# UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

FACULTAD DE INGENIERÍA CIVIL

# DESARROLLO DE UNA HERRAMIENTA DE CÁLCULO PARA EL DISEÑO DE MUROS DE CONTENCIÓN

Tesis profesional para obtener el grado de:

**INGENIERO CIVIL**


Autor:

**Héctor Alfonso Sam Arroyo**


Asesor de tesis:

**Dr. Carlos Chávez Negrete**

Morelia, Michoacán, noviembre de 2021

# RESUMEN

El proceso de diseño de un muro de contención, es un proceso complicado e iterativo. La presente tesis ofrece una alternativa rápida y eficaz mediante la programación de una herramienta de cálculo en el lenguaje de programación Python, capaz de ofrecer una propuesta de diseño de un muro de contención, tras realizar todas las etapas del proceso, siendo estas:

- Cálculo de los empujes activo y pasivo actuantes sobre el muro debido al relleno contenido (homogéneo o estratificado) utilizando la teoría de Rankine.
- Cálculo del empuje debido a distintos tipos de sobrecargas mediante la ecuación de Boussinesq.
- Cálculo del incremento del empuje activo debido a sismo utilizando la teoría de Mononobe-Okabe.
- Pre dimensionamiento del muro.
- Cálculo de la estabilidad del muro ante volcamiento y deslizamiento.
- Cálculo de la capacidad de carga del terreno utilizando la teoría de Vesić.
- Diseño del armado del muro apegado al reglamento de construcción de la Ciudad de México y las normas NTC-17.

**Conceptos clave:** Muro de contención, Python, estabilidad, capacidad de carga, armado.

# ABSTRACT

The design process of a retaining wall is a complicated and iterative process. This thesis presents a fast and efficient alternative by programming a calculation tool in the Python programming language, capable of providing a design proposal for a retaining wall, after performing all the stages of the process, these being:

- Calculation of the active and passive thrusts acting on the wall due to the contained backfill (homogeneous or stratified) using Rankine theory.
- Calculation of the thrust due to different types of surcharges using the Boussinesq equation.
- Calculation of the active thrust increment due to earthquake using the Mononobe-Okabe theory.
- Pre-sizing of the wall.
- Calculation of the stability of the wall against overturning and sliding.
- Calculation of the bearing capacity of the soil using Vesić's theory.
- Design of the wall steel reinforcement according to Mexico City construction code and NTC-17 standards.

**Key concepts:** retaining wall, Python, stability, bearing capacity, steel reinforcement.

# AGRADECIMIENTOS

A todos mis profesores de la Facultad de Ingeniería Civil que, con sus enseñanzas, dedicación y amistad, me ayudaron a crecer como profesional.

A mi asesor, Dr. Carlos Chávez Negrete que, con su dirección y conocimientos, permitió el desarrollo de este trabajo.

# DEDICATORIA

A mis padres, Claudia Arroyo Rodríguez y Héctor Sam Vargas, por todo el apoyo, cariño y ánimo que me brindaron a lo largo de mis estudios.

A mi abuelita Tita, por toda su ayuda y amor a lo largo de estos años.

A mi pareja y a mis amigos, por siempre motivarme a superarme.

# CONTENIDO

# ÍNDICE DE FIGURAS

# 1 INTRODUCCIÓN

Los muros de contención son estructuras útiles para retener una masa de suelo cuando ocurre un cambio abrupto de nivel en el terreno, con una pendiente que el propio suelo no podría mantener por sí mismo (vertical o casi vertical).

El proceso de diseño de un muro de contención, realizado de forma manual, es un proceso largo e iterativo, pudiéndose tornar bastante complejo. La presente tesis busca ofrecer una solución a ese problema.

Hoy en día, con ayuda de las computadoras y de software especializado, es posible hacer mucho más eficiente el proceso de diseño de cualquier estructura. Por lo que la propuesta de solución a la problemática planteada surge de la programación de un código en Python, capaz de realizar el proceso completo del diseño de un muro de contención de gravedad o en voladizo, tomando como entrada los datos de campo del terreno donde se ubicará el muro.

Para desarrollar el programa, se escogió el lenguaje Python debido a que se trata de un lenguaje interpretado, lo que lo vuelve sencillo de comprender para cualquier persona con nociones básicas de programación, además de ser de código abierto lo que facilita que la comunidad de programadores haya desarrollado una variedad de módulos o librerías útiles para el propósito que interesa a esta tesis.

En este documento, se explica detalladamente el marco teórico utilizado para la programación de la herramienta de cálculo, se enumeran casos de verificación en los que se comprueba el correcto funcionamiento del código escrito y se presenta un ejemplo de aplicación del programa desarrollado para el diseño completo de un muro de contención.

# 1.1 OBJETIVOS

## 1.1.1 GENERAL

Desarrollar una herramienta de cálculo que realice el proceso de diseño de muros de contención de gravedad o en voladizo, para una amplia variedad de condiciones físicas del relleno contenido y sobrecargas actuantes.

## 1.1.2 PARTICULARES

- Desarrollar programas individuales para la resolución de cada una de las partes del proceso de diseño de muros de contención e integrarlos en una sola aplicación.
- Creación de una GUI para facilitar el uso del programa desarrollado mediante Ipywidgets.
- Automatizar la generación de un reporte de resultados en un archivo Markdown.
- Comprobar el correcto funcionamiento del programa desarrollado mediante la resolución de casos de verificación, propuestos en diversas bibliografías.

# 1.2 METODOLOGÍA

Para dar solución a la problemática planteada, se siguió la siguiente metodología:

Se realizó una revisión bibliográfica de los temas necesarios para el diseño de muros de contención, con la finalidad de obtener un marco teórico.

Se programó una herramienta en Python que realiza, por separado, cada uno de los pasos del proceso de diseño de muros de contención de gravedad y en voladizo,

siendo estos: cálculo del empuje activo, cálculo del empuje pasivo, cálculo de los diferentes tipos de sobrecargas, cálculo del incremento del empuje activo debido a sismo, cálculo del empuje total, estabilidad del muro ante volteo y deslizamiento, capacidad de carga del terreno y armado del muro.

Para desarrollar el programa, se escogió el lenguaje Python debido a que cuenta de manera gratuita con una amplia variedad de módulos y librerías útiles para la manipulación de datos.

Como entorno de desarrollo integrado o IDE, se escogió Google Colaboratory que es un servicio en la nube gratuito basado en Jupyter Notebooks y que actualmente trabaja con Python 3.6. Una IDE, es un sistema de software que contiene las herramientas principales necesarias para facilitar al programador el desarrollo de aplicaciones. Las ventajas que ofrece esta IDE, son la versatilidad en el manejo de archivos debido a que las notebooks se pueden conectar a Google Drive y la facilidad de compartir las notebooks con otros usuarios.

Se integraron todos los códigos en una herramienta que funciona en conjunto para la resolución del proceso completo del diseño de un muro de contención.

Para ofrecer una mejor experiencia al utilizar la herramienta de cálculo, se creó una interfaz gráfica de usuario o GUI, cuya finalidad es proporcionar un entorno visual para facilitar la comunicación entre el usuario y la aplicación. La GUI se programó utilizando ipywidgets, una librería de widgets basada en HTML, un lenguaje marcado que se utiliza para indicar la estructura de un documento mediante etiquetas y CSS un lenguaje de diseño visual para definir la presentación de un documento HTML útil para volver interactiva una notebook de Python.

Por último, se automatizó la creación de un reporte de resultados en un archivo Markdown. Markdown es un lenguaje de marcado ligero que convierte el texto en documentos HTML utilizando la traducción a Python de la librería html2text.

# 2 PRESIÓN LATERAL DE TIERRA

El diseño de una estructura de contención parte del cálculo de las presiones que estarán actuando sobre la misma.

La presión lateral de tierra, depende de varios factores como son: las propiedades físicas del suelo, la resistencia al corte por la consolidación del suelo, la interacción entre la masa de suelo contenida, la estructura de contención y las cargas presentes (debidas al propio material contenido, sobrecargas o cargas por sismo).

La presión lateral de tierras, se divide en presión activa y presión pasiva. El tipo de presión que se genera en un muro de contención, está relacionado con las deformaciones que se presentan en el muro. En la interacción muro - terreno, pueden ocurrir desde deformaciones prácticamente nulas hasta deformaciones que permitan que se desarrolle una falla por corte en el suelo de relleno.

Las dos teorías clásicas para la determinación de la presión lateral, pertenecen al francés Coulomb (1776) y al escocés Rankine (1857). Se decidió trabajar con la teoría de Rankine, debido a que la teoría de Coulomb no es aplicable en casos donde el muro de contención tenga un relleno estratificado.

## 2.1 TEORÍA DE RANKINE

La teoría de Rankine (1857), está basada en la hipótesis de que existe equilibrio plástico (condición en la que cada punto de una masa de suelo está a punto de fallar) y una relación entre las presiones vertical y horizontal de una masa de suelo homogénea, isotrópica y sin cohesión, contenida por un muro de contención liso, es decir, que no existe fricción entre el material contenido y la estructura de contención.

Rankine propone el uso de un material sin cohesión (un material granular) debido a que es un material que tiene un ángulo de fricción interna $\varphi$, por lo tanto, un ángulo

de falla θ, relativamente grande. Esto quiere decir que la cuña de falla es de menor tamaño, el peso del material contenido es menor y por lo tanto el muro o la estructura de contención debe ser de menores dimensiones.

Un punto de la masa de suelo, está sometido a presiones efectivas horizontales $\sigma'_h$ y verticales $\sigma'_v$, a cierta profundidad $z$. Donde el empuje horizontal $\sigma'_h$ estará dado por la Ecuación 1.

$$\sigma'_h = k\,\sigma'_v \qquad \textit{Ecuación 1}$$

La variable $k$, representa el coeficiente de presión lateral, es decir, es la relación entre el esfuerzo horizontal y el esfuerzo vertical como se muestra en la Ecuación 2.

$$k = \frac{\sigma'_h}{\sigma'_v} = \frac{\sigma'_3}{\sigma'_1} \qquad \textit{Ecuación 2}$$

## 2.1.1 PRESIÓN ACTIVA (ESTADO ACTIVO DE RANKINE)

En caso de que el muro de contención ceda ante las presiones que el suelo ejerce sobre él, el material de relleno se expandirá horizontalmente, lo que en cierto punto originará esfuerzos de corte en el suelo y la presión horizontal que ejerce el suelo de relleno sobre el muro disminuirá lentamente hasta el punto límite inferior, también conocido como presión activa de tierras.

En otras palabras, el esfuerzo principal horizontal, $\sigma_h$, disminuirá mientras que el esfuerzo vertical, $\sigma_v$, permanecerá constante. Si el desplazamiento del muro continúa, el suelo llegará hasta un punto de equilibrio plástico y la falla por corte del suelo. El ángulo del plano de falla formado está dado por la Ecuación 3 y se representa en la Figura 1. En ese momento, a la relación entre el esfuerzo horizontal y vertical se le denomina coeficiente de presión activa de Rankine, $k_a$.

$$\theta = 45° + \varphi/2$$ *Ecuación 3*

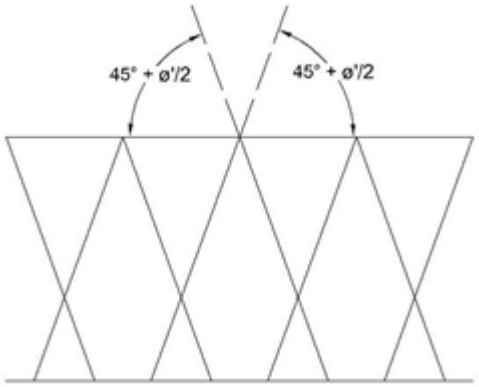



*Figura 1 Definición del ángulo de plano de falla para el caso de empuje activo (replicado de Braja M. Das, 2013).*

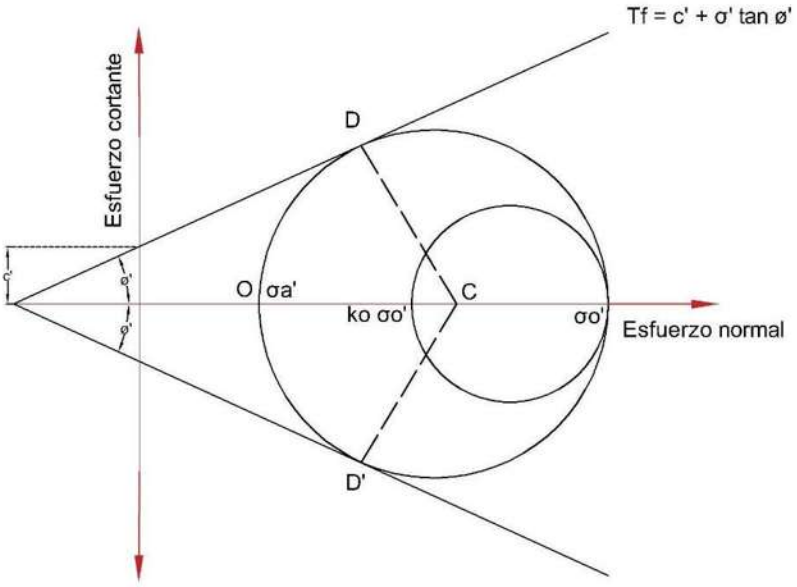



*Figura 2 Condición de presión activa representada por el círculo de Mohr (replicado de Braja M. Das, 2013).*

El ángulo de fricción interna de los materiales, depende principalmente de su grado de compactación y de su contenido de humedad, pero es difícil garantizar que el relleno detrás del muro consiste de un material definido o que su contenido de humedad permanecerá constante.

Entonces, se puede plantear una relación para el estado activo, es decir, cuando $\sigma_1 > \sigma_3$. Como se muestra en la Ecuación 4.

$$sen\ \varphi = \frac{(\sigma_1 - \sigma_3)/2}{(\sigma_1 + \sigma_3)/2} = \frac{\sigma_1 - \sigma_3}{\sigma_1 + \sigma_3} \qquad \textit{Ecuación 4}$$

Resolviendo para el esfuerzo horizontal, $\sigma_3$. Se obtiene la Ecuación 5.

$$\sigma_3 = \sigma_1 \frac{1 - sen\ \varphi'}{1 + sen\ \varphi'} \qquad \textit{Ecuación 5}$$

Y teniendo en cuenta que el esfuerzo vertical está dado por la Ecuación 6.

$$\sigma_1 = \gamma\ z \qquad \textit{Ecuación 6}$$

Donde *γ,* es el peso específico del suelo y $z$, es la profundidad a la que se está calculando la presión lateral.

Cuando los planos de los esfuerzos principales son horizontal y vertical, es decir, cuando existe un muro vertical y el relleno por encima del muro es horizontal. El coeficiente de presión lateral en el estado activo de Rankine, $K_a$, se calcula como lo indica la Ecuación 7.

$$k_a = tan^2\left(45 - \frac{\varphi'}{2}\right)$$ *Ecuación 7*

Pero, en caso de que el relleno por encima del muro tenga una inclinación, $\beta$, el coeficiente de presión lateral se calcula como lo indica la Ecuación 8.

$$k_a = \cos\beta \frac{cos\beta - \sqrt{cos^2\beta - cos^2\varphi}}{cos\beta + \sqrt{cos^2\beta - cos^2\varphi}}$$ *Ecuación 8*

Hay que tomar en cuenta que cuando el ángulo del relleno por encima del muro, $\beta$, sea menor al ángulo de fricción interna del material, $\varphi$. Se deberá calcular el coeficiente de presión activa con la Ecuación 7.

Por lo tanto, el empuje activo de Rankine se calcula como se indica en la Ecuación 9.

$$P_a = k_a\, \gamma\, z$$ *Ecuación 9*

La componente horizontal del empuje activo está dada por la Ecuación 10.

$$P_{a,h} = P_a \cos\beta$$ *Ecuación 10*

La componente vertical del empuje activo está dada por la Ecuación 11.

$$P_{a,v} = P_a\, sen\, \beta$$ *Ecuación 11*

## 2.1.2 PRESIÓN PASIVA (ESTADO PASIVO DE RANKINE)

En caso contrario a la presión activa, si el muro empuja en dirección horizontal hacia la masa de suelo contenido, el esfuerzo horizontal $\sigma'_h$, aumentará hasta llegar a un punto donde el suelo falle por corte. Como indica la Figura 3.

El estado pasivo de Rankine se presenta cuando el esfuerzo horizontal $\sigma_3$, es mayor que el esfuerzo vertical $\sigma_1$. Como se aprecia en el círculo de Mohr de la Figura 4.

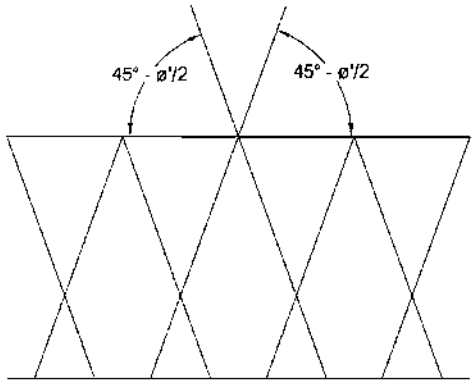



*Figura 3 Definición del ángulo de plano de falla para el caso de empuje pasivo (Braja M. Das, 2013).*

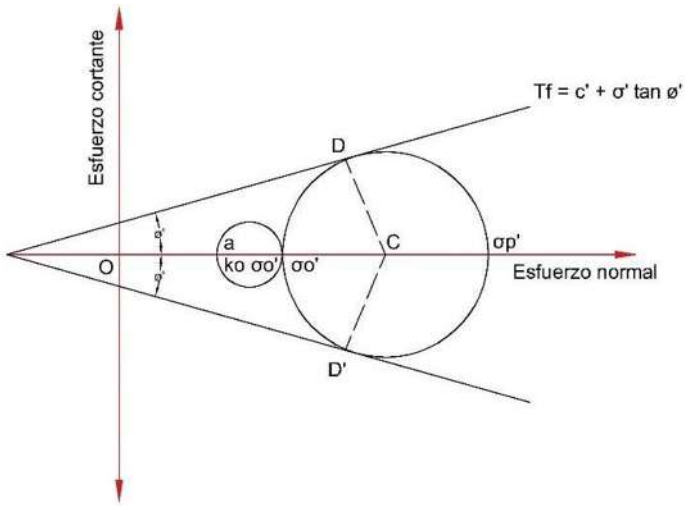



*Figura 4 Condición de presión pasiva representada por el círculo de Mohr (Braja M. Das, 2013).*

El ángulo del plano de falla generado, está dado por la Ecuación 12.

$$\theta = 45° - \varphi/2$$

*Ecuación 12*

En dirección del plano principal menor, es decir, en la dirección vertical. Entonces se puede plantear una relación para el estado pasivo como se muestra en la Ecuación 13.

$$sen\ \varphi = \frac{(\sigma_3 - \sigma_1)/2}{(\sigma_3 + \sigma_1)/2} = \frac{\sigma_3 - \sigma_1}{\sigma_3 + \sigma_1}$$

*Ecuación 13*

Resolviendo para el esfuerzo horizontal $\sigma_3$. Como se muestra en la Ecuación 14.

$$\sigma_3 = \sigma_1 \frac{1 + sen\ \varphi'}{1 - sen\ \varphi'}$$

*Ecuación 14*

Y teniendo en cuenta que el esfuerzo vertical está dado por la Ecuación 15.

$$\sigma_1 = \gamma\ z$$

*Ecuación 15*

Donde $\gamma$, es el peso específico del suelo y $z$, es la profundidad a la que se está calculando la presión lateral.

Cuando los planos de los esfuerzos principales son horizontal y vertical, es decir, cuando existe un muro de pared interna vertical y el relleno por encima del muro es horizontal. El coeficiente de presión lateral en el estado pasivo de Rankine $K_p$, se calcula como lo indica la Ecuación 16.

$$k_p = tan^2\left(45 + \frac{\varphi'}{2}\right)$$ *Ecuación 16*

Sin embargo, en caso de que el relleno por encima del muro tenga una inclinación $\beta$, el coeficiente de presión lateral se calcula como lo indica la Ecuación 17.

$$k_p = \cos\beta \frac{cos\beta + \sqrt{cos^2\beta - cos^2\varphi}}{cos\beta - \sqrt{cos^2\beta - cos^2\varphi}}$$ *Ecuación 17*

Hay que tomar en cuenta que cuando el ángulo del relleno por encima del muro $\beta$, sea menor al ángulo de fricción interna del material $\varphi$. Se deberá calcular el coeficiente de presión pasiva con la Ecuación 16.

Por lo tanto, el empuje pasivo de Rankine se calcula como se indica en la Ecuación 18.

$$P_p = k_p\, \gamma\, z$$ *Ecuación 18*

La componente horizontal del empuje pasivo está dada por la Ecuación 19.

$$P_{p,h} = P_p \cos\beta$$ *Ecuación 19*

La componente vertical del empuje pasivo está dada por la Ecuación 20.

$$P_{p,v} = P_p\, sen\, \beta$$ *Ecuación 20*

## 2.2 SUELO PARCIALMENTE SUMERGIDO

En caso de contar con suelo parcialmente sumergido, cuando el punto en el que se calcula la presión lateral a una profundidad $z$, se encuentre por debajo del nivel de aguas freáticas, se utilizará la Ecuación 21.

$$\gamma' = \gamma_{sat} - \gamma_w \qquad \textit{Ecuación 21}$$

Hay que tener en cuenta si la estructura de contención contará con drenes para de esa manera decidir si tomar en cuenta la presión intersticial.

## 2.3 EMPUJE LATERAL DE TIERRAS

Como la presión lateral aumenta linealmente con la profundidad, se calcula la presión horizontal a las profundidades donde se presentan los distintos estratos de suelo en el relleno y se grafica la presión contra la profundidad.

Se puede definir al empuje como la fuerza que actúa paralela a la estructura de contención, que se genera por la distribución de los esfuerzos horizontales. La fuerza de empuje lateral se obtiene calculando el área de la gráfica de distribución de las presiones laterales o aplicando la Ecuación 22 cuando no se trata de un relleno estratificado.

$$P_{a,p} = \frac{1}{2} k_{a,p}\, \gamma\, H^2 \qquad \textit{Ecuación 22}$$

La resultante o punto de acción del empuje, está dada por el centro de gravedad del diagrama de distribución de presiones y en rellenos homogéneos no estratificados se encuentra a 1/3 de la altura de la estructura de contención.

Conocer el punto de acción del empuje lateral, es importante para calcular los momentos que actúan sobre la estructura de contención.

Para efectos de diseño, se toma siempre un largo unitario, es decir, las unidades de la fuerza de empuje serán por ejemplo $\mathrm{Ton/m}$ o $\mathrm{KN/m}$.

## 2.3.1 PRESIÓN LATERAL EN SUELO CON RELLENO COHESIVO

Al contar con un material de relleno con cohesión, la resistencia del suelo ya no depende solamente del ángulo de fricción interna.

A partir del círculo de Mohr, se puede obtener la relación mostrada en la Ecuación 23.

$$sen\ \varphi = \frac{\frac{(\sigma_1 - \sigma_3)}{2}}{\frac{(\sigma_1 - \sigma_3)}{2} + c\ cot\ (\varphi)} = \frac{(\sigma_1 - \sigma_3)}{\sigma_1 + \sigma_3 + 2c\ cot\ (\varphi)} \qquad \textit{Ecuación 23}$$

Resolviendo para el esfuerzo horizontal $\sigma_3$, como se muestra en la Ecuación 24.

$$\sigma_3 = \sigma_1 \left(\frac{1 - sen(\varphi)}{1 + sen(\varphi)}\right) - 2c \left(\frac{cos\ (\varphi)}{1 + sen(\varphi)}\right)\left(\frac{cos\ (\varphi)}{1 + sen(\varphi)}\right) \qquad \textit{Ecuación 24}$$

Donde:

$$\left(\frac{cos\ (\varphi)}{1 + sen(\varphi)}\right) = \sqrt{\frac{1 - sen(\varphi)}{1 + sen(\varphi)}} \qquad \textit{Ecuación 25}$$

Por lo tanto, el esfuerzo horizontal $\sigma_3$, es igual a la Ecuación 26.

$$\sigma_3 = \sigma_1 \left(\frac{1 - sen(\varphi)}{1 + sen(\varphi)}\right) - 2c\sqrt{\frac{1 - sen(\varphi)}{1 + sen(\varphi)}} \qquad \textit{Ecuación 26}$$

Y recordando que el esfuerzo vertical $\sigma_1$, se calcula como se muestra en la Ecuación 27.

$$\sigma_1 = \gamma\, z \qquad \textit{Ecuación 27}$$

$$\left(\frac{1 - sen(\varphi)}{1 + sen(\varphi)}\right) = tan^2\left(45 - \frac{\varphi'}{2}\right) \qquad \textit{Ecuación 28}$$

Se puede calcular la presión lateral como se indica en la Ecuación 29.

$$\sigma_3 = \sigma_1\, tan^2\left(45 - \frac{\varphi'}{2}\right) - 2c\, tan\left(45 - \frac{\varphi'}{2}\right) \qquad \textit{Ecuación 29}$$

Y de manera simplificada, el esfuerzo horizontal $\sigma_3$, se obtiene con la Ecuación 30.

$$\sigma_3 = \gamma\, z\, k_a - 2c\sqrt{k_a} \qquad \textit{Ecuación 30}$$

Al graficar el diagrama de distribución de presiones para suelos cohesivos, es posible que se generen zonas de tensión las cuales generalmente son despreciadas para ofrecer una solución más conservadora.

### 2.3.2 AGUA EN EL RELLENO

La presencia de agua en el relleno, ya sea producto de lluvia o de infiltraciones, es indeseable y debe minimizarse por medio de obras de drenaje. Si el material de relleno es permeable, como es el caso de los materiales granulares, el agua evacúa fácilmente, pero en el caso de materiales de relleno impermeables como arcillas o limos, se debe tomar en cuenta el peso específico del material húmedo y en caso de que no sea posible drenar el agua retenida en el muro, es necesario sumar al empuje de tierras la presión hidrostática.

Otro problema que puede causar el agua en zonas con bajas temperaturas, es que el agua en el relleno se congele y empuje a la estructura de contención hacia adelante, este desplazamiento no se recupera una vez que ocurre la descongelación.

La mayoría de los problemas asociados con el agua, se pueden evitar colocando agujeros de drenaje en la base del muro o colocando tuberías para el drenaje a lo largo del muro, pero hay que prestar atención en evitar que el relleno se erosione a través de los drenajes o se obstruyan las tuberías de drenaje con material del relleno.

## 2.4 PRESIÓN LATERAL POR SOBRECARGA, LA ECUACIÓN DE BOUSSINESQ

El método de la teoría de la elasticidad, se puede utilizar para calcular el perfil de presiones laterales sobre una estructura de contención debido a distintos tipos de sobrecargas en la superficie. Dichas sobrecargas pueden ser cargas puntuales, áreas de carga con variación lineal o áreas cargadas uniformemente. Para este método se utiliza la ecuación de Boussinesq, descrita en la Ecuación 31.

$$\sigma_r = \frac{P}{2\pi z^2}\left[3\, sen^2\theta\, cos^3\theta - \frac{(1-2\mu)\, cos^2\theta}{1+cos\theta}\right] \qquad \textit{Ecuación 31}$$

En la Figura 5, se representan los términos de la ecuación.

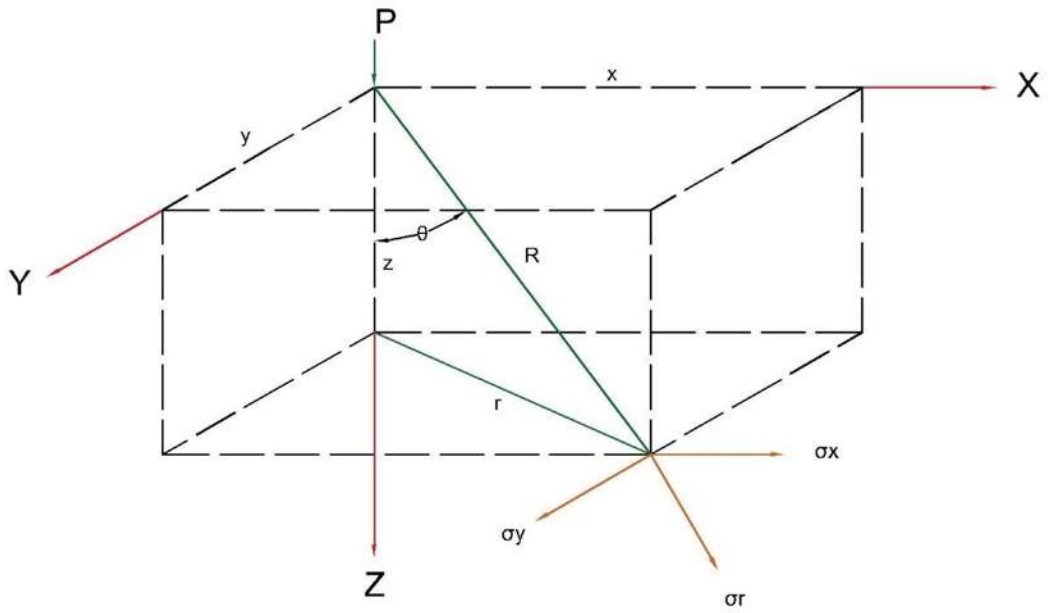



*Figura 5 Identificación de los términos utilizados en la ecuación de Boussinesq.*

$$R = \sqrt{r^2 + z^2}$$

$$r = \sqrt{x^2 + y^2}$$

La ecuación de Boussinesq, también se puede escribir como se muestra en la Ecuación 32.

$$\sigma_r = \frac{P}{2\pi}\left[\frac{3r^2 z}{R^5} - \frac{(1-2\mu)}{R(R+z)}\right] \qquad \textit{Ecuación 32}$$

Se puede analizar fácilmente un área cargada uniformemente (como un terraplén) o un área con variación lineal (como pueden ser los taludes de un terraplén). En cualquiera de esos casos, el área se divide en tiras con la misma intensidad de

carga $q$, de un ancho unitario $B$, que van del orden de 0.25 a 0.5 m. Las tiras a su vez se dividen en áreas de largo unitario $L$, que también son del orden de 0.25 a 0.5 m.

Las “áreas unitarias”, son tratadas como una serie de cargas puntuales cuya magnitud se calcula como indica la Ecuación 33 y sus resultantes actúan en el centro de dichas áreas unitarias.

$$Q = q\ B\ L \qquad \textit{Ecuación 33}$$

La sumatoria de las aportaciones de cada una de las áreas unitarias, da como resultado la presión lateral actuando en un punto definido a la profundidad de interés (ya sea en el suelo o en la estructura de contención).

La relación de Poisson es un parámetro muy importante y la teoría de la elasticidad limita la relación de Poisson a entre $-1 \leq \mu \leq 0.5$.

La relación de Poisson, es una constante que se refiere a la medida de la compresibilidad de un material, perpendicular al esfuerzo aplicado, es decir la relación entre la deformación vertical y la deformación longitudinal. Debe su nombre al matemático Simeon Poisson (1781 – 1840).

Hay que notar que una relación de Poisson positiva (+). Significa una deformación por elongación vertical y contracción lateral y cabe remarcar que ningún material utilizado en la construcción produce una relación de Poisson negativa.

### 2.4.1 EMPUJE LATERAL DE TIERRAS DEBIDO A SOBRECARGA

Una vez se calcula la presión horizontal debido a sobrecarga, en una serie de puntos a distintas profundidades, se grafican presión entre profundidad de la misma manera que para el empuje debido a la masa de suelo contenida. El empuje está definido por la distribución de los esfuerzos horizontales por lo que se puede calcular el empuje total debido a la sobrecarga calculando el área de la gráfica.

El centroide o centro de gravedad en el sentido Y de la gráfica, indica la resultante del empuje debido a sobrecarga y es útil para calcular el momento que estará actuando sobre la estructura de contención debido a esta fuerza.

## 2.5 EMPUJE LATERAL DEBIDO A SISMOS

El empuje sísmico depende de la magnitud del desplazamiento que experimenta el muro. Se considerará que se alcanza un estado activo cuando el desplazamiento del muro permita el desarrollo de resistencia al corte en el relleno.

A los muros de contención que se desplazan lo suficiente para desarrollar un estado de presión activa, se les conoce como muros que pueden ceder. Las presiones dinámicas que actúan sobre los muros de contención son estimadas mediante métodos pseudo estáticos como el método de Mononobe-Okabe. El método de Mononobe-Okabe es una extensión del método estático de Coulomb para determinar los empujes laterales.

Cuando se crea una condición de presiones activas, la aceleración sísmica $A_o$ causa un incremento del empuje activo debido al relleno del muro. Este incremento del empuje se conoce como incremento del empuje activo por sismo o $\Delta E_{ae}$. La suma del empuje activo que el relleno ejerce sobre el muro $E_a$, sumado al incremento del empuje activo por sismo $\Delta E_{ae}$, es igual al empuje activo debido al sismo $E_{ae}$.

La aceleración sísmica $A_o$, es la aceleración que se presenta en la base del muro, se expresa como una fracción de la aceleración de la gravedad y está en función de la regionalización sísmica del país. Por ejemplo, México está dividido en 4 zonas sísmicas siendo la D la más peligrosa y la A la menos peligrosa.

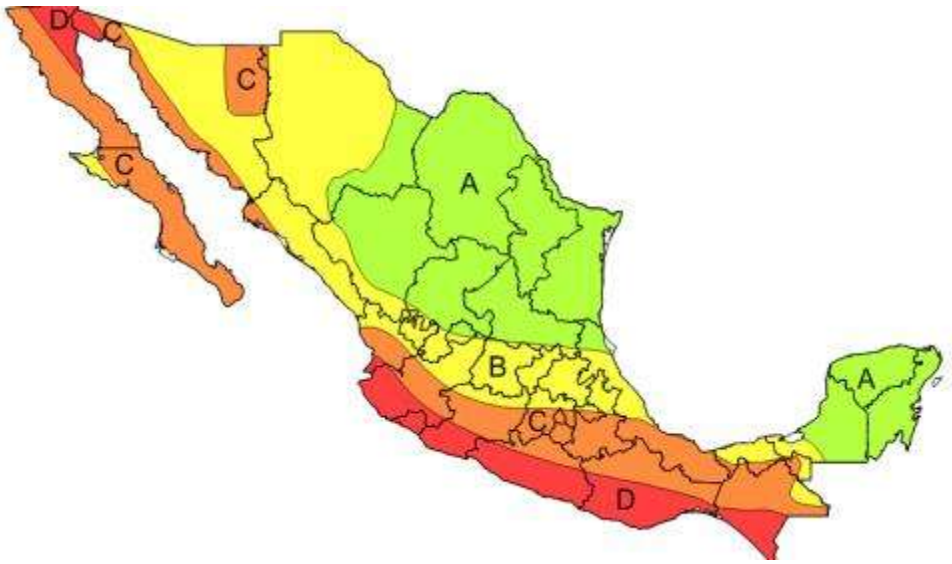



*Figura 6 Regionalización sísmica de México (Manual de diseño de obras civiles de la CFE).*

El coeficiente de presión dinámica activa $K_{ae}$, se calcula a partir de la ecuación de Mononobe-Okabe (Ecuación 34). Este coeficiente considera el efecto estático y el efecto dinámico sobre el muro, colocando la resultante de ambos empujes en la misma ubicación.

$$K_{ae} = \frac{cos^2(\varphi - \theta - \psi)}{\cos\psi \;\; cos^2\theta \;\; \cos(\delta + \theta + \psi)\left[1 + \sqrt{\frac{sen(\delta + \varphi)\, sen(\varphi - \beta - \psi)}{cos(\delta + \theta + \psi)\, sen(\beta - \theta)}}\right]^2} \qquad \textit{Ecuación 34}$$

Donde:

$\delta$, es el ángulo de fricción entre el material de relleno y la estructura de contención.

$\theta$, es ángulo de la cara interna del muro respecto a la vertical (Si el muro no está inclinado, $\theta = 0$).

φ, es el ángulo de fricción interna del material de relleno contenido.

β, es el ángulo que tiene el relleno por encima del muro de contención.

$\psi$, es el ángulo del sismo y se obtiene mediante la Ecuación 35.

$$\psi = \arctan\left(\frac{K_h}{1 - K_v}\right) \qquad \textit{Ecuación 35}$$

Donde:

$K_h$, es el coeficiente sísmico horizontal y se calcula como se indica en la Ecuación 36.

$$K_h = \frac{2}{3} A_o \qquad \textit{Ecuación 36}$$

Y $K_v$, es el coeficiente sísmico vertical calculado como se indica en la Ecuación 37.

$$K_v = \frac{2}{3} K_h \qquad \textit{Ecuación 37}$$

A pesar de que el método de Coulomb implica que la resultante del empuje lateral se encuentra a una altura de $\frac{H}{3}$ m a partir de la base del muro, resultados experimentales del método de Mononobe-Okabe demuestran que la resultante en realidad se encuentra más elevada. Por lo que el empuje lateral debido a sismo $E_{ae}$, se puede dividir en una componente estática $E_a$ y una componente dinámica $\Delta E_{ae}$.

Por lo que el empuje total debido a sismo, es igual a la Ecuación 38.

$$E_{ae} = E_a + \Delta E_{ae}$$ *Ecuación 38*

Como se menciona anteriormente, la componente estática del empuje total debido a sismo, se encuentra a una altura de $\frac{H}{3}$ a partir de la base del muro. Mientras que diversos resultados experimentales, demuestran que la componente dinámica actúa a una altura de aproximadamente $0.6\,H$, a partir de la base del muro. Por lo que la resultante del empuje total debido a sismo se calcula como se muestra en la Ecuación 39.

$$h_e = \frac{E_a\left(\frac{H}{3}\right) + \Delta E_{ae}(0.6\,H)}{E_{ae}}$$ *Ecuación 39*

Conociendo el empuje y la resultante de la condición pseudoestática, se calcula la componente horizontal del momento de sismo $M_e$. Como se muestra en la Ecuación 40.

$$M_e = E_{ae}\ h_e\ \cos(\delta)$$ *Ecuación 40*

# 3 MUROS DE CONTENCIÓN

## 3.1 INTRODUCCIÓN

Los muros de contención, son estructuras cuyo propósito es contener al suelo con una pendiente que no mantendría por sí mismo naturalmente (casi vertical o vertical). Por lo tanto, su función se puede explicar cómo resistir las fuerzas ejercidas por el suelo contenido, evitando que este se deslice debido a la gravedad.

Los muros de contención, se pueden clasificar de acuerdo a cómo producen la estabilidad, pudiendo ser:

a) Muros de gravedad.
b) Muros en voladizo.
c) Muros con contrafuertes.

El diseño de los muros de contención, implica que sean resistentes al deslizamiento, al vuelco y que la capacidad de carga del terreno sea suficiente para soportar al muro.

## 3.2 TIPOS DE MUROS DE CONTENCIÓN

La elección del tipo de muro de contención, depende principalmente de lo siguiente:

- Superficie con la que se cuente para la construcción.
- Presupuestos.
- Altura y longitud de la masa de suelo que se debe contener.
- Colindancias.

### 3.2.1 MUROS DE GRAVEDAD

Los muros de gravedad, son el tipo de muro de contención más antiguo que existe y suelen ser la opción más económica tratándose de alturas menores a 5 m. Como lo indica su nombre, los muros de gravedad dependen de su propio peso para lograr la estabilidad.

Este tipo de muros, generalmente son construidos de concreto simple o mampostería y debido a su grosor, desarrollan muy poca o nula tensión, por lo que no es necesario reforzarlos con acero.

La dimensión de la base de este tipo de muros, oscila entre 0.4 a 0.7 veces la altura. Por economía, la base debe ser lo más angosta posible, pero debe ser lo suficientemente ancha para proporcionar estabilidad contra el volteo y deslizamiento, además de para aplicar presiones al suelo debajo de su base no mayores que las máximas permisibles.

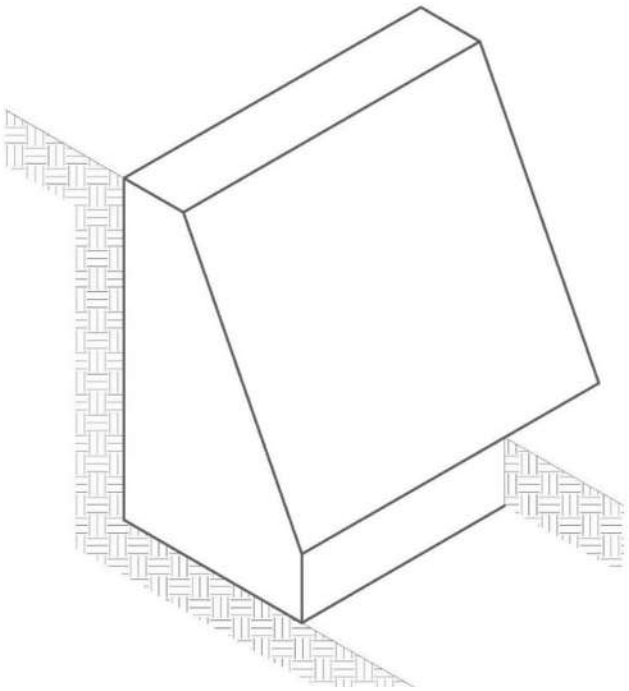


*Figura 7 Diagrama de muro de contención de gravedad.*

### 3.2.2 MUROS EN VOLADIZO

Los muros en voladizo, están hechos de concreto reforzado. Son formados por una pantalla delgada empotrada en una zapata o base. Los muros en voladizo son eficientes y económicos hasta una altura de 10 m.

La geometría más utilizada es en forma de “T”, cuya estabilidad está en función del ancho de su zapata y se ve aumentada por el peso de la masa de suelo que se encuentra sobre el talón del muro, que ayuda para impedir el volcamiento, además de volver más pesada la estructura, lo que ayuda para evitar el deslizamiento.

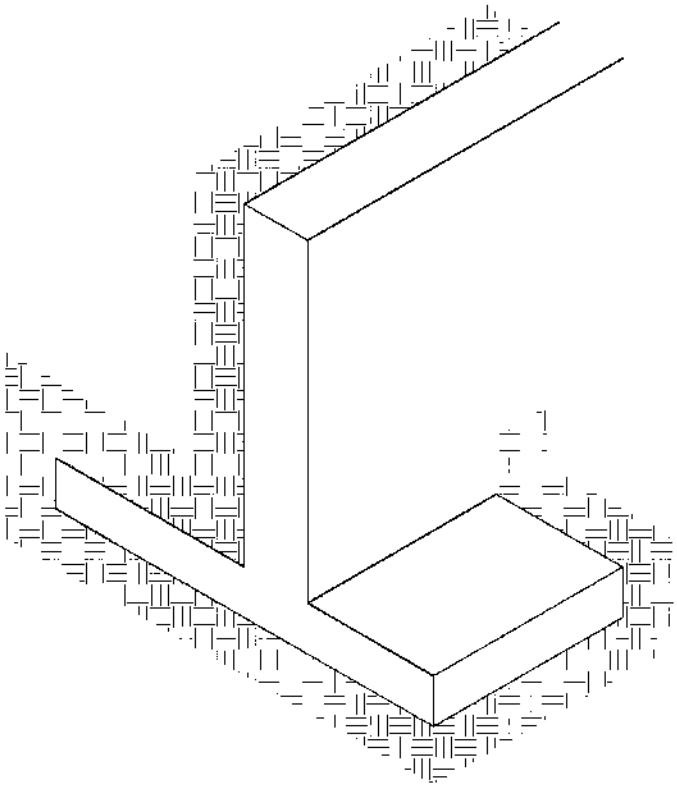


*Figura 8 Diagrama de muro de contención en voladizo.*

# 3.3 PREDIMENSIONAMIENTO

## 3.3.1 DIMENSIONES PRELIMINARES

Para el diseño de muros de contención, se deben suponer unas dimensiones preliminares, con las que se calcula la estabilidad del muro. En caso de producir resultados no deseados, se modifican las dimensiones y se calcula nuevamente la estabilidad. Se trata de un proceso iterativo el cual se puede optimizar por medio de programas de cómputo como es el caso de esta tesis.

Las dimensiones mínimas recomendadas, varían de acuerdo con la bibliografía consultada. En este caso las dimensiones preliminares propuestas para muros de gravedad se indican en la Figura 9.

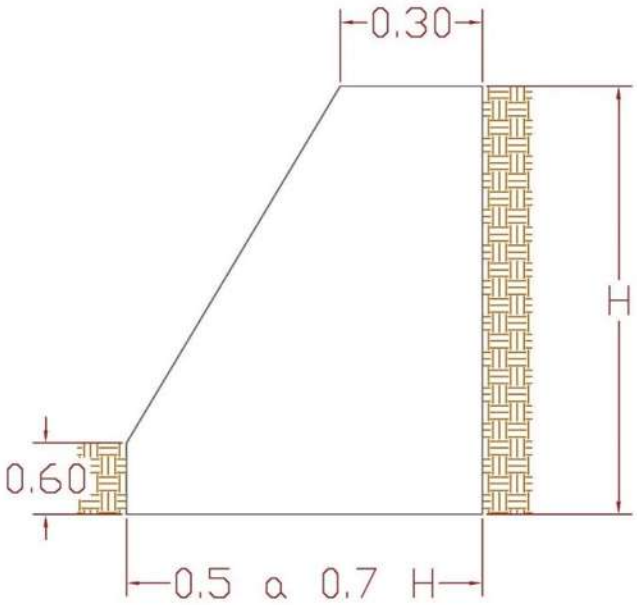



*Figura 9 Dimensiones preliminares para muros de contención de gravedad.*

La corona, debe tener mínimo 30 cm de espesor, para permitir la correcta colocación del concreto hidráulico.

La base del muro, por lo general mide entre 0.5 y 0.7 veces la altura, pero debe ser lo más angosta posible por economía.

La profundidad de emplazamiento, debe ser mínimo de 60 cm.

Las dimensiones preliminares recomendadas para muros de contención en voladizo, se muestran en la Figura 10.

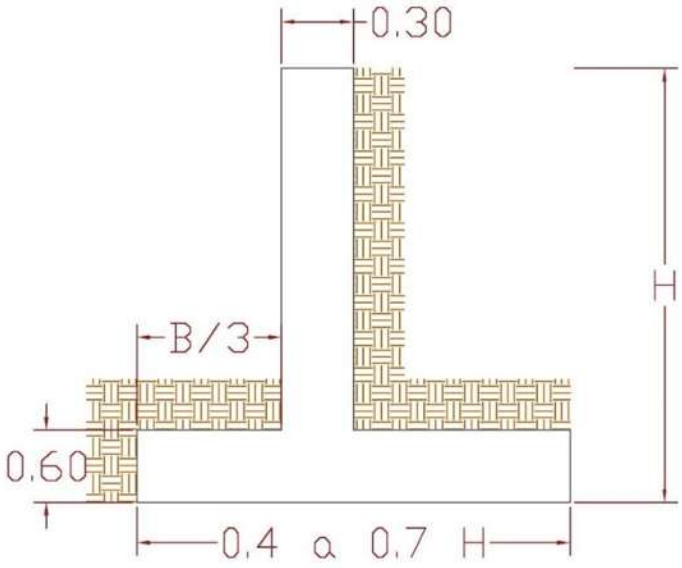



*Figura 10 Dimensiones preliminares para muros de contención en voladizo.*

La corona, debe tener mínimo 30 cm de espesor para permitir la correcta colocación del concreto hidráulico.

La dimensión de la base se propone entre 0.4 y 0.7 veces la altura, mientras que sea suficiente para garantizar la estabilidad del muro.

La profundidad de emplazamiento, debe ser mínimo de 60 cm.

El ancho del pie, se propone de un tercio de la base del muro.

### 3.3.2 REVISIÓN DEL PRE DIMENSIONAMIENTO

Una vez se ha realizado el pre dimensionamiento, se procede a revisar que las dimensiones propuestas cumplan con los factores de seguridad. La revisión del pre dimensionamiento de un muro de contención comprende lo siguiente:

- Evitar el volteo en la punta del muro.
- Evitar el deslizamiento a lo largo de la base del muro.
- Verificar que las presiones sobre el terreno no excedan la capacidad de carga máxima admisible.

En caso de que los factores de seguridad contra volteo y contra deslizamiento no cumplan o que la presión ejercida al terreno por la base del muro exceda la capacidad de carga máxima admisible, se procede a modificar las dimensiones del muro y verificar la estabilidad nuevamente.

#### 3.3.2.1 CÁLCULO DEL FACTOR DE SEGURIDAD CONTRA VOLTEO

El factor de seguridad contra el volteo, se obtiene dividiendo el momento resistente o estabilizante entre el momento de volteo. Para que se pueda considerar al muro de contención seguro contra el volteo, el factor de seguridad contra volteo, $FSV$, debe ser mayor o igual a 1.5, cuando el suelo presente en campo sea un suelo granular y mayor o igual a 2, cuando se trate de un suelo cohesivo. Como se indica en la Ecuación 41.

$$FSV = \frac{M_R}{M_v} \geq 1.5 \; o \; 2.0 \qquad \textit{Ecuación 41}$$

El momento resistente $M_R$, es la suma de los momentos provocados por las fuerzas que evitan el volteo, como lo son, el peso de una longitud unitaria de muro y en el caso de muros en voladizo, la masa de suelo que se encuentra sobre el talón del muro también aporta al momento resistente.

El momento de volteo $M_V$, es la suma de los momentos provocados por las fuerzas de empuje lateral debido al suelo contenido, a las sobrecargas y al incremento de presiones debido a sismo, que tienden a provocar el volteo en la punta del muro de contención.

### 3.3.2.2 CÁLCULO DEL FACTOR DE SEGURIDAD CONTRA DESLIZAMIENTO.

El factor de seguridad contra deslizamiento, se obtiene dividiendo las fuerzas que se oponen al deslizamiento a lo largo de la base del muro, entre las fuerzas que provocan el deslizamiento, como se muestra en la Ecuación 42. Para que se pueda considerar un muro de contención como resistente contra el deslizamiento, el factor de seguridad contra el deslizamiento $FSD$ debe ser mayor o igual a 1.5.

$$FSD = \frac{\mu \ W_{muro} + E_p}{E_a} \geq 1.5 \qquad \textit{Ecuación 42}$$

Donde:

$\mu$, es el coeficiente de fricción entre el material del muro y el suelo debajo del mismo.

$E_p$, es la fuerza del empuje pasivo generado por el suelo en la punta del muro

$E_a$, son las fuerzas del empuje activo generadas por el suelo contenido y las sobrecargas.

### 3.3.2.3 CÁLCULO DEL FACTOR DE SEGURIDAD POR CAPACIDAD DE CARGA DEL SUELO

El suelo debe ser capaz de soportar las presiones transmitidas por la estructura de contención construida encima, sin presentar falla por cortante o asentamientos imprevistos.

Las magnitudes de las presiones transmitidas al suelo, dependen de las cargas aplicadas y de las dimensiones de la base del muro. Si las presiones transmitidas son lo suficientemente grandes, los esfuerzos cortantes inducidos pueden superar la resistencia al cortante de los suelos dando como resultado una falla por capacidad de carga.

La capacidad de carga última del suelo, $q_u$, dividida entre la presión máxima transmitida al suelo por el muro de contención $q_{max} = q_{punta}$. (Ver Figura 11). Debe ser mayor o igual a un factor de seguridad de 3.

Note que $q_{punta}$ y $q_{talón}$, corresponden a la presión máxima y mínima transmitidas al suelo, estas se producen en el extremo de la punta y del talón respectivamente como se muestra en la Figura 11.

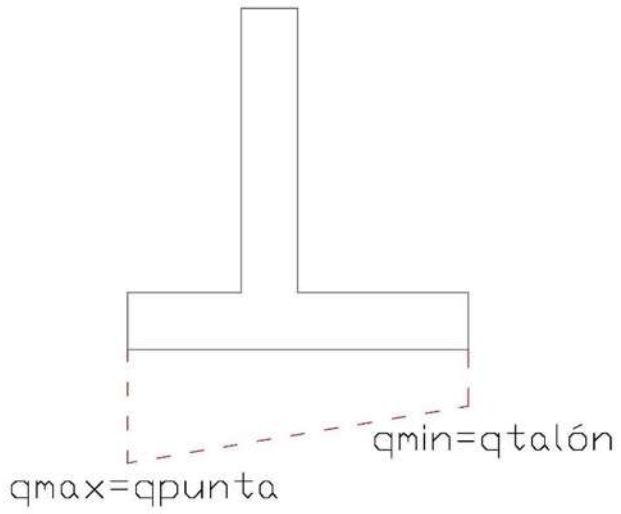



*Figura 11 Presiones transmitidas al suelo por el muro de contención.*

Las magnitudes de $q_{punta}$ y $q_{talón}$, pueden calcularse de la siguiente manera:

La resultante $R$ de las fuerzas, es igual a la suma de los pesos $(\Sigma V)$, que actúan sobre la base del muro (El peso del muro y el peso del relleno sobre el talón del

muro), más la componente horizontal de las presiones debidas al suelo contenido ($P_a$) y a otras sobrecargas como indica la Ecuación 43.

$$R = \Sigma V + P_a \cos\beta \qquad \textit{Ecuación 43}$$

Donde $\beta$, es el ángulo del relleno por encima del muro.

El momento neto generado por las fuerzas mencionadas anteriormente, actuando sobre el extremo de la punta del muro, se calcula con la Ecuación 44.

$$M_{neto} = \Sigma MR - MV \qquad \textit{Ecuación 44}$$

Donde $MR$, es el momento resistente y $MV$, el momento de volteo.

La distancia $x$, entre el punto de aplicación de las fuerzas y el extremo de la punta del muro, se expresa como en la Ecuación 45.

$$x = \frac{M_{neto}}{\Sigma V} \qquad \textit{Ecuación 45}$$

Por lo tanto, la excentricidad $e$, de la resultante $R$, se puede expresar como se indica en la Ecuación 46.

$$e = \frac{B}{2} - x \qquad \textit{Ecuación 46}$$

Donde $B$, es la base del muro.

Para calcular la presión máxima en la punta del muro, se utiliza la Ecuación 47.

$$q_{max} = \frac{\Sigma V}{B}\left(1 + \frac{6e}{B}\right)$$ *Ecuación 47*

Del mismo modo, para la presión mínima en el talón del muro, se utiliza la Ecuación 48.

$$q_{min} = \frac{\Sigma V}{B}\left(1 - \frac{6e}{B}\right)$$ *Ecuación 48*

Es importante remarcar que cuando la excentricidad $e \geq B/6$, $q_{min}$ se vuelve negativa representando un esfuerzo de tensión en el extremo de la sección del talón. Dicha tensión es indeseable debido a que el suelo no posee resistencia a la tensión, por lo que, de presentarse este caso, será necesario aumentar las dimensiones del muro en la sección de la punta en muros en voladizo o aumentar la base en muros de gravedad.

Por otro lado, para obtener la capacidad de carga última del suelo, se deben analizar las relaciones entre las cargas, las dimensiones de la base del muro y las propiedades del suelo donde está desplantado el muro de contención.

Existen numerosas teorías para obtener la capacidad de carga última, pero en este caso se eligió la teoría de Terzaghi modificada utilizando los factores de capacidad de carga de Vesić (Coduto, 2016). Estos factores de capacidad de carga son muy utilizados para el caso de cimentaciones donde se presentan momentos. Por otro lado, a pesar de que esta teoría es más compleja, ofrece resultados de capacidad de carga más precisos debido a que toma en cuenta un rango más amplio de condiciones de carga y de geometría del suelo.

La teoría de capacidad de carga de Vesić toma en cuenta los siguientes factores:

a) Factores de capacidad de carga de Vesić:

$$N_q = e^{\pi\, tan\varphi} tan^2\left(45 + \frac{\varphi}{2}\right)$$ *Ecuación 49*

$$N_c = \frac{N_q - 1}{tan\varphi}\ (para\ \varphi > 0)$$

$$N_c = 5.14\ (para\ \varphi = 0)$$ *Ecuación 50*

$$N_\gamma = 2\left(N_q + 1\right)tan\varphi$$ *Ecuación 51*

b) Factores de forma:

$$S_c = \ 1 + \left(\frac{B}{L}\right)\left(\frac{N_q}{N_c}\right)$$ *Ecuación 52*

$$S_q = \ 1 + \left(\frac{B}{L}\right)tan\varphi'$$ *Ecuación 53*

$$S_\gamma = \ 1 - 0.4\left(\frac{B}{L}\right)$$ *Ecuación 54*

Hay que tener en cuenta que, para estructuras corridas, como es el caso de los muros de contención, la relación $B/L$ es pequeña, por lo que $S_c, S_q\ y\ S_\gamma$ tienen un valor muy cercano a 1 por lo que estos factores pueden ignorarse.

c) Factores de profundidad:

$$d_c = 1 + 0.4k$$ *Ecuación 55*

$$d_q = 1 + 2k\ tan\varphi\ (1 - sen\varphi)^2$$ *Ecuación 56*

$$d_\gamma = 1$$ *Ecuación 57*

Para profundidades de desplante relativamente pequeñas $(D/B \leq 1)$ como es el caso de los muros de contención, $k = tan^{-1}(D/B)$, donde la tangente debe calcularse en radianes.

d) Factores de carga inclinada:

Los factores de carga inclinada se utilizan cuando existen cargas que no actúan de manera perpendicular a la base del muro. La variable P, se refiere a la componente de la carga que actúa de manera perpendicular a la base del muro, la variable V, se refiere a la componente que actúa de manera paralela a la base, la variable c, es la cohesión efectiva del suelo y A es el área de la base del muro.

$$i_c = 1 - \frac{mV}{A\ c\ N_c} \geq 0 \qquad \textit{Ecuación 58}$$

$$i_q = \left[1 - \frac{V}{P + \frac{A\ c}{tan\varphi}}\right]^m \geq 0 \qquad \textit{Ecuación 59}$$

$$i_\gamma = \left[1 - \frac{V}{P + \frac{A\ c}{tan\varphi}}\right]^{m+1} \geq 0 \qquad \textit{Ecuación 60}$$

Donde:

$$m = \frac{2 + \frac{B}{L}}{1 + \frac{B}{L}} \qquad \textit{Ecuación 61}$$

Si la carga actúa perpendicular a la base del muro o el ángulo de fricción interna del suelo $\varphi = 0$, los factores de inclinación de carga son iguales a 1 y pueden ser ignorados.

e) Factores de inclinación en la base:

La mayoría de los muros de contención son construidos con una base horizontal, sin embargo, si la carga aplicada tiene una inclinación muy grande a partir de la vertical, puede ser buena idea inclinar la base del muro de contención al mismo ángulo, como se muestra en la Figura 12. Para que las fuerzas actúen de manera perpendicular.

$$b_c = 1 - \frac{\alpha}{147°} \qquad \textit{Ecuación 62}$$

$$b_q = b_\gamma = \left(1 - \frac{\alpha\ tan\varphi}{57°}\right)^2 \qquad \textit{Ecuación 63}$$

De cualquier modo, si la base del muro de contención es horizontal (el caso habitual), los factores de inclinación en la base son igual a 1 y pueden ser ignorados.

f) Factores de inclinación del suelo:

Los muros de contención ubicados cerca de la parte superior de un talud. Como se muestra en la Figura 12. Tienen una capacidad de carga menor que los ubicados a nivel de suelo. Para tomar en cuenta esto, Vesić agregó los factores de inclinación del suelo.

$$g_c = 1 - \frac{\beta}{147°} \qquad \textit{Ecuación 64}$$

$$g_q = g_\gamma = (1 - tan\beta)^2$$ *Ecuación 65*

Si la superficie de suelo se encuentra nivelada ($\beta = 0$), los factores por inclinación deben ser ignorados.

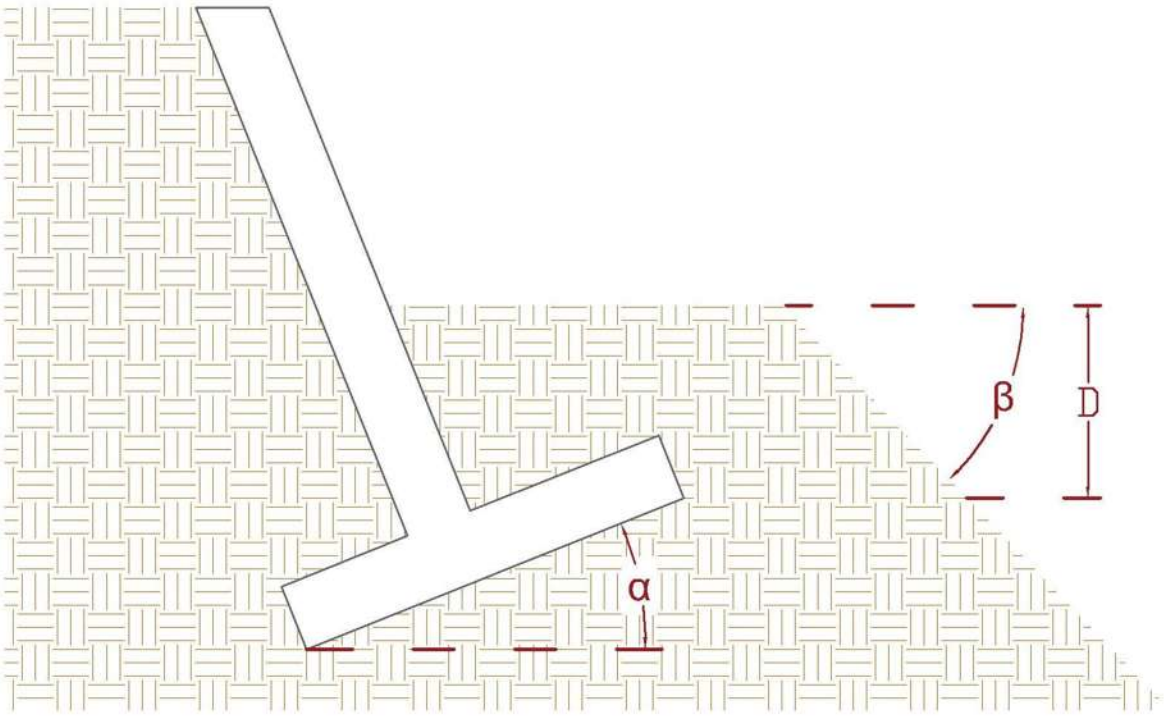



*Figura 12 Diagrama de los ángulos de inclinación para los factores de inclinación del suelo e inclinación de la base.*

Una vez obtenidos todos los factores tomados en cuenta por la teoría de Vesić, se procede a aplicar la Ecuación 66 para obtener la capacidad de carga última del suelo.

$$q_u = c'N_c S_c d_c i_c b_c g_c + \sigma' D N_q S_q d_q i_q b_q g_q + 0.5\gamma' B N_\gamma S_\gamma d_\gamma i_\gamma b_\gamma g_\gamma$$ *Ecuación 66*

Por último, se compara la capacidad de carga última obtenida $q_u$, con la presión máxima transmitida al suelo por el muro $q_{max}$, para obtener el factor de seguridad por capacidad de carga, el cual deberá ser mayor o igual a 3. Como se indica en la Ecuación 67.

$$FS_{capacidad\ de\ carga} = \frac{q_u}{q_{max}} \geq 3 \qquad \textit{Ecuación 67}$$

#### 3.3.2.3.1 INFLUENCIA DEL NIVEL DE AGUAS FREÁTICAS EN LA CAPACIDAD DE CARGA

Si existe la suficiente agua en el suelo como para que se desarrolle el nivel freático y el nivel freático se encuentra en el rango de la zona potencial de falla por cortante. La capacidad de carga unitaria nominal se verá reducida.

Se debe considerar este efecto para el peor caso (el nivel freático más superficial), que podría esperarse durante la vida útil del muro de contención.

En la Figura 13, se muestran los 3 casos que pueden describir la ubicación del nivel freático en condiciones de campo.

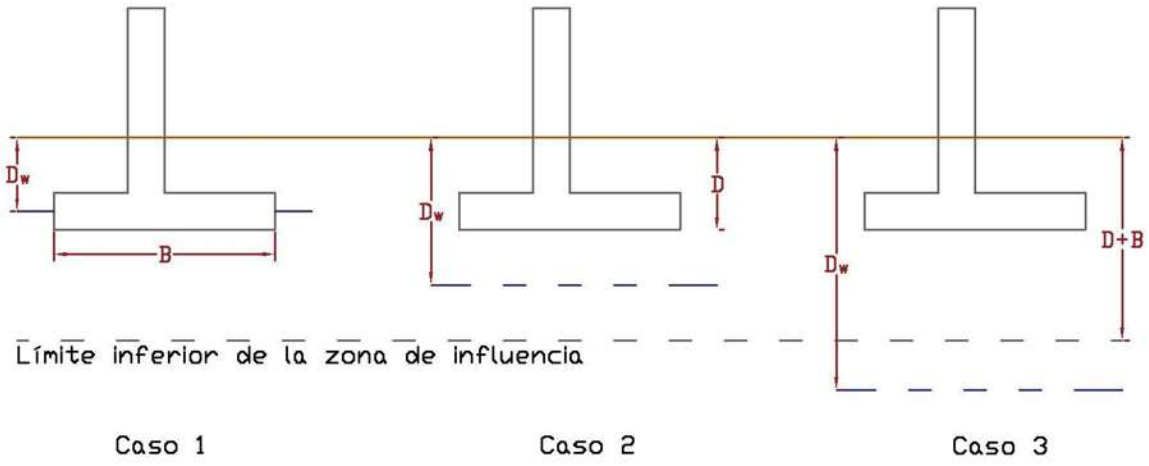



*Figura 13 Casos de la ubicación del nivel freático respecto a la zona de influencia y la profundidad de desplante. (Replicado de Foundation design, principles and practices, Dondald P. Coduto, 2016).*

Se toma en cuenta el cambio en las presiones efectivas a lo largo de la superficie de falla, ajustando el peso específico efectivo $\gamma'$, en el tercer término de la ecuación de Vesić.

Dependiendo del caso que se presente, el peso específico efectivo $\gamma'$ se calcula como se indica en la Ecuación 68, Ecuación 69 o Ecuación 70.

- Caso 1 ($D_w \leq D$):

$$\gamma' = \gamma - \gamma_w \qquad \textit{Ecuación 68}$$

- Caso 2 ($D < D_w < D + B$):

$$\gamma' = \gamma - \gamma_w \left(1 - \frac{D_w - D}{B}\right) \qquad \textit{Ecuación 69}$$

- Caso 3 ($D + B \leq D_w$); No es necesario hacer correcciones:

$$\gamma' = \gamma \qquad \textit{Ecuación 70}$$

## 3.4 DISEÑO DE MUROS DE CONTENCIÓN EN VOLADIZO COMO ESTRUCTURA DE CONCRETO ARMADO

El diseño del acero de refuerzo para muros de contención en voladizo se realiza apegado a las Normas Técnicas Complementarias para el Diseño y Construcción de Estructuras de Concreto, edición 2017. La filosofía de diseño a la que obedece el reglamento se le conoce como factores de carga y resistencia.

El diseño se realiza por separado para cada uno de los elementos que componen al muro de contención, siendo estos: la punta, el talón y la pantalla. Los elementos se analizan como si se tratara de vigas en voladizo y cada uno está sometido a distintas fuerzas cortantes y momentos flexionantes.

La pantalla está sometida a las fuerzas de empuje debido al relleno contenido que, al multiplicarse por su brazo de palanca resulta un momento flexionante que origina tensiones en la cara interna de la pantalla.

La punta está sometida a la distribución de presiones de reacción que ejerce el suelo, además de a su peso propio, pero este es despreciable. El momento flexionante origina tensión en la cara inferior de la punta.

El talón está sometido al peso del material de relleno que se encuentra sobre el mismo, sumado al peso propio del talón, puede además estar sometido a la variación de presiones de reacción que ejerce el suelo sobre la base del muro, aunque el efecto de estas presiones es despreciable. En cualquier caso, el momento flector origina tensión en la cara superior del talón.

En la Figura 14, se muestran las partes del muro de contención y las acciones sobre los mismos.

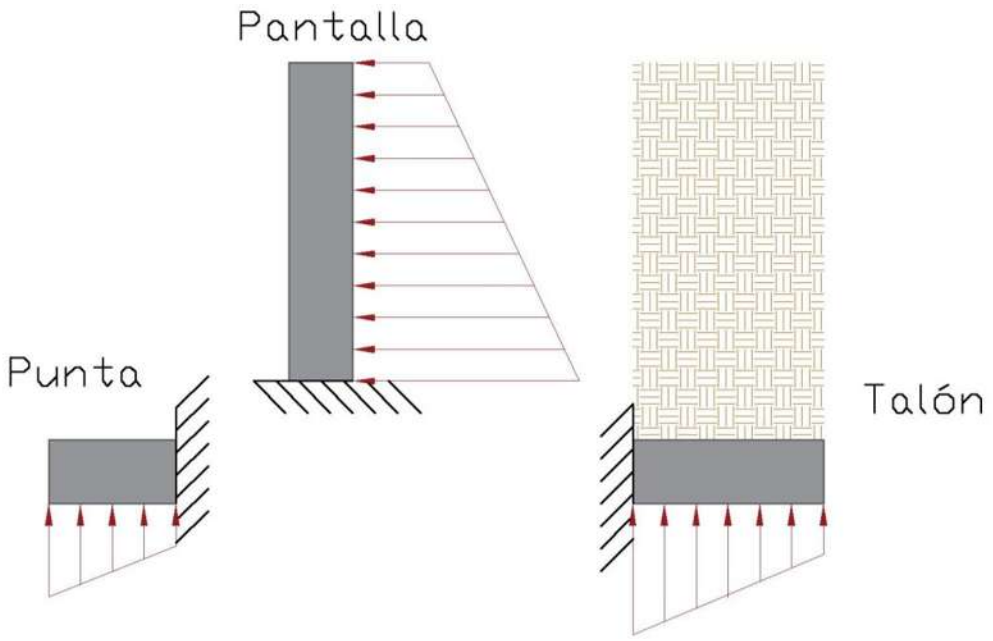



*Figura 14 Partes del muro de contención y fuerzas que actúan sobre ellos.*

### 3.4.1 MÉTODO DE DISEÑO POR RESISTENCIA DE ACUERDO A LA NORMA NTC-17

El fin último de este análisis es determinar el área de acero necesaria para que el elemento que se está diseñando sea resistente al momento último y la sección de concreto sea resistente a la fuerza cortante última.

Se considera que las varillas de refuerzo a tensión estarán trabajando en su punto de cedencia antes de que falle el concreto en el lado de compresión del elemento.

La distribución de esfuerzos de compresión en el concreto, cuando se alcanza la resistencia de la sección, es uniforme y toma un valor de $f_c''$ igual a $0.85\, f_c'$ hasta una profundidad de la zona de compresión igual a $\beta_1\, c$. Como se muestra en los diagramas de esfuerzo y deformación de la Figura 15 y donde, $\beta_1$, se calcula como se indica en la Ecuación 71.

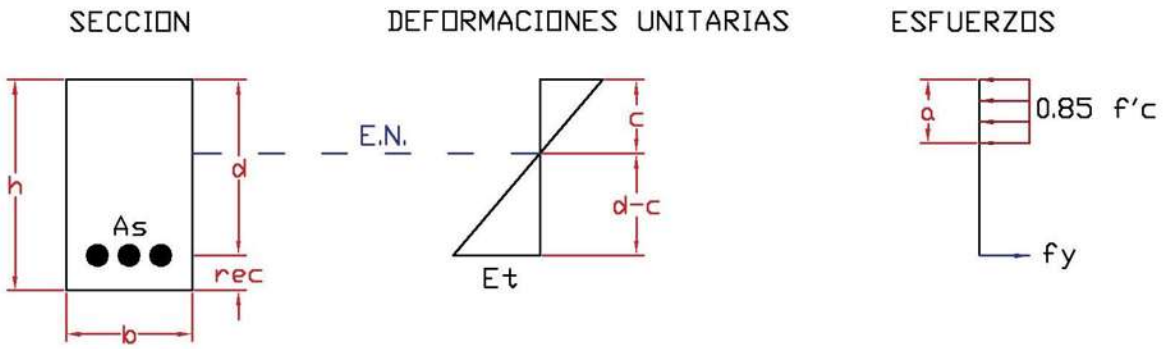



*Figura 15 Diagrama esfuerzo-deformación de una viga sujeta a flexión.*

$$\beta_1 = 0.85, \quad si\ f_c' \leq 280\,\frac{kg}{cm^2}$$

$$\beta_1 = 1.05 - \frac{f_c'}{1400} \geq 0.65, \quad si\ f_c' > 280\,\frac{kg}{cm^2} \qquad \textit{Ecuación 71}$$

En el apartado 5.1.3 de las NTC-17 se menciona que el momento resistente $M_R$ se calcula como se muestra en la Ecuación 72.

$$M_R = F_R\, b\, d^2\, f_c''\, q\, (1 - 0.5\, q) \qquad \textit{Ecuación 72}$$

Donde, de acuerdo con las NTC sobre Criterios y Acciones para el Diseño Estructural de las Edificaciones, las resistencias deben afectarse por un factor de reducción, siendo este factor $F_R = 0.9$ para flexión.

La variable $q$, que es el índice de refuerzo a tensión, se calcula como se indica en la Ecuación 73.

$$q = 1 - \sqrt{1 - \frac{2\ M_u}{F_R\ 0.85\ f_c'\ b\ d^2}} \qquad \textit{Ecuación 73}$$

La variable $\rho$, es la cuantía de acero de refuerzo longitudinal trabajando a tensión y se calcula como se indica en la Ecuación 74.

$$\rho = q\ \frac{0.85\ f_c'}{f_y} \qquad \textit{Ecuación 74}$$

#### 3.4.1.1 DETERMINACIÓN DEL ACERO EN TENSIÓN

Consiste en determinar el acero necesario para que un elemento de secciones definidas, sea capaz de resistir el momento último de diseño $M_u$.

En el apartado 5.1.4 de las NTC-17 indica que se debe comprobar que la cuantía de acero de refuerzo trabajando a tensión no sea menor que la calculada con la Ecuación 75.

$$\rho_{min} = \frac{0.7\sqrt{f_c'}}{f_y}\ bd \qquad \textit{Ecuación 75}$$

En caso de que la cuantía de acero calculada con la Ecuación 74, sea menor que la cuantía de acero mínimo $\rho_{min}$, calculada con la Ecuación 75. Se deberá utilizar la cuantía de acero mínimo $\rho_{min}$.

El área de acero requerida $A_s$, se calculará entonces mediante la Ecuación 76.

$$A_s = \rho\, b\, d$$ *Ecuación 76*

Conociendo el área de acero requerida y el diámetro de la varilla propuesta, se puede determinar el número de varillas necesarias para que el momento resistente $M_R$, sea mayor o igual que el momento último $M_u$.

La separación entre las varillas de refuerzo, se obtiene dividiendo la base del elemento (en este caso siempre será de 100 cm debido a que analizamos al muro de contención por franjas de 1 m de longitud), entre el número de varillas necesarias.

### 3.4.1.2 RESISTENCIA A FUERZA CORTANTE

Se debe comprobar que la sección crítica sea resistente a la fuerza cortante, es decir, que la fuerza cortante que resiste el concreto, sea mayor o igual que la fuerza cortante última. La revisión de la fuerza cortante resistente se realiza de acuerdo al apartado 5.3.3 de las NTC-17.

El apartado 5.3.3.1a de las NTC-17, menciona que, en elementos con relación claro a peralte total, $L/h$, no menor que 5, la fuerza cortante que toma el concreto se calcula mediante la Ecuación 77 o la Ecuación 78.

Si $\rho$ < 0.015:

$$V_c = F_R(0.2 + 20\rho)\sqrt{f_c'}\, b\, d$$ *Ecuación 77*

Si $\rho \geq 0.015$:

$$V_c = F_R\, 0.5\, \sqrt{f_c'}\, b\, d$$ *Ecuación 78*

Mientras que, si la relación $L/h$, es menor que 4, el cortante que recibe el concreto se calcula con la Ecuación 79.

$$V_c = F_R \left(3.5 - 2.5 \frac{M}{V\ d}\right) 0.5\sqrt{f_c'}\ b\ d \qquad \textit{Ecuación 79}$$

Donde:

$$\left(3.5 - 2.5 \frac{M}{V\ d}\right) > 1$$

$M$ y $V$, es el momento flexionante y la fuerza cortante que actúan sobre la sección respectivamente.

Sin embargo, la fuerza cortante que recibe el concreto nunca debe ser mayor que la calculada mediante la Ecuación 80.

$$V_c \leq F_R\ 1.5\ \sqrt{f_c'}\ b\ d \qquad \textit{Ecuación 80}$$

El apartado 5.3.3.1b de las NTC-17, menciona que, en elementos anchos en los que se cumpla que el ancho $b$, no sea menor que 4 veces el peralte efectivo $d$, el espesor del elemento no sea menor a 600 mm y la relación $M/V\ d$, no exceda de 2.0, el cortante que toma el concreto se calculará mediante la Ecuación 78 independientemente de la cuantía de acero. En caso de que el espesor del elemento, sea mayor de 600 mm y la relación $M/V\ d$, exceda de 2.0, la resistencia a la fuerza cortante se determinará como indica el apartado 5.3.3.1a, es decir, con la Ecuación 77 o la Ecuación 78.

Del mismo modo, de acuerdo con las NTC sobre Criterios y Acciones para el Diseño Estructural de las Edificaciones, las resistencias deben afectarse por un factor de reducción, siendo este factor $F_R = 0.75$ para cortante.

### 3.4.1.3 LONGITUD DE DESARROLLO

De acuerdo al apartado 6.1.2 de las NTC-17, para que el acero de refuerzo sea capaz de desarrollar tensión, este debe prolongarse una longitud de desarrollo base,

$l_{db}$, más allá de la sección crítica. Esta distancia se calcula como se indica en la Ecuación 81.

$$l_{db} = \frac{a_{var}\ f_y}{3(c + K_{tr})\sqrt{f_c'}} \geq 0.11\frac{d_b\ f_y}{\sqrt{f_c'}} \qquad \textit{Ecuación 81}$$

Donde:

$a_{var}$, es el área nominal de la varilla en cm$^2$.

$d_b$, es el diámetro nominal de la varilla en cm.

$c$, es el recubrimiento utilizado en cm.

$K_{tr}$, es el índice de refuerzo transversal que se supondrá igual a 0 por sencillez en el diseño.

La longitud de desarrollo $l_{db}$, en ningún caso deberá ser menos a 30 cm.

### 3.4.1.4 ACERO REQUERIDO POR CONTRACCIÓN Y TEMPERATURA

Es necesario armar los elementos con acero transversal con el fin de controlar las deformaciones provocadas por contracción y temperatura.

De acuerdo al apartado 6.7 de las NTC-17, por sencillez del cálculo, puede suministrarse un refuerzo mínimo con cuantía igual a $\rho = 0.003$, tratándose de elementos estructurales expuestos a la intemperie o en contacto con el terreno natural. Por lo que el área de acero requerida se calcula con la Ecuación 82.

$$A_{sc} = \rho\ b\ h \qquad \textit{Ecuación 82}$$

Si el elemento estructural tiene un espesor menor a 15 cm, el refuerzo se coloca en una sola capa. De lo contrario, el refuerzo debe colocarse en dos capas próximas a las caras del elemento.

### 3.4.1.5 FACTORES DE CARGA

Para poder determinar la resistencia de diseño, se deben multiplicar las acciones actuantes en los elementos por los factores de carga indicados en el apartado 3.4 de las Normas técnicas complementarias sobre criterios y acciones para el diseño estructural de las edificaciones. Una vez contamos con las cargas factorizadas, se deben obtener los elementos mecánicos, como fuerza cortante y momento flexionante.

En este caso, las presiones se analizarán como cargas vivas y serán multiplicadas por un factor de carga de 1.5, mientras que las cargas muertas o peso propio de los elementos, será multiplicado por un factor de carga de 1.3.

### 3.4.1.6 CONSTANTES DE DISEÑO

El diseño de cada elemento, parte de definir las constantes de diseño. Algunas de estas son propuestas por el calculista mientras que otras son determinadas por la geometría de los elementos del muro de contención.

- **Resistencia a la compresión del concreto ($f'_c$):** debe ser propuesto por el calculista. Se expresa en $kg/cm^2$ o $MPa$.
- **Esfuerzo de fluencia del acero ($f_y$):** el valor más utilizado es de 4200 $kg/cm^2$ o 420 $MPa$, que corresponde al acero de alta resistencia.
- **Base del elemento ($b$):** en el caso de muros de contención toma un valor de 100 $cm$ debido a que el análisis se realiza en franjas de 1 m de muro.

- **Espesor del elemento ($h$)**: determinado por la geometría de la sección crítica.
- **Diámetro de varilla por número de designación ($\phi$):** propuesto por el calculista, en la Tabla 1, se observan los diámetros comerciales más comunes.
- **Recubrimiento libre:** Depende del tipo de exposición que tendrá el concreto.

*Tabla 1 Diámetros comerciales de varilla (MX-C-407-ONNCCE-2001 Varilla Corrugada).*

| Número de designación | Masa nominal en kg/m | Dimensiones nominales | | |
|---|---|---|---|---|
| | | Diámetro en mm | Área de la sección transversal en $mm^2$ | Perímetro en mm |
| 2,5 | 0,388 | 7,9 | 49 | 24,8 |
| 3 | 0,560 | 9,5 | 71 | 29,8 |
| 4 | 0,994 | 12,7 | 127 | 39,9 |
| 5 | 1,552 | 15,9 | 198 | 50,0 |
| 6 | 2,235 | 19,0 | 285 | 60,0 |
| 7 | 3,042 | 22,2 | 388 | 69,7 |
| 8 | 3,973 | 25,4 | 507 | 79,8 |
| 9 | 5,033 | 28,6 | 642 | 89,8 |
| 10 | 6,225 | 31,8 | 794 | 99,9 |
| 11 | 7,503 | 34,9 | 957 | 109,8 |
| 12 | 8,938 | 38,1 | 1140 | 119,7 |
| 14 | 12,147 | 44,5 | 1552 | 139,6 |
| 16 | 15,89 | 50,8 | 2026 | 159,6 |

Respecto al recubrimiento libre del acero de refuerzo, el apartado 4.9.3 de las NTC-17 menciona que, para elementos en contacto directo con el terreno, el recubrimiento libre no será menor a 75 milímetros.

# 4 PROGRAMACIÓN Y DESARROLLO DE LA HERRAMIENTA COMPUTACIONAL DE CÁLCULO

## 4.1 PLANTEAMIENTO Y ANÁLISIS DEL PROBLEMA

El desarrollo de una herramienta de cálculo, capaz de automatizar el proceso de diseño completo de un muro de contención, ofrece una alternativa de solución a realizar el proceso de diseño de forma manual, debido a que se trata de un proceso largo e iterativo.

En la actualidad existe software especializado capaz de realizar el cálculo referente a la presión lateral de tierra ejercida sobre el muro (Por ejemplo, LateralK de NovoTech). Pero dicho programa no toma en cuenta variables adicionales que se presentan en campo, como pueden ser, un relleno estratificado, material de relleno cohesivo o presiones debido a diferentes tipos de sobrecarga.

También existen aplicaciones útiles para el dimensionamiento y la revisión de la estabilidad de muros de contención, pero toman como variables de entrada los empujes a los que va a estar sometido, por lo que el usuario debe calcularlos previamente.

Además, una licencia comercial anual de alguno de estos softwares tiene un costo del orden de 200 dólares, un costo difícil de afrontar para profesionales que recién se integran al mercado laboral.

En conclusión, se busca crear una alternativa completa, funcional, fácil de usar y accesible a un problema complejo pero computable.

### 4.1.1 DATOS DE ENTRADA Y DE SALIDA

La herramienta de cálculo necesita información para poder ejecutarse y resolver el problema. Se denomina como datos de entrada a aquellos que permiten introducir información al programa desde el exterior. Los datos de entrada que necesita el programa, se muestran en la Tabla 2.

*Tabla 2 Datos de entrada*

| DATOS DE ENTRADA | | |
|---|---|---|
| DATOS | DATA TYPE | DESCRIPCIÓN |
| HTWALL | float | Altura del muro (m) |
| sobrecraga | float | Magnitud de la sobrecarga (kN) |
| γms_1 | float | Peso volumétrico del relleno contenido (kN/m$^3$) |
| Hs_1 | float | Espesor del estrato (m) |
| φ_1 | float | Ángulo de fricción interna del relleno (°) |
| c_1 | float | Cohesion del material de relleno (kPa) |
| β | float | Ángulo del relleno por encima del muro (°) |
| NAF | float | Profundidad del NAF (m) |
| Hay_NAF | dict | Indica la presencia o aunsencia del NAF |
| γm_p1 | float | Peso volumétrico del relleno al frente (kN/m$^3$) |
| Hs_p1 | float | Espesor del estrato al frente (m) |
| φ_p1 | float | Ángulo de fricción interna del relleno al frente (°) |
| c_p1 | float | Cohesion del material de relleno al frente (kPa) |
| A0 | float | Coeficiente sísmico |
| tipo_de_sobrecarga | dict | Indica el tipo de sobrecarga |
| B | float | Ancho del área cargada (m) |
| W | float | Largo del área cargada (m) |
| q | float | Magnitud de la sobrecarga distribuida (kPa) |
| μ | float | Coeficiente de Poisson |
| NSQW | float | Divisiones del área cargada a lo ancho |
| NSQL | float | Divisiones del área cargada a lo largo |
| DTWALL | float | Distancia entre el muro y la sobrecarga |
| NVERT | float | Intervalos de la altura a los que se calcula la presión |
| P | float | Magnitud de la carga puntual (kN) |
| tipo_muro | dict | Define el tipo de muro |
| γ_muro | float | Peso específico del material del muro (kN/m$^3$) |
| coef_deslizam | float | Coeficiente de fricció entre el muro y el suelo |
| α | float | Inclinación de la base del muro |
| NAF_CC | float | Profundidad del NAF debajo del muro (m) |

| | | |
|---|---|---|
| fc | float | Resistencia del concreto a compresión (kg/cm$^2$) |
| fy | float | Esfuerzo de fluencia del acero (kg/cm$^2$) |
| E | float | Módulo de elasticidad del concreto |
| Φ | float | Factor de reducción de Flexión |
| rec_libre | float | Recubrimiento libre de la varilla (cm) |
| var_prop_num | float | Diámetro de la varilla (in) |

Una vez que la información ha sido procesada, la herramienta de cálculo arroja datos de salida para mostrar al usuario los resultados obtenidos. Los datos de salida, arrojados por la herramienta de cálculo se muestran en la Tabla 3.

*Tabla 3 Datos de salida*

| DATOS DE SALIDA | | |
|---|---|---|
| DATOS | DATA TYPE | DESCRIPCIÓN |
| Empuje_activo | float | Magnitud del empuje activo (kN) |
| Resultante_EA | float | Ubicación de la resultante empuje activo (m) |
| Presion_pasiva | float | Magnitud del empuje pasivo (kN) |
| Resultante_EP | float | Ubicación de la resultante empuje pasivo (m) |
| ΔEa | float | Magnitud del incremento del EA por sismo (kN) |
| resultante_sismo | float | Ubicación del incremento del EA (m) |
| Empuje_SAC | float | Magnitud del empuje debido a sobrecarga AC (kN) |
| Resultante_SAC | float | Ubicación del empuje debido a sobrecarga AC (m) |
| Empuje_SLL | float | Magnitud del empuje debido a sobrecarga LL (kN) |
| Resultante_SLL | float | Ubicación del empuje debido a sobrecarga LL (m) |
| Empuje_SP | float | Magnitud del empuje debido a sobrecarga P (kN) |
| Resultante_SP | float | Ubicación del empuje debido a sobrecarga P (m) |
| Empuje_total | float | Magnitud del empuje total sobre el muro (kN) |
| brazo_palanca | float | Ubicación de el empuje total sobre el muro (m) |
| FSV | float | Factor de seguridad por volteo |
| FSD | float | Factor de seguridad por deslizamiento |
| FSCC | float | Factor de seguridad por capacidad de carga |
| qn | float | Capacidad de carga última del terreno (kPa) |
| var_necesarias_punta | float | Cantidad de varillas necesarias en la punta |
| var_propuesta_num | float | Diámetro de la varilla propuesta (in) |
| long_var_punta | float | Longitud necesaria para la varilla de la punta (cm) |
| separacion_punta | float | Separación entre las varillas de la punta (cm) |
| Mu_punta | float | Momento último actuante en la punta (t.m) |
| Mn_punta | float | Momento resistente de la punta (t.m) |

| | | |
|---|---|---|
| Vu_punta | float | Cortante último en la punta (t) |
| Vc_punta | float | Cortante resistente en la punta (t) |
| var_necesarias_talon | float | Cantidad de varillas necesarias en el talón |
| long_var_talon | float | Longitud necesaria para la varilla del talón (cm) |
| separacion_talon | float | Separación entre las varillas del talón (cm) |
| Mu_talon | float | Momento último actuante en el talón (t.m) |
| Mn_talon | float | Momento resistente del talón (t.m) |
| Vu_talon | float | Cortante último en el talón (t) |
| Vc_talon | float | Cortante resistente en el talón (t) |
| var_necesarias_pantalla | float | Cantidad de varillas necesarias en la pantalla |
| long_var_pantalla | float | Longitud necesaria para la varilla de la pantalla (cm) |
| separacion_pantalla | float | Separación entre las varillas de la pantalla (cm) |
| Mu_pantalla | float | Momento último actuante en la pantalla (t.m) |
| Mn_pantalla | float | Momento resistente de la pantalla (t.m) |
| Vu_pantalla | float | Cortante último en la pantalla (t) |
| Vc_pantalla | float | Cortante resistente en la pantalla (t) |
| cant_var_base | float | Varillas necesarias por temperatura en la base |
| s_base | float | Separación de varillas por temp en la base (cm) |
| Long_temp_base | float | Longitud de varilla por temp en la base (cm) |
| cant_var_pan_hor | float | Varillas necesarias por temperatura en la pantalla h |
| s_pantalla_hor | float | Separación de varillas por temp en la pantalla h (cm) |
| Long_temp_pan_h | float | Longitud de varilla por temp en la pantalla h (cm) |
| cant_var_pan_ver | float | Varillas necesarias por temperatura en la pantalla v |
| s_pantalla_ver | float | Separación de varillas por temp en la pantalla v (cm) |
| Long_temp_pan_v | float | Longitud de varilla por temp en la pantalla v (cm) |

### 4.1.2 ESTRUCTURA Y PROCESOS PRINCIPALES DE CÁLCULO

La herramienta de cálculo, se programó basada en el paradigma de programación orientada a objetos. La programación orientada a objetos se basa en el concepto de clases y objetos, sus principales ventajas son que el código escrito bajo este paradigma es reutilizable, organizado y fácil de corregir o mantener.

La programación orientada a objetos estructura un programa en clases, las clases son plantillas simples y reutilizables, útiles para crear instancias de objetos. Los objetos tienen una serie de atributos o propiedades y comportamientos, que son implementados mediante métodos (funciones).

Para la programación de la herramienta de cálculo, se hizo uso de las siguientes librerías:

- **Numpy:** Es una librería que permite gestionar fácilmente matrices de n dimensiones. Además de contener una amplia variedad de funciones matemáticas complejas para operar entre las matrices.
- **Matplotlib:** Es una librería útil para la generación de gráficas a partir de datos contenidos en arrays de Python o Numpy**.**
- **Shapely:** Es una librería de Python para la creación, manipulación y análisis de objetos geométricos planos.
- **Libdibujo:** Es una librería de dibujo tipo CAD, útil para aplicaciones de Ingeniería Civil. Está basada en Matplotlib y fue programada por el Dr. Carlos Chávez Negrete.

La herramienta de cálculo consta de dos clases principales, la clase $Empuje\_muros$ y la clase $Muros$.

Al ser llamada la clase $Empuje\_muros$, crea un objeto al que se le asignan como atributos las variables de entrada y mediante una serie de métodos, calcula la magnitud del empuje que se ejercerá sobre el muro debido a distintos factores, además del empuje total. Los métodos que contiene esta clase son los siguientes:

- $presion\_activa\_Rankine$: Calcula la magnitud del empuje debido a la presión activa del relleno contenido y la ubicación de su resultante.
- $presion\_pasiva\_Rankine$: Calcula la magnitud del empuje debido a la presión pasiva debido al suelo al frente del muro y la ubicación de su resultante.
- $presion\_activa\_Okabe$: Calcula el incremento de la presión activa debido a sismo por el método de Mononobe-Okabe y la ubicación de su resultante.
- $prelat\_sobrecarga\_areacargada1$: Calcula el empuje debido a una sobrecarga con la forma de un área cargada de magnitud constante.

- $prelat\_sobrecarga\_linearload$: Calcula el empuje debido a una sobrecarga con la forma de un área cargada con variación lineal ascendente o descendente.
- $Prelat\_sobrecarga\_cargapuntual$: Calcula el empuje debido a una sobrecarga con la forma de carga puntual.
- $suma$: Calcula la suma de todos los empujes laterales existentes por medio de una interpolación entre los diagramas de distribución de presiones.
- $graficar$: Grafica cada uno de los diagramas de distribución de presiones laterales.

Al ser llamada la clase $Muros$, crea un objeto al que se le asignan como atributos las variables de salida de la clase $Empuje\_muros$ y mediante una serie de métodos, dimensiona un muro para resistir los empujes ejercidos, calcula su estabilidad, calcula la capacidad de carga y en caso de tratarse de un muro en voladizo, diseña el acero de refuerzo. Los métodos que contiene esta clase son los siguientes:

- $predim\_gravedad$: Realiza el dimensionamiento de un muro de contención de gravedad, partiendo de las dimensiones mínimas recomendadas e iterando hasta que las dimensiones del muro garanticen seguridad contra el volcamiento, el deslizamiento y calcula la capacidad de carga última del terreno por el método de Vesić.
- $predim\_voladizo$: Realiza el dimensionamiento de un muro de contención en voladizo, partiendo de las dimensiones mínimas recomendadas e iterando hasta que las dimensiones del muro garanticen seguridad contra el volcamiento, el deslizamiento y calcula la capacidad de carga última del terreno por el método de Vesić.
- $graficar\_muro$: Realiza el diagrama del muro de contención diseñado.
- $graficar\_CC$: Realiza el diagrama de la distribución de presiones que el muro de contención ejerce sobre el suelo.
- $diseno\_acero$: Calcula el acero de refuerzo necesario para resistir el cortante y momento último en muros en voladizo.

- $acero\_contraccion$: Calcula el acero de refuerzo necesario por contracción y temperatura para cada uno de los elementos del muro de contención.
- $dibujar\_acero$: Dibuja el diagrama del armado del muro en voladizo con el acero necesario y las dimensiones finales.

### 4.1.3 PRESENTACIÓN DE RESULTADOS

Para mostrar los resultados al usuario, se programó una GUI mediante la librería Ipywidgets que hace uso de widgets tipo FloatText para desplegar los datos de salida tipo Float y widgets tipo Output para presentar las figuras generadas por varios métodos mencionados anteriormente utilizando la librería Matplotlib.

Además, se automatizó la generación de un reporte de resultados utilizando una plantilla en Markdown almacenada en Google drive a la que el programa accede y reemplaza placeholders por los resultados obtenidos en las variables de salida y por las figuras generadas por los métodos, las cuales son guardadas automáticamente en Google Drive. Un ejemplo del reporte generado se puede consultar en el apartado 9.8 de anexos.

### 4.1.4 CODIFICACIÓN

Para la codificación del programa de cálculo, se aplicó la siguiente metodología:

1. Planteamiento del problema.
2. Formulación del algoritmo.
3. Codificación.
4. Diseño de la GUI.
5. Ejecución.
6. Verificación de resultados.

El programa se programó en Python 3.6, utilizando como IDE Google Colaboratory y el código completo está listado en el Anexo. En el apartado 5 de esta tesis, se realiza la validación del programa mediante la resolución de casos de verificación.

## 4.1.5 DIAGRAMA DE FLUJO

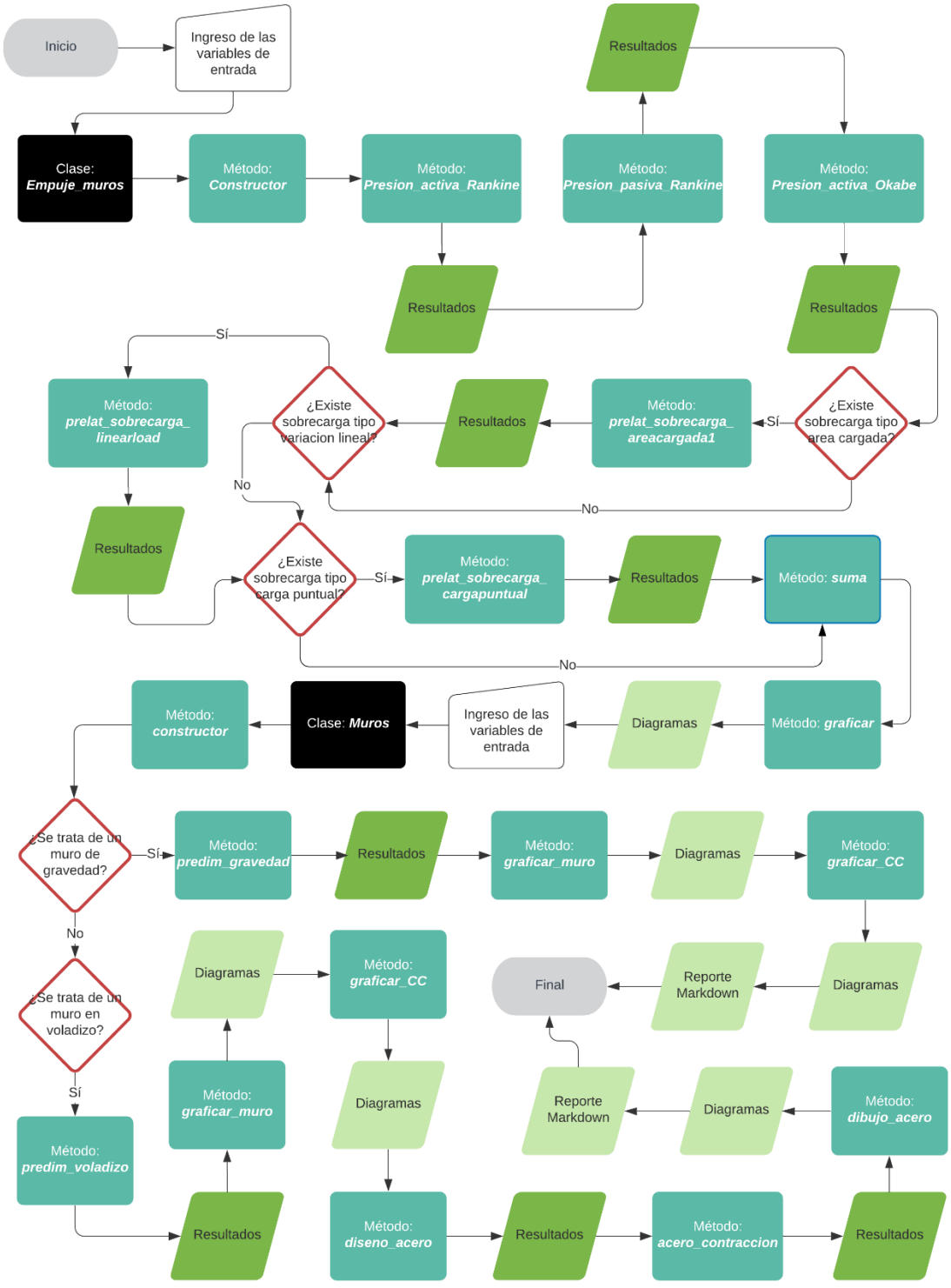



*Figura 16 Diagrama de flujo de la herramienta de cálculo.*

# 4.2 DOCUMENTACIÓN (MANUAL DE USUARIO)

1. **Archivos necesarios para el funcionamiento y conexión con Google Drive.**

Para el funcionamiento del programa, es necesario tener en Google Drive la carpeta “Tesis_python” que contiene las librerías y plantillas en archivos Markdown necesarias para generar el reporte de resultados.

Una vez se ha copiado a Google Drive la carpeta, hay que montar el cuaderno de Google Colaboratory a una cuenta de Google Drive, abriendo una celda de código y ejecutando las líneas de código que aparecen a continuación.

```
from google.colab import drive
drive.mount('/content/drive')
```

2. **Conectar el cuaderno y ejecutar todas las celdas de código.**

3. **Datos de entrada referentes a las características del relleno.**

En la pestaña DATOS EMPUJES, indicada en la captura de la Figura 17, aparecen 4 conjuntos para la entrada de datos y la sobrecarga.

En el apartado Presión activa de Rankine, llenar los datos referentes a las características del relleno, la herramienta de cálculo puede calcular empujes generados por relleno estratificado en hasta tres estratos. En caso de que el suelo esté dividido en menos de tres estratos diferentes, llenar los espacios de los demás estratos con ceros.

En el apartado Presión pasiva de Rankine, se deben llenar las celdas referentes a las características del terreno al frente del muro.

En el apartado Presión debido a sismo, se deben ingresar en la celda la magnitud de la aceleración sísmica en la zona.

En el apartado Presión debido a sobrecarga, en caso de existir, se deben llenar en las celdas las características y magnitud de la sobrecarga.

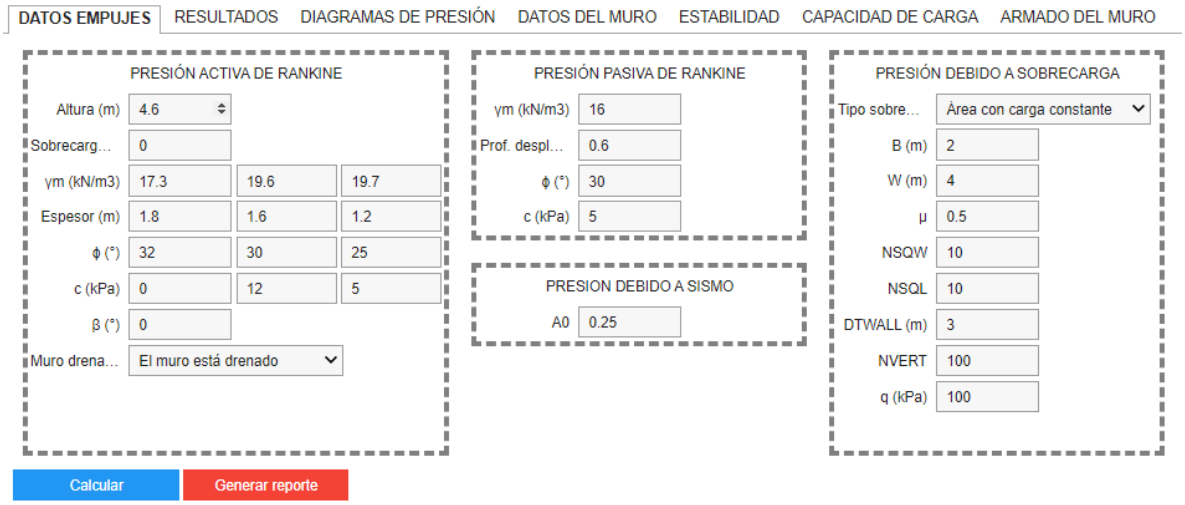



*Figura 17 DATOS EMPUJES. Manual de usuario.*

Una vez ingresados los datos, hacer clic en el botón “Calcular” para generar los resultados.

## 4. Observación y análisis de los resultados de los empujes actuantes sobre el muro.

En la pestaña RESULTADOS, se pueden observar la magnitud y resultante de todos los empujes actuantes sobre el muro, además del empuje total. Como aparece en la captura de la Figura 18.

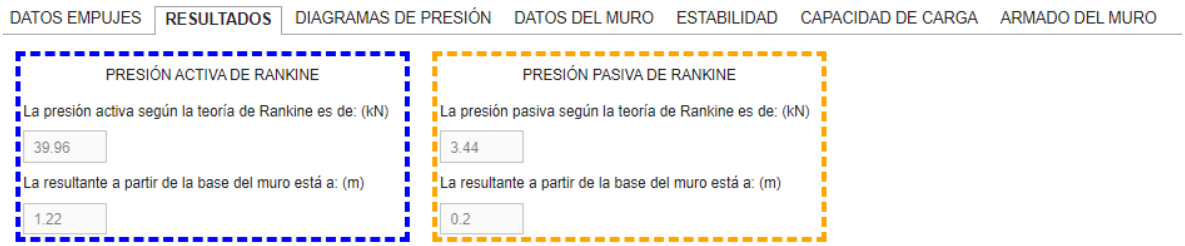



*Figura 18 RESULTADOS. Manual de usuario.*

En la pestaña DIAGRAMAS PRESIÓN, se pueden observar los diagramas de distribución de presiones de cada empuje. Como aparece en la captura de la Figura 19.

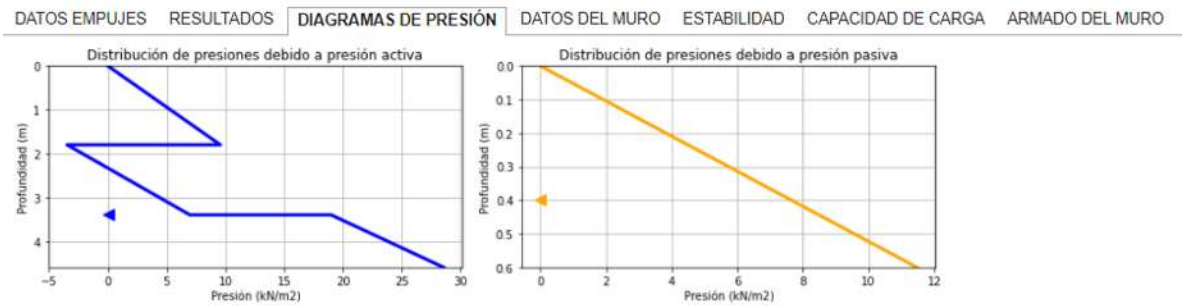



*Figura 19 DIAGRAMAS DE PRESIÓN. Manual de usuario.*

## 5. Datos de entrada referentes a las características del muro.

En la pestaña DATOS DEL MURO, indicada en la captura de la Figura 20, se deben ingresar los datos referentes a las características del muro. En caso de seleccionar muro en voladizo, se desplegará un menú en el que se deberán ingresar los datos referentes a los materiales del muro, necesarios para diseñar el acero de refuerzo.

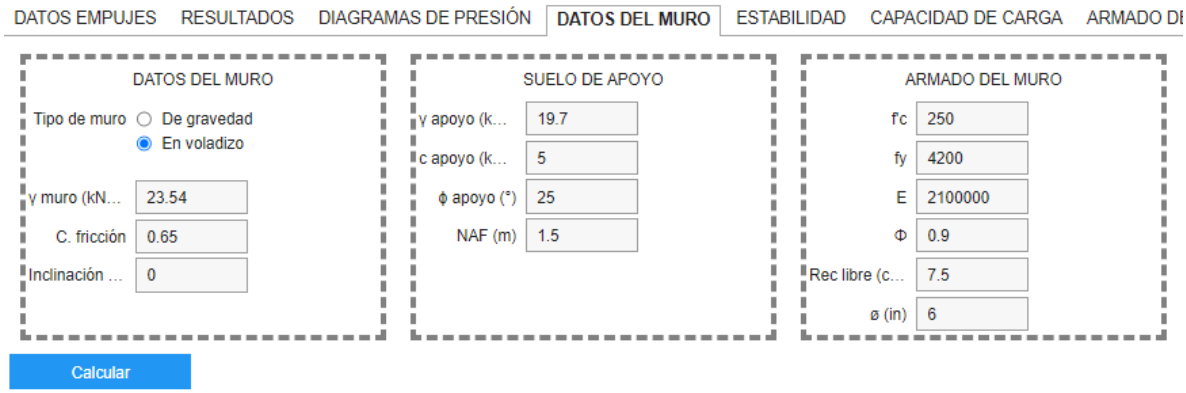



*Figura 20 DATOS DEL MURO. Manual de usuario.*

Para generar los resultados, hacer clic en el botón “calcular”.

## 6. Observación y análisis de los resultados de estabilidad, capacidad de carga y armado del muro.

En la pestaña ESTABILIDAD, se muestran los resultados de los factores de seguridad contra volcamiento y contra deslizamiento, además de un diagrama con las dimensiones mínimas necesarias por el muro para garantizar su estabilidad. Como se muestra en la captura de la Figura 21.

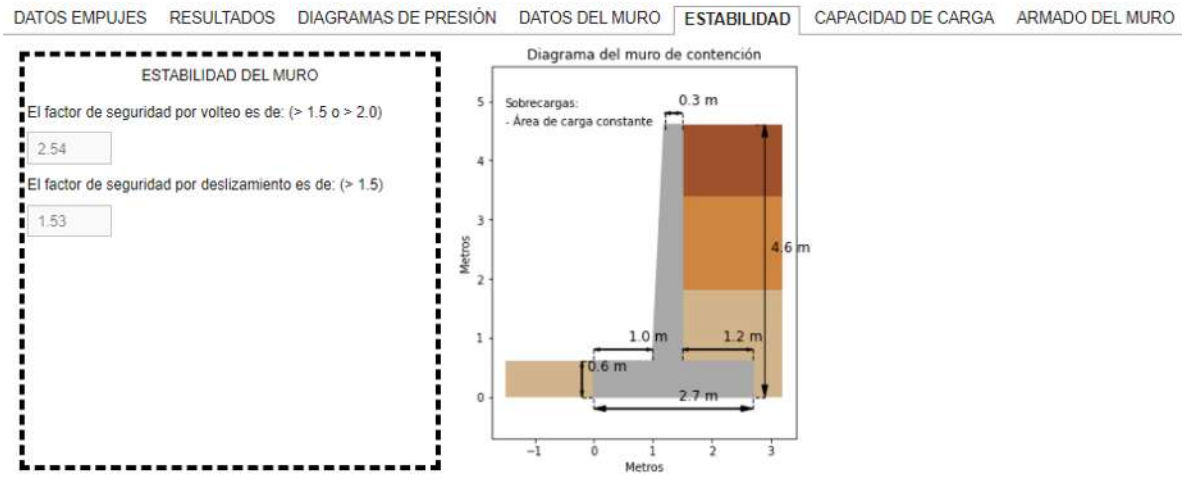



*Figura 21 ESTABILIDAD. Manual de usuario.*

En la pestaña CAPACIDAD DE CARGA, aparecen la capacidad de carga última del terreno, el factor de seguridad por capacidad de carga y el diagrama de distribución de presiones debajo de la base del muro. Como se muestra en la captura de la Figura 22.

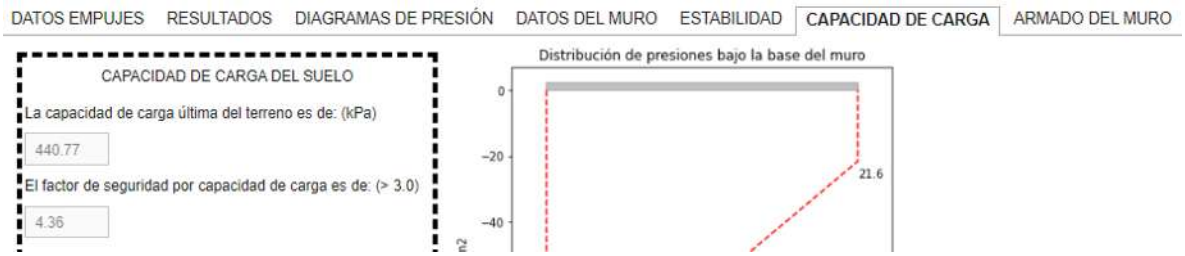



*Figura 22 CAPACIDAD DE CARGA. Manual de usuario.*

En la pestaña ARMADO DEL MURO, aparecen los detalles del armado necesario para que cada elemento del muro resista el cortante y momento último, además de un diagrama del armado del muro. Como se muestra en la captura de la Figura 23.

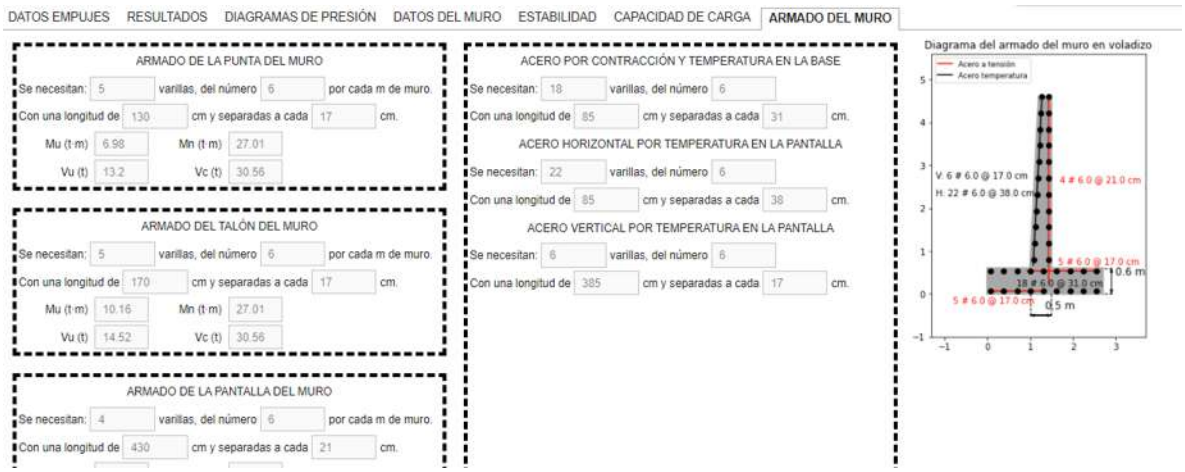



*Figura 23 ARMADO DEL MURO. Manual de usuario.*

Una vez obtenidos todos los resultados, regresar a la primera pestaña, mostrada en la Figura 17 y hacer clic en el botón “generar reporte”. El reporte se generará automáticamente en un archivo Markdown y se descargará automáticamente en formato ZIP.

# 5 CASOS DE VERIFICACIÓN

Para verificar y validar el correcto funcionamiento del código escrito, se utilizaron fragmentos del mismo para la resolución de problemas propuestos en distintas bibliografías.

A continuación, se plantea cada uno de los problemas, se muestra la solución obtenida por el autor, se compara dicha solución con la obtenida con el código programado para esta tesis y se anexa el código utilizado para la resolución.

## 5.1 CASO DE VERIFICACIÓN 1: Presión activa por el método de Rankine (Ejemplo 14.4. Fundamentos de Ingeniería Geotécnica, Braja M. Das, 2013).

En la Figura 24, se muestra un muro de contención. Determine la fuerza activa de Rankine $P_a$, por unidad de longitud de la pared. También determine la ubicación de la resultante.

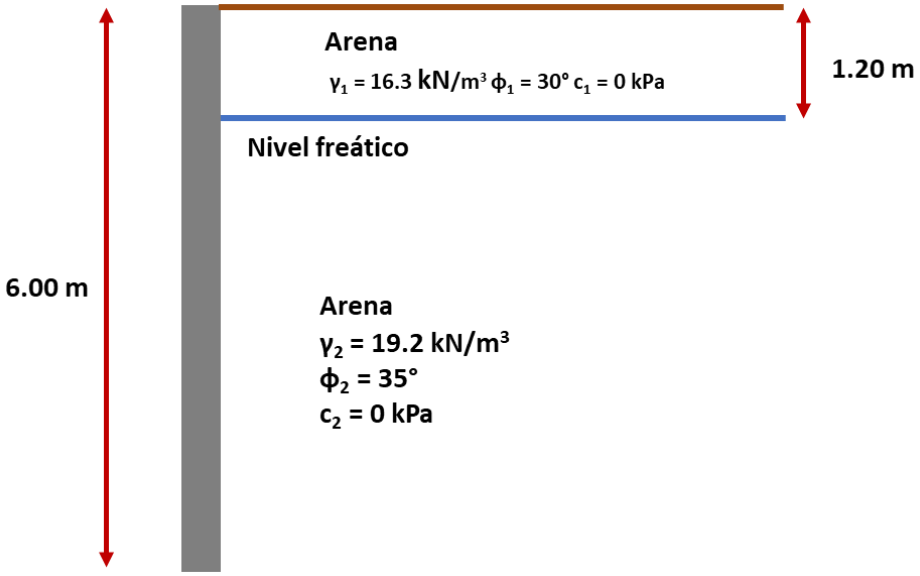



*Figura 24 Caso de verificación 1 (Replicado de Fundamentos de Ingeniería Geotécnica, Braja M. Das, 2013).*

### 5.1.1 RESULTADOS OBTENIDOS POR EL AUTOR

El resultado obtenido por el autor, es una presión activa total de 172.08 $kN/m$, con una resultante ubicada a 1.8 $m$ de altura, a partir de la base del muro. El diagrama de distribución de presiones se muestra en la Figura 25.

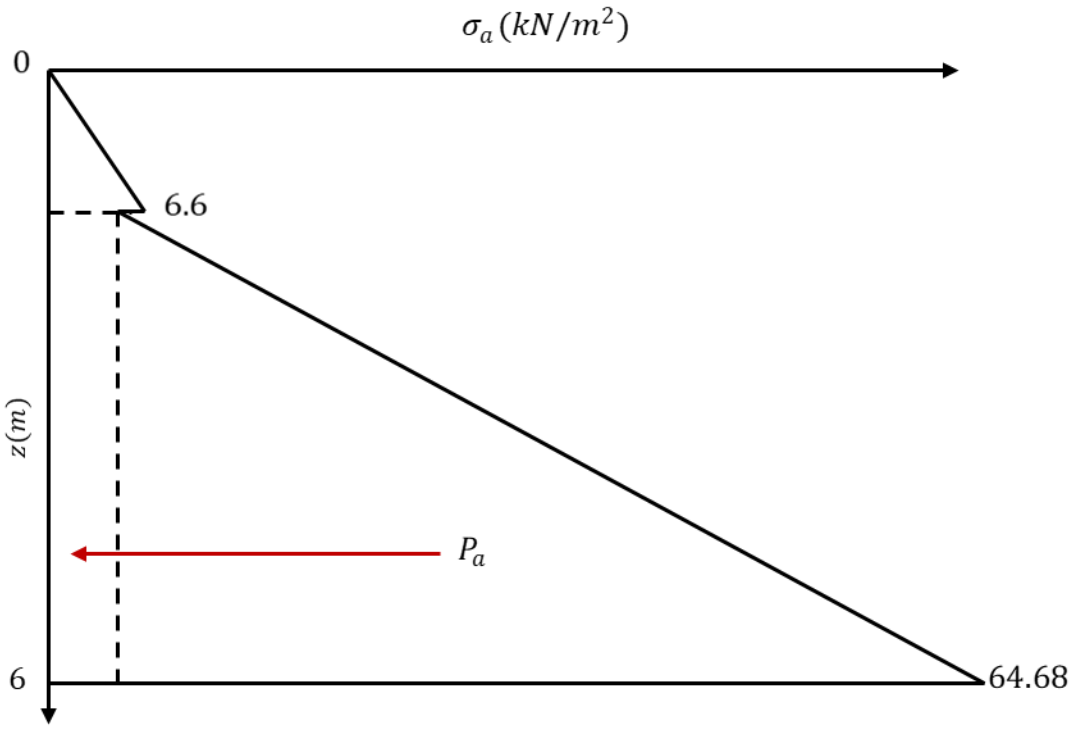



*Figura 25 Diagrama de distribución de presiones (Replicado de Fundamentos de Ingeniería Geotécnica, Braja M. Das, 2013)*

### 5.1.2 RESULTADOS OBTENIDOS CON LA HERRAMIENTA DE CÁLCULO.

Resolviendo el problema con la herramienta programada, se obtiene una presión activa total de 172.04 $kN/m$, con una resultante ubicada a 1.8 $m$ de altura a partir de la base del muro. En la Figura 26 se muestra el diagrama de distribución de presiones y la ubicación de la resultante. El código utilizado para la resolución de este problema, se encuentra anexo en el apartado 9.1 Caso de verificación 1.

```
Presión activa total: 172.04 kN/m2
Resultante (tomada desde la base): 1.8 m
```

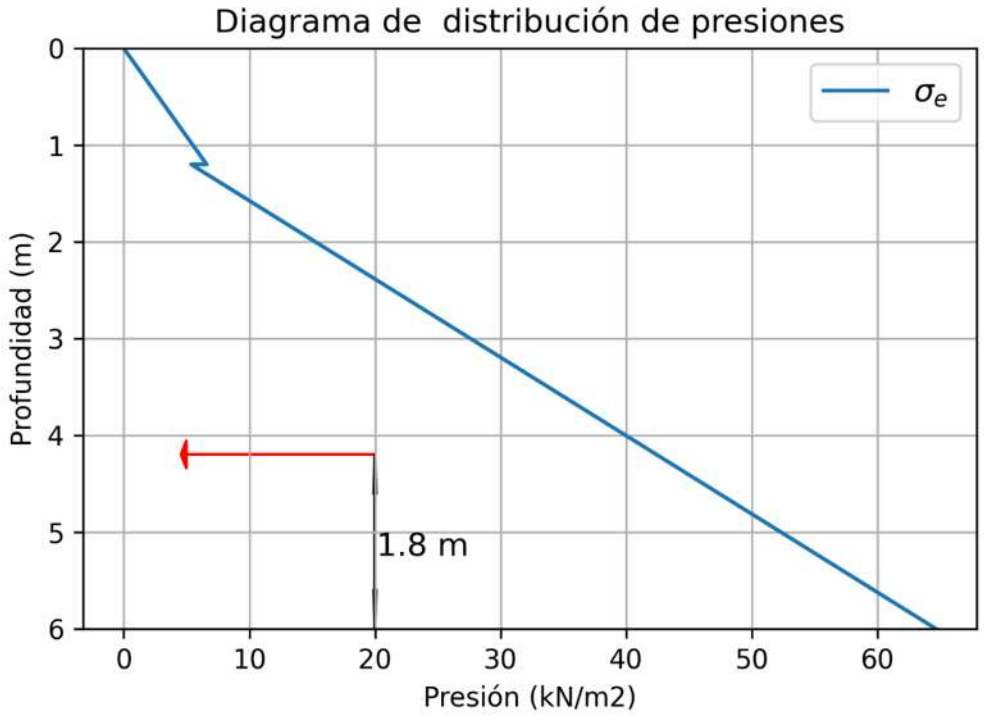



*Figura 26 Diagrama de distribución de presiones obtenido con la herramienta de cálculo programada. (Caso de verificación 1).*

### 5.1.3 CONCLUSIÓN

El resultado obtenido con la herramienta de cálculo es 0.04 kN/m menor, debido a que el autor utiliza solamente dos decimales para realizar los cálculos. Mientras que el código utiliza 15 decimales debido a que se utilizaron variables de tipo $float$.

## 5.2 CASO DE VERIFICACIÓN 2: Presión activa cuando existe cohesión y sobrecarga (Ejemplo 11-5 Foundation Analysis and Design, Joseph E. Bowles, 1997).

Grafique el diagrama de presión activa de tierras y calcule la resultante $R$ y su ubicación para el sistema de muros mostrado en la Figura 27.

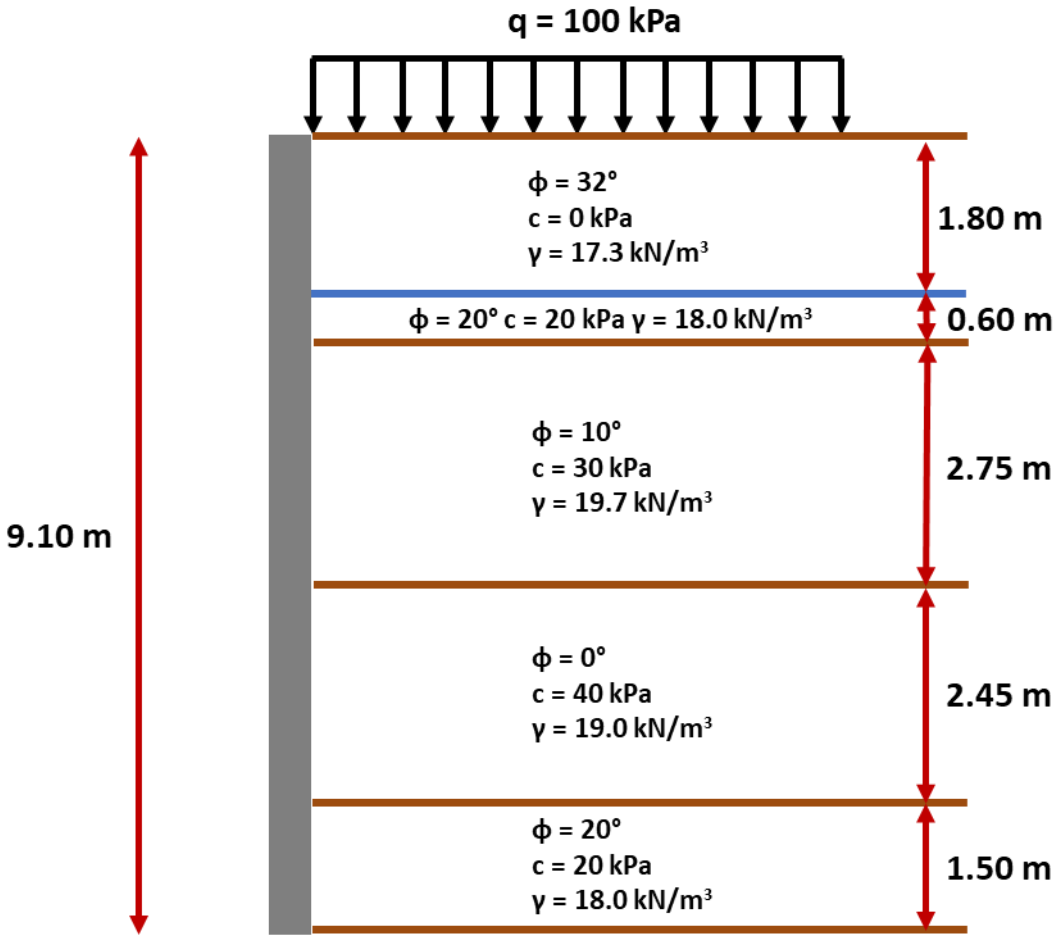



*Figura 27 Caso de verificación 2 (Replicado de Foundation analysis and design, Joseph E. Bowles, 1997).*

## 5.2.1 RESULTADOS OBTENIDOS POR EL AUTOR

El resultado obtenido por el autor, es una presión activa total de 550.7 $kN$, cuya resultante se encuentra a 3.67 $m$ de altura, a partir de la base del muro. El diagrama de distribución de presiones se muestra en la Figura 28.

El autor toma en cuenta las siguientes consideraciones:

- Se asume que el relleno del muro de contención estará drenado, por lo que se desprecia la presión del agua. De no tomar en cuenta esta consideración, la presión activa total sería 261.3 $kN$ mayor.
- Resta la zona de tensión del diagrama de distribución de presiones a la presión activa total, sin embargo, sugiere no hacer esto en la práctica para ofrecer un resultado más conservador.

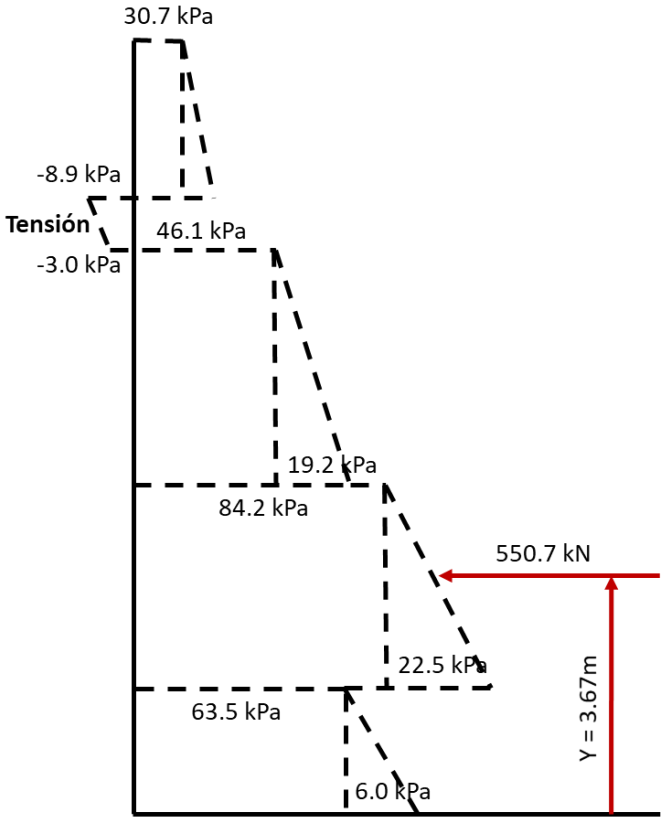



*Figura 28 Diagrama de distribución de presiones (Replicado de Foundation analysis and design, Joseph E. Bowles, 1997).*

### 5.2.2 RESULTADOS OBTENIDOS CON LA HERRAMIENTA DE CÁLCULO.

Calculando con la herramienta programada, se obtiene una presión activa total de 547.27 $kN/m$, con una resultante ubicada a 3.65 $m$ de altura a partir de la base del muro. En la Figura 29, se muestra el diagrama de distribución de presiones y la ubicación de la resultante. El código utilizado para la resolución de este problema, se encuentra anexo en el apartado 9.2 Caso de verificación 2.

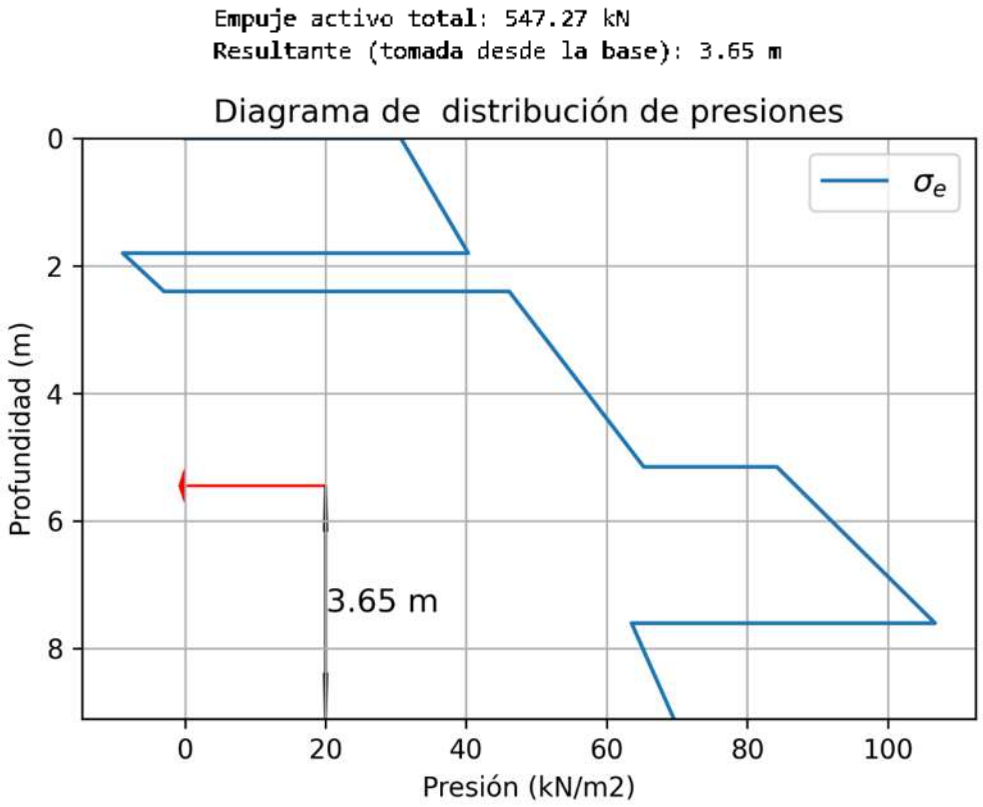



*Figura 29 Diagrama de distribución de presiones obtenido con la herramienta de cálculo programada. (Caso de verificación 2).*

### 5.2.3 CONCLUSIÓN

En el resultado obtenido con la herramienta de cálculo, con el fin de ofrecer un resultado más conservador, como recomienda el autor, no se toma en cuenta el área negativa que representa la zona de tensión.

La zona de tensión tiene un área equivalente a 3.53 kN, por lo que, de tomarla en cuenta, la presión activa total tendría un valor de 550.8 kN cuya resultante se encontraría a 3.67 m, a partir de la base del muro.

## 5.3 CASO DE VERIFICACIÓN 3: Presión pasiva por el método de Rankine (Problema 14.3. Fundamentos de Ingeniería Geotécnica, Braja M. Das, 2013).

A partir de la Figura 30, determine la fuerza pasiva, $P_p$, por unidad de longitud de la pared para el caso Rankine. También determine el estado de presión pasiva de Rankine en la parte inferior de la pared.

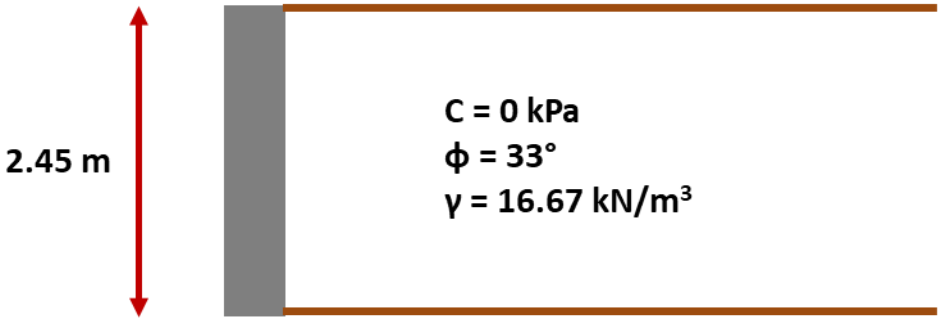



*Figura 30 Caso de verificación 1 (Replicado de Fundamentos de Ingeniería Geotécnica, Braja M. Das, 2013).*

### 5.3.1 RESULTADOS OBTENIDOS POR EL AUTOR.

El autor obtiene una presión pasiva total $P_p$, de 169.6 $kN$ por cada metro lineal de muro. También calcula que la presión pasiva en la base del muro es de 138.5 $kN/m^2$.

### 5.3.2 RESULTADOS OBTENIDOS CON LA HERRAMIENTA DE CÁLCULO.

Resolviendo el problema con la herramienta programada, se obtiene una presión pasiva total de 169.71 $kN$ por metro lineal de muro, con una resultante ubicada a 0.82 $m$ de altura a partir de la base del muro. En la Figura 31, se muestra el diagrama de distribución de presiones y la ubicación de la resultante. El código utilizado para la resolución de este problema, se encuentra anexo en el apartado 9.3 Caso de verificación 3.

```
Presión pasiva total: 169.71 kN
Resultante (tomada desde la base): 0.82 m
La presión pasiva en la base del muro es de: 138.54 kN/m2
```

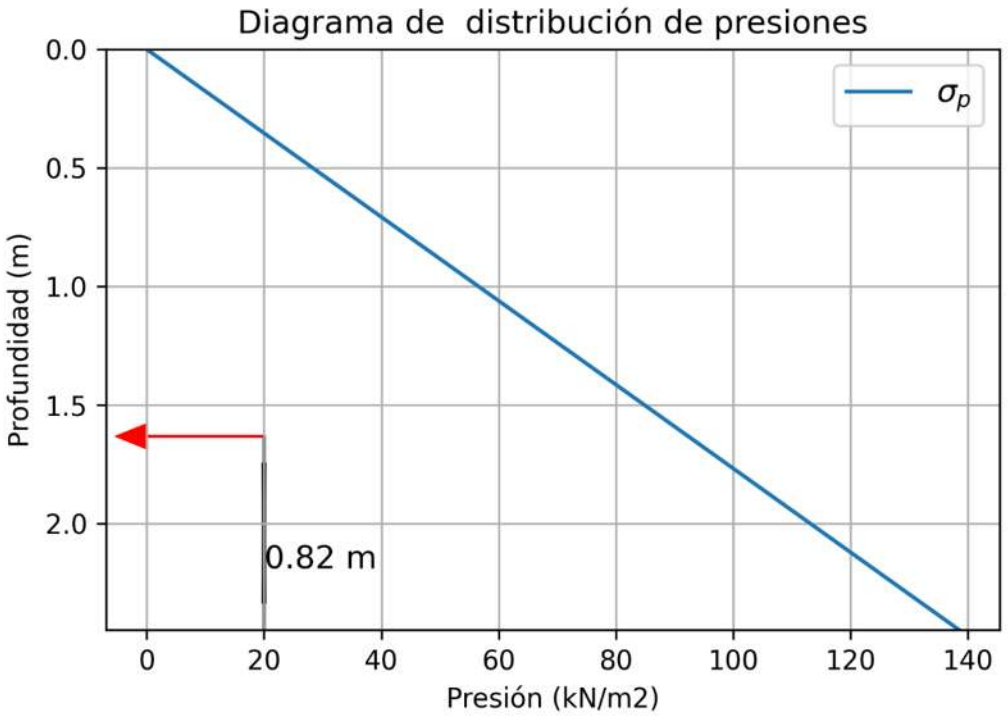



*Figura 31 Diagrama de distribución de presiones obtenido con la herramienta de cálculo programada. (Caso de verificación 3).*

### 5.3.3 CONCLUSIÓN.

El resultado obtenido con la herramienta de cálculo es de 0.11 $kN$ por metro lineal de muro menor debido a que el autor utiliza solamente un decimal para realizar los cálculos, mientras que el código utiliza 15 decimales debido a que se utilizaron variables de tipo $float$. En caso de redondear todos los cálculos a una posición decimal, se obtiene el mismo resultado.

## 5.4 CASO DE VERIFICACIÓN 4: Presión lateral debido a sobrecarga usando la ecuación de Boussinesq (Ejemplo 11-8 Foundation Analysis and Design, Joseph E. Bowles, 1997).

Calcular la fuerza lateral ejercida sobre un muro de 7.5 $m$ de altura, ejercida por un área cargada de 2 x 4 $m$, ubicada a 3 $m$ de distancia del muro. El diagrama del muro y las cargas se muestra en la Figura 32.

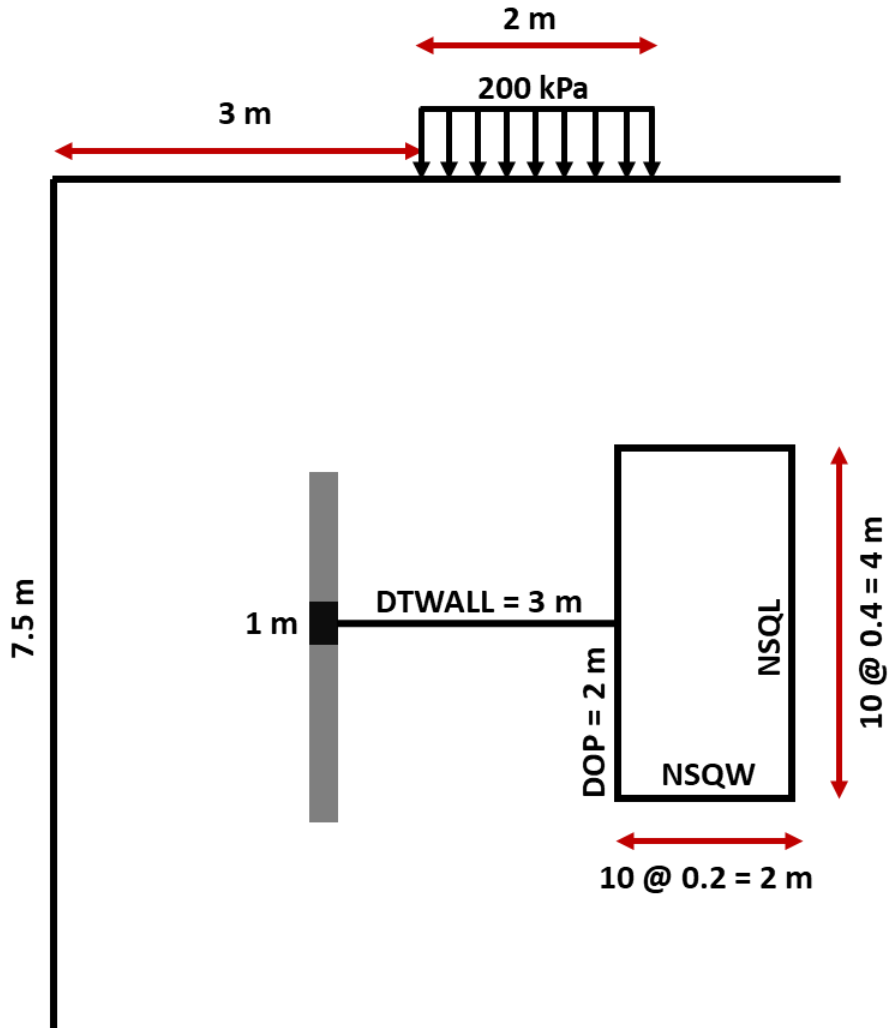



*Figura 32 Caso de verificación 4 (Replicado de Foundation analysis and design, Joseph E. Bowles, 1997).*

### 5.4.1 RESULTADOS OBTENIDOS POR EL AUTOR.

Para la resolución del problema, el autor utiliza un programa llamado SMBLP1, el cual toma como argumentos los datos del problema y arroja como resultado una fuerza horizontal total de 52.672 $kN$, cuya resultante se encuentra ubicada a una altura de 4.401 $m$ a partir de la base del muro.

### 5.4.2 RESULTADOS OBTENIDOS CON LA HERRAMIENTA DE CÁLCULO.

Resolviendo el problema con la herramienta programada, se obtiene una fuerza horizontal total de 54.83 $kN$, con una resultante ubicada a 4.4 $m$ de altura a partir de la base del muro. En la Figura 33, se muestra el diagrama de distribución de presiones y la ubicación de la resultante. El código utilizado para la resolución de este problema, se encuentra anexo en el apartado 9.4 Caso de verificación 4.

```
La presión lateral generada por la sobrecarga es de: 54.83 kN
La resultante de la presión lateral está en: 4.4 m desde la base
```

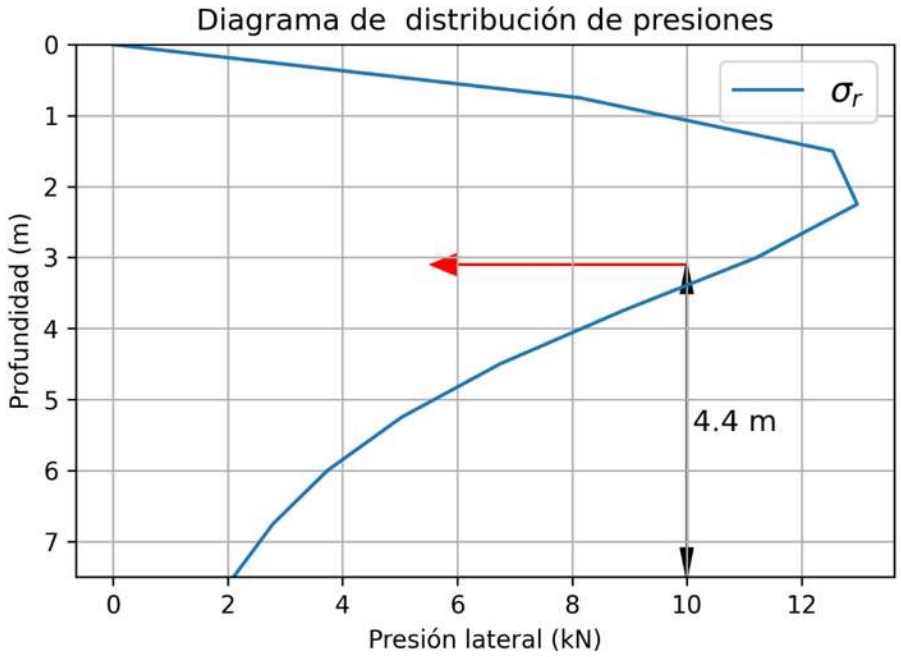



*Figura 33 Diagrama de distribución de presiones obtenido con la herramienta de cálculo programada. (Caso de verificación 4).*

### 5.4.3 CONCLUSIÓN.

El resultado obtenido con la herramienta de cálculo es 2.158 $kN$ mayor que el calculado por medio de la herramienta de cálculo.

## 5.5 CASO DE VERIFICACIÓN 5: Presión lateral debido a sobrecarga usando la ecuación de Boussinesq (Ejemplo 11-9 Foundation Analysis and Design, Joseph E. Bowles, 1997).

Para el área cargada con variación lineal, calcular la fuerza ejercida sobre el muro de la Figura 34.

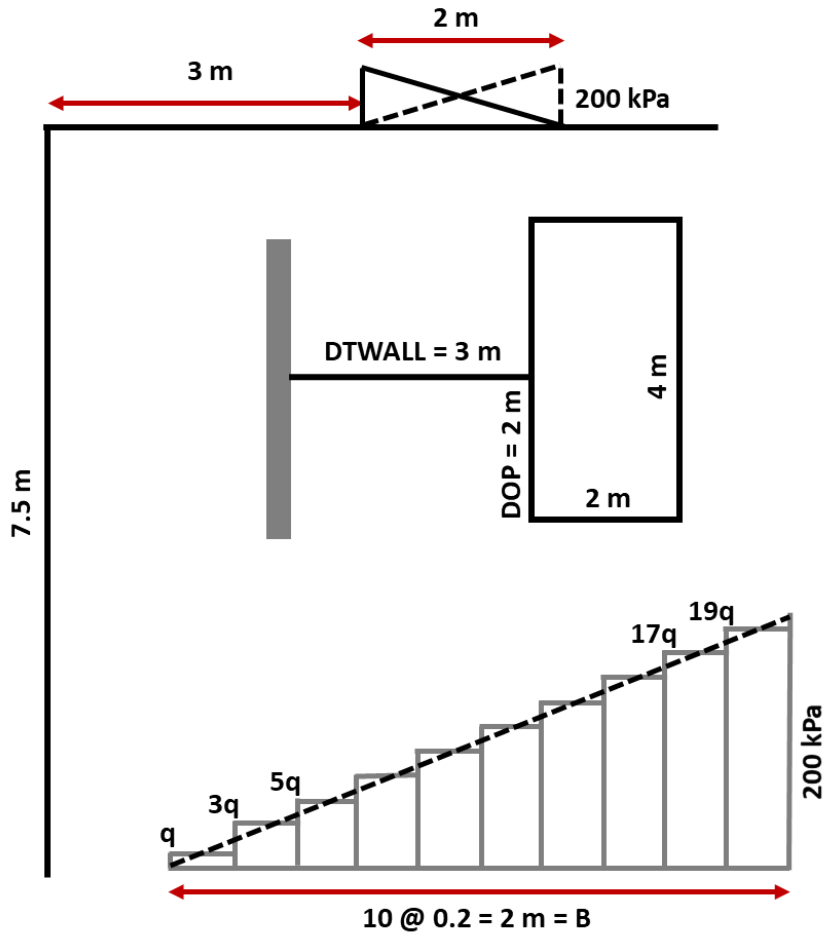



*Figura 34 Caso de verificación 5 (Replicado de Foundation analysis and design, Joseph E. Bowles, 1997).*

## 5.5.1 RESULTADOS OBTENIDOS POR EL AUTOR.

Para la resolución del problema, el autor utiliza un programa llamado SMBLP1, el cual toma como argumentos los datos del problema y arroja como resultado una fuerza horizontal total de 23.898 $kN$, cuya resultante se encuentra ubicada a una distancia de 4.247 $m$ a partir de la base del muro.

### 5.5.2 RESULTADOS OBTENIDOS POR LA HERRAMIENTA DE CÁLCULO.

Calculando con la herramienta programada, se obtiene una fuerza horizontal total de 24.72 $kN$, con una resultante ubicada a 4.24 $m$ de altura, a partir de la base del muro. En la Figura 35, se muestra el diagrama de distribución de presiones y la ubicación de la resultante. El código utilizado para la resolución de este problema, se encuentra anexo en el apartado 9.5 Caso de verificación 5.

```
La presión lateral generada por la sobrecarga es de: 24.72 kN
La resultante de la presión lateral está en: 4.24 m desde la base
```

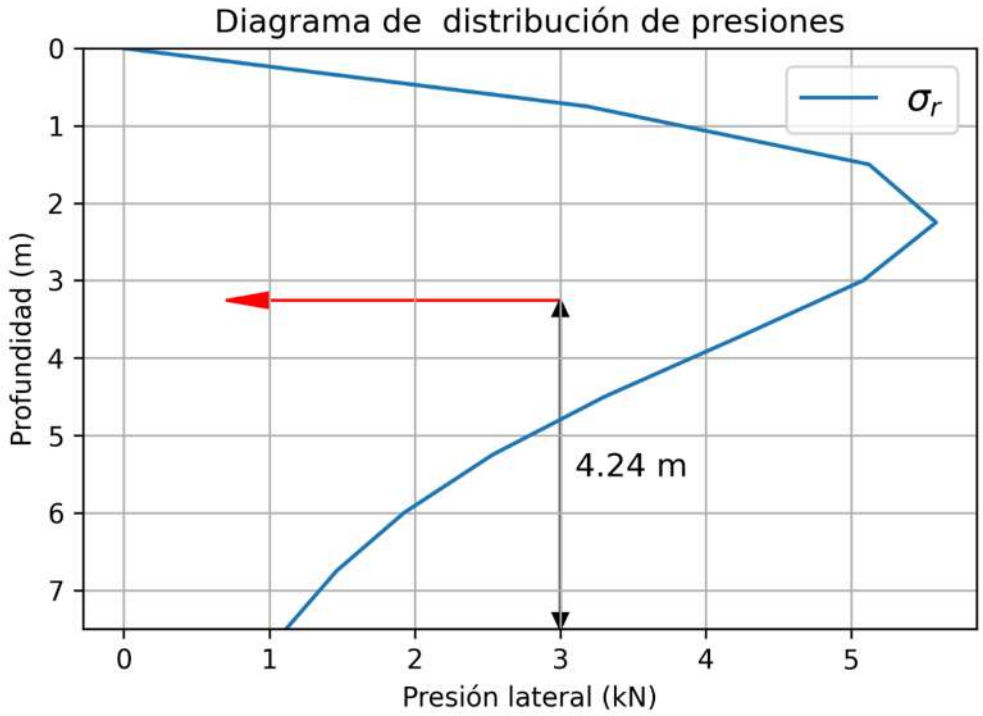



*Figura 35 Diagrama de distribución de presiones obtenido con la herramienta de cálculo programada. (Caso de verificación 5).*

### 5.5.3 CONCLUSIÓN.

El resultado obtenido con la herramienta de cálculo es 0.822 $kN$ mayor que el calculado por medio de la herramienta de cálculo.

Resolviendo el problema con los mismos datos, pero suponiendo que ahora la magnitud máxima de la carga se encuentra al inicio y esta tiene un incremento negativo. Se puede comprobar que el resultado es correcto, sumando ambas presiones laterales totales. La suma de ambas debe ser igual al resultado obtenido al ejecutar la herramienta de cálculo para el caso de verificación 4.

## 5.6 CASO DE VERIFICACIÓN 6: Empuje lateral debido a sismos por el método de Mononobe-Okabe. (Ejemplo 11.1 Geotechnical earthquake engineering, Steven L. Kramer, 1996).

Calcular el momento de volteo en la base del muro mostrado en la Figura 36. Para un coeficiente sísmico horizontal, $k_h$= 0.15, y un coeficiente sísmico vertical, $k_v$ = 0.075.

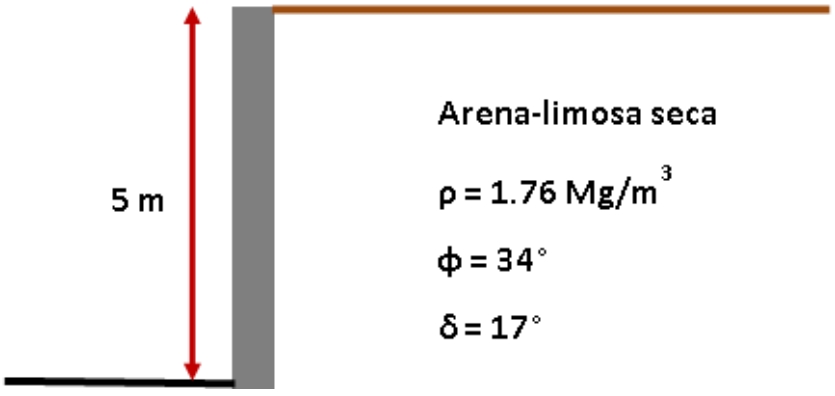



*Figura 36 Caso de verificación 6 (Replicado de Geotechnical Earthquake Engineering, Steven L. Kramer, 1996).*

### 5.6.1 RESULTADOS OBTENIDOS POR EL AUTOR.

El autor obtiene una presión activa total $P_a$, de 55.3 $kN$, una presión activa total debido a sismo $P_{ae}$, de 72.3 $kN$ por lo que el incremento de presiones debido al sismo $\Delta P_{ae}$, es igual a 17 $kN$.

La resultante del incremento de presiones debido a sismo se ubica a 1.98 $m$ a partir de la base del muro y tomando en cuenta solamente la componente horizontal del empuje debido al sismo, el autor obtiene que el momento de volteo provocado por el empuje debido al sismo es igual a 137 $kN\ m$.

### 5.6.2 RESULTADOS OBTENIDOS POR LA HERRAMIENTA DE CÁLCULO.

Resolviendo el problema con la herramienta programada, se obtiene una presión activa total $P_a$, de 61.02 $kN$ como se muestra en la Figura 37, una presión activa

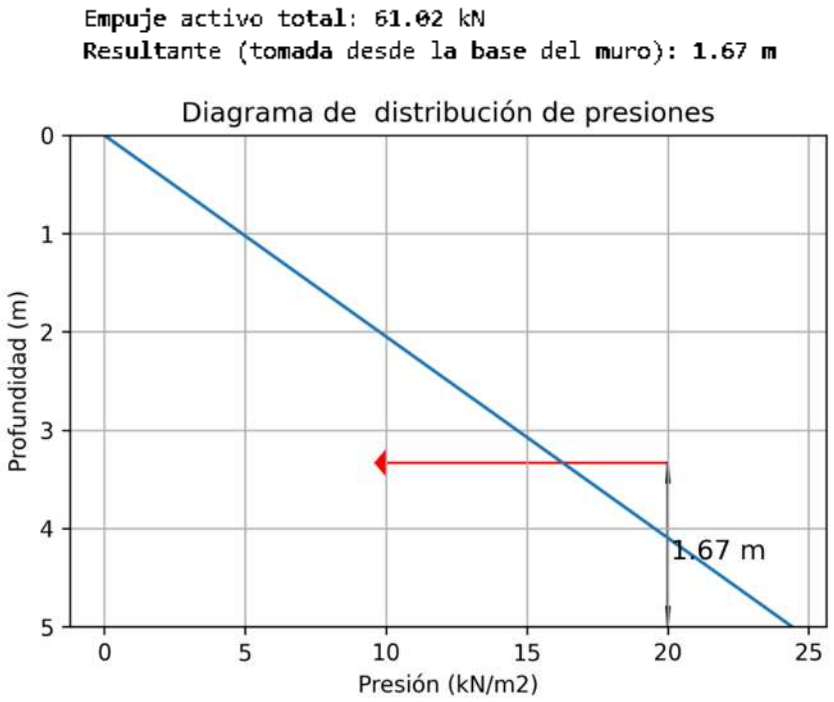



*Figura 37 Diagrama de distribución de presiones activas. Obtenido con la herramienta de cálculo programada. (Caso de verificación 6).*

total debido a sismo, $P_{ae}$, de 72.34 $kN$ como se muestra en la Figura 38, por lo que el incremento de presiones debido al sismo, $\Delta P_{ae}$, es igual a 11.32 $kN$.

```
Empuje activo total debido a sismo: 72.34 kN
Resultante Pae (tomada desde la base): 3 m
```

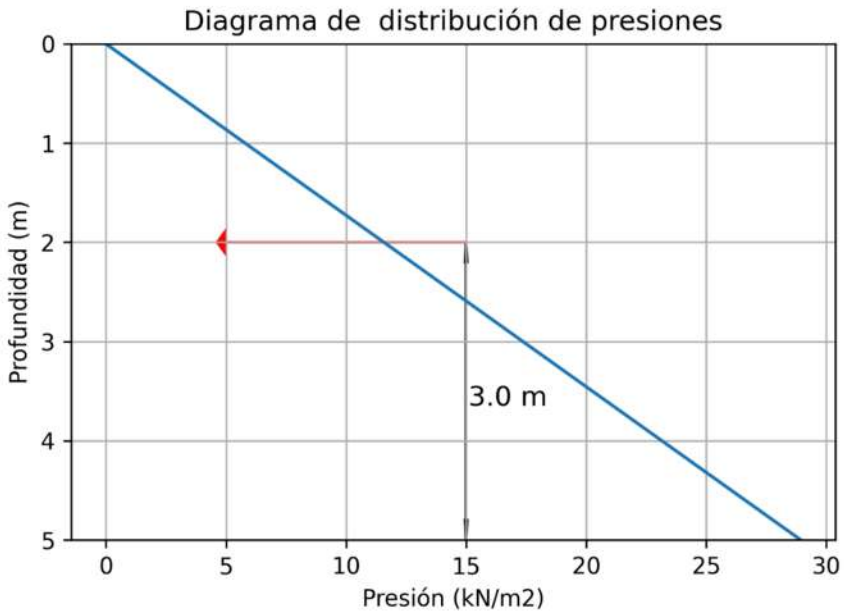



*Figura 38 Diagrama de distribución de presiones debido a sismo. Obtenido con la herramienta de cálculo programada. (Caso de verificación 6).*

La resultante del incremento de presiones debido a sismo, se ubica a 1.88 $m$ a partir de la base del muro y tomando en cuenta solamente la componente horizontal del empuje debido a sismo, se obtiene que el momento de volteo provocado por el empuje debido al sismo, es igual a 129.93 $kN\ m$.

```
El incremento de la presión activa debido a sismo es de: 11.32 kN
El empuje total debido a sismo es de  72.34 kN
La resultante de la presión activa debido a sismo se encuentra en 1.88 m
El momento sísmico es igual a 129.93 kN/m m
```

El código utilizado para la resolución de este problema se encuentra anexo en el apartado 9.6 Caso de verificación 6.

### 5.6.3 CONCLUSIÓN.

La principal diferencia entre el resultado obtenido por el autor y el resultado obtenido por la herramienta de cálculo es el empuje activo. Esto se debe a que el autor calcula el empuje activo usando la ecuación de Coulomb, mientras que la herramienta de cálculo está programada utilizando la ecuación de Rankine.

El resultado obtenido para el empuje activo debido a sismo, calculado en ambos casos con la ecuación de Mononobe-Okabe es el mismo.

## 5.7 CASO DE VERIFICACIÓN 7: Capacidad de carga por el método de Vesić. (Ejemplo 7.3 Foundation design principles and practices, Donald P. Coduto, 2016).

Calcular la capacidad de carga para la zapata cuadrada de 3 $m$ de ancho mostrada en la Figura 39.

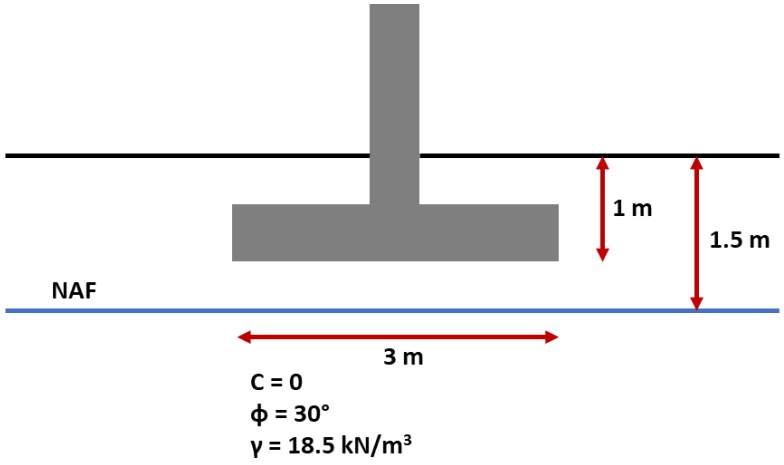



*Figura 39 Caso de verificación 7 (Replicado de Foundation design principles and practices, Donald P. Coduto, 2016).*

### 5.7.1 RESULTADOS OBTENIDOS POR EL AUTOR.

El autor obtiene una capacidad de carga de 799 $kPa$.

### 5.7.2 RESULTADOS OBTENIDOS POR LA HERRAMIENTA DE CÁLCULO.

Resolviendo el problema con la herramienta de cálculo, se obtiene una capacidad de carga de 796.81 $kPa$.

```
La capacidad de carga última es de 796.81 kPa
```

El código utilizado para la resolución de este problema se encuentra anexo en el apartado 9.7 Caso de verificación 7.

### 5.7.3 CONCLUSIÓN.

La capacidad de carga calculada por el autor es 0.19 $kPa$ mayor que la calculada por la herramienta de cálculo, debido a que el autor utiliza 2 decimales para realizar los cálculos, mientras que la herramienta de cálculo utiliza 15 decimales debido a que se utilizaron variables tipo $float$.

## 5.8 CASO DE VERIFICACIÓN 8: Comparación entre el empuje activo y el empuje pasivo utilizando la herramienta de cálculo programada y los obtenidos utilizando un software comercial.

Para este caso de verificación, se utilizó la versión de prueba del software LateralK de NovoTech. Este software calcula la presión activa y la presión pasiva actuantes sobre el muro, así como la ubicación de sus resultantes.

Se calcularon los empujes que se ejercen sobre el muro mostrado en la Figura 40.

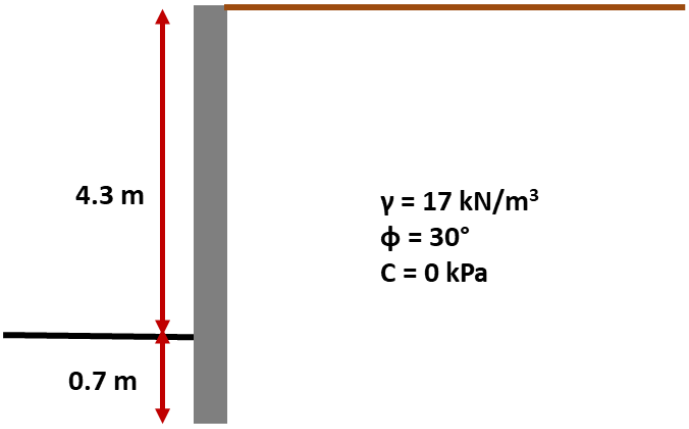



*Figura 40 Caso de verificación 8, condiciones del muro.*

### 5.8.1 RESULTADOS OBTENIDOS UTILIZANDO EL SOFTWARE COMERCIAL.

Los resultados obtenidos utilizando el software comercial LateralK de NovoTech, se muestran en las capturas de la Figura 41. En la primera parte se muestran los resultados pertenecientes a la condición estática, es decir, la presión activa y la

presión pasiva y en la segunda parte aparecen los resultados pertenecientes a la condición sísmica.

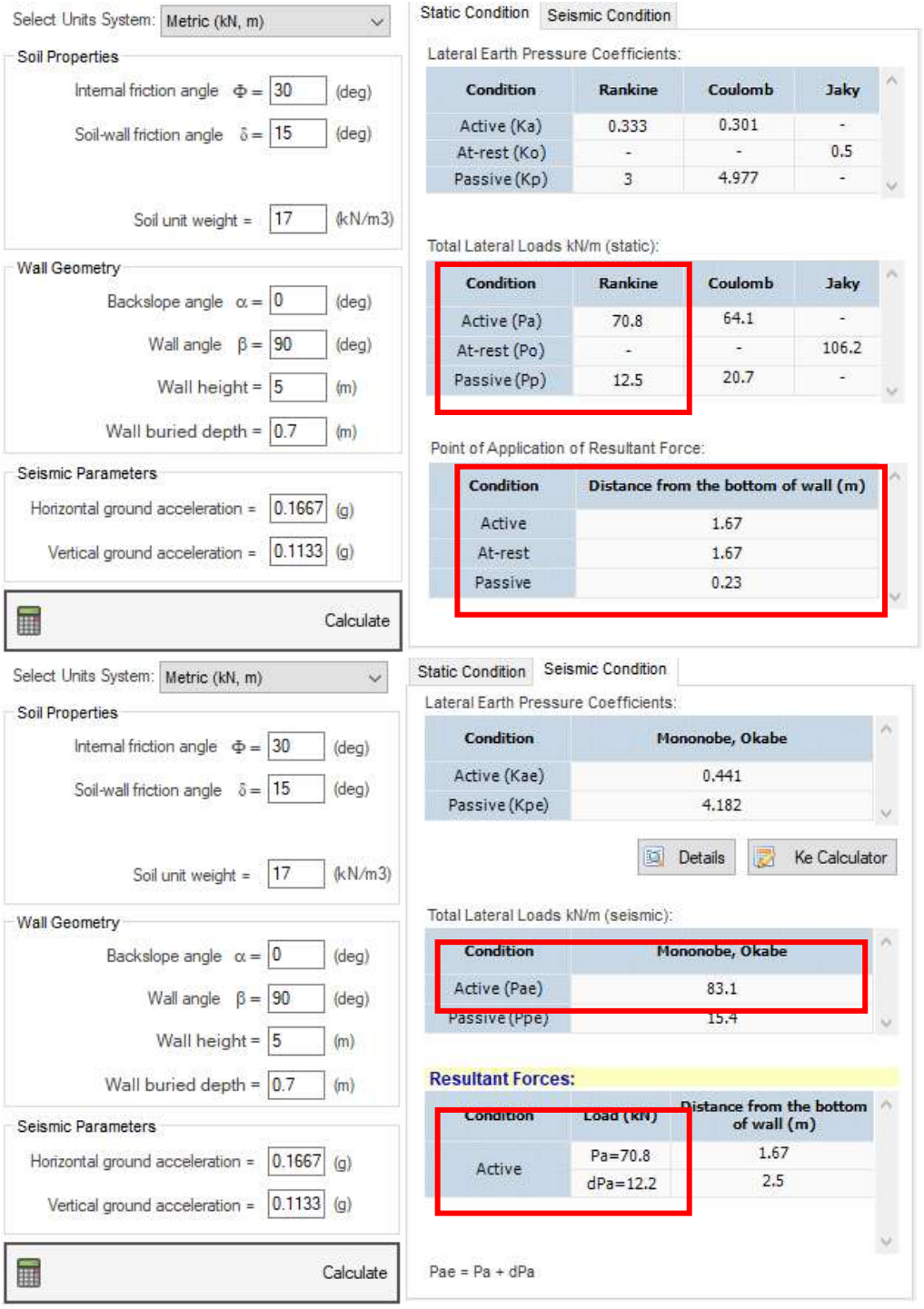



*Figura 41 Resultados obtenidos utilizando el software LateralK.*

## 5.8.2 RESULTADOS OBTENIDOS POR LA HERRAMIENTA DE CÁLCULO.

Resolviendo el problema con la herramienta de cálculo, se obtienen los resultados mostrados en la Figura 42.

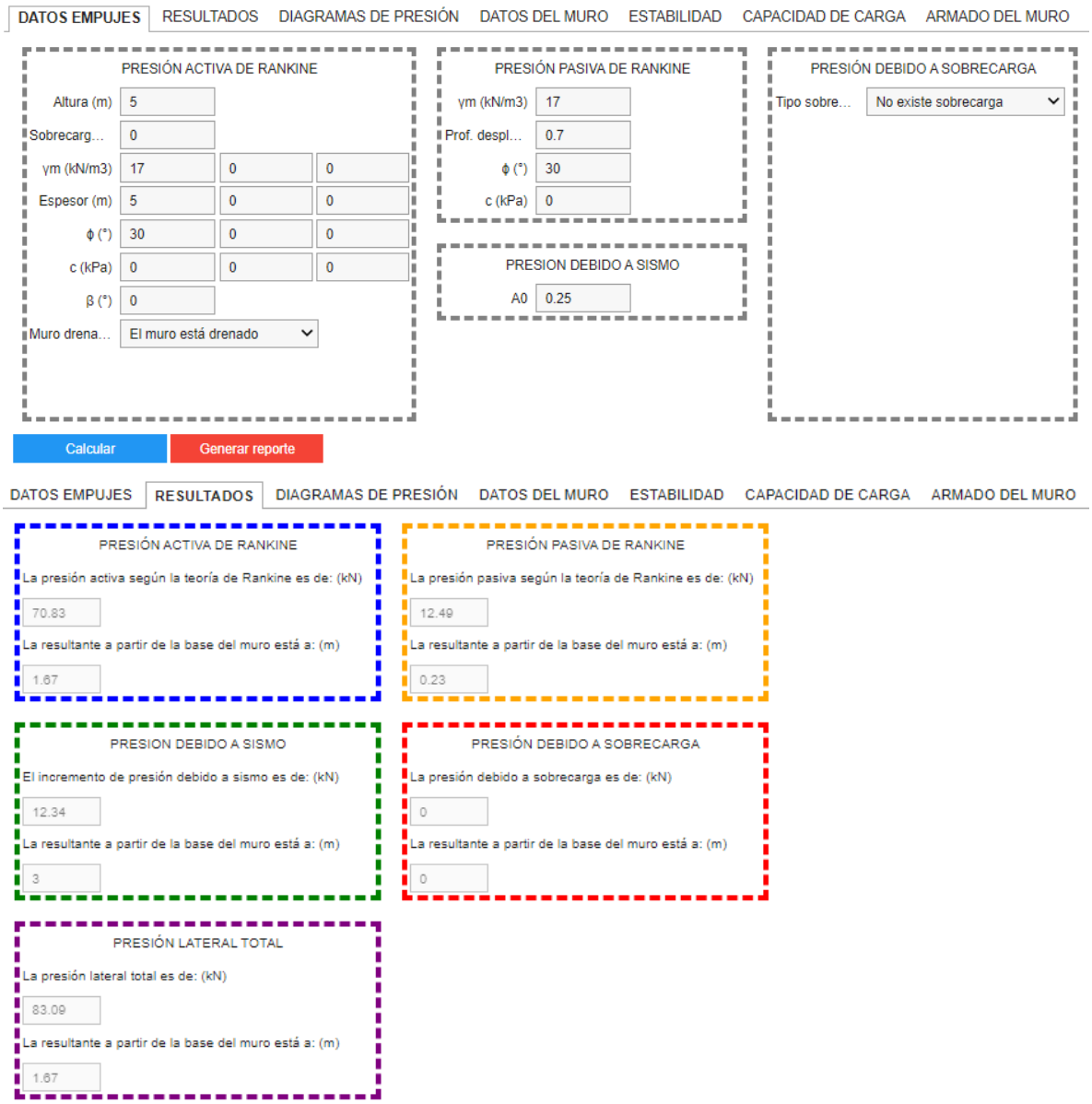



*Figura 42 Resultados obtenidos con la herramienta de cálculo (Caso de verificación 8).*

### 5.8.3 CONCLUSIÓN.

El empuje activo, calculado con el software LateralK es de 70.8 kN, cuya resultante se ubica a 1.67 m de altura, el calculado utilizando la herramienta de cálculo programada es de 70.83 kN, y su resultante se ubica a 1.67 m de altura.

El empuje pasivo, calculado con el software LateralK es de 12.5 kN cuya resultante se ubica a 0.23 m de altura, el calculado utilizando la herramienta de cálculo programada es de 12.49 kN y su resultante se ubica a 0.23 m de altura.

Para calcular el incremento del empuje debido a sismo, la herramienta de cálculo programada calcula el coeficiente sísmico horizontal como, $k_h = \frac{2}{3} A_o$ y el coeficiente sísmico vertical como, $k_v = \frac{2}{3} k_h$, por lo que para ingresar una aceleración de 0.25 g en el software LateralK, se ingresa 0.1667 en la aceleración sísmica horizontal y 0.1133 en la aceleración sísmica vertical. El ángulo de fricción entre el material de relleno y el muro, se calcula como el ángulo de fricción interna del material entre dos, por lo que en el software LateralK se computa como 15°.

El incremento del empuje activo debido a sismo, calculado con el software LateralK es de 12.2 kN, mientras que el calculado utilizando la herramienta de cálculo programada es de 12.34 kN.

La presión activa total, calculada con el software LateralK es de 83.1 kN, mientras que el calculado con la herramienta de cálculo programada es de 83.09 kN. Por lo tanto, se concluye que los resultados obtenidos por ambos programas son prácticamente los mismos.

# 6 EJEMPLO DE FUNCIONAMIENTO DEL PROGRAMA.

Para explicar el funcionamiento del programa, se realizó el diseño de un muro de contención de 4.6 m de altura, capaz de contener el relleno mostrado en la Figura 43. Tomando las siguientes consideraciones:

- El relleno del muro se encuentra drenado.
- Existe una sobrecarga distribuida en un área de 2 x 4 m, con una intensidad de 200 KPa, que se encuentra a 3 m del muro. Como se muestra en la Figura 44.
- La relación de Poisson del material $\mu$, es de 0.5.
- Para el cálculo del incremento del empuje debido a sismo, se usó una aceleración sísmica de 0.25 veces la gravedad.

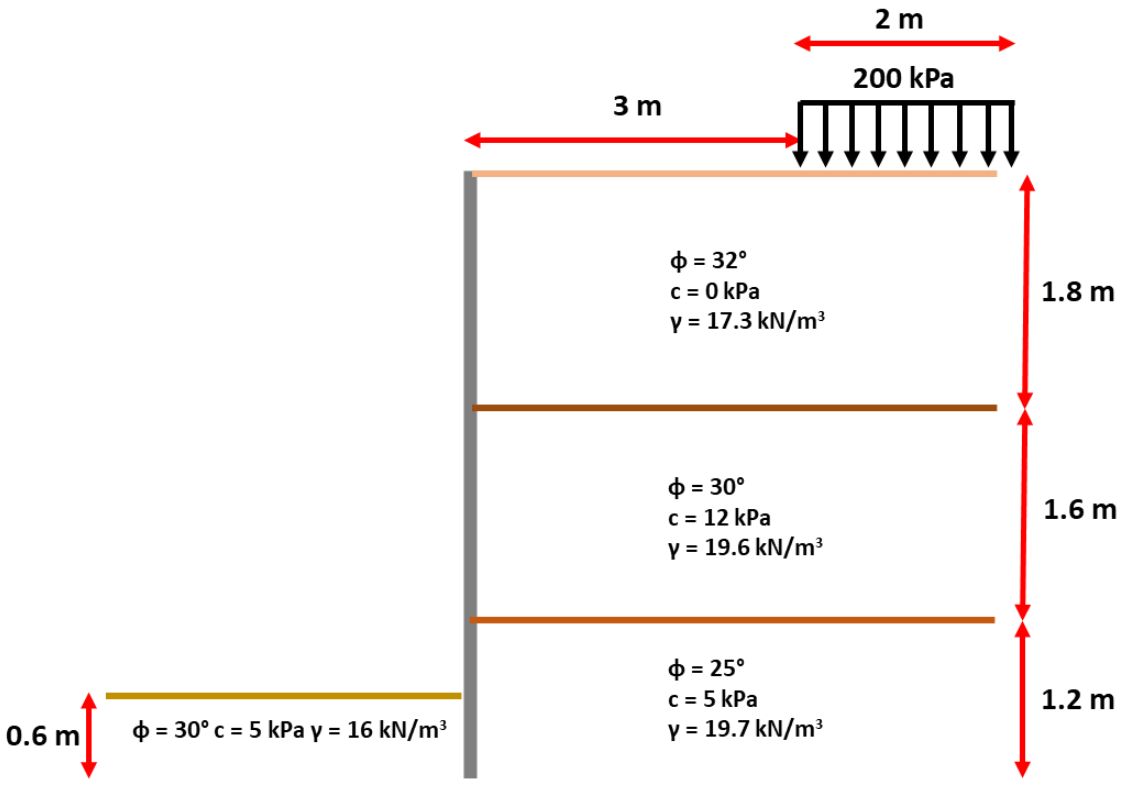



*Figura 43 Diagrama del perfil del muro a resolver con la herramienta de cálculo.*

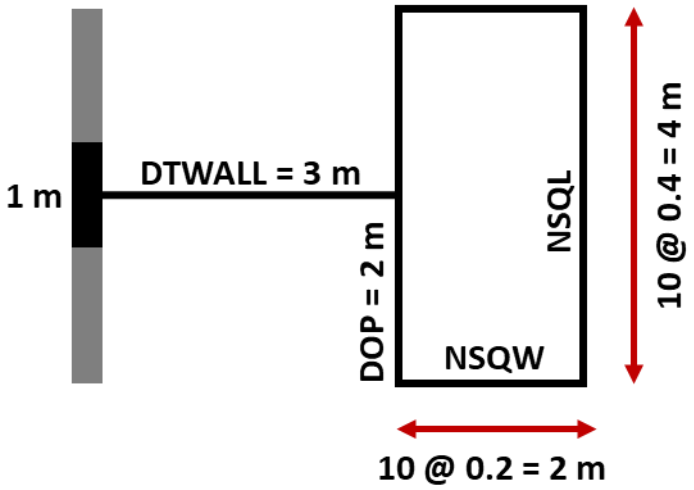



*Figura 44 Diagrama de la sobrecarga del muro.*

Se ingresan los datos necesarios para calcular los empujes actuantes sobre el muro en la primera pestaña de la GUI. Como se muestra en la Figura 45.

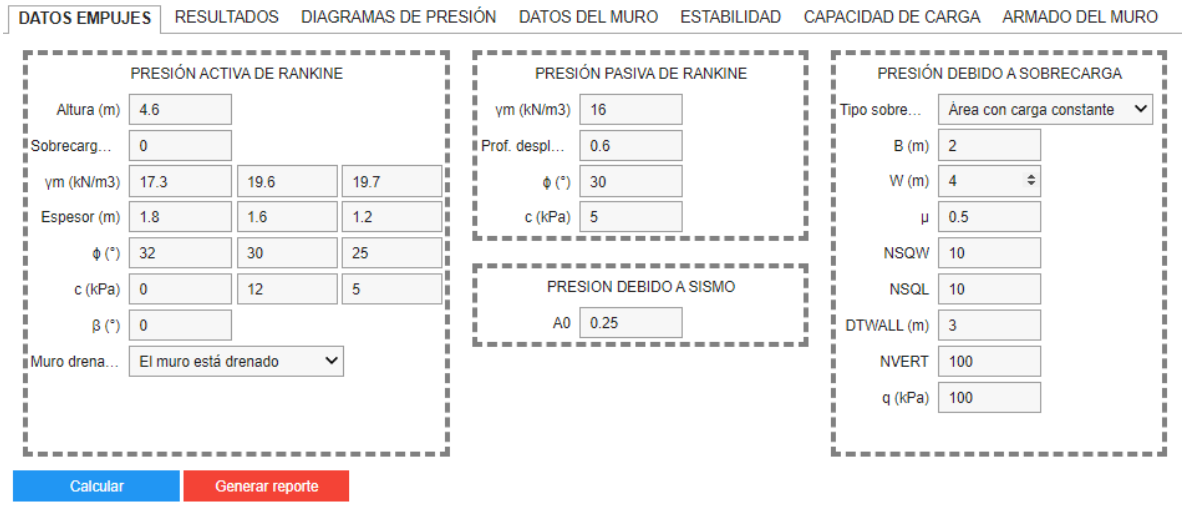



*Figura 45 Ejemplo de funcionamiento, datos necesarios para calcular los empujes.*

En la Figura 46, se muestra la pestaña de resultados, en ella aparecen los resultados de todos los empujes actuantes sobre el muro.


*Figura 46 Ejemplo de funcionamiento, resultados de los empujes actuantes sobre el muro.*

En la Figura 47, se muestra la pestaña diagramas de presión en la que aparece cada uno de los diagramas de distribución de presiones actuantes sobre el muro.

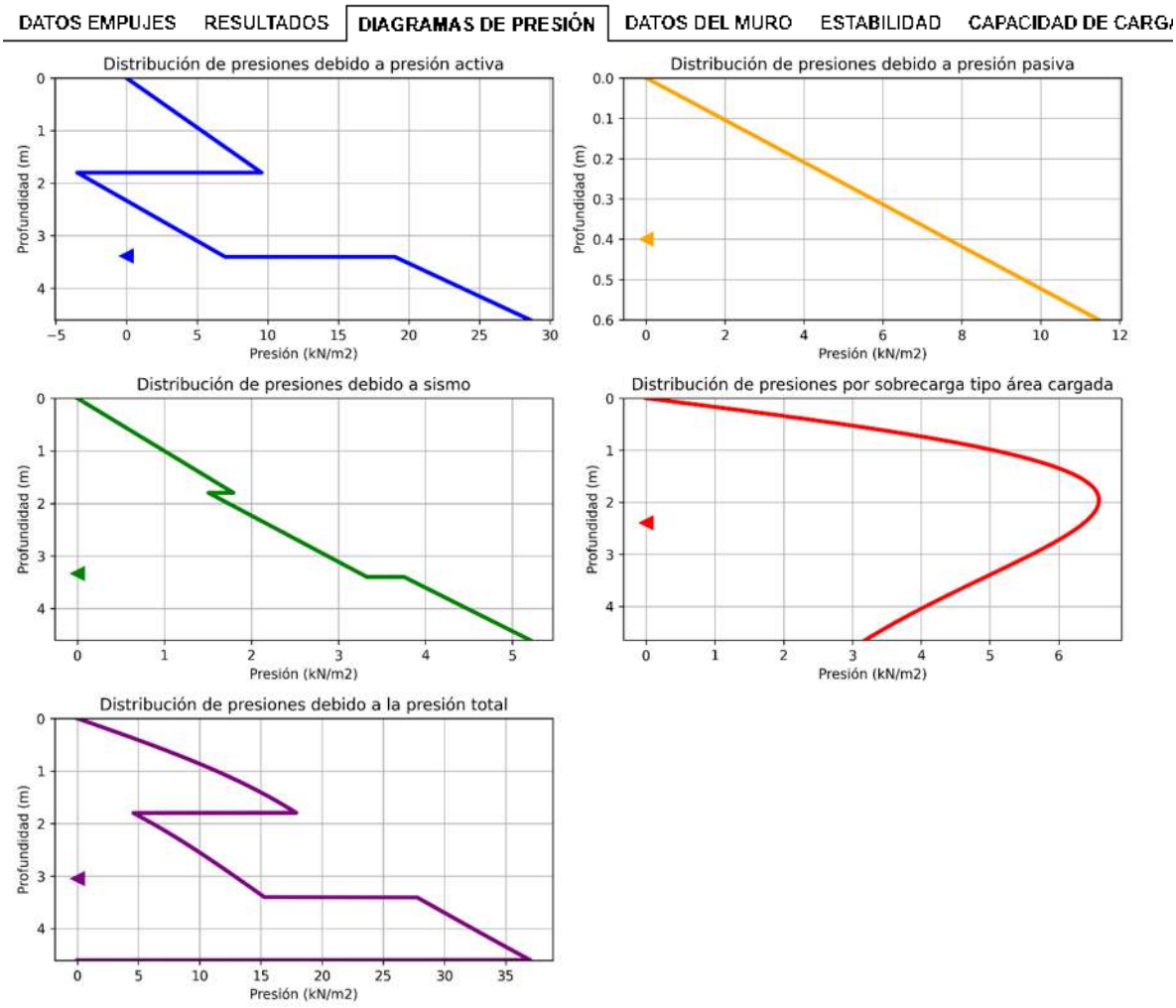



*Figura 47 Ejemplo de funcionamiento, diagramas de distribución de presiones.*

En la Figura 48, se muestra la pestaña datos del muro. En ella el usuario debe ingresar datos referentes al tipo de muro, a la interacción muro-suelo y a las propiedades mecánicas de los materiales para el armado del muro en caso de haber seleccionado el muro en voladizo.

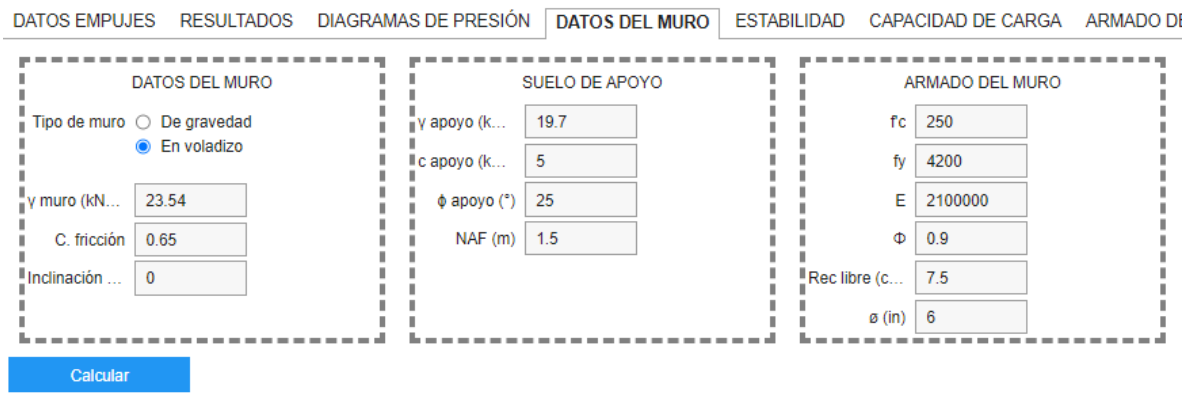



*Figura 48 Ejemplo de funcionamiento, datos del muro.*

En la Figura 49, se muestra la pestaña estabilidad. En ella aparecen los factores de seguridad contra volteo y contra deslizamiento, además de un diagrama del muro de contención con las dimensiones mínimas necesarias para que el muro sea estable.

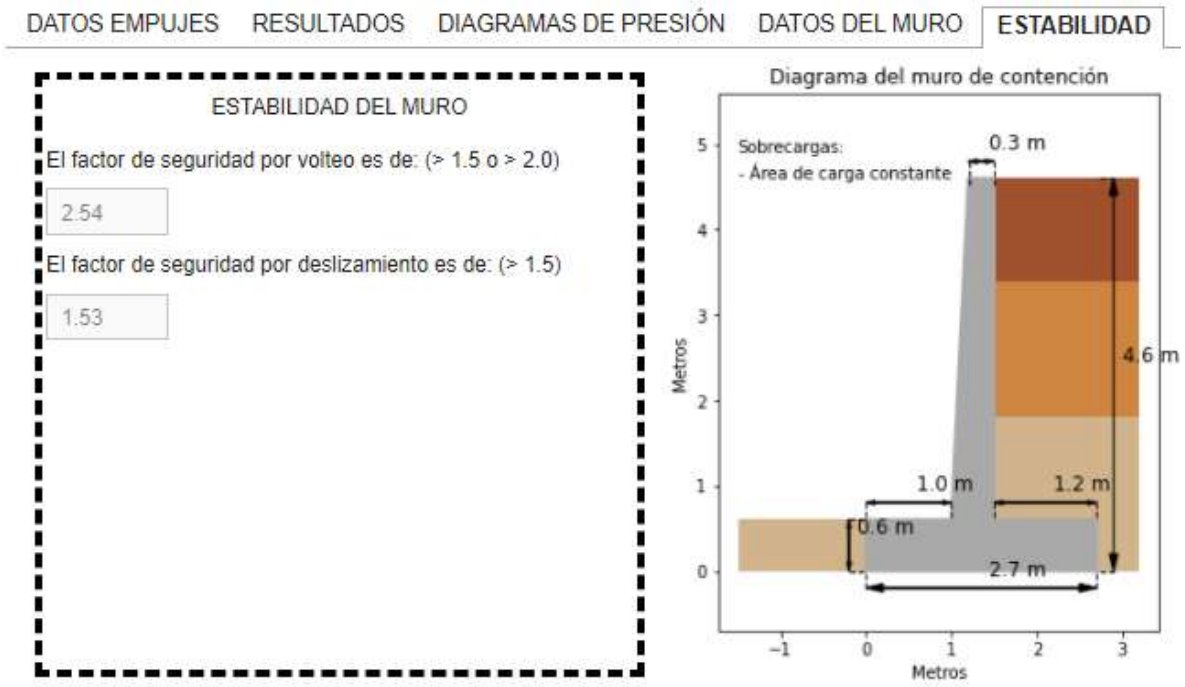



*Figura 49 Ejemplo de funcionamiento, estabilidad del muro.*

En la Figura 50, se muestra la pestaña Capacidad de carga. En ella aparece la capacidad de carga última del suelo debajo del muro, el factor de seguridad por capacidad de carga, además de un diagrama con la distribución de presiones que el muro ejerce sobre el suelo.

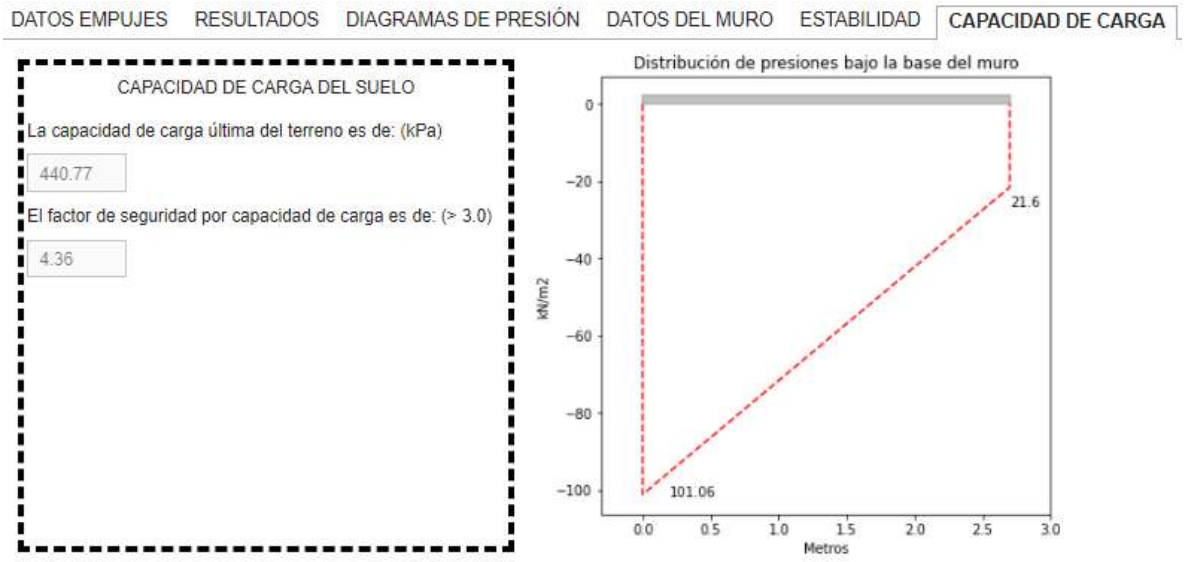
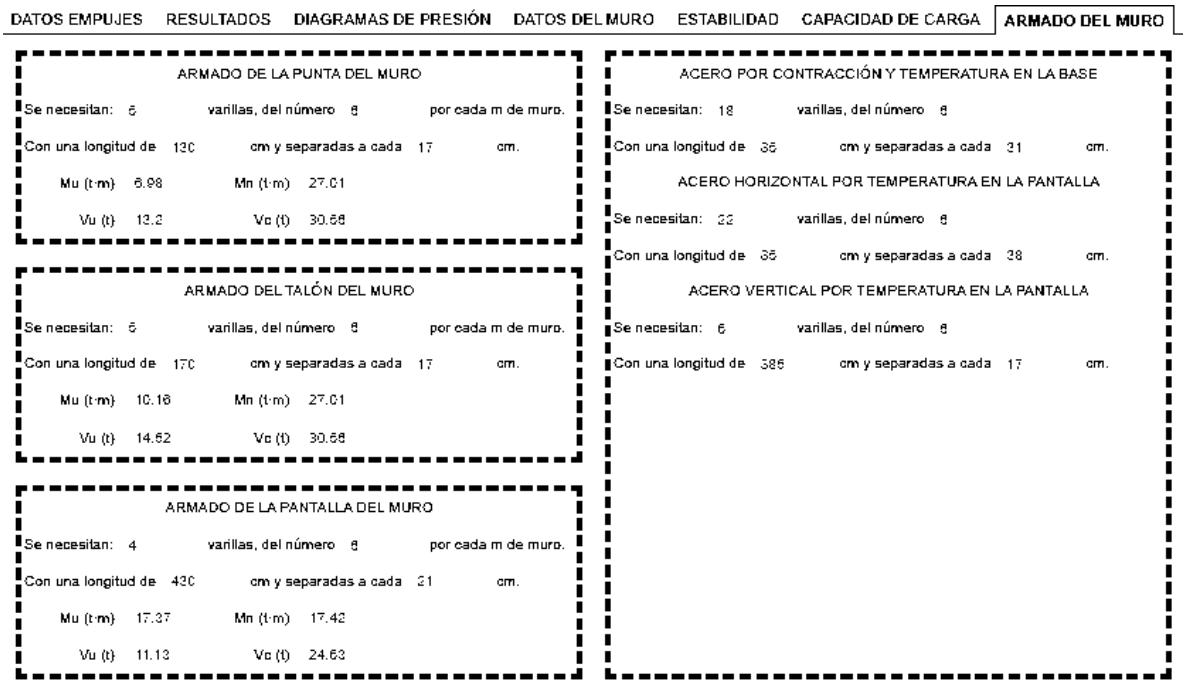



*Figura 50 Ejemplo de funcionamiento, capacidad de carga.*

En caso de haber seleccionado muro en voladizo, en la pestaña armado del muro, aparecen las varillas necesarias para resistir los elementos mecánicos a los que está sometida la pantalla, la punta y el talón del muro de contención. También se muestra el acero necesario por contracción y temperatura. Como se muestra en la Figura 51.



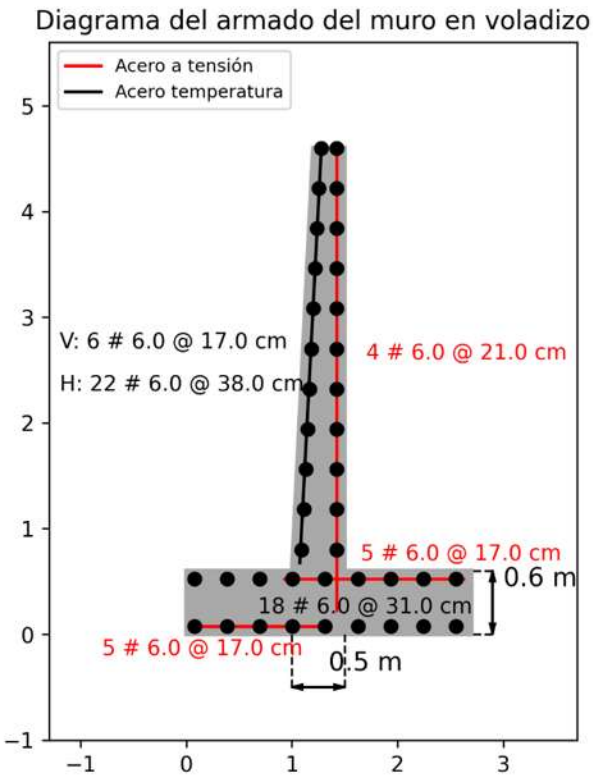




*Figura 51 Ejemplo de funcionamiento, armado del muro en voladizo.*

El reporte de resultados de este ejemplo de funcionamiento se puede consultar en el apartado de 9.9 de anexos.

# 7 CONCLUSIONES

En los últimos años, el diseño constructivo ha experimentado una migración casi total hacia la computarización debido a que esto disminuye considerablemente el tiempo que le toma a un ingeniero realizar tareas y cálculos rutinarios.

En la presente tesis, se desarrolló una herramienta de cálculo que automatiza el proceso de diseño de muros de contención de gravedad o en voladizo para una amplia variedad de condiciones físicas del terreno, parámetros del sitio y sobrecargas actuantes, ofreciendo una propuesta de diseño que ofrece economía de materiales.

Mediante la resolución de casos de verificación, se comprobó que el programa desarrollado, calcula con precisión cada una de las partes del proceso de diseño del muro de contención respecto a los resultados obtenidos por los autores en diferentes bibliografías.

La GUI, implementada mediante Ipywidgets en Google Colaboratory (Cloud), ofrece al usuario una interacción amigable con el programa, fácil de entender para cualquier persona con conocimientos básicos de geotecnia y estructuras.

Como trabajo futuro, se plantea implementar el uso de librerías tipo GIS (Sistemas de información geográfica), que permiten la manipulación de información geográfica. Con lo que se conseguirá automatizar el diseño de muros de contención ingresando como datos de entrada, por ejemplo, la topografía de secciones transversales de carreteras.

# 8 REFERENCIAS BIBLIOGRÁFICAS


- Bowles, J. E. (1997). Foundation analysis and design. 5 ed. Peoria, Illinois. McGraw-Hill. 589-723.

- Das, B. M. (2013). Fundamentos de ingeniería geotécnica. 4 ed. Ciudad de México. Cengage Learning. 379-510.

- Coduto, D. P. (2001). Foundation design: principles and practices. 2 ed. Upper Saddle River, New Jersey. Prentice Hall. 170-205.

- Kramer, S. L. (1996). Geotechnical earthquake engineering. Upper Saddle River, New Jersey. Prentice Hall. 466-503.

- Villalaz, C. (2004). Mecánica de suelos y cimentaciones. 5 ed. Ciudad de México. Limusa. 507-528.

- Hunt, J. (2020). A Beginners Guide to Python 3 Programming. Cham, Suiza. Springer. 433.

- Calavera, J. (1987). Muros de contención y muros en sótano. 2 ed. Madrid, España. Intemac. 307.

- Secretaría de Desarrollo Urbano y Vivienda, Secretaría de Obras y Servicios (2017). Normas Técnicas Complementarias para el Diseño de Estructuras de Concreto. Ciudad de México.

- Secretaría de Desarrollo Urbano y Vivienda, Secretaría de Obras y Servicios (2017). Normas Técnicas Complementarias Sobre Criterios y Acciones para el Diseño Estructural de las Edificaciones. Ciudad de México.

# 9 ANEXOS

## 9.1 CASO DE VERIFICACIÓN 1

```
import numpy as np
import matplotlib.pyplot as plt
from shapely.geometry import Polygon
from libdibujo import DibujaCota

def Presion_activa_Rankine(NAF, γms, Hs, φ):
  """
  CALCULA LA PRESIÓN ACTIVA TOTAL Y SU RESULTANTE POR EL MÉTODO DE RANKINE
  NAF, la profundidad a la que se encuentra el nivel freático
  γms, es un array con el peso específico del material de cada estrato
  Hs, es un array con el espesor de cada estrato
  φ, es un array con el ángulo de fricción interna de cada material
  """
  γw = 9.81 # peso específico del agua (kN/m3)
  Profs = np.cumsum(Hs) # Vector de espesores acumulados
  γe = γms.copy()
  ProfsW = np.zeros_like(Hs)
  Prof_total = sum(Hs) # Variable con la profundidad total

  # Vector con los coeficientes de presión activa
  ka = (1 - np.sin(np.radians(φ))) / (1 + np.sin(np.radians(φ)))
  ProfsW = np.where(Profs > NAF, Profs - NAF, ProfsW)
  u = γw * ProfsW # Vector con la presión del agua
  γe = np.where(Profs > NAF, γms - γw, γe) # Vector con γms - γw (si es el caso)

  σa = np.cumsum(Hs * γe) # Vector con las profundides acumuladas
  σe = (ka * σa) + u # Vector con las presiones efectivas

  for i in range (0, len(γms), 2): # Agrega la presión con el coeficiente del
estrato siguiente
    σe = np.insert(σe, i+1, σa[i] * ka[i + 1])
    Profs = np.insert(Profs, i+1, Profs[i])

  σe = np.insert(σe, 0, 0) # Agrega el punto (0, 0)
  Profs = np.insert(Profs, 0, 0) # Agrega el punto (0, 0)

  a = np.stack((σe,Profs), axis=1) # Junta la matriz de profundidac con la de
presión efectivos
  point = np.array([[0, Prof_total]]) # Agrega el punto necesario para cerrar el
polígono
  b = np.vstack((a, point)) # Tuple con las coordenadas para el polígono

  pre_lat = Polygon(b) # Polígono con la distribución de presión lateral

  plt.plot(σe, Profs, label=r'$\sigma_e$') # Gráfica presión activa /
profundidad
  plt.ylim(np.max(Profs),0)
  plt.legend(fontsize=12)
  plt.ylabel('Profundidad (m)')
  plt.xlabel('Presión (kN/m2)')
```

```
  plt.title("Diagrama de  distribución de presiones")
  plt.arrow(20, pre_lat.centroid.y, -15, 0, shape='full', head_width=0.3,
color="red")
  DibujaCota((20,pre_lat.centroid.y), (20,np.max(Profs)), dist=0, lw=2,
AnchoFlecha=0.2, color="black", decimales=2)
  plt.grid()
  plt.savefig("Caso de verificación 1 (mío)", dpi=300)


  print(f"Presión activa total: {round(pre_lat.area,2)} kN/m2")
  print(f"Resultante (tomada desde la base): {round((Prof_total -
pre_lat.centroid.y),1)} m" )


# DATOS
NAF = 1.2 # m
γms = np.array([16.5,19.2]) # (m) definición de pesos especificos
Hs = np.array([1.2,4.8]) # Vector de espesores
φ = np.array([30,35]) # vector de ángulo de fricción del material

Presion_activa_Rankine(NAF, γms, Hs, φ)
```

# 9.2 CASO DE VERIFICACIÓN 2

```
import numpy as np
import matplotlib.pyplot as plt
from shapely.geometry import Polygon
from libdibujo import DibujaCota

def presion_activa_rankine(NAF, sobrecarga, γms, Hs, φ, c):
  """
  CALCULA LA PRESIÓN ACTIVA TOTAL Y SU RESULTANTE POR EL MÉTODO DE RANKINE
(Cuando se puresenta cohesión y/o sobrecargas).
  NAF, la profundidad a la que se encuentra el nivel freático.
  sobrecarga, es la magnitud de la sobrecarga en kPa.
  γms, es un array con el peso específico del material de cada estrato.
  Hs, es un array con el espesor de cada estrato.
  φ, es un array con el ángulo de fricción interna de cada material.
  c, es un array con la fuerza de cohesión de cada material en kPa.
  """

  γw = 9.81 #kN/m3
  Profs = np.cumsum(Hs)
  γe = γms.copy() # Array para almacenar los gammas efectivos
  ProfsW = np.zeros_like(Hs) # Array de ceros donde se va a indicar el NAF
  Prof_total = sum(Hs) # Variable con la profundidad total

  ka = (1 - np.sin(np.radians(φ))) / (1 + np.sin(np.radians(φ))) # Vector con los
coeficientes de presión activa de Rankine
  ProfsW = np.where(Profs > NAF, Profs - NAF, ProfsW)
  u = γw * ProfsW # Vector con la presión del agua
  γe = np.where(Profs > NAF, γms - γw, γe) # Vector con γms - γw (si es el caso)

  # Agrega a las arrays lo referente a la sobrecarga
  if sobrecarga >= 0:
    γe = np.insert(γe, 0, sobrecarga) # Agrega el valor de la sobrecarga en la
primera posición
```

```
    Hs = np.insert(Hs, 0, 1)
    u = np.insert(u, 0, 0)
    Profs = np.insert(Profs, 0, 0)
  σa = np.cumsum(Hs * γe) # Vector con en el empuje acumulado en la profundidad

  ###### REPITE LAS ARRAYS PARA HACER EL CAMBIO DE ESTRATO CON LOS KA DE LOS DOS
MATERIALES #######

  σa = np.repeat(σa, 2)
  σa = np.delete(σa, 0)
  σa = np.delete(σa, len(σa) - 1)
  u = np.repeat(u, 2)
  u = np.delete(u, 0)
  u = np.delete(u, len(u) - 1)
  Profs = np.repeat(Profs, 2)
  Profs = np.delete(Profs, 0)
  Profs = np.delete(Profs, len(Profs) - 1)
  ka = np.repeat(ka, 2)

  σe = (ka * σa) # + u # Vector con las presiones efectivas (El ejercicio del
libro desprecia la presión del agua 608 Bowles)

  ######  COHESIÓN (2 * c * √ka)  #######

  raiz_ka = ka ** (1/2)
  c = np.repeat(c, 2)
  tension = 2 * c * raiz_ka

  Empuje_activo = σe - tension # Empuje activo total

  Empuje_activo = np.insert(Empuje_activo, 0, 0) # Agrega el punto 0 en Y
  Profs = np.insert(Profs, 0, 0) # Agrega el punto 0 en X

  a = np.stack((Empuje_activo,Profs), axis=1) # Junta la matriz de profundidad
con la de presión efectivos
  point = np.array([[0, Prof_total]]) # Agrega el punto necesario para cerrar el
polígono
  b = np.vstack((a, point)) # Tuple con las coordenadas para el polígono

  pre_act_fig = Polygon(b) # Polígono representa la presión lateral

  fig, ax = plt.subplots()
  plt.title("Diagrama de  distribución de presiones")
  ax.plot(Empuje_activo, Profs, label=r'$\sigma_e$') # Gráfica presión activa /
profundidad
  ax.arrow(20, pre_act_fig.centroid.y, -20, 0, shape='full', head_width=0.5,
color="red")
  plt.ylim(np.max(Profs),0)
  plt.legend(fontsize=12)
  plt.ylabel('Profundidad (m)')
  plt.xlabel('Presión (kN/m2)')
  DibujaCota((20,pre_act_fig.centroid.y), (20,np.max(Profs)), dist=0, lw=2,
AnchoFlecha=0.35, color="black", decimales=2)
  plt.grid()
  plt.savefig("Caso de verificación 2 (mío)", dpi=300)


  presion_activa = round(pre_act_fig.area,2)
```

```
  centroide = round((Prof_total - pre_act_fig.centroid.y),2)

  return presion_activa, centroide, Profs
  # Sí toma en cuenta el área negativa para el total 547.27 + 3.53 = 550.8 kN


############### Datos ################

NAF = float(1.8)
sobrecarga = float(100)
γms = np.array([17.3,19.6,19.7,19.0,18.0])
Hs = np.array([1.8,0.6,2.75,2.45,1.5])
φ = np.array([32,0,10,0,20])
c = np.array([0.0,70.0,30.0,40.0,20.0])


presion_activa, centroide, ka = presion_activa_rankine(NAF, sobrecarga, γms, Hs,
φ, c)
print(f"Empuje activo total: {presion_activa} kN")
print(f"Resultante (tomada desde la base): {centroide} m" )
```

## 9.3 CASO DE VERIFICACIÓN 3

```
import numpy as np
import matplotlib.pyplot as plt
from shapely.geometry import Polygon
from libdibujo import DibujaCota

def presion_pasiva_rankine(NAF, γm, espesor, φ_p, c_p):

  """
  CALCULA LA PRESIÓN PASIVA TOTAL Y SU RESULTANTE POR EL MÉTODO DE RANKINE.
  NAF, es la profundidad a la que se encuentra el nivel freático.
  γm, es un array con el peso específico del material de cada estrato.
  espesor, es un array con el espesor de cada estrato.
  φ_p, es un array con el ángulo de fricción interna de cada material.
  c_p, es un array con la cohesión de cada  material.
  """

  Profs = np.cumsum(espesor) # Array con las profundidades acumuladas
  γe = γm.copy() # Array para almacenar los gammas efectivos

  Prof_total = sum(espesor) # Variable con la profundidad total

  kp = (np.tan(np.radians(45 + φ_p / 2))) ** 2 # Vector con los coeficientes de
presión activa de Rankine

  σa = np.cumsum(espesor * γe) # Vector con en el empuje acumulado en la
profundidad

  ###### REPITE LAS ARRAYS PARA HACER EL CAMBIO DE ESTRATO CON LOS kp DE LOS DOS
MATERIALES #######
  σa = np.repeat(σa, 2)
  Profs = np.repeat(Profs, 2)
  kp = np.repeat(kp, 2)

  σe = (kp * σa)
```

```
  ######  COHESIÓN (2 * c * √kp)  #######

  raiz_kp = kp ** (1/2)
  c_p = np.repeat(c_p, 2)
  tension = 2 * c_p * raiz_kp

  Empuje_pasivo = σe - tension # Empuje activo total
  pbm = np.round(Empuje_pasivo[len(Empuje_pasivo)-1], 2) # Presión pasiva en la
base del muro

  Empuje_pasivo = np.insert(Empuje_pasivo, 0, 0) # Agrega el punto 0 en Y

  Profs = np.insert(Profs, 0, 0) # Agrega el punto 0 en X

  a = np.stack((Empuje_pasivo,Profs), axis=1) # Junta la matriz de profundidac
con la de presión efectivos
  point = np.array([[0, Prof_total]]) # Crea el punto necesario para cerrar el
polígono
  b = np.vstack((a, point)) # Agrega el punto necesario para cerrar el polígono

  pre_pas_fig = Polygon(b) # Polígono de distribucion de la presión lateral
pasiva

  fig, ax = plt.subplots()
  plt.title("Diagrama de  distribución de presiones")
  ax.plot(Empuje_pasivo, Profs, label=r'$\sigma_p$') # Gráfica presión activa /
profundidad
  ax.arrow(20, pre_pas_fig.centroid.y, -20, 0, shape='full', head_width=0.1,
head_length=5, color="red")
  plt.ylim(np.max(Profs),0)
  plt.legend(fontsize=12)
  plt.ylabel('Profundidad (m)')
  plt.xlabel('Presión (kN/m2)')
  DibujaCota((20,pre_pas_fig.centroid.y), (20,np.max(Profs)), dist=0, lw=2,
AnchoFlecha=0.35, color="black", decimales=2)
  plt.grid()

  presion_pasiva = round(pre_pas_fig.area,2)
  centroide = round((Prof_total - pre_pas_fig.centroid.y),2)

  return presion_pasiva, centroide, pbm


############### Datos ###############

NAF = float(2.45) # m
γm = np.array([16.67]) # kN/m3
espesor = np.array([2.45]) # m
φ_p = np.array([33])
c_p = np.array([0.0])

presion_pasiva, centroide, pbm = presion_pasiva_rankine(NAF, γm, espesor, φ_p,
c_p)
print(f"Presión pasiva total: {presion_pasiva} kN")
print(f"Resultante (tomada desde la base): {centroide} m" )
print(f"La presión pasiva en la base del muro es de: {pbm} kN/m2")
```

# 9.4 CASO DE VERIFICACIÓN 4

```
import numpy as np
import matplotlib.pyplot as plt
from shapely.geometry import Polygon
from libdibujo import DibujaCota

def prelat_sobrecarga_areacargada1(B, W, q, μ, NSQW,  NSQL, HTWALL, DTWALL,
NVERT):
  """
  CALCULA LA PRESIÓN LATERAL DEBIDO A UNA SOBRECARGA CON LA FORMA DE UN ÁREA
CARGADA CON   MAGNITUD CONSTANTE.
  B, es el ancho del área cargada en m.
  W, es el largo del área cargada en m.
  q, es la magnitud de la carga en kPa.
  μ, es la relación de Poisson.
  NSQW, es el número de divisiones del área cargada a lo ancho.
  NSQL, es el número de divisiones del área cargada a lo largo.
  HTWALL, es la altura del muro de contención en m.
  DTWALL, es la distancia entre el muro de contención y el área cargada en m.
  NVERT, el número de divisiones de la altura del muro a las que se calculará la
presión lateral.
  """

  DDY = HTWALL / (NVERT - 1) # Incremento de la distancia vertical
  DOP = W / 2 # Distancia al centro de la carga en el eje y

  B1 = B / NSQW # Dimensiones del área eje x
  W1 = W / NSQL # Dimensiones del área eje y

  P = q * (B1 * W1) # Carga por área unitaria (kN)

  # Crea un arreglo separando el largo y el ancho entre el número de las
separaciones
  y = np.arange(B1 / 2, B, B1)
  x = np.arange(W1 / 2, W, W1)
  z = np.arange(0, HTWALL + DDY, DDY)
  z = np.where(z == 0, 0.000000001, z) # Si dejamos z = 0 y μ = 0.5, se hace
división entre 0 y da error

  xx, yy = np.meshgrid(x, y) # Crea el grid con las coordenadas de las áreas de
carga unitarias

  r = ((abs(DOP - xx) ** 2) + (DTWALL + yy) ** 2) ** (1/2)

  σr_polygon = np.array([]) # Array vacía para almacenar la presión a cada
profundidad
  for l in range(len(z)): # Calcular para cada profundidad

    R = (r ** 2 + z[l] ** 2) ** (1/2)
    R = np.ravel(R) # Transforma la array (10,10) a (100, )
    r = np.ravel(r) # Transforma la array (10,10) a (100, )

    θ = np.arccos(z[l] / R)

    term1 =  3 * np.sin(θ) ** 2 * np.cos(θ) ** 3
    term2 = (1 - 2 * μ) * np.cos(θ) ** 2  / (1 + np.cos(θ))
```

```
    σr = (P / (2 * np.pi * z[1] ** 2)) * (term1 - term2)
    σr_total = np.sum(σr) # Suma de todas las cargas de las áras unitarias en
determinada profundidad z
    σr_polygon = np.append(σr_polygon, σr_total)

  σr_polygon = np.where(σr_polygon < 0, 0, σr_polygon) # Hacer 0 el primer punto
que sale negativo

  t = np.stack((σr_polygon, z), axis=1) # Junta las arrays de presiones y
profundidades
  t = np.vstack((t, [0, HTWALL])) # Agregar el punto necesario para cerrar el
polígono
  p = Polygon(t) # Crea el polígono

  fig, ax = plt.subplots()
  plt.title("Diagrama de  distribución de presiones")
  ax.plot(σr_polygon, z, label=r'$\sigma_r$') # Gráfica presión lateral por
sobrecarga / profundidad
  ax.set_ylim(HTWALL, 0)  # Eje Y en orden descendiente
  ax.arrow(10, p.centroid.y, -4, 0, shape='full', head_width=0.3, color="red")
  plt.legend(fontsize=14)
  plt.ylabel('Profundidad (m)')
  plt.xlabel('Presión lateral (kN)')
  DibujaCota((10,p.centroid.y), (10,np.max(z)), dist=0, lw=2, AnchoFlecha=0.2,
color="black", decimales=2)
  plt.grid()

  prelat = round(p.area, 2)
  centroide = round(HTWALL - p.centroid.y, 2)
  return prelat, centroide


############    DATOS    ############    CON FÓRMULA 1    #### la del theta
B = 2 # (m)
W = 4 # (m)
q = 200 # (kPa)
μ = 0.5 # Relación de Poisson
NSQW = 10 # Número de áreas a  lo ancho
NSQL = 10 # Número de áreas a lo largo
HTWALL = 7.5 # (m)
DTWALL = 3 # (m)
NVERT = 11 # Número de puntos en el eje z (segmentos = NVERT - 1)

prelat, centroide = prelat_sobrecarga_areacargada1(B, W, q, μ, NSQW,  NSQL,
HTWALL, DTWALL, NVERT)

print(f"La presión lateral generada por la sobrecarga es de: {prelat} kN")
print(f"La resultante de la presión lateral está en: {centroide} m desde la
base")
def prelat_sobrecarga_areacargada1(B, W, q, μ, NSQW,  NSQL, HTWALL, DTWALL,
NVERT):

  DDY = HTWALL / (NVERT - 1) # Incremento de la distancia vertical
  DOP = W / 2 # Distancia al centro de la carga en el eje y

  B1 = B / NSQW # Dimensiones del área eje x
  W1 = W / NSQL # Dimensiones del área eje y
```

```
  P = q * (B1 * W1) # Carga por área unitaria (kN)

  y = np.arange(B1 / 2, B, B1)
  x = np.arange(W1 / 2, W, W1)
  z = np.arange(0, HTWALL + DDY, DDY)
  z = np.where(z == 0, 0.000000001, z) # Si dejamos z = 0 y μ = 0.5, se hace
división entre 0 y da error

  xx, yy = np.meshgrid(x, y) # Crea el grid con las coordenadas de las áreas de
carga unitarias

  r = ((abs(DOP - xx) ** 2) + (DTWALL + yy) ** 2) ** (1/2)

  σr_polygon = np.array([]) # Array vacía para almacenar la presión a cada
profundidad
  for l in range(len(z)): # Calcular para cada profundidad

    R = (r ** 2 + z[l] ** 2) ** (1/2)
    R = np.ravel(R) # Transforma la array (10,10) a (100, )
    r = np.ravel(r) # Transforma la array (10,10) a (100, )

    θ = np.arccos(z[l] / R)

    term1 =  3 * np.sin(θ) ** 2 * np.cos(θ) ** 3
    term2 = (1 - 2 * μ) * np.cos(θ) ** 2  / (1 + np.cos(θ))
    σr = (P / (2 * np.pi * z[l] ** 2)) * (term1 - term2)
    σr_total = np.sum(σr) # Suma de todas las cargas de las áras unitarias en
determinada profundidad z
    σr_polygon = np.append(σr_polygon, σr_total)

  σr_polygon = np.where(σr_polygon < 0, 0, σr_polygon) # Hacer 0 el primer punto
que sale negativo

  t = np.stack((σr_polygon, z), axis=1) # Junta las arrays de presiones y
profundidades
  t = np.vstack((t, [0, HTWALL])) # Agregar el punto necesario para cerrar el
polígono
  p = Polygon(t) # Crea el polígono

  fig, ax = plt.subplots()
  plt.title("Diagrama de  distribución de presiones")
  ax.plot(σr_polygon, z, label=r'$\sigma_r$') # Gráfica presión lateral por
sobrecarga / profundidad
  ax.set_ylim(HTWALL, 0)  # Eje Y en orden descendiente
  ax.arrow(10, p.centroid.y, -4, 0, shape='full', head_width=0.3, color="red")
  plt.legend(fontsize=14)
  plt.ylabel('Profundidad (m)')
  plt.xlabel('Presión lateral (kN)')
  DibujaCota((10,p.centroid.y), (10,np.max(z)), dist=0, lw=2, AnchoFlecha=0.2,
color="black", decimales=2)
  plt.grid()

  prelat = round(p.area, 2)
  centroide = round(HTWALL - p.centroid.y, 2)
  return prelat, centroide


###########    DATOS   ###########    CON FÓRMULA 1   #### la del theta
```

```
B = 2 # (m)
W = 4 # (m)
q = 200 # (kPa)
μ = 0.5 # Relación de Poisson
NSQW = 10 # Número de áreas a  lo ancho
NSQL = 10 # Número de áreas a lo largo
HTWALL = 7.5 # (m)
DTWALL = 3 # (m)
NVERT = 11 # Número de puntos en el eje z (segmentos = NVERT - 1)

prelat, centroide = prelat_sobrecarga_areacargada1(B, W, q, μ, NSQW,  NSQL,
HTWALL, DTWALL, NVERT)

print(f"La presión lateral generada por la sobrecarga es de: {prelat} kN")
print(f"La resultante de la presión lateral está en: {centroide} m desde la
base")
```

## 9.5 CASO DE VERIFICACIÓN 5

```
import numpy as np
import matplotlib.pyplot as plt
from shapely.geometry import Polygon
from libdibujo import DibujaCota

def prelat_sobrecarga_variacionlineal(B, W, q, μ, NSQW,  NSQL, HTWALL, DTWALL,
NVERT):
  """
  CALCULA LA PRESIÓN LATERAL DEBIDO A UNA SOBRECARGA CON LA FORMA DE UN ÁREA
CARGADA CON VARIACIÓN LINEAL
  B, es el ancho del área cargada en m.
  W, es el largo del área cargada en m.
  q, es la magnitud de la carga en su punto máximo en kPa.
  μ, es la relación de Poisson.
  NSQW, es el número de divisiones del área cargada a lo ancho.
  NSQL, es el número de divisiones del área cargada a lo largo.
  HTWALL, es la altura del muro de contención en m.
  DTWALL, es la distancia entre el muro de contención y el área cargada en m.
  NVERT, el número de divisiones de la altura del muro a las que se calculará la
presión lateral.
  """

  DDY = HTWALL / (NVERT - 1) # Incremento de la distancia vertical
  DOP = W / 2 # Distancia al centro en el eje y

  B1 = B / NSQW # Dimensiones del área eje x
  W1 = W / NSQL # Dimensiones del área eje y

  P = (q * (B * W)) / 2 # Carga de todas las strips juntas

  y = np.arange(B1 / 2, B, B1)
  x = np.arange(W1 / 2, W, W1)
  z = np.arange(0, HTWALL + DDY, DDY)
  z = np.where(z == 0, 0.000000001, z) # Si dejamos z = 0 y μ = 0.5, se hace
división entre 0 y da error
```

```
  xx, yy = np.meshgrid(x, y) # Crea el grid con las coordenadas de los
centroides de las áreas de carga unitarias

  nq = yy[:, 0] * 10 # Calcula las nqs

  q_unit = P / (sum(nq) * B1 * W1 * 10) # Carga unitaria

  strip_intensities = np.array([]) # Array vacía para almacenar las strip
intensities
  strip_intensities = nq * B1 * W1 * q_unit
  # strip_intensities = np.flip(strip_intensities)
################################## COMO ESTA LINEA SE CALCULA INVERTIDO
##############################
  strip_intensities = np.repeat(strip_intensities, 10) # Convierte la array
(10,0) a (100,0) para que cada carga unitaria tenga su intensidad

  r = ((abs(DOP - xx) ** 2) + (DTWALL + yy) ** 2) ** (1/2)

  σr_polygon = np.array([]) # Array vacía para almacenar la presión a cada
profundidad
  for l in range(len(z)): # Calcular para cada profundidad

    R = (r ** 2 + z[l] ** 2) ** (1/2)
    R = np.ravel(R) # Transforma la array (10,10) a (100, )
    r = np.ravel(r) # Transforma la array (10,10) a (100, )

    θ = np.arccos(z[l] / R)

    term1 =  3 * np.sin(θ) ** 2 * np.cos(θ) ** 3
    term2 = (1 - 2 * μ) * np.cos(θ) ** 2  / (1 + np.cos(θ))
    σr = (strip_intensities / (2 * np.pi * z[l] ** 2)) * (term1 - term2)
    σr_total = np.sum(σr) # Suma de todas las cargas de las áras unitarias en
determinada profundidad z
    σr_polygon = np.append(σr_polygon, σr_total)

  σr_polygon = np.where(σr_polygon < 0, 0, σr_polygon) # Hacer 0 el primer punto
que sale negativo

  t = np.stack((σr_polygon, z), axis=1) # Junta las arrays de presiones y
profundidades
  t = np.vstack((t, [0, HTWALL])) # Agregar el punto necesario para cerrar el
polígono
  p = Polygon(t) # Crea el polígono

  fig, ax = plt.subplots()
  plt.title("Diagrama de  distribución de presiones")
  ax.plot(σr_polygon, z, label=r'$\sigma_r$') # Gráfica presión lateral por
sobrecarga / profundidad
  ax.set_ylim(HTWALL, 0)  # Eje Y en orden descendiente
  ax.arrow(3, p.centroid.y, -2, 0, shape='full', head_width=0.2, color="red")
  plt.legend(fontsize=14)
  plt.ylabel('Profundidad (m)')
  plt.xlabel('Presión lateral (kN)')
  DibujaCota((3,p.centroid.y), (3,np.max(z)), dist=0, lw=2, AnchoFlecha=0.1,
color="black", decimales=2)
  plt.grid()

  prelat = round(p.area, 2)
```

```
  centroide = round(HTWALL - p.centroid.y, 2)
  return prelat, centroide

###########    DATOS   ###########
B = 2 # (m)
W = 4 # (m)
q = 200 # (kPa)
μ = 0.5 # Relación de Poisson
NSQW = 10 # Número de áreas a  lo ancho
NSQL = 10 # Número de áreas a lo largo
HTWALL = 7.5 # (m)
DTWALL = 3 # (m)
NVERT = 11 # Número de puntos en el eje z (segmentos = NVERT - 1)

prelat, centroide = prelat_sobrecarga_variacionlineal(B, W, q, μ, NSQW,  NSQL,
HTWALL, DTWALL, NVERT)
print(f"La presión lateral generada por la sobrecarga es de: {prelat} kN")
print(f"La resultante de la presión lateral está en: {centroide} m desde la
base")
```

# 9.6 CASO DE VERIFICACIÓN 6

```
def presion_activa_rankine(sobrecarga, γms, Hs, φ, c, β):
  """
  CALCULA LA PRESIÓN ACTIVA TOTAL Y SU RESULTANTE POR EL MÉTODO DE RANKINE.
  sobrecarga, es la magnitud de la sobrecarga en kPa.
  γms, es un array con el peso específico del material de cada estrato.
  Hs, es un array con el espesor de cada estrato.
  φ, es un array con el ángulo de fricción interna de cada material.
  c, es un array con la fuerza de cohesión de cada material en kPa.
  β, es el ángulo del relleno por encima del muro.
  """

  γw = 9.81 #kN/m3
  Profs = np.cumsum(Hs)
  γe = γms.copy() # Array para almacenar los gammas efectivos
  Prof_total = sum(Hs) # Variable con la profundidad total

  β = np.radians(β) # Ángulo de inclinación del relleno por encima del muro
  φ = np.radians(φ) # Transforma a radianes para poder utilizar las funciones
trigonométricas
  ka = np.array([])
  # Calcula los coeficientes de presión activa dependiendo de que término es
mayor, se usa una fórmula o otra (Pag 489 Kramer)
  for i in range(len(φ)):
    if β >= φ[i]:
      a = (1 - np.sin(φ[i])) / (1 + np.sin(φ[i]))
      ka = np.append(ka, a)
    elif β < φ[i]:
      nume = (np.cos(β) - ((np.cos(β)) ** 2 - (np.cos(φ[i])) ** 2) ** (1/2))
      deno = (np.cos(β) + ((np.cos(β)) ** 2 - (np.cos(φ[i])) ** 2) ** (1/2))
      a = np.cos(β) * (nume / deno)
      ka = np.append(ka, a)

  # Agrega a las arrays lo referente a la sobrecarga
  if sobrecarga >= 0:
```

```
    γe = np.insert(γe, 0, sobrecarga) # Agrega el valor de la sobrecarga en la primera posición
    Hs = np.insert(Hs, 0, 1)
    Profs = np.insert(Profs, 0, 0)
  σa = np.cumsum(Hs * γe) # Vector con en el empuje acumulado en la profundidad

  ###### REPITE LAS ARRAYS PARA HACER EL CAMBIO DE ESTRATO CON LOS KA DE LOS DOS MATERIALES #######

  σa = np.repeat(σa, 2)
  σa = np.delete(σa, 0)
  σa = np.delete(σa, len(σa) - 1)
  Profs = np.repeat(Profs, 2)
  Profs = np.delete(Profs, 0)
  Profs = np.delete(Profs, len(Profs) - 1)
  ka = np.repeat(ka, 2)

  σe = (ka * σa)

  ######  COHESIÓN (2 * c * √ka)  #######

  raiz_ka = ka ** (1/2)
  c = np.repeat(c, 2)
  tension = 2 * c * raiz_ka

  Presion_activa = σe - tension # Empuje activo total

  Presion_activa = Presion_activa * np.cos(β) # Componente horizontal del empuje activo de Rankine

  Presion_activa = np.insert(Presion_activa, 0, 0) # Agrega el punto 0 en Y
  Profs = np.insert(Profs, 0, 0) # Agrega el punto 0 en X

  a = np.stack((Presion_activa, Profs), axis=1) # Junta la matriz de profundidac con la de presión efectivos
  point = np.array([[0, Prof_total]]) # Agrega el punto necesario para cerrar el polígono
  b = np.vstack((a, point)) # Tuple con las coordenadas para el polígono

  pre_act_fig = Polygon(b) # Polígono representa la presión lateral

  fig, ax = plt.subplots()
  plt.title("Diagrama de  distribución de presiones")
  ax.plot(Presion_activa, Profs) # Gráfica presión activa / profundidad
  ax.arrow(20, pre_act_fig.centroid.y, -10, 0, shape='full', head_width=0.25, color="red")
  plt.ylim(np.max(Profs),0)
  plt.ylabel('Profundidad (m)')
  plt.xlabel('Presión (kN/m2)')
  DibujaCota((20,pre_act_fig.centroid.y), (20,np.max(Profs)), dist=0, lw=2, AnchoFlecha=0.1, color="black", decimales=2)
  plt.grid()
  plt.savefig("Caso de verificación 6 (mío)", dpi=300)

  Empuje_activo = round(pre_act_fig.area,2)
  centroide = round((Prof_total - pre_act_fig.centroid.y),2)
  return Empuje_activo, centroide
```

```
def presion_activa_okabe(sobrecarga, γms, Hs, φ, c, β, θ, A0, Presion_activa,
resultante):
  """
  CALCULA LA PRESIÓN ACTIVA DEBIDO A SISMO POR EL MÉTODO DE MONONOBE-OKABE
  sobrecarga, es la magnitud de la sobrecarga en kPa
  γms, es un array con el peso específico del material de cada estrato
  Hs, es un array con el espesor de cada estrato
  φ, es un array con el ángulo de fricción interna de cada material
  c, es un array con la fuerza de cohesión de cada material en kPa
  β, es el ángulo que tiene el relleno por encima del muro de contención.
  A0, es la aceleración máxima en roca en la base del muro.
  Presión_activa, es la presión activa total.
  Resultante, es la ubicación de la resultante de la presión activa total
  """

  Profs = np.cumsum(Hs)
  γe = γms.copy() # Array para almacenar los gammas efectivos
  Prof_total = sum(Hs) # Variable con la profundidad total

  β = np.radians(β) # Ángulo de inclinación del relleno por encima del muro
  φ = np.radians(φ) # Transforma a radianes para poder utilizar las funciones
trigonométricas de numpy

  # Kh = 2/3 * A0 # Coeficiente sísmico horizontal
  Kh = 0.15
  # Kv = 2/3 * Kh # Coeficiente sísmico vertical
  Kv = 0.075
  θ = np.radians(θ) # Ángulo de la cara interna del muro respecto a la vertical
(Es 0 si el muro no se encuentra inclinado)
  δ = np.radians(np.degrees(φ) / 2) # Ángulo de fricción entre el material de
relleno y la estructura de contención

  ψ = np.arctan(Kh / (1 - Kv))

  # Ecuación del coeficiente dinámico de presión activa resulta por partes
  numerador = (np.cos(φ - θ - ψ)) ** 2
  deno1 = np.cos(ψ) * ((np.cos(θ)) ** 2) * np.cos(δ + θ + ψ)
  deno2 = (1 + ((np.sin(φ + δ) * np.sin(φ - β - ψ)) / (np.cos(δ + θ + ψ) *
np.cos(β - θ))) ** (1/2)) ** 2

  kae = numerador / (deno1 * deno2)

  # Agrega a las arrays lo referente a la sobrecarga
  if sobrecarga >= 0:
    γe = np.insert(γe, 0, sobrecarga) # Agrega el valor de la sobrecarga en la
primera posición
    Hs = np.insert(Hs, 0, 1)
    Profs = np.insert(Profs, 0, 0)
  σa = np.cumsum(Hs * γe) # Vector con en el empuje acumulado en la profundidad

  ###### REPITE LAS ARRAYS PARA HACER EL CAMBIO DE ESTRATO CON LOS kae DE LOS
DOS MATERIALES #######

  δ = np.repeat(δ, 2)

  σa = np.repeat(σa, 2)
  σa = np.delete(σa, 0)
  σa = np.delete(σa, len(σa) - 1)
```

```
  Profs = np.repeat(Profs, 2)
  Profs = np.delete(Profs, 0)
  Profs = np.delete(Profs, len(Profs) - 1)
  kae = np.repeat(kae, 2)

  σe = (kae * σa)

  ######  COHESIÓN (2 * c * √kae)  #######

  raiz_kae = kae ** (1/2)
  c = np.repeat(c, 2)
  tension = 2 * c * raiz_kae

  Presion_sismo = σe - tension # Empuje activo total

  Presion_sismo = (Presion_sismo) * (1 - Kv) # Componente horizontal del empuje
activo de Rankine

  Presion_sismo = np.insert(Presion_sismo, 0, 0) # Agrega el punto 0 en Y
  Profs = np.insert(Profs, 0, 0) # Agrega el punto 0 en X

  a = np.stack((Presion_sismo, Profs), axis=1) # Junta la matriz de profundidac
con la de presión efectivos
  point = np.array([[0, Prof_total]]) # Agrega el punto necesario para cerrar el
polígono
  b = np.vstack((a, point)) # Tuple con las coordenadas para el polígono

  pre_sismo = Polygon(b) # Polígono representa la presión lateral

  fig, ax = plt.subplots()
  plt.title("Diagrama de  distribución de presiones")
  ax.plot(Presion_sismo, Profs) # Gráfica presión activa / profundidad
  ax.arrow(15, max(Profs) - round(0.6 * max(Profs)), -10, 0, shape='full',
head_width=0.25, color="red")
  plt.ylim(np.max(Profs),0)
  plt.ylabel('Profundidad (m)')
  plt.xlabel('Presión (kN/m2)')
  DibujaCota((15,max(Profs) - round(0.6 * max(Profs))), (15,np.max(Profs)),
dist=0, lw=2, AnchoFlecha=0.1, color="black", decimales=2)
  plt.grid()

  Pae = round(pre_sismo.area,2)
  resultante_sismo = round(0.6 * max(Profs))

  ΔEa = Pae - Pa
  print(f"El incremento del empuje activo debido a sismo es de: {ΔEa} kN")
  Resultante_ambas = (Pa * resultante + ΔEa * resultante_sismo) / Pae
  print(f"Resultante total (tomada desde la base){round(Resultante_ambas, 2)}
m")
  # Momento sísmico
  Me = float(Pae * Resultante_ambas * np.cos(np.radians(φ) / 2)) # Es cos(δ) que
es igual a cos(φ / 2)
  print(f"El momento de volteo debido a la fuerza sísmica es de: {round(Me, 2)}
kN.m")

  return Pae, resultante_sismo
```

```
############### Datos ################
sobrecarga = float(0)
γms = np.array([1.76*9.81])
Hs = np.array([5])
φ = np.array([34])
c = np.array([0])
β = 0

θ = 0
A0 = 0 # Aceleración máxima en roca (A0 veces la aceleración de la gravedad)

Pa, resultante = presion_activa_rankine(sobrecarga, γms, Hs, φ, c, β)
print(f"Empuje activo total: {Pa} kN")
print(f"Resultante (tomada desde la base del muro): {resultante} m" )

Pae, resultante_sismo = presion_activa_okabe(sobrecarga, γms, Hs, φ, c, β, θ, A0,
Presion_activa, resultante)
print(f"Empuje activo total debido a sismo: {Pae} kN")
print(f"Resultante Pae (tomada desde la base): {resultante_sismo} m" )


ΔEa = Pae - Pa
print(f"El incremento de la presión activa debido a sismo es de: {round(ΔEa,2)}
kN")

print(f"El empuje total debido a sismo es de  {round(Pae, 2)} kN ")

Resultante_ambas = (Pa * resultante + ΔEa * resultante_sismo) / Pae
print(f"La resultante de la presión activa debido a sismo se encuentra en
{round(Resultante_ambas,2)} m")

# Momento de volteo debido al sismo
Me = float(Pae * Resultante_ambas * np.cos(np.radians(φ) / 2)) # Es cos(δ) que es
igual a cos(φ / 2)
print(f"El momento sísmico es igual a {round(Me, 2)} kN/m m")
```

## 9.7 CASO DE VERIFICACIÓN 7

```
import numpy as np

def Capacidad_de_carga_Vesic(NAF, Prof_emplazamiento, B, L, c, φ, α, in_suelo,
γ):
  """
  Calcula la capacidad de carga última.
  NAF, es la profundidad del nivel de aguas freáticas.
  Prof_emplazamiento, es la profundidad de desplante de la zapata.
  B, es el ancho de la zapata.
  L,  es el largo de la zapata.
  c, es la cohesión del material.
  φ, es el ángulo de fricción interna del material.
  α, es el ángulo de la inclinación de la base de la zapata.
  in_suelo, ángulo de inclinación del suelo.
  γ, es el peso específico del material.
  """

  γw = 9.81 # Peso específico del agua kN/m3
```

```
  # Influencia del NAF
  if NAF <= Prof_emplazamiento:
    γe = γ - γw
  elif (Prof_emplazamiento < NAF) and (NAF < Prof_emplazamiento + B):
    γe = γ - γw * (1 - ((NAF - Prof_emplazamiento) / B))
  elif Prof_emplazamiento + B <= NAF:
    γe = γ

  σ_zD = γ * Prof_emplazamiento

  # Depth factors
  k = Prof_emplazamiento / B
  dc = 1 + 0.4 * k
  dq = 1 + 2 * k * np.tan(np.radians(φ)) * (1 - np.sin(np.radians(φ))) ** 2
  dγ = 1

  # Bearing capacity factors
  Nq = ((np.exp(1) ** (np.pi * np.tan(np.radians(φ)))) * (np.tan(np.radians(45 +
(φ/2)))) ** 2)
  if φ == 0:
    Nc = 5.14
  elif φ > 0:
    Nc = (Nq - 1) / np.tan(np.radians(φ))
  Nγ = 2 * (Nq + 1) * np.tan(np.radians(φ))

  # Shape factors
  Sc = 1 + (B / L) * (Nq / Nc)
  Sq = 1 + (B / L) * np.tan(np.radians(φ))
  Sγ = 1 - 0.4 * (B / L)

  # Base inclination factors
  if α == 0:
    bc = 1
    bq = 1
    bγ = 1
  elif α > 0:
    bc = 1 - α / 147 , 2
    bq = (1 - (α * np.tan(np.radians(φ))) / 57) ** 2
    bγ = bq

  # Ground inclination factors (Cuando se encuentra cerca de la orilla de un
talud (no es el caso))
  if in_suelo == 0:
    gc = 1
    gq = 1
    gγ = 1
  elif in_suelo > 0:
    gc = (1 - in_suelo / 147)
    gq = (1 - np.tan(np.radians(in_suelo))) ** 2
    gγ = gq

  # Load inclination factors
  ic = 1
  iq = 1
  iγ = 1

  qn = c * Nc * Sc * dc * ic * bc * gc + σ_zD * Nq * Sq * dq * iq * bq * gq +
0.5 * γe * B * Nγ * Sγ * dγ * iγ * bγ * gγ
```

```
    return qn


##### Datos #####
NAF = 1.5 # m
Prof_emplazamiento = 1 # m
B = 3 # m
L = 3 # m

c = 0 # Cohesión del material
φ = 30 # Ángulo del fricción del material debajo del muro (Tomo el estrato de
hasta abajo)
α = 0 # Ángulo de inclinación de la base del muro
in_suelo = 0 # Ángulo de inclinación del suelo
γ = 18.5 # kN/m3
γw = 9.81 # Peso específico del agua kN/m3

qn = Capacidad_de_carga_Vesic(NAF, Prof_emplazamiento, B, L, c, φ, α, in_suelo,
γ, γw)
print(f"La capacidad de carga última es de {round(qn, 2)} kPa")
```

# 9.8 CÓDIGO DE LA HERRAMIENTA DE CÁLCULO

```
import numpy as np
import matplotlib.pyplot as plt
from shapely.geometry import Polygon

import ipywidgets as ipw
from google.colab import widgets, files
from ipywidgets import Layout

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import sys
sys.path.insert(0, "/content/drive/MyDrive/Tesis_python")
from libdibujo import DibujaCota, Poligono

class Empuje_muros():

  def __init__(self, NAF, sobrecarga, γms, Hs, φ, c, β, B, W, q, μ, NSQW, NSQL,
HTWALL, DTWALL, NVERT, P,
               γm_p, Hs_p, φ_p, c_p, A0, a_o_d, Hay_NAF):

    # Para la presión activa
    self.NAF = NAF
    self.sobrecarga = sobrecarga # Magnitud de la sobrecarga
    self.γms = γms # Peso específico del material
    self.Hs = Hs # Espesor del estrato
    self.φ = φ # Ángulo de fricción interna del material
    self.c = c # Cohesión del material
    self.γw = 9.81 #kN/m3
    self.β = β # Ángulo de inclinación del relleno por encima del muro
    self.Hay_NAF = Hay_NAF # Para ver si el muro esta dreanado y tomar en cuenta
el γ sumergido y la u

    # Para la presión pasiva
    self.γm_p = γm_p
    self.Hs_p = Hs_p
    self.φ_p = φ_p
    self.c_p = c_p

    #Para la(s) sobrecargas(s)
    self.B = B
    self.W = W
    self.q = q
    self.μ = μ
    self.NSQW = NSQW
    self.NSQL = NSQL
    self.HTWALL = HTWALL
    self.DTWALL = DTWALL
    self.NVERT = NVERT
    self.P = P # Magnitud de la sobrecarga puntual
    self.a_o_d = a_o_d # Para la de variación lineal si es ascendente o
descendente
```

```
    # Para el sismo
    self.A0 = A0



  def presion_activa_rankine(self):
    """
    CALCULA LA PRESIÓN ACTIVA TOTAL Y SU RESULTANTE POR EL MÉTODO DE RANKINE.
    Puede tomar en cuenta:
    - Rellenos con cohesión.
    - Sobrecargas sobre el muro.
    - Relleno con una inclinación por encima del muro
    """

    γw = 9.81 #kN/m3
    Profs = np.cumsum(self.Hs)
    γe = self.γms.copy() # Array para almacenar los gammas efectivos
    ProfsW = np.zeros_like(self.Hs) # Array de ceros donde se va a indicar el
NAF
    Prof_total = sum(self.Hs) # Variable con la profundidad total


    β = np.radians(self.β) # Ángulo de inclinación del relleno por encima del
muro
    φ = np.radians(self.φ) # Transforma a radianes para poder utilizar las
funciones trigonométricas

    # Calcula los coeficientes de presión activa dependiendo de que término es
mayor, se usa una fórmula o otra (Pag 489 Kramer)
    ka = np.array([])
    for i in range(len(φ)):
      if β >= φ[i]:
        a = (1 - np.sin(φ[i])) / (1 + np.sin(φ[i]))
        ka = np.append(ka, a)
      elif β < φ[i]:
        nume = (np.cos(β) - ((np.cos(β)) ** 2 - (np.cos(φ[i])) ** 2) ** (1/2))
        deno = (np.cos(β) + ((np.cos(β)) ** 2 - (np.cos(φ[i])) ** 2) ** (1/2))
        a = np.cos(β) * (nume / deno)
        ka = np.append(ka, a)

    if self.Hay_NAF == 1:
      pass
    elif self.Hay_NAF == 2:
      ProfsW = np.where(Profs > self.NAF, Profs - self.NAF, ProfsW)
      γe = np.where(Profs > self.NAF, self.γms - self.γw, γe) # Vector con γms -
γw (si es el caso)

    u = self.γw * ProfsW # Vector con la presión del agua

    # Agrega a las arrays lo referente a la sobrecarga
    γe = np.insert(γe, 0, self.sobrecarga) # Agrega el valor de la sobrecarga en
la primera posición
    Hs = np.insert(self.Hs, 0, 1)
    u = np.insert(u, 0, 0)
    Profs = np.insert(Profs, 0, 0)
    σa = np.cumsum(Hs * γe) # Vector con en el empuje acumulado en la
profundidad
```

```
    ###### REPITE LAS ARRAYS PARA HACER EL CAMBIO DE ESTRATO CON LOS KA DE LOS
DOS MATERIALES #######

    σa = np.repeat(σa, 2)
    σa = np.delete(σa, 0)
    σa = np.delete(σa, len(σa) - 1)
    u = np.repeat(u, 2)
    u = np.delete(u, 0)
    u = np.delete(u, len(u) - 1)
    Profs = np.repeat(Profs, 2)
    Profs = np.delete(Profs, 0)
    Profs = np.delete(Profs, len(Profs) - 1)
    ka = np.repeat(ka, 2)
    c = np.repeat(self.c, 2)

    if self.Hay_NAF == 1:
      σe = (ka * σa)
    elif self.Hay_NAF == 2:
      σe = (ka * σa) + u # Vector con las presiones efectivas (El ejercicio del
libro desprecia la presión del agua)

    ######  COHESIÓN (2 * c * √ka)  #######

    raiz_ka = ka ** (1/2)
    tension = 2 * c * raiz_ka

    Presion_activa = σe - tension # Empuje activo total

    Presion_activa = np.insert(Presion_activa, 0, 0) # Agrega el punto 0 en Y
    Profs = np.insert(Profs, 0, 0) # Agrega el punto 0 en X

    i = np.stack((Presion_activa, Profs), axis=1) # Junta la matriz de
profundidac con la de presión efectivos
    point = np.array([[0, Prof_total]]) # Agrega el punto necesario para cerrar
el polígono
    b = np.vstack((i, point)) # Tuple con las coordenadas para el polígono

    a = Polygon(b) # Polígono representa la presión lateral

    self.Empuje_activo = round(a.area,2)
    self.Resultante_EA = round((Prof_total - a.centroid.y),2)

    # Para graficar
    self.X_PA = Presion_activa
    self.Y_PA = Profs
    self.Poli_PA = a
    self.Text_PA = "debido a presión activa"

    self.empuje_Rankine = b # Almacena la variable donde está la gráfica para
poder usarla en otros métodos.

    return self.Empuje_activo, self.Resultante_EA



  def presion_pasiva_rankine(self):
    """
```

```
    CALCULA LA PRESIÓN PASIVA TOTAL Y SU RESULTANTE POR EL MÉTODO DE RANKINE.
    """

    Profs = np.cumsum(self.Hs_p) # Array con las profundidades acumuladas
    γe = self.γm_p.copy() # Array para almacenar los gammas efectivos

    Prof_total = sum(self.Hs_p) # Variable con la profundidad total

    kp = (np.tan(np.radians(45 + self.φ_p / 2))) ** 2 # Vector con los
coeficientes de presión activa de Rankine

    σa = np.cumsum(self.Hs_p * γe) # Vector con en el empuje acumulado en la
profundidad

    ###### REPITE LAS ARRAYS PARA HACER EL CAMBIO DE ESTRATO CON LOS kp DE LOS
DOS MATERIALES #######
    σa = np.repeat(σa, 2)
    Profs = np.repeat(Profs, 2)
    kp = np.repeat(kp, 2)

    σe = (kp * σa) #+ u # Vector con las presiones efectivas (El ejercicio del
libro desprecia la presión del agua)

    ######  COHESIÓN (2 * c * √kp)  #######

    raiz_kp = kp ** (1/2)
    self.c_p = np.repeat(self.c_p, 2)
    tension = 2 *self.c_p * raiz_kp

    Empuje_pasivo = σe - tension # Empuje activo total

    Empuje_pasivo = np.insert(Empuje_pasivo, 0, 0) # Agrega el punto 0 en Y
    Profs = np.insert(Profs, 0, 0) # Agrega el punto 0 en X

    a = np.stack((Empuje_pasivo, Profs), axis=1) # Junta la matriz de
profundidac con la de presión efectivos
    point = np.array([[0, Prof_total]]) # Agrega el punto necesario para cerrar
el polígono
    b = np.vstack((a, point)) # Tuple con las coordenadas para el polígono

    pre_pas_fig = Polygon(b) # Polígono representa la presión lateral

    # Para graficar
    self.X_PP = Empuje_pasivo
    self.Y_PP = Profs
    self.Poli_PP = pre_pas_fig
    self.Text_PP = "debido a presión pasiva"

    Presion_pasiva = round(pre_pas_fig.area,2)
    Resultante_EP = round((Prof_total - pre_pas_fig.centroid.y),2)

    return Presion_pasiva, Resultante_EP



  def presion_activa_okabe(self):
    """
```

```
    CALCULA EL INCREMENTO DE LA PRESIÓN ACTIVA DEBIDO A SISMO POR EL MÉTODO DE
MONONOBE-OKABE.
    """

    Profs = np.cumsum(self.Hs)
    γe = self.γms.copy() # Array para almacenar los gammas efectivos
    Prof_total = sum(self.Hs) # Variable con la profundidad total

    β = np.radians(self.β) # Ángulo de inclinación del relleno por encima del
muro
    φ = np.radians(self.φ) # Transforma a radianes para poder utilizar las
funciones trigonométricas de numpy

    Kh = 2/3 * self.A0 # Coeficiente sísmico horizontal
    Kv = 2/3 * Kh # Coeficiente sísmico vertical

    θ = 0 # Ángulo de la cara interna del muro respecto a la vertical (Es 0 si
el muro no se encuentra inclinado)
    δ = np.radians(np.degrees(φ) / 2) # Ángulo de fricción entre el material de
relleno y la estructura de contención

    ψ = np.arctan(Kh / (1 - Kv))

    # Ecuación del coeficiente dinámico de presión activa resulta por partes
    numerador = (np.cos(φ - θ - ψ)) ** 2
    deno1 = np.cos(ψ) * ((np.cos(θ)) ** 2) * np.cos(δ + θ + ψ)
    deno2 = (1 + ((np.sin(φ + δ) * np.sin(φ - β - ψ)) / (np.cos(δ + θ + ψ) *
np.cos(β - θ))) ** (1/2)) ** 2

    kae = numerador / (deno1 * deno2)

    # Agrega a las arrays lo referente a la sobrecarga
    γe = np.insert(γe, 0, self.sobrecarga) # Agrega el valor de la sobrecarga en
la primera posición
    Hs = np.insert(self.Hs, 0, 1)
    Profs = np.insert(Profs, 0, 0)
    σa = np.cumsum(Hs * γe) # Vector con en el empuje acumulado en la
profundidad

    ###### REPITE LAS ARRAYS PARA HACER EL CAMBIO DE ESTRATO CON LOS kae DE LOS
DOS MATERIALES #######

    δ = np.repeat(δ, 2)

    σa = np.repeat(σa, 2)
    σa = np.delete(σa, 0)
    σa = np.delete(σa, len(σa) - 1)
    Profs = np.repeat(Profs, 2)
    Profs = np.delete(Profs, 0)
    Profs = np.delete(Profs, len(Profs) - 1)
    kae = np.repeat(kae, 2)

    σe = (kae * σa)

    ######  COHESIÓN (2 * c * √kae)  #######

    raiz_kae = kae ** (1/2)
    c = np.repeat(self.c, 2)
```

```
    tension = 2 * c * raiz_kae

    Presion_sismo = σe - tension # Empuje activo total

    Presion_sismo = (Presion_sismo) * (1 - Kv) # Componente horizontal del
empuje activo de Rankine

    Presion_sismo = np.insert(Presion_sismo, 0, 0) # Agrega el punto 0 en Y
    Profs = np.insert(Profs, 0, 0) # Agrega el punto 0 en X

    a = np.stack((Presion_sismo, Profs), axis=1) # Junta la matriz de
profundidac con la de presión efectivos
    point = np.array([[0, Prof_total]]) # Agrega el punto necesario para cerrar
el polígono
    b = np.vstack((a, point)) # Tuple con las coordenadas para el polígono
    self.empuje_Sismo = a

    pre_sismo = Polygon(b) # Polígono representa la presión lateral

    Pae = round(pre_sismo.area,2)
    resultante_sismo = round(0.6 * max(Profs), 2)

    ΔEa = round(Pae - self.Empuje_activo, 2)
    Resultante_ambas = (self.Empuje_activo * self.Resultante_EA + ΔEa *
resultante_sismo) / Pae

    if self.A0 == 0: # Quiere decir en los widgets que no se analiza por sismo
      ΔEa = 0
      resultante_sismo =0

    self.X_Sismo = Presion_sismo - self.X_PA
    self.Y_Sismo = Profs
    self.Poli_Sismo = pre_sismo
    self.Text_Sismo = "debido a sismo"

    return ΔEa, resultante_sismo



  def prelat_sobrecarga_areacargada1(self):
    """
    CALCULA EL EMPUJE DEBIDO A UNA SOBRECARGA CON LA FORMA DE UN ÁREA CARGADA CON
MAGNITUD CONSTANTE.
    """

    DDY = self.HTWALL / (self.NVERT - 1) # Incremento de la distancia vertical
    DOP = self.W / 2 # Distancia al centro en el eje y

    B1 = self.B / self.NSQW # Dimensiones del área eje x
    W1 = self.W / self.NSQL # Dimensiones del área eje y

    P = self.q * (B1 * W1) # Carga por área unitaria (kN)

    y = np.arange(B1 / 2, self.B, B1)
    x = np.arange(W1 / 2, self.W, W1)
    z = np.arange(0, self.HTWALL + DDY, DDY)
    z = np.where(z == 0, 0.000000001, z) # Si dejamos z = 0 y μ = 0.5, se hace
división entre 0 y da error
```

```
    # z = 5 # Prueba

    xx, yy = np.meshgrid(x, y) # Crea el grid con las coordenadas de las áreas
de carga unitarias

    r = ((abs(DOP - xx) ** 2) + (self.DTWALL + yy) ** 2) ** (1/2)

    σr_polygon = np.array([]) # Array vacía para almacenar la presión a cada
profundidad
    for l in range(len(z)): # Calcular para cada profundidad

      R = (r ** 2 + z[l] ** 2) ** (1/2)
      R = np.ravel(R) # Transforma la array (10,10) a (100, )
      r = np.ravel(r) # Transforma la array (10,10) a (100, )

      θ = np.arccos(z[l] / R)

      term1 =  3 * np.sin(θ) ** 2 * np.cos(θ) ** 3
      term2 = (1 - 2 * self.μ) * np.cos(θ) ** 2  / (1 + np.cos(θ))
      σr = (P / (2 * np.pi * z[l] ** 2)) * (term1 - term2)
      σr_total = np.sum(σr) # Suma de todas las cargas de las áras unitarias en
determinada profundidad z
      σr_polygon = np.append(σr_polygon, σr_total)

    σr_polygon = np.where(σr_polygon < 0, 0, σr_polygon) # Hacer 0 el primer
punto que sale negativo

    t = np.stack((σr_polygon, z), axis=1) # Junta las arrays de presiones y
profundidades
    t = np.vstack((t, [0, self.HTWALL])) # Agregar el punto necesario para
cerrar el polígono
    a = Polygon(t) # Crea el polígono

    # Para graficar
    self.X_SA = σr_polygon
    self.Y_SA = z
    self.Poli_SA = a
    self.Text_SA = "por sobrecarga tipo área cargada"

    self.empuje_sobre_area = t # Almacena la variable para poder usarla en otros
métodos.

    Empuje_SAC = round(a.area, 2)
    Resultante_SAC = round(self.HTWALL - a.centroid.y, 2)

    return Empuje_SAC, Resultante_SAC



  def prelat_sobrecarga_linearload(self): # Para invertir la carga, descomentar
la línea 173
    """
    CALCULA EL EMPUJE DEBIDO A UNA SOBRECARGA CON LA FORMA DE UN ÁREA CARGADA CON
VARIACIÓN LINEAL.
    La variación de la carga puede ser:
    - Ascendente en dirección del muro.
    - Descendete en dirección del muro.
    """
```

```
    DDY = self.HTWALL / (self.NVERT - 1) # Incremento de la distancia vertical
    DOP = self.W / 2 # Distancia al centro en el eje y

    B1 = self.B / self.NSQW # Dimensiones del área eje x
    W1 = self.W / self.NSQL # Dimensiones del área eje y

    P = (self.q * (self.B * self.W)) / 2 # Carga de todas las strips juntas

    y = np.arange(B1 / 2, self.B, B1)
    x = np.arange(W1 / 2, self.W, W1)
    z = np.arange(0, self.HTWALL + DDY, DDY)
    z = np.where(z == 0, 0.000000001, z) # Si dejamos z = 0 y μ = 0.5, se hace
división entre 0 y da error

    xx, yy = np.meshgrid(x, y) # Crea el grid con las coordenadas de los
centroides de las áreas de carga unitarias

    nq = yy[:, 0] * 10 # Calcula las nqs

    q_unit = P / (sum(nq) * B1 * W1 * 10) # Carga unitaria

    strip_intensities = np.array([]) # Array vacía para almacenar las strip
intensities
    strip_intensities = nq * B1 * W1 * q_unit

    if self.a_o_d == 4: # 4 es el value de descendente en el widget de dropdown
      strip_intensities = np.flip(strip_intensities) # Con esta línea se calcula
invertido

    strip_intensities = np.repeat(strip_intensities, 10) # Convierte la array
(10,0) a (100,0) para que cada carga unitaria tenga su intensidad

    r = ((abs(DOP - xx) ** 2) + (self.DTWALL + yy) ** 2) ** (1/2)

    σr_polygon = np.array([]) # Array vacía para almacenar la presión a cada
profundidad
    for l in range(len(z)): # Calcular para cada profundidad

      R = (r ** 2 + z[l] ** 2) ** (1/2)
      R = np.ravel(R) # Transforma la array (10,10) a (100, )
      r = np.ravel(r) # Transforma la array (10,10) a (100, )

      θ = np.arccos(z[l] / R)

      term1 =  3 * np.sin(θ) ** 2 * np.cos(θ) ** 3
      term2 = (1 - 2 * self.μ) * np.cos(θ) ** 2  / (1 + np.cos(θ))
      σr = (strip_intensities / (2 * np.pi * z[l] ** 2)) * (term1 - term2)
      σr_total = np.sum(σr) # Suma de todas las cargas de las áras unitarias en
determinada profundidad z
      σr_polygon = np.append(σr_polygon, σr_total)

    σr_polygon = np.where(σr_polygon < 0, 0, σr_polygon) # Hacer 0 el primer
punto que sale negativo

    t = np.stack((σr_polygon, z), axis=1) # Junta las arrays de presiones y
profundidades
```

```
    t = np.vstack((t, [0, self.HTWALL])) # Agregar el punto necesario para
cerrar el polígono
    a = Polygon(t) # Crea el polígono

    # Para graficar
    self.X_SL = σr_polygon
    self.Y_SL = z
    self.Poli_SL = a
    self.Text_SL = "por sobrecarga tipo variación lineal"

    self.empuje_sobre_linear = t # Almacena la variable para poder usarla en
otros métodos.

    Empuje_SLL = round(a.area, 2)
    Resultante_SLL = round(self.HTWALL - a.centroid.y, 2)

    return Empuje_SLL, Resultante_SLL



  def prelat_sobrecarga_cargapuntual(self):
    """
    CALCULA LA PRESIÓN LATERAL DEBIDO A UNA SOBRECARGA CON LA FORMA DE UNA CARGA
PUNTUAL.
    """

    DDY = self.HTWALL / (self.NVERT - 1) # Incremento de la distancia vertical

    z = np.arange(0, self.HTWALL + DDY, DDY)
    z = np.where(z == 0, 0.0000001, z) # Si dejamos z = 0 y μ = 0.5, se hace
división entre 0 y da error
    r = (0 ** 2 + self.DTWALL ** 2) ** (1 / 2)

    σr_polygon = np.array([]) # Array vacía para almacenar la presión a cada
profundidad

    for i in range(int(self.NVERT)): # Calcular para cada profundidad

      R = (r ** 2 + z[i] ** 2) ** (1/2)

      θ = np.arccos(z[i] / R)

      term1 =  3 * np.sin(θ) ** 2 * np.cos(θ) ** 3
      term2 = (1 - 2 * self.μ) * np.cos(θ) ** 2  / (1 + np.cos(θ))
      σr = (self.P / (2 * np.pi * z[i] ** 2)) * (term1 - term2)

      σr_polygon = np.append(σr_polygon, σr)

    σr_polygon = np.where(σr_polygon < 0, 0, σr_polygon) # Hacer 0 el primer
punto que sale negativo
    t = np.stack((σr_polygon, z), axis=1) # Junta las arrays de presiones y
profundidades
    t = np.vstack((t, [0, self.HTWALL])) # Agregar el punto necesario para
cerrar el polígono
    a = Polygon(t) # Crea el polígono

    # Para graficar
    self.X_SP = σr_polygon
```

```
    self.Y_SP = z
    self.Poli_SP = a
    self.Text_SP = "por sobrecarga tipo puntual"

    self.empuje_sobre_puntual = t # Almacena la variable para poder usarla en
otros métodos.

    Empuje_SP = round(a.area, 2)
    Resultante_SP = round(self.HTWALL - a.centroid.y, 2)

    return Empuje_SP, Resultante_SP



  def suma(self):
    """
    CALCULA LA SUMA DE TODOS LOS EMPUJES LATERALES POR MEDIO DE UNA INTEPORLACIÓN
ENTRE LOS DIAGRAMAS DE
    DISTRIBUCIÓN DE PRESIONES.
    """

    x_total = 0
    new_y = np.linspace(0, self.HTWALL, num=1000) # Mientras mas puntos sean,
mas preciso da
    self.cuales_sobrecargas = np.array([]) # Array que almacena cuales
sobrecargas fueron llamadas

    ##### REVISA CUALES SON LAS FUNCIONES DE PRESIÓN LATERAL QUE FUERON LLAMADAS
#####
    try:
      new_x1 = np.interp(new_y, self.empuje_Rankine[:,1],
self.empuje_Rankine[:,0]) # Interpola respecto a los nuevos puntos de y
      new_x1 = new_x1 * np.cos(np.radians(self.β)) # Suma solamente el
componente horizontal del empuje Pa porque solo ese afecta al momento de volteo
      x_total = x_total + new_x1 # Suma las coordenadas en x de las
distribuciones de presiones
    except AttributeError: self.empuje_Rankine = None # Se activa el
AttributeError cuando no se llamó la función

    try:
      new_x2 = np.interp(new_y, self.empuje_sobre_area[:,1],
self.empuje_sobre_area[:,0]) # Interpola respecto a los nuevos puntos de y
      x_total = x_total + new_x2 # Suma las coordenadas en x de las
distribuciones de presiones
      self.cuales_sobrecargas = np.append(self.cuales_sobrecargas, "area
cargada")
    except AttributeError: self.empuje_sobre_area = None

    try:
      new_x3 = np.interp(new_y, self.empuje_sobre_linear[:,1],
self.empuje_sobre_linear[:,0]) # Interpola respecto a los nuevos puntos de y
      x_total = x_total + new_x3 # Suma las coordenadas en x de las
distribuciones de presiones
      self.cuales_sobrecargas = np.append(self.cuales_sobrecargas, "variación
lineal")
    except AttributeError: self.empuje_sobre_linear = None

    try:
```

```
      new_x4 = np.interp(new_y, self.empuje_sobre_puntual[:,1],
self.empuje_sobre_puntual[:,0]) # Interpola respecto a los nuevos puntos de y
      x_total = x_total + new_x4 # Suma las coordenadas en x de las
distribuciones de presiones
      self.cuales_sobrecargas = np.append(self.cuales_sobrecargas, "carga
puntual")
    except AttributeError: self.empuje_sobre_puntual = None

    if self.A0 > 0: # Si es = 0 quiere decir que el usuario no toma en cuenta el
incremento debido a sismo
      new_x5 = np.interp(new_y, self.empuje_Sismo[:,1], self.empuje_Sismo[:,0]-
self.X_PA) # La resta es para tomar en cuenta solamente el incremento
      x_total = x_total + new_x5


    x_total[0] = 0 # Agrega los puntos necesarios para cerrar el polígono
    x_total[len(x_total) - 1] = 0 # Agrega los puntos necesarios para cerrar el
polígono

    poly = np.stack((x_total, new_y), axis=1) # Junta las presiones con la prof
para hacer el polígono
    a = Polygon(poly) # Crea el polígono

    self.X_Suma = x_total
    self.Y_Suma = new_y
    self.Poli_Suma = a
    self.Text_Suma = "debido a la presión total"

    # Guarda los resultados en self para poderlos utilizar en la clase de Predim
de muros.
    Empuje_total = round(a.area, 2)
    brazo_palanca = round(self.HTWALL - a.centroid.y, 2)

    return Empuje_total, brazo_palanca #, self.cuales_sobrecargas



  def graficar(self):
    """
    GRAFICA CADA UNO DE LOS DIAGRAMAS DE DISTRIBUCIÓN DE PRESIONES LATERALES.
    """

    fig = plt.figure(figsize=(12,10))
    gs = plt.GridSpec(nrows=3, ncols=2)

    # Para la presión activa
    ax0 = fig.add_subplot(gs[0, 0])
    ax0.plot(self.X_PA, self.Y_PA, color="blue", linewidth=3)
    plt.ylim(np.max(self.Y_PA), 0)
    ax0.plot(0, self.Poli_PA.centroid.y, marker="<", ms=10, c="blue")
    # ax0.arrow(np.max(self.X_PA), self.Poli_PA.centroid.y, -np.max(self.X_PA)/5,
0,
    #           head_width=np.max(self.Y_PA)/15,
head_length=np.max(self.X_PA)/100, shape="full", color="red")
    plt.ylabel("Profundidad (m)")
    plt.xlabel("Presión (kN/m2)")
    ax0.set_title(f"Distribución de presiones {self.Text_PA}", fontsize=12)
    plt.grid()
```

```
    # Para la presión pasiva
    ax1 = fig.add_subplot(gs[0, 1])
    ax1.plot(self.X_PP, self.Y_PP, color="orange", linewidth=3)
    plt.ylim(np.max(self.Y_PP), 0)
    # ax1.arrow(np.max(self.X_PP), self.Poli_PP.centroid.y, -np.max(self.X_PP)/5,
0,
    #          head_width=np.max(self.Y_PA)/15,
head_length=np.max(self.X_PA)/100, shape="full", color="red")
    ax1.plot(0, self.Poli_PP.centroid.y, marker="<", ms=10, c="orange")
    plt.ylabel("Profundidad (m)")
    plt.xlabel("Presión (kN/m2)")
    ax1.set_title(f"Distribución de presiones {self.Text_PP}", fontsize=12)
    plt.grid()

    # Para el incremento de presiones debido a sismo
    if self.A0 > 0:
      ax2 = fig.add_subplot(gs[1, 0])
      ax2.plot(self.X_Sismo, self.Y_Sismo, color="green", linewidth=3)
      plt.ylim(np.max(self.Y_Sismo), 0)
      # ax.arrow(np.max(self.X_Sismo), self.Poli_Sismo.centroid.y, -
np.max(self.X_Sismo)/5, 0, head_width=0.2, shape="full", color="red")
      ax2.plot(0, self.Poli_Sismo.centroid.y, marker="<", ms=10, c="green")
      plt.ylabel("Profundidad (m)")
      plt.xlabel("Presión (kN/m2)")
      ax2.set_title(f"Distribución de presiones {self.Text_Sismo}", fontsize=12)
      plt.grid()

    # Para la sobrecarga tipo área con carga constante
    if "area cargada" in self.cuales_sobrecargas:
      ax3 = fig.add_subplot(gs[1, 1])
      ax3.plot(self.X_SA, self.Y_SA, color="red", linewidth=3)
      plt.ylim(np.max(self.Y_SA), 0)
      # ax.arrow(np.max(self.X_SA), self.Poli_SA.centroid.y, -
np.max(self.X_SA)/5, 0, head_width=0.2, shape="full", color="red")
      ax3.plot(0, self.Poli_SA.centroid.y, marker="<", ms=10, c="red")
      plt.ylabel("Profundidad (m)")
      plt.xlabel("Presión (kN/m2)")
      ax3.set_title(f"Distribución de presiones {self.Text_SA}", fontsize=12)
      plt.grid()

    # Para la sobrecarga tipo área con carga variación lineal
    if "variación lineal" in self.cuales_sobrecargas:
      ax3 = fig.add_subplot(gs[1, 1])
      ax3.plot(self.X_SL, self.Y_SL, color="red", linewidth=3)
      plt.ylim(np.max(self.Y_SL), 0)
      # ax.arrow(np.max(self.X_SL), self.Poli_SL.centroid.y, -
np.max(self.X_SL)/5, 0, head_width=0.15, shape="full", color="red")
      ax3.plot(0, self.Poli_SL.centroid.y, marker="<", ms=10, c="red")
      plt.ylabel("Profundidad (m)")
      plt.xlabel("Presión (kN/m2)")
      ax3.set_title(f"Distribución de presiones {self.Text_SL}", fontsize=12)
      plt.grid()

    # Para sobrecarga tipo carga puntual
    if "carga puntual" in self.cuales_sobrecargas:
      ax3 = fig.add_subplot(gs[1, 1])
      ax3.plot(self.X_SP, self.Y_SP, color="red", linewidth=3)
```

```
      plt.ylim(np.max(self.Y_SP), 0)
      # ax.arrow(np.max(self.X_SP), self.Poli_SP.centroid.y, -
np.max(self.X_SP)/5, 0, head_width=0.2, shape="full", color="red")
      ax3.plot(0, self.Poli_SP.centroid.y, marker="<", ms=10, c="red")
      plt.ylabel("Profundidad (m)")
      plt.xlabel("Presión (kN/m2)")
      ax3.set_title(f"Distribución de presiones {self.Text_SP}", fontsize=12)
      plt.grid()

    # Para la sobrecarga total de la función suma
    ax4 = fig.add_subplot(gs[2, 0])
    ax4.plot(self.X_Suma, self.Y_Suma, color="purple", linewidth=3)
    plt.ylim(np.max(self.Y_Suma), 0)
    # ax.arrow(np.max(self.X_Suma), self.Poli_Suma.centroid.y, -
np.max(self.X_Suma)/5, 0, head_width=0.25, shape="full", color="red")
    ax4.plot(0, self.Poli_Suma.centroid.y, marker="<", ms=10, c="purple")
    plt.ylabel("Profundidad (m)")
    plt.xlabel("Presión (kN/m2)")
    ax4.set_title(f"Distribución de presiones {self.Text_Suma}", fontsize=12)
    plt.grid()

    plt.tight_layout()

 plt.savefig("/content/drive/MyDrive/Tesis_python/Reporte_descargar/Diagramas_pr
esiones", dpi=250)
    plt.show()



class Muros():

  def __init__(self, altura_muro, Hs, empuje_total, brazo_palanca,
peso_especif_material_muro, coef_deslizam, cuales_sobrecargas,
               presion_pasiva, centroide, prof_emplazamiento, γms, c, β, α,
in_suelo, φ, NAF,
               fc, fy, E, Φ, rec_libre, var_propuesta_num,\
               γ_apoyo, c_apoyo, φ_apoyo):

    self.altura_muro = altura_muro
    self.Hs_estratos = Hs
    self.empuje_total = empuje_total
    self.brazo_palanca = brazo_palanca
    self.peso_especif_material_muro = peso_especif_material_muro
    self.coef_deslizam = coef_deslizam
    self.cuales_sobrecargas = cuales_sobrecargas
    self.empuje_pasivo = presion_pasiva
    self.centroide_pasiva = centroide
    self.prof_emplazamiento = prof_emplazamiento
    self.γ_suelo = γms
    self.c = c
    self.β = β
    self.α = α
    self.φ = φ
    self.in_suelo = in_suelo
    self.NAF = NAF
    self.colores =
["tan","peru","sienna","brown","maroon","rosybrown","sandybrown","goldenrod","co
rnsilk","burlywood"]
```

```
    self.fc = fc
    self.fy = fy
    self.E = E
    self.Φ = Φ
    self.rec_libre = rec_libre
    self.var_propuesta_num = var_propuesta_num
    self.γ_apoyo = γ_apoyo
    self.c_apoyo = c_apoyo
    self.φ_apoyo = φ_apoyo



  def predim_gravedad(self):
    """
    REALIZA EL DIMENDIONAMIENTO DE UN MURO DE GRAVEDAD PARTIENDO DE LAS
DIMENSIONES MÍNIMAS RECOMENDADAS
    E ITERANDO HASTA QUE LAS DIMENSIONES DEL MURO GARANTICEN SEGURIDAD CONTRA EL
VOLCAMIENTO, EL DESLIZAMIENTO Y
    CALCULA LA CAPACIDAD DE CARGA ÚLTIMA DEL TERRENO POR EL MÉTODO DE VESIC.
    """

    global base_muro, corona, prof_emplazamiento
    self.tipo_muro = "gravedad"

    # alturas_acum = np.cumsum(self.Hs_estratos)
    # alturas_acum = np.insert(alturas_acum, 0, 0)

    ### Dimensiones tentativas ###
    base_muro = self.altura_muro / 2 # Min H/2
    corona = 0.30 # min 30 cm

    # Calcula los factores de seguridad  de volteo y deslizamiento hasta que
pasen
    while True:

      # Coordenadas y diagrama del muro
      x = np.array([0, 0, base_muro-corona, base_muro, base_muro])
      y = np.array([0, self.prof_emplazamiento, self.altura_muro,
self.altura_muro, 0])
      coords_muro = np.stack((x, y), axis=1)
      poly = Polygon(coords_muro)

      global MR, MV, q_max
      w_muro = poly.area * self.peso_especif_material_muro # Peso del muro
      MR = w_muro * poly.centroid.x # Momento resistente (kN.m)
      MV = self.empuje_total * self.brazo_palanca # Momento del volteo (kN.m)

      FSV = MR / MV # factor de seguridad por volteo (Debe ser > 1.5 suelo
granular o 2 suelo cohesivo)
      FSD = ((w_muro * self.coef_deslizam) + self.empuje_pasivo) /
self.empuje_total # factor de seguridad por deslizamiento (Debe ser > 1.5)

      ### CAPACIDAD DE CARGA ###
      Mneto = MR - MV # Momento neto sobre el punto inicial de la punta del muro

      CE = Mneto / w_muro

      e = (base_muro / 2) - CE # Excentricidad de la resultante R (m)
```

```
      # Cuando la excentricidad > B/6, se producen tensiones en el suelo, hay
que aumentar la punta del muro
      if e > (base_muro / 6):
        base_muro = base_muro + 0.1
        corona = corona + 0.05
        continue

      q_max = (w_muro / base_muro) * (1 + ((6 * e) / base_muro))
      q_min = (w_muro / base_muro) * (1 - ((6 * e) / base_muro))

      # # Tomo el estrato de hasta abajo para las propiedades del suelo
      # c = self.c[len(self.c) - 1]
      # φ = np.radians(self.φ[len(self.φ) - 1])
      # γ = self.γ_suelo[len(self.γ_suelo) - 1]
      c = self.c_apoyo
      φ = np.radians(self.φ_apoyo)
      γ = self.γ_apoyo
      γw = 9.81 # Peso específico del agua (kN/m3)

      # Influencia del NAF en la capacidad de carga
      if self.NAF <= self.prof_emplazamiento:
        γe = γ - γw
      elif (self.prof_emplazamiento < self.NAF) and (self.NAF <
self.prof_emplazamiento + base_muro):
        γe = γ - γw * (1 - ((self.NAF - self.prof_emplazamiento) / base_muro))
      elif self.prof_emplazamiento + base_muro <= self.NAF:
        γe = γ

      σ_zD = γ * self.prof_emplazamiento

      # Shape factors (En zapatas continuas como es el caso, se les da un valor
de 1)
      Sc = 1
      Sq = 1
      Sγ = 1

      # Depth factors
      k = self.prof_emplazamiento / base_muro
      dc = 1 + 0.4 * k
      dq = 1 + 2 * k * np.tan(φ) * (1 - np.sin(φ)) ** 2
      dγ = 1

      # Bearing capacity factors
      Nq = ((np.exp(1) ** (np.pi * np.tan(φ))) * (np.tan(np.radians(45 +
(np.degrees(φ)/2)))) ** 2)
      if φ == 0:
        Nc = 5.14
      elif φ > 0:
        Nc = (Nq - 1) / np.tan(φ)
      Nγ = 2 * (Nq + 1) * np.tan(φ)

      # Base inclination factors
      if self.α == 0:
        bc = 1
        bq = 1
        bγ = 1
      elif self.α > 0:
```

```
        bc = 1 - self.α / 147
        bq = (1 - (self.α * np.tan(φ)) / 57) ** 2
        bγ = bq

      # Ground inclination factors (Cuando se encuentra cerca de la orilla de un talud (no es el caso))
      if self.in_suelo == 0:
        gc = 1
        gq = 1
        gγ = 1
      elif self.in_suelo > 0:
        gc = 1 - self.in_suelo / 147
        gq = (1 - np.tan(np.radians(self.in_suelo))) ** 2
        gγ = gq

      # Load inclination factors
      ic = 1
      iq = 1
      iγ = 1

      qn = c * Nc * Sc * dc * ic * bc * gc + σ_zD * Nq * Sq * dq * iq * bq * gq + 0.5 * γe * base_muro * Nγ * Sγ * dγ * iγ * bγ * gγ

      # SI YA CUMPLE ESTAS CONDICIONES SALE DEL LOOP
      if c == 0: # FSV 1.5 para materiales granulares, 2.0 para materiales cohesivos
        if (FSV >= 1.5 and FSD >= 1.5): # Si ya cumplen los factores, salir del ciclo infinito
          break
        else: # Si no cumplen los factores, aumentar 10cm a todas las dimensiones
          base_muro = base_muro + 0.1
          corona = corona + 0.1
      else:
        if (FSV >= 2 and FSD >= 1.5): # Si ya cumplen los factores, salir del ciclo infinito
          break
        else: # Si no cumplen los factores, aumentar 10cm a todas las dimensiones
          base_muro = base_muro + 0.1
          corona = corona + 0.1

    FSCC = round(qn / q_max, 2)

    self.coords_muro = coords_muro
    self.base_muro = base_muro
    self.corona = corona
    self.q_max = q_max
    self.q_min = q_min

    return round(FSV, 2), round(FSD, 2), FSCC, round(qn, 2)



  def predim_voladizo(self):
    """
    REALIZA EL DIMENDIONAMIENTO DE UN MURO EN VOLADIZO PARTIENDO DE LAS DIMENSIONES MÍNIMAS RECOMENDADAS
```

```
    E ITERANDO HASTA QUE LAS DIMENSIONES DEL MURO GARANTICEN SEGURIDAD CONTRA EL
VOLCAMIENTO, EL DESLIZAMIENTO Y
    CALCULA LA CAPACIDAD DE CARGA ÚLTIMA DEL TERRENO POR EL MÉTODO DE VESIC.
    """

    global base_muro, corona, base_punta, altura_talon
    self.tipo_muro = "voladizo"

    alturas_acum = np.cumsum(self.Hs_estratos)
    alturas_acum = np.insert(alturas_acum, 0, 0)

    base_muro = 0.5 * self.altura_muro # Entre 0.5 a 0.7 H
    corona = 0.30 # Min 0.3 m
    corona_aumento = 0 # Dimension que crece la base de la pantalla con cada
iteracion
    base_punta = np.ceil(base_muro / 3) # B/3
    altura_talon = np.round(self.altura_muro / 10, 1) # H/12 to H/10


    ### Dibujo y características del muro ###
    # Calcula los factores de seguridad hasta que pasen (volteo y deslizamiento)
    while True:

      base_talon = base_muro - base_punta - corona_aumento - corona

      # Array con las dimensinoes en x del muro
      x = np.array([0, 0, base_punta, base_punta+corona_aumento,
base_punta+corona+corona_aumento,
                    base_punta+corona+corona_aumento, base_muro, base_muro])

      y = np.array([0, altura_talon, self.altura_muro, altura_talon, 0]) # Array
con las dimensiones en y del muro
      y = np.repeat(y, 2)
      y = np.delete(y, len(y) - 1)
      y = np.delete(y, 0)
      coords_poly = np.stack((x, y), axis=1)
      poly_muro = Polygon(coords_poly)

      # Dimensiones del relleno que aportan al momento resistente para calcuar
el centroide
      relleno_x = np.array([base_muro-base_talon, base_muro])
      relleno_x = np.repeat(relleno_x, 2)
      relleno_y = np.array([altura_talon, self.altura_muro])
      relleno_y = np.append(relleno_y, np.flipud(relleno_y)) # Inivierte las
coordenadas y las agrega
      relleno = np.stack((relleno_x, relleno_y), axis=1)
      poly_relleno = Polygon(relleno)

      # Para obtener el peso de los estratos que aportan resistencia
      w_relleno = 0 # Contador para el peso del relleno

      # Cálcula el peso de cada estrato que se encuentra sobre el talón del muro
      for j in range (alturas_acum.shape[0] - 1):

        coords_estratos_x = np.array([base_muro-base_talon, base_muro-
base_talon, base_muro, base_muro])
        coords_estratos_y = np.array([alturas_acum[j], alturas_acum[j+1],
alturas_acum[j+1], alturas_acum[j]])
```

```
        union_estratos = np.stack((coords_estratos_x, coords_estratos_y),
axis=1)

        poligono_estratos = Polygon(union_estratos) # Crea el polígono referente
al estrato j
        area = poligono_estratos.area
        w_relleno = w_relleno + (area * self.γ_suelo[j]) # Cálculo del peso del
relleno

      global MR, MV, q_max
      w_relleno = w_relleno - ((base_talon) * altura_talon *
np.mean(self.γ_suelo)) # Peso del relleno sobre el talón del muro (kN)
      w_muro = poly_muro.area * self.peso_especif_material_muro # Cálculo del
peso del muro (kN)
      MR = w_muro * poly_muro.centroid.x  + w_relleno * poly_relleno.centroid.x
# Cálculo del MR total
      MV = self.empuje_total * self.brazo_palanca # Momento de volteo (KN-m)

      FSV = MR / MV # factor de seguridad por volteo (Debe ser > 1.5)
      FSD = (((w_muro + w_relleno) * self.coef_deslizam) + self.empuje_pasivo) /
self.empuje_total # factor de seguridad por deslizamiento (Debe ser > 1.5)

      ## CAPACIDAD DE CARGA ##

      Mneto = MR - MV # Momento neto sobre el punto inicial de la punta del muro
(izquierda)

      CE = Mneto / (w_muro + w_relleno)

      e = (base_muro / 2) - CE # Excentricidad de la resultante R

      # Cuando la excentricidad > B/6, se producen tensiones en el suelo, hay
que aumentar la punta del muro
      if e > (base_muro / 6):
        base_muro = base_muro + 0.1
        # base_punta = base_punta + 0.05 # Apagué esta línea parace que funciona
sin ella 05-07-21
        continue

      q_max = ((w_muro + w_relleno) / base_muro) * (1 + ((6 * e) / base_muro))
      q_min = ((w_muro + w_relleno) / base_muro) * (1 - ((6 * e) / base_muro))

      # Tomo el estrato de hasta abajo para las propiedades del suelo
      # c = self.c[len(self.c) - 1]
      # φ = np.radians(self.φ[len(self.φ) - 1])
      # γ = self.γ_suelo[len(self.γ_suelo) - 1]
      c = self.c_apoyo
      φ = np.radians(self.φ_apoyo)
      γ = self.γ_apoyo
      γw = 9.81 # Peso específico del agua kN/m3

      # Influencia del NAF en la capacidad de carga
      if self.NAF <= self.prof_emplazamiento:
        γe = γ - γw
      elif (self.prof_emplazamiento < self.NAF) and (self.NAF <
self.prof_emplazamiento + base_muro):
        γe = γ - γw * (1 - ((self.NAF - self.prof_emplazamiento) / base_muro))
```

```
        elif self.prof_emplazamiento + base_muro <= self.NAF:
            γe = γ

        σ_zD = γ * self.prof_emplazamiento

        # Shape factors (En zapatas continuas como es el caso, se les da un valor
de 1)
        Sc = 1
        Sq = 1
        Sγ = 1

        # Depth factors
        k = self.prof_emplazamiento / base_muro
        dc = 1 + 0.4 * k
        dq = 1 + 2 * k * np.tan(φ) * (1 - np.sin(φ)) ** 2
        dγ = 1

        # Bearing capacity factors
        Nq = ((np.exp(1) ** (np.pi * np.tan(φ))) * (np.tan(np.radians(45 +
(np.degrees(φ)/2)))) ** 2)
        if φ == 0:
            Nc = 5.14
        elif φ > 0:
            Nc = (Nq - 1) / np.tan(φ)
        Nγ = 2 * (Nq + 1) * np.tan(φ)

        # Base inclination factors
        if self.α == 0:
            bc = 1
            bq = 1
            bγ = 1
        elif self.α > 0:
            bc = 1 - self.α / 147
            bq = (1 - (self.α * np.tan(φ)) / 57) ** 2
            bγ = bq

        # Ground inclination factors (Cuando se encuentra cerca de la orilla de un
talud (no es el caso))
        if self.in_suelo == 0:
            gc = 1
            gq = 1
            gγ = 1
        elif self.in_suelo > 0:
            gc = 1 - self.in_suelo / 147
            gq = (1 - np.tan(np.radians(self.in_suelo))) ** 2
            gγ = gq

        # Load inclination factors
        ic = 1
        iq = 1
        iγ = 1

        qn = c * Nc * Sc * dc * ic * bc * gc + σ_zD * Nq * Sq * dq * iq * bq * gq
+ 0.5 * γe * base_muro * Nγ * Sγ * dγ * iγ * bγ * gγ


        # SI YA CUMPLE ESTAS CONDICIONES, SALE DEL WHILE LOOP
```

```
      if c == 0: # FSV 1.5 para materiales granulares, 2.0 para materiales
cohesivos
        if (FSV >= 1.5 and FSD >= 1.5): # Si ya cumplen los factores, salir del
ciclo infinito
          break
        else: # Si no cumplen los factores, aumentar 10cm a todas las
dimensiones
          base_muro = base_muro + 0.05
          # base_punta = base_punta + 0.05
          corona_aumento = corona_aumento + 0.05
          altura_talon = altura_talon + 0.05
          if altura_talon > self.prof_emplazamiento:
            altura_talon = self.prof_emplazamiento
      else:
        if (FSV >= 2 and FSD >= 1.5): # Si ya cumplen los factores, salir del
ciclo infinito
          break
        else: # Si no cumplen los factores, aumentar 10cm a todas las
dimensiones
          base_muro = base_muro + 0.05
          # base_punta = base_punta + 0.05
          corona_aumento = corona_aumento + 0.05
          altura_talon = altura_talon + 0.05
          if altura_talon > self.prof_emplazamiento:
            altura_talon = self.prof_emplazamiento

    FSCC = round(qn / q_max, 2) # Factor de seguridad por capacidad de carga

    self.altura_talon = altura_talon
    self.base_punta = base_punta
    self.base_muro = base_muro
    self.corona = corona
    self.coords_poly = coords_poly
    self.q_max = q_max
    self.q_min = q_min
    self.w_relleno = w_relleno
    self.base_talon = base_muro - base_punta - corona
    self.corona_aumento = corona_aumento

    return round(FSV, 2), round(FSD, 2), FSCC, round(qn, 2)



  def graficar_muro(self):
    """
    REALIZA EL DIAGRAMA DEL MURO DE CONTENCIÓN CALCULADO
    """

    if self.tipo_muro == "gravedad":

      fig, ax = plt.subplots(figsize=(6, 6))

      # Grafica el muro de conteción y las cotas del mismo
      Poligono(self.coords_muro, color='darkgray', ColorBorde='darkgray',
alpha=1)
      # Poligono(coords_relleno,color='#CD853F',ColorBorde='#CD853F',alpha=0.7)
      DibujaCota((0,-0.2), (self.base_muro,-0.2), dist=-0.2, lw=2,
AnchoFlecha=0.1, color="black") # base
```

```
      DibujaCota((self.base_muro+0.2,0), (self.base_muro+0.2, self.altura_muro),
dist=-0.2, lw=2, AnchoFlecha=0.1, color="black") # altura
      DibujaCota((self.base_muro-self.corona,self.altura_muro+0.2),
(self.base_muro,self.altura_muro+0.2),
                 dist=0.3, lw=2, AnchoFlecha=0.06, color="black") # corona
      DibujaCota((0.1,0), (0.1, self.prof_emplazamiento), dist=-0.15, lw=2,
AnchoFlecha=0.07, color="black") # emplazamiento
      ax.set_title("Diagrama del muro de contención")
      ax.set_ylim(-0.7,self.altura_muro+1)
      plt.ylabel("Metros")
      plt.xlabel("Metros")

      # Dibuja el suelo del lado izquierdo (Profundidad de emplazamiento)
      prof_emplazam_x = np.array([-2, -2, 0, 0])
      prof_emplazam_y = np.array([0, self.prof_emplazamiento,
self.prof_emplazamiento, 0])
      ax.fill(prof_emplazam_x, prof_emplazam_y, facecolor="tan")

      # En caso de que el terreno tenga pendiente por encima del muro, lo
grafica
      angulo_sobre_muro_x = np.array([self.base_muro, self.base_muro+1,
self.base_muro+1])
      if self.β > 0:
        angulo_sobre_muro_y = np.array([self.altura_muro, self.altura_muro+0.7,
self.altura_muro])
      else:
        angulo_sobre_muro_y = np.array([self.altura_muro, self.altura_muro,
self.altura_muro])
      ax.fill(angulo_sobre_muro_x, angulo_sobre_muro_y,
facecolor=self.colores[len(self.Hs_estratos)-1])

      # Loop para dibujar los diferentes estratos del suelo contenido
      alturas_acum = np.cumsum(self.Hs_estratos)
      alturas_acum = np.insert(alturas_acum, 0, 0)
      for i in range (alturas_acum.shape[0] - 1):
        estratos_x = np.array([self.base_muro, self.base_muro, self.base_muro+1,
self.base_muro+1])
        estratos_y = np.array([alturas_acum[i], alturas_acum[i+1],
alturas_acum[i+1], alturas_acum[i]])
        ax.fill(estratos_x, estratos_y, facecolor=self.colores[i], ec=None)

      # Revisa cuales sobrecargas fueron llamadas para agregarlas al diagrama
      ax.text(-2, self.altura_muro+0.3, "Sobrecargas:", color="black")
      if self.cuales_sobrecargas == 1:
        sc = "Sin sobrecarga"
      elif self.cuales_sobrecargas == 2:
        sc = "Área de carga constante"
      elif self.cuales_sobrecargas == 3:
        sc = "Variación lineal ascendente"
      elif self.cuales_sobrecargas == 4:
        sc = "Variación lineal descendente"
      elif self.cuales_sobrecargas == 5:
        sc = "Sobrecarga puntual"
      ax.text(-2, (self.altura_muro), "- " + sc, color="black")
      ax.set_aspect('equal', 'box')

 plt.savefig("/content/drive/MyDrive/Tesis_python/Reporte_descargar/Diagrama_mur
o", dpi=250)
```

```
      plt.show()


    elif self.tipo_muro == "voladizo":

      fig, ax = plt.subplots(figsize=(6, 6))

      # Para graficar la profundidad de emplazamiento
      if self.prof_emplazamiento <= self.altura_talon: # Cuando la prof de
emplazamiento es menor o igual a la altura del talón
        prof_emplazam_x = np.array([-1.5, -1.5, 0, 0])
        prof_emplazam_y = np.array([0, self.prof_emplazamiento,
self.prof_emplazamiento, 0])
        ax.fill(prof_emplazam_x, prof_emplazam_y, facecolor="tan")
      else: # Cuando la prof de emplazamiento es mayor a la altura del talón
        prof_emplazam_x = np.array([-1.5, -1.5, self.base_punta,
self.base_punta, 0, 0])
        prof_emplazam_y = np.array([0, self.prof_emplazamiento,
self.prof_emplazamiento, self.altura_talon, self.altura_talon, 0])
        ax.fill(prof_emplazam_x, prof_emplazam_y, facecolor="tan")

      # En caso de que el terreno tenga pendiente por encima del muro, lo
grafica
      angulo_sobre_muro_x = np.array([self.base_punta+self.corona,
self.base_muro+0.5, self.base_muro+0.5])
      if self.β > 0:
        angulo_sobre_muro_y = np.array([self.altura_muro, self.altura_muro+0.7,
self.altura_muro])
      else:
        angulo_sobre_muro_y = np.array([self.altura_muro, self.altura_muro,
self.altura_muro])
      ax.fill(angulo_sobre_muro_x, angulo_sobre_muro_y,
facecolor=self.colores[len(self.Hs_estratos)-1])

      # Para graficar los estratos de suelo contenidos
      alturas_acum = np.cumsum(self.Hs_estratos)
      alturas_acum = np.insert(alturas_acum, 0, 0)
      for i in range (alturas_acum.shape[0] - 1):
        estratos_x = np.array([self.base_punta+self.corona+self.corona_aumento,
self.base_punta+self.corona+self.corona_aumento,
                               self.base_muro+0.5, self.base_muro+0.5])
        estratos_y = np.array([alturas_acum[i], alturas_acum[i+1],
alturas_acum[i+1], alturas_acum[i]])
        ax.fill(estratos_x, estratos_y, facecolor=self.colores[i], ec=None)


      # Dibujo del muro de contención y de las cotas
      Poligono(self.coords_poly, color='darkgray', ColorBorde='darkgray',
alpha=1)
      # ax.arrow(base_muro+0.7, brazo_palanca, -0.5, 0, shape='full',
head_width=0.1, color="red")
      DibujaCota((0,-0.2), (self.base_muro,-0.2), dist=-0.2, lw=2,
AnchoFlecha=0.1, color="black") # base
      DibujaCota((self.base_muro+0.2,0), (self.base_muro+0.2, self.altura_muro),
dist=-0.2, lw=2, AnchoFlecha=0.1, color="black") # altura
      DibujaCota((self.base_punta+self.corona_aumento,self.altura_muro+0.2),
(self.base_punta+self.corona+self.corona_aumento,self.altura_muro+0.2),
                 dist=0.3, lw=2, AnchoFlecha=0.05, color="black") # corona
```

```
      DibujaCota((self.base_punta+self.corona+self.corona_aumento,
self.altura_talon+0.2), (self.base_muro, self.altura_talon+0.2),
                 dist=0.2, lw=2, AnchoFlecha=0.05, color="black") # base talón

      DibujaCota((-0.2,0), (-0.2,self.altura_talon), dist=0.2, lw=2,
AnchoFlecha=0.05, color="black") # altura talón
      DibujaCota((0,self.altura_talon+0.2),
(self.base_punta,self.altura_talon+0.2), dist=0.2, lw=2, AnchoFlecha=0.05,
color="black") # base punta
      plt.title("Diagrama del muro de contención")
      ax.set_ylim(-0.7,self.altura_muro+1)
      plt.ylabel("Metros")
      plt.xlabel("Metros")

      # Revisa cuales sobrecargas fueron llamadas para agregarlas al diagrama
      ax.text(-1.5, self.altura_muro+0.3, "Sobrecargas:", color="black")
      if self.cuales_sobrecargas == 1:
        sc = "Sin sobrecarga"
      elif self.cuales_sobrecargas == 2:
        sc = "Área de carga constante"
      elif self.cuales_sobrecargas == 3:
        sc = "Variación lineal ascendente"
      elif self.cuales_sobrecargas == 4:
        sc = "Variación lineal descendente"
      elif self.cuales_sobrecargas == 5:
        sc = "Sobrecarga puntual"
      ax.text(-1.5, (self.altura_muro), "- " + sc, color="black")
      ax.set_aspect('equal', 'box')

 plt.savefig("/content/drive/MyDrive/Tesis_python/Reporte_descargar/Diagrama_mur
o", dpi=250)
      plt.show()



  def graficar_CC(self):
    """
    REALIZA EL DIAGRAMA DE LA DISTRIBUCIÓN DE PRESIONES QUE EL MURO EJERCE SOBRE
EL SUELO
    """

    # GRÁFICAS CORRESPONDIENTES A LA CAPACIDAD DE CARGA
    fig, ax = plt.subplots(figsize=(6, 6))

    # Distribución de la presión bajo la losa de la base
    presiones_x = np.array([0, 0, self.base_muro, self.base_muro])
    presiones_y = np.array([0, -self.q_max, -self.q_min, 0])

    # Dibuja la base del muro
    base_x = np.array([0, 0, self.base_muro, self.base_muro])
    base_y = np.array([0, 2, 2, 0])
    base = np.stack((base_x, base_y), axis=1)

    Poligono(base, color="darkgrey", ColorBorde="darkgrey")
    ax.plot(presiones_x, presiones_y, "--", color="red")
    ax.set_title("Distribución de presiones bajo la base del muro")
    ax.set_xlabel("Metros")
    ax.set_ylabel("kN/m2")
```

```
    ax.text(0.2, -self.q_max-0.5, round(self.q_max,2))
    ax.text(self.base_muro, -self.q_min-5, round(self.q_min,2))
    ax.set_xlim(-0.3, self.base_muro+0.3)
    plt.show()



  def diseno_acero(self):
    """
    CALCULA EL ACERO DE REFUERZO NECESARIO PARA RESISTIR EL CORTANTE Y MOMENTO ÚLTIMO PARA MUROS EN VOLADIZO
    """

    # Para la punta (Sección crítica 1 - Presión que el suelo transmite al muro)
    qmax = self.q_max / 9.81 # Estoy conviertiendo kPa en t/m2
    qmin = self.q_min / 9.81 # Estoy conviertiendo kPa en t/m2

    interp_diag_punta = qmax - (self.base_punta * (qmax - qmin) / self.base_muro) # interpolacion del diag de  presiones a una dist sec crit de la punta

    F_pres_max_pun = 1.5 * qmax # Multiplicando por factor de carga (Cv)
    F_pres_min_pun = 1.5 * interp_diag_punta # Multiplicando por factor de carga (Cv)

    # Para el talón (Sección crítica 2 - Peso del relleno y del talón del muro)
    F_pres_talon = 1.3 * ((self.w_relleno / 9.81) + (2.4 * self.base_talon * self.altura_talon)) # Estoy conviertiendo kN a ton

    # Para la pantalla (Sección crítica 3 - Empujes laterales)
    F_pres_pan = 1.5 * (self.empuje_total / 9.81) # Aquí va el valor del empuje total (Cv)

    db = round((self.var_propuesta_num / 8) * 2.54, 2) # Diámetro de varilla en (cm)
    a_var = (np.pi * ((self.var_propuesta_num / 8) * 2.54) ** 2) / 4 # Área nominal de la varilla en (cm2)
    rec_efect = round(self.rec_libre + db / 2, 2) # Recubrimiento del bordo libre al centro de varilla (cm)


    ######    PARA LA PUNTA    ######

    while True:

      d_punta = (self.altura_talon * 100) - rec_efect # Peralte efectivo (cm)

      # Para obtener el momento último de diseño
      coords_diam_pun_x = np.array([0, 0, self.base_punta, self.base_punta])
      coords_diam_pun_y = np.array([0, F_pres_max_pun, F_pres_min_pun, 0])
      coords_diam_pun = np.stack((coords_diam_pun_x, coords_diam_pun_y), axis=1)
      diagrama_punta = Polygon(coords_diam_pun)

      Mu_punta = round(diagrama_punta.area * (self.base_punta - diagrama_punta.centroid.x), 2) # Momento último de diseño (t.m)

      # Porcentaje de acero requerido
```

```
      ρ_punta = ((0.85 * self.fc) / self.fy) * (1 - (1 - (2 * Mu_punta * 100000)
/ (0.9 * 0.85 * self.fc * 100 * d_punta ** 2)) ** 0.5) # *100000 para que Mu
sean kg.cm

      # Cuantía de acero mínimo requerido
      ρ_min_punta = ((0.7 * self.fc ** 0.5) / self.fy)

      # Si la cuantía de acero mínima es mayor que la calculada, usar esa
      if ρ_min_punta > ρ_punta:
        ρ_punta = ρ_min_punta

      # Área de acero requerida
      As_punta = ρ_punta * 100 * d_punta # Se analiza un ancho de muro de 1 m =
100 cm (cm2)

      var_necesarias_punta = int(np.ceil(As_punta / a_var))

      # Porcentaje de acero necesario calculado con el área de las varillas
totales utilizadas
      ρt_punta = (var_necesarias_punta * a_var) / (100 * d_punta)

      # indice de refuerzo a tension
      q_punta = (ρt_punta * self.fy) / (0.85 * self.fc)

      Mn_punta = round((0.9 * 100 * (d_punta ** 2) * 0.85 * self.fc * q_punta *
(1 - 0.5 * q_punta)) / 100000, 2) # Momento resistente (t.m)

      separacion_punta = np.floor((100 - 2 * self.rec_libre) /
var_necesarias_punta) # sep_basearación centro a centro de las varillas

      l_db1_punta = (a_var * self.fy) / (3 * rec_efect * (self.fc ** 0.5))
      l_db2_punta = 30 # cm
      if l_db1_punta > l_db2_punta:
        ld_punta = l_db1_punta
      else:
        ld_punta = l_db2_punta

      self.long_var_punta = np.ceil(self.base_punta * 100 + ld_punta) # Longitud
necesaria de la varilla (cm)

      Vu_punta = round(diagrama_punta.area, 2) # Cortante útlimo de diseño (t)

      # Cálculo del cortante resistente Vc
      Vc_punta_77 = round((0.75 * (0.2 + 20 * ρt_punta) * (self.fc ** 0.5) * 100
* d_punta) / 1000, 2)

      Vc_punta_78 = round((0.75 * 0.5 * (self.fc ** 0.5) * 100 * d_punta) /
1000, 2)

      Vc_punta_79 = abs(round(((0.75 * (3.5 - 2.5 * ((Mu_punta*100000) /
((Vu_punta*1000) * d_punta)))
                               * 0.5 * (self.fc ** 0.5) * 100 * d_punta)) / 1000,
2))

      # 5.3.3.1a Elementos sin presfuerzo
      if (self.base_punta / self.altura_talon) >= 5: # Cuando L/h >= 5
        if ρt_punta < 0.015:
          Vc_punta = Vc_punta_77 # (t)
```

```
        elif pt_punta >= 0.015:
          Vc_punta = Vc_punta_78 # (t)

      elif (self.base_punta / self.altura_talon) <= 4: # Cuando L/h < 4
        Vc_punta = Vc_punta_79

      elif (self.base_punta / self.altura_talon) > 4 and (self.base_punta /
self.altura_talon) < 5: # Interpolación lineal L/h > 4 and < 5
        if pt_punta < 0.015:
          dif1 = (self.base_punta / self.altura_talon) - 4
          dif2 = abs(Vc_punta_77 - Vc_punta_79)
          Vc_punta = ((dif1 * dif2) / 1) + Vc_punta_77
        if pt_punta >= 0.015:
          dif1 = (self.base_punta / self.altura_talon) - 4
          dif2 = abs(Vc_punta_78 - Vc_punta_79)
          Vc_punta = ((dif1 * dif2) / 1) + Vc_punta_78

      # 5.3.3.1b Elementos achos
      if 100 < (4 * d_punta) and self.altura_talon <= 0.6 and (Mu_punta /
(Vu_punta * d_punta)) <= 2:
        Vc_punta = round((0.75 * 0.5 * (self.fc ** 0.5) * 100 * d_punta) / 1000,
2) # (t)

      elif self.altura_talon >= 0.6 and (Mu_punta / (Vu_punta * d_punta)) > 2:
        # 5.3.3.1a Elementos sin presfuerzo
        if (self.base_punta / self.altura_talon) >= 5: # Cuando L/h >= 5
          if pt_punta < 0.015:
            Vc_punta = Vc_punta_77 # (t)
          elif pt_punta >= 0.015:
            Vc_punta = Vc_punta_78 # (t)

        elif (self.base_punta / self.altura_talon) <= 4: # Cuando L/h < 4
          Vc_punta = Vc_punta_79

        elif (self.base_punta / self.altura_talon) > 4 and (self.base_punta /
self.altura_talon) < 5: # Interpolación lineal L/h > 4 and < 5
          if pt_pantalla < 0.015:
            dif1_pun = (self.base_punta / self.altura_talon) - 4
            dif2_pun = abs(Vc_punta_77 - Vc_punta_79)
            Vc_punta = round(((dif1_pun * dif2_pun) / 1) + Vc_punta_77, 2)
          if pt_pantalla >= 0.015:
            dif1_pun = (self.base_punta / self.altura_talon) - 4
            dif2_pun = abs(Vc_punta_78 - Vc_punta_79)
            Vc_punta = round(((dif1_pun * dif2_pun) / 1) + Vc_punta_78, 2)

      # Valor máximo de cortante que puede tomar el concreto (t)
      Vc_max_punta = round((0.75 * 1.5 * (self.fc ** 0.5) * 100 * d_punta) /
1000, 2)
      if Vc_punta > Vc_max_punta:
        Vc_punta = Vc_max_punta

      # Si ya pasa por cortante sale del ciclo infinito
      if Vc_punta >= Vu_punta:
        break
      else:
        self.altura_talon = self.altura_talon + 0.1
```

```
    self.resultado_punta = f"{int(var_necesarias_punta)} #
{self.var_propuesta_num} @ {separacion_punta} cm"


    ######    PARA EL TALÓN    ######

    while True:

      largo_talon = self.base_muro - self.base_punta - self.corona -
self.corona_aumento

      d_talon = (self.altura_talon * 100) - rec_efect # Peralte efectivo (cm)

      # Para obtener el momento último de diseño
      coords_diam_tal_x = np.array([0, 0, self.base_talon, self.base_talon])
      coords_diam_tal_y = np.array([0, F_pres_talon, F_pres_talon, 0])
      coords_diam_tal = np.stack((coords_diam_tal_x, coords_diam_tal_y), axis=1)
      diagrama_talon = Polygon(coords_diam_tal)

      Mu_talon = round(F_pres_talon * (self.base_talon / 2), 2) # Momento último
de diseño 1/2 q l^2 (t·m)

      # Porcentaje de acero requerido
      ρ_talon = ((0.85 * self.fc) / self.fy) * (1 - (1 - (2 * Mu_talon * 100000)
/ (0.9 * 0.85 * self.fc * 100 * d_talon ** 2)) ** 0.5)

      # Porcentaje de acero mínimo
      ρ_min_talon = ((0.7 * self.fc ** 0.5) / self.fy)

      # Si el porcentaje de acero mínimo es mayor, usar ese
      if ρ_min_talon > ρ_talon:
        ρ_talon = ρ_min_talon

      # Área de acero requerida
      As_talon = ρ_talon * 100 * d_talon # Se analiza un ancho de muro de 1 m =
100 cm (cm2)

      var_necesarias_talon = int(np.ceil(As_talon / a_var))

      # Porcentaje de acero necesario calculado con el área de las varillas
totales utilizadas
      ρt_talon = (var_necesarias_talon * a_var) / (100 * d_talon)

      # Índice de refuerzo a tensión
      q_talon = (ρt_talon * self.fy) / (0.85 * self.fc)

      Mn_talon = round((0.9 * 100 * (d_talon ** 2) * 0.85 * self.fc * q_talon *
(1 - 0.5 * q_talon)) / 100000, 2) # Momento resistente (kN.m)

      separacion_talon = np.floor((100 - 2 * self.rec_libre) /
var_necesarias_talon) # sep_baseaeración centro a centro de las varillas

      l_db1_talon = (a_var * self.fy) / (3 * rec_efect * (self.fc ** 0.5))
      l_db2_talon = 30 # cm
      if l_db1_talon > l_db2_talon:
        ld_talon = l_db1_talon
      else:
        ld_talon = l_db2_talon
```

```
      self.long_var_talon = np.ceil(self.base_talon * 100 + ld_talon) # Longitud
necesaria de la varilla (cm)

      Vu_talon = round(F_pres_talon, 2) # Cortante útlimo de diseño (t)

      # Cálculo del cortante resistente Vc
      Vc_talon_77 = round((0.75 * (0.2 + 20 * pt_talon) * (self.fc ** 0.5) * 100
* d_talon) / 1000, 2)

      Vc_talon_78 = round((0.75 * 0.5 * (self.fc ** 0.5) * 100 * d_talon) /
1000, 2)

      Vc_talon_79 = abs(round(((0.75 * (3.5 - 2.5 * ((Mu_talon*100000) /
((Vu_talon*1000) * d_talon)))
                               * 0.5 * (self.fc ** 0.5) * 100 * d_talon)) / 1000,
2))

      # 5.3.3.1a Elementos sin presfuerzo
      if (largo_talon / self.altura_talon) >= 5: # Cuando L/h >= 5
        if pt_talon < 0.015:
          Vc_talon = Vc_talon_77 # (t)
        elif pt_talon >= 0.015:
          Vc_talon = Vc_talon_78 # (t)

      elif (largo_talon / self.altura_talon) <= 4: # Cuando L/h < 4
        Vc_talon = Vc_talon_79

      elif (largo_talon / self.altura_talon) > 4 and (largo_talon /
self.altura_talon) < 5: # Interpolación lineal L/h > 4 and < 5
        if pt_talon < 0.015:
          dif1 = (largo_talon / self.altura_talon) - 4
          dif2 = abs(Vc_talon_77 - Vc_talon_79)
          Vc_talon = ((dif1 * dif2) / 1) + Vc_talon_77
        if pt_talon >= 0.015:
          dif1 = (largo_talon / self.altura_talon) - 4
          dif2 = abs(Vc_talon_78 - Vc_talon_79)
          Vc_talon = ((dif1 * dif2) / 1) + Vc_talon_78

      # 5.3.3.1b Elementos achos
      if 100 < (4 * d_talon) and self.altura_talon <= 0.6 and (Mu_talon /
(Vu_talon * d_talon)) <= 2:
        Vc_talon = round((0.75 * 0.5 * (self.fc ** 0.5) * 100 * d_talon) / 1000,
2) # (t)

      elif self.altura_talon >= 0.6 and (Mu_talon / (Vu_talon * d_talon)) > 2:
        # 5.3.3.1a Elementos sin presfuerzo
        if (largo_talon / self.altura_talon) >= 5: # Cuando L/h >= 5
          if pt_talon < 0.015:
            Vc_talon = Vc_talon_77 # (t)
          elif pt_talon >= 0.015:
            Vc_talon = Vc_talon_78 # (t)

        elif (largo_talon / self.altura_talon) <= 4: # Cuando L/h < 4
          Vc_talon = Vc_pantalla_79

        elif (largo_talon / self.altura_talon) > 4 and (largo_talon /
self.altura_talon) < 5: # Interpolación lineal L/h > 4 and < 5
```

```
          if pt_pantalla < 0.015:
            dif1_tal = (largo_talon / self.altura_talon) - 4
            dif2_tal = abs(Vc_talon_77 - Vc_talon_79)
            Vc_talon = round(((dif1_tal * dif2_tal) / 1) + Vc_talon_77, 2)
          if pt_pantalla >= 0.015:
            dif1_tal = (largo_talon / self.altura_talon) - 4
            dif2_tal = abs(Vc_talon_78 - Vc_talon_79)
            Vc_talon = round(((dif1_tal * dif2_tal) / 1) + Vc_talon_78, 2)

      # Valor máximo de cortante que puede tomar el concreto (t)
      Vc_max_talon = round((0.75 * 1.5 * (self.fc ** 0.5) * 100 * d_talon) /
1000, 2)
      if Vc_talon > Vc_max_talon:
        Vc_talon = Vc_max_talon


      # Si ya pasa por cortante sale del ciclo infinito
      if Vc_talon >= Vu_talon:
        break
      else:
        self.altura_talon = self.altura_talon + 0.1

    self.resultado_talon = f"{int(var_necesarias_talon)} #
{self.var_propuesta_num} @ {separacion_talon} cm"


    ######    PARA LA PANTALLA    ######

    while True:

      peralte_pantalla = self.corona_aumento + self.corona
      d_pantalla = (peralte_pantalla * 100) - rec_efect # Peralte efectivo (cm)

      Mu_pantalla = round(F_pres_pan * self.brazo_palanca, 2) # Momento último
de diseño (t·m)

      # Porcentaje de acero requerido
      ρ_pantalla = ((0.85 * self.fc) / self.fy) * (1 - (1 - (2 * Mu_pantalla *
100000) / (0.9 * 0.85 * self.fc * 100 * d_pantalla ** 2)) ** 0.5)

      # Porcentaje de acero mínimo
      ρ_min_pantalla = ((0.7 * self.fc ** 0.5) / self.fy)

      # Si el porcentaje mínimo es mayor, usar el mínimo
      if ρ_min_pantalla > ρ_pantalla:
        ρ_pantalla = ρ_min_pantalla

      # Área de acero requerida
      As_pantalla = ρ_pantalla * 100 * d_pantalla # Se analiza un ancho de muro
de 1 m = 100 cm (cm2)

      var_necesarias_pantalla = int(np.ceil(As_pantalla / a_var))

      # Porcentaje de acero calculado con las varillas que se van a utilizar
      pt_pantalla = (var_necesarias_pantalla * a_var) / (100 * d_pantalla)

      # Índice de refuerzo a tensión
      q_pantalla = (pt_pantalla * self.fy) / (0.85 * self.fc)
```

```
      Mn_pantalla = round((0.9 * 100 * (d_pantalla ** 2) * 0.85 * self.fc *
q_pantalla * (1 - 0.5 * q_pantalla)) / 100000, 2) # Momento resistente (t.m)

      separacion_pantalla = np.floor((100 - 2 * self.rec_libre) /
var_necesarias_pantalla) # separación centro a centro de las varilla

      l_db1_pantalla = (a_var * self.fy) / (3 * rec_efect * (self.fc ** 0.5))
      l_db2_pantalla = 30 # cm
      if l_db1_pantalla > l_db2_pantalla:
        ld_pantalla = l_db1_pantalla
      else:
        ld_pantalla = l_db2_pantalla

      l_pantalla = self.altura_muro - self.altura_talon
      self.long_var_pantalla = np.ceil(l_pantalla * 100 + ld_pantalla) #
Longitud necesaria de la varilla (cm)

      # Colocar aquí la variable correcta
      Vu_pantalla = round(F_pres_pan, 2) # Cortante útlimo de diseño (kN)


      # Cálculo del cortante resistente Vc
      Vc_pantalla_77 = round((0.75 * (0.2 + 20 * pt_pantalla) * (self.fc ** 0.5)
* 100 * d_pantalla) / 1000, 2)

      Vc_pantalla_78 = round((0.75 * 0.5 * (self.fc ** 0.5) * 100 * d_pantalla)
/ 1000, 2)

      Vc_pantalla_79 = abs(round(((0.75 * (3.5 - 2.5 * ((Mu_pantalla*100000) /
((Vu_pantalla*1000) * d_pantalla)))
                                  * 0.5 * (self.fc ** 0.5) * 100 * d_pantalla)) /
1000, 2))

      # 5.3.3.1a Elementos sin presfuerzo
      if (l_pantalla / peralte_pantalla) >= 5: # Cuando L/h >= 5
        if pt_pantalla < 0.015:
          Vc_pantalla = Vc_pantalla_77 # (t)
        elif pt_pantalla >= 0.015:
          Vc_pantalla = Vc_pantalla_78 # (t)

      elif (l_pantalla / self.corona) <= 4: # Cuando L/h < 4
        Vc_pantalla = Vc_pantalla_79

      elif (l_pantalla / peralte_pantalla) > 4 and (l_pantalla /
peralte_pantalla) < 5: # Interpolación lineal L/h > 4 and < 5
        if pt_pantalla < 0.015:
          dif1 = (l_pantalla / peralte_pantalla) - 4
          dif2 = abs(Vc_pantalla_77 - Vc_pantalla_79)
          Vc_pantalla = ((dif1 * dif2) / 1) + Vc_pantalla_77
        if pt_pantalla >= 0.015:
          dif1 = (l_pantalla / peralte_pantalla) - 4
          dif2 = abs(Vc_pantalla_78 - Vc_pantalla_79)
          Vc_pantalla = ((dif1 * dif2) / 1) + Vc_pantalla_78

      # 5.3.3.1b Elementos achos
      if 100 < (4 * d_pantalla) and peralte_pantalla <= 0.6 and (Mu_pantalla /
(Vu_pantalla * d_pantalla)) <= 2:
```

```
        Vc_pantalla = round((0.75 * 0.5 * (self.fc ** 0.5) * 100 * d_pantalla) /
1000, 2) # (t)

      elif peralte_pantalla >= 0.6 and (Mu_pantalla / (Vu_pantalla *
d_pantalla)) > 2:
        # 5.3.3.1a Elementos sin presfuerzo
        if (l_pantalla / peralte_pantalla) >= 5: # Cuando L/h >= 5
          if ρt_pantalla < 0.015:
            Vc_pantalla = Vc_pantalla_77 # (t)
          elif ρt_pantalla >= 0.015:
            Vc_pantalla = Vc_pantalla_78 # (t)

        elif (l_pantalla / peralte_pantalla) <= 4: # Cuando L/h < 4
          Vc_pantalla = Vc_pantalla_79

        elif (l_pantalla / peralte_pantalla) > 4 and (l_pantalla /
peralte_pantalla) < 5: # Interpolación lineal L/h > 4 and < 5
          if ρt_pantalla < 0.015:
            dif1_pan = (l_pantalla / peralte_pantalla) - 4
            dif2_pan = abs(Vc_pantalla_77 - Vc_pantalla_79)
            Vc_pantalla = round(((dif1_pan * dif2_pan) / 1) + Vc_pantalla_77, 2)
          if ρt_pantalla >= 0.015:
            dif1_pan = (l_pantalla / peralte_pantalla) - 4
            dif2_pan = abs(Vc_pantalla_78 - Vc_pantalla_79)
            Vc_pantalla = round(((dif1_pan * dif2_pan) / 1) + Vc_pantalla_78, 2)

      # Valor máximo de cortante que puede tomar el concreto (t)
      Vc_max_pantalla = round((0.75 * 1.5 * (self.fc ** 0.5) * 100 * d_pantalla)
/ 1000, 2)
      if Vc_pantalla > Vc_max_pantalla:
        Vc_pantalla = Vc_max_pantalla


      # Si ya pasa por cortante sale del ciclo infinito
      if Vc_pantalla >= Vu_pantalla:
        break
      else:
        self.corona_aumento = self.corona_aumento + 0.1

    self.resultado_pantalla = f"{int(var_necesarias_pantalla)} #
{self.var_propuesta_num} @ {separacion_pantalla} cm"


    R_punta = np.array([var_necesarias_punta, self.var_propuesta_num,
self.long_var_punta, separacion_punta, Mu_punta, Mn_punta, Vu_punta, Vc_punta])
    R_talon = np.array([var_necesarias_talon, self.var_propuesta_num,
self.long_var_talon, separacion_talon, Mu_talon, Mn_talon, Vu_talon, Vc_talon])
    R_pantalla = np.array([var_necesarias_pantalla, self.var_propuesta_num,
self.long_var_pantalla, separacion_pantalla, Mu_pantalla, Mn_pantalla,
Vu_pantalla, Vc_pantalla])
    return R_punta, R_talon, R_pantalla

# Si quisiera regresar los resultados de Mu, Mc, Vu, Vc en kN·m o kN solo tengo
que multiplicarlos por 9.81

  def acero_contraccion(self):
    """
```

```
    CALCULA EL ACERO POR CONTRACCIÓN Y TEMPERATURA NECEARIO PARA CADA UNO DE LOS ELEMENTOS DEL MURO EN VOLADIZO
    """

    ρ = 0.003 # Porcentaje de acero mínimo por temparatura NTC-17

    a_var = (np.pi * ((self.var_propuesta_num / 8) * 2.54) ** 2) / 4 # Área de la varilla en cm2
    db = round((self.var_propuesta_num / 8) * 2.54, 2) # Diámetro de varilla en cm

    # PARA LA PUNTA Y EL TALÓN DEL MURO
    As_base = round(ρ * self.base_muro * self.altura_talon * 10000, 2) # Área de acero requerida en función de las dimensiones de la base

    cant_var_base = np.ceil((As_base / a_var) / 2) * 2 # Cantidad de varillas necesarias de acuerdo al diámetro propuesto
    self.cant_var_base = cant_var_base

    L_base = round((self.base_muro * 100) - (2 * self.rec_libre)) # Longitud de la base en la que se deben distribuir las varillas

    s_base = np.floor((L_base / (cant_var_base - 2)) * 2) # Separación de la varilla
    self.s_base = s_base

    Long_temp_base = 100 - (2 * self.rec_libre)


    # PARA LA PANTALLA DEL MURO
    # Acero horizontal
    As_pantalla_hor = round(ρ * (self.corona_aumento+self.corona) * (self.altura_muro-self.altura_talon) * 10000, 2)
    cant_var_pan_hor = np.ceil((As_pantalla_hor / a_var) / 2) * 2 # Cantidad de varillas necesarias de acuerdo al diámetro propuesto
    self.cant_var_pan_hor = cant_var_pan_hor

    L_pantalla_hor = round(((self.altura_muro-self.altura_talon) * 100) - (2 * (self.rec_libre)), 0)

    s_pantalla_hor = np.floor((L_pantalla_hor / (cant_var_pan_hor - 2)) * 2) # separación de la varilla va en ambos lechos
    self.s_pantalla_hor = s_pantalla_hor

    Long_temp_pan_h = 100 - (2 * self.rec_libre)

    # Acero vertical
    As_pantalla_ver = round(ρ * (self.corona_aumento+self.corona) * 1 * 10000, 2)
    cant_var_pan_ver = np.ceil(As_pantalla_ver / a_var) # Cantidad de varillas necesarias de acuerdo al diámetro propuesto
    self.cant_var_pan_ver = cant_var_pan_ver

    L_pantalla_ver = round((self.altura_muro-self.altura_talon) * 100 - 2 * (self.rec_libre), 0)

    s_pantalla_ver = np.floor(((100 - 2*self.rec_libre) / (cant_var_pan_ver - 1))) # separación de la varilla va solo en 1 lecho
```

```
    self.s_pantalla_ver = s_pantalla_ver


    # Strings con los resultados para ponerlos en el diagrama
    self.resultado_cont_base = f"{int(cant_var_base)} # {self.var_propuesta_num}
@ {s_base} cm"
    self.resultado_cont_pantalla_h = f"{int(cant_var_pan_hor)} #
{self.var_propuesta_num} @ {s_pantalla_hor} cm"
    self.resultado_cont_pantalla_v = f"{int(cant_var_pan_ver)} #
{self.var_propuesta_num} @ {s_pantalla_ver} cm"

    Cont_base = np.array([cant_var_base, s_base, Long_temp_base])
    Cont_pantalla_h = np.array([cant_var_pan_hor, s_pantalla_hor,
Long_temp_pan_h])
    Cont_pantalla_v = np.array([cant_var_pan_ver, s_pantalla_ver,
L_pantalla_ver])
    return Cont_base, Cont_pantalla_h, Cont_pantalla_v



  def dibujar_acero(self):
    """
    DIBUJA EL DIAGRAMA DEL ARMADO DEL MURO EN VOLADIZO
    """

    # ACERO A TENSIÓN
    # Varillas de la punta
    punta_x = np.array([self.rec_libre/100 , self.long_var_punta/100])
    punta_y = np.array([self.rec_libre/100, self.rec_libre/100])

    # Varillas del talón
    talon_x = np.array([(self.base_muro-
(self.rec_libre+self.long_var_talon)/100), (self.base_muro-self.rec_libre/100)])
    talon_y = np.array([self.altura_talon-self.rec_libre/100, self.altura_talon-
self.rec_libre/100])

    # Varillas de la pantalla
    h_pantalla = self.altura_muro - self.altura_talon
    pantalla_x = np.array([self.base_punta+self.corona+self.corona_aumento-
self.rec_libre/100,
                           self.base_punta+self.corona+self.corona_aumento-
self.rec_libre/100])
    pantalla_y = np.array([self.altura_talon+h_pantalla-self.rec_libre/100,
                           (self.altura_talon+h_pantalla-self.rec_libre/100)-
self.long_var_pantalla/100])

    fig, ax = plt.subplots(figsize=(6,6))
    # ax.fill(muro_x, muro_y, color="gray")
    # Poligono(self.coords_poly, color='darkgray', ColorBorde='darkgray',
alpha=1)


    # Array con las dimensinoes finales en x del muro
    x_final = np.array([0, 0, self.base_punta,
self.base_punta+self.corona_aumento,
self.base_punta+self.corona+self.corona_aumento,
                  self.base_punta+self.corona+self.corona_aumento,
self.base_muro, self.base_muro])
```

```
    # Array con las dimensiones finales en y del muro
    y_final = np.array([0, self.altura_talon, self.altura_talon,
self.altura_muro, self.altura_muro, self.altura_talon, self.altura_talon, 0])

    coords_final = np.stack((x_final, y_final), axis=1)
    Poligono(coords_final, color='darkgray', ColorBorde='darkgray', alpha=1)

    # Cotas
    DibujaCota((self.base_muro+0.2,0), (self.base_muro+0.2,self.altura_talon),
dist=-0.2, lw=2, AnchoFlecha=0.05, color="black") # altura talón
    DibujaCota((self.base_punta,-0.5),
(self.base_punta+self.corona_aumento+self.corona,-0.5),
                dist=-0.5, lw=2, AnchoFlecha=0.05, color="black") # ancho corona
aumento + corona

    plt.title("Diagrama del armado del muro en voladizo")
    ax.plot(punta_x, punta_y, color="red")
    ax.text(-0.8, -0.2, self.resultado_punta, color="red") # acero de la punta
    ax.plot(talon_x, talon_y, color="red", label="Acero a tensión")
    ax.text(self.base_punta+self.corona+self.base_talon/4,
self.altura_talon+0.1, self.resultado_talon, color="red") # acero del talón
    ax.plot(pantalla_x, pantalla_y, color="red")
    ax.text(self.base_punta+self.corona+self.corona_aumento+0.2,
self.altura_talon+h_pantalla/2, self.resultado_pantalla, color="red") # acero de
la pantalla


    # Grafica los puntos correspondientes a las varillas de acero por contracción
y temperatura
    # Base
    for i in range(int(self.cant_var_base / 2)):
      ax.plot((self.rec_libre + self.s_base * i) / 100, self.rec_libre / 100,
marker="o", color="black")
      ax.plot((self.rec_libre + self.s_base * i) / 100, self.altura_talon -
self.rec_libre / 100, marker="o", color="black")

    # Pantalla
    # Refuerzo horizontal
    incremento = self.corona_aumento / (self.cant_var_pan_hor / 2)
    for k in range(int(self.cant_var_pan_hor / 2)):
      ax.plot(self.base_punta + self.corona + self.corona_aumento -
(self.rec_libre / 100), self.altura_talon + h_pantalla
              -(self.s_pantalla_hor / 100) * k, marker="o", color="black") #
lado derecho
      ax.plot(self.base_punta + self.corona_aumento + (self.rec_libre / 100) -
(incremento * k), self.altura_talon + h_pantalla
              -(self.s_pantalla_hor / 100) * k, marker="o", color="black") #
lado izquierdo

    # Refuerzo vertical
    pantalla_x_hor = np.array([self.base_punta+self.rec_libre/100 +
self.corona_aumento, self.base_punta+self.rec_libre/100])
    pantalla_y_hor = np.array([self.altura_muro-self.rec_libre/100,
self.altura_talon+self.rec_libre/100])
    ax.plot(pantalla_x_hor, pantalla_y_hor, color="black", label="Acero
temperatura")
```

```
    ax.text(self.base_muro/4, self.altura_talon/3, self.resultado_cont_base)
    ax.text(-1.2, self.altura_muro/2, f"H: {self.resultado_cont_pantalla_h}")
    ax.text(-1.2, self.altura_muro/1.7, f"V: {self.resultado_cont_pantalla_v}")

    ax.set_xlim(-1.3, self.base_muro+1)
    ax.set_ylim(-1, self.altura_muro+1)
    ax.set_aspect('equal', "box")
    ax.legend(loc="upper left", fontsize="small")

 plt.savefig("/content/drive/MyDrive/Tesis_python/Reporte_descargar/Diagrama_acero", dpi=250)
    plt.show()


######################### EMPUJES ##########################

# RECOPILANDO LAS VARIABLES DEL USUARIO
# Para la presión activa de Rankine
Label_PA = ipw.Label(value="PRESIÓN ACTIVA DE RANKINE",
layout=Layout(display="flex", justify_content="center"))

Hay_NAF = ipw.Dropdown(
    options=[("El muro está drenado",1), ("El muro no está drenado",2)],
    value=1,
    description="Muro drenado",
    layout=Layout(width="280px", display="flex", justify_content="center"),
    disabled=False)

NAF = ipw.FloatText(value=3.6, min=0, step=1, description='NAF (m)',
layout=Layout(width='180px', visibility="hidden"))
sobrecarga = ipw.FloatText(value=0, min=0, step=1, description='Sobrecarga (kPa)', layout=Layout(width='180px'))
γms_1 = ipw.FloatText(value=17.3, min=0, step=1, description='γm (kN/m3)',
layout=Layout(width='180px'))
γms_2 = ipw.FloatText(value=19.6, min=0, step=1, description='',
layout=Layout(width='90px'))
γms_3 = ipw.FloatText(value=19.7, min=0, step=1, description='',
layout=Layout(width='90px'))
Hs_1 = ipw.FloatText(value=1.8, min=0, step=1, description='Espesor (m)',
layout=Layout(width='180px'))
Hs_2 = ipw.FloatText(value=1.6, min=0, step=1, description='',
layout=Layout(width='90px'))
Hs_3 = ipw.FloatText(value=1.2, min=0, step=1, description='',
layout=Layout(width='90px'))
φ_1 = ipw.FloatText(value=32, min=0, step=1, description='φ (°)',
layout=Layout(width='180px'))
φ_2 = ipw.FloatText(value=30, min=0, step=1, description='',
layout=Layout(width='90px'))
φ_3 = ipw.FloatText(value=25, min=0, step=1, description='',
layout=Layout(width='90px'))
c_1 = ipw.FloatText(value=0.0, min=0, step=1, description='c (kPa)',
layout=Layout(width='180px'))
c_2 = ipw.FloatText(value=12.0, min=0, step=1, description='',
layout=Layout(width='90px'))
c_3 = ipw.FloatText(value=5.0, min=0, step=1, description='',
layout=Layout(width='90px'))
β = ipw.FloatText(value=0, min=0, step=1, description='β (°)',
layout=Layout(width='180px'))
```

```
# Para la sobrecarga
Label_Sobrecarga = ipw.Label(value="PRESIÓN DEBIDO A
SOBRECARGA",layout=Layout(display="flex", justify_content="center"))
B = ipw.FloatText(value=2, min=0, step=1, description='B (m)',
layout=Layout(width='180px'))
W = ipw.FloatText(value=4, min=0, step=1, description='W (m)',
layout=Layout(width='180px'))
q = ipw.FloatText(value=100, min=0, step=1, description='q (kPa)',
layout=Layout(width='180px', visibility="hidden"))
μ = ipw.FloatText(value=0.5, min=0, step=1, description='μ',
layout=Layout(width='180px'))
NSQW = ipw.FloatText(value=10, min=0, step=1, description='NSQW',
layout=Layout(width='180px'))
NSQL = ipw.FloatText(value=10, min=0, step=1, description='NSQL',
layout=Layout(width='180px'))
HTWALL = ipw.FloatText(value=4.6, min=0, step=1, description='Altura (m)',
layout=Layout(width='180px'))
DTWALL = ipw.FloatText(value=3, min=0, step=1, description='DTWALL (m)',
layout=Layout(width='180px'))
NVERT = ipw.FloatText(value=100, min=0, step=1, description='NVERT',
layout=Layout(width='180px'))
P = ipw.FloatText(value=5, min=0, step=1, description='P (kN)',
layout=Layout(width='180px', visibility="hidden"))

tipo_de_sobrecarga = ipw.Dropdown(
    options=[("No existe sobrecarga",1), ("Área con carga constante",2), ("Área
con variación lineal ascendente",3), \
             ("Área con variación lineal descendente",4), ("Sobrecarga
puntual",5)],
    value=1,
    description="Tipo sobrecarga:",
    layout=Layout(width="280px", display="flex", justify_content="center"),
    disabled=False)

# Para la presión pasiva
Label_PP = ipw.Label(value="PRESIÓN PASIVA DE RANKINE",
layout=Layout(display="flex", justify_content="center"))
γm_p1 = ipw.FloatText(value=16, min=0, step=1, description='γm (kN/m3)',
layout=Layout(width='180px'))
Hs_p1 = ipw.FloatText(value=0.6, min=0, step=1, description='Prof. desplante
(m)', layout=Layout(width='180px'))
φ_p1 = ipw.FloatText(value=30, min=0, step=1, description='φ (°)',
layout=Layout(width='180px'))
c_p1 = ipw.FloatText(value=5.0, min=0, step=1, description='c (kPa)',
layout=Layout(width='180px'))

# Para el incremento de presión debido a sismo
Label_Sismo = ipw.Label(value="PRESION DEBIDO A SISMO",
layout=Layout(display="flex", justify_content="center"))
A0 = ipw.FloatText(value=0.25, min=0, step=1, description='A0',
layout=Layout(width='180px'))


# Para los resultados
Label_Resultados = ipw.Label(value="PRESIÓN LATERAL TOTAL",
layout=Layout(display="flex", justify_content="center"))
```

```
Label_E1 = ipw.Label(value="La presión activa según la teoría de Rankine es de: (kN)")
Text_E1 = ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Label_R1 = ipw.Label(value="La resultante a partir de la base del muro está a: (m)")
Text_R1 = ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Label_E2 = ipw.Label(value="La presión pasiva según la teoría de Rankine es de: (kN)")
Text_E2 = ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Label_R2 = ipw.Label(value="La resultante a partir de la base del muro está a: (m)")
Text_R2 = ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Label_E3 = ipw.Label(value="La presión debido a sobrecarga es de: (kN)")
Text_E3 = ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Label_R3 = ipw.Label(value="La resultante a partir de la base del muro está a: (m)")
Text_R3 = ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Label_ESismo = ipw.Label(value="El incremento de presión debido a sismo es de: (kN)")
Text_ESismo = ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Label_RSismo = ipw.Label(value="La resultante a partir de la base del muro está a: (m)")
Text_RSismo = ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Label_ESuma = ipw.Label(value="La presión lateral total es de: (kN)")
Text_ESuma = ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Label_RSuma = ipw.Label(value="La resultante a partir de la base del muro está a: (m)")
Text_RSuma = ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))


out = ipw.Output()
# BOTÓN DEL CÁLCULO
BotonCalculo = ipw.Button(description = "Calcular", button_style="primary")
def BotonCalculo_clicked(b):
  !rm /content/drive/MyDrive/Tesis_python/Reporte_descargar/Diagramas_presiones.png
  !rm /content/drive/MyDrive/Tesis_python/Reporte_descargar/Diagrama_muro.png
  !rm /content/drive/MyDrive/Tesis_python/Reporte_descargar/Diagrama_acero.png

  global Hs, γms, φ, c
  Hs = np.array([Hs_1.value, Hs_2.value, Hs_3.value])
  γms = np.array([γms_1.value, γms_2.value, γms_3.value]) # Almacena los valores del FloatText en una np.array
  γms = np.delete(γms, np.where(Hs <= 0))
  φ = np.array([φ_1.value, φ_2.value, φ_3.value])
  φ = np.delete(φ, np.where(Hs <= 0))
  c = np.array([c_1.value, c_2.value, c_3.value])
  c = np.delete(c, np.where(Hs <= 0))
  Hs = np.delete(Hs, np.where(Hs <= 0)) # Guarda solo los elementos mayores que 0, (quiere decir que no existe el estrato)
  γm_p = np.array([γm_p1.value])
  Hs_p = np.array([Hs_p1.value])
  φ_p = np.array([φ_p1.value])
  c_p = np.array([c_p1.value])

  # Llama a la clase Empuje_muros para crear una instancia (iniciarla)
  Empujes = Empuje_muros(NAF.value, sobrecarga.value, γms, Hs, φ, c, β.value, # Hay_NAF es para la presión activa como en PARCPEM
```

```
                        B.value, W.value, q.value, μ.value, NSQW.value,
NSQL.value, HTWALL.value, DTWALL.value, NVERT.value, P.value,
                        γm_p, Hs_p, φ_p, c_p, A0.value, tipo_de_sobrecarga.value,
Hay_NAF.value) # tipo_de_sobrecarga es para la condicion de acendente o
descendente

  E1, R1 = Empujes.presion_activa_rankine()
  E2, R2 = Empujes.presion_pasiva_rankine()
  if tipo_de_sobrecarga.value == 1:
    E3 = 0
    R3 = 0
  elif tipo_de_sobrecarga.value == 2:
    E3, R3 = Empujes.prelat_sobrecarga_areacargada1()
  elif tipo_de_sobrecarga.value == 3 or tipo_de_sobrecarga.value == 4:
    E3, R3 = Empujes.prelat_sobrecarga_linearload()
  elif tipo_de_sobrecarga.value == 5:
    E3, R3 = Empujes.prelat_sobrecarga_cargapuntual()
  E_Sismo, R_Sismo = Empujes.presion_activa_okabe()
  ESuma, RSuma = Empujes.suma()

  Text_E1.value = E1
  Text_R1.value = R1
  Text_E2.value = E2
  Text_R2.value = R2
  Text_E3.value = E3
  Text_R3.value = R3
  Text_ESismo.value = E_Sismo
  Text_RSismo.value = R_Sismo
  Text_ESuma.value = ESuma
  Text_RSuma.value = RSuma

  # Para Graficar los diagramas
  out.clear_output() # Para borrar los diagramas anteriores
  with out:
    Empujes.graficar()

BotonCalculo.on_click(BotonCalculo_clicked)

# ORDENANDO LAS VARIABLES EN CONTENEDORES

# Pestaña de datos
γ = ipw.HBox([γms_1, γms_2, γms_3]) # Presión activa de Rankine
H = ipw.HBox([Hs_1, Hs_2, Hs_3])
φ = ipw.HBox([φ_1, φ_2, φ_3])
c = ipw.HBox([c_1, c_2, c_3])
Contenedor_PA = ipw.VBox([Label_PA, HTWALL, sobrecarga, γ, H, φ, c, β, Hay_NAF,
NAF], layout=Layout(border="5px dashed gray", margin="10px"))

labels_sobrecarga = ipw.VBox([Label_Sobrecarga, tipo_de_sobrecarga])
datos_sobrecarga = ipw.VBox([B, W, μ, NSQW, NSQL, DTWALL, NVERT, q, P],
layout=Layout(visibility="hidden"))
Contenedor_sobrecarga = ipw.VBox([labels_sobrecarga, datos_sobrecarga],
layout=Layout(width="300px", border="5px dashed gray", margin="10px"))

Contenedor_Sismo = ipw.VBox([Label_Sismo, A0], layout=Layout(width="300px",
border="5px dashed gray", margin="10px"))
```

```
Contenedor_PP = ipw.VBox([Label_PP, γm_p1, Hs_p1, φ_p1, c_p1],
layout=Layout(width="300px", border="5px dashed gray", margin="10px"))

Contenedor_datos = ipw.HBox([Contenedor_PA, ipw.VBox([Contenedor_PP,
Contenedor_Sismo]), Contenedor_sobrecarga])

# Pestaña de resultados
CR1 = ipw.VBox([Label_PA, Label_E1, Text_E1, Label_R1, Text_R1],
layout=Layout(border="5px dashed blue", margin="10px", width="350px"))
CR2 = ipw.VBox([Label_PP, Label_E2, Text_E2, Label_R2, Text_R2],
layout=Layout(border="5px dashed orange", margin="10px", width="350px"))
CR3 = ipw.VBox([Label_Sobrecarga, Label_E3, Text_E3, Label_R3, Text_R3],
layout=Layout(border="5px dashed red", margin="10px", width="350px"))
CR4 = ipw.VBox([Label_Sismo, Label_ESismo, Text_ESismo, Label_RSismo,
Text_RSismo], layout=Layout(border="5px dashed green", margin="10px",
width="350px"))
CR5 = ipw.VBox([Label_Resultados, Label_ESuma, Text_ESuma, Label_RSuma,
Text_RSuma], layout=Layout(border="5px dashed purple", margin="10px",
width="350px"))
Contenedor_resultados = ipw.HBox([ipw.VBox([CR1, CR4, CR5]), ipw.VBox([CR2,
CR3])])

# Mostrar o no mostrar casillas para datos
def Ocultar_carga(value):
  if value == 1: # No hay
    datos_sobrecarga.layout.visibility = 'hidden'
    q.layout.visibility = 'hidden'
    P.layout.visibility = 'hidden'
    B.layout.visibility = 'hidden'
    W.layout.visibility = 'hidden'
  elif value == 2 or value == 3 or value == 4: # Todas menos puntual
    datos_sobrecarga.layout.visibility = 'visible'
    q.layout.visibility = 'visible'
    P.layout.visibility = 'hidden'
    B.layout.visibility = 'visible'
    W.layout.visibility = 'visible'
  elif value == 5: # Puntual
    datos_sobrecarga.layout.visibility = 'visible'
    q.layout.visibility = 'hidden'
    P.layout.visibility = 'visible'
    B.layout.visibility = 'hidden'
    W.layout.visibility = 'hidden'
ipw.interactive(Ocultar_carga, value=tipo_de_sobrecarga)

def Ocultar_NAF(value):
  if value == 1:
    NAF.layout.visibility = 'hidden'
  else:
    NAF.layout.visibility = 'visible'
ipw.interactive(Ocultar_NAF, value=Hay_NAF)


########################## MUROS ##########################

# RECOPILANDO LAS VARIABLES DEL USUARIO
# DATOS muro
Label_Muro = ipw.Label(value="DATOS DEL MURO", layout=Layout(display="flex",
justify_content="center"))
```

```
tipo_muro = ipw.RadioButtons(options=[("De gravedad",1), ("En voladizo",2)],
    value=1,
    layout={'width': 'max-content'},
    description='Tipo de muro',
    disabled=False)

γ_muro = ipw.FloatText(value=23.54, min=0, step=1, description='γ muro (kN/m3)',
layout=Layout(width='180px'))
coef_deslizam = ipw.FloatText(value=0.65, min=0, step=1, description='C.
fricción', layout=Layout(width='180px'))
α = ipw.FloatText(value=0, min=0, step=1, description='Inclinación de la base
del muro', layout=Layout(width='180px'))
NAF_CC = ipw.FloatText(value=1.5, min=0, step=1, description='NAF (m)',
layout=Layout(width='180px'))

# Contenedor muro
Contenedor_muro = ipw.VBox([Label_Muro, tipo_muro, γ_muro, coef_deslizam, α],
                           layout=Layout(width="300px", border="5px dashed
gray", margin="10px"))

# DATOS suelo de apoyo
Label_Apoyo = ipw.Label(value="SUELO DE APOYO", layout=Layout(display="flex",
justify_content="center"))

γ_apoyo = ipw.FloatText(value=19.7, min=0, step=1, description='γ apoyo
(kN/m3)', layout=Layout(width='180px'))
c_apoyo = ipw.FloatText(value=5.0, min=0, step=1, description='c apoyo (kPa)',
layout=Layout(width='180px'))
φ_apoyo = ipw.FloatText(value=25, min=0, step=1, description='φ apoyo (°)',
layout=Layout(width='180px'))

# Contenedor apoyo
Contenedor_apoyo = ipw.VBox([Label_Apoyo, γ_apoyo, c_apoyo, φ_apoyo, NAF_CC],
                           layout=Layout(width="300px", border="5px dashed
gray", margin="10px"))


# DATOS armado del muro
Label_Armado = ipw.Label(value="ARMADO DEL MURO", layout=Layout(display="flex",
justify_content="center"))

fc = ipw.FloatText(value=250, min=0, step=1, description="f'c",
layout=Layout(width='180px')) # 24516.6 kPa = 250 kg/cm2
fy = ipw.FloatText(value=4200, min=0, step=1, description="fy",
layout=Layout(width='180px')) # 411879.3 kPa = 4200 kg/cm2
E = ipw.FloatText(value=2100000, min=0, step=1, description="E",
layout=Layout(width='180px')) # 205939650 kPa = 2100000 kg/cm2
Φ = ipw.FloatText(value=0.9, min=0, step=1, description="Φ",
layout=Layout(width='180px'))
rec_libre = ipw.FloatText(value=7.5, min=0, step=1, description="Rec libre
(cm)", layout=Layout(width='180px'))
var_propuesta_num = ipw.FloatText(value=6, min=0, step=1, description="ø (in)",
layout=Layout(width='180px'))

# Contenedor armado
Contenedor_armado = ipw.VBox([Label_Armado, fc, fy, E, Φ, rec_libre,
var_propuesta_num],
```

```
                              layout=Layout(width="300px", border="5px dashed
gray", margin="10px", visibility='hidden'))

# RESULTADOS
Label_Estabilidad = ipw.Label(value="ESTABILIDAD DEL MURO",
layout=Layout(display="flex", justify_content="center"))

Label_FSV = ipw.Label(value="El factor de seguridad por volteo es de: (> 1.5 o >
2.0)")
Text_FSV = ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Label_FSD = ipw.Label(value="El factor de seguridad por deslizamiento es de: (>
1.5)")
Text_FSD = ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))

Label_CapacidadC = ipw.Label(value="CAPACIDAD DE CARGA DEL SUELO",
layout=Layout(display="flex", justify_content="center"))

Label_FSCC = ipw.Label(value="El factor de seguridad por capacidad de carga es
de: (> 3.0)")
Text_FSCC = ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Label_CC = ipw.Label(value="La capacidad de carga última del terreno es de:
(kPa)")
Text_CC = ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))

# Resultados del armado del muro

Label_var1 = ipw.Label(value="Se necesitan: ")
Label_var2 = ipw.Label(value="varillas, del número ")
Label_var3 = ipw.Label(value="por cada m de muro. ")
Label_var4 = ipw.Label(value="Con una longitud de ")
Label_var5 = ipw.Label(value="cm y separadas a cada ")
Label_var6 = ipw.Label(value="cm.")
# Punta
Label_armado_punta = ipw.Label(value="ARMADO DE LA PUNTA DEL MURO",
layout=Layout(display="flex", justify_content="center"))
Text_var_punta =
ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Text_num_punta = ipw.FloatText(value=0,
disabled=True,layout=Layout(width='75px'))
Text_lon_punta =
ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Text_sep_punta =
ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Text_Mu_punta = ipw.FloatText(value=0,disabled=True,description='Mu
(t·m)',layout=Layout(width='150px'))
Text_Mn_punta = ipw.FloatText(value=0,disabled=True,description='Mn
(t·m)',layout=Layout(width='150px'))
Text_Vu_punta = ipw.FloatText(value=0,disabled=True,description='Vu
(t)',layout=Layout(width='150px'))
Text_Vn_punta = ipw.FloatText(value=0,disabled=True,description='Vc
(t)',layout=Layout(width='150px'))
# Talón
Label_armado_talon = ipw.Label(value="ARMADO DEL TALÓN DEL MURO",
layout=Layout(display="flex", justify_content="center"))
Text_var_talon =
ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Text_num_talon = ipw.FloatText(value=0,
disabled=True,layout=Layout(width='75px'))
```

```
Text_lon_talon =
ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Text_sep_talon =
ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Text_Mu_talon = ipw.FloatText(value=0,disabled=True,description='Mu
(t·m)',layout=Layout(width='150px'))
Text_Mn_talon = ipw.FloatText(value=0,disabled=True,description='Mn
(t·m)',layout=Layout(width='150px'))
Text_Vu_talon = ipw.FloatText(value=0,disabled=True,description='Vu
(t)',layout=Layout(width='150px'))
Text_Vn_talon = ipw.FloatText(value=0,disabled=True,description='Vc
(t)',layout=Layout(width='150px'))
# Pantalla
Label_armado_pantalla = ipw.Label(value="ARMADO DE LA PANTALLA DEL MURO",
layout=Layout(display="flex", justify_content="center"))
Text_var_pantalla =
ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Text_num_pantalla = ipw.FloatText(value=0,
disabled=True,layout=Layout(width='75px'))
Text_lon_pantalla =
ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Text_sep_pantalla =
ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Text_Mu_pantalla = ipw.FloatText(value=0,disabled=True,description='Mu
(t·m)',layout=Layout(width='150px'))
Text_Mn_pantalla = ipw.FloatText(value=0,disabled=True,description='Mn
(t·m)',layout=Layout(width='150px'))
Text_Vu_pantalla = ipw.FloatText(value=0,disabled=True,description='Vu
(t)',layout=Layout(width='150px'))
Text_Vn_pantalla = ipw.FloatText(value=0,disabled=True,description='Vc
(t)',layout=Layout(width='150px'))
# Contracción
Label_cont_base = ipw.Label(value="ACERO POR CONTRACCIÓN Y TEMPERATURA EN LA
BASE", layout=Layout(display="flex", justify_content="center"))
Text_cant_base =
ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Text_s_base = ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Text_L_base = ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Label_cont_pantalla_h = ipw.Label(value="ACERO HORIZONTAL POR TEMPERATURA EN LA
PANTALLA", layout=Layout(display="flex", justify_content="center"))
Text_cant_pantalla_h =
ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Text_s_pantalla_h =
ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Text_L_pantalla_h =
ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Label_cont_pantalla_v = ipw.Label(value="ACERO VERTICAL POR TEMPERATURA EN LA
PANTALLA", layout=Layout(display="flex", justify_content="center"))
Text_cant_pantalla_v =
ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Text_s_pantalla_v =
ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))
Text_L_pantalla_v =
ipw.FloatText(value=0,disabled=True,layout=Layout(width='75px'))

# Contenedor Estabilidad
CR_Estabilidad = ipw.VBox([Label_Estabilidad, Label_FSV, Text_FSV, Label_FSD,
Text_FSD],
```

```
                  layout=Layout(border="5px dashed black", margin="10px",
width="370px"))

# Contenedor Capacidad de carga
CR_CapacidadC = ipw.VBox([Label_CapacidadC, Label_CC, Text_CC, Label_FSCC,
Text_FSCC],
                  layout=Layout(border="5px dashed black", margin="10px",
width="370px"))

# Contendor Resultados Armado
CR_armado_punta = ipw.VBox([Label_armado_punta, ipw.HBox([Label_var1,
Text_var_punta, Label_var2, Text_num_punta, Label_var3]),
                      ipw.HBox([Label_var4, Text_lon_punta, Label_var5,
Text_sep_punta, Label_var6]),
                      ipw.HBox([Text_Mu_punta, Text_Mn_punta]),
ipw.HBox([Text_Vu_punta, Text_Vn_punta])],
                      layout=Layout(border="5px dashed black", margin="10px",
width="500px"))
CR_armado_talon = ipw.VBox([Label_armado_talon, ipw.HBox([Label_var1,
Text_var_talon, Label_var2, Text_num_talon, Label_var3]),
                      ipw.HBox([Label_var4, Text_lon_talon, Label_var5,
Text_sep_talon, Label_var6]),
                      ipw.HBox([Text_Mu_talon, Text_Mn_talon]),
ipw.HBox([Text_Vu_talon, Text_Vn_talon])],
                      layout=Layout(border="5px dashed black", margin="10px",
width="500px"))
CR_armado_pantalla = ipw.VBox([Label_armado_pantalla, ipw.HBox([Label_var1,
Text_var_pantalla, Label_var2, Text_num_pantalla, Label_var3]),
                      ipw.HBox([Label_var4, Text_lon_pantalla, Label_var5,
Text_sep_pantalla, Label_var6]),
                      ipw.HBox([Text_Mu_pantalla, Text_Mn_pantalla]),
ipw.HBox([Text_Vu_pantalla, Text_Vn_pantalla])],
                      layout=Layout(border="5px dashed black", margin="10px",
width="500px"))
CR_contraccion = ipw.VBox([Label_cont_base, ipw.HBox([Label_var1,
Text_cant_base, Label_var2, Text_num_pantalla]),
                           ipw.HBox([Label_var4, Text_L_base, Label_var5,
Text_s_base, Label_var6]),
                           Label_cont_pantalla_h, ipw.HBox([Label_var1,
Text_cant_pantalla_h, Label_var2, Text_num_pantalla]),
                           ipw.HBox([Label_var4, Text_L_pantalla_h, Label_var5,
Text_s_pantalla_h, Label_var6]),
                           Label_cont_pantalla_v, ipw.HBox([Label_var1,
Text_cant_pantalla_v, Label_var2, Text_num_pantalla]),
                           ipw.HBox([Label_var4, Text_L_pantalla_v, Label_var5,
Text_s_pantalla_v, Label_var6])],
                           layout=Layout(border="5px dashed black",
margin="10px", width="500px"))
CR_armado = ipw.HBox([ipw.VBox([CR_armado_punta, CR_armado_talon,
CR_armado_pantalla]), CR_contraccion],
                     layout=Layout(visibility="hidden"))


out_muro = ipw.Output()
out_CC = ipw.Output()
out_armado = ipw.Output()
# BOTÓN DEL CÁLCULO
BotonCalculo_muro = ipw.Button(description = "Calcular", button_style="primary")
```

```
def BotonCalculo_muro_clicked(b):

  # Llama a la clase Muros para crear una instancia (iniciarla)
  Muro_1 = Muros(HTWALL.value, Hs, Text_ESuma.value, Text_RSuma.value,
γ_muro.value, coef_deslizam.value, tipo_de_sobrecarga.value,
                 Text_E2.value, Text_R2.value, Hs_p1.value, γms, c, β.value,
α.value, 0, φ, NAF_CC.value,
                 fc.value, fy.value, E.value, Φ.value, rec_libre.value,
var_propuesta_num.value,
                 γ_apoyo.value, c_apoyo.value, φ_apoyo.value)

  if tipo_muro.value == 1:
    FSV, FSD, FSCC, qn = Muro_1.predim_gravedad()

    out_armado.clear_output() # Lo pongo tambien aquí para que lo borre si se
generó antes

  else:
    FSV, FSD, FSCC, qn = Muro_1.predim_voladizo()
    R_punta, R_talon, R_pantalla = Muro_1.diseno_acero()
    Cont_base, Cont_pantalla_h, Cont_pantalla_v = Muro_1.acero_contraccion()

    Text_var_punta.value = R_punta[0]
    Text_num_punta.value = R_punta[1]
    Text_lon_punta.value = R_punta[2]
    Text_sep_punta.value = R_punta[3]
    Text_Mu_punta.value = R_punta[4]
    Text_Mn_punta.value = R_punta[5]
    Text_Vu_punta.value = R_punta[6]
    Text_Vn_punta.value = R_punta[7]
    Text_var_talon.value = R_talon[0]
    Text_num_talon.value = R_talon[1]
    Text_lon_talon.value = R_talon[2]
    Text_sep_talon.value = R_talon[3]
    Text_Mu_talon.value = R_talon[4]
    Text_Mn_talon.value = R_talon[5]
    Text_Vu_talon.value = R_talon[6]
    Text_Vn_talon.value = R_talon[7]
    Text_var_pantalla.value = R_pantalla[0]
    Text_num_pantalla.value = R_pantalla[1]
    Text_lon_pantalla.value = R_pantalla[2]
    Text_sep_pantalla.value = R_pantalla[3]
    Text_Mu_pantalla.value = R_pantalla[4]
    Text_Mn_pantalla.value = R_pantalla[5]
    Text_Vu_pantalla.value = R_pantalla[6]
    Text_Vn_pantalla.value = R_pantalla[7]

    Text_cant_base.value = Cont_base[0]
    Text_s_base.value = Cont_base[1]
    Text_L_base.value = Cont_base[2]
    Text_cant_pantalla_h.value = Cont_pantalla_h[0]
    Text_s_pantalla_h.value = Cont_pantalla_h[1]
    Text_L_pantalla_h.value = Cont_pantalla_h[2]
    Text_cant_pantalla_v.value = Cont_pantalla_v[0]
    Text_s_pantalla_v.value = Cont_pantalla_v[1]
    Text_L_pantalla_v.value = Cont_pantalla_v[2]

    out_armado.clear_output()
```

```
    with out_armado:
      Muro_1.dibujar_acero()

  Text_FSV.value = FSV
  Text_FSD.value = FSD
  Text_FSCC.value = FSCC
  Text_CC.value = qn

  out_muro.clear_output() # Para borrar los diagramas anteriores
  with out_muro:
    Muro_1.graficar_muro()

  out_CC.clear_output() # Para borrar los diagramas anteriores
  with out_CC:
    Muro_1.graficar_CC()

BotonCalculo_muro.on_click(BotonCalculo_muro_clicked)


def Ocultar_armado(value):
  if value == 1:
    Contenedor_armado.layout.visibility = 'hidden'
    CR_armado.layout.visibility = 'hidden'
  else:
    Contenedor_armado.layout.visibility = 'visible'
    CR_armado.layout.visibility = 'visible'

ipw.interactive(Ocultar_armado, value=tipo_muro)


BotonGenerar_reporte = ipw.Button(description = "Generar reporte",
button_style="danger")
def BotonGenerar_reporte_clicked(b):

    #input file
  f_in = open("/content/drive/MyDrive/Tesis_python/Reporte_in.md", "rt")
  data = f_in.read()

  #output file to write the result to
  f_out =
open("/content/drive/MyDrive/Tesis_python/Reporte_descargar/Reporte_out.md",
"wt")

  data = data.replace("P01", str(HTWALL.value))
  data = data.replace("P02", str(Text_E1.value))
  data = data.replace("P03", str(Text_R1.value))
  data = data.replace("P04", str(Text_E2.value))
  data = data.replace("P05", str(Text_R2.value))
  data = data.replace("P07", str(Text_E3.value))
  data = data.replace("P08", str(Text_R3.value))
  data = data.replace("P09", str(Text_ESismo.value))
  data = data.replace("P10", str(Text_RSismo.value))
  data = data.replace("P11", str(Text_ESuma.value))
  data = data.replace("P12", str(Text_RSuma.value))

  if tipo_muro.value == 1:
    tipo = "de gravedad"
  elif tipo_muro.value == 2:
```

```
    tipo = "en voladizo"
  data = data.replace("P13", tipo)
  data = data.replace("P14", str(Text_FSV.value))
  data = data.replace("P15", str(Text_FSD.value))
  data = data.replace("P16", str(round(q_max, 2)))
  data = data.replace("P17", str(Text_CC.value))
  data = data.replace("P18", str(Text_FSCC.value))
  data = data.replace("P19", str(round(MV, 2)))
  data = data.replace("P20", str(round(MR, 2)))

  Info_armado = f"### ARMADO DEL MURO EN VOLADIZO. \n\n \
  ##### ACERO A TENSIÓN \n\n \
  Punta del muro: {Text_var_punta.value} varillas # {Text_num_punta.value} @
{Text_sep_punta.value} cm, con longitud de {Text_lon_punta.value} cm. \n\n \
  Talón del muro: {Text_var_talon.value} varillas # {Text_num_talon.value} @
{Text_sep_talon.value} cm, con longitud de {Text_lon_talon.value} cm. \n\n \
  Pantalla del muro: {Text_var_pantalla.value} varillas #
{Text_num_pantalla.value} @ {Text_sep_pantalla.value} cm, con longitud de
{Text_lon_pantalla.value} cm. \n\n \
  ##### ACERO POR TEMPERATURA \n\n \
  Base del muro: {Text_cant_base.value} varillas # {Text_num_punta.value} @
{Text_s_base.value} cm, con longitud de {Text_L_base.value} cm. \n\n \
  Base del muro: {Text_cant_pantalla_h.value} varillas # {Text_num_punta.value} @
{Text_s_pantalla_h.value} cm, con longitud de {Text_L_pantalla_h.value} cm. \n\n
\
  Base del muro: {Text_cant_pantalla_v.value} varillas # {Text_num_punta.value} @
{Text_s_pantalla_v.value} cm, con longitud de {Text_L_pantalla_v.value} cm.
\n\n"

  if tipo_muro.value == 2: # muro en voladizo
    data = data.replace("P21", Info_armado)
    data = data.replace("P25", "![](Diagrama_acero.png)")

    dimensiones_voladizo = f"- Base: {round(base_muro, 2)} m. \n\n- Altura:
{HTWALL.value} m. \n\n- Corona: {round(corona, 2)} m. \n\n\
- Profundidad de desplante: {round(altura_talon, 2)} m. \n\n- Punta:
{round(base_punta, 2)} m."
    data = data.replace("P22", dimensiones_voladizo)
  elif tipo_muro.value == 1:
    data = data.replace("P21", "")
    data = data.replace("P25", "")

    dimensiones_gravedad = f"- Base: {round(base_muro, 2)} m. \n\n- Altura:
{HTWALL.value} m. \n\n- Corona: {round(corona, 2)} m. \n\n\
- Profundidad de desplante: {Hs_p1.value} m."
    data = data.replace("P22", dimensiones_gravedad)

  data = data.replace("P23", "![](Diagrama_muro.png)")
  data = data.replace("P24", "![](Diagramas_presiones.png)")

  f_out.write(data) # Sobre escribe data en el archivo md de salida

  #close input and output files
  f_in.close()
  f_out.close()

  !zip -r reporte.zip /content/drive/MyDrive/Tesis_python/Reporte_descargar #
Comprime los archivos necesarios para visualizar el reporte
```

```
  files.download("reporte.zip") # Descarga el archivo comprimido

BotonGenerar_reporte.on_click(BotonGenerar_reporte_clicked)



########################## MENÚ ##########################

def Menu():
  tab_bar = widgets.TabBar(["DATOS EMPUJES", "RESULTADOS", "DIAGRAMAS DE
PRESIÓN", "DATOS DEL MURO", "ESTABILIDAD", "CAPACIDAD DE CARGA", "ARMADO DEL
MURO"])

  with tab_bar.output_to(1):
    display(Contenedor_resultados)

  with tab_bar.output_to(2):
    display(out)

  with tab_bar.output_to(3):
    display(ipw.HBox([Contenedor_muro, Contenedor_apoyo, Contenedor_armado]),
BotonCalculo_muro)

  with tab_bar.output_to(4):
    display(ipw.HBox([CR_Estabilidad, out_muro]))

  with tab_bar.output_to(5):
    display(ipw.HBox([CR_CapacidadC, out_CC]))

  with tab_bar.output_to(6):
    display(ipw.HBox([CR_armado, out_armado]))

  with tab_bar.output_to(0):
    display(Contenedor_datos, ipw.HBox([BotonCalculo, BotonGenerar_reporte]))

Menu()
```

# 9.9 REPORTE.

Utilizando la herramienta de cálculo. Para un muro de contención **en voladizo** con 4.6 m de altura, se obtuvieron los siguientes resultados:

## PRESIONES LATERALES.

**PRESIÓN ACTIVA POR EL MÉTODO DE RANKINE.**

La presión activa de Rankine ocasiona un empuje horizontal de 39.96 kN. Cuya resultante se ubica a 1.22 m a partir de la base del muro.

**PRESIÓN PASIVA POR EL MÉTODO DE RANKINE.**

La presión pasiva de Rankine ocasiona un empuje horizontal de 3.44 kN. Cuya resultante se ubica a 0.2 m a partir de la base del muro.

**PRESIÓN DEBIDO A SOBRECARGA.**

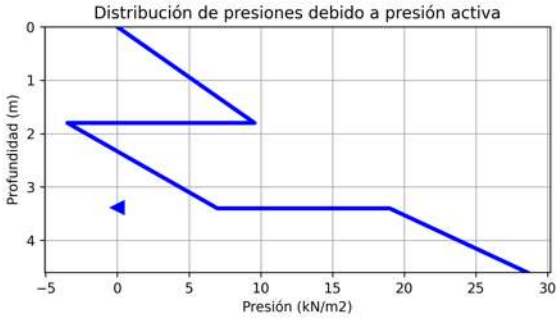
La sobrecarga ocasiona un empuje horizontal de 22.17 kN. Cuya resultante se ubica a 2.21 m a partir de la base del muro.

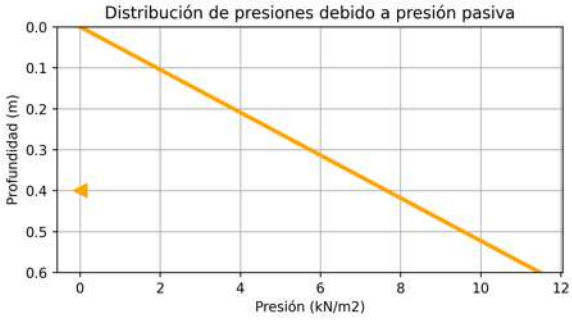
**PRESIÓN DEBIDO A SISMO POR EL MÉTODO DE MONONOBE-OKABE.**

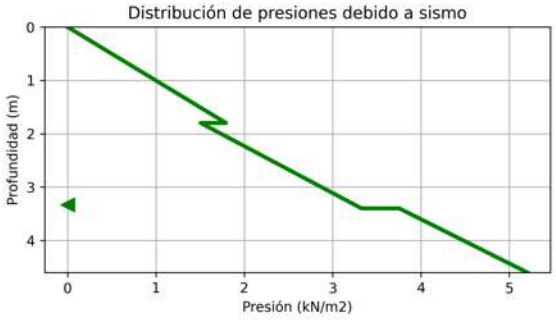
El incremento de la presión activa debido a sismo ocasiona un empuje horizontal de 10.86 kN. Cuya resultante se ubica a 2.76 m a partir de la base del muro.

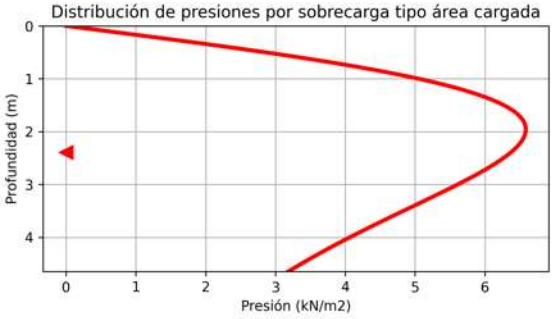
**PRESIÓN TOTAL.**

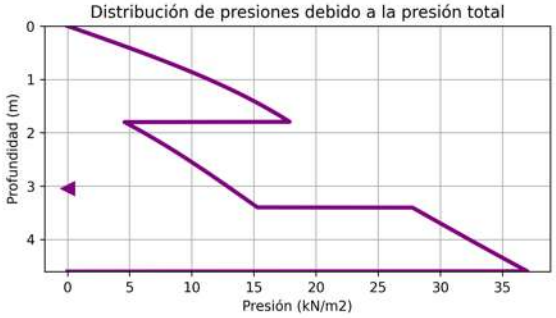
La sumatoria de todos los empujes actuantes es de 72.8 kN. Cuya resultante se ubica a 1.56 m a partir de la base del muro.



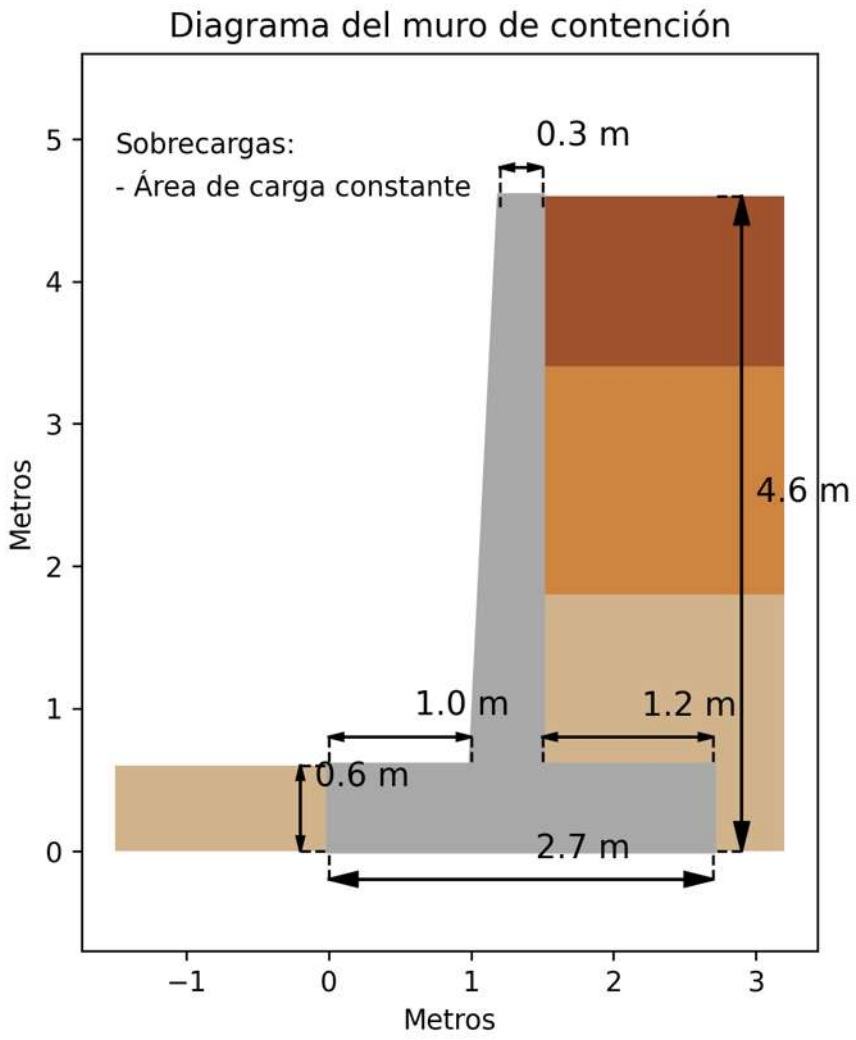




# DIMENSIONAMIENTO DEL MURO.

### DIMENSIONES PROPUESTAS POR LA HERRAMIENTA DE CÁLCULO.

- Base: 2.7 m.
- Altura: 4.6 m.
- Corona: 0.3 m.
- Profundidad de desplante: 0.6 m.
- Punta: 1.0 m.




**ESTABILIDAD DEL MURO.**

El momento de que provoca el volcamiento del muro es de 113.57 kN-m, mientras que el momento que evita el volcamiento del muro es de 288.83 kN-m.

El factor de seguridad contra volteo es de 2.54, este debe ser mayor o igual que 1.5 tratándose de un material granular o mayor o igual a 2.0 si se trata de un material cohesivo.

El factor de seguridad contra deslizamiento es de 1.53, este debe ser mayor o igual que 1.5.

**CAPACIDAD DE CARGA DEL TERRENO.**

La presión máxima que el muro ejerce sobre el terreno es de 101.06 kPa, mientras que la capacidad de carga última del suelo es de 440.77 kPa. Por lo tanto, el factor de seguridad por capacidad de carga es de 4.36.

# ARMADO DEL MURO EN VOLADIZO.

**ACERO A TENSIÓN**

Punta del muro: 5.0 varillas # 6.0 @ 17.0 cm, con longitud de 130.0 cm.

Talón del muro: 5.0 varillas # 6.0 @ 17.0 cm, con longitud de 170.0 cm.

Pantalla del muro: 4.0 varillas # 6.0 @ 21.0 cm, con longitud de 430.0 cm.

**ACERO POR TEMPERATURA**

Base del muro: 18.0 varillas # 6.0 @ 31.0 cm, con longitud de 85.0 cm.

Base del muro: 22.0 varillas # 6.0 @ 38.0 cm, con longitud de 85.0 cm.

Base del muro: 6.0 varillas # 6.0 @ 17.0 cm, con longitud de 385.0 cm.