U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Análise Numérica

(M2018)

Relatório do Trabalho Prático 4

Amadeu Marques
Eduardo Santos
Francisco Ribeiro
Miguel Ramos
Ricardo Ribeiro

Introdução

Pretendemos neste relatório resolver os problemas propostos no quarto trabalho prático de Análise Numérica (M2018).
À semelhança dos primeiros dois trabalhos práticos os programas foram todos escritos na linguagem C, devido à facilidade de uso dentro do grupo e a boa eficiência da linguagem.
Incluímos ainda as bibliotecas stdio.h e math.h para a impressão de resultados e utilização recorrente das funções: pow(calcular potência), sin(cálculo do seno) e cos(cálculo do cosseno). Todas as outras funções são da nossa autoria.

Exercício 1

Neste exercício, os três programas são muito semelhantes, sendo que todas as funções têm o mesmo comportamento. Assim, não vale a pena comentar em detalhe os três códigos, pelo que só vão ser referidas as alterações depois da primeira explicação.

Nestas implementações obtivemos os valores M_integral_rect, M_integral_trap e M_integral_simp que representam uma majoração do máximo das respetivas derivadas necessárias para o cálculo do erro. Estes cálculos foram feitos à parte de forma a diminuir a extensão dos códigos e facilitando o processo do cálculo dos valores.

Método dos retângulos:

Código do programa:

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

double a = 0.0, b = 3.0;
double M_integral_rect = 1.261; //calculado externamente

double func(double x){
	return sin(cos(sin(cos(pow(x,2.0)))));
}

unsigned long intervalo_rect(double epsilon){
	unsigned long n = 1; // numero de subintervalos
	double h = (double)(b - a)/n; // tamanho de cada subintervalo
	double erro = (double)((b - a)/2.0 * h * M_integral_rect);
```

```
	while( erro > epsilon ){
		n <<= 1;
		h = (double)(b - a)/n;
		erro = (double)((b - a)/2.0 * h * M_integral_rect);
	}

	return n;
}

double integral_rect(double epsilon){
	unsigned long n = intervalo_rect(epsilon);
	double h = (double)(b - a)/n;
	double value = 0.0;

	for(unsigned long i = 1; i <= n ; i++){
		value += func(a + (i-1)*h);
	}

	value *= h;
	printf("n = %li\n",n);
	printf("value = %.15f\n\n", value);
	return value;
}

int main(){
	integral_rect(pow(10,-5));
	integral_rect(pow(10,-7));
	integral_rect(pow(10,-9));
}
```

Este programa, à semelhança dos outros, contém três funções para além do main.

A primeira, func, recebe um x e retorna o resultado da função $sin(cos(sin(cos(x^2))))$.

A segunda, intervalo_rect, recebe uma precisão épsilon e vai retornar um valor n que corresponde ao número de iterações que vão correr na próxima função.

A última função e mencionada anteriormente é a integral_rect, que recebe uma precisão épsilon. Esta função define um h, do tipo $\frac{b-a}{n}$, em que b é igual a 3.0 e a é igual a 0.0. Depois, vai inicializar um value a 0.0 e entra num ciclo que irá correr n vezes, onde a value é adicionado o valor de func(a + (i-1)*h). Quando o ciclo termina, a função imprime o produto de value por h.

Método dos trapézios:

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

double a = 0.0, b = 3.0;
double M_integral_trap = 11.26; //calculado externamente

double func(double x){
	return sin(cos(sin(cos(pow(x,2.0)))));
}

unsigned long intervalo_trap(double epsilon){
	unsigned long n = 1; // numero de subintervalos
	double h = (double)(b - a)/n; // tamanho de cada subintervalo
	double erro = (double)(pow(h,2)/12.0 * (b - a) * M_integral_trap);

	while( erro > epsilon ){
		n <<= 1;
		h = (double)(b - a)/n;
		erro = (double)(pow(h,2)/12.0 * (b - a) * M_integral_trap);
	}

	return n;
}

double integral_trap(double epsilon){
	unsigned long n = intervalo_trap(epsilon);
	double h = (double)(b - a)/n;
	double value = 0.0;

	for(unsigned long i = 1; i < n ; i++){
		value += func(a + (i)*h);
	}

	value = value*h + h/2.0*(func(a) + func(b));
	printf("n = %li\n",n);
	printf("value = %.15f\n\n", value);
	return value;
}

int main(){
	integral_trap(pow(10,-5));
	integral_trap(pow(10,-7));
	integral_trap(pow(10,-9));
}
```

Como os outros programas, neste existem três funções, func, intervalo_trap e integral_trap. As maiores diferenças serão no cálculo do erro e no cálculo do valor da integral na última função.

Método de Simpson:

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

double a = 0.0, b = 3.0;
//maximize fourth derivate sin(cos(sin(cos(x^2)))) over [2.854120,2.854128]
double M_integral_simp = 3385; //calculado externamente //y = 3384.8

double func(double x){
	return sin(cos(sin(cos(pow(x,2.0)))));
}

unsigned long intervalo_simp(double epsilon){
	unsigned long n = 1; // numero de subintervalos
	double h = (double)(b - a)/n; // tamanho de cada subintervalo
	double erro = (double)((pow(h,4.0)/180)*(b - a)* M_integral_simp);

	while( erro > epsilon ){
		n <<= 1;
		h = (double)(b - a)/n;
		erro = (double)((pow(h,4.0)/180)*(b - a)* M_integral_simp);
	}

	return n;
}

double integral_simp(double epsilon){
	unsigned long n = intervalo_simp(epsilon);
	double h = (double)(b - a)/n;
	double value = 0.0;

	for(unsigned long i = 1; i <= n-1 ; i++){
		if(i%2!=0) value += 4*func(a + i*h);
		else value += 2*func(a + i*h);
	}

	value += func(a) + func(b);
	value *= h/3;
	printf("n = %li\n",n);
	printf("value = %.15f\n\n", value);
	return value;
}
```

```
int main(){
	integral_simp(pow(10,-5));
	integral_simp(pow(10,-7));
	integral_simp(pow(10,-9));
}
```

Igual aos anteriores, neste existem três funções, func, intervalo_simp e integral_simp. As maiores diferenças serão no cálculo do erro e no cálculo do valor da integral na última função.

Exercício 2

Neste exercício foi-nos pedidos para calcular, se possível, valores aproximados de ***I*** com:

Erro absoluto majorado inferior a $\varepsilon = 10^{-5}$
Método dos retângulos: 2.126853
Método dos trapézios: 2.126853
Método de Simpson: 2.126853

´Erro absoluto majorado inferior a $\varepsilon = 10^{-7}$
Método dos retângulos: 2.12685380
Método dos trapézios: 2.12685394
Método de Simpson: 2.12685395

Erro absoluto majorado inferior a $\varepsilon = 10^{-9}$
Método dos retângulos: 2.1268539559
Método dos trapézios: 2.1268539521
Método de Simpson: 2.1268539519

Exercício 3

Neste exercício iremos comentar os resultados obtidos nas 3 implementações e averiguar qual é o método que se revela mais eficaz no cálculo aproximado do integral. Como critérios de avaliação consideramos o tempo de execução e o número de subdivisões necessárias para obter um resultado em que o erro é inferior ao erro pedido.

$$I = \int_0^3 \sin(\cos(\sin(\cos(x^2)))) \, \mathrm{d}x$$

Fig.1: Integral a calcular

Método dos retângulos:

Para este método utilizamos as seguintes fórmulas estudadas no decorrer das aulas:

$$I_f \approx R_n = h \sum_{i=1}^{n} f(x_{i-1})$$

Fig.2: Fórmula de aproximação do método dos retângulos

Tal como explicado no código em primeiro lugar necessitamos de obter o valor de h, ou seja, o valor da amplitude entre cada ponto de forma a obter uma amplitude que nos assegure que ao calcular o erro será inferior ao erro pedido. Para isso, utilizamos a seguinte fórmula sabendo que *f*'(x) é contínua e derivável no intervalo [a,b]:

$$E_n^R = \frac{b-a}{2} h f'(t), \quad a < t < b$$

Fig.3: Fórmula do erro do método dos retângulos

Após a aplicação dos algoritmos obtivemos então os seguintes resultados:

ε	10^{-5}	10^{-7}	10^{-9}
Nº de iterações	1048576	67108864	8589934592
I	2.126853	2.12685380	2.1268539559
Tempo de execução	0m0,074s	0m4,468s	9m25,741s

Método dos trapézios:

Para este método utilizamos as seguintes fórmulas dadas no decorrer das aulas:

$$\frac{h}{2}(f(a)+f(b)) + h \sum_{i=1}^{n-1} f(x_i) = T_n$$

Fig 4: Fórmula de aproximação do método dos trapézios

Inicialmente é necessário calcular h, que é a amplitude entre cada ponto. Para obter um valor de erro inferior ao pedido utilizámos a seguinte fórmula sabendo que $f''(x)$ é contínua e derivável no intervalo [a,b]:

$$E_n^T = -\frac{h^2}{12}(b-a)\, f''(t), \quad a < t < b$$

Fig 5: Fórmula do erro do método dos trapézios

ε	10^{-5}	10^{-7}	10^{-9}
Nº de iterações	2048	16384	262144
I	2.126853	2.12685394	2.1268539519
Tempo de execução	0m0,001s	0m0,002s	0m0,019s

Método de Simpson:

O Método de Simpson composto, utiliza parábolas que interpolam os pontos de um dado intervalo para aproximar a curvatura da função a integrar:

$$S_n = \frac{h}{3}(f(a) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(b))$$

Fig 6: Fórmula de aproximação do método de Simpson

Começamos por calcular h que é a amplitude entre cada ponto, para obter um valor de erro inferior ao pedido utilizamos a seguinte fórmula sabendo que $f^4(x)$ é contínua e derivável no intervalo [a,b]:

$$E_n^S = -\frac{h^4}{180}(b-a)\, f^4(t), \quad a < t < b$$

Fig 7: Fórmula do erro do método de Simpson

ε	10^{-5}	10^{-7}	10^{-9}
Nº de iterações	256	512	2048
I	2.126853	2.12685395	2.1268539519
Tempo de execução	0m0,001s	0m0,001s	0m0,001s

Após a implementação destes métodos podemos perceber que apesar de serem eficazes a calcular os valores com os erros necessários o método de Simpson é sem dúvida o mais eficaz. Aumentando a precisão verificamos que o nº de iterações necessárias aos diferentes métodos varia significativamente. Porém verificamos que essa mesma variação é bastante menos acentuada no método de Simpson. Podemos verificar, por exemplo que necessitamos de 2048 iterações para calcular um valor com erro menor a 10^{-9} através do método de simpson que é exatamente o mesmo número de iterações necessário para calcular um valor com erro menor a 10^{-5} através do método dos trapézios. Podemos concluir então que de um ponto de vista computacional o último método implementado é o mais eficiente e eficaz.

Conclusões

Neste trabalho decidimos fazer uma implementação mais virada para programação visto que, dadas as características do problema, torna-se bastante fácil a implementação das fórmulas. No final, dados os resultados obtidos conseguimos perceber que o método de Simpson é o mais eficiente apesar de, por vezes, o cálculo das derivadas poder ser um problema. Dadas as implementações consideramos bastante interessante a utilização destes métodos, já que a partir das mesmas conseguimos obter resultados bastante aproximados ao resultado que seria de esperar e, desta forma, conseguimos evitar o problema de descrever uma função analiticamente complexa.
Gostámos de trabalhar neste projeto, tal como nos restantes trabalhos, e pensamos que correu bastante bem e todos os membros cumpriram bem os seus objetivos para a realização do trabalho.