DESARROLLO DE UN SOFTWARE PARA EL DISEÑO DE ZAPATAS RECTANGULARES POR FLEXIÓN BIAXIAL

César Rodolfo Bocanegra Malca

Piura, 17 de Junio de 2005

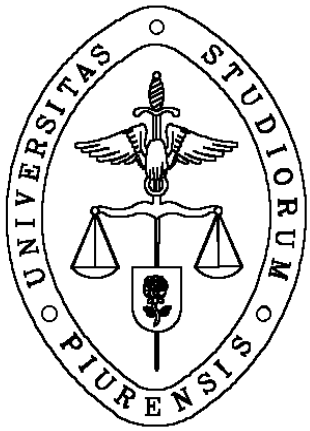
FACULTAD DE INGENIERÍA

Departamento de Ingeniería Civil

Junio 2005

UNIVERSIDAD DE PIURA

Facultad de Ingeniería



"DESARROLLO DE UN SOFTWARE PARA EL DISEÑO DE ZAPATAS RECTANGULARES POR FLEXIÓN BIAXIAL"

Tesis para optar el Título de:
Ingeniero Civil

CÉSAR RODOLFO BOCANEGRA MALCA

Asesor: Mgtr. Arturo Martínez Ramírez

Piura, Agosto 2005

Dedicatoria

Dedico esta tesis a Dios, a mis padres: César y Nancy por su amor y apoyo constante que siempre me han brindado.

PROLOGO

La cimentación es parte vital de cualquier edificación, debido a que es la parte estructural de una edificación o de cualquier otra obra civil, que tiene la función principal de transmitir las cargas al terreno, el cual es el único elemento que no podemos elegir, por lo que es necesario obtener un diseño adecuado que se acerque a la realidad.

Actualmente en el diseño de zapatas rectangulares sometidas a flexión biaxial, se utilizan algunos criterios de diseño que no reflejan el comportamiento de una zapata. Este desconocimiento del comportamiento de la zapata rectangular, ha dado lugar en algunos casos que se le añadan a la zapata vigas de cimentación, para que así esta absorba los momentos, los cuales son en muchos casos innecesarias.

Es responsabilidad de los ingenieros estudiar este caso, realizando una investigación adecuada con el fin de resolver dicho problema.

Motivado por este problema de ingeniería, presento la tesis titulada **"Desarrollo de un software para el diseño de zapatas rectangulares por flexión biaxial"**, con el desarrollo de esta tesis se busca desarrollar un programa de cómputo que permita resolver los casos de flexión biaxial en zapatas, incluso la redistribución de esfuerzos en el suelo, tanto para zapatas centradas como excéntricas. Asimismo, se efectuará el diseño estructural de la zapata.

Finalmente quiero expresar mi más sincero agradecimiento a mi asesor Mgtr. Ing. Arturo Martínez Ramírez, por su apoyo continuo en la elaboración de la presente tesis.

RESUMEN

Éste trabajo tiene como objetivo desarrollar un software que permita diseñar una zapata rectangular sometida a flexión biaxial, para ello se utilizará como lenguaje de programación el Visual Basic 6.0.

El desarrollo del trabajo se divide en tres grandes partes. La primera parte comprende la solución analítica de los diversos casos de distribución de esfuerzos para flexión biaxial. La segunda parte comprende el desarrollo del software, manual de usuario y principales rutinas. La tercera parte comprende la validación del software desarrollado a través de su comparación con ejemplos de libros de texto.

De acuerdo a los resultados obtenidos en la validación del software, se concluye que el software permite diseñar de forma confiable zapatas rectangulares sometidas a flexión biaxial.

INDICE

Dedicatoria
Prólogo
Resumen
Introducción

Capítulo I: Introducción al Problema.

1.1. Formas de Analizar una Zapata, y su diseño. 3
1.2. Búsqueda de Bibliografía y temas relacionados. 12

Capítulo II: Desarrollo al problema.

2.1. Casos de distribución de esfuerzos en zapatas sometidas a flexión biaxial. 13
2.2. Solución analítica propuesta por R. Irles y F. Irles. 17
2.3. Solución analítica al problema de R. Irles y F. Irles. 23

Capítulo III: Desarrollo del software Diza 1.0.

3.1. Visual Basic 6.0: Introducción. 25
3.2. Diagramas de Flujo. 28

Capítulo IV: Validación del software. 32

Capítulo V: Conclusiones y recomendaciones. 48

5.1. Conclusiones.
5.2. Recomendaciones.

Bibliografía. 50

Anexo A.
Manual de Uso. 51

Anexo B.
Variables, Definición y entorno de Visual Basic. 56

Anexo C.
Algoritmos y códigos del software. 66

Diccionario de términos.

INTRODUCCIÓN

*R. Irles y F. Irles***,** en un artículo publicado en el Journal of Geotechnical Engineering, Vol 120, N° 2, ASCE. Feb, 1994, basándose en una distribución plana de esfuerzos y las ecuaciones del equilibrio estático presentan una solución analítica de los esfuerzos actuantes en la zapata rectangular en flexión biaxial, diferente al resto de autores, encuentran las ecuaciones que se adecuan al comportamiento de la zapata según su excentricidad, sin embargo no terminan de solucionar el problema debido a que no encuentran la solución de una ecuación, esta tesis resuelve dicho problema.

A nivel internacional existen muchos el software que tratan este tema, destacando STAAD – Pro, el cual contiene un software aplicativo para el diseño de zapatas aisladas, el autor de esta tesis no tiene conocimiento de un software similar a nivel local.

El desarrollo de la presente tesis se ha dividido en cinco capítulos más dos anexos. El primer capítulo es una descripción de los métodos utilizados por diversos autores para resolver una zapata rectangular por flexión biaxial. El segundo capítulo muestra una descripción de los diversos casos de distribución de esfuerzos para flexión biaxial, así como su solución analítica.

El tercer capítulo hace referencia al lenguaje de programación utilizado, Visual Basic 6.0; además que muestra los diagramas de flujo que realiza el software desarrollado. El cuarto capítulo realizará una comparación de los resultados de diversos autores con el software desarrollado con el fin de demostrar su validez. El último capítulo muestra las conclusiones y recomendaciones del presente trabajo. El anexo 1, es una manual de usuario del programa por último el anexo 2, muestra las principales variables y entorno de Visual Basic.

CAPITULO 1

INTRODUCCIÓN AL PROBLEMA

1.1. FORMAS DE ANALIZAR UNA ZAPATA, Y SU DISEÑO.

Actualmente en el diseño de zapatas rectangulares se utilizan algunos criterios de diseño que no reflejan el comportamiento en flexión biaxial. Se suele asumir lo siguiente:

- Diseñar la zapata uniaxialmente para la peor condición, es decir en el lado de la zapata con mayores esfuerzos, y luego correr los momentos y aceros hallados en ambas direcciones.

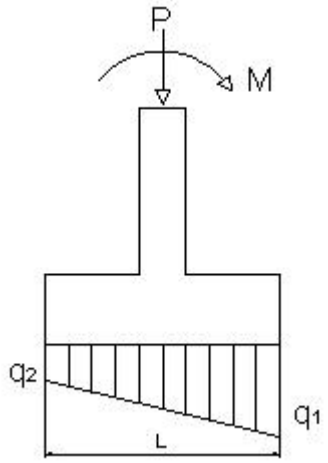
- Siempre con el criterio de diseño uniaxial, si la excentricidad sale del núcleo central, los esfuerzos se redistribuyen. Esto mismo se repite en la otra dirección de la zapata.

Este tipo de idealización tiene cierta validez si no se tienen momentos importantes.
Este desconocimiento del comportamiento de la zapata rectangular, ha dado lugar en algunos casos que se le añadan a la zapata vigas de cimentación, para que así esta absorba los momentos, los cuales son en muchos casos innecesarias. Se explicará a continuación detalladamente las formas de analizar una zapata rectangular sometidas a carga vertical y momento biaxial y además su diseño.

En primer lugar se realiza un predimensionamiento, con la finalidad de obtener las dimensiones de la zapata y comprobar que con estas dimensiones los esfuerzos actuantes son menores que la capacidad portante del suelo. Se asume distribución lineal de esfuerzos.

A continuación se amplifican las cargas según la Norma Técnica E060 – Concreto Armado, se realizan estos cálculos “sin carga de sismo” y “con carga de sismo”, y se obtienen una carga y un momento de diseño. Luego se obtiene la excentricidad, que sería resultado de la división entre el momento y la carga. Obtenido este resultado se revisa si la excentricidad es menor que la sexta parte de la longitud mayor de la zapata. De ser así se utilizarán las siguientes expresiones de mecánica de materiales:

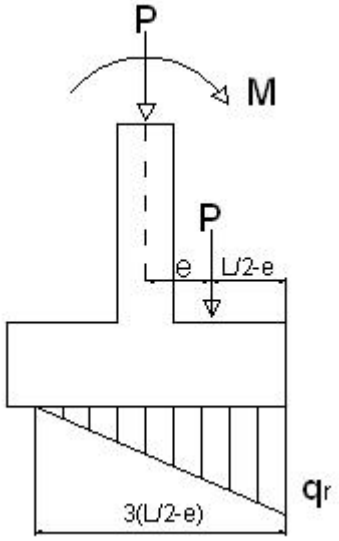
Solicitaciones uniaxiales



$$q1 = \frac{P}{b \times l} + \frac{6 \times M}{b \times l^2} \qquad q2 = \frac{P}{b \times l} - \frac{6 \times M}{b \times l^2}$$

donde:
l = Longitud en la dirección del momento.
b = La otra dimensión de la zapata.

Si la excentricidad “e” es mayor que la sexta parte de la longitud mayor de la zapata se realiza una redistribución de esfuerzos mediante la siguiente expresión:



$$qr = \frac{2 \times P}{3 \times \left(\frac{l}{2} - e \right) \times b}$$

Una vez obtenidos los esfuerzos de diseño en la zapata se procede a obtener los momentos actuantes Mu con carga de sismo y sin carga de sismo, y se elige el mayor de ambos. Con este momento se procede a obtener el refuerzo requerido así como su respectivo espaciamiento.

A continuación se realiza el diseño por corte y punzonamiento, usando las fórmulas de la Norma Técnica E060 – Concreto Armado.

Corte

$$Vu \le \phi.Vn = 0.85x0.53x\sqrt{f'c}xbxd$$

Punzonamiento

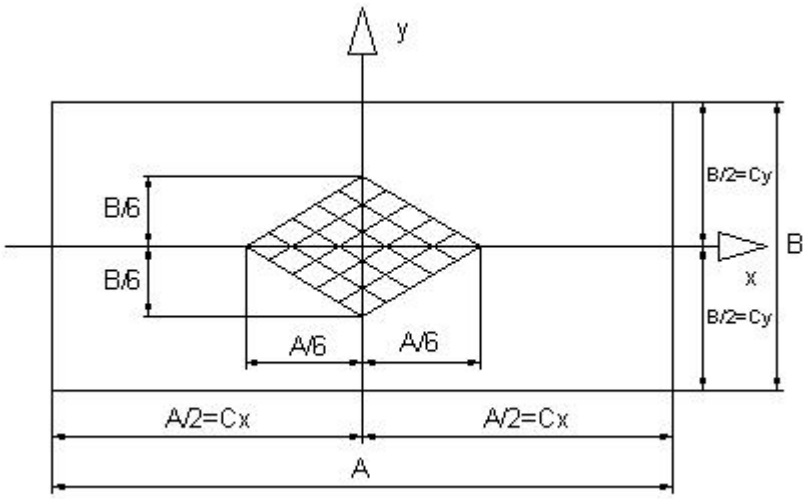
$$Vu \le \phi.Vn = \phi.Vc$$

$$Vc = 0.27x\left(2 + \frac{4}{\beta.c}\right)x\sqrt{f'c}xbxd \le 1.1x\sqrt{f'c}xbxd$$

βc=lado largo de la columna/lado corto de la columna.

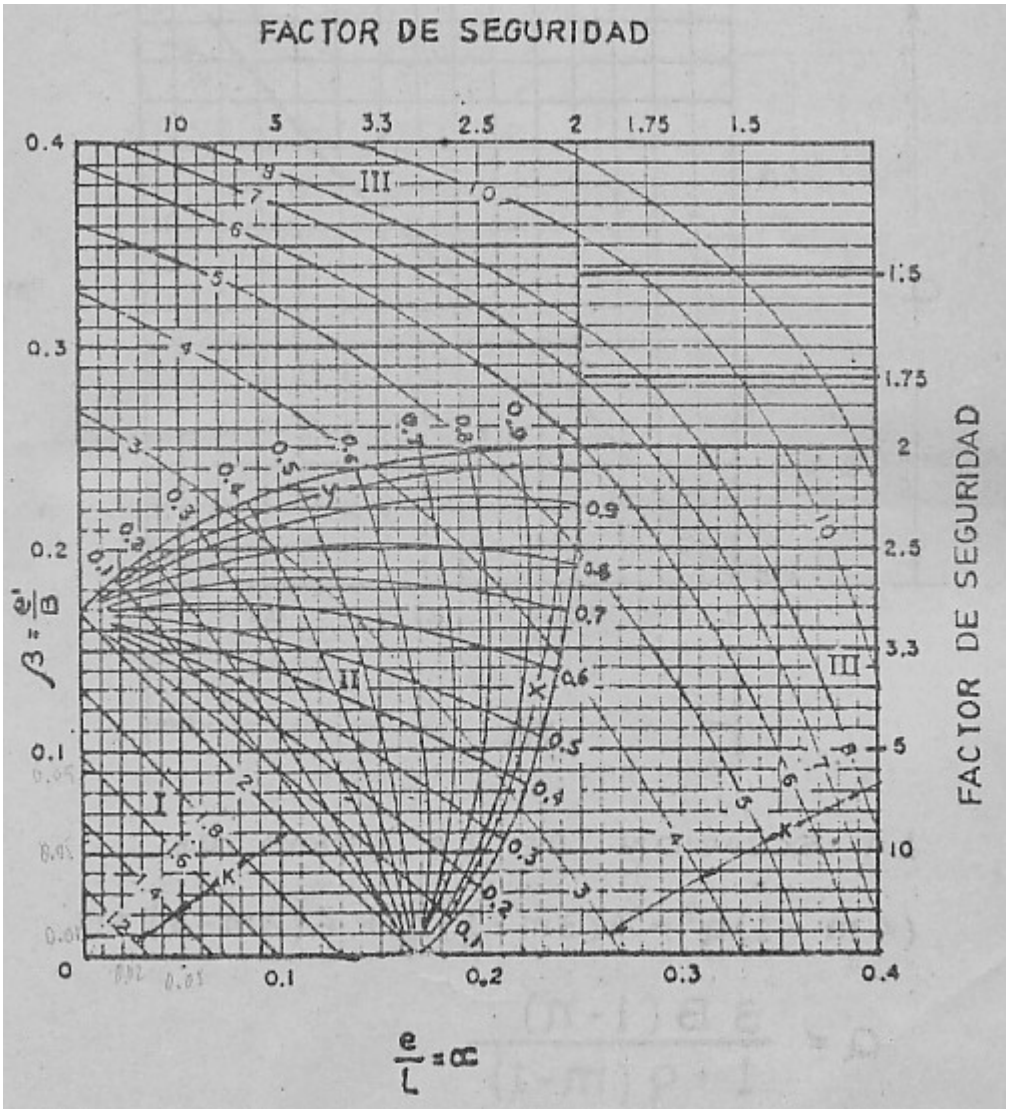
Existen otros autores como el Ing. ***Juan Ortega García*** y ***Julio Rivera Feijoo*** que utilizan tablas que relacionan las excentricidades en cada dirección con la evaluación de un factor de seguridad al volteo. Cuando ese factor de seguridad es menor que 2.5 se deben cambiar las dimensiones de la zapata. A continuación el método de ***Juan Ortega García***:

Análisis de Presión en la Base



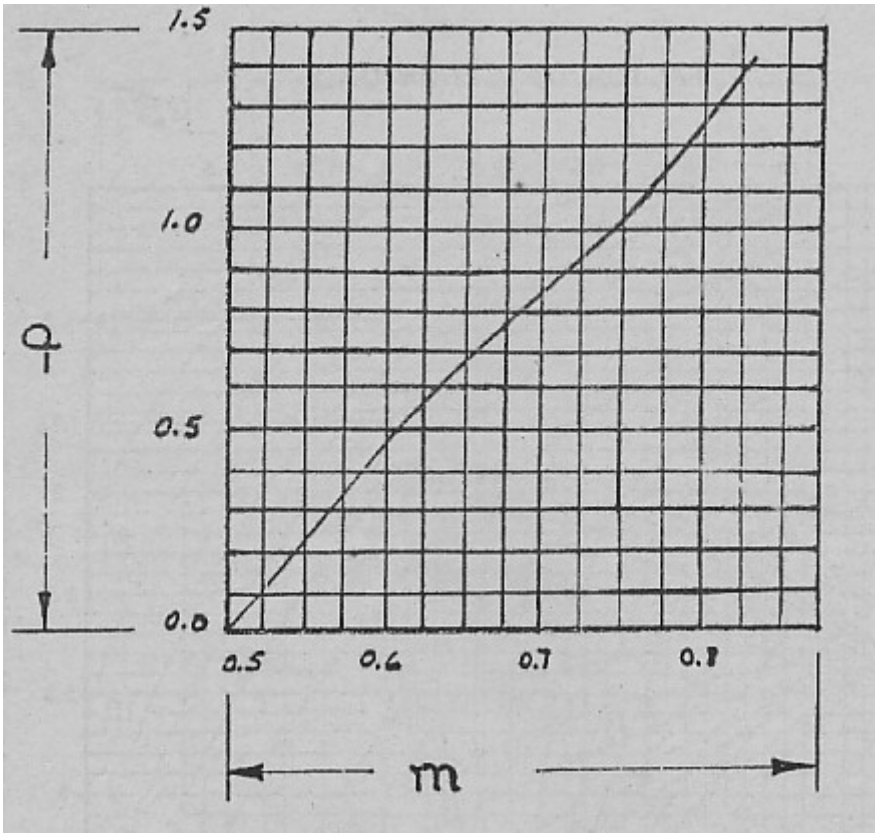
A continuación se presentan las tablas que se utiliza en el presente método:

Tabla 1.1



Abaco para determinar parámetros K, FS, X, Y

Tabla 1.2



Abaco para determinar el parámetro q

Caso 0:

Carga dentro del núcleo central:

$$q\max = \frac{P}{A \times B} + \frac{Mx \times Cy}{Ix} + \frac{My \times Cx}{Iy}$$

$$q\min = \frac{P}{A \times B} - \frac{Mx \times Cy}{Ix} - \frac{My \times Cx}{Iy}$$

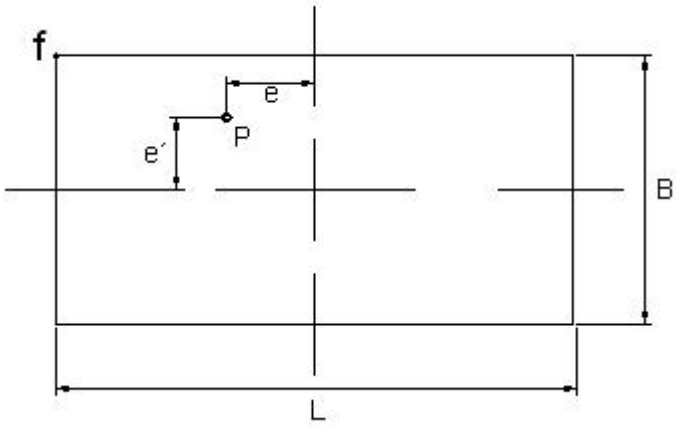
$$q\max = \frac{P}{A \times B} + \frac{6 \times P \times ex}{A^2 \times B} + \frac{6 \times P \times ey}{B^2 \times A}$$

$$q\min = \frac{P}{A \times B} - \frac{6 \times P \times ex}{A^2 \times B} - \frac{6 \times P \times ey}{B^2 \times A}$$

Cuando el punto de aplicación de la carga "P" está dentro del núcleo la presión se produce en toda la base, sino será presión parcial (hay 3 casos).

Caso I

Presión Total en la Base



Datos:
M, M`, P, σt

Solución:

$$e = \frac{M}{P} \quad e' = \frac{M'}{P}$$

$$\frac{e}{L} = \alpha \qquad \frac{e`}{B} = \beta$$ ……Con estos datos se entra a la tabla 1.1, se obtienen K y FS.

De tabla (a): K, F.S.

Se obtiene:

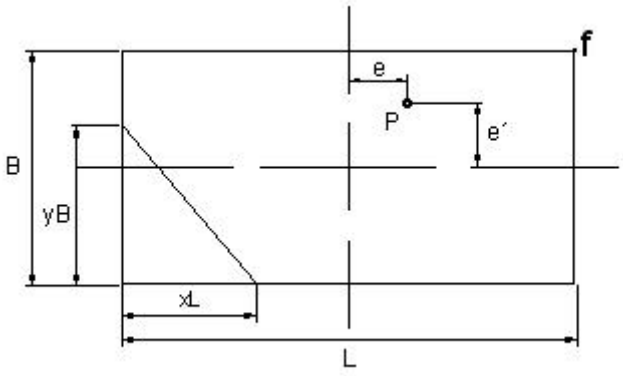
$$f = k \times \frac{P}{A}$$

Condición: Verificación de esfuerzos del terreno

$f \leq \sigma t$

Caso II

Presión Parcial: Zona no Comprimida Triangular



Datos:

M, M`, P, σt

Solución:

$e = \frac{M}{P}$ $\quad$ $e' = \frac{M'}{P}$

$\frac{e}{L} = \alpha$ $\quad$ $\frac{e`}{B} = \beta$ ……Ver Tabla 1.1

De tabla 1.1: K, F.S., X, Y

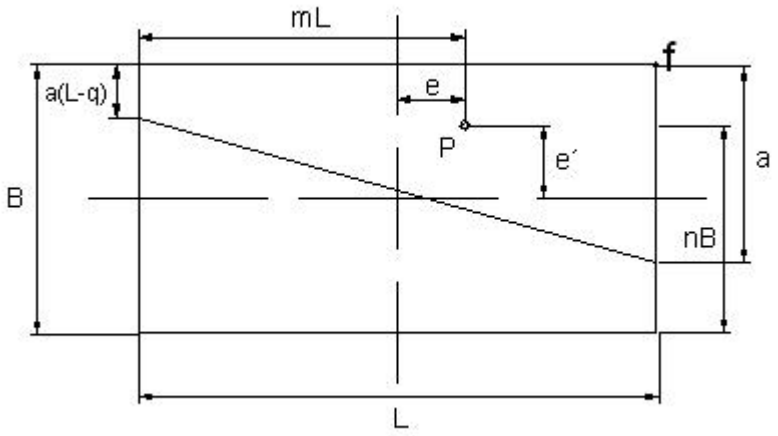
Se obtiene:

$f = k \times \frac{P}{A} \leq \sigma t$

XL, YB

Caso III

Presión Parcial Trapezoidal en la Base



Datos:

M, M`, P, σt

Solución:

$$e = \frac{M}{P} \qquad e' = \frac{M'}{P}$$

$$\frac{e}{L} = \alpha \qquad \frac{e`}{B} = \beta \quad \ldots\ldots \text{Ver Tabla 1.1}$$

De tabla 1.1: K, F.S.

Se obtiene :

$$f = k \times \frac{P}{A} \leq \sigma t$$

$$nB = \frac{B}{2} + e` \qquad mL = \frac{L}{2} + e$$

Se obtiene:

n, m

Con m .....tabla 1.2 ..... se obtiene q

Dimensiones de Zona en Compresión:

$$a = \frac{3 \times B \times (1-n)}{1 + q \times (m-1)}$$

a(l-q)

f´ = f(l-q)

Donde:

A, B, L	Dimensiones de la zapata.
f	Esfuerzo actuante.
e, e´	Excentricidades en las dos direcciones.
XL, YB	Dimensiones de los lados del área producida por los esfuerzos.
P	Carga vertical.
M, M´	Momentos en ambas direcciones de la zapata.
FS, K, X, Y, q, m	Parámetros obtenidos en los abacos 1.1 y 1.2.
σt	Capacidad portante del suelo.

A continuación el método de ***Julio Rivera Feijoo***:
Al igual que el método de Juan Ortega García, el método consiste en usar formulas para los diferentes casos en flexión biaxial y una tabla.

Fórmulas:

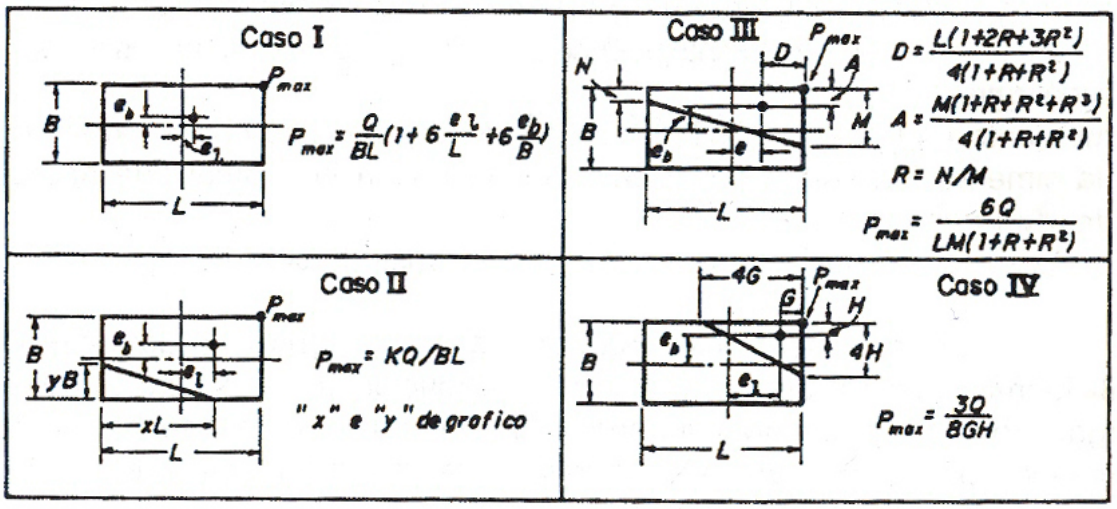
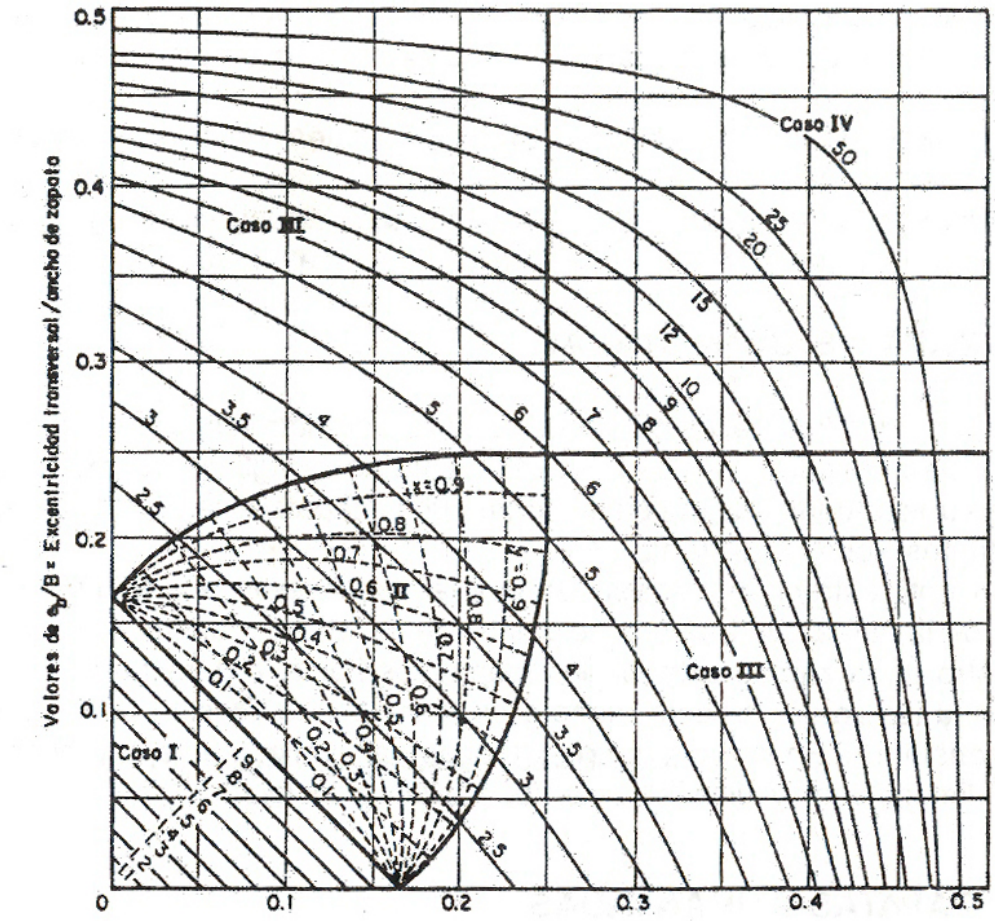


FIG. 4 ZAPATAS RECTANGULARES CON DOBLE EXCENTRICIDAD



Valores de e_l/L= Excentricidad longitudinal / Longitud de Zapata
Las líneas llenas dan valores de K
Presión máximo p_{max} = K x Q/BL
Q = Carga concentrada sobre la zapata

1.2. Búsqueda de Bibliografía y Temas Relacionados.

En 1994, los autores R. Iries y F. Iries en el paper "Explicit Stresses Under Rectangular Footings", publicado en el Journal of Geotechnical Engineering Vol 120, N°2 ASCE; desarrollaron de manera explícita los esfuerzos que se desarrollan bajo una zapata para las distintas solicitaciones de flexión biaxial.

Este tema es desarrollado por el Ing. Juan Ortega y Julio Rivera Feijoo, el primero desarrolla diferentes ejercicios para las diferentes solicitaciones de flexión biaxial, para lo que posteriormente se realizará la validación del software.

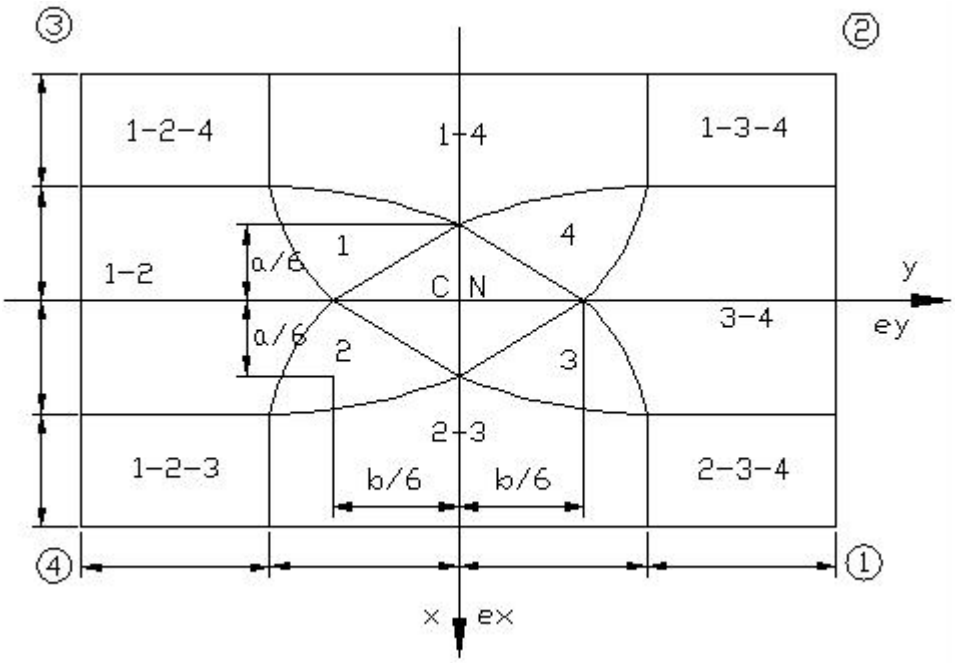
En la comprobación del cálculo estructural se utilizaron dos autores, Delgado Vargas M y el ejercicio desarrollado en clase de Concreto armado II por el Ing Arturo Martínez.

CAPITULO 2

DESARROLLO DEL PROBLEMA

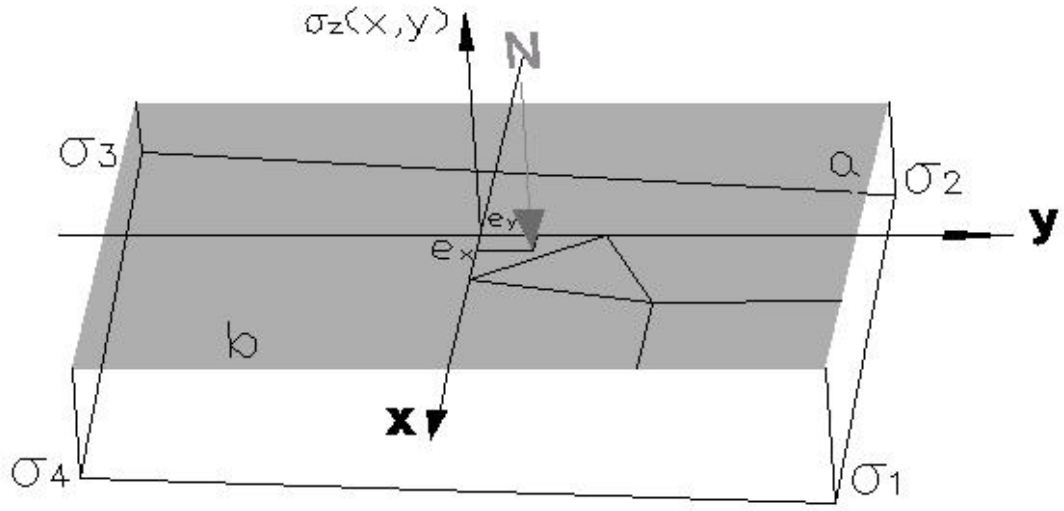
2.1. CASOS DE DISTRIBUCIÓN DE ESFUERZOS EN ZAPATAS SOMETIDAS A FLEXIÓN BIAXIAL.

Las cargas actuantes producen esfuerzos en la zapata, los cuales se pueden agrupar según su excentricidad de la siguiente forma:



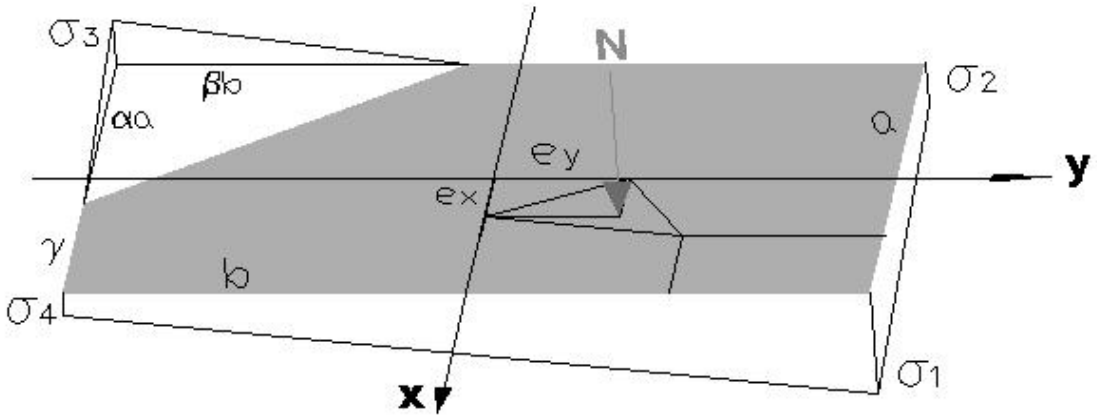
Esfuerzos en Núcleo Central (CN)

Los esfuerzos en el área CN se comportan de la siguiente forma:



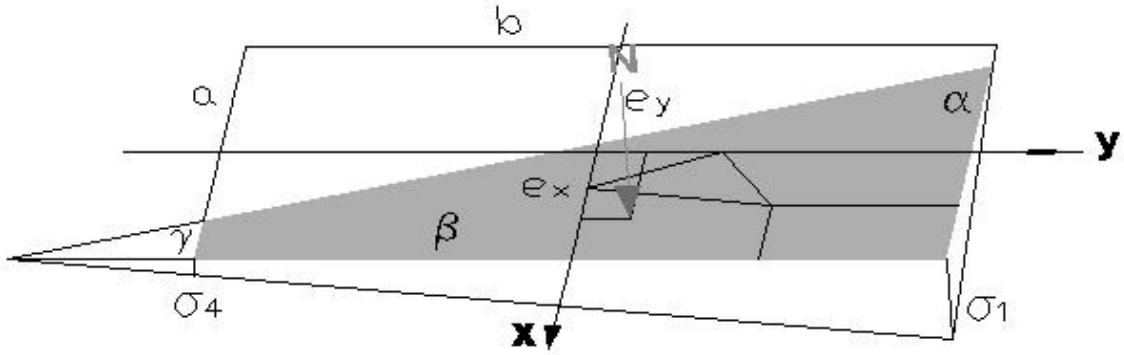
Esfuerzos en 3

Los esfuerzos en el área 3 se comportan de la siguiente forma:



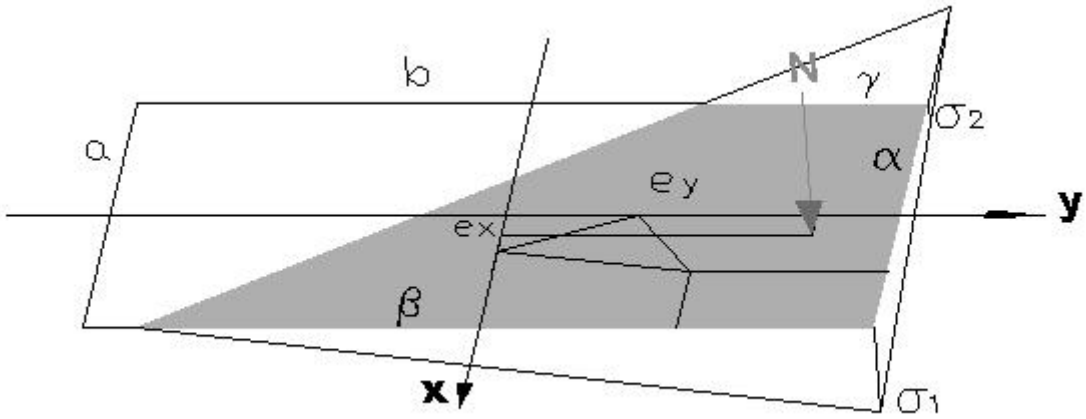
Esfuerzos en 2-3

Los esfuerzos en el área 2-3 se comportan de la siguiente forma:



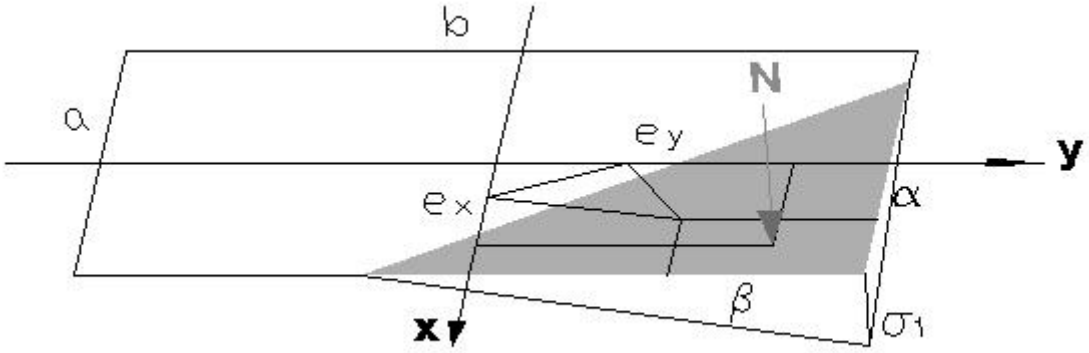
Esfuerzos en 3-4

Los esfuerzos en el área 3-4 se comportan de la siguiente forma:



Esfuerzos en 2-3-4

Los esfuerzos en el área 2-3-4 se comportan de la siguiente forma:



Donde:

a, b	Dimensiones de la zapata.
σ1, σ2, σ3, σ4	Esfuerzos en las zapatas.
ex, ey	Excentricidades en las dimensiones x e y respectivamente.
α, β, γ	Dimensiones de los lados del área producida por los esfuerzos.
N	Carga vertical.

2.2. SOLUCIÓN ANALÍTICA PROPUESTA POR R. IRLES Y F. IRLES.

El cálculo de esfuerzos bajo zapatas rectangulares con flexión biaxial se lleva a cabo resolviendo un sistema de 3 ecuaciones que prepara la carga y equilibra los esfuerzos del suelo. Cuando se asume una distribución de esfuerzos plana, el área esta toda a compresión, el sistema mencionado es lineal y su solución es inmediata. Con el aumento de las excentricidades, una esquina puede separarse del suelo, el sistema empieza no lineal y *aumenta las dificultades en su resolución*. A continuación se presenta una alternativa analítica, que llevará a obtener las soluciones específicas para esos esfuerzos y la ubicación del eje neutro.

Introducción

La derivación de esfuerzos debajo de zapatas rectangulares sujeto a flexión biaxial se calcula del siguiente sistema de ecuaciones:

$$\iint_{\Omega} \sigma z(x,y)\,dxdy = N \quad \text{..... (2.1a)}$$

$$\iint_{\Omega} x\,\sigma z(x,y)\,dxdy = Nex \quad \text{..... (2.1b)}$$

$$\iint_{\Omega} y\,\sigma z(x,y)\,dxdy = Ney \quad \text{..... (2.1c)}$$

Cuando se asume una distribución de esfuerzos plana, y el área esta completamente en compresión (resultante en el núcleo central), el sistema (1) es linear y es manejada inmediatamente a una fórmula muy conocida.

$$\sigma z(x,y) = \frac{N}{A} + \frac{xNex}{Iy} + \frac{yNey}{Ix} \quad \text{..... (2.2)}$$

Donde A = ab; Ix = $ab^3/12$; y Iy = $ba^3/12$

La ecuación 2.2 no se aplica para grandes excentricidades, debido a que una parte de la zapata rectangular no está apoyada totalmente en el suelo. La resolución general del sistema no lineal, puede ser resuelto mediante las siguientes formas:

1. Por medios de iteración que conduce a obtener una posición de eje neutro, indicado por Peck et al. (1974) y otros.

2. Por medios de iteración para valores discretos de ciertos parámetros geométricos, habilitando el dibujo de un ábaco para el cálculo directo de la ubicación del eje neutro y esfuerzos máximos.

Se presenta una solución diferente, eso llevará a una solución específica para todos los posibles casos, exceptuando cuando no está apoyada totalmente en el suelo. Sin embargo, incluso en este caso, se reduce el sistema de ecuaciones a una simple ecuación.

Derivación

Debido al hecho de que el problema es simétrico, será suficiente considerar un solo cuadrante.

2-3-4

$$N = \frac{\alpha\beta\sigma 1}{6} \quad \text{.....} \quad (2.3a)$$

$$N\left(\frac{b}{2} - ey\right) = \frac{\alpha\beta^2\sigma 1}{24} \quad \text{. .....} \quad (2.3b)$$

$$N\left(\frac{a}{2} - ex\right) = \frac{\alpha^2\beta\sigma 1}{24} \quad \text{. . .....} \quad (2.3c)$$

y obtenemos fácilmente

$$\alpha = 2a - 4ex \quad \text{.....} \quad (2.4a)$$

$$\beta = 2b - 4ey \quad \text{.....} \quad (2.4b)$$

$$\sigma 1 = \frac{6N}{\alpha\beta} = \frac{3N}{2(a - 2ex)(b - 2ey)} \quad \text{.....} \quad (2.4c)$$

2-3

$$N = \frac{\alpha\beta\sigma 1}{6}\left[1 - \left(\frac{\beta - b}{\beta}\right)^3\right] \quad \text{.....} \quad (2.5a)$$

$$N\left(\frac{b}{2} - ey\right) = \frac{\alpha\beta\sigma 1}{24}\left[\beta - \left(\frac{\beta - b}{\beta}\right)^3(\beta + 3b)\right] \quad \text{.....} \quad (2.5b)$$

$$N\left(\frac{a}{2}-ex\right)=\frac{\alpha\beta\sigma 1}{24}\left[\alpha-\left(\frac{\beta-b}{\beta}\right)^4\alpha\right] \quad \text{.....} \quad (2.5c)$$

Dividiendo (2.5b) entre (2.5a), obtenemos:

$$\left(\frac{b}{2}\right)-ey=\frac{3b^3-8\beta b^2+6\beta^2 b}{4\left(3\beta^2-3\beta b+b^2\right)} \quad \text{.....} \quad (2.6)$$

$\mu=2-\left(\frac{4ey}{b}\right)$ y $\delta=\frac{\beta}{b}$ luego:

$$(6-3\mu)\delta^2-(8-3\mu)\delta+3-\mu=0 \quad \text{.....} \quad (2.7)$$

Obtenemos:

$$\delta=\frac{8-3\mu+\left(12\mu-3\mu^2-8\right)^{1/2}}{12-6\mu} \quad \text{.....} \quad (2.8a)$$

y dividiendo (2.5c) entre (2.5a), obtenemos

$$\alpha=4\delta\left(\frac{a}{2}-ex\right)\frac{\delta^3-(\delta-1)^3}{\delta^4-(\delta-1)^4} \quad \text{.....} \quad (2.8b)$$

Finalmente de (2.5a)

$$\sigma 1=\frac{6N\delta^2}{\alpha b\left[\delta^3-(\delta-1)^3\right]} \quad \text{.....} \quad (2.8c)$$

y de relaciones similares

$$\sigma 4=\sigma 1\left(1-\frac{1}{\delta}\right) \quad \text{.....} \quad (2.8d)$$

$$\gamma=\alpha\left(1-\frac{1}{\delta}\right) \quad \text{.....} \quad (2.8e)$$

3

$$N = \frac{ab\sigma 3}{2\alpha\beta}\left(\alpha + \beta - 2\alpha\beta + \frac{\alpha^2\beta^3}{3}\right) \quad \text{.....} \quad (2.9a)$$

$$N\left(ey + \frac{b}{2}\right) = \frac{ab^2\sigma 3}{24\alpha\beta}\left(6\beta - 12\alpha\beta + 8\alpha + \alpha^2\beta^3\right) \quad \text{.....} \quad (2.9b)$$

$$N\left(ex + \frac{a}{2}\right) = \frac{a^2b\sigma 3}{24\alpha\beta}\left(6\alpha - 12\alpha\beta + 8\beta + \beta^2\alpha^3\right) \quad \text{.....} \quad (2.9c)$$

Dividiendo (2.9b) y (2.9c) por (2.9a)

$$\frac{4ex}{a}\left(3\alpha + 3\beta - 6\alpha\beta + \alpha^2\beta^2\right) = 2\beta - 2\alpha^2\beta^2 + \beta^2\alpha^3 \quad \text{.....} \quad (2.10a)$$

$$\frac{4ey}{b}\left(3\alpha + 3\beta - 6\alpha\beta + \alpha^2\beta^2\right) = 2\alpha - 2\alpha^2\beta^2 + \alpha^2\beta^3 \quad \text{.....} \quad (2.10b)$$

entonces, dividiendo (2.9b) y (2.9a)

Un sistema de dos ecuaciones no lineales, los cuales no se ha podido encontrar una solución especifica. Sin embargo, con algunas transformaciones algebraicas, se puede simplificar bastante; utilizando suma y división de ecuaciones.

$$A\left(3\alpha + 3\beta - 6\alpha\beta + \alpha^2\beta^2\right) = 2(\alpha + \beta) - 4\alpha^2\beta^2 + (\alpha + \beta)\alpha^2\beta^2 \quad \text{.....} \quad (2.11a)$$

$$C\left(2\alpha - 2\alpha^2\beta^2 + \alpha^2\beta^3\right) = 2\beta - 2\alpha^2\beta^2 + \beta^2\alpha^3 \quad \text{.....} \quad (2.11b)$$

Donde:

$$A = 4\left[\left(\frac{ex}{a}\right) + \left(\frac{ey}{b}\right)\right] \quad \text{.....} \quad (2.12a)$$

$$C = \frac{bex}{aey} \quad \text{.....} \quad (2.12b)$$

Sabiendo que $u = \alpha + \beta$; $v = \alpha \times \beta$ (2.11a)

$A(3u-6v+v^2) = 2u-4v^2+uv^2$ ..... (2.13)

Luego:

$$u = v\frac{6A - (A+4)v}{(3A - 2 - v^2)} \quad \text{.....} \quad (2.14)$$

$$\beta = \frac{u\left(v^2 - 2C\right) + 2v^2\left(C-1\right)}{\left(C+1\right)\left(v^2-2\right)} \quad \text{..... (2.15)}$$

$$\alpha = \frac{u\left(cv^2 - 2\right) - 2v^2\left(C-1\right)}{\left(C+1\right)\left(v^2-2\right)} \quad \text{..... (2.16)}$$

Así, en orden resolvemos el sistema de ecuaciones, solo $\alpha(v)\beta(v) = v$ se necesita. Después de racionalizarlo, se encuentra un polinomio en función de v.

$P_9(v) = 0$..... (2.17)

Finalmente, removiendo el factor común v, se encuentra una sola solución independiente de A y C, de acuerdo a (2.14), (2.15), (2.16) y (2.12)

$P8(v) = 0$ ..... (2.18)

Un polinomio de grado ocho por el cual se puede graficar (aunque es necesario un exhaustivo análisis numérico-gráfico), y donde se encuentra una única solución posible para la correspondiente área.

Para el caso particular de ex/a = ey/b, (C=1), el polinomio se reduce a cuarto grado.

$$P_4(v) = \sum_{i=0}^{4} B_i v^i = 0 \quad \text{..... (2.19)}$$

y v puede ser obtenido de la siguiente forma:

$$p = B_2 - \frac{3B_3^2}{8} \quad \text{..... (2.20a)}$$

$$q = B_1 - \frac{B_3 B_2}{2} + \frac{B_3^3}{8} \quad \text{..... (2.20b)}$$

$$r = B_0 - \frac{3B_3^4}{256} + \frac{B_3^2 B_2}{16} - \frac{B_3 B_1}{4} \quad \text{..... (2.20c)}$$

$$p' = \frac{p^2 - 4r}{16} - \frac{p^2}{12} \quad \text{..... (2.21a)}$$

$$q' = \frac{p^3}{108} - p\frac{p^2 - 4r}{96} - \frac{q^2}{64} \quad \text{..... (2.21b)}$$

$$\phi = \left(-\frac{p'}{3}\right)^{3/2} \quad \text{..... (2.22a)}$$

$$\theta = ar\cos\frac{-q'}{2\phi} \quad \text{.....} \quad (2.22b)$$

$$y_i = 2\phi^{1/3}\cos\left[\frac{\theta}{3}+120(i-1)\right]-\frac{p}{6} \quad \text{para i = 1, 2, 3.....} \quad (2.23)$$

finalmente (con S = q/q/)

$$v_1 = S\left(-\sqrt{y_1}+\sqrt{y_2}+\sqrt{y_3}\right)-\frac{B_3}{4} \quad \text{.....} \quad (2.24a)$$

$$v_2 = S\left(-\sqrt{y_1}+\sqrt{y_2}-\sqrt{y_3}\right)-\frac{B_3}{4} \quad \text{.....} \quad (2.24b)$$

$$v_3 = S\left(-\sqrt{y_1}-\sqrt{y_2}-\sqrt{y_3}\right)-\frac{B_3}{4} \quad \text{.....} \quad (2.24c)$$

$$v_4 = S\left(\sqrt{y_1}-\sqrt{y_2}+\sqrt{y_3}\right)-\frac{B_3}{4} \quad \text{.....} \quad (2.24d)$$

De (9a) y los esfuerzos respectivamente

$$\sigma 3 = \frac{2N\alpha\beta}{ab\left(\alpha+\beta-2\alpha\beta+\frac{\alpha^2\beta^2}{3}\right)} \quad \text{.....} \quad (2.25a)$$

$$\sigma 1 = \sigma 3\left(\frac{1}{\alpha}+\frac{1}{\beta}-1\right) \quad \text{.....} \quad (2.25b)$$

$$\sigma 2 = \sigma 3\left(\frac{1}{\beta}-1\right) \quad \text{.....} \quad (2.25c)$$

$$\sigma 4 = \sigma 3\left(\frac{1}{\alpha}-1\right) \quad \text{.....} \quad (2.25d)$$

2.3. SOLUCIÓN ANALÍTICA AL PROBLEMA DE R. IRLES Y F. IRLES.

R. Irles y F. Irles lograron determinar las ecuaciones para cada región donde caen las cargas según su excentricidad, sin embargo no llegaron a completar las ecuaciones cuando las cargas caen en 1, 2, 3, 4 por lo que a continuación se explica el método seguido para hallarlas.

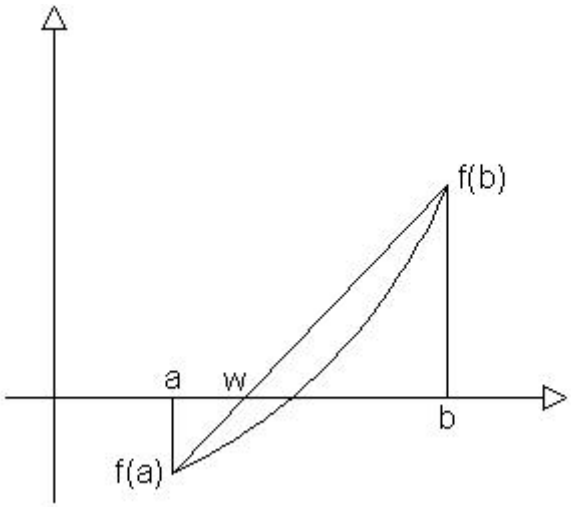
Se sabe que:

$$\alpha(v)\times\beta(v)=v$$

Reemplazando de 2.14, 2.15, 2.16 se obtiene el siguiente polinomio:

$$\begin{aligned}
&v^8\times\left(-(C+1)^2\right)+v^7\times\left[(A+4)^2\times C+2\times(C-1)^2\times(A+4)-4\times(C-1)^2\right]+\\
&v^6\times\left[-12\times A\times(A+4)\times C-12\times(C-1)^2\times A+6\times(C+1)^2\times A\right]+\\
&v^5\times\left[36\times A^2\times C-2\times(A+4)^2\times(C^2+1)+2\times(C-1)^2\times(A+4)\times(4-3\times A)+8\times(C-1)^2\times(3\times A-2)\right]+\\
&v^4\times\left[24\times A\times(A+4)\times(C^2+1)+12\times(C-1)^2\times A\times(3\times A-4)-(C+1)^2\times\left[(3\times A-2)\times(3\times A+6)+4\right]\right]+\\
&v^3\times\left[-72\times A^2\times(C^2+1)+4\times(A+4)^2\times C+4\times(C-1)^2\times(A+4)\times(2-3\times A)-4\times(C-1)^2\times(3\times A-2)^2\right]+\\
&v^2\times\left[-48\times A\times(A+4)\times C+12\times(C-1)^2\times A\times(6\times A-4)+12\times A\times(3\times A-2)\times(C+1)^2\right]\\
&+v\times\left[144\times A^2\times C\right]-4\times(C+1)^2\times(3\times A-2)^2=0
\end{aligned}$$

Para hallar el valor de v, se utilizó el método iterativo del Regular Falsi.



$$w = \frac{af(b) - bf(a)}{f(b) - f(a)}$$

Una vez obtenido v, se reemplaza en las ecuaciones 2.14, 2.15 y 2.16; para obtener u, α y β respectivamente.

Para encontrar los valores iniciales de a y b, se procedió a graficar el polinomio utilizando el software Matlab, debido a que el polinomio en mención se encuentra en función de A y C y estos a su vez en función de las longitudes de la zapata y sus excentricidades, se gráfico dándole valores los cuales caen en las regiones 1, 2, 3, 4.
En estos gráficos se puede observar que de las posibles ocho raíces, cuatro son reales y las restantes se encuentran en el campo de los números complejos. De las cuatro raíces reales la mitad son valores positivos, por lo tanto de las ocho posibles raíces sólo dos pueden ser tomadas como posibles valores de v.
Posteriormente se tomo los intervalos de ambas raíces tomando las crestas de las curvas de los gráficos obtenidos por Matlab, se tomaron dichos intervalos en el software y se compararon los resultados con los ejercicios resueltos por *Juan Ortega García*, se encontró que el intervalo correcto era a = 0 y b = 0.7.

Con este desarrollo se conoce las presiones de contacto en la zapata, con esta presión se procede a calcular los momentos y cortantes asumiendo de manera conservadora, que es uniforme en toda el área de la zapata. Los códigos de los momentos y cortantes se encuentran en el anexo 3.

CAPÍTULO 3

DESARROLLO DEL SOFTWARE DIZA 1.0

3.1. VISUAL BASIC: INTRODUCCIÓN

El lenguaje de programación BASIC (Beginner's All purpose Symbolic Instruction Code) nació en el año 1964 como una herramienta destinado a principiantes, buscando una forma sencilla de realizar programas, empleando un lenguaje casi igual al usado en la vida ordinaria (en inglés), y con instrucciones muy sencillas y escasas. Teniendo en cuenta el año de su nacimiento, este lenguaje cubría casi todas las necesidades para la ejecución de programas. Téngase en cuenta que las máquinas existentes en aquella época estaban estrenando los transistores como elementos de conmutación, los ciclos de trabajo llegaban a la impensable cifra de 10.000 por segundo y la memoria no pasaba de unos pocos k´s en toroides de ferrita.

La evolución del BASIC por los años 70 fue escasa, dado el auge que tomaron en aquella época lenguajes de alto nivel como el FORTRAN y el COBOL. En 1978 se definió una norma para unificar los Basics existentes creándose la normativa BASIC STANDARD

Con la aparición de los primeros ordenadores personales, dedicados comercialmente al usuario particular, allá por la primera mitad de los ochenta, el BASIC resurgió como lenguaje de programación pensado para principiantes, y muchos de estos pequeños ordenadores domésticos lo usaban como único sistema operativo (Sinclair, Spectrum, Amstrad)

Con la popularización del PC, salieron varias versiones del BASIC que funcionaban en este tipo de ordenadores (Versiones BASICA, GW-BASIC), pero todas estas versiones del BASIC no hicieron otra cosa que terminar de rematar este lenguaje. Los programadores profesionales no llegaron a utilizarlo, habida cuenta de las desventajas de este lenguaje respecto a otras herramientas (PASCAL, C, CLIPPER). El BASIC con estas versiones para PC llegó incluso a perder crédito entre los profesionales de la informática.

Las razones para ello eran obvias:

- No era un lenguaje estructurado.

- No existían herramientas de compilación fiables.

- No disponía de herramientas de intercambio de información.

- No tenía librerías.

- No se podía acceder al interior de la máquina.

- Una gran cantidad de desventajas respecto a otros lenguajes de programación.

Tal fue ese abandono por parte de los usuarios, que la aparición del Quick-BASIC de Microsoft, una versión ya potente del BASIC, que corregía casi todos los defectos de las versiones pasó prácticamente inadvertida, a no ser porque las últimas versiones del sistema operativo MS-DOS incluían una versión de Quick-BASIC algo recortada (Q-Basic) como un producto mas dentro de la amplia gama de ficheros ejecutables que acompañan al sistema operativo, y aprovecha de él el editor de textos (Cada vez que se llama al EDIT estamos corriendo el editor del Q-Basic).

Esta versión del popular BASIC ya es un lenguaje estructurado, lo que permite crear programas modularmente, mediante subrutinas y módulos, capaz de crear programas ya competitivos con otros lenguajes de alto nivel. Sin embargo llegaba tarde, pues los entornos MS-DOS estaban ya superados por el entorno gráfico Windows.

Sin embargo algo había en el BASIC que tentaba a superarse: su gran sencillez de manejo. Si a esto se le añade el entorno gráfico Windows, el aprovechamiento al máximo de las posibilidades de Windows en cuanto a intercambio de información, de sus librerías, de sus drivers y controladores, manejo de bases de datos, etc. el producto resultante puede ser algo que satisfaga todas las necesidades de programación en el entorno Windows. La suma de todas estas cosas es VISUAL - BASIC. Esta herramienta conserva del BASIC de los años 80 únicamente su nombre y su sencillez, y tras su lanzamiento al mercado, la aceptación a nivel profesional hizo borrar por fin el "mal nombre" asociado a la palabra BASIC.

Actualmente se está comercializando la versión 6.0 de este producto. Desde su salida al mercado, cada versión supera y mejora la anterior. Dados los buenos resultados a nivel profesional de este producto, y el apoyo prestado por el fabricante para la formación de programadores, Visual-Basic se ha convertido en la primera herramienta de desarrollo de aplicaciones en entorno Windows.

Es obligado decir sin embargo, que sigue siendo BASIC. No se pueden comparar sus prestaciones con otros lenguajes cuando deseamos llegar al fondo de la máquina y controlar uno a uno sus registros. No es ese el fin perseguido con VB y si es necesario llegar a esas precisiones será necesario utilizar otro lenguaje que permita bajar el nivel de programación. (Visual-C). o realizar librerías (DLLs) que lo hagan. En la mayor parte de las aplicaciones, las herramientas aportadas por VB son mas que suficiente para lograr un programa fácil de realizar y de altas prestaciones.

Características Generales de Visual-Basic

Visual-Basic es una herramienta de diseño de aplicaciones para Windows, en la que estas se desarrollan en una gran parte a partir del diseño de una interface gráfica. En una aplicación Visual - Basic, el programa está formado por una parte de código puro, y otras partes asociadas a los objetos que forman la interface gráfica.

Es por tanto un termino medio entre la programación tradicional, formada por una sucesión lineal de código estructurado, y la programación orientada a objetos. Combina ambas tendencias. Ya que no podemos decir que VB pertenezca por completo a uno de esos dos tipos de programación, debemos inventar una palabra que la defina : PROGRAMACION VISUAL.

La creación de un programa bajo Visual Basic lleva los siguientes pasos:

- Creación de un interface de usuario. Este interface será la principal vía de comunicación hombre máquina, tanto para salida de datos como para entrada. Será necesario partir de una ventana - Formulario - a la que le iremos añadiendo los controles necesarios.

- Definición de las propiedades de los controles - Objetos - que hayamos colocado en ese formulario. Estas propiedades determinarán la forma estática de los controles, es decir, como son los controles y para qué sirven.

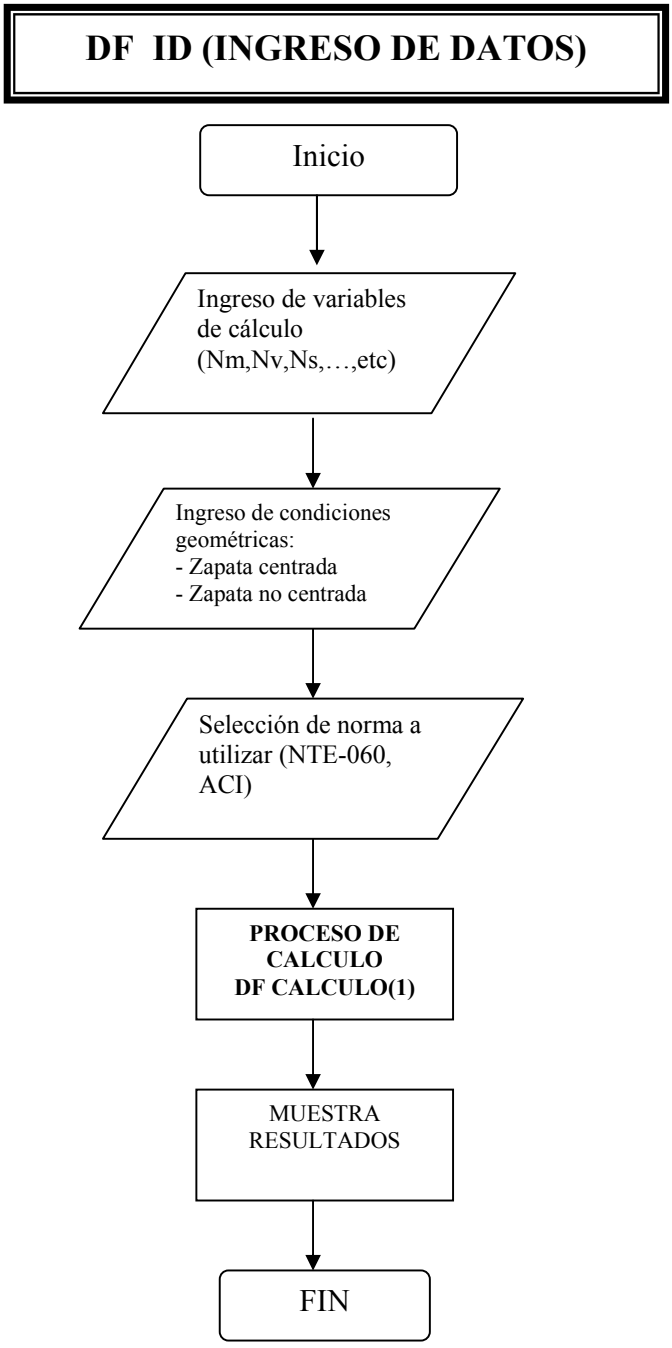
- Generación del código asociado a los eventos que ocurran a estos objetos. A la respuesta a estos eventos (click, doble click, una tecla pulsada, etc.) le llamamos Procedimiento, y deberá generarse de acuerdo a las necesidades del programa.

- Generación del código del programa. Un programa puede hacerse solamente con la programación de los distintos procedimientos que acompañan a cada objeto. Sin embargo, VB ofrece la posibilidad de establecer un código de programa separado de estos eventos. Este código puede introducirse en unos bloques llamados Módulos, en otros bloques llamados Funciones, y otros llamados Procedimientos. Estos Procedimientos no responden a un evento acaecido a un objeto, sino que responden a un evento producido durante la ejecución del programa.

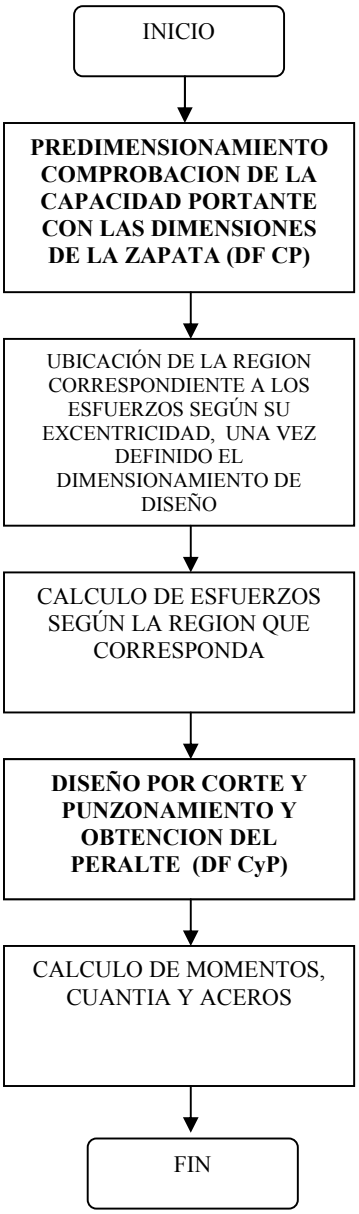
No es necesario entender de momento lo anterior. Visual Basic introduce un concepto nuevo de programación, y es necesario cambiar hasta el argot del programador. Posiblemente se le habrán acumulado demasiados términos de una sola vez. Es normal. A poco que siga leyendo verá las cosas mas claras cuando se explique una por una.

3.2. DIAGRAMAS DE FLUJO DE DIZA 1.0.

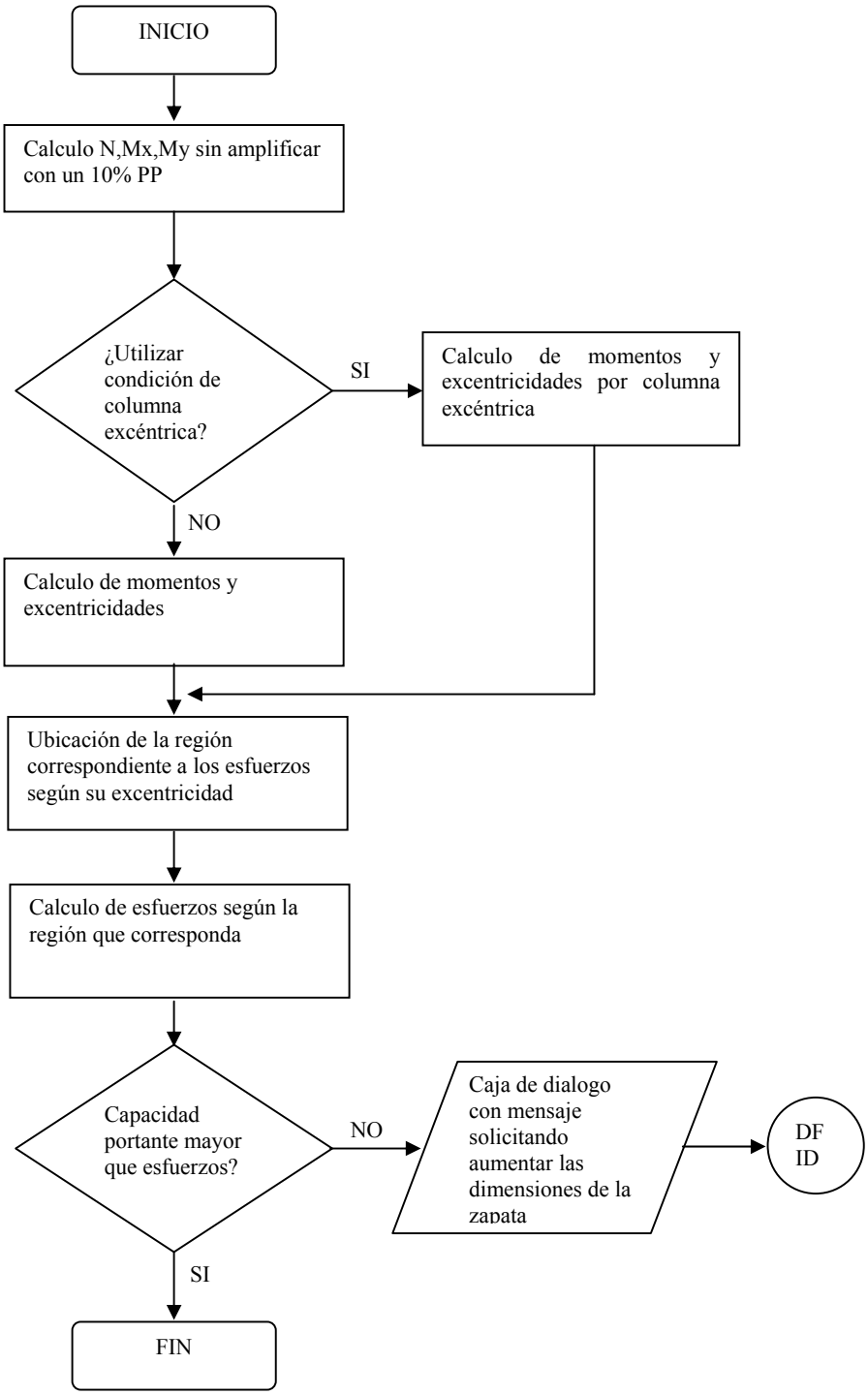
Estos diagramas muestran la secuencia de diseño que realiza el programa y cuyo desarrollo se detalla en el capítulo 4.

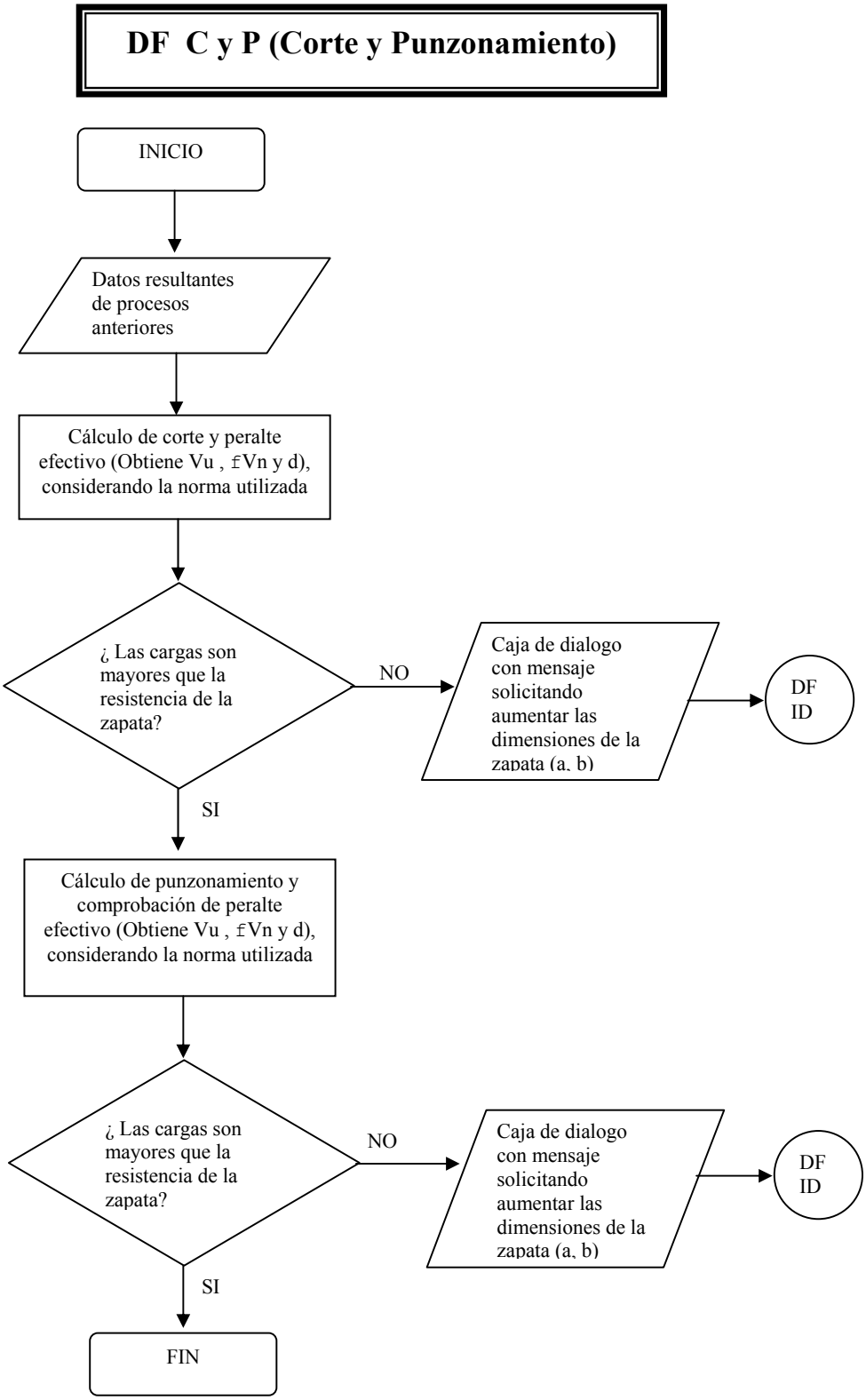


DF CALCULO (1)


INICIO
PREDIMENSIONAMIENTO COMPROBACION DE LA CAPACIDAD PORTANTE CON LAS DIMENSIONES DE LA ZAPATA (DF CP)
UBICACIÓN DE LA REGION CORRESPONDIENTE A LOS ESFUERZOS SEGÚN SU EXCENTRICIDAD, UNA VEZ DEFINIDO EL DIMENSIONAMIENTO DE DISEÑO
CALCULO DE ESFUERZOS SEGÚN LA REGION QUE CORRESPONDA
DISEÑO POR CORTE Y PUNZONAMIENTO Y OBTENCION DEL PERALTE (DF CyP)
CALCULO DE MOMENTOS, CUANTIA Y ACEROS
FIN

DF CP (Comprobación de la Capacidad Portante)




DF C y P (Corte y Punzonamiento)
INICIO
Datos resultantes de procesos anteriores
Cálculo de corte y peralte efectivo (Obtiene Vu , fVn y d), considerando la norma utilizada
¿ Las cargas son mayores que la resistencia de la zapata?
NO
Caja de dialogo con mensaje solicitando aumentar las dimensiones de la zapata (a, b)
DF ID
SI
Cálculo de punzonamiento y comprobación de peralte efectivo (Obtiene Vu , fVn y d), considerando la norma utilizada
¿ Las cargas son mayores que la resistencia de la zapata?
NO
Caja de dialogo con mensaje solicitando aumentar las dimensiones de la zapata (a, b)
DF ID
SI
FIN

CAPÍTULO 4

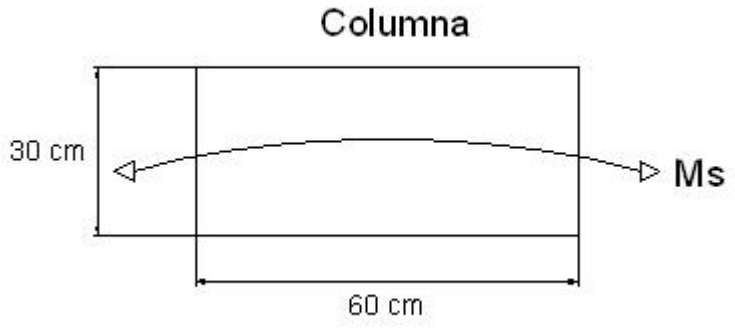
VALIDACIÓN DEL SOFTWARE.

A continuación se procederá a comparar ejercicios resueltos de diferentes autores con los resultados del programa Diza 1.0.
El desarrollo de estos ejemplos tienen las mismas hipótesis que el programa desarrollado, principalmente que las presiones se distribuyen linealmente

Ejercicio resuelto en clase de Concreto Armado II con el Ing. Arturo Martínez Ramírez:

Ejercicio 1

Datos
Pm = 80 Ton
Pv = 60 Ton
Ps = 0
Ms = ± 20 Ton*m
Mm = 0
Mv = 0
σt = 3 kg/cm2 ….. capacidad portante del suelo



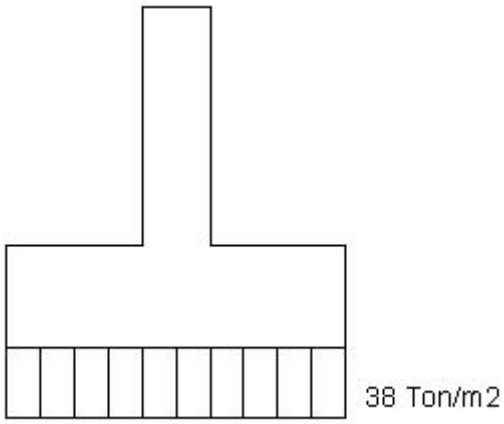
Dimensiones de zapata: 2.3*2.6m

Solución

Sin Sismo

P = 1.5*80+1.8*60 = 228 Ton

$$qu = \frac{228}{2.3 \times 2.6} = 38 ton/m^2$$



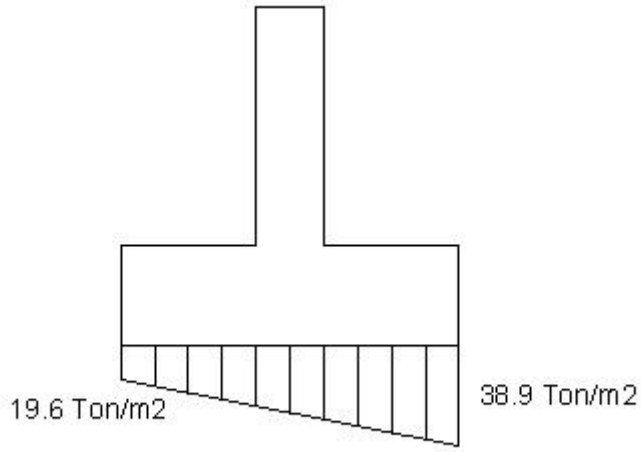
Con Sismo

P = 1.25(80+60) = 175 Ton

M = 1.25*20 = 25 Ton

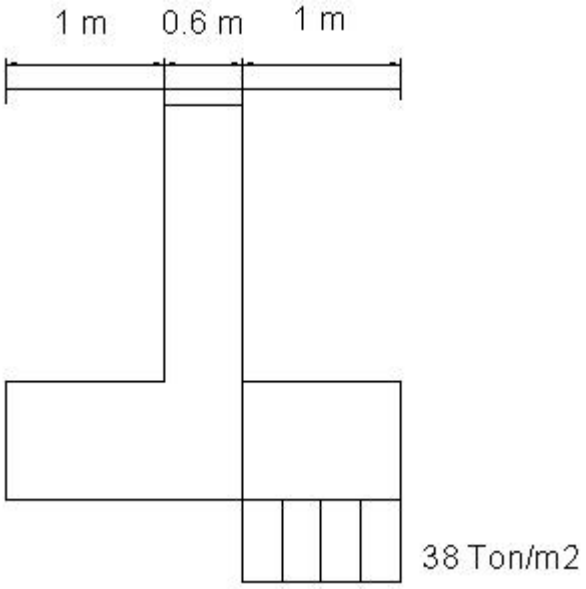
$$qu = \frac{175}{2.3 \times 2.6} \pm \frac{6 \times 25}{2.3 \times 2.6^2} = 29.26 \pm 9.64 ton/m^2$$

q1 = 38.9 Ton/m^2
q2 = 19.6 Ton/m^2



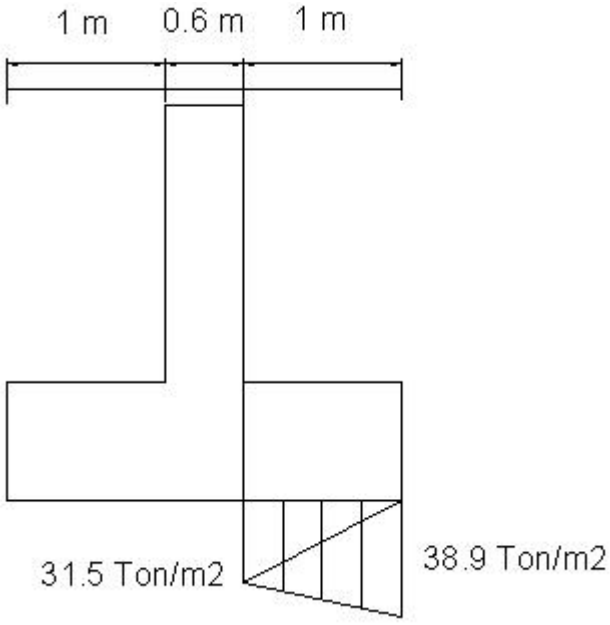
Diseño por Flexión

Sin Sismo



$$\text{Mu} = \frac{38\left(ton/m^2\right)\times 1(m)}{2} = 19 \text{ Ton*m/m}$$

Con Sismo



$$\text{Mu} = \left(\frac{31.5\left(Ton/m^2\right)\times 1(m)}{2}\right)\times\left(\frac{1}{3}\times 1(m)\right)+\left(\frac{38.9\left(ton/m^2\right)\times 1(m)}{2}\right)\times\left(\frac{2}{3}\times 1(m)\right) =$$

18.2 Ton*m/m

Diseño de acero

Mu = 19 Ton/m^2
b = 1 m
d = 0.5 m

$$ku = \frac{19 \times 10^5}{100 \times 50^2} = 7.6$$

f´c = 210 Kg/cm^2
ρ = 0.0021
ϕ5/8¨@20cm en ambas direcciones

Corte

Vu = 19 Ton

$$\phi Vn = 0.85 \times 0.53 \times \sqrt{210} \times 100 \times 50 = 32.6 Ton$$

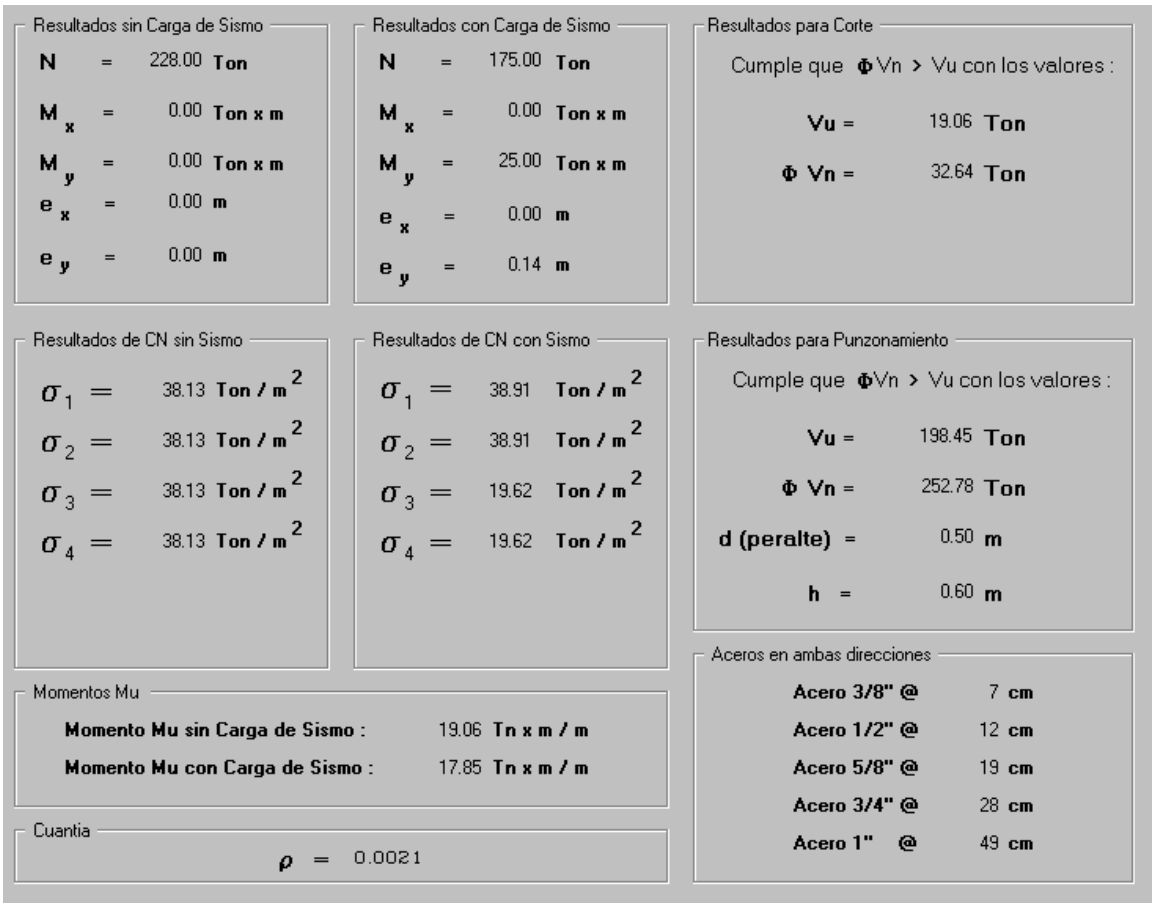
ϕVn>Vu

Punzonamiento

Vu = 38*(2.3*2.6-1.1*0.8) = 193.8 Ton
ϕVn = 253 Ton

ϕVn>Vu

Resultados de Diza 1.0.



Nota:
CN : Núcleo Central

Ejercicio resuelto por el Ing. *Manuel Delgado Vargas* en el libro "Diseño de Estructuras de Concreto Armado"

Ejercicio 2

Datos
Columna 40x80cm
Cargas Pm = 130 Ton
Pv = 70 Ton

Mmx = 10 Tonxm.	Mmy = 2 Tonxm.
Mvx = 6 Tonxm	Mvy = 1 Tonxm.
Msx = 15 Tonxm	Msy = 13 Tonxm
Psx = 10 Ton	Psy = 9 Ton

Resistencia del terreno = 3 kg/cm^2

Dado que hay momentos en las dos direcciones, se tendrá que verificar diversas hipótesis, teniendo presente que el sismo no actúa simultáneamente en las dos direcciones.

Dimensionamiento:

1era Verificación (Momentos sin sismo)

Pm = 130 Ton
Pv = 70 Ton

Mmx = 10 Tonxm.	Mvx = 6 Tonxm.
Mmy = 2 Tonxm.	Mvy = 1 Tonxm.

Ptotal = 200 Ton

Area tentativa = $\frac{200 \times 1.05}{27} = 7.77 m^2$

Se considera 27 en lugar de 30 (dato del suelo) debido a que en este tanteo no se están tomando en cuenta los momentos.

Columna 40x80; diferencia de lados = 40cm.

Buscamos dos lados de zapata que tengan una diferencia de 40cm. Y un producto de 7.8 m^2 aproximadamente:

B = 2.6m L = 3.00m

Area = 2.6x3.0 = 7.8 m^2

Verificamos Momentos en x

$$\sigma = \frac{P}{A} + \frac{6 \times M}{B \times L^2} = \frac{200 \times 1.05}{2.6 \times 3.0} + \frac{6(10+6)}{2.6x3^2} = 26.92 + 4.1$$

entonces $\sigma = 31.02$ Ton/m^2

Como la presión obtenida es mayor a la admisible y todavía falta verificar la condición real de flexión biaxial, aumentamos las dimensiones de la zapata.

B = 2.7m L = 3.1m

Verificamos biaxialmente.

$$\sigma = \frac{200 \times 1.05}{2.7 \times 3.1} + \frac{6 \times (10+6)}{2.7 \times 3.1^2} + \frac{6 \times (2+1)}{3.1 \times 2.7^2}$$

$\sigma = 25.08+3.69+0.79=29.56$ Ton/m^2

2da Verificación (Sismo en x)

Pm = 130 Ton
Pv = 70 Ton
Mmx = 10 Tonxm Mmy = 2 Tonxm
Mvx = 6 Tonxm Mvy = 1 Tonxm
Psx = 10 Ton
Msx = 15 Tonxm

$$\sigma = \frac{210 \times 1.05}{2.7 \times 3.1} + \frac{6 \times (10+6+15)}{2.7 \times 3.1^2} + \frac{6(2+1)}{3.1 \times 2.7^2}$$

$\sigma = 26.34 + 7.16 + 0.79 = 34.29$Ton/m^2

Esta presión es mayor a la resistente, sin embargo, la mayoría de los Ingenieros de Suelos reconocen que se puede considerar hasta un 33% de incremento en la presión resistente al tratarse el sismo de un efecto eventual y de corta duración.

3da Verificación (Sismo en y)

Pm = 130 Ton
Pv = 70 Ton
Mmx = 10 Tonxm Mmy = 2 Tonxm
Mvx = 6 Tonxm Mvy = 1 Tonxm
Psy = 9 Ton
Msx = 13 Tonxm

$$\sigma = \frac{209 \times 1.05}{2.7 \times 3.1} + \frac{6 \times (10+6)}{2.7 \times 3.1^2} + \frac{6 \times (2+1+13)}{3.1 \times 2.7^2}$$

$\sigma = 26.21 + 3.69 + 4.24 = 34.14$ Ton/m^2

Diseño:

- Cuando no se consideró sismo se obtuvo $\sigma = 29.56$ Ton/m^2
$\sigma u = 29.56x1.6 = 47.29$ Ton/m^2
- Cuando se consideró sismo en x se obtuvo $\sigma = 34.29$ Ton/m^2
$\sigma u = 34.29x1.25 = 42.68$ Ton/m^2
- Cuando no se consideró sismo se obtuvo $\sigma = 34.14$ Ton/m^2
$\sigma u = 34.14x1.25 = 42.67$ Ton/m^2

Por lo tanto se efectuará el diseño con $\sigma u = 47.29$ Ton/m^2

(*) 1.6 es un valor intermedio entre 1.5 y 1.8 considerando mayor incidencia en la carga muerta.

Por Punzonamiento

Se tienen volados iguales = 1.15m

Suponiendo d (peralte efectivo) = 50cm
bo = 2(130)+2(90) = 440cm = 4.40m
Ao = 0.9x1.3 = 1.17 m^2
A_{total} = 3.1x2.7 = 8.37 m^2

Cortante de diseño por punzonamiento
$Vu = \sigma u(A_{total} - Ao) = 47.3(8.37-1.17) = 340.5$ Ton

Cortante resistente por Punzonamiento

$Vc = (0.53+1.1/Bc)x\sqrt{f'c}$ xboxd

Donde Bc = 80/40 = 2 entonces (0.53+1.1/2)=1.08

$Vc = 1.08x\sqrt{210}$ x440x50 = 344,315 Kg

Por tanto ϕVc=0.35x344.3 Ton = 292.6 Ton

Como Vu = 340.5 Ton no cumple el valor de d = 50cm

Aumentamos el peralte efectivo a d=60cm

bo = 2(60+80)+2(40+60)=480cm

Ao = 1x1.40=1.40m^2

A_{total} = 8.37 m^2

Cortante de diseño por punzonamiento

Vc = 1.08x $\sqrt{210}$ x480x60 = 450,739.7 Kg

Por tanto ϕVc = 0.85x450.7 Ton = 383.1 Ton

Como Vu = 329.6 Ton, el peralte efectivo de 60cm, es adecuado.

Por Cortante

Cortante de diseño:
Vu = 47.3(2.7)(1.15-06) = 70.24 Ton

Cortante resistente:
Vc=0.53x $\sqrt{210}$ x270x60 = 124422.9 Kg

Por tanto ϕVc = 0.85x124.4 = 105.7 Ton

Como Vu = 70.24 Ton, el peralte supuesto es adecuado.

Por Flexión

$$Mu = \frac{\sigma u(1.15)^2(2.7)}{2}$$

$$Mu = \frac{47.3(1.15)^2(2.7)}{2} = 84.44 Ton \times m$$

Como b=270 y d=60

Se tiene

$bd = 16200 cm^2$
$bd^2 = 972000\ cm^3$

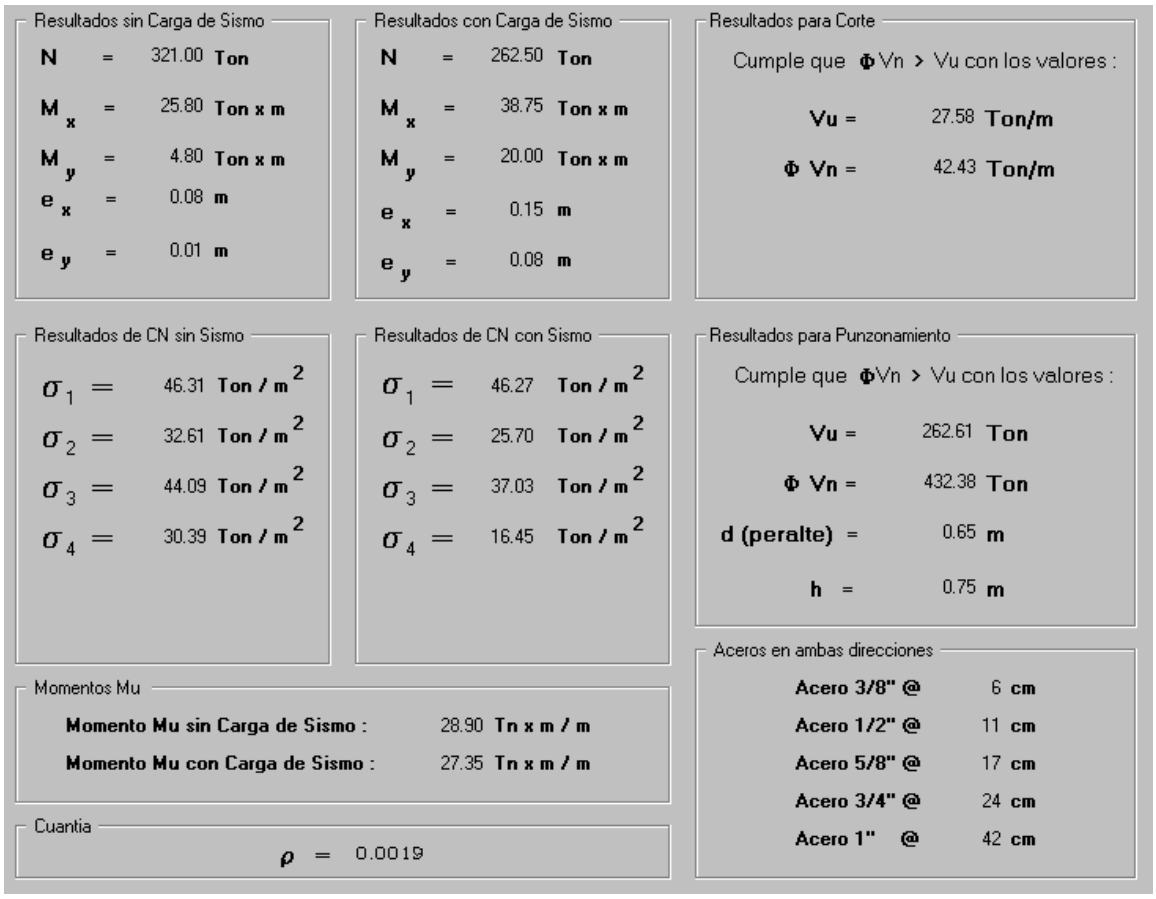
$Ku = Mu/bd^2 = 8.68$

$\rho = 0.00235$

$As = 0.00235x270x60 = 38.07\ cm^2$

Por tanto, usando fierro de 5/8´´ se tiene: 19ϕ5/8´´ en 2.7m de ancho, lo cual equivale a un espaciamiento de 15cm.

Resultados de Diza 1.0.



Comentario:

Debido a que *Manuel Delgado Vargas* realiza diversas simplificaciones como: diseñar uniaxialmente y elegir la mayor de las presiones halladas, o suponer que la presión es uniforme sus resultados con los del software difieren ligeramente.

Ejercicio resueltos por el Ing. *Juan Ortega García* (método visto en el capitulo I)

Ejercicio 3
Flexión biaxial con resultado en el núcleo central.

Datos

Caso I
P = 110 Ton
M = 8 Ton-m
M´ = 2 Ton-m
$\sigma t = 3 Kg/cm^2$

Asumir Zapata de 2*2.5m para un primer planteo

Solución

$$e = \frac{8}{110} = 0.0727 < \frac{L}{6} = \frac{2.5}{6} = 0.416 \text{ ...Presión total en la base}$$

$$e' = \frac{2}{110} = 0.0182 < \frac{B}{6} = \frac{2}{6} = 0.333 \text{ ...Presión total en la base}$$

$$\alpha = \frac{e}{L} = \frac{0.0727}{2.5} = 0.0291$$

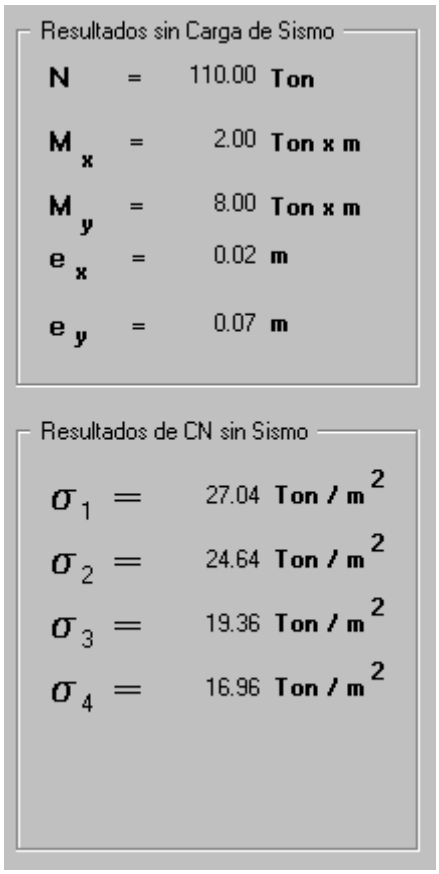
$$\beta = \frac{e'}{B} = \frac{0.0182}{2.0} = 0.0091$$

De tabla a
K ≅ 1.22
F.S. > 10 (ambos sentidos)

$$f = k\frac{P}{A} = 1.22 \times \frac{110}{2.5 \times 2.0} = 26.84 Ton/m^2$$

26.84 Ton/m^2 < 30 Ton/m^2…….. OK

Resultados de Diza 1.0.



Ejercicio resueltos por el Ing. *Juan Ortega García*

Ejercicio 4

Datos

Caso II
P = 25 Ton
M = 15 Ton-m
M´ = 5 Ton-m
$\sigma t = 25\ Ton/m^2$

Asumir Zapata de 2.9*1.4m para un primer planteo

Solución

$$e = \frac{15}{25} = 0.6 > \frac{L}{6} = \frac{2.9}{6} = 0.48$$

$$e' = \frac{5}{25} = 0.2 < \frac{B}{6} = \frac{1.4}{6} = 0.233$$

$$\alpha = \frac{e}{L} = \frac{0.6}{2.9} = 0.207$$

$$\beta = \frac{e'}{B} = \frac{0.2}{1.4} = 0.143$$

De tabla a
K ≅ 3.6
F.S. = 2.43 (La mínima)

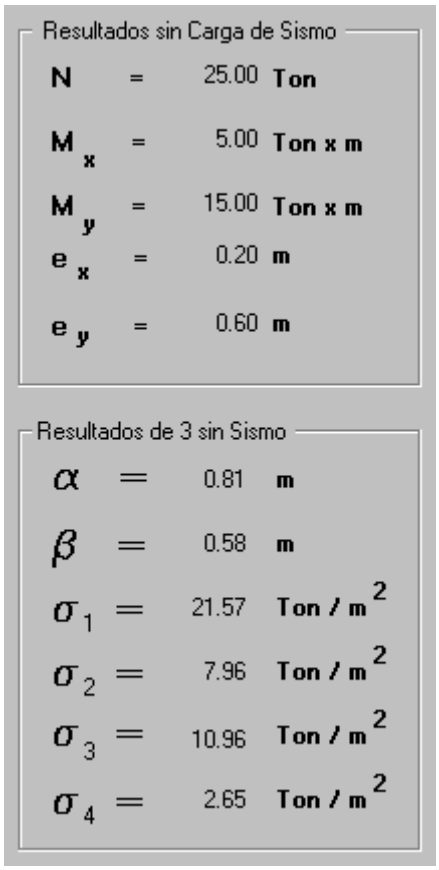
$$f = k\frac{P}{A} = 3.6 \times \frac{25}{2.9 \times 1.4} = 22.17 Ton/m^2$$

$22.17\ Ton/m^2 < 25\ Ton/m^2$…….. OK

Tabla a

X = 0.57
Y = 0.81
xL = 0.81*2.9 = 2.45m
yB = 0.51*14 = 0.71m

Resultados de Diza 1.0.



Ejercicio resueltos por el Ing. *Juan Ortega García*

Ejercicio 5

Datos

Caso III
P = 100 Ton
M = 45 Ton-m
M´ = 85 Ton-m
$\sigma t = 35\ Ton/m^2$

Asumir Zapata de 4.3*3m para un primer planteo

Solución

$$e = \frac{45}{100} = 0.45 < \frac{L}{6} = \frac{4.3}{6} = 0.717$$

$$e' = \frac{85}{100} = 0.85 > \frac{B}{6} = \frac{3.0}{6} = 0.5$$

$$\alpha = \frac{e}{L} = \frac{0.45}{4.3} = 0.105$$

$$\beta = \frac{e'}{B} = \frac{0.85}{3.0} = 0.283$$

De tabla a
K = 4.2
F.S. = 1.75 (La mínima)

$$f = k\frac{P}{A} = 4.2 \times \frac{100}{4.3 \times 3} = 32.56 Ton/m^2$$

$33.56\ Ton/m^2 < 35\ Ton/m^2$…….. OK

$$nB = \frac{3.0}{2} + 0.85 = 2.35 \ldots\ldots \quad n = 0.783$$

$$mL = \frac{4.3}{2} + 0.45 = 2.6 \ldots\ldots \quad m = 0.605$$

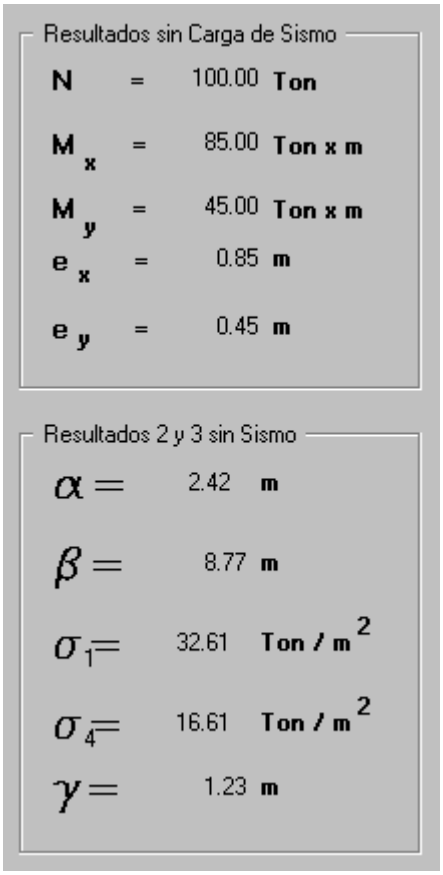
Con m = 0.605 Tabla b

$$a = \frac{3 \times 3(1 - 0.783)}{1 + 0.5(0.605 - 1)} = 2.43m$$

a(1-q) = 1.22m

f′ = 32.56(1-0.5) = 16.22 Ton/m^2

Resultados de Diza 1.0.



CAPITULO 5

CONCLUSIONES Y RECOMENDACIONES

5.1. CONCLUSIONES.

- Se concluye que la solución numérica aplicada al planteamiento de R. Iries y F. Iries ha sido satisfactoria al comprobarlo a través de varios ejemplos durante su validación.

- Como se demuestra la validación de los resultados del programa tienen bastante semejanza con los encontrados en las referencias bibliográficas, a nivel de esfuerzos destaca la similitud mientras que a nivel de diseño se obtienen resultados conservadores debido a que la hipótesis considera que la distribución de presiones de diseño actúa uniformemente en toda la zapata.

5.2. RECOMENDACIONES.

Para finalizar el trabajo, se recomienda lo siguiente:

- Se recomienda desarrollar un algoritmo que integre numéricamente los esfuerzos de compresión con la finalidad de hallar los cortantes y momentos de diseño.

- Antes de instalar el programa Diza 1.0, para un correcto funcionamiento del programa, tener en cuenta lo siguiente:

  Requerimientos del Sistema:

  - Resolución Minima 1,024x768 pixels

  - Copiar e instalar los fonts greekc y greeks en la carpeta \Windows\Fonts

  - Instalar el programa en el Directorio c:\DIZA

- Para diseñar, tener siempre en consideración de utilizar el eje Y para la mayor dimensión de la zapata, por lo tanto usar el eje X para la menor dimensión.

- Aumentar el software para los diferentes tipos de cimentaciones en próximas tesis, Diza 1.0 es el primer paso para poder obtener un software completo que permita resolver todo tipo de cimentaciones, ya sea zapatas combinadas, vigas de cimentación, losas, etc.

BIBLIOGRAFÍA

- Apuntes de Concreto Armado II, Ing. Arturo Martínez Ramírez, 2004
- Arthur H. Nilson. Diseño de Estructuras de Concreto. Editorial Mc Graw Hill. Octubre 1990.
- Delgado Vargas M. "Ingeniería de Cimentaciones" II Edición. Editorial alfa-omega. 1999. Colombia
- Juan Ortega García, "Diseño de Estructuras de Concreto Armado", Septiembre de 1990.
- Julio Rivera Feijoo, "Análisis y diseño estructural de cimentaciones superficiales", Diciembre, 1998.
- Normas de Construcciones en Concreto Estructural ACI 318-99, 2000.
- R. Iries y F. Iries. "Explicit Stresses Under Rectangular Footings", Journal of Geotechnical Engineering, Vol 120, N° 2, ASCE. Feb, 1994.
- SENCICO, "Norma Técnica E060 – Concreto Armado" , Año 2000.

ANEXO A

MANUAL DE USUARIO DIZA 1.0

Introducción:

Diza 1.0 es un programa que permite resolver los casos de flexión biaxial en zapatas rectangulares con columnas excéntricas.

- Primero, realiza una comprobación de las dimensiones de la zapata, es decir el predimensionamiento.
- Segundo amplifica las cargas y calcula la excentricidad, asimismo reconoce la zona de la zapata donde recaen dichas cargas y calcula los esfuerzos para dicha zona, estos esfuerzos son redistribuidos para que se proceda a diseño estructural de la zapata.
- Tercero, realiza el diseño estructural, calculando corte, punzonamiento y peralte de la zapata para luego proceder a calcular los momentos últimos, cuantía, aceros y espaciamientos correspondientes.

Diza además permite diseñar zapatas con columnas excéntricas y poder elegir que norma a utilizar NTP E060 o ACI.

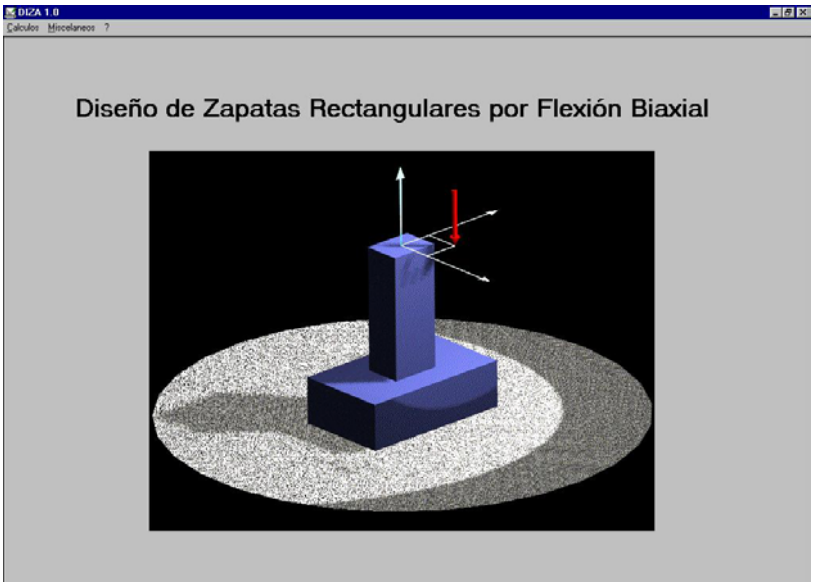
Este programa además da recomendaciones al usuario al correr el programa, para un correcto diseño.

Además Diza a medida que va dando resultados, muestra gráficos que permiten un mejor entendimiento al usuario.

Existen otras opciones como resumen de cálculos, formulas utilizadas, gráficos utilizados, entre otros, lo cual permite entender el método empleado para el diseño de zapatas.

Entorno de Diza 1.0

Este programa posee una barra de control, la cual contiene tres opciones: Cálculo, misceláneos y ?.



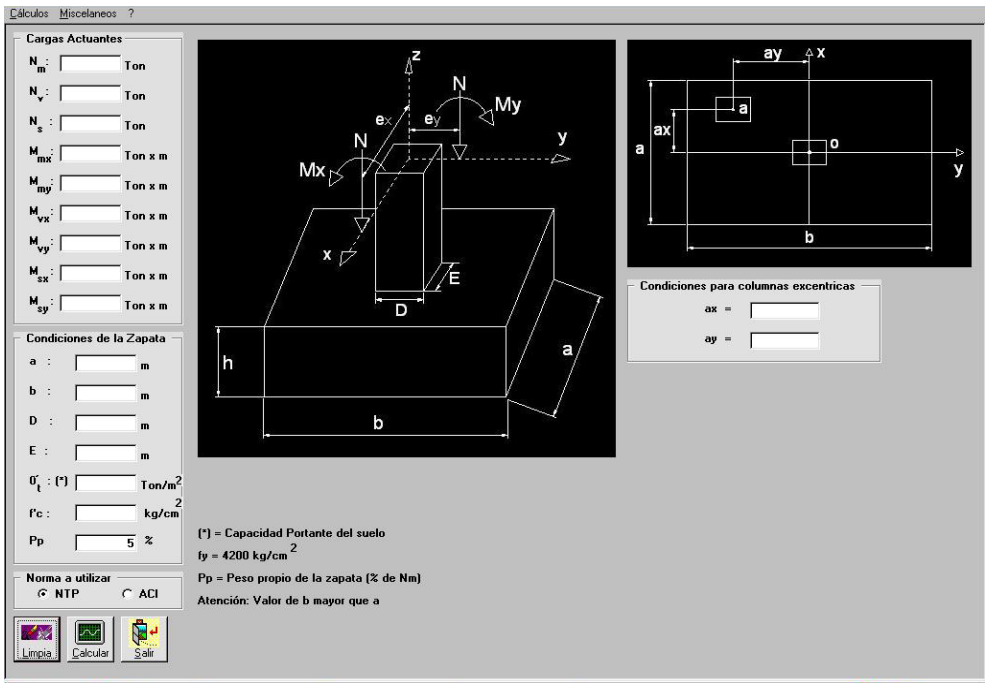
La opción Calculos cuenta con 3 sub opciones: Cálculo, Resumen de Cálculos y Salir del sistema.

La opción Misceláneos cuenta con 4 sub opciones: Problemática, Gráficos Utilizados, Formulas Utilizadas y Créditos.

La opción "?" posee una sub opción, la cual es Acerca de.

Cálculos

Para iniciar a calcular se debe ir a la opción Cálculos y a la sub Opción Cálculo, donde aparecerá la siguiente caja de dialogo.



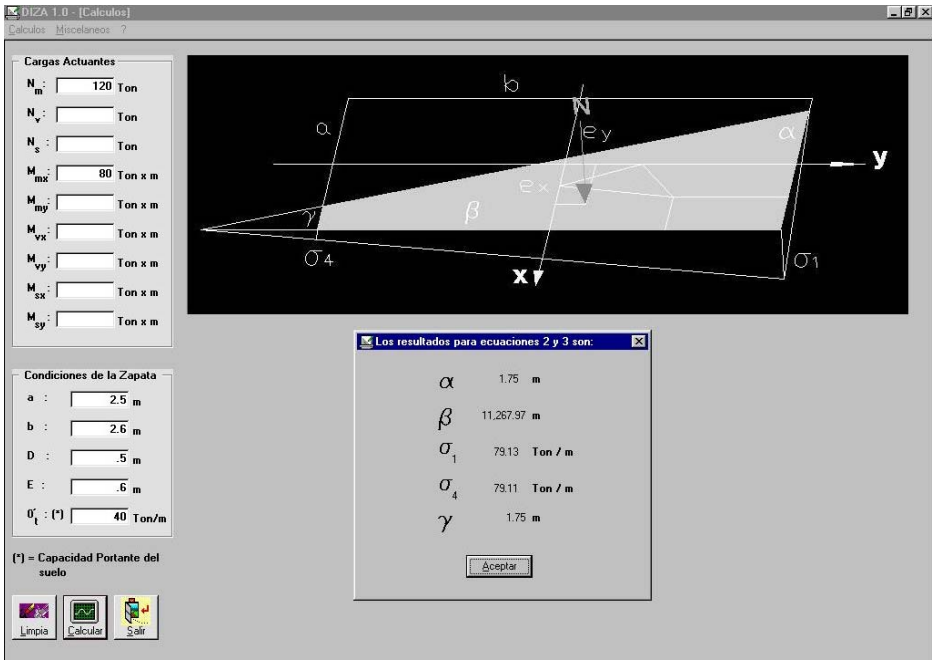
Esta caja de dialogo, mediante los gráficos guía al usuario a llenar los datos de una manera correcta, una vez llenado las casillas, se procederá a hacer clic en Calcular.

Si se desea llenar nuevos datos, se procederá a hacer clic en Limpiar, de esta forma el programa limpiará las casillas.

Por defecto Diza utiliza la norma peruana NTP, sin embargo el usuario podrá cambiarla por la norma ACI.

Para diseñar tener siempre en consideración de utilizar el eje y para la mayor dimensión de la zapata, por lo tanto usar el eje x para la menor dimensión.

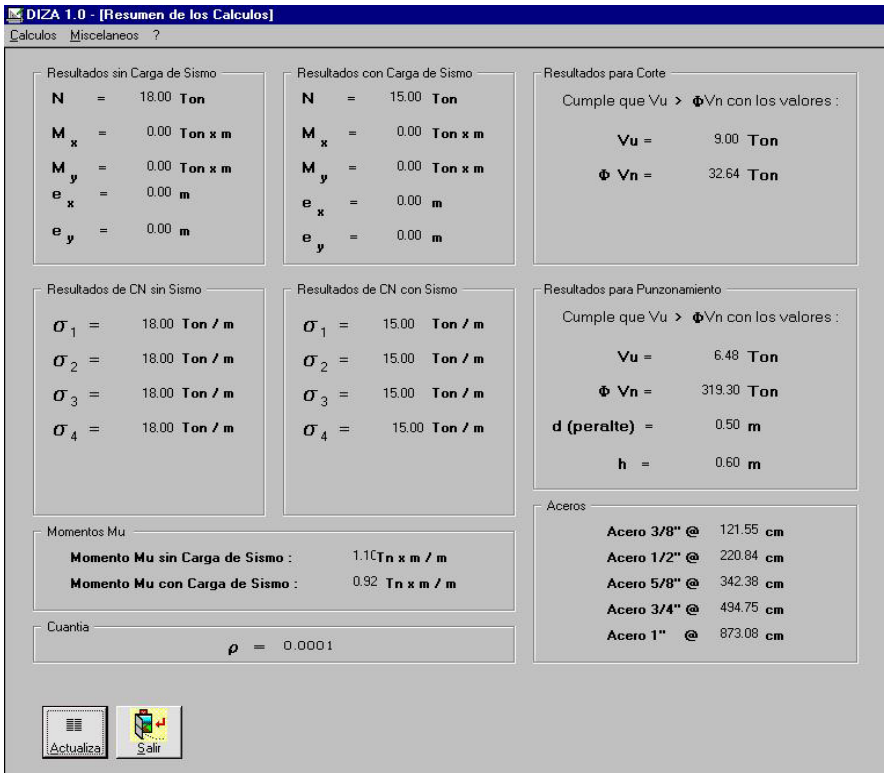
A medida que el programa va mostrando los resultados muestra diferentes gráficos, tal como se muestra a continuación.



Nota:

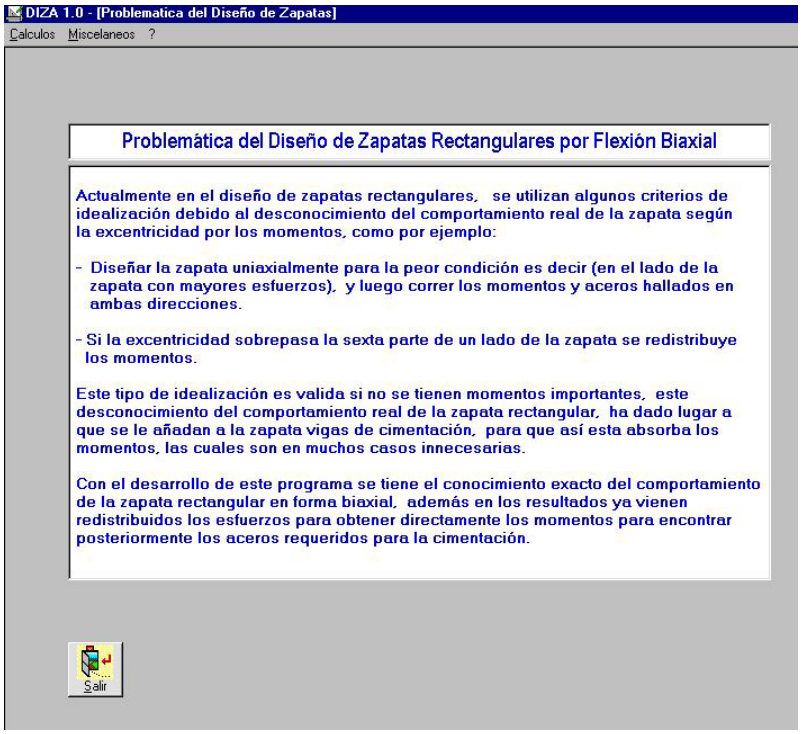
El programa elige para los cálculos (Cortantes, punzonamientos y momentos), los resultados mayores de Con carga de sismo y Sin carga de sismo.

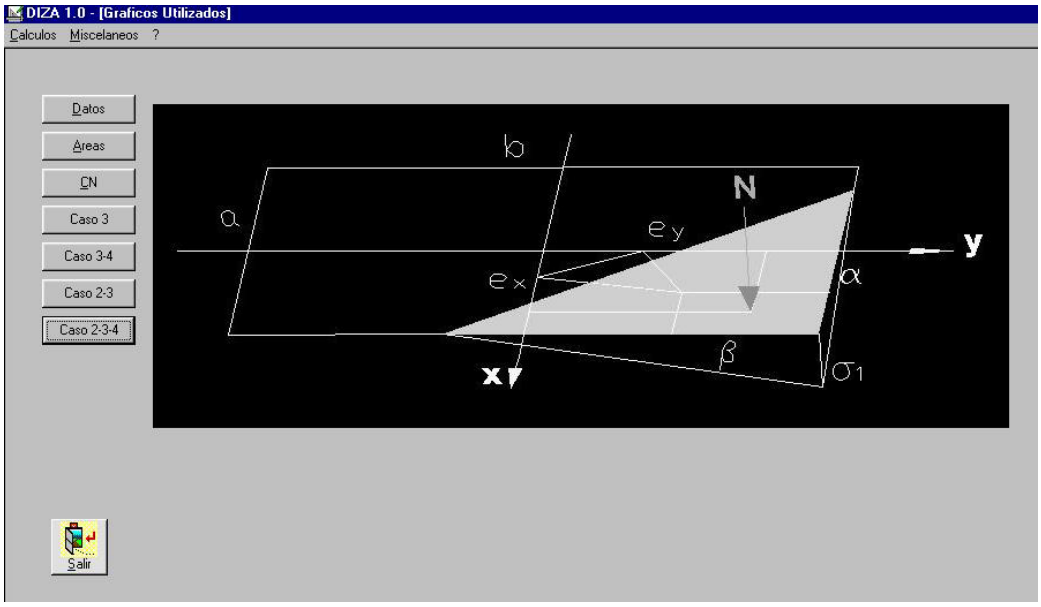
Al final de todo el usuario podrá revisar todos los resultados obtenidos en la sub opción, Resumén de Cálculos. Tal como se muestra a continuación:



Misceláneos

Esta opción permite visualizar el método de diseño empleado, explicando la problemática del diseño de zapatas rectangulares por Flexión biaxial, mostrando las formulas utilizadas y los gráficos empleados, además muestra los créditos de Diza 1.0.





Antes de instalar el programa Diza 1.0, para un correcto funcionamiento del programa, tener en cuenta lo siguiente:

Requerimientos del Sistema:

- Resolución Minima 1,024x768 pixels

- Copiar e instalar los fonts greekc y greeks en la carpeta \Windows\Fonts

- Instalar el programa en el Directorio c:\DIZA

ANEXO B

VISUAL BASIC 6.0:

VARIABLES, DEFINICIÓN Y ENTORNO UTILIZADOS EN DIZA 1.0

Basic, desde siempre, al contrario de otros sistemas de programación, no exigió la definición previa de una variable. Una variable, como Vd. seguro que conoce, es un nombre que en el programa le asignamos a un dato. Ese dato podrá cambiar. Piense por ejemplo, en un programa consistente en la toma de datos de los alumnos de un centro escolar. Existirán varias variables para poder introducir los datos de los alumnos. Estas variables pueden tener nombre tales como:

Nombre, Apellido1, Apellido2, Dirección, Teléfono, DNI

La variable Nombre tomará valores distintos según vayamos introduciendo los datos de los distintos alumnos. Es posible, que a lo largo de la ejecución del programa, esta variable *Nombre* valga:

José, Pedro, María, Luis

Decíamos que Basic no exige la definición previa de las variables. Otras herramientas exigen que se haga así. Por lo tanto es normal encontrar, en otros sistemas de programación, que un programa comienza de la siguiente forma:

Declare Nombre As String Le dice que Nombre es una sucesión de letras

Declare Apellido1 As String

Declare Apellido2 As String

Declare Dirección As String

Declare Teléfono As String Le dice que Teléfono es una sucesión de letras

Declare DNI As Número Le dice que DNI es un número

Mediante estas declaraciones, el programa sabe de que tipo de dato se trata y por tanto cómo debe trabajar con él. En otros sistemas de programación distintos de Basic, es necesario realizar esta declaración antes de introducir una variable.

Basic permite que no se declaren. Cuando a lo largo del programa le introducimos una variable nueva, asume que es una variable y que el tipo es el adecuado para el valor que le estamos introduciendo en ese momento.

Por ejemplo, si Basic encuentra estas instrucciones

DNI=50000000

Nombre ="Pedro"

Teléfono = "1234567"

entiende que *DNI, Nombre y Teléfono* son variables, que *DNI* es un número (No hemos metido su valor entre comillas), y que *Nombre* y *Teléfono* son sucesiones de caracteres alfanuméricos (su valor está entre comillas)

Esta particularidad de no necesitar declarar las variables hace que sea sencillo introducir una variable nueva. Sin embargo entraña un gran peligro. Imagínese que en un paso posterior del programa, le mandamos escribir esos tres datos anteriores con la instrucción PRINT

Print DNI

Print Nombre

Print Telwfono

Habrá observado en tercer lugar la palabra **Telwfono**, que por error ha introducido el programador. Basic interpreta que Telwfono es una variable e irá a leer en memoria el valor que tiene. No tendrá ningún valor. Por lo tanto no escribirá nada y encima no dará ningún aviso de que se ha cometido un error, por lo que la facilidad para introducir variables se paga con la posibilidad de un error.

OPTION EXPLICIT

Obliga a declarar previamente las variables que se vayan a usar. De no haberla declarado antes de usarla, el programa dará una comunicación de error.

TIPOS DE VARIABLES

Las variables pueden ser de los siguientes tipos: (El número indicado en segundo lugar indica el número de Bytes que ocupa en memoria.)

Booleana (2) Admite los valores 0 y 1, o True (verdadero) y False (falso)

Byte (1) Números enteros, en el rango de 0 a 255

Integer (2) Números enteros en el rango de -32768 a 32767

Long (4) Números enteros en el rango de -2147483648 a 2147483647

Single (4) Punto flotante, simple precisión

Doble (8) Punto flotante, doble precisión.

Currency (8) Entero, con punto decimal fijo (Típico de monedas)

String (*) Cadenas alfanuméricas de longitud variable o fija

Date (8) Fechas

Objet (4) Referencia a objetos

Variant (**) Otros tipos de datos

(*) Una variable tipo String ocupa el mismo número de bytes que caracteres tenga la cadena.

(**) Una variable tipo Variant ocupa 16 bytes si se trata de un número y 22 bytes + longitud de la cadena si se trata de un dato tipo cadena de caracteres.

Cada tipo de variable ocupa unos determinados bytes. Si no se define una variable, VB toma como tipo por defecto para la variable el tipo Variant. Este tipo ocupa mas bytes que, por ejemplo, un integer. Si el tipo de dato que vamos a introducir en una variable es un integer, y no la hemos declarado como tal, VB asumirá para esa variable que es del tipo Variant, lo que le llevará a gastar mas bytes de memoria (16) que los que necesitaría (2) si la hubiésemos declarado previamente.

Puede declarar el tipo de la variable mediante un carácter después del nombre de la variable. Esta técnica, obligatoria en Quick-Basic, está en desuso en VB. No es recomendable definir el tipo de esta forma, pues existe un serio peligro de error. De cualquier forma, eso es potestativo del programador y de sus costumbres.

Ambito de las variables.

Denominamos ámbito de una variable a las partes del programa donde esa variable está declarada. Para entenderlo mejor, veamos someramente la forma de un programa desarrollado en VB.

Un programa VB tiene uno o varios formularios. Cada formulario tiene varios controles. Tanto el formulario como cada uno de sus controles tienen una parte del programa, justamente la parte relacionada con cada uno de los eventos que pueden suceder bien al formulario o a los controles. A estas partes las habíamos llamado Procedimientos. Podemos tener procedimientos que no estén relacionados con ningún evento ocurrido al formulario o a sus controles. (Los Procedimientos que iremos insertando a lo largo de la aplicación)

Aparte de formularios y controles, un programa puede tener Módulos, y en cada uno de los módulos podemos insertar cuantos Procedimientos y Funciones queramos. La estructura de un programa VB puede ser de la siguiente forma:

Formulario1 Formulario2 Formulario3 Módulo1 Modulo2

Declaraciones Declaraciones Declaraciones Declaraciones Declaraciones

Proc. A1 Proc.B1 Proc.C1 Proc.D1 Proc.E1

Proc. A2 Proc.B2 Proc.C2 Proc.D2 Proc.E2

Proc. A3 Proc.B3 Proc.C3 Proc.D3 FunciónE1

Proc. A4 Proc.B4 Proc.C4 Proc.D4 FunciónE2

Proc. A5 Proc.BB1 Proc.CC1 FunciónE3

Proc. AA1 Proc.BB2 Proc.CC2 FunciónE4

Proc. AA2 Proc.B33 Proc.CC3 FunciónE5

Proc. AA3 Proc.CC4 FunciónE6

Si se declara una variable dentro de un procedimiento o Función, esa variable "NO SALE" del Procedimiento o Función donde se declaró. El procedimiento puede estar en un Formulario

En un Formulario, una variable puede declararse de dos formas : **Privada** o **Pública**. Para declarar una variable a nivel de formulario debe hacerse en la sección de declaraciones, que está la ventana de código ***Objeto* = *General***, ***Proc.* = *Declaraciones***. Si se declara **Privada**, esa variable se puede mover por todo el formulario, (es decir, por todos los procedimientos de todos los controles del formulario y por los Procedimientos que pudiésemos insertar en ese formulario), pero no sale de dicho formulario. Si se declara como **Pública**, esa variable puede moverse por todo el formulario, de la misma forma que lo haría declarada como Privada, y además puede ser usada desde otro Formulario o Módulo, citándola con el nombre del Formulario, seguido del nombre de la variable (**Formulario. Variable**)

En un Módulo una variable puede declararse como **Privada**, con lo que no saldrá de ese Módulo, o **Pública**, pudiendo en este caso usarse en todo el programa. Cuando se declara una variable como pública en un Módulo, basta referirse a ella por su nombre, sin citar el nombre del Módulo donde se declaró.

Tipos de declaración de variables.

Sentencia DIM Es la forma mas común de declarar una variable como Privada. Puede emplearse en un Procedimiento, Función, Formulario o Módulo. La sintaxis es de la siguiente forma:

Dim nombre variable **As Integer** (o el tipo que sea)

Declarando una variable con la sentencia DIM, en un ***formulario***, ***Función***, ***procedimiento*** o ***módulo***, el entorno de la variable será el explicado anteriormente para una variable

declarada como Privada. Es decir, esa variable no sale del formulario, procedimiento ó módulo donde se declaró. Cada vez que entremos al formulario, procedimiento o módulo, esa variable tomará el valor cero (si es numérica) o nulo (si es string).

Sentencia PRIVATE Es la forma de declarar una variable como Privada. Puede emplearse solamente en la sección de declaraciones de un Formulario o Módulo. La sintaxis es de la siguiente forma:

Private nombre variable **As** Tipo variable

Declarando una variable mediante la sentencia **PRIVATE** en un Formulario o Módulo, esa variable puede usarse en todo ese Formulario o Módulo (En todos sus Procedimientos y Funciones), pero NO fuera del Formulario o Módulo donde se declaró.

La sentencia **Private** no puede usarse en un procedimiento o función.

Sentencia PUBLIC Es la forma de declarar una variable como Pública. Puede emplearse solamente en la sección de declaraciones de un Formulario o Módulo. La sintaxis es de la siguiente forma:

Public nombre variable **As** Tipo variable

Declarando una variable de esta forma en la sección de declaraciones de un **Módulo**, esa variable puede usarse en cualquier parte del programa citándola simplemente por su nombre.

Si se declara de esta forma en la sección de declaraciones de un Formulario, esa variable puede usarse en toda el programa. Para nombrarla, si estamos en el Formulario donde se declaró basta con citarla por su nombre. Si no estamos en ese Formulario, habrá que citarla por el nombre del Formulario, seguido del nombre de la variable, separados por un punto :

Nombre Formulario. **Nombre variable**

En un Módulo puede usarse también la sentencia **Global** en vez de **Public** :

Sentencia GLOBAL Declara una variable que es válida en todo el programa. La sintaxis es:

Global nombre variable **As** tipo variable

La sentencia **Global** sólo puede usarse en el apartado de declaraciones de un **Módulo**.

Mediante la sentencia **Global** la variable puede usarse en todo el espacio del programa.

Sentencia STATIC

Como se dijo anteriormente, una variable declarada en un procedimiento pierde su valor al salir de él. Lo peor es que una vez que el programa vuelva a entrar en ese procedimiento, la variable estará puesta a cero. Afortunadamente, esto último tiene solución. Si declarásemos una variable en un procedimiento o función, como estática, esa variable, aunque no la

podremos utilizar fuera de ese procedimiento o función, cuando volvamos a él conservará el valor que tenía cuando lo abandonamos. Esta declaración como estática se realiza mediante la intrucción **Static**

Static nombre variable **As** tipo variable

El nombre de una variable puede ser tan largo como queramos. hasta un máximo de 40 caracteres. En la versión VB para España se pueden usar incluso la Ñ y vocales acentuadas. Es indiferente usar mayúscula ó minúsculas. No se sorprenda, si por ejemplo, la ha declarado con mayúsculas y luego la cita con minúsculas al escribir el código, que automáticamente se cambie a mayúsculas. El nombre de una variable siempre debe comenzar por una letra.

Lenguaje Basic del Visual Basic.

No ha sido un juego de palabras. VB emplea unas instrucciones casi iguales a las que emplea Quick Basic. Sin embargo ha añadido otras nuevas instrucciones, inherentes con la programación visual. Vamos a estudiar aquí las instrucciones y definiciones mas sencillas, comunes a QB y VB.

Sentencias condicionales.

Llamamos sentencias condicionales a aquellas que se realizan si se cumple una determinada condición. Son las sentencias por las que empieza cualquier texto de Basic, y este no va ser menos.

La sentencia condicional mas usada es:

Si se cumple una condición **Entonces**

Realiza estas instrucciones

Si no se cumple

Realiza estas otras instrucciones

Fin de la sentencia.

Así de fácil es programar en Basic. Lo que ocurre es que esta herramienta habla inglés, y lo descrito anteriormente toma la forma:

If condición **Then**

Instrucciones

Else

Otras instrucciones

End If

En este ejemplo, la condición era que, o se cumple una condición y ejecuta unas determinadas instrucciones, o no se cumple, y ejecuta otras condiciones distintas. Puede ocurrir que, caso de no cumplirse la condicione primera, se abra un abanico de dos o tres posibilidades. La sentencia condicional tendría entonces la forma:

If condición 1 **Then**

Instrucciones

ElseIf Condición 2

Otras instrucciones

ElseIf Condición 3

Otro juego de instrucciones

Else

Instrucciones que debe realizar caso de no cumplir las condiciones 1, 2 y 3.

End If

Como decíamos anteriormente, este es el tipo de sentencia condicional mas usada. Existe otra:

Select Case

Su nombre casi nos define lo que es: Selecciona, dependiendo del caso, un determinado juego de instrucciones:

Select Case variable ' variable es una variable que puede tomar los valores (p.e.) de 1 a 4

Case 1

Instrucciones a ejecutar en caso de que variable = 1

Case 2

Instrucciones a ejecutar en caso de que variable = 2

Case 3

Instrucciones a ejecutar en caso de que variable = 3

Case 4

Instrucciones a ejecutar en caso de que variable = 4

End Select

Este procedimiento resulta mucho mas sencillo y rápido que las sentencias **If Then Else**

vistas anteriormente, cuando el margen de elección es mayor que 2.

Cuando lo que queremos es elegir un valor, no ejecutar instrucciones como hacíamos anteriormente, disponemos de otras dos funciones: **Choose** y **Switch**.

Switch toma una serie de parámetros, todos por parejas. El primer término de cada pareja es la expresión a evaluar. El segundo es el valor que tiene que devolver. En realidad **Switch** es una función (las funciones las veremos muy pronto)

A = **Switch** (B=1, 5, B=2, 7, B=3, 11)

Esta instrucción obtiene un valor para A que dependerá del valor que tome B entre los valores posibles (1, 2 ó 3)

La sentencia **Choose** es casi igual, cambia solamente la forma. La misma intrucción anterior puede realizarse con **Choose** de la siguiente forma:

A = **Choose** (B, 5, 7, 11)

En estas sentencias, **Switch** y **Choose**, si el valor de B no coincide con ninguno de los valores que se le habían establecido (1, 2 ó 3 en nuestro caso), la sentencia devuelve el valor Nulo (**Null**). Esto puede producir algún error si no se contempla esa posibilidad.

Con estas sentencias condicionales es posible realizar bifurcaciones del programa, cambiar las propiedades de un objeto, obtener resultados de operaciones, ...

Sentencias de bucle.

Es muy común utilizar bucles a lo largo de un programa. Un bucle es una sucesión repetitiva de instrucciones, que se estarán realizando mientras se cumpla una condición o mientras no se cumpla otra condición. Es tan sencillo como esto:

Mientras condición

Instrucciones

Fin del bucle

Existen dos formas de bucle: Una, que realiza un número determinado de recorridos por el bucle. Es el denominado bucle por contador. Otra, realiza el bucle hasta que se cumpla (o deje de cumplirse) una condición. Es el llamado bucle por condición.

Bucle por contador

Realiza el bucle tantas veces como le indiquemos. Por ejemplo, en este bucle nos va a presentar las 26 letras mayúsculas del alfabeto inglés

For N=65 To 90

Label1.caption = Chr (N)

Next N

Este "programa" nos presentará en una caja (Label) los caracteres cuyo número ASCII vaya desde el 65 (A) al 90 (Z) Comenzará presentando el correspondiente al número 65, e irá presentando sucesivamente el 66, el 67, etc., hasta llegar al 90, donde se parará.

Bucles por condición

Ejecuta las instrucciones del bucle mientras se cumple una condición

X = 0

Do While X < 1000

X = X + 1

Loop

El programa toma una variable (X) que previamente tuvimos la curiosidad de ponerla a cero, e incrementa su valor una unidad. Analiza si el valor de X es menor que 1000, y si es cierto, vuelve a realizar el bucle. Así hasta que X ya no sea menor que 1000. Al dejar de cumplirse que X sea menor que 1000, sale del bucle. Acabamos de realizar un temporizador, y también de exponer las sentencias condicionales y los bucles, inicio de cualquier curso de Basic. Como final de lección, se propone un problema. Con el primer bucle, donde visualizábamos los caracteres A a la Z, posiblemente no nos diese tiempo de ver cada una de las letras que iban apareciendo en la pantalla, en la etiqueta Label1, dado que cambiaría con mucha velocidad, y solamente veríamos la Z, que es donde se detuvo el programa.

Si lo que queremos es que el programa se ejecute mientras no se cumpla una determinada condición, la sentencia será:

X = 0

Do Until X > 1000

X = X + 1

Loop

Observe que la diferencia entre una y otra es la condición, **While** para indicar **Mientras se cumpla que ...** y **Until** para indicar **Mientras no se cumpla que ....**

Para terminar bien el programa anterior utilizaremos la condición de **While** (Mientras se cumpla la condición)

For N=65 To 90

Label1.caption = Chr (N)

Label1.RefreshN ' Refresca la etiqueta

X = 0

Do While X < 1000

X = X + 1

Loop

Next N

ANEXO C

ALGORITMOS Y CÓDIGOS DEL SOFTWARE

El presente anexo corresponde a los algoritmos y códigos utilizados en el software Diza 1.0.

La primera parte corresponde a la comprobación de que los esfuerzos actuantes según las dimensiones de la zapata no sobrepasen la capacidad portante del suelo, tanto para cargas con sismo o sin sismo.

Cálculo sin carga de sismo
Esta parte corresponde al ingreso de datos iniciales los cuales van a ser posteriormente utilizados a lo largo del programa.

```
cond = "SS"
N = 0
Ex = 0
Ey = 0
Nm = Val(txtNm)
Nv = Val(txtNv)
Mmx = Val(txtMmx)
Mvx = Val(txtMvx)
Mmy = Val(txtMmy)
Mvy = Val(txtMvy)
Mx = 0
My = 0
Mox = 0
Moy = 0
```

La siguiente parte corresponde a la comprobación de la capacidad portante con las dimensiones de la zapata.

```
'Realizando Calculos
N = ((1 + (Pp / 100)) * Nm) + Nv      .....     Algoritmos
Mx = Mmx + Mvx
My = Mmy + Mvy
If ax > 0 And ay > 0 Then             .....     Condiciones para columna excéntrica
Mox = Mx + (N * ax)                   .....     Algoritmos de columna excéntrica
Moy = My + (N * ay)
Ex = Round(Mox / N, 5)                .....     Tipo de variable utilizada en este caso
Ey = Round(Moy / N, 5)                          redondeo de 5 decimales de aproximación.
```

```
Mx = Mox                    .....    Esto representa que en el caso de que
My = Moy                             exista momento por columna excéntrica
Else                                 se tomará este momento para los cálculos
Ex = Round(Mx / N, 5)
Ey = Round(My / N, 5)
End If
```

Calcula los esfuerzos tomando las formulas para cada uno de los casos de esfuerzo por flexión biaxial (los algoritmos se encuentran posteriormente).

```
Condicionantes
If condicionSS = "Otro" Then Exit Sub
```

Compara el esfuerzo actuante con el resistente, si el primero es menor que el segundo, seguirá corriendo el programa de lo contrario presentará el siguiente aviso, de que los esfuerzos actuantes sean menores que los resistentes, de lo contrario el programa regresa al inicio de cálculos.

```
If sigmat <= sigma1 Then
msg = MsgBox("Se recomienda aumentar las dimensiones de la zapata (a,b), " + Chr(13) +
"debido a que los esfuerzos actuantes sobrepasan la capacidad portante del suelo",
vbOKOnly + vbCritical, "Cuidado")
Exit Sub
End If
frmRptaSigmat.Show 1
```

```
 'Calculo con carga de sismo
cond = "CS"
  N = 0
  Ex = 0
  Ey = 0
  Nm = Val(txtNm)
  Nv = Val(txtNv)
  Ns = Val(txtNs)
  Mmx = Val(txtMmx)
  Mvx = Val(txtMvx)
  Msx = Val(txtMsx)
  Mmy = Val(txtMmy)
  Mvy = Val(txtMvy)
  Msy = Val(txtMsy)
  Mx = 0
  My = 0
  Mox = 0
  Moy = 0

  N = ((1 + (Pp / 100)) * Nm) + Nv + Ns
  Mx = Mmx + Mvx + Msx
  My = Mmy + Mvy + Msy
  If ax > 0 And ay > 0 Then
  Mox = Mx + (N * ax)
```

```
  Moy = My + (N * ay)
  Ex = Round(Mox / N, 5)
  Ey = Round(Moy / N, 5)
  Mx = Mox
  My = Moy
  Else
  Ex = Round(Mx / N, 5)
Ey = Round(My / N, 5)
End If

Condicionantes
If condicionCS = "Otro" Then Exit Sub
If 1.3 * sigmat <= sigma1 Then
msg = MsgBox("Se recomienda aumentar las dimensiones de la zapata (a,b), " + Chr(13) +
"debido a que los esfuerzos actuantes sobrepasan la capacidad portante del suelo",
vbOKOnly + vbCritical, "Cuidado")
Exit Sub
End If
frmRptaSigmat.Show 1
xxSW = True
End Sub
```

Private Condicionantes

A continuación las condiciones utilizadas para las diferentes regiones donde caen las cargas

'**Cn** : Cuando los valores Ex y Ey están entre 0<=Ex<=a/6 and Ey<=(-b/a)*Ex+(b/6)

```
f1 = 0
g1 = 0
f1 = a / 6
g1 = Round((-b / a) * Ex + (b / 6), 5)

If 0 <= Ex <= f1 And Ey <= g1 Then

'If (Ex >= 0 And Ex <= f1) And (Ey <= g1) Then
EcuacionCN
If xxSW = False Then Exit Sub
If cond = "SS" Then
condicionSS = "CN"
Else
condicionCS = "CN"
End If
End If
```

'2-3-4 : (a/4)<=Ex<(a/2) and (b/4)<=Ey<(b/2)

```
f1 = 0
f2 = 0
g1 = 0
```

```
g2 = 0
f1 = a / 4
f2 = a / 2
g1 = b / 4
g2 = b / 2

If (Ex >= f1 And Ex < f2) And (Ey >= g1 And Ey < g2) Then
Ecuacionp234
If xxSW = False Then Exit Sub
Corte234
If cond = "SS" Then
condicionSS = "234"
Else
condicionCS = "234"
End If

'2-3 : (a/6)<Ex<=(a/4) and 0<=Ey<(3b/a)*Ex - (b/2)  or (a/4)<Ex<(a/2) and 0<=Ey<(b/4)
f1 = 0
f2 = 0
g1 = 0
g2 = 0
g3 = 0
f1 = a / 6
f2 = a / 4
g1 = Round((((3 * b) / a) * Ex - (b / 2)), 6)
g2 = a / 2
g3 = b / 4

If ((Ex > f1 And Ex <= f2) And (Ey >= 0 And Ey < g1)) Or ((Ex > f2 And Ex < g2) And
(Ey >= 0 And Ey < g3)) Then
Ecuacionp23
If xxSW = False Then Exit Sub
CorteAll
If cond = "SS" Then
condicionSS = "23"
Else
condicionCS = "23"
End If
End If

'3-4:  0<=Ex<(a/4) and (b/(3a)*Ex + (b/6) < Ey <= (b/4) or   0<=Ex<(a/4) and
(b/4)<Ey<(b/2)
f1 = 0
f2 = 0
f3 = 0
f4 = 0
f1 = a / 4
f2 = (b / (3 * a)) * Ex + (b / 6)
f3 = b / 4
f4 = b / 2
```

```
If ((Ex >= 0 And Ex < f1) And (Ey > f2 And Ey <= f3)) Or ((Ex >= 0 And Ex < f1) And
(Ey > f3 And Ey < f4)) Then
Ecuacionp34
If xxSW = False Then Exit Sub
Corte34
If cond = "SS" Then
condicionSS = "34"
Else
condicionCS = "34"
End If
End If

'3 :    0<Ex<(a/4) and Ey>((-b/a) * Ex )+(b/6) and Ey<=((b*Ex)/(3a))+(b/6) and
Ey>=((3b*Ex)/a)-(b/2)
f1 = 0
f2 = 0
f3 = 0
f4 = 0
f1 = a / 4
f2 = Round(((-b / a) * Ex) + (b / 6), 5)
f3 = ((b * Ex) / (3 * a)) + (b / 6)
f4 = Round((((3 * b) / a) * Ex) - (b / 2), 5)

If (Ex > 0 And Ex < f1) And (Ey > f2) And (Ey <= f3) And (Ey >= f4) Then
Ecuacionp3
If xxSW = False Then Exit Sub
CorteAll
If cond = "SS" Then
condicionSS = "3"
Else
condicionCS = "3"
End If
End If

'Otherwise : a/2<=Ex<=100*a or b/2<=Ey<=100*b
f1 = 0
f2 = 0
f3 = 0
f4 = 0
f1 = a / 2
f2 = 100 * a
f3 = b / 2
f4 = 100 * b
If (Ex >= f1 And Ex <= f2) Or (Ey >= f3 And Ey <= f4) Then
msg = MsgBox("Se recomienda aumentar las dimensiones de la zapata (a,b), debido a que
los momentos son grandes", vbOKOnly + vbCritical, "Cuidado")
If cond = "SS" Then
condicionSS = "Otro"
Else
```

```
condicionCS = "Otro"
End If
End If

End Sub
```

Amplificación de cargas

A continuación la amplificación de cargas tanto para la norma ACI como la NTE060.

```
If optNTP.Value = True Then
N = (1.5 * Nm) + (1.8 * Nv)
Mx = (1.5 * Mmx) + (1.8 * Mvx)
My = (1.5 * Mmy) + (1.8 * Mvy)
Else
N = (1.4 * Nm) + (1.7 * Nv)
Mx = (1.4 * Mmx) + (1.7 * Mvx)
My = (1.4 * Mmy) + (1.7 * Mvy)
End If
Ex = Mx / N
Ey = My / N

xxMsg = "Resultados sin Carga de Sismo"

frmRptaSSismo.Show 1
```

Calculos cuando exista columna excéntrica

```
 If ax > 0 And ay > 0 Then
 Mox = Mx + (N * ax)
 Moy = My + (N * ay)
 Ex = Mox / N
 Ey = Moy / N
 Mx = Mox
 My = Moy
msg = MsgBox("N = " + Str(Format(N, "#,##0.00")) + " Ton" + Chr(13) + "Mx = " +
Str(Format(Mx, "#,##0.00")) + " Ton x m" + Chr(13) + "My = " + Str(Format(My,
"#,##0.00")) + " Ton x m" + Chr(13) + "Ex = " + Str(Format(Ex, "#,###.00")) + " m" +
Chr(13) + "Ey = " + Str(Format(Ey, "#,###.00")) + " m", vbOKOnly, "Cálculos para
columna excéntrica sin sismo")
End If
End Sub

Private Sub CalConSismo()
  N = 0
  Ex = 0
  Ey = 0
  Nm = Val(txtNm)
  Nv = Val(txtNv)
  Ns = Val(txtNs)
```

```
  Mmx = Val(txtMmx)
  Mvx = Val(txtMvx)
  Msx = Val(txtMsx)
  Mmy = Val(txtMmy)
  Mvy = Val(txtMvy)
  Msy = Val(txtMsy)
  ax = Val(txtAx)
  ay = Val(txtAy)
  Mx = 0
  My = 0
  Mox = 0
  Moy = 0

  If optNTP.Value = True Then
  N = 1.25 * (Nm + Nv + Ns)
  Mx = 1.25 * (Mmx + Mvx + Msx)
  My = 1.25 * (Mmy + Mvy + Msy)
  Else
  N = (1.05 * Nm) + (1.275 * Nv) + (1.4 * Ns)
  Mx = (1.05 * Mmx) + (1.275 * Mvx) + (1.4 * Msx)
  My = (1.05 * Mmy) + (1.275 * Mvy) + (1.4 * Msy)
  End If
  Ex = Mx / N
  Ey = My / N

  'Mostrando Valores
  xxMsg = "Resultados con Carga de Sismo"
  frmRptaSSismo.Show 1
  'msg = MsgBox("Resultados con Carga de Sismo:" + Chr(13) + "N = " + Str(N) +
Chr(13) + "Mx = " + Str(Mx) + Chr(13) + "My = " + Str(My) + Chr(13) + "Ex = " +
Str(Ex) + Chr(13) + "Ey = " + Str(Ey), vbOKOnly, "Resultados")

    If ax > 0 And ay > 0 Then
    Mox = Mx + (N * ax)
    Moy = My + (N * ay)
    Ex = Mox / N
    Ey = Moy / N
    Mx = Mox
    My = Moy
    msg = MsgBox("N = " + Str(Format(N, "#,##0.00")) + " Ton" + Chr(13) + "Mx = " +
Str(Format(Mx, "#,##0.00")) + " Ton x m" + Chr(13) + "My = " + Str(Format(My,
"#,##0.00")) + " Ton x m" + Chr(13) + "Ex = " + Str(Format(Ex, "#,###.00")) + " m" +
Chr(13) + "Ey = " + Str(Format(Ey, "#,###.00")) + " m", vbOKOnly, "Cálculos para
columna excéntrica con sismo")
  End If
End Sub
```

Cálculos para cortante y punzonamientos

A continuación los algoritmos y códigos para corte, punzonamiento según la región donde caen las cargas.

```
Private Sub CorteAll()
  dd = 0.5
  Vu = 0
  X = ((sigma1 - sigma4) * (b + D) + (2 * dd)) / (2 * b)
  Vu = ((X + sigma4 + sigma1) * (b - D - (2 * dd))) / 4
End Sub

Private Sub Corte34()
dd = 0.5
Vu = 0
X = ((sigma1 - sigma2) * (b + D) + (2 * dd)) / (2 * b)
Vu = ((X + sigma2 + sigma1) * (b - D - (2 * dd))) / 4
End Sub

Private Sub Corte234()
dd = 0.5
Vu = 0
X = (sigma1 * (beta + D) + (2 * dd)) / (2 * beta)
Vu = ((X + 0 + sigma1) * (beta - D - (2 * dd))) / 4
End Sub

Private Sub ValidaVuVn()

Dim vtft As Boolean

dd = 0.5
t = 0.5
vtft = True
dd1 = 0
dd2 = 0
Do While vtft
For dd = t To 1000 Step 0.05
If condCorteP = "234" Then
If optNTP.Value = True Then
If ((X + 0 + sigma1) * (beta - D - (2 * dd))) / 4 <= 65.28 * dd Then Exit For
Else
If ((X + 0 + sigma1) * (beta - D - (2 * dd))) / 4 <= 246.35 * dd Then Exit For
End If
ElseIf condCorteP = "34" Then
If optNTP.Value = True Then
If ((X + sigma2 + sigma1) * (b - D - (2 * dd))) / 4 <= 65.28 * dd Then Exit For
Else
If ((X + sigma2 + sigma1) * (b - D - (2 * dd))) / 4 <= 246.35 * dd Then Exit For
End If
Else
If optNTP.Value = True Then
If ((X + sigma4 + sigma1) * (b - D - (2 * dd))) / 4 <= 65.28 * dd Then Exit For
Else
If ((X + sigma4 + sigma1) * (b - D - (2 * dd))) / 4 <= 246.35 * dd Then Exit For
```

```
End If
End If
Next

If condCorteP = "234" Then
X = (sigma1 * (beta + D) + (2 * dd)) / (2 * beta)
Vu = ((X + 0 + sigma1) * (beta - D - (2 * dd))) / 4
ElseIf condCorteP = "34" Then
X = ((sigma1 - sigma2) * (b + D) + (2 * dd)) / (2 * b)
Vu = ((X + sigma2 + sigma1) * (b - D - (2 * dd))) / 4
Else
X = ((sigma1 - sigma4) * (b + D) + (2 * dd)) / (2 * b)
Vu = ((X + sigma4 + sigma1) * (b - D - (2 * dd))) / 4
End If
dd2 = dd
If dd1 = dd2 Then
vtft = False
Else
t = dd
dd1 = dd2
End If
Loop

'rpt1 = ((X + sigma4 + sigma1) * dd) / 2
rpt1 = Vu
rpt2 = 65.28 * dd
rpt1C = rpt1
rpt2C = rpt2
xxMsg = "Resultados para Corte"
frmRptasCorte.Show 1
'msg = MsgBox("Para Corte cumple que Vu > Vn con los valores : " + Chr(13) + " Vu = "
+ Str(rpt1) + Chr(13) + " Vn = " + Str(rpt2) + Chr(13) + "d = " + Str(dd), vbOKOnly,
"Resultados")
End Sub

Private Sub Punzonamiento()
pzSigma = 0
D = Val(txtD)
E = Val(txtE)

If condCorteP = "234" Then
If sigma1ss > sigma1cs Then
pzSigma = sigma1ss
Else
pzSigma = sigma1cs
End If
ElseIf condCorteP = "34" Then
If (sigma1ss + sigma2ss) / 2 > (sigma1cs + sigma2cs) / 2 Then
pzSigma = (sigma1ss + sigma2ss) / 2
Else
```

```
pzSigma = (sigma1cs + sigma2cs) / 2
End If
Else
If (sigma1ss + sigma4ss) / 2 > (sigma1cs + sigma4cs) / 2 Then
pzSigma = (sigma1ss + sigma4ss) / 2
Else
pzSigma = (sigma1cs + sigma4cs) / 2
End If
End If

If D > E Then
llc = D
lcc = E
Else
llc = E
lcc = D
End If

t = dd
For dd = t To 1000 Step 0.05
If optNTP.Value = True Then
If condCorteP = "234" Then
If (pzSigma / 2) * ((a * beta) - (D + dd) * (E + dd)) < 33.26 * (2 + 4 * (lcc / llc)) * dd *(2 *
(D + dd) + 2 * (E + dd)) Then Exit For
Else
If pzSigma * ((a * b) - (D + dd) * (E + dd)) < 33.26 * (2 + 4 * (lcc / llc)) * dd * (2 * (D +
dd) + 2 * (E + dd)) Then Exit For
End If
Else
If condCorteP = "234" Then
If (pzSigma / 2) * ((a * beta) - (D + dd) * (E + dd)) < 123.177 * (2 + 4 * (lcc / llc)) * dd *
(2 * (D + dd) + 2 * (E + dd)) Then Exit For
Else
If pzSigma * ((a * b) - (D + dd) * (E + dd)) < 123.177 * (2 + 4 * (lcc / llc)) * dd * (2 * (D
+ dd) + 2 * (E + dd)) Then Exit For
End If
End If
Next

rpt1 = pzSigma * ((a * b) - (D + dd) * (E + dd))
rpt2 = 33.26 * (2 + 4 * (lcc / llc)) * dd * (2 * (D + dd) + 2 * (E + dd))

rpt1P = rpt1
rpt2P = rpt2

xxMsg = "Resultados para Punzonamiento"
frmRptasCorte.Show 1
'msg = MsgBox("Para Punzonamiento cumple que Vu > Vn con los valores : " + Chr(13) +
" Vu = " + Str(rpt1) + Chr(13) + " Vn = " + Str(rpt2) + Chr(13) + "d = " + Str(dd),
vbOKOnly, "Resultados")
```

```
End Sub
```

Cálculos para momentos

A continuación los algoritmos y códigos para la obtención de los momentos..

```
Private Sub MomentoMuAll()
'X = ((sigma1ss - sigma4ss) * (b + D)) / (2 * b)
Muss = ((Xss + sigma4ss) * (b - D) ^ 2) / 24 + (sigma1ss * (b - D) ^ 2) / 12
Mucs = ((Xcs + sigma4cs) * (b - D) ^ 2) / 24 + (sigma1cs * (b - D) ^ 2) / 12
If Muss > Mucs Then
Mu1 = Muss
Else
Mu1 = Mucs
End If
msg = MsgBox("El valor de Mu sin Carga de Sismo es : " + Format(Muss, "#,##0.00") + "
Tn x m / m" + Chr(13) + Chr(13) + _
"El valor de Mu con Carga de Sismo es : " + Format(Mucs, "#,##0.00") + " Tn x m / m" +
Chr(13), vbOKOnly, "Resultados")
End Sub

Private Sub MomentoMu34()
'X = ((sigma1ss - sigma4ss) * (b + D)) / (2 * b)
Muss = ((Xss + sigma2ss) * (b - D) ^ 2) / 24 + (sigma1ss * (b - D) ^ 2) / 12
Mucs = ((Xcs + sigma2cs) * (b - D) ^ 2) / 24 + (sigma1cs * (b - D) ^ 2) / 12
If Muss > Mucs Then
Mu1 = Muss
Else
Mu1 = Mucs
End If
msg = MsgBox("El valor de Mu sin Carga de Sismo es : " + Format(Muss, "#,##0.00") + "
Tn x m / m" + Chr(13) + Chr(13) + _
"El valor de Mu con Carga de Sismo es : " + Format(Mucs, "#,##0.00") + " Tn x m / m" +
Chr(13), vbOKOnly, "Resultados")
End Sub

Private Sub MomentoMu234()
'X = (sigma1 * (beta + D)) / (2 * beta)
Muss = (Xss * (beta - D) ^ 2) / 24 + (sigma1ss * (beta - D) ^ 2) / 12
Mucs = (Xcs * (beta - D) ^ 2) / 24 + (sigma1cs * (beta - D) ^ 2) / 12
If Muss > Mucs Then
Mu1 = Muss
Else
Mu1 = Mucs
End If
msg = MsgBox("El valor de Mu sin Carga de Sismo es : " + Format(Muss, "#,##0.00") + "
Tn x m / m" + Chr(13) + Chr(13) + _
"El valor de Mu con Carga de Sismo es : " + Format(Mucs, "#,##0.00") + " Tn x m / m" +
Chr(13), vbOKOnly, "Resultados")
End Sub
```

Cálculos para los refuerzos

A continuación los algoritmos y códigos para el cálculo del refuerzo, para el cual se utilizará el método numérico Regular Falsi Modificado, para esto se tomará unos intervalos (a,b).

```
Private Sub Aceros()

'Valores iniciales de a, b y m
av = 0
bv = 0.04
mv = 0
sigma = 0
Fc = txtFc
For t = 1 To 100
av = mv
bv = 0.04

' Desarrollando f(a)
f_av = ((2478 * av ^ 2) / Fc) - av + (Mu1 / (37800 * dd ^ 2))

' Desarrollando f(b)
f_bv = ((2478 * bv ^ 2) / Fc) - bv + (Mu1 / (37800 * dd ^ 2))

'Calculando m
mv = ((av * f_bv) - (bv * f_av)) / (f_bv - f_av)

'msg = MsgBox("Los valores calculados son : (Bucle = " + Str(t) + ")" + Chr(13) + _
'    "av = " + Str(av) + Chr(13) + _
'    "f_av = " + Str(f_av) + Chr(13) + _
'    "bv = " + Str(bv) + Chr(13) + _
'    "f_bv = " + Str(f_bv) + Chr(13) + _
'    "mv = " + Str(mv) + Chr(13) + _
Next

sigma = mv

msg = MsgBox("El valor de la cuantia es : " + Format(sigma, "#,##0.0000"), vbOKOnly,
"Resultado")

Asx = sigma * 100 * dd * 100
```

Algoritmos para el espaciamiento de aceros de diferentes diametros

```
z0 = (0.71 * 100) / Asx 'Acero3/8

z1 = (1.29 * 100) / Asx 'Acero1/2

z2 = (2 * 100) / Asx 'Acero 5/8
```

```
z3 = (2.89 * 100) / Asx ' Acero 3/4
z4 = (5.1 * 100) / Asx 'Acero 1

msg = MsgBox("Los valores calculados son : " + Chr(13) + Chr(13) + _
"Acero 3/8' @ " + Format(z0, "#,##0") + " cm" + Chr(13) + Chr(13) + _
   "Acero 1/2' @ " + Format(z1, "#,##0") + " cm" + Chr(13) + Chr(13) + _
   "Acero 5/8' @ " + Format(z2, "#,##0") + " cm" + Chr(13) + Chr(13) + _
   "Acero 3/4' @ " + Format(z3, "#,##0") + " cm" + Chr(13) + Chr(13) + _
   "Acero   1' @ " + Format(z4, "#,##0") + " cm", vbOKOnly, "Resultados")
End Sub
```

Cálculo de esfuerzos

A continuación los algoritmos y códigos utilizados para el cálculo de esfuerzos en cada región según sus cargas.

```
Private Sub EcuacionCN()
σz = 0
'Ex = Val(txtEx)
'Ey = Val(txtEy)
'a = Val(txtA)
'b = Val(txtB)
'N = Val(txtN)
sigma1 = 0
sigma2 = 0
sigma3 = 0
sigma4 = 0

'σz = (N / (a * b)) + (6 * N * Ex) / (a * b ^ 2) + (6 * N * Ey) / (b * a ^ 2)

sigma1 = (N / (a * b)) + (6 * N * Ex) / (b * a ^ 2) + (6 * N * Ey) / (a * b ^ 2)
sigma4 = (N / (a * b)) - (6 * N * Ex) / (b * a ^ 2) - (6 * N * Ey) / (a * b ^ 2)

' Aqui adicione sigma2 y sigma3
sigma2 = (N / (a * b)) - (6 * N * Ex) / (b * a ^ 2) + (6 * N * Ey) / (a * b ^ 2)
sigma3 = (N / (a * b)) + (6 * N * Ex) / (b * a ^ 2) - (6 * N * Ey) / (a * b ^ 2)

If xxSW = False Then Exit Sub

frmRptasCN.Show 1

'msg = MsgBox("Resultados de CN es : " + Chr(13) + _
'     "Sigma1 : " + Str(sigma1) + Chr(13) + _
'     "Sigma2 : " + Str(sigma2) + Chr(13) + _
'     "Sigma3 : " + Str(sigma3) + Chr(13) + _
'     "Sigma4 : " + Str(sigma4), vbOKOnly, "Resultados")

xRpta = Chr(3)

End Sub
```

```
Private Sub Ecuacionp234()
sigma1 = 0
sigma4 = 0

alfa = 0
beta = 0
'Ex = Val(txtEx)
'Ey = Val(txtEy)
'a = Val(txtA)
'b = Val(txtB)
'N = Val(txtN)

alfa = (2 * a) - (4 * Ex)
beta = (2 * b) - (4 * Ey)

sigma1 = (6 * N) / (alfa * beta)
If xxSW = False Then Exit Sub

frmRptasCaso234.Show 1

'msg = MsgBox("El resultado de Sigma1 para 2-3-4 es : " + Str(sigma1) + Chr(13) + _
"Alfa = " + Str(alfa) + Chr(13) + _
"Beta = " + Str(beta), vbOKOnly, "Resultados")
End Sub

Private Sub Ecuacionp23()
sigma1 = 0
alfa = 0
beta = 0
delta = 0
Mu = 0
sigma4 = 0
gamma = 0

'Ex = Val(txtEx)
If Ey = 0 Then
   Ey = 0.0001
End If

'a = Val(txtA)
'b = Val(txtB)
'N = Val(txtN)

Mu = 2 - ((4 * Ey) / b)

delta = (8 - 3 * Mu + (12 * Mu - 3 * Mu ^ 2 - 8) ^ (1 / 2)) / (12 - 6 * Mu)

alfa = 4 * delta * ((a / 2) - Ex) * ((delta ^ 3 - (delta - 1) ^ 3) / (delta ^ 4 - (delta - 1) ^ 4))

beta = delta * b
```

```
sigma1 = (6 * N * delta ^ 2) / (alfa * b * (delta ^ 3 - (delta - 1) ^ 3))

sigma4 = sigma1 * (1 - (1 / delta))

gamma = alfa * (1 - (1 / delta))

If xxSW = False Then Exit Sub 'Ultima modificacion

muestra_areas

muestra_2y3

frmRptasCasos23.Show 1

' Todos los resultados
'msg = MsgBox("Los resultados para ecuaciones 2 y 3 son : " + Chr(13) + _
'    "Mu = " + Str(Mu) + Chr(13) + _
'    "Delta = " + Str(Delta) + Chr(13) + _
'    "Alfa = " + Str(alfa) + Chr(13) + _
'    "Beta = " + Str(beta) + Chr(13) + _
'    "Sigma1 = " + Str(sigma1) + Chr(13) + _
'    "Sigma4 = " + Str(sigma4) + Chr(13) + _
'    "Gamma = " + Str(Gamma), vbOKOnly, "Resultados")

End Sub

Private Sub Ecuacionp34()
sigma1 = 0
sigma2 = 0
alfa = 0
beta = 0
delta = 0
Mu = 0
gamma = 0

If Ex = 0 Then
  Ex = 0.0001
End If

'Ey = Val(txtEy)
'a = Val(txtA)
'b = Val(txtB)
'N = Val(txtN)

Mu = 2 - ((4 * Ex) / a)

delta = (8 - 3 * Mu + (12 * Mu - 3 * Mu ^ 2 - 8) ^ (1 / 2)) / (12 - 6 * Mu)

alfa = delta * a
```

```
beta = 4 * delta * ((b / 2) - Ey) * ((delta ^ 3 - (delta - 1) ^ 3) / (delta ^ 4 - (delta - 1) ^ 4))

sigma1 = (6 * N * (delta ^ 2)) / (beta * a * (delta ^ 3 - (delta - 1) ^ 3))
sigma2 = sigma1 * (1 - (1 / delta))

gamma = beta * (1 - (1 / delta))

If xxSW = False Then Exit Sub

frmRptasCasos34.Show 1

' Todos los resultados
'msg = MsgBox("Los resultados para ecuaciones 3 y 4 son : " + Chr(13) + _
'    "Mu = " + Str(Mu) + Chr(13) + _
'    "Alfa = " + Str(alfa) + Chr(13) + _
'    "Beta = " + Str(beta) + Chr(13) + _
'    "Delta = " + Str(Delta) + Chr(13) + _
'    "Sigma1 = " + Str(sigma1) + Chr(13) + _
'    "Sigma2 = " + Str(sigma2) + Chr(13) + _
'    "Gamma = " + Str(Gamma), vbOKOnly, "Resultados")

End Sub
```

Private Sub Ecuacionp3()

Se utilizará el método numérico Regular Falsi Modiicado cuyo inicial es (av,bv).

```
'Ex = Val(txtEx)
'Ey = Val(txtEy)
'a = Val(txtA)
'b = Val(txtB)
'N = Val(txtN)

A1 = 0
c = 0
t = 0
Mu = 0
alfa = 0
sigma1 = 0
sigma2 = 0
sigma3 = 0
sigma4 = 0
v = 0

A1 = 4 * ((Ex / a) + (Ey / b))

c = (b * Ex) / (a * Ey)

'Valores iniciales de a, b y m
```

```
av = 0.01
bv = 0.7
mv = 0.01
vv = 0

For t = 1 To 100
av = mv
bv = 0.7

' Desarrollando f(a)
f_av = ((av ^ 9) * -((c + 1) ^ 2)) + ((av ^ 8) * ((c * (A1 + 4) ^ 2) + (2 * ((c - 1) ^ 2) * (A1 +
4)) - (4 * ((c - 1) ^ 2)))) + ((av ^ 7) * ((-12 * A1 * (A1 + 4) * c) - (12 * ((c - 1) ^ 2) * A1) +
(6 * ((c + 1) ^ 2) * A1))) + ((av ^ 6) * ((36 * (A1 ^ 2) * c) - (2 * ((A1 + 4) ^ 2) * ((c ^ 2) +
1)) + (2 * ((c - 1) ^ 2) * (A1 + 4) * (4 - (3 * A1))) + (8 * ((c - 1) ^ 2) * ((3 * A1) - 2)))) +
((av ^ 5) * ((24 * A1 * (A1 + 4) * ((c ^ 2) + 1)) + (12 * ((c - 1) ^ 2) * A1 * ((3 * A1) - 4)) -
(((c + 1) ^ 2) * (((3 * A1) - 2) * ((3 * A1) + 6) + 4)))) + ((av ^ 4) * ((-72 * (A1 ^ 2) * ((c ^
2) + 1)) + (4 * ((A1 + 4) ^ 2) * c) + (4 * ((c - 1) ^ 2) * (A1 + 4) * (2 - (3 * A1))) - (4 * ((c -
1) ^ 2) * ((3 * A1) - 2) ^ 2))) + ((av ^ 3) * ((-48 * A1 * (A1 + 4) * c) + (12 * ((c - 1) ^ 2) *
A1 * ((6 * A1) - 4)) + (12 * A1 * ((3 * A1) - 2) * (c + 1) ^ 2))) + ((av ^ 2) * (144 * (A1 ^
2) * c)) - (av * (4 * ((c + 1) ^ 2) * (((3 * A1) - 2) ^ 2)))

' Desarrollando f(b)

f_bv = ((bv ^ 9) * -((c + 1) ^ 2)) + ((bv ^ 8) * ((c * (A1 + 4) ^ 2) + (2 * ((c - 1) ^ 2) * (A1 +
4)) - (4 * ((c - 1) ^ 2)))) + ((bv ^ 7) * ((-12 * A1 * (A1 + 4) * c) - (12 * ((c - 1) ^ 2) * A1) +
(6 * ((c + 1) ^ 2) * A1))) +  ((bv ^ 6) * ((36 * (A1 ^ 2) * c) - (2 * ((A1 + 4) ^ 2) * ((c ^ 2) +
1)) + (2 * ((c - 1) ^ 2) * (A1 + 4) * (4 - (3 * A1))) + (8 * ((c - 1) ^ 2) * ((3 * A1) - 2)))) +
((bv ^ 5) * ((24 * A1 * (A1 + 4) * ((c ^ 2) + 1)) + (12 * ((c - 1) ^ 2) * A1 * ((3 * A1) - 4)) -
(((c + 1) ^ 2) * (((3 * A1) - 2) * ((3 * A1) + 6) + 4)))) +  ((bv ^ 4) * ((-72 * (A1 ^ 2) * ((c ^
2) + 1)) + (4 * ((A1 + 4) ^ 2) * c) + (4 * ((c - 1) ^ 2) * (A1 + 4) * (2 - (3 * A1))) - (4 * ((c -
1) ^ 2) * ((3 * A1) - 2) ^ 2))) +  ((bv ^ 3) * ((-48 * A1 * (A1 + 4) * c) + (12 * ((c - 1) ^ 2) *
A1 * ((6 * A1) - 4)) + (12 * A1 * ((3 * A1) - 2) * (c + 1) ^ 2))) + ((bv ^ 2) * (144 * (A1 ^
2) * c)) - (bv * (4 * ((c + 1) ^ 2) * (((3 * A1) - 2) ^ 2)))

'Calculando m
mv = ((av * f_bv) - (bv * f_av)) / (f_bv - f_av)

' Desarrollando f(m)
f_mv = ((mv ^ 9) * -((c + 1) ^ 2)) + ((mv ^ 8) * ((c * (A1 + 4) ^ 2) + (2 * ((c - 1) ^ 2) * (A1
+ 4)) - (4 * ((c - 1) ^ 2)))) + ((mv ^ 7) * ((-12 * A1 * (A1 + 4) * c) - (12 * ((c - 1) ^ 2) *
A1) + (6 * ((c + 1) ^ 2) * A1))) + ((mv ^ 6) * ((36 * (A1 ^ 2) * c) - (2 * ((A1 + 4) ^ 2) * ((c
^ 2) + 1)) + (2 * ((c - 1) ^ 2) * (A1 + 4) * (4 - (3 * A1))) + (8 * ((c - 1) ^ 2) * ((3 * A1) -
2)))) +  ((mv ^ 5) * ((24 * A1 * (A1 + 4) * ((c ^ 2) + 1)) + (12 * ((c - 1) ^ 2) * A1 * ((3 *
A1) - 4)) - (((c + 1) ^ 2) * (((3 * A1) - 2) * ((3 * A1) + 6) + 4)))) + ((mv ^ 4) * ((-72 * (A1
^ 2) * ((c ^ 2) + 1)) + (4 * ((A1 + 4) ^ 2) * c) + (4 * ((c - 1) ^ 2) * (A1 + 4) * (2 - (3 * A1)))
- (4 * ((c - 1) ^ 2) * ((3 * A1) - 2) ^ 2))) + ((mv ^ 3) * ((-48 * A1 * (A1 + 4) * c) + (12 *
((c - 1) ^ 2) * A1 * ((6 * A1) - 4)) + (12 * A1 * ((3 * A1) - 2) * (c + 1) ^ 2))) +       ((mv ^
2) * (144 * (A1 ^ 2) * c)) - (mv * (4 * ((c + 1) ^ 2) * (((3 * A1) - 2) ^ 2)))

'msg = MsgBox("Los valores calculados son : (Bucle = " + Str(t) + ")" + Chr(13) + _
```

```
'    "av = " + Str(av) + Chr(13) + _
'    "f_av = " + Str(f_av) + Chr(13) + _
'    "bv = " + Str(bv) + Chr(13) + _
'    "f_bv = " + Str(f_bv) + Chr(13) + _
'    "mv = " + Str(mv) + Chr(13) + _
'    "f_mv = " + Str(f_mv), vbOKOnly, "Resultados")

Next

v = mv

Mu = v * ((6 * A1) - ((A1 + 4) * v)) / (3 * A1 - 2 - v ^ 2)

alfa = ((Mu * (c * (v ^ 2) - 2)) - (2 * v ^ 2 * (c - 1))) / ((c + 1) * ((v ^ 2) - 2))

beta = ((Mu * ((v ^ 2) - (2 * c))) + (2 * (v ^ 2) * (c - 1))) / ((c + 1) * ((v ^ 2) - 2))

sigma3 = (2 * N * alfa * beta) / (a * b * ((alfa + beta) - (2 * alfa * beta) + ((alfa ^ 2 * beta ^
2) / 3)))
sigma1 = sigma3 * ((1 / alfa) + (1 / beta) - 1)
sigma2 = sigma3 * ((1 / beta) - 1)
sigma4 = sigma3 * ((1 / alfa) - 1)

'msg = MsgBox("Los resultados para ecuaciones 3 son : " + Chr(13) + _
'    "v = " + Str(v) + Chr(13) + _
'    "u = " + Str(Mu) + Chr(13) + _
'    "Alfa = " + Str(alfa) + Chr(13) + _
'    "Beta = " + Str(beta) + Chr(13) + _
'    "Sigma1 = " + Str(sigma1) + Chr(13) + _
'    "Sigma2 = " + Str(sigma2) + Chr(13) + _
'    "Sigma3 = " + Str(sigma3) + Chr(13) + _
'    "Sigma4 = " + Str(sigma4), vbOKOnly, "Resultados")

If xxSW = False Then Exit Sub

frmRptasCaso3.Show 1

'msg = MsgBox("Los resultados para ecuaciones 3 son : " + Chr(13) + _
'    "v = " + Str(v) + Chr(13) + _
'    "u = " + Str(Mu) + Chr(13) + _
'    "Alfa = " + Str(alfa) + Chr(13) + _
'    "Beta = " + Str(beta) + Chr(13) + _
'    "Sigma1 = " + Str(sigma1) + Chr(13) + _
'    "Sigma2 = " + Str(sigma2) + Chr(13) + _
'    "Sigma3 = " + Str(sigma3) + Chr(13) + _
'    "Sigma4 = " + Str(sigma4), vbOKOnly, "Resultados")
```