Ludger O. Suárez-Burgoa & Exneyder A. Montoya Araque

CIRCULAR SLOPE STABILTY PYPROGRAM (PYCSS)

PROGRAMA DE CÓDIGO ABIERTO EN PYTHON PARA EL ANÁLISIS DE ESTABILIDAD DE TALUDES EN 2D, MÉTODOS DE FELLENIUS Y DE BISHOP.

JULIO , 2016

Manual de Usuario

Universidad Nacional de Colombia – Medellín

A los que sembraron la buena semilla que germinó y produjo este fruto lleno de esfuerzo y dedicación.

E.A.M.A.

A los desarrolladores de software libre en Ciencias de la Tierra.

L.O.S.B.

Prefacio

El programa pyCSS® fue diseñado para el cálculo del factor de seguridad al deslizamiento por falla circular en taludes en dos dimensiones (2D), bajo condiciones de equilibrio límite mediante los métodos de Fellenius (*i.e.* Método Ordinario de las Dovelas) y de Bishop Simplificado, ambos de uso extensivo en geotécnia.

En muchas ocasiones no se tiene la capacidad económica para comprometerse a una licencia comercial de un *software* especializado y no queda más alternativa que realizar los extensos y tediosos cálculos de forma manual, o recurrir a mecanismos ilícitos como la obtención de *software* pirata o aplicaciones externas que violentan el acceso al *software* comercial. Esto último representa una violación a los derechos de autor, además de tener potencial de múltiples riesgos informáticos.

Las razones por presentar pyCSS® como código abierto tiene el fin de posibilitar al usuario acceder a una herramienta informática no comercial, que le facilitará realizar cálculos que serían tediosos de forma manual y sin que tenga que hacer uso de *software* pirata. Asimismo, debido a que el código fuente está disponible, el presente programa incentivará a los usuarios a estudiar, modificar y mejorarlo.

Esperamos que el presente programa sea de gran utilidad para el usuario y que su impacto sea positivo y duradero. Medellín, julio de 2016

Ludger O. Suárez-Burgoa
Profesor Asistente
Departamento de Ingeniería Civil
Universidad Nacional de Colombia

Exneyder A. Montoya Araque
Estudiante Ingeniería Geológica
Departamento de Geociencias y Medio Ambiente
Universidad Nacional de Colombia

Índice general

Capítulo 1
Descripción

El presente programa **pyCSS**® tiene un acrónimo tomado del idioma Inglés de *python-program for Circular Slope Stability*; que significa *Programa en PYTHON® para el análisis de estabilidad de taludes por falla circular*.

El objetivo principal del código abierto del programa pyCSS® es el de facilitar al usuario realizar cálculos para la obtención del factor de seguridad al deslizamiento por falla circular en *taludes estándar* en dos dimensiones (2D) considerando condiciones de equilibrio límite, usando los métodos: Fellenius (*i.e.* método ordinario de las dovelas) [8] y Bishop simplificado [1].

Denominamos talud estándar al que se muestra en la Figura 3.2, el cual cumple con las siguientes características:

- la cara del talud se encuentra buzando a la derecha;
- no hay presencia de bermas a lo largo de la cara del talud;
- su corona y pie son superficies horizontales;
- está compuesto por un solo material;
- puede haber presencia de nivel freático con las siguientes alternativas:
  - ser coincidente con la superficie del talud;
  - estar por debajo de la superficie del talud con una geometría paralela a las caras del mismo;
  - ser totalmente horizontal, cubriendo el pie del talud (*i.e.* talud parcialmente sumergido).

El código abierto está compuesto por 22 funciones, cada una asociada a un módulo (*i.e.* archivo cuyo contenido es código escrito en lenguaje PYTHON®, y que tiene extensión `.py`). Estas funciones se ejecutan dentro del lenguaje de programación PYTHON® en su versión 3.x. Sin embargo, solo un módulo final (`finalModule.py`) es el que se ejecuta, el cual recibe las variables de entrada. El módulo entonces invoca a dos funciones (según sea una única superficie evaluada, o múltiples para hallar la superficie crítica); donde cada una

de ellas a su vez va llamando a las demás funciones internamente como un sistema anidado de subfunciones, el cual se muestra en la Figura 1.1.

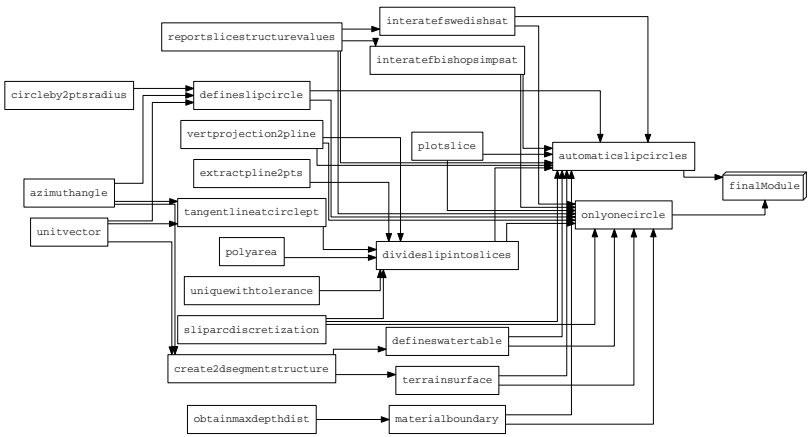


Figura 1.1 Esquema anidado de las funciones del programa pyCSS® .

Para lograr la ejecución del programa basta con editar las variables de entrada (vea la Sección 3.1) en el módulo `finalModule.py` y ejecutar el mismo, el cual hará el papel de un archivo de lotes. Otra alternativa más intuitiva para muchos usuarios será haciendo uso de la sencilla interfaz gráfica correspondiente al módulo `pyCSS.py`, donde bastará con ingresar la información solicitada y dar *clic* en el botón *Ejecutar análisis*.

Cualquiera que sea la elección a la hora de ejecutar el programa, se obtendrá como resultado dos archivos: el primer archivo es una imagen de formato `.svg` (con posibilidad de cambiar este formato entre una amplia oferta, tanto de tipo vectorial como rasterizado) la cual esquematiza el talud evaluado e incluye el resultado obtenido por uno o ambos métodos; y el segundo es un archivo de *texto plano* de formato `.txt` que resume toda la información del proyecto.

La estructura del directorio raíz del programa es la que se muestra en las Figuras 1.2 y 1.3. Ahí observará que existe tres carpetas junto a los módulos `finalModule.py` y `pyCSS.py` mencionados anteriormente.

La primera de las carpetas contiene un par de módulos que corresponden a los ejemplos usados en este manual, y que se emplean para ilustrar el uso del programa. Junto a los módulos están los archivos obtenidos con la ejecución de los mismos (extensiones `.svg` y `.txt`).

La segunda carpeta corresponde al paquete de módulos que conforman las funciones del programa.

La tercera carpeta corresponde a cinco módulos que se usaron para validar la calidad de los resultados obtenidos por el programa, además de los archivos asociados a la ejecución de los mismos y una subcarpeta que contiene un par de archivos externos `.slim` y `.xlsx` usados para la validación, ejecutándolos con SLIDE® y MICROSOFT EXCEL® (u otro programa similar que abra ese tipo de hojas de cálculo) respectivamente.

```
/home/userName/python/pyCSS/
├── finalModule.py
├── pyCSS.py
├── examples
│   ├── Example-01.svg
│   ├── example01.py
│   ├── Example-01.txt
│   ├── Example-02.svg
│   ├── example02.py
│   └── Example-02.txt
└── functions
    ├── automaticslipcircles.py
    ├── azimuthangle.py
    ├── circleby2ptsradius.py
    ├── create2dsegmentstructure.py
    ├── defineslipcircle.py
    ├── defineswatertable.py
    ├── divideslipintoslices.py
    ├── extractplinefrom2pts.py
    ├── interatefbishopsimpsat.py
    ├── interatefswedishsat.py
    ├── materialboundary.py
    ├── obtainmaxdepthdist.py
    ├── onlyonecircle.py
    ├── plotslice.py
    ├── polyarea.py
    ├── reportslicestructurevalues.py
    ├── sliparcdiscretization.py
    ├── tangentlineatcirclept.py
    ├── terrainsurface.py
    ├── uniquewithtolerance.py
    ├── unitvector.py
    └── vertprojection2pline.py
```

Figura 1.2 Estructura de la carpeta `pyCSS` (continúa).

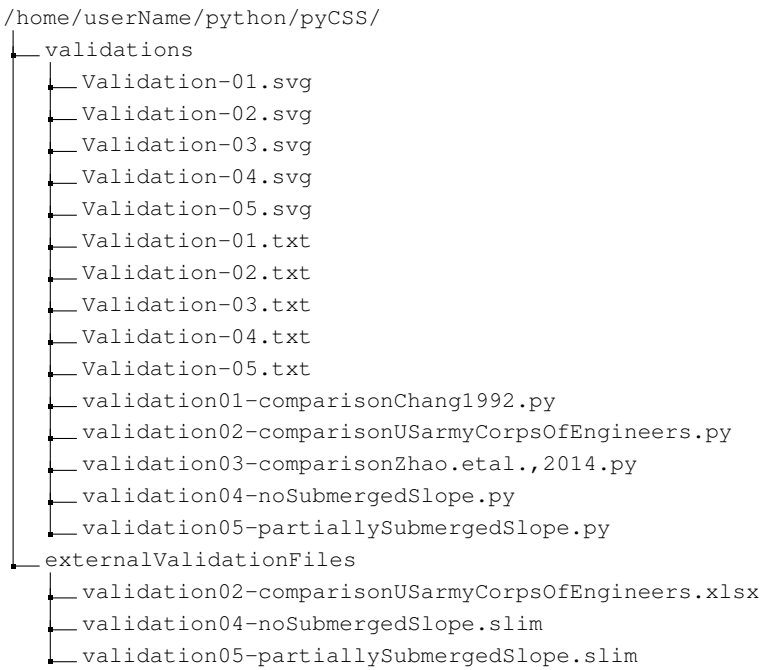


Figura 1.3 Estructura de la carpeta `pyCSS` (continuación).

Capítulo 2
Métodos de equilibrio límite

Los problemas de análisis de estabilidad de taludes para la mecánica de suelos clásica normalmente se abordan en primera instancia mediante los *métodos de equilibrio límite* (LEM de las siglas en Inglés de *Limit Equilibrium Methods*) para superficie de fallas no planas; los cuales existen en una variedad que obliga a clasificarlos.

Por tanto, los LEM se clasifican en: métodos inexactos y los exactos; donde los inexactos se dividen en el método del círculo de fricción y el método de las dovelas. El método de las dovelas se dividen en aproximados y precisos. Mientras que los métodos exactos tienen nombres específicos de los autores que plantearon; por ejemplo el método de Spencer [11], el método de Morgenstern & Price [10], o método de Bishop riguroso y el método Global de Equilibrio Límite o llamado método GLE (de las siglas en Inglés de *Global Limit Equilibrium*) [6]. La Figura 2.1 muestra la estructura relacional de esta clasificación.

En el método de las dovelas el inconveniente radica en la indeterminación del planteamiento estático de la solución. Cada dovela de las n que conforman la geometría de la superficie de falla con la superficie del terreno está sometida a ocho variables: dos fuerzas tangenciales a las caras verticales de la dovela (*i.e.* V_i y V_{i+1}), dos fuerzas normales a las caras verticales de la dovela (*i.e.* E_i y E_{i+1}), una fuerza tangencial (T) y normal (N) en la base de la dovela, dos abscisas desde la base de la dovela hacia el punto de aplicación de las fuerzas normales a las caras verticales (*i.e.* b_i y b_{i+1}).

Debido a que dos dovelas adyacentes comparten una misma cara, para el caso de n dovelas en la discretización se tiene las siguientes variables a resolver[4]:

- fuerzas tangenciales a las caras verticales de la dovela, en $(n-1)$ incógnitas;
- fuerzas normales a las caras verticales de la dovela, en $(n-1)$ incógnitas;
- abscisas hacia las fuerzas normales a las caras verticales de la dovela, en $(n-1)$ incógnitas;
- fuerzas tangenciales a las base de la dovela, en n incógnitas;
- fuerzas normales a las base de la dovela, en n incógnitas;
- localización de las fuerzas normales a la base de la dovela en n incógnitas; y

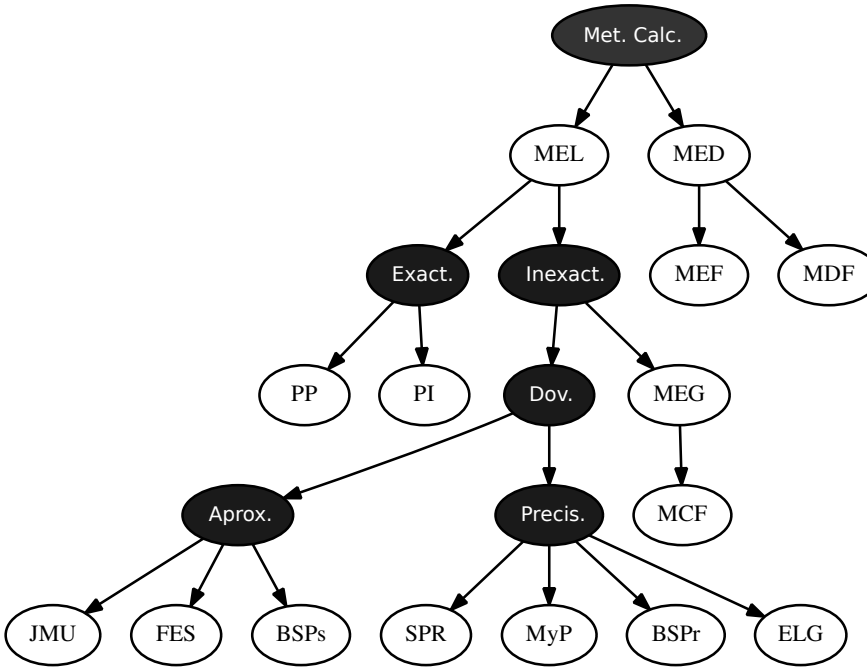


Figura 2.1 Clasificación de los métodos de cálculo bidimensionales de estabilidad de taludes. MEL: método de equilibrio límite; MED: método de esfuerzo-deformación; MEF: método de los elementos finitos; MDF: método de las diferencias finitas; PP: ruptura plana paralela a la superficie; PI: ruptura plana inclinada a la superficie; MEG: método de estabilidad global; MCF: método del círculo de fricción; JMU: método de Jambú; FES: método de Felenuius; BSPs: método de Bishop simplificado; SPR: método de Spencer; MyP: método de Morgenstern y Price; BSPr: método de Bishop riguroso; ELG: método de equilibrio global; Met. Calc.: métodos de cálculo; Exact: exactos; Inexact: inexactos; Dov: de dovelas; Aprox: aproximados; Precis: precisos.

- el factor de seguridad global, en 1 incógnita;

para un total de $6n-2$ incógnitas.

Por otro lado, por cada dovela se tiene tres ecuaciones de equilibrio estático en el plano dados por la suma de fuerzas y momentos, además de una ecuación referida al criterio de falla de Mohr-Coulomb, es decir

- ecuaciones de equilibrio de fuerzas horizontales, en n cantidades;
- ecuaciones de equilibrio de fuerzas verticales, en n cantidades;
- ecuaciones de equilibrio de momentos, en n cantidades; y
- ecuaciones de criterio de falla de Mohr-Coulomb en n cantidades;

para un total de $4n$ ecuaciones. Esto corrobora el problema de la indeterminación del planteamiento estático de la solución., ya que $6n-2-4n=2n-2$.

Para posibilitar una solución a este problema se puede incrementar el número de ecuaciones si se define las relaciones esfuerzo y deformación del material, o se puede disminuir las incógnitas conociendo alguno de sus valores por otros métodos. Por ejemplo, se puede hacer un análisis esfuerzo deformación y dada una superficie de falla conocer todos los esfuerzos en las dovelas.

También para posibilitar una solución con la intención de disminuir el número de incógnitas se adopta alguna hipótesis, de modo de conocer de forma aproximada las fuerzas tangenciales y normales a las paredes verticales de las dovelas. Esto es lo que hacen los LEM aproximados. Mientras que los LEM precisos adoptan hipótesis de tal modo que las fuerzas tangenciales y normales a las paredes verticales a las dovelas sigan una ley general que puede depender de una o pocas más variables relativas al material, pero que son conocidas.

Además del problema de la indeterminación de ecuaciones, los LEM presentan otras limitaciones [14]:

- Hay que asumir que el factor de seguridad es constante a lo largo de toda la potencial superficie de falla.
- Las propiedades de Esfuerzo-Deformación no están representadas explícitamente.
- El estado de esfuerzos iniciales dentro de la masa de suelo tampoco está representado explícitamente.
- Fuerzas anómalamente muy altas y/o negativas pueden ser calculadas bajo ciertas condiciones.
- Los métodos que requieren iteraciones, puede llegar a ocurrir que no converja en ciertos casos.

2.1. Análisis de falla circular en condiciones indrenadas

Observe que el análisis con $\phi = 0^\circ$ (que simula las condiciones indrenadas tomando $c = c_\mathrm{u} = s_\mathrm{u}$) es apropiado sólo para excavaciones temporales a corto plazo donde se asume una resistencia indrenada o de corto tiempo para controlar la estabilidad.

El método de la falla circular —para $\phi = 0^\circ$ y $c = c_\mathrm{u}$— se esquematiza en la Figura 2.2, donde el talud se divide en un número conveniente de dovelas y el factor de seguridad f_s es la razón de la suma de los momentos estabilizadores M_e respecto la suma de los momentos desestabilizadores M_d . De este modo, y refiriéndose a la Figura 2.2, se tiene que

$$f_\mathrm{s} = \frac{\text{esfuerzos estabilizantes}}{\text{esfuerzos movilizantes}} = \frac{M_\mathrm{e}}{M_\mathrm{d}},$$

donde para una dovela $M_\mathrm{e} = s_\mathrm{u} l R$ y $M_\mathrm{d} = W x$; luego la expresión queda

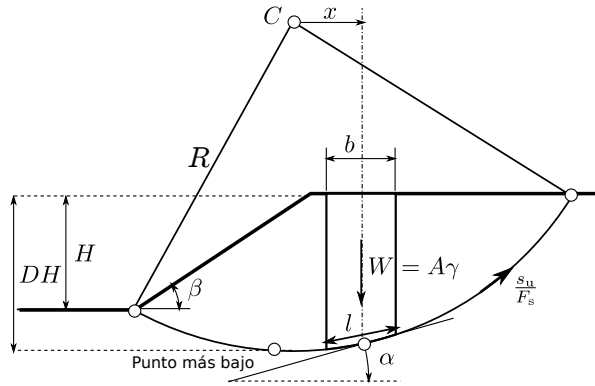


Figura 2.2 Esquema clave para el análisis de esfuerzos totales de un talud al asumir un deslizamiento circular por el método de las dovelas.

$$f_s = R\frac{\sum s_u l}{\sum W x}; \tag{2.1}$$

observe que $s_u = c_u$, $l = b\sec\alpha$, y $x = R\sin\alpha$, de este modo la expresión queda

$$f_s = \frac{\sum c_u b \sec\alpha}{\sum W \sin\alpha}, \tag{2.2}$$

y para dovelas de ancho constante, que es la forma convencional para los cálculos a mano quedaría finamente

$$f_s = \frac{\sum c_u \sec\alpha}{\sum p \sin\alpha}, \tag{2.3}$$

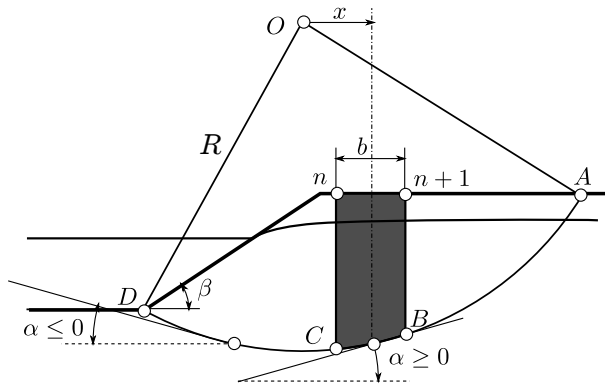
donde p es el esfuerzo vertical total en la base de la dovela.

Las soluciones son posibles para geometrías simples y bajo las siguientes condiciones:

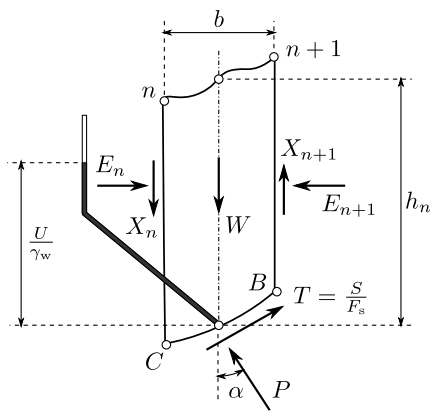
- resistencia a corte indrenada y constante;
- resistencia a corte que incrementa en forma lineal con la profundidad desde un valor cero desde la superficie del terreno [9].

2.2. Análisis de falla circular en condiciones drenadas

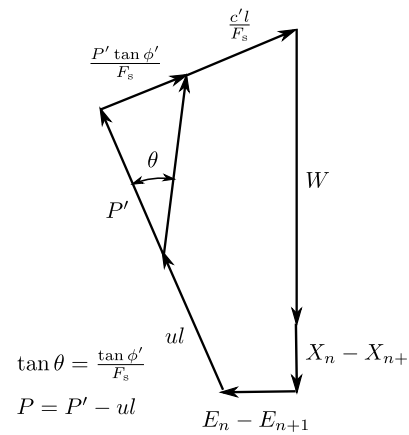
La figura 2.3 representa un talud típico, en el cual la masa de suelo por encima de la superficie circular se divide en n dovelas a conveniencia, donde cada una de ellas está sometida a una serie de fuerzas representadas allí mismo.



(a) mecanismo de falla circular



(b) fuerzas actuando en una dovela individual



(c) polígono de fuerzas

Figura 2.3 Esquema de la definición para el análisis de esfuerzos efectivos en un talud, método de las dovelas.

2.2.1. Método de Fellenius u Ordinario de las dovelas[8]

Corresponde al planteamiento que sirvió de base para muchos de los otros métodos más rigurosos que consiste en separar la masa de suelo que se encuentra entre la superficie de ruptura considerada y la superficie del terreno en una serie de *n* rebanadas verticales (llamadas técnicamente como dovelas), y evaluar el equilibrio de momento individual de cada una respecto al centro del circulo que define la superficie de deslizamiento, esto se logra asumiendo algunas hipótesis como por ejemplo que no se consideran las fuerzas

entre dovelas. Este método ha tenido diferentes denominaciones: *Método Ordinario de las dovelas* o simplemente *Método Ordinario*, sin embargo también se conoce también por el nombre del autor, siendo entonces *Método de Fellenius*[8].

El factor de seguridad se define como

$$f_s = \frac{\text{momentos estabilizantes}}{\text{momentos movilizantes}} = \frac{M_e}{M_d},$$

donde para una dovela $M_e = \tau l R$ y $M_d = W x$; luego la expresión queda

$$f_s = R\frac{\sum \tau l}{\sum W x}. \tag{2.4}$$

Por el criterio de Mohr-Coulomb se sabe que $\tau = c' + \sigma' \tan\phi'$ (en términos de esfuerzos efectivos), además $x = R\sin\alpha$; de este modo la expresión queda

$$f_s = \frac{\sum c' l + \sigma' \tan\phi' l}{\sum W \sin\alpha}. \tag{2.5}$$

Del equilibrio normal a la base de la dovela se sabe que $P = W_\perp = W\cos\alpha$; por lo tanto el esfuerzo total normal σ a la base de la dovela es

$$\sigma = \frac{W\cos\alpha}{l},$$

y como el esfuerzo efectivo σ' es igual a $\sigma - \mu$, se tiene

$$\sigma' = \sigma - \mu = \frac{W\cos\alpha}{l} - \mu, \tag{2.6}$$

donde μ es la presión de poro en la superficie de deslizamiento de la dovela. Reemplazando 2.6 en 2.5 se tiene

$$f_s = \frac{\sum c' l + (W\cos\alpha - \mu l)\tan\phi'}{\sum W \sin\alpha}. \tag{2.7}$$

La ecuación 2.7 representa una expresión para el factor de seguridad por el Métotodo de Fellenius (Ordinario de las dovelas), sin embargo la suposición de que $\sigma' = W\cos\alpha l^{-1} - \mu$, puede dar valores bajos y poco realistas, incluso negativos [7]; esto se demuestra con facilidad del siguiente modo:

- El peso de la dovela es $W = \gamma_s h b$, y como $b = l\cos\alpha$, $W = \gamma_s h l \cos\alpha$.
- Sustituyendo la anterior expresión en la ecuación 2.7 se tiene

$$f_s = \frac{\sum c' l + (\gamma_s h \cos^2\alpha - \mu) l \tan\phi'}{\sum W \sin\alpha}. \tag{2.8}$$

- El término en paréntesis , $\gamma_s h\cos^2\alpha - \mu$, de la ecuación 2.8 representa el esfuerzo efectivo σ'; por lo tanto se puede escribir de la forma

$$\frac{\sigma'}{\gamma_s h} = \cos^2\alpha - \frac{\mu}{\gamma_s h},$$

donde $\frac{\sigma'}{\gamma_s h}$ es la razón entre el esfuerzo efectivo y la presión de sobrecarga del material, y $\frac{\mu}{\gamma_s h}$ es la razón entre la presión de poro y la presión de sobrecarga del suelo.

- Ahora, supóngase que la relación de la presión de poro es un tercio de la sobrecarga, es decir $\frac{\mu}{\gamma_s h} = \frac{1}{3}$ y que la superficie de deslizamiento tiene un ángulo con la horizontal de $60°$, es decir $\alpha = 60°$.

$$\frac{\sigma'}{\gamma_s h} = \cos^2 60° - \frac{1}{3} = -0.08.$$

Lo anterior demuestra entonces que, en especial para altos valores de α, pueden darse esfuerzos efectivos negativos. Esto puede ocurrir debido a que el método ordinario de las dovelas no considera las fuerzas en las paredes de las dovelas y no hay nada para contrarrestar la presión de poros [7].

Sin embargo Turnbull y Hvorslev [13] proponen expresar el peso de la dovela en término de "peso efectivo" así:

$$W' = W - \mu b$$

y nuevamente del equilibrio normal a la base de la dovela se sabe ahora que $P' = W'_{\perp} = W'\cos\alpha$; es decir

$$P' = W'\cos\alpha,$$

y como $b = l\cos(\alpha)$,

$$P' = W\cos\alpha - \mu l\cos^2\alpha.$$

El esfuerzo efectivo entonces resulta de dividir la fuerza normal P' entre el área de la base de la dovela, dando entonces la siguiente expresión para el esfuerzo efectivo:

$$\sigma' = \frac{W\cos\alpha}{l} - \cos^2\alpha. \tag{2.9}$$

Finalmente, reemplazando 2.9 en 2.5 se tiene la ecuación 2.10, la cual es la expresión definitiva para el factor de seguridad por el método de Fellenius u Ordinario de las dovelas.

$$f_s = \frac{\sum c'l + (W\cos\alpha - \mu l\cos^2\alpha)\tan\phi'}{\sum W\sin\alpha}. \tag{2.10}$$

2.2.2. *Método de Bishop simplificado[1]*

El mecanismo de falla circular se muestra en la Figura 2.3(a) y las fuerzas que actúan en una dovela individual en la Figura 2.3(b).

El polígono de fuerzas se muestra en la Figura 2.3(c). Los puntos importantes de notar son los siguientes:

- el factor de seguridad es

$$f_s = \frac{\text{esfuerzos estabilizantes}}{\text{esfuerzos movilizantes}}$$

y por tanto la resistencia movilizada se da según

$$s = \frac{c'}{f_s} + (\sigma_n - u)\frac{\tan\phi'}{f_s};$$

- no es posible una solución analítica generalizada, y se requiere de una solución numérica [2];
- para estimar los valores de σ_n en cada punto de la superficie de falla se usa el método de las dovelas de Fellenius [8] que se publicó por primera vez en el congreso de grandes presas.

Para el caso de la solución planteada por Bishop [1], que se plantea en este texto, observe la Figura 2.3; el análisis es el siguiente.

La resistencia mobilizadora es:

$$s = \frac{c'}{f_s} + (\sigma_n - u)\frac{\tan\phi'}{f_s}. \tag{2.11}$$

El esfuerzo normal total en la base de la dovela es

$$\sigma_n = \frac{P}{l} \quad \text{de donde} \tag{2.12a}$$

$$s = \frac{c'}{f_s} + \left[\frac{P}{l} - u\right]\frac{\tan\phi'}{f_s} \tag{2.12b}$$

La fuerza de corte s que actúa en la base de la dovela resulta ser sl; y tomando momentos alrededor de O se tiene lo siguiente para un equilibrio límite:

$$\sum Wx = \sum SR = \sum slR \tag{2.13}$$

de la Ecuación 2.12b

$$\sum Wx = R\sum \frac{c'l}{f_s} + (P - ul)\tan\phi' f_s, \tag{2.14a}$$

es decir

$$f_s = \frac{R}{\sum Wx} \sum \left[c'l + (P - ul) \tan \phi' \right], \tag{2.14b}$$

para obtener P se resuelve normal a la superficie de deslizamiento

$$P = (W + X_n - X_{n-1}) \cos \alpha - (E_n - E_{n-1}) \sin \alpha. \tag{2.15}$$

Si se inserta la ecuación 2.15 en la ecuación 2.14b se tiene

$$f_s = \sum \left\{ c'l + \tan \phi' \left(W \cos \alpha - ul \right) + \tan \phi' \left[(X_n = X_{n-1}) \cos \alpha - (E_n - E_{n-1}) \sin \alpha \right] \right\} \tag{2.16}$$

Debido a que no hay fuerzas externas en el talud, se considera que la suma de las fuerzas internas deberían ser iguales a cero, es decir

$$\sum (X_n - X_{n+1}) \tag{2.17a}$$

$$\sum (E_n - E_{n+1}) \tag{2.17b}$$

pero los términos en $X_n \ldots$ y $E_n \ldots$ no desaparecen; ellos sólo desaparecen si ϕ' y α son constantes; es decir, se tiene deslizamiento en un plano.

Una propuesta de solución por la Agencia de Administración de Recursos Hídricos de los EE.UU. (USBR: United States Bureau of Reclamation) es asumir

$$\sum \tan \phi' \left[(X_n - X_{n-1}) \cos \alpha - (E_n - E_{n+1}) \sin \alpha \right] = 0$$

De este modo, si se iguala $x = R \sin \alpha$ se tiene que

$$f_s = \sum \left[c'l + \tan \phi' \left(W \cos \alpha - ul \right) \right]. \tag{2.18}$$

Esta forma de solución da resultados del lado de la conservación, particularmente para círculos profundos.

En términos simples, se resolvió el peso total W normal a la superficie de deslizamiento y luego se substrajo la fuerza debido a la presión de poros ul (ver la ecuación 2.18).

Siguiendo la fórmula de Taylor, se puede considerar el peso sumergido de la dovela igual a $W - ub$ y resolver ésto como sigue:

$$(W - ub) \cos \alpha = W \cos \alpha - u \cos \alpha \frac{1}{\sec a}$$

$$= W \cos \alpha - ul \cos^2 \alpha$$

y obtener

$$f_s = \frac{1}{\sum W \sin \alpha} \sum \left[c'l + \tan \phi' \left(W \cos \alpha - ul \cos^2 \alpha \right) \right] \tag{2.20}$$

Para evitar estos errores, se regresa a la Ecuación 2.14b de arriba, donde se coloca $P' = (P - ul)$ y se resuelve verticalmente de forma de eliminar los términos E_n como sigue:

$$W + X_n - X_{n-1} = ul\cos\alpha + P'\cos\alpha + P'\frac{\tan\phi'}{f_s}\sin\alpha + \frac{c'}{f_s}l\sin\alpha$$

entonces

$$P' = \frac{W + X_n - X_{n-1} - l\left[u\cos\alpha + \left(c' f_s^{-1}\sin\alpha\right)\right]}{\cos\alpha + f_s^{-1}\tan\phi'\sin\alpha}. \tag{2.21}$$

Substituyendo esta ecuación en la Ecuación 2.14b y colocando $l = b\sec\alpha$, $x = R\sin\alpha$ se llega a:

$$f_s = \frac{\sum\left\{c'b + \tan\phi'\left[W - ub + (X_n - X_{n+1})\right]\frac{\sec\alpha}{1+f_s^{-1}\tan\phi'\tan\alpha}\right\}}{\sum W\sin\alpha}. \tag{2.22}$$

Con una suficiente exactitud para los fines más prácticos, se puede asumir que $X_n - X_{n+1} = 0$ y obtener la solución simplificada de Bishop [1], que resulta ser igual a

$$f_s = \frac{\sum\left\{c'b + \tan\phi'\left(W - ub\right)\frac{\sec\alpha}{1+f_s^{-1}\tan\phi'\tan\alpha}\right\}}{\sum W\sin\alpha}; \tag{2.23}$$

que converge de forma rápida.

Para el caso especial de dovelas de igual ancho, la Ecuación 2.23 se escribe como sigue:

$$f_s = \frac{\sum c' + (p - u)\tan\phi' m_\alpha^{-1}}{\sum p\sin\alpha} \tag{2.24}$$

donde

$$m_\alpha = \cos\alpha\left(1 + f_s^{-1}\tan\alpha\tan\phi'\right) \tag{2.25}$$

y donde p es el esfuerzo vertical total en la base de la dovela.

Para el caso particular de taludes parcialmente sumergidos hay diferentes planteamientos para considerar las fuerzas y/o momentos externos que ejerce la columna de agua por encima de una dovela.

Las funciones que evalúan los métodos de Fellenius y de Bishop simplificado dentro de pyCSS® están configuradas para que en el caso que se de la condición especial mencionada, se ejecuten de a cuerdo a unos lineamientos presentados en el manual del Cuerpo de Ingenieros del Ejército de los Estados Unidos (USACE) [14], cuya expresión matemática para el método de Bishop simplificado es la 2.26.

$$f_s = \frac{\sum c'b + (W + P_w\cos\beta - ub)\tan\phi' m_\alpha^{-1}}{\sum W\sin\alpha - \frac{1}{R}\sum m_\alpha} \tag{2.26}$$

donde

- P_w es la fuerza externa ejercida por la columna de agua por encima de la dovela sumergida, es decir el resultado de dividir la presión ejercida por la columna de agua entre la longitud del tope de la dovela,
- β es la inclinación del tope de la dovela y
- R es el radio del circulo.

En el presente programa se ha desarrollado las funciones que calculan de forma respectiva f_s por los métodos de Fellenius y de Bishop simplificado.

En sí, el cálculo de f_s no toma muchas líneas de código; en el primer método las ecuaciones no necesitan iteraciones numéricas para llegar a la solución, mientras que el segundo sí. Lo que consume varias líneas de código y recursos de la máquina son otras funciones con las que se crea la estructura de datos de entrada para hacer correr las anteriores funciones, la que muestra de forma gráfica la solución del problema (*i.e.* el dibujo del talud con la superficie del terreno y el contorno de cálculo, la polilínea del nivel freático, el arco de circunferencia que representa la superficie de falla circular y la discretización de las dovelas) y particularmente la función con la que se obtiene la superficie de falla crítica.

2.3. La búsqueda de la superficie de falla crítica

Uno de los beneficios de realizar un código computacional para evaluar el factor de seguridad al deslizamiento por falla circular en un talud bidimensional para una superficie dada, es evitar los tediosos cálculos manuales que se requieren para tal objetivo.

Sin embargo, tal código es relativamente sencillo para cada método. La complejidad está en las múltiples funciones previas que transforman la información de entrada en la estructura de datos para cada dovela en la que será dividida la masa de suelo, y de este modo proceder a evaluarla.

Una virtud mayor es lograr evaluar una cantidad considerable de superficies para determinar aquella cuyo factor de seguridad es más bajo de todos, esta recibe el nombre de *superficie de falla crítica*.

El programa pyCSS® incluye una función que posibilita determinar la superficie de falla crítica, ésta lleva el nombre de `automaticslipcircles` A.1. El algoritmo que la función sigue se describe a continuación:

1. Definir la superficie del talud y el nivel freático si es que se desea (Figura 2.4).
2. Seleccionar un par de puntos aleatorios sobre la superficie del talud (Figura 2.5).
3. Encontrar el circulo con el radio mínimo posible que pase por el par de puntos aleatorios y evaluar esta superficie (Figura 2.6).
4. Aumentar el radio inicial una magnitud l introducida por el usuario para el mismo par de puntos aleatorios.

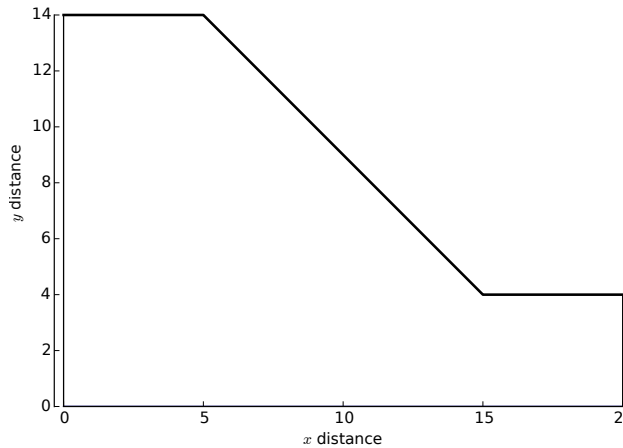


Figura 2.4 Definición de la superficie del talud.

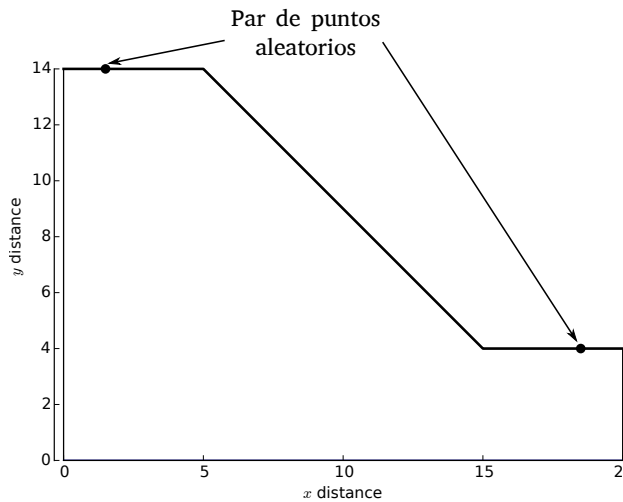


Figura 2.5 Par de puntos aleatorios

5. Verificar que la superficie circular no corte la cara y la pata del talud simultáneamente y corregirla dado caso para proceder a evaluar la nueva superficie (Figura 2.7).

   Como se observa en la Figura 2.7, la corrección consiste en descartar la porción de la superficie circular que no se encuentra dentro de la masa de suelo, conservando el radio intacto.

6. Repetir los dos ítem anterior n veces, donde n también lo introduce el usuario hasta lograr un esquema como el que se muestra en la Figura 2.8.

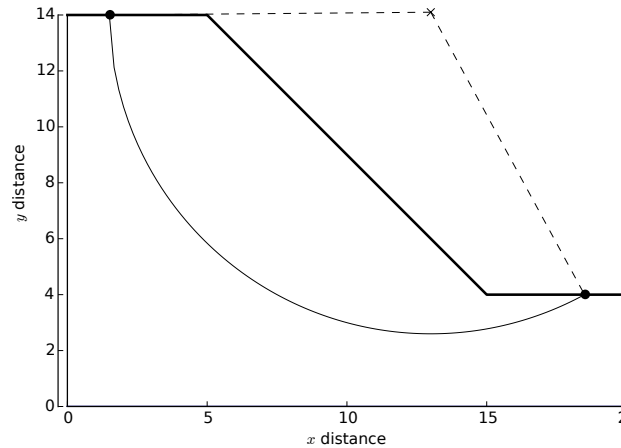


Figura 2.6 Circulo inicial.

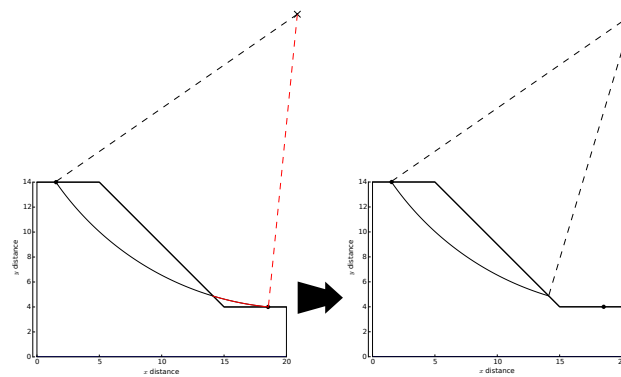


Figura 2.7 Corrección de una superficie circular errónea

7. Definir un nuevo par de puntos aleatorios sobre la superficie y repetir todo el ciclo desde el numeral 2. Esta cantidad de puntos aleatorios es función de la cantidad de superficies que se desean evaluar, lo cual es introducido por el usuario.

Del procedimiento anterior se obtiene una nube de puntos asociados a todos los centros de las superficies circulares evaluadas, donde cada uno también está ligado a un valor de factor de seguridad. De este modo, teniendo esta distribución espacial bidimensional de puntos, es posible interpolar los valore factor de seguridad para visualizar la distribución de los mismos.

Con la visualización de la interpolación mencionada se podrá distinguir la región donde se encontraría el mínimo valor, aunque internamente la función se encarga de buscarlo y mostrárselo al usuario. Es de mencionar, que tal como se cita a varios autores en [4], hay la posibilidad de encontrar una distribución con varios mínimos locales, tal como se ejemplifi-

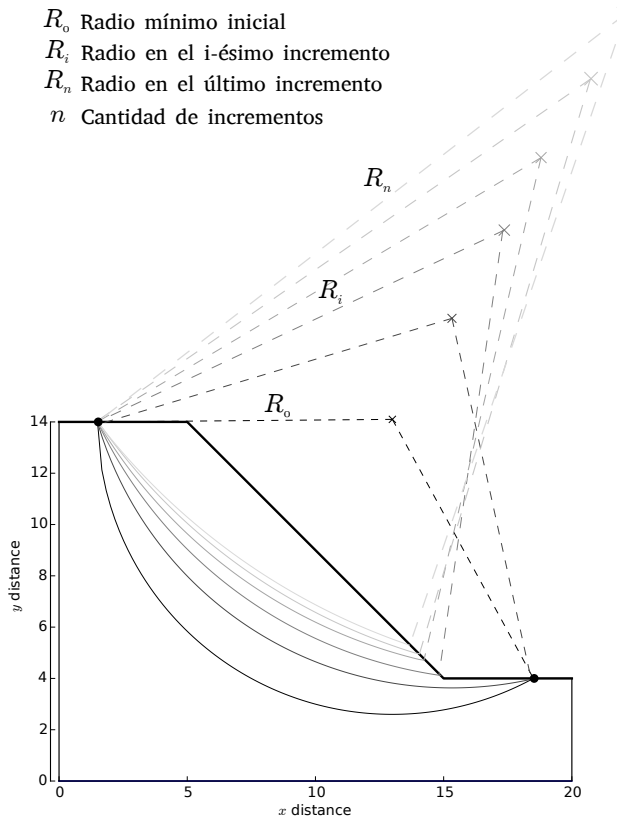


Figura 2.8 Desarrollo de múltiples círculos que pasan por un par de puntos fijos sobre la superficie del talud.

ca en la Figura 2.9 (obtenida con una versión modificada de la función `automaticslipcircles` para este fin didáctico), sin embargo el mínimo global será el que defina la superficie de falla crítica.

Un ejemplo más completo sobre la visualización real de la gráfica que se obtiene con la ejecución de la función se muestra en la Figura 3.3(a).

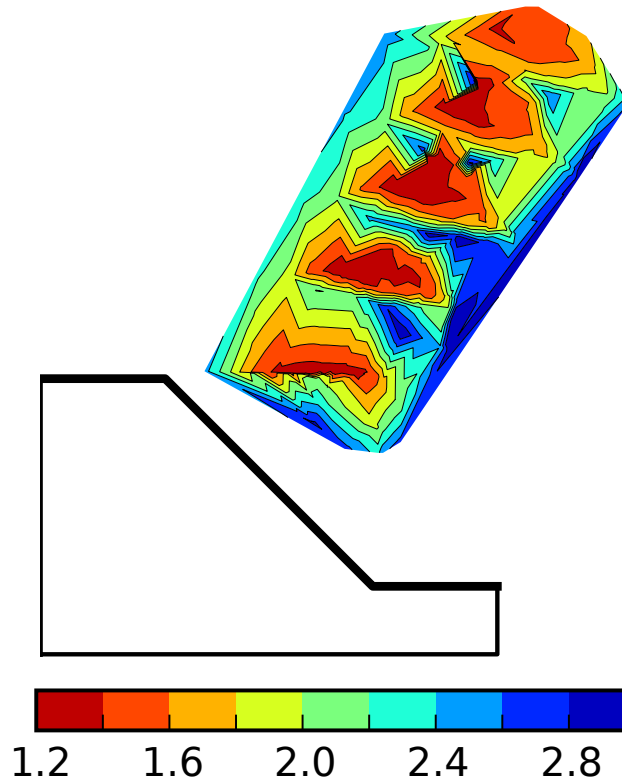


Figura 2.9 Distribución de los valores de factores de seguridad después de la evaluación de un talud; se identifica claramente la presencia de 5 regiones asociadas a mínimos locales.

Capítulo 3
Variables de entrada y archivos de salida

El programa pyCSS® funciona, como es de esperarse, bajo la lógica secuencial *entrada* → *operaciones* → *salida*, esto se puede evidenciar remitiéndose a la Figura 3.1 en la que se muestra la relación entre las variables de entrada y las funciones que usa el módulo `finalModule`.

A continuación se describe las variables de entrada requeridas y los archivos de salida que se obtienen con la ejecución del programa.

3.1. Variables de entrada

Como se mencionó anteriormente, el programa pyCSS® puede ser ejecutado vía archivo de lotes o vía interfaz gráfica; por lo tanto, las variables de entrada se ingresan modificando el módulo `finalModule.py`. Esto se puede realizar con mucha facilidad, modificando el archivo con un editor de texto (vea la sección Recomendaciones en el capítulo 4, según sea el sistema operativo de su preferencia); o por otro lado, simplemente ingresando la información que solicita la interfaz gráfica, la cual internamente convertirá dicha información en los valores de las variables de entrada.

A manera general se comenta que las variables de entrada pueden ser de varios tipos: valores numéricos escalares (`type int` y `type float`); arreglos vectoriales y matriciales (`type array`), cuyas componentes son escalares; cadenas de caracteres (`type str`); variables lógicas (`type bool`); o listas (`type list`), cuyos elementos pueden ser los anteriores o una combinación de ellos.

Se hace un llamado especial al usuario a que sea consciente que el programa ejecutará sus operaciones sean o no lógicos los valores y unidades ingresados a él; por lo tanto, es importante revisar las magnitudes de los mismos y la coherencia de las unidades usadas (*e.g.* usar solo el Sistema Métrico o el Sistema Imperial de Unidades).

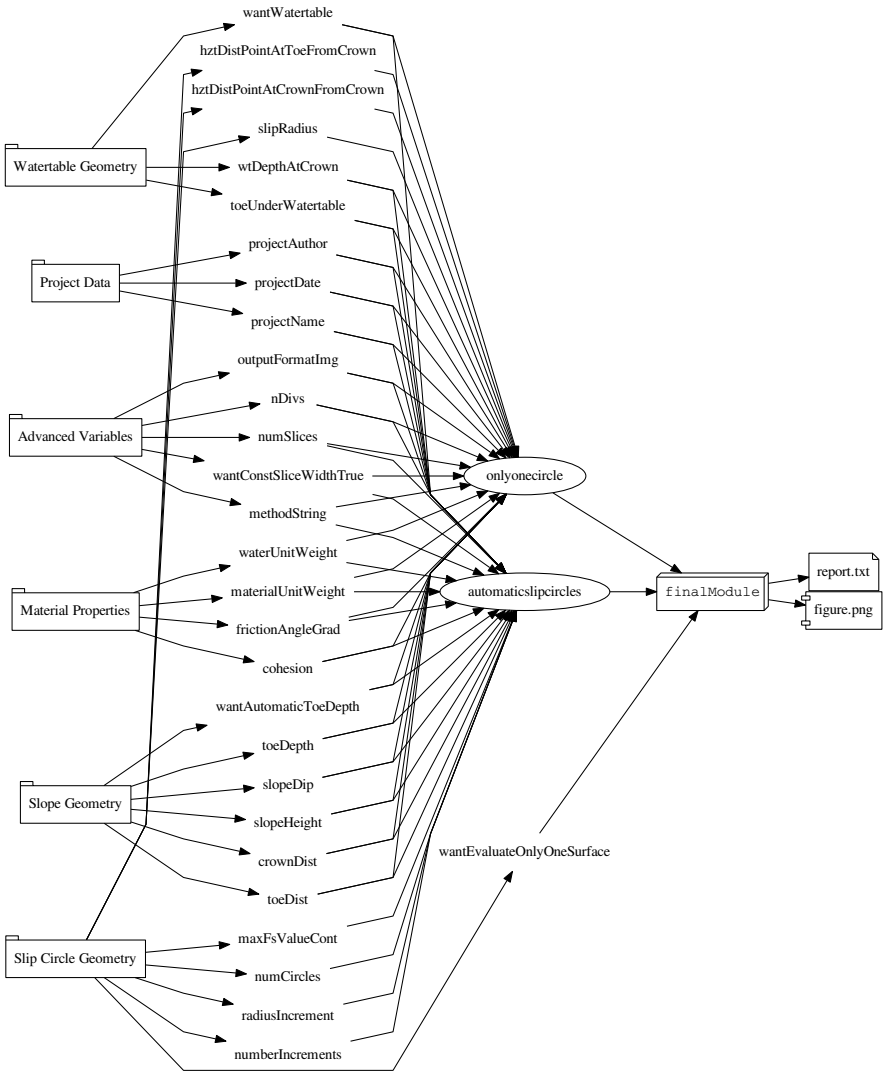


Figura 3.1 Distribución de las variables de entrada y las principales funciones del programa pyCSS® dentro del módulo `finalModule`.

A continuación se describe las variables de entrada básicas que debe ingresar el usuario. Para editar otras variables con valores por defecto, se recomienda leer el capítulo 5.

3.1.1. Información del proyecto

Nombre del proyecto El nombre de la variable es `projectName`. Corresponde a una cadena de caracteres con el nombre o título del proyecto.

Autor del proyecto El nombre de la variable es `projectAuthor`. Corresponde a una cadena de caracteres con el nombre del autor del proyecto (*e.g.*persona natural, empresa, institución, etc.).

Fecha del proyecto El nombre de la variable es `projectDate`. Corresponde a una cadena de caracteres con la fecha en que se ejecuta el programa. Está configurada para que tome la fecha del día en curso usando el módulo `time` que viene predeterminado en PYTHON®; sin embargo se puede modificar a gusto escribiendo la fecha requerida de forma manual.

3.1.2. Geometría del talud

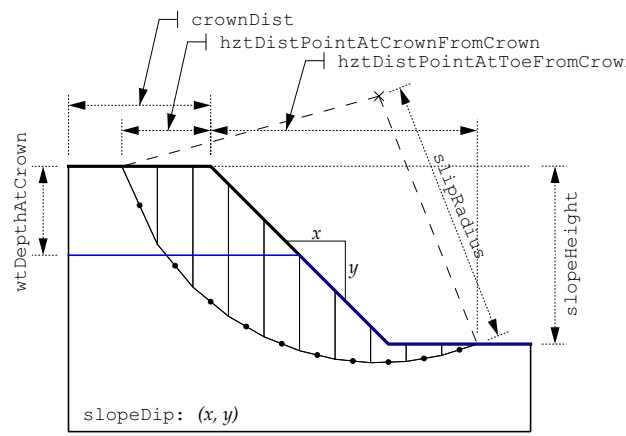


Figura 3.2 Variables de entrada que definen la geometría de un talud estándar, la superficie circular de deslizamiento y el nivel freático.

Altura del talud El nombre de la variable es `slopeHeight`. Corresponde a una lista de longitud 2, donde el primer elemento es de tipo *escalar* con el valor de la altura del talud, y el segundo es de tipo *cadena de caracteres* con la respectiva unidad de medida.

Inclinación de la cara del talud El nombre de la variable es `slopeDip`. Corresponde a un arreglo vectorial bidimensional de tipo *array*, con ambos elementos de tipo *escalar* que representan una relación de distancia horizontal:vertical respectivamente, para indicar la pendiente del talud.

Distancia de la corona El nombre de la variable es `crownDist`. Corresponde a una lista de longitud 2, donde el primer elemento es de tipo *escalar* con el valor de la longitud de la corona del talud, y el segundo es de tipo *cadena de caracteres* con la respectiva unidad de medida.

Distancia del pie El nombre de la variable es `toeDist`. Corresponde a una lista de longitud 2, donde el primer elemento es de tipo *escalar* con el valor de la longitud del pie del talud, y el segundo es de tipo *cadena de caracteres* con la respectiva unidad de medida.

Profundidad del talud El nombre de la variable es `toeDepth`. Corresponde a una lista de longitud 2, donde el primer elemento es de tipo *escalar* con el valor de la profundidad del talud medido desde el pie, y el segundo es de tipo *cadena de caracteres* con la respectiva unidad de medida. Esta variable solo tiene influencia si `wantAutomaticToeDepth` es igual a `False`, de lo contrario será calculada automáticamente de acuerdo al arco circular de mayor longitud de radio que puede quedar circunscrito en el talud.

3.1.3. Superficie de deslizamiento

3.1.3.1. Evaluando una única superficie

La superficie circular de deslizamiento que será analizada se define con dos puntos localizados en la superficie del talud y el radio.

Primer punto El nombre de la variable es `hztDistPointAtCrownFromCrown`. Corresponde a una lista de longitud 2, donde el primer elemento es de tipo *escalar* con el valor de la longitud horizontal del primer punto del circulo (*i.e.*el que está más cerca al vértice de la corona) medido desde el extremo de la corona, justo donde empieza la superficie inclinada del talud (puntos a la izquierda del vértice son negativos, y a la derecha positivos). El segundo elemento es de tipo *cadena de caracteres* con la respectiva unidad de medida.

Segundo punto El nombre de la variable es `hztDistPointAtToeFromCrown`. Corresponde a una lista de longitud 2, donde el primer elemento es de tipo *escalar* con el valor de la longitud horizontal del segundo punto del circulo (*i.e.*el que está más cerca al vértice del pie) medido desde el extremo de la corona, justo donde empieza la superficie inclinada del talud (se entiende que es un punto a la derecha del vértice, por lo tanto se espera que sea positivo). El segundo elemento es de tipo *cadena de caracteres* con la respectiva unidad de medida.

Radio El nombre de la variable es `slipRadius`. Corresponde a una lista de longitud 2, donde el primer elemento es de tipo *escalar* con el valor de la longitud del radio del circulo de deslizamiento, y el segundo es de tipo *cadena de caracteres* con la respectiva unidad de medida.

Las anteriores variables se pueden entender mejor remitiéndose a la Figura 3.2.

3.1.3.2. Evaluando múltiples superficies

Evaluar múltiples superficies es uno de los máximos beneficios de analizar la estabilidad de un talud bajo condiciones de equilibrio límite haciendo uso de un ordenador, ya que dependiendo del número de superficies (hasta miles, o más) podría encontrarse aquella más crítica; es decir, aquella que representa el mínimo valor global del factor de seguridad.

La función implementada en el programa pyCSS® que permite lograr lo anterior se llama `automaticslipcircles`, y su mecanismo se puede entender remitiéndose a la sección 2.3. A continuación solo se describe las variables de entrada asociadas a ella.

Número de superficies consideradas El nombre de la variable es `numCircles`. Corresponde a una variable de tipo *escalar* con el valor entero de la cantidad de superficies que se desean evaluar.

Longitud que aumenta el radio por cada iteración El nombre de la variable es `radiusIncrementlip`. Corresponde a una lista de longitud 2, donde el primer elemento es de tipo *escalar* con la longitud que irá aumentando un radio inicial que junto a dos puntos aleatorios fijos definirán un nuevo círculo cada vez que aumente el radio; el segundo elemento es de tipo *cadena de caracteres* con la respectiva unidad de medida.

Numero de veces que un radio inicial aumenta su longitud El nombre de la variable es `numberIncrements`. Corresponde a una variable de tipo *escalar* con el valor entero de la cantidad de veces que un radio inicial dado aumentará una longitud fijada con la variable `radiusIncrementlip`

Máximo valor de factor de seguridad para mostrar El nombre de la variable es `maxFsValueCont`. Corresponde a una variable de tipo *escalar* con el valor máximo de factor de seguridad que se desea mostrar en el diagrama de contornos que muestra la distribución de los valores obtenidos al evaluar todas las superficies.

3.1.4. Nivel freático

El programa está configurado por defecto para que el usuario defina un nivel freático; sin embargo, en caso de no requerirlo basta con identificar la variable `wantWatertable` y cambiar su valor por defecto de `True` a `False` y no modificar las demás variables.

Profundidad del nivel freático El nombre de la variable es `wtDepthAtCrown`. Corresponde a una lista de longitud 2, donde el primer elemento es de tipo *escalar* con el valor de la longitud de la profundidad del nivel freático, medido desde la corona del talud, y el segundo es de tipo *cadena de caracteres* con la respectiva unidad de medida.

Talud parcialmente sumergido El nombre de la variable es `toeUnderWatertable`. Es de tipo *booleano* con valor por defecto `False`. Modificar esta variable es opcional: cuando se desee que el talud esté parcialmente sumergido se debe cambiar por `True`.

3.1.5. Propiedades de los materiales

Peso específico del agua El nombre de la variable es `waterUnitWeight`. Corresponde a una lista de longitud 2, donde el primer elemento es de tipo *escalar* con el valor del peso específico del agua, y el segundo es de tipo *cadena de caracteres* con la respectiva unidad de medida.

Peso específico del material del talud El nombre de la variable es `materialUnitWeight`. Corresponde a una lista de longitud 2, donde el primer elemento es de tipo *escalar* con el valor del peso específico del material que conforma el talud, y el segundo es de tipo *cadena de caracteres* con la respectiva unidad de medida.

Ángulo de fricción interna del material del talud El nombre de la variable es `frictionAngleGrad`. Corresponde a una lista de longitud 2, donde el primer elemento es de tipo *escalar* con el ángulo en grados del material que conforma el talud, y el segundo es de tipo *cadena de caracteres* con la respectiva unidad de medida.

Cohesión del material del talud El nombre de la variable es `cohesion`. Corresponde a una lista de longitud 2, donde el primer elemento es de tipo *escalar* con el ángulo en grados del material que conforma el talud, y el segundo es de tipo *cadena de caracteres* con la respectiva unidad de medida.

3.2. Archivos de salida

Son dos archivos los que se obtiene con la ejecución del programa pyCSS® : un archivo imagen y un archivo de texto plano.

3.2.1. Archivo de imagen

Corresponde a un archivo en formato .svg, en el cuál se esquematiza el análisis hecho. Esto incluye la superficie del talud a escala, la superficie del nivel freático, el circulo de deslizamiento, las dovelas, y una de las siguientes dos opciones:

- el factor de seguridad obtenido mediante los dos métodos en el caso de evaluar una única superficie circular, o
- el mapa de contornos de los factores de seguridad y la superficie crítica en el caso de evaluar múltiples superficies.

Las figuras 3.3(a) y 3.3(a) muestran un ejemplo de cómo se visualizarán los archivos de imagen de salida según corresponda a uno de los dos casos anteriores.

Para usuarios que prefieran un formato rasterizado diferente al .png, o que necesiten un formato vectorial, se recomienda leer el capítulo 5, para que identifique la forma de lograr tal interés.

3.2.2. Archivo de texto

Corresponde a un archivo en formato .txt, en el cuál se resume la información del análisis hecho. Este está estructurado en secciones, incluyendo las siguientes:

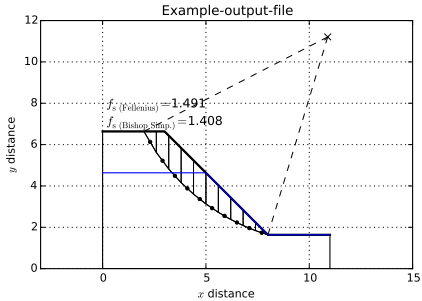
1. Información general.
2. Geometría del talud.
3. Superficie circular de deslizamiento.
4. Geometría y datos del nivel freático.
5. Propiedades de los materiales.
6. Datos de las dovelas.

A continuación se muestra el contenido del archivo de texto asociado a la Figura 3.3(a); uno similar tomaría lugar para la figura 3.3(b).

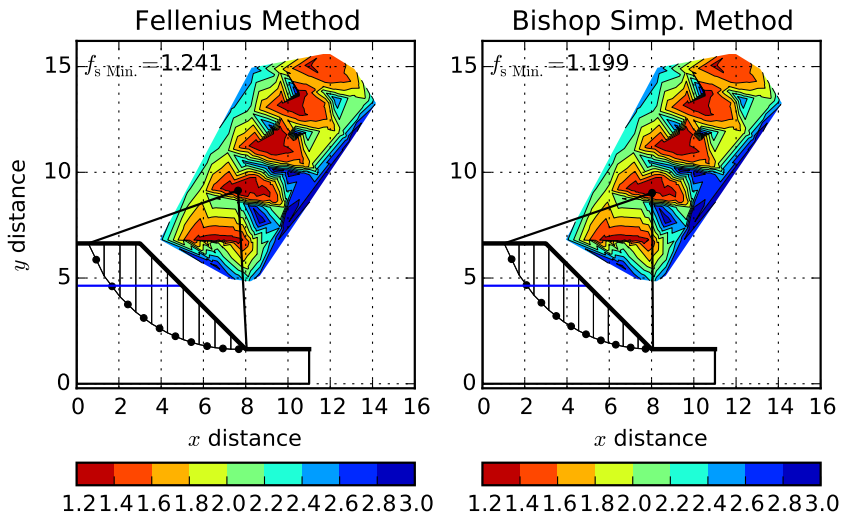
```
---SUMMARY OF PROJECT---

Project name: Example-output-file
Author: Exneyder A. Montoya Araque
Date: 22/04/16
Safety factors:
        -Fellenius method: 1.49130375181
        -Bishop method: 1.40821871205

--Slope geometry--
Height: 5 m
Dip: 89.2328896038 degrees
Crown distance: 3 m
Toe distance: 3 m
Toe depth: 1.63636363636 m
```



(a) Ejemplo del archivo de imagen que se obtiene con la ejecución de la función `onlyonecircle` del programa.



(b) Ejemplo del archivo de imagen que se obtiene con la ejecución de la función `automaticslipcircles` del programa.

Figura 3.3 Tipos de archivos de imagen que se pueden obtener con la ejecución de pyCSS®.

```
Surface coordinates:
[[  0.          6.63636364]
 [  3.          6.63636364]
 [  8.          1.63636364]
```

```
 [ 11.           1.63636364]]

--Watertable geometry--
Depth at crown: 2 m
Is the slope partially submerged? No
Watertable coordinates:
[[  0.           4.63636364]
 [  5.           4.63636364]
 [  8.           1.63636364]
 [ 11.           1.63636364]]

--Slip circle--
Radius: 10 m
Center coordinates: [ 10.89352242  11.20859054]

--Materials properties--
Water unit weight: 9.81 kN/m3
Soil unit weight: 17 kN/m3
Friction angle 25 degrees
Cohesion: 10 kPa

--Slices data--
Number of slices: 10
Has the surface slip constant width slices? Yes

Slices structures data:
[['Index' 'Abscissa' 'Ordinate' 'Area' 'Width' 'Height'
'Secant Angle Grad at Bottom' 'Angle Grad at Top' 'Water Height'
'Water Height Above Slope' 'Horizontal Moment Arm' 'Vertical Moment Arm']
['0.0' '2.3' '6.1' '0.3' '0.6' '0.5' '59.4' '-0.0' '0.0' '0.0' '-8.6'
 '4.6']
['1.0' '2.9' '5.2' '0.8' '0.6' '1.3' '53.2' '18.4' '0.0' '0.0' '-8.0'
 '4.6']
['2.0' '3.5' '4.5' '1.0' '0.6' '1.6' '47.7' '45.0' '0.1' '0.0' '-7.4'
 '5.1']
['3.0' '4.1' '3.9' '1.0' '0.6' '1.7' '42.8' '45.0' '0.8' '0.0' '-6.8'
 '5.7']
['4.0' '4.7' '3.4' '0.9' '0.6' '1.6' '38.3' '45.0' '1.3' '0.0' '-6.2'
 '6.3']
['5.0' '5.3' '2.9' '0.8' '0.6' '1.4' '34.0' '45.0' '1.4' '0.0' '-5.6'
 '6.9']
['6.0' '5.9' '2.6' '0.7' '0.6' '1.2' '30.0' '45.0' '1.2' '0.0' '-5.0'
 '7.5']
['7.0' '6.5' '2.2' '0.5' '0.6' '0.9' '26.1' '45.0' '0.9' '0.0' '-4.4'
 '8.1']
['8.0' '7.1' '2.0' '0.3' '0.6' '0.6' '22.3' '45.0' '0.6' '0.0' '-3.8'
 '8.7']
['9.0' '7.7' '1.7' '0.1' '0.6' '0.2' '18.6' '45.0' '0.2' '0.0' '-3.2'
 '9.3']]

Note: This program calculated the safety factor to circular slip,
under limit equilibrium considerations, using Fellenius and Bishop
methods. The imagen attached shows the calculation performed.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Capítulo 4
Instalación y ejecución

4.1. Descarga

El programa está alojado en la plataforma para el hospedaje de códigos, que permite la colaboración y control de versiones, denominada *GitHub* bajo el nombre de pyCSS.

Allí podrá obtener un archivo comprimido `.zip` que contiene el programa pyCSS® y el presente manual del usuario en formato `.pdf`.

Descargue el archivo en cualquier directorio y descomprímalo (basta con dar clic derecho sobre el archivo `.zip` y seleccionar la opción "descomprimir aquí" o "extraer aquí"), de tal modo que el directorio raíz del programa (*i.e.* donde se aloja el módulo `finalModule.py` y la carpeta *functions*) sea por ejemplo, si está usando un sistema operativo basado en Unix:

```
/home/userName/python/pyCSS/
```

o si está usando un sitema opertivo WINDOWS®:

```
C:\Users\userName\python\pyCSS\
```

4.2. Instalación y preparativos previos

Si bien el programa pyCSS® no requiere un proceso de instalación estricto para su ejecución, es necesario cumplir unos preparativos en el sistema operativo para su funcionamiento:

4.2.1. GNU/Linux: Ubuntu ®

Instalación del intérprete: lo primero es asegurarse que se tiene instalado el intérprete PYTHON® 3.x en la máquina; aquí se encuentra una ventaja para la distribución UBUNTU®, ya que ésta lo trae incorporado; esto lo puede corroborar escribiendo

```
python3 --version
sudo apt-get install python-support
sudo update-python-modules -a
```

en la terminal; en caso que no esté instalado, basta con ejecutar la siguiente línea allí mismo:

```
sudo apt-get install python3
```

Instalación de módulos externos: una vez instalado el intérprete, es necesario instalar 2 paquetes (*i.e.* conjunto de módulos de PYTHON®) en este caso con utilidad para el cálculo científico–ingenieril, un paquete para la esquematización bidimensional, y un paquete que permite cargar la interfaz gráfica. Estos paquetes puede que no vengan precargados con la instalación de PYTHON® pero que son llamadas en algún momento por pyCSS® Ḋe este modo, su instalación se logra ejecutando las siguientes lineas en la terminal:

```
sudo apt-get install python3-numpy
sudo apt-get install python3-scipy
sudo apt-get install python3-matplotlib
sudo apt-get install python3-tk
```

Instalación del editor de texto: finalmente solo queda asegurarse que se dispone de un buen editor de texto, que para el caso de UBUNTU®, no es necesario realizar una extensa búsqueda, ya que éste trae incorporado GEDIT®, el cual entiende la sintaxis de PYTHON®, permitiendo una mejor visualización del código.

4.2.2. Microsoft Windows ®

Instalación del intérprete: lo primero es asegurarse que se tiene instalado el intérprete PYTHON® 3.x en la máquina. Esto se logra descargando el ejecutable de la versión más reciente de PYTHON®3.x desde la web oficial de la *Python Software Foundation [US]*.

Configuración de las variables de entorno: una vez instalado el intérprete, es necesario establecer su directorio de instalación en el `PATH` de WINDOWS®. Esto se logra escribiendo las siguientes lineas en la ventana del Símbolo del Sistema:

```
set path=%path%;C:\Python3x\
set path=%path%;C:\Python3x\Scripts
```

donde la `x` cambia según la versión que haya instalado de PYTHON®, por ejemplo si instaló PYTHON®3.4.3, quedaría `set path=%path%;C:\Python34\`.

Instalación de módulos externos: a continuación se deben instalar 3 paquetes (*i.e.* conjunto de módulos de PYTHON®; en este caso con utilidad para el cálculo científico–ingenieril y para la esquematización bidimensional), que no vienen pre-cargados con PYTHON® pero que son llamadas en algún momento por pyCSS® Ėsto se logra con el administrador de paquetes `pip` ejecutando las siguientes lineas en la terminal

```
pip install numpy
pip install scipy
pip install matplotlib
```

Segunda opción para la instalación el intérprete PYTHON® y los paquetes externos: en algunas máquinas se ha visto que el administrador `pip` presenta errores al instalar los paquetes; por lo que hay que recurrir a otros administradores como por ejemplo `conda`, el cual viene incluido en la distribución de PYTHON®denominada ANACONDA®, la cual incluye el intérprete, los paquetes que necesita pyCSS® (y otros tantos más), el administrador de paquetes y un *IDE* llamado SPYDER®.

Instalación del editor de texto: finalmente solo queda asegurarse que se dispone de un buen editor de texto, que para el caso de WINDOWS®bastaría con el bloc de notas, pero se recomienda uno más potente como NOTEPAD^{++}® el cual entiende la sintaxis de PYTHON®, permitiendo una mejor visualización del código.

4.3. Requerimientos del sistema

Intérprete: para usar el presente código se requiere de un lenguaje intérprete específico que lea el código desarrollado; este es PYTHON®en su versión 3.x.

Máquina: se requiere de un computador *común* de la temporada (*e.g.* computador de escritorio o portátil) con dos principales software: un sistema operativo y el intérprete de un lenguaje de programación específico mencionado anteriormente.

Sistema Operativo: en lo que respecta a esto, el código abierto desarrollado corre en múltiples plataformas. Es decir, que puede ser ejectudado bajo WINDOWS® (*e.g.* XP, 7, 8, 8.1 y 10), MacOS® o Linux (*e.g.* DEBIAN, UBUNTU® RedHat®).

Las pruebas que se hicieron para este código fueron en los siguientes sistemas operativos: GNU/Linux UBUNTU® 15.10 y WINDOWS® 7 y 8.1.

Instalación de paquetes externos: la instalación de los paquetes que usa PYTHON® es más fácil bajo los sistemas operativos basados en Linux, tal y como se describió en la sección 4.2.

4.4. Ejecución del programa

4.4.1. Vía interfaz gráfica

4.4.1.1. GNU/Linux: Ubuntu ®

1. Copie la ruta del directorio donde tiene almacenado el módulo `pyCCS.py` junto a la carpeta *functions*.
2. Abra la terminal de comandos y acceda a la ruta que acaba de copiar escribiendo la siguiente línea en la terminal:

```
cd /home/userName/pyCSS
```

   donde `/home/userName/pyCSS` es la ruta del directorio que contiene el módulo `pyCCS.py` localizado junto a la carpeta *functions*.
3. Finalmente ejecute el módulo `pyCCS.py` escribiendo la siguiente línea en la terminal:

```
python3 pyCCS.py
```

Lo anterior mostrará la interfaz gráfica del programa (Figura 4.1), la cual es muy intuitiva, y después de ingresar las variables de entrada, solo bastará con dar clic al botón de *Ejecutar análisis*.

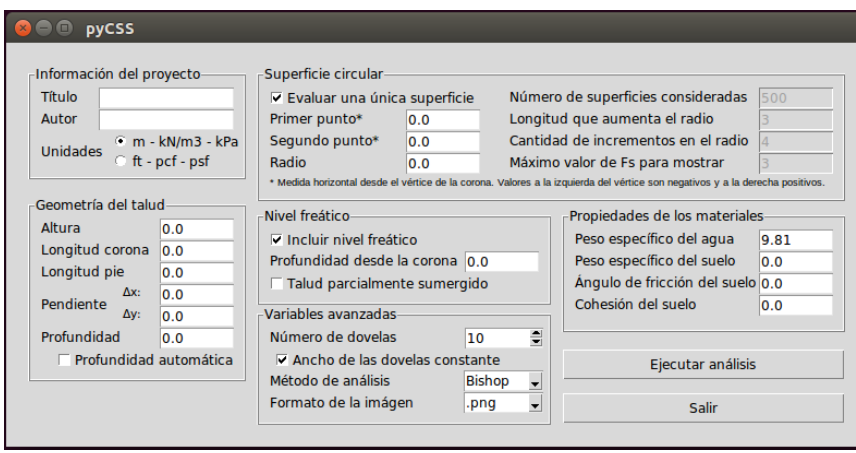


Figura 4.1 Vista de la interfaz gráfica del programa pyCSS® ejecutada en UBUNTU®.

4.4.1.2. Microsoft Windows ®

1. Una vez instalado el intérprete, solo es necesario dar doble clic sobre el archivo `pyCSS.py` localizado junto a la carpeta *functions*.

Lo anterior mostrará la interfaz gráfica del programa (Figura 4.2), la cual es muy intuitiva, y después de ingresar las variables de entrada, solo bastará con dar clic al botón de *Ejecutar análisis*.

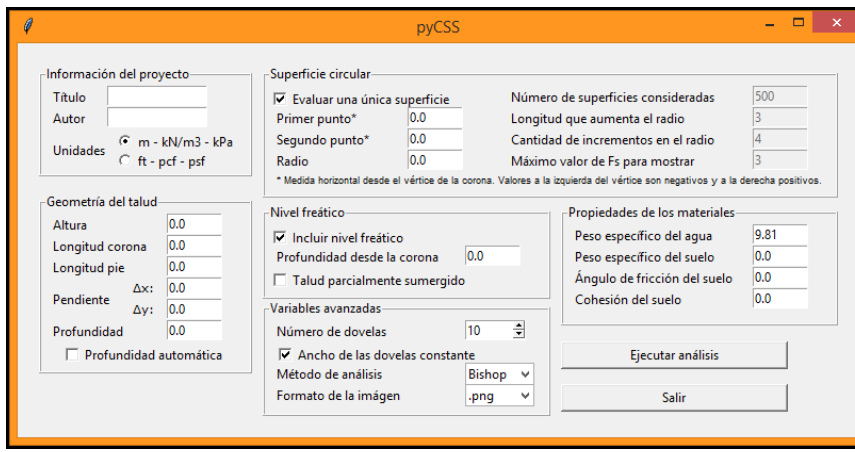


Figura 4.2 Vista de la interfaz gráfica del programa pyCSS® ejecutada en WINDOWS®.

4.4.2. Vía archivo de lotes desde la Terminal de comandos o el Símbolo de sistema

4.4.2.1. GNU/Linux: Ubuntu ®

1. Edite las variables de entrada en el módulo `finalModule.py` de acuerdo a la necesidad de su proyecto.
2. Copie la ruta del directorio donde tiene almacenado el módulo `finalModule.py` junto a la carpeta *functions*.
3. Abra la terminal de comandos y acceda a la ruta que acaba de copiar escribiendo la siguiente línea en la terminal:

```
cd /home/userName/pyCSS
```

donde `/home/userName/pyCSS` es la ruta del directorio que contiene el módulo `finalModule.py` junto a la carpeta *functions*.

4. Finalmente ejecute el módulo `finalModule.py` escribiendo la siguiente línea en la terminal:

```
python3 finalModule.py
```

Note que `finalModule.py` es el nombre del módulo, y en caso de cambiarlo debe conservar la extensión `.py`.

4.4.2.2. Microsoft Windows ®

1. Edite las variables de entrada en el módulo `finalModule.py` de acuerdo a la necesidad de su proyecto.
2. Copie la ruta del directorio donde tiene almacenado el módulo `finalModule.py` junto a la carpeta *functions*.
3. Abra la ventana del Símbolo de sistema y acceda a la ruta que acaba de copiar escribiendo la siguiente línea en la consola:

```
cd C:\Users\userName\pyCSS
```

donde `C:\Users\userName\pyCSS` es la ruta del directorio que contiene el módulo `finalModule.py` junto a la carpeta *functions*.

4. Finalmente ejecute el módulo `finalModule.py` escribiendo la siguiente línea en la consola:

```
python finalModule.py
```

Note que `finalModule.py` es el nombre del módulo, y en caso de cambiarlo debe conservar la extensión `.py`.

Capítulo 5
Opciones avanzadas

5.1. Otras variables de entrada

En la sección 3.1 se describieron la variables de entrada básicas para ejecutar el programa pyCSS® . A continuación se describirán otro conjunto de variables, que aunque en el módulo `finalModule.py` traen valores predeterminados, pueden ser modificados de acuerdo a la necesidad del usuario.

Ancho constante de las dovelas: El nombre de la variable es `wantConstSliceWidthTrue`. Corresponde a una variable lógica tal que si su valor es `True` el ancho de todas las dovelas será el mismo, mientras que si su valor es `False` las dovelas se distribuirán de tal forma que los límites entre ellas coincidan con los vértices de la superficie del talud.

Número de dovelas: El nombre de la variable es `numSlices`. Corresponde a una variable de tipo *escalar*, que define el número de dovelas en que se dividirá la masa de suelo en consideración.

Número de segmentos en que se discretiza la superficie de deslizamiento: El nombre de la variable es `nDivs`. Corresponde a una variable de tipo *escalar*, que define el número de segmentos en el que se dividirá el arco circular que define la superficie de deslizamiento. Se recomienda que este sea igual al valor definido en la variable `numSlices`.

Método de análisis: El nombre de la variable es `methodString`. Corresponde a una cadena de caracteres con la abreviación del método de análisis de equilibrio límite que se usará. Puede ser `'Flns'` para el método de Fellenius, `'Bshp'` para el método de Bishop, o `'Allm'` para ambos.

Formato de la imagen: El nombre de la variable es `outputFormatImg`. Corresponde a una cadena de caracteres con la extensión del archivo de imagen que genera el programa. Puede elegir entre múltiples formatos, tanto rasterizados como vectoriales: `'.eps'`, `'.jpeg'`, `'.jpg'`, `'.pdf'`, `'.pgf'`, `'.png'`, `'.raw'`, `'.rgba'`, `'.svg'`, `'.svgz'`, `'.tif'`, o `'.tiff'`.

5.2. Modifique el código a su agrado

La ventaja de tener un código libre es que usted puede mejorar y adaptar el mismo a su agrado para que él sea un complemento o una herramienta de otras aplicaciones que está desarrollando.

Si usted estudia el código de pyCSS® podrá ver que su modificación y adaptación a sus necesidades es una tarea poco difícil. Siéntase libre de hacerlo.

Capítulo 6
Validación

6.1. Fuente de información

Para la validación del programa pyCSS® se procedió a comparar los resultados obtenidos con él respecto a cinco diferentes fuentes.

1. Ejemplo 2 (figura 6) del artículo de Chang [3].
2. Cálculo manual del ejemplo de la figura 4-3 del manual del Cuerpo de Ingenieros del Ejército de los Estados Unidos (USACE) [14], siguiendo los lineamientos planteados por ellos mismos.
3. Ejemplo 1 (figura 2, sección 4.1) del artículo de Zhao y coautores [15].
4. Igual que el anterior, adicionando nivel freático, evaluado en el software comercial SLIDE®.
5. Igual que el anterior, considerando el talud parcialmente sumergido.

6.2. Procedimiento

Se generaron cinco copias del módulo `finalModule.py`, guardando cada una con los nombres que aparecen en la carpeta *validations*.

Luego de esto, se procedió a ingresar las variables de entrada de acuerdo a las características de los ejemplos que se usaron para la validación.

Finalmente se ejecutaron estos cinco módulos y se obtuvieron los archivos de salida que llevan los mismos nombres que su respectivo módulo, y se encuentran igualmente en la carpeta *validations*.

6.3. Verificación con archivos externos

Para el caso de la validación 02, se tiene el archivo `validation02-comparisonUSarmyCorpsOf Engineers.xlsx`, localizado en la subcarpeta *externalValidationFiles*, que corresponde a una hoja de cálculo en la que se realizaron las operaciones manuales siguiendo los lineamientos del manual del USACE [14], para encontrar el factor de seguridad.

Para los casos de las validaciones 04 y 05 se tienen los archivos `validation04-no-SubmergedSlope.slim` y `validation05-partiallySubmergedSlope.slim` respectivamente, localizados igualmente en la subcarpeta *externalValidationFiles*, los cuales se ejecutan en el software comercial SLIDE® con el fin de realizar las últimas pruebas comparativas finales.

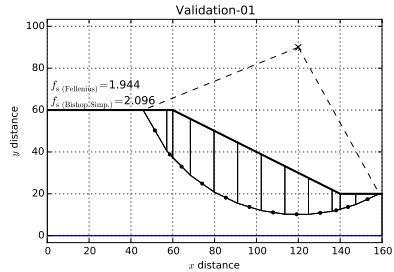
6.4. Resultados

Cuadro 6.1 Comparación de los resultados obtenidos entre pyCSS® y las otras fuentes.

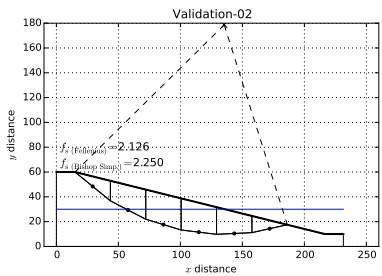
Validación	pyCSS		Otra fuente			Error absoluto e_{abs}	
	Fellenius	Bishop Simp.	Fellenius	Bishop Simp.	Referencia	Fellenius	Bishop Simp.
01	1.944	2.096	1.928	2.080	[3]	1.6 %	1.6 %
02	2.126	2.250	2.133	2.302	[14]	0.7 %	5.2 %
03	0.967	0.992	0.967	0.992	[15]	0.0 %	0.0 %
04	0.750	0.736	0.749	0.736	SLIDE®	0.1 %	0.0 %
05	0.962	1.004	0.938	0.966	SLIDE®	2.4 %	3.8 %

Asumiendo que la información tomada de las otras fuentes para las validación del programa corresponde a los valores correctos, se puede observar que en general el programa pyCSS® presenta diferencias en los resultados por debajo del 5 %, lo que demuestra que está desempeñando correctamente los cálculos.

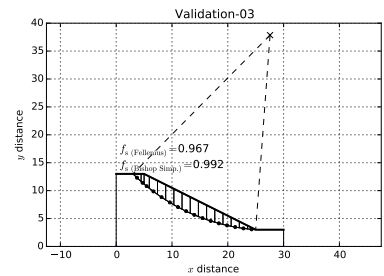
Particularmente se observa que los valores de e_{abs} más altos corresponden al método de Bishop Simplificado para los numerales 02 y 05, donde la característica en común entre ellos es que en ambos se consideró el talud parcialmente sumergido.



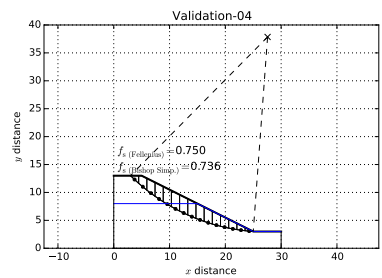
(a) Esquema de la validación 01



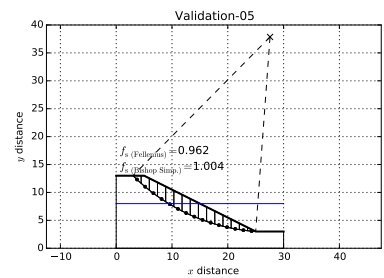
(b) Esquema de la validación 02



(c) Esquema de la validación 03



(d) Esquema de la validación 04



(e) Esquema de la validación 05

Figura 6.1 Esquemas de las 5 validaciones realizadas.

Capítulo 7
Ejemplos

7.1. Cálculo del factor de seguridad al deslizamiento por falla circular de un talud sin nivel freático

Este ejemplo corresponde a la traducción del Inglés del ejercicio 15.20-a del libro de Braja M. Das [5] (página 573):

> Refiriéndose a la figura 7.1 y usando el método ordinario de las dovelas **(*i.e.* Método de Fellenius) y el método de Bishop simplificado**, encuentre el factor de seguridad al deslizamiento para el siguiente caso.
>
> $\beta = 45\,°$, $\phi = 20\,°$, $c' = 400\,\text{lb ft}^{-2}$, $\gamma = 115\,\text{lb ft}^{-3}$, $H = 40\,\text{ft}$, $\alpha = 30\,°$, y $\theta = 70\,°$

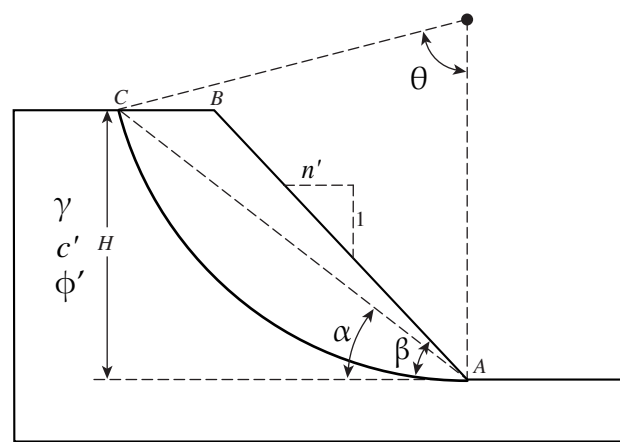


Figura 7.1 Esquema de la Figura 15.50 de la referencia [5].

El texto en negrita fue añadido por los autores del presente manual para complementar el ejercicio.

7.1.1. Solución

1. Cálculo de la longitud $\overline{AC}$

$$\overline{AC} = \frac{H}{\sin(30)} = \frac{40\,\text{m}}{\sin(30)} = 80\,\text{m}$$

2. Cálculo del radio R:

$$R = \frac{0.5\overline{AC}}{\sin(0.5\,\theta)} = \frac{0.5 \times 80\,\text{m}}{\sin(0.5 \times 70^\circ)} = 69.738\,\text{m}$$

3. Teniendo todos los datos previos, se procede a ejecutar el siguiente archivo de lotes para la solución del ejercicio

```
'''
# Description.
This is a minimal module in order to perform a circular arc slope stability
analysis for the example number 01. #
'''

#-------------------------------------------------------------------------------#
### Add functions directory ###
import sys
sys.path += ['../functions']

#-------------------------------------------------------------------------------#
## Modules/Functions import
import numpy as np
import time
from onlyonecircle import onlyonecircle

#-------------------------------------------------------------------------------#

### Poject data ###
projectName = 'Example-01'
projectAuthor = 'Exneyder A. Montoya Araque'
projectDate = time.strftime("%d/%m/%y")

#-------------------------------------------------------------------------------#
### Previous  calculations ###
segmentAC = 40/np.sin(np.radians(30))

### Define inputs ###
# The slope geometry #
slopeHeight = [40, 'ft']
slopeDip = np.array([1, np.tan(np.radians(45))])
crownDist = [30, 'ft']
toeDist = [30, 'ft']
wantAutomaticToeDepth = False
```

```
toeDepth = [10, 'ft']
# The slip arc-circle #
hztDistPointAtCrownFromCrown = [40/np.tan(np.radians(45))-\
        40/np.tan(np.radians(30)), 'ft']
hztDistPointAtToeFromCrown = [40/np.tan(np.radians(45)), 'ft']
slipRadius = [0.5*80/np.sin(0.5*np.radians(70)), 'ft']
# Watertable #
wantWatertable = False
wantWatertable = False
wtDepthAtCrown = ['No watertable']
toeUnderWatertable = False
# Materials properties #
waterUnitWeight = [62.4, 'pcf']
materialUnitWeight = [115, 'pcf']
frictionAngleGrad = [20, 'degrees']
cohesion = [400, 'psf']

### Advanced inputs ###
# Want divide the slip surface in constant width slices? #
wantConstSliceWidthTrue = True
# Number of discretizations of slip surface. #
numSlices = 15
# Number of discretizations of circular arcs. #
nDivs = numSlices
# Select the method to calcualte the safety factor ['Flns', 'Bshp' or 'Allm'] #
methodString = 'Allm'
# Select the output format image ['.eps', '.jpeg', '.jpg', '.pdf', '.pgf', \ #
# '.png', '.ps', '.raw', '.rgba', '.svg', '.svgz', '.tif', '.tiff']. #
outputFormatImg = '.pdf'

#-----------------------------------------------------------------------------#
# Operations for only one slip surface #
msg = onlyonecircle(projectName, projectAuthor, projectDate, slopeHeight, \
        slopeDip, crownDist, toeDist, wantAutomaticToeDepth, toeDepth, \
        hztDistPointAtCrownFromCrown, hztDistPointAtToeFromCrown, \
        slipRadius, wantWatertable, wtDepthAtCrown, toeUnderWatertable, \
        waterUnitWeight, materialUnitWeight, frictionAngleGrad, cohesion, \
        wantConstSliceWidthTrue, numSlices, nDivs, methodString, \
        outputFormatImg)

#'''
BSD 2 license.

Copyright (c) 2016, Universidad Nacional de Colombia, Ludger O.
Suarez-Burgoa and Exneyder Andres Montoya Araque.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
```

```
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#'''#
```

4. Resultados: (ver figura 7.2)

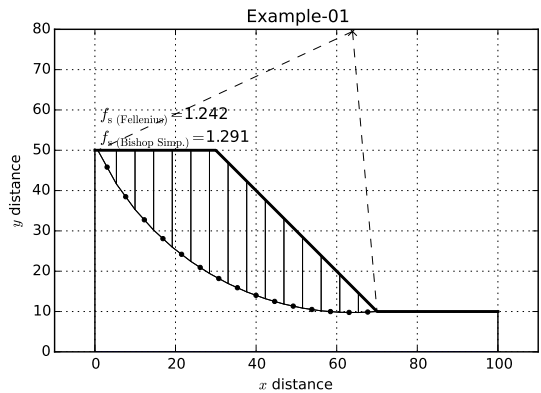


Figura 7.2 Resultados obtenidos para el ejemplo 01: $f_{s\,(\text{Fellenius})} = 1.242$, $f_{s\,(\text{Bishop Simp.})} = 1.291$.

7.2. Cálculo del factor de seguridad al deslizamiento por falla circular de un talud con nivel freático

Este ejemplo corresponde al ejercicio 2.4 del libro de Suárez-Burgoa [12] (página 57):

> En un punto específico de una vía, se decidió analizar un talud con una superficie de falla circular, como se muestra en la Figura 7.3.
>
> Calcule el factor de seguridad por el método de Bishop simplificado solo para un factor de seguridad semilla. El factor de seguridad semilla se escoge de forma *aleatoria* entre el rango $[0.21, 1.89]$.
>
> Se encontró por medio de ensayos de laboratorio que el ángulo de fricción interna saturado en condiciones drenadas del suelo (ϕ) es de 20 °; la cohesión saturada drenada del mismo suelo (c') igual a 4.5 kN m^{-2}; el peso unitario seco (γ_d) igual a 13 kN m^{-3}; y la gravedad específica (G_s) de 2.4.

Se extrajo muestras inalteradas en la porción del suelo por encima del nivel freático, donde se vio que ellas tenían un contenido de humedad (w) de 0.18.

La posición del nivel freático es horizontal detrás del talud a una cota de 15.5 m y todos coincidentes con el nivel del terreno a partir del punto donde el nivel freático corte con la superficie la cara del talud. Use el peso unitario saturado en los cálculos para el cálculo del peso para aquellas porciones de suelo que estén por encima del nivel freático (*i.e.* $\gamma^* = \gamma_{sat}$). Divida la superficie de falla en 10 dovelas de ancho constante.

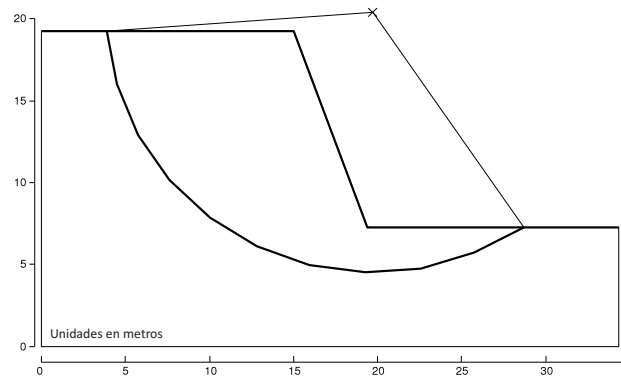


Figura 7.3 Figura 2.14 de la referencia [12].

7.2.1. Solución

1. Cálculo de la relación de vacíos e

$$e = \frac{\gamma_w}{\gamma_d} G_s - 1 = \frac{9.8\,\text{kN}\,\text{m}^{-3}}{13\,\text{kN}\,\text{m}^{-3}} \times 2.4 - 1 = 0.809$$

2. Como el ejercicio indica que se debe considerar $\gamma^* = \gamma_{sat}$, se procede a calcular γ_{sat}:

$$\gamma_{sat} = \frac{G_s + e}{1 + e} \times \gamma_w = \frac{2.4 + 0.809}{1 + 0.809} \times 9.8\,\text{kN}\,\text{m}^{-3} = 17.383\,\text{kN}\,\text{m}^{-3}$$

3. Teniendo todos los datos previos, se procede a ejecutar el siguiente archivo de lotes para la solución del ejercicio:

```
'''
# Description.
This is a minimal module in order to perform a circular arc slope stability
```

```
analysis for the example number 02. #
'''

#-------------------------------------------------------------------------------#
### Add functions directory ###
import sys
sys.path += ['../functions']

#-------------------------------------------------------------------------------#
## Modules/Functions import
import numpy as np
import time
from onlyonecircle import onlyonecircle

#-------------------------------------------------------------------------------#
### Poject data ###
projectName = 'Example-02'
projectAuthor = 'Exneyder A. Montoya Araque'
projectDate = time.strftime("%d/%m/%y")

#-------------------------------------------------------------------------------#
### Previous  calculations ###
waterUnitWeight = [9.8, 'kN/m3']
materialDryUnitWeight = [13, 'kN/m3']
specificGravity = 2.4
moisture = 0.18
voidRatio = (waterUnitWeight[0]*specificGravity/materialDryUnitWeight[0])-1
materialUnitWeight = [specificGravity*(1+moisture)*waterUnitWeight[0]/\
        (1+voidRatio), 'kN/m3']
materialSatUnitWeight = [(specificGravity+voidRatio)*waterUnitWeight[0]/\
        (1+voidRatio), 'kN/m3']

### Define inputs ###
# The slope geometry #
slopeHeight = [11.5, 'm']
slopeDip = np.array([3, 8])
crownDist = [15, 'm']
toeDist = [15, 'm']
wantAutomaticToeDepth = True
toeDepth = ['automatic toe Depth']
# The slip arc-circle #
hztDistPointAtCrownFromCrown = [-11, 'm']
hztDistPointAtToeFromCrown = [13.5, 'm']
slipRadius = [15.6, 'm']
# Water table depth #
wantWatertable = True
wtDepthAtCrown = [3.7, 'm']
toeUnderWatertable = False
# Materials properties #
waterUnitWeight = waterUnitWeight[:]
materialUnitWeight = materialSatUnitWeight[:]
frictionAngleGrad = [21, 'degrees']
cohesion = [4.5, 'kPa']

### Advanced inputs ###
# Want divide the slip surface in constant width slices? #
wantConstSliceWidthTrue = True
# Number of discretizations of slip surface. #
numSlices = 10
# Number of discretizations of circular arcs. #
nDivs = numSlices
# Select the method to calcualte the safety factor ['Flns', 'Bshp' or 'Allm'] #
```

```
methodString = 'Bshp'
# Select the output format image ['.eps', '.jpeg', '.jpg', '.pdf', '.pgf', \ #
# '.png', '.ps', '.raw', '.rgba', '.svg', '.svgz', '.tif', '.tiff']. #
outputFormatImg = '.pdf'

#-----------------------------------------------------------------------------#
# Operations for only one slip surface #
msg = onlyonecircle(projectName, projectAuthor, projectDate, slopeHeight, \
        slopeDip, crownDist, toeDist, wantAutomaticToeDepth, toeDepth, \
        hztDistPointAtCrownFromCrown, hztDistPointAtToeFromCrown, \
        slipRadius, wantWatertable, wtDepthAtCrown, toeUnderWatertable, \
        waterUnitWeight, materialUnitWeight, frictionAngleGrad, cohesion, \
        wantConstSliceWidthTrue, numSlices, nDivs, methodString, \
        outputFormatImg)
#'''
BSD 2 license.

Copyright (c) 2016, Universidad Nacional de Colombia, Ludger O.
Suarez-Burgoa and Exneyder Andres Montoya Araque.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
'''#
```

4. Resultados: (ver Figura 7.4)

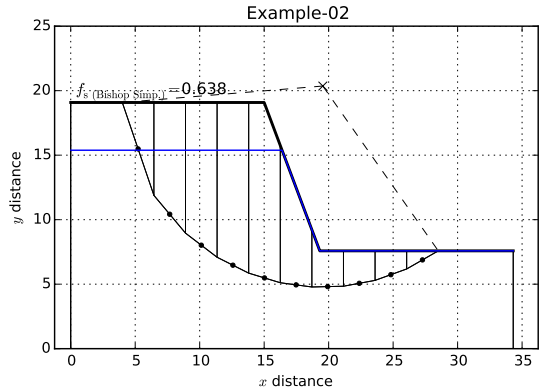


Figura 7.4 Resultado obtenido para el ejemplo 02: $f_{s-\text{Bishop Simp.}} = 0.638$.

Referencias

[1] BISHOP, A. W.: «The use of the slip circle in the stability analysis of slopes». *Géotechnique*, 1955, **5**, pp. 7–17.

[2] BISHOP, A. W. y MORGENSTERN, N.R.: «Stability coefficients for earth slopes». *Géotechnique*, 1960, **10**, pp. 129–150.

[3] CHANG, C. S.: «Discrete elements method for slope stability analysis». *Journal of Geothecnical Engineering*, 1992, **118(12)**, pp. 1889–1905.

[4] CHENG, Y. M. y LAU, C. K.: *Slope stability analysis and stabilization: new methods and insight*. Routledge, New York, 1^a edición, 2008.

[5] DAS, B. M.: *Principles of Geotechnical Engineering*. Cengage Learning, 7^a edición, 2009.

[6] DJ. ALZATE CASTAÑO, R.: *Estabilidad de taludes con aplicación en zonas húmedas tropicales*. Editorial Universidad Tecnológica del Chocó, Quibdó, 1^a edición, 2005.

[7] DUNCAN, J. M.; WRIGHT, S. G. y BRANDON, T. L.: *Soil Strength and Slope Stability*. Wiley, New Jersey, 2^a edición, 2014.

[8] FELLENIUS, W.: «Calculation of the stability of earth dams». En: ICOLD (Ed.), *Transactions of the 2nd Congress on Large Dams*, volumen 4, pp. 445–459. Washington D.C., 1936.

[9] GIBSON, R. E. y MORGENSTERN, N.: «A note of the stability of cuttings in normally consolidated clays». *Géotechnique*, 1962, **12(3)**, pp. 212–216.

[10] MORGENSTERN, N. R. y PRICE, V.E.: «A numerical method for solving the equations of stability of general slip surfaces». *Computer Journal*, 1967, **9(4)**, pp. 388–393.

[11] SPENCER, E.: «A method of analysis of the stability of emabankments assuming parallel inter-slice forces». *Geotéchnique*, 1967, **17(1)**, pp. 11–26.

[12] SUÁREZ-BURGOA, L. O.: *Análisis de estabilidad de taludes. Con aplicaciones en MATLAB®*. Medellín, 1^a edición, 2016.

[13] TURNBULL, W. J. y HVORSLEV, M. J.: «Special problems in slope stability». *Journal of the Soil Mechanics and Foundations Division*, 1967, **93**, pp. 499–528.

[14] U. S. Army Corps of Engineers: *Engineearing and Design. Slope Stability*. Department of the Army, 2003.

[15] Zhao, Y.; Tong, Z. Y. y Lü, Q.: «Slope stability analysis using slice-wise factor of safety». *Mathematical problems in engineering*, 2014, **2014**, p. 6.

Apéndice A
Descripción de las funciones

A.1. automaticslipcircles

```
   automaticslipcircles(projectName, projectAuthor, projectDate, slopeHeight, slopeDip,
 crownDist, toeDist, wantAutomaticToeDepth, toeDepth, numCircles, radiusIncrement, numberIn
, maxFsValueCont, wantWatertable, wtDepthAtCrown, toeUnderWatertable, waterUnitWeight,
materialUnitWeight, frictionAngleGrad, cohesion, wantConstSliceWidthTrue, numSlices, nDivs
, methodString, outputFormatImg)
```

A.1.1. Descripción

Llama a las demás funciones para evaluar la estabilidad del talud en múltiples superficies circulares generadas automáticamente, y de esa forma encontrar la que tenga el factor de seguridad más bajo (*i.e.*la superficie crítica).

También genera el gráfico del talud con el mapa de contornos que muestra la distribución de los valores de factor de seguridad encontrados, definiendo la localización de la superficie crítica, de manera separada para el Método de Fellenius y el Método de Bishop simplificado.

Finalmente arroja un archivo de texto con el resumen del análisis realizado.

A.1.2. Subfunciones externas

defineslipcircle: Ver su descripción en la sección A.5.
defineswatertable: Ver su descripción en la sección A.6.

divideslipintoslices: Ver su descripción en la sección A.7.
interatefbishopsimpsat: Ver su descripción en la sección A.9.
interateffelleniussat: Ver su descripción en la sección A.10.
materialboundary: Ver su descripción en la sección A.11.
obtainmaxdepthdist: Ver su descripción en la sección A.12.
plotslice: Ver su descripción en la sección A.14.
reportslicestructurevalues: Ver su descripción en la sección A.16.
sliparcdiscretization: Ver su descripción en la sección A.17.
terrainsurface: Ver su descripción en la sección A.19.
vertprojection2pline: Ver su descripción en la sección A.22.

A.1.3. Variables de entrada

projectName: Nombre o título del proyecto. Es de tipo *cadena de caracteres*.
projectAuthor: Nombre del autor del proyecto. Es de tipo *cadena de caracteres*.
projectDate: Fecha de realización del análisis. Es de tipo *cadena de caracteres*. Por defecto genera automáticamente la fecha tomándola de la máquina en la que se ejecuta
slopeHeight: Lista de dos elementos con la altura del talud de tipo *escalar* y la unidad de medida de tipo *cadena de caracteres*.
slopeDip: Pendiente del talud, expresada en un vector de $\mathbb{R}^2$, donde la primer componente representa una distancia horizontal, y la segunda una vertical, las cuales no son necesariamente las distancias reales del talud. Es de tipo *array*.
crownDist: Lista de dos elementos con la longitud de la corona del talud de tipo *escalar* y la unidad de medida de tipo *cadena de caracteres*.
toeDist: Lista de dos elementos con la longitud del pie del talud de tipo *escalar* y la unidad de medida de tipo *cadena de caracteres*.
wantAutomaticToeDepth: Variable de verificación sobre si se desea o no generar la profundidad del talud. Es de tipo *booleano*.
toeDepth: Lista de dos elementos con la profundidad del talud de tipo *escalar* y la unidad de medida de tipo *cadena de caracteres*.
numCircles: Variable que define la cantidad de superficies circulares que serán evaluadas. Es de tipo *escalar*.
radiusIncrement: Lista de dos elementos con la longitud que irá aumentando el radio de un circulo que pasa por un mismo par de putos aleatorios sobre la superficie del talud de tipo *escalar*, y la unidad de medida de tipo *cadena de caracteres*.
numberIncrements: Variable que define la cantidad de veces que será aumentando el radio inicial (con magnitud definida por la variable `radiusIncrement`) de los círculos que pasan por un mismo par de puntos aleatorios sobre la superficie del talud. Es de tipo *escalar*.

maxFsValueCont: Variable que limita el valor máximo de factor de seguridad que se mostrará en el mapa de contornos sobre la imagen de salida. Es de tipo *escalar*.

wantWatertable: Variable de verificación sobre si se desea o no generar una superficie de nivel freático. Es de tipo *booleano*.

wtDepthAtCrown: Lista de dos elementos con la profundidad del nivel freático medido desde la corona del talud de tipo *escalar* y la unidad de medida de tipo *cadena de caracteres*.

toeUnderWatertable: Variable de verificación sobre si se desea o no que el nivel freático esté por encima del pié del talud (*i.e.* talud parcialmente sumergido). Es de tipo *booleano*.

waterUnitWeight: Lista de dos elementos con el peso específico del agua de tipo *escalar* y la unidad de medida de tipo *cadena de caracteres*.

materialUnitWeight: Lista de dos elementos con el peso específico del material que compone el talud de tipo *escalar* y la unidad de medida de tipo *cadena de caracteres*.

frictionAngleGrad: Lista de dos elementos con el ángulo de fricción del material que compone el talud de tipo *escalar* y la unidad de medida de tipo *cadena de caracteres*.

cohesion: Lista de dos elementos con la cohesión del material que compone el talud de tipo *escalar* y la unidad de medida de tipo *cadena de caracteres*.

wantConstSliceWidthTrue: Variable de verificación sobre si se desea o no que el ancho de las dovelas sea el mismo para todas. Es de tipo *booleano*.

numSlices: Número de dovelas en las que se quiere dividir la masa de suelo. Es de tipo *escalar*.

nDivs: Número de divisiones en las que se dividirá el arco circular de deslizamiento. Se recomienda que sea igual al valor de la variable `numSlices`. Es de tipo *escalar*.

methodString: Abreviación del método que se usará para el análisis. Es de tipo *cadena de caracteres*, y se puede escoger entre `'Flns'` para el método de Fellenius, `'Bshp'` para el método de Bishop Simplificado o `'Allm'` para ambos.

outputFormatImg: Extensión de la imagen de salida. Es de tipo *cadena de caracteres*, y se puede escoger entre `'.eps'`, `'.jpeg'`, `'.jpg'`, `'.pdf'`, `'.pgf'`, `'.png'`, `'.raw'`, `'.rgba'`, `'.svg'`, `'.svgz'`, `'.tif'`, o `'.tiff'`.

A.1.4. Variables de salida

Esta función no genera variables de salida, pero si dos archivos de salida: un archivo de imagen con el esquema del problema analizado, y un archivo de texto con el resumen del análisis. Estos archivos se describen mejor en la sección 3.2.

A.1.5. Ejemplo 01

```
#-------------------------------------------------------------------- #
### Add functions directory ###
import sys
sys.path += ['../functions']

#-------------------------------------------------------------------- #
### Modules/Functions import ###
import numpy as np
import time
from onlyonecircle import onlyonecircle

#-------------------------------------------------------------------- #
### Poject data ###
projectName = 'Example-01'
projectAuthor = 'Exneyder A. Montoya Araque'
projectDate = time.strftime("%d/%m/%y")

#-------------------------------------------------------------------- #
### Define inputs ###
# previous values #
segmentAC = 40/np.sin(np.radians(30))
# The slope geometry #
slopeHeight = [40, 'ft']
slopeDip = np.array([1, np.tan(np.radians(45))])
crownDist = [30, 'ft']
toeDist = [30, 'ft']
wantAutomaticToeDepth = False
if wantAutomaticToeDepth == True:
        toeDepth = ['automatic toe Depth']
else:
        toeDepth = [10, 'ft']
# The slip arc-circle #
numCircles = 1000
radiusIncrement = [2, 'm']
numberIncrements = 20
maxFsValueCont = 1.5
# Water table depth #
wantWatertable = False
if wantWatertable == True:
        wtDepthAtCrown = [0, 'm']
else:
        wtDepthAtCrown = ['No watertable']
toeUnderWatertable = False
# Materials properties. #
waterUnitWeight = [62.4, 'pcf']
materialUnitWeight = [115, 'pcf']
frictionAngleGrad = [20, 'degrees']
cohesion = [400, 'psf']

### Advanced inputs ###
# Want divide the slip surface in constant width slices? #
wantConstSliceWidthTrue = True
# Number of discretizations of slip surface. #
numSlices = 15
# Number of discretizations of circular arcs. #
nDivs = numSlices
# Select the method to calcualte the safety factor#
methodString = 'Allm'
# Select the output format image #
```

```
outputFormatImg = '.pdf'

#---------------------------------------------------------------------- #
# Operations for only one slip surface #
automaticslipcircles(projectName, projectAuthor, projectDate, slopeHeight,\
        slopeDip, crownDist, toeDist, wantAutomaticToeDepth, toeDepth, \
        numCircles, radiusIncrement, numberIncrements, maxFsValueCont, \
        wantWatertable, wtDepthAtCrown, toeUnderWatertable, waterUnitWeight, \
        materialUnitWeight, frictionAngleGrad, cohesion, \
        wantConstSliceWidthTrue, numSlices, nDivs, methodString, \
        outputFormatImg)
```

A.1.6. Ejemplo 02

```
#---------------------------------------------------------------------- #
### Add functions directory ###
import sys
sys.path += ['../functions']

#---------------------------------------------------------------------- #
### Modules/Functions import ###
import numpy as np
import time
from onlyonecircle import onlyonecircle

#---------------------------------------------------------------------- #
### Poject data ###
projectName = 'Example-02'
projectAuthor = 'Exneyder A. Montoya Araque'
projectDate = time.strftime("%d/%m/%y")

#---------------------------------------------------------------------- #
### Define inputs ###
# previous values #
segmentAC = 40/np.sin(np.radians(30))
# The slope geometry #
slopeHeight = [11.5, 'm']
slopeDip = np.array([3, 8])
crownDist = [15, 'm']
toeDist = [15, 'm']
wantAutomaticToeDepth = True
if wantAutomaticToeDepth == True:
        toeDepth = ['automatic toe Depth']
else:
        toeDepth = [10, 'ft']
# The slip arc-circle #
numCircles = 100
radiusIncrement = [7, 'm']
numberIncrements = 10
maxFsValueCont = 1
# Water table depth #
wantWatertable = True
if wantWatertable == True:
        wtDepthAtCrown = [3.7, 'm']
else:
        wtDepthAtCrown = ['No watertable']
toeUnderWatertable = False
```

```
# Materials properties. #
waterUnitWeight = [9.8, 'kN/m3']
materialUnitWeight = [18, 'kN/m3']
frictionAngleGrad = [21, 'degrees']
cohesion = [4.5, 'kPa']

### Advanced inputs ###
# Want divide the slip surface in constant width slices? #
wantConstSliceWidthTrue = False
# Number of discretizations of slip surface. #
numSlices = 10
# Number of discretizations of circular arcs. #
nDivs = numSlices
# Select the method to calcualte the safety factor#
methodString = 'Allm'
# Select the output format image #
outputFormatImg = '.pdf'

#------------------------------------------------------------------- #
# Operations for only one slip surface #
automaticslipcircles(projectName, projectAuthor, projectDate, slopeHeight,\
        slopeDip, crownDist, toeDist, wantAutomaticToeDepth, toeDepth, \
        numCircles, radiusIncrement, numberIncrements, maxFsValueCont, \
        wantWatertable, wtDepthAtCrown, toeUnderWatertable, waterUnitWeight, \
        materialUnitWeight, frictionAngleGrad, cohesion, \
        wantConstSliceWidthTrue, numSlices, nDivs, methodString, \
        outputFormatImg)
```

A.2. azimuthangle

```
angleRad = azimuthangle(vector)
```

A.2.1. Descripción

Dado un vector en $\mathbb{R}^2$, esta función obtiene el ángulo del azimut medido desde el lado positivo del eje *x* en el sentido contrario al de las manecillas del reloj.

El sistema coordenado es tal que el eje *x* está en las abscisas, y el eje *y* en las ordenadas, es decir, el eje *x* apunta hacia la derecha, y el eje *y* hacia arriba.

A.2.2. Variables de entrada

vector: Vector bidimensional. Es de tipo *array*.

A.2.3. Variables de salida

angleRad: Azimut del vector introducido según el eje de referencia explicado, dado en radianes. Es de tipo *escalar*

A.2.4. Ejemplo 01

```
vector = np.array([3,-5])
print(azimuthangle(vector))
```

A.2.5. Ejemplo 02

```
vector = np.array([-10,-2])
print(azimuthangle(vector))
```

A.3. circleby2ptsradius

```
centerVec1, centerVec2 = circleby2ptsradius(pt1Vec, pt2Vec, radius)
```

A.3.1. Descripción

Calcula los 2 posibles centros de un circulo, dados dos puntos y el radio.

A.3.2. Variables de entrada

pt1Vec: Primer punto por el que pasa el circulo. Es de tipo *array*.
pt2Vec: Segundo punto por el que pasa el circulo. Es de tipo *array*.
radius: Valor del radio del círculo. Es de tipo *escalar*.

A.3.3. Variables de salida

centerVec1: Vector del punto del primer posible centro del circulo. Es de tipo *array*.
centerVec2: Vector del punto del primer posible centro del circulo.. Es de tipo *array*.

A.3.4. Ejemplo 01

```
pt1Vec = np.array([40, 12])
pt2Vec = np.array([4.347, 24])
radius = 34.4848
print(circleby2ptsradius(pt1Vec, pt2Vec, radius))
```

A.3.5. Ejemplo 02

```
pt1Vec = np.array([-5, 0])
pt2Vec = np.array([5, 0])
radius = 5
print(circleby2ptsradius(pt1Vec, pt2Vec, radius))
```

A.4. create2dsegmentstructure

```
segmentSTR = create2dsegmentstructure(iniPnt2dRowVec, endPnt2dRowVec)
```

A.4.1. Descripción

Crea una estructura de datos de un segmento de linea bidimensional a partir de su punto inicial y final.

A.4.2. Subfunciones externas

azimuthangle: Ver su descripción en la sección A.2.
unitvector: Ver su descripción en la sección A.21.

A.4.3. Variables de entrada

iniPnt2dRowVec: Vector bidimensional que representa el punto inicial del segmento. Es de tipo *array*.

endPnt2dRowVec: Vector bidimensional que representa el punto final del segmento. Es de tipo *array*.

A.4.4. Variables de salida

segmentSTR: Estructura de datos de tipo *diccionario*, que contiene los siguientes campos:

iniPtVec Vector de tipo *array* con las coordenadas del primer punto de la linea.

endPtVec Vector de tipo *array* con las coordenadas del último punto de la linea.

unitVec Vector unitario de tipo *array* que define una dirección.

lambda Valor de tipo *escalar* que define la longitud del segmento.

slope Valor de tipo *escalar* que define la pendiente de la ecuación de la linea del segmento.

azimuthRad Valor de tipo *escalar* que define el ángulo (en radianes) del segmento, medido en contra de las manecillas del reloj desde el eje de referencia $[1,0]$.

intercept Valor de tipo *escalar* que define el intercepto de la ecuación de la linea del segmento.

A.4.5. Ejemplo 01

```
iniPnt2dRowVec = np.array([53.8973, 43.2314])
endPnt2dRowVec = np.array([69.0489, 50.5464])
print(create2dsegmentstructure(iniPnt2dRowVec, endPnt2dRowVec))
```

A.4.6. Ejemplo 02

```
iniPnt2dRowVec = np.zeros(2)
endPnt2dRowVec = np.array([5, 10])
print(create2dsegmentstructure(iniPnt2dRowVec, endPnt2dRowVec))
```

A.5. defineslipcircle

```
slipArcSTR = defineslipcircle(pointAtToeVec, pointAtCrownVec, slipRadius)
```

A.5.1. Descripción

Define la estructura del arco de circunferencia de deslizamiento (incluye los valores del punto central del arco, radio y ángulo inicial y final), dados dos puntos extremos y el radio del círculo.

A.5.2. Subfunciones externas

azimuthangle: Ver su descripción en la sección A.2
circleby2ptsradius: Ver su descripción en la sección A.3
unitvector: Ver su descripción en la sección A.21

A.5.3. Variables de entrada

pointAtToeVec: Vector bidimensional con las coordenadas que define el deslizamiento en el pie del talud. Es de tipo *array*.
pointAtCrownVec: Vector bidimensional con las coordenadas que define el deslizamiento en la corona del talud. Es de tipo *array*.
slipRadius: Valor del radio del arco. Es de tipo *escalar*.

A.5.4. Variables de salida

existSlipCircleTrue: Variable lógica dada como True— si es posible un arco dados las variables de entrada y con el sentido de una superficie de deslizamiento. Es de tipo *booleano*.
slipArcSTR: Estructura de datos de tipo *diccionario* del arco de circunferencia del deslizamiento; contiene los siguientes campos:

center Centro del arco del deslizamiento.
radius Radio del arco del deslizamiento.

iniAngGrad Ángulo sexagesimal medido en contra de las manecillas del reloj desde el vector unitario de referencia [1, 0] hasta el radio inicial que define el arco.

endAngGrad Ángulo sexagesimal medido en contra de las manecillas del reloj desde el vector unitario de referencia [1, 0] hasta el radio final que define el arco.

deepDist Distancia más profunda por donde pasa el arco.

leftDist Distancia más extrema hacia la izquierda por donde pasa el arco.

A.5.5. Ejemplo 01

```
pointAtToeVec = np.array([40, 12])
pointAtCrownVec = np.array([4.347, 24])
slipRadius = 34.4848
print(defineslipcircle(pointAtToeVec, pointAtCrownVec, slipRadius))
```

A.5.6. Ejemplo 02

```
pointAtToeVec = np.array([-5, 5])
pointAtCrownVec = np.array([5, 0])
slipRadius = 15
print(defineslipcircle(pointAtToeVec, pointAtCrownVec, slipRadius))
```

A.6. defineswatertable

```
   wtCoordsArray = defineswatertable(wtDepthAtCrown, surfaceDataCell, toeUnderWatertable
 = False, want2plot = False)
```

A.6.1. Descripción

Genera un nivel freático horizontal que puede ser: coincidente con la superficie del talud, estar debajo de la superficie del talud, o estar encima del pie del talud, únicamente para el talud estándar.

A.6.2. Subfunciones externas

create2dsegmentstructure: Ver su descripción en la sección A.4

A.6.3. Variables de entrada

wtDepthAtCrown: Distancia desde la superficie horizontal de la corona del talud hasta el nivel freático. Es de tipo *escalar*

surfaceDataCell: Lista que almacena la estructura de datos para cada linea que conforma el polígono abierto de la superficie del talud. Se obtiene al ejecutar previamente la función `terrainsurface`. Es de tipo *lista*. Cada elemento de la lista es de tipo *diccionario* y contiene los siguientes campos:

- **iniPtVec** Vector de tipo *array* con las coordenadas del primer punto de la linea.
- **endPtVec** Vector de tipo *array* con las coordenadas del último punto de la linea.
- **unitVec** Vector unitario de tipo *array* que define una dirección.
- **lambda** Valor de tipo *escalar* que define la longitud del segmento.
- **slope** Valor de tipo *escalar* que define la pendiente de la ecuación de la linea del segmento.
- **azimuthRad** Valor de tipo *escalar* que define el ángulo (en radianes) del segmento, medido en contra de las manecillas del reloj desde el eje de referencia $[1,0]$ hasta el $[0,1]$.
- **intercept** Valor de tipo *escalar* que define el intercepto de la ecuación de la linea del segmento.

toeUnderWatertable Corresponde a una variable de verificación sobre si desea o no considerar el talud parcialmente sumergido, es decir que el pie del talud esté bajo el nivel freático. Es de tipo *booleano*, su valor por defecto es False

want2plot: Corresponde a una variable de verificación sobre si desea o no graficar el nivel freático. Es de tipo *booleano*, su valor por defecto es False.

A.6.4. Variables de salida

watertableDataCell: Lista similar a surfaceDataCell, donde se almacena la información del polígono abierto que representa el nivel freático. Cada elemento de la lista corresponde a una estructura de tipo *diccionario*, donde se almacenan los mismos campos que los introducidos en la variable surfaceDataCell.

A.6.5. Ejemplo 01

```
# inputs #
slopeHeight = 12; slopeDip = np.array([1, 2.5]); crownDist = 10.0;
toeDist = 10.0; fromToeOriginRowVec = np.array([-14.8, -3.30]); wtDepthAtCrown = 0
# Previous functions #
surfaceDataCell, surfaceChordsArray = terrainsurface(fromToeOriginRowVec, slopeHeight,
      slopeDip, crownDist, toeDist)
# This function #
print(defineswatertable(wtDepthAtCrown, surfaceDataCell, False, True))
```

A.6.6. Ejemplo 02

```
# inputs #
slopeHeight = 12; slopeDip = np.array([1, 2.5]); crownDist = 10.0;
toeDist = 10.0; fromToeOriginRowVec = np.array([-14.8, -3.30]); wtDepthAtCrown = 0
# Previous functions #
surfaceDataCell, surfaceChordsArray = terrainsurface(fromToeOriginRowVec, slopeHeight,
      slopeDip, crownDist, toeDist)
# This function #
print(defineswatertable(wtDepthAtCrown, surfaceDataCell))
```

A.7. divideslipintoslices

```
   slicesSTRCell = divideslipintoslices(slipArcSTR, surfaceDataCell, watertableDataCell
, numSlices, pointAtToeVec, pointAtCrownVec, wantConstSliceWidthTrue = False)
```

A.7.1. Descripción

Divide el arco circular de deslizamiento en el número de dovelas requeridas de acuerdo a la configuración de la superficie del terreno, y crea la estructura de datos de cada dovela, con la información necesaria para realizar futuros anánlisis de deslizamiento por métodos de equilibrio límite.

A.7.2. Subfunciones externas

extractplinefrom2pts: Ver su descripción en la sección A.8.

polyarea: Ver su descripción en la sección A.15.
sliparcdiscretization: Ver su descripción en la sección A.17.
tangentlineatcirclept: Ver su descripción en la sección A.18.
uniquewithtolerance: Ver su descripción en la sección A.20.
vertprojection2pline: Ver su descripción en la sección A.22.

A.7.3. Variables de entrada

slipArcSTR: Estructura de datos de tipo *diccionario* del arco de circunferencia del deslizamiento; contiene los campos obtenidos con la función `defineslipcircle` (A.5).

surfaceDataCell: Lista que almacena la estructura de datos para cada linea que conforma el polígono abierto de la superficie del talud. Cada elemento de la lista es de tipo *diccionario* y contiene los campos obtenidos con la función `terrainsurface` (A.19).

watertableDataCell Lista similar a surfaceDataCell, donde se almacena la información del polígono abierto que representa el nivel freático. Cada elemento de la lista corresponde a una estructura de tipo *diccionario*, donde se almacenan los mismos campos obtenidos con la función `defineswatertable` (A.6).

numSlices: Numero de dovelas en las que se quiere dividir el arco circular. Es de tipo *escalar*.

pointAtToeVec: Vector bidimensional con las coordenadas que define el deslizamiento en el pie del talud. Es de tipo *array*.

pointAtCrownVec: Vector bidimensional con las coordenadas que define el deslizamiento en la corona del talud. Es de tipo *array*.

wantConstSliceWidthTrue: Variable lógica de verificación sobre si se quiere que las dovelas tengan un ancho constante. Su valor predeterminado es False, y es de tipo *booleano*.

A.7.4. Variables de salida

slicesSTRCell: Lista que almacena todas las estructuras de datos de tipo *diccionario* de las dovelas. Cada estructura contiene los siguientes campos:

- **plineCords:** Matriz $n \times 2$ con las coordenadas que definen la dovela como un polígono cerrado. Es de tipo *array*.
- **area:** Valor del área del polígono cerrado que define la dovela. Es de tipo *escalar*.
- **midPoint:** Vector bidimensional con las coordenadas del punto medio de la dovela en su base. Es de tipo *array*.

midHeight: Valor de la altura media de la dovela, tomando los puntos extremos y su punto medio en la base hasta la superficie del terreno. Es de tipo *escalar*.

width: Valor del ancho de la dovela.

inclinationAngleGradAtBottom: Ángulo en grados sexagesimales de la secante que pasa por los puntos extremos de la base de la dovela. Es de tipo *escalar*.

inclinationAngleGradAtTop: Ángulo en grados sexagesimales de la secante que pasa por los puntos extremos del tope de la dovela. Es de tipo *escalar*.

wtMidHeight: Valor de la altura media del nivel freático, tomando los puntos extremos y su punto medio en la base hasta el nivel freático. Es de tipo *escalar*.

wtMidHeightAboveSlope: Valor de la altura media la columna de agua que está por encima de la superficie del talud, tomando los

hrzMomentArm: Valor de la componente horizontal del brazo del momento que actúa sobre la superficie del talud debido a la columna de agua por encima de esta.

vrtMomentArm: Valor de la componente vertical del brazo del momento que actúa sobre la superficie del talud debido a la columna de agua por encima de esta.

A.7.5. Ejemplo 01

```
# inputs: #
slopeHeight = 12.0; slopeDip = np.array([1, 2.5]); crownDist = 10.0
toeDist = 10.0; wtDepthAtCrown = 0; numSlices = 10; slipRadius = 15
pointAtToeVec = np.array([23, 3.3]); pointAtCrownVec = np.array([2, 15.3])
# Previous functions #
boundPointsCordsArray, fromToeOriginRowVec, coordTransMat = materialboundary(
     slopeHeight, slopeDip, crownDist, toeDist)
surfaceDataCell, surfaceChordsArray = terrainsurface(fromToeOriginRowVec, slopeHeight,
      slopeDip, crownDist, toeDist)
watertableDataCell, wtCoordsArray = defineswatertable(wtDepthAtCrown, surfaceDataCell)
existSlipCircleTrue, slipArcSTR = defineslipcircle(pointAtToeVec, pointAtCrownVec,
     slipRadius)
# This function #
print(divideslipintoslices(slipArcSTR, surfaceDataCell, watertableDataCell, numSlices,
      pointAtToeVec, pointAtCrownVec))
```

A.7.6. Ejemplo 02

```
# inputs: #
slopeHeight = 15.0; slopeDip = np.array([1, 1]); crownDist = 10.0
toeDist = 10.0; wtDepthAtCrown = 0; numSlices = 10; slipRadius = 25
pointAtToeVec = np.array([30, 5]); pointAtCrownVec = np.array([1, 20])
# Previous functions #
boundPointsCordsArray, fromToeOriginRowVec, coordTransMat = materialboundary(
     slopeHeight, slopeDip, crownDist, toeDist)
```

```
surfaceDataCell, surfaceChordsArray = terrainsurface(fromToeOriginRowVec, slopeHeight,
      slopeDip, crownDist, toeDist)
watertableDataCell, wtCoordsArray = defineswatertable(wtDepthAtCrown, surfaceDataCell)
existSlipCircleTrue, slipArcSTR = defineslipcircle(pointAtToeVec, pointAtCrownVec,
     slipRadius)
# This function #
print(divideslipintoslices(slipArcSTR, surfaceDataCell, watertableDataCell, numSlices,
      pointAtToeVec, pointAtCrownVec, True))
```

A.8. extractplinefrom2pts

```
extractplinefrom2pts(pointOneVec, pointTwoVec, plineStructureCell)
```

A.8.1. Descripción

Obtiene las coordenadas de los puntos que definen una polilínea *A*, extraída a partir de otra, y dos puntos extremos que corresponden al inicio y final de *A*.

A.8.2. Variables de entrada

pointOneVec: Primer punto dado por un vector bidimensional. Es de tipo *array*.

pointTwoVec: Segundo punto dado por un vector bidimensional. Es de tipo *array*.

surfaceDataCell: Lista que almacena la estructura de datos para cada linea que conforma la polilínea sobre la que se va a extraer la nueva polilínea. Cada elemento de la lista es de tipo *diccionario* y contiene los mismos campos que las estructuras obtenidas con la función `create2dsegmentstructure`. Es obtenida previamente con función `terrainsurface` (A.19).

A.8.3. Variables de salida

plineChordsArray: Matriz $n \times 2$ que contiene las coordenadas que pertenecen a la nueva polilínea ordenadas ascendentemente. Es de tipo *array*.

A.8.4. Ejemplo 01

```
pointOneVec = np.array([4.3470, 24])
pointTwoVec = np.array([12.0085, 23.1966])
plineStructureCell = [
{'iniPtVec':np.array([0, 24]), 'endPtVec':np.array([10, 24]), 'unitVec':\
np.array([1, 0]), 'lambda':10, 'slope':0, 'azimuthRad':0, 'intercept':24},
{'iniPtVec':np.array([10, 24]), 'endPtVec':np.array([40, 12]), 'unitVec':\
np.array([0.9285, -0.3714]), 'lambda':32.3110, 'slope':-0.4228,
'azimuthRad':5.9027, 'intercept':34.1470},
{'iniPtVec':np.array([40, 12]), 'endPtVec':np.array([50, 12]), 'unitVec':\
np.array([1, 0]), 'lambda':10, 'slope':0, 'azimuthRad':0, 'intercept':12}]
print(extractplinefrom2pts(pointOneVec, pointTwoVec, plineStructureCell))
```

A.8.5. Ejemplo 02

```
# inputs #
slopeHeight = 12; slopeDip = np.array([1, 2.5]); crownDist = 10.0; toeDist = 10.0;
     fromToeOriginRowVec = np.array([-14.8, -3.30]); pointOneVec = np.array([-25, 8]);
      pointTwoVec = np.array([-15, 5])
# previous functions #
surfaceDataCell, surfaceChordsArray = terrainsurface(fromToeOriginRowVec, slopeHeight,
      slopeDip, crownDist, toeDist)
# this function #
print(extractplinefrom2pts(pointOneVec, pointTwoVec, surfaceDataCell))
```

A.9. interatefbishopsimpsat

```
   sf = interatefbishopsimpsat(slicesSTRCell, waterUnitWeight, dryMaterialUnitWeight, fricti
, cohesion, slipRadius,seedSafetyFactor = 1.5, iterations = 5)
```

A.9.1. Descripción

Obtiene el factor de seguridad al deslizamiento por falla circular bajo un análisis de equilibrio límite con el método de Bishop simplificado [1], para cualquier nivel freático coincidente, bajo el terreno del talud, o con el talud parcialmente sumergido.

A.9.2. Subfunciones externas

reportslicestructurevalues: Ver su descripción en la sección A.16.

A.9.3. Variables de entrada

slicesSTRCell: Lista que almacena todas las estructuras de datos de tipo *diccionario* de las dovelas. Cada estructura contiene los campos obtenidos con la función `divideslipintoslices` (A.7).
waterUnitWeight: Peso unitario del agua. Es de tipo *escalar*.
dryMaterialUnitWeight: Peso unitario del geomaterial en estado seco. Es de tipo *escalar*.
frictionAngleGrad: Ángulo de fricción del geomaterial en la base de la dovela. Es de tipo *escalar*.
cohesion: Cohesión del geomaterial en la base de la dovela. Es de tipo *escalar*.
slipRadius Radio del circulo de deslizamiento. Es de tipo *escalar*.
seedSafetyFactor: Valor semilla del factor de seguridad para dar inicio a la iteración. Es de tipo *escalar*. Su valor por defecto es 1.5.
iterations: Numero de iteraciones a realizar. Es de tipo *escalar*. Su valor por defecto es 5.

A.9.4. Variables de salida

sf: Valor del factor de seguridad al deslizamiento por falla circular. Es de tipo *escalar*.

A.9.5. Ejemplo 01

```
# inputs: #
slopeHeight = 12.0; slopeDip = np.array([1, 2.5]); crownDist = 10.0;
toeDist = 10.0; wtDepthAtCrown = 10; numSlices = 10; nDivs = numSlices; pointAtToeVec
     = np.array([23, 3]); pointAtCrownVec = np.array([2, 15]); slipRadius = 14;
     waterUnitWeight = 9.81; dryMaterialUnitWeight = 19.5; frictionAngleGrad = 23;
     cohesion = 18; wantConstSliceWidthTrue = False;
# Previous functions #
boundPointsCordsArray, fromToeOriginRowVec, coordTransMat = materialboundary(
     slopeHeight, slopeDip, crownDist, toeDist)
surfaceDataCell, surfaceChordsArray = terrainsurface(fromToeOriginRowVec, slopeHeight,
      slopeDip, crownDist, toeDist)
watertableDataCell, wtCoordsArray = defineswatertable(wtDepthAtCrown, surfaceDataCell)
```

```
existSlipCircleTrue, slipArcSTR = defineslipcircle(pointAtToeVec, pointAtCrownVec,
     slipRadius)
slicesSTRCell = divideslipintoslices(slipArcSTR, surfaceDataCell, watertableDataCell,
     numSlices, pointAtToeVec, pointAtCrownVec, wantConstSliceWidthTrue)
# This function #
print(interatefbishopsimpsat(slicesSTRCell, waterUnitWeight, \
dryMaterialUnitWeight, frictionAngleGrad, cohesion, slipRadius))
```

A.9.6. Ejemplo 02

```
# inputs: #
slopeHeight = 10.0; slopeDip = np.array([1, 1]); crownDist = 10.0;
toeDist = 10.0; wtDepthAtCrown = 5; numSlices = 10; nDivs = numSlices; pointAtToeVec =
      np.array([25, 7]); pointAtCrownVec = np.array([5, 17]); slipRadius = 15;
     waterUnitWeight = 9.81; dryMaterialUnitWeight = 18; frictionAngleGrad = 25;
     cohesion = 20; wantConstSliceWidthTrue = True;
# Previous functions #
boundPointsCordsArray, fromToeOriginRowVec, coordTransMat = materialboundary(
     slopeHeight, slopeDip, crownDist, toeDist)
surfaceDataCell, surfaceChordsArray = terrainsurface(fromToeOriginRowVec, slopeHeight,
      slopeDip, crownDist, toeDist)
watertableDataCell, wtCoordsArray = defineswatertable(wtDepthAtCrown, surfaceDataCell)
existSlipCircleTrue, slipArcSTR = defineslipcircle(pointAtToeVec, pointAtCrownVec,
     slipRadius)
slicesSTRCell = divideslipintoslices(slipArcSTR, surfaceDataCell, watertableDataCell,
     numSlices, pointAtToeVec, pointAtCrownVec, wantConstSliceWidthTrue)
# This function #
print(interatefbishopsimpsat(slicesSTRCell, waterUnitWeight, \
dryMaterialUnitWeight, frictionAngleGrad, cohesion, slipRadius))
```

A.10. interateffelleniussat

```
   sf = interateffelleniussat(slicesSTRCell, waterUnitWeight, dryMaterialUnitWeight, frictio
, cohesion, slipRadius)
```

A.10.1. Descripción

Obtiene el factor de seguridad al deslizamiento por falla circular bajo un análisis de equilibrio límite con el *método de Fellenius* [8] (también llamado de *Método ordinario de las dovelas* o simplemente *método de las dovelas*) modificado por [13], para cualquier nivel freático coincidente, bajo el terreno del talud, o con el talud parcialmente sumergido.

A.10.2. Subfunciones externas

reportslicestructurevalues: Ver su descripción en la sección A.16.

A.10.3. Variables de entrada

slicesSTRCell: Lista que almacena todas las estructuras de datos de tipo *diccionario* de las dovelas. Cada estructura contiene los campos obtenidos con la función divideslipintoslices—(A.7).
waterUnitWeight: Peso unitario del agua. Es de tipo *escalar*.
dryMaterialUnitWeight: Peso unitario del geomaterial en estado seco. Es de tipo *escalar*.
frictionAngleGrad: Ángulo de fricción del geomaterial en la base de la dovela. Es de tipo *escalar*.
cohesion: Cohesión del geomaterial en la base de la dovela. Es de tipo *escalar*.
slipRadius Radio del circulo de deslizamiento. Es de tipo *escalar*.

A.10.4. Variables de salida

sf: Valor del factor de seguridad. Es de tipo *escalar*.

A.10.5. Ejemplo 01

```
# inputs: #
slopeHeight = 12.0; slopeDip = np.array([1, 2.5]); crownDist = 10.0;
toeDist = 10.0; wtDepthAtCrown = 10; numSlices = 10; nDivs = numSlices; pointAtToeVec
     = np.array([23, 3]); pointAtCrownVec = np.array([2, 15]); slipRadius = 14;
     waterUnitWeight = 9.81; dryMaterialUnitWeight = 19.5; frictionAngleGrad = 23;
     cohesion = 18; wantConstSliceWidthTrue = False;
# Previous functions #
boundPointsCordsArray, fromToeOriginRowVec, coordTransMat = materialboundary(
     slopeHeight, slopeDip, crownDist, toeDist)
surfaceDataCell, surfaceChordsArray = terrainsurface(fromToeOriginRowVec, slopeHeight,
      slopeDip, crownDist, toeDist)
watertableDataCell, wtCoordsArray = defineswatertable(wtDepthAtCrown, surfaceDataCell)
existSlipCircleTrue, slipArcSTR = defineslipcircle(pointAtToeVec, pointAtCrownVec,
     slipRadius)
slicesSTRCell = divideslipintoslices(slipArcSTR, surfaceDataCell, watertableDataCell,
     numSlices, pointAtToeVec, pointAtCrownVec, wantConstSliceWidthTrue)
# This function #
print(interateffelleniussat(slicesSTRCell, waterUnitWeight, \
```

```
dryMaterialUnitWeight, frictionAngleGrad, cohesion, slipRadius))
```

A.10.6. Ejemplo 02

```
# inputs: #
slopeHeight = 10.0; slopeDip = np.array([1, 1]); crownDist = 10.0;
toeDist = 10.0; wtDepthAtCrown = 5; numSlices = 10; nDivs = numSlices; pointAtToeVec =
      np.array([25, 7]); pointAtCrownVec = np.array([5, 17]); slipRadius = 15;
     waterUnitWeight = 9.81; dryMaterialUnitWeight = 18; frictionAngleGrad = 25;
     cohesion = 20; wantConstSliceWidthTrue = True;
# Previous functions #
boundPointsCordsArray, fromToeOriginRowVec, coordTransMat = materialboundary(
     slopeHeight, slopeDip, crownDist, toeDist)
surfaceDataCell, surfaceChordsArray = terrainsurface(fromToeOriginRowVec, slopeHeight,
      slopeDip, crownDist, toeDist)
watertableDataCell, wtCoordsArray = defineswatertable(wtDepthAtCrown, surfaceDataCell)
existSlipCircleTrue, slipArcSTR = defineslipcircle(pointAtToeVec, pointAtCrownVec,
     slipRadius)
slicesSTRCell = divideslipintoslices(slipArcSTR, surfaceDataCell, watertableDataCell,
     numSlices, pointAtToeVec, pointAtCrownVec, wantConstSliceWidthTrue)
# This function #
print(interateffelleniussat(slicesSTRCell, waterUnitWeight, \
dryMaterialUnitWeight, frictionAngleGrad, cohesion, slipRadius))
```

A.11. materialboundary

```
   boundPointsCordsArray, fromToeOrginRowVec, coordTransMat = materialboundary(slopeHeight
, slopeDip, crownDist, toeDist, want2plot = False)
```

A.11.1. Descripción

Define y grafica los límites del material donde tomará lugar el análisis de estabilidad del talud. El análisis solo se llevará a cabo en taludes que miran hacia la derecha (*i.e.* el deslizamiento ocurre en sentido contrario a las manecillas del reloj.)

Por otro lado, esta función solo crea un talud estándar, el cual corresponde a una superficie horizontal detrás de la corona y después del pie del talud. De esta forma, el límite queda definido por la altura e inclinación del talud, junto a las dos superficies horizontales.

A.11.2. Subfunciones externas

obtainmaxdepthdist: Ver su descripción en la sección A.12.

A.11.3. Variables de entrada

slopeHeight: Altura del talud. Es de tipo *escalar*.
slopeDip: Pendiente del talud, expresada en un vector de $\mathbb{R}^2$, donde la primer componente representa una distancia horizontal, y la segunda una vertical, las cuales no son necesariamente las distancias reales del talud. Es de tipo *array*.
crownDist: Distancia del plano horizontal en la cabeza del talud. Es de tipo *escalar*.
toeDist: Distancia horizontal del plano en el pie del talud. Es de tipo *escalar*.
want2plot: Corresponde a una variable de verificación sobre si desea o no graficar los límites del material. Es de tipo *booleano*, su valor por defecto es False—

A.11.4. Variables de salida

boundPointsCordsArray: Coordenadas del contorno del material dadas en un arreglo $n \times 2$ de tipo *array*.
fromToeOrginRowVec: Vector de desplazamiento que especifica el problema del origen del sistema de coordenadas relativo al pie del talud. Es de tipo *array*.
coordTransMat: Matriz de transformación del sistema coordenado, de tamaño 2×2. Es de tipo *array*.

A.11.5. Ejemplo 01

```
slopeHeight = 12
slopeDip = np.array([2.5, 1])
crownDist = 10.0
toeDist = 10.0
print(materialboundary(slopeHeight, slopeDip, crownDist, toeDist))
```

A.11.6. Ejemplo 02

```
slopeHeight = np.random.randint(1, 20)
slopeDip = np.array([1.5, 1])
crownDist = np.random.randint(1, 10)
toeDist = np.random.randint(1, 10)
print(materialboundary(slopeHeight, slopeDip, crownDist, toeDist))
```

A.12. obtainmaxdepthdist

```
toeDepth = obtainmaxdepthdist(slopeHeight, slopeDip, crownDist, toeDist)
```

A.12.1. Descripción

Dados los puntos extremos de un talud definido, esta función obtiene el límite inferior posible en el cual puede estar una superficie de falla circular.

A.12.2. Variables de entrada

slopeHeight: Altura del talud. Es de tipo *escalar*.
slopeDip: Pendiente del talud, expresada en un vector de $\mathbb{R}^2$, donde la primer componente representa una distancia horizontal, y la segunda una vertical, las cuales no son necesariamente las distancias reales del talud. Es de tipo *array*.
crownDist: Distancia del plano horizontal en la cabeza del talud. Es de tipo *escalar*.
toeDist: Distancia horizontal del plano en el pie del talud. Es de tipo *escalar*.

A.12.3. Variables de salida

toeDepth: Distancia vertical desde el pie del talud hacia abajo. Es de tipo *escalar*.

A.12.4. Ejemplo 01

```
slopeHeight = 12
slopeDip = np.array([2.5, 1])
crownDist = 10.0
```

```
toeDist = 10.0
print(obtainmaxdepthdist(slopeHeight, slopeDip, crownDist, toeDist))
```

A.12.5. Ejemplo 02

```
slopeHeight = np.random.randint(1,20)
slopeDip = np.array([1.5, 1])
crownDist = np.random.randint(1,10)
toeDist = np.random.randint(1,10)
print(obtainmaxdepthdist(slopeHeight, slopeDip, crownDist, toeDist ))
```

A.13. onlyonecircle

```
   msg = onlyonecircle(projectName, projectAuthor, projectDate, slopeHeight, slopeDip,
crownDist, toeDist, wantAutomaticToeDepth, toeDepth, hztDistPointAtCrownFromCrown, hztDistPointAtT
, slipRadius, wantWatertable, wtDepthAtCrown, toeUnderWatertable, waterUnitWeight, materialUnitWei
, frictionAngleGrad, cohesion, wantConstSliceWidthTrue, numSlices, nDivs, methodString,
 outputFormatImg)
```

A.13.1. Descripción

Llama a las demás funciones para evaluar la estabilidad del talud en una sola superficie circular dada.

También genera el gráfico del talud con la superficie circular analizada, dividida en dovelas y con los valores de los factores de seguridad encontrados por los métodos de de Fellenius y de Bishop simplificado.

Finalmente arroja un archivo de texto con el resumen del análisis realizado.

A.13.2. Subfunciones externas

defineslipcircle: Ver su descripción en la sección A.5.
defineswatertable: Ver su descripción en la sección A.6.
divideslipintoslices: Ver su descripción en la sección A.7.
interatefbishopsimpsat: Ver su descripción en la sección A.9.

interateffelleniussat: Ver su descripción en la sección A.10.
materialboundary: Ver su descripción en la sección A.11.
obtainmaxdepthdist: Ver su descripción en la sección A.12.
plotslice: Ver su descripción en la sección A.14.
reportslicestructurevalues: Ver su descripción en la sección A.16.
sliparcdiscretization: Ver su descripción en la sección A.17.
terrainsurface: Ver su descripción en la sección A.19.
vertprojection2pline: Ver su descripción en la sección A.22.

A.13.3. Variables de entrada

projectName: Nombre o título del proyecto. Es de tipo *cadena de caracteres*.
projectAuthor: Nombre del autor del proyecto. Es de tipo *cadena de caracteres*.
projectDate: Fecha de realización del análisis. Es de tipo *cadena de caracteres*. Por defecto genera automáticamente la fecha tomándola de la máquina en la que se ejecuta
slopeHeight: Lista de dos elementos con la altura del talud de tipo *escalar* y la unidad de medida de tipo *cadena de caracteres*.
slopeDip: Pendiente del talud, expresada en un vector de $\mathbb{R}^2$, donde la primer componente representa una distancia horizontal, y la segunda una vertical, las cuales no son necesariamente las distancias reales del talud. Es de tipo *array*.
crownDist: Lista de dos elementos con la longitud de la corona del talud de tipo *escalar* y la unidad de medida de tipo *cadena de caracteres*.
toeDist: Lista de dos elementos con la longitud del pie del talud de tipo *escalar* y la unidad de medida de tipo *cadena de caracteres*.
wantAutomaticToeDepth: Variable de verificación sobre si se desea o no generar la profundidad del talud. Es de tipo *booleano*.
toeDepth: Lista de dos elementos con la profundidad del talud de tipo *escalar* y la unidad de medida de tipo *cadena de caracteres*.
hztDistPointAtCrownFromCrown: Lista de dos elementos: el primero corresponde a la distancia horizontal del punto más cerca a la corona que define la superficie de deslizamiento, medida desde el límite de la corona con la superficie inclinada del talud (puntos a la izquierda son negativos), es de tipo *escalar*; y es segundo elemento es la unidad de medida de tipo *cadena de caracteres*.
hztDistPointAtToeFromCrown: Lista de dos elementos: el primero corresponde a la distancia horizontal del punto más cerca al pie que define la superficie de deslizamiento, medida desde el límite de la corona con la superficie inclinada del talud, es de tipo *escalar*; y es segundo elemento es la unidad de medida de tipo *cadena de caracteres*.
slipRadius: Lista de dos elementos con la longitud el radio de la superficie circular de deslizamiento de tipo *escalar* y la unidad de medida de tipo *cadena de caracteres*.

wantWatertable: Variable de verificación sobre si se desea o no generar una superficie de nivel freático. Es de tipo *booleano*.

wtDepthAtCrown: Lista de dos elementos con la profundidad del nivel freático medido desde la corona del talud de tipo *escalar* y la unidad de medida de tipo *cadena de caracteres*.

toeUnderWatertable: Variable de verificación sobre si se desea o no que el nivel freático esté por encima del pié del talud (*i.e.* talud parcialmente sumergido). Es de tipo *booleano*.

waterUnitWeight: Lista de dos elementos con el peso específico del agua de tipo *escalar* y la unidad de medida de tipo *cadena de caracteres*.

materialUnitWeight: Lista de dos elementos con el peso específico del material que compone el talud de tipo *escalar* y la unidad de medida de tipo *cadena de caracteres*.

frictionAngleGrad: Lista de dos elementos con el ángulo de fricción del material que compone el talud de tipo *escalar* y la unidad de medida de tipo *cadena de caracteres*.

cohesion: Lista de dos elementos con la cohesión del material que compone el talud de tipo *escalar* y la unidad de medida de tipo *cadena de caracteres*.

wantConstSliceWidthTrue: Variable de verificación sobre si se desea o no que el ancho de las dovelas sea el mismo para todas. Es de tipo *booleano*.

numSlices: Número de dovelas en las que se quiere dividir la masa de suelo. Es de tipo *escalar*.

nDivs: Número de divisiones en las que se dividirá el arco circular de deslizamiento. Se recomienda que sea igual al valor de la variable `numSlices`. Es de tipo *escalar*.

methodString: Abreviación del método que se usará para el análisis. Es de tipo *cadena de caracteres*, y se puede escoger entre `'Flns'` para el método de Fellenius, `'Bshp'` para el método de Bishop Simplificado o `'Allm'` para ambos.

outputFormatImg: Extensión de la imagen de salida. Es de tipo *cadena de caracteres*, y se puede escoger entre `'.eps'`, `'.jpeg'`, `'.jpg'`, `'.pdf'`, `'.pgf'`, `'.png'`, `'.raw'`, `'.rgba'`, `'.svg'`, `'.svgz'`, `'.tif'`, o `'.tiff'`.

A.13.4. Variables de salida

msg Mensaje informativo dependiendo de si la función se ejecutó satisfactoriamente, o si hubo algun error en las variables de entrada. De ser la segunda opción, describe en qué variable está el error. Es de tipo *cadena de caracteres*.

Esta función también genera dos archivos de salida: un archivo de imagen con el esquema del problema analizado, y un archivo de texto con el resumen del análisis. Estos archivos se describen mejor en la sección 3.2.

A.13.5. Ejemplo 01

```
#---------------------------------------------------------------------- #
### Add functions directory ###
import sys
sys.path += ['../functions']

#---------------------------------------------------------------------- #
### Modules/Functions import ###
import numpy as np
import time
from onlyonecircle import onlyonecircle

#---------------------------------------------------------------------- #
### Poject data ###
projectName = 'Example-01'
projectAuthor = 'Exneyder A. Montoya Araque'
projectDate = time.strftime("%d/%m/%y")

#---------------------------------------------------------------------- #
### Define inputs ###
# previous values #
segmentAC = 40/np.sin(np.radians(30))
# The slope geometry #
slopeHeight = [40, 'ft']
slopeDip = np.array([1, np.tan(np.radians(45))])
crownDist = [30, 'ft']
toeDist = [30, 'ft']
wantAutomaticToeDepth = False
if wantAutomaticToeDepth == True:
        toeDepth = ['automatic toe Depth']
else:
        toeDepth = [10, 'ft']
# The slip arc-circle #
hztDistPointAtCrownFromCrown = [40/np.tan(np.radians(45))-\
40/np.tan(np.radians(30)), 'ft']
hztDistPointAtToeFromCrown = [40/np.tan(np.radians(45)), 'ft']
slipRadius = [0.5*80/np.sin(0.5*np.radians(70)), 'ft']
# Water table depth #
wantWatertable = False
if wantWatertable == True:
        wtDepthAtCrown = [0, 'm']
else:
        wtDepthAtCrown = ['No watertable']
toeUnderWatertable = False
# Materials properties. #
waterUnitWeight = [62.4, 'pcf']
materialUnitWeight = [115, 'pcf']
frictionAngleGrad = [20, 'degrees']
cohesion = [400, 'psf']

### Advanced inputs ###
# Want divide the slip surface in constant width slices? #
wantConstSliceWidthTrue = False
# Number of discretizations of slip surface. #
numSlices = 15
# Number of discretizations of circular arcs. #
nDivs = numSlices
# Select the method to calcualte the safety factor#
methodString = 'Allm'
# Select the output format image #
```

```
outputFormatImg = '.pdf'

#-------------------------------------------------------------------- #
# Operations for only one slip surface #
onlyonecircle(projectName, projectAuthor, projectDate, slopeHeight, \
        slopeDip, crownDist, toeDist, wantAutomaticToeDepth, toeDepth, \
        hztDistPointAtCrownFromCrown, hztDistPointAtToeFromCrown, \
        slipRadius, wantWatertable, wtDepthAtCrown, toeUnderWatertable, \
        waterUnitWeight, materialUnitWeight, frictionAngleGrad, cohesion, \
        wantConstSliceWidthTrue, numSlices, nDivs, methodString, \
        outputFormatImg)
```

A.13.6. Ejemplo 02

```
#-------------------------------------------------------------------- #
### Add functions directory ###
import sys
sys.path += ['../functions']

#-------------------------------------------------------------------- #
### Modules/Functions import ###
import numpy as np
import time
from onlyonecircle import onlyonecircle

#-------------------------------------------------------------------- #
### Poject data ###
projectName = 'Example-02'
projectAuthor = 'Exneyder A. Montoya Araque'
projectDate = time.strftime("%d/%m/%y")

#-------------------------------------------------------------------- #
### Define inputs ###
# previous values #
segmentAC = 40/np.sin(np.radians(30))
# The slope geometry #
slopeHeight = [11.5, 'm']
slopeDip = np.array([3, 8])
crownDist = [15, 'm']
toeDist = [15, 'm']
wantAutomaticToeDepth = True
if wantAutomaticToeDepth == True:
        toeDepth = ['automatic toe Depth']
else:
        toeDepth = [10, 'ft']
# The slip arc-circle #
hztDistPointAtCrownFromCrown = [-11, 'm']
hztDistPointAtToeFromCrown = [13.5, 'm']
slipRadius = [15.6, 'm']
# Water table depth #
wantWatertable = True
if wantWatertable == True:
        wtDepthAtCrown = [3.7, 'm']
else:
        wtDepthAtCrown = ['No watertable']
toeUnderWatertable = False
# Materials properties. #
```

```
waterUnitWeight = [9.8, 'kN/m3']
materialUnitWeight = [18, 'kN/m3']
frictionAngleGrad = [21, 'degrees']
cohesion = [4.5, 'kPa']

### Advanced inputs ###
# Want divide the slip surface in constant width slices? #
wantConstSliceWidthTrue = False
# Number of discretizations of slip surface. #
numSlices = 10
# Number of discretizations of circular arcs. #
nDivs = numSlices
# Select the method to calcualte the safety factor#
methodString = 'Allm'
# Select the output format image #
outputFormatImg = '.pdf'

#-------------------------------------------------------------------- #
# Operations for only one slip surface #
onlyonecircle(projectName, projectAuthor, projectDate, slopeHeight, \
        slopeDip, crownDist, toeDist, wantAutomaticToeDepth, toeDepth, \
        hztDistPointAtCrownFromCrown, hztDistPointAtToeFromCrown, \
        slipRadius, wantWatertable, wtDepthAtCrown, toeUnderWatertable, \
        waterUnitWeight, materialUnitWeight, frictionAngleGrad, cohesion, \
        wantConstSliceWidthTrue, numSlices, nDivs, methodString, \
        outputFormatImg)
```

A.14. plotslice

```
slicePlineCordsArray = plotslice(slicesSTR)
```

A.14.1. Descripción

Dibuja una dovela a partir de su estructura.

A.14.2. Variables de entrada

slicesSTR: Estructuras de datos de tipo *diccionario* de una dovela. Cada estructura contiene los campos obtenidos con la función `divideslipintoslices` (A.7).

A.14.3. Variables de salida

slicePlineCordsArray: Matriz $n \times 2$ que contiene las coordenadas del contorno de la dovela. Es de tipo *array*.

A.14.4. Ejemplo 01

```
slicesSTR = \
{'area': 18.063276613019383,
'hrzMomentArm': -5.4527963142320601,
'inclinationAngleGradAtBottom': 21.379968728885775,
'inclinationAngleGradAtTop': 68.198590513648185,
'midHeight': 8.6015602919139909,
'midPoint': np.array([ 11.45      ,   3.07666551]),
'plineCords': np.array([[ 10.4       ,  14.30322581],
[ 10.4       ,   3.4877326 ],
[ 11.45      ,   3.07666551],
[ 12.5       ,   2.66559843],
[ 12.5       ,   9.05322581],
[ 10.4       ,  14.30322581]]),
'vrtMomentArm': 5.3266677434544878,
'width': 2.0999999999999996,
'wtMidHeight': 8.6015602919139909,
'wtMidHeightAboveSlope': 0.0}
print(slicePlineCordsArray = plotslice(slicesSTR))
```

A.14.5. Ejemplo 02

```
slicesSTR = \
{'area': 0.79712195542504105,
'hrzMomentArm': 5.0472036857679434,
'inclinationAngleGradAtBottom': -19.719424036464194,
'inclinationAngleGradAtTop': -0.0,
'midHeight': 0.37958188353573608,
'midPoint': np.array([ 21.95      ,   2.92364392]),
'plineCords': np.array([[ 20.9       ,   3.30322581],
[ 20.9       ,   2.54728785],
[ 21.95      ,   2.92364392],
[ 23.        ,   3.3       ],
[ 23.        ,   3.30322581],
[ 20.9       ,   3.30322581]]),
'vrtMomentArm': 13.70166774345449,
'width': 2.0999999999999979,
'wtMidHeight': 0.37958188353573608,
'wtMidHeightAboveSlope': 0.0}
print(slicePlineCordsArray = plotslice(slicesSTR))
```

A.15. polyarea

```
area = polyarea(xCoord, yCoord)
```

A.15.1. Descripción

Calcula el área de un polígono simple cuyos vértices están definidos por pares ordenados en el plano. Está basada en la fórmula del área de Gauss.

A.15.2. Variables de entrada

xCoord: Vector $1 \times n$ con las coordenadas en x del polígono. Es de tipo *array*.
yCoord: Vector $1 \times n$ con las coordenadas en y del polígono. Es de tipo *array*.

A.15.3. Variables de salida

area: Valor del área del polígono. Es de tipo *escalar*.

A.15.4. Ejemplo 01

```
xCoord = np.arange(1,10)
yCoord = xCoord**2
print(polyarea(xCoord, yCoord))
```

A.15.5. Ejemplo 02

```
xCoord = np.arange(10, 50)
yCoord = np.sqrt(xCoord)
print(polyarea(xCoord, yCoord))
```

A.16. reportslicestructurevalues

```
reportCell, reportedArray = reportslicestructurevalues(slicesSTRCell)
```

A.16.1. Descripción

Transforma y resume la información de las estructuras de datos de todas las dovelas.

A.16.2. Variables de entrada

slicesSTRCell: Lista que almacena todas las estructuras de datos de tipo *diccionario* de las dovelas. Se obtiene ejecutando previamente la función `divideslipintoslices` (A.7).

A.16.3. Variables de salida

reportedArray: Matriz $n \times 7$ de tipo *array* que resume las estructuras de datos de todas las dovelas, donde cada fila corresponde a la información de cada dovela, y cada columna corresponde a los siguientes campos:

Abscisa del punto medio de la base
Ordenada del punto medio de la base
Área de la dovela
Ancho de la dovela
Altura promedio de la dovela (tomando los puntos extremos y medio de la base)
Ángulo de inclinación de la secante de la base de la dovela
Ángulo de inclinación de la secante del tope de la dovela
Altura promedio de la columna de agua (tomando los puntos extremos y medio de la base)
Altura promedio de la columna de agua que está por encima de la superficie del talud
Componente horizontal del brazo del momento ejercido por la columna de agua por encima de la superficie del talud
Componente vertical del brazo del momento ejercido por la columna de agua por encima de la superficie del talud

reportCell: Arreglo $n+1 \times 8$ de tipo *array* que asemeja una tabla, que resume las estructuras de datos de todas las dovelas. Los campos de la primer fila corresponde a las

etiquetas de la tabla así; la primer columna corresponde a una numeración índice, y las demás celdas son las mismas que reportedArray.

A.16.4. Ejemplo 01

```
# inputs: #
slopeHeight = 12.0; slopeDip = np.array([1, 2.5]); crownDist = 10.0
toeDist = 10.0; wtDepthAtCrown = 0; numSlices = 10; slipRadius = 15
pointAtToeVec = np.array([23, 3.3]); pointAtCrownVec = np.array([2, 15.3])
# Previous functions #
boundPointsCordsArray, fromToeOriginRowVec, coordTransMat = materialboundary(
     slopeHeight, slopeDip, crownDist, toeDist)
surfaceDataCell, surfaceChordsArray = terrainsurface(fromToeOriginRowVec, slopeHeight,
      slopeDip, crownDist, toeDist)
watertableDataCell, wtCoordsArray = defineswatertable(wtDepthAtCrown, surfaceDataCell)
existSlipCircleTrue, slipArcSTR = defineslipcircle(pointAtToeVec, pointAtCrownVec,
     slipRadius)
slicesSTRCell = divideslipintoslices(slipArcSTR, surfaceDataCell, watertableDataCell,
     numSlices, pointAtToeVec, pointAtCrownVec)
# This function #
print(reportslicestructurevalues(slicesSTRCell))
```

A.16.5. Ejemplo 02

```
# inputs: #
slopeHeight = 15.0; slopeDip = np.array([1, 1]); crownDist = 10.0
toeDist = 10.0; wtDepthAtCrown = 0; numSlices = 10; slipRadius = 25
pointAtToeVec = np.array([30, 5]); pointAtCrownVec = np.array([1, 20])
# Previous functions #
boundPointsCordsArray, fromToeOriginRowVec, coordTransMat = materialboundary(
     slopeHeight, slopeDip, crownDist, toeDist)
surfaceDataCell, surfaceChordsArray = terrainsurface(fromToeOriginRowVec, slopeHeight,
      slopeDip, crownDist, toeDist)
watertableDataCell, wtCoordsArray = defineswatertable(wtDepthAtCrown, surfaceDataCell)
existSlipCircleTrue, slipArcSTR = defineslipcircle(pointAtToeVec, pointAtCrownVec,
     slipRadius)
slicesSTRCell = divideslipintoslices(slipArcSTR, surfaceDataCell, watertableDataCell,
     numSlices, pointAtToeVec, pointAtCrownVec)
# This function #
print(reportslicestructurevalues(slicesSTRCell))
```

A.17. sliparcdiscretization

`arcPointsCoordsArray = sliparcdiscretization(pointAtToeVec, pointAtCrownVec, nDivs, slipArcSTR)`

A.17.1. Descripción

Obtiene las coordenadas y también genera la gráfica del arco de la superficie de deslizamiento, conocidos los puntos inicial y final, el número de divisiones que tendrá y la estructura del arco.

El arco está representado por una polilínea, por lo tanto hay que ingresar el número de segmentos que el arco puede ser dividido, pero es recomendado que sea igual al número de dovelas para futuros cálculos.

A.17.2. Variables de entrada

pointAtToeVec: Vector bidimensional con las coordenadas que define el deslizamiento en el pie del talud. Es de tipo *array*.

pointAtCrownVec: Vector bidimensional con las coordenadas que define el deslizamiento en la corona del talud. Es de tipo *array*.

nDivs: Número de segmentos de la polilínea que define el arco. Es de tipo *escalar*.

slipArcSTR: Estructura de datos de tipo *diccionario* del arco de circunferencia del deslizamiento. Se obtiene previamente ejecutando la función defineslipcircle—.

want2plot: Corresponde a una variable de verificación sobre si desea o no graficar el arco resultante. Es de tipo *booleano*, su valor por defecto es False—.

A.17.3. Variables de salida

arcPointsCoordsArray: Matriz $n \times 2$ que representa las coordenadas de los puntos que definen el arco. Es de tipo *array*.

A.17.4. Ejemplo 01

```
# inputs #
pointAtToeVec = np.array([23, 3.3]); pointAtCrownVec = np.array([2, 15.3])
radius = 34.4848; nDivs = 6
# previous functions #
existSlipCircleTrue, slipArcSTR = defineslipcircle(pointAtToeVec, pointAtCrownVec,
    radius)
# this funtion #
print(sliparcdiscretization(pointAtToeVec, pointAtCrownVec, nDivs, slipArcSTR))
```

A.17.5. Ejemplo 02

```
# inputs #
pointAtToeVec = np.array([50, 5]); pointAtCrownVec = np.array([0, 20])
radius = 65; nDivs = 10
# previous functions #
existSlipCircleTrue, slipArcSTR = defineslipcircle(pointAtToeVec, pointAtCrownVec,
    radius)
# this funtion #
print(sliparcdiscretization(pointAtToeVec, pointAtCrownVec, nDivs, slipArcSTR))
```

A.18. tangentlineatcirclept

```
   isPtBetweenArcLimitsTrue, tangentLineSTR = tangentlineatcirclept(atCirclePointVec, slipC
)
```

A.18.1. Descripción

Calcula una línea que es tangente a un punto especificado que pertenece a un arco de circunferencia.

El código verifica si el punto pertenece a la ecuación del arco circular, o tolera algunos puntos que están bastante cerca del circulo, recalculando la coordenada y, basado en la coordenada x del punto. Una vez asegurado que el punto pertenece al circulo del arco, la función verifica si el punto está entre los límites que definen el arco.

A.18.2. Subfunciones externas

azimuthangle: Ver su descripción en la sección A.2
unitvector: Ver su descripción en la sección A.21

A.18.3. Variables de entrada

atCirclePointVec: Vector bidimensional que define el punto que pertenece al arco de circunferencia. Es de tipo *array*.

slipCircleSTR: Estructura de datos de un arco circular del cual se desea obtener la línea tangente. Se obtiene previamente con lafunción `defineslipcircle` (A.5). Es de tipo *diccionario*.

A.18.4. Variables de salida

isPtBetweenArcLimitsTrue: Variable lógica con valor *True*, si el punto dado est'a dentro del arco circular. Es de tipo *booleano*.

tangentLineSTR: Estructura de datos de la linea infinita tangente al punto dado. Es de tipo *diccionario*, y tiene los siguientes campos:

refPtVec Punto de referencia por donde pasa la linea obtenida.

unitVec Vector director de la linea.

slope Pendiente de la linea.

nearestAzimuthAngRad Ángulo en radianes del azimut de uno de los sentidos de la linea tangente, medido en contra de las manecillas del reloj desde el vector unitario de referencia [1, 0] y que está más cercano a este vector de referencia.

farestAzimuthAngRad Similar que el anterior, pero corresponde al sentido más alejado del vector de referencia [1, 0].

intercept Valor del intercepto de la ecuación de la linea.

A.18.5. Ejemplo 01

```
atCirclePointVec = np.array([16.7036, 14.1941])
slipCircleSTR = {'center':np.array([31.3936, 45.3936]), 'radius':34.4848,
'iniAngGrad': 218.3437, 'endAngGrad': 284.4522, 'deepDist': 10.9088,
'leftDist': -3.0912}
print(tangentlineatcirclept(atCirclePointVec, slipCircleSTR))
```

A.18.6. Ejemplo 02

```
atCirclePointVec = np.array([16.7036, 14.1941])
pointAtToeVec = np.array([40, 12])
pointAtCrownVec = np.array([4.347, 24])
slipRadius = 34.4848
existSlipCircleTrue, slipCircleSTR = defineslipcircle(pointAtToeVec, pointAtCrownVec,
    slipRadius)
print(tangentlineatcirclept(atCirclePointVec, slipCircleSTR))
```

A.19. terrainsurface

```
surfaceDataCell = terrainsurface(fromToeOriginRowVec, boundPointsCordsArray, want2plot
= False
```

A.19.1. Descripción

Define y grafica la superficie del talud como un polígono abierto. El polígono es almacenado en una lista, en la cual cada elemento es la estructura de datos de un segmento.

A.19.2. Subfunciones externas

create2dsegmentstructure: Ver su descripción en la sección A.4

A.19.3. Variables de entrada

fromToeOriginRowVec: Vector de desplazamiento que conecta el pie del talud con el sistema global de coordenadas. Se obtiene al ejecutar previamente la función material-boundary—. Es de tipo *array*.

slopeHeight: Altura del talud. Es de tipo *escalar*.

slopeDip: Pendiente del talud, expresada en un vector de $\mathbb{R}^2$, donde la primer componente representa una distancia horizontal, y la segunda una vertical, las cuales no son necesariamente las distancias reales del talud. Es de tipo *array*.

crownDist: Distancia del plano horizontal en la cabeza del talud. Es de tipo *escalar*.

toeDist: Distancia horizontal del plano en el pie del talud. Es de tipo *escalar*.

want2plot: Corresponde a una variable de verificación sobre si desea o no graficar los límites del material. Es de tipo *booleano*, su valor por defecto es False—.

A.19.4. Variables de salida

surfaceDataCell: Lista que almacena la estructura de datos para cada linea que conforma el polígono abierto de la superficie del talud. Cada elemento de la lista es de tipo *diccionario* y contiene los siguientes campos:

iniPtVec Vector de tipo *array* con las coordenadas del primer punto de la linea.

endPtVec Vector de tipo *array* con las coordenadas del último punto de la linea.

unitVec Vector unitario de tipo *array* que define una dirección.

lambda Valor de tipo *escalar* que define la longitud del segmento.

slope Valor de tipo *escalar* que define la pendiente de la ecuación de la linea del segmento.

azimuthRad Valor de tipo *escalar* que define el ángulo (en radianes) del segmento, medido en contra de las manecillas del reloj desde el eje de referencia $[1,0]$ hasta el $[0,1]$.

intercept Valor de tipo *escalar* que define el intercepto de la ecuación de la linea del segmento.

surfaceChordsArray Arreglo matricial $n \times 2$ que almacena las coordenadas de los vértices de la polilínea que define la superficie del talud. Es de tipo *array*.

A.19.5. Ejemplo 01

```
slopeHeight = 12
slopeDip = np.array([2.5, 1])
crownDist = 10.0
toeDist = 10.0
fromToeOriginRowVec = np.array([-14.8, -3.30])
print(terrainsurface(fromToeOriginRowVec, slopeHeight, slopeDip, crownDist, toeDist))
```

A.19.6. Ejemplo 02

```
slopeHeight = np.random.randint(1,20)
slopeDip = np.array([1.5, 1])
crownDist = np.random.randint(1,10)
toeDist = np.random.randint(1,10)
temp1, fromToeOriginRowVec, temp2 = materialboundary(slopeHeight, slopeDip, crownDist,
    toeDist)
print(terrainsurface(fromToeOriginRowVec, slopeHeight, slopeDip, crownDist, toeDist))
```

A.20. uniquewithtolerance

```
uniqueArray = uniquewithtolerance(array, tolerance)
```

A.20.1. Descripción

Dado un *array* $n \times 2$, extrae las filas de valores únicos, los cuales no son necesariamente ni estrictamente igual a los de otras filas. La decisión es gobernada por un valor de tolerancia.

A.20.2. Variables de entrada

array: Arreglo matricial de dimensión $n \times 2$ del que se extraer'an las filas de valores únicos. Es de tipo *array*.

tolerance: Valor que definirá la tolerancia del criterio de selección. Su valor por defecto es 0.001. Es de tipo *escalar*.

A.20.3. Variables de salida

uniqueArray: Nuevo arreglo matricial de filas únicas. Es de tipo *array*.

A.20.4. Ejemplo 01

```
array  = np.array([[9, 2], [7,4.123], [5,6.129], [7,8.12], [0,1.1], [9.001, 2]])
print(uniquewithtolerance(array))
```

A.20.5. Ejemplo 02

```
array = np.array = np.random.rand(10,2)
print(uniquewithtolerance(array, 0.05))
```

A.21. unitvector

```
uVector = unitvector(vector)
```

A.21.1. Descripción

Devuelve el vector unitario de un vector con magnitud distinta de la unidad para $\mathbb{R}^n$.

A.21.2. Variables de entrada

vector: Vector con magnitud distinta de la unidad de dimensión $n \times 1$. Es de tipo *array*, que es como se denominan los arreglos vectoriales y matriciales en *Python*.

A.21.3. Variables de salida

uVector: Vector de magnitud unitaria, el cual tiene la misma dimensión, dirección y sentido que el vector de entrada vector—. Es igualmente de tipo *array*.

A.21.4. Ejemplo 01

```
vector = np.array([0.20, 0.58, 1.36])
print(unitvector(vector))
```

A.21.5. Ejemplo 02

```
vector =np.random.rand(2)
print(unitvector(vector))
```

A.21.6. Validación

```
import numpy as np
import scipy.linalg as la
from unitvector import *

v = np.random.rand(3)
vNorm = la.norm(v)
```

```
vuVector = unitvector(v)
w = vNorm*vuVector

isValid = True
i = 0
for j in range(len(v)):
    if v[i] == w[i]:
        i += 1
        pass
    else:
        i += 1
        isValid = False

if isValid:
    print('v is equal to w, function is verified.')
else:
    print('v is not equal to w, function is not robust.')
```

A.22. vertprojection2pline

```
projectedPointVec = vertprojection2pline(pointVec, plineStructureCell)
```

A.22.1. Descripción

Obtiene las coordenadas de un punto (o grupo de puntos) proyectado verticalmente sobre una polilínea abierta.

A.22.2. Variables de entrada

pointVec: Vector del punto que se desea proyectar verticalmente sobre la poliínea. Es de tipo *array*.

plineStructureCell: Lista que almacena la estructura de datos para cada linea que conforma el polígono abierto de la sobre el que se desea proyectar el punto. Cada elemento de la lista es de tipo *diccionario* y contiene los mismos campos que las estructuras obtenidas con la función `create2dsegmentstructure`.

A.22.3. Variables de salida

projectedPointVec: Vector con las coordenadas del punto proyectado sobre la polilínea. Es de tipo *array*.

A.22.4. Ejemplo 01

```
pointVec = np.array([7.35, 18])
plineStructureCell = [
    {'iniPtVec':np.array([0, 24]), 'endPtVec':np.array([10, 24]), 'unitVec':\
    np.array([1, 0]), 'lambda':10, 'slope':0, 'azimuthRad':0, 'intercept':24},
    {'iniPtVec':np.array([10, 24]), 'endPtVec':np.array([40, 12]), 'unitVec':\
    np.array([0.9285, -0.3714]), 'lambda':32.3110, 'slope':-0.4228,
    'azimuthRad':5.9027, 'intercept':34.1470},
    {'iniPtVec':np.array([40, 12]), 'endPtVec':np.array([50, 12]), 'unitVec':\
    np.array([1, 0]), 'lambda':10, 'slope':0, 'azimuthRad':0, 'intercept':12}]
print(vertprojection2pline(pointVec, plineStructureCell))
```

A.22.5. Ejemplo 02

```
pointVec = np.array([-20, 15]); slopeHeight = 12; slopeDip = np.array([2.5, 1]);
     crownDist = 10.0; toeDist = 10.0; fromToeOriginRowVec = np.array([-14.8, -3.30])
plineStructureCell, surfaceChordsArray = terrainsurface(fromToeOriginRowVec,
     slopeHeight, slopeDip, crownDist, toeDist)
print(vertprojection2pline(pointVec, plineStructureCell))
```

Apéndice B
Consideraciones finales

B.1. Alojamiento y desarrollo del código

El desarrollo y administración del presente código está alojado en la plataforma para el hospedaje de códigos, que permite la colaboración y control de versiones, denominada *GitHub* bajo el nombre de pyCSS.

Allí podrá obtener un archivo comprimido `.zip` que contiene el programa pyCSS® y el presente manual del usuario en formato `.pdf`.

A través de este sitio se hace las respectivas descargas, aportes y peticiones de participación en el proyecto.

B.2. Licencia

Los autores son miembros del *Semillero de Geología Matemática y Computacional* parte del Grupo de Investigación en Geotecnia de la Facultad de Minas de la Universidad Nacional de Colombia en Medellín.

B.3. Descargo de responsabilidades

El presente código computacional se distribuye con la esperanza de que sea útil, pero sin ninguna garantía; sin la garantía implícita en su comercialización o idoneidad para un propósito particular. Consulte la Licencia BSD-2 para mayores detalles.