Implementação da Eliminação de Gauss

Diferenças em Tipos de Dados e Acesso à Memória

- **C**
  Utiliza arrays estáticos, o que implica que a memória é alocada de forma fixa na stack ou na data segment.
  As operações de acesso são diretas, porém sem proteção contra erros, por exemplo acesso fora dos limites.

- **Rust**
  Utiliza Vec<Vec<f32>> para a matriz e Vec<f32> para os vetores, com alocação dinâmica.
  Garante segurança de memória e checagem de limites em tempo de execução (a menos que use código unsafe).
  O gerenciamento de memória é feito automaticamente sem um coletor de lixo, usando um sistema de empréstimos.

- **Golang**
  Utiliza slices (que são mais flexíveis que arrays) para armazenar a matriz e os vetores.
  Go possui coleta de lixo, o que simplifica a gestão de memória, embora introduza um overhead mínimo.
  A linguagem também verifica limites em tempo de execução.

Organização dos Códigos e Chamadas de Função

- **C**
  Estrutura modular com funções separadas para parâmetros, inicialização, impressão, e o algoritmo de eliminação Gaussiana.
  As chamadas de função são diretas e o código é estruturado de forma procedural.

- **Rust**
  Organiza o código em funções como main() e gauss().
  Faz uso da biblioteca padrão para manipulação de argumentos e tempo, resultando num código conciso e seguro.
  O uso de crates (como rand) e o sistema de módulos do Rust ajudam a manter o código organizado.

- **Golang**
  Possui uma estrutura simples com função main() e funções auxiliares.

O gerenciamento de argumentos é feito pela biblioteca os, e o tempo é medido usando time.Now().
A simplicidade e a legibilidade são pontos fortes da implementação em Go.

Comandos de Controle de Fluxo

- **C**
  Usa os loops for com contadores inteiros tradicionais.
  A estrutura de controle é mais “manual”, como os incrementos e decrementos explícitos.

- **Rust e Golang**
  Ambos utilizam laços for com range, o que torna o código mais legível e reduz a chance de erros de indexação.
  Em Rust, o iterador rev() é usado para a substituição regressiva, enquanto no Go, o laço for com decremento simula essa lógica.

Métricas

- **Número de linhas**

	C	**Rust**	**Golang**
Linhas	101	74	58

- **Desempenho**

CPU: AMD Ryzen 3 3100 4-Core Processor 3.60 GHz
Memória: 16GB DDR4
Sistema Operacional: Windows 10

O Rust por padrão faz sua execução no modo debug, então para isso foi feita a execução para coleta do tempo de execução de ambas maneiras, com o modo debug(padrão) e no modo sem debug.

Matriz	C	Rust	Rust(otimizado)	Golang
500	125 ms	412 ms	6 ms	46 ms
1000	990 ms	3000 ms	47 ms	366 ms
1500	3371 ms	9739 ms	176 ms	1257 ms
2000	7970 ms	22914 ms	874 ms	3135 ms

Todos os valores são aproximados.

Conclusão

O Golang demonstrou grande eficiência em diversos aspectos, principalmente por possuir um código simples e oferecer um tempo de execução competitivo. Por outro lado, o C apresentou uma estrutura mais complexa, alcançando um desempenho superior apenas quando comparado ao Rust executado em modo debug. Quanto ao Rust, ele se destacou em termos de desempenho; contudo, ao ser utilizado com debug, seu desempenho fica comprometido se tornando o pior, e somente sem o debug ele revela todo o seu potencial.