



Documentación del programa: Wallter

Francisco Rivera Álvarez
CN 034

<https://www.github.com/franterminator/Wallter>

16/05/2017

Archivos generados: result/resultados.html

Índice general

Capítulos	Página
I Documentación del programa	3
1. Descripción del programa	4
1.1. Herramientas	5
1.1.1. Lenguaje usado	5
1.1.2. Compilador	5
1.1.3. IDE (entorno de desarrollo del programa)	5
1.1.4. Debugger	6
1.1.5. Controlador de versiones Git	6
1.2. Estructura de carpetas	7
1.2.1. config (directorio)	8
1.2.2. doc (directorio)	8
1.2.3. result (directorio)	8
1.2.4. Numerico.bcp (archivo)	9
1.2.5. README.org (archivo)	9
1.2.6. archivos en fortran	9
1.3. Limitaciones del programa	10
2. Ejecución	11
2.1. Ejecutar el programa, paso a paso	11
2.2. Ejecución mediante scripts	14
2.3. Ejecución manual	15
2.3.1. Ayuda	16
2.3.2. Imprimir resultados de factorización	16
2.3.3. Archivos de configuración	17
2.4. Documentación FORD	18
3. Funcionamiento interno del programa	20
3.1. Subrutinas	20
3.2. Pseudocódigo	21
3.3. Flujo del programa	23



Capítulos	Página
4. Métodos de cálculo	24
4.1. Cálculo numérico por diferencias	24
4.2. Matriz	26
4.3. Cálculo analítico de Navier	27
4.3.1. Placa apoyada con carga de agua hasta la mitad	29
5. Resultados	30
5.1. Como se muestran los resultados	31
5.2. Precisión	32
5.3. Importancia de las variables	37
6. Conclusión	43
II Código del programa	44

Parte I

Documentación del programa

Capítulo 1

Descripción del programa

El programa Wallter esta diseñado para el cálculo de flechas de una placa apoyada en todo su contorno con agua hasta la mitad de su largo.

El nombre del programa, Wallter, es la combinación de water + wall. Describe el problema que resuelve este programa, el cálculo de flechas de una placa con agua.

Las flechas son las deformaciones de la placa, la respuesta del material frente a las solicitudes. Una placa siempre se deforma frente a cualquier esfuerzo. Pero hay que analizar si esta deformación es "aceptable", es decir, no impide que la estructura deje de ser funcional o segura.

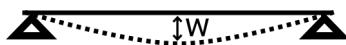


Figura 1.1: Flecha de una viga

Con este programa el ingeniero solo se tiene que preocuparse de tomar la decisión de si los valores de placa son los deseados. Las cuentas las hace el programa.

Para más precisión el programa usa dos modos de cálculo. Un método numérico y un método analítico. Ambos son más precisos cuantas más iteraciones se realicen.

Al estar programado en FORTRAN. La ejecución es muy rápida, por lo que se puede aumentar el número de iteraciones necesarias para conseguir precisión sin repercutir mucho en los tiempos de cálculo.

El resultado final es un programa funcional y útil. Un solo archivo que simplifica el tedioso cálculo de flechas a un par de porrazos de teclado.



1.1. Herramientas

1.1.1. Lenguaje usado

Se ha escrito en FORTRAN 95, aunque la mayor parte del código es compatible con FORTRAN 77.

Se ha preferido este lenguaje ya que está preparado para escribir código matemático. Internamente ya tiene integradas las funciones básicas de los cálculos elementales: funciones trigonométricas directas e inversas, potencias, operaciones aritméticas... En otros lenguajes estos recursos suelen ser importados de diferentes librerías, lo cual es más incomodo a la hora de programar.

FORTRAN 95 cuenta con las funciones allocate y allocatable. Estos métodos permiten manejar matrices dinámicas, por lo que el ejecutable solo usa el almacenamiento necesario para los cálculos. No hace falta reservar espacio adicional para aceptar diferentes tamaños de matriz.

1.1.2. Compilador

El programa se ha compilado en un ordenador de 64 bits con el sistema operativo Windows 10 con el compilador gcc en su versión 5.3.0. La compilación se ha optimizado hasta un nivel 2.

Para compilar el programa, se puede usar la siguiente línea de comando:

```
gfortran result.f95 main.f95 -o2 -o Wallter.exe
```

Se recomienda que el archivo ejecutable esté junto a la carpeta *result*, la cual contiene los archivos necesarios para una correcta visualización de este archivo. El programa al ser llamada desde la raíz detectará la carpeta.

1.1.3. IDE (entorno de desarrollo del programa)

Para programar el archivo se ha utilizado el programa Code::Blocks en su versión 16.01. Este es un programa pensado para programar en C/C++ pero acepta varios lenguajes como fortran.

El proyecto de Code::Blocks contiene la configuración de compilación y ejecución. Este archivo se incluye junto al programa por si el usuario necesita modificar el programa.



1.1.4. Debugger

Durante la implementación del código, se usó el debugger GDB (incluido en la instalación del compilador). Junto al programa no se incluyen los test que se usaron del programa.

El proyecto de Code::Blocks contiene la configuración del debugger. Si se necesitara usar el GDB de forma manual se recomienda compilar el programa, con las siguientes opciones:

```
gfortran result.f95 main.f95 -g -o Wallter.exe
```

El siguiente comando inicia el debugger para el programa Wallter.

```
gdb Wallter.exe
```

1.1.5. Controlador de versiones Git

El programa se ha implementado como un repositorio Git. De esta manera es más fácil programar diferentes versiones funcionales del programa, que se van mejorando siempre partiendo de un paso anterior (commit).

Para acceder a la configuración local de las carpetas y archivos del repositorio, hay que entrar en la carpeta .git. La cual está oculta por defecto.

Instalando git en el sistema operativo que se esté usando se puede ver el historial de commits. Para ello se debe ejecutar el siguiente comando:

```
git log --pretty=format:'%h %s' --graph
```

Para el desarrollo de este proyecto se usó el programa [GitKraken](#), el cual permite manejar git mediante una interfaz gráfica.

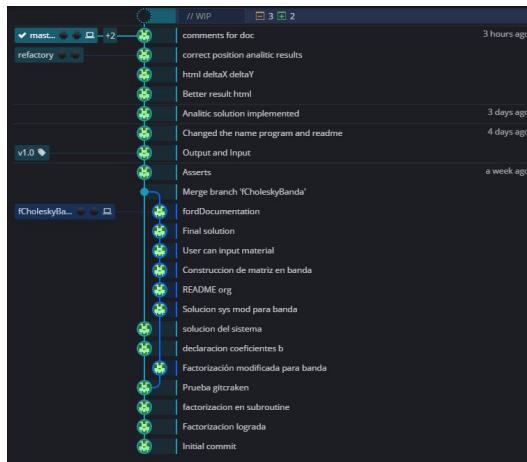
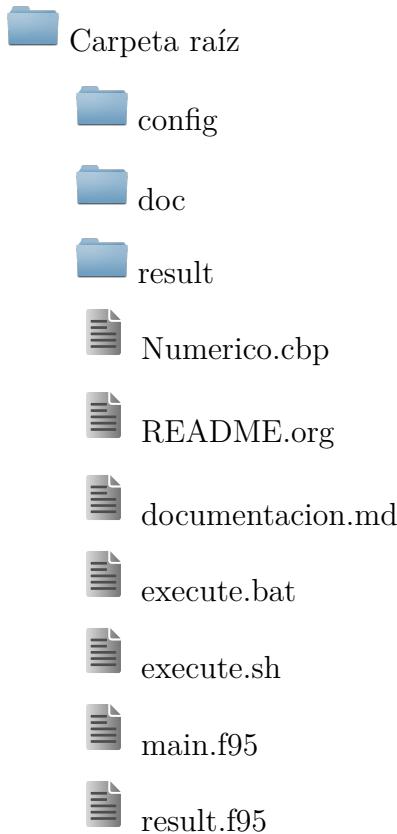


Figura 1.2: Historial de commits del repositorio en GitKraken



1.2. Estructura de carpetas



Esta es la estructura de carpetas del programa. Cualquiera de estos archivos puede ser modificado por el usuario. Los programas recomendados para ello se encuentran en el apartado 1.1. En cualquier caso, no se recomienda mover los archivos de carpeta.

A la hora de ejecutar el programa, se debe ejecutar desde dentro de la carpeta de la raíz. Si se ejecuta dentro de alguna subcarpeta (por ejemplo *doc*) el archivo de resultados se creará al lado del programa. En ese caso la página web no se verá correctamente porque no encuentra los archivos **.css* y **.js* necesarios. Para corregirlo se puede mover el html hasta la carpeta *result*.

A continuación se detallan los diferentes directorios (archivos y carpetas) que componen el proyecto.



1.2.1. config (directorio)

En esta carpeta se encuentran varios archivos de configuración. Si se usan los scripts para lanzar Wallter.exe entonces, los archivos de configuración deben estar en la carpeta. Pero si se ejecuta manualmente el programa, no tienen porque guardarse las configuraciones en una carpeta específica.

Dentro de la carpeta está el archivo *material.html* que sirven para generar los archivos de configuración. Para más detalles de como se usan los archivos de configuración ver el apartado [2.3.3.](#)

1.2.2. doc (directorio)

En esta carpeta se guardan los archivos de documentación. La documentación se ha generado usando el programa [FORD](#).

La documentación está en html, por lo tanto, se recomienda abrir los archivo sin moverlos de las carpetas para que no se pierdan las referencias a los archivos css y js.

El archivo principal es el *index.html*. Este contiene la documentación técnica del programa: argumentos del programa, variables, prototipos de las subrutinas...

La documentación se puede encontrar en línea en la siguiente dirección <http://franterminator.com/fortran/Wallter/doc/index.html>.

El archivo *documentacion.md* se usa para generar la documentación. Contiene la configuración necesaria para generar los archivos usando FORD. Para más detalles de como usar la documentación y generarla ver el apartado [2.4.](#)

1.2.3. result (directorio)

En la carpeta hay varios archivos **.css* y **.js* que facilitan el manejo y la visualización de los resultados. También se encuentra un archivo html ejemplo de como se verían las flechas.

En este carpeta se genera por defecto los resultados del programa Wallter. Para que Wallter encuentre la carpeta se debe ejecutar en la carpeta raíz del proyecto.

Se explica como usar el archivo de resultados en el capítulo [5](#).



Figura 1.3: Ejemplo del archivo de resultados

1.2.4. Numerico.bcp (archivo)

Este es el archivo de configuración del proyecto para el programa Code::Blocks. El programa es libre y se puede descargar en la siguiente página: <http://www.codeblocks.org/downloads>.

1.2.5. README.org (archivo)

Se ha escrito en Emacs y contiene la información básica de como ejecutar y usar el programa, una versión resumida de este documento.

Está en org-mode. Para poder visualizarlo se puede abrir con cualquier editor de texto. Aún así existen editores online de org-mode, por ejemplo [org.js](#).

1.2.6. archivos en fortran

El archivo principal de Wallter es el *main.f95*. Contiene la resolución analítica y numérica del problema. Las subrutinas para la generación del archivo **.html* de resultados están en *result.f95*

La compilación del programa se detalla en el apartado [1.1](#).



1.3. Limitaciones del programa

Wallter es una aplicación de consola. Solo puede entender los datos que escriba el usuario si los escribe de la manera indicada por el programa. Escribir una letra, cuando el programa pide un número provoca un fallo en tiempo de ejecución.

Los programas de consola tampoco son muy cómodos de usar. Al no tener interfaz gráfica los usuarios tienen mas dificultades para correr el programa y usarlo de forma adecuada.

Otra limitación son los errores de maquina. Los errores de truncamiento y redondeo impiden al programa conseguir una solución exacta.

Capítulo 2

Ejecución

El programa se ejecuta mediante la consola o shell. No tiene ninguna interfaz gráfica implementada. Esto quiere decir que se controla mediante el teclado.

Para ejecutarlo hay dos scripts dentro de la carpeta del proyecto: *execute.sh* y *execute.bat*. Estos programas permiten ejecutar el programa con sus diferentes opciones.

La ejecución más simple es llamar al programa desde la consola. También se puede lanzar mediante el ratón. De esta manera el programa se ejecutará con las opciones por defecto. Si se configura el sistema operativo también se pueden ejecutar los scripts con un click y desde ahí usar el teclado para ejecutar el programa.

Al estar escrito en FORTRAN se necesitan las librerías de FORTRAN para poder ejecutar o compilar. Se pueden descargar en:

- [Mingw](#)
- [Cygwin](#)

A veces saltan los antivirus al ejecutar los scripts o el programa principal. Los programas estan libres de virus ([VirusTotal](#)). Es recomendable añadir la excepción de los scripts y el programa Wallter al antivirus para un correcto funcionamiento del programa.

2.1. Ejecutar el programa, paso a paso

Al ejecutar el programa se presenta un ventana de bienvenida. Debemos pulsar Enter para continuar.



2.2.1. Ejecutar el programa, paso a paso

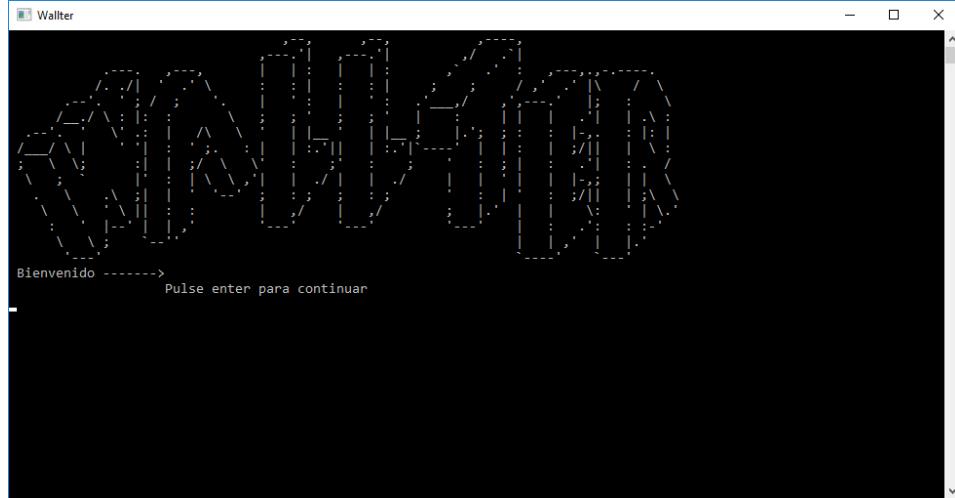


Figura 2.1: Mensaje de bienvenida

El programa necesita los datos técnicos de la placa y el material para poder trabajar. Se necesitan las medidas de placa (ancho, alto y espesor), modulo de Young y coeficiente de Poisson del material que conforma la placa. También el número de puntos en que quiere dividir la placa para los cálculos numéricos y las iteraciones que desea para los cálculos analíticos. Son necesarios todos los datos para poder obtener los resultados.

Las unidades en que deben de estar los datos los indica el programa. No se puede cambiar las unidades que acepta el programa. Si no se usan las unidades que el programa indica, los resultados no tendrán sentido.

Introduzca el dato que le pide el programa y pulse la tecla enter para validar. Haga este paso una y otra vez para introducir los.

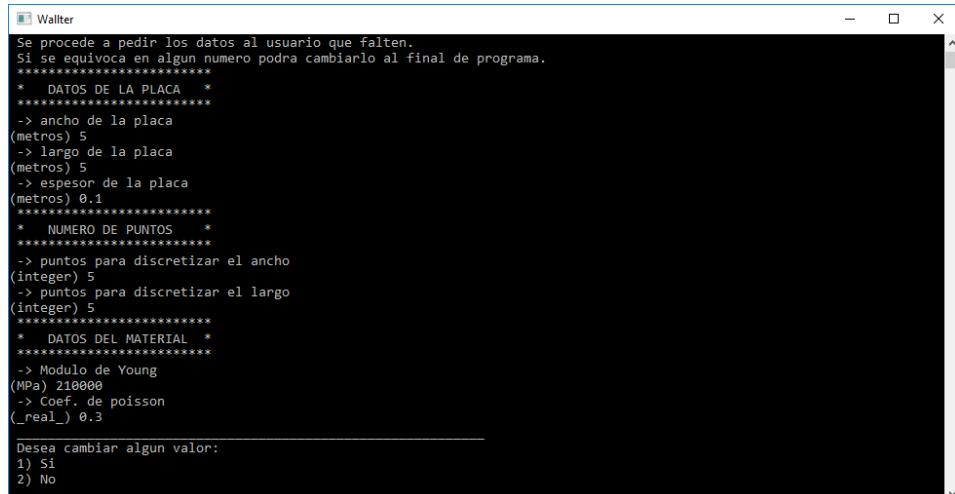


Figura 2.2: El programa lee los datos del usuario



2.2.1. Ejecutar el programa, paso a paso

Si se equivoca al escribir el valor de alguna variable no pulse enter. Borre el valor con la tecla retroceso y a continuación escriba el nuevo valor.

En el caso de que pulse enter, no se preocupe porque tiene una segunda oportunidad para corregir el fallo. Cuando acabe de introducir el resto de datos, el programa le preguntará si desea cambiar algún valor y que valor quiere modificar. Puede hacerlo tantas veces como desee.

```
Walter
-> puntos para discretizar el ancho
(integer) 5
-> puntos para discretizar el largo
(integer) 5
*****
* DATOS DEL MATERIAL *
*****
-> Modulo de Young
(MPa) 210000
-> Coef. de poisson
(_real_) 0.3

Desea cambiar algun valor:
1) Sí
2) No
1 Que valor desea cambiar?
1) ancho
2) largo
3) espesor
4) n
5) m
6) Young
7) poisson
6
*****
* DATOS DEL MATERIAL *
*****
-> Modulo de Young
(MPa) 208000
```

Figura 2.3: Menú para cambiar valores

A continuación se muestran los datos que usa el programa y las unidades en las que deben introducirse.

Archivos de configuracion		
Id	Variable	Unidades
1	ancho::	Metros (m)
2	largo::	Metros (m)
3	espesor::	Metros (m)
4	n::	Un número entero (integer)
5	m::	Un número entero (integer)
6	young::	MegaPascal (MPa)
7	poisson::	Un número decimal (double)
8	r::	Un número entero (integer)
9	s::	Un número entero (integer)

Cuadro 2.1: Tabla con las unidades

Cuando acabe de introducir los datos para el cálculo, el programa le mostrará la solución numérica en pantalla. Luego, le pedirá la información que falta para calcular la analítica.



2.2.2. Ejecución mediante scripts

```
Walter
5) m
6) Young
7) poisson
6
*****
* DATOS DEL MATERIAL *
*****
-> Modulo de Young
(MPa) 208000

Desea cambiar algun valor:
1) Si
2) No
2
[A,B,C] -> [ -5.76 1.44 1.44 ]
Solucion numerica
.0002 .0004 .0004 .0004 .0002
.0003 .0005 .0006 .0005 .0003
.0003 .0005 .0005 .0005 .0003
.0002 .0003 .0004 .0003 .0002
.0001 .0002 .0002 .0002 .0001
*****
* NUMERO DE ITERACIONES *
*****
-> Numero de iteraciones para el calculo analitico:
r (integer): 20
s (integer): 20
```

Figura 2.4: Solución numérica. A continuación, el programa pedirá los datos de la analítica.

Finalmente se imprimirán en pantalla los resultados analíticos. Recuerde que puede ver todos los resultados que ha generado el programa. Están almacenados en el archivo *resultados.html* en la carpeta result.

```
Walter
Solucion numerica
.0002 .0004 .0004 .0004 .0002
.0003 .0005 .0006 .0005 .0003
.0003 .0005 .0005 .0005 .0003
.0002 .0003 .0004 .0003 .0002
.0001 .0002 .0002 .0002 .0001
*****
* NUMERO DE ITERACIONES *
*****
-> Numero de iteraciones para el calculo analitico:
r (integer): 20
s (integer): 20

Desea cambiar algun valor:
1) Si
2) No
2
Solucion analitica
.0002 .0004 .0004 .0004 .0002
.0003 .0006 .0006 .0006 .0003
.0003 .0005 .0006 .0005 .0003
.0002 .0004 .0004 .0004 .0002
.0001 .0002 .0002 .0002 .0001
Gracias por usar el programa...
```

Figura 2.5: Fin del programa

2.2. Ejecución mediante scripts

El script *execute.sh* lanza el programa en Linux. El script *execute.bat* solo se puede usar en Windows. Se explicará como usar el script para Windows. Ambos scripts son muy parecidos por lo que el script de Linux se usa de forma similar.



Al arrancar *execute.bat* saldrá el siguiente menú:

```
.....  
Selecciona el modo de ejecucion  
.....  
  
1 - Ejecucion por defecto  
2 - Imprimir resultados factorizacion  
3 - Usar archivo de configuracion  
4 - EXIT  
  
Escribe 1, 2, 3 o 4 y pulsa ENTER:
```

Figura 2.6: Menú principal del script

Escribiendo un número de 1 a 4 y pulsando enter se arranca el archivo Wallter con la opción escogida. La opción 1 carga las opciones por defecto. La opción 2 obliga a imprimir los resultados y pasos intermedios de la factorización. Si se escribe el 3 y se pulsa enter, se abre un nuevo menú donde se puede escoger un archivo de configuración. Con el número 4 se sale del script.

```
.....  
Selecciona el modo de ejecucion  
.....  
  
1 - Ejecucion por defecto  
2 - Imprimir resultados factorizacion  
3 - Usar archivo de configuracion  
4 - EXIT  
  
Escribe 1, 2, 3 o 4 y pulsa ENTER: 3  
1 - config.txt  
numero archivo:1  
Escribe 1, 2, 3 o 4 y pulsa ENTER:
```

Figura 2.7: Nuevo menú para la opción 3

2.3. Ejecución manual

En la carpeta raíz está el programa Wallter.exe. Se ha compilado con el gcc en su versión 5.3.0 en un ordenador con Windows 10. Si este binario no funciona, se puede compilar siguiendo las instrucciones del subapartado compilador [1.1.2](#).



Si se prefiere ejecutar de manera manual el programa, se puede usar el siguiente comando:

En windows: Wallter.exe

En linux: ./Wallter.exe

A parte el programa acepta varias opciones avanzadas. Las cuales se detallan a continuación.

2.3.1. Ayuda

La ayuda imprime un resumen de estas opciones y sale del programa. Para ver la ayuda ejecutar el comando:

Wallter.exe -h

```
command >wallter.exe -h
Las opciones disponibles son:
-h para ver esta ayuda
-a para activar la impresion de los datos de factorizacion
-c [FILENAME] para usar un programa de configuracion con los datos

command >
```

Figura 2.8: Ayuda del programa

2.3.2. Imprimir resultados de factorización

El programa no solo mostrará las flechas obtenidas por ambos métodos, también imprimirá los resultados intermedios de la factorización y resolución del sistema. Para ello se debe escribir lo siguiente:

Wallter.exe -a

```
command >wallter.exe -a
Se ha activado la impresion de los datos de factorizacion.
```

Figura 2.9: Arranque del programa con la opción asserts activada



2.3.3. Archivos de configuración

Los archivos de configuración permiten un arranque muy cómodo del programa. Contienen los parámetros que se usan luego en los cálculos de flecha.

Si el programa se ejecuta sin esta opción se piden los datos necesarios para los cálculos. Pero al activar los archivos de configuración txt, el programa tomará los datos del txt. Si falta algún dato por definir en la configuración, se le pedirá al usuario.

Otra diferencia con el resto de modos de arranque es que el programa considera los datos como correctos. **No deja al usuario cambiar los datos una vez ejecutado el programa**. La idea es que el usuario puede cambiar los datos en los archivos de configuración y estos pueden ser comprobados antes de arrancar el archivo Wallter.exe.

De esta manera se pierde menos tiempo en la ejecución, ya que no es necesario pedir los datos al usuario. Obteniendo los resultados en tiempo récord.

Para crear los archivos de configuración se pueden usar la página web *material.html* disponible en la carpeta *config*. También se puede crear a mano.

Si se opta por la opción de la web, hay que abrir el archivo *material.html*. En la página hay un formulario y debajo un botón. Se deben escribir los datos que se deseen. Luego al pulsar el botón se genera un texto. Este texto se debe copiar en un *.txt y guardarlo en la carpeta *config*.

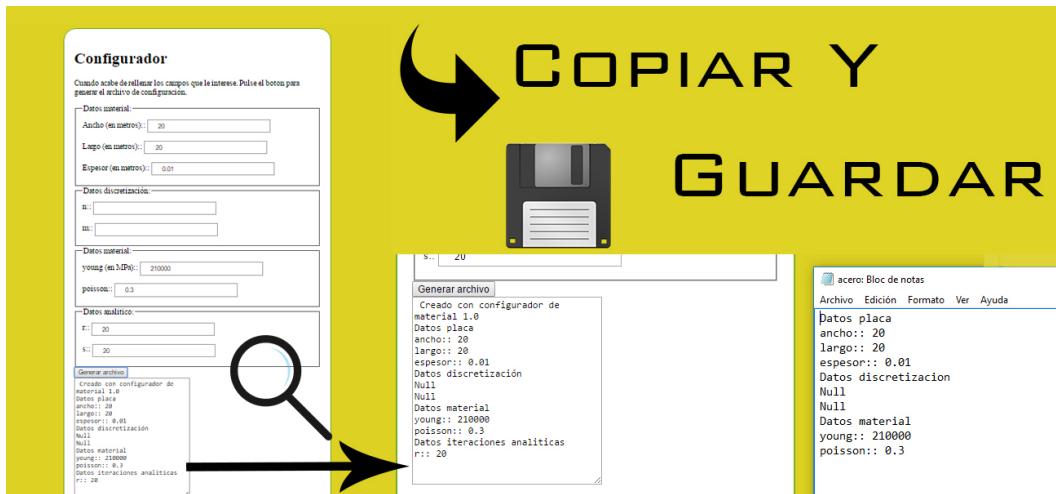


Figura 2.10: Instrucciones de uso que acompañan al archivo web

Si se crean a mano hay que tener cuidado en la forma en que se escriben los datos. Se recomienda coger un archivo de configuración existente y modificarlo para no equivocarse con el formato.



El programa lee línea por línea. En cada línea debe haber un texto y/o un número. Si en una línea hay más información Wallter la omite. El texto le indica al programa la variable y el número el valor que debe asignar a esa variable. No vale cualquier texto, solo detectará unas palabras clave. Si se pone un texto que no es una palabra clave el programa lo omitirá.

En la siguiente tabla se muestran como deben de ser esas palabras y a que variable se refieren. No tienen porque escribirse en un orden determinado.

Archivos de configuracion

Id	Keyword	Descripción
1	ancho::	Ancho de la placa
2	largo::	Largo de la placa
3	espesor::	Espesor de la placa
4	n::	Número de puntos de discretización en el modelo numérico
5	m::	Número de puntos de discretización en el modelo numérico
6	young::	Modulo de Young del material
7	poisson::	Coeficiente de poisson del material
8	r::	Iteraciones en el cálculo analítico
9	s::	Iteraciones en el cálculo analítico

Cuadro 2.2: Tabla con las palabras clave de los archivos de configuración

2.4. Documentación FORD

Además de este documento, en la carpeta doc del proyecto se encuentra una documentación más técnica. En ella se describen las variables del programa principal, los prototipos de las subrutinas y sus variables. Se incluye también un diagrama de flujo.

Esta documentación se genera de forma automática con el programa [FORD](#). El cual lee los comentarios y el código de los archivos a documentar. Luego genera un archivo html. Este archivo usa la plantilla bootstrap que permite ver el html en pequeñas pantallas (como la de los móviles).

Para poder usar [FORD](#) se necesita crear un archivo MarkDown con los datos de configuración y un texto de presentación.

Si se descarga el programa [FORD](#) y se instala, para generar la documentación técnica del programa se puede ejecutar este comando desde la carpeta raíz:

```
ford documentacion.md
```



2.2.4. Documentación FORD

Se generarán en la carpeta doc varios archivos. El archivo principal es el *index.html*. Para abrirlo se necesita un navegador actualizado instalado en el ordenador.

The screenshot shows a web-based documentation interface. At the top, there's a dark header bar with the text "Fortran Program", "Contents ▾", and a search bar. Below the header, the main content area has a light gray background. On the left, there's a sidebar with the title "I program" and a dropdown menu showing "Source Files", "Procedures", and "Program". To the right of the sidebar, the main content area contains the text "sroom". Below this, there's a section titled "Find us on..." with two blue buttons: "GitHub" and "The Web".

Fortran Program

Hi, my name is .

This is a project which I wrote. This file will provide the documents. I'm writing the body of the text here. It contains an overall description of the project. It might explain how to go about installing/compiling it. It might provide a change-log for the code. [[finalg]] Maybe it will talk about the history and/or motivation for this software.

Note

You can include any notes (or bugs, warnings, or todos) like so.

Figura 2.11: Ejemplo de una documentacion generada por FORD

La documentación se ha subido a un servidor para comodidad del usuario. Se puede ver en esta página web: <http://franterminator.com>.

Capítulo 3

Funcionamiento interno del programa

3.1. Subrutinas

PROGRAM Wallter

- ⇒ commandLine(asserts):: analiza los argumentos cuando se ejecuta el programa.
 - logical, intent(inout):: asserts → activa la impresión de los datos de factorización.
- ⇒ bienvenido():: escribe en pantalla un mensaje de bienvenida.
- ⇒ datos(ancho,largo,espesor,rigidez,n,m):: solicita al usuario los datos de la placa y material.
 - integer,intent(out):: n,m → número de puntos para discretizar la placa a lo ancho y largo (respectivamente).
 - real*8,intent(out):: largo,ancho,espesor → medidas de la placa
 - real*8,intent(out):: rigidez → rigidez a flexión de la placa.
- ⇒ construcMatriz(ancho,largo,matriz,n,m):: construye la matriz de los coeficientes del sistema.
 - integer,intent(in):: n,m → número de puntos para discretizar la placa a lo ancho y largo (respectivamente).
 - real*8,intent(in):: largo,ancho → medidas de la placa.
 - real*8,dimension(n*m,n*m),intent(inout):: matriz → matriz con los coeficientes del sistema.



- ⇒ construcVector(largo,rigidez,vector,n,m):: construye el vector de términos independientes.
- integer,intent(in):: n,m → número de puntos para discretizar la placa a lo ancho y largo (respectivamente).
 - real*8,intent(in):: largo → medidas de la placa.
 - real*8,intent(in):: rigidez → rigidez a flexión de la placa.
 - real*8,dimension(n*m),intent(inout):: vector → vector con los términos independientes.
- ⇒ fCholesky(matriz,n,m):: factorizada la matriz por el método de cholesky.
- integer,intent(in):: n,m → número de puntos para discretizar la placa a lo ancho y largo (respectivamente).
 - real*8,dimension(n*m,n*m),intent(inout):: matriz → matriz con los coeficientes del sistema.
- ⇒ linearSystem(matriz,f,n,m):: resuelve el sistema con los coeficientes de la matriz y los términos independientes f.
- integer,intent(in):: n,m → número de puntos para discretizar la placa a lo ancho y largo (respectivamente).
 - real*8,dimension(n*m,n*m),intent(inout):: matriz → matriz con los coeficientes del sistema.
 - real*8,dimension(n*m),intent(inout):: f → vector con los términos independientes.
- ⇒ analiticaNavier(w,ancho,largo,rigidez,n,m):: obtiene las flechas por el método analítico de Navier.
- real*8,dimension(n,m),intent(inout):: w → matriz con las flechas del método de Navier.
 - real*8,intent(in):: ancho,largo → medidas de la placa.
 - real*8,intent(in):: rigidez → rigidez a flexión de la placa.
 - integer,intent(in):: n,m → número de puntos para discretizar la placa a lo ancho y largo (respectivamente).

END PROGRAM Wallter

3.2. Pseudocódigo



```
PROGRAM Wallter
    call opcionesEjecucion(archivo,imprimir)
    call mensajeBienvenida

    call pedirDatos(ancho,largo,espesor,rigidez)
    fijar dimension matriz
    call construirMatriz(ancho,largo,espesor,matriz)
    fijar dimension vector
    call construirVector(largo,rigidez,vector)

    if(imprimir)
        write matriz
        write vector

        call cholesky(matriz)
        write matrizFactorizada

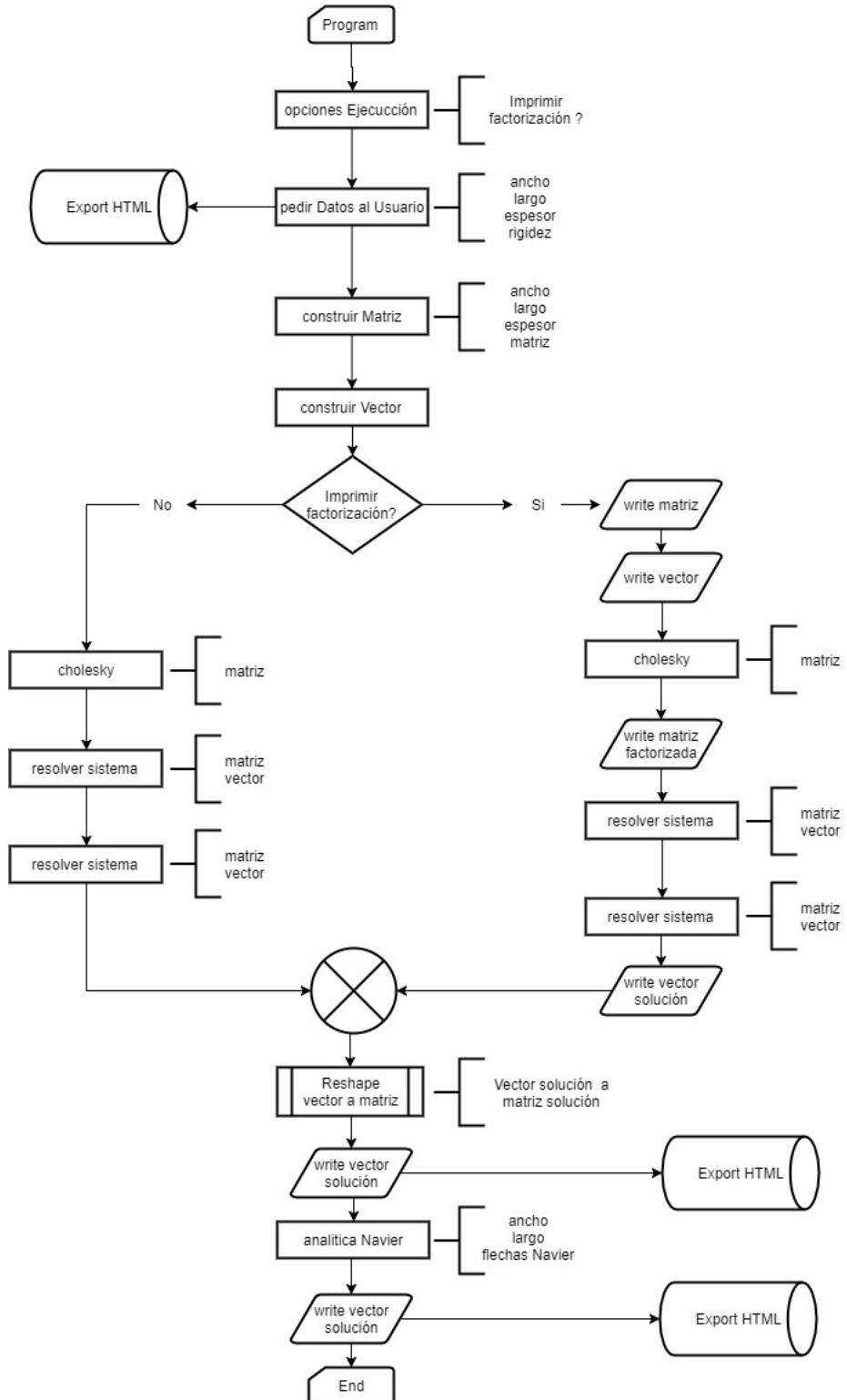
        call resolverSistema(matriz,vector)
        call resolverSistema(matriz,vector)
        write vectorSolucion
    else
        call cholesky(matriz)
        call resolverSistema(matriz,vector)
        call resolverSistema(matriz,vector)
    end if

    reshape vector a matrizSolucion
    write matrizSolucion
    call exportHtmlNumerica(matrizSolucion)

    call analiticaNavier(flechasNavier)
    write navierSolucion
    call exportHtmlAnalitica(flechasNavier)
END PROGRAM
```



3.3. Flujo del programa



Capítulo 4

Métodos de cálculo

4.1. Cálculo numérico por diferencias

El esquema de la placa sería el siguiente:

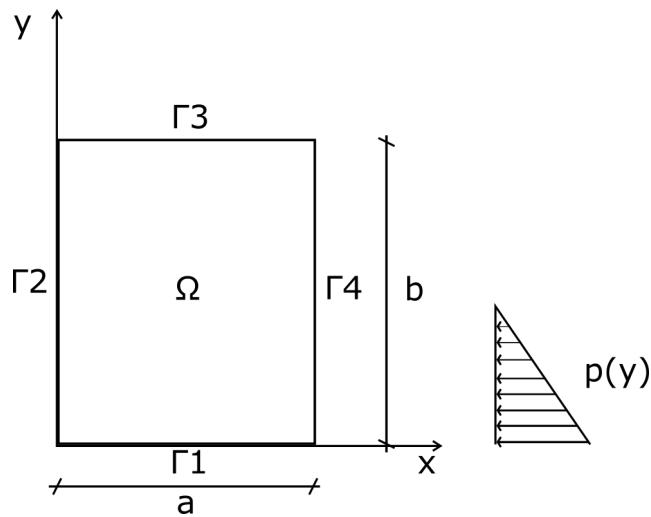


Figura 4.1: Esquema de la placa y la carga.

La flecha en una placa se calcula como:

$$\nabla^4 w = \frac{P}{D} = f$$

siendo: $D = \frac{E \cdot t^3}{12(1 - v^2)}$



Con las condiciones de contorno:

$$\text{c.c.} \left\{ \begin{array}{ll} w = 0 ; \frac{\delta^2 w}{\delta y^2} = 0 & \text{en } \Gamma_1 \\ w = 0 ; \frac{\delta^2 w}{\delta x^2} = 0 & \text{en } \Gamma_2 \\ w = 0 ; \frac{\delta^2 w}{\delta y^2} = 0 & \text{en } \Gamma_3 \\ w = 0 ; \frac{\delta^2 w}{\delta x^2} = 0 & \text{en } \Gamma_4 \end{array} \right.$$

Para facilitar los cálculos posteriores, se define ϕ como el segundo gradiente de la flecha. $\phi = \nabla^2 w$. De esta manera la fórmula general queda como:

$$\begin{aligned} \nabla^2 \phi &= f && \text{en } \Omega \\ \phi &= 0 && \text{en } \Gamma i = 1, \dots, 4 \end{aligned}$$

$$\begin{aligned} \nabla^2 w &= \phi && \text{en } \Omega \\ w &= 0 && \text{en } \Gamma i = 1, \dots, 4 \end{aligned}$$

Para poder calcular las flechas de forma numérica con ayuda de un programa informático se ha dividido la zona de cálculo en varios puntos.

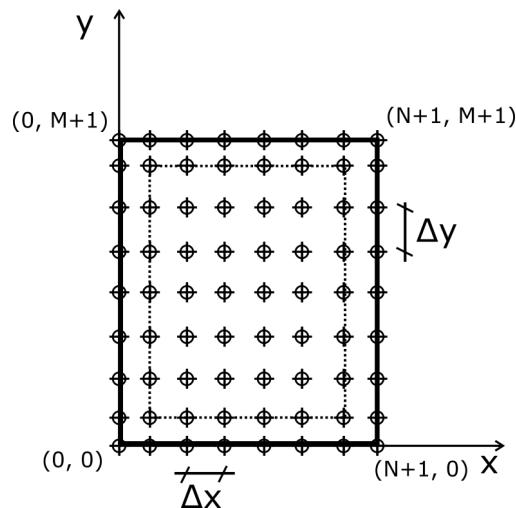


Figura 4.2: La zona enmarcada por la línea a puntos es la zona interior de la placa.

Queremos calcular la flecha, para ello tenemos que calcular $\nabla^2 \phi$. Mediante diferencias la derivada en un punto se calcula a partir de los puntos



anteriores y sus derivadas. Para poder calcular $\nabla^2\phi$ vamos usar el siguiente artificio:

$$\begin{aligned}\phi_{i+1,j} &= \phi_{i,j} + \frac{\delta\phi}{\delta x}\Big|_{i,j} \Delta x + \frac{1}{2} \frac{\delta^2\phi}{\delta x^2}\Big|_{i,j} \Delta x^2 + \frac{1}{6} \frac{\delta^3\phi}{\delta x^3}\Big|_{i,j} \Delta x^3 + \theta(\Delta x^4) \\ \phi_{i-1,j} &= \phi_{i,j} - \frac{\delta\phi}{\delta x}\Big|_{i,j} \Delta x + \frac{1}{2} \frac{\delta^2\phi}{\delta x^2}\Big|_{i,j} \Delta x^2 - \frac{1}{6} \frac{\delta^3\phi}{\delta x^3}\Big|_{i,j} \Delta x^3 + \theta(\Delta x^4) \\ &\quad + \dots \\ \phi_{i+1,j} + \phi_{i-1,j} &= 2\phi_{i,j} + \frac{\delta^2\phi}{\delta x^2}\Big|_{i,j} \Delta x^2 + \theta(\Delta x^4)\end{aligned}$$

Despejando $\frac{\delta^2\phi}{\delta x^2}$ obtenemos:

$$\frac{\delta^2\phi}{\delta x^2}\Big|_{i,j} = \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\theta(\Delta x^4)}{\Delta x^2}$$

Los errores disminuyen: $\frac{\theta(\Delta x^4)}{\Delta x^2} = \theta(\Delta x^2)$

Este proceso lo hemos hecho avanzando en i , sumando $\phi_{i+1,j} + \phi_{i-1,j}$. Si hacemos los mismos cálculos avanzado en j , obtenemos:

$$\frac{\delta^2\phi}{\delta y^2}\Big|_{i,j} = \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} + \frac{\theta(\Delta y^4)}{\Delta y^2}$$

Como ya conocemos $\frac{\delta^2\phi}{\delta y^2}$ y $\frac{\delta^2\phi}{\delta x^2}$, podemos calcular $\nabla^2\phi = f$.

$$\nabla^2\phi = \frac{\delta^2\phi}{\delta x^2}\Big|_{i,j} + \frac{\delta^2\phi}{\delta y^2}\Big|_{i,j} = \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} = f_{i,j} \quad (4.1)$$

Las incógnitas de esta ecuación son las ϕ . La variable f se calcula como P/D y los Δx y Δy se calculan cuando se escoge el número de puntos:

$$\begin{cases} \Delta x = \text{ancho placa/nº puntos} = \frac{a}{N+1} \\ \Delta y = \text{largo placa/nº puntos} = \frac{b}{M+1} \end{cases}$$

4.2. Matriz

Podemos desarrollar la ecuación (4.1) y separar las variables conocidas de las incógnitas.

$$\frac{1}{\Delta x^2}\phi_{i+1,j} - 2\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right)\phi_{i,j} + \frac{1}{\Delta x^2}\phi_{i-1,j} + \frac{1}{\Delta y^2}\phi_{i,j+1} + \frac{1}{\Delta y^2}\phi_{i,j-1} = f_{i,j}$$



Haciendo cambios de variables en las Δ .

$$B \cdot \phi_{i+1,j} + A \cdot \phi_{i,j} + B \cdot \phi_{i-1,j} + C \cdot \phi_{i,j+1} + C \cdot \phi_{i,j-1} = f_{i,j} \quad (4.2)$$

$$\begin{cases} A = -2\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right) \\ B = \frac{1}{\Delta x^2} \\ C = \frac{1}{\Delta y^2} \end{cases}$$

Para resolver el sistema se plantea de forma matricial.

Matriz para $n = 3$ y $m = 2$

$$\begin{bmatrix} A & B & 0 & C & 0 & 0 \\ B & A & B & 0 & C & 0 \\ 0 & B & A & B & 0 & C \\ C & 0 & B & A & B & 0 \\ 0 & C & 0 & B & A & B \\ 0 & 0 & C & 0 & B & A \end{bmatrix} \cdot \begin{bmatrix} \phi_{1,1} \\ \phi_{2,1} \\ \phi_{3,1} \\ \phi_{1,2} \\ \phi_{2,2} \\ \phi_{3,2} \end{bmatrix} = \begin{bmatrix} f_{1,1} \\ f_{2,1} \\ f_{3,1} \\ f_{1,2} \\ f_{2,2} \\ f_{3,2} \end{bmatrix}$$

Una vez obtenido ϕ podemos obtener las flechas w . Como $\nabla^2 w = \phi$, el sistema tiene la misma forma:

Matriz para $n = 3$ y $m = 2$

$$\begin{bmatrix} A & B & 0 & C & 0 & 0 \\ B & A & B & 0 & C & 0 \\ 0 & B & A & B & 0 & C \\ C & 0 & B & A & B & 0 \\ 0 & C & 0 & B & A & B \\ 0 & 0 & C & 0 & B & A \end{bmatrix} \cdot \begin{bmatrix} w_{1,1} \\ w_{2,1} \\ w_{3,1} \\ w_{1,2} \\ w_{2,2} \\ w_{3,2} \end{bmatrix} = \begin{bmatrix} \phi_{1,1} \\ \phi_{2,1} \\ \phi_{3,1} \\ \phi_{1,2} \\ \phi_{2,2} \\ \phi_{3,2} \end{bmatrix}$$

Finalmente obtenemos w , las flechas en los puntos equiespaciados ΔX e ΔY .

4.3. Cálculo analítico de Navier

Una placa sometida a una distribución de cargas $p(x, y)$ en dirección perpendicular a su plano medio, la flecha de la placa debe cumplir la siguiente ecuación diferencial:

$$\nabla^4 w = \frac{p(x, y)}{D}$$



Donde D es la rigidez de la placa a flexión expuesta antes:

$$D = \frac{E \cdot t^3}{12(1 - v^2)}$$

La solución que propuso Navier para la flecha w es una doble serie de Fourier de la forma:

$$w(x, y) = \sum_{m=1}^{\infty} \sum_{n=1}^{\infty} w_{mn} \operatorname{sen}\left(\frac{m\pi x}{a}\right) \operatorname{sen}\left(\frac{n\pi y}{b}\right)$$

Esta solución cumple todas las condiciones de borde apoyado en los cuatro lados, es decir:

$$w|_{x=0} = 0 \quad w|_{x=ancho} = 0 \quad w|_{y=0} = 0 \quad w|_{y=largo} = 0$$

$$\frac{\delta^2 w}{\delta x^2} \Big|_{x=0} = 0 \quad \frac{\delta^2 w}{\delta x^2} \Big|_{x=ancho} = 0 \quad \frac{\delta^2 w}{\delta x^2} \Big|_{y=0} = 0 \quad \frac{\delta^2 w}{\delta x^2} \Big|_{x=largo} = 0$$

Derivando la flecha en Series de Fourier podemos introducirla en la primera ecuación e igualarla a la flecha:

$$\nabla^4 w = \frac{p(x, y)}{D} = \sum_{m=1}^{\infty} \sum_{n=1}^{\infty} \left(\frac{m^4}{a^4} + \frac{n^4}{b^4} + 2 \frac{m^2 n^2}{a^2 b^2} \right) \pi^4 w_{mn} \operatorname{sen}\left(\frac{m\pi x}{a}\right) \operatorname{sen}\left(\frac{n\pi y}{b}\right) \quad (4.3)$$

La placa se ve sometida a una distribución de cargas $p(x, y)$. Representando la intensidad de la carga en doble serie de Fourier:

$$p(x, y) = \sum_{m=1}^{\infty} \sum_{n=1}^{\infty} p_{mn} \operatorname{sen}\left(\frac{m\pi x}{a}\right) \operatorname{sen}\left(\frac{n\pi y}{b}\right) \quad (4.4)$$

Para despejar p_{mn} multiplicaremos por el $\operatorname{sen}\left(\frac{m'\pi x}{a}\right)$ y $\operatorname{sen}\left(\frac{n'\pi y}{b}\right)$.

$$\int_0^a \int_0^b p(x, y) \operatorname{sen}\left(\frac{m'\pi x}{a}\right) \operatorname{sen}\left(\frac{n'\pi y}{b}\right) dx dy$$

Esta integral vale 0 cuando $n \neq n'$ y $m \neq m'$. Cuando $n = n'$ y $m = m'$ entonces el resultado es:

$$\int_0^a \int_0^b p(x, y) \operatorname{sen}\left(\frac{m'\pi x}{a}\right) \operatorname{sen}\left(\frac{n'\pi y}{b}\right) dx dy = p_{mn} \frac{a \cdot b}{4}$$



Igualando (4.3) con (4.4) obtenemos:

$$w_{mn} = \frac{p_{mn}}{\pi^4 D \left(\frac{m^2}{a^2} + \frac{n^2}{b^2} \right)^2}$$

Siendo: $p_{mn} = \frac{4}{a \cdot b} \int_0^a \int_0^b p(x, y) \sin\left(\frac{m\pi x}{a}\right) \sin\left(\frac{n\pi y}{b}\right) dx dy$

Finalmente ya tenemos todos los coeficiente para poder calcular la flecha.

$$w(x, y) = \sum_{m=1}^{\infty} \sum_{n=1}^{\infty} w_{mn} \sin\left(\frac{m\pi x}{a}\right) \sin\left(\frac{n\pi y}{b}\right)$$

4.3.1. Placa apoyada con carga de agua hasta la mitad

Una carga de presión hasta la mitad del largo de la placa se escribe como:

$$p(x, y) = \gamma_{\text{agua}} \left(\frac{\text{largo}}{2} - y \right)$$

Se calcula el coeficiente p_{mn} como:

$$p_{mn} = \frac{4 \cdot \text{largo} \cdot \sin\left[\frac{m\pi}{2}\right] (n\pi - 2\sin\left[\frac{n\pi}{2}\right])}{mn^2\pi^3}$$

Finalmente la flecha la calculamos como:

$$w_{mn} = \frac{4 \cdot \text{largo} \cdot \sin\left[\frac{m\pi}{2}\right] (n\pi - 2\sin\left[\frac{n\pi}{2}\right])}{mn^2\pi^8 \cdot \text{rigidez} \left(\frac{m^2}{\text{largo}^2} + \frac{n^2}{\text{ancho}^2} \right)^2}$$

$$w = \sum_{n=1}^{\infty} \sum_{m=1}^{\infty} w_{mn} \sin\left(\frac{m\pi x}{a}\right) \sin\left(\frac{n\pi y}{b}\right)$$

Capítulo 5

Resultados

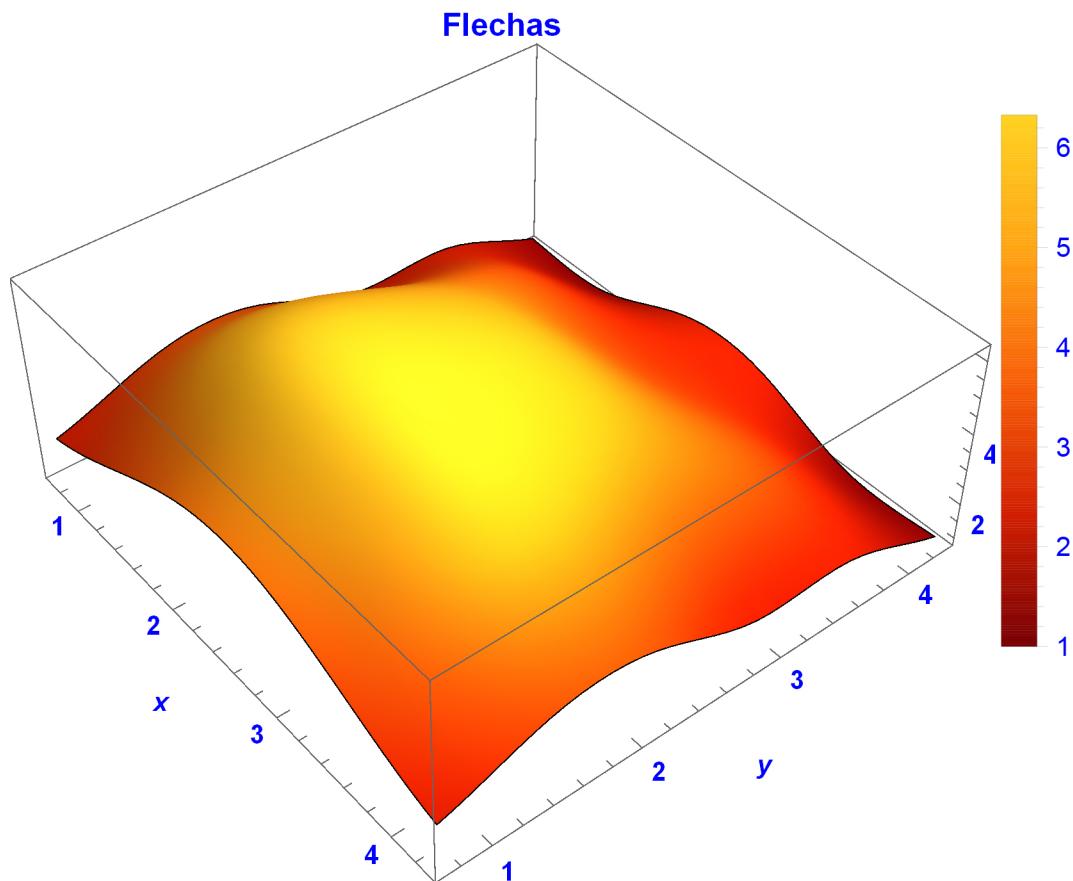


Figura 5.1: Flechas de una placa de acero de 5x5x0.01

Los resultados obtenidos por el programa son las flechas que experimenta la placa al estar sometida a una carga de agua hasta la mitad de su largo.



Los contornos están apoyados, por lo que no experimentarán flechas. Por lo tanto los contornos no se incluyen en los resultados mostrados por el programa.

Las flechas no se calculan para toda la placa. El programa calcula las flechas de varios puntos. Los puntos donde se hacen los cálculos vienen marcados por las variables n y m.

Las variables n y m representan el número de puntos equiespaciados que se desean calcular en la dirección del ancho y largo de la placa respectivamente. Cuanto mayor sea n y m, más puntos calculará el programa. En el caso del método numérico un aumento del número de puntos supone un aumento de precisión. El método se basa en los resultados del punto anterior para calcular el siguiente. Cuantos más puntos se calculen más se ajustarán las flechas calculadas a las reales.

En las gráficas que se van a mostrar a continuación los resultados de las flechas son continuos, no por puntos. Esto es porque se usó una función de interpolación para calcular los puntos intermedios entre los puntos calculados por Wallter.

5.1. Como se muestran los resultados

Las soluciones al problema las imprime en pantalla Wallter al ejecutarse. También se pueden ver en el archivo *resultados.html* en la carpeta result. El archivo muestra los datos de la placa y las flechas obtenidas. Para ver la solución analítica hay que pulsar un botón en la cabecera. Hay que pulsar el otro botón para volver a ver la solución numérica.

En la parte central de la web están las soluciones en forma de tabla. La flechas de la tabla están en cm. La distancia del punto al origen esta en metros, lo que aparece en negrita. Debajo de la tabla hay un botón para copiar los resultados al portapapeles.

La parte de arriba de la tabla serían las flechas en la parte superior, zona donde no habría agua. En la zona inferior hay agua y las flechas son mayores, tal como se aprecia en la parte de abajo de la tabla. Las flechas mayores se encuentran en medio de la región mojada aproximadamente.

Esto es el comportamiento real de una placa sometida a unas cargas de agua hasta la mitad. Lo que demuestra que los datos devueltos del programa en FORTRAN son coherentes.



5.2. Precisión

En este apartado intentaremos calcular cual es el número de puntos mínimo para obtener precisiones aceptables. Se entiende precisión aceptable por una precisión inferior al milímetro, ya que precisiones inferiores al milímetro ya no son medibles por los aparatos habituales: reglas/cintas, flexos, estaciones totales...

Los resultados del método numérico son más precisos cuanto más puntos se calculen. Por otro lado, los calculados por el método de Navier, son más precisos cuantas más iteraciones se hagan.

Lo primero que vamos a calcular es el número de iteraciones necesarias para obtener resultados precisos por el método analítico de Navier. Para ello introduciremos una placa de 20 m de largo, 20 m de ancho y 10 cm de espesor fabricada en acero (modulo de Young 210000 MPa y coeficiente de poisson 0.3) en el programa. Variando el número de iteraciones llegará un momento en que la diferencia entre hacer más o menos iteraciones será inferior al milímetro. Ese sería el valor recomendado de iteraciones.

	2.857	19.4318	32.8924	39.5852	39.5852	32.8924	19.4318
	5.714	28.9790	50.0668	60.8499	60.8499	50.0668	28.9790
	8.571	28.9497	50.9258	62.4997	62.4997	50.9258	28.9497
Y	11.429	23.6751	42.1076	52.0317	52.0317	42.1076	23.6751
	14.286	16.1587	28.9607	35.9509	35.9509	28.9607	16.1587
	17.143	8.1825	14.6749	18.2402	18.2402	14.6749	8.1825
	/	2.857	5.714	8.571	11.429	14.286	17.143
O	X						

Figura 5.2: Resultados analíticos para r y s igual 5



	2.857	19.4292	32.9153	39.5909	39.5909	32.9153	19.4292
	5.714	28.9489	50.0446	60.8155	60.8155	50.0446	28.9489
	8.571	28.9912	50.9796	62.5510	62.5510	50.9796	28.9912
Y	11.429	23.6241	42.0493	51.9716	51.9716	42.0493	23.6241
	14.286	16.1990	29.0071	35.9986	35.9986	29.0071	16.1990
	17.143	8.1621	14.6511	18.2157	18.2157	14.6511	8.1621
	/	2.857	5.714	8.571	11.429	14.286	17.143
O	X						

Figura 5.3: Resultados analíticos para r y s igual 10

	2.857	19.4188	32.9069	39.5881	39.5881	32.9069	19.4188
	5.714	28.9416	50.0386	60.8130	60.8130	50.0386	28.9416
	8.571	28.9913	50.9802	62.5528	62.5528	50.9802	28.9913
Y	11.429	23.6271	42.0525	51.9747	51.9747	42.0525	23.6271
	14.286	16.1991	29.0072	35.9987	35.9987	29.0072	16.1991
	17.143	8.1601	14.6490	18.2136	18.2136	14.6490	8.1601
	/	2.857	5.714	8.571	11.429	14.286	17.143
O	X						

Figura 5.4: Resultados analíticos para r y s igual 15



	2.857	19.4209	32.9081	39.5891	39.5891	32.9081	19.4209
	5.714	28.9422	50.0387	60.8130	60.8130	50.0387	28.9422
	8.571	28.9914	50.9802	62.5527	62.5527	50.9802	28.9914
Y	11.429	23.6268	42.0521	51.9744	51.9744	42.0521	23.6268
	14.286	16.1988	29.0069	35.9984	35.9984	29.0069	16.1988
	17.143	8.1598	14.6488	18.2134	18.2134	14.6488	8.1598
	/	2.857	5.714	8.571	11.429	14.286	17.143
O	X						

Figura 5.5: Resultados analíticos para r y s igual 20

En las tablas se puede ver como va variando la solución a medida que aumenta el número de iteraciones. Entre la última tabla y la penúltima las diferencias de flecha son menores del milímetro. Habríamos alcanzado una precisión aceptable con 15 iteraciones.

Aunque el número de iteraciones necesarias para alcanzar un valor preciso varía según el caso, en general unas 20 iteraciones es suficiente.

En el caso del método numérico. La precisión depende del número de puntos tomados, es decir, del valor de n y m. Cogiendo la placa anterior vamos a calcular la flecha para 4,8,10,15 y 20 puntos y veremos cuando se obtienen resultados precisos.

En las gráficas siguientes se representan las deformaciones de la placa. En cada caso se va aumentando el número de puntos de cálculo (n y m). En amarillo se representa la solución analítica y en azul la numérica.

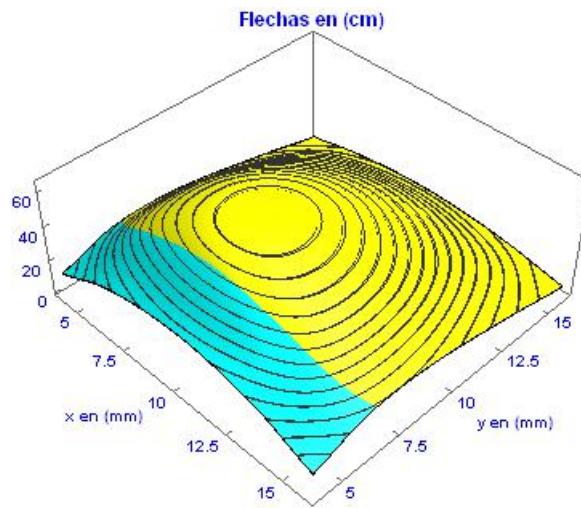


Figura 5.6: Flechas para la placa de acero para 8 puntos

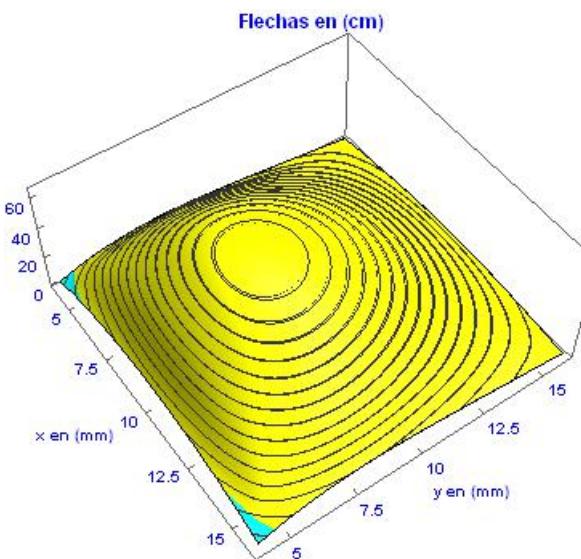


Figura 5.7: Flechas para la placa de acero para 15 puntos

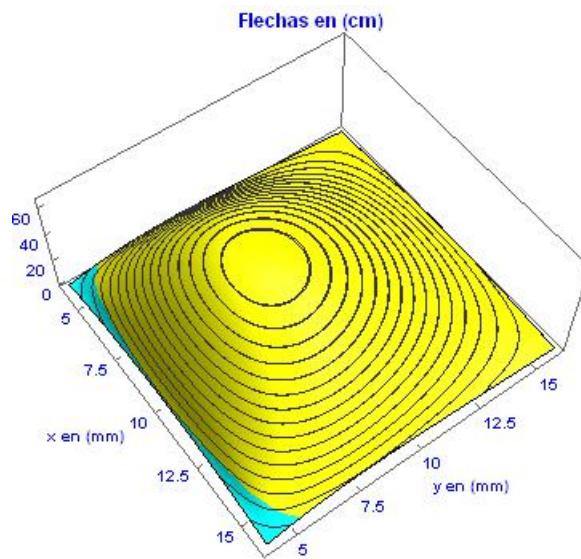


Figura 5.8: Flechas para la placa de acero para 25 puntos

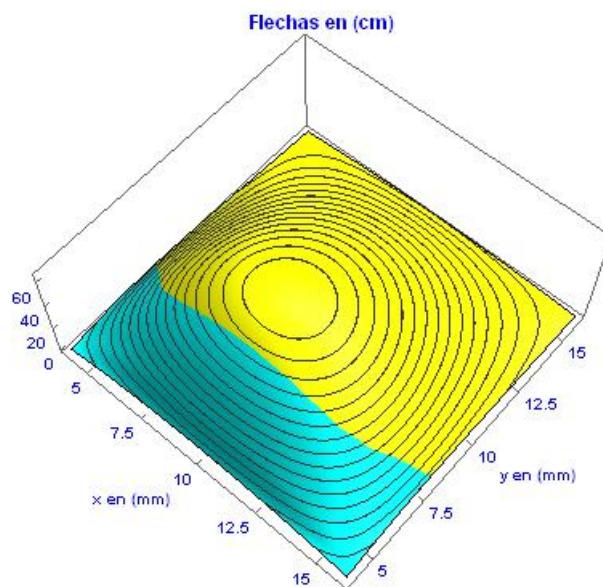


Figura 5.9: Flechas para la placa de acero para 30 puntos

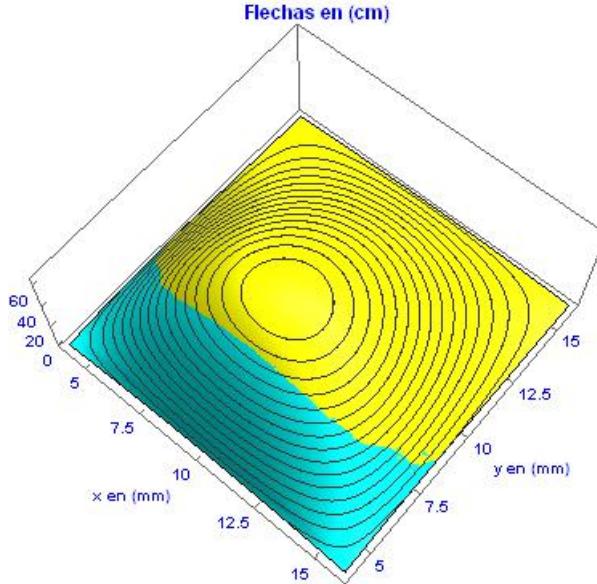


Figura 5.10: Flechas para la placa de acero para 50 puntos

En todos los casos se ha calculado la solución analítica para 20 iteraciones. Como vimos antes con 20 iteraciones se consigue una precisión aceptable, por lo tanto si la solución numérica se parece hasta el milímetro a la analítica se puede considerar una solución con precisión aceptable.

Esto se puede ver en las gráficas. En todas ellas se representan los resultados de ambos métodos y sus curvas de nivel. Si las curvas de nivel de ambos métodos se acercan quiere decir que nos acercamos a una solución aceptable. Para este caso fueron necesarios unos 30 puntos ($n=m=30$) para conseguir los resultados aceptables.

El problema de aumentar el número de puntos es que aumenta el tamaño de la matriz. Para $n=30$ los tiempos de ejecución ya fueron de casi 10 segundos (depende del ordenador). Para $n=50$ se tardó casi 30 segundos en obtener resultados. Sin embargo, aumentar las iteraciones del resultado analítico casi no repercute en el tiempo de cálculo.

En resumen, se aconseja usar la solución analítica para obtener resultados precisos. Una vez que se obtienen unos resultados válidos. Se puede ir variando el número de puntos en la solución numérica y compararlos con la solución analítica.

5.3. Importancia de las variables

No todos los parámetros afectan de la misma manera a los resultados. En el apartado anterior, se muestra como la variación de n , m y las iteraciones



5.5.3. Importancia de las variables

del proceso analítico solo afecta a la precisión de los cálculos. Pero los datos que tienen un sentido físico si que afectan a las flechas obtenidas.

Las variables más importantes son las que si se modifican provocan una apreciable variación en los resultados. Para descubrir cuales son, usaremos el cálculo de Navier. En el apartado 4.3.1 se obtiene que la flecha en la placa depende del factor w_{mn} :

$$w_{mn} = \frac{4 \cdot \text{largo} \cdot \sin\left[\frac{m\pi}{2}\right] (n\pi - 2\sin\left[\frac{n\pi}{2}\right])}{mn^2\pi^8 \cdot \text{rigidez} \left(\frac{m^2}{\text{largo}^2} + \frac{n^2}{\text{ancho}^2}\right)^2}$$

siendo la rigidez: $D = \frac{E \cdot t^3}{12(1 - v^2)}$

Tanto el modulo de Young como el coeficiente de poisson afectan a la rigidez a flexión. La cual divide el factor w_{mn} , es decir, cuanta mayor sea la rigidez menor será flecha.

El coeficiente de poisson no varía mucho de un material a otro. Suele tener valores comprendidos entre 0.2 (vidrio) y 0.5 (goma). En estas gráficas se muestran las flechas para una placa de acero de 20 x 20 x 0.1 con un modulo de Young de 210000 MPa.

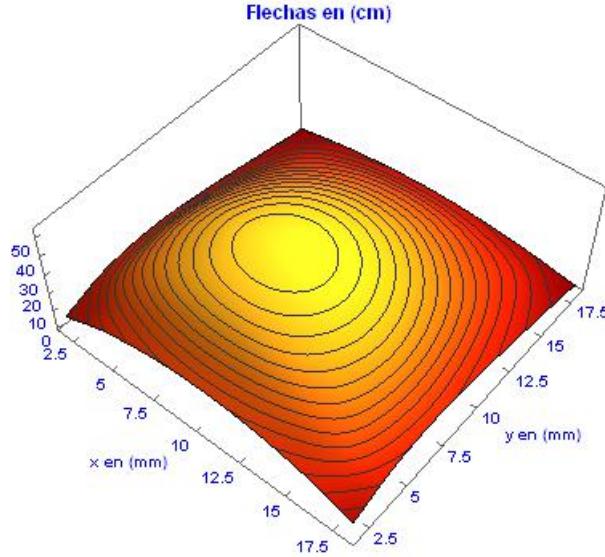


Figura 5.11: Coeficiente de poisson 0,4

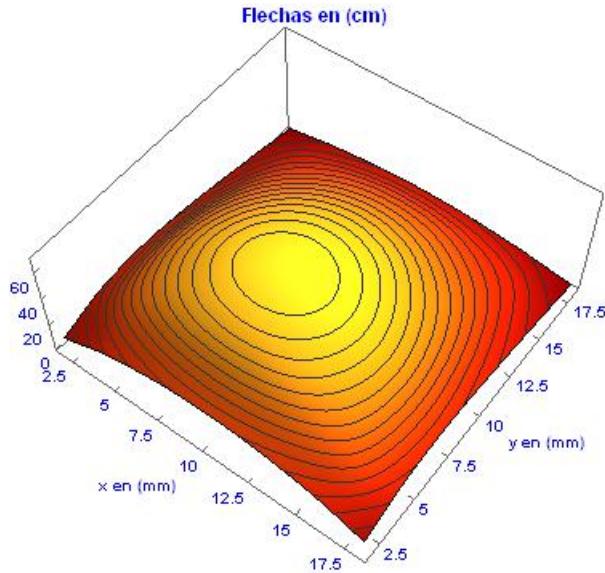


Figura 5.12: Coeficiente de poisson 0,2

Cuanto mayor es el coeficiente de poisson, mayor es la rigidez y por tanto menores son las flechas. En la figura de la derecha las flechas máximas son de 60 cm con un coeficiente de poisson de 0,2. En la figura de la izquierda las flechas alcanzan 50 cm para un poisson de 0,4.

El módulo de Young mide la relación entre las deformaciones longitudinales y la tensión aplicada. Cuanto mayor sea el módulo menores serán las deformaciones.

Para el ejemplo de la placa de 20x20x0,1 m y 0,3 de poisson se vario el módulo de Young y se obtuvieron los siguientes resultados. Las mayores deformaciones las sufre la placa con Young de 200000, ya que se deforma con más facilidad.

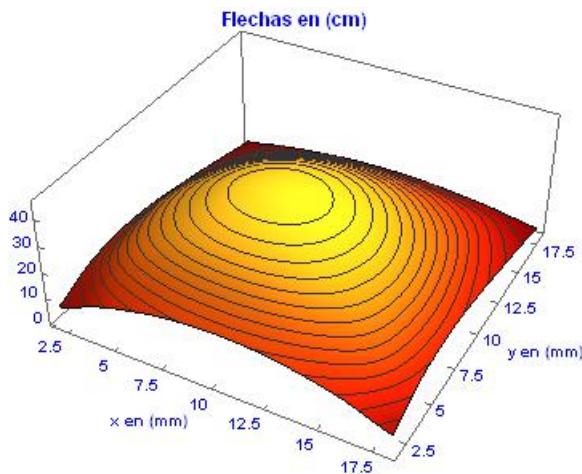


Figura 5.13: Módulo de Young de 300000 MPa



5.5.3. Importancia de las variables

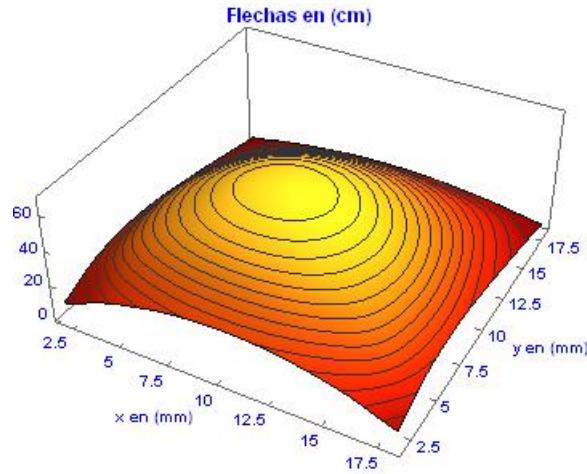


Figura 5.14: Módulo de Young de 200000 MPa

El último coeficiente que afecta a la rigidez es el espesor. Para calcular la rigidez a flexión hay que elevar el espesor al cubo, esto lo convierte en un factor importante. En las siguientes gráficas se muestran las deformaciones de una placa de acero con Young 210000 MPa y poisson 0,3. El largo y el ancho son de 20 metros, pero el espesor varía.

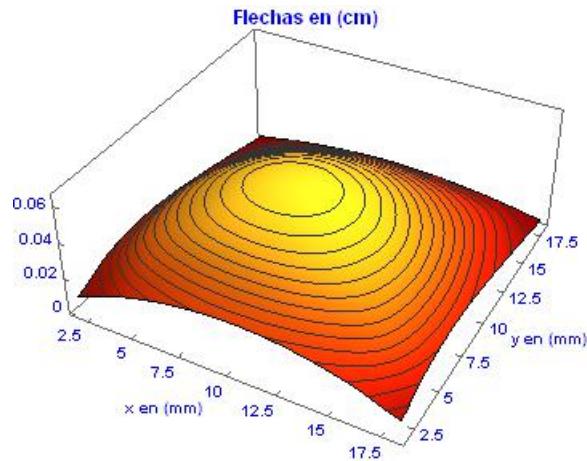


Figura 5.15: Placa con espesor de 0,1 m

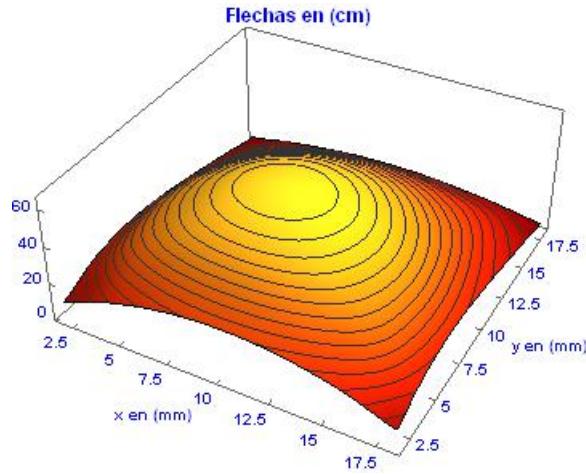


Figura 5.16: Placa con espesor de 1 m

Como cabía esperar, una variación en el espesor implica un cambio importante del resultado. Llegando a ser un cambio de varias magnitudes. La placa con espesor de 0,1 m llega a deformarse 60 cm. La placa de mayor espesor solo 0,06 cm.

Queda solo analizar el largo y el ancho de la placa. Son variables importantes porque están elevados a la cuarta:

$$\left(\frac{m^2}{largo^2} + \frac{n^2}{ancho^2} \right)^2$$

Un cambio de una de estas variables supone un cambio de varias magnitudes en las deformaciones.

Las siguientes gráficas muestran las flechas para una placa de acero con 0,1 metros de espesor. En el primer caso la placa es de 40 metros de largo. En el segundo caso tiene 10 metros de largo.



5.5.3. Importancia de las variables

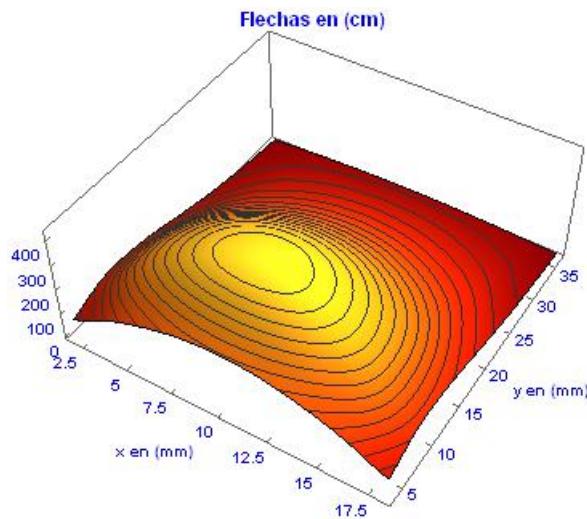


Figura 5.17: Placa de 40 metros de largo y 20 de ancho.

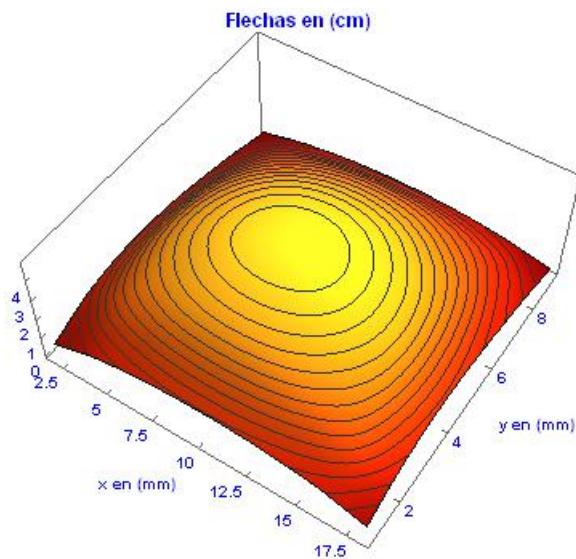


Figura 5.18: Placa de 10 metros de largo y 20 de ancho.

En resumen, un error en el módulo de Young y coeficiente de poisson repercute notablemente en los resultados pero no destaca tanto como un error en las medidas de la placa. Los errores en la medida de la placa suponen un cambio importante en la solución final.

Capítulo 6

Conclusión

En conclusión, el programa Wallter es un programa válido para el cálculo de flechas. Su precisión es buena con pocas iteraciones. Además cuenta con dos modos de cálculo que permite contrastar los valores resultantes.

Sus múltiples opciones permiten ejecutar el programa de formas rápidas y sencillas. Con opciones para comprobar los resultados intermedios y comprobar que no hay ningún fallo lógico.

Las flechas obtenidas se pueden ver en un archivo web bien estructurado. En el cual se ve claramente cada uno de los datos introducidos.

Este programa destaca por su velocidad. Otros programas de cálculos de estructuras como Abaqus son más robustos y permiten mas opciones. Pero sus interfaces pesadas y lentas les impiden alcanzar la velocidades de ejecución de Wallter. Este programa es una buena herramienta para el ingeniero que necesite conocer las deformaciones de una placa sumergida.

Parte II

Código del programa



Datos del programa	
Nombre::	Wallter
Archivo::	main.f95
Autor::	Francisco Rivera Álvarez
Versión::	3.1
Comentarios::	

```
!< author: francisco rivera alvarez
! Programa para el calculo de flechas de una placa apoyada
! en sus cantos con carga hidraulica hasta la mitad de su largo
Program WALLTER

real*8, dimension(:,:),allocatable:: matriz !! sistema de
ecuaciones
real*8, dimension(:),allocatable:: f      !! vector terminos
independientes
real*8,dimension(:,:),allocatable:: navier !! vector terminos
analiticos

real*8, dimension(:,:),allocatable:: fMatriz!! solucion en forma
matricial
integer,dimension(2):: forma           !! forma de la matriz
solucion
integer,dimension(2):: orden = (/2,1/) !! orden de los numeros
al cambiar vector a matriz solucion

character(len=50):: resultsFile = './result/resultados.html' !!
directorio donde se escribiran los resultados
character(len=50):: configFile = ''          !!
directorio donde se encuentran los config file
logical:: asserts = .FALSE.           !! si se activa se
imprimen los datos de la factorizacion
logical:: exists                      !! comprueba la
existencia de los directorios de resultados

real*8:: ancho,largo,espesor           !! dimensiones de la placa
real*8:: rigidez                      !! rigidez a flexion de
la placa
integer:: n, m                         !! numero de puntos para
discretizar largo(n) y ancho(m)

COMMON resultsFile,configFile        !! variables globales
```



```
! comprueba si existe el directorio de resultados
inquire(file=resultsFile,exist=exists)
if(.NOT.exists) resultsFile = 'result.html' ! si no existe crea
    el archivo al lado del programa

! inicio del programa
call commandLine(asserts)           ! opciones de ejecucion
call bienvenido()                   ! mensaje de bienvenida
(header)
call datos(ancho,largo,espesor,rigidez,n,m) ! datos para poder
    funcionar el programa

allocate(matriz(n*m,n*m))           ! fija la matriz de
    resultados numericos
call constructMatriz(ancho,largo,matriz,n,m)! construye la
    matriz para los resultados numericos
allocate(f(n*m))                   ! fija el vector de
    resultados numericos
call constructVector(largo,rigidez,f,n,m) ! construye el vector
    para los resultados numericos

! muestra la matriz y factorizacion
if (asserts) then
    call printMatrix(matriz, 'Matriz') ! imprime en pantalla la
        matriz
    call printVector(f, 'Vector')    ! imprime en pantallas el
        vector de terminos independientes

! factorizacion y resultados
call fCholesky(matriz,n,m)          ! factorizada la matriz por
    cholesky
call printMatrix(matriz , 'Matriz (factorizada)')

! resolucion del sistema
call linearSystem(matriz,f,n,m)
call linearSystem(matriz,f,n,m)

! muestra solo la solucion
else

    ! factoriza y resuelve
    call fCholesky(matriz,n,m)
    call linearSystem(matriz,f,n,m)
    call linearSystem(matriz,f,n,m)
end if

! cambio de vector a matriz solucion
```



```
f = f *100
allocate(fMatriz(m,n))
forma(1) = n
forma(2) = m
fMatriz = reshape(f,forma,order=orden)
call printMatrix(fMatriz, 'Solucion numerica')
call resNumericos(ancho,largo,fMatriz,n,m)
call wolframNumerico(ancho,largo,fMatriz,n,m)

! calcula los resultados analiticos por Navier
allocate(navier(m,n))
call analiticaNavier(navier,ancho,largo,rigidez,n,m)
call printMatrix(navier, 'Solucion analitica')
call resAnaliticos(ancho,largo,navier,n,m)
call wolframAnalitico(ancho,largo,navier,n,m)

! para que no se cierre el programa derepente
write(*,*) "Gracias por usar el programa..."
read(*,*)

contains
!< Imprime una linea, un texto centrado y una linea.
! Creando como un titulo.
subroutine header(label)
    character(len=*),intent(in):: label
    character(len=30):: formato
    character(len=2):: iString
    integer:: i

    ! 30 is the line length / 2
    i = 30 - len(label)/2
    write(iString,'(i2)') i
    formato = '(//iString//X,A)'
    call linea()
    write(*,formato) label
    call linea()
end subroutine

!< Imprime una matriz mas un texto como titulo.
subroutine printMatrix(matriz,label)
    real*8,dimension(:, :):: matriz
    character(len=*),intent(in):: label
    integer:: i,j

    call header(label)
    do i=1,ubound(matriz,1)
        write(*, '(*(f0.4,5x))') (matriz(i,j),j=1,ubound(matriz,2))
```



```
    end do
    call linea()
end subroutine

!< Imprime un vector mas un texto como titulo
subroutine printVector(f,label)
    real*8,dimension(:),intent(in):: f
    character(len=*),intent(in):: label
    integer:: i

    call header(label)
    do i=1,ubound(f,1)
        write(*,*) f(i)
    end do
    call linea()
end subroutine
End Program WALLTER

!< Imprime una linea de 60 caracteres.
! Se usa para separar la informacion que se imprime en la pantalla
subroutine linea()
    write(*,*)
    "-----"
end subroutine

!< Analiza los argumentos cuando se ejecuto el programa.
! -h -> help: imprime las opciones disponibles.
! -a -> asserts: activa la impresion de los datos de la
    factorizacion.
subroutine commandLine(asserts)
    logical,intent(out):: asserts !! cuando es verdadera se activa
        la impresion de los datos de factorizacion

    integer:: i                      !! numero de argumentos
    character(len=30):: cmd          !! el argumento que se analiza

    character(len=50):: resultsFile,configFile
    COMMON resultsFile,configFile

    i = iargc()
    if(i>0) then
        do i=1,iargc()
            call getarg(i,cmd)
            if(cmd == '-h') then
                write(*,*) 'Las opciones disponibles son:'
                write(*,*) '-h para ver esta ayuda'
```



```
        write(*,*) '-a para activar la impresion de los datos
                    de factorizacion'
        write(*,*) '-c [FILENAME] para usar un programa de
                    configuracion con los datos'
        call exit(0)
    end if
    if(cmd == '-a') then
        write(*,*) 'Se ha activado la impresion de los datos
                    de factorizacion.'
        asserts = .TRUE.
    end if
    if(cmd == '-c') then
        write(*,*) 'Se usara para los datos el siguiente
                    config file: '
        call getarg(i+1,configFile)
        write(*,*) configFile
        exit
    end if
end do

end if
end subroutine

!< Imprime un mensaje de bienvenida
subroutine bienvenido()

    write(*,*) "                                     ,--,
      ,---,                                     ,--,
    write(*,*) "                                     ,---.'| ,---.'| ,/
      .'|
    write(*,*) "         .---. ,---, | | : | | : ,'
      .' : ,---,.,-.-. ,"
    write(*,*) "         / . /| ' . ' \   : | : | : | ; ;
      / , ' . ' \| / \ "
    write(*,*) "         .--'. ' ; / ; ' . | ' : | ' : .'_--,/ '
      , '---.' | ; : \| "
    write(*,*) "         /__./ \ : | : ; \ ; ; ; ; ' | :
      | | | .'| | .\ : "
    write(*,*) "         .--'. ' \^ .: | / \ \ ' | | _' | | _';
      | .'; ; : | : | -,. : | : | "
    write(*,*) "/ __/ \ | ' ' | : ' ; . : | | | : ' || | : ' | '----'
      | | : | ; / | | | \ : "
    write(*,*) "; \ \ ; : | | ; / \ \ \ ' : ; ' : ; ; ' :
      : ; | : | .'| : . / "
    write(*,*) " \ ; ' ' | ' : | \ \ \ , ' | | . / | | . /
      | ' | | | -;, | | | \ : "
    write(*,*) " . \ \ \ ; | | ' ' --' ; : ; ; : ; ' :
```



```
: | ; : ;/|| | ;\ \ "
write(*,*) " \ \ \ ' \| || : : | ,/ | ,/ ;
| . ' | | \: ' | \. ' "
write(*,*) " : ' | --' | | , '---' '---'
'---' | : . ' : :--' "
write(*,*) " \ \ ; '---' | | , ' | | . ' "
write(*,*) " '---'
"
""

write(*,*) "Bienvenido -----> "
write(*,'(20X,A)') "Pulse enter para continuar"
read(*,*)
end subroutine

!< Solicita los datos necesarios para el calculo de la flecha.
subroutine datos(ancho,largo,espesor,rigidez,n,m)
! ----- Variables -----
integer*4,intent(out):: n,m !! numero de puntos para
discretizar la placa
real*8,intent(out):: largo, ancho, espesor !! dimensiones de la
placa
real*8,intent(out):: rigidez !! rigidez a flexion -> D
= E*t^3/(12(1-v^2))

real*8:: Young !! Modulo de Young
real*8:: poisson !! Coeficiente de Poisson

logical:: exists = .FALSE. !! existe el archivo de
configuracion?
character(len=50):: label !! etiquetas del archivo
de configuracion
real*8:: numero = 0

integer*4,dimension(7):: modif = 0
integer:: i
integer:: sel=1
character(len=7),dimension(7):: etiquetas = &
(/"ancho ","largo ","espesor","n      ","m      ","Young
","poisson"/)

integer*4:: ios=0
character(len=50):: resultsFile,configFile
COMMON resultsFile,configFile
```



```
! ***** Código ***** !  
  
! comprueba que se haya definido un archivo de configuracion y  
de que existe  
if(configFile /= '') then  
    inquire(file=configFile,exist=exists)  
    if(.NOT.exists) then  
        write(*,*) 'No se ha encontrado el archivo de  
        configuracion.'  
    end if  
end if  
  
! si existe se abre y se lee la informacion  
if(exists) then  
    open(unit=24,file=configFile,status='old',action='read') ! se  
    abre  
    ! se lee la informacion, quitando las etiquetas  
    ! importante poner las etiquetas como en los archivos ejemplo  
  
    do while(ios>=0)  
        read(24,*,<filestat>iostat=ios) label, numero  
  
        ! check if there were problems reading the file  
        if(ios < 0) then  
            continue  
        end if  
  
        ! check label reference  
        if(label=='ancho::') then  
            ancho = numero  
            modif(1) = 1  
        end if  
        if(label=='largo::') then  
            largo = numero  
            modif(2) = 1  
        end if  
        if(label=='espesor::') then  
            espesor = numero  
            modif(3) = 1  
        end if  
        if(label=='n::') then  
            n = numero  
            modif(4) = 1  
        end if  
        if(label=='m::') then  
            m = numero  
            modif(5) = 1
```



```
end if
if(label=='young::') then
    Young = numero
    modif(6) = 1
end if
if(label=='poisson::') then
    poisson = numero
    modif(7) = 1
end if
end do
close(24)

! se imprime los datos obtenidos por si el usuario los quiere
cambiar
write(*,*) "Datos config file ::"
write(*,'(5X,A,f0.3,A,f0.3,A,f0.3)') "Datos placa ->",ancho,"
    x ",largo," x ",espesor
write(*,'(5X,A,i2,A,i2,A)') "Datos discretizacion [n,m] ->
    [",n,",",",m,"]"
write(*,'(5X,A,f0.1,A,f0.3)') "Datos material -> Young::
    ",Young,"; poisson:: ",poisson

endif

call linea()
write(*,*) 'Se procede a pedir los datos al usuario que falten.'
write(*,*) 'Si se equivoca en algun numero podra cambiarlo al
final de programa.'

do while(sel==1)
    ! Datos tecnicos de la placa
    if(modif(1)==0 .OR. modif(2)==0 .OR. modif(3)==0) then
        write(*,*) "*****"
        write(*,*) "* DATOS DE LA PLACA *"
        write(*,*) "*****"
        if(modif(1)==0) then
            write(*,*) "-> ancho de la placa"
            write(*,'(A,$)') "(metros) "
            read(*,*) ancho; modif(1) = 1
        end if
        if (modif(2)==0) then
            write(*,*) "-> largo de la placa"
            write(*,'(A,$)') "(metros) "
            read(*,*) largo; modif(2) = 1
        end if
        if (modif(3)==0) then
            write(*,*) "-> espesor de la placa"
```



```
        write(*,'(A,$)') "(metros) "
        read(*,*) espesor; modif(3) = 1
    end if
end if

! datos para la discretizacion del modelo
if(modif(4)==0 .OR. modif(5)==0) then
    write(*,*) "*****"
    write(*,*) "* NUMERO DE PUNTOS *"
    write(*,*) "*****"
    if(modif(4)==0) then
        write(*,*) "-> puntos para discretizar el ancho"
        write(*,'(A,$)') "(integer) "
        read(*,*) n; modif(4) = 1
    endif
    if(modif(5)==0) then
        write(*,*) "-> puntos para discretizar el largo"
        write(*,'(A,$)') "(integer) "
        read(*,*) m; modif(5) = 1
    endif
endif

! propiedades del material
if(modif(6)==0 .OR. modif(7)==0) then
    write(*,*) "*****"
    write(*,*) "* DATOS DEL MATERIAL *"
    write(*,*) "*****"
    if(modif(6)==0) then
        write(*,*) "-> Modulo de Young"
        write(*,'(A,$)') "(MPa) "
        read(*,*) Young; modif(6) = 1
    endif
    if(modif(7)==0) then
        write(*,*) "-> Coef. de poisson"
        write(*,'(A,$)') "(_real_) "
        read(*,*) poisson; modif(7) = 1
    endif
endif

! cambiar valores si se han introducido valores a mano
! si hay un archivo se entiende que el usuario no tiene
! la necesidad de cambiar ningun dato.
if(.NOT.exists) then
    call linea()
    write(*,*) "Desea cambiar algun valor:"
    write(*,*) "1) Si"
    write(*,*) "2) No"
```



```
read(*,*) sel
if(sel==1) then
    write(*,"(5X,A)") "Que valor desea cambiar?"
    do i = 1,7
        write(*,"(10X,I2,A,A)") i,") ",etiquetas(i)
    end do
    read(*,*) i
    modif(i) = 0
end if
else
    sel = 20
end if
end do

! MPa (N/mm3) -> 1000 KPa (kN/m2)
rigidez = Young*1000*espesor**3
rigidez = rigidez/(12*(1-poisson**2))

! EXPORT HTML
call createHTML("resultados")
call exportMaterial(Young,poisson)
call exportPlaca(largo,ancho,espesor)
end subroutine

!< Calculo los terminos de la matriz de resultados numericos y
! los coloca en su sitio. La matriz se almacena en banda.
subroutine constructMatriz(ancho,largo,matriz,n,m)
    ! ----- Variables -----
    real*8,dimension(n*m,n*m),intent(inout):: matriz
    real*8,intent(in):: ancho,largo
    integer,intent(in):: n,m

    real*8:: A,B,C,deltaX,deltaY
    integer:: i,j

    ! calculo de los coef A, B y C
    deltaX = ancho / (n + 1)
    deltaY = largo / (m + 1)
    B = 1 / (deltaX**2)
    C = 1 / (deltaY**2)
    A = -2 * (B + C)
    write(*,'(A,3(f0.2,X),A)') "[A,B,C] -> [ ",A,B,C," ]"

    !matriz y vector a cero
    do i=1,n*m
        do j=1,n*m
            matriz(i,j) = 0
```



```
      end do
    end do

    ! construccion matriz
    do i=1,n*m
      matriz(i,1) = A
      if (mod(i,n) == 0 .AND. i > 1) then
        matriz(i,2) = 0
      else
        matriz(i,2) = B
      end if
      if(i+n <= n*m) matriz(i,n+1) = C
    end do
  end subroutine

!< Construye el vector de resultados numericos
subroutine constructVector(largo,rigidez,vector,n,m)
  ! ----- Variables -----
  real*8,dimension(n*m),intent(inout):: vector
  real*8,intent(in):: largo,rigidez
  integer,intent(in):: n,m

  real*8:: deltaY,presion

  ! vector a cero
  do i=1,n*m
    vector(i) = 0
  end do

  deltaY = largo / (m + 1)

  ! construccion vector
  j = 1
  do i=1,n*m
    ! peso especifico agua = 10000 N / m3 -> 10 kN / m3
    presion = 10*(largo/2-j*deltaY)/rigidez

    !solo hasta la mitad
    if (presion < 0) then
      vector(i) = 0
    else
      vector(i) = presion
    end if

    ! las filas tienen la misma presion
    if(mod(i,m) == 0) j = j + 1
  end do
```



```
end subroutine

!< Factorización de Cholesky -> A = L * D * Transpose[L]
subroutine fCholesky(matriz, n, m)
    ! ----- Variables -----
    real*8,dimension(n*m,n*m),intent(inout):: matriz
    integer:: k,i,j
    real*8:: suma

    ! factorizacion de cholesky
    do k=1,n*m-1
        do i=1,k
            suma = 0
            do j=1,i-1
                suma = suma + matriz(j,i-j+1)*matriz(j,k+2-j)
            end do
            matriz(i,k+2-i) = (matriz(i,k+2-i) - suma)
        end do
        do i=1,k
            matriz(i,k+2-i) = matriz(i,k+2-i)/matriz(i,1)
        end do

        suma = 0
        do j=1,k
            suma = suma + matriz(j,k+2-j)*matriz(j,1)*matriz(j,k+2-j)
        end do
        matriz(k+1,1) = matriz(k+1,1) - suma
    end do
end subroutine

!< Resolucion del sistema de ecuaciones para una matriz en banda
factorizada
subroutine linearSystem (matriz, f, n, m)
    ! ----- Variables -----
    real*8,dimension(n*m,n*m),intent(inout):: matriz
    real*8,dimension(n*m),intent(inout):: f
    integer,intent(in):: n, m
    integer:: i,j
    real*8:: suma = 0

    do i=2,n*m
        suma = 0
        do j=1,i-1
            suma = suma + matriz(j,i-j+1)*f(j)
        end do
        f(i) = f(i) - suma
    end do
```



```
do i=1,n*m
   f(i) = f(i) / matriz(i,1)
end do

do i=n*m-1,1,-1
   suma = 0
   do j=i+1,n*m
      suma = suma + matriz(i,j-i+1)*f(j)
   end do
   f(i) = f(i) - suma
end do
end subroutine

!< Resolucion del sistema mediante el metodo de Navier
subroutine analiticaNavier(w,ancho,largo,rigidez,n,m)
! ----- Variables -----
real*8,intent(in):: ancho,largo
real*8,intent(in):: rigidez
real*8,dimension(m,n),intent(out):: w
integer,intent(in):: n, m

integer:: i, j, r, s
integer:: k, u
real*8:: deltaX, deltaY, X, Y
real*8:: p_ku, w_ku
real*8:: pi = acos(-1.0d0)

integer,dimension(2):: modif = 0
integer:: sel=1
integer:: h

logical:: exists = .FALSE.           !! existe el archivo de
configuracion?
character(len=50):: label          !! etiquetas del archivo
de configuracion
real*8:: numero

integer:: ios
character(len=50):: resultsFile,configFile
COMMON resultsFile,configFile

! ***** Código *****
! comprueba que se haya definido un archivo de configuracion y
de que existe
if(configFile /= '') then
```



```
inquire(file=configFile,exist=exists)
if(.NOT.exists) then
    write(*,*) 'No se ha encontrado el archivo de
                configuracion.'
    write(*,*)
end if
end if

if(exists) then
    open(unit=24,file=configFile,status='old',action='read') ! se
        abre
    do while(ios>=0)
        read(24,*,iostat=ios) label, numero

        ! check if there were problems reading the file
        if(ios < 0) then
            write(*,*) "Fin de archivo"
            continue
        end if

        ! check label reference
        if(label=='r::') then
            r = numero
            modif(1) = 1
        end if
        if(label=='s::') then
            s = numero
            modif(2) = 1
        end if
    end do
    close(24)

    write(*,*) "Los datos de ",configFile," son:"
    write(*,'(A,I4,I4,A)') "Numero de iteraciones para el calculo
                                analitico -> [r,s] = [",r,s,"]"
end if

do while(sel==1)
    ! datos para el bucle de calculo de flecha
    if (modif(1) == 0 .OR. modif(2) == 0) then
        write(*,*) "*****"
        write(*,*) "* NUMERO DE ITERACIONES *"
        write(*,*) "*****"
        write(*,*) "-> Numero de iteraciones para el calculo
                            analitico: "
        if(modif(1) == 0) then
            write(*,'(A,$)') 'r (interger): '
        end if
    end if
```



```
        read(*,*) r; modif(1) = 1
end if
if(modif(2) == 0) then
    write(*,'(A,$)') 's (interger): '
    read(*,*) s; modif(2) = 1
end if
end if

! cambiar valores si se han introducido valores a mano
! si hay un archivo se entiende que el usuario no tiene
! la necesidad de cambiar ningun dato.
if(.NOT.exists) then
    call linea()
write(*,*) "Desea cambiar algun valor:"
write(*,*) "1) Si"
write(*,*) "2) No"
read(*,*) sel
if(sel==1) then
    write(*,"(5X,A)") "Que valor desea cambiar?"
    write(*,"(10X,A)") "1) r"
    write(*,"(10X,A)") "2) s"
    read(*,*) h
    modif(h) = 0
end if
else
    sel = 20
end if
end do

!calculamos deltaX y deltaY
deltaX = ancho / (n + 1)
deltaY = largo / (m + 1)

! ceros en w
do i=1,n
    do j=1,m
        w(j,i) = 0
    end do
end do
!bucle para cada x e y
do i=1,n
    X = i*deltaX
    do j=1,m
        Y = j*deltaY
```



```
!bucle para el calculo de la flecha
do k=1,r
    do u=1,s
        p_ku = 4 * largo * 10 * sin(k*pi/2)**2 / (k * u**2
            * pi**3)
        p_ku = p_ku * (u * pi - 2 * sin(u*pi/2))

        w_ku =(k**2 / ancho**2 + u**2 / largo**2)
        w_ku = w_ku**2
        w_ku = p_ku / (pi**4 * rigidez * w_ku)

        w(j,i) = w(j,i) + w_ku * sin(k*pi*X/ancho) *
            sin(u*pi*Y/largo) * 100
    end do
end do
end do
end do

end subroutine
```



Datos del programa	
Nombre::	<i>Wallter</i>
Archivo::	<i>results.f95</i>
Autor::	<i>Francisco Rivera Álvarez</i>
Versión::	<i>3.1</i>
Comentarios::	<i>Exporta los resultados a HTML</i>

```
!< Crea el head y el principio del body del html de resultados
! para una mejor visualizacion del html, se recomienda crearlo7
! en la carpeta result. Por defecto el programa lo hara asi.
subroutine createHTML(name)
    character(len=*) ,intent(in):: name

    character(len=50):: resultsFile,configFile
    COMMON resultsFile,configFile

    open(unit=12,file=resultsFile)
    write(12,'(A,$)') '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
        Transitional//EN"'
    write(12,*) '<html xmlns="http://www.w3.org/1999/xhtml"
        xml:lang="en" lang="en">'
    write(12,*)<head>
    write(12, '(5X,3A)') '<title>',name,'</title>'
    write(12, '(5X,A)') '<meta http-equiv="Content-Type"
        content="text/html; charset=utf-8" />'
    write(12, '(5X,A)') '<meta name="description" content="" />'
    write(12, '(5X,A)') '<meta name="keywords" content="" />'
    write(12, '(5X,A)') '<meta name="robots" content="index,follow"
        />'
    write(12, '(5X,A)') '<link rel="stylesheet" type="text/css"
        href=".//css/style.css" />'
    write(12, '(5X,A)') '<script
        src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-3.2.1.min.js">'
    write(12, '(5X,A)') '</script>'
    write(12, '(5X,A)') '<script
        src="https://cdn.jsdelivr.net/clipboard.js/1.6.0/clipboard.min.js">'
    write(12, '(5X,A)') '</script>'
    write(12,*)</head>
    write(12,*)<body>
    write(12, '(5X,A)') '<div class="row">'
```



```
write(12,'(10X,A)') '<div class="header">'  
write(12,'(15X,A)') '<h1> Resultados del programa </h1>'  
write(12,'(15X,A)') '<button id="bNumerico">Resultados  
    Numericos</button>'  
write(12,'(15X,A)') '<button id="bAnalitico">Resultados  
    Analiticos</button>'  
write(12,'(10X,A)') '</div>'  
write(12,'(5X,A)') '</div>'  
close(12)  
end subroutine  
  
!< Escribe en la table de materiales en el html de resultados  
subroutine exportMaterial(Young,poisson)  
real*8,intent(in):: Young,poisson  
character(len=50):: resultsFile,configFile  
COMMON resultsFile,configFile  
  
open(unit=12,file=resultsFile,status='old',position="append")  
write(12,'(5X,A)') '<div class="row">'  
write(12,'(10X,A)') '<div class="box-item">'  
write(12,'(15X,A)') '<header> Material </header>'  
write(12,'(15X,A)') '<p> El material empleado es : </p>'  
write(12,'(15X,A)') '<table>'  
write(12,'(20X,A,F0.3,A)') '<tr><td>E = ',Young,' MPa</td></tr>'  
write(12,'(20X,A,F4.3,A)') '<tr><td>v = ',poisson,'</td></tr>'  
write(12,'(15X,A)') '</table>'  
close(12)  
end subroutine  
  
!< Escribe la table de datos de placa en el html de resultados  
subroutine exportPlaca(largo,ancho,espesor)  
real*8,intent(in):: largo,ancho,espesor  
character(len=50):: resultsFile,configFile  
COMMON resultsFile,configFile  
  
open(unit=12,file=resultsFile,status='old',position="append")  
write(12,'(15X,A)') '<header> Placa </header>'  
write(12,'(15X,A)') '<p>Las medidas de la placa son:</p>'  
write(12,'(15X,A)') '<table>'  
write(12,'(20X,A,F6.3,A)') '<tr><td>Largo = ',largo,',  
    m</td></tr>'  
write(12,'(20X,A,F6.3,A)') '<tr><td>Ancho = ',ancho,',  
    m</td></tr>'  
write(12,'(20X,A,F4.3,A)') '<tr><td>Espesor = ',espesor,',  
    m</td></tr>'  
write(12,'(15X,A)') '</table>'  
write(12,'(10X,A)') '</div>'
```



```
    close(12)
end subroutine

!< Escribe la tabla de resultados numericos en el html de resultados
subroutine resNumericos(ancho,largo,fMatriz,n,m)
    real*8,dimension(n,m),intent(in):: fMatriz
    real*8,intent(in):: ancho,largo
    integer,intent(in):: n,m

    real*8:: deltaX, deltaY

    character(len=50):: resultsFile,configFile
COMMON resultsFile,configFile

deltaX = ancho / (n + 1)
deltaY = largo / (m + 1)

open(unit=12,file=resultsFile,status='old',position="append")
write(12,'(10X,A)') '<div class="box-item" id="resultados">'
write(12,'(15X,A)') '<header id="azul"> Resultados </header>',
write(12,'(15X,A)') '<p> Tabla con las flechas (en cm) </p>',
write(12,'(15X,A)') '<p> X es la distancia a lo ancho hasta el
    Origen (en metros) </p>',
write(12,'(15X,A)') '<p> Y es la distancia a lo largo hasta el
    Origen (en metros) </p>',
write(12,'(15X,A)') '<table id="tNumerico">',
write(12,'(20X,A)') '<tr class="tr-h">'
write(12,'(25X,A,I5,A)') '<td rowspan="" ,m+2, '"> Y </th>',
write(12,'(20X,A)') '</tr>

do i=1,n,1
    write(12,'(20X,A)') '<tr>'
    write(12,'(25X,A,F0.3,A)') '<td
        class="td-h">,i*deltaX,'</td>',
    write(12,'(25X,*(A,f0.4,A))')
        ('<td>',fMatriz(i,j),'</td>',j=1,m)
    write(12,'(20X,A)') '</tr>'
end do

write(12,'(20X,A)') '<tr class="tr-h">'
write(12,'(25X,A)') '<td>/</td>',
write(12,'(25X,*(A,F0.3,A))') ('<td>',j*deltaY,'</td>',j=1,m)
write(12,'(20X,A)') '</tr>'
write(12,'(20X,A)') '<tr class="tr-h">'
write(12,'(25X,A,I4,A)') '<td>0</td><td colspan="" ,n+1, '">X</td>',
write(12,'(20X,A)') '</tr>'
write(12,'(15X,A)') '</table>'
```



```
    close(12)
end subroutine

!< Escribe la tabla de resultados analiticos en el html de resultados
subroutine resAnaliticos(ancho,largo,navier,n,m)
    real*8,dimension(n,m),intent(in):: navier
    real*8,intent(in):: ancho,largo
    integer,intent(in):: n,m

    real*8:: deltaX, deltaY

    character(len=50):: resultsFile,configFile
COMMON resultsFile,configFile

deltaX = ancho / (n + 1)
deltaY = largo / (m + 1)

open(unit=12,file=resultsFile,status='old',position="append")
write(12,'(15X,A)') '<table id="tAnalitico">'
write(12,'(20X,A)') '<tr class="tr-h">'
write(12,'(25X,A,I5,A)') '<td rowspan="" ,m+2, '"> Y </th>'
write(12,'(20X,A)') '</tr>

do i=1,n,1
    write(12,'(20X,A)') '<tr>
    write(12,'(25X,A,F0.3,A)') '<td
        class="td-h">,i*deltaX,</td>
    write(12,'(25X,*(A,f0.4,A))')
        ('<td>',navier(i,j),'</td>',j=1,m)
    write(12,'(20X,A)') '</tr>
end do

write(12,'(20X,A)') '<tr class="tr-h">
write(12,'(25X,A)') '<td></td>
write(12,'(25X,*(A,F0.3,A))') ('<td>,j*deltaY,</td>',j=1,m)
write(12,'(20X,A)') '</tr>
write(12,'(20X,A)') '<tr class="tr-h">
write(12,'(25X,A,I4,A)') '<td>0</td><td colspan="" ,n+1, '">X</td>
write(12,'(20X,A)') '</tr>
write(12,'(15X,A)')'</table>

write(12,'(10X,A)') '<button id="copyButton"
    data-clipboard-action="copy" data-clipboard-target=".active">
write(12,'(15X,A)') 'Copia la tabla</button>
write(12,'(10X,A)') '</div>
write(12,'(5X,A)') '</div>
write(12,'(5X,A)') '<script src="js/buttons.js"></script>
```



```
write(12,* ) '</body>'  
write(12,* ) '</html>'  
close(12)  
end subroutine  
  
subroutine wolframNumerico(ancho,largo,flechas,n,m)  
real*8,dimension(n,m),intent(in):: flechas  
real*8,intent(in):: ancho,largo  
integer,intent(in):: n,m  
  
real*8:: deltaX, deltaY  
  
deltaX = ancho / (n + 1)  
deltaY = largo / (m + 1)  
  
open(unit=25,file='./result/wolframNumerico.txt')  
write(25,* ) 'ListPlot3D[{'  
  
do i=1,n,1  
    write(25,'(A,$)') '{'  
    do j=1,m  
        write(25,'(f0.4,$)') flechas(i,j)  
        if(j/=m) write(25,'(A,$)') ','  
    end do  
    if(i/=n) then  
        write(25,'(A)') '},'  
    else  
        write(25,'(A,$)') ']  
    end if  
end do  
write(25,'(A)') '},'  
  
write(25,'(A,f0.3,A,f0.3,A,f0.3,A,f0.3,A)') 'DataRange ->  
    {{',deltaX,',',n*deltaX,'},{',deltaY,',',m*deltaY,'}},'  
write(25,'(A)') 'InterpolationOrder -> 3,'  
write(25,'(A)') 'AxesLabel -> {"x en (mm)", "y en (mm)"},'  
write(25,'(A)') 'LabelStyle -> Directive[Blue, Bold], PlotLabel  
    -> "Flechas en (cm)",'  
write(25,'(A)') 'ColorFunction -> "SolarColors", PlotTheme ->  
    "Web",PlotLegends -> Automatic]  
close(25)  
end subroutine  
  
subroutine wolframAnalitico(ancho,largo,flechas,n,m)  
real*8,dimension(n,m),intent(in):: flechas  
real*8,intent(in):: ancho,largo
```



```
integer,intent(in):: n,m

real*8:: deltaX, deltaY

deltaX = ancho / (n + 1)
deltaY = largo / (m + 1)

open(unit=25,file='./result/wolframAnalitico.txt')
write(25,*)'ListPlot3D[{'

do i=1,n,1
    write(25,'(A,$)') '{'
    do j=1,m
        write(25,'(f0.4,$)') flechas(i,j)
        if(j/=m) write(25,'(A,$)') ','
    end do
    if(i/=n) then
        write(25,'(A)') '},'
    else
        write(25,'(A,$)') '}]'
    end if
end do
write(25,'(A)') '},'

write(25,'(A,f0.3,A,f0.3,A,f0.3,A,f0.3,A)') 'DataRange ->
{{,deltaX,,n*deltaX,},{,deltaY,,m*deltaY,}},'
write(25,'(A)') 'InterpolationOrder -> 3,
write(25,'(A)') 'AxesLabel -> {"x en (mm)", "y en (mm)"},'
write(25,'(A)') 'LabelStyle -> Directive[Blue, Bold], PlotLabel
-> "Flechas en (cm)",
write(25,'(A)') 'ColorFunction -> "SolarColors", PlotTheme ->
"Web",PlotLegends -> Automatic],
close(25)
end subroutine
```
