U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Análise Numérica (M2018)

Relatório do Trabalho Prático 2

Amadeu Marques
Eduardo Santos
Miguel Ramos
Ricardo Ribeiro

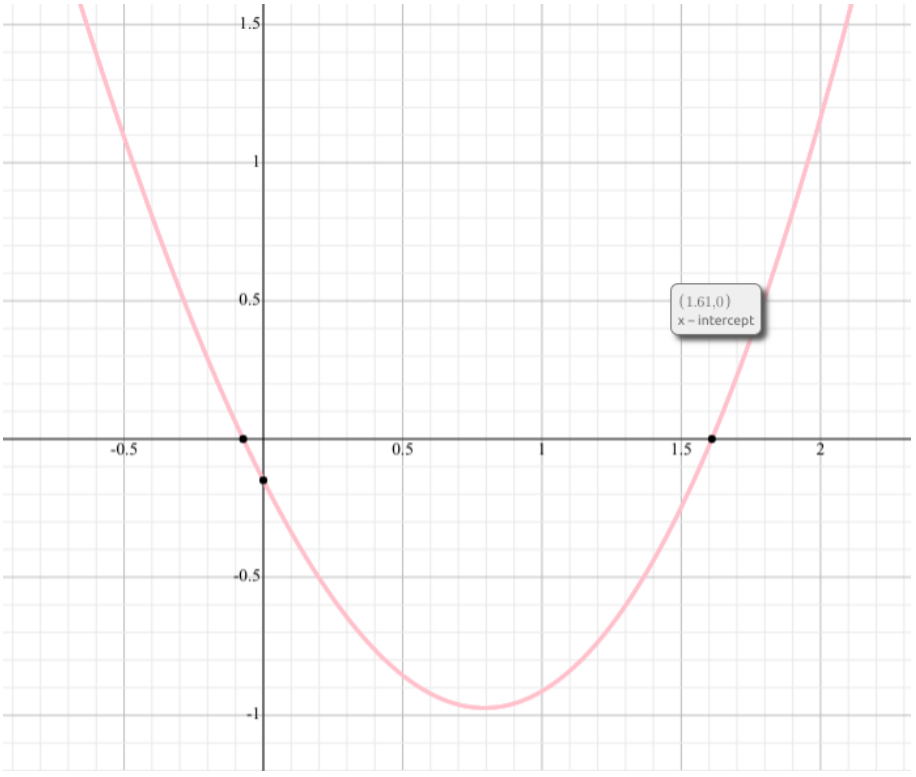
Introdução

Pretendemos neste relatório resolver os problemas propostos no segundo trabalho prático de Análise Numérica (M2018).
À semelhança do primeiro trabalho prático, os programas foram todos escritos na linguagem C, pelas mesmas razões, como a facilidade de uso dentro do grupo e a boa eficiência do C. Para suporte visual e de apoio à prova, utilizámos ainda as ferramentas de representação gráfica dos websites Symbolab Graph e Wolfram Alpha.
Incluímos ainda as bibliotecas stdio.h e math.h para a impressão de resultados e utilização recorrente da função `pow`. Todas as outras funções são da nossa autoria.

Exercício 1

Alínea a)

Entrando a função dada, $x^2 - x - sen(x + 0.15)$, no Symbolab, podemos ver que a função tem dois zeros, o maior dos dois intercetando o eixo das abcissas aproximadamente em 1.61. Sabendo isto, escolhemos o intervalo [1.55,1.65] de amplitude 10^{-1} onde o zero está contido.



Alínea b)

Método das bisseções sucessivas:

i.

O método das bisseções sucessivas tem duas condições: F(x) contínua em [a,b], com F(a) x F(b) < 0, ou seja , F(a) e F(b) têm sinais opostos. Se estas condições forem válidas, o algoritmo consiste na divisão do intervalo no ponto médio m, $m = \frac{a+b}{2}$ e, depois, verificar nos intervalos [a,m] e [m,b] em qual destes subintervalos existe uma raíz, e assim sucessivamente. Deste modo, o erro absoluto é reduzido a metade a cada iteração.
No nosso exercício, a função escolhida é $F(x) = x^2 - x - sen(x + 0.15)$ com intervalo [1.55,1.65]. Para conseguir utilizar o algoritmo, F(x) tem que ser contínua no intervalo. A função F(x) é composta por uma divisão de uma função trigonométrica por uma função polinomial, ambas contínuas em $\mathbb{R}$. Assim, o único ponto onde não há continuidade é em x=0. Como o intervalo é [1.55,1.65], podemos afirmar que a função é contínua nesse intervalo.
Como F(1.55) x F(1.65) < 0 podemos assim afirmar que existe pelo menos uma raiz nesse intervalo. Com as duas condições confirmadas, podemos aplicar o algoritmo que foi utilizado no programa abaixo.

ii.

```
#include <stdio.h>
#include <math.h>

double absoluto(double x){
  if(x >= 0.0) return x;
  else return -x;
}

double func(double x){
   return pow(x,2) - x - sin(x + 0.15);
}

void bissec_suc(double a, double b){
   int i,precisao = 8;
   double m,f_m,f_a, epsilon = 5 * pow(0.1,precisao), erro = absoluto(b - a);
   for(i=0; erro > epsilon; i++){
      m = (a + b)/2.0;
      f_m = func(m);
      f_a = func(a);
      if(f_m == 0.0) erro = 0.0;
      else if((f_m * f_a) < 0.0) b = m;
      else a = m;
      erro /= 2;
   }
```

```
   printf("Metodo de bissecao sucessiva: n = %2d, v = %.*f, erro = %.*f\n", i,
precisao+1, m, precisao+1, erro);
}

int main(){
    bissec_suc(1.55,1.65);
}
```

Este programa funciona com base no algoritmo presente nos slides teóricos da cadeira, e é composto por três funções.

A primeira, `absoluto`, simplesmente retorna o valor absoluto de qualquer número.

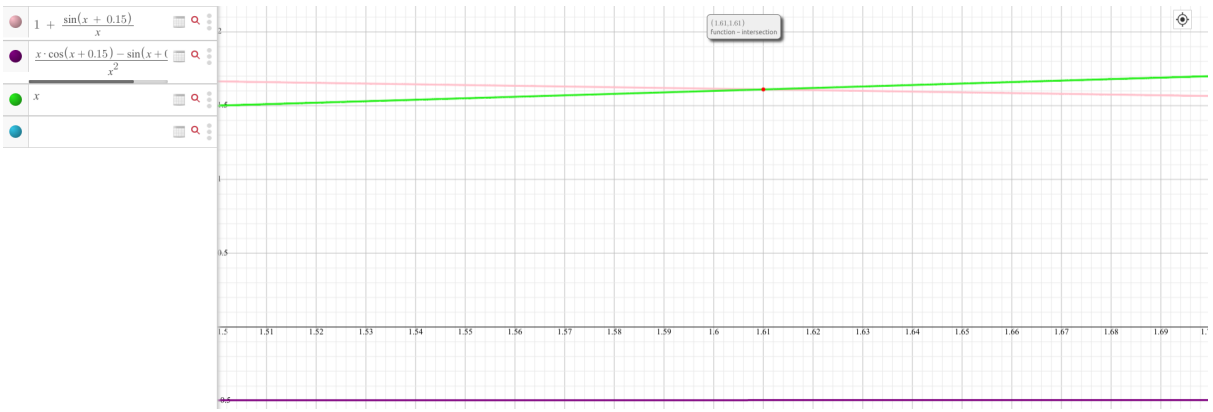
A segunda, `func`, retorna o valor da função para qualquer x que receba.

Por último, temos o algoritmo em si, `bissec_suc`, que recebe um intervalo onde a raíz que queremos encontrar se encontra. Neste programa, temos um valor *a* e um valor *b*, para os valores mínimos e máximos do intervalo respetivamente. Logo começa um ciclo, onde temos um valor *m*, que é o valor da média entre *a* e *b*. Guardamos nas variáveis *f_m* e *f_a* os valores da função sobre *m* e *a*, respetivamente. Se *f_m* for igual a 0, então o erro da iteração fica a 0 também. Senão, se o produto de *f_m* e *f_a* for menor que 0, então o valor de *m* fica guardado em *b*. Caso contrário, fica o valor de *m* guardado em *a*. Finalmente, o erro da iteração passa para metade do seu valor atual. O ciclo termina quando o erro for igual ou menor a épsilon.
Concluído o ciclo, o programa imprime o número de iterações necessárias para o exercício, juntamente com o valor obtido e o erro associado.

Método iterativo simples:

i.

O método iterativo simples tem três condições postas sobre a função iterativa, que terão de ser cumpridas para o algoritmo funcionar. A função interativa que escolhemos, tal que x = f(x), é definida como $F(x) = 1 + \frac{sen(x+0.15)}{x}$. Para suporte à prova, temos o gráfico da função (linha cor-de-rosa) e a sua derivada $\frac{xcos(x+0.15) - sen(x+0.15)}{x^2}$ (linha roxa) na imagem seguinte, para o intervalo [1.55,1.65]. Para mostrar que a função de iteração é correta, fizemos ainda uma interseção com a reta da função y = x (linha verde) para mostrar que se encontra no zero da função original.



A primeira condição dita que a função tem de ser contínua no intervalo [1.55,1.65]. Como a função é composta por uma função seno, contínua em $\mathbb{R}$, sobre x, será contínua em $\mathbb{R}$ exceto 0, onde os limites pela esquerda e pela direita apontam para -/+ infinito respetivamente. Então, podemos afirmar que F(x) é contínua em [1.55,1.65].

Como a derivada de F(x) no intervalo é negativa (como podemos ver no gráfico), temos que $F(x0) \geq F(x1)$, para todo o x pertencente a [1.55,1.65] e $x0 \leq x1$. Assim, temos que o valor máximo da função no intervalo será F(1.55), e o valor mínimo F(1.65), e toma valores em [1.59021... ,1.63978...] , sendo que então a segunda condição se cumpre, que dita que f(x) pertence a [a,b] para todo o x pertencente [a,b].

A terceira condição diz ainda que o módulo da derivada de f(x) terá de ser menor ou igual a L, que por sua vez é menor que 1, para todo o x pertencente ao intervalo]a,b[. Como podemos ver no gráfico, no intervalo pretendido, todos os valores de F'(x) são superiores a -1 e inferiores a 0, logo o seu módulo está compreendido entre]0,1[. Assim se cumpre a condição.

Desta forma podemos afirmar que esta função iterativa é válida para este exercício, e será demonstrado em baixo o programa onde ela é aplicada e os resultados obtidos.

ii.

```
#include <stdio.h>
#include <math.h>

double absoluto(double x){
  if(x >= 0.0) return x;
  else return -x;
}

double func_iterativa(double x){
  return 1 + (sin(x + 0.15) / x);
  //Função obtida por manipulação algébrica da função original
}

void iter_simples(double x0, int nmax){
  int i = 1, precisao = 8;
  double x1 = func_iterativa(x0);
  double erroiter = absoluto(x1 - x0), epsilon = 5 * pow(0.1,precisao);
```

```
  while( i <= nmax && erroiter > epsilon){
    x0 = x1;
    x1 = func_iterativa(x0);
    erroiter = absoluto(x1 - x0);
    i++;
  }

  if(i > nmax) printf("Metodo iterativo simples: não foi possível ao fim de %d
iteracoes encontrar a solucao com o erro pretendido\n", i);
  else printf("Metodo iterativo simples:     n = %2d, v = %.*f, erro = %.*f\n", i,
precisao+1, x1, precisao+1, erroiter);
}

int main(){
    iter_simples(1.55,22);
}
```

Este programa funciona com base no algoritmo presente nos slides teóricos da cadeira, e é composto por três funções.

A primeira, `absoluto`, simplesmente retorna o valor absoluto de qualquer número.

A segunda, `func_iterativa`, recebe um número e retorna o valor da função iterativa em que x é igual ao número recebido.

Por fim, o algoritmo da iteração simples, `iter_simples`, vai receber um x0 e um limite máximo de iterações, que através de análise de resultados sabemos que não ultrapassa os 22 para este exercício. Neste algoritmo, começamos por definir um x1 como igual ao resultado da função iterativa por x0. Em seguida, vamos entrar num ciclo que irá guardando o valor de x1 em x0, e igualando x1 ao valor da função iterativa sobre o novo x0. Ainda dentro do ciclo, vamos calcular o erro da iteração, que será o valor absoluto da subtração de x1 por x0. O ciclo termina quando o erro é menor do que valor de épsilon dado ou chegamos ao limite de iterações.

Concluído o ciclo, o programa imprime o número de iterações necessárias para o exercício, juntamente com o valor obtido e o erro associado, ou uma mensagem de erro se o limite de iterações foi ultrapassado.

Método de Newton:

i.

Para a recorrência do método de Newton convergir para a raiz de F(x) são suficientes 5 condições: F,F' e F'' existem e são contínuas em [a,b], isto é simples de concluir pois como a função F(x) é composta por 3 funções(linear,quadrática e trigonométrica), todas elas contínuas em [a,b] então a função F(x) também é contínua no mesmo intervalo.

$F(1.55) \approx -0.1391648104$ e $F(1.65) \approx 0.09865236912$, logo $F(a)F(b) < 0$.

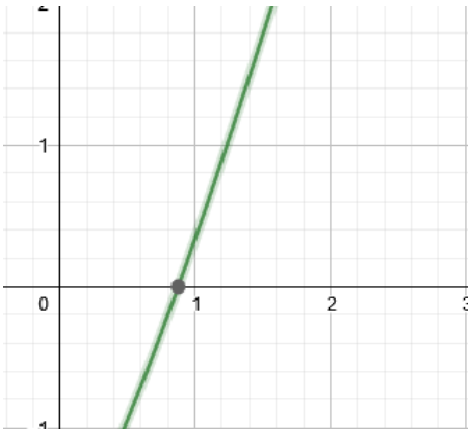


gráfico da função F’(x).

Como podemos observar no gráfico da função F’(x), F’(x) ≠ 0 em [1.55,1.65].

Também pelo gráfico conseguimos ver que F’(x) é constantemente crescente no intervalo [a,b], logo a sua derivada será positiva, então $\forall x \in [a,b]$, F’’(x) > 0.

Por fim, F(x0) $>$ 0 e F”(x0) $>$ 0 pois x0 = 1.65, logo F(x0)F”(x0) > 0.

ii.

```
#include <stdio.h>
#include <math.h>

double absoluto(double x){
  if(x >= 0.0) return x;
  else return -x;
}

double func(double x){
   return pow(x,2) - x - sin(x + 0.15);
}

double df(double x){
    return 2*x - 1 - cos(x - 0.15);
}

void newton(double x0){
    int iter = 1, precisao = 8;
    double x1 = x0 - (func(x0)/df(x0));
    double erroiter = absoluto(x1-x0), epsilon = 5 * pow(0.1,precisao);

    while(erroiter >= epsilon){
        x0 = x1;
        x1 = x0 - func(x0)/df(x0);
        erroiter = absoluto(x1 - x0);
        iter++;
    }
```

```
       printf("Metodo de Newton:             n = %2d, v = %.*f, erro = %.*f\n",
iter, precisao+1, x1, precisao+1, erroiter);
}

int main(){
    newton(1.65);
}
```

Este programa está fundamentado no método de Newton presente nos slides teóricos para a aproximação de uma raíz, e é composto por 4 funções.

A primeira, `absoluto`, simplesmente retorna o valor absoluto de qualquer número.

A segunda, `func`, retorna o valor da função para qualquer x que receba.

A terceira, `df`, retorna o valor da derivada da função original para qualquer x que receba.

Por fim, o algoritmo do método, `newton`, que recebe o valor inicial do intervalo, *x0*. Logo no início, é definido uma variável *x1*, que será igual à diferença entre *x0* e o valor da divisão dos valores de `func(x0)` e `df(x0)`, ou seja, da função sobre a sua derivada. Inicializamos o valor do erro para a primeira iteração como o valor absoluto da diferença entre x1 e x0.
Começa então o ciclo, onde a cada iteração guardamos em *x0* o valor de *x1*, e em *x1* registamos o valor da diferença entre *x0* e o valor da divisão dos valores de `func(x0)` e `df(x0)`. Adicionamos 1 ao contador de iterações, e o ciclo recomeça. O ciclo termina quando o valor do erro é inferior ao valor do épsilon e atingimos a precisão pretendida.

Finalmente, o programa imprime o número de iterações necessárias para o exercício, juntamente com o valor obtido e o erro associado.

Alínea c)

	N	Valor	Erro
Método de bisseções sucessivas	21	1.610022593	0.000000048
Método iterativo simples	22	1.610022544	0.000000036
Método de Newton	8	1.610022561	0.000000040

Como é possível verificar na tabela de resultados acima, o método de Newton terá sido o que teve menos iterações, com apenas 8, para este problema. Os métodos de bisseções sucessivas e iterativo simples resolveram o problema com o número semelhante de iterações, sendo o método de bisseções sucessivas melhor neste caso. De todos os métodos, o método iterativo simples obteve a melhor aproximação com o menor erro.

Exercício 2

Alínea a)

A função F(x) é uma função polinomial logo contínua em [3.5,6.5]. Como F(3.5) = -8.5625 e F(6.5) = 5.3125, então F(3.5)F(6.5) < 0. Desta forma, fica provado pelo teorema de Bolzano-Cauchy que existe pelo menos um zero de F(x) no intervalo [3.5,6.5].

Alínea b)

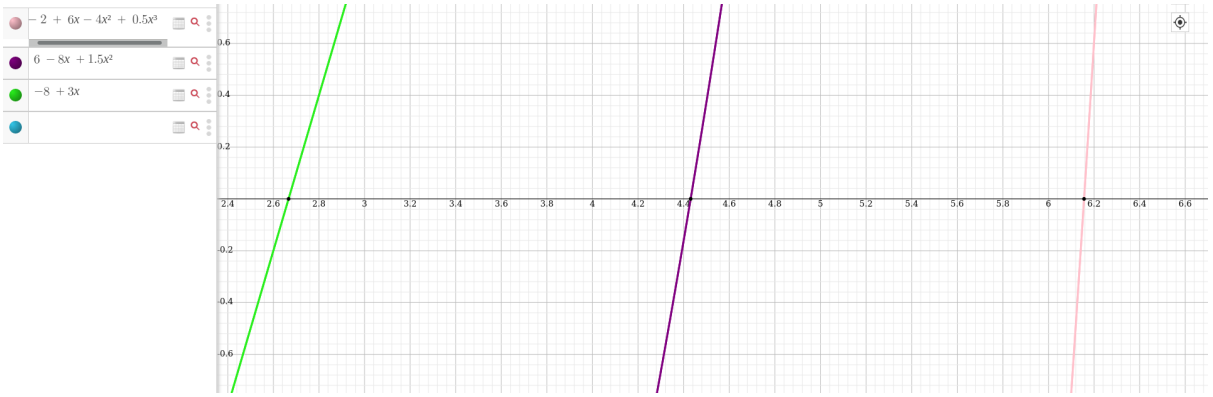
```
#include <stdio.h>
#include <math.h>

double absoluto(double x){
  if(x >= 0.0) return x;
  else return -x;
}

double f(double x){
    return -2 + 6*x - 4*pow(x,2) + 0.5*pow(x,3);
}

double df(double x){
    return 6 - 8*x + 1.5*pow(x,2);
}

void newton(double x0){
    int iter = 1, precisao = 6;
    double x1 = x0 - (f(x0)/df(x0));
    double erroiter = absoluto(x1-x0), epsilon = pow(0.1,precisao);

    while(erroiter >= epsilon){
        x0 = x1;
        x1 = x0 - f(x0)/df(x0);
        erroiter = absoluto(x1 - x0);
        iter++;
    }

       printf("Metodo de Newton: n = %2d, v = %.*f, erro = %.*f\n", iter,
precisao+1, x1, precisao+1, erroiter);
}

int main(){
    newton(3.5);
    newton(6.5);
    newton(4.4);
}
```

Alínea c)

Podemos observar na tabela abaixo os resultados obtidos. A partir destes resultados e observando o gráfico podemos concluir que apenas o valor de x0=6.5 está correto.

	N	Valor	Erro
[3.5,6.5] x0 = 3.5	6	1.3691024	0.0000000
[6.0,6.5] x0 = 6.5	4	6.1563252	0.0000001
[4.4,6.5] x0 = 4.4	15	0.4745724	0.0000001

No intervalo [4.4,6.5] e consequentemente em [3.5,6.5] não podemos assegurar que o método de Newton irá convergir para a raiz de F(x), pois existe uma raiz de F'(x) que pertence a ambos os intervalos. Já no intervalo [6.0,6.5] podemos assegurar a convergência pois satisfaz todas as condições de convergência do método de Newton.



Conclusões

Em termos de programação, não sentimos grandes dificuldades na criação do código, assim como na elaboração do relatório.
Embora, na nossa opinião, todos os programas sejam eficientes, obtivemos uma grande diferença no número de iterações no método de Newton comparando com o método das bisseções sucessivas e o método iterativo simples. Isto pode ter como explicação o facto de o método de Newton ter mais condições necessárias para a execução correta do algoritmo. Ao ter mais restrições, o algoritmo acaba por ser executado menos vezes
Gostámos de trabalhar neste projeto , tal como no primeiro trabalho, e pensamos que correu bastante bem e todos os membros cumpriram bem os seus objetivos para a realização do trabalho.