U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Análise Numérica (M2018)

Relatório do Trabalho Prático 1

Amadeu Marques ______________________

Eduardo Santos ______________________

Miguel Ramos ______________________

Ricardo Ribeiro ______________________

Introdução

Pretendemos neste relatório resolver os problemas propostos no primeiro trabalho prático de Análise Numérica (M2018).
Os programas foram todos escritos na linguagem C, uma linguagem onde o grupo se sente confortável e que permite um bom desempenho devido a ser uma linguagem de baixo nível.

Usufruímos do uso de doubles, que são pontos flutuantes com precisão de 10^{-15}, que achamos perfeito para o contexto deste trabalho.
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

Escrever um programa que permita calcular um valor aproximado de:

$$S = 18 \sum_{k=1}^{\infty} \frac{k!^2}{k^2(2k)!}$$

com erro absoluto inferior a ϵ.

Programa:

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

// verifiquei que k <= 23 e suficiente
double mem_fact[47];

void m_fact_init(){
   mem_fact[0] = 1.0;
   for(int i=1;i<47;i++) mem_fact[i] = 0.0;
}

double m_fact(int n){
   if(mem_fact[n] != 0.0) return mem_fact[n];
   mem_fact[n] = n*m_fact(n-1);
   return mem_fact[n];
}

void serie(){
   int k,precisao = 8;
   double sum = 0.0, termo, epsilon = pow(0.1,precisao);

   for(k=1;precisao <= 15;k++) {
      termo = 18.0 * pow(m_fact(k),2) / (pow(k,2) * m_fact(2*k));
```

```
        if(termo < epsilon){
            printf("N = %d E = 10^-%d\n",k-1,precisao);
            printf("         S = %.*f\n",precisao+1,sum);
            precisao++;
            epsilon *= 0.1;
        }
        sum += termo;
    }
}

int main(){
    m_fact_init();
    serie();
}
```

O programa tem como objetivo calcular um valor aproximado da série acima com erro absoluto inferior a εe imprimir o seu valor ao longo dos n termos calculados.

Resultados Obtidos:

N	Erro (ε)	Valor de S
13	10^{-8}	9.869604398
14	10^{-9}	9.8696044004
16	10^{-10}	9.86960440105
17	10^{-11}	9.869604401081
19	10^{-12}	9.8696044010889
20	10^{-13}	9.86960440108926
22	10^{-14}	9.869604401089354
23	10^{-15}	9.8696044010893580

Análise do Programa:

Neste programa, utilizámos 3 funções, que iremos explicar seguidamente. Inicializámos ainda um *array* “mem_fact”, onde guardamos os valores dos fatoriais já utilizados, de maneira que cada fatorial é calculado uma única vez. Isto torna o programa mais eficiente, pois elimina a necessidade de calcular cada fatorial na sua totalidade.

Embora o k só chegue a 23, como na expressão do exercício temos um fatorial de (k*2), o *array* tem que ter um tamanho igual ou superior ao dobro do valor máximo do k.

```
// verifiquei que k <= 23 e suficiente
double mem_fact[47];
```

A primeira função do programa a ser percorrida é a `m_fact_init`, que iguala todos os valores do *array* “mem_fact” a 0.0, com exceção do mem_fact[0], que é 1.0, sendo que 0! = 1. Isto é importante porque vai ser o caso base da recursão presente na função a seguir.

```
void m_fact_init(){
   mem_fact[0] = 1.0;
   for(int i=1;i<47;i++) mem_fact[i] = 0.0;
}
```

A seguinte função chama-se `m_fact` e retorna o fatorial de um número n. Primeiro, verifica se esse fatorial já se encontra em “mem_fact”. Caso não se encontre (ou seja, quando mem_fact[n] == 0.0), chama a função recursivamente, guarda o valor em mem_fact[n] e devolve-o. Se n! já estiver guardado no *array*, simplesmente devolve o valor.

```
double m_fact(int n){
   if(mem_fact[n] != 0.0) return mem_fact[n];
   mem_fact[n] = n*m_fact(n-1);
   return mem_fact[n];
}
```

A terceira e última função é a mais importante, usando a anterior como auxiliar sempre que for necessário calcular um fatorial.

Primeiro, inicializamos dois inteiros. Um `k`, que será equivalente ao contador de termos + 1, ou seja, N+1. O outro é a `precisão`, que será o módulo do expoente de 10 para a precisão que queremos atingir, começando por 8 (para ficar 10^{-8}).
Em seguida, inicializamos três doubles. Um `sum`, que será a soma dos termos, um `termo`, onde guardamos o valor do termo mais recente, e `epsilon`, que contém o valor de ε, através da função da biblioteca math.h, `pow`. Como ainda não foi calculado nenhum termo, a variável `sum` é inicializada com o valor 0.0.

```
void serie(){
   int k,precisao = 8;
   double sum = 0.0, termo, epsilon = pow(0.1,precisao);
   (...)
}
```

Em seguida, onde ficaria (...), temos o ciclo onde é calculado cada termo. Como temos a função auxiliar `m_fact`, onde podemos guardar os valores de fatoriais já calculados, o cálculo de cada termo é trivial, e é simplesmente uma tradução da expressão da série S para código. À medida que calculamos os termos, vamos somando o seu valor ao valor guardado em `sum`.

```
for(k=1;precisao <= 15;k++) {
      termo = 18.0 * pow(m_fact(k),2) / (pow(k,2) * m_fact(2*k));
      (...)
      sum += termo;
}
```

Por último, ainda dentro do ciclo onde temos novamente (...), fica a condição que verifica se o termo mais recente é menor que o valor do erro absoluto (ε) que estamos a analisar. Se for o caso, então o programa imprime o valor de N, que será igual a k-1, e o valor da soma até esse momento, com o mesmo número de casas decimais que a precisão admite. Em seguida, o valor de `precisao` é aumentado por 1, e multiplicamos o `epsilon` por 0.1 (passamos de, por exemplo, 10^{-8} para 10^{-9}).

```
if(termo < epsilon){
         printf("N = %d E = 10^-%d\n",k-1,precisao);
         printf("         S = %.*f\n",precisao+1,sum);
         precisao++;
         epsilon *= 0.1;
}
```

Quando o valor de `precisao` atinge valores superiores a 15, o ciclo e o programa são terminados. Impressos na consola ficam todos os valores de N para os quais o termo para qual k = N é menor que os diferentes ε.

Exercício 2

Escrever um programa que permita calcular um valor aproximado de

$$S = 12 \sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k^2}$$

com erro absoluto inferior a ε, dado.

Programa:

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

void serie(){
   int k,precisao = 8;
   double sum = 0.0,termo,epsilon = pow(0.1,precisao);

   for(k=1;precisao <=15;k++){
      termo = 12.0/pow(k,2);
      if(termo < epsilon){
         printf("N = %d E = 10^-%d\n",k-1,precisao);
         printf("         S = %.*f\n",precisao+1,sum);
         precisao++;
         epsilon *= 0.1;
      }
      if(k%2 == 0) sum-=termo;
      else sum+=termo;
```

```
    }
}

int main(){
   serie();
}
```

De maneira semelhante ao exercício 1, este programa calcula um valor aproximado da série e imprime o seu valor, juntamente com o número de termos calculados até que o erro absoluto seja menor que o εa ser analisado até aí.

Resultados Obtidos:

N	Erro (ε)	Valor de S
34641	10^{-8}	9.869604406
109544	10^{-9}	9.8696044006
346410	10^{-10}	9.86960440104
1095445	10^{-11}	9.869604401094
3464101	10^{-12}	9.8696044010898
10954451	10^{-13}	9.86960440108936
34641016	10^{-14}	9.869604401089306
109544511	10^{-15}	9.8696044010893100

Análise do Programa:

Neste exercício, usamos apenas uma função, a qual chamamos `serie`, que simultaneamente faz os cálculos e imprime os valores pedidos.

Primeiro, inicializamos dois inteiros. Um k, que será equivalente ao contador de termos + 1, ou seja, N+1. O outro é uma `precisao`, que será o módulo do número a qual o 10 está elevado para a precisão que queremos atingir, começando por 8 (para ficar 10^{-8}).
Em seguida, inicializamos três doubles. Um `sum`, que será a soma dos termos, um `termo`, onde guardamos o valor do termo mais recente, e `epsilon`, que contém o valor de ε, através da função da biblioteca math.h, `pow`.

```
int k,precisao = 8;
double sum = 0.0,termo,epsilon = pow(0.1,precisao);
```

De seguida, temos o ciclo que faz os cálculos para cada termo, e o adiciona o seu valor à variável `sum` no final de cada iteração. Como calculamos cada termo individualmente, foi simples utilizar a

expressão original de S e traduzi-la para código. Neste caso, `termo = 12.0/pow(k,2);`.
Omitimos o $(-\ 1)^{k-1}$ porque apenas nos informa se o número é negativo ou positivo, e estar a calcular essa potência sempre que tivéssemos um novo termo seria pouco eficiente. Assim só verificamos a sua paridade, se for par, o termo será subtraído a `sum`, se for ímpar, será antes adicionado.

```
for(k=1;precisao <=15;k++){
      termo = 12.0/pow(k,2);
      (...)
      if(k%2 == 0) sum-=termo;
      else sum+=termo;
   }
```

Onde tínhamos (...) no excerto anterior, temos a seguinte condição. O programa só entra nesta condição quando o valor do termo corrente é inferior ao valor de ε. Quando isto se verifica, o programa imprime o valor de N, que será igual a k-1, e o valor da soma até esse momento, com o mesmo número de casas decimais que a precisão admite. Em seguida, o valor de `precisao` é aumentado por 1, e multiplicamos o `epsilon` por 0.1 (passamos de, por exemplo, 10^{-8} para 10^{-9}).

```
if(termo < epsilon){
         printf("N = %d E = 10^-%d\n",k-1,precisao);
         printf("         S = %.*f\n",precisao+1,sum);
         precisao++;
         epsilon *= 0.1;
      }
```

Quando o valor `precisao` atinge valores superiores a 15, o ciclo e o programa terminam. Impressos na consola ficam todos os valores de N para os quais o termo para qual k = N é menor que os diferentes ε.

Exercício 3

Sabendo que nos dois exercícios anteriores $S = \pi^2$, alterar os programas para imprimir também o erro absoluto efetivamente cometido no cálculo de π^2,
$E = |\ \pi^2 - S\ |$.

Programas:

Em ambos os programas, foi preciso adicionar uma linha de código dentro da condição em `serie`, esta sendo :

```
printf("|pi^2 - S| = %.*f\n\n",precisao+1,pow(M_PI,2) - sum);
```

Nesta linha o programa calcula | $\pi^2 - S$ | com o valor de π definido na biblioteca `math.h` e imprime-o imediatamente com a mesma precisão de S.

No programa do exercício 2, foi ainda preciso adicionar uma função `d_abs`, que serve para devolver o módulo do número que recebe. Foi necessário implementar esta função porque existem casos onde o erro relativo é negativo.

```
double d_abs(double n){
   if(n>=0) return n;
   else return -n;
}
```

1.

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

// verifiquei que k <= 23 e suficiente
double mem_fact[47];

void m_fact_init(){
   mem_fact[0] = 1.0;
   for(int i=1;i<47;i++) mem_fact[i] = 0.0;
}

double m_fact(int n){
   if(mem_fact[n] != 0.0) return mem_fact[n];
   mem_fact[n] = n*m_fact(n-1);
   return mem_fact[n];
}

void serie(){
   int k,precisao = 8;
   double sum = 0.0, termo, epsilon = pow(0.1,precisao);

   for(k=1;precisao <= 15;k++) {
      termo = 18.0 * pow(m_fact(k),2) / (pow(k,2) * m_fact(2*k));
      if(termo < epsilon){
         printf("N = %d E = 10^-%d\n",k-1,precisao);
         printf("         S = %.*f\n",precisao+1,sum);
         printf("|pi^2 - S| = %.*f\n\n",precisao+1,pow(M_PI,2) - sum);
         precisao++;
         epsilon *= 0.1;
      }
      sum += termo;
   }
}

int main(){
   m_fact_init();
   serie();
}
```

2.

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

double d_abs(double n){
   if(n>=0) return n;
   else return -n;
}

void serie(){
   int k,precisao = 8;
   double sum = 0.0,termo,epsilon = pow(0.1,precisao);

   for(k=1;precisao <=15;k++){
      termo = 12.0/pow(k,2);
      if(termo < epsilon){
         printf("N = %d E = 10^-%d\n",k-1,precisao);
         printf("         S = %.*f\n",precisao+1,sum);
         printf("|pi^2 - S| = %.*f\n\n",precisao+1,d_abs(pow(M_PI,2) - sum));
         precisao++;
         epsilon *= 0.1;
      }
      if(k%2 == 0) sum-=termo;
      else sum+=termo;
   }
}

int main(){
   serie();
}
```

Resultados Obtidos:

1.

N	Erro (ε)	Valor de S	E = | π² − S|
13	10^{-8}	9.869604398	0.000000003
14	10^{-9}	9.8696044004	0.0000000007
16	10^{-10}	9.86960440105	0.00000000003
17	10^{-11}	9.869604401081	0.000000000008
19	10^{-12}	9.8696044010889	0.0000000000004
20	10^{-13}	9.86960440108926	0.00000000000010
22	10^{-14}	9.869604401089354	0.000000000000004
23	10^{-15}	9.8696044010893580	0.0000000000000000

2.

N	Erro (ε)	Valor de S	E = | π² − S|
34641	10^{-8}	9.869604406	0.000000005
109544	10^{-9}	9.8696044006	0.0000000005
346410	10^{-10}	9. 86960440104	0.00000000005
1095445	10^{-11}	9.869604401094	0.000000000005
3464101	10^{-12}	9.8696044010898	0.0000000000004
10954451	10^{-13}	9.86960440108936	0.00000000000000
34641016	10^{-14}	9.869604401089306	0.000000000000052
109544511	10^{-15}	9.8696044010893100	0.0000000000000480

Nestes casos, o novo valor representa a diferença do valor real com o aproximado da soma que foi calculada no nosso programa a cada nível de precisão.
Em certos casos, o resultado é representado apenas com 0's. Isso acontece porque a diferença entre o valor real e a soma a esse nível de precisão é mais pequena do que a precisão que esse ε nos permite representar.

Conclusões

Sentimos que conseguimos concluir todos os exercícios com êxito. Em geral não houve dificuldades na resolução deste trabalho prático, seja a nível da criação dos programas como na escrita do relatório. Achamos que ainda será possível melhorar a eficiência do segundo programa, que demora alguns segundos a ser executado devido ao grande número de termos a serem calculados, mas não pensamos em nenhuma boa maneira até ao momento de escrita.
Gostámos de trabalhar neste projeto e como primeiro trabalho em grupo, correu bastante bem e todos os membros contribuíram bem para a sua realização.