correlacaoDePearson.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define QUANTIDADE_CANDLES 100
#define TAMANHO_STRING 50

double leituraCotacoes[QUANTIDADE_CANDLES];
char nomeRobo[50],nomeTipoGrafico[2];
double metodoCorrelacao(int tempoCorrelacao);
void detectaRoboETipoDeGrafico();

int main(){
   metodoCorrelacao(21);
   printf("%f\n",metodoCorrelacao(21));
   return 0;
}

double metodoCorrelacao(int tempoCorrelacao){
    FILE *arquivo;
    double somaOrdenadas = 0, somaAbcissas = 0,
           somaOrdenadasQuadrado = 0, somaAbcissasQuadrado = 0,
           somaXvezesY = 0, correlacao,
           numeroAbcissa, numeroOrdenada,
           numerador, denominador_1,denominador;
    int c;
    detectaRoboETipoDeGrafico();

    if( (strcmp(nomeTipoGrafico,"M1")) == 0)
            arquivo = fopen("tabela1Minuto.csv","rt");
    else if( (strcmp(nomeTipoGrafico,"M5")) == 0)
            arquivo = fopen("tabela5Minutos.csv","rt");
    else if( (strcmp(nomeTipoGrafico,"H1")) == 0)
```

```
34.             arquivo = fopen("tabela1Hora.csv","rt");
35.     else
36.             printf("Erro, tabela nao encontrada\n");
37.
38.     for(c=0; c<QUANTIDADE_CANDLES; c++){
39.         fscanf(arquivo, "%lf",&leituraCotacoes[c]);
40.     }
41.
42.     for(c=0; c<tempoCorrelacao; c++){
43.         numeroAbcissa = leituraCotacoes[c];
44.         numeroOrdenada = leituraCotacoes[c+1];
45.
46.         somaAbcissas =   somaAbcissas + numeroAbcissa;
47.         somaAbcissasQuadrado += (numeroAbcissa*numeroAbcissa);
48.         somaOrdenadas = somaOrdenadas + numeroOrdenada;
49.         somaOrdenadasQuadrado += (numeroOrdenada*numeroOrdenada);
50.         somaXvezesY = somaXvezesY + (numeroOrdenada*numeroAbcissa);
51.      }
52.
53.     numerador =((tempoCorrelacao*somaXvezesY)-((somaAbcissas)*(somaOrdenadas)));
54.     denominador_1 =((tempoCorrelacao*somaAbcissasQuadrado)-(somaAbcissas*somaAbcissas))*
55.     ((tempoCorrelacao*somaOrdenadasQuadrado)-(somaOrdenadas*somaOrdenadas));
56.
57.     denominador = sqrt(denominador_1);
58.     correlacao = numerador/denominador;
59.
60.     return correlacao;
61.
62.     printf("%f\n",correlacao);
63.     fclose(arquivo);
64. }
65. void detectaRoboETipoDeGrafico(){
66.     FILE *arquivo;
67.
```

```
68.     arquivo = fopen("criterioEntrada.txt","rt");
69.     fgets(nomeRobo, 50,arquivo);
70.     fgets(nomeTipoGrafico, 3,arquivo);
71.     fclose(arquivo);
72. }
```

testeCorrelacaoDePearson.c

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <CUnit/Basic.h>
4.
5.  int init_suite(void) {
6.      return 0;
7.  }
8.  int clean_suite(void) {
9.      return 0;
10. }
11.
12. double metodoCorrelacao(int tempoCorrelacao);
13.
14. void testMetodoCorrelacao() {
15.     int tempoCorrelacao = 21;
16.     double resultadoCorrelacao = metodoCorrelacao(tempoCorrelacao);
17.
18.     CU_ASSERT_DOUBLE_EQUAL(0.748820,resultadoCorrelacao, 0.001 );
19. }
20. int main() {
21.     CU_pSuite pSuite = NULL;
22.
23.     /* Initialize the CUnit test registry */
24.     if (CUE_SUCCESS != CU_initialize_registry())
25.         return CU_get_error();
26.
27.     /* Add a suite to the registry */
```

```
    pSuite = CU_add_suite("correlacaoLinearTeste", init_suite, clean_suite);
    if (NULL == pSuite) {
        CU_cleanup_registry();
        return CU_get_error();
    }
    /* Add the tests to the suite */
    if ((NULL == CU_add_test(pSuite, "testMetodoCorrelacao", testMetodoCorrelacao))) {
        CU_cleanup_registry();
        return CU_get_error();
    }
    /* Run all tests using the CUnit Basic interface */
    CU_basic_set_mode(CU_BRM_VERBOSE);
    CU_basic_run_tests();
    CU_cleanup_registry();
    return CU_get_error();
}
```

fibonacci

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

double calculoSuporte(int quantidadeVelas);
double calculoResistencia(int quantidadeVelas);
double calculoRegressaoFibonacci(double fatorDeRegressao, int quantidadeVelas);

int main(){

    printf("Suporte = %lf, resistencia =
%lf\n",calculoSuporte(13),calculoResistencia(13));
    printf("Regressao De Fibonacci = %lf\n",calculoRegressaoFibonacci(0.23, 13));
    return 0;
}

double calculoSuporte(int quantidadeVelas){
    FILE *arquivo;
```

```
    double cotacao[quantidadeVelas];
    double suporte = 0;
    int i;
    arquivo = fopen("dadosFibonacci.txt","rt");

    for(i = 0; i < quantidadeVelas; i++){
        fscanf(arquivo, "%lf",&cotacao[i]);

        if(suporte < cotacao[i])
            suporte = cotacao[i];
    }
    fclose(arquivo);
    return suporte;
}
double calculoResistencia(int quantidadeVelas){
    FILE *arquivo;
    double cotacao[quantidadeVelas];
    double resistencia = 777;
    int i;

    arquivo = fopen("dadosFibonacci.txt","rt");

    for(i = 0; i < quantidadeVelas; i++){
        fscanf(arquivo, "%lf",&cotacao[i]);
        if(resistencia > cotacao[i])
            resistencia = cotacao[i];
    }
    fclose(arquivo);
    return resistencia;
}

double calculoRegressaoFibonacci(double fatorDeRegressao, int quantidadeVelas){
    double variacaoDePontos = calculoSuporte(quantidadeVelas) -
calculoResistencia(quantidadeVelas);
```

```
    return variacaoDePontos*fatorDeRegressao;
}
```

testeFibonnaci

```
#include <stdio.h>
#include <stdlib.h>
#include <CUnit/Basic.h>

int init_suite(void) {
    return 0;
}
int clean_suite(void) {
    return 0;
}
double calculoRegressaoFibonacci(double fatorDeRegressao, int quantidadeVelas);

void testCalculoRegressaoFibonacci() {
    double resultadoRegressaoFibonacci = calculoRegressaoFibonacci(0.23, 13);
    CU_ASSERT_DOUBLE_EQUAL(0.162380,resultadoRegressaoFibonacci, 0.001 );
}

double calculoResistencia(int quantidadeVelas);

void testCalculoResistencia() {
    int quantidadeVelas = 13;
    CU_ASSERT_DOUBLE_EQUAL(136.290,calculoResistencia(quantidadeVelas), 0.001 );
}

double calculoSuporte(int quantidadeVelas);

void testCalculoSuporte() {
     int quantidadeVelas = 13;
    CU_ASSERT_DOUBLE_EQUAL(136.996,calculoSuporte(quantidadeVelas), 0.001 );
}
```

```
31.
32. int main() {
33.     CU_pSuite pSuite = NULL;
34.     /* Initialize the CUnit test registry */
35.     if (CUE_SUCCESS != CU_initialize_registry())
36.         return CU_get_error();
37.     /* Add a suite to the registry */
38.     pSuite = CU_add_suite("fibonacciTeste", init_suite, clean_suite);
39.     if (NULL == pSuite) {
40.         CU_cleanup_registry();
41.         return CU_get_error();
42.     }
43.     /* Add the tests to the suite */
44.     if ((NULL == CU_add_test(pSuite, "testCalculoRegressaoFibonacci",
    testCalculoRegressaoFibonacci)) ||
45.             (NULL == CU_add_test(pSuite, "testCalculoResistencia",
    testCalculoResistencia)) ||
46.             (NULL == CU_add_test(pSuite, "testCalculoSuporte", testCalculoSuporte))) {
47.         CU_cleanup_registry();
48.         return CU_get_error();
49.     }
50.     /* Run all tests using the CUnit Basic interface */
51.     CU_basic_set_mode(CU_BRM_VERBOSE);
52.     CU_basic_run_tests();
53.     CU_cleanup_registry();
54.     return CU_get_error();
55. }
```

minimosQuadarados.c

```
1. #include <stdio.h>
2. #include <stdlib.h>
3.
4. double calculoCoeficienteLinear();
5. double calculoCoeficienteAngular();
6.
```

```c
double calculoCoeficienteLinear(int quantidadeVelas){
    FILE *arquivo;
    double x[quantidadeVelas], y[quantidadeVelas];
    double soma_x = 0, soma_y = 0;
    double numerador, denominador;
    double variacaoLinear;
    int i;

    arquivo = fopen("dadosMinimosQuadrados.txt","rt");

    for(i = 1; i < quantidadeVelas; i++){
        fscanf(arquivo, "%lf",&x[i]);
        fscanf(arquivo, "%lf",&y[i]);
        soma_x = soma_x + x[i];
        soma_y = soma_y + y[i];
    }

    for(i = 1; i < quantidadeVelas; i++){
        numerador = x[i]*(y[i] - soma_x/quantidadeVelas);
        denominador = y[i]*(x[i] - soma_y/quantidadeVelas);
    }
    variacaoLinear = numerador/denominador;
    fclose(arquivo);
    return variacaoLinear;
}

double calculoCoeficienteAngular(int quantidadeVelas){
    FILE *arquivo;
    double x[quantidadeVelas], y[quantidadeVelas];
    double soma_x = 0, soma_y = 0;
    double variacaoAngular;
    int i;
    arquivo = fopen("dadosMinimosQuadrados.txt","rt");

```

```
    for(i = 1; i < quantidadeVelas; i++){
        fscanf(arquivo, "%lf",&x[i]);
        fscanf(arquivo, "%lf",&y[i]);
        soma_x = soma_x + x[i];
        soma_y = soma_y + y[i];
    }

    variacaoAngular = soma_y/quantidadeVelas -
(calculoCoeficienteLinear(quantidadeVelas)*soma_x/quantidadeVelas);

    fclose(arquivo);
    return variacaoAngular;
}
```

testeMinimosQuadrados

```
#include <stdio.h>
#include <stdlib.h>
#include <CUnit/Basic.h>

int init_suite(void) {
    return 0;
}

int clean_suite(void) {
    return 0;
}

double calculoRegressaoFibonacci(double fatorDeRegressao, int quantidadeVelas);

void testCalculoRegressaoFibonacci() {
    double resultadoRegressaoFibonacci = calculoRegressaoFibonacci(0.23, 13);

    CU_ASSERT_DOUBLE_EQUAL(0.162380,resultadoRegressaoFibonacci, 0.001 );
}

```

```c
double calculoResistencia(int quantidadeVelas);

void testCalculoResistencia() {
    int quantidadeVelas = 13;

    CU_ASSERT_DOUBLE_EQUAL(136.290,calculoResistencia(quantidadeVelas), 0.001 );
}

double calculoSuporte(int quantidadeVelas);

void testCalculoSuporte() {
     int quantidadeVelas = 13;

    CU_ASSERT_DOUBLE_EQUAL(136.996,calculoSuporte(quantidadeVelas), 0.001 );
}

int main() {
    CU_pSuite pSuite = NULL;

    /* Initialize the CUnit test registry */
    if (CUE_SUCCESS != CU_initialize_registry())
        return CU_get_error();

    /* Add a suite to the registry */
    pSuite = CU_add_suite("fibonacciTeste", init_suite, clean_suite);
    if (NULL == pSuite) {
        CU_cleanup_registry();
        return CU_get_error();
    }

    /* Add the tests to the suite */
    if ((NULL == CU_add_test(pSuite, "testCalculoRegressaoFibonacci",
testCalculoRegressaoFibonacci)) ||
```

```
53.             (NULL == CU_add_test(pSuite, "testCalculoResistencia",
    testCalculoResistencia)) ||
54.             (NULL == CU_add_test(pSuite, "testCalculoSuporte", testCalculoSuporte))) {
55.         CU_cleanup_registry();
56.         return CU_get_error();
57.     }
58.
59.     /* Run all tests using the CUnit Basic interface */
60.     CU_basic_set_mode(CU_BRM_VERBOSE);
61.     CU_basic_run_tests();
62.     CU_cleanup_registry();
63.     return CU_get_error();
64. }
```