UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
COORDENAÇÃO DO CURSO DE ENGENHARIA CIVIL

MATHEUS SERRA DA SILVA

OTIMIZAÇÃO DE TALUDES UTILIZANDO UM ALGORITMO GENÉTICO ACOPLADO AO MÉTODO DE CONFIABILIDADE FOSM

SÃO LUÍS
2026

MATHEUS SERRA DA SILVA

OTIMIZAÇÃO DE TALUDES UTILIZANDO UM ALGORITMO GENÉTICO ACOPLADO AO MÉTODO DE CONFIABILIDADE FOSM

Trabalho de conclusão de curso I (TCCI) apresentado ao Curso de Engenharia Civil da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Engenheiro Civil.

Orientador: Prof. Dr. Felipe Alexander Vargas Bazán.

SÃO LUÍS

2026

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Diretoria Integrada de Bibliotecas/UFMA

Silva, Matheus Serra da.
Otimização de taludes utilizando um algoritmo genético acoplado ao método de confiabilidade FOSM / Matheus Serra da Silva. - 2026.
74 f.

Orientador(a): Felipe Alexander Vargas Bazán.
Curso de Engenharia Civil, Universidade Federal do Maranhão, São Luís, 2026.

1. Estabilidade de Taludes. 2. Método de Fellenius. 3. Algoritmo Genético Binário. 4. Otimização. 5. Método Fosm de Confiabilidade. I. Bazán, Felipe Alexander Vargas. II. Título.

MATHEUS SERRA DA SILVA

OTIMIZAÇÃO DE TALUDES UTILIZANDO UM ALGORITMO GENÉTICO ACOPLADO AO MÉTODO DE CONFIABILIDADE FOSM

Trabalho de conclusão de curso I (TCCI) apresentado ao Curso de Engenharia Civil da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Engenheiro Civil.

Aprovado em: _____/_____/_____

BANCA EXAMINADORA

Prof. Dr. Felipe Alexander Vargas Bazán

(Orientador)

Universidade Federal do Maranhão

Prof. Dr. Marcos Aurélio Araújo Santos

Universidade Federal do Maranhão

Prof. Dr. José Antônio Pires Ferreira Marão

Universidade Federal do Maranhão

AGRADECIMENTOS

Agradeço a Deus pela vida e capacidade, pela oportunidade de me levantar a cada manhã, pela força para lutar e alcançar. À minha família, por ser meu ponto de segurança, por construir as pontes que me ajudaram, me ajudam e me ajudarão a superar obstáculos e chegar mais longe. À minha amada companheira, Máydilla, pelo apoio incondicional em cada desafio, pelo carinho, pelo cuidado, pela paciência e pela felicidade que é estar com você a cada dia. Agradeço ao Prof. Dr. Felipe Alexander Vargas Bazán, pela orientação, dedicação e pela confiança depositada em meu trabalho, sua contribuição foi essencial para a conclusão desta fase tão importante. Por fim, agradeço a todos que, de alguma forma, fizeram parte desta jornada, oferecendo palavras de incentivo, gestos de apoio e amizade. Cada contribuição foi importante para que este momento se tornasse possível.

“Uma lição sem dor não tem significado. [...] Mas, quando você supera essa dor e a transforma em aprendizado, conquista um coração forte que não perde para nada... um coração forte como aço...”

(Edward Elric, Fullmetal Alchemist)

RESUMO

A análise de estabilidade de taludes é um problema recorrente na Geotecnia, cuja avaliação tradicional é frequentemente realizada por métodos determinísticos baseados no equilíbrio limite. Entretanto, as incertezas associadas às propriedades dos solos podem comprometer a representatividade dessas análises. Nesse contexto, o presente trabalho teve como objetivo desenvolver uma ferramenta computacional para análise da estabilidade de taludes, utilizando o método de Fellenius para o cálculo do fator de segurança, associado à aplicação de um algoritmo genético binário e do método FOSM (*First Order Second Moment*) para identificação da superfície crítica de ruptura correspondente à maior probabilidade de falha. A implementação foi realizada em linguagem Python, permitindo a determinação automática da geometria da superfície crítica, do fator de segurança, do índice de confiabilidade e da probabilidade de falha associada. Adicionalmente, uma abordagem determinística baseada na minimização do fator de segurança foi empregada para fins de validação e comparação dos resultados. Os resultados obtidos evidenciaram que a consideração das incertezas geotécnicas pode alterar a avaliação da estabilidade do talude, demonstrando que superfícies associadas aos menores fatores de segurança não necessariamente correspondem às maiores probabilidades de falha. Dessa forma, a metodologia proposta contribui para a integração entre métodos determinísticos e probabilísticos aplicados à Engenharia Geotécnica e Civil.

Palavras-chave: estabilidade de taludes; método de Fellenius; algoritmo genético binário; otimização; método FOSM de confiabilidade.

ABSTRACT

Slope stability analysis is a recurring problem in Geotechnics, traditionally assessed through deterministic methods based on limit equilibrium. However, uncertainties associated with soil properties and slope geometry may compromise the representativeness of such analyses. In this context, this study aims to develop a computational tool for slope stability analysis using the Fellenius method to calculate the safety factor, combined with a binary genetic algorithm and a numerical reliability procedure to optimize the search for the critical slip surface corresponding to the minimum safety factor and the maximum probability of failure. The binary genetic algorithm is employed as a deterministic optimization technique to improve the efficiency of identifying the critical slip surface, while the numerical probabilistic procedure performs an optimized stochastic search within the solution space, accounting for the variability of the problem parameters. The methods are implemented in the Python programming language, and a simple computational interface is developed to facilitate data input and result visualization. The proposed approach is expected to provide a more comprehensive and realistic assessment of slope stability, contributing to the integration of deterministic and probabilistic methods, as well as to the application of optimization techniques in Civil and Geotechnical Engineering contexts.

Keywords: slope stability; Fellenius method; binary genetic algorithm; optimization; FOSM reliability method.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização

Um talude é qualquer superfície inclinada de solo ou rocha, podendo ser uma encosta natural ou um corte ou aterro criado artificialmente (Gerscovich, 2016). Essas estruturas estão sujeitas a movimentos de massa e podem romper devido a diversos fatores, como a ação da água e mudanças nas condições do solo (Duncan; Wright; Brandon, 2014). Nesse contexto, a análise da estabilidade de taludes assume papel fundamental na Geotecnia, uma vez que influencia diretamente a segurança de estruturas como estradas, barragens e áreas de encosta. Falhas nesses maciços podem resultar em perdas significativas, o que torna essencial a aplicação de métodos seguros e eficazes para a avaliação da estabilidade.

A análise de estabilidade de taludes baseia-se, em grande parte, na determinação de um fator de segurança (FS), definido como a razão entre as forças resistentes e as forças solicitantes ao longo de uma superfície potencial de ruptura (Gerscovich, 2016). Para isso, é necessária uma investigação geotécnica para obter as características do talude e, com base nelas, utilizar algum método conhecido para o cálculo do fator de segurança. Nesse contexto, o método de Fellenius, também conhecido como método sueco, figura como um dos procedimentos mais tradicionais para a análise de taludes. Segundo Fellenius (1936), o método baseia-se na análise do equilíbrio limite de uma massa de solo que pode deslizar ao longo de uma superfície circular.

Embora o fator de segurança seja amplamente utilizado na prática, ele representa apenas uma estimativa determinística da condição de estabilidade do talude, obtida a partir de valores únicos para os parâmetros geotécnicos. Entretanto, propriedades como coesão, ângulo de atrito e peso específico apresentam variabilidade inerente decorrente da heterogeneidade dos solos, dos métodos de investigação e das limitações experimentais. Dessa forma, dois taludes com o mesmo fator de segurança podem apresentar níveis distintos de confiabilidade, evidenciando a importância de abordagens probabilísticas na avaliação da estabilidade (Doan, 2023). Nesse contexto, a análise probabilística surge como uma abordagem complementar, permitindo incorporar essas incertezas à avaliação da estabilidade por meio da estimativa da probabilidade de falha (PF) associada às superfícies potenciais de ruptura.

Além da consideração das incertezas, outro grande desafio associado à análise de estabilidade de taludes é a identificação da superfície crítica de ruptura (SCR). Como um mesmo talude admite um grande número de superfícies geometricamente possíveis, a determinação daquela que representa a condição mais crítica torna-se um problema de otimização, envolvendo um espaço de busca amplo e complexo.

Visando superar a complexidade do problema, técnicas computacionais de otimização podem ser usadas para identificar a SCR, como por exemplo os algoritmos genéticos (AG), que são inspirados em mecanismos de evolução natural e se caracterizam pela robustez e simplicidade, possuindo recursos de auto orientação na busca de soluções (Goldberg, 1989). Quando associados a métodos probabilísticos de análise de confiabilidade, como o método *First Order Second Moment* (FOSM), tornam possível não apenas localizar superfícies com baixos fatores de segurança, mas também avaliar sua probabilidade de falha, considerando a variabilidade dos parâmetros geotécnicos.

Nesse cenário, este trabalho propõe o desenvolvimento de uma ferramenta computacional para análise da estabilidade de taludes, utilizando o método de Fellenius para o cálculo do fator de segurança, um algoritmo genético binário para a busca otimizada de superfícies de ruptura e o método FOSM para a estimativa da probabilidade de falha de cada superfície analisada. A ferramenta foi implementada em linguagem *Python* e permite realizar otimizações determinísticas (minimizando o FS) e probabilísticas (maximizando a PF), contribuindo para uma avaliação mais abrangente da estabilidade de taludes.

1.2 Justificativa

A estabilidade de taludes é um tema de grande relevância na Engenharia Civil, pois está diretamente relacionada à segurança de obras como rodovias, barragens e encostas. Rupturas nesses sistemas podem causar prejuízos econômicos, impactos ambientais e riscos à vida humana, tornando indispensável a utilização de métodos confiáveis para a avaliação da estabilidade (Duncan; Wright; Brandon, 2014). Entretanto, a identificação da SCR constitui um problema complexo, devido à grande quantidade de combinações geométricas possíveis, as incertezas presentes nas características do solo e à natureza não linear do problema.

Métodos tradicionais, baseados em tentativas sucessivas ou em procedimentos determinísticos, como os métodos de equilíbrio limite, podem não garantir a

identificação da condição mais desfavorável. Nesse contexto, o emprego de métodos computacionais de otimização e confiabilidade, como os algoritmos genéticos e o FOSM, apresenta-se como uma alternativa eficiente para a busca da SCR. Essas técnicas permitem explorar o espaço de soluções de forma sistemática e automatizada, reduzindo a dependência da experiência do projetista e aumentando a robustez das análises.

Do ponto de vista acadêmico, o trabalho justifica-se pela integração entre métodos clássicos da estabilidade de taludes, como o método de Fellenius, e ferramentas modernas de otimização e confiabilidade. Além disso, a implementação desses métodos em uma ferramenta computacional confere caráter prático ao estudo, contribuindo para aplicações em análises preliminares de estabilidade de taludes.

1.3 Objetivos

1.3.1 Objetivo principal

Implementar uma ferramenta computacional em linguagem *Python* para identificação da superfície crítica de ruptura em taludes por meio de um algoritmo genético binário (AGB), cuja busca é orientada pela probabilidade de falha das superfícies candidatas, estimada pelo método FOSM a partir do fator de segurança calculado pelo método de Fellenius.

1.3.2 Objetivos específicos

- Implementar o método de Fellenius para a análise determinística através do cálculo do fator de segurança de taludes considerando superfícies circulares de ruptura.
- Modelar geometricamente o talude e as superfícies potenciais de ruptura em um ambiente computacional, estabelecendo as restrições geométricas necessárias ao processo de otimização.
- Implementar um algoritmo genético binário para explorar automaticamente o espaço de busca das superfícies potenciais de ruptura.
- Implementar uma análise probabilística baseada no método FOSM para estimar a probabilidade de falha das superfícies candidatas a partir do fator de segurança calculado pelo método de Fellenius.

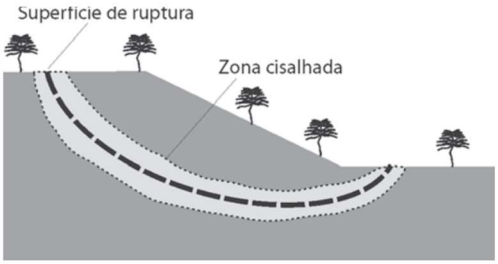
- Integrar o algoritmo genético binário ao método FOSM, de modo que a probabilidade de falha seja utilizada como critério de avaliação das superfícies durante o processo de otimização.
- Comparar os resultados obtidos pelas abordagens determinística e probabilística quanto às superfícies críticas identificadas e aos parâmetros de estabilidade obtidos.
- Analisar a coerência física das superfícies críticas identificadas pela ferramenta implementada.

2 REFERENCIAL TEÓRICO

2.1 Estabilidade de taludes

Gerscovich (2016) caracteriza a ruptura de taludes como a formação de uma superfície de cisalhamento que se estende por todo o maciço, criando uma camada de solo que perde suas características resistentes, como mostrado na Figura 1.

Figura 1 - Exemplo de superfície de ruptura em um talude.



Fonte: Gerscovich (2016).

A partir disso, a estabilidade de taludes é compreendida como a capacidade de uma massa de solo manter-se em equilíbrio diante das forças atuantes, sendo naturais ou de ação humana. Esse tema assume grande importância na Geotecnia, uma vez que taludes estão presentes em diversas obras civis, como cortes e aterros rodoviários, barragens, escavações e encostas naturais (Duncan; Wright; Brandon, 2014).

De forma geral, a ruptura de um talude ocorre quando as forças solicitantes, associadas principalmente ao peso próprio do solo e a ações externas, superam as forças resistentes mobilizadas ao longo de uma superfície de ruptura. Essas forças resistentes estão diretamente relacionadas aos parâmetros de resistência ao

cisalhamento do solo, como a coesão e o ângulo de atrito interno, além das condições de tensões e da presença de água nos poros (Das; Sobhan, 2014).

Czap e Marques Filho (2024) explicam que métodos de equilíbrio limite são abordagens determinísticas que envolvem equações de equilíbrio de forças ou momentos para calcular o FS e determinar a SCR. Segundo Duncan, Wright e Brandon (2014), o FS de solos é definido como a razão entre a resistência disponível ao cisalhamento e a resistência mobilizada necessária para manter o equilíbrio. Valores de FS superiores a 1 indicam condições estáveis quanto mais elevados forem, enquanto valores próximos a 1 sinalizam a possibilidade de ruptura.

Diversos fatores influenciam a estabilidade de um talude, incluindo a geometria do maciço, as propriedades mecânicas dos materiais envolvidos, o nível do lençol freático, as condições de drenagem e as ações externas, como carregamentos adicionais ou variações climáticas (Gerscovich, 2016).

Embora os métodos de equilíbrio limite sejam amplamente empregados na avaliação da estabilidade de taludes, sua aplicação tradicional baseia-se na utilização de valores médios para representar os parâmetros geotécnicos do maciço. Entretanto, por se tratar de um material natural, o solo apresenta elevada heterogeneidade, fazendo com que suas propriedades variem espacialmente. Dessa forma, os parâmetros utilizados nas análises representam estimativas obtidas por meio de investigações de campo, ensaios laboratoriais e interpretações do engenheiro, estando sujeitos a diferentes fontes de incerteza (Campello, 2020).

Segundo Campello (2020), as incertezas associadas às propriedades geotécnicas podem ser classificadas em dois tipos: incertezas aleatórias e incertezas epistêmicas. As incertezas aleatórias estão relacionadas à variabilidade natural do solo, decorrente de sua própria formação geológica, não podendo ser eliminadas mesmo com elevada investigação. Já as incertezas epistêmicas decorrem da limitação do conhecimento disponível sobre o maciço, sendo influenciadas pela falta de investigação, pelos erros de medição, pelas simplificações adotadas nos modelos de cálculo e pelas correlações empregadas para estimar propriedades geotécnicas a partir de ensaios indiretos.

Nesse contexto, análises exclusivamente determinísticas podem não representar adequadamente o nível de segurança de um talude, uma vez que desconsideram a variabilidade inerente às propriedades do solo. Assim, a utilização de abordagens probabilísticas torna-se uma alternativa para incorporar essas

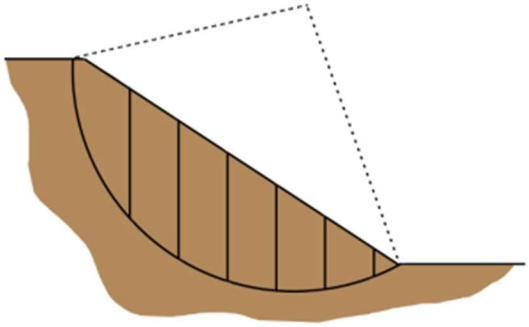
incertezas ao processo de avaliação da estabilidade, permitindo estimar não apenas o fator de segurança, mas também a probabilidade de ocorrência de ruptura.

2.2 Método de Fellenius

O método de Fellenius, também conhecido como método sueco, é um dos procedimentos mais tradicionais e simples utilizados na análise de estabilidade de taludes por meio dos métodos de equilíbrio limite, pois estabelece uma equação linear para a determinação do fator de segurança (Silva, 2011). Proposto originalmente por Wolmar Fellenius na década de 1930, esse método baseia-se na análise do equilíbrio global de uma massa de solo limitada por uma superfície de ruptura circular (Fellenius, 1936). Sendo amplamente empregado em estudos preliminares e em aplicações acadêmicas devido à sua simplicidade conceitual e facilidade de implementação.

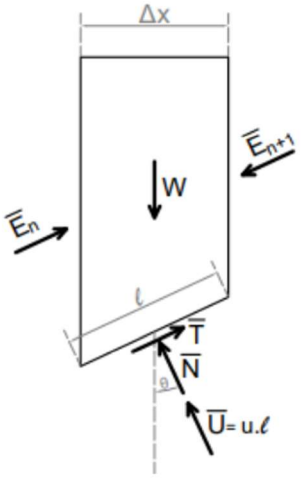
No método de Fellenius, a partir de uma possível superfície circular de ruptura o maciço é dividido em fatias verticais, como mostrado na Figura 2, e o equilíbrio é avaliado considerando apenas as forças atuantes em cada fatia, como o peso próprio e as forças resistentes ao cisalhamento mobilizadas ao longo da superfície de ruptura, como exemplificado na Figura 3.

Figura 2 - Fatias verticais usadas no método de Fellenius.



Fonte: o próprio autor (2026).

Figura 3 - Forças atuantes em uma fatia no método de Fellenius.



Fonte: Massad (2003).

Silva (2011) explica que o método assume que as forças de interação entre fatias adjacentes são paralelas as bases das mesmas, podendo ser desprezadas no cálculo, porém, essa simplificação na prática não é verdadeira, o que torna o método mais conservador no quesito segurança.

De acordo com Gerscovich (2016), fator de segurança, nesse método, é definido como a razão entre a soma das resistências ao cisalhamento disponíveis ao longo da superfície de ruptura e a soma das forças solicitantes responsáveis pelo movimento da massa de solo. A resistência ao cisalhamento é expressa em função dos parâmetros de resistência do solo, como a coesão e o ângulo de atrito interno, enquanto as forças solicitantes estão associadas principalmente ao peso das fatias e à inclinação da superfície de ruptura, como mostrado na Equação 1.

$$FS = \frac{R}{S} = \frac{\sum\left[c' \cdot \left(\frac{b}{cos\theta_i}\right) + \left(W_i \cdot cos\theta_i - u_i \cdot \left(\frac{b}{cos\theta_i}\right)\right) \cdot tan\phi'\right]}{\sum W_i \cdot sen\theta_i}, \qquad i = 1,2,\dots,n. \tag{1}$$

Onde: n é o número de fatias verticais, c' é a coesão do solo, b é a largura das fatias, W_i é o peso próprio da i-ésima fatia, u_i é a parcela de poropressão da i-ésima fatia, ϕ' é o ângulo de atrito interno do solo e θ_i é o ângulo formado entre o ponto central da base da i-ésima fatia e o centro do círculo que descreve a superfície de ruptura. Vale ressaltar que c' e ϕ' são tomados com os valores médios provenientes de ensaios amostrais prévios.

Outro aspecto relevante do método de Fellenius diz respeito à definição da superfície de ruptura. A precisão da análise depende fortemente da escolha dessa superfície, uma vez que diferentes geometrias podem conduzir a valores significativamente distintos de fator de segurança. Assim, a identificação da superfície crítica de ruptura, associada ao menor fator de segurança, torna-se um ponto-chave da análise, motivando o uso de técnicas computacionais para automatizar e otimizar o processo de busca.

2.3 Problemas de otimização

De acordo com Silva (2001), entende-se por otimização as várias técnicas usadas para encontrar o melhor resultado de uma função a partir da combinação dos valores de suas variáveis, considerando as restrições impostas ao problema. Podendo ser uma otimização de minimização, quando o melhor valor é o mínimo global da função a ser otimizada, ou de maximização, quando o valor ideal se trata do máximo global da função em questão. Um clássico exemplo de problema de otimização é o do "caixeiro viajante", onde um certo homem precisa visitar um número C de cidades, passando por cada cidade uma única vez e finalizando sua jornada na cidade inicial, devendo escolher a rota mais curta possível. Linden (2006 *apud* Silva, 2024) explica que, para esse problema, o número de rotas possíveis entre as cidades é de $C!$, dessa forma, a complexidade do problema aumenta à medida que mais cidades precisarem ser visitadas, por exemplo, existem apenas 6 rotas possíveis para 3 cidades, enquanto que para 10 cidades a quantidade de rotas é superior a 3,5 milhão. Assim, a busca pela solução ótima de um problema complexo pode exigir o uso de técnicas computacionais, visto que em alguns casos torna-se impossível o cálculo direto de todas as possibilidades e a aplicação de métodos numéricos pode ser ineficiente para problemas não lineares.

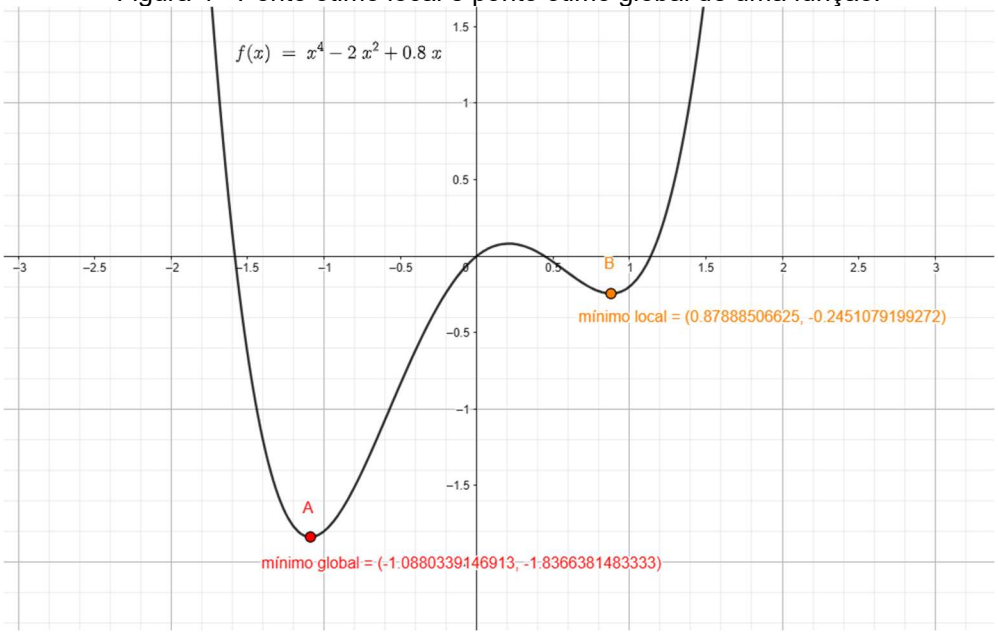
Problemas dessa natureza estão presentes em diversas áreas do conhecimento, como engenharia, finanças, nutrição, logística, entre outras. Isso se deve ao fato de que a busca por soluções mais eficientes para a realização de atividades está presente nas mais variadas áreas de atuação humana.

De maneira geral, um problema de otimização é composto por três elementos principais: as variáveis de projeto, que representam os parâmetros passíveis de alteração; a função objetivo, que é a função a ser minimizada ou maximizada; e as restrições, que definem os limites físicos, geométricos ou operacionais do problema.

2.3.1 Variáveis de projeto

Silva (2024) explica que as variáveis de projeto são as variáveis presentes em uma função a ser otimizada, podendo ser representadas por um vetor solução $\boldsymbol{x} = [x_1, x_2, \dots, x_n]^T$, sendo n o número de variáveis de projeto. Quando os valores das variáveis de projeto retornam o valor ótimo global buscado na otimização, a representação do vetor solução passa a ser $\boldsymbol{x}^* = [x_1{}^*, x_2{}^*, \dots, x_n{}^*]^T$. Considerando o espaço das soluções de uma função, define-se como um ponto ótimo local aquele que representa a melhor solução em uma vizinhança desse ponto, enquanto um ponto ótimo global é o melhor valor de todo o espaço de soluções (Kirsch, 1993 apud Silva, 2001). A Figura 4 exemplifica a concepção de ótimos locais e globais para um caso de minimização, mostrando tais pontos para função $f(x) = x^4 - 2x^2 + 0{,}8x$, representada graficamente por meio da versão online e gratuita do *software GeoGebra*.

Figura 4 - Ponto ótimo local e ponto ótimo global de uma função.



Fonte: o próprio autor (2026).

De acordo com o exemplo da Figura 4, o ponto B com $\boldsymbol{x} \cong [0{,}879]$ representa um ótimo local pois é o menor valor da função na região em que x ∈ [0, 1], enquanto o ponto A com $\boldsymbol{x}^* \cong [-1{,}088]$ representa o ótimo global da função, pois em todo seu domínio nenhum outro valor da variável de projeto retornará um valor menor de solução para a função.

2.3.2 Função objetivo

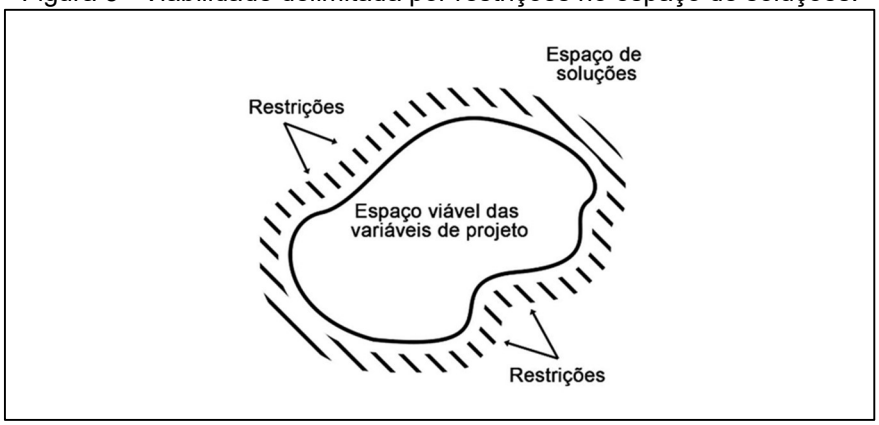
A função $f(\boldsymbol{x})$ da qual se deseja encontrar o ótimo global (máximo ou mínimo) em um problema de otimização é denominada função objetivo. Quando há apenas uma função objetivo a ser otimizada, o problema é classificado como otimização de objetivo único (*Single Objective Optimization*), por outro lado, quando o problema envolve a otimização simultânea de duas ou mais funções objetivo, ele é caracterizado como um problema de otimização multiobjetivo (*Multi Objective Optimization*) (Silva, 2024).

Para sua resolução, podem ser empregados algoritmos baseados em gradientes, métodos heurísticos e estratégias de busca determinísticas ou estocásticas. Nos métodos determinísticos, uma mesma condição inicial conduz sempre ao mesmo resultado, enquanto nos métodos estocásticos a presença de componentes aleatórios pode produzir soluções distintas para uma mesma entrada (Deb, 2001).

2.3.3 Funções de restrição

Os problemas de otimização geralmente possuem algumas restrições nos valores que as variáveis de projeto podem assumir, essas restrições limitam o espaço de soluções em regiões de valores viáveis para a otimização, fora dessas regiões a combinação de variáveis de projeto assumem valores que desrespeitam alguma restrição imposta, ocasionando soluções inviáveis para o problema. O efeito das restrições no espaço de soluções de um problema de otimização é exemplificado na Figura 5.

Figura 5 - Viabilidade delimitada por restrições no espaço de soluções.



Fonte: Silva (2024).

Silva (2001) explica que as restrições podem ser geométricas ou de comportamento. As restrições geométricas definem diretamente os limites (superior e inferior) de cada variável de projeto e devem ser impostas mesmo que o problema de otimização não especifique limites, isto para evitar um espaço infinito de soluções. Já as restrições de comportamento limitam as variáveis de projeto indiretamente através de funções de desigualdade ($g_i(\boldsymbol{x}) \leq 0$) e/ou de igualdade ($h_k(\boldsymbol{x}) = 0$), um problema de otimização pode não impor nenhum tipo de restrição de comportamento.

As funções de restrição podem envolver uma ou mais variáveis de projeto e ambas devem ser iguais ou desiguais ao mesmo valor, preferencialmente a zero, para facilitar a análise do descumprimento das restrições na otimização.

2.3.4 Formulação padrão dos problemas de otimização

Silva (2024) explica que, de maneira padronizada, os problemas de otimização podem ser formulados de acordo com a Equação 2, para o caso de minimização.

$$\begin{aligned} &\text{Minimize} \quad f(\boldsymbol{x}). \\ &\text{Sujeito a} \quad g_j(\boldsymbol{x}) \leq 0 \ , \qquad j = 1,2,\dots p. \\ &\qquad\qquad\ \ h_k(\boldsymbol{x}) = 0 \ , \qquad k = 1,2,\dots q. \\ &\qquad\qquad\ \ x_i^l \leq x_i \leq x_i^u \ , \qquad i = 1,2,\dots n. \end{aligned} \tag{2}$$

Onde f é a função objetivo; g_j são as j-ésimas restrições de comportamento de desigualdade em quantidade igual a p; h_k são as k-ésimas restrições de comportamento de igualdade em quantidade igual a q; x_i^l e x_i^u são respectivamente os i-ésimos limites superior e inferior da i-ésimas variáveis de projeto x_i em quantidade igual a n.

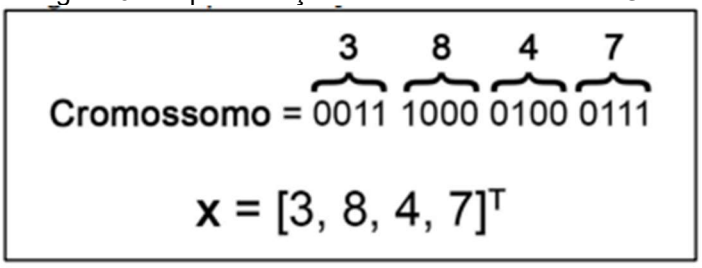
2.4 Algoritmo Genético Binário

Os algoritmos genéticos são métodos de otimização e busca inspirados nos princípios da seleção natural propostos por Darwin, sendo classificados como técnicas de computação evolutiva. Propostos originalmente por Holland (1992) em meados dos anos 60 e posteriormente difundidos por Goldberg (1989), esses algoritmos simulam mecanismos biológicos, como seleção natural, recombinação genética e mutação, com o objetivo de evoluir sucessivas populações de soluções candidatas até obter indivíduos com o melhor desempenho em relação a um determinado problema.

Diferentemente dos métodos clássicos de otimização, que frequentemente dependem do cálculo de derivadas ou da continuidade da função a ser otimizada, os algoritmos genéticos realizam uma busca estocástica baseada na evolução de uma população de indivíduos. Essa característica permite sua aplicação em problemas com funções não lineares, descontínuas ou multimodais, além de reduzir a probabilidade de convergência para ótimos locais quando comparados a métodos determinísticos tradicionais (Goldberg, 1989).

Linden (2006) explica que a representação das soluções deve ser montada a partir de elementos simples, assim como o DNA (*Deoxyribonucleic Acid*) humano, portanto, em um algoritmo genético binário (AGB) as soluções são formadas apenas por caracteres 0 e 1. Essas cadeias de *bits* simulam cromossomos (ou indivíduos), onde cada *bit* é um gene e cada variável do problema a ser otimizada é representada por um grupo de genes, que por sua vez devem ser normalizados em valores do espaço de busca natural do problema, como mostrado na Figura 6 (Silva, 2024).

Figura 6 - Representação cromossomial em um AGB.



Fonte: Silva (2024).

O processo evolutivo inicia-se com a geração de uma população inicial de indivíduos, normalmente criada de forma aleatória dentro dos limites estabelecidos para o problema, a fim de promover diversidade no espaço de busca. Cada indivíduo é então avaliado por meio de uma função de aptidão (*fitness*), responsável por quantificar a qualidade da solução representada pelo cromossomo (Deb, 2001). A função de aptidão leva em conta a função objetivo e as restrições do problema de otimização. Quanto melhor é a aptidão de um indivíduo em relação aos demais, maior sua probabilidade de ser selecionado para gerar descendentes nas gerações seguintes.

Mezura-Montes e Coello (2005) descrevem um tipo de formulação para a função de aptidão de um problema de minimização, considerando as restrições no processo de avaliação dos cromossomos, de acordo com a Equação 3.

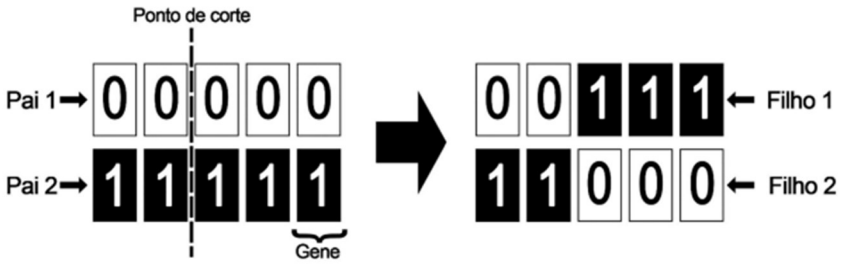
$$Fitness(\boldsymbol{x}) = \begin{cases} f(\boldsymbol{x}), & se\ \boldsymbol{x}\ \text{é}\ viável. \\ \sum_{j=1}^{p} \begin{cases} g_j(\boldsymbol{x}), & g_j(\boldsymbol{x}) > 0 \\ 0, & g_j(\boldsymbol{x}) \leq 0 \end{cases} + \sum_{k=1}^{q} |h_k(\boldsymbol{x})|, & se\ \boldsymbol{x}\ \text{é}\ inviável. \end{cases} \quad (3)$$

Basicamente, a aptidão de um cromossomo será exatamente igual ao valor da função objetivo avaliada no vetor solução que ele representa, desde que nenhuma restrição seja violada, isto é, desde que o vetor solução seja viável para o problema. Se o vetor solução for inviável, violando uma ou mais restrições, sua aptidão será igual à soma de todas as restrições que foram ultrapassadas; quanto mais inviável, maior será o valor de aptidão e consequentemente pior será aquela solução para uma minimização.

Após a avaliação da população, são aplicados os operadores genéticos responsáveis pela evolução das soluções. Deb (2001) explica que o operador de seleção privilegia indivíduos com melhor aptidão, aumentando suas chances de reprodução, simulando o princípio da sobrevivência dos mais adaptados. Esse operador consiste em técnicas para escolher cromossomos que serão combinados entre si para gerar novos cromossomos, uma dessas técnicas consiste em simular um sorteio de roleta, onde indivíduos mais aptos terão uma porção maior na roleta, tendo mais chances de serem sorteados; outra técnica de seleção consiste em simular um torneio, onde dois indivíduos escolhidos aleatoriamente têm suas aptidões comparadas entre si, vencendo aquele com a melhor aptidão.

Em seguida, o operador de cruzamento (*crossover*) é aplicado, promovendo a recombinação dos genes entre pares de indivíduos selecionados, permitindo a troca de informações e a geração de novas soluções que combinam características dos pais (Holland, 1992). Durante o cruzamento, um ponto de corte é definido separando cada cromossomo em duas partes, depois essas partes são misturadas formando dois novos indivíduos, como mostrado na Figura 7.

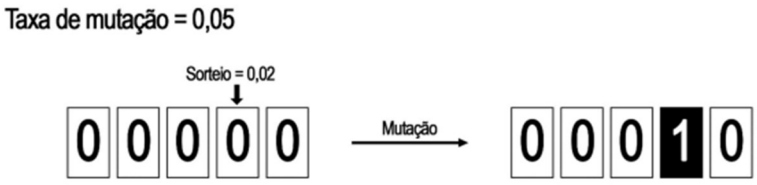
Figura 7 - Cruzamento em um AGB.



Fonte: Silva (2024).

Por fim, o operador de mutação simula o processo de mutação genética que pode acontecer em organismos, existem diversas formas de simular isso em um AG, Araujo e Cervigón (2009) apresentam a mutação *bit* a *bit*, onde a partir de uma taxa de probabilidade um sorteio é realizado para cada gene, e caso o valor sorteado seja igual ou menor que a taxa de probabilidade, a mutação ocorre, invertendo o valor binário do gene, como mostrado na Figura 8.

Figura 8 - Mutação *bit* a *bit* em um AGB.

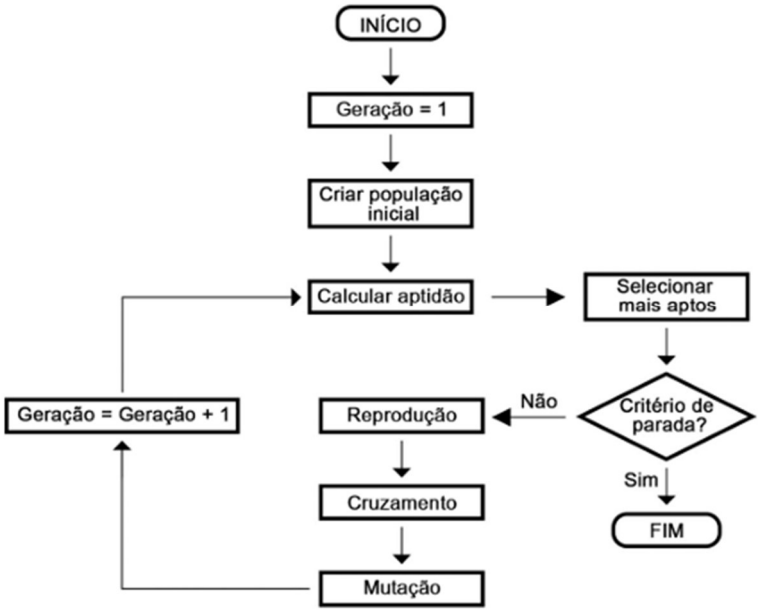


Fonte: Silva (2024).

Ao passar por esses operadores, a população inicial dá início a uma nova geração, com indivíduos que herdam as boas características dos seus antecessores. Além desses operadores, muitos AGs empregam estratégias de elitismo, nas quais os melhores indivíduos de uma geração são preservados para a geração seguinte sem sofrer modificações por cruzamento ou mutação. Essa estratégia contribui para garantir que a qualidade da melhor solução encontrada não seja perdida ao longo do processo evolutivo, favorecendo a convergência do algoritmo.

O processo evolutivo é finalizado quando um critério de parada previamente definido é alcançado, por exemplo, um determinado número de gerações, ou um valor alvo da função objetivo a ser alcançado. A Figura 9 apresenta, de forma esquemática, as principais etapas do funcionamento de um AG.

Figura 9 - Fluxograma do funcionamento de um AG.



Fonte: Silva (2024).

Os AGs apresentam vantagens significativas na resolução de problemas complexos, não lineares e com múltiplos mínimos locais, características frequentemente encontradas em aplicações de engenharia. Uma das vantagens na otimização com um AG está no fato de não ser necessário definir uma solução inicial de onde partirá a busca, como em alguns métodos numéricos, além disso, várias soluções são avaliadas simultaneamente, guiando o processo de busca automaticamente.

2.5 Confiabilidade estrutural

Ferreira (2015) explica que em projetos estruturais, existem pequenas incertezas quanto a carregamentos, condições ambientais, propriedades mecânicas e geométricas, etc. Essas incertezas geram uma pequena probabilidade de falha na estrutura, e deve ser avaliada para garantir a segurança. No contexto da estabilidade de taludes, essas incertezas são particularmente relevantes, uma vez que os parâmetros de resistência e deformabilidade dos solos são obtidos a partir de amostragens limitadas e ensaios que apresentam variabilidade espacial e estatística.

2.5.1 Variáveis aleatórias

No contexto da confiabilidade estrutural, quando os resultados associados a um determinado fenômeno não podem ser previstos com exatidão devido às

incertezas inerentes aos materiais, às ações atuantes, às condições ambientais ou aos processos construtivos, esse fenômeno é classificado como aleatório. Nesses casos, os possíveis resultados de cada experimento são descritos em termos de probabilidades de ocorrência e podem ser representados por uma função real (Bazán, 2018).

Segundo Naghettini e Pinto (2007 *apud* Campello, 2020), essa função real, que associa um valor numérico a cada resultado possível de um experimento aleatório, é denominada variável aleatória. Campello (2020) complementa os conceitos explicando que uma variável aleatória X pode assumir um determinado valor x ($X = x$); se x puder assumir apenas valores finitos ou numeráveis a variável aleatória é considerada discreta, por outro lado, se x puder assumir qualquer valor real a variável aleatória é classificada como contínua.

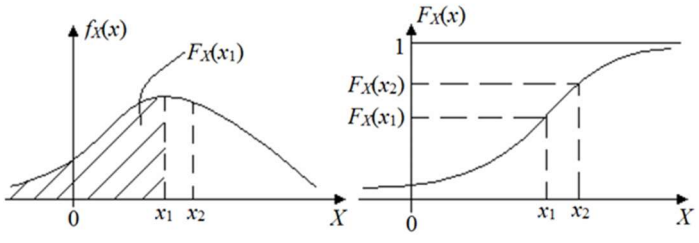
A função que indica a probabilidade de uma variável aleatória X assumir qualquer valor menor ou igual a x, sendo $-\infty \leq x \leq +\infty$, é chamada de função de distribuição cumulativa de probabilidades ou *cumulative distribution function* (CDF). Já a derivada de uma CDF em relação a x, é chamada de função densidade de probabilidade ou *probability density function* (PDF) (Santos, 2021). A CDF é dada pela Equação 4, enquanto a PDF é dada pela Equação 5.

$$F_X(x) = P[\{X \leq x\}], \tag{4}$$

$$f_X(x) = \frac{dF_X(x)}{dx}. \tag{5}$$

A Figura 10 apresenta a representação gráfica da PDF e CDF de uma variável aleatória contínua, sendo que a área abaixo da PDF é igual a 1, representando a probabilidade de 100% do valor da variável aleatória ser menor ou igual a $-\infty$.

Figura 10 - PDF e CDF de uma variável aleatória contínua.



Fonte: adaptado de Lima e Sagrilo (2003 *apud* Bazán,2018).

Outros conceitos importantes relacionados às variáveis aleatórias são a média ou valor esperado (Equação 6), variância (Equação 7), desvio padrão (Equação 8) e coeficiente de variação (Equação 9), ambos bastante conhecidos e essenciais em estudos probabilísticos e de confiabilidade.

$$E[X] = \mu_X = \int_{-\infty}^{+\infty} x \cdot f_X(x)dx, \tag{6}$$

$$Var[X] = E[(X - \mu_X)^2] = \sigma_X^2 = \int_{-\infty}^{+\infty} (x - \mu_X)^2 \cdot f_X(x)dx, \tag{7}$$

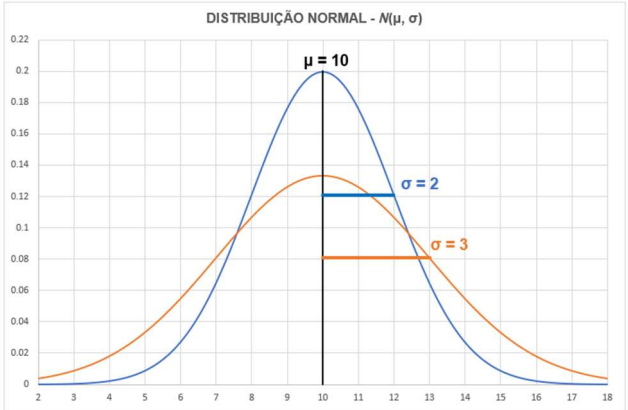
$$\sigma_X = \sqrt{Var[X]}, \tag{8}$$

$$CV_X = \delta_X = \frac{\sigma_X}{\mu_X}. \tag{9}$$

Cada variável aleatória está sujeita a um modelo estatístico que melhor representa a distribuição de probabilidade do fenômeno analisado, e dentre os diversos modelos de distribuição existentes a distribuição normal é a mais pertinente para este trabalho.

Santos (2021) explica que a distribuição normal de uma variável aleatória, isto é, $X \sim N(\mu, \sigma)$, também chamada de distribuição gaussiana, é a mais conhecida e utilizada para descrever fenômenos aleatórios, sendo modelada a partir da média e do desvio padrão de uma variável aleatória, além de possuir simetria em torno do valor médio, o que lhe confere um formato de “sino”, como mostrado graficamente utilizando a versão online e gratuita do *software GeoGebra*, na Figura 11.

Figura 11 - Exemplo de distribuição normal.



Fonte: o próprio autor (2026).

É possível observar na Figura 11 que o vértice da distribuição normal está localizado exatamente no valor da média, representando o resultado mais provável de acontecer. Também é possível notar que, para o mesmo valor de média, valores de desvio padrão menores tornam o formato de “sino” mais estreito e esticado em torno do valor médio, enquanto desvios padrão maiores alargam e achatam o formato, isso porque o desvio padrão representa justamente o quão afastados da média estão os demais valores.

A formulação da PDF e CDF em uma distribuição normal são apresentas respectivamente pela Equação 10 e Equação 11.

$$f_X(x) = \frac{1}{\sigma\sqrt{2 \cdot \pi}} \cdot exp\left[-\frac{1}{2} \cdot \left(\frac{x-\mu}{\sigma}\right)^2\right] \;\; ; \;\; -\infty \leq x \leq +\infty, \tag{10}$$

$$F_X(x) = \int_{-\infty}^{x} \frac{1}{\sigma\sqrt{2 \cdot \pi}} \cdot exp\left[-\frac{1}{2} \cdot \left(\frac{z-\mu}{\sigma}\right)^2\right] dz \;\; ; \;\; -\infty \leq x \leq +\infty. \tag{11}$$

Não é possível resolver a integral da CDF de maneira analítica, apenas numericamente, de forma que para facilitar a busca é possível consultar tabelas de referência disponíveis na literatura. Essas tabelas são construídas usando valores para uma variável aleatória normal padrão Y, com média igual a 0 e desvio padrão igual a 1, assim, é necessário transformar a variável aleatória normal $X \sim N(\mu_X, \sigma_X)$ em uma variável aleatória normal padrão $Y \sim N(0,1)$, por meio da Equação 12.

$$Y = \frac{X - \mu_X}{\sigma_X}. \tag{12}$$

Dessa forma, a PDF padrão e a CDF padrão são dadas respectivamente pela Equação 13 e Equação 14.

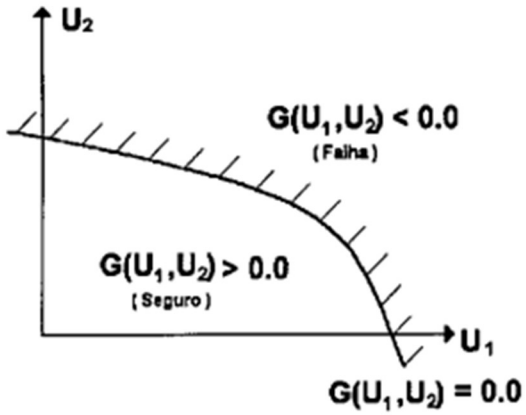
$$f_Y(y) = \phi(y) = \frac{1}{\sqrt{2 \cdot \pi}} \cdot exp\left(-\frac{1}{2} \cdot (y)^2\right) \;\; ; \;\; -\infty \leq y \leq +\infty, \tag{13}$$

$$F_Y(y) = \Phi(y) = \int_{-\infty}^{y} \phi(z)\, dz \;\; ; \;\; -\infty \leq y \leq +\infty. \tag{14}$$

2.5.2 Função de estado limite

A análise de confiabilidade estrutural é baseada em uma função de estado limite (ou função de falha) $G(\boldsymbol{U})$, onde $\boldsymbol{U} = (U_1, U_2, \dots, U_n)$ é o conjunto de todas as variáveis aleatórias, isto é, um ponto no espaço de soluções, de forma que $G(\boldsymbol{U}) = 0$ é a superfície de falha, que separa o domínio de falha ($G(\boldsymbol{U}) < 0$) do domínio seguro $G(\boldsymbol{U}) > 0$ (Sagrilo, 1994), como mostrado na Figura 12.

Figura 12 - Definição de função de estado limite.



Fonte: Sagrilo (1994).

Para exemplificar, o chamado problema fundamental da confiabilidade estrutural, envolve duas variáveis aleatórias independentes e é definido de acordo com a função de estado limite da Equação 15:

$$g(R, S) = R - S. \tag{15}$$

Onde R é a resistência e S é a solicitação existente no problema. A probabilidade de falha, nesse caso, é definida a partir da Equação 16:

$$P_f = P[\{R \leq S\}] = P[\{R - S \leq 0\}] = P[g(R, S) \leq 0], \tag{16}$$

onde $P[\{R \leq S\}]$ é a probabilidade de R ser menor que S e consequentemente da função de estado limite assumir um valor no domínio de falha (Bazán, 2018).

2.5.3 Índice de confiabilidade β

Uma característica interessante nas funções de estado limite é que se $\boldsymbol{U}$ for um conjunto de variáveis aleatórias independentes entre si e ambas assumirem distribuição normal, $G(\boldsymbol{U})$ também será uma variável aleatória de distribuição normal, com média μ_G e desvio padrão σ_G.

Segundo Cornell (1969 *apud* Bazán, 2018), o índice de confiabilidade β, também conhecido como índice de *Cornell*, é uma medida de probabilidade de falha que incorpora as incertezas do problema e indica a distância entre o valor médio da distribuição e a superfície de falha da função de estado limite, admitindo-se que a função de estado limite seja linear e que as variáveis aleatórias possuem distribuição normal sendo definido de acordo com a Equação 17.

$$\beta = \frac{\mu_G}{\sigma_G}. \tag{17}$$

Considerando o problema fundamental da confiabilidade estrutural, temos que $g(R,S)$ é uma variável aleatória, com $\mu_g = \mu_R - \mu_S$ e $\sigma_g = \sqrt{\sigma_R^2 + \sigma_S^2}$, assim podemos definir o índice de confiabilidade para essa função de estado limite de acordo com a Equação 18.

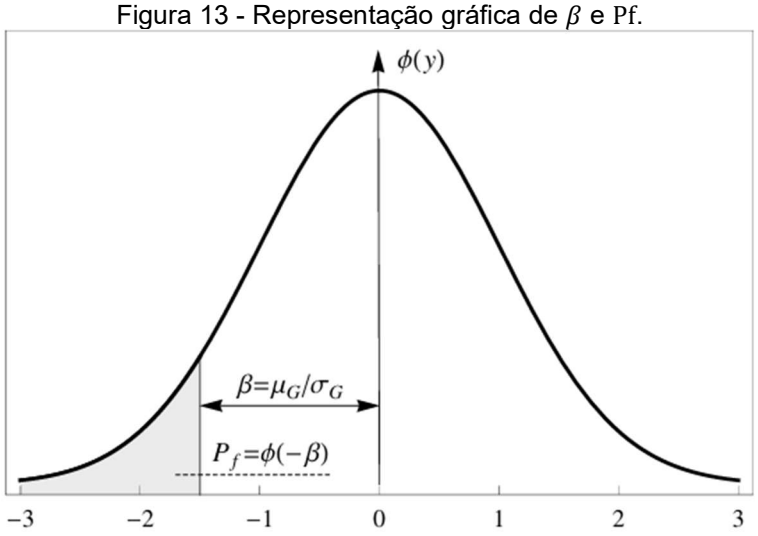
$$\beta_g = \frac{\mu_g}{\sigma_g} = \frac{\mu_R - \mu_S}{\sqrt{\sigma_R^2 + \sigma_S^2}}. \tag{18}$$

O índice β está diretamente relacionado à probabilidade de falha por meio da distribuição normal padrão, uma vez que aplicado na CDF padrão retorna o próprio valor da probabilidade de falha, como mostrado na Equação 19.

$$P_f = \Phi(-\beta). \tag{19}$$

Portanto, quanto maior o valor de β, menor é a probabilidade de falha e maior é o nível de segurança. A Figura 13 mostra a representação gráfica do índice β e a probabilidade de falha Pf, que corresponde à área sombreada abaixo da curva.

Figura 13 - Representação gráfica de β e Pf.



Fonte: Beck (2012).

2.5.4 Método FOSM

Entre os métodos probabilísticos utilizados na análise de confiabilidade estrutural, o método FOSM (*First Order Second Moment*) destaca-se por sua simplicidade de implementação e pelo reduzido custo computacional quando comparado a métodos baseados em simulação, como o método de Monte Carlo. Sua principal característica consiste em estimar a influência das incertezas das variáveis de entrada sobre a resposta do sistema por meio de uma aproximação de primeira ordem da função de estado limite, utilizando apenas os dois primeiros momentos estatísticos das variáveis aleatórias, isto é, suas médias e variâncias (Beck, 2012).

O nome do método deriva justamente dessa característica. O termo *First Order* refere-se à utilização da expansão da função de estado limite em série de Taylor truncada nos termos de primeira ordem, enquanto *Second Moment* indica que a propagação das incertezas é realizada a partir do segundo momento estatístico das variáveis aleatórias, representado pela variância. Dessa forma, o método permite estimar a média e o desvio padrão da função de estado limite sem a necessidade de inúmeras simulações, constituindo uma alternativa eficiente para problemas de engenharia em que se deseja incorporar os efeitos das incertezas ao processo de análise (Beck, 2012).

A ideia fundamental do método é avaliar como as incertezas presentes nas variáveis aleatórias propagam-se até a resposta do sistema. Campello (2020) explica que em problemas geotécnicos, parâmetros como coesão, ângulo de atrito interno e

peso específico apresentam variabilidade natural decorrente da heterogeneidade do solo, além de incertezas associadas às investigações de campo, ensaios laboratoriais e simplificações adotadas na modelagem. Como consequência, a função de estado limite também passa a apresentar comportamento aleatório, tornando possível estimar através do FOSM a probabilidade de ocorrência da falha.

Matematicamente, o método parte da linearização da função de estado limite $G(\boldsymbol{U})$ em torno do vetor formado pelos valores médios das variáveis aleatórias $\boldsymbol{\mu_U} = [\mu_{U_1}, \mu_{U_2}, \dots, \mu_{U_n}]$, utilizando a expansão em série de Taylor de primeira ordem, como mostrado na Equação 20 com sua respectiva forma vetorial.

$$\tilde{G}(\boldsymbol{U}) = G(\boldsymbol{\mu_U}) + \sum_{i=1}^{n} \frac{\partial G(\boldsymbol{U})}{\partial U_i}\bigg|_{\boldsymbol{\mu_U}} \cdot \left(U_i - \mu_{U_i}\right) = G(\boldsymbol{\mu_U}) + \nabla G(\boldsymbol{\mu_U})^T \cdot (\boldsymbol{U} - \boldsymbol{\mu_U}). \quad (20)$$

Essa linearização permite aproximar a resposta do sistema por uma função linear das variáveis de entrada, possibilitando que sua média seja obtida diretamente pela avaliação da função nos valores médios das variáveis. Bazán (2018) e Beck (2012) explicam que, de maneira análoga, a variabilidade da resposta pode ser estimada por meio da propagação das variâncias das variáveis aleatórias, desde que independentes, ponderadas pela sensibilidade da função de estado limite em relação a cada variável, representada por suas derivadas parciais, como é aplicado na Equação 21.

$$Var\left[\tilde{G}(\boldsymbol{U})\right] = \sum_{i=1}^{n} \left(\frac{\partial G(\boldsymbol{U})}{\partial U_i}\bigg|_{\boldsymbol{\mu_U}} \right)^2 = \nabla G(\boldsymbol{\mu_U})^T \cdot \nabla G(\boldsymbol{\mu_U}). \quad (21)$$

Observa-se que as derivadas parciais de $G(\boldsymbol{U})$ correspondem à sensibilidade da resposta em relação às variáveis aleatórias. Assim, quanto maior a influência de determinado parâmetro sobre a função de estado limite, maior será sua contribuição para a variabilidade da resposta. Essa característica permite identificar quais parâmetros exercem maior influência sobre a confiabilidade do sistema analisado.

Obtidas a média e o desvio padrão da função de estado limite linearizada, calcula-se o índice de confiabilidade β. Como abordado anteriormente, valores elevados de β indicam maior afastamento da região de falha e, consequentemente,

maior confiabilidade do sistema, enquanto valores reduzidos estão associados a maiores probabilidades de ocorrência da ruptura.

Beck (2012) aborda sobre outro importante conceito para o método FOSM, a transformação de Hassofer e Lind, que semelhante a padronização apresentada na Equação 12, transforma um vetor de variáveis aleatórias normais $\boldsymbol{U}$ avaliado no espaço $\mathbb{U}$ em um vetor de variáveis aleatórias normais padrão $\boldsymbol{Y}$ avaliado no espaço $\mathbb{Y}$, com média nula e desvio padrão unitário, como mostrado na Equação 22, para as i-ésimas variáveis aleatórias do vetor $\boldsymbol{U}$.

$$Y_i = \frac{U_i - \mu_{U_i}}{\sigma_{U_i}}. \tag{22}$$

Torna-se conveniente trabalhar com a transformação de $\mathbb{U} \rightarrow \mathbb{Y}$ ou de $\mathbb{Y} \rightarrow \mathbb{U}$ na forma matricial, a saber $\boldsymbol{y} = \boldsymbol{D^{-1}}\{\boldsymbol{u} - \boldsymbol{M}\}$ e $\boldsymbol{u} = \boldsymbol{Dy} + \boldsymbol{M}$ onde $\boldsymbol{D}$ é a matriz diagonal dos desvios padrão e $\boldsymbol{M}$ é o vetor transposto das médias das variáveis aleatórias normais. Matrizes Jacobianas também são usadas para facilitar as transformações, são elas $\boldsymbol{J_{yu}} = \boldsymbol{D^{-1}}$ e $\boldsymbol{J_{uy}} = \boldsymbol{D}$.

A ideia principal é sair de um ponto $\boldsymbol{U}$ qualquer e percorrer o espaço de soluções por meio dessas transformações até encontrar um ponto de projeto $\boldsymbol{U}^*$ sobre o domínio de falha que tenha a maior probabilidade de ocorrência, através de um processo de otimização. Para isso, Bazán (2018) sugere a utilização do algoritmo simples de Hasofer, Lind, Rackwitz e Fiessler (HLRF), que se baseia na aproximação de um ponto qualquer à superfície de falha em k-ésimas iterações, tal algoritmo é apresentado na Equação 23.

$$\boldsymbol{U}_{k+1} = \frac{\nabla G(\boldsymbol{U}_k)^T \boldsymbol{U}_k - G(\boldsymbol{U}_k)}{\|\nabla G(\boldsymbol{U}_k)\|^2} \nabla G(\boldsymbol{U}_k). \tag{23}$$

Assim, Bazán (2018) e Beck (2012) resumem o passo a passo do algoritmo de solução pelo método FOSM como:

1. Avaliação das matrizes Jacobianas ($\boldsymbol{J_{yu}}$ e $\boldsymbol{J_{uy}}$) e do vetor de médias $\boldsymbol{M}$.
2. Escolha das médias $\boldsymbol{\mu_U}$ como ponto inicial $\boldsymbol{u_k}$ para $k = 0$.
3. Transformação do ponto $\boldsymbol{u_0}$ de $\mathbb{U}$ para $\mathbb{Y}$ e avaliação do β_0.

4. Avaliação de $g(\boldsymbol{u_k})$.
5. Cálculo do vetor gradiente:
    a) Cálculo das derivadas parciais de $g(\boldsymbol{u})$ no espaço $\mathbb{U}$.
    b) Transformação do gradiente para o espaço $\mathbb{Y}$.
    c) Cálculo dos fatores de sensibilidade.
6. Cálculo de um novo ponto $\boldsymbol{y}_{k+1}$ pelo algoritmo HLRF.
7. Avaliação do índice de confiabilidade β_{k+1}.
8. Transformação de $\boldsymbol{y}_{k+1}$ para o espaço $\mathbb{U}$.
9. Verificação do critério de convergência. Se $\frac{\|y_{k+1}\|-\|y_k\|}{\|y_{k+1}\|} \leq tolerância$ o algoritmo é interrompido, se não, repetem-se os passos 4 a 9 até a convergência.
10. Ao final, o β da última iteração é avaliado e usado para definir a probabilidade de falha $Pf = \Phi(-\beta)$.

3 METODOLOGIA

O presente trabalho é estruturado como uma pesquisa de natureza aplicada, uma vez que busca desenvolver e avaliar uma ferramenta computacional para otimização de problemas práticos da Engenharia Civil relacionados à análise de estabilidade de taludes incorporando os efeitos das incertezas inerentes aos parâmetros geotécnicos. Possui uma abordagem quantitativa, focada na modelagem matemática de problemas e na avaliação numérica dos resultados. Em relação aos objetivos, o trabalho possui caráter exploratório e metodológico, uma vez que propõe a integração entre o método de equilíbrio limite de Fellenius, o método probabilístico FOSM e um algoritmo genético binário, visando ao desenvolvimento de uma metodologia capaz de identificar automaticamente superfícies de ruptura associadas às maiores probabilidades de falha. Os procedimentos técnicos deste trabalho são fundamentados em implementações computacionais e simulações numéricas.

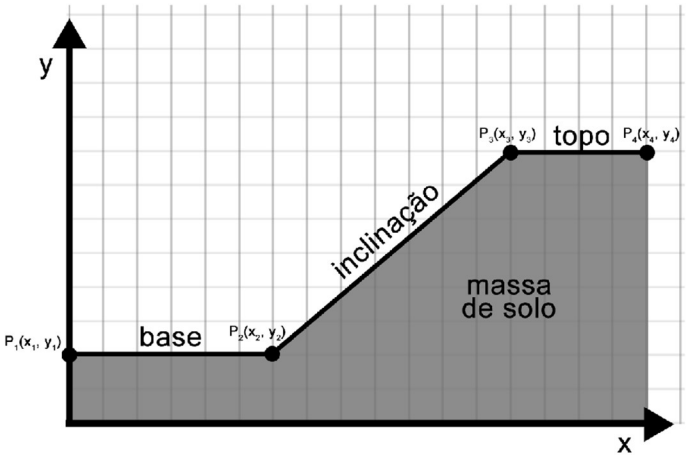
3.1 Caracterização do problema

O trabalho aborda a análise de estabilidade de taludes em solos homogêneos, considerando uma seção transversal bidimensional simplificada para representar o talude. Para caracterização dos parâmetros geométricos, como altura e inclinação, a seção bidimensional é atribuída a um plano cartesiano, onde as coordenadas (X, Y)

dos pontos de mudança de direção da superfície do talude são conectadas por linhas retas, dessa forma facilitando a modelagem do problema em *Python*.

Para a análise, considera-se que a ruptura ocorre por meio de superfícies circulares, definidas por coordenadas de centro e valores de raio. Adota-se regime estático de carregamento, sem considerar efeitos de poropressão e ações externas. A Figura 14 mostra um exemplo de representação de um talude que pode ser analisado neste trabalho.

Figura 14 - Representação simplificada de um talude no plano cartesiano.



Fonte: o próprio autor (2026).

O critério para avaliação da estabilidade de um talude é a probabilidade de falha estimada pelo fator de segurança utilizando o método FOSM, onde cada superfície circular analisada resulta em características distintas de estabilidade, sendo considerada a superfície crítica de ruptura aquela que apresenta a maior probabilidade de falha.

3.2 Cálculo do fator de segurança

O cálculo do fator de segurança das superfícies de ruptura analisadas é obtido através do método das fatias de Fellenius, onde a massa de solo delimitada pela superfície circular de ruptura e pelo perfil do terreno é discretizada em um determinado número de fatias verticais.

Por meio da geometria de cada fatia, compreendida computacionalmente através das coordenadas no plano cartesiano, e utilizando definições matemáticas de retas, circunferências, distância entre pontos e interseções de funções, é possível

calcular as forças atuantes em cada fatia, considerando os parâmetros de resistência do solo previamente definidos em cada problema.

O fator de segurança é definido como a razão entre a soma das resistências ao cisalhamento disponíveis ao longo da superfície de ruptura e a soma das forças solicitantes que tendem a provocar o movimento de deslizamento. Dessa forma, para cada superfície circular analisada, obtém-se um valor de FS, o qual é utilizado como base para avaliação da estabilidade do talude através da abordagem probabilística.

O procedimento de cálculo será implementado computacionalmente, permitindo a avaliação automática do fator de segurança para diferentes superfícies de ruptura geradas pelo algoritmo genético

3.3 Análise probabilística

Considerando a variabilidade inerente aos parâmetros geotécnicos devido à heterogeneidade natural dos solos, às limitações dos processos de investigação geotécnica e às simplificações adotadas em modelagens, foi incorporada nesse trabalho uma abordagem probabilística como a principal ferramenta de análise da estabilidade de taludes. Para isso, os parâmetros de resistência do solo, representados pela coesão efetiva c', pelo ângulo de atrito interno efetivo ϕ' e pelo peso específico γ, foram tratados como variáveis aleatórias $\boldsymbol{X} = [c', \phi', \gamma]$, sendo definidos por seus respectivos valores médios e desvios padrão. Admitiu-se, para fins de simplificação, e para o uso do método FOSM, que essas variáveis são estatisticamente independentes e seguem distribuição normal.

A análise probabilística foi realizada por meio do método FOSM (*First Order Second Moment*), o qual permite estimar a média e o desvio padrão da função de estado limite a partir de uma aproximação de primeira ordem baseada na expansão em série de Taylor. Para implementação nesse trabalho, a função de estado limite foi definida como:

$$g(\boldsymbol{X}) = FS - 1,$$

em que FS corresponde ao fator de segurança obtido pelo método de Fellenius considerando os valores das variáveis aleatórias, vale lembrar que o valor mínimo para o fator de segurança é 1. Dessa forma, a condição de ruptura ocorre quando:

$$g(\boldsymbol{X}) \leq 0,$$

com a média e o desvio padrão de $g(\boldsymbol{X})$ obtidos pelo FOSM, o índice β e a probabilidade de falha são calculados. Assim, diferentemente da abordagem

determinística, na qual a superfície crítica corresponde àquela associada ao menor fator de segurança, a abordagem probabilística adotada busca identificar a superfície de ruptura associada à maior probabilidade de ocorrência da falha.

Dessa forma, a metodologia proposta permite incorporar explicitamente os efeitos das incertezas geotécnicas à análise de estabilidade de taludes, fornecendo uma medida adicional de desempenho por meio da probabilidade de falha e possibilitando a identificação de superfícies críticas de ruptura sob uma perspectiva probabilística.

3.4 Aplicação do algoritmo genético binário

O AGB foi implementado e aplicado com o objetivo de identificar a superfície crítica de ruptura associada a maior probabilidade de falha dos taludes analisados. Para isso, cada indivíduo da população representa uma superfície circular de ruptura, definida a partir de suas variáveis de projeto, a saber: coordenadas do centro do círculo (x_c,y_c) e de seu raio (R_c), respeitando os limites geométricos previamente estabelecidos de acordo com a geometria do talude. Os parâmetros que definem cada superfície de ruptura são codificados em cadeias binárias, permitindo a aplicação dos operadores genéticos de seleção, cruzamento e mutação. A população inicial é gerada de forma aleatória, assegurando a variabilidade genética e de características.

A função objetivo a ser otimizada consiste na maximização da probabilidade de falha calculada para cada indivíduo por meio do FOSM, com base no FS obtido pelo método de Fellenius, ambos implementados previamente. As superfícies de ruptura devem ser geometricamente admissíveis. Para isso, são definidas restrições de posicionamento, priorizando que a superfície circular intercepte o perfil do talude em dois pontos no mínimo. As superfícies que não atendem aos critérios geométricos estabelecidos são penalizadas, de modo a direcionar a busca para soluções fisicamente plausíveis.

Ao longo do processo evolutivo, os indivíduos são avaliados e selecionados de acordo com seu desempenho, sendo aplicados os operadores de cruzamento e mutação para a geração de novas soluções. O algoritmo é executado até que seja atingido o número máximo de gerações e, ao final do processo, a solução encontrada com a maior probabilidade de falha, respeitando as restrições, é considerada a superfície crítica de ruptura do talude. De forma resumida, o AGB é responsável por gerar novas superfícies variando suas características geométricas, cada uma dessas

superfícies é submetida à análise probabilística e avaliada com base nisso, o AGB entende quais soluções são mais aptas e passa suas características geométricas adiante.

3.4.1 Função objetivo e função de aptidão

A definição da função objetivo é uma das etapas mais importantes no processo de otimização, uma vez que estabelece o critério utilizado pelo AGB para avaliar a qualidade das soluções candidatas. Em abordagens determinísticas aplicadas à estabilidade de taludes, é comum que a SCR seja definida como aquela associada ao menor FS. Entretanto, conforme discutido anteriormente, a consideração exclusiva do FS pode não representar adequadamente o nível de segurança do talude, uma vez que desconsidera as incertezas inerentes aos parâmetros geotécnicos.

Assim, a função objetivo foi estabelecida em termos da PF estimada pelo método FOSM para cada superfície potencial de ruptura analisada. Adicionalmente, foram incorporadas penalizações às superfícies consideradas inviáveis do ponto de vista físico ou geométrico, como superfícies que não interceptam o talude em pelo menos dois pontos e superfícies que interceptam o talude fora de intervalos predefinidos.

Essas restrições têm como objetivo evitar que o algoritmo direcione a busca para soluções incompatíveis com o mecanismo esperado de ruptura, superfícies que interceptam inadequadamente a geometria do talude ou mecanismos de ruptura considerados improváveis. A incorporação dessas penalizações permite restringir o espaço de busca do algoritmo genético, favorecendo a obtenção de superfícies de ruptura fisicamente admissíveis e contribuindo para a estabilidade numérica do processo de otimização.

A função objetivo do problema de otimização pode ser representado por:

$$\text{Maximize} \qquad Pf = P[\{FS(x_c, y_c, R_c) - 1 \leq 0\}],$$

$$\text{Sujeito a} \qquad g(x_c, y_c, R_c) = D_1(x_c, y_c, R_c) + D_2(x_c, y_c, R_c) \leq 0,$$

$$h(x_c, y_c, R_c) = |I(x_c, y_c, R_c) - 2| = 0,$$

$$x_c^l \leq x_c \leq x_c^u,$$

$$y_c^l \leq y_c \leq y_c^u,$$

$$R_c^l \leq R_c \leq R_c^u,$$

sendo $D_1(x_c, y_c, R_c)$ a distância que a primeira interseção está do primeiro intervalo de interseção definido no talude; $D_2(x_c, y_c, R_c)$ a distância que a segunda interseção está do segundo intervalo de interseção definido no talude e $I(x_c, y_c, R_c)$ a quantidade de interseções existentes entre a superfície de ruptura e o talude. Todas essas medidas são encontradas em função das variáveis de projeto (x_c, y_c, R_c). Assim, $g(x_c, y_c, R_c)$ é a restrição referente aos intervalos de interseção do talude, se as interseções estiverem dentro do limite nenhum valor é somada, mas à medida que se afastem dos limites definidos maiores será a soma da restrição. Já $h(x_c, y_c, R_c)$ garante que a superfície seja penalizada caso não tenha exatamente duas interseções.

Para definir a função de aptidão (ou função *fitness*), apenas superfícies que desrespeitam a restrição $h(x_c, y_c, R_c)$ são consideradas inviáveis, e pela impossibilidade de definir o FS pelo método de Fellenius nesse caso, a aptidão dessas superfícies é apenas uma penalização muito alta. Assim, a função de aptidão é dada por:

$$Aptidão(x_c, y_c, R_c) = \begin{cases} Pf - g(x_c, y_c, R_c), & se\ é\ viável. \\ 1000, & se\ é\ inviável. \end{cases}$$

3.5 Ferramentas computacionais

Os métodos descritos neste trabalho foram integrados em uma ferramenta computacional desenvolvida em linguagem *Python*, com o objetivo de facilitar o cálculo do fator de segurança e a busca pela superfície crítica de ruptura. A escolha da linguagem deve-se à sua grande utilização em aplicações científicas de otimização e a facilidade de integração à outras linguagens.

O algoritmo geral foi modularizado em classes, separando o algoritmo que calcula o fator de segurança pelo método de Fellenius do algoritmo genético binário e do método FOSM, com a camada de iteração com o usuário também sendo modular. Essa organização facilita a manutenção do código e possibilita futuros aprimoramentos da ferramenta, como a incorporação de outros métodos de análise de estabilidade.

Com o intuito de facilitar a utilização da ferramenta, foi desenvolvida uma interface simples apresentada no terminal do Python e a possibilidade de salvar dados da otimização em um documento .txt

3.6 Avaliação e comparação de resultados

Após a implementação das metodologias determinística e probabilística, procedeu-se à avaliação dos resultados obtidos, buscando analisar o comportamento do algoritmo genético e verificar a influência das incertezas geotécnicas na identificação das superfícies críticas de ruptura.

Inicialmente, realizou-se a validação da implementação do método de Fellenius de forma isolada, comparando os fatores de segurança obtidos pelo programa desenvolvido com valores determinados por meio de um procedimento parcialmente manual utilizando a versão gratuita online do software *GeoGebra*. Essa etapa teve como objetivo verificar a consistência dos cálculos implementados e garantir a correta determinação do fator de segurança das superfícies analisadas. Em seguida, avaliou-se o desempenho do AGB em uma abordagem determinística, considerando como critério de otimização a minimização do fator de segurança. Nessa etapa, foram analisados aspectos relacionados à convergência do processo evolutivo, à estabilidade das soluções encontradas e à coerência geométrica das superfícies críticas obtidas.

Posteriormente, aplicou-se a metodologia probabilística proposta, na qual a função objetivo passou a ser definida em termos da probabilidade de falha associada às superfícies candidatas. Para essa análise, foram considerados diferentes conjuntos de parâmetros geotécnicos, permitindo investigar a influência das propriedades do solo e de suas incertezas sobre os resultados obtidos.

Para auxiliar a interpretação dos resultados, foram elaborados gráficos contendo a evolução do melhor indivíduo ao longo das gerações, bem como representações gráficas das superfícies críticas encontradas para cada cenário analisado. Também foram utilizados arquivos de saída gerados automaticamente pelo programa, contendo os melhores indivíduos de cada geração, permitindo uma análise mais detalhada do comportamento do processo evolutivo.

4 RESULTADOS E DISCUSSÕES

4.1 Validação do método de Fellenius implementado

Com o objetivo de verificar a correta implementação computacional do método de Fellenius, realizou-se inicialmente uma análise determinística utilizando apenas a classe responsável pelo cálculo do fator de segurança, sem a utilização das

ferramentas de otimização implementadas neste trabalho. Essa etapa é importante, pois permite avaliar isoladamente o funcionamento do método de equilíbrio limite empregado, garantindo que os resultados obtidos posteriormente pelo processo de otimização não sejam influenciados por possíveis inconsistências na implementação do cálculo do fator de segurança.

Para isso, foi criado um objeto da classe “Fellenius” (presente no Apêndice C) para o talude apresentado por Tenório *et al.* (2018), utilizado os atributos presentes na Tabela 1.

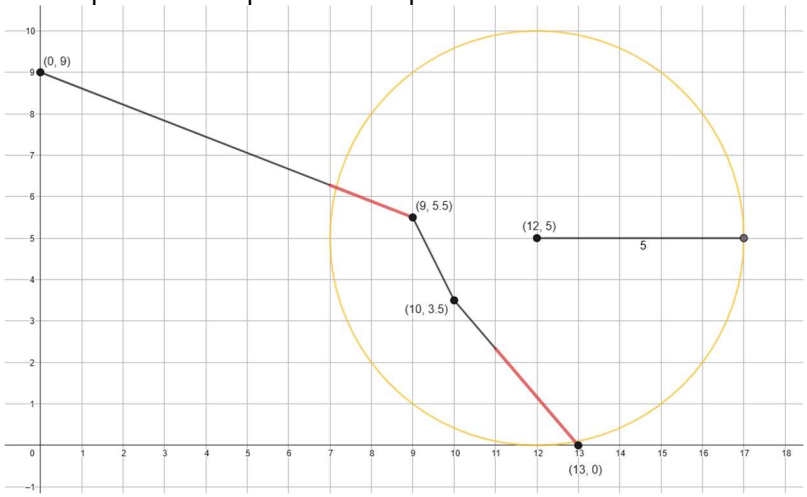
Tabela 1 – Atributos do solo para o cálculo do FS.

Atributos	**Valores**
Pontos do talude	(0; 9), (9; 5,5), (10; 3,5), (13; 0)
Nº de fatias	8
c’ (kPa)	4,15
ϕ' (°)	35,05
γ (kN/m^3)	20,71
Intervalos de interseção	[7; 9], [11; 13]

Fonte: adaptado de Tenório *et al* (2018).

Uma superfície circular de ruptura foi definida arbitrariamente, apenas para fins de validação, essa superfície possui centro em (12; 5) e raio de 5 m, respeitando os intervalos de interseção, que são os segmentos vermelhos representados na Figura 15.

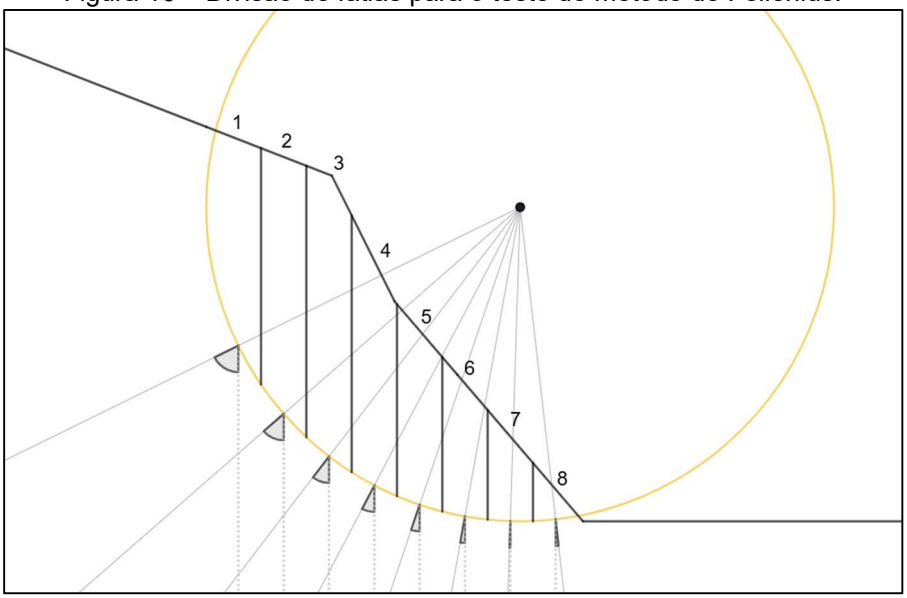
Figura 15 – Talude e superfície de ruptura usadas para teste do método de Fellenius implementado.



Fonte: o próprio autor (2026).

Ao executar o código desenvolvido para o método de Fellenius, o fator de segurança retornado foi $\boldsymbol{FS = 1,2271}$, com interseções entre talude e superfície de ruptura aproximadas em (7,15; 6,22) e (12,92; 0,09), não desrespeitando o limite predeterminado para tais interseções. A fim de validação, o mesmo cálculo foi realizado de maneira parcialmente manual, com auxílio da versão online e gratuita do *software Geogebra*. A Figura 16 mostra a divisão e numeração das fatias, cada uma com largura aproximada de 0,7219 m, mostra também a representação dos ângulos centrais θ em cada fatia.

Figura 16 – Divisão de fatias para o teste do método de Fellenius.



Fonte: o próprio autor (2026).

Os valores necessários para o cálculo do FS (Equação 1), encontrados de maneira parcialmente manual, são apresentados na Tabela 2.

Tabela 2 – Características das fatias no teste do método de Fellenius.

Fatia	Área (m²)	W (kN)	θ (°)	$\cos(\theta)$	$sen(\theta)$	R (kN)	S (kN)
1	2,2374	46,3366	63,8487	0,4407	0,8976	21,1239	41,5932
2	2,9162	60,3945	48,8735	0,6577	0,7533	32,4209	45,4927
3	3,0335	62,8238	37,5089	0,7933	0,6089	38,7367	38,2524
4	2,5772	53,3738	27,6785	0,8856	0,4645	36,5406	24,7927
5	1,9892	41,1963	18,6711	0,9474	0,3201	30,5410	13,1884
6	1,5162	31,4005	10,1229	0,9844	0,1758	24,7281	5,5190
7	0,9646	19,9769	1,7986	0,9995	0,0314	17,0044	0,6270

8	0,3352	6,9420	6,4876	0,9936	0,1130	7,8539	0,7844
TOTAL						208,9495	170,2498

Fonte: o próprio autor (2026).

A tangente do ângulo de atrito é $\tan(35{,}05°) = 0{,}7015$ e para o cálculo de resistências e solicitações foi considerada a ausência de água subterrânea no talude avaliado, portanto, a parcela de poropressão da equação do FS é nula ($u = 0$). Dessa forma, o FS é dado por:

$$FS = \frac{R}{S} = \frac{208{,}9495}{170{,}2498} = 1{,}2273.$$

A proximidade entre os valores encontrados indica que a implementação desenvolvida reproduz adequadamente o comportamento esperado do método de Fellenius, apresentando diferenças atribuídas principalmente às aproximações geométricas inerentes ao processo de discretização e ao arredondamento numérico empregado no cálculo parcialmente manual. Dessa forma, considera-se validada a implementação do método, possibilitando sua utilização nas etapas posteriores do trabalho, nas quais o cálculo do FS é integrado à análise probabilística e ao processo de otimização por algoritmo genético.

4.2 Validação do algoritmo genético binário implementado.

Essa etapa teve como objetivo avaliar a capacidade do AGB implementado em identificar automaticamente a superfície crítica de ruptura de um talude, verificando se o processo de otimização é capaz de convergir para soluções coerentes e que respeitem as restrições impostas. Para essa finalidade, optou-se por utilizar a abordagem determinística, na qual a superfície crítica é definida como aquela que apresenta o menor FS. Dessa forma, a função de aptidão do AGB foi estabelecida de modo a favorecer indivíduos associados a menores valores de FS, considerando simultaneamente penalizações aplicadas a superfícies geometricamente inviáveis ou pouco representativas do mecanismo real de ruptura.

A utilização da abordagem determinística nesta etapa de validação permite avaliar isoladamente o desempenho do algoritmo genético, uma vez que elimina a influência das incertezas probabilísticas introduzidas posteriormente pelo método FOSM. Assim, torna-se possível verificar se o processo evolutivo implementado é

capaz de explorar adequadamente o espaço de busca e direcionar a população para regiões associadas às superfícies potencialmente mais críticas.

Essa validação foi feita utilizando o mesmo talude e parâmetros da seção 4.1, ou seja, o objeto criado na classe “Fellenius” foi mantido e um outro objeto foi criado, dessa vez na classe “*BinaryGeneticOperators*” (Apêndice B), os atributos desse novo objeto são apresentados na Tabela 3.

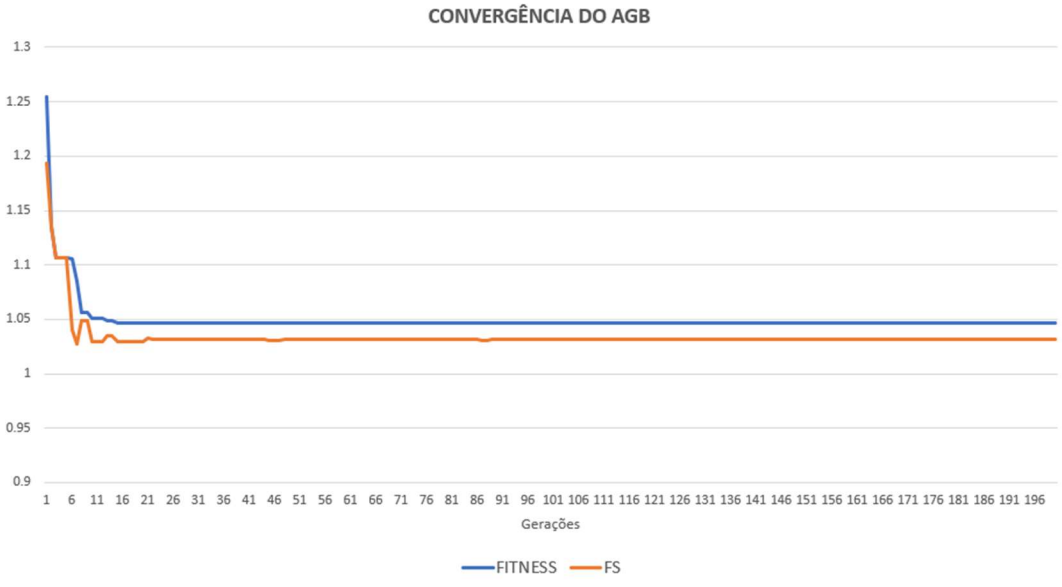
Tabela 3 – Atributos do AGB para otimização do FS.

Atributos	**Valores**
N° de variáveis de projeto	3 (xc; yc; R_c)
Nº de genes por variável de projeto	20
Intervalo das variáveis de projeto	[8; 14], [0;10], [1; 15]
Tamanho da população	1000
Gerações	200
Taxa de mutação (%)	1
Taxa de elitismo (%)	1

Fonte: o próprio autor (2026).

A Figura 17 apresenta a convergência do AGB ao longo das gerações, usando os melhores valores de aptidão (*fitness*) de cada geração.

Figura 17 – Convergência no AGB.

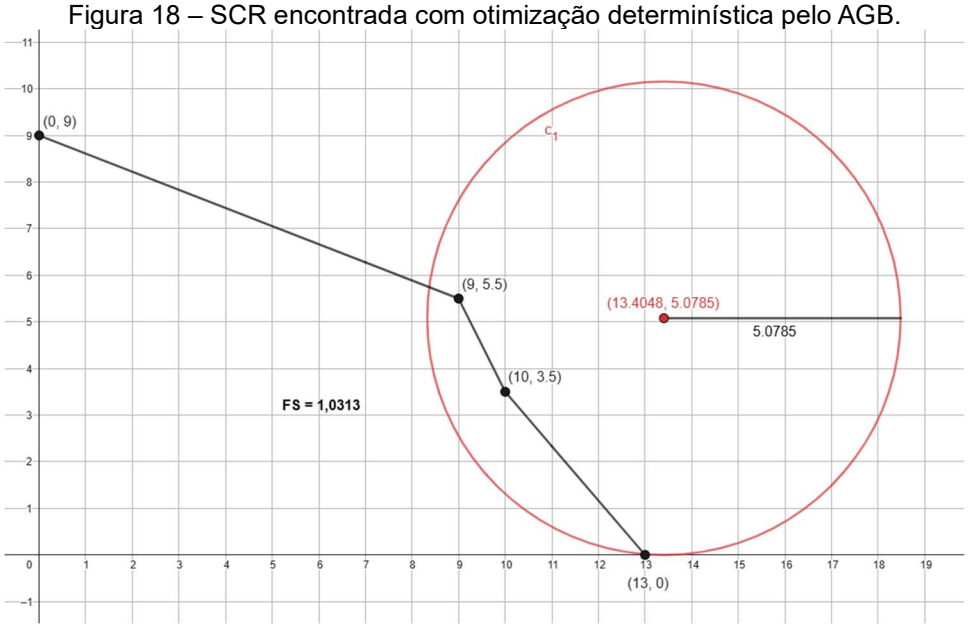


Fonte: o próprio autor (2026).

É possível observar que o FS geralmente é menor que o valor de aptidão, uma vez que a aptidão é medida considerando as restrições. Logo, mesmo que um cromossomo represente um FS muito baixo, ele não será tão apto se desrespeitar muitas restrições.

Observa-se que, nas gerações iniciais, ocorre uma redução significativa do fator de segurança, indicando uma rápida exploração do espaço de busca pelo AGB. À medida que o processo evolutivo avança, a taxa de melhoria torna-se progressivamente menor, sugerindo a convergência da população para uma região próxima da solução ótima. A superfície crítica de ruptura identificada pelo AGB possui $x_c = 13{,}4048$, $y_c = 5{,}0785$ e $R_c = 5{,}0785$, e o fator de segurança associado a ela é $\boldsymbol{FS = 1.0313}$. A Figura 18 apresenta a SCR identificada pelo AGB, juntamente com o valor do FS associado.

Figura 18 – SCR encontrada com otimização determinística pelo AGB.



Fonte: o próprio autor (2026).

Os resultados obtidos indicam que o algoritmo foi capaz de identificar uma superfície de ruptura compatível com o mecanismo esperado para o problema analisado, apresentando um fator de segurança inferior ao das demais superfícies avaliadas durante o processo de busca. Além disso, o comportamento convergente observado ao longo das gerações evidencia a capacidade do algoritmo em direcionar progressivamente a população para regiões mais promissoras do espaço de soluções.

Dessa forma, considera-se que AGB apresentou desempenho satisfatório na identificação da SCR em uma abordagem determinística, permitindo sua posterior integração com a análise probabilística proposta neste trabalho, na qual o critério de otimização deixa de ser exclusivamente a minimização do FS e passa a considerar a probabilidade de falha associada a cada superfície candidata.

4.3 Otimização da probabilidade de falha

Após a validação da implementação determinística, realizou-se a aplicação da metodologia probabilística proposta neste trabalho, na qual o processo de otimização deixa de utilizar exclusivamente o FS como critério de avaliação das superfícies candidatas e passa a considerar a função de aptidão baseada na probabilidade de falha associada a cada mecanismo potencial de ruptura. Com o objetivo de avaliar o comportamento da metodologia proposta, foram considerados três cenários distintos, correspondentes ao mesmo talude das seções 4.1 e 4.2, porém, com diferentes conjuntos de propriedades geotécnicas e parâmetros estatísticos do solo. A adoção de múltiplos cenários permite investigar a influência das incertezas dos parâmetros geotécnicos sobre a SCR e sobre a probabilidade de falha estimada.

Para cada caso, o AGB foi executado utilizando os mesmos parâmetros evolutivos empregados na etapa determinística, isto é, o mesmo objeto da classe "*BinaryGeneticOperators*" com atributos apresentados na Tabela 3, alterando apenas a função de aptidão, que passou a privilegiar superfícies associadas a maiores probabilidades de falha, considerando simultaneamente as penalizações aplicadas às superfícies geometricamente inviáveis.

Para o primeiro solo foi utilizado o mesmo objeto da classe "Fellenius" com atributos mostrados na Tabela 1 e um novo objeto da classe "*ProbabilisticSlopeAnalysis*" (Apêndice D) com os atributos estatísticos (média, desvio padrão e coeficiente de variação) das variáveis aleatórias (coesão, ângulo de atrito e peso específico).

O segundo solo utilizou os mesmos atributos estatísticos do primeiro solo, apenas alterando os valores médios de cada variável aleatória.

Já o terceiro solo utilizou as mesmas médias do primeiro, modificando apenas o coeficiente de variação e consequentemente o desvio padrão, como apresentado na Tabela 4.

Tabela 4 – Atributos dos solos usados na otimização da PF.

Atributos		**Solo 1**	**Solo 2**	**Solo 3**
c' (kPa)	μ	4,15	10	4,15
	σ	0,581	0,581	0,83
	CV **(%)**	14	5,81	20
ϕ' ()	μ	35,05	28	35,05
	σ	3,505	3,505	5,2575
	CV **(%)**	10	12,518	15
γ (kN/m^3)	μ	20,71	19	20,71
	σ	0,4142	0,4142	1,0355
	CV **(%)**	2	2,18	5

Fonte: o próprio autor (2026).

Adicionalmente, a Tabela 5 apresenta os principais resultados obtidos para cada solo, incluindo a SCR, o ponto de projeto, a porcentagem de influência que cada variável de projeto apresentou, o índice de confiabilidade e a probabilidade de falha estimada.

Tabela 5 – Resultados da otimização da PF em cada solo.

	Solo 1			**Solo 2**		
SCR (m)	x_c	y_c	R_c	x_c	y_c	R_c
	13,4934	5,0996	5,0996	8,0147	1,0341	3,8976
Ponto de projeto	**c' (kPa)**	**ϕ' (°)**	**γ (kN/m^3)**	**c' (kPa)**	**ϕ' (°)**	**γ (kN/m^3)**
	4,0972	34,2501	20,7153	≈ 10	≈ 28	18,9999
Influência (%)	13,6339	86,0951	0,2711	3,6845	95,7968	0,5187
β	0,2459			$4{,}69 \cdot 10^{-6}$		
Pf	0,4029			0,50		

	Solo 3		
SCR (m)	x_c	y_c	R_c

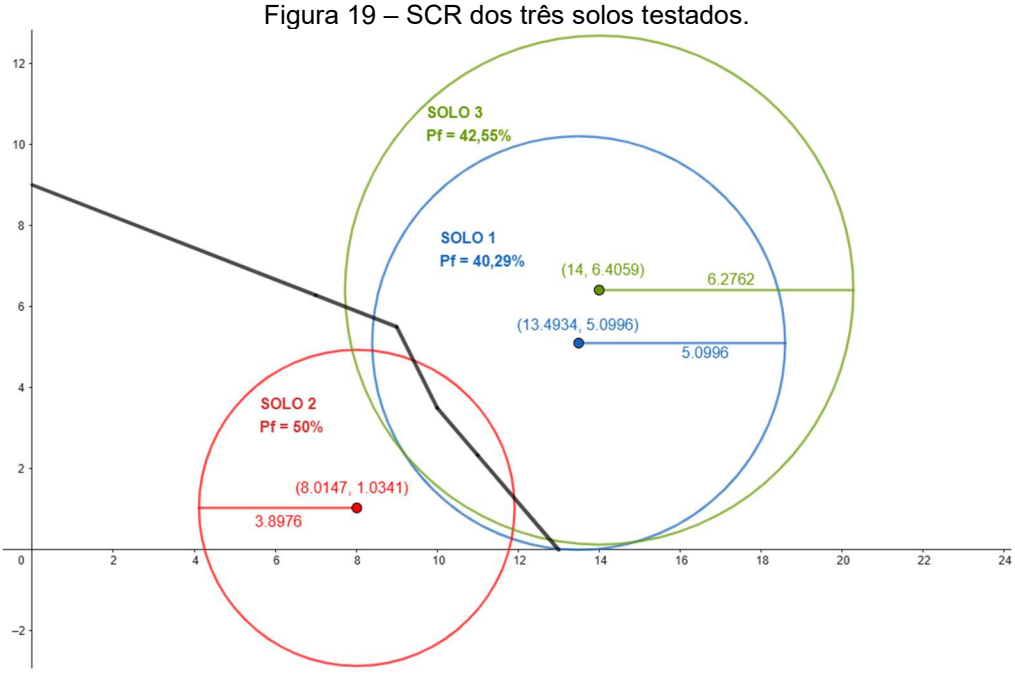
	14	6,4059	6,2762
Ponto de projeto	**c' (kPa)**	**ϕ' (°)**	**γ (kN/m^3)**
	4,0893	34,1444	20,7286
Influência (%)	15,1184	83,9658	0,9158
β	0,1880		
Pf	0,4255		

Fonte: o próprio autor (2026).

O solo 1, com baixa coesão e elevado ângulo de atrito, tem características de um solo arenoso, apresentando 40,29 % de probabilidade de romper. As incertezas presentes nesse solo, inerentes ao coeficiente de variação, serão usadas de base para comparação com os outros dois solos. O solo 2, com coesão mais alta e ângulo de atrito menor, possuindo características de um solo argiloso, apresentou probabilidade de falha igual a 50. Nota-se que ao modificar o tipo de solo mantendo o mesmo nível de incertezas, como no solo 2, a probabilidade de falha é influenciada pelas características resistentes do solo, mais especificamente pela diminuição do ângulo de atrito, que influencia cerca de 95 % da probabilidade de falha no solo 2. Quanto ao solo 3, que é o mesmo tipo de solo arenoso que solo 1, porém com um nível de incerteza maior, apresentou 42,55 % de chance de romper, comprovando o comportamento que já era esperado: se as incertezas quanto ao tipo de solo aumentam, sua probabilidade de falha refletirá a baixa precisão dos dados, forçando um estado mais conservador de segurança.

A Figura 19 apresenta as superfícies críticas identificadas para cada solo analisado.

Figura 19 – SCR dos três solos testados.



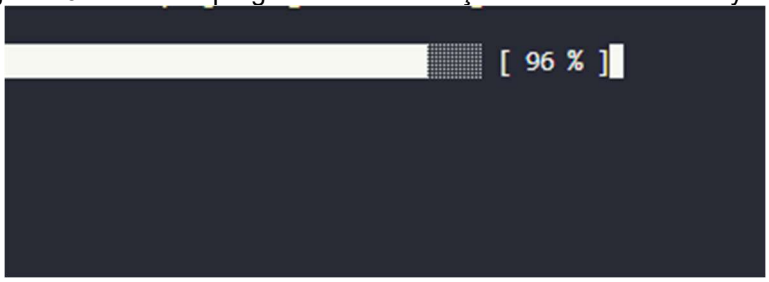
Fonte: o próprio autor (2026).

A SCR encontrada para o solo 2 apresenta uma reentrância acentuada no talude, considerando que os parâmetros geotécnicos foram modificados de maneira arbitrária, possivelmente a combinação deles não representam fielmente um tipo de solo argiloso existente, gerando uma superfície de ruptura incomum. O que se pode inferir, é que para esses parâmetros a ruptura se daria com uma grande quantidade de solo de rompendo.

4.4 Saída de dados

A ferramenta computacional implementada conta com uma barra de progresso simples para acompanhar a progressão da otimização quando o algoritmo está sendo executado, mostrando no terminal a porcentagens de gerações que já foram avaliadas, como mostrado na Figura 20.

Figura 20 - Barra de progresso da otimização vista no terminal Python.

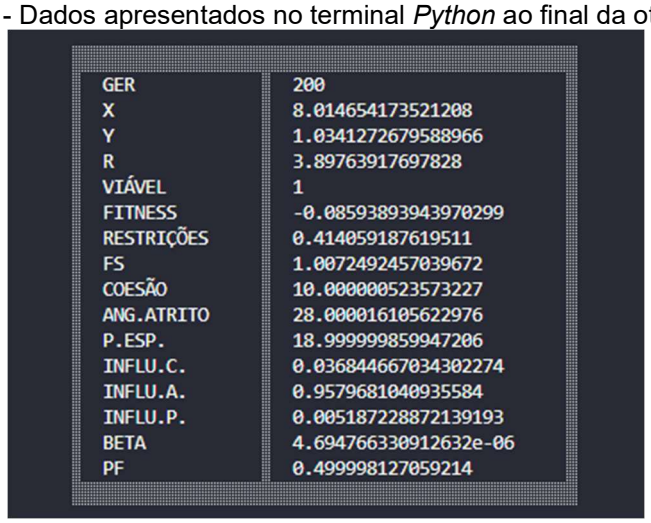


Fonte: o próprio autor (2026).

Após a execução do processo de otimização, o programa desenvolvido fornece os principais resultados obtidos durante a análise. As informações são apresentadas diretamente no terminal do *Python*, permitindo ao usuário visualizar de forma imediata os parâmetros associados à SCR identificada pelo algoritmo.

Entre os resultados exibidos encontram-se as coordenadas do centro da superfície circular, o respectivo raio, o FS calculado pelo método de Fellenius, o índice de confiabilidade e a probabilidade de falha associada à solução encontrada. Além disso, são apresentadas informações referentes ao desempenho do algoritmo genético, como o número de gerações executadas e o valor da função objetivo correspondente ao melhor indivíduo. A Figura 21 apresenta um exemplo da saída de dados exibida no terminal ao final do processo de otimização probabilística.

Figura 21 - Dados apresentados no terminal *Python* ao final da otimização.

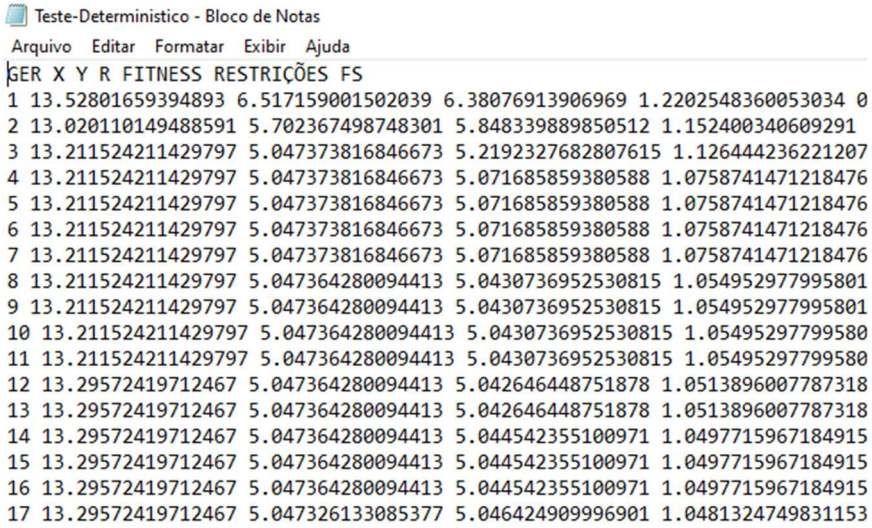


Fonte: o próprio autor (2026).

Com o objetivo de permitir análises posteriores e facilitar a interpretação do processo evolutivo, o programa também realiza o armazenamento automático dos resultados em um arquivo no formato .txt em um diretório predefinido no código. Nesse arquivo são registrados os melhores indivíduos obtidos em cada geração, juntamente com seus respectivos parâmetros geométricos e valores de aptidão.

Esse procedimento possibilita acompanhar a evolução do AGB ao longo das gerações, permitindo verificar o comportamento da convergência e identificar possíveis oscilações ou estabilizações do processo de busca. Além disso, o armazenamento dos dados em arquivo externo possibilita sua posterior utilização na construção de gráficos, tabelas e análises complementares. A Figura 22 apresenta um exemplo do arquivo gerado automaticamente pelo programa.

Figura 22 - Arquivo de texto gerado pelo algoritmo.



Fonte: o próprio autor (2026).

A disponibilização simultânea dos resultados no terminal e em arquivo externo torna a ferramenta desenvolvida mais flexível do ponto de vista prático, permitindo tanto a análise imediata dos resultados quanto a realização de estudos posteriores sem a necessidade de novas execuções do algoritmo.

4.5 Código fonte

Todo o código fonte implementado em Python foi disponibilizado através dos apêndices neste trabalho, a fim de que a reprodução e testes possam ser realizados por qualquer pessoa. O projeto na íntegra também está disponível em repositório online na plataforma *GitHub* através do link https://github.com/0Serra/slope-stability-genetic-algorithm.

5 CONCLUSÃO

O presente trabalho teve como objetivo desenvolver uma ferramenta computacional para identificação automática de superfícies críticas de ruptura em taludes, integrando o método de Fellenius, o método probabilístico FOSM (*First Order Second Moment*) e um algoritmo genético binário. A metodologia proposta buscou incorporar as incertezas inerentes aos parâmetros geotécnicos ao processo de análise da estabilidade, permitindo que a avaliação da segurança do talude fosse realizada sob uma perspectiva probabilística.

Inicialmente, procedeu-se à implementação e validação do método de Fellenius, cujos resultados apresentaram boa concordância com os cálculos

realizados de forma parcialmente manual, evidenciando a consistência da formulação empregada para o cálculo do fator de segurança. Em seguida, verificou-se o adequado funcionamento do algoritmo genético na identificação de superfícies críticas de ruptura em uma abordagem determinística, demonstrando sua capacidade de explorar o espaço de busca e convergir para soluções geotecnicamente coerentes.

Posteriormente, incorporou-se à metodologia uma análise probabilística baseada no método FOSM, na qual os parâmetros de resistência do solo foram tratados como variáveis aleatórias. Essa abordagem possibilitou a estimativa da probabilidade de falha associada a cada superfície candidata, permitindo que o processo de otimização passasse a buscar superfícies com maior propensão à ocorrência da ruptura, em substituição ao critério tradicional baseado exclusivamente na minimização do fator de segurança.

Os resultados obtidos evidenciaram a influência das incertezas geotécnicas na avaliação da estabilidade dos taludes, demonstrando que a consideração da variabilidade dos parâmetros do solo pode alterar a interpretação do nível de segurança do talude. Além disso, observou-se que a superfície crítica determinada a partir de uma abordagem probabilística não necessariamente coincide com aquela obtida por meio de uma análise puramente determinística. Esse comportamento evidencia que o fator de segurança, embora constitua um importante indicador de estabilidade, pode não representar integralmente o desempenho do sistema quando as incertezas inerentes aos parâmetros geotécnicos são consideradas.

Ressalta-se que a comparação realizada neste trabalho não teve como finalidade determinar qual metodologia apresenta melhor desempenho, mas sim investigar de que forma a consideração explícita das incertezas geotécnicas pode influenciar a identificação das superfícies críticas de ruptura e a avaliação da segurança de taludes. Nesse contexto, a abordagem probabilística não deve ser interpretada como uma substituição da análise determinística tradicional, mas sim como uma ferramenta complementar, capaz de fornecer informações adicionais relevantes para o processo de tomada de decisão em problemas de Engenharia.

De maneira geral, a metodologia desenvolvida mostrou-se capaz de integrar satisfatoriamente técnicas de otimização e confiabilidade estrutural, constituindo uma ferramenta computacional promissora para aplicações em análises de estabilidade de taludes.

Como sugestões para trabalhos futuros, recomenda-se a aplicação da metodologia proposta em casos reais de estabilidade de taludes, utilizando parâmetros geotécnicos obtidos a partir de investigações de campo e ensaios laboratoriais. Adicionalmente, sugere-se a consideração de outras técnicas de análise probabilística, bem como a utilização de métodos de equilíbrio limite mais sofisticados. Também podem ser exploradas futuras implementações envolvendo a consideração da inclusão do nível d'água e de carregamentos externos, além do estudo da influência da correlação estatística entre os parâmetros geotécnicos sobre a probabilidade de falha estimada.

Por fim, espera-se que os resultados obtidos contribuam para o desenvolvimento de metodologias computacionais aplicadas à Engenharia Civil e Engenharia Geotécnica, incentivando a utilização de abordagens probabilísticas na análise da estabilidade de taludes e evidenciando a importância da consideração das incertezas inerentes aos materiais geotécnicos na avaliação da segurança de obras de terra.

REFERÊNCIAS

ARAUJO, L.; CERVIGÓN, C. **Algoritmos evolutivos:** un enfoque práctico. 1. ed. Cidade do México: Alfaomega Grupo Editor, 2009. ISBN 978-607-7686-29-3.

BAZÁN, F. A. V. **Confiabilidade estrutural:** notas de aula. São Carlos: Programa de Pós-Graduação em Estruturas e Construção Civil / UFSCar, 2018.

BECK, A. T. **Curso de confiabilidade estrutural**: material didático. São Carlos: Universidade de São Paulo, Escola de Engenharia de São Carlos, Departamento de Engenharia de Estruturas, 2012.

CAMPELLO, I. C. **Abordagem probabilística aplicada ao estudo da variabilidade geotécnica dos solos**. 2020. Dissertação (Mestrado em Geotecnia e Transportes) – Escola de Engenharia, Universidade Federal de Minas Gerais, Belo Horizonte, 2020.

CZAP, M. M. F.; MARQUES FILHO, J. Análise teórica das abordagens determinísticas e probabilísticas para localização de superfícies de ruptura em taludes. **Contribuciones a las Ciencias Sociales**, São José dos Pinhais, v. 17, n. 13, p. 1-21, 2024. DOI: 10.55905/revconv.17n.13-123.

DAS, B. M.; SOBHAN, K. **Fundamentos de engenharia geotécnica**. Tradução da 8. ed. norte-americana. São Paulo: Cengage Learning, 2014. ISBN 978-85-221-1824-3.

DEB, K. **Multi-Objective Optimization using Evolutionary Algorithms**. 1. ed. Chichester, England: John Wiley & Sons, 2001. ISBN 0-471-87339-X.

DOAN, N. S. Reliability analysis and uncertainty quantification of clay and sand slopes stability evaluated by Fellenius and Bishop's simplified methods. **International Journal of Geo-Engineering**, v. 14, art. 22, 2023. DOI: 10.1186/s40703-023-00200-2.

DUNCAN, J. M.; WRIGHT, S. G.; BRANDON, T. L. **Soil strength and slope stability**. 2. ed. Hoboken: John Wiley & Sons, 2014. ISBN 978-1-118-65165-0.

FELLENIUS, W. **Calculation of Stability of earth dams**. 2nd Congress on Large Dams, Washington, v. 4, p. 445-463, 1936.

FERREIRA, E. G. **Análise de confiabilidade estrutural via Método SORM DG**. 2025. Tese de Doutorado (Engenharia Civil) – Universidade Federal de Ouro Preto, Ouro Preto, 2015.

GERSCOVICH, D. M. S. **Estabilidade de taludes**. 2. ed. São Paulo: Oficina de Textos, 2016. ISBN 978-85-7975-241-4.

GOLDBERG, D. E. **Genetic algorithms in search, optimization, and machine learning**. 1. ed. Boston, MA: Addison-Wesley, 1989. ISBN 0-201-15767-5.

HOLLAND, J. H. **Adaptation in natural and artificial systems:** an introductory analysis with applications to biology, control, and artificial intelligence. Cambridge: MIT Press, 1992. ISBN: 0-262-58111-6.

LINDEN, R. **Algoritmos genéticos:** Uma importante ferramenta da Inteligência Computacional. Rio de Janeiro: Brasport, 2006. ISBN 85-7452-265-1.

MASSAD, F. **Obras de terra:** Curso básico de geotécnica. São Paulo: Oficina de Textos, 2003.

SAGRILO, L. V. S. **Análise de confiabilidade estrutural utilizando os métodos FORM e SORM**. 1994. 142 p. Tese (Doutorado em Engenharia Civil) – Universidade Federal do Rio de Janeiro, COPPE/UFRJ, Rio de Janeiro, 1994.

SANTOS, M. F. **Análise de confiabilidade estrutural de fundações tipo sapata:** aplicação em uma sapata corrida. 2021. Trabalho de Conclusão de Curso (Engenharia Civil) – Universidade Federal do Maranhão, São Luís, 2021.

SILVA, E. E. **Otimização de estruturas de concreto armado utilizando algoritmos genéticos**. 2001. Dissertação (Mestrado em Engenharia de Estruturas) – Escola Politécnica, Universidade de São Paulo, São Paulo, 2001.

SILVA, J. P. M. **Os Métodos de Equilíbrio Limite e dos Elementos Finitos na Análise de Estabilidade de Taludes**. 2011. Dissertação de Mestrado (Engenharia Civil, especialização em Geotecnia) - Faculdade de Engenharia da Universidade do Porto, Porto-Portugal, 2011.

SILVA, M. S. **Implementação e aplicações de um algoritmo genético para problemas de otimização**. 2024. Trabalho de Conclusão de Curso (Bacharelado Interdisciplinar em Ciência e Tecnologia) – Universidade Federal do Maranhão, São Luís, 2024.

TENÓRIO, E. A. G.; SILVA, J. H. G.; TORRES, P. R. B.; MENEZES, W. R.; SILVANI, C. Comparação entre a análise de estabilidade de taludes por método determinístico e probabilístico. **Congresso nacional de pesquisa e ensino em ciências (CONAPESC)**, 3., 2018. Campina Grande: Realize Editora, 2018.

APÊNDICE A – ARQUIVO .PY (*MAIN*)

```
from classes import BinaryGeneticOperators, Fellenius, ProbabilisticSlopeAnalysis
from utils import (
    fitness_fellenius,
    fitness_fellenius_fosm,
    progress_bar,
    export_to_txt,
    print_best_result,
)

slope_points = [[0, 9], [9, 5.5], [10, 3.5], [13, 0], [16, 0]]
number_slices = 8
soil_specific_weight = 20.71
soil_cohesion = 4.15
soil_friction_angle = 35.05
intersection_intervals = [[7, 9], [11, 13]]

variables_number = 3
variables_length = 20
variables_range = [((8, 14), 1), ((0, 10), 1), ((1, 15), 1)]
population_length = 1000
generations = 200
mutation_rate = 1
elitism_rate = 1

m = [4.15, 35.05, 20.71]
sd = [0.83, 5.2575, 1.0355]
t = 1e-4

soil_parameters = Fellenius(
    slope_points,
    number_slices,
    soil_specific_weight,
    soil_cohesion,
    soil_friction_angle,
    intersection_intervals,
)

ga_parameters = BinaryGeneticOperators(
    variables_number,
    variables_length,
    variables_range,
    population_length,
    generations,
    mutation_rate,
    elitism_rate,
)

fosm_parameters = ProbabilisticSlopeAnalysis(m, sd, t)

function_fitness = fitness_fellenius_fosm
fitness_args = [ga_parameters, soil_parameters, fosm_parameters]

initial_population = ga_parameters.create_population()
current_generation = 1
new_population = initial_population
data = []
```

```
progress_bar(current_generation, ga_parameters.generations)

while current_generation <= ga_parameters.generations:

    evaluated_population, elite, selected = ga_parameters.evaluation_and_selection(
        new_population,
        function_fitness,
        *fitness_args,
    )

    new_population = ga_parameters.generate_new_population(selected, 1)

    new_population += elite

    best_norm = ga_parameters.normalize_chromosome(evaluated_population[0][0])

    data.append(
        [current_generation]
        + best_norm
        + [evaluated_population[0][1]]
        + [evaluated_population[0][2]]
        + [evaluated_population[0][3]]
        + evaluated_population[0][4]
    )

    current_generation += 1

    progress_bar(current_generation, ga_parameters.generations)


export_to_txt(
    "Resultados",
    r"C:\Users\Lenovo\Desktop\ARTIGO",
    " ",
    function_fitness,
    data,
)

print_best_result(function_fitness, data)
```

APÊNDICE B – ARQUIVO .PY (*BINARY_GENETIC_OPERATORS*)

```
import random
import math as mt


class BinaryGeneticOperators:

    def __init__(
        self,
        variables_number,
        variables_length,
        variables_range,
        population_length,
        generations,
        mutation_rate,
        elitism_rate,
    ):
        self.variables_number = variables_number
        self.variables_length = variables_length
        self.genes_number = variables_number * variables_length
        self.variables_range = variables_range
        self.population_length = population_length
        self.generations = generations
        self.mutation_rate = mutation_rate / 100
        self.elite_number = mt.ceil((elitism_rate / 100) * population_length)

    def create_chromosome(self):

        return "".join(str(random.choice([0, 1])) for _ in range(self.genes_number))

    def normalize_chromosome(self, chromosome):
        normalized_chromosome = []
        normalized_variables = 0
        separate_chromosome = [
            chromosome[i : i + self.variables_length]
            for i in range(0, len(chromosome), self.variables_length)
        ]

        for (initial_range, final_range), variable in self.variables_range:

            for _ in range(variable):
                normalized_variables += 1
                int_value = int(separate_chromosome[normalized_variables - 1], 2)
                normalized_value = (
                    initial_range
                    + ((final_range - initial_range) / (2**self.variables_length -
1))
                    * int_value
                )
                normalized_chromosome.append(normalized_value)

        return normalized_chromosome

    def create_population(self):

        return [self.create_chromosome() for _ in range(self.population_length)]
```

```
    def evaluate_chromosome(self, chromosome, fitness_function, *args):
        viable, fitness, constraints, utilities = fitness_function(chromosome,
*args)

        return viable, fitness, constraints, utilities

    def tournament(self, chromosome1, chromosome2, fitness_function, *args):
        viable1, fitness1, constraints1, utilities1 = self.evaluate_chromosome(
            chromosome1, fitness_function, *args
        )
        viable2, fitness2, constraints2, utilities2 = self.evaluate_chromosome(
            chromosome2, fitness_function, *args
        )

        if viable1 and not viable2:
            return [chromosome1, viable1, fitness1, constraints1, utilities1]
        elif viable2 and not viable1:
            return [chromosome2, viable2, fitness2, constraints2, utilities2]
        elif fitness1 <= fitness2:
            return [chromosome1, viable1, fitness1, constraints1, utilities1]
        else:
            return [chromosome2, viable2, fitness2, constraints2, utilities2]

    def evaluation_and_selection(self, population, fitness_function, *args):
        evaluated_population = []

        index = [i for i in range(len(population)) for _ in range(2)]
        random.shuffle(index)

        for i in range(0, len(index) - 2 * self.elite_number, 2):
            best = self.tournament(
                population[index[i]], population[index[i + 1]], fitness_function,
*args
            )
            evaluated_population.append(best)

        evaluated_population.sort(key=lambda item: (not item[1], item[2]))

        elite = [i[0] for i in evaluated_population[: self.elite_number]]
        selected = [
            i[0]    for    i    in    evaluated_population[:    len(population)    -
self.elite_number]
        ]

        return evaluated_population, elite, selected

    def crossover(self, parents, cuts):
        cut_points = sorted(random.sample(range(1, len(parents[0])), cuts))
        offspring = ""
        start = 0

        for i in range(len(cut_points) + 1):
            end = cut_points[i] if i < len(cut_points) else len(parents[0])
            donor = 0 if i % 2 == 0 else 1
            offspring += parents[donor][start:end]
            start = end

        return offspring
```

```
def mutation(self, chromosome):
    genes = list(chromosome)

    for i in range(len(genes)):
        if random.random() < self.mutation_rate:
            genes[i] = "1" if genes[i] == "0" else "0"

    mutated_chromosome = "".join(genes)

    return mutated_chromosome

def generate_new_population(self, selected, cuts):
    new_population = []

    if len(selected) < 2:
        return selected

    while len(new_population) < len(selected):
        parents = random.sample(selected, 2)
        offspring = self.crossover(parents, cuts)

        if self.mutation_rate > 0:
            new_chromosome = self.mutation(offspring)
        else:
            new_chromosome = offspring

        new_population.append(new_chromosome)

    return new_population
```

APÊNDICE C – ARQUIVO .PY (*FELLENIUS*)

```
import math
import sys

import numpy as np
import sympy as sp

x, y = sp.symbols("x, y")


class Fellenius:

    def __init__(
        self,
        slope_points,
        number_slices,
        soil_specific_weight,
        soil_cohesion,
        soil_friction_angle,
        intersection_intervals,
    ):

        self.slope_points = slope_points
        self.number_slices = number_slices
        self.soil_specific_weight = soil_specific_weight
        self.soil_cohesion = soil_cohesion
        self.soil_friction_angle = soil_friction_angle
        self.intersection_intervals = intersection_intervals
        self.slope_segments = self.define_slope_surface()
        self.inicializado = False

        [[x1, x2], [x3, x4]] = self.intersection_intervals

        if not (
            self.slope_segments[0]["x_min"]
            <= x1
            < x2
            < x3
            < x4
            <= self.slope_segments[-1]["x_max"]
        ):
            raise ValueError(
                f"Intervalos inválidos para interseções.\nGaranta que:
{self.slope_segments[0]["x_min"]} <= x1 < x2 < x3 < x4 <= {self.slope_segments[-
1]["x_max"]}"
            )

    def __getattr__(self, atribute):

        return f"\nO atributo '{atribute}' ainda não foi gerado.\nUse
'.set_circle_surface()' para gerá-lo\n"

    def define_slope_surface(self):
        slope_segments = []

        for i in range(len(self.slope_points) - 1):
            x0, y0 = self.slope_points[i]
            x1, y1 = self.slope_points[i + 1]
```

```
            dx = x1 - x0
            dy = y1 - y0

            segment = {}

            if abs(dx) < 1e-9:
                segment["type"] = "vertical"
                segment["x_const"] = x0
            elif abs(dy) < 1e-9:
                segment["type"] = "horizontal"
                segment["y_const"] = y0

            else:
                m = dy / dx
                b = y0 - m * x0

                segment["type"] = "regular"
                segment["m"] = m
                segment["b"] = b

            segment["x0"] = x0
            segment["y0"] = y0
            segment["x1"] = x1
            segment["y1"] = y1

            segment["x_min"] = min(x0, x1) - 1e-9
            segment["x_max"] = max(x0, x1) + 1e-9
            segment["y_min"] = min(y0, y1) - 1e-9
            segment["y_max"] = max(y0, y1) + 1e-9

            slope_segments.append(segment)

        return slope_segments

    def line_circle_intersections(self, segment, cx, cy, r):
        intersections = []
        eps = 1e-9

        if segment["type"] == "vertical":
            x = segment["x_const"]
            inside = r**2 - (x - cx) ** 2  # MODIFICADO;

            if inside < -eps:  # FORA DO CÍRCULO;
                return []

            if abs(inside) <= eps:  # TANGENTE AO CÍRCULO;
                y = cy

                if segment["y_min"] <= y <= segment["y_max"]:
                    intersections.append((x, y))

                    return intersections

            root = math.sqrt(inside)
            y1 = cy + root
            y2 = cy - root

            if segment["y_min"] <= y1 <= segment["y_max"]:
                intersections.append((x, y1))
```

```
    if segment["y_min"] <= y2 <= segment["y_max"]:
        intersections.append((x, y2))

    return intersections

if segment["type"] == "horizontal":
    y = segment["y_const"]
    inside = r**2 - (y - cy) ** 2

    if inside < -eps:
        return []

    if abs(inside) <= eps:
        x = cx

        if segment["x_min"] <= x <= segment["x_max"]:
            intersections.append((x, y))

        return intersections

    root = math.sqrt(inside)
    x1 = cx + root
    x2 = cx - root

    if segment["x_min"] <= x1 <= segment["x_max"]:
        intersections.append((x1, y))
    if segment["x_min"] <= x2 <= segment["x_max"]:
        intersections.append((x2, y))

    return intersections

m = segment["m"]
b = segment["b"]

a = 1 + m**2
bc = -2 * cx + 2 * m * (b - cy)
c = cx**2 + (b - cy) ** 2 - r**2

delta = bc * bc - 4 * a * c

if delta < -eps:
    return []

if abs(delta) <= eps:
    x = -bc / (2 * a)
    if segment["x_min"] <= x <= segment["x_max"]:
        y = m * x + b
        intersections.append((x, y))

    return intersections

root_delta = math.sqrt(delta)
x1 = (-bc + root_delta) / (2 * a)
x2 = (-bc - root_delta) / (2 * a)

if segment["x_min"] <= x1 <= segment["x_max"]:
    y1 = m * x1 + b
    intersections.append((x1, y1))
```

```
        if segment["x_min"] <= x2 <= segment["x_max"]:
            y2 = m * x2 + b
            intersections.append((x2, y2))

        return intersections

    def set_circle_surface(self, circle_center, circle_radius):

        self.circle_center = circle_center
        self.circle_radius = circle_radius

        self.intersections = self.find_intersections_slope_and_circle(
            self.slope_segments
        )

        self.inicializado = True

        if self.intersections == 0:
            return

        v1, v2, v3, v4, v5, v6, v7 = self.define_slice_properties(
            self.intersections, self.slope_segments
        )
        self.slice_width = v1
        self.slice_x = v2
        self.slice_center_x = v3
        self.slice_height = v4
        self.slice_area = v5
        self.slice_center_angle = v6
        self.slice_base_length = v7
        self.slice_total_area = sum(self.slice_area)
        self.safety_factor = self.fellenius_safety_factor(
            self.slice_area, self.slice_center_angle, self.slice_base_length
        )

    def find_intersections_slope_and_circle(self, slope_segments):
        cx = self.circle_center[0]
        cy = self.circle_center[1]
        r = self.circle_radius

        all_intersections = []
        eps = 1e-7

        for segment in slope_segments:
            intersections = self.line_circle_intersections(segment, cx, cy, r)

            for intersection in intersections:
                already = False

                for i in all_intersections:
                    if (
                        abs(intersection[0] - i[0]) < eps
                        and abs(intersection[1] - i[1]) < eps
                    ):
                        already = True
                        break

                if not already:
                    all_intersections.append(intersection)
```

```
    if len(all_intersections) != 2:
        return 0

    all_intersections.sort(key=lambda t: t[0])

    return all_intersections

def define_slice_properties(self, intersections, slope_segments):

    if intersections == 0:
        return

    x_initial, x_final = intersections[0][0], intersections[1][0]

    slice_width = (x_final - x_initial) / self.number_slices
    slice_x = np.linspace(x_initial, x_final, self.number_slices + 1)
    slice_center_x = (slice_x[:-1] + slice_x[1:]) / 2

    cx, cy = self.circle_center
    r = self.circle_radius

    inside = (r**2) - (slice_x - cx) ** 2
    inside[inside < 0] = 0
    circle_y = cy - np.sqrt(inside)
    circle_center_y = cy - np.sqrt((r * r) - (slice_center_x - cx) ** 2)

    slope_y = np.zeros_like(slice_x)

    for segment in slope_segments:
        m = segment.get("m", None)
        b = segment.get("b", None)

        mask = (slice_x >= segment["x_min"]) & (slice_x <= segment["x_max"])

        if segment["type"] == "vertical":
            x0 = segment["x_const"]
            slope_y[mask] = segment["y0"]
        elif segment["type"] == "horizontal":
            slope_y[mask] = segment["y_const"]
        else:
            slope_y[mask] = m * slice_x[mask] + b

    slice_height = slope_y - circle_y

    slice_area = (slice_height[:-1] + slice_height[1:]) * slice_width * 0.5

    m_ang = (cy - circle_center_y) / (cx - slice_center_x)

    slice_center_angle = 90 - abs(np.degrees(np.arctan(m_ang)))

    slice_base_length = slice_width / np.cos(np.deg2rad(slice_center_angle))

    return (
        slice_width,
        slice_x.tolist(),
        slice_center_x.tolist(),
        slice_height.tolist(),
        slice_area.tolist(),
```

```
        slice_center_angle.tolist(),
        slice_base_length.tolist(),
    )

def fellenius_safety_factor(
    self, slice_area, slice_center_angle, slice_base_length
):

    slice_area = np.array(slice_area)
    slice_center_angle = np.array(slice_center_angle)
    slice_base_length = np.array(slice_base_length)

    t1 = np.sum(self.soil_cohesion * slice_base_length)
    t2 = np.sum(
        slice_area
        * self.soil_specific_weight
        * np.cos(np.radians(slice_center_angle))
        * np.tan(np.radians(self.soil_friction_angle))
    )
    t3 = np.sum(
        slice_area
        * self.soil_specific_weight
        * np.sin(np.radians(slice_center_angle))
    )

    safety_factor = (t1 + t2) / t3

    return safety_factor
```

APÊNDICE D – ARQUIVO .PY (*PROBABILISTIC_ANALYSIS*)

```
import numpy as np
from scipy.stats import norm


class ProbabilisticSlopeAnalysis:
    def __init__(self, mean, standard_deviation, tolerance):
        self.m = np.array(mean)
        self.sd = np.array(standard_deviation)
        self.tolerance = tolerance

    def limit_state_function(
        self, x, slice_base_length, slice_area, slice_center_angle
    ):
        self.soil_cohesion = x[0]
        self.soil_friction_angle = x[1]
        self.soil_specific_weight = x[2]
        self.slice_base_length = np.array(slice_base_length)
        self.slice_area = np.array(slice_area)
        self.slice_center_angle = np.array(slice_center_angle)
        self.t1 = np.sum(self.soil_cohesion * self.slice_base_length)
        self.t2 = np.sum(
            self.slice_area
            * self.soil_specific_weight
            * np.cos(np.radians(self.slice_center_angle))
            * np.tan(np.radians(self.soil_friction_angle))
        )
        self.t3 = np.sum(
            self.slice_area
            * self.soil_specific_weight
            * np.sin(np.radians(self.slice_center_angle))
        )

        lef = ((self.t1 + self.t2) / self.t3) - 1

        return lef

    def gradient_x(self):
        grad_1 = np.sum(self.slice_base_length) / self.t3
        grad_2 = (
            (
                np.sum(
                    self.slice_area
                    * self.soil_specific_weight
                    * np.cos(np.radians(self.slice_center_angle))
                )
            )
            * (1 + (np.tan(np.radians(self.soil_friction_angle))) ** 2)
            * (np.pi / 180)
            / self.t3
        )
        grad_3 = (
            (
                self.t3
                * np.sum(
                    self.slice_area
                    * np.cos(np.radians(self.slice_center_angle))
```

```
                    * np.tan(np.radians(self.soil_friction_angle))
                )
            )
            - (
                (self.t1 + self.t2)
                *                     np.sum(self.slice_area                     *
np.sin(np.radians(self.slice_center_angle)))
            )
        ) / (self.t3**2)
        return [grad_1, grad_2, grad_3]

    def gradient_y(self):
        jacobian_xy = np.diag(self.sd)
        return jacobian_xy.T @ self.gradient_x()

    def new_y(self, y, x):
        return (
            (
                self.gradient_y() @ y
                - self.limit_state_function(
                    x,          self.slice_base_length,          self.slice_area,
self.slice_center_angle
                )
            )
            / (np.linalg.norm(self.gradient_y()) ** 2)
        ) * self.gradient_y()

    def beta(self, x):
        return np.linalg.norm(x)

    def find_probability_failure(self):
        x = [3, 33, 22]
        y = (x - self.m) / self.sd

        beta = self.beta(y)
        stop_condition = self.tolerance + 1

        while stop_condition > self.tolerance:
            g_x = self.limit_state_function(
                x, self.slice_base_length, self.slice_area, self.slice_center_angle
            )
            grad_y = self.gradient_y()
            direction_cosines = grad_y / np.linalg.norm(grad_y)
            sensitivity = direction_cosines**2

            previous_y = y
            y = self.new_y(y, x)
            beta = self.beta(y)
            x = y * self.sd + self.m
            stop_condition = abs(
                (np.linalg.norm(y)      -      np.linalg.norm(previous_y))      /
np.linalg.norm(y)
            )

        pf = norm.cdf(-beta)

        return x, sensitivity, beta, pf
```

APÊNDICE E – ARQUIVO .PY (*FITNESS_FUNCTIONS*)

```
import math
import time as tm


def fitness_fellenius(chromosome, ga_parameters, soil_parameters):
    normalized_individual = ga_parameters.normalize_chromosome(chromosome)

    soil_parameters.set_circle_surface(
        normalized_individual[:2], normalized_individual[2]
    )

    constraints = 0

    if soil_parameters.intersections == 0:
        viability = False

        return viability, 1000, constraints, [0]
    else:
        viability = True

    intersection_intervals = soil_parameters.intersection_intervals

    constraints = max(
        intersection_intervals[0][0] - soil_parameters.intersections[0][0],
        0,
        soil_parameters.intersections[0][0] - intersection_intervals[0][1],
    ) + max(
        intersection_intervals[1][0] - soil_parameters.intersections[-1][0],
        0,
        soil_parameters.intersections[-1][0] - intersection_intervals[1][1],
    )

    fs = soil_parameters.safety_factor

    fitness = fs + constraints
    utilities = [fs]

    return viability, fitness, constraints, utilities


def     fitness_fellenius_fosm(chromosome,     ga_parameters,     soil_parameters,
fosm_parameters):

    normalized_individual = ga_parameters.normalize_chromosome(chromosome)

    soil_parameters.set_circle_surface(
        normalized_individual[:2], normalized_individual[2]
    )

    constraints = 0

    if soil_parameters.intersections == 0:
        viability = False

        return viability, 1000, constraints, [0, 0]
    else:
```

```
            viability = True

        intersection_intervals = soil_parameters.intersection_intervals

        constraints = max(
            intersection_intervals[0][0] - soil_parameters.intersections[0][0],
            0,
            soil_parameters.intersections[0][0] - intersection_intervals[0][1],
        ) + max(
            intersection_intervals[1][0] - soil_parameters.intersections[-1][0],
            0,
            soil_parameters.intersections[-1][0] - intersection_intervals[1][1],
        )

        bases = soil_parameters.slice_base_length
        areas = soil_parameters.slice_area
        angles = soil_parameters.slice_center_angle

        x = fosm_parameters.m
        fosm_parameters.limit_state_function(x, bases, areas, angles)

        fs = soil_parameters.safety_factor
        x_proj, sensitivity, beta, pf = fosm_parameters.find_probability_failure()
        fitness = -pf + constraints

        cohesion, friction_angle, specific_weight = x_proj
        sens_cohesion, sens_friction, sens_weight = sensitivity

        utilities = [
            fs,
            cohesion,
            friction_angle,
            specific_weight,
            sens_cohesion,
            sens_friction,
            sens_weight,
            beta,
            pf,
        ]

        return viability, fitness, constraints, utilities


    fitness_fellenius_fosm.header = [
        "GER",
        "X",
        "Y",
        "R",
        "VIÁVEL",
        "FITNESS",
        "RESTRIÇÕES",
        "FS",
        "COESÃO",
        "ANG.ATRITO",
        "P.ESP.",
        "INFLU.C.",
        "INFLU.A.",
        "INFLU.P.",
        "BETA",
```

```
    "PF",
]

fitness_fellenius.header = [
    "GER",
    "X",
    "Y",
    "R",
    "VIÁVEL",
    "FITNESS",
    "RESTRIÇÕES",
    "FS",
]


def fitness_rastrigin(chromosome, ga_parameters):
    x = ga_parameters.normalize_chromosome(chromosome)
    a = 10
    n = len(x)

    fitness = (a * n) + sum(
        [(x[i] ** 2 - a * math.cos(2 * math.pi * x[i])) for i in range(n)]
    )
    viable = True

    return viable, fitness
```

APÊNDICE F – ARQUIVO .PY (*AUXILIARY_FUNCTIONS*)

```
def progress_bar(current_value, total_value):
    percent = (current_value / total_value) * 100
    percent = min(max(percent, 0), 100)
    progress = int(percent)
    bar = "█" * progress + "░" * (100 - progress)
    print(f"\r{bar} [ {int(percent)} % ]", end="")


def export_to_txt(name, path, delimiter, function_fitness, data):
    with open(f"{path}\\{name}.txt", "w", encoding="utf-8") as file:
        file.write(delimiter.join(map(str, function_fitness.header)) + "\n")

        for row in data:
            formatted_row = delimiter.join(map(str, row))
            file.write(formatted_row + "\n")


def print_best_result(function_fitness, data):
    max_column = max(len(column) for column in function_fitness.header) + 5
    max_value = max(len(str(value)) for value in data[-1]) + 5

    print("\n")
    print("     " + "░" * (max_column + max_value + 7))

    for column, value in zip(function_fitness.header, data[-1]):
        print(f"     ░  {column:<{max_column}}  {value:<{max_value}}░")

    print("     " + "░" * (max_column + max_value + 7))
    print("")
```