Go possui uma sintaxe muito mais simples e intuitiva que a adotada por Rust. Rust é bem mais restrito em questão de tipagens e também pareceu apresentar uma faixa de tipos mais extensa e minuciosa (um bom exemplo é o tipo usize utilizado em laços), enquanto Go por sua vez permite inferência de tipos.

Ainda na sintaxe Rust implementa laços utilizando declaração da range e passo com incremento de 1 por padrão, requerendo a especificação caso, por exemplo, se deseje percorrer a faixa em ordem reversa; Já Go utiliza o mesmo padrão de sintaxe observado em C.

Pode-se observar também que Rust apresentou menor tolerância quanto ao uso de variáveis globais, sendo estas evitadas por padrão pelo compilador, bem como impedimento da declaração de vetores muito grandes, requerendo uma alocação dinâmica para permitir este tipo de uso da memória. Go além de não apresentar resistência quanto ao uso de globais também não apresentou problemas na declaração de grandes vetores. Outro ponto interessante a se ressaltar é o fato que Rust não possui matrizes nativamente, sendo assim requer ou uma biblioteca externa ou, como foi o nosso caso, uso de um vetor sendo tratado como matriz na execução.

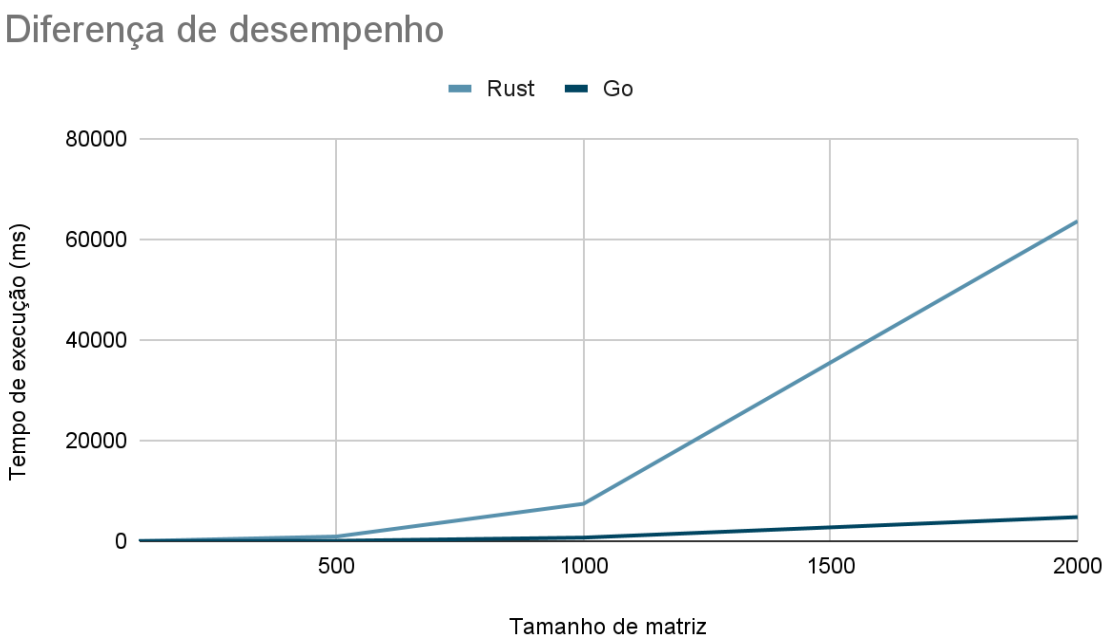
Rust também, por não aprovar o uso de globais, acabou necessitando de uma passagem de referência das matrizes, bem como o tamanho, em todas as funções que as utilizavam, algo que não foi necessário em momento algum nas funções implementadas em Go.

Da maneira que o código foi implementado, ambas as linguagens atingiram um número de linhas semelhantes, sendo Go com 124 linhas enquanto Rust em 114 linhas. Esta pequena diferença porém foi causada em grande parte devido a Go ser mais restrito no tratamento de erros (linhas 45 e 54 por exemplo) e também a maneira com que os packages foram importados (a configuração padrão da IDE utilizada impedia importação na mesma linha), de maneira que caso estes dois não existissem ambas teriam atingido 114 linhas.

Um pequeno detalhe também observado foi que, devido às maiores restrições de tipo em Rust, seu código acabou por apresentar maior quantidade de declarações explícitas, enquanto Go resolvia estes em tempo de compilação. E quanto à compilação, enquanto Go não necessita de um comando específico para esta, que é feita antes da execução, seu tempo de compilação pareceu ser um pouco mais lento, porém as diferenças em tempo de compilação não foram medidas.

Ambos os códigos foram rodados em uma máquina Windows 8.1 de 64 bits, Intel Core i5 1.70GHz, 8GB de RAM.

A partir dos testes pode-se notar um aumento exponencial no tempo de execução em Rust em comparação com Go, que permaneceu rápido e com aumento linear. Concluímos que um dos possíveis motivos possa ser a abordagem adotada pela dupla em relação às matrizes em Rust, talvez o uso da biblioteca de matrizes ou outra abordagem mais eficiente poderia ser menos custosa; Outro possível motivo também pode ter sido a alocação dinâmica em Rust, visto que este não permite a declaração de vetores utilizando números não constantes, comportamento que não se repetiu em Go.



Após os testes foi realizada uma tentativa de compilar Rust com flags de otimização para observar se haveria melhora significativa no tempo de execução, porém a diferença pareceu ser muito casual e aleatória e não surtiu o efeito esperado.

Foi observado que Rust possui grande atenção à segurança da memória, deixando de responsabilidade para o programador criar um código mais específico e evitar erros de execução, com sua tipagem estrita e, por padrão, diversas checagem de código para impedir os mais diversos erros, por mais pequenos que sejam. Go demonstrou ser mais focada em velocidade e simplicidade de implementação, porém aparenta não ter a mesma expressabilidade presente em Rust, potencialmente sendo um empecilho caso o algoritmo fosse um sistema maior.

Também percebeu-se grande diferença na velocidade de implementação do algoritmo, com a sintaxe intuitiva e mais “relaxada” de Go o código foi concluído em poucas horas, enquanto Rust, sendo mais robusto e rigoroso, necessitou mais que o dobro de tempo; Deve-se levar em conta também que foi o primeiro contato de ambas as integrantes com ambas as linguagens porém com algum conhecimento tanto em Python quanto em C que certamente auxiliou na velocidade de entendimento das sintaxes.