MAC0214 – Cálculo Numérico com Aplicações em Física
Exercício Programa 3 – Integração Numérica

1a)

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

// Função para o integrando do problema
float func(float x) {
	return 7 - 5 * pow(x, 4);
}

int main() {
	// Arquivo para salvar dados em tabela para uso posterior
	FILE* fptr;
	fptr = fopen("data_1a.txt", "w");

	// Loop principal iterante sobre todos os valores de p
	for (int p = 1; p < 26; p++) {
		int N = pow(2, p);	// Número de partições do intervalo

		// Fazemos a soma das áreas dos trapézios
		float sum = 0.0f;
		for (int i = 0; i < N; i++)
			sum += ( func((float)i / N) + func((float)(i+1) / N) ) * (1.0f / N) / 2;
		float error = fabs(6 - sum);  // Erro comparado com o valor obtido analiticamente: 6

		// Imprimimos a tabela com valores na tela e no arquivo de dados
		fprintf(fptr, "%d\t%f\n", p, (float)log10(error));
		printf("%d\t%d\t%f\t%f\n", p, N, sum, error);
	}

	// Finalização do algoritmo
	fclose(fptr);
	return 0;
}
```

1b)

Tabela 1. Valores dos parâmetros de integração por trapézios usando precisão simples (single).

p	N	I_{num}	erro
1	2	5.593750	0.406250
2	4	5.896484	0.103516
3	8	5.973999	0.026001
4	16	5.993492	0.006508
5	32	5.998372	0.001628
6	64	5.999592	0.000408
7	128	5.999898	0.000102
8	256	5.999973	0.000027
9	512	5.999988	0.000012
10	1024	5.999997	0.000003
11	2048	5.999996	0.000004
12	4096	6.000007	0.000007
13	8192	6.000003	0.000003
14	16384	6.000004	0.000004
15	32768	5.999997	0.000003
16	65536	6.000094	0.000094
17	131072	6.000277	0.000277
18	262144	5.999516	0.000484
19	524288	6.002491	0.002491
20	1048576	6.007617	0.007617
21	2097152	5.995615	0.004385
22	4194304	6.044366	0.044366
23	8388608	5.722580	0.277420
24	16777216	6.564703	0.564703
25	33554432	4.000000	2.000000

1c)

Tabela 2. Valores dos parâmetros de integração por trapézios usando precisão dupla (double).

p	N	I_{num}	erro
1	2	5.5937500000000000	0.4062500000000000
2	4	5.8964843750000000	0.1035156250000000
3	8	5.9739990234375000	0.0260009765625000
4	16	5.9934921264648438	0.0065078735351562
5	32	5.9983725547790527	0.0016274452209473
6	64	5.9995931088924408	0.0004068911075592
7	128	5.9998982753604650	0.0001017246395350
8	256	5.9999745687237009	0.0000254312762991
9	512	5.9999936421736493	0.0000063578263507
10	1024	5.9999984105431476	0.0000015894568524
11	2048	5.9999996026356648	0.0000003973643352
12	4096	5.9999999006589277	0.0000000993410723
13	8192	5.9999999751647461	0.0000000248352539
14	16384	5.9999999937911603	0.0000000062088397
15	32768	5.9999999984478265	0.0000000015521735
16	65536	5.9999999996119406	0.0000000003880594
17	131072	5.9999999999031610	0.0000000000968390
18	262144	5.9999999999757589	0.0000000000242411
19	524288	5.9999999999941851	0.0000000000058149
20	1048576	5.9999999999985478	0.0000000000014522
21	2097152	5.9999999999993276	0.0000000000006724
22	4194304	6.0000000000007025	0.0000000000007025
23	8388608	6.0000000000001634	0.0000000000001634
24	16777216	5.9999999999978266	0.0000000000021734
25	33554432	5.9999999999994325	0.0000000000005675

1d)

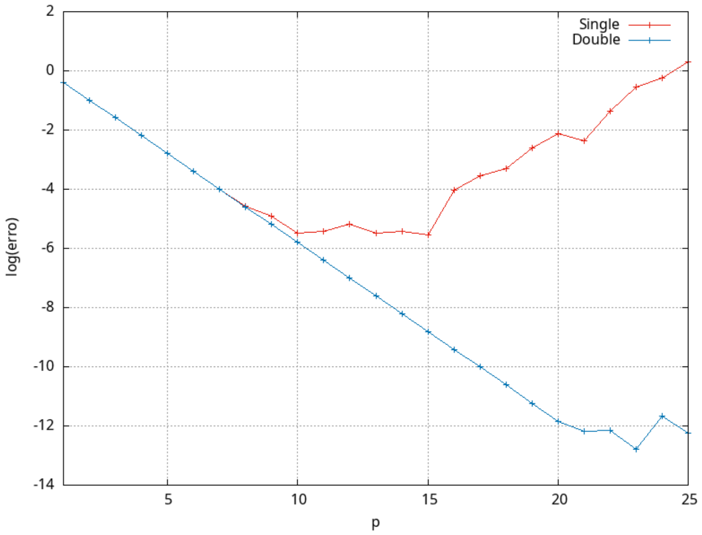

Figura 1. Gráfico do $\log_{10}$ do erro do método dos trapézios para cada precisão (simples e dupla).

O **erro de truncamento** é a divergência entre o valor real de uma soma infinita e o valor calculado por uma soma finita, portanto, no gráfico do log do erro total (Figura 1), vemos que a contribuição pelo erro de truncamento descresce linearmente com o expotente p. As retas do log dos erros para precisão simples e dupla se interceptam no intervalo de [0, 7].

O **erro de round-off** ocorre quando o algortitmo tenta representar um número com precisão finita, ou seja, quando o valor atinge o limite de precisão do computador e os cálculos são influênciados por arredondamentos dos números. Vemos as influências desses erros em intervalos diferentes para cada precisão. Para a precisão simples, começa em p=8, e para a precisão dupla em p=21. No gráfico, notamos como o log do erro para a se comportar de maneira caótica, crescendo para expoentes maiores.

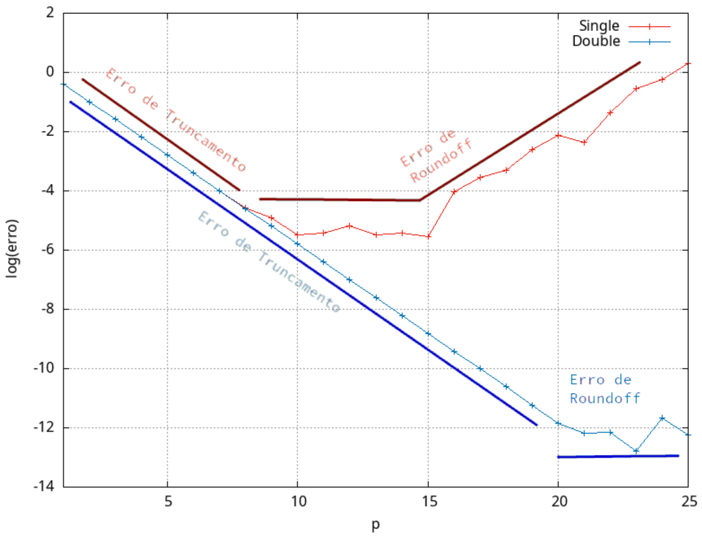

Figura 2. Indição dos erros de truncamento e roudoff para cada precisão.

2)

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

// Número máximo de pontos gerados para análise
#define       MAX_POINTS     99999

// Função do integrando da integral presente na relação de T
double func(double x, double k) {
       return 1.0f / sqrt(1 - pow(k * sin(x), 2));
}

int main() {
       // Período para ângulos pequenos
       double T_0 = 2 * M_PI * sqrt( 1 / 9.81 );

       int N = 100;   // Número de valores de theta entre [0, pi)
       for (int i = 0; i < N; i++) {
              double S              = 0.0f;                       // Soma da integral
              double theta   = M_PI * i / N;               // thata_0 inicial
              double k              = sin(theta / 2);      // Constante k

              // Iteramos sobre os pontos entre os limites de integração
              for (int j = 0; j < MAX_POINTS; j++) {
                     int weight = 1;
              // Peso predefinido
                     // Muda o peso de acordo com j
                     if (j != 0 && j != MAX_POINTS - 1) weight = 4 - 2 * (j % 2)
                     // Soma o produto do valor do ponto com o peso respectivo
                     S += weight * func((M_PI / 2) * ((float)j / MAX_POINTS), k);
              }
       // Faz o produto pelo fator restante
              S = S * ((M_PI / 2) / (MAX_POINTS - 1)) / 3;

              // Calcula T para um pêndulo de 1 metro de comprimento
              double T = 4 * sqrt(1 / 9.81) * S;

              // Imprime os valore obtidos na tela
              printf("%.16f\t%.16f\n", theta, T/T_0);
       }

       // Finaliza
       return 0;
}
```

Tabela 3. Valores calculados de θ_0 e de T.

θ_0 (rad)	T (s)
0.0000000000000000	1.9924199685969697
0.3141592653589793	2.0046516608092420
0.6283185307179586	2.0421795600304185
0.9424777960769379	2.1076706010489916
1.2566370614359172	2.2062080625568852
1.5707963267948966	2.3465851564836098
1.8849555921538759	2.5441329644479822
2.1991148575128552	2.8275574153906931
2.5132741228718345	3.2590462795913959
2.8274333882308138	4.0191313127537498

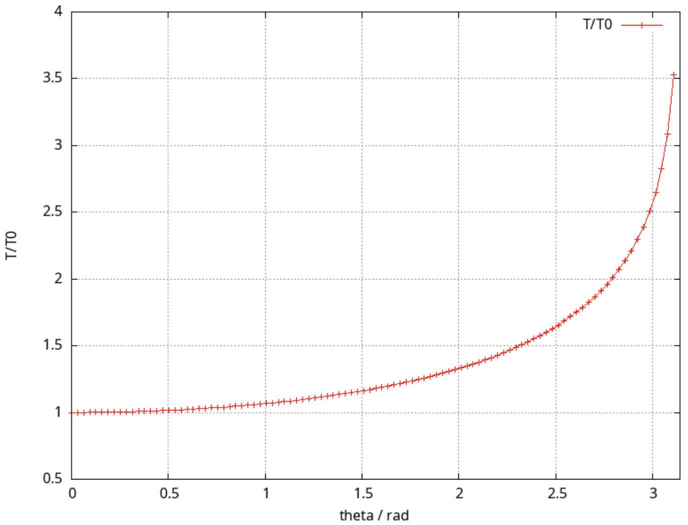

Figura 3. Gráfico da razão entre o período para um pêndulo oscilando com ângulo apreciável e período de um pêndulo oscilando com ângulos pequenos.

3)

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

// Função que gera um aleatório Z baseado numa semente inicial.
unsigned long int random(unsigned long int z) {
        return (1103515245 * z + 12345) % 2147483647;
}

int main() {

        int N                           = 100;  // Número total de pontos por tentativa
        unsigned long int z = 9;        // Semente inicial fornecida ao gerador randômico

        float Im[17]            = {};   // Array com valores médios de I
        float sI[17]            = {};   // Array com desvios-padrões amostrais de I
        float sIm[17]           = {};   // Array com desvios-padrões amostrais de I

        // Loop principal
        for (int i = 0; i < 17; i++) {
                int M = pow(2, i+1);    // Número de tentativas M por linha

                float I[M];                     // Array com valores da área armazenados

                // Fazemos cada simulação por Monte Carlo
                for (int j = 0; j < M; j++) {
                        int n = 0;              // Número de pontos abaixo do gráfico de y

                        // Geração dos pontos
                        for (int i = 0; i < N; i++) {
                                z = random(z);                                  // Geramos novo z
                                float x = (float)z / 2147483647;        // Geramos x baseado em z

                                z = random(z);                                  // Geramos novo z
```

```
                        float y = (float)z / 2147483647;      // Geramos y baseado em z

                        // Se o ponto está abaixo de y, então incrementamos n por 1
                        if (y <= pow(x, 4)) n++;
                }

                // Área do gráfico em 0 < x < 1 (a área total é unitária, portanto a área sob
                // o gráfico é equivalente à razão dos pontos internos e pontos totais)
                float ratio = (float)n / N;
                I[j] = ratio;
        }

        // Calculamos o valor médio de I
        Im[i] = 0.0f;
        for (int j = 0; j < M; j++)
                Im[i] += I[j] / M;

        // Calculamos o desvio-padrão amostral de I
        sI[i] = 0.0f;
        for (int j = 0; j < M; j++)
                sI[i] += pow(I[j] - Im[i], 2);
        sI[i] = sqrt(sI[i] / (M - 1));

        // Calculamos o desvio-padrão da média de Im
        sIm[i] = sI[i] / sqrt(M);
    }

    // Imprimimos a tabela
    for (int i = 0; i < 17; i++)
        printf("%d\t%f\t%f\t%f\n", (int)pow(2, i), Im[i], sI[i], sIm[i]);

    return 0;
}
```

Tabela 4. Estimativa da área sob curva através do Método de Monte Carlo.

N_t	I_m	σ	σ_m
2	0.225000	0.007071	0.005000
4	0.180000	0.029439	0.014720
8	0.197500	0.038079	0.013463
16	0.186250	0.036674	0.009169
32	0.190313	0.044103	0.007796
64	0.198437	0.036219	0.004527
128	0.196328	0.043215	0.003820
256	0.200351	0.042216	0.002639
512	0.199961	0.039804	0.001759
1024	0.198643	0.039610	0.001238
2048	0.201197	0.039637	0.000876
4096	0.200312	0.040203	0.000628
8192	0.199911	0.040202	0.000444
16384	0.199474	0.039489	0.000309
32768	0.199621	0.040068	0.000221
65536	0.199943	0.039991	0.000156
131072	0.200015	0.040082	0.000111