

Escola Politécnica da Universidade de São Paulo  
Engenharia Elétrica  
MAP3121 - Métodos Numéricos e Aplicações

# Fatoração de Matrizes para Sistemas de Classificação de *Machine Learning*

Alunos:

Lucas Reis Werner - 10336038

Pedro Rabelo Vasconcelos Dias - 10273904

Professor:

Pedro da Silva Peixoto

São Paulo

2019

# Sumário

<b>1</b>	<b>Introdução</b>	<b>2</b>
1.1	Relevância do Exercício-Programa . . . . .	2
1.2	Objetivo . . . . .	2
<b>2</b>	<b>Implementações</b>	<b>2</b>
2.1	Rotação de Givens . . . . .	2
2.2	Fatoração QR . . . . .	4
2.2.1	Primeira Tarefa . . . . .	9
2.3	Fatoração Não Negativa . . . . .	23
2.3.1	Segunda Tarefa . . . . .	25
<b>3</b>	<b>Machine Learning</b>	<b>27</b>
3.1	Tarefa Principal . . . . .	27
<b>4</b>	<b>Considerações Finais</b>	<b>42</b>

# 1 Introdução

## 1.1 Relevância do Exercício-Programa

Sem dúvidas, *Machine Learning* e *Data Science* são palavras-chave para descrever o papel atual da Computação no mercado de trabalho. Suas aplicações são imensas e são, de fato, parte importante para fazer o aprendizado completo de um engenheiro. É de suma relevância poder utilizar tais ciências em ambientes práticos durante a graduação, tal qual fizemos neste Exercício-Programa, onde buscamos, de forma complexa e efetiva, realizar a identificação de dígitos manuscritos através da aproximação e fatoração de matrizes complexas.

## 1.2 Objetivo

Como já citado anteriormente, o objetivo principal deste Exercício-Programa é o treinamento de um algoritmo para conseguir identificar, de maneira hábil e efetiva, dígitos manuscritos de um banco de dados previamente fornecido. Utilizaremos, para tal, diversas operações de matrizes, como Rotação de Givens, Fatoração QR e Fatoração Não Negativa.

# 2 Implementações

## 2.1 Rotação de Givens

O algoritmo da Rotação de Givens utilizado neste EP, foi o seguinte:

---

---

```
Entrada: matriz real W; inteiros i, j, m, n; reais c e s
1 início
2   inteiro r
3   real aux
4   para r = 1 até m faça
5     aux = c * wi,r - s * wj,r
6     wj,r = s * wi,r + c * wj,r
7     wi,r = aux
8   fim
9 fim
```

---

A rotação de Givens será referenciada por  $Q(i, j, \theta)$  nos próximos pseudocódigos.

**Implementação em C++** Tal algoritmo foi implementado em C++ da seguinte forma: duas versões da função, uma recebendo como parâmetro, uma matriz e um vetor, e outra recebendo duas matrizes, além de outros parâmetros, evidentemente, que são iguais em ambos os casos: os índices das linhas i e j, o cosseno e o seno (c e s) utilizado na função.

Para a função que recebe como parâmetro uma matriz e um vetor:

---

```
1 void Givens(vector<vector<double>> &matriz, vector<double> &vetor, int i, int
   j, double seno, double cosseno)
2 {
3     vector<vector<double>> temp = matriz;
4     vector<double> tempvec = vetor;
5     for (int k = 0; k < matriz[i].size(); k++) {
6         matriz[i][k] = cosseno * temp[i][k] - seno * temp[j][k];
7     }
8     vetor[i] = cosseno * tempvec[i] - seno * tempvec[j];
9
10    for (int k = 0; k < matriz[i].size(); k++) {
11        matriz[j][k] = cosseno * temp[j][k] + seno * temp[i][k];
12    }
13    vetor[j] = cosseno * tempvec[j] + seno * tempvec[i];
14 }
```

---

Neste caso, perceba que na linha 6 do código, as alterações da linha 5 do algoritmo é aplicada na matriz. Em seguida, na linha 8, ela é aplicada no vetor. O mesmo vale para as alterações da linha 6 do algoritmo, que é aplicada nas linhas 11 (na matriz) e 13 (no vetor).

E para a função que recebe como parâmetro duas matrizes:

---

```
1 void Givens(vector<vector<double>> &matriz, vector<vector<double>> &matriz2,
   int i, int j, double seno, double cosseno)
2 {
3     vector<vector<double>> temp = matriz;
4     vector<vector<double>> tempmat = matriz2;
5
6     for (int k = 0; k < matriz[i].size(); k++)
7         matriz[i][k] = cosseno * temp[i][k] - seno * temp[j][k];
8     for (int k = 0; k < matriz2[i].size(); k++)
9         matriz2[i][k] = cosseno * tempmat[i][k] - seno * tempmat[j][k];
10
11    for (int k = 0; k < matriz[i].size(); k++)
12        matriz[j][k] = cosseno * temp[j][k] + seno * temp[i][k];
13    for (int k = 0; k < matriz2[i].size(); k++)
14        matriz2[j][k] = cosseno * tempmat[j][k] + seno * tempmat[i][k];
15 }
```

---

Aqui, as alterações mencionadas da linha 5 e 6 do código são aplicadas nas linhas 7 e 12, 9 e 14, respectivamente, em ambas as matrizes. Sendo a *matriz2* a matriz solução.

## 2.2 Fatoração QR

O código utilizado para a implementação da Fatoração QR seguiu o pseudocódigo fornecido pelo enunciado do Exercício-Programa, reproduzido a seguir o pseudocódigo referente à solução de Sistemas sobredeterminados:

---

**Entrada:** matriz W, vetor b  
**Saída:** vetor x

```
1 início
2   para  $k=1$  até  $m$  faça
3     para  $j=n$  até  $k+1$  com passo -1 faça
4        $i = j - 1$ 
5       se  $w_{j,k} \neq 0$  então
6         aplique  $Q(i, j, \theta)$  a matriz W (com  $c$  e  $s$  definidos por  $w_{i,k}$  e  $w_{j,k}$ ) e
          ao vetor b
7       fim
8     fim
9   fim
10  para  $k=m$  até 1 com passo -1 faça
11     $x_k = (b_k - \sum_{j=k+1}^m w_{k,j}x_j)/w_{k,k}$ 
12  fim
13 fim
```

---

No código acima, percebe-se que a saída é a solução do sistema, dada pelo vetor x.

Já o algoritmo utilizado para a solução de Sistemas simultâneos é:

---

**Entrada:** matriz W, matriz A  
**Saída:** matriz H

```

1 início
2   para k = 1 até p faça
3     para j=n até k+1 com passo -1 faça
4       i = j - 1
5       se  $w_{j,k} \neq 0$  então
6         aplique Q(i,j, $\theta$ ) à matriz W (com c e s definidos por  $w_{i,k}$  e  $w_{j,k}$  e à
          matriz A
7       fim
8     fim
9   fim
10  para k=p até 1 com passo -1 faça
11    para j=1 até m faça
12       $h_{k,j} = (a_{k,j} - \sum_{i=k+1}^p w_{k,i} h_{i,j}) / w_{k,k}$ 
13    fim
14  fim
15 fim

```

---

No código acima, a matriz H possui as soluções dos sistemas simultâneos.

**Implementação em C++** A implementação da Fatoração QR em C++ deu-se de maneira similar à Rotação de Givens, com uma função recebendo como parâmetro um vetor e uma matriz (para sistemas simples) e outra função recebendo como parâmetro duas matrizes (para sistemas simultâneos).

---

```

1 vector<double> solucaoSistemas(vector<vector<double> >& matriz, vector<double>&
   vetor)
2 {
3   // parte 1: realiza a decomposicao QR na matriz e no vetor
4   int tamColMatriz = matriz[0].size();
5   int tamLinMatriz = matriz.size();
6   double cosseno, seno, temp;
7   vector<vector<double>> temporario;
8   vector<double> tempvec;
9
10  for (int k = 0; k < tamColMatriz; k++) {
11    for (int j = tamLinMatriz - 1; j > k; j--) {
12      int i = j - 1;
13      if (fabs(matriz[j][k]) > ZERO) {
14
15        if (fabs(matriz[i][k]) > fabs(matriz[j][k])) {
16          temp = -matriz[j][k] / matriz[i][k];
17          cosseno = 1 / sqrt(1 + temp * temp);
18          seno = cosseno * temp;

```

```

19         //Givens(matriz, vetor, i, j, seno, cosseno);
20
21         temporario = matriz;
22         tempvec = vetor;
23         for (int k = 0; k < tamColMatriz; k++) {
24             matriz[i][k] = cosseno * temporario[i][k] - seno *
                temporario[j][k];
25         }
26
27         vetor[i] = cosseno * tempvec[i] - seno * tempvec[j];
28
29         for (int k = 0; k < tamColMatriz; k++) {
30             matriz[j][k] = cosseno * temporario[j][k] + seno *
                temporario[i][k];
31         }
32
33         vetor[j] = cosseno * tempvec[j] + seno * tempvec[i];
34     }
35     else {
36         temp = -matriz[i][k] / matriz[j][k];
37         seno = 1 / sqrt(1 + temp * temp);
38         cosseno = seno * temp;
39         //Givens(matriz, vetor, i, j, seno, cosseno);
40
41         temporario = matriz;
42         tempvec = vetor;
43         for (int k = 0; k < tamColMatriz; k++) {
44             matriz[i][k] = cosseno * temporario[i][k] - seno *
                temporario[j][k];
45         }
46
47         vetor[i] = cosseno * tempvec[i] - seno * tempvec[j];
48
49         for (int k = 0; k < tamColMatriz; k++) {
50             matriz[j][k] = cosseno * temporario[j][k] + seno *
                temporario[i][k];
51         }
52
53         vetor[j] = cosseno * tempvec[j] + seno * tempvec[i];
54     }
55 }
56 }
57 }
58
59 // parte 2: resolve o sistema
60 vector<double> x;

```

```

61     x.resize(tamColMatriz);
62
63     for (int k = matriz[0].size() - 1; k >= 0; k--) {
64         if (k == matriz[0].size() - 1) { x[k] = vetor[k] / matriz[k][k]; }
65         else {
66             double soma = 0;
67             for (int j = k + 1; j <= matriz[0].size() - 1; j++) {
68                 soma += matriz[k][j] * x[j] / matriz[k][k];
69             }
70             x[k] = vetor[k] / matriz[k][k] - soma;
71         }
72     }
73
74     return x;
75 }

```

---

Dentro dessa função, percebe-se que ocorrem diversas rotações de Givens, tanto quanto na matriz quanto no vetor. E retornam o vetor x, contendo a solução do sistema. Percebe-se que a parte 1 refere-se aos dois primeiros *loops* do algoritmo, enquanto que a parte 2 se refere aos dois últimos *loops* do algoritmo.

---

```

1  vector<vector<double>> solucaoSimultaneos(vector<vector<double>>& W,
    vector<vector<double>>& A) {
2      // Realiza a decomposicao QR
3      int tamColW = W[0].size();
4      int tamLinW = W.size();
5      int tamCol2 = A[0].size();
6      double cosseno, seno, temp;
7      double temporario;
8      double tempmat;
9
10     for (int k = 0; k < tamColW; k++){
11         for (int j = tamLinW - 1; j > k; j--){
12             int i = j - 1;
13             if (fabs(W[j][k]) > ZERO){
14
15                 if (fabs(W[i][k]) > fabs(W[j][k])){
16                     temp = -W[j][k] / W[i][k];
17                     cosseno = 1 / sqrt(1 + temp * temp);
18                     seno = cosseno * temp;
19                     //Givens(W, A, i, j, seno, cosseno);
20
21                     for (int k = 0; k < tamColW; k++){
22                         temporario = cosseno * W[i][k] - seno * W[j][k];
23                         W[j][k] = cosseno * W[j][k] + seno * W[i][k];
24                         W[i][k] = temporario;
25                     }

```



```

26
27         for (int k = 0; k < tamCol2; k++){
28             tempmat = cos seno * A[i][k] - seno * A[j][k];
29             A[j][k] = cos seno * A[j][k] + seno * A[i][k];
30             A[i][k] = tempmat;
31         }
32     }
33     else {
34         temp = -W[i][k] / W[j][k];
35         seno = 1 / sqrt(1 + temp * temp);
36         cos seno = seno * temp;
37         //Givens(W, A, i, j, seno, cos seno);
38
39         for (int k = 0; k < tamColW; k++){
40             temporario = cos seno * W[i][k] - seno * W[j][k];
41             W[j][k] = cos seno * W[j][k] + seno * W[i][k];
42             W[i][k] = temporario;
43         }
44
45         for (int k = 0; k < tamCol2; k++){
46             tempmat = cos seno * A[i][k] - seno * A[j][k];
47             A[j][k] = cos seno * A[j][k] + seno * A[i][k];
48             A[i][k] = tempmat;
49         }
50     }
51 }
52 }
53 }
54
55 // Resolve o sistema
56 int tamLinA = A.size();
57 int tamColA = A[0].size();
58
59 vector<vector<double>> H(tamColW, vector<double>(tamColA));
60
61 int k = tamColW - 1;
62
63 for (int j = 0; j < tamColA; j++) // loop proprio para o caso em que k =
    W[0].size() - 1
64     H[k][j] = A[k][j] / W[k][k];
65
66 for (int k = tamColW - 2; k >= 0; k--){ // loop para os demais casos
67     for (int j = 0; j < tamColA; j++){
68         double soma = 0;
69         for (int i = k + 1; i < tamColW; i++){
70             soma += W[k][i] * H[i][j] / W[k][k];

```

```

71         }
72         H[k][j] = A[k][j] / W[k][k] - soma;
73     }
74 }
75
76 return H;
77 }

```

---

A implementação dessa função já é mais complexa, segue a mesma lógica do pseudocódigo, mas dessa vez a complexidade aqui deve-se ao fato de que todas as alterações devem ser feitas em ambas as matrizes, que podem ser de tamanhos diferentes, o que exige que essas alterações sejam feitas em cada matriz em *loops* diferentes.

No código da Fatoração QR, percebe-se que, ao invés de chamar a função *Givens*, esta foi implementada diretamente nas funções que exigiam a utilização de Rotação de Givens devido ao fato de que, dessa forma, o código teve uma diferença de tempo de execução bastante notória. Neste mesmo propósito de otimização, as quantidades de linhas e colunas das matrizes foram salvas em variáveis ao invés de chamar a função *size()* diretamente.

### 2.2.1 Primeira Tarefa

Com os códigos da Fatoração QR e Rotação de Givens implementados, já é possível realizar a Primeira Tarefa. Sendo necessário apenas uma *main.cpp* para a construção da matriz.

**Item A** Neste item, foi utilizado o seguinte código como *main.cpp* para construção das matrizes e impressão dos resultados:

---

```

1  int main() {
2
3      int n = 64;
4      int m = 64;
5      vector<vector<double>> W (n, vector<double> (m));
6      vector<double> b(n);
7
8      for (int i = 0; i < n; i++) {
9          for (int j = 0; j < m; j++){
10             W[i][i] = 2;
11             if(abs(i - j) == 1)
12             {
13                 W[i][j] = 1;
14             }
15             if(abs(i - j) > 1)
16             {
17                 W[i][j] = 0;

```

```

18         }
19     }
20 }
21
22 for(int k = 0; k < n; k++){
23     b[k] = 1;
24 }
25
26 vector<double> x = solucaoSistemas(W, b);
27
28 cout << "Matriz R:" << endl;
29 for(int i = 0; i < n; i++){
30     cout << "|" << " ";
31     for(int j = 0; j < m; j++){
32         cout << W[i][j] << " ";
33     }
34     cout << "|" << endl;
35 }
36
37 cout << "Vetor b modificado:" << endl;
38
39 for(int i = 0; i < n; i++){
40     cout << b[i] << " ";
41 }
42
43 cout << endl;
44 cout << "Vetor x:" << endl;
45 for(int i = 0; i < n; i++){
46     cout << x[i] << " ";
47 }
48
49 cout << endl;
50
51 return 0;
52 }

```

---

E os resultados são:

[illegible]



```

21
22     for(int k = 0; k < b.size(); k++){
23         b[k] = k + 1;
24     }
25
26     printf("Matriz Inicial: \n");
27
28     for (int i = 0; i < W.size(); i++){
29         cout << "|";
30         for (int j = 0; j < W[0].size(); j++){
31             cout << " " << W[i][j] << " ";
32         }
33         cout << "|" << endl;
34     }
35
36     printf("Vetor B: \n");
37
38     for(int k = 0; k < b.size(); k++){
39         cout << b[k] << " ";
40     }
41
42     cout << endl;
43
44     x = solucaoSistemas(W, b);
45
46     printf("Matriz R: \n");
47
48     for (int i = 0; i < W.size(); i++){
49         cout << "|";
50         for (int j = 0; j < W[0].size(); j++){
51             cout << " " << W[i][j] << " ";
52         }
53         cout << "|" << endl;
54     }
55
56     printf("Vetor B modificado: \n");
57
58     for(int k = 0; k < b.size(); k++){
59         cout << b[k] << " ";
60     }
61
62     cout << endl;
63
64     printf("Vetor X: \n");
65
66     for(int i = 0; i < x.size(); i++){

```



```

17         if(abs(i - j) > 1)
18         {
19             W[i][j] = 0;
20         }
21     }
22 }
23
24 for (int i = 1; i < (A.size()+1); i++){
25     for (int j = 0; j < W[0].size(); j++){
26         if(j == 0){
27             A[i-1][j] = 1;
28         }
29         if (j == 1)
30         {
31             A[i-1][j] = i;
32         }
33         if(j == 2)
34         {
35             A[i-1][j] = 2*i - 1;
36         }
37     }
38 }
39
40 printf("Matriz Inicial: \n");
41
42 for (int i = 0; i < W.size(); i++){
43     cout << "|";
44     for (int j = 0; j < W[0].size(); j++){
45         cout << " " << W[i][j] << " ";
46     }
47     cout << "|" << endl;
48 }
49
50 printf("Matriz A: \n");
51
52 for (int i = 0; i < A.size(); i++){
53     cout << "|";
54     for (int j = 0; j < A[0].size(); j++){
55         cout << " " << A[i][j] << " ";
56     }
57     cout << "|" << endl;
58 }
59
60 cout << endl;
61
62 H = solucaoSimultaneos(W, A);

```



```

63
64     printf("Matriz final: \n");
65
66     for (int i = 0; i < W.size(); i++){
67         cout << "|";
68         for (int j = 0; j < W[0].size(); j++){
69             cout << " " << W[i][j] << " ";
70         }
71         cout << "|" << endl;
72     }
73
74     printf("Matriz A modificado: \n");
75
76     for (int i = 0; i < A.size(); i++){
77         cout << "|";
78         for (int j = 0; j < A[0].size(); j++){
79             cout << " " << A[i][j] << " ";
80         }
81         cout << "|" << endl;
82     }
83
84     cout << endl;
85
86     printf("Matriz H: \n");
87
88     for (int i = 0; i < H.size(); i++){
89         cout << "|";
90         for (int j = 0; j < H[0].size(); j++){
91             cout << " " << H[i][j] << " ";
92         }
93         cout << "|" << endl;
94     }
95
96     cout << endl;
97
98     return 0;
99 }

```

---

Perceba que neste caso, foram utilizadas 3 matrizes e um valor p, que também é passado como parâmetro para a função. Os resultados obtidos deste item, foram:

17



```

0.0923077 6 11.9077 |
0.4 1.01252e-13 -0.4 |
0.107692 7 13.8923 |
0.384615 1.03029e-13 -0.384615 |
0.123077 8 15.8769 |
0.369231 9.9476e-14 -0.369231 |
0.138462 9 17.8615 |
0.153846 1.06581e-13 -0.353846 |
0.153846 10 19.8462 |
0.338462 1.1724e-13 -0.338462 |
0.169231 11 21.8308 |
0.123077 1.16003e-13 -0.323077 |
0.184615 12 23.8154 |
0.307692 1.49214e-13 -0.307692 |
0.2 13 25.8 |
0.292308 1.59872e-13 -0.292308 |
0.215385 14 27.7846 |
0.276923 1.59872e-13 -0.276923 |
0.230769 15 29.7692 |
0.261538 1.56319e-13 -0.261538 |
0.246154 16 31.7538 |
0.246154 1.49214e-13 -0.246154 |
0.261538 17 33.7385 |
0.230769 1.36003e-13 -0.230769 |
0.276923 18 35.7231 |
0.215385 1.20792e-13 -0.215385 |
0.292308 19 37.7077 |
0.2 1.13687e-13 -0.2 |
0.307692 20 39.6923 |
0.184615 1.13687e-13 -0.184615 |
0.323077 21 41.6769 |
0.169231 1.13687e-13 -0.169231 |
0.338462 22 43.6615 |
0.153846 9.9476e-14 -0.153846 |
0.353846 23 45.6462 |
0.138462 7.01597e-14 -0.138462 |
0.369231 24 47.6308 |
0.123077 6.39488e-14 -0.123077 |
0.384615 25 49.6154 |
0.107692 5.55271e-14 -0.107692 |
0.4 26 51.6 |
0.0923077 0 -0.0923077 |
0.415385 27 53.5846 |
0.0769231 -2.13163e-14 -0.0769231 |
0.430769 28 55.5692 |
0.0615385 -3.55271e-14 -0.0615385 |
0.446154 29 57.5538 |
0.4461538 -2.84217e-14 -0.0461538 |
0.461538 30 59.5385 |
0.0307692 -2.13163e-14 -0.0307692 |
0.476923 31 61.5231 |
0.0153846 -1.42109e-14 -0.0153846 |
0.492308 32 63.5077 |

```

**Item D** A matriz  $W$  aqui construída é idêntica à matriz  $W$  construída no item b, o que difere, no entanto, é a saída e a matriz  $A$ , ao invés de vetor  $b$ . Tudo isso foi construído da seguinte forma:

```

1 int main(){
2
3     int m = 3;
4     int n = 20;
5     int p = 17;
6     vector<vector<double>> W (n, vector<double> (p));
7     vector<vector<double>> A(n, vector<double> (m));
8     vector<vector<double>> H;
9
10    for (int i = 0; i < W.size(); i++) {
11        for (int j = 0; j < W[0].size(); j++){
12            if((i - j) <= 4 && (i - j) >= -4){
13                a = (i + j + 1);
14                W[i][j] = 1/a;
15            }
16            else {
17                W[i][j] = 0;
18            }
19        }
20    }
21
22    for (int i = 1; i < (A.size()+1); i++){
23        for (int j = 0; j < W[0].size(); j++){

```

```

24         if(j == 0){
25             A[i-1][j] = 1;
26         }
27         if (j == 1)
28         {
29             A[i-1][j] = i;
30         }
31         if(j == 2)
32         {
33             A[i-1][j] = 2*i - 1;
34         }
35     }
36 }
37
38 printf("Matriz Inicial: \n");
39
40 for (int i = 0; i < W.size(); i++){
41     cout << "|";
42     for (int j = 0; j < W[0].size(); j++){
43         cout << " " << W[i][j] << " ";
44     }
45     cout << "|" << endl;
46 }
47
48 printf("Matriz A: \n");
49
50 for (int i = 0; i < A.size(); i++){
51     cout << "|";
52     for (int j = 0; j < A[0].size(); j++){
53         cout << " " << A[i][j] << " ";
54     }
55     cout << "|" << endl;
56 }
57
58 cout << endl;
59
60 H = solucaoSimultaneos(W, A);
61
62 printf("Matriz final: \n");
63
64 for (int i = 0; i < W.size(); i++){
65     cout << "|";
66     for (int j = 0; j < W[0].size(); j++){
67         cout << " " << W[i][j] << " ";
68     }
69     cout << "|" << endl;

```

```

70     }
71
72     printf("Matriz A modificado: \n");
73
74     for (int i = 0; i < A.size(); i++){
75         cout << "|";
76         for (int j = 0; j < A[0].size(); j++){
77             cout << " " << A[i][j] << " ";
78         }
79         cout << "|" << endl;
80     }
81
82     cout << endl;
83
84     printf("Matriz H: \n");
85
86     for (int i = 0; i < H.size(); i++){
87         cout << "|";
88         for (int j = 0; j < H[0].size(); j++){
89             cout << " " << H[i][j] << " ";
90         }
91         cout << "|" << endl;
92     }
93
94     cout << endl;
95
96     return 0;
97 }

```

---

Os resultados deste item d foram:

```
Matriz final:
1.2098 0.68882 0.492814 0.385411 0.317759 0.132975 0.0663076 0.0325624 0.0127167 0 0 0 0 0 0 0 0 |
0 0.192193 0.18681 0.171496 0.156294 0.282641 0.450937 0.160785 0.073883 0.0492967 0 0 0 0 0 0 0 |
0 -9.28437e-19 0.1131 0.103167 0.0943648 0.0960241 0.0885834 0.0763883 0.064319 0.0536576 0.0577893 0 0 0 0 0 0 |
0 -5.11568e-19 1.47543e-19 0.0912401 0.0839478 0.0799673 0.0770848 0.0702668 0.0624793 0.0549659 0.0544816 0.0524406 0 0 0 0 0 |
0 -1.13151e-19 9.0726e-20 -4.71823e-20 0.0770187 0.0719042 0.0699867 0.0652932 0.0595686 0.0537727 0.0519213 0.0497261 0.0475599 0 0 0 |
0 -3.60785e-20 3.66412e-20 -3.36617e-19 2.76996e-21 0.0975697 0.0347847 0.0110038 0.00353221 -0.00136402 0.026944 0.0110092 0.0086099 0.0297075 0 0 0 |
4.9985e-18 -1.25623e-18 4.85951e-19 -3.0748e-19 0 -0.115364 -0.0776697 -0.054035 -0.037728 -0.0350032 -0.0358996 -0.0361527 -0.0358431 -0.0203958 0 0 |
1.09933e-18 1.46957e-20 9.20791e-20 -4.46578e-19 0 -3.82883e-22 0.100965 0.0752633 0.0566339 0.0442761 0.0408342 0.0390357 0.0376865 0.0263693 0.0193069 0 |
0 -2.5704e-18 8.23112e-19 -4.89698e-19 -1.807e-19 0 6.48358e-20 2.21752e-18 -0.08749 -0.0679983 -0.0523955 -0.0464643 -0.0430883 -0.0407098 -0.030921 -0.0239963 -0.0187683 |
0 2.38219e-18 -7.38633e-19 4.37082e-19 -0.70841e-20 0 6.74115e-20 -1.9223e-18 0 0.0768958 0.0597137 0.0522218 0.0476065 0.0444847 0.0351428 0.0281051 0.022634 |
0 -1.01774e-18 2.92031e-19 -2.4928e-19 3.64921e-20 0 -3.19065e-21 2.95493e-18 0 1.78196e-18 0.0463745 0.0474861 0.0467244 0.0454391 0.0545698 0.0477853 0.039753 |
0 -3.56694e-19 1.03371e-19 -7.09018e-20 1.0542e-20 0 -1.53246e-21 5.47978e-19 0 3.83697e-19 -7.54849e-20 0.0374139 0.0363747 0.0352561 0.0361045 0.0351292 0.0327808 |
0 -1.81501e-19 5.33131e-20 -3.19406e-20 4.42788e-21 0 -9.98201e-22 1.49746e-19 0 1.32412e-19 -3.98204e-20 0.73515e-21 0.0345528 0.0334724 0.032967 0.0325316 0.0312391 |
0 -1.03865e-19 3.09361e-20 1.65798e-20 2.05232e-21 0 -7.00595e-22 3.07722e-20 0 4.89186e-20 -2.32482e-20 -8.80621e-22 -4.50488e-21 0.0322793 0.0314058 0.0310034 0.0300726 |
0 -2.70242e-19 8.31081e-20 -1.46214e-19 1.09394e-20 0 -9.08939e-22 3.00378e-18 0 1.62581e-18 -7.37333e-21 2.34046e-19 -2.65104e-21 0 0.0361266 0.0246044 0.0188544 |
0 -1.72246e-18 4.36347e-19 -1.52568e-19 1.00965e-19 0 -1.18323e-21 -1.90612e-18 0 -1.03237e-18 -5.13429e-19 -1.5115e-19 -2.23173e-21 -0 -4.03829e-22 0.0410288 0.033789 |
0 8.49059e-19 -6.17522e-20 9.09256e-20 -2.50802e-19 0 -7.99652e-22 -1.71067e-18 0 1.86288e-18 3.88428e-19 -3.07837e-19 -1.74217e-19 0 -4.33071e-20 8.4049e-19 0.0281374 |
0 -0.32342e-19 2.98974e-19 -2.84119e-19 -4.29177e-20 0 -1.48437e-21 3.85979e-18 0 3.27821e-18 -6.07688e-20 2.27086e-19 -7.62162e-20 0 -1.94287e-20 2.26997e-18 -1.73472e-18 |
0 2.48489e-18 -7.3386e-19 3.67908e-19 -2.50195e-20 0 2.12016e-21 0 -1.6375e-18 5.75888e-19 1.0399e-19 1.04045e-19 0 2.60489e-20 3.03103e-18 0 |
0 -5.04842e-21 2.52908e-21 -1.26559e-21 -1.10494e-19 0 5.46189e-20 0 0 3.4873e-18 0 -1.09719e-19 -1.32542e-21 0 -3.31463e-22 0 0 |
Matriz A modificado:
1.88737 4.11292 6.37848 |
1.51558 8.07637 14.6372 |
1.03653 7.07216 13.1078 |
1.01139 8.03167 15.052 |
1.00599 9.03195 17.0579 |
0.588219 5.14367 9.69912 |
-0.928738 -9.7095 -18.4903 |
0.987681 11.4485 21.9093 |
-1.02702 -12.741 -24.455 |
1.07552 14.0013 26.927 |
1.24792 18.1887 35.1295 |
1.02181 16.2349 31.4479 |
1.00634 17.0698 33.1333 |
1.00307 18.0411 35.0792 |
0.792518 14.5672 28.3419 |
0.964127 19.054 37.1438 |
0.410258 8.3962 16.3822 |
-0.0848568 -1.87506 -3.66526 |
0.530541 10.9695 21.4084 |
-0.0329633 -0.749369 -1.46577 |
Matriz H:
2.88155 56.3578 109.834 |
-1.83376 -45.8748 -89.9159 |
-1.51399 -43.489 -85.4641 |
-1.52191 -48.5765 -95.6312 |
-0.453795 -30.1402 -59.8266 |
5.8507 89.8121 173.768 |
3.42193 48.7136 94.0054 |
3.65656 59.2392 114.822 |
1.20368 11.4464 21.689 |
6.12534 109.163 212.2 |
-2.47971 -72.8728 -143.266 |
-1.07793 -54.3629 -107.248 |
1.09933e-18 1.46957e-20 9.20791e-20 -4.46578e-19 0 -3.82883e-22 0.100965 0.0752633 0.0566339 0.0442761 0.0408342 0.0390357 0.0376865 0.0263693 0.0193069 0 |
0 2.38219e-18 -7.38633e-19 4.37082e-19 -0.70841e-20 0 6.74115e-20 -1.9223e-18 0 0.0768958 0.0597137 0.0522218 0.0476065 0.0444847 0.0351428 0.0281051 0.022634 |
0 -1.01774e-18 2.92031e-19 -2.4928e-19 3.64921e-20 0 -3.19065e-21 2.95493e-18 0 1.78196e-18 0.0463745 0.0474861 0.0467244 0.0454391 0.0545698 0.0477853 0.039753 |
0 -3.56694e-19 1.03371e-19 -7.09018e-20 1.0542e-20 0 -1.53246e-21 5.47978e-19 0 3.83697e-19 -7.54849e-20 0.0374139 0.0363747 0.0352561 0.0361045 0.0351292 0.0327808 |
0 -1.81501e-19 5.33131e-20 -3.19406e-20 4.42788e-21 0 -9.98201e-22 1.49746e-19 0 1.32412e-19 -3.98204e-20 0.73515e-21 0.0345528 0.0334724 0.032967 0.0325316 0.0312391 |
0 -1.03865e-19 3.09361e-20 1.65798e-20 2.05232e-21 0 -7.00595e-22 3.07722e-20 0 4.89186e-20 -2.32482e-20 -8.80621e-22 -4.50488e-21 0.0322793 0.0314058 0.0310034 0.0300726 |
0 -2.70242e-19 8.31081e-20 -1.46214e-19 1.09394e-20 0 -9.08939e-22 3.00378e-18 0 1.62581e-18 -7.37333e-21 2.34046e-19 -2.65104e-21 0 0.0361266 0.0246044 0.0188544 |
0 -1.72246e-18 4.36347e-19 -1.52568e-19 1.00965e-19 0 -1.18323e-21 -1.90612e-18 0 -1.03237e-18 -5.13429e-19 -1.5115e-19 -2.23173e-21 -0 -4.03829e-22 0.0410288 0.033789 |
0 8.49059e-19 -6.17522e-20 9.09256e-20 -2.50802e-19 0 -7.99652e-22 -1.71067e-18 0 1.86288e-18 3.88428e-19 -3.07837e-19 -1.74217e-19 0 -4.33071e-20 8.4049e-19 0.0281374 |
0 -0.32342e-19 2.98974e-19 -2.84119e-19 -4.29177e-20 0 -1.48437e-21 3.85979e-18 0 3.27821e-18 -6.07688e-20 2.27086e-19 -7.62162e-20 0 -1.94287e-20 2.26997e-18 -1.73472e-18 |
0 2.48489e-18 -7.3386e-19 3.67908e-19 -2.50195e-20 0 2.12016e-21 0 -1.6375e-18 5.75888e-19 1.0399e-19 1.04045e-19 0 2.60489e-20 3.03103e-18 0 |
0 -5.04842e-21 2.52908e-21 -1.26559e-21 -1.10494e-19 0 5.46189e-20 0 0 3.4873e-18 0 -1.09719e-19 -1.32542e-21 0 -3.31463e-22 0 0 |
Matriz A modificado:
1.88737 4.11292 6.37848 |
1.51558 8.07637 14.6372 |
1.03653 7.07216 13.1078 |
1.01139 8.03167 15.052 |
1.00599 9.03195 17.0579 |
0.588219 5.14367 9.69912 |
-0.928738 -9.7095 -18.4903 |
0.987681 11.4485 21.9093 |
-1.02702 -12.741 -24.455 |
1.07552 14.0013 26.927 |
1.24792 18.1887 35.1295 |
1.02181 16.2349 31.4479 |
1.00634 17.0698 33.1333 |
1.00307 18.0411 35.0792 |
0.792518 14.5672 28.3419 |
0.964127 19.054 37.1438 |
0.410258 8.3962 16.3822 |
-0.0848568 -1.87506 -3.66526 |
0.530541 10.9695 21.4084 |
-0.0329633 -0.749369 -1.46577 |
Matriz H:
2.88155 56.3578 109.834 |
-1.83376 -45.8748 -89.9159 |
-1.51399 -43.489 -85.4641 |
-1.52191 -48.5765 -95.6312 |
-0.453795 -30.1402 -59.8266 |
5.8507 89.8121 173.768 |
3.42193 48.7136 94.0054 |
3.65656 59.2392 114.822 |
1.20368 11.4464 21.689 |
6.12534 109.163 212.2 |
-2.47971 -72.8728 -143.266 |
-1.07793 -54.3629 -107.248 |
-1.2039 -51.4444 -101.085 |
0.120415 -25.0146 -50.158 |
6.50159 98.5719 190.642 |
11.4911 218.659 425.827 |
14.5805 298.4 582.22 |
```

## 2.3 Fatoração Não Negativa

A fatoração não negativa é a última das funções prévias pedidas para a resolução do Exercício-Programa, sua implementação foi de grande complexidade e seguiu o algoritmo: O objetivo desse algoritmo é fatorar a matriz A em duas matrizes menores W e H com

---

**Entrada:** Matriz A não negativa, valor p  
**Saída:** Matriz W

---

```
1 início
2   Inicialize randomicamente a matriz W com valores positivos
3   Armazene uma cópia da matriz A
4   para  $ERRO > 10^{-5}$  ou  $itmax < 100$  faça
5       Normalize W tal que a norma de cada uma de suas colunas seja 1
           ( $w_{i,j} = w_{i,j}/s_j$ , com  $s_j = \sqrt{\sum_{i=1}^n w_{i,j}^2}$ ) Resolva o problema de mínimos
           quadrados  $WH = A$ , determinando a matriz H (são  $m$  sistemas
           simultâneos! Cuidado, pois A é modificada no processo de solução. Na
           iteração seguinte deve-se usar a matriz A original novamente. Por isso
           armazena-se uma cópia de A!) Redefina H, com  $h_{i,j} = \max\{0, h_{i,j}\}$ 
           Compute a matriz  $A^t$  (transposta da matriz A original) Resolva os
           problemas de mínimos quadrados  $H^t W^t = A^t$ , determinando a nova matriz
            $W^t$ . (são  $n$  sistemas simultâneos!) Compute a matriz W (transposta de
            $W^t$ ) Redefina W, com  $w_{i,j} = \max\{0, w_{i,j}\}$ 
6   fim
7 fim
```

---

elementos restritivamente positivos, a matriz W contendo "a base" da matriz A e a matriz H contendo os "pesos" de cada elemento da matriz W na matriz A. Essa função implementada requer um treinamento para realizar melhores aproximações das matrizes.

Essa função foi implementada em C++ da seguinte forma:

---

```
1 vector<vector<double>> NMF(vector<vector<double>> &matriz, int p,
   vector<vector<double>> &saidaH) {
2     vector<vector<double>> A;
3     vector<vector<double>> W(matriz.size(), vector<double>(p));
4     vector<vector<double>> H;
5     vector<vector<double>> At; // transposta de A
6     vector<vector<double>> Ht; // transposta de H
7     vector<vector<double>> Wt;
8
9     int tamLinW = W.size();
10    int tamColW = W[0].size();
11
12    for (int i = 0; i < tamLinW; i++) {
13        for (int j = 0; j < tamColW; j++) {
```



```

14         W[i][j] = rand();
15         W[i][j]++;
16     }
17 }
18
19 double erro = 1;
20
21 for (int itmax = 0; erro > EPS && itmax < 100; itmax++) {
22
23     A = matriz;
24
25     double s = 0;
26     for (int j = 0; j < tamColW; j++) {
27         double soma = 0;
28         for (int i = 0; i < tamLinW; i++) {
29             soma += W[i][j] * W[i][j];
30         }
31         s = sqrt(soma);
32         for (int i = 0; i < tamLinW; i++) {
33             W[i][j] = W[i][j] / s;
34         }
35     }
36
37     H = solucaoSimultaneos(W, A);
38
39     int tamLinH = H.size();
40     int tamColH = H[0].size();
41
42     for (int i = 0; i < tamLinH; i++) {
43         for (int j = 0; j < tamColH; j++) {
44             if (H[i][j] < EPS)
45                 H[i][j] = 0;
46         }
47     }
48
49     At = MTranspose(matriz);
50     Ht = MTranspose(H);
51
52     Wt = solucaoSimultaneos(Ht, At);
53
54     int tamLinWt = Wt.size();
55     int tamColWt = Wt[0].size();
56     // Redefina W, com w(i,j) = max{0, w(i,j)}
57     for (int i = 0; i < tamLinWt; i++) {
58         for (int j = 0; j < tamColWt; j++) {
59             if (Wt[i][j] < EPS)

```

```

60         Wt[i][j] = 0;
61     }
62 }
63 W = MTranspose(Wt);
64
65
66 erro = 0;
67 vector<vector<double>> Aerro; // Aerro sera a matris WH
68 Aerro = MMultiplication(W, H);
69 int tamLinAerro = Aerro.size();
70 int tamColAerro = Aerro[0].size();
71
72 for (int i = 0; i < tamLinAerro; i++) {
73     for (int j = 0; j < tamColAerro; j++) {
74         double provisorio;
75         provisorio = matriz[i][j] - Aerro[i][j];
76         erro += provisorio * provisorio;
77     }
78 }
79 }
80
81 saidaH = H;
82 return W;
83 }

```

As linhas 12 a 17 é um loop para iniciar a matriz com valores positivos e aleatórios. O loop das linhas 26 a 35 é para normalizar a matriz W através de  $w_{i,j} = w_{i,j}/s_j$ . O loop de 42 a 47 redefine a matriz H apenas com valores positivos. Depois disso, há o cálculo das matrizes transpostas para a resolução de  $H^t W^t = A^t$ , obtendo a matriz  $W^t$  (linhas 49 a 52), então redefinimos  $W^t$  para valores positivos e então obtemos a matriz W através da transposta de  $W^t$ . As linhas seguintes são referentes a obtenção do erro.

Importante notar que no arquivo *fatoracaoNaoNeg.cpp* existem duas versões de função NMF, a primeira pede como parâmetro a matriz a ser fatorada, o valor p e também a matriz que receberá H após a fatoração (o código dessa função é o código colocado acima), a segunda versão possui basicamente o mesmo funcionamento, apenas diverge que ela não recebe como parâmetro a matriz de saída H, para ser utilizada em casos de que a matriz H não é necessária.

### 2.3.1 Segunda Tarefa

Nessa tarefa, utilizamos, evidentemente o código da Fatoração Não Negativa (*fatoracaoNaoNeg.cpp*) para a elaboração dos resultados. Além disso, o código da *main.cpp* utilizado para construção e impressão das matrizes, foi:

```

1 int main() {
2

```

```

3     int m = 3;
4     int n = 3;
5     vector<vector<double>> W(m, vector<double>(n));
6     vector<double> b(n);
7     vector<double> x(n);
8     int k = 0;
9     double a;
10
11     W[0] = {0.3, 0.6, 0};
12     W[1] = {0.5, 0, 1};
13     W[2] = {0.4, 0.8, 0};
14
15     printf("Matriz Inicial: \n");
16
17     vector<vector<double>> A;
18     vector<vector<double>> B;
19
20     for (int i = 0; i < W.size(); i++) {
21         cout << "|";
22         for (int j = 0; j < W[0].size(); j++) {
23             cout << " " << W[i][j] << " ";
24         }
25         cout << "|" << endl;
26     }
27     A = NMF(W, 2, B);
28     printf("Matriz 1: \n");
29
30     for (int i = 0; i < A.size(); i++) {
31         cout << "|";
32         for (int j = 0; j < A[0].size(); j++) {
33             cout << " " << A[i][j] << " ";
34         }
35         cout << "|" << endl;
36     }
37
38     printf("Matriz 2: \n");
39
40     for (int i = 0; i < B.size(); i++) {
41         cout << "|";
42         for (int j = 0; j < B[0].size(); j++) {
43             cout << " " << B[i][j] << " ";
44         }
45         cout << "|" << endl;
46     }
47
48     vector<vector<double>> M;

```

```

49     M = MMultiplication(A, B);
50
51     printf("\n Matriz Multiplicada: \n");
52
53     for (int i = 0; i < M.size(); i++) {
54         cout << "|";
55         for (int j = 0; j < M[0].size(); j++) {
56             cout << " " << M[i][j] << " ";
57         }
58         cout << "|" << endl;
59     }
60
61     return 0;
62 }

```

O resultado do teste referente a esta tarefa é:

```

C:\Users\Pedro\Desktop\Numerico\EP1\bin\Debug\EP1.exe
Matriz Inicial:
| 0.3 0.6 0 |
| 0.5 0 1 |
| 0.4 0.8 0 |
Matriz 1:
| 0 0.6 |
| 1 0 |
| 0 0.8 |
Matriz 2:
| 0.5 0 1 |
| 0.5 1 0 |

Matriz Multiplicada:
| 0.3 0.6 0 |
| 0.5 0 1 |
| 0.4 0.8 0 |

Process returned 0 (0x0) execution time : 0.412 s
Press any key to continue.

```

## 3 Machine Learning

### 3.1 Tarefa Principal

Para a tarefa principal, adicionamos mais algumas funções. Todas elas estão no arquivo `preProcessamento.cpp`

A função `adquireImagem` abre um arquivo que já esteja organizado na forma pedida pelo enunciado (com todos os elementos de uma imagem em uma coluna), divide os elementos por 255, para que o módulo de cada elemento seja no máximo 1, e retorna a matriz.

```

1 vector<vector<double>> adquireImagem(string nomeDoArquivo, int numLinhas, int
    numColunas, int numImagens) {
2     fstream input(nomeDoArquivo, ios::in);

```

```

3   if (!input.is_open()) {
4       cout << endl << "Nao foi possivel abrir o arquivo" << endl;
5       return vector<vector<double>>();
6   }
7
8   int numLinA = numLinhas * numColunas;
9   vector<vector<double>> A(numLinA, vector<double>(numImagens));
10  double temp;
11  for (int i = 0; i < numLinA; i++) {
12      for (int j = 0; j < numImagens && !(input.eof()); j++) {
13          input >> temp;
14          A[i][j] = temp/255;
15      }
16      input.ignore(99999999, '\n');
17  }
18
19  input.close();
20
21  return A;
22 }

```

---

A função *aprendizagem* cria um arquivo para salvar o resultado, aplica a fatoração por matrizes não nulas no A, que foi a matriz obtida na função *adquireImagem*, e salva os elementos de W no arquivo de texto. W consiste da matriz de treinamento daquele dígito.

---

```

1  vector<vector<double>> aprendizagem(string nomeDoArquivo,
    vector<vector<double>> A, int p, int numLinhas, int numColunas, int
    numImagens) {
2      vector<vector<double>> W;
3      fstream output(nomeDoArquivo, ios::out);
4      if (!output.is_open()) {
5          cout << endl << "Nao foi possivel abrir o arquivo" << endl;
6          return vector<vector<double>>();
7      }
8
9      // obter W
10     W = NMF(A, p);
11
12     // salvando W no arquivo:
13     int tamLinW = W.size();
14     int tamColW = W[0].size();
15     for (int i = 0; i < tamLinW; i++) {
16         for (int j = 0; j < tamColW; j++) {
17             output << W[i][j] << " ";
18         }
19         output << "\n";

```

```

20     }
21     output.close();
22     return W;
23 }

```

---

A função *classificaDigito* calcula a solução do sistema, com o *Wdigito* e a matriz *A* inicial. Com a solução, verifica se essa é a que possui o menor erro quadrado, caso seja, atualiza o valor do erro e o índice de qual é o dígito provável dessa imagem.

---

```

1 void classificaDigito(vector<vector<double>> Wdigito, vector<vector<double>> A,
    vector<double>& erro, vector<double>& indice, int digito,
2 int p, int numLinhas, int numColunas, int n_teste) {
3     if (n_teste > 10000) {
4         cout << endl << "Numero de teste invalido" << endl;
5         return;
6     }
7
8     int numLinA = numColunas * numLinhas;
9
10    vector<vector<double>> salvaA; // matriz para deixar sempre salvo os valores
    a serem testados
11    salvaA = A;
12    vector<vector<double>> H;
13
14    H = solucaoSimultaneos(Wdigito, A); // solucao do sistema
15    A = salvaA;
16
17    vector<vector<double>> produto;
18    produto = MMultiplication(Wdigito, H); // produto de W e H
19    int tamColProd = produto[0].size();
20    int tamLinProd = produto.size();
21
22    double provisorio; // para salvar A - WH
23    // calculo do erro
24    double soma;
25    double erro_atual;
26    for (int j = 0; j < tamColProd; j++) {
27        soma = 0;
28        for (int i = 0; i < tamLinProd; i++) {
29            provisorio = A[i][j] - produto[i][j];
30            soma += provisorio * provisorio; // calcula o quadrado de cada elemento
            da coluna e soma todos eles
31        }
32        erro_atual = sqrt(soma); // raiz do somatorio, esse e o erro
33        if (digito != 0) { // atualiza o erro para os digitos subsequentes
34            if (erro_atual > erro[j]) {
35                erro[j] = erro_atual;

```

```

36         indice[j] = digito;
37     }
38 }
39 else { // erro inicial corresponde ao erro do digito 0
40     erro[j] = erro_atual;
41     indice[j] = 0;
42 }
43 }
44 }

```

---

A função *taxaDeAcerto* verifica a taxa de acerto total e a taxa de acerto por dígito. Possui como entrada o nome do arquivo onde estão salvos os testes, o vetor onde foram salvos os valores prováveis de dígitos para cada imagem, quantos testes foram feitos e uma variável para cada dígito que irá salvar por referência a taxa de acerto por dígito.

---

```

1 double taxaDeAcerto(string arquivoIndices, vector<double> indice, int n_teste,
2     double& acerto0, double& acerto1, double& acerto2, double& acerto3, double&
3     acerto4, double& acerto5,
4     double& acerto6, double& acerto7, double& acerto8, double& acerto9) {
5     // pegando os dados do gabarito
6     fstream input(arquivoIndices, ios::in);
7     if (!input.is_open()) {
8         cout << endl << "Nao foi possivel abrir o arquivo: " << arquivoIndices <<
9         endl;
10        return 0;
11    }
12    vector<double> gabarito(n_teste);
13    for (int i = 0; i < n_teste && !(input.eof()); i++) { // Pegando os dados do
14        gabarito
15        input >> gabarito[i];
16    }
17
18    input.close();
19
20    double acertos = 0;
21    acerto0 = 0;
22    acerto1 = 0;
23    acerto2 = 0;
24    acerto3 = 0;
25    acerto4 = 0;
26    acerto5 = 0;
27    acerto6 = 0;
28    acerto7 = 0;
29    acerto8 = 0;
30    acerto9 = 0;
31    double quantidade0 = 0;
32    double quantidade1 = 0;

```

```

30 double quantidade2 = 0;
31 double quantidade3 = 0;
32 double quantidade4 = 0;
33 double quantidade5 = 0;
34 double quantidade6 = 0;
35 double quantidade7 = 0;
36 double quantidade8 = 0;
37 double quantidade9 = 0;
38
39 for (int i = 0; i < n_teste; i++) {
40     // verificar o numero total de cada digito no arquivo
41     // utilizando uma busca binaria por ser mais rapida
42     if (gabarito[i] < 4.5) { // pode ser 0,1,2,3,4
43
44         if (gabarito[i] < 2.5) { // pode ser 0,1,2
45
46             if (gabarito[i] < 1.5) { // pode ser 0,1
47
48                 if (gabarito[i] == 0) { // apenas 0
49                     quantidade0++;
50                 }
51                 else { // apenas 1
52                     quantidade1++;
53                 }
54             }
55             else { // apenas 2
56                 quantidade2++;
57             }
58         }
59         else { // pode ser 3,4
60             if (gabarito[i] == 3) { // apenas 3
61                 quantidade3++;
62             }
63             else { // apenas 4
64                 quantidade4++;
65             }
66         }
67     }
68     else { // pode ser 5,6,7,8,9
69
70         if (gabarito[i] < 7.5) { // pode ser 5,6,7
71
72             if (gabarito[i] < 6.5) { // pode ser 5 ou 6
73
74                 if (gabarito[i] == 5) { // apenas 5
75                     quantidade5++;

```



```

76         }
77         else { // apenas 6
78             quantidade6++;
79         }
80     }
81     else { // apenas 7
82         quantidade7++;
83     }
84 }
85 else { // pode ser 8 ou 9
86
87     if (indice[i] == 8) { // apenas 8
88         quantidade8++;
89     }
90     else { // apenas 9
91         quantidade9++;
92     }
93 }
94 }
95
96 if (gabarito[i] == indice[i]) {
97     acertos++;
98     // vamos categorizar utilizando uma arvore de busca, de modo a reduzir
99     // o tempo medio de execucao
100     if (indice[i] < 4.5) { // pode ser 0,1,2,3,4
101
102         if (indice[i] < 2.5) { // pode ser 0,1,2
103
104             if (indice[i] < 1.5) { // pode ser 0,1
105
106                 if (indice[i] == 0) { // apenas 0
107                     acerto0++;
108                 }
109                 else { // apenas 1
110                     acerto1++;
111                 }
112             }
113             else { // apenas 2
114                 acerto2++;
115             }
116         }
117         else { // pode ser 3,4
118             if (indice[i] == 3) { // apenas 3
119                 acerto3++;
120             }
121             else { // apenas 4

```

```

121         acerto4++;
122     }
123 }
124 }
125 else { // pode ser 5,6,7,8,9
126
127     if (indice[i] < 7.5) { // pode ser 5,6,7
128
129         if (indice[i] < 6.5){ // pode ser 5 ou 6
130
131             if (indice[i] == 5) { // apenas 5
132                 acerto5++;
133             }
134             else { // apenas 6
135                 acerto6++;
136             }
137         }
138         else { // apenas 7
139             acerto7++;
140         }
141     }
142     else { // pode ser 8 ou 9
143
144         if (indice[i] == 8) { // apenas 8
145             acerto8++;
146         }
147         else { // apenas 9
148             acerto9++;
149         }
150     }
151 }
152 }
153 }
154
155 acertos = acertos / n_teste;
156 acerto0 = acerto0 / quantidade0;
157 acerto1 = acerto1 / quantidade1;
158 acerto2 = acerto2 / quantidade2;
159 acerto3 = acerto3 / quantidade3;
160 acerto4 = acerto4 / quantidade4;
161 acerto5 = acerto5 / quantidade5;
162 acerto6 = acerto6 / quantidade6;
163 acerto7 = acerto7 / quantidade7;
164 acerto8 = acerto8 / quantidade8;
165 acerto9 = acerto9 / quantidade9;
166

```

```
167     return acertos;
168 }
```

---

Para realizar os testes, utilizamos a main na forma abaixo. Repare que a biblioteca *chrono* e *iomanip* foram utilizadas para calcular o tempo de execução do programa. Pode-se retirar ambas as bibliotecas e apagar a parte do código para medir o tempo de execução. O exemplo abaixo é para o caso com  $p = 15$ , realizando 10 mil testes e com 4000 treinos por dígito .

---

```
1  #include <iostream>
2  #include <vector>
3  #include <fstream>
4  #include <string>
5  #include <math.h>
6  #include <chrono> // biblioteca p/ medir tempo de execucao
7  #include <iomanip> // tirar essa biblioteca depois
8  #include "fatoracaoQR.h"
9  #include "MatrixOperations.h"
10 #include "rotGivens.h"
11 #include "fatoracaoNaoNeg.h"
12 #include "preProcessamento.h"
13
14 #define numeroLinhas 28
15 #define numeroColunas 28
16 #define tamanhoP 15
17 #define numeroImagens 4000
18 #define numeroTestes 10000
19
20 using namespace std;
21 using namespace std::chrono;
22
23 int main() {
24     // Parte para calcular o tempo de execucao
25     auto start = chrono::high_resolution_clock::now();
26     // unsync the I/O of C and C++.
27     ios_base::sync_with_stdio(false);
28
29     vector<vector<double>> b, W0, W1, W2, W3, W4, W5, W6, W7, W8, W9;
30
31     cout << "EP1 Calculo Numerico" << endl;
32     cout << "Alunos: Lucas Werner e Pedro Rabelo" << endl;
33     cout << endl;
34     cout << "Parametros utilizados: " << endl;
35     cout << "p = " << tamanhoP << endl;
36     cout << "ndig_treino = " << numeroImagens << endl;
37     cout << "n_test = " << numeroTestes << endl;
```

```

38
39     b = acquireImagem("train_dig0.txt", numeroLinhas, numeroColunas,
40                       numeroImagens);
41     W0 = aprendizagem("treino_d0_p15_treino4000.txt", b, tamanhoP, numeroLinhas,
42                      numeroColunas, numeroImagens);
43     b = acquireImagem("train_dig1.txt", numeroLinhas, numeroColunas,
44                       numeroImagens);
45     W1 = aprendizagem("treino_d1_p15_treino4000.txt", b, tamanhoP, numeroLinhas,
46                      numeroColunas, numeroImagens);
47     b = acquireImagem("train_dig2.txt", numeroLinhas, numeroColunas,
48                       numeroImagens);
49     W2 = aprendizagem("treino_d2_p15_treino4000.txt", b, tamanhoP, numeroLinhas,
50                      numeroColunas, numeroImagens);
51     b = acquireImagem("train_dig3.txt", numeroLinhas, numeroColunas,
52                       numeroImagens);
53     W3 = aprendizagem("treino_d3_p15_treino4000.txt", b, tamanhoP, numeroLinhas,
54                      numeroColunas, numeroImagens);
55     b = acquireImagem("train_dig4.txt", numeroLinhas, numeroColunas,
56                       numeroImagens);
57     W4 = aprendizagem("treino_d4_p15_treino4000.txt", b, tamanhoP, numeroLinhas,
58                      numeroColunas, numeroImagens);
59     b = acquireImagem("train_dig5.txt", numeroLinhas, numeroColunas,
60                       numeroImagens);
61     W5 = aprendizagem("treino_d5_p15_treino4000.txt", b, tamanhoP, numeroLinhas,
62                      numeroColunas, numeroImagens);
63     b = acquireImagem("train_dig6.txt", numeroLinhas, numeroColunas,
64                       numeroImagens);
65     W6 = aprendizagem("treino_d6_p15_treino4000.txt", b, tamanhoP, numeroLinhas,
66                      numeroColunas, numeroImagens);
67     b = acquireImagem("train_dig7.txt", numeroLinhas, numeroColunas,
68                       numeroImagens);
69     W7 = aprendizagem("treino_d7_p15_treino4000.txt", b, tamanhoP, numeroLinhas,
70                      numeroColunas, numeroImagens);
71     b = acquireImagem("train_dig8.txt", numeroLinhas, numeroColunas,
72                       numeroImagens);
73     W8 = aprendizagem("treino_d8_p15_treino4000.txt", b, tamanhoP, numeroLinhas,
74                      numeroColunas, numeroImagens);
75     b = acquireImagem("train_dig9.txt", numeroLinhas, numeroColunas,
76                       numeroImagens);
77     W9 = aprendizagem("treino_d9_p15_treino4000.txt", b, tamanhoP, numeroLinhas,
78                      numeroColunas, numeroImagens);
79
80     vector<double> erro(numeroTestes);
81     vector<double> indice(numeroTestes, 0);
82
83     // arquivo com as imagens para testar se acertou a classificacao

```

```

64     fstream input("test_images.txt", ios::in);
65     if (!input.is_open()) {
66         cout << endl << "Nao foi possivel abrir o arquivo de teste de imagens" <<
            endl;
67         return 0;
68     }
69     int numLinA = numeroLinhas * numeroColunas;
70     vector<vector<double>> A(numLinA, vector<double>(numeroTestes));
71     for (int i = 0; i < numLinA; i++) {
72         for (int j = 0; j < numeroTestes && !(input.eof()); j++) {
73             input >> A[i][j];
74         }
75         input.ignore(99999999, '\n');
76     }
77     input.close();
78
79     classificaDigito(W0, A, erro, indice, 0, tamanhoP, numeroLinhas,
        numeroColunas, numeroTestes);
80     classificaDigito(W1, A, erro, indice, 1, tamanhoP, numeroLinhas,
        numeroColunas, numeroTestes);
81     classificaDigito(W2, A, erro, indice, 2, tamanhoP, numeroLinhas,
        numeroColunas, numeroTestes);
82     classificaDigito(W3, A, erro, indice, 3, tamanhoP, numeroLinhas,
        numeroColunas, numeroTestes);
83     classificaDigito(W4, A, erro, indice, 4, tamanhoP, numeroLinhas,
        numeroColunas, numeroTestes);
84     classificaDigito(W5, A, erro, indice, 5, tamanhoP, numeroLinhas,
        numeroColunas, numeroTestes);
85     classificaDigito(W6, A, erro, indice, 6, tamanhoP, numeroLinhas,
        numeroColunas, numeroTestes);
86     classificaDigito(W7, A, erro, indice, 7, tamanhoP, numeroLinhas,
        numeroColunas, numeroTestes);
87     classificaDigito(W8, A, erro, indice, 8, tamanhoP, numeroLinhas,
        numeroColunas, numeroTestes);
88     classificaDigito(W9, A, erro, indice, 9, tamanhoP, numeroLinhas,
        numeroColunas, numeroTestes);
89
90     double acertos;
91     double acerto0, acerto1, acerto2, acerto3, acerto4, acerto5, acerto6,
        acerto7, acerto8, acerto9;
92     acertos = taxaDeAcerto("test_index.txt", indice, numeroImagens, acerto0,
        acerto1, acerto2, acerto3, acerto4, acerto5, acerto6, acerto7, acerto8,
        acerto9);
93     cout << "Taxa de acerto total: " << acertos*100 << "%" << endl;
94     cout << "Acertos do digito 0: " << acerto0 * 100 << "%" << endl;
95     cout << "Acertos do digito 1: " << acerto1 * 100 << "%" << endl;

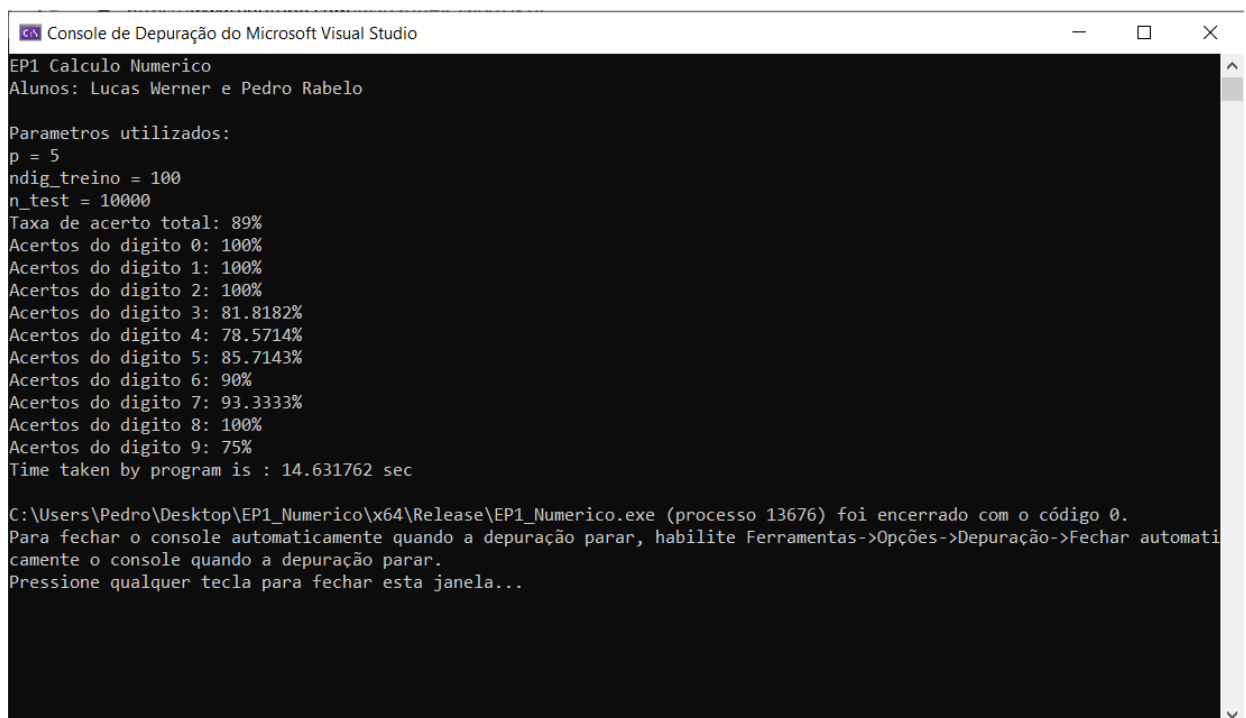
```

```

96     cout << "Acertos do digito 2: " << acerto2 * 100 << "%" << endl;
97     cout << "Acertos do digito 3: " << acerto3 * 100 << "%" << endl;
98     cout << "Acertos do digito 4: " << acerto4 * 100 << "%" << endl;
99     cout << "Acertos do digito 5: " << acerto5 * 100 << "%" << endl;
100    cout << "Acertos do digito 6: " << acerto6 * 100 << "%" << endl;
101    cout << "Acertos do digito 7: " << acerto7 * 100 << "%" << endl;
102    cout << "Acertos do digito 8: " << acerto8 * 100 << "%" << endl;
103    cout << "Acertos do digito 9: " << acerto9 * 100 << "%" << endl;
104
105    // Parte para calcular o tempo de execucao
106    auto end = chrono::high_resolution_clock::now();
107    // Calculating total time taken by the program.
108    double time_taken =
109        chrono::duration_cast<chrono::nanoseconds>(end - start).count();
110    time_taken *= 1e-9;
111    cout << "Time taken by program is : " << fixed
112        << time_taken << setprecision(9);
113    cout << " sec" << endl;
114
115    return 0;
116 }

```

Abaixo têm-se os resultados obtidos para todos os casos (com 100,1000 e 4000 treinamentos por dígito, com p valendo 5, 10 e 15, em todos os casos foram feitos 10 mil testes).



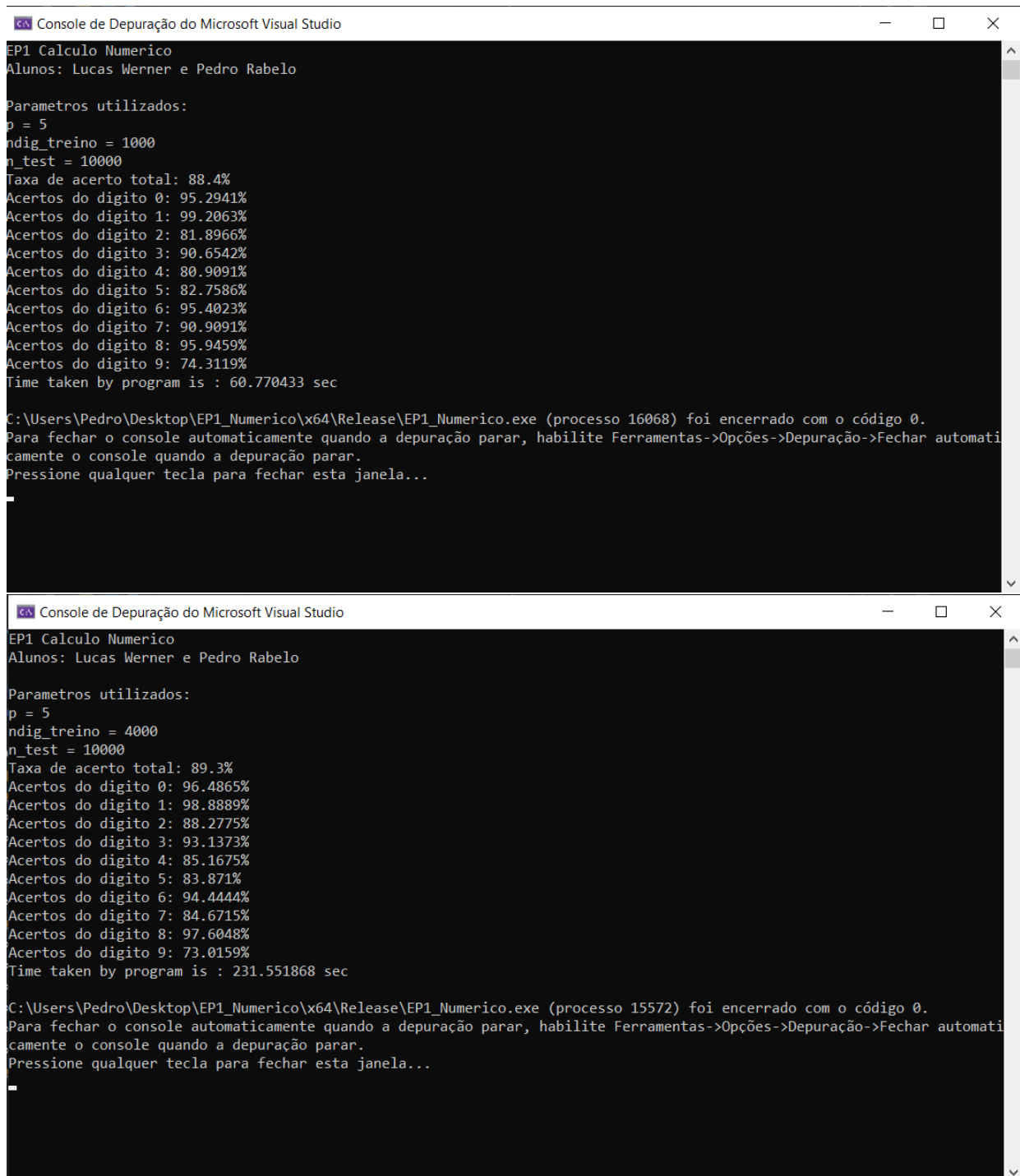
```

Microsoft Visual Studio
EP1 Calculo Numerico
Alunos: Lucas Werner e Pedro Rabelo

Parametros utilizados:
p = 5
ndig_treino = 100
n_test = 10000
Taxa de acerto total: 89%
Acertos do digito 0: 100%
Acertos do digito 1: 100%
Acertos do digito 2: 100%
Acertos do digito 3: 81.8182%
Acertos do digito 4: 78.5714%
Acertos do digito 5: 85.7143%
Acertos do digito 6: 90%
Acertos do digito 7: 93.3333%
Acertos do digito 8: 100%
Acertos do digito 9: 75%
Time taken by program is : 14.631762 sec

C:\Users\Pedro\Desktop\EP1_Numerico\x64\Release\EP1_Numerico.exe (processo 13676) foi encerrado com o código 0.
Para fechar o console automaticamente quando a depuração parar, habilite Ferramentas->Opções->Depuração->Fechar automaticamente o console quando a depuração parar.
Pressione qualquer tecla para fechar esta janela...

```



Console de Depuração do Microsoft Visual Studio

EP1 Calculo Numerico  
Alunos: Lucas Werner e Pedro Rabelo

Parametros utilizados:  
p = 5  
ndig\_treino = 1000  
n\_test = 10000  
Taxa de acerto total: 88.4%  
Acertos do digito 0: 95.2941%  
Acertos do digito 1: 99.2063%  
Acertos do digito 2: 81.8966%  
Acertos do digito 3: 90.6542%  
Acertos do digito 4: 80.9091%  
Acertos do digito 5: 82.7586%  
Acertos do digito 6: 95.4023%  
Acertos do digito 7: 90.9091%  
Acertos do digito 8: 95.9459%  
Acertos do digito 9: 74.3119%  
Time taken by program is : 60.770433 sec

C:\Users\Pedro\Desktop\EP1\_Numerico\x64\Release\EP1\_Numerico.exe (processo 16068) foi encerrado com o código 0.  
Para fechar o console automaticamente quando a depuração parar, habilite Ferramentas->Opções->Depuração->Fechar automaticamente o console quando a depuração parar.  
Pressione qualquer tecla para fechar esta janela...

Console de Depuração do Microsoft Visual Studio

EP1 Calculo Numerico  
Alunos: Lucas Werner e Pedro Rabelo

Parametros utilizados:  
p = 5  
ndig\_treino = 4000  
n\_test = 10000  
Taxa de acerto total: 89.3%  
Acertos do digito 0: 96.4865%  
Acertos do digito 1: 98.8889%  
Acertos do digito 2: 88.2775%  
Acertos do digito 3: 93.1373%  
Acertos do digito 4: 85.1675%  
Acertos do digito 5: 83.871%  
Acertos do digito 6: 94.4444%  
Acertos do digito 7: 84.6715%  
Acertos do digito 8: 97.6048%  
Acertos do digito 9: 73.0159%  
Time taken by program is : 231.551868 sec

C:\Users\Pedro\Desktop\EP1\_Numerico\x64\Release\EP1\_Numerico.exe (processo 15572) foi encerrado com o código 0.  
Para fechar o console automaticamente quando a depuração parar, habilite Ferramentas->Opções->Depuração->Fechar automaticamente o console quando a depuração parar.  
Pressione qualquer tecla para fechar esta janela...

```
Console de Depuração do Microsoft Visual Studio
EP1 Calculo Numerico
Alunos: Lucas Werner e Pedro Rabelo

Parametros utilizados:
p = 10
ndig_treino = 100
n_test = 10000
Taxa de acerto total: 90%
Acertos do digito 0: 100%
Acertos do digito 1: 100%
Acertos do digito 2: 87.5%
Acertos do digito 3: 90.9091%
Acertos do digito 4: 78.5714%
Acertos do digito 5: 71.4286%
Acertos do digito 6: 80%
Acertos do digito 7: 100%
Acertos do digito 8: 100%
Acertos do digito 9: 90.9091%
Time taken by program is : 22.661392 sec

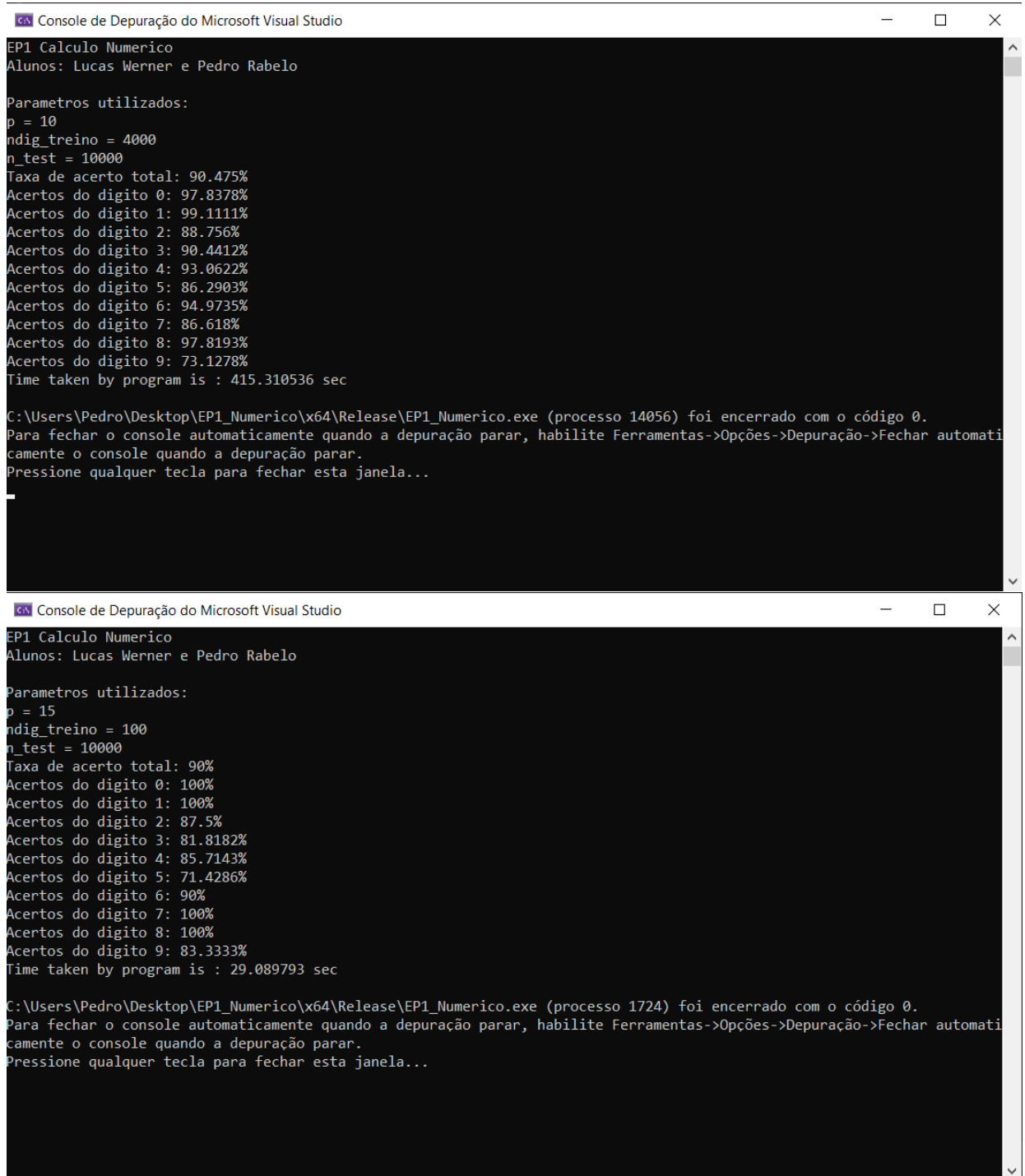
C:\Users\Pedro\Desktop\EP1_Numerico\x64\Release\EP1_Numerico.exe (processo 15812) foi encerrado com o código 0.
Para fechar o console automaticamente quando a depuração parar, habilite Ferramentas->Opções->Depuração->Fechar automati
camente o console quando a depuração parar.
Pressione qualquer tecla para fechar esta janela...
```

```
Console de Depuração do Microsoft Visual Studio
EP1 Calculo Numerico
Alunos: Lucas Werner e Pedro Rabelo

Parametros utilizados:
p = 10
ndig_treino = 1000
n_test = 10000
Taxa de acerto total: 90.9%
Acertos do digito 0: 98.8235%
Acertos do digito 1: 100%
Acertos do digito 2: 86.2069%
Acertos do digito 3: 87.8505%
Acertos do digito 4: 93.6364%
Acertos do digito 5: 87.3563%
Acertos do digito 6: 93.1034%
Acertos do digito 7: 89.899%
Acertos do digito 8: 95.122%
Acertos do digito 9: 77.2277%
Time taken by program is : 112.152493 sec

C:\Users\Pedro\Desktop\EP1_Numerico\x64\Release\EP1_Numerico.exe (processo 9068) foi encerrado com o código 0.
Para fechar o console automaticamente quando a depuração parar, habilite Ferramentas->Opções->Depuração->Fechar automati
camente o console quando a depuração parar.
Pressione qualquer tecla para fechar esta janela...
```





Console de Depuração do Microsoft Visual Studio

EP1 Calculo Numerico  
Alunos: Lucas Werner e Pedro Rabelo

Parametros utilizados:  
p = 10  
ndig\_treino = 4000  
n\_test = 10000  
Taxa de acerto total: 90.475%  
Acertos do digito 0: 97.8378%  
Acertos do digito 1: 99.1111%  
Acertos do digito 2: 88.756%  
Acertos do digito 3: 90.4412%  
Acertos do digito 4: 93.0622%  
Acertos do digito 5: 86.2903%  
Acertos do digito 6: 94.9735%  
Acertos do digito 7: 86.618%  
Acertos do digito 8: 97.8193%  
Acertos do digito 9: 73.1278%  
Time taken by program is : 415.310536 sec

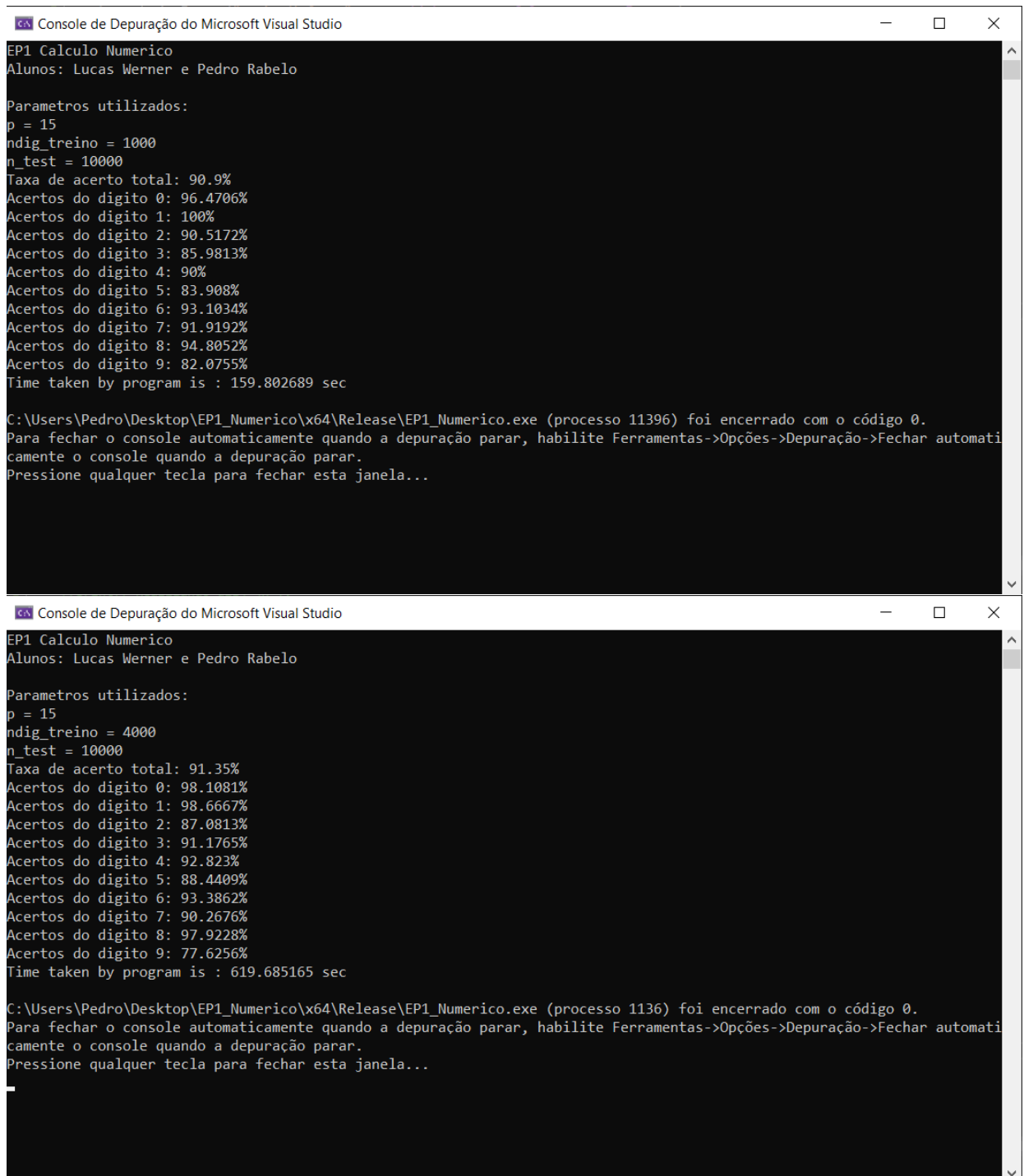
C:\Users\Pedro\Desktop\EP1\_Numerico\x64\Release\EP1\_Numerico.exe (processo 14056) foi encerrado com o código 0.  
Para fechar o console automaticamente quando a depuração parar, habilite Ferramentas->Opções->Depuração->Fechar automaticamente o console quando a depuração parar.  
Pressione qualquer tecla para fechar esta janela...

Console de Depuração do Microsoft Visual Studio

EP1 Calculo Numerico  
Alunos: Lucas Werner e Pedro Rabelo

Parametros utilizados:  
p = 15  
ndig\_treino = 100  
n\_test = 10000  
Taxa de acerto total: 90%  
Acertos do digito 0: 100%  
Acertos do digito 1: 100%  
Acertos do digito 2: 87.5%  
Acertos do digito 3: 81.8182%  
Acertos do digito 4: 85.7143%  
Acertos do digito 5: 71.4286%  
Acertos do digito 6: 90%  
Acertos do digito 7: 100%  
Acertos do digito 8: 100%  
Acertos do digito 9: 83.3333%  
Time taken by program is : 29.089793 sec

C:\Users\Pedro\Desktop\EP1\_Numerico\x64\Release\EP1\_Numerico.exe (processo 1724) foi encerrado com o código 0.  
Para fechar o console automaticamente quando a depuração parar, habilite Ferramentas->Opções->Depuração->Fechar automaticamente o console quando a depuração parar.  
Pressione qualquer tecla para fechar esta janela...



Console de Depuração do Microsoft Visual Studio

EP1 Calculo Numerico  
Alunos: Lucas Werner e Pedro Rabelo

Parametros utilizados:  
p = 15  
ndig\_treino = 1000  
n\_test = 10000  
Taxa de acerto total: 90.9%  
Acertos do digito 0: 96.4706%  
Acertos do digito 1: 100%  
Acertos do digito 2: 90.5172%  
Acertos do digito 3: 85.9813%  
Acertos do digito 4: 90%  
Acertos do digito 5: 83.908%  
Acertos do digito 6: 93.1034%  
Acertos do digito 7: 91.9192%  
Acertos do digito 8: 94.8052%  
Acertos do digito 9: 82.0755%  
Time taken by program is : 159.802689 sec

C:\Users\Pedro\Desktop\EP1\_Numerico\x64\Release\EP1\_Numerico.exe (processo 11396) foi encerrado com o código 0.  
Para fechar o console automaticamente quando a depuração parar, habilite Ferramentas->Opções->Depuração->Fechar automaticamente o console quando a depuração parar.  
Pressione qualquer tecla para fechar esta janela...

Console de Depuração do Microsoft Visual Studio

EP1 Calculo Numerico  
Alunos: Lucas Werner e Pedro Rabelo

Parametros utilizados:  
p = 15  
ndig\_treino = 4000  
n\_test = 10000  
Taxa de acerto total: 91.35%  
Acertos do digito 0: 98.1081%  
Acertos do digito 1: 98.6667%  
Acertos do digito 2: 87.0813%  
Acertos do digito 3: 91.1765%  
Acertos do digito 4: 92.823%  
Acertos do digito 5: 88.4409%  
Acertos do digito 6: 93.3862%  
Acertos do digito 7: 90.2676%  
Acertos do digito 8: 97.9228%  
Acertos do digito 9: 77.6256%  
Time taken by program is : 619.685165 sec

C:\Users\Pedro\Desktop\EP1\_Numerico\x64\Release\EP1\_Numerico.exe (processo 1136) foi encerrado com o código 0.  
Para fechar o console automaticamente quando a depuração parar, habilite Ferramentas->Opções->Depuração->Fechar automaticamente o console quando a depuração parar.  
Pressione qualquer tecla para fechar esta janela...

Os testes foram realizados pelo VS Code Community (IDE gratuito do VS Code) com o comando -O2 e -m64 e também direto do terminal do Ubuntu 18.04, através do compilador GCC, com o comando -O2.

Analisando os resultados, percebe-se uma correlação entre a taxa de acerto e o número de treinamentos por dígitos. No geral, quanto maior o numero de treinamentos, maior é a precisão. No entanto, a precisão aumenta em uma proporção pequena, principalmente se

levado em conta o maior tempo de execução. Por esse motivo, devido a elevada precisão inicial (cerca de 90%) pode ser pouco vantajoso realizar diversos treinamentos por dígito.

Ao aumentar o valor de  $p$ , em geral, também eleva-se a taxa de acerto. Mas, novamente, deve-se analisar qual a aplicação desejada, visto que o tempo de execução aumenta consideravelmente para um ganho relativamente pequeno na taxa de acerto.

Em alguns casos, ocorreu uma queda na taxa de acerto ao se elevar os parâmetros (como de  $p = 5$  e  $ndig\_treino = 100$  para  $p = 5$  e  $ndig\_treino = 100$ ). Pelo método de fatoração por matrizes não negativas não ser exato, isto é, ele realiza uma aproximação iterativa de MMQ que utiliza como valor inicial números aleatórios, acreditamos que, caso o número de iterações fosse maior, não ocorreria essa queda na taxa de acerto. Outra causa pode ser o valor inicial, que em um dos casos levou para uma aproximação melhor.

## 4 Considerações Finais

Após essas semanas trabalhando no código, o resultado foi satisfatório. É interessante trabalhar com temas atuais, como Machine Learning.

Esse exercício nos proporcionou uma série de novos desafios. O principal deles foi na questão de otimização. Todos os exercícios que tínhamos feito para as outras disciplinas de computação levavam pouco tempo para execução. Por outro lado, a versão inicial desse código demorava muito tempo para rodar (cerca de 15 minutos para o 1º caso). Após conversar com colegas descobrimos que era um tempo bem superior à média.

Conseguimos reduzir consideravelmente o tempo de execução ao tomar diversas medidas de otimização. A principal delas ocorreu na função *solucaoSimultaneos*.

Inicialmente, a rotação de Givens era chamada dentro dela. No entanto, a função era pouco eficiente, por criar duas grandes matrizes salvando os termos iniciais de  $W$  e  $A$ . Após a mudança na implementação da função a queda no tempo de execução foi considerável.

Acredito que, em termos gerais, o trabalho foi produtivo, tanto no fato de nos colocar em contato com um tema importante e atual de computação, como nos forçar a pensar em tópicos que foram pouco trabalhados nas outras disciplinas, como é o caso de otimização.