DESARROLLO DE UN PROGRAMA PARA LA SOLUCIÓN DE ECUACIONES NO LINEALES APLICADO A LA ENSEÑANZA Y APRENDIZAJE DE LA ESTÁTICA EN LA INGENIERÍA MECÁNICA.

WILLIAM ARCADIO SUÁREZ MOLINA
RAUL ALBERTO CULMA PÉREZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO - MECÁNICAS
ESCUELA DE INGENIERÍA MECÁNICA
BUCARAMANGA
2009

DESARROLLO DE UN PROGRAMA PARA LA SOLUCIÓN DE ECUACIONES NO LINEALES APLICADO A LA ENSEÑANZA Y APRENDIZAJE DE LA ESTÁTICA EN LA INGENIERÍA MECÁNICA.

WILLIAM ARCADIO SUÁREZ MOLINA
RAUL ALBERTO CULMA PÉREZ

Trabajo de Grado para optar al título de
Ingeniero Mecánico

Director
DAVID FUENTES
Ingeniero Mecánico

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE CIENCIAS FÍSICO – MECÁNICAS
ESCUELA DE INGENIERÍA MECÁNICA
BUCARAMANGA
2009

DEDICATORIA

A Dios mi única razón para vivir.

A mi padre Arcadio Suárez, a quien agradezco el inmenso deseo de salir adelante que sembró en mí, su gran amor y todo su generoso apoyo.

DEDICATORIA

Que difícil es plasmar en un papel el cúmulo de sentimientos que suscito mi pensamiento en escasos segundos. Escasos segundos de valor temporal pero de gran valor emocional. Quiero dedicar este triunfo a mi madre, Raimunda Pérez Cepeda quien cuando nací era un ser que aparecía para aplaudir mis últimos logros. Cuando me iba haciendo mayor, era una figura que me enseñaba la diferencia entre el mal y el bien. Durante mi adolescencia era la autoridad que ponía límites a mis deseos. Ahora que soy adulto, es la mejor consejera y amiga que tengo. Madre, Tu fuerza y tu amor me han dirigido por la vida y me han dado las alas que necesitaba para volar.

AGRADECIMIENTOS

Expreso mis agradecimientos a todas aquellas personas e instituciones que han colaborado y hecho posible la realización de este trabajo.

En primer lugar y muy especialmente le doy gracias Dios por ser mi mejor aliado y amigo, gracias a David Fuentes, Ing. Mecánico UIS, director del proyecto por su inmensa dedicación y paciencia, quien prestó su decidido apoyo e invaluable colaboración poniendo a disposición toda su experiencia, trabajo, tiempo y valioso conocimiento.

A todos aquellos compañeros de academia, con los cuales vivimos y compartimos los mejores momentos, gracias por su amistad.

A las personas que colaboraron con sus valiosos aportes: Nelly Vargas, Manuel Cabarcas, Alberto Silva, Milton Galván.

CONTENIDO

pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABLAS

pág.

LISTA DE ANEXOS

pág.

RESUMEN

TITULO: DESARROLLO DE UN PROGRAMA PARA LA SOLUCIÓN DE ECUACIONES NO LINEALES APLICADO A LA ENSEÑANZA Y APRENDIZAJE DE LA ESTÁTICA EN LA INGENIERÍA MECÁNICA.[*]

AUTOR: WILLIAM ARCADIO SUÁREZ MOLINA
RAUL ALBERTO CULMA PÉREZ[**]

PALABRAS CLAVES: Programación orientada a objetos, clases, librerías, ecuaciones no lineales método Newton-Raphson.

DESCRIPCION: El objetivo de este proyecto es dotar a la escuela de Ingeniería Mecánica de la Universidad Industrial de Santander, de un programa que pueda utilizarse como complemento de la asignatura Estática.

El algoritmo del programa se diseñó con base en la programación orientada a objetos, adaptando de las respectivas librerías, las clases para la edición e introducción de ecuaciones de forma literal, clases para solucionar sistemas de ecuaciones no lineales utilizando el método de Newton Raphson de múltiples variables y las clases para visualización de la solución de forma numérica, paramétrica y gráfica, usando como lenguaje de programación **Visual C++** por su versatilidad y la facilidad con que permite crear interfaces gráficas en un entorno de Windows.

El resultado es un software de fácil manejo que permite solucionar una amplia variedad de problemas concernientes a la estática en forma rápida, presentado los resultados en un formato claro y de fácil interpretación, donde incluye soluciones a los sistemas de ecuaciones representadas en las ventanas de forma numérica, paramétrica y gráfica. Estas características hacen el programa muy útil tanto para estudiantes y profesores como para ingenieros, ya que evita tediosos y largos procedimientos de cálculo, facilitando más tiempo para el análisis y comprensión de los problemas a analizar.

[*] Trabajo de Grado
[**] Facultad de Ciencias Físico-Mecánicas, Escuela de Ingeniería Mecánica, Ing. David Fuentes

SUMMARY

TITLE: PROGRAM FOR THE SOLUTION OF NONLINEAR EQUATIONS DEVELOPMENT APPLIED TO STATIC TEACHING AND LEARNING IN MECHANICAL ENGINEERING.*

AUTHORS: WILLIAM ARCADIO SUÁREZ MOLINA
RAUL ALBERTO CULMA PÉREZ**

KEY WORDS: Object Oriented Programming, classes, libraries, nonlinear equations, Newton-Raphson method.

DESCRIPTION: The Project objective is to endow to mechanical engineering school of Santander's University Industrial with a program that can be used as a complement of the static subject.

The program algorithm was designed based on object oriented programming, adapting to respective libraries, the classes for edition and introduction of equations in literal forms, the ones to solve nonlinear equations systems by using the Newton Raphson method of multiple variables and the ones for the visualization of numerical form solution, graphic and parametric form, using the **VISUAL C++** programming language due to the versatility and facility that it allows creating graphics interfaces on windows environments.

The result is a software of easy handling that allows to solve a wide range of static subject problems quickly, giving a view of the results in a clear format of easy interpretation, that includes solutions to the equations systems represented in the numerical form windows, parametric and graphic ones. This characteristics make a useful program for students, teachers and engineers, since it avoids tedious and large calculation procedures, facilitating more time to the analysis and understanding of the problems to examine.

* Degree Work
** Physical-Mechanical Sciences Faculty, Mechanical Engineering, Eng. David Fuentes

INTRODUCCIÓN

En los últimos años, el desarrollo de herramientas computacionales en el campo de la educación e ingeniería, ha originado un cambio total en nuestra visión del futuro. Antiguamente, el estudiante o profesor dedicaba gran parte de su tiempo al proceso de cálculos dispendiosos y repetitivos con el fin de solucionar un problema. Hoy, esto ha cambiado gracias al uso de los computadores y de los lenguajes de programación, los cuales permiten modelar un sinnúmero de problemas, reduciendo al mínimo el tiempo de ejecución de los cálculos y permitiendo el análisis de diversas situaciones, con sólo modificar cualquiera de las variables que intervienen en el mismo.

En la escuela de Ingeniería Mecánica, muchas asignaturas se están beneficiando con la introducción de paquetes de software versátiles y amigables, que le permiten al estudiante y al profesor familiarizarse con el uso de las herramientas computacionales y al mismo tiempo aplicarlas a la solución de problemas de ingeniería.

Por las razones anteriores, se desarrolló la presente tesis de grado, titulada "Desarrollo de un programa para la solución de ecuaciones no lineales aplicado a la enseñanza y aprendizaje de la estática en la ingeniería mecánica", **STATICSOLVER 1.0**. Este programa permite solucionar un sistema de ecuaciones no lineales para cualquier problema de estática, que no sólo da la solución de cada variable sino que están incluidas herramientas como tabla paramétrica, que ayuda al estudiante a ingresar diferentes valores de una variable elegida y observar cómo se ven afectadas las demás variables, también se dispone de una herramienta gráfica que le permite al estudiante visualizar los resultados de la tabla paramétrica. Con estas herramientas el estudiante tiene una solución global del problema, dándole un mejor análisis al problema.

El capítulo 1 tiene como fin identificar el problema a resolver de este proyecto, para darle un mejor desarrollo a la asignatura de estática mediante la creación de un software que resuelva sistemas de ecuaciones no lineales aplicado a la asignatura, también se describen los antecedentes de librerías y clases que ayudarán para la solución del problema.

En el capítulo 2, se plantea la solución al problema una vez identificado, presentando herramientas que se tienen en cuenta como el método de solución "Newton Raphson", P.O.O (Programación Orientada a Objetos), lenguaje de programación C++, Visual C, librerías y diferentes clases, que llevaron al desarrollo del problema.

El capítulo 3 muestra los resultados obtenidos del proyecto, comparando el cálculo de diferentes problemas de estática.

Se incluye como anexo a este documento, el **Manual del Usuario de STATICSOLVER 1.0**.

1 IDENTIFICACIÓN DEL PROBLEMA

1.1 GENERALIDADES

Dentro de las estrategias de la Universidad en materia de la utilización de las Tecnologías de la Información y Comunicación (TICs), la escuela debe proyectar acciones de tal manera que sus prácticas docentes estén de acuerdo con las políticas de la Universidad. Dentro de este marco, es posible disponer de un sistema basado en las TICs que permitirá plantear y resolver problemas relacionados de la estática.

Actualmente, la enseñanza de la estática se realiza basada en la explicación que realiza el profesor sobre la solución del problema, y ante el eventual hecho de desear la participación del estudiante, las operaciones necesarias para la solución de los problemas son, algunas veces, demasiado extensas, de manera que se pierde sentido práctico en el desarrollo de la clase. Además, dentro de la gran variedad de asignaturas vistas durante el transcurso de la carrera, existe la posibilidad de aplicar la metodología desarrollada en este proyecto a algunas de ellas. Por otro lado, la solución de ecuaciones no lineales está presente en la solución de muchos problemas prácticos en la ingeniería.

Con este proyecto de grado se desea disponer de un sistema de solución de ecuaciones no lineales que permita resolver problemas de estática usando tanto notación vectorial como escalar.

1.2 MÉCANICA

Mecánica es una rama de la física que describe y predice condiciones de reposo o movimiento de los cuerpos bajo la acción de fuerzas.
La mecánica se divide en tres partes principales:

1. Mecánica de cuerpos rígidos.
2. Mecánica de cuerpos deformables.
3. Mecánica de fluidos.

La mecánica de cuerpos rígidos es aquella que estudia el movimiento y equilibrio de sólidos, ignorando sus deformaciones. Esta a su vez se divide en ESTÁTICA y DINAMICA. La primera estudia los cuerpos en reposo y la segunda los cuerpos en movimiento. En esta parte de la Mecánica se supone que los cuerpos son perfectamente rígidos. Sin embargo, las estructuras y las máquinas reales nunca lo son y se deforman bajo las cargas que están sometidas. Estas deformaciones son casi siempre pequeñas y no afectan de manera apreciable las condiciones de equilibrio o de movimiento de la estructura en consideración.

1.2.1 Definición de Estática

La **Estática** como su nombre lo indica es la parte de la mecánica que estudia el equilibrio de fuerzas, sobre un cuerpo en reposo.

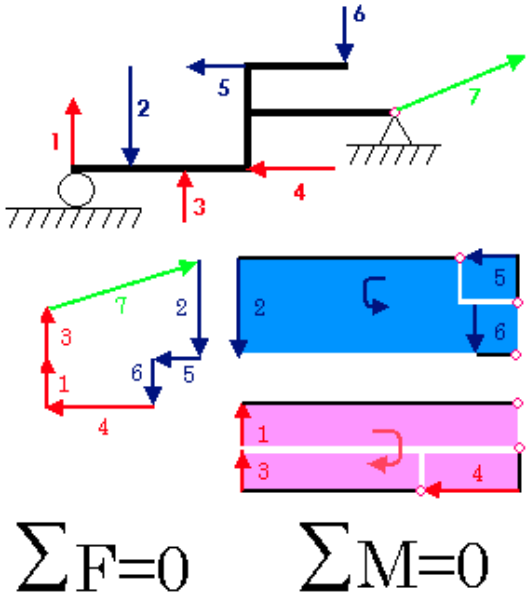
1.2.2 Análisis del Equilibrio

El equilibrio en un sistema es una situación estacionaria en la que se cumplen estas dos condiciones:

1. La sumatoria de fuerzas es igual a cero.
2. La sumatoria de momentos respecto a un punto es cero.

Para problemas estáticamente determinados, es suficiente plantear las condiciones básicas de equilibrio como se muestra en la Figura 1, en la cual se tiene un sistema en equilibrio dado que la suma de las siete fuerzas y momentos que actúan en la viga es cero.

Figura 1. Diagrama de fuerzas y momentos de una viga en equilibrio



Estas dos condiciones, mediante el álgebra vectorial, se convierten en un sistema de ecuaciones, cuya solución se obtiene resolviendo la condición de equilibrio. Según las leyes de la mecánica, una partícula en equilibrio no sufre aceleración lineal ni de rotación, pero puede estar moviéndose a velocidad uniforme o rotar a velocidad angular uniforme. Esto es ampliable a un sólido rígido. Las ecuaciones necesarias y suficientes de equilibrio son:

- Una partícula o un sólido rígido esta en equilibrio de traslación cuando la suma de todas las fuerzas que actúan sobre el cuerpo es cero:

$$\sum_{i=1}^{n} \vec{F}_i = 0 \qquad (1)$$

En el espacio se tienen tres ecuaciones de fuerzas, una por dimensión. Descomponiendo cada fuerza en sus coordenadas se tiene:

$$\vec{F}_i = F_{i,x}\vec{u}_x + F_{i,y}\vec{u}_y + F_{i,z}\vec{u}_z$$

Dado que un vector es cero, cuando cada una de sus componentes es cero, se tiene:

1. $\sum_{i=1}^{n} \vec{F}_{i,x} = 0$
2. $\sum_{i=1}^{n} \vec{F}_{i,y} = 0$
3. $\sum_{i=1}^{n} \vec{F}_{i,z} = 0$

Un sólido rígido está en equilibrio de traslación cuando la suma de las componentes de las fuerzas que actúan sobre él es cero.

- Un sólido rígido está en equilibrio de rotación, si la suma de momentos sobre el cuerpo es cero.

$$\sum_{i=1}^{n} \vec{M}_i = 0 \qquad (2)$$

En el espacio se tienen las tres ecuaciones una por dimensión; tiene un razonamiento similar al de las fuerzas:

$$\vec{M}_i = M_{i,x}\vec{u}_x + M_{i,y}\vec{u}_y + M_{i,z}\vec{u}_z$$

Resultando:

1. $\sum_{i=1}^{n} \vec{M}_{i,x} = 0$
2. $\sum_{i=1}^{n} \vec{M}_{i,y} = 0$
3. $\sum_{i=1}^{n} \vec{M}_{i,z} = 0$

Un sólido rígido está en equilibrio de rotación cuando la suma de las componentes de los momentos que actúan sobre él es cero.

Un sólido rígido está en equilibrio si está en equilibrio de traslación y de rotación.

Para comprender mejor cómo se plantea un sistema de ecuaciones, en la Figura 2, se muestra una bicicleta estacionada la cual está sometida a un sistema de fuerzas en equilibrio; el peso, dirigido hacia el centro de la Tierra y las reacciones de ésta sobre los neumáticos. Para que se mueva, se debe vencer primero su inercia y después el rozamiento aplicándole un impulso, ese impulso es el que permite avanzar los primeros metros. Si el impulso es débil, las fuerzas descritas son iguales, permanecen en equilibrio y la bicicleta se mantiene inmóvil. Si el impulso es suficiente, la suma total arrojará su resultado a favor del impulso y empezará a moverse. Factores que se tienen en cuenta para generar el sistema de ecuaciones:

Peso Wb: es la medida de la fuerza que ejerce la gravedad sobre un cuerpo y depende de la masa del mismo.

Centro de gravedad: es la posición donde se puede considerar actuando la fuerza de gravedad neta.

Figura 2. Sistema de fuerzas en equilibrio

Momento de una fuerza: es el producto de la magnitud de la fuerza por la distancia perpendicular desde el punto de apoyo a la línea de acción de dicha fuerza.

El sistema de ecuaciones resulta así:

$\sum_{i=1}^{n} \vec{F}_{i,y} = 0$ R1 + R2 –Wb = 0 (3)

$\sum_{i=1}^{n} \vec{M}_{i,z} = 0$ R2***a** – R1***b** = 0 (4)

Donde **a** es la distancia entre R1-Wb y **b** es la distancia entre Wb-R2 de la Figura 2.

Se conoce un tipo particular de equilibrio mecánico llamado equilibrio estático que correspondería a una situación en que el cuerpo está en reposo, con velocidad cero. Una hoja de papel sobre un escritorio está en equilibrio mecánico y estático, un paracaidista cayendo a velocidad constante, dada por la velocidad límite estaría en equilibrio mecánico pero no estático.

Existen métodos de solución de este tipo de problemas de estática mediante gráficos, heredados de los tiempos en que la complejidad de la solución de sistemas de ecuaciones se evitaba mediante la geometría, si bien actualmente se tiende al cálculo por computador.

1.2.3 Hiperestática

En estática, una estructura es **hiperestática** o estáticamente indeterminada cuando está en equilibrio pero las ecuaciones de la estática resultan insuficientes para determinar todas las fuerzas internas o las reacciones. (Una estructura en equilibrio estable que no es hiperestática es **isoestática**). Existen diversas formas de hiperestaticidad:

- Una estructura es **internamente hiperestática** si las ecuaciones de la estática no son suficientes para determinar los esfuerzos internos de la misma.

- Una estructura es **externamente hiperestática** si las ecuaciones de la estática no son suficientes para determinar fuerzas de reacción de la estructura al suelo o a otra estructura.

Una estructura es **completamente hiperestática** si es internamente y externamente hiperestática.

1.2.4 Aplicaciones de la Estática

La estática abarca el estudio del equilibrio tanto del conjunto como de sus partes constituyentes, incluyendo las porciones elementales de material.
Uno de los principales objetivos de la estática es la obtención de esfuerzos cortantes, fuerza normal, de torsión y momento flector a lo largo de una pieza, que puede ser desde una viga de un puente o los pilares de un rascacielos.
Su importancia reside en que una vez trazados los diagramas y obtenidas sus ecuaciones, se puede decidir el material con el que se construirá, las dimensiones que deberá tener, límites para un uso seguro, entre otros, mediante un análisis de materiales. Por tanto, resulta de aplicación en ingeniería estructural, ingeniería mecánica, construcción, siempre que se quiera construir una estructura fija. Para el análisis de una estructura en movimiento es necesario considerar la aceleración de las partes y las fuerzas resultantes.

El estudio de la Estática suele ser el primero dentro del área de la ingeniería mecánica, debido a que los procedimientos que se realizan suelen usarse a lo largo de los demás cursos de ingeniería mecánica. En esta rama de la mecánica se generan diversas situaciones en las que la solución se da de manera matemática, en la mayoría de los problemas se obtiene un elevado número de ecuaciones con el mismo número de variables, creando la necesidad de desarrollar un software para darle solución al problema de la forma más rápida.

1.3 ANTECEDENTES

- ✓ **CREACIÓN DE OBJETOS EN C++ PARA LA SOLUCIÓN DE PROBLEMAS DE INGENIERÍA:** Este proyecto de grado fue desarrollado por los estudiantes de Ingeniería Mecánica de la UIS, Elkin Arroyo y Pedro José Díaz, en el año de 1997. Su objetivo fue diseñar y construir los objetos básicos necesarios, que permitan hacer un análisis utilizando el método de elementos finitos. Crearon una serie de objetos que permiten las

operaciones entre vectores y matrices dando como resultado dos clases padres, la clase **Vector**, y la clase **Matrix**, la clase vector se genera con un puntero y una variable entera que almacena el tamaño del arreglo. A diferencia de la clase **Vector**, la clase **Matrix** es una clase abstracta que controlará a sus hijas, la clase **DensMatrix**, la clase **SpaMatrix**, y la clase **DiagMatrix**. Se desarrollaron librerías básicas necesarias para atacar cualquier problema de ingeniería, usando el método de elementos finitos, también dentro de los objetos desarrollados en C++, se incluyen formas a la solución de sistemas de ecuaciones lineales, a través de una eliminación gaussiana con substitución regresiva, también con el cálculo de la matriz inversa.

- ✓ **SOLVERLIB:** Es una librería actualizada por el profesor David Fuentes de la Escuela de Ingeniería Mecánica de la UIS, con el propósito de dar solución a problemas de ingeniería, se incluyen formas a la solución de sistemas de ecuaciones no lineales, utilizando el método de Newton Raphson.

- ✓ **TEVALUAR:** Hace parte de un listado del código de fuente de la versión 2.0 del programa INTERCAM desarrollado en el proyecto de grado "**CONDENSADORES DE VAPOR TEORÍA, DISEÑO Y SISTEMATIZACIÓN", Tomo 2**, por el estudiante Oscar Iván Del Rio Guzmán en el año de 1990 en la UIS. Este código fue adaptado para clases y para C++ por el profesor David Fuentes.

  La clase TEvaluar es un analizador de expresiones matemáticas para reales, operadores, funciones y variables. En el parámetro `prog` se pasa la cadena a ser evaluada y el resultado se devuelve en el parámetro `result`. Si ocurre algún error durante el análisis, se devuelve un mensaje de lo que ha ocurrido en `msj_err` y `result` se hace 0.0. La rutina llamadora debe encargarse de chequear si hay algún error.

- **XGRAPH2005:** Graficar grandes cantidades de datos técnicos puede ser una tarea frustrante. Se pueden encontrar muchos controles de diagramación con efectos y características inútiles, pero en lo que se refiere a graficar muchas curvas de inmediato, independientemente escaladas en ejes diferentes, la mayor parte de ellos fallan. Por esto la clase CXGraph es muy eficiente. CXGraph se creó para visualizar una serie ilimitada de datos en ejes múltiples. El autor de la librería XGRAPH2005 Gunnar Bolle desarrollador de web, permite el libre uso descrito de la siguiente manera:

```
////////////////////////////////////////////////////////////
// Main header file for the CXGraph charting control
//
//You are free to use, modify and distribute this source, as long
as
// there is no charge, and this HEADER stays intact. This source is
// supplied "AS-IS", without WARRANTY OF ANY KIND, and the user
// holds me blameless for any or all problems that may arise
// from the use of this code.
//
// Expect bugs.
//
// Gunnar Bolle (webmaster@beatmonsters.com)
//
// Known issues
// - clipping in print preview doesn't work
//
// Additional credits
//
// - Ishay Sivan (Bugfixing)
// - Pavel Klocek (Bugfixing)
// - James White (color button)
// - Chris Maunder (color popup)
// - Keith Rule (MemDC)
// - J.G. Hattingh (parts of Device Context Utilities)
// - Jörg Blattner (Bugfixing, improvements)
////////////////////////////////////////////////////////////
```

Características:

- Serie ilimitada de datos.
- Ejes ilimitados (la pantalla es el límite).
- Escalas logarítmicas.
- Auto escalado.

- ✓ Colocación manual o automática del eje.
- ✓ La mayoría de aspectos visuales pueden ser personalizados.
- ✓ Leyendas.
- ✓ Bitmaps.
- ✓ Cursor (agrega anotaciones de datos haciendo clic derecho).
- ✓ Zoom (ampliación de la gráfica).
- ✓ Traslado de la gráfica.
- ✓ Rangos de color.
- ✓ Selección del objeto con el ratón.
- ✓ Tendencia lineal.
- ✓ Tendencia cúbica.
- ✓ Tendencia de polinomio de enésimo orden.
- ✓ Impresión de datos.
- ✓ Promedio variable.
- ✓ Guardar y cargar propiedades, datos y ejes.
- ✓ Marcadores de ejes.
- ✓ Soporte para Windows.
- ✓ Relleno entre curvas.
- ✓ Sincronización del cursor.
- ✓ Edición de datos.
- ✓ Modo de medida.

✓ **GRIDCTRLLIB:** Es una librería creada por el programador canadiense Chris Maunder uno de los fundadores principales de “The Code Project”. Se creó la clase CGridCtrl para la representación visual y edición de datos dispuestos en tablas.

El autor permite el libre uso de la librería **GRIDCTRLLIB** descrito de la siguiente manera:

```
////////////////////////////////////////////////////////////////
// MFC Grid Control v2.20
// Written by Chris Maunder <cmaunder@mail.com>
// Copyright (c) 1998-2000. All Rights Reserved.
//
// The code contained in this file is based on the original
// WorldCom Grid control written by Joe Willcoxson,
//        mailto:chinajoe@aol.com
//        http://users.aol.com/chinajoe
// (These addresses may be out of date) The code has gone through
so many modifications that I'm not sure if there is even a single
original line of code. In any case Joe's code was a great framework
on which to build.
//
// This code may be used in compiled form in any way you desire.
This file may be redistributed unmodified by any means PROVIDING it
is not sold for profit without the authors written consent, and
providing that this notice and the authors name and all copyright
notices remains intact.
//
// An email letting me know how you are using it would be nice as
well.
// This file is provided "as is" with no expressed or implied
warranty. The author accepts no liability for any damage/loss of
business that this product may cause.
//
// Expect bugs!
//
// Please use and enjoy, and let me know of any
bugs/mods/improvements that you have found/implemented and I will
fix/incorporate them into this file.
//
////////////////////////////////////////////////////////////////
```

Características de control:

- ✓ La selección de celdas puede estar deshabilitada.
- ✓ Ajuste del tamaño de la fila y Columna. El dimensionamiento puede estar deshabilitado para fila, columnas o ambos.
- ✓ Auto dimensionamiento de filas o de columnas.
- ✓ Cualquier número de columnas y filas fijas.
- ✓ En celdas individuales, el texto y el fondo pueden tener colores diferentes.
- ✓ Las celdas individuales pueden tener tipos de letra diferentes.
- ✓ Las celdas individuales pueden ser de sólo lectura.

- ✓ Arrastre y colocación de celdas.
- ✓ Edición del contenido de la celda.
- ✓ La edición puede ser deshabilitada.
- ✓ Líneas de la cuadrícula opcionales.
- ✓ Imágenes en cualquier celda.
- ✓ El modo opcional "lista", que selecciona una fila completa con un solo clic en el encabezado de la columna.
- ✓ Soporte Unicode.
- ✓ Títulos de información para celdas que son muy pequeñas para exhibir sus datos.
- ✓ Ocultar columnas y filas

✓ **CRYSTAL EDIT-SYNTAX COLORING TEXT EDITOR:** Es una librería de edición de texto, que le proporciona color, consta de tres clases principales:

La clase **CCrystalTextBuffer** esta encargada de almacenar líneas, cargar y borrar texto para un archivo. Para simplificar implementaciones de comandos **Undo / Redo**, cada operación de edición, es dividida dentro de una secuencia de acciones de insertar y borrar texto. Consecuentemente, las clases derivadas de CView (clase que incluye una vista básica de un documento) sólo intentan responder a estas operaciones primitivas. La clase **CCrystalTextView** es la encargada de visualizar en la ventana el texto. Se deriva de la clase CView, y proporciona color al texto, funciones para el resaltado de sintaxis, diferentes tipos de selecciones del texto, movimientos del cursor, el diálogo **Buscar** etc. Sin embargo esta clase, no permite realizar cambios al texto.

Las vistas derivadas de CCrystalTextView son usualmente usadas con el objeto CCrystalTextBuffer. Una vez la vista está relacionada al objeto CCrystalTextBuffer, es capaz de seguir los cambios hechos en el texto.

La clase **CCrystalEditView** esta derivada de la clase CCrystalTextView. A diferencia de la anterior clase, la cuál sólo puede mostrar el texto y actualizar la vista cuando es necesario, CCrystalEditView tiene funciones para realizar toda clase de edición, incluyendo el diálogo **Reemplazar**, arrastrar y colocar texto. En la vista no se hacen los cambios de texto directamente, en lugar de eso, se cambia el comando dentro de una secuencia de operaciones primitivas descritas anteriormente, y las delega al objeto CCrystalTextBuffer. Una vez que los cambios son realizados, el objeto CCrystalTextBuffer actualiza todas las vistas que se conecten a él.

El autor Stcherbatchenko Andrei permite el libre uso de la librería **CRYSTAL EDIT** descrito de la siguiente manera:

```
////////////////////////////////////////////////////////////////////
//    File:       CCrystalEditView.h
//    Version:    1.0.0.0
//    Created:    29-Dec-1998
//
//    Author:           Stcherbatchenko Andrei
//    E-mail:           windfall@gmx.de
//
//    Interface of the CCrystalEditView class, a part of Crystal
Edit - syntax
//    coloring text editor.
//
//   You are free to use or modify this code to the following
restrictions:
//    - Acknowledge me somewhere in your about box, simple "Parts
of code by.."
//    will be enough. If you can't (or don't want to), contact me
personally.
//    - LEAVE THIS HEADER INTACT
////////////////////////////////////////////////////////////////////
```

Limitaciones y desventajas de la librería:

- ✓ Sólo el tipo de letra Courier es soportado.
- ✓ Ningún soporte para negrita y letra itálica en elementos de sintaxis.
- ✓ Ningún soporte para la selección de la columna.

- **APLICACIONES COMERCIALES SIMILARES:** a continuación se menciona varios software similares que evalúan sistemas de ecuaciones no lineales:

  TK SOLVER: Es un software creado por Ahl, David H. en el año de 1983, TKSolver es un lenguaje de programación declarativo, solucionador algebraico de ecuaciones, un solucionador iterativo de ecuaciones, y estructurado, basado en objetos de interfaz. La interfaz comprende nueve clases de objetos que pueden ser compartidos y anexarse en otros archivos TK:

  - **Reglas**: ecuaciones, fórmulas, llamadas a funciones que pueden incluir las condiciones lógicas.

  - **Variables**: una lista de las variables que se utilizan en las normas, junto con los valores (numéricos o no numéricos) que han sido introducidos por el usuario o calculado por el software.

  - **Unidades**: unidades de todos los factores de conversión, en un solo lugar, para permitir la actualización automática de los valores cuando se cambian las unidades.

  - **Tablas**: Las colecciones de listas presentadas conjuntamente.

  - **Gráficos**: Los diagramas lineales, los gráficos de dispersión, los gráficos de barras, y los gráficos circulares.

  - **Funciones**: basadas en reglas, cuadro de búsqueda, y de programación de componentes.

  - **Comentarios:** para la explicación y documentación.

Utiliza tres métodos incluyendo el de Newton-Raphson para resolver el sistema de ecuaciones. TKSolver tiene alrededor de 150 funciones incorporadas: matemáticas, trigonométricas, booleanos, cálculo numérico, acceso a bases de datos, y funciones de programación.

El TKSolver tiene un costo en el mercado de $ 400 dólares o más según su versión.

ENGINEERING EQUATION SOLVER (EES): es un software desarrollado por dos profesores, el doctor William Beckman y el doctor Sanford Klein, ambos de la Universidad de Wisconsin. Su experiencia en la enseñanza de la ingeniería mecánica, en la termodinámica y transferencia de calor, puso en manifiesto que los estudiantes gastan demasiado tiempo buscando información de propiedades y resolviendo ecuaciones para problemas de sus tareas. Problemas prácticos interesantes no podían ser asignados a causa de la complejidad matemática de la solución numérica del conjunto de ecuaciones algebraicas. El doctor Beckman y el doctor Klein han creado el EES para permitir al usuario concentrarse más en el diseño que en tediosas tareas como buscar información de propiedades y la solución de ecuaciones. EES es un software que brinda la solución numérica de un conjunto de ecuaciones algebraicas. Sus principales características son:

- ✓ Resolver ecuaciones diferenciales e integrales.
- ✓ Realizar análisis de regresión lineal y no lineal, etc.
- ✓ Incluye funciones de las propiedades termodinámicas y psicrométricas, útiles para cálculos de ingeniería: tablas de vapor, refrigerantes, amoníaco, metano, anhídrido carbónico y otros fluidos.
- ✓ Tablas paramétricas que funcionan como una hoja de cálculo.
- ✓ Visualizar gráficas de calidad.
- ✓ Resuelve simultáneamente no más de 6000 ecuaciones no lineales.
- ✓ Unidades SI e Inglés.

La versión demo del EES puede ser descargada en la siguiente dirección: http://www.mhhe.com/engcs/mech/ees/download.html

MathCad: fue desarrollado por la compañía Mathsoft con sede en Cambridge (Massachussets, USA), caracterizada por proporcionar soluciones de cálculo para ingeniería a instituciones académicas, centros de investigación y la industria, destacando éstas por su potencia y sencillez de uso. MathCad es un software de cálculo extremadamente versátil y potente, que permite la solución a problemas de disciplinas como en Ingeniería, Arquitectura, Química, Física y Matemáticas entre otras, con una amplia variedad de herramientas y soporta una gran variedad de técnicas de análisis y visualización.

Mathcad es una herramienta técnica que incorpora todas las funcionalidades de interactividad propias de las hojas de cálculo, una extensa librería de funciones y una completa integración con muchas de las herramientas utilizadas en ingeniería (Excel, AutoCAD, MATLAB, Visio, ODBC, etc.). Sus prestaciones de documentación técnica, junto con una notación matemática, permiten que los cálculos sean auto-documentados y ahorrar así mucho tiempo al usuario a la hora de crear informes técnicos sobre los cálculos realizados.

MathCad se encuentra organizado como una hoja de trabajo, en las que las ecuaciones y expresiones se muestran gráficamente, no como simple texto. Dentro de las capacidades de MathCad se encuentran:

- ✓ Resolver ecuaciones diferenciales con varios métodos numéricos.
- ✓ Graficar funciones en dos o tres dimensiones.
- ✓ El uso del alfabeto griego (letras griegas mayúsculas y minúsculas).
- ✓ Cálculo de expresiones simbólicas.
- ✓ Operaciones con arreglos (vectores y matrices).

- ✓ Solución simbólica de un sistema de ecuaciones.
- ✓ Encontrar la gráfica (la curva de tendencia) de un grupo de datos.
- ✓ Implementación de subprogramas.
- ✓ Encontrar raíces de polinomios y funciones.
- ✓ Funciones estadísticas y distribuciones de probabilidad.

2 PLANTEAMIENTO DE LA SOLUCIÓN AL PROBLEMA

2.1 SISTEMAS ECUACIONES LINEALES Y NO LINEALES

Un **sistema de ecuaciones lineales** está compuesto por planteamientos de igualdad, involucrando una o más variables a la primera potencia, que no contiene productos entre variables, involucra solamente sumas y restas entre las variables. Un **sistema de ecuaciones no lineales**, tiene planteamientos de igualdad, involucrando Potencias, Funciones Trigonométricas, Hiperbólicas, Logarítmicas, Exponenciales, entre otras. Utilizando operadores como Suma, Resta, División, Multiplicación entre variables.

2.2 SOLUCIÓN DE SISTEMAS DE ECUACIONES NO LINEALES

2.2.1 Introducción

En muchas ocasiones aparece la necesidad de resolver sistemas de ecuaciones algebraicas no lineales de la forma:

$$f_1(x_1, x_2 \dots, x_n) = 0 \quad (5)$$

$$f_2(x_1, x_2 \dots, x_n) = 0 \quad (6)$$

$$\vdots$$

$$f_n(x_1, x_2 \dots, x_n) = 0 \quad (7)$$

Encontrar las raíces de un sistema de ecuaciones es encontrar los valores de las variables del sistema de ecuaciones, tal que al sustituir dichos valores en las ecuaciones del sistema, cada una de ellas sea igual a cero. Una de las dificultades asociadas con estos sistemas es que no es fácil saber si existen raíces y cuántas son. Existen métodos que bajo ciertas condiciones son capaces de encontrar una de las raíces de un sistema de ecuaciones.

2.2.2 Historia

Se consideran ecuaciones no lineales de la forma: $f(x) = 0$ donde f es una función real de variable real. Un caso sencillo es cuando la función $f(x)$ es un polinomio. Los polinomios de primero y segundo grado fueron tratados con éxito en tiempos muy remotos, los babilonios eran capaces de resolver ecuaciones cuadráticas. La historia del descubrimiento de la solución algebraica cúbica, enfrentó a dos grandes rivales italianos: Cardano y Tartaglia hacia 1540, y Ferrari, alumno y secretario de Cardano, resolvió en 1545 la ecuación de cuarto grado. Posteriormente fueron muchos los matemáticos eminentes que trataron de resolver las ecuaciones de grado superior a cuatro, aunque en vano puesto que el matemático noruego Abel en 1893 demostró que es imposible resolver por radicales la ecuación general de grado mayor a cuatro. En consecuencia, para calcular las raíces de polinomios de grado mayor que cuatro es imprescindible usar técnicas numéricas[1].

El método de Newton fue descrito por Isaac Newton en *De analysi per aequationes número terminorum infinitas* (escrito en 1669, publicado en 1711 por William Jones) y en *De metodis fluxionum et serierum infinitarum* (escrito en 1671, traducido y publicado como *Método de fluxiones* en 1736 por John Colson). Sin embargo, su

[1] VADILLO, Fernando. Ecuaciones no lineales p. 1-2.

descripción difiere en forma sustancial de la descripción moderna, Newton aplica el método sólo a polinomios. El método no computa las aproximaciones sucesivas x_n, sino que computa una secuencia de polinomios y al final llega a la aproximación de la raíz x. Newton ve el método como puramente algebraico y falla al no ver la conexión con el cálculo. Isaac Newton probablemente deriva su método de forma similar aunque menos precisa que el método de François Viète.

2.2.3 El Método de Newton-Raphson (NR)

En un principio las soluciones analíticas de ecuaciones no lineales eran directas y no iterativas, pero excesivamente largas en su solución y con acumulación de errores. En muchas situaciones, en especial en problemas de estática, plantear una solución directa sin iterar para un grupo de *n* ecuaciones y *n* variables casi siempre suele ser imposible de alcanzar. Es decir, hallar las soluciones de las ecuaciones es una tarea ardua sin la ayuda de un algoritmo.

Cualquier método de solución iterativa requiere que se proporcione al comienzo del cálculo uno o más valores iniciales. Estos valores iniciales se utilizan para obtener una nueva solución. Este proceso de iteración se repite hasta que converja en una solución muy próxima a la correcta. ¿Cuál es el peligro? La solución puede no ser única.

Un algoritmo que proporciona una solución que converge con la considerada correcta y muy útil para la solución de sistemas de ecuaciones generados por problemas en la Estática es el método de **Newton- Raphson**. Éste método resuelve sistemas de ecuaciones no lineales para aproximaciones sucesivas a partir de una configuración inicial dada para cada una de las variables involucradas en el sistema de ecuaciones y del entorno de resultados obtenidos en el paso anterior.

2.2.3.1 Funcionamiento del Algoritmo

Una función no lineal puede tener múltiples raíces de las cuales una se define como la intersección de la función con cualquier recta. Usualmente se toma el eje cero de la variable independiente como línea recta.

A continuación se muestra cómo funciona el algoritmo para determinar las raíces de una función no lineal de variable independiente:
En la Figura 3 se tiene:

- ✓ Una función cuyo valor depende de x: $y = f(x)$
- ✓ Elección de un valor $x1$ inicial: $y1 = f(x1)$
- ✓ NR evalúa la función para este valor de $x1$ hallando $y1$.

El valor $y1$ se compara con una tolerancia seleccionada a priori, para ver si la diferencia entre el valor buscado de y, y el valor obtenido en $y1$ es bastante cercano a cero. Si es así, $x1$ será una raíz idónea.

Si la tolerancia escogida es mucho menor que la diferencia obtenida entre el valor obtenido de $y1$ y el valor de y buscado, se debe repetir la iteración.

¿Cómo se busca un nuevo valor inicial?

- ✓ Se evalúa la pendiente de la función $f(x)$ en el punto $x1$, y $y1$ se calcula utilizando una expresión analítica para la derivada de la función.
- ✓ Se calcula la pendiente de la función $f(x)$ en el punto ($x1$, $y1$).

- ✓ La ecuación de la recta tangente se evalúa para encontrar la intersección en el eje x, este punto $x2$, se utiliza como nuevo valor inicial.

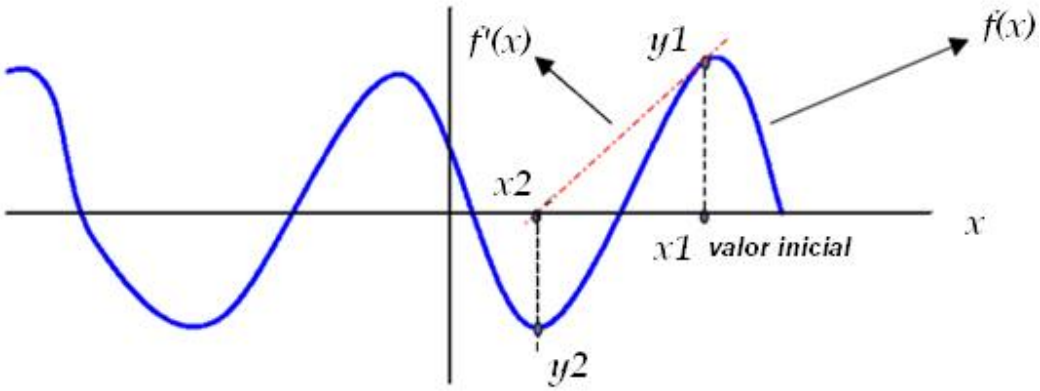
- ✓ Se comprueba de nuevo la tolerancia y se procede en consecuencia.

Siendo:

$x2 = x1 - (y1/m)$

$m = f'(x)$

Figura 3. Algoritmo NR en una función $f(x)$



Problemas que presenta la aplicación de este algoritmo:

- ✓ Puede no converger.
- ✓ Es sensible al valor inicial.
- ✓ La función debe ser diferenciable.

Ventajas del algoritmo Newton-Raphson:

- ✓ Las derivadas son continuas en la región de la raíz, dependiendo de la función.
- ✓ Determinación multidimensional de las raíces.

De un sistema de ecuaciones no lineales, se define:

- La función $y_i = f(x_i)$, siendo i un número entero.
- La pendiente $m = f'(x_i)$.
- El error $\Delta x = x_{(i+1)} - x_i$, si Δx tiende a cero la solución converge.

Si no converge, se debe hallar el siguiente valor inicial $x_{(i+1)}$ mediante las expresiones:

$x_{(i+1)} = x_i - \frac{y_i}{m}$ si se despeja m se tiene:

$m\,(x_{(i+1)} - x_i) = -y_i$

Por lo tanto queda la ecuación:

$$f'(x_i).\Delta x = -f(x_i) \qquad (8)$$

Interpretando estas ecuaciones, y ampliando el resultado a un sistema de ecuaciones no lineales, para cualquier problema a solucionar por NR se tendrá un conjunto de ecuaciones definido por:

$$\begin{pmatrix} f_1(x_i) \\ . \\ . \\ . \\ f_n(x_i) \end{pmatrix} = \vec{B} = residuo\ de\ las\ ecuaciones.$$

Para hallar los términos de la pendiente *m*, se requieren las derivadas parciales de la función en cada punto:

$$\begin{pmatrix} \frac{\delta f_1}{\delta x_1} & \frac{\delta f_1}{\delta x_2} & \frac{\delta f_1}{\delta x_3} & \frac{\delta f_1}{\delta x_4} & \frac{\delta f_1}{\delta x_5} \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\delta f_n}{\delta x_1} \vdots & \frac{\delta f_n}{\delta x_2} & \frac{\delta f_n}{\delta x_3} & \frac{\delta f_n}{\delta x_4} & \frac{\delta f_n}{\delta x_5} \dots \end{pmatrix} = \vec{A}$$

$\vec{A}$, matriz de derivadas parciales del sistema de las ecuaciones o matriz jacobiana del sistema.

$$\begin{pmatrix} \Delta x_1 \\ \vdots \\ \Delta x_n \end{pmatrix} = X = términos\ de\ error.$$

Si $\Delta x = x_i - x_{(i+1)}$, para un sistema de ecuaciones según (8) queda de la forma:

$$A.X = B$$
$$X = [A]^{-1}\{B\} \qquad (9)$$

Donde *A* y *B* se calculan para cualquier valor inicial supuesto. Se debe dar un criterio de convergencia adecuado.

Las ecuaciones se resolverán simultáneamente teniendo en cuenta el criterio de error escogido. Después de cada iteración se debe comparar el error con la tolerancia.

Se debe tener en cuenta que el sistema de ecuaciones sólo tiene solución si el determinante de la matriz jacobiana es diferente de cero.

En definitiva, se puede observar que la solución de un sistema de ecuaciones no lineales mediante el método de Newton se reduce en cada iteración, a la solución de un sistema de ecuaciones lineales.

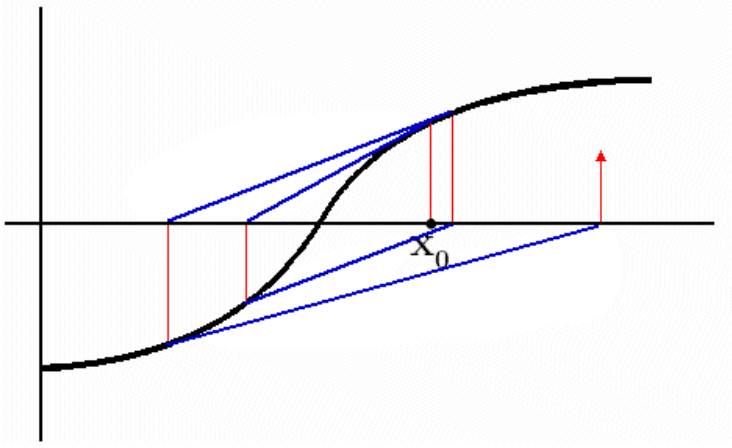
2.2.3.2 Análisis de la convergencia del Método de Newton-Raphson:

Se debe tener en cuenta que la naturaleza de la función puede originar diferentes dificultades, llegando a hacer que el método no converja.

Observaciones del método:

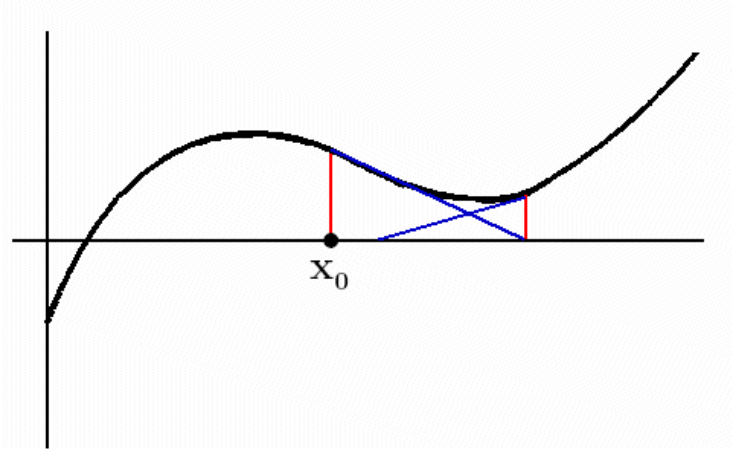
a) Si existe un punto de inflexión en las proximidades de la raíz, las iteraciones divergen progresivamente como en la Figura 4.

Figura 4. Observación a)



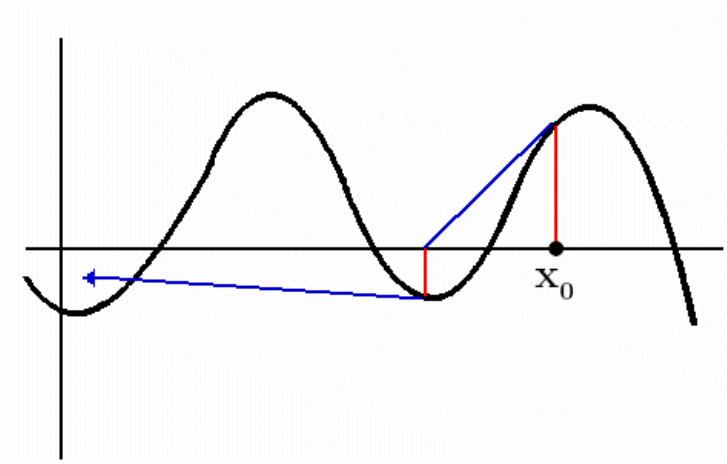
b) El método de Newton-Raphson oscila en los alrededores de un máximo o un mínimo local, persistiendo o llegando a encontrarse con pendientes cercanas a cero, en cuyo caso las iteraciones se alejan del área de interés (Figura 5).

Figura 5. Observación b)



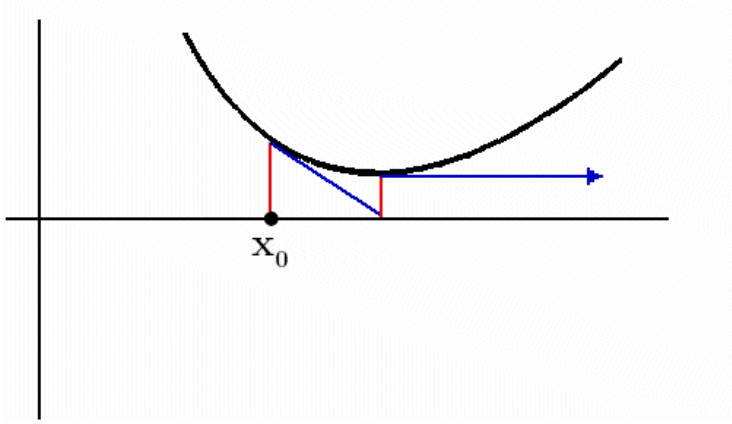
c) Un valor inicial a una raíz como en la Figura 6, puede converger a otra raíz muy distante de la anterior como consecuencia de encontrarse pendientes cercanas a cero.

Figura 6. Observación c)



d) Una pendiente nula (Figura 7), provoca una división por cero (geométricamente, una tangente horizontal que jamás corta al eje de abscisas).

Figura 7. Observación d)



En general aunque el método de Newton-Raphson es muy eficiente, existen situaciones en que presenta dificultades. Un caso especial es en el de las raíces múltiples. En algunos casos es posible que para raíces simples se presenten dificultades por su lenta convergencia.

2.2.4 Ejemplo de aplicación[2]

Se tiene el siguiente sistema de ecuaciones no lineales:

$$x^2 - 2x - y + 0.5 = 0 \quad (10)$$

$$x^2 + 4y^2 - 4 = 0 \quad (11)$$

Puede considerarse como la ecuación $f(x) = 0$ (cero representa ahora al vector nulo, es decir, que $0 = (0,0)^T$), con $x = (x, y)^T$ y $f = (f1, f2)^T$ y si $f(x) = 0$ se escribe de la forma $X = F(x)$, se obtiene $F(x) = X + f(x)$ siendo que:

$$f1(x) = x^2 - 2x - y + 0.5$$

$$f2(x) = x^2 + 4y^2 - 4$$

[2] GAVALA, Javier C. Algebra Numérica. p. 34-38

Una vez transformado el sistema en ecuaciones del tipo $f(x) = 0$, se puede resolver por el método de NR. Despejando x y y, se obtiene:

$$x^2 - 2x - y + 0.5 = 0 \;\rightarrow 2x = x^2 - y + 0.5 \quad \rightarrow x = \frac{x^2 - y + 0.5}{2}$$
$$x^2 + 4y^2 - 4 = 0 \;\rightarrow\; x^2 + 4y^2 + y - 4 = y \;\rightarrow\; y = x^2 + 4y^2 + y - 4$$

Es decir:

$$X = F(x)\ con\ \begin{cases} x = (x, y)^T \\ y \\ F(x) = (\frac{x^2 - y + 0.5}{2}, x^2 + 4y^2 + y - 4)^T \end{cases}$$

Donde X es una solución de $F(x)$ y $X_{(i+1)}$ una aproximación obtenida. Tomando como solución inicial $X_0 = (-0.5, 1)^T$ se tiene que:

$$f(x_0) = (0.75, 0.25)^T$$
$$J(x) = \begin{pmatrix} 2x - 2 & -1 \\ 2x & 8y \end{pmatrix} \rightarrow f'(x_0) = J(x_0) = \begin{pmatrix} -3 & -1 \\ -1 & 8 \end{pmatrix}$$

Por lo que se debe resolver el sistema:

$$J(x_0).(\Delta_{x1}) = -f(x_0)$$

$$\begin{pmatrix} -3 & -1 \\ -1 & 8 \end{pmatrix} \begin{pmatrix} \Delta_{x1}^1 \\ \Delta_{x2}^1 \end{pmatrix} = \begin{pmatrix} -0.75 \\ -0.25 \end{pmatrix}$$

Cuya solución es $\Delta_{x1} = \begin{pmatrix} \Delta_{x1}^1 \\ \Delta_{x2}^1 \end{pmatrix} = \begin{pmatrix} 0.25 \\ 0 \end{pmatrix}$ y, por lo tanto:

$$X_1 = X_0 + \Delta_{x1} = \begin{pmatrix} -0.25 \\ 1 \end{pmatrix}$$

Realizando una nueva iteración se obtiene:

$$f(x_1) = \begin{pmatrix} 0.0625 \\ 0.0625 \end{pmatrix} \qquad f'(x_1) = J(x_1) = \begin{pmatrix} -0.25 & -1 \\ -0.5 & 8 \end{pmatrix}$$

Se obtiene el sistema:

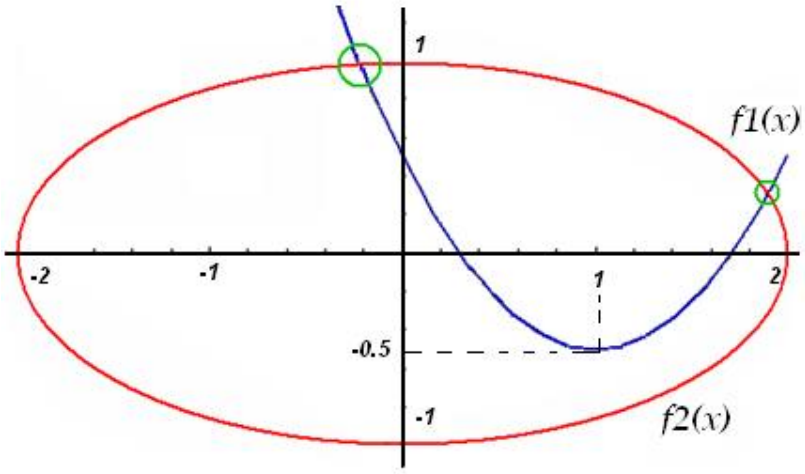
$$\begin{pmatrix} -0.25 & -1 \\ -0.5 & 8 \end{pmatrix} \begin{pmatrix} \Delta_{x1}^1 \\ \Delta_{x2}^1 \end{pmatrix} = \begin{pmatrix} -0.0625 \\ -0.0625 \end{pmatrix}$$

Cuya solución es $\Delta_{x2} = \begin{pmatrix} \Delta_{x1}^1 \\ \Delta_{x2}^1 \end{pmatrix} = \begin{pmatrix} 0.022561 \ldots \\ -0.006 \ldots \end{pmatrix}$ y, por tanto:

$$X_2 = X_1 + \Delta_{x2} = \begin{pmatrix} -0.227439 \ldots \\ 0.994 \ldots \end{pmatrix}$$

Con dos iteraciones se tiene que $\Delta_x \approx 0$, es decir converge la solución. En la Figura 8 se observa dos puntos de intersección de las funciones $f1(x)$ y $f2(x)$, quiere decir que existen dos soluciones al anterior sistema de ecuaciones no lineales. Los puntos en la gráfica son: (1.901, 0.3112) y (-0.222, 0.9938).

Figura 8. Intersección de las funciones $f1(x)$ y $f2(x)$



2.2.5 Otros Métodos Iterativos para Sistemas de Ecuaciones No Lineales

El método de Newton es el más sencillo de los métodos disponibles para resolver sistemas de ecuaciones no lineales. Una clase de métodos ampliamente utilizada son los denominados métodos cuasi-Newton de los cuales un ejemplo es el denominado método de Broyden.

Estos métodos intentan evitar la dificultad que supone tener que calcular (o evaluar) en cada paso de la iteración la matriz jacobiana y su inversa. Si se supone en primer lugar que $f(x) = 0$ es un sistema de ecuaciones lineales $f(x) = Ax - b = 0$, entonces si se restan los valores en dos puntos de un proceso iterativo para obtener la solución se tiene:

$$f(x^k) - f(x^{k-1}) = A(x^k - x^{k-1}) \tag{12}$$

En el caso no lineal esta igualdad no es cierta, pero suponiendo que se elija adecuadamente A_k de modo que:

$$A_k(x^k - x^{k-1}) \cong f(x^k) - f(x^{k-1}) \tag{13}$$

Los métodos cuasi-Newton construyen una sucesión A_k que aproxime de la mejor manera posible a la matriz jacobiana $J(x^k)$.

Otros métodos que tienen mejores propiedades de convergencia global son los métodos de minimización. Estos métodos se basan en la observación de que el problema de encontrar la solución de un sistema de ecuaciones lineales o no lineales puede transformarse en uno de minimización. Por ejemplo, el problema de encontrar los ceros de la función $f(x)$ es equivalente a minimizar la función escalar:

$$S(x_1, \ldots, x_n) = f_1^2(x_1, x_2 \ \ldots, x_n) + \ldots + f_n^2(x_1, x_2 \ \ldots, x_n)$$

Estas u otras identificaciones permiten utilizar algoritmos de minimización para buscar ceros a sistemas de ecuaciones lineales o no lineales.

2.3 LENGUAJE C++

La historia del lenguaje de programación C++ comienza en los años 70, con un programador de nombre Dennis Ritchie que trabajaba en los laboratorios de AT&T (American Telephone and Telegraph). Trabajando con un lenguaje llamado BCPL inventado por Martin Richards (que luego influyó para crear el lenguaje B de Ken Thompson), Dennis deseaba un lenguaje que le permitiese manejar el hardware de la misma manera que el ensamblador pero con algo de programación estructurada como los lenguajes de alto nivel. Fue entonces que creó el lenguaje C que primeramente corría en computadoras con el sistema operativo UNIX. Pero los verdaderos alcances de lo que sería éste, se verían poco tiempo después cuando Dennis volvió a escribir el compilador C de UNIX en el mismo C, y luego Ken Thompson (diseñador del sistema) escribió UNIX completamente en C y ya no en ensamblador[3].

En 1983, el Instituto Americano de Normalización (ANSI) se dio a la tarea de estandarizar el lenguaje C, aunque esta tarea tardó 6 años en completarse, y además con la ayuda de la Organización Internacional de Normalización (ISO) en el año de 1989 definió el C Estándar. Luego se dio pie para evolucionar el lenguaje de programación C. Fue en los mismos laboratorios de AT&T Bell, que Bjarnes Stroutstrup diseñó y desarrolló C++ buscando un lenguaje con las opciones de programación orientada a objetos (POO). En ese momento el

[3] OSORIO, Rojas Alan D. Manual teórico práctico. p. 10-14.

desarrollo del estándar de C++ acaparaba la atención de los diseñadores. En el año 1995, se incluyeron algunas bibliotecas de funciones al lenguaje C. Y con base en ellas, se pudo en 1998 definir el lenguaje estándar de C++.

"C" es un lenguaje de alto nivel, basado en funciones, que permite desarrollos estructurados. Entre otras muchas características contempla la definición de estructuras de datos, recursividad o indirecciones a datos o código (punteros). "C ++", por su parte, es un súper conjunto de "C", al que recubre con una capa de soporte a la POO. Permite por tanto la definición, creación y manipulación de objetos. El C++ es a la vez un lenguaje orientado a algoritmos y orientado a objetos. C++ como lenguaje se asemeja a C y es compatible con él, aunque ya se ha dicho que presenta ciertas ventajas. Como lenguaje orientado a objetos se basa en una filosofía completamente diferente, que exige del programador un completo cambio de mentalidad. Las características propias de la POO de C++ son modificaciones mayores que cambian radicalmente su naturaleza.

En el año de 1992, la compañía Microsoft introduce C/C++ 7.0 y la biblioteca de clases MFC (Microsoft Foundation Classes), las cuales tenían la finalidad de que el desarrollo de aplicaciones para Windows, escritas en C, fuera más fácil. Sin embargo, no resultó como esperaban, así que un año más tarde fue creado Visual C++ 1.0, que parecía más amigable a los desarrolladores, porque tenía una versión mejorada de las clases MFC. Con Visual C++ se introdujo una tecnología de desarrollo a base de asistentes. En general es un entorno de desarrollo diseñado para crear aplicaciones gráficas orientadas a objetos. A continuación, se presentan los grupos de clases más utilizadas de la MFC. Esta librería está compuesta por cientos de clases distintas, que desde el punto de vista funcional, las podemos dividir en 4 grupos:

1. Clases orientadas al interfaz de usuario: permiten representar ventanas, menús, diálogos, etc.

2. Clases de propósito general: representan ficheros, cadenas, datos de fecha y hora, etc.
3. Clases orientadas a bases de datos: representan tanto bases de datos como conjuntos de registros seleccionados después de una consulta sobre una tabla, etc.
4. Clases para manejo de excepciones: utilizadas para control avanzado de errores de ejecución.

2.4 PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

La POO es una nueva filosofía de programación que se basa en la utilización de objetos. El objetivo de la POO es la meta de cualquier modelo de programación estructurada convencional: "imponer" una serie de normas de desarrollo que aseguren y faciliten la mantenibilidad y reusabilidad del código. Los mecanismos básicos de la POO son: objetos, mensajes, métodos y clases.

- ✓ **Objetos:** un objeto es una entidad que tiene unos atributos particulares (datos) y unas formas de operar sobre ellos (los métodos o funciones miembro). Es decir, un objeto incluye, por una parte, una serie de operaciones que definen su comportamiento, y una serie de variables manipuladas por esas funciones que definen su estado. Por ejemplo, una ventana Windows contendrá operaciones como "maximizar" y variables como "ancho" y "alto" de la ventana.

- ✓ **Mensajes:** en C++, un mensaje se corresponde con el nombre de uno de los métodos de un objeto. Cuando se pasa un mensaje a un objeto, este responde ejecutando el código de la función asociada.

- ✓ **Método:** un método (función miembro) se implementa dentro de un objeto y determina cómo tiene que actuar el objeto cuando se produce el mensaje asociado. En C++ un método se corresponde con la definición de la función miembro del objeto.

- ✓ **Clases:** una clase es la definición de un tipo de objetos. De esta manera, una clase "Empleado" representaría todos los empleados de una empresa, mientras que un objeto de esa clase (también denominado instancia) representaría a uno de esos empleados en particular.

Las principales características de la **POO** son las siguientes:

- ✓ *Abstracción*: Es el mecanismo de diseño en la POO. Permite extraer de un conjunto de entidades datos y comportamientos comunes para almacenarlos en clases.

- ✓ *Encapsulamiento*: Mediante esta técnica se consigue que cada clase sea una caja negra, de tal manera que los objetos de esa clase se puedan manipular como unidades básicas. Los detalles de la implementación se encuentran dentro de la clase, mientras que desde el exterior, un objeto será simplemente una entidad que responde a una serie de mensajes públicos (también denominados interfaz de la clase).

- ✓ *Herencia*: Es el mecanismo que permite crear clases derivadas (especialización) a partir de clases bases (generalización). Es decir, podría tener la clase "Empleado" (clase base) y la clase "Vendedor" derivando de la anterior. Una librería de clases (como la MFC) no es más que un conjunto de definiciones de clases interconectadas por múltiples relaciones de herencia.

- ✓ *Polimorfismo:* Esta característica permite disponer de múltiples implementaciones de un mismo método de clase, dependiendo de la clase en la que se realice. Es decir, se puede acceder a una variedad de métodos distintos mediante el mismo mecanismo de acceso. En C++ el polimorfismo se consigue mediante la definición de clases derivadas, funciones virtuales y el uso de punteros a objetos.

2.5 DESCRIPCIÓN DEL CÓDIGO FUENTE DE STATICSOLVER 1.0

Haciendo uso de la teoría descrita, se desarrolló el software **StaticSolver 1.0** con la finalidad de complementar la asignatura Estática de la Escuela de Ingeniería Mecánica, debido a que es una herramienta que permite al profesor y al estudiante concentrarse más en el análisis de un determinado problema de la asignatura, que en resolver el sistema de ecuaciones planteadas para la solución del problema, pues es común invertir prolongados tiempos haciendo esta operación.

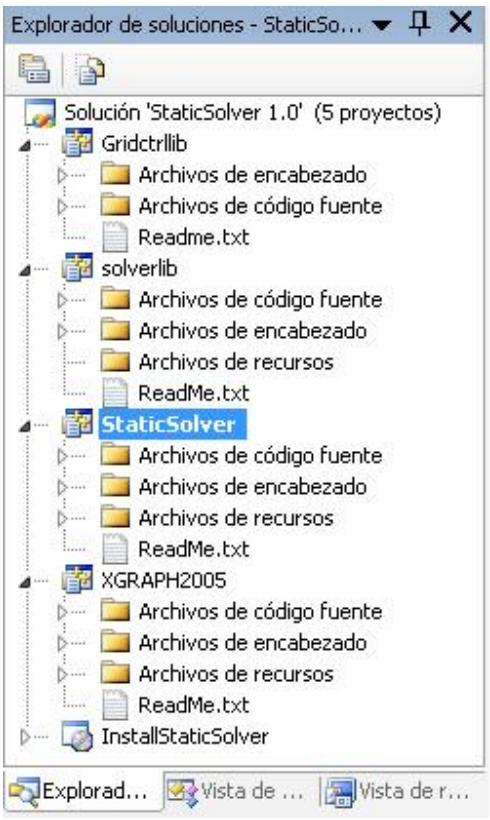
2.5.1 Librerías

Las **librerías** son trozos de código que contienen alguna funcionalidad pre-construida que puede ser utilizada para desarrollar software. Las librerías contienen en su interior variables y funciones. En librerías de C++, estas variables y funciones pueden estar encapsuladas en forma de clases.

La programación del software StaticSover 1.0, se desarrolló en Visual C++ debido a su versatilidad y uso extendido en este medio. El software contiene cinco

proyectos (ver Figura 9), cuatro librerías (Gridctrllib, Solverlib, Xgraph2005 y StaticSolver) y el proyecto del instalador.

Figura 9. Proyectos del software



Las librerías se acoplaron en la plataforma de **MICROSOFT VISUAL STUDIO.NET (2005)**, para desarrollar el software **StaticSolver 1.0**. Las librerías contienen sus respectivos archivos de código fuente, archivos de encabezado y archivos de recursos.

StaticSolver se creó como proyecto principal debido a que se necesita enlazar las demás librerías para hacer uso de todas las aplicaciones de las librerías para desarrollar el software. La librería **Solverlib** se acopló para dar solución a las ecuaciones no lineales debido a que en este proyecto se encuentra el código para resolver el sistema de ecuaciones con el método de Newton Raphson, **Gridctrlib**

se utiliza como herramienta en el uso de tablas paramétricas para dar múltiples valores a las variables independientes de las ecuaciones y determinar su efecto en las variables dependientes según la solución del sistema de ecuaciones no lineales, **Xgraph2005** como herramienta para graficar los datos obtenidos en las tablas paramétricas.

Dentro de la librería **StaticSolver** se incluye la clase **TEvaluar** descrita en el capítulo 1, la cual ayuda a analizar las expresiones matemáticas como reales, operadores, funciones y variables que se encuentren en el sistema de ecuaciones a resolver.

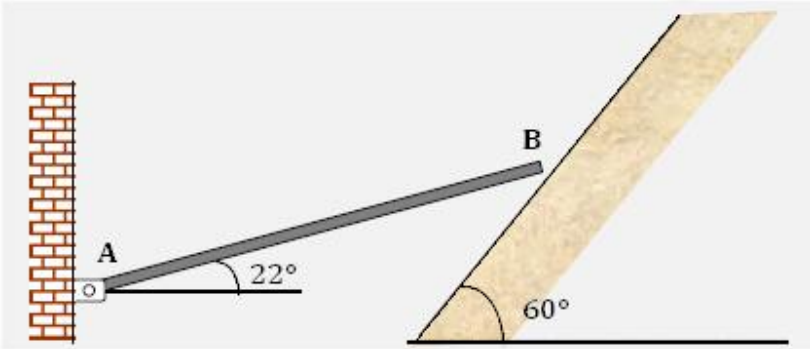
Se crearon las clases CStaticSolverDoc (utilizada para almacenar los datos del documento introducidos en la ventana **Hoja de Ecuaciones**) y CStaticSolverView (utilizada para editar y visualizar el texto de las ecuaciones no lineales en la ventana) mediante la relación de las clases CCrystalTextBuffer, CCrystalTextView, CCrystalEditView, con la clase CStaticSolverView. Las clases CEditReplaceDlg, CFindTextDlg ayudan a reemplazar y buscar texto de las ecuaciones planteadas. Todas estas clases hacen parte de librería **StaticSolver.**

2.5.2 Descripción de la solución de un Sistema de Ecuaciones No Lineales con el Staticsolver 1.0

Mediante el Ejemplo 1, típico de la estática, se describe paso a paso el desarrollo para la solución del sistema de ecuaciones no lineales con el código fuente del software:

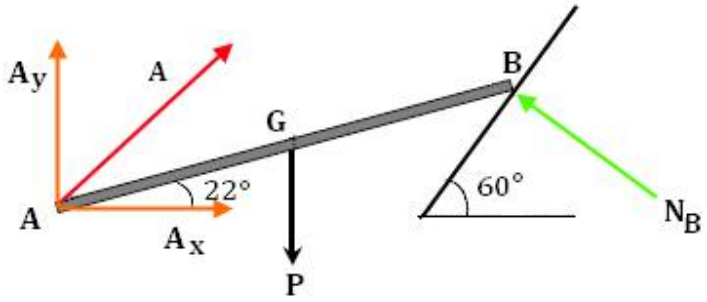
Ejemplo 1: una barra homogénea de peso P y longitud *l* está articulada en su extremo A y se apoya en su extremo B sobre una superficie lisa tal como se muestra en la Figura 10. Determinar la reacción A_x y A_y en la articulación.

Figura 10. Barra homogénea AB



Lo primero es realizar el diagrama de cuerpo libre de la barra (Figura 11) donde se tienen las fuerzas que actúan en la barra, que son: el peso P, la reacción R_A y la normal N_B.

Figura 11. Diagrama de cuerpo libre de la barra AB



Para resolver el problema se plantean las condiciones de equilibrio para la barra, generando las ecuaciones:

$$\sum F_x = 0 \quad A_x = N_B * \cos(30°)$$

$$\sum F_y = 0 \quad A_y + N_B * sen(30°) = P$$

$$\sum M_A = 0 \quad N_B * sen(52°) * l = \frac{1}{2} P * l * \cos(22°)$$

Como l multiplica a ambos lados de la igualdad se elimina y con el valor de peso P = 369 N (Newtons) se tiene un sistema de cuatro ecuaciones con cuatro incógnitas:

$$A_x = N_B * \cos(30°) \quad (14)$$

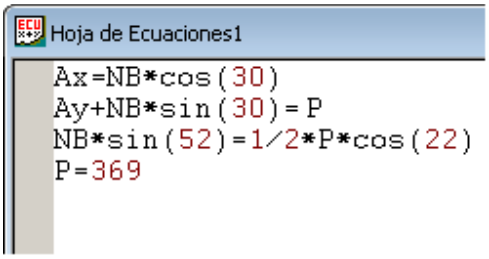
$$A_y + N_B * sen(30°) = P \quad (15)$$

$$N_B * sen(52°) = \frac{1}{2} P * \cos(22°) \quad (16)$$

$$P = 369 \quad (17)$$

El sistema se representa en la Hoja de Ecuaciones (ver Figura 12) así:

Figura 12. Sección de la Hoja de Ecuaciones1



Una vez las ecuaciones son introducidas correctamente a la Hoja de Ecuaciones1, se utiliza el botón calcular, para resolver el sistema de ecuaciones. La anterior operación inicia la función donde empieza el cálculo: `OnEjecutarEjecutar()`, la clase CStaticSolverApp es la clase base que junto con CStaticSolverView y CStaticSolverDoc forman las clases para la interfaz de usuario. A continuación se explican fragmentos del código de la función:

```
void CStaticSolverApp::OnEjecutarEjecutar()
{
 CMDIFrameWnd *wnd=(CMDIFrameWnd *)AfxGetMainWnd();
 CMDIChildWnd*  mdi=wnd->MDIGetActive();
 CView *vistacti=mdi->GetActiveView(); // vista activa
 POSITION posdoc,posview;
 CStaticSolverView *pvista=NULL;
 CString strDocName,strTarget; //nombre del documento
```

```
CDocTemplate* pSelectedTemplate; // Clase abstracta que aporta la
funcionalidad básica para plantillas
CView *pview; //clase que suministra la funcionalidad de la vista
CDocument *doc; // suministra la funcionalidad del documento
POSITION pos=GetFirstDocTemplatePosition();
```

Donde `pos` recupera la posición del primer documento. Enseguida se busca el nombre del tipo del documento (visualizado en la cadena `strDocName`). Se crean cuatro tipos de documentos, que son:

- ✓ Hoja de Ecuaciones.
- ✓ Tabla Paramétrica.
- ✓ Gráfica.
- ✓ Solución.

```
CStaticSolverView *pvista=NULL;
strTarget=_T("Hoja de Ecuaciones");
while (pos !=NULL) {
    pSelectedTemplate=(CDocTemplate*)GetNextDocTemplate(pos);
    ASSERT(pSelectedTemplate!=NULL);
   ASSERT(pSelectedTemplate->IsKindOf(RUNTIME_CLASS(CDocTemplate)));
   pSelectedTemplate
   ->GetDocString(strDocName,CDocTemplate::docName);
  if (strDocName==strTarget)
      posdoc = pSelectedTemplate->GetFirstDocPosition();
      while (posdoc !=NULL) {
           doc=pSelectedTemplate->GetNextDoc(posdoc);
           ASSERT(doc!=NULL);
           ASSERT(doc->IsKindOf(RUNTIME_CLASS(CDocument)));
           posview =doc->GetFirstViewPosition();
              if (posview !=NULL) {
                    pview=doc->GetNextView(posview);
                    ASSERT(pview!=NULL);
                     ASSERT(pview- >IsKindOf(RUNTIME_CLASS(CView)));
                    pvista=(CStaticSolverView*)pview;
                    if (!pvista) return;
```

En el fragmento de código anterior, si el tipo de documento es `"Hoja de Ecuaciones"`, es decir `if(strDocName==strTarget)`, continúa hasta que se acople a la clase `CView *pview` las siguientes clases:

- ✓ CStaticSolverDoc: conserva los datos del documento, como el nombre del documento (en este caso Hoja de Ecuaciones1), tipo de documento etc.

- ✓ CStaticSolverView: añade las clases CCrystalEditView y CProblemaGeneral. CProblemaGeneral es la clase que tiene las características principales de determinado problema, la lista de ecuaciones, cantidad de variables, solución inicial y solución del problema.

Si la vista activa (ventana con la que se está trabajando) es la hoja de ecuaciones1, continúa con el siguiente código:

```
if (pvista==vistacti){
      StaticSolverView=pvista;
      StaticSolverView->PasarEcuaciones();
      StaticSolverView->miproblema->SolInicial();
      if (error>0) return;
      StaticSolverView->miproblema->calcular();
      if (error>0) return...
      break;
}
```

Del código anterior se realizan las siguientes operaciones:

1. `StaticSolverView->PasarEcuaciones();`
2. `StaticSolverView->miproblema->SolInicial();`
3. `if (error>0) return;`
4. `SolverView->miproblema->calcular();`

1. En la función `PasarEcuaciones(void)` de la clase CStaticSolverView se realiza las siguientes acciones:

- ✓ Contar el número de líneas editadas en la hoja de ecuaciones con la función `GetLineCount()`. Para el Ejemplo 1 serían 4 líneas, debido a que son 4 ecuaciones.

- ✓ Borrar la lista de ecuaciones con la función `LiberarMemoria()` de la clase CProblemaGeneral denominada `miproblema`.

- ✓ Contar el número de caracteres en cada línea con `GetLineLength(j)`. En este caso la primera línea tiene 13, la segunda 15, la tercera 24 y la cuarta 5 caracteres.

- ✓ La línea se almacena en una cadena de caracteres `str` con la función `GetText(j, 0,j,nLineLength,str)`.

- ✓ Borrar caracteres de los comentarios de cada línea, es decir los caracteres después de las comillas “, con `str.Delete(index,500)`. Para el Ejemplo 1 no se incluyen comentarios.

- ✓ Si la línea no tiene comentarios y no está vacía en seguida se añade al vector `arr` la ecuación que se obtiene de `str`.

- ✓ Borrar el vector `arr`.

Se puede observar el código de la función `PasarEcuaciones()`:

```
int CStaticSolverView::PasarEcuaciones(void)
{
  int j,i,k=0;
  error=0;
  cont=0;
  int nLineLength;
  int index=0;
  i=LocateTextBuffer()->GetLineCount(); //Número de lineas
  miproblema->LiberarMemoria();
  CString strLine,str;
  CStringArray *arr=new CStringArray();
    for (j=0;j<i;j++) {
       nLineLength = 255;
       nLineLength=LocateTextBuffer()->GetLineLength(j);
     if (nLineLength) {
          LocateTextBuffer()->GetText(j, 0,j,nLineLength,str);
```

```
        str.ReleaseBuffer(nLineLength);
        index=str.Find("\"");
          if (index>=0){
               str.Delete(index,500);
               index=-1;
               }
               if (str.Find("\"")<0  && !str.IsEmpty()) {
                    str.Trim(" ");
                    if (str.GetLength()>0)
                        arr->Add(str);}
                else {
                }
             }
        }
      miproblem->PasarEcuaciones(arr); //pasar ecuaciones
      if(miproblema->Xini.size() >1){
        int NVar=arr->GetCount();
        GetDocument()->nvariables=NVar;
      }
      delete arr;
      return 1;}
```

Cuando se realice este procedimiento con cada línea, lo siguiente es pasar a la función `PasarEcuaciones(arr)` de CProblemaGeneral.

```
void CProblemaGeneral::PasarEcuaciones(CStringArray *arr)
{
      necuaciones=arr->GetSize(); //número de ecuaciones
      listaec=new char*[necuaciones];
      int i;
      for (i=0;i<necuaciones;i++) {
           listaec[i]=new char[255];
           strcpy(listaec[i],arr->GetAt(i)); //copia arr a listaec[i]
      }
}
```

En el anterior código se realiza el conteo del número de ecuaciones del vector (`arr->GetSize()`) y copia cada ecuación del vector `arr` a la lista de ecuaciones denominada `listaec[i]`. Cuando la lista contiene todas las ecuaciones retorna a la función `PasarEcuaciones(void)` de StaticSolverView, para borrar el vector `arr`.

2. Cuando termina la anterior operación, la función `SolInicial()` de CProblemaGeneral, enlazada con CStaticSolverView, impone la solución inicial al problema, identificando las variables de las ecuaciones. Ver el siguiente código:

```
int CProblemaGeneral::SolInicial(void)
{
	char evaluar[255],result[255],err[255];
	int i;
	listaglb=&lista;
	cont++; // Cuenta las veces que se recorre SolInicial(void)
	for (i=0;i<necuaciones;i++) {
		QuitarIgual(listaec[i]);
		if (error>0) return 1;
	}...
```

Dentro de este código se encuentran las siguientes funciones:

- ✓ `QuitarIgual(listaec[i])`, es una función de la clase TEvaluar (`listaec[i]= prog1`), esta encargada de:

  - Pasar todo lo que está enseguida de la igualdad de cada ecuación dentro de paréntesis, como negativo al lado izquierdo de la igualdad y borra el signo igual (=) de la ecuación. Para Ejemplo 1, la ecuación $A_x = N_B * \cos(30)$ queda $A_x - (N_B * \cos(30))$ y así con el resto de ecuaciones. Ver el siguiente código:

```
void QuitarIgual(char* prog1)
{
TEvaluar *TEvaluar;
char *prog;
numdelim=0;
prog=strdup(prog1);// duplica  prog
strcat(prog1,"$"); //anexa el "$" a prog1
int len =strlen(prog); //longitud de prog
int i,j;
for (i=0,j=0;i<len;i++,j++) {
     if (prog[i]!='=') {
     prog1[j]=prog[i];
     }
     else {
           igual.Add(prog1[j]);
```

```
            prog1[j]='-';  //remplaza el iguial por un (-)
            prog1[j+1]='(';// remplaza el caracter por (
            j++;
        }
    if (prog1[j]!='(' ){
    }
    else {
       numdelim ++;
    }
    if (prog1[j]!=')'){
    }
    else{
        numdelim --;
    }
}
strcat(prog1,")");//anexa ")"...
```

- En `QuitarIgual(char* prog1)` se encuentra la variable entera `numiguales` (ver código adjunto), ésta almacena el número de iguales de cada ecuación gracias al conteo del vector `igual`, que contiene estos signos. Si se encuentra que hay más de un signo igual o no hay ningún signo igual, se llama la función `Error_Sint(int i)` de la clase TEvaluar es decir Error_Sint(5) ó `Error_Sint(6)`, generando un mensaje de error visible al usuario.

```
...
numiguales=igual.GetCount();
igual.RemoveAll();
delete prog;
if(numiguales>1){
    TEvaluar->Error_Sint(5);
    return ;
}
if (numiguales<1 && cont==1){
     TEvaluar->Error_Sint(6);
     return ;
}...
```

- Si `cont==2` entonces se verifica que número de ecuaciones es el mismo que el de variables (ver código adjunto), de lo contrario se origina el mensaje de error `Error_Sint(0)`. En este caso con el Ejemplo 1 se tiene un sistema de 4 X 4.

```
if (cont==2 ){  /////Verifica si el número de ecuaciones
esdiferente de número de variables////////
  TEvaluar->Nvar=listaglb->lista->size();//Num. de variables
   if(TEvaluar->Nvar!=necuaciones){
       TEvaluar->Error_Sint(0);
        return;
    } }...
```

- Por último si el número de paréntesis no es equilibrado, es decir si el número de paréntesis no es par, origina el mensaje de error (ver código siguiente) con la función `Error_Sint(7).`

```
...
 if (numdelim!=1){
    if(cont>1) return;
    TEvaluar->Error_Sint(7);
    return ;
 }
```

✓ Continuando con la función `SolInicial(void)`, en el código siguiente, se borran los datos de listas anteriores y se crea una nueva lista para almacenar las variables. Se le da un valor cero al tamaño del vector Xent, este vector que almacena los valores de la solución del sistema de ecuaciones del problema.

```
...
   if (!inicial) return 1;
   Xent.chaSize(0);
   Dec.chaSize(0);
   lista.BorrarDatos();
   lista.CrearLista();
   eval.lista=&lista;
   eval.Xent=&Xent; //iguala el vector Xent con Xent de TEvaluar
   eval.Xini=&Xini; //iguala el vector Xini con Xini de TEvaluar
   eval.Dec=&Dec;//El vector Dec guarda el número de decimales de
   la solución
   if (cont>1){
        cont=0;
   }
```

✓ El código de la función `SolInicial(void)`, termina de la siguiente forma:

```
Vector<int> vecres(necuaciones);
   for (e=0;e<necuaciones;e++) {
            vecres[e]=eval.Evaluar(listaec[e],result,err);
            if (error>0)return 1;
   }
   int maximo=vecres.MMax();//retorna el valor a la posicion   del
    indice maximo
   int minimo=vecres.MMin();//retorna el valor a la posicion del
    indice maximo
   if (minimo<0 || maximo > 0)
            return 0;
   Xini=Xent;
   inicial=0;
   return 1;
}
```

El vector `Xent` se iguala al vector `Xini` que almacena los valores que el usuario define para la solución inicial del sistema de ecuaciones. Estos valores se toman del diálogo de **Información de variables**.

Cuando se accede a la función `Evaluar(listaec[e],result,err)` (de la clase TEvaluar) la cual evalúa las expresiones, lleva al siguiente código:

```
int TEvaluar::Evaluar ( char * prog1, char * result,
char * msj_err)   //evalua las expreciones
{
   contieneigual=0;
   strcpy(msj,"");
   strcpy(result,"0.0");
   if (!strlen(prog1)) return 1;
   prog=strdup(prog1);
   int ll=strlen(prog);
   error = 0;
   Tomar_Exp (result,result);//Entra a tomar la expresión
   if (error>0) return 1;
   if (error) result = "";
   prog=prog-(ll-strlen(prog));
   delete prog;
   return error;
 }
```

Entrando a la función `Evaluar` se accede a otra función (ver código adjunto) llamada `Tomar_Exp (result,result)`:

```
void  TEvaluar::Tomar_Exp (char * result,char * valor)
{
      strcpy(objeto,nombre);
      Obtener_Token();
      if (strlen(Token) != 0 &&  stricmp(Token,"$") &&   error==0)
            Nivel1 (result,valor);
      }
}
```

La función `Tomar_Exp (char * result,char * valor)` se caracteriza por contener `Obtener_Token()`, función que se encarga de analizar el conjunto de los caracteres de cada ecuación y así identificar el papel que cumple cada carácter o grupo de caracteres, que puede ser:

- ✓ Numérico (`int TEvaluar::EsDigit (char car)`) `"0123456789"`.
- ✓ Alfabético (`int TEvaluar::EsAlfa (char car)`) `"A…Z"`.
- ✓ Función (`int TEvaluar::EsFuncion(char *funcion)`), si es una función trigonométrica, trigonométrica inversa, hiperbólica, logarítmica en base 10 y logarítmica natural, exponencial, ABS, SQRT, PI, FLOOR, etc.
- ✓ Función binaria (`int TEvaluar::EsFuncionBIN(char *funcion)`) `"MAX","MIN","MOD"`.
- ✓ Corchete (`int TEvaluar::EsCorchete (char car)`) `"{}"`.
- ✓ Espacio en blanco (`int TEvaluar::EsBlanco (char car)`)" ".
- ✓ Operador (`int TEvaluar::EsDelim (char car)`) `"+-/*^()$"`.
- ✓ Punto (`int TEvaluar::EsPunto (char car)`) `"."`.
- ✓ Coma (`int TEvaluar::EsComa (char car)`) `","`.
- ✓ Indebido (`int TEvaluar::EsIndebido (char car)`) `"#%&~@?¿¡!¬|°;_[]´¨`'"`.
- ✓ Cadena (`int TEvaluar::EsCadena(char *cadena)`).

Definido un carácter o grupo de caracteres como una determinada variable, con la función de TEvaluar `Encontrar_Var(char * var)`, se busca y crea una lista de variables que se encuentran en el sistema de ecuaciones, añadiendo a cada posición del vector `Xent` el índice asignado a cada

variable, para que cuando se obtengan los valores de la solución de `Xent` éstos se almacenen correctamente dentro del vector, de acuerdo al indice de cada variable (ver siguiente código).

```
char * TEvaluar::Encontrar_Var(char * var)
{
char valor[255];
int muesca;
valor[0]=0;
muesca = toupper(*var) - 'A';
CDato *dato;
dato=lista->BuscarVariable(var);
if (dato) {
    _gcvt( Xent->elem(dato->indice-1), 32, valor);
 }// busca una lista de variables asignandole un indice en Xent
 else {
     Xent->addVal(1.0);
     Dec->addVal(4.0);
     Lista->AgregarDato(Xent->size(),Xent->elem(Xent->size()-
     1),var);
     dato=lista->BuscarVariable(var);
    _gcvt( Xent->elem(dato->indice-1), 32, valor);
   }//crea una lista de variables asignandole un indice en Xent
      return valor;
}
```

La lista de variables se va almacenando en `Vector<PDato> *lista` de la clase `CDatosLista` como se muestra en el siguiente código:

```
int CDatosLista::AgregarDato(CDato *dato)
{
  CDato *enc=BuscarVariable(dato->nombre);
  if (!enc) {
     lista->addVal(dato); //agrega las variables a este vector
            return 1;
      }
      return 0;
 }
```

Para el Ejemplo 1 el orden en que se encuentran las variables del sistema de ecuaciones es: Ax, NB ,Ay, P.

Ya definidos todos los caracteres del sistema de ecuaciones se procede al cálculo del sistema de ecuaciones, si no se presentan errores durante el procedimiento descrito.

3. Si durante el anterior procedimiento se presenta un error, TEvaluar contiene la función `Error_Sint (int i)` que visualiza un mensaje de error, según sea el caso. La variable `int error` toma el valor de 1 si se presenta algun error y aborta el procedimiento de cálculo, si no hay errores la variable toma el valor de cero (0).

4. `StaticSolverView->miproblema->calcular()`. Si no existen errores entonces se procede al cálculo de las ecuaciones no lineales, se realiza mediante la siguiente función `calcular()` de la clase CProblemaGeneral, dentro de esta función se realiza el cálculo de las ecuaciones. El enlace de la librería Solverlib con la función `calcular()`, permite darle la solución al sistema de ecuaciones por el método de Newton-Raphson. El método de calculo utiliza NonLinEqSolver, que es la clase base para resolver sistemas de ecuaciones no lineales. La clase base tiene la información básica de convergencia, el número de iteraciones y el error estimado. La clase derivada `NonLinEqSolverprm` se utiliza para la inicialización de la clase base, contiene todos los parámetros que se necesitan. Algunos de los parámetros más importantes del método de cálculo son :

- ✓ `method`: Cadena que contiene el nombre del método de iteración no lineal, ej: NewtonRaphson.

- ✓ `max_iterations`**:** máximo número de iteraciones antes que el algoritmo se detenga, sino converge (la función success detecta la divergencia en `NonLinEqSol`).

- ✓ `intermediate_results`**:** indicador para reporte de la convergencia de los métodos.

- `diag`: es un arreglo de longitud n. Si el modo = 1, `diag` es internamente determinado. Si el modo = 2, `diag` debe contener entradas positivas que sirven de factores multiplicativos de escala para las variables.

- `mode`**:** modo de escalamiento. Es una variable de entrada entera, si su valor es 1, las variables serán modificadas a escala internamente. Si es 2, la escalada es especificada por la variable de entrada `diag`.

- `factor`: es una variable de entrada positiva usada para determinar el límite del paso inicial. Este límite es determinado por el producto del factor y la norma euclidiana de `diag` si no es cero, de lo contrario será el mismo factor. En la mayoría de casos el factor debería estar sobre el intervalo (.1-100). Un valor generalmente recomendado es 100.

- `stopping_criterion`: criterio de parada.

- `max_eps_increases`: detector de divergencia.

- `max_eps`: máximo épsilon "error" según stopping_criterion.

- `xtol`**:** es una variable de entrada no negativa. Finaliza cuando el error relativo entre dos iteraciones consecutivas es `xtol` como máximo.

- `maxfev`: es una variable de entrada de número entero positivo. Finaliza cuando el número de llamadas para la variable fcn es por lo menos `maxfev`: al final de una iteración.

- `epsfcn`: es una variable de aporte usada para determinar una longitud adecuada de paso, para la aproximación de la siguiente diferencia. Esta

aproximación asume que los errores relativos en las funciones son del orden de `epsfcn`:

A continuación se presenta el código para calcular el sistema de ecuaciones:

```
int CProblemaGeneral::calcular()
{
    if (!SolInicial()) {
            LiberarMemoria(); /*Calculo de la Solución inicial*/
            return 0;}
    if (error>0)return 1;
      int n=Xent.size();
      Xent=Xini;
      /*..............Datos para el método de calculo...........*/
      ofstream ofs("salida.txt");
      pm.diag.chaSize(n);
      pm.diag.fill(1.0);
      pm.method="NEWTON_RAPHSON";//Método de cálculo
      pm.ldfjac = n;
      pm.eps_norm_tp=L2;
      GaussNewton solver(pm);//Introduccion de parametros de
      NonLinEqSolverprm pm a solver.
      DensMatrix<double> Jac(n,n);  /*matriz jacobiana de n x n */
      Vector<real> lin_sol(n);
      Vector<real> escal(n);
      solver.attachLinSol(lin_sol);
      solver.attachMatrix(Jac);
      residual.chaSize(n);
      solver.attachNonLinSol(Xent);
      solver.attachResidual(residual);
      solver.attachUserCode(*this);
      escal=1.0;
      solver.attachescala(escal);
      CPararDlg* m_pModeless;
      m_pModeless = new CPararDlg(::AfxGetMainWnd());
      int result=0;;
      if (m_pModeless->Create() == TRUE) {
            solver.solve();//Resueve el sistema de ecuaciones
            result=solver.stopcrit;
            Xent=solver.GetNonLinearSol();//valores de la solución
      }
      cout << Xent;
      return LiberarMemoria();
}
```

La solución del sistema de ecuaciones termina cuando converge el método de Newton-Raphson, en el vector `Xent` quedan almacenados los valores de la

solución del problema. Para el Ejemplo 1 el vector `Xent` almacena los siguientes valores:

Xent[0] = 188.0012
Xent[1] = 217.0851
Xent[2] = 260.4574
Xent[3] = 369.0000

Resuelto el sistema de ecuaciones lo siguiente es visualizar la solución en una ventana. Para este fin se puede utilizar tres tipos de ventanas que son:

1. **Ventana Solución:** el código adjunto busca el documento solución, al encontrarlo se da la orden de abrir el documento solución.

```
if( solución==1) {
strTarget=_T("Solución");
CDocTemplate* pSelectedTemplate;
POSITION pos=GetFirstDocTemplatePosition();
while (pos !=NULL) {
      pSelectedTemplate=(CDocTemplate*)GetNextDocTemplate(pos)
      ASSERT(pSelectedTemplate!=NULL);
      ASSERT(pSelectedTemplate->  IsKindOf(RUNTIME_CLASS(CDocTemplate)));
      pSelectedTemplate->
      GetDocString(strDocName,CDocTemplate::docName);
      if (strDocName==strTarget)
           pSelectedTemplate->OpenDocumentFile(NULL);
          }
      }
        pvista->miproblema->LiberarMemoria();
}
```

El anterior código que está dentro de la función `OnEjecutarEjecutar()`, lleva a la función `OnInitialUpdate()` de la clase CSolucionView creada para visualizar los resultados de un sistema de ecuaciones. CSolucionView está enlazada con clase CGridCtrl denominada `m_Grid`, de la librería **Gridctrllib,** de manera que la solución se muestre en foma de tabla. A continuación se muestran las variables de la función que permiten especificar las propiedades de la tabla:

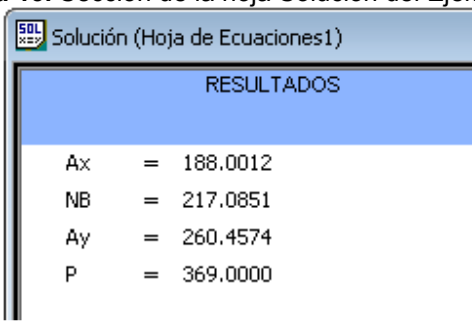
```
void CSolucionView::OnInitialUpdate()
{
 numfilas=prmsolucion->nfilas;
 numcol=prmsolucion->ncol;
 arrvar.Copy(prmsolucion->arrvariables);

//////////////////PROPIEDADES DE LA TABLA/////////////////
 CDato *dato;
 arrvar.RemoveAll();
 arrvar.SetSize(0,2); //Vector arrvar
 int n=listaglb->lista->size();/número de variables de listaglb
 for (int i=0;i<n;i++) {
      dato=listaglb->lista->elem(i);
      if (dato)
          arrvar.Add(dato->nombre);//Añade cada variable al vector
 }
 numfilas=arrvar.GetCount();//número variables del vector
 numcol=4;
 m_nFixCols = 1;
 m_nFixRows = 1;
 m_nRows = 1+numfilas+m_nFixRows;// numero de filas a mostrar
 m_nCols = numcol+m_nFixCols; // numero de columnas a mostrar
```

Para visualizar los valores del vector Xent junto con la lista de variables del sistema de ecuaciones se utiliza la función `SetItemText(j+1,3,str)` de la clase CGridCtrl así:

```
//////////////////VISUALIZAR LOS DATOS//////////////////////

...
for (j=0;j<numfilas;j++) {
var=m_Grid.GetItemText(j+1,1);
strcpy(cadena,var);
dato=listaglb->BuscarVariable(cadena);
if (dato) {
int decimal= StaticSolverView->miproblema->Dec(dato->indice);
strfor.Format("%d",decimal);
strfor="%."+strfor+"f";
str.Format(strfor,StaticSolverView->miproblema->Xent(dato->indice));
m_Grid.SetItemText(j+1,3,str);...
```

Para el Ejemplo 1 una vez finalice la función `OnInitialUpdate()` y el comando `OnEjecutarEjecutar()` aparece una ventana solución como en la Figura 13:

Figura 13. Sección de la hoja Solución del Ejemplo 1



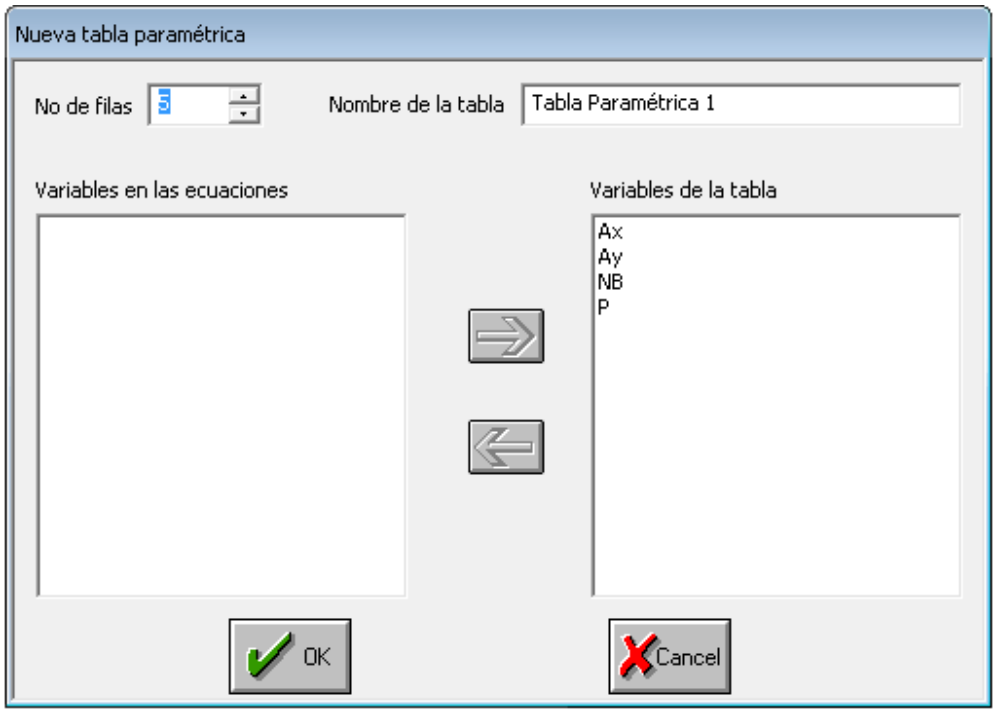
2. **Ventana tabla paramétrica:** para crear una tabla paramétrica se debe tener en cuenta que por cada valor dado (en cada fila) a la variable independiente se genera una nueva ecuación en el sistema ocasionando que por cada fila de la tabla se genere un sistema de ecuaciones diferente. Si en el Ejemplo 1 se toma como parámetro la variable P, se debe eliminar la ecuación 4 de la Hoja de Ecuaciones1 y así poder dar diferentes valores a la variable P. Para desarrollar la tabla paramétrica se presiona el botón Nueva tabla paramétrica, esta acción lleva a la función `OnTablaNuevatabla()` de la clase CStaticSolverApp para crear una tabla, una vez se comprueba que la ventana activa es la Hoja de Ecuaciones1 y se ejecutan las funciones `PasarEcuaciones()` y `SolInicial()` se continua con el siguiente código, que permite visualizar el dialogo Nueva Tabla Paramétrica:

```
...
CNuevaTablaParDlg dialg;
dlg.lista=& StaticSolverView->miproblema->lista;//lista de variables
if (dlg.DoModal()==IDOK) {// abre el diálogo NuevaTablaParamétrica
numfilas=dlg.m_nfilas; //lleva el número de filas a la tabla
numcol=dlg.arrvarsel.GetUpperBound()+1;// lleva el número de columnas a
la tabla
m_nombretable=dlg.m_nombretable; //Nombre de la tabla
strarr.RemoveAll();
strarr.SetSize(0,2);
for (int i=0;i<numcol;i++) {
   strarr.Add(dialg.arrvarsel.GetAt(i));
}...
```

Al final del código anterior se adicionan las variables de arrvarsel al vector `Strarr`. Se creó la clase CNuevaTablaParDlg para el diálogo **Nueva tabla paramétrica**, en esta clase se toma la lista de variables, para visualizarlas en el diálogo (Figura

14) y se puedan escoger las variables para construir la tabla paramétrica. Las variables seleccionadas se almacenan en el vector `arrvarsel` de la clase CNuevaTablaParDlg, lo mismo el número de filas con la variable `m_nfilas` y el nombre de la tabla con la variable `m_nombretable`.

Figura 14. Nueva tabla paramétrica



Variables de `CNuevaTablaParDlg`:

- ✓ `CString m_nombre`: almacena el nombre de la tabla a generar.
- ✓ `CListBox m_listavarecuaciones`: lista que almacena las variables que están en el sistema de ecuaciones.
- ✓ `CListBox m_vartabla`: lista que guarda las variables a mostrar en la tabla.
- ✓ `m_nfilas`: almacena el número de filas a mostrar en la tabla.
- ✓ `OnBnClickedButtonadd`: Agrega la variable seleccionada.
- ✓ `OnBnClickedButtonremove`: Borra la varible seleccionada.

Luego de introducir los datos requeridos en el diálogo Nueva tabla paramétrica se abre el tipo de documento Tabla Paramétrica mediante la clase CTableView creada para visualizar la tabla, ya que se inicia la función `OnInitialUpdate()` (ver siguiente código).

```
void CTableView::OnInitialUpdate()
{
///Parametros para la creación de la tabla paramétrica/////
m_nFixCols = 1;// número de filas fijas
m_nFixRows = 1; //número de columnas
m_nRows = numfilas+m_nFixRows;// número de filas a mostrar
m_nCols=numcol+m_nFixCols; // número de columnas a mostrar

...

for (i=0;i<numcol;i++) { // visualiza las variables en la columna 0
    str.Format("%s",strarr.GetAt(i));//strarr contiene las variables
      m_Grid.SetItemText(0,i+1,str);
}

for (i=0;i<numfilas;i++) {  // muestra la numeración de las filas
      str.Format("%d",i+1);
      m_Grid.SetItemText(i+1,0,str);
}
```

Una vez se recorre la función OnInitialUpdate() se obtiene la ventana de la Tabla paramétrica 1 (Figura 15):

Figura 15. Ventana Tabla Paramétrica 1

Lo siguiente es introducir los valores de la variable independiente y así ver como se afectan las demás variables. Para el Ejemplo 1, P es la variable independiente y se le darán 5 valores (300, 400, 500, 600 y 700).

Para resolver la tabla se presiona el botón Calcular tabla paramétrica, esta acción conduce a la función `OnEjecutarEjecutarTabla()` de CTableView. Esta función permite visualizar el cálculo de la tabla. Características de la función:

- ✓ Se comprueba la existencia de las variables dependientes, que son aquellas cuyas celdas están vacías, y variables independientes, donde las celdas se especifican los valores.

- ✓ Con la función `AddEquation(str)`, de la clase CProblemaGeneral, se adiciona la ecuación faltante completando el sistema de ecuaciones para cada fila. La ecuación se genera al igualar la variable independiente con el valor introducido en la celda de la respectiva variable (columna). Para el Ejemplo 1 por cada fila se genera un sistema de 4 ecuaciones con 4 incógnitas.

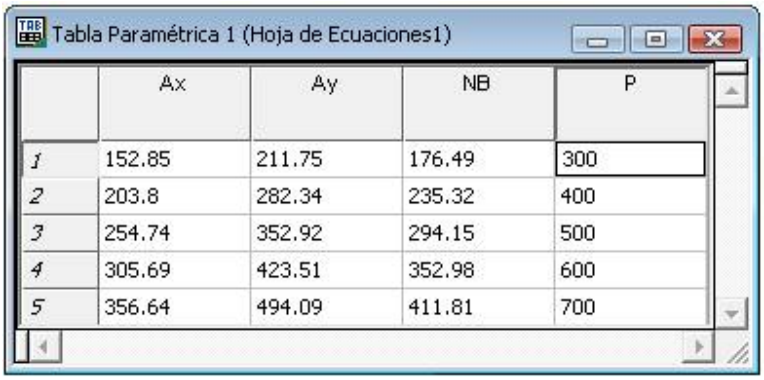
- ✓ Luego de completar el sistema de ecuaciones para cada fila, la función `calcular()` obtendrá la solución en el vector Xent y se visualiza la solución del sistema de ecuaciones en la tabla mediante la función `m_Grid.SetItemText(i+1,j+1,str)`. A continuación se muestra el código:

```
//////////////////////VISUALIZAR SOLUCIÓN/////////
   ...
   for (j=0;j<numcol;j++) {
   if (vardep_b[i][j]==1) {
   vardep=m_Grid.GetItemText(0,j+1);
   strcpy(cadena,vardep);
   dato=listaglb->BuscarVariable(cadena);
   if (dato) {
   strfor.Format("%d",4);
   strfor="%."+strfor+"f";
```

```
str.Format(strfor,StaticSolverView->miproblema->Xent(dato-
>indice));//valores de las variables
m_Grid.SetItemText(i+1,j+1,str);// muestra la solución en la tabla
```

Al final se obtiene resuelta la tabla paramétrica, Figura 16.

Figura 16. Tabla Paramétrica 1 resuelta



	Ax	Ay	NB	P
1	152.85	211.75	176.49	300
2	203.8	282.34	235.32	400
3	254.74	352.92	294.15	500
4	305.69	423.51	352.98	600
5	356.64	494.09	411.81	700

3. **Ventana Gráfica:** apoyado con los valores de la Tabla Paramétrica 1, se procede a graficar estos valores de la siguiente manera: se presiona el botón Nueva Gráfica para crear una ventana Gráfica. Esta acción lleva a ejecutar la función `OnGraficarNuevaGrafica()` de la clase CStaticSolverApp, `que` una vez se compruebe que la ventana activa es la Tabla Paramétrica 1, realiza las siguientes operaciones:

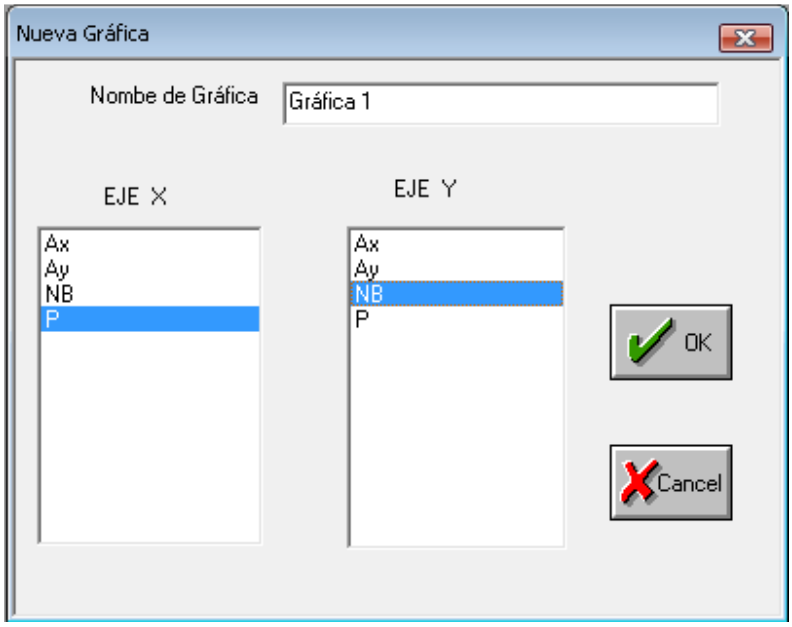
- ✓ `CargarGlobales()`: esta función de la clase CTableView se construyó para cargar los valores de la tabla activa. En la matriz `Global` se almacenan los valores de la tabla y en el vector `arrvar` se almacenan las variables de la tabla.

- ✓ La clase CDialogoGrafica se usa para mostrar el dialogo Nueva grafica. En esta clase se almacena la variable seleccionada, para el eje x y para el eje y , en los vectores `arrXs` y `arrYs` respectivamente, además guarda el nombre de la gráfica en la variable `m_nombregrafica`.

Una vez se obtienen los valores de la tabla, las variables para el eje x y eje y y el nombre de la gráfica en el diálogo Nueva Gráfica, lo siguiente es abrir el tipo de documento Gráfica presionando el botón OK del diálogo (Figura 17).

La clase CPlotView se creó para visualizar la gráfica por el enlace con la clase CXGraph, nombrada en CPlotView como m_Graph. Antes de abrir el documento primero inicia la función `OnInitialUpdate()` de CPlotView con el objetivo de cargar una serie de datos necesarios para construir la gráfica, esto se logra con la función `AddDataSerie(bool bAddAxes)`.

Figura 17. Diálogo Nueva Gráfica



Dentro de `AddDataSerie` se encuentra una función con el nombre de `TDataPoint m_Valuesglb[MAX_SERIES][VALCOUNT+2]` de la clase CXGraphLabel, función que organiza los datos para la construcción de la gráfica. Las funciones `m_Valuesglb[i][j].fXVal` y `m_Valuesglb[i][j].fYVal` se igualan a la matriz `values[i][indice]` que contiene los valores de la matriz Global, de acuerdo a un índice (valor numérico que referencia un vector de la matriz Global), para darle un valor a cada punto en la gráfica. Con la función `m_Graph.SetData`

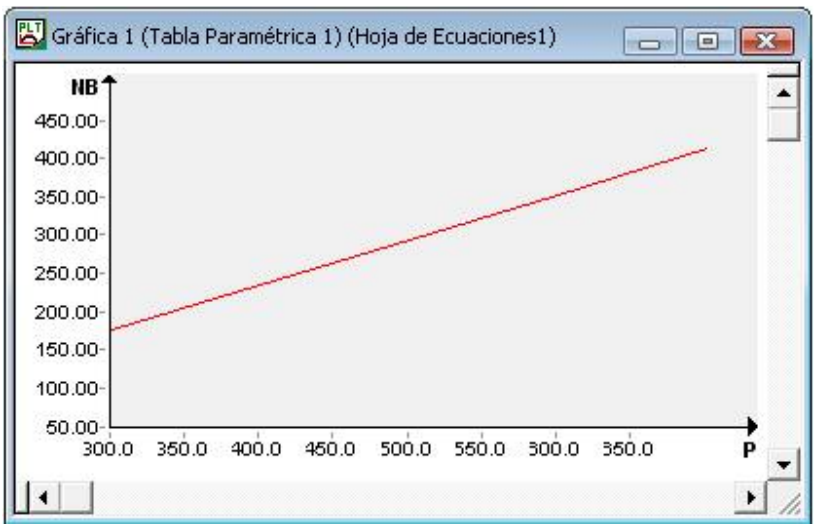
`(m_Valuesglb[j], nx[j], j, 0, 0)`, se ubican los datos para la construcción de la gráfica, donde nx, es el número de puntos a graficar. Una vez obtenidos estos datos se procede a graficar los puntos.

A Continuación se puede ver el código de la función addDataserie:

```
void CPlotView::AddDataSerie(bool bAddAxes)
{
...
for (i=0;i<nXs;i++) {
     varX=arrXs.GetAt(i);
     indice=BuscarIndiceVar(varX);
for (j=0;j<prmplot->values.getColumn(indice).size();j++)
      m_Valuesglb[i][j].fXVal=prmplot->values[j][indice];
}
for (i=0;i<nYs;i++) {
    varY=arrYs.GetAt(i);
    indice=BuscarIndiceVar(varY);
for (j=0;j<prmplot->values.getColumn(indice).size();j++)
     m_Valuesglb[i][j].fYVal=prmplot->values[j][indice];
      }
}
else
int val=0;
double fMin, fMax, fStep;
for (i = 0; i< k; i++){

// serie de datos
for (j=0;j<k;j++) {
     m_Graph.SetData (m_Valuesglb[j], nx[j], j, 0, 0,true);
     if (j==0)
         m_Graph.GetCurve (j).SetColor(RGB(255,0,0));
     else
         m_Graph.GetCurve (j).SetColor(RGB(0,0,0));

// Ajuste de aspectos visuales para la grafica
m_Graph.SetSeparatedLegend(false);
m_Graph.GetYAxis (0).SetAutoScale(true);
m_Graph.GetYAxis (0).SetDisplayFmt(_T("%5.2f"));
m_Graph.GetXAxis (0).SetLabel(varX);  nombre del eje X
m_Graph.GetYAxis (0).SetLabel(varY);  nombre del eje Y

m_Graph.GetXAxis (0).SetShowGrid(false);
m_Graph.GetYAxis (0).SetShowGrid(false);
m_Graph.GetXAxis (0).SetShowMarker(false);
m_Graph.GetYAxis (0).SetShowMarker(false);
m_Graph.SetShowLegend(false);
                  }
        }
}
```

Para el Ejemplo 1, si se escoge en el diálogo Nueva Gráfica la variable P (para el eje X) y la variable NB (para el eje Y) se obtiene una gráfica como la mostrada en la Figura 18.

Figura 18. Gráfica 1



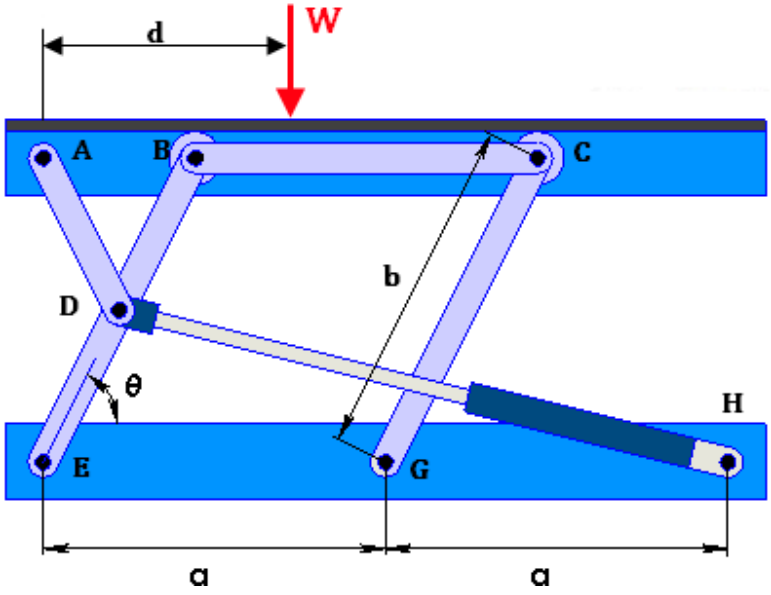
3 RESULTADOS

A continuación se compara la solución arrojada por el Static Solver 1.0 para problemas de estática con la solución encontrada mediante otros métodos, ya sea de manera manual o utilizando el software TKSolver.

3.1 PROBLEMA 1

Un elevador hidráulico (ver Figura 19) se emplea para levantar una caja de 8000 lb. El elevador consta de una plataforma de dos eslabones idénticos sobre los cuales los cilindros hidráulicos ejercen fuerzas iguales. (En la figura sólo se muestra uno de los cilindros y uno de los eslabones y el peso lo dividimos en dos, es decir **W**=4000 lb.) Determinar la fuerza en DH al variar el ángulo de 5^{o} a 90^{o} si se tiene DE=a=25", b=70".

Figura 19. Elevador hidráulico

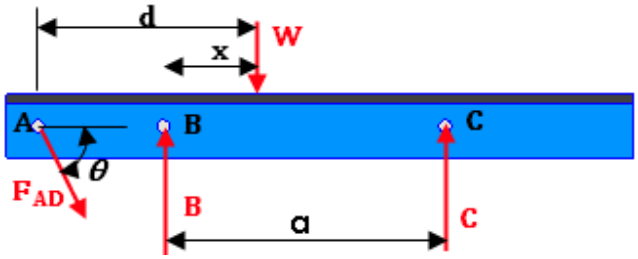


Como la fuerza $F_{AD} = 0$, entonces, del diagrama de cuerpo libre de la viga ABC (Figura 20) se plantean las siguientes ecuaciones:

$$\sum Fy = 0 \quad By + Cy = W \tag{18}$$

$$\sum M_B = 0 \quad W * X = C * a \tag{19}$$

Figura 20. Diagrama de cuerpo libre de la viga ABC



Para la viga GC (Ver Figura 21) se plantea la siguiente ecuación:

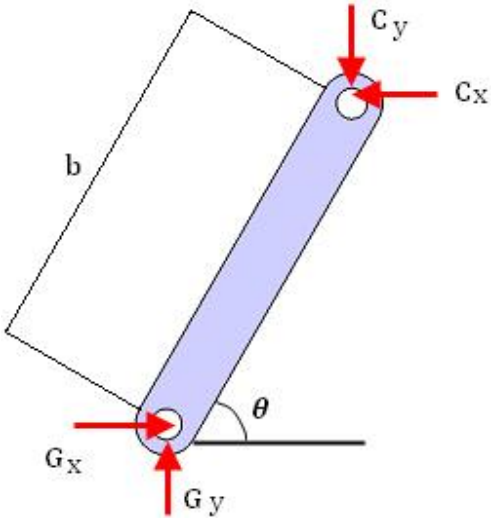
$$\sum Fy = 0 \quad Gy = Cy$$

$$\sum Fx = 0 \quad Cx = Gx$$

$$\sum M_G = 0 \quad Cx * b * sen(\theta) = Cy * b * cos(\theta)$$

$$Cx * sen(\theta) = Cy * cos(\theta) \tag{20}$$

Figura 21. Diagrama de cuerpo libre de la viga GC



Para la viga EB (Ver Figura 22) las ecuaciones que se plantean son:

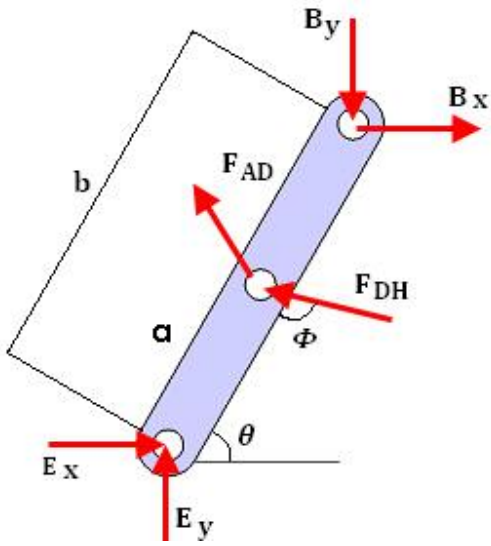
$$Bx = Cx \tag{21}$$

$$\Phi = 90° + ß \tag{22}$$

$$\sum ME = 0$$

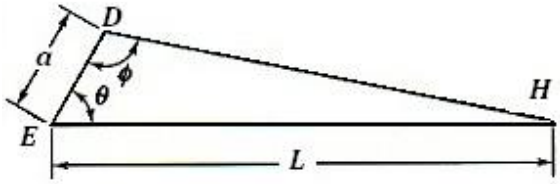
$$F_{DH} * cos\,(ß) * a = Bx * b * sen\,(\theta) + By * b * cos\,(\theta) \tag{23}$$

Figura 22. Diagrama de cuerpo libre de la viga EB



De la geometría del triángulo de la Figura 23, se deducen varias ecuaciones que son:

Figura 23. Triangulo a-L-DH



$$L = 2a = 50" \tag{24}$$

$$DH^2 = L^2 + a^2 - 2 * L * a * cos\,(\theta) \tag{25}$$

$$a^2 = DH^2 + L^2 - 2 * DH * L * Cos(f) \quad (26)$$

$$180 = f + t + u \quad (27)$$

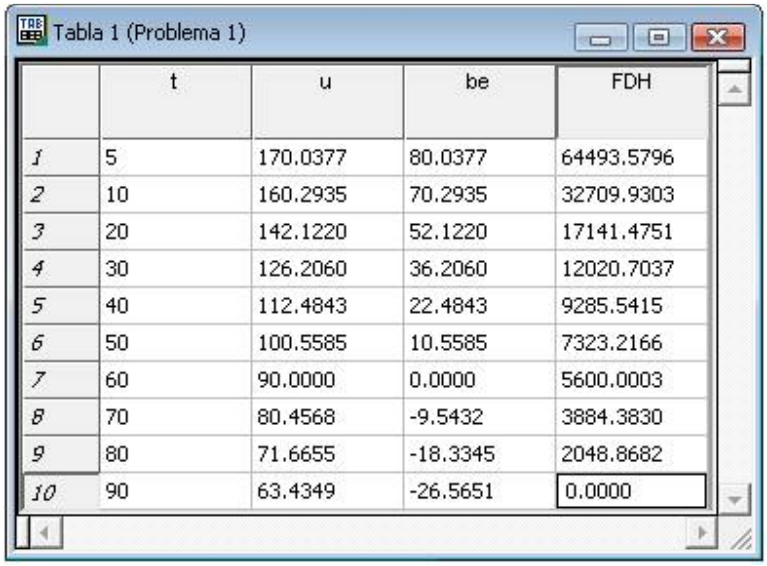
Si se tienen en cuenta las variables conocidas W, a, b, X, se forma un sistema de 10 ecuaciones con 11 incógnitas. Si $t = \theta$, $ß = be$ y $\Phi = u$ entonces:

1. $By + Cy = 4000$
2. $Cy = 1600$
3. $Cx * sin(t) = Cy * cos(t)$
4. $Bx = Cx$
5. $u = 90 + be$
6. $FDH * cos(be) * a = Bx * b * sin(t) + By * b * cos(t)$
7. $L = 50$
8. $DH^2 = L^2 + 25^2 - 2 * L * 25 * Cos(t)$
9. $25^2 = DH^2 + L^2 - 2 * DH * L * Cos(f)$
10. $180 = f + t + u$

Para completar el sistema de ecuaciones utilizando la tabla paramétrica de StaticSolver 1.0 se colocan valores de 5° hasta 90° en la columna de la variable t.

Resolviendo el sistema de ecuaciones se obtendrá la solución en la Tabla 1.

Tabla 1. Solución al Problema 1 con StaticSolver 1.0



	t	u	be	FDH
1	5	170.0377	80.0377	64493.5796
2	10	160.2935	70.2935	32709.9303
3	20	142.1220	52.1220	17141.4751
4	30	126.2060	36.2060	12020.7037
5	40	112.4843	22.4843	9285.5415
6	50	100.5585	10.5585	7323.2166
7	60	90.0000	0.0000	5600.0003
8	70	80.4568	-9.5432	3884.3830
9	80	71.6655	-18.3345	2048.8682
10	90	63.4349	-26.5651	0.0000

En la Tabla 2 se muestran los valores que se obtienen de la solución del sistema de ecuaciones de manera manual:

Tabla 2. Solución manual al Problema 1

t(°)	u(°)	be(°)	FDH(Lb)
5	170	80,04	64494
10	160,3	70,29	32710
20	142,1	52,12	17141
30	126,2	36,21	12021
40	112,5	22,48	9286
50	100,6	10,56	7323
60	90	0	5600
70	80,46	-9,543	3884
80	71,67	-18,33	2049
90	63,43	-26,57	0

Comparando los resultados que se obtienen de manera manual con los obtenidos con StaticSolver 1.0, la diferencia entre las soluciones, no es significativa. En la

Tabla 3 se comprueba esta diferencia utilizando los valores de la variable FDH determinando el error absoluto (**Ea**), relativo (**Er**), el promedio ($\bar{x}$**)** y la desviación estándar (σ). Todos los valores del **Er** tienden a cero y la σ es muy pequeña, es decir no esta tan alejado del $\bar{x}$ del error relativo. Comprobando así que el StaticSolver 1.0 proporciona soluciones confiables a sistemas de ecuaciones.

Tabla 3. Diferencia entre solución manual y la de StaticSolver 1.0

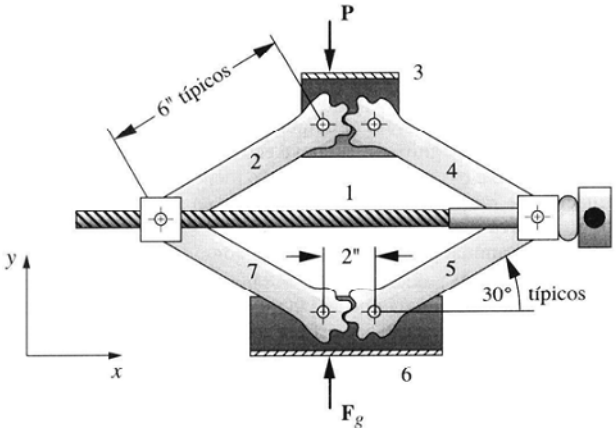
Manual (FDH)	Software (FDH)	Er	Ea
64494	64493,5796	-6,518E-06	-0,4204
32710	32709,9303	-2,130E-06	-0,0697
17141	17141,4751	2,771E-05	0,4751
12021	12020,7037	-2,464E-05	-0,2963
9286	9285,5415	-4,937E-05	-0,4585
7323	7323,2166	2,95E-05	0,2166
5600	5600	0	0
3884	3884,3830	9,860E-05	0,383
2049	2048,8682	-6,432E-05	-0,1318
0	0	0	0
	$\bar{x}$	8,907E-07	
	Σ	4,544E-05	

3.2 PROBLEMA 2

Determinar las fuerzas sobre los elementos del gato de tijera en la posición mostrada en la Figura 24. La geometría es conocida y el gato soporta una fuerza de P=1000 lb en la posición mostrada. Las aceleraciones son despreciables, el gato está sobre el piso a nivel, el ángulo del bastidor del automóvil elevado no transmite o imparte momento de volteo al gato, todas las fuerzas son coplanares y en dos dimensiones[4]

[4] NORTON, Robert L. Diseño de Norton, caso 3A. p. 119

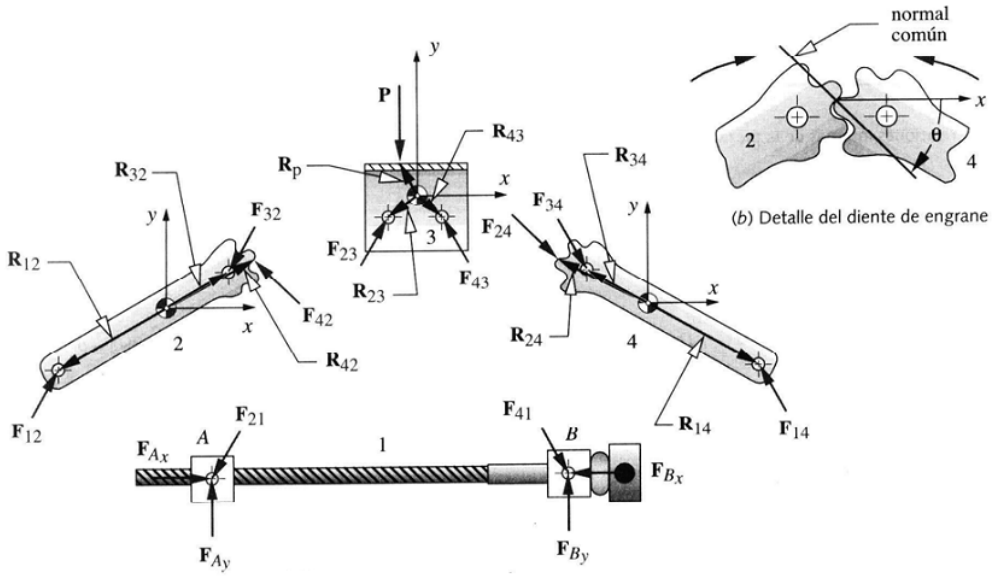
Figura 24. Gato de tijera



Fuente: Diseño de Norton, cap. 3, pag. 120.

Debido a la simetría entre las porciones superior e inferior, para simplificar el análisis se elimina la mitad inferior, entonces las ecuaciones se plantean de acuerdo al análisis de equilibrio de cada una de las piezas de la Figura 25*:*

Figura 25. Sistema de fuerzas de las piezas del gato de tijera



Fuente: Diseño de Norton, cap. 3, pag. 123.

Analizando la pieza 2, se plantean las ecuaciones:

$$\sum FX = 0 \;\; F12x + F32x + F42x = 0 \tag{28}$$

$$\sum FY = 0 \;\; F12y + F32y + F42y = 0 \tag{29}$$

$$\sum Mz = 0 \;\; (R12x * F12y - R12y * F12x) + (R32x * F32y - R32y * F32x) + \; (R42x * F42y - R42y * F42x) = 0 \tag{30}$$

Analizando la pieza 3, se deducen las ecuaciones:

$$\sum FX = 0 \;\; F23x + F43x + Px = 0 \tag{31}$$

$$\sum FY = 0 \;\; F23y + F43y + Py = 0 \tag{32}$$

$$\sum Mz = 0 \;\; (R23x * F23y - R23y * F23x) + (R43x * F43y - R43y * F43x) + (Rpx * Py - Rpy * Px) = 0 \tag{33}$$

Analizando la pieza 4, se plantean las ecuaciones:

$$\sum FX = 0 \;\; F34x + F14x + F24x = 0 \tag{34}$$

$$\sum FY = 0 \;\; F34y + F14y + F24y = 0 \tag{35}$$

$$\sum Mz = 0 \;\; (R14x * F14y - R14y * F14x) + (R24x * F24y - R24y * F24x) + (R34x * F34y - R34y * F34x) = 0 \tag{36}$$

Según la geometría del diente de engrane se generan las ecuaciones:

$$F24y = F24x * tan(t) \tag{37}$$

$$t = -45 \tag{38}$$

De las relaciones cruzadas de las piezas se obtienen las ecuaciones:

$$F32x = -F23x \tag{39}$$

$$F32y = -F23y \tag{40}$$

$$F34x = -F43x \tag{41}$$

$$F34y = -F43y \tag{42}$$

$$F24x = -F42x \tag{43}$$

$$F24y = -F42y \tag{44}$$

$$F12x = -F21x \quad (45)$$

$$F12y = -F21y \quad (46)$$

$$F14x = -F41x \quad (47)$$

$$F14y = -F41y \quad (48)$$

De los datos dados se plantean las siguientes ecuaciones:

$$Py = -1000 \quad (49)$$

$$Px = 0 \quad (50)$$

$$Rpx = -0.5 \quad (51)$$

$$Rpy = 0.9 \quad (52)$$

$$R12x = -3.1 \quad (53)$$

$$R12y = -1.8 \quad (54)$$

$$R32x = 2.1 \quad (55)$$

$$R32y = 1.2 \quad (56)$$

$$R42x = 2.7 \quad (57)$$

$$R42y = 1.0 \quad (58)$$

$$R23x = -0.8 \quad (59)$$

$$R23y = -0.8 \quad (60)$$

$$R43x = 0.8 \quad (61)$$

$$R43y = -0.8 \quad (62)$$

$$R14x = 3.1 \quad (63)$$

$$R14y = -1.8 \quad (64)$$

$$R24x = -2.6 \quad (65)$$

$$R24y = 1.0 \quad (66)$$

$$R34x = -2.1 \quad (67)$$

$$R34y = 1.2 \quad (68)$$

Integrando todas las ecuaciones se genera un sistema de 41 variables con 41 incógnitas, si se utiliza el StaticSolver 1.0 se obtiene la siguiente solución:

RESULTADOS

F12x	=	880.7971	F43x	=	-598.1884	F24x	=	282.6087
F32x	=	-598.1884	Px	=	0.0000	F34y	=	-187.5000
F42x	=	-282.6087	F23y	=	812.5000	F14y	=	470.1087
F12y	=	529.8913	F43y	=	187.5000	F24y	=	-282.6087
F32y	=	-812.5000	Py	=	-1000.0000	R14x	=	3.1000
F42y	=	282.6087	R23x	=	-0.8000	R14y	=	-1.8000
R12x	=	-3.1000	R23y	=	-0.8000	R24x	=	-2.6000
R12y	=	-1.8000	R43x	=	0.8000	R24y	=	1.0000
R32x	=	2.1000	R43y	=	-0.8000	R34x	=	-2.1000
R32y	=	1.2000	Rpx	=	-0.5000	R34y	=	1.2000
R42x	=	2.7000	Rpy	=	0.9000	t	=	-45.0000
R42y	=	1.0000	F34x	=	598.1884	F21x	=	-880.7971
F23x	=	598.1884	F14x	=	-880.7971	F21y	=	-529.8913
F41x	=	880.7971						
F41y	=	-470.1087						

Las unidades de las fuerzas o reacciones son libras (lb) y de los ángulos grados (°).

A continuación se muestra en la Tabla 4 la solución al sistema de ecuaciones del Problema 2, obtenida por el software TKSolver que se encuentra en el libro Diseño de Máquinas de Norton:

Tabla 4. Solución del Problema 2 con TKSolver

Variable	TKSolver	Unidades
		Pieza 2
F12x	877.9	Lb
F12y	530.0	Lb
F32x	-586.5	Lb
F32y	-821.4	Lb
F42x	-291.4	Lb
F42y	291.4	Lb
		Pieza 3
F23x	586.5	Lb

F23y	821.4	Lb
F43x	-586.5	Lb
F43y	178.6	Lb
		Pieza 4
F14x	-877.9	Lb
F14y	470.0	Lb
F24x	291.4	Lb
F24y	-291.4	Lb
F34x	586.5	Lb
F34y	-178.6	Lb

Fuente: Diseño de Norton, cap. 3, pag. 126.

Comparando la solución del problema del gato de tijera obtenida con TKSolver con la solución del **StaticSolver 1.0**, En la Tabla 5 se comprueba que los resultados son parecidos utilizando los valores de la variables determinando el error absoluto (**Ea**), relativo (**Er**), el promedio ($\bar{x}$**)** y la desviación estándar (σ). Todos los valores del **Er** tienden a cero y la σ es muy pequeña, es decir no esta tan alejado del $\bar{x}$ del error relativo.

Tabla 5. Comparación de la solución con TKSolver y StaticSolver

Variable	TKSolver	StaticSolver	Er	Ea
F12x	877.9	880.7971	0,0033	2,8971
F12y	530.0	529.8913	-0,00020	-0,1087
F32x	-586.5	-598.1884	0,01992	-11,6884
F32y	-821.4	-812.5	-0,01083	8,9000
F42x	-291.4	-282.6087	-0,03016	8,7913
F42y	291.4	282.6087	-0,03016	-8,7913
F23x	586.5	598.1884	0,01992	11,6884
F23y	821.4	812.5	-0,01083	-8,9000

F43x	-586.5	-598.1884	0,01992	-11,6884
F43y	178.6	187.5	0,04983	8,9000
F14x	-877.9	-880.7971	0,00330	-2,8971
F14y	470.0	470.1087	0,00023	0,1087
F24x	291.4	282.6087	-0,03016	-8,7913
F24y	-291.4	-282.6087	-0,03016	8,7913
F34x	586.5	598.1884	0,01992	11,6884
F34y	-178.6	-187.5	0,04983	-8,9000
		$\overline{x}$	0,002897	
		Σ	0,02818	

Para comprobar la exactitud de la solución del TKSolver y StaticSolver se reemplaza los valores en las ecuaciones y se verificar la igualdad de cada una de las ecuaciones. Tomando la ecuación 34 y reemplazando los valores de la solución del TKSolver se tiene:

$$(-0.8 * 821.4 - (-0.8) * 586.5) + (0.8 * 178.6 - (-0.8) * -586.5) + (-0.5 * (-1000) - \quad 0.9 * 0) = 0$$

$$-14.24 = 0$$

Hay una buen diferencia en la igualdad, por lo tanto los valores de la solución que brindo el TKsolver para el Problema 2 no son muy exactos.

Reemplazando los valores de la solución del StaticSolver para la ecuación 34 se obtiene:

$$(-0.8 * 812.5 - (-0.8) * 598.1884) + (0.8 * 187.5 - (-0.8) * -598.1884) + (-0.5 * (-1000) - \quad 0.9 * 0) = 0$$

$0 = 0$

La igualdad se cumple, quiere decir que los valores de la solución del StaticSolver 1.0 son muy exactos, brindando al usuario confiabilidad en el desarrollo a sistemas de ecuaciones.

4 CONCLUSIONES

El proyecto de grado y más específicamente el software desarrollado, cumplió a cabalidad el objetivo de convertirse en una herramienta de gran valor pedagógico y que además servirá como apoyo tanto al estudiante como para el profesor, en la asignatura Estática.

El software presenta gran facilidad de manejo y posee herramientas que dan una mayor cobertura a la solución de sistemas de ecuaciones no lineales, permitiendo que el profesor de una solución rápida a problemas y favoreciendo un desarrollo activo del estudiante dentro de la clase.

La desviación estándar y el error relativo de las soluciones comparadas de los problemas 1 y 2, tienen valores cercanos a cero, quiere decir que no hay una diferencia notoria entre las soluciones del StaticSolver 1.0 con las obtenidas de manera manual y mediante el TKSolve y también se comprobó la exactitud de la solución del StaticSolver 1.0 reemplazando los valores en la ecuación 34. Debido a lo anterior se deduce que el StaticSolver 1.0 brinda soluciones muy confiables a sistema de ecuaciones no lineales.

Para algunos problemas se requiere un mayor tiempo de cálculo debido a que su solución inicial está muy alejada de la correcta, en estos casos se recomienda detener el cálculo y utilizar otra solución inicial para cada variable del sistema de ecuaciones utilizando la opción **Información de Variables** de la barra de herramientas.

RECOMENDACIONES

El StaticSolver 1.0 se encuentra en desarrollo lo que quiere decir que está disponible para agregar nuevas características y de solucionar posibles errores. El uso de este software es de carácter académico y está sujeto a los criterios definidos por la Universidad Industrial de Santander.

Se recomienda utilizar su código fuente como base para realizar versiones futuras de este tipo de programas, para ampliar las aplicaciones y mejorar la funcionalidad del software, agregando propiedades de fluidos que permitan resolver problemas aplicados a termodinámica, transferencia de calor, etc.

Debido a que el software interpreta cada ecuación carácter por carácter ocasiona que la solución de un sistema de ecuaciones tome mucho tiempo. Se puede mejorar el modo de interpretación de ecuaciones, desarrollando en el programa un reconocimiento por palabras y así ahorrar tiempo en la solución de ecuaciones.

BIBLIOGRAFIA

- **BEER, Ferdinand P. Johnston, E. Russell & Eisenberg, Elliot R.** Estática mecánica vectorial para ingenieros. Mc. Graw-Hill.

- **HIBBELER, Russel Charles.** Mecánica vectorial para ingenieros. Mc. Graw-Hill.

- **NORTON, Robert L.** Diseño de Norton. Mc. Graw-Hill.

- **CHAPRA, Steven C.** Métodos numéricos para ingenieros: con aplicaciones en computadoras personales**.** Mc. Graw Hill.

- **GABALA, Javier C.** Algebra Numérica. Ingeniería informática.

- **VADILLO, Fernando.** Ecuaciones no lineales. Universidad del país Vasco.

- **OSORIO, Rojas Alan D.** Manual teórico práctico C++. España 2006.

- **GARCIA, Ignacio A.** Introducción para la programación para Windows con Visual C++. Universidad de Oviedo.

- http://www.codeproject.com. Pagina donde se encuentran las librerías XGRAPH2005, GRIDCTRLLIB y CRYSTAL EDIT-SYNTAX COLORING TEXT EDITOR.

ANEXOS

Anexo A. MANUAL DE USUARIO

INTRODUCCIÓN

Este manual le permitirá aprender a utilizar todas las funcionalidades básicas de **StaticSolver 1.0.** ¿Qué es **StaticSolver 1.0**?

StaticSolver 1.0, es una herramienta de apoyo al estudiante de ingeniería, diseñada para resolver sistemas de ecuaciones no lineales. Este software se caracteriza por la introducción de las ecuaciones de la forma como se plantean a mano, gracias al procesador de texto de la hoja de ecuaciones. En el momento en que el usuario escriba las ecuaciones y el sistema no tenga ninguna clase de error se puede ejecutar el cálculo de la solución de las variables de las ecuaciones planteadas, el usuario tiene la opción de utilizar la tabla paramétrica para poder variar la solución de acuerdo a la variable independiente escogida y también cuenta con una herramienta para visualizar los valores de las columnas de la tabla paramétrica por medio de una gráfica.

REQUERIMIENTOS:

- ✓ StaticSolver 1.0 está diseñado para trabajar bajo los sistemas operativos Microsoft Windows 2000, xp/vista®.
- ✓ Ordenadores personales compatibles AMD/IBM/Intel ®.
- ✓ Mínimo 32 mega bytes de memoria RAM.
- ✓ Monitor a color.
- ✓ Mouse.

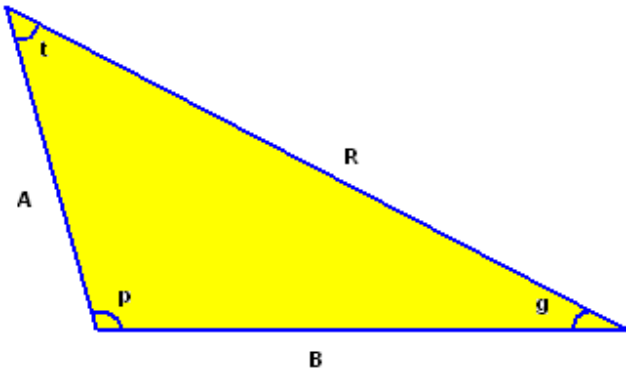
El programa está constituido por varias ventanas y cuadros de diálogos:

1. VENTANA DE HOJAS DE ECUACIONES

Esta ventana es una herramienta de escritura que funciona como un procesador de texto, se utiliza para plantear las ecuaciones tal como se escriben en el papel, en esta ventana se consigna todo el sistema de ecuaciones a solucionar y comentarios del sistema de ecuaciones. Cada ecuación se escribe en su respectiva línea de escritura.

Mediante el siguiente ejemplo se pretende capacitar al usuario para el uso de las aplicaciones del software. Un ejemplo es el triángulo escaleno de lados A-B-R que contiene los ángulos t, g y p, como se muestra en la Figura 1, donde la distancia $A = 20\ cm,\ B = 40\ cm$ y el ángulo $t = 30\ °$. Hallar la distancia R del triángulo.

Figura 1. Geometría del triangulo escaleno A-B-R



Se plantea es siguiente sistema de ecuaciones para hallar da distancia R:

$$A = 20 \quad (1)$$

$$B = 40 \quad (2)$$

$$t = 30 \quad (3)$$

$$R^2 = A^2 + B^2 - 2 \times A \times B \times \cos(p) \quad (4)$$

$$\frac{A}{\sin(g)} = \frac{B}{\sin(t)} \quad (5)$$

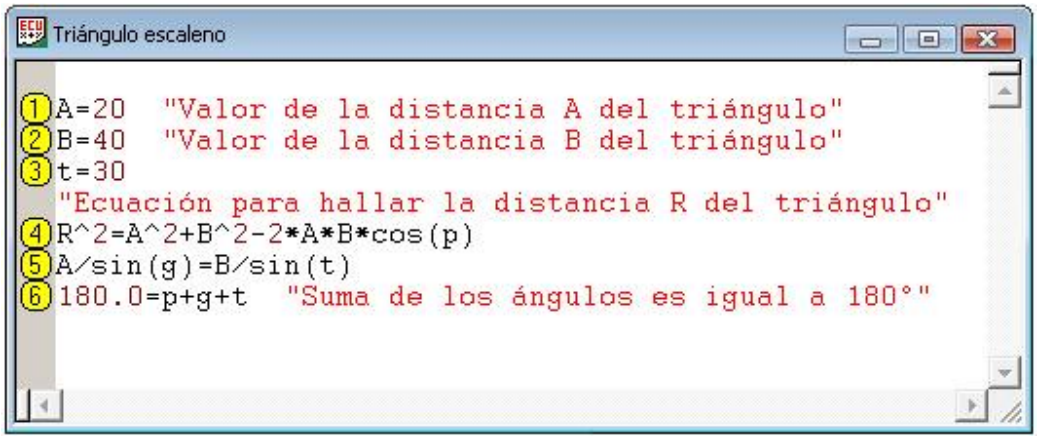
$$180 = p + g + t \quad (6)$$

Las ecuaciones pueden ser introducidas en cualquier orden. El usuario debe entender que el orden de las ecuaciones no tiene el efecto en la solución.

Los operadores matemáticos usados para la ventana de hoja de ecuaciones se describen: SUMA, RESTA, PRODUCTO, DIVISOR (+,-, *, /). Por ejemplo, la ecuación 4 para elevar la variable R al cuadrado, se utiliza el signo de intercalación (^). También las funciones, como las trigonométricas como sin y cos se utilizan seguidamente de paréntesis.

Los nombres de las variables deben comenzar con una letra y deben constar de cualquier caracter del teclado excepto ( ) * + - ^ { }. Los siguientes son caracteres indebidos (# % & ~ @ ? ¿ ¡ ! ¬ | ° ; _ [ ] ´ ¨ ` ') que no se deben usar ya que el StaticSolver 1.0 no los reconocerá. La máxima longitud de una variable es 20 caracteres. Los valores numéricos en la ventana pueden tener una coma (,) o un punto (.). Por regla general, las variables no deberían recibir nombres que corresponden a las funciones incorporadas, como por ejemplo la función PI ( ).
En la Figura 2 se representan las ecuaciones dentro de la ventana de Hoja de Ecuaciones:

Figura 2. Ventana de Hoja de Ecuaciones

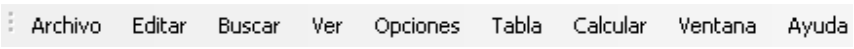


Las líneas en blanco no afectan la solución al sistema de ecuaciones debido a que el software reconoce solo las líneas donde existan caracteres, puede haber

espacios entre ecuaciones, esto hace de la ventana Hoja de Ecuaciones más legible. Los comentarios serán exhibidos en la hoja de ecuaciones dentro de comillas, por ejemplo la ecuación 6 de la figura 2, tiene el cometario " Suma de los ángulos es igual a 180°", cabe resaltar que en el momento de crear un cometario el color del texto cambia a rojo. Los comentarios pueden extenderse a lo largo de líneas múltiples.

La Hoja de ecuaciones hace uso del menú principal de la Figura 3:

Figura 3. Menú principal

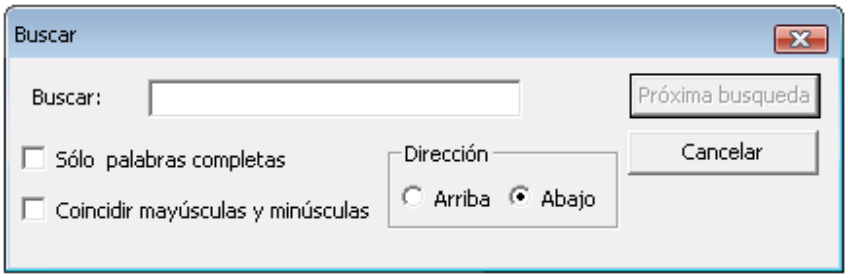


Archivo: Permite crear un nuevo documento, abrir, guardar al usuario archivos de la hoja de ecuaciones. También contiene las opciones de impresión del documento.

Editar: Permite las opciones de deshacer, rehacer, cortar, copiar, pegar, borrar, seleccionar todo, del texto seleccionado, la opción para solo lectura y los marcadores de ecuaciones.

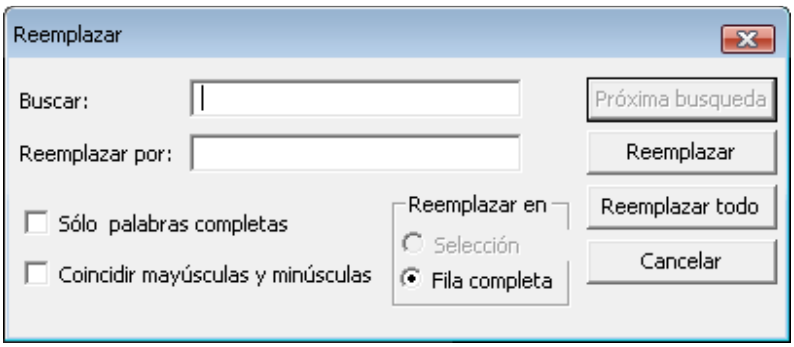
Buscar: contiene el diálogo Buscar usado para encontrar letras (Figura 4):

Figura 4. Diálogo Buscar



También la opción que muestra el diálogo Remplazar utilizado para reemplazar palabras o letras (figura 5):

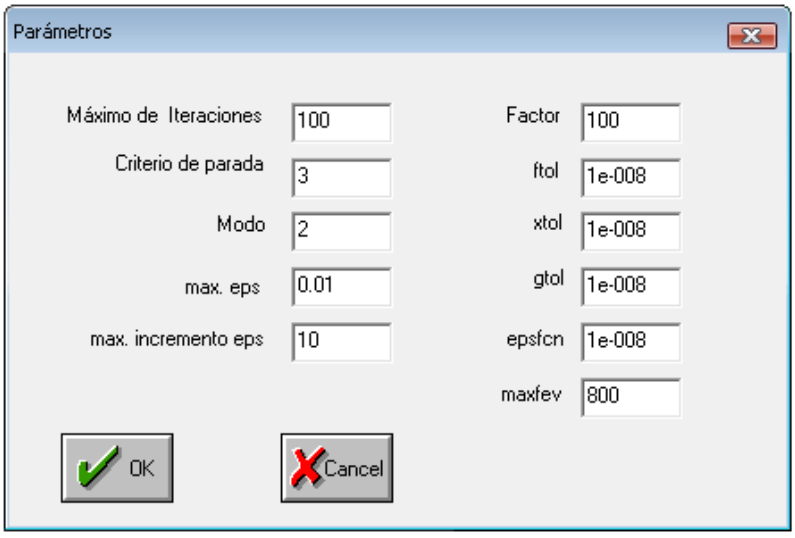
Figura 5. Diálogo Reemplazar



Ver: visualiza la barra de herramientas y de estado.

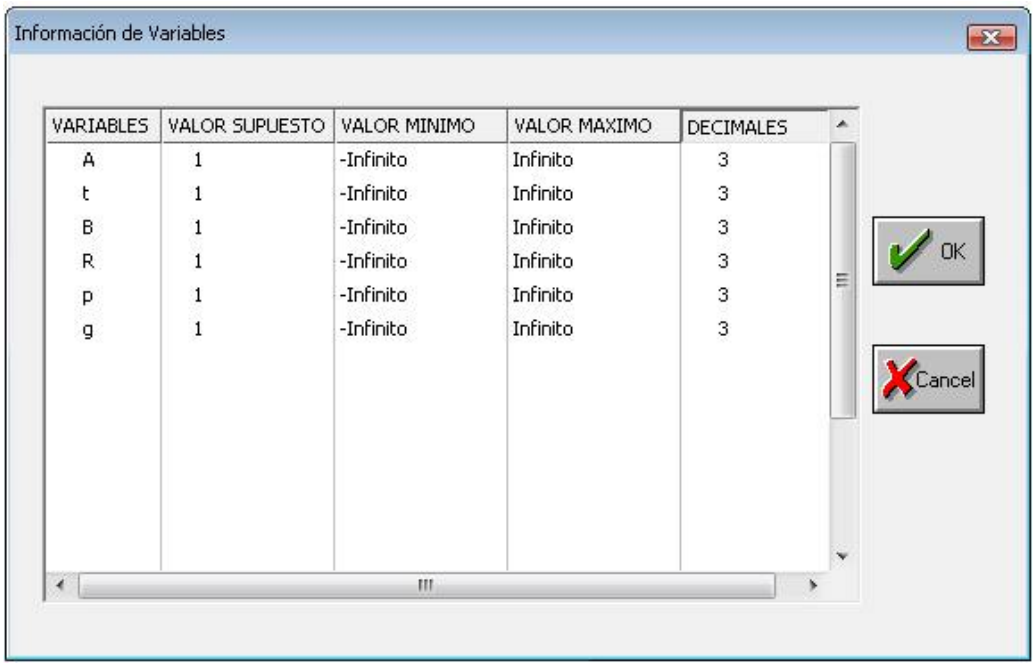
Opciones: permite el acceso al diálogo de parámetros, al diálogo Información de variables y detener el cálculo. En el primer diálogo (Figura 6) se encuentran los parámetros para la solución del sistema de ecuaciones tales como el número máximo de iteraciones, factor, modo, entre otros, dando al usuario la opción para variar estos valores, los cuales afectan en la exactitud de la solución.

Figura 6. Diálogo Parámetros



Para el ejemplo del triángulo, en la Figura 7, el diálogo Información de variables contiene varias columnas: una de variables (depende del sistema de ecuaciones a solucionar), de valor supuesto (es el valor de la solución inicial para cada variable), valor máximo y mínimo al que puede llegar la solución de determinada variable y el número de decimales que contiene la solución de cada variable.

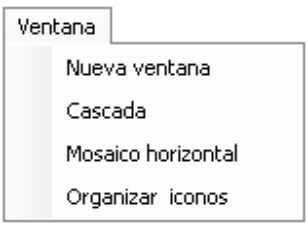
Figura 7. Diálogo Información de variables



Calcular: contiene el comando para calcular el sistema de ecuaciones.

Ventana: contiene opciones para la ventana, nueva ventana, cascada, mosaico horizontal y organizar iconos, como en la Figura 8.

Figura 8. Menú Ventana



Ayuda: viene con el menú acerca de StaticSolver 1.0 (Figura 9).

Figura 9. Acerca de StaticSolver

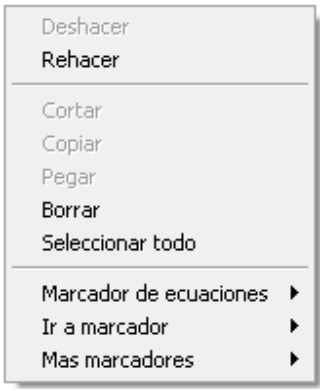
En la Hoja de ecuaciones dispone de la barra de herramientas Estándar como en la Figura 10, tiene los iconos de las opciones del menú Archivo, Editar y Ayuda.

Figura 10. Barra de herramientas estándar

Se habilitan tres iconos de la barra de herramientas  al introducir las ecuaciones, que son: Opciones de parámetros, Información de variables y calcular sistema de ecuaciones.

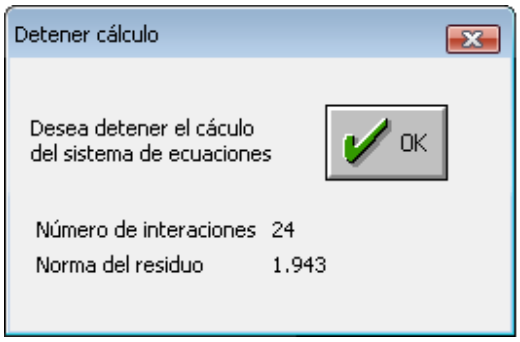
Dando un clic sobre el botón derecho del ratón en la ventana de Hoja de Ecuaciones mostrará un menú emergente, tiene las opciones que se ven en la Figura 11, en particular tiene la opción de marcadores de ecuaciones que sirve para numerar las ecuaciones con un círculo amarillo al inicio de la línea de la hoja de ecuaciones como en la Figura 2.

Figura 11. Menú emergente de la Hoja de Ecuaciones



Al iniciar la solución del sistema de ecuaciones con el botón  aparecerá el cuadro de dialogo de la Figura 12 donde aparece la opción para detener el cálculo, muestra el número de iteraciones y la norma respectiva que tiende a cero cuando se acerca a la convergencia.

Figura 12. Diálogo Detener cálculo

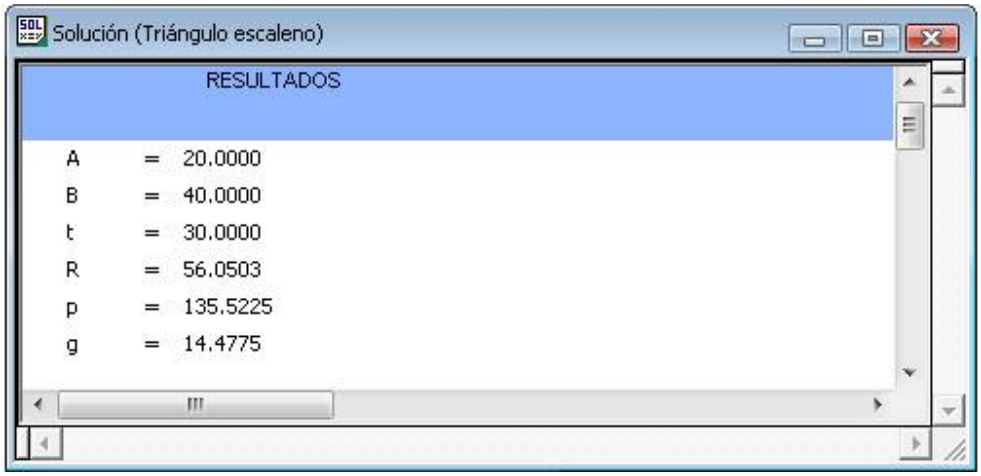


2. VENTANA SOLUCIÓN

La ventana de Solución aparecerá adelante de todas las demás ventanas después de que los cálculos sean completados.

Una vez el número de ecuaciones sea igual al número de incógnitas y estén correctamente escritas cada ecuación de la ventana de hoja de ecuaciones, se le dará solución con **Calcular** al sistema de ecuaciones y aparecerá la ventana de solución, con los respectivos resultados con valores de las cada una de las variables del sistema de ecuaciones así como en la Figura 13:

Figura 13. Ventana de la solución



El menú Opciones de la ventana Solución, solo se habilitará el menú Cambiar Fuente.

Aparecerá una nueva ventana de Solución, si ocurre cualquier cambio en la ventana de Ecuaciones y se calcula de nuevo el sistema de ecuaciones.

3. VENTANA TABLA PARAMÉTRICA

Cada Tabla Paramétrica funciona como una hoja de cálculo que está constituida por columnas y filas, con sus respectivas celdas. La tabla es generada con el comando **Nueva tabla paramétrica**. No hay límite para el número de tablas paramétricas que puede ser generado, aparte de la memoria disponible. Cada columna en una Tabla Paramétrica mantiene valores para una variable del sistema se ecuaciones seleccionada. El valor numérico se designa en las celdas. Se asume que valores introducidos en las celdas de cada una de las variables, estas pasan a ser variables independientes (no dependen del valor de otras variables). Introducir un valor en una celda de una variable, produce el efecto de añadir a la hoja de ecuaciones una ecuación con valor de la variable de la celda, quiere decir, que no se le debe dar valor a la variable independiente en la hoja de ecuaciones. Una advertencia se muestra si una variable es colocada para un valor en ambas ventanas hoja de ecuaciones y la Tabla Paramétrica cuando la orden de la solución de la tabla es dada. Las variables dependientes son por defecto las celdas vacías debido a que en estas celdas se acomodan los valores que se determinen en la solución de la tabla. Por cada Fila en la tabla se debe asignar las variables dependientes e independientes y estas filas representan un sistema de ecuaciones diferente.

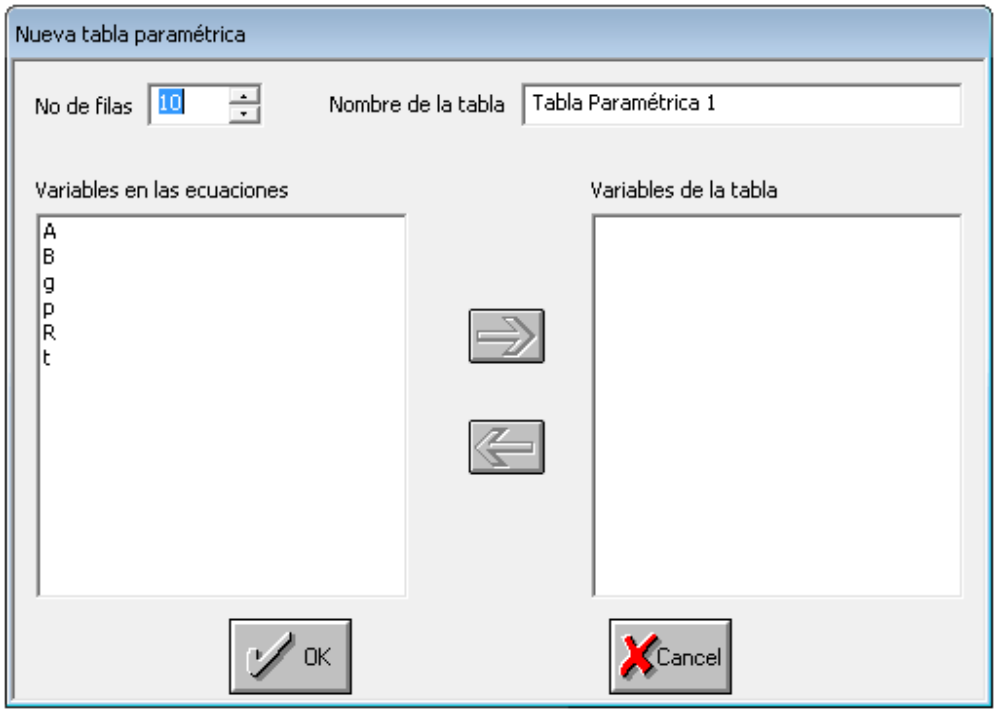
Las variables que aparecen en la tabla son la lista de variables de de la hoja de ecuaciones, las cuales serán el encabezado de las columnas. El número de las filas también se podrán escoger. Cada tabla paramétrica nueva debe recibir un nombre por el usuario, el nombre aparece en la parte superior de la ventana Tabla Paramétrica.

Una tabla es generada usando el comando  **Nueva tabla paramétrica** de la barra de herramientas, pero antes de ser visualizada la ventana de tabla

paramétrica, aparecerá un con el cuadro de diálogo llamado **Nueva tabla paramétrica** como el de la Figura 14. En el diálogo aparecen dos listas principales, una para las variables de las ecuaciones y la otra para las variables de la tabla, también un cuadro para el nombre de la tabla paramétrica y la opción para escoger el número de filas.

Las variables pueden agregarse o suprimirse de la tabla paramétrica, usando el botón agregar  o borrar  del diálogo **Nueva tabla paramétrica.**

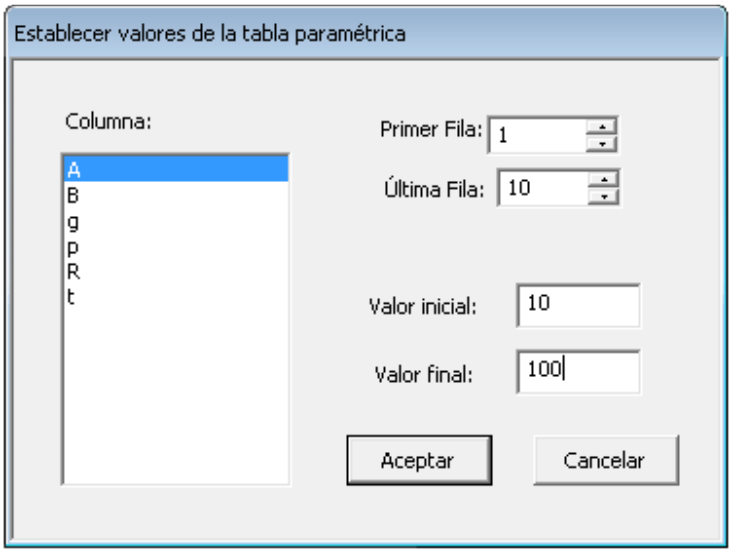
Figura 14. Cuadro de diálogo Nueva tabla paramétrica



Una vez escogidas las variables de la tabla, mediante el botón  se genera automáticamente la tabla paramétrica. En caso de anular la anterior operación se presiona  que es el botón de cancelar.

Los valores para cualquier columna de la tabla paramétrica pueden ser introducidos en cada celda o dando un clic sobre el botón derecho del ratón dentro de las celdas de la tabla paramétrica con el comando Establecer valores de tabla o con el botón de la barra de herramientas. Al ejecutar el comando mostrará un cuadro de diálogo **Establecer los valores de la tabla paramétrica** como aparece en la Figura 15. En el diálogo se escoge la columna, es decir la variable de la tabla, el rango de filas que se quieren introducir los valores y valor inicial y final de la columna seleccionada.

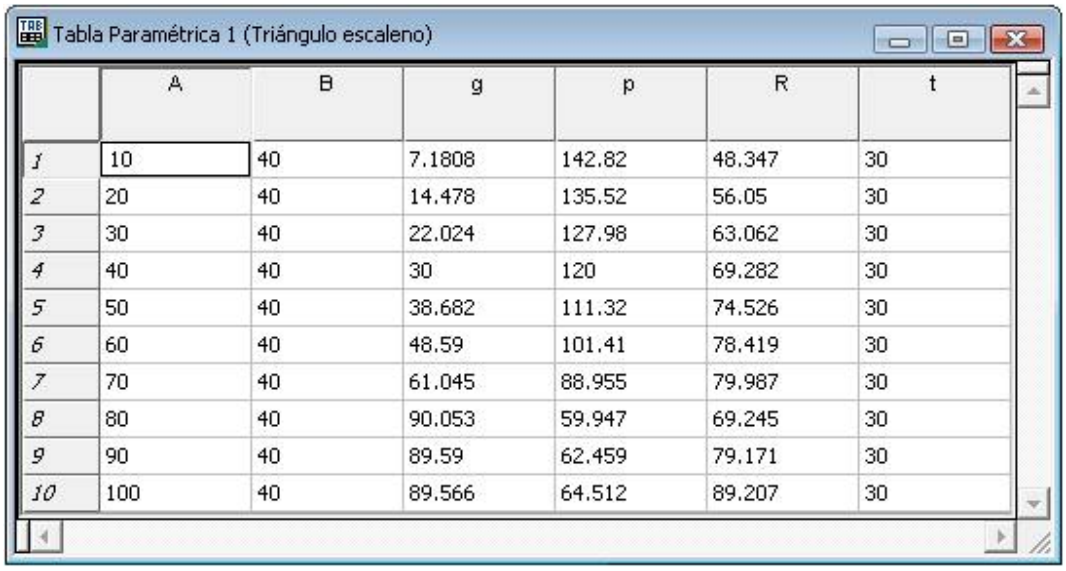
Figura 15. Establecer valores de tabla



Cada fila de la Tabla Paramétrica es un cálculo separado que se inicia por la orden de **Calcular tabla** con el botón de la barra de herramientas. El número de filas es escogido antes que la tabla sea generada.

Para el ejemplo de triángulo escaleno se escogieron 10 filas, las columnas de variables A, B, g, p, R y t, se tomo columna de la variable A como variable independiente y se le dieron valores de 10 en 10 hasta 100, luego se calculó la tabla así como se ve en la Figura 16:

Figura 16. Resultados de la Tabla paramétrica 1



	A	B	g	p	R	t
1	10	40	7.1808	142.82	48.347	30
2	20	40	14.478	135.52	56.05	30
3	30	40	22.024	127.98	63.062	30
4	40	40	30	120	69.282	30
5	50	40	38.682	111.32	74.526	30
6	60	40	48.59	101.41	78.419	30
7	70	40	61.045	88.955	79.987	30
8	80	40	90.053	59.947	69.245	30
9	90	40	89.59	62.459	79.171	30
10	100	40	89.566	64.512	89.207	30

En la Tabla paramétrica 1 (Triángulo escaleno), la solución del valor de cada variable dependiente (B, g, p, R, t) varia a medida que el valor de variable independiente A va incrementando su valor de 10 en 10 hasta 100.

En la ventana Tabla paramétrica aparece el menú Tabla (ver Figura 17), que contiene diferentes opciones para realizar cambios en la tabla paramétrica.

Figura 17. Menú Tabla

Selección de celda: seleccionar, seleccionar fila sobre celda fija, seleccionar columna sobre celda fija.

Tamaño de Celda: autoajuste de celdas, ampliar última columna, ajustar celdas.

Líneas de Cuadrícula: mostrar líneas verticales, mostrar líneas horizontales.

Operaciones de Cuadrícula: insertar fila, borrar fila, cambiar fuente.

Opciones de Cuadrícula: edición, título de información, foco de celda, marco de celda, ordenar sobre encabezado.

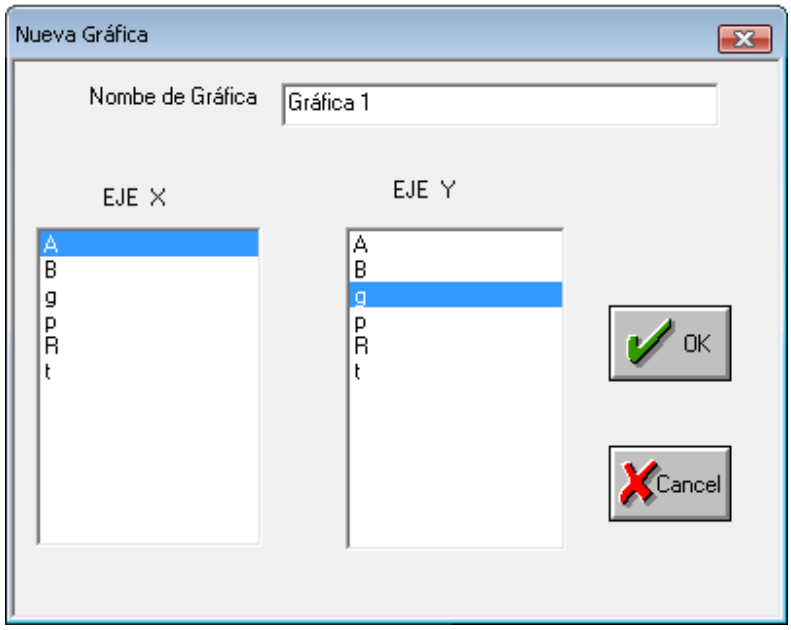
Modo Lista: activar modo lista y selección simple.

Cambiar Tamaño: tamaño de fila y tamaño de columna.

4. VENTANA DE GRÁFICOS

Los valores de las variables que aparecen en la tabla paramétrica pueden ser graficados en la ventana de gráficos, a partir del diálogo **Nueva gráfica** que aparece mediante la orden del botón  Nueva Gráfica de la barra de herramientas o del menú de Nueva Gráfica. Dada la orden para generar la nueva gráfica, aparecerá el cuadro de diálogo de la Figura 18.

Figura 18. Cuadro de diálogo Nueva Gráfica

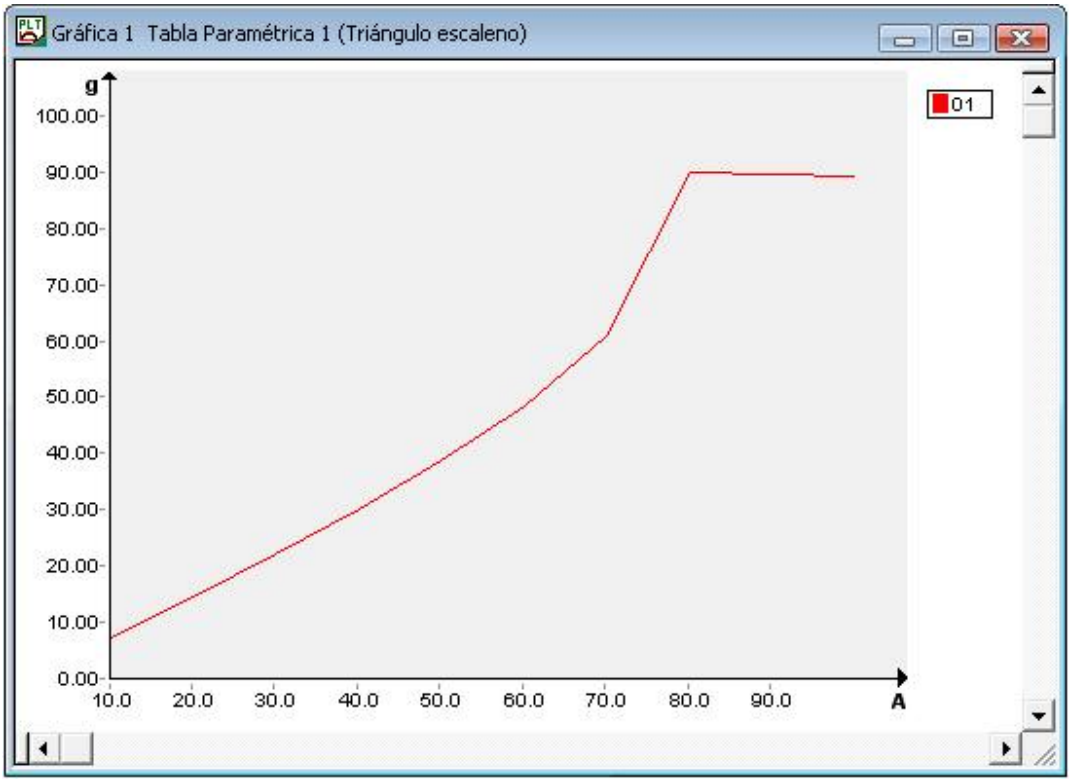


El diálogo Nueva Gráfica provee un cuadro de texto para darle nombre a la gráfica y dos listas de las variables seleccionadas de la tabla paramétrica respectiva para la elección de ejes X-Y. Se escogen dos variables para establecer el eje x y eje y. Una vez se escogen los ejes y se le da el nombre a la gráfica, se presiona el botón **OK** para que aparezca la ventana de gráfica, donde resultan graficados los valores de las variables escogidas de la tabla paramétrica. Para cancelar la gráfica se presiona el botón **Cancel.**

Los campos del texto para etiquetar las ejes X y Y son generados cuando un gráfico se construye. El texto adicional puede ser añadido al gráfico dando clic derecho dentro de la ventana de gráficos en el menú **comentario.**

Para el ejemplo de triángulo escaleno se nombra la gráfica como **Gráfica 1** y se escogen las variables A y g que representan el eje X y el eje Y respectivamente y presionado el botón **OK** se genera la siguiente ventana de gráfica como en la Figura 19:

Figura 19. Ventana de la Gráfica 1



En la Tabla 1 se encuentran los valores de las variables A y g del problema del triángulo escaleno graficados en la Figura 19.

Tabla 1. Valores de la Tabla Paramétrica 1 graficados

	A	G
1	10	7.1808
2	20	14.478
3	30	22.024
4	40	30
5	50	38.682
6	60	48.59
7	70	61.045
8	80	90.053
9	90	80.59
10	100	80.566

La ventana de gráficos y los cuadros de diálogo de esta ventana, ofrecen un número de opciones gráficas como la elección del tipo de línea, selección de ejes y de la curva, gráfica escalada, tendencia lineal, cúbica y polinomial, cuadrícula, ampliación, introducción de texto, colores de ejes y curvas, entre otras. Estas opciones pueden ser determinadas una vez el gráfico es dibujado utilizando la barra de herramientas o el menú que aparece dando clic derecho en cualquier punto de la gráfica.

No hay límite para el número de ventanas de gráfica que pueden ser creadas (aparte de las limitaciones de memoria). La apariencia del gráfico puede variarse de muchas formas usando el menú o la barra Herramientas. El menú Herramientas esta determinado como se muestra a en la Figura 20:

Figura 20. Menú de Herramientas de la ventana de Gráficos

Selección: utilizada para seleccionar un eje, una curva, cuadro de texto o cualquier objeto que este dentro de la gráfica.

Zoom: amplia la gráfica, funciona trazando un cuadro para ampliar en cualquier parte de la gráfica que se desee, presionado el clic derecho hasta darle tamaño al cuadro del zoom.

Editar: Esta opción permite editar los datos de la curva de la gráfica de tal manera que la curva puede moverse de la posición original.

Restablecer: una vez se amplia la gráfica, si se requiere que la gráfica vuelva a su posición original se utiliza esta orden.

Trasladar: es la manera para mover la gráfica en distintas direcciones. El gráfico asociado puede ser movido a cualquier posición de la ventana de gráficos, dando un clic izquierdo sobre el botón del ratón dentro del gráfico y arrastrando el ratón.

Medición: esta opción del menú de Herramientas, se utiliza para medir distancias entre dos puntos de la gráfica.

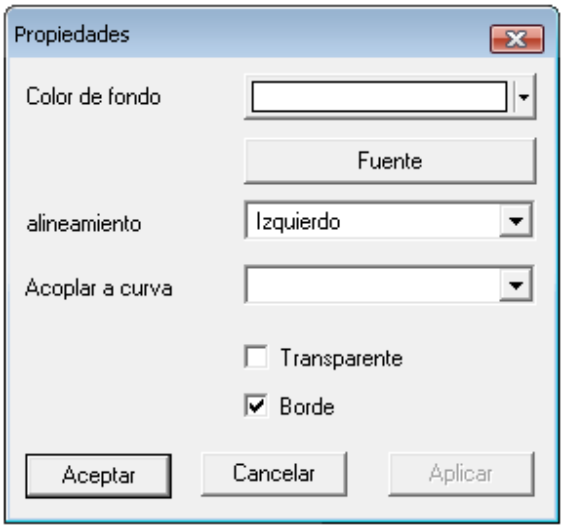
Cursor: un cursor aparece mostrando los valores de la curva en la posición actual del ratón dando clic en los puntos de la curva. Aparecerá un cuadro en la esquina superior izquierda de la ventana, indicando los valores de las variables según la curva.

Comentario: dar un clic sobre este comando, aparecerá un cuadro para la introducción de texto como en la Figura 21:

Figura 21. Cuadro de texto

Para seleccionar el cuadro de texto se debe hacer clic dentro del cuadro, el cuadro de texto seleccionado también pueden ser movido cuando el cursor tome la forma de flechas. Para dimensionar el cuadro arrastramos el ratón de los lados o bordes del cuadro. Para deseleccionar el texto, se debe dar un clic sobre el ratón en la ventana de gráficos en una posición donde no hay texto, líneas o cuadros. Para cambiar las propiedades del texto se da clic derecho dentro del cuadro de texto y aparecerá la opción propiedades que visualiza el diálogo Propiedades de la Figura 22. En el diálogo se pueden cambiar las características del texto como el color del texto, color de fondo, fuente de letra, el alineamiento (izquierdo, centrado y Derecho), acoplar curva, transparencia y borde del cuadro.

Figura 22. Diálogo Propiedades



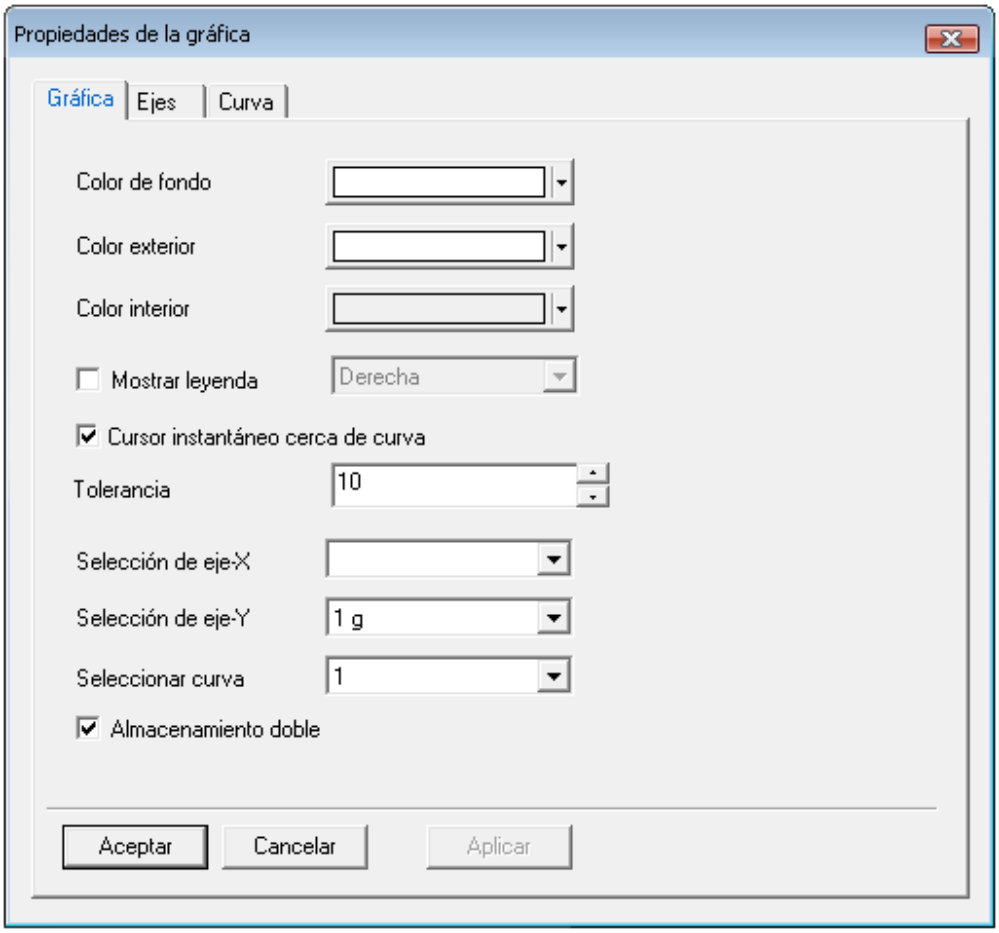
Propiedades: En la ventana de gráficos dispone de un cuadro de diálogo llamado **Propiedades de gráfica** que ofrece tres tipos de propiedades, una para la gráfica, para los ejes y para la curva. Estas propiedades se mencionan a continuación:

1. **Gráfica:** tiene la opción para cambiar el color de fondo de la gráfica, el color exterior, el color interior, estas opciones muestran un cuadro de colores como el de la Figura 23:

Figura 23. Cuadro de colores

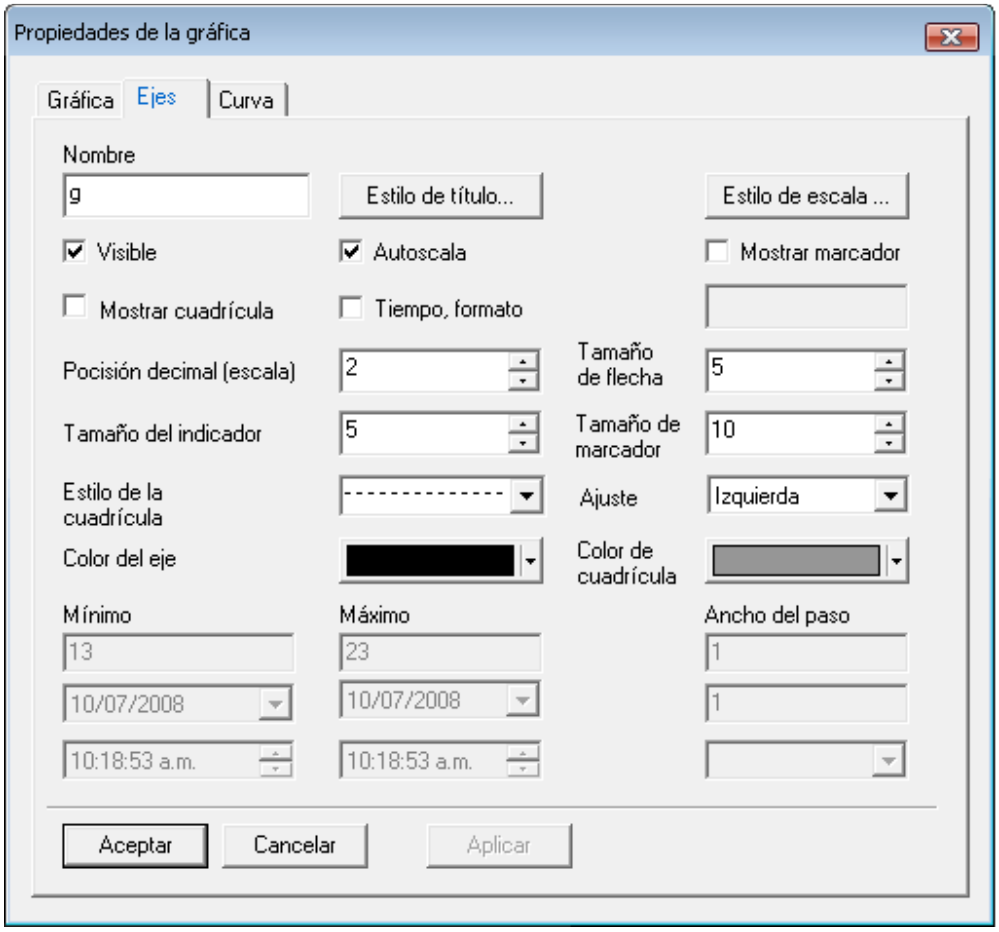
También contiene la opción para mostrar la leyenda, tolerancia, selección de la curva, selección de eje x y del eje y como se muestra en la Figura 24:

Figura 24. Diálogo Propiedades de gráfica: Gráfica



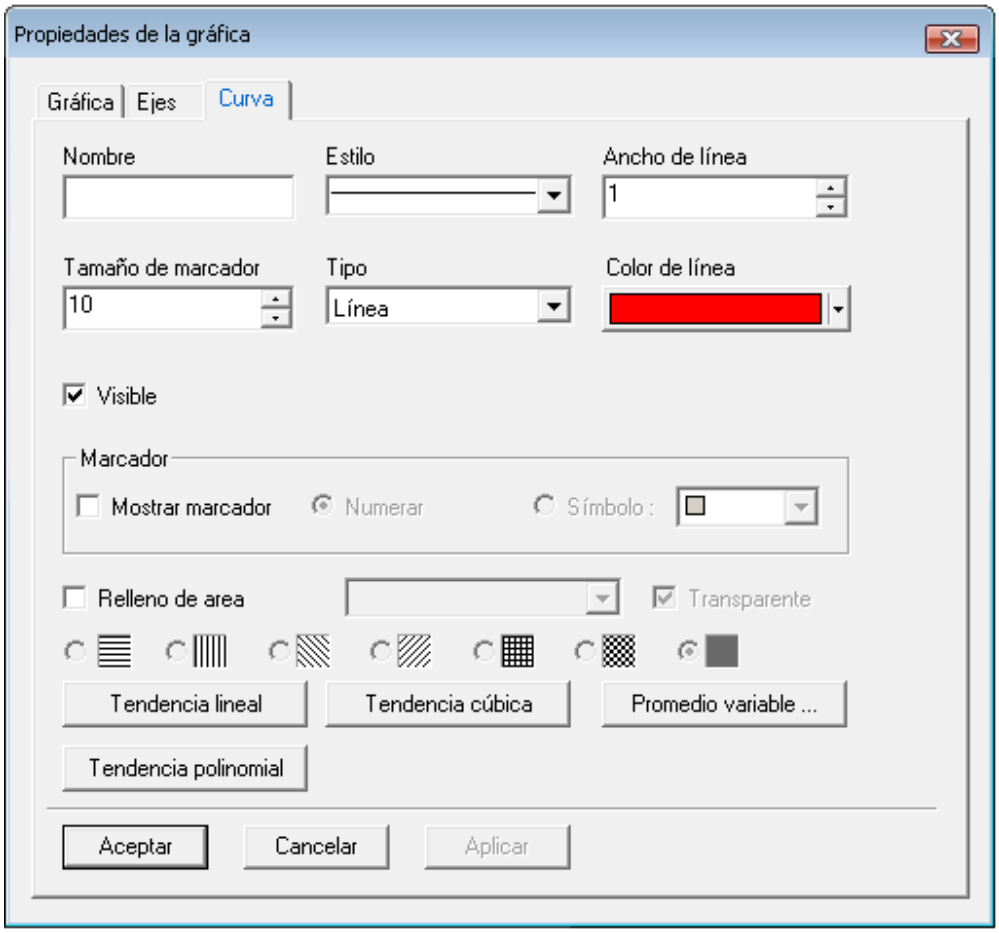
2. **Ejes:** presenta las opciones para los ejes (ver Figura 25) tales como: cambiar de nombre, estilo de titulo y escala (fuente del título y la escala), ejes visibles, la autoescala, tamaño del indicador, mostrar el marcador, tamaño de flecha y marcador, mostrar cuadrícula, estilo y color de la cuadrícula, muestra el ancho de paso, valor mínimo y máximo en el eje.

Figura 25. Diálogo Propiedades de gráfica: Ejes



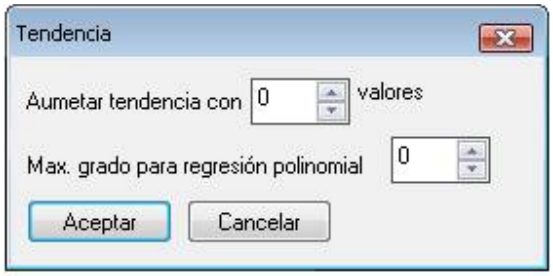
3. **Curva:** opciones para la curva (ver Figura 26) de nombre, color, estilo, ancho de línea de la curva. Opciones de marcador como mostrar marcador, tamaño de marcador, numerar, forma de símbolo. Opciones de tipo de curva ya sea por línea o dispersión, color de línea, curva visible, relleno de área, tendencia lineal, cúbica, polinomial y promedio variable de la curva.

Figura 26. Diálogo Propiedades de gráfica: Curva



En las opciones de la tendencia aparecerá un cuadro de diálogo (Figura 27) donde se puede aumentar la tendencia y limitar el máximo grado de regresión polinomial.

Figura 27. Tendencia



En las opciones de promedio variable aparecerá un cuadro de diálogo (Figura 28) donde se puede variar el tipo de promedio variable ya sea simple, lineal, exponencial, triangular y de seno aritmético. También el tamaño en la ventana

Figura 28. Promedio variable

Los archivos tienen diferentes extensiones ya sea de hoja de ecuación (.ecu), de solución (.sol), tabla paramétrica (.tab) y gráficos (.plt). Estos se podrán guardar independientemente. Cada archivo simplemente se guarda dando clic en la opción guardar del menú archivo.

Para imprimir cualquier tipo de archivo, también se puede desde el menú archivo con la opción imprimir.

Anexo B. GLOSARIO

NEWTON-RAPHSON: El método de Newton es un algoritmo eficiente para encontrar aproximaciones de los ceros o raíces de una función real.

PROGRAMACIÓN ORIENTADA A OBJETOS: es un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (es decir, datos), comportamiento (esto es, procedimientos o métodos) e identidad (propiedad del objeto que lo diferencia del resto). La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

CLASES: una clase es la definición de un tipo de objetos.

LIBRERÍAS: son códigos que contienen funciones que son frecuentemente utilizadas y que no necesita ser modificados. Las librerías se usan para desarrollar software.

VISUAL STUDIO.NET (2005): Visual Studio 2005 es un software de programación que proporciona una variedad de herramientas que ofrece amplios beneficios tanto para desarrolladores individuales como para equipos de desarrollo.

Anexo C. INSTALACIÓN Y DESINSTALACION DEL PROGRAMA STATICSOLVER 1.0

INSTALACIÓN DEL SOFTWARE STATICSOLVER 1.0

1. Ir a la carpeta donde esta el instalador y hacer doble clic sobre el icono Setup.
2. Seguir las instrucciones que indica el programa y luego aceptar.
3. Al completar la instalación, presionar el botón cerrar.
4. Para entrar al programa ir a: inicio/todos los programas/StaticSolver 1.0

DESINSTALACIÓN DEL SOFTWARE STATICSOLVER 1.0

1. Ir a inicio/panel de control/agregar o quitar programas/seleccione el programa y oprima en el botón cambiar o quitar.
2. Confirmar que desea borrar el programa.