pi_accuracy_graficos.c

```c

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h> // Necessário para M_PI

double calculate_pi_sequential(long long num_iterations) {
    double sum = 0.0;
    int sign = 1;

    for (long long i = 0; i < num_iterations; i++) {
        double term = (double)sign / (2.0 * i + 1.0);
        sum += term;
        sign *= -1;
    }

    return 4.0 * sum;
}


double calculate_pi_ilp(long long num_iterations) {
    double sum1 = 0.0, sum2 = 0.0, sum3 = 0.0, sum4 = 0.0;

    for (long long i = 0; i < num_iterations; i += 4) {
        sum1 += 1.0 / (2.0 * i + 1.0);
        sum2 -= 1.0 / (2.0 * i + 3.0);
        sum3 += 1.0 / (2.0 * i + 5.0);
        sum4 -= 1.0 / (2.0 * i + 7.0);
    }

    double total_sum = sum1 + sum2 + sum3 + sum4;

    return 4.0 * total_sum;
}


int main() {
    long long iterations[] = {
        10, 100, 1000, 10000, 100000, 1000000,
        10000000, 100000000, 1000000000
    };
    int num_tests = sizeof(iterations) / sizeof(iterations[0]);

    // Abre o ficheiro CSV para escrita
    FILE *csv_file = fopen("pi_results.csv", "w");
    if (csv_file == NULL) {
        perror("Erro ao abrir o ficheiro CSV");
        return 1;
    }

    // Escreve o cabeçalho do CSV
```

```c
    fprintf(csv_file, "Metodo,Iteracoes,PiCalculado,ErroAbsoluto,Tempo_s\n");
    printf("A gerar resultados em pi_results.csv...\n");

    // --- Testes para a Versão Sequencial ---
    for (int i = 0; i < num_tests; i++) {
        long long current_iterations = iterations[i];
        struct timespec start_time, end_time;

        clock_gettime(CLOCK_MONOTONIC, &start_time);
        double pi_approximation = calculate_pi_sequential(current_iterations);
        clock_gettime(CLOCK_MONOTONIC, &end_time);

        double time_spent = (end_time.tv_sec - start_time.tv_sec) +
                            (end_time.tv_nsec - start_time.tv_nsec) / 1e9;
        double error = fabs(M_PI - pi_approximation);

        // Escreve os dados no ficheiro CSV
        fprintf(csv_file, "Sequencial,%lld,%.18f,%.18f,%.9f\n",
               current_iterations, pi_approximation, error, time_spent);
    }

    // --- Testes para a Versão Otimizada com ILP ---
    for (int i = 0; i < num_tests; i++) {
        long long current_iterations = iterations[i];
        struct timespec start_time, end_time;

        clock_gettime(CLOCK_MONOTONIC, &start_time);
        double pi_approximation = calculate_pi_ilp(current_iterations);
        clock_gettime(CLOCK_MONOTONIC, &end_time);

        double time_spent = (end_time.tv_sec - start_time.tv_sec) +
                            (end_time.tv_nsec - start_time.tv_nsec) / 1e9;
        double error = fabs(M_PI - pi_approximation);

        // Escreve os dados no ficheiro CSV
        fprintf(csv_file, "ILP_Otimizado,%lld,%.18f,%.18f,%.9f\n",
               current_iterations, pi_approximation, error, time_spent);
    }

    // Fecha o ficheiro
    fclose(csv_file);

    printf("Resultados guardados com sucesso em pi_results.csv\n");
    printf("Valor de referência de M_PI: %.18f\n", M_PI);

    return 0;
}

```