## correlacaoDePearson.c

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <math.h>
5. #define QUANTIDADE_CANDLES 100
6. #define TAMANHO_STRING 50
7.
8. double leituraCotacoes[QUANTIDADE_CANDLES];
9. char nomeRobo[50],nomeTipoGrafico[2];
10. double metodoCorrelacao(int tempoCorrelacao);
11. void detectaRoboETipoDeGrafico();
12.
13. int main(){
14.
      metodoCorrelacao(21);
15. printf("%f\n", metodoCorrelacao(21));
16. return 0;
17. }
18.
19. double metodoCorrelacao(int tempoCorrelacao){
20.
       FILE *arquivo;
21.
       double somaOrdenadas = 0, somaAbcissas = 0,
22.
              somaOrdenadasQuadrado = 0, somaAbcissasQuadrado = 0,
23.
              somaXvezesY = 0, correlacao,
24.
              numeroAbcissa, numeroOrdenada,
25.
              numerador, denominador_1,denominador;
26.
       int c;
27.
       detectaRoboETipoDeGrafico();
28.
       if( (strcmp(nomeTipoGrafico, "M1")) == 0)
29.
30.
               arquivo = fopen("tabela1Minuto.csv", "rt");
       else if( (strcmp(nomeTipoGrafico, "M5")) == 0)
31.
               arquivo = fopen("tabela5Minutos.csv","rt");
32.
       else if( (strcmp(nomeTipoGrafico, "H1")) == 0)
33.
```

```
34.
               arquivo = fopen("tabela1Hora.csv","rt");
35.
       else
36.
               printf("Erro, tabela nao encontrada\n");
37.
38.
       for(c=0; c<QUANTIDADE_CANDLES; c++){</pre>
39.
           fscanf(arquivo, "%lf",&leituraCotacoes[c]);
40.
       }
41.
42.
       for(c=0; c<tempoCorrelacao; c++){</pre>
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.
       numerador =((tempoCorrelacao*somaXvezesY)-((somaAbcissas)*(somaOrdenadas)));
53.
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.
       /* Initialize the CUnit test registry */
23.
24.
      if (CUE_SUCCESS != CU_initialize_registry())
25.
           return CU_get_error();
26.
27.
      /* Add a suite to the registry */
```

```
28.
       pSuite = CU_add_suite("correlacaoLinearTeste", init_suite, clean_suite);
29.
       if (NULL == pSuite) {
30.
           CU_cleanup_registry();
31.
           return CU_get_error();
32.
       /* Add the tests to the suite */
33.
34.
       if ((NULL == CU_add_test(pSuite, "testMetodoCorrelacao", testMetodoCorrelacao))) {
35.
           CU_cleanup_registry();
36.
           return CU_get_error();
37.
38.
       /* Run all tests using the CUnit Basic interface */
39.
       CU_basic_set_mode(CU_BRM_VERBOSE);
40.
       CU_basic_run_tests();
41.
       CU_cleanup_registry();
       return CU_get_error();
42.
43. }
```

## fibonacci

```
1. #include <stdio.h>
2. #include <stdlib.h>
3.

    double calculoSuporte(int quantidadeVelas);

5. double calculoResistencia(int quantidadeVelas);

    double calculoRegressaoFibonacci(double fatorDeRegressao, int quantidadeVelas);

7.
8. int main(){
9.
10.
       printf("Suporte = %lf, resistencia =
   %lf\n", calculoSuporte(13), calculoResistencia(13));
       printf("Regressao De Fibonacci = %1f\n",calculoRegressaoFibonacci(0.23, 13));
11.
12.
       return 0;
13. }
14.
15. double calculoSuporte(int quantidadeVelas){
16.
       FILE *arquivo;
```

```
17.
       double cotacao[quantidadeVelas];
18.
       double suporte = 0;
19.
       int i;
20.
       arquivo = fopen("dadosFibonacci.txt","rt");
21.
22.
       for(i = 0; i < quantidadeVelas; i++){</pre>
23.
           fscanf(arquivo, "%lf",&cotacao[i]);
24.
25.
           if(suporte < cotacao[i])</pre>
26.
               suporte = cotacao[i];
27.
       }
28.
       fclose(arquivo);
29.
       return suporte;
30.}
31. double calculoResistencia(int quantidadeVelas){
32.
       FILE *arquivo;
33.
       double cotacao[quantidadeVelas];
34.
       double resistencia = 777;
35.
       int i;
36.
       arquivo = fopen("dadosFibonacci.txt","rt");
37.
38.
39.
       for(i = 0; i < quantidadeVelas; i++){</pre>
40.
           fscanf(arquivo, "%lf",&cotacao[i]);
41.
           if(resistencia > cotacao[i])
42.
               resistencia = cotacao[i];
43.
44.
       fclose(arquivo);
45.
       return resistencia;
46.}
47.
48. double calculoRegressaoFibonacci(double fatorDeRegressao, int quantidadeVelas){
49.
       double variacaoDePontos = calculoSuporte(quantidadeVelas) -
   calculoResistencia(quantidadeVelas);
```

```
50. return variacaoDePontos*fatorDeRegressao;
51. }
```

## testeFibonnaci

```
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. double calculoRegressaoFibonacci(double fatorDeRegressao, int quantidadeVelas);
12.
13. void testCalculoRegressaoFibonacci() {
       double resultadoRegressaoFibonacci = calculoRegressaoFibonacci(0.23, 13);
15.
       CU_ASSERT_DOUBLE_EQUAL(0.162380,resultadoRegressaoFibonacci, 0.001 );
16.}
17.
18. double calculoResistencia(int quantidadeVelas);
19.
20. void testCalculoResistencia() {
21.
       int quantidadeVelas = 13;
22.
       CU_ASSERT_DOUBLE_EQUAL(136.290,calculoResistencia(quantidadeVelas), 0.001 );
23.}
24.
25. double calculoSuporte(int quantidadeVelas);
26.
27. void testCalculoSuporte() {
28.
        int quantidadeVelas = 13;
       CU_ASSERT_DOUBLE_EQUAL(136.996, calculoSuporte(quantidadeVelas), 0.001 );
30.}
```

```
31.
  32. int main() {
         CU_pSuite pSuite = NULL;
  33.
         /* Initialize the CUnit test registry */
  34.
  35.
         if (CUE_SUCCESS != CU_initialize_registry())
  36.
             return CU get error();
         /* Add a suite to the registry */
  37.
  38.
         pSuite = CU_add_suite("fibonacciTeste", init_suite, clean_suite);
  39.
         if (NULL == pSuite) {
  40.
             CU_cleanup_registry();
             return CU_get_error();
  41.
  42.
         /* Add the tests to the suite */
  43.
  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();
         CU_cleanup_registry();
  53.
  54.
         return CU_get_error();
  55. }
minimosQuadarados.c
  1. #include <stdio.h>
  2. #include <stdlib.h>
  3.

    double calculoCoeficienteLinear();

  5. double calculoCoeficienteAngular();
  6.
```

```
7. double calculoCoeficienteLinear(int quantidadeVelas){
8.
       FILE *arquivo;
9.
       double x[quantidadeVelas], y[quantidadeVelas];
10.
       double soma_x = 0, soma_y = 0;
11.
       double numerador, denominador;
12.
       double variacaoLinear;
13.
       int i;
14.
15.
       arquivo = fopen("dadosMinimosQuadrados.txt","rt");
16.
       for(i = 1; i < quantidadeVelas; i++){</pre>
17.
           fscanf(arquivo, "%lf",&x[i]);
18.
           fscanf(arquivo, "%lf",&y[i]);
19.
20.
          soma_x = soma_x + x[i];
21.
          soma_y = soma_y + y[i];
22.
       }
23.
24.
       for(i = 1; i < quantidadeVelas; i++){</pre>
25.
           numerador = x[i]*(y[i] - soma_x/quantidadeVelas);
26.
           denominador = y[i]*(x[i] - soma_y/quantidadeVelas);
27.
28.
       variacaoLinear = numerador/denominador;
29.
       fclose(arquivo);
30.
       return variacaoLinear;
31. }
32.
33. double calculoCoeficienteAngular(int quantidadeVelas){
34.
       FILE *arquivo;
35.
       double x[quantidadeVelas], y[quantidadeVelas];
36.
       double soma_x = 0, soma_y = 0;
37.
       double variacaoAngular;
38.
       int i;
       arquivo = fopen("dadosMinimosQuadrados.txt","rt");
39.
40.
```

```
41.
          for(i = 1; i < quantidadeVelas; i++){</pre>
  42.
              fscanf(arquivo, "%lf",&x[i]);
             fscanf(arquivo, "%lf",&y[i]);
  43.
  44.
              soma_x = soma_x + x[i];
             soma_y = soma_y + y[i];
  45.
  46.
          }
  47.
  48.
          variacaoAngular = soma_y/quantidadeVelas -
      (calculoCoeficienteLinear(quantidadeVelas)*soma_x/quantidadeVelas);
  49.
          fclose(arquivo);
  50.
  51.
          return variacaoAngular;
  52.}
testeMinimosQuadrados
  1. #include <stdio.h>
  2. #include <stdlib.h>
  3. #include <CUnit/Basic.h>
  4.
  5. int init_suite(void) {
  6.
         return 0;
  7. }
  8.
  9. int clean_suite(void) {
  10.
         return 0;
  11. }
  12.
  13. double calculoRegressaoFibonacci(double fatorDeRegressao, int quantidadeVelas);
  14.
  15. void testCalculoRegressaoFibonacci() {
          double resultadoRegressaoFibonacci = calculoRegressaoFibonacci(0.23, 13);
  16.
  17.
  18.
          CU_ASSERT_DOUBLE_EQUAL(0.162380,resultadoRegressaoFibonacci, 0.001 );
  19.}
  20.
```

```
21. double calculoResistencia(int quantidadeVelas);
22.
23. void testCalculoResistencia() {
24.
       int quantidadeVelas = 13;
25.
26.
       CU_ASSERT_DOUBLE_EQUAL(136.290, calculoResistencia(quantidadeVelas), 0.001 );
27. }
28.
29. double calculoSuporte(int quantidadeVelas);
30.
31. void testCalculoSuporte() {
32.
        int quantidadeVelas = 13;
33.
34.
       CU_ASSERT_DOUBLE_EQUAL(136.996,calculoSuporte(quantidadeVelas), 0.001 );
35.}
36.
37. int main() {
38.
       CU_pSuite pSuite = NULL;
39.
40.
       /* Initialize the CUnit test registry */
41.
       if (CUE_SUCCESS != CU_initialize_registry())
42.
           return CU_get_error();
43.
44.
       /* Add a suite to the registry */
45.
       pSuite = CU_add_suite("fibonacciTeste", init_suite, clean_suite);
46.
      if (NULL == pSuite) {
47.
           CU_cleanup_registry();
48.
           return CU_get_error();
49.
       }
50.
       /* Add the tests to the suite */
51.
52.
       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. }
```