOLVENDO A EQUAÇÃO

- Dada uma aproximação inicial para a raiz, refinamos essa aproximação atravez de métodos interativos.
- Obter um intervalo que contém a raiz da equação
- Melhorar sequentimente as aproximações dada até obter a aproximação da raiz dentro da precisão prefixiada

LOCALIZANDO A RAIZ

- Regra de sinal de Descartes para determinar número de zeros reais com coeficientes reais
- Teoremas da localização no Círculo

MÉTODOS INTERATIVOS

- Método de Newton para polinômios
- Método de Newton para multiplicidade
- Método da secante para multiplicidade
- Há outros métodos...

MÉTODO DE NEWTON PARA POLINÔMIOS

- Polinômio da forma $p_n(x) = a_0+a_1x+a_2x^2+...+a_{n-1}x^{n-1}+a_nx^n$, $a_n \neq 0$
- Teoremas para auxiliar na localização das raízes
- Método Newton para polinômios

MÉTODO DE NEWTON PARA POLINÔMIOS - MÉTODO

- Entrada: n, coeficientes a_0,a_1, ..., $a_{n,}$ $\varepsilon_{1,}$ iterMax

EXPLORER

OPEN EDITORS

METODOS_NUMERICOS V2

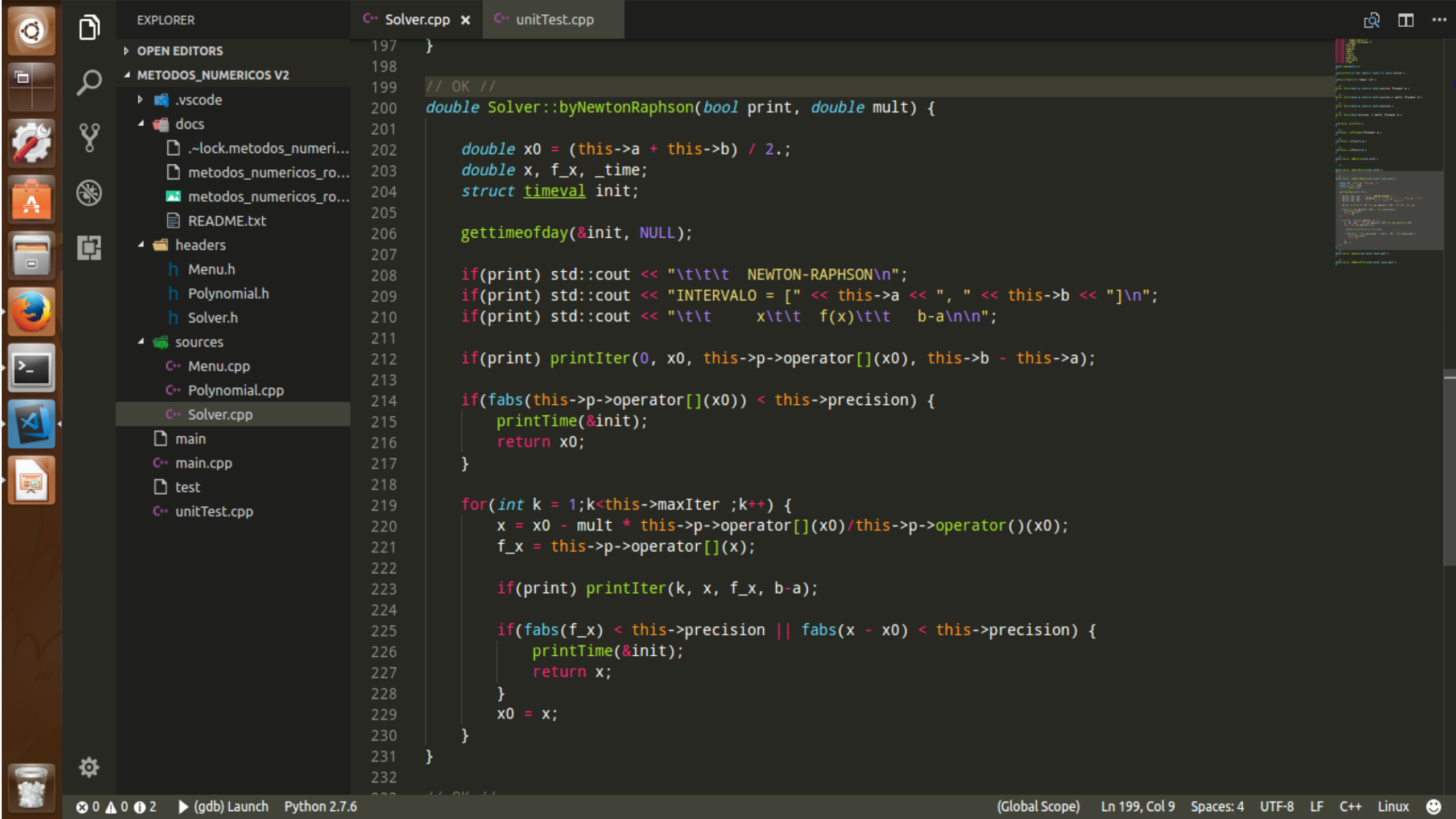
- .vscode
- docs
  - .~lock.metodos_numeri...
  - metodos_numericos_ro...
  - metodos_numericos_ro...
  - README.txt
- headers
  - Menu.h
  - Polynomial.h
  - Solver.h
- sources
  - Menu.cpp
  - Polynomial.cpp
  - Solver.cpp
- main
- main.cpp
- test
- unitTest.cpp

Solver.cpp | unitTest.cpp

```
312     double x = (this->a + this->b) / 2.;
313     double dx = x;
314     double b, c, f_x, _time;
315     struct timeval init, end;
316
317     gettimeofday(&init, NULL);
318
319     if(print) std::cout << "\t\t\t  NEWTON-PARA-POLINÔMIOS\n";
320     if(print) std::cout << "INTERVALO = [" << this->a << ", " << this->b << "]\n";
321     if(print) std::cout << "\t\t     x\t\t  f(x)\t\t   b-a\n\n";
322
323     if(print) printIter(0, x, this->p->operator[](x), this->b - this->a);
324
325     if(fabs(this->p->operator[](x)) < this->precision) {…
328     }
329
330     for(int k = 1; k < this->maxIter; k++) {
331         b = this->p->getCoeficient(this->p->getDegree());
332         c = b;
333
334         for(int i = this->p->getDegree()-1; i > 0; i--) {
335             b = this->p->getCoeficient(i) + b*x;
336             c = b + c*x;
337         }
338
339         b = this->p->getCoeficient(0) + b*x;
340
341         if(fabs(b) < this->precision) {…
344         }
345
346         dx = b/c;
347         x = x - mult * dx;
348         f_x = this->p->operator[](x);
349
350         if(fabs(dx) < this->precision) {…
353         }
```

MÉTODO DE NEWTON PARA MULTIPLICIDADE

- Polinômio da forma $p_n(x) = a_0+a_1x+a_2x^2+...+a_{n-1}x^{n-1}+a_nx^n$, $a_n \neq 0$
- Termo de multiplicidade $p.\ (x_{k+1} = x_k - (p*f(x)/f'(x))$
- Teoremas para auxiliar na localização das raízes
- Método Newton para polinômios


EXPLORER
Solver.cpp
unitTest.cpp
OPEN EDITORS
METODOS_NUMERICOS V2
.vscode
docs
.~lock.metodos_numeri...
metodos_numericos_ro...
metodos_numericos_ro...
README.txt
headers
Menu.h
Polynomial.h
Solver.h
sources
Menu.cpp
Polynomial.cpp
Solver.cpp
main
main.cpp
test
unitTest.cpp
197 }
198
199 // OK //
200 double Solver::byNewtonRaphson(bool print, double mult) {
201
202 double x0 = (this->a + this->b) / 2.;
203 double x, f_x, _time;
204 struct timeval init;
205
206 gettimeofday(&init, NULL);
207
208 if(print) std::cout << "\t\t\t NEWTON-RAPHSON\n";
209 if(print) std::cout << "INTERVALO = [" << this->a << ", " << this->b << "]\n";
210 if(print) std::cout << "\t\t x\t\t f(x)\t\t b-a\n\n";
211
212 if(print) printIter(0, x0, this->p->operator[](x0), this->b - this->a);
213
214 if(fabs(this->p->operator[](x0)) < this->precision) {
215 printTime(&init);
216 return x0;
217 }
218
219 for(int k = 1;k<this->maxIter ;k++) {
220 x = x0 - mult * this->p->operator[](x0)/this->p->operator()(x0);
221 f_x = this->p->operator[](x);
222
223 if(print) printIter(k, x, f_x, b-a);
224
225 if(fabs(f_x) < this->precision || fabs(x - x0) < this->precision) {
226 printTime(&init);
227 return x;
228 }
229 x0 = x;
230 }
231 }
232
0 0 2
(gdb) Launch
Python 2.7.6
(Global Scope)
Ln 199, Col 9
Spaces: 4
UTF-8
LF
C++
Linux

MÉTODO DA SECANTE PARA MULTIPLICIDADE

- Polinômio da forma $p_n(x) = a_0+a_1x+a_2x^2+...+a_{n-1}x^{n-1}+a_nx^n$, $a_n \neq 0$
- Termo de multiplicidade p.
- $x_{k+2} = x_{k+1} - p*f(x_{k+1})(x_{k+1} - x_k)/(f(x_{k+1})-f(x_k))$
- Teoremas para auxiliar na localização das raízes

MÉTODO DA SECANTE PARA MULTIPLICIDADE - MÉTODO

- Entrada: n, coeficientes a0,a1, ..., an, ε1, p, IterMax

(98%) 20:47

EXPLORER

OPEN EDITORS

METODOS_NUMERICOS V2

- .vscode
- docs
  - .~lock.metodos_numeri...
  - metodos_numericos_ro...
  - metodos_numericos_ro...
  - README.txt
- headers
  - Menu.h
  - Polynomial.h
  - Solver.h
- sources
  - Menu.cpp
  - Polynomial.cpp
  - Solver.cpp
- main
- main.cpp
- test
- unitTest.cpp

Solver.cpp | unitTest.cpp

```
235     double x0 = this->a;
236     double x1 = this->b;
237     double f_x0 = this->p->operator[](x0);
238     double f_x1 = this->p->operator[](x1);
239     double x, f_x, _time;
240     struct timeval init;
241
242     gettimeofday(&init, NULL);
243
244     if(print) std::cout << "\t\t\t  SECANTE\n";
245     if(print) std::cout << "INTERVALO = [" << this->a << ", " << this->b << "]\n";
246     if(print) std::cout << "\t\t     x\t\t  f(x)\t\t   b-a\n\n";
247
248     if(print) printIter(0, x0, this->p->operator[](x0), this->b - this->a);
249
250     if(fabs(f_x0) < this->precision) {…
253     }
254
255     if(fabs(f_x1) < this->precision || fabs(x1 - x0) < this->precision) {…
258     }
259
260     for(int k = 1;k<this->maxIter ;k++) {
261         f_x0 = this->p->operator[](x0);
262         f_x1 = this->p->operator[](x1);
263         x = x1 - mult * (f_x1/(f_x1 - f_x0)) * (x1 - x0);
264         f_x = this->p->operator[](x);
265
266         if(print) printIter(k, x, f_x, this->b - this->a);
267
268         if(fabs(f_x) < this->precision || fabs(x - x1) < this->precision) {
269             printTime(&init);
270             return x;
271         }
272         x0 = x1;
273         x1 = x;
274     }
```

PROBLEMA: TEMA 4

- Uma determinada reação química produz uma quantidade c de CO2 medida em ppm (parte por milhão) dada pela equação polinomial $f(c) = a_4c_4 + a_3c_3 + a_2c_2 + a_1c_1 + a_0$.
- Foi desenvolvido um sistema para calcular a quantidade de c de CO_2 de uma determinada reação química

ANÁLISE DOS MÉTODOS

Método Newton para Polinômios

- Polinômio $x^4 - 5x^3 + 6x^2 + 4x - 8$
- Tempo: 0,21 segundos
- Intervalo: [0, 9]
- Multiplicidade: 1
- Precisão: 0.001

Iteração	x	f(x)
0	4,50000	85,937500
1	3,77632	26,778300
2	3,24952	8,298200
3	2,87819	2,558200
4	2,68835	0,779839
5	2,41077	0,236402
6	2,27913	0,071316

Iteração	x	f(x)
7	2,18866	0,021410
8	2,12699	0,006403
9	2,08522	0,001910
10	2,05707	0,000568
11	2,03017	0,000168
12	**2,02550**	**0,000050**

ANÁLISE DOS MÉTODOS

Método Newton Raphson

- Polinômio $x^4 - 5x^3 + 6x^2 + 4x - 8$
- Tempo: 0,18 segundos
- Intervalo: [0, 9]
- Multiplicidade: 1
- Precisão: 0.001

Iteração	x	f(x)
0	4,50000	85,937500
1	3,77632	26,778300
2	3,24952	8,298200
3	2,87819	2,558200
4	2,68835	0,779839
5	2,41077	0,236402
6	2,27913	0,071316

Iteração	x	f(x)
7	2,18866	0,021410
8	2,12699	0,006403
9	2,08522	0,001910
10	2,05707	0,000568
11	2,03017	0,000168
12	**2,02550**	**0,000050**

ANÁLISE DOS MÉTODOS

Método da Secante

- Polinômio $x^4 - 5x^3 + 6x^2 + 4x - 8$
- Tempo: 0,14 segundos
- Intervalo: [0, 9]
- Multiplicade: 1
- Precisão: 0.001

Iteração	x	f(x)
0	0,00000	-8,000000
1	0,02894	-7,913640
2	0,04161	-7,823520
3	1,83589	-0,012530
4	1,83077	-0,01198
5	1,89262	-0,00358
6	1,91581	-0,00174

Iteração	x	f(x)
7	1,93772	-0,000710
8	1,95281	-0,000310
9	1,96454	-0,000130
10	**1,97324**	**-0,000056**

ANÁLISE DOS MÉTODOS

Método Newton para Polinômios

- Polinômio $x^4 - 5x^3 + 6x^2 + 4x - 8$
- Tempo: 0,082 segundos
- Intervalo: [0, 9]
- Multiplicidade: 3
- Precisão: 0.001

Iteração	x	f(x)
0	4,5	85,9375
1	2,32895	0,11948
2	**2,01049**	**0,00000347**

ANÁLISE DOS MÉTODOS

Método Newton Raphson

- Polinômio $x^4 - 5x^3 + 6x^2 + 4x - 8$
- Tempo: 0,061 segundos
- Intervalo: [0, 9]
- Multiplicidade: 3
- Precisão: 0.001

Iteração	x	f(x)
0	4,5	85,9375
1	2,32895	0,11948
2	**2,01049**	**0,00000347**

ANÁLISE DOS MÉTODOS

Método da Secante

- Polinômio $x^4 - 5x^3 + 6x^2 + 4x - 8$
- Tempo: 0,14 segundos
- Intervalo: [0, 9]
- Multiplicade: 3

Iteração	x	f(x)
0	0,00000	-8
1	-17.9372	134224
2	64.9936	1.64964e+07
3	-185.84	1.22506e+09
4	576.931	1.09831e+11
5	-1737.19	9.13359e+12
6	5289.68	7.82179e+14

Iteração	x	f(x)
7	-16040	6.62145e+16
8	48714	5.63079e+18
9	-147860	4.77982e+20
10	448891	4.0603e+22
...	...	...