Como Funciona IA

*Mário Leite*

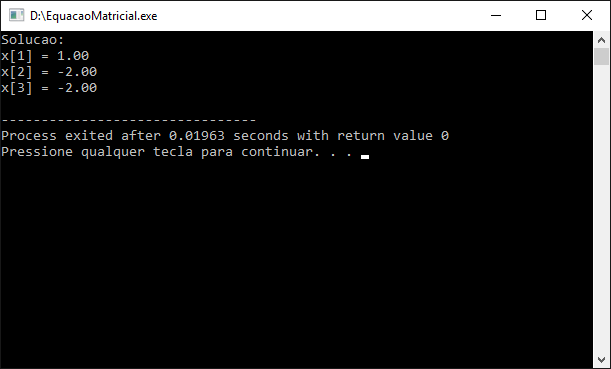
O conceito de **IA** (**I**nteligência **A**rtificial) não é novo; historicamente, surgiu com o inglês *Alan Turing* em 1936 (portanto há quase 90 anos) com a denominada “Máquina de Turing”. Esse conceito progrediu com os estudos sobre redes neurais, concebidas por *Warren McCulloch* e *Walter Pitts* em 1943. O objetivo era criar algoritmos que pudessem resolver problemas de usuários usando a lógica a partir de treinamentos incessantes desses algoritmos sobre a análise de grandes bases de dados alimentadas ao sistema. Resumidamente, **IA** é um programa de computador baseado em redes neurais que tenta resolver diversos tipos de problemas a partir de solicitações do usuário; solicitações essas que podem ser uma simples operação matemática até a criação de vídeos, pintura de um quadro abstrato ou fatos históricos! Até o momento já existem mais de dez **IA’s**, divididas em três categorias: Limitada, Geral e Superinteligência; sendo que, atualmente, apenas a primeira está disponível, pois as duas últimas ainda são alvos de pesquisas e discussões filosóficas e morais.

Uma **IA** opera por meio de algoritmos estruturados para reconhecer padrões e tomar decisões com sua lógica construída em três etapas principais:

1. **Entrada de Dados:** Recebe informações (texto, números, imagens) e as converte em representações matemáticas (vetores, matrizes).
2. **Processamento por Camadas:** Em redes neurais, camadas interligadas aplicam operações matemáticas (como multiplicação de matrizes e funções de ativação) para extrair características e relações complexas.
3. **Aprendizado e Adaptação:** Durante o treinamento ajusta parâmetros (pesos) minimizando erros via algoritmos de otimização (ex: gradiente descendente).

Um clássico de uso de **IA** é na solução de equações complexas; por exemplo, em equações matriciais do tipo **Ax = B**, onde o objetivo é encontrar os valores de **x** que satisfaçam a expressão, usando decomposições matemáticas (LU, QR) ou métodos iterativos: dependendo da estrutura do problema. O programa “**EquacaoMatricial**” é um exemplo de código gerado por **IA** (*Qwen-2.5 Max*); a **figura 1** mostra a saída desse programa para uma equação matricial dada. Mas, é importante frisar o seguinte: “*IA codifica, mas não programa!*”

-----------------------------------------------------------------------------------------------------------------------------------



**Figura 1 - Saída do programa “EquacaoMatricial”**

**//EquacaoMatricial.C**

#include <stdio.h>

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define N 3 **//tamanho da matriz (NxN)**

**void** **trocarLinhas**(**float** A[N][N], **float** B[N], **int** i, **int** j) {

**float** temp;

**for** (**int** k = 0; k < N; k++) {

temp = A[i][k];

A[i][k] = A[j][k];

A[j][k] = temp;

}

temp = B[i];

B[i] = B[j];

B[j] = temp;

}

**void** **eliminacaoGaussiana**(**float** A[N][N], **float** B[N], **float** X[N]) {

**int** i, j, k;

**float** fator;

**//Eliminação Gaussiana com pivoteamento parcial**

**for** (i = 0; i < N; i++) {

**//Encontrar o pivô máximo na coluna atual**

**int** maxRow = i;

**for** (j = i + 1; j < N; j++) {

**if** (fabs(A[j][i]) > fabs(A[maxRow][i])) {

maxRow = j;

}

}

**//Trocar linhas se necessário**

**if** (maxRow != i) {

**trocarLinhas**(A, B, i, maxRow);

}

**//Verificar se a matriz é singular ou próxima de singular**

**if** (fabs(A[i][i]) < 1e-10) { **//usar uma tolerância pequena para evitar erros**

**printf**("Erro: Matriz singular ou próxima de singular!\n");

**exit**(1);

}

**//Eliminação Gaussiana**

**for** (j = i + 1; j < N; j++) {

fator = A[j][i] / A[i][i];

**for** (k = i; k < N; k++) {

A[j][k] -= fator \* A[i][k];

}

B[j] -= fator \* B[i];

}

}

**//Substituição retroativa**

**for** (i = N - 1; i >= 0; i--) {

X[i] = B[i];

**for** (j = i + 1; j < N; j++) {

X[i] -= A[i][j] \* X[j];

}

X[i] /= A[i][i];

}

}

**int** **main**() {

**//Monta a equação com os valores dados**

**float** A[N][N] = {{3, 2, -1}, {2, -2, 4}, {-1, 0.5, -1}};

**float** B[N] = {1, -2, 0};

**float** X[N];

**eliminacaoGaussiana**(A, B, X);

**printf**("Solucao:\n");

**for** (**int** i = 0; i < N; i++) {

**printf**("x[%d] = %.2f\n", i + 1, X[i]);

}

**return** 0;

}