Uso de .NET Framework para el Desarrollo de Software Matemático y de Ingeniería

Esquivel, Gabriel A., Pellettieri, Daniel N.

Universidad Tecnológica Nacional, Facultad Regional Buenos Aires

Abstract

El presente trabajo pretende presentar un resumen de los logros alcanzados en el desarrollo de bibliotecas para aplicaciones matemáticas, científicas y de ingeniería utilizando la plataforma .NET Framework y las bibliotecas gráficas DirectX de Microsoft.

Palabras Clave

Bode – Clases - C# - Estructuras - FFT - Fourier - GDI – IDE - Instancia - Laplace - Nyquist- .NET Framework - POO – Parser – Realimentación - String - Tipo de dato - Visual Basic - Visual Studio 2005

Introducción

Durante el dictado de la asignatura Electrónica Aplicada II (en FRBA de la UTN) es frecuente estudiar el comportamiento de los circuitos electrónicos mediante análisis gráficos que responden a las ecuaciones circuitales.

Desafortunadamente no todo el software disponible hace algunos años atrás permitía realizar esto de forma efectiva y simple, sin mencionar el costo del software especializado, que por lo general se halla fuera del alcance de los alumnos.

El objetivo inicial en ese entonces fue poder trazar gráficas de Bode para funciones racionales de variable compleja, a partir de la especificación de sus polos y ceros, utilizando una aplicación Windows realizada en Visual Basic, pero los buenos resultados obtenidos derivaron en el trabajo que aquí se describe.

Elementos del Trabajo y metodología

Como lenguaje de programación en un principio se utilizó Visual Basic 6.0, intentando con éste graficar funciones matemáticas de diferente índole.

A fines de 2003 el software denominado para ese entonces BodeViewer permitía graficar funciones a partir de polos y ceros o de su expresión algebraica, en 2 o 3 dimensiones. Incluso era posible obtener gráficas de Nyquist y lugar de raíces para las funciones racionales, o buscar las raíces reales en forma interactiva por diferentes métodos.

Sin embargo, el desarrollo de la aplicación con programación estructurada impuso muchísimas limitaciones, sobre todo para la expansión del soft, ya que cada mejora agregada implicaba la modificación de gran parte del código.

La inclusión del software en un proyecto de desarrollo patrocinado por Microsoft a comienzos de 2004, motivó la necesidad de migración a un lenguaje relativamente nuevo llamado C# (léase C sharp) que estaba montado sobre una plataforma muy robusta denomina .NET Framework.

Esto implicó el cambiar el paradigma de programación, adoptando desde ese momento POO[1] (programación orientada a objetos), y obligó a reescribir el código completo, propiciando todo tipo de mejoras.

Sin embargo, diferentes desarrollos paralelos implementados en visual Basic .NET nos brindaron más experiencia en este último. Esa es la razón por la cual el código se rescribió definitivamente en Visual Ba-

[1] Programación Orientada a Objetos

sic.NET, y dado que ambos se compilan en el lenguaje intermedio denominado IL[2], el resultado obtenido es tan eficiente y estable como si fuese escrito en C#.

Desde el punto de vista de un programador, el .NET Framework ofrece una vasta y robusta base de desarrollo que permite crear tanto aplicaciones como bibliotecas (archivos con extensión .DLL) de la más variada índole. En cuanto al entorno de desarrollo (IDE[3] por sus siglas en ingles) denominado Visual Studio 2005, ofrece un entorno muy amigable y a la vez potente por ofrecer una gran variedad de herramientas útiles para el programador.

Con respecto al código, fue posible crear en forma simple y eficaz clases y estructuras básicas para los números complejos y demás tipos básicos, y a la vez contar en éstos con operadores y funciones que permiten resolver las más variadas expresiones matemáticas.

Por ejemplo, para definir el tipo de un número complejo se puede escribir una estructura simple como la siguiente:

```
Public Structure Complex

    Public Real As Double
    Public Imaginary As Double

    Public Shared Operator +(ByVal
Z1 As Complex, ByVal Z2 As Complex)
As Complex
        Return New Complex(Z1.Real
+ Z2.Real, Z1.Imaginary +
Z2.Imaginary)
    End Operator

    Public Shared Operator -(ByVal
Z1 As Complex, ByVal Z2 As Complex)
As Complex
        Return New Complex(Z1.Real
- Z2.Real, Z1.Imaginary -
Z2.Imaginary)
    End Operator
End Structure
```

En éste ejemplo básico la estructura posee dos variables miembro o campos (*Real* e *Imaginary*) que almacenarán respectivamente la parte real e imaginaria del número complejo al que representan.

Además, se han definido en éste caso dos operadores, con lo cual podremos de ahora en adelante en nuestro código sumar y restar variables complejas como si de cualquier otro tipo numérico se tratase, por ejemplo:

```
Sub CodigoCualquiera()
    Dim a, b, c As Complex

    a.Real = 10
    b.Imaginary = 5
    c = a + b

End Sub
```

Si se observa la penúltima línea, a la variable *c* se le asigna el resultado de la suma de *a* con *b*, mediante el uso del operador + definido dentro de la estructura, y con una sintaxis que no podía ser más clara.

Es más, es posible darle aún mayor versatilidad a la estructura si en lugar de campos utilizamos propiedades, ya que éstas se comportan en parte como campos, pero admiten además la inclusión de código, con lo cual se le brinda mayor operatividad a éste pseudo-campo.

También reutilizamos y reemplazamos miembros heredados, como por ejemplo la función *ToString*, la cual ahora devuelve un string con el formato $R \pm I \cdot i$, donde i es la unidad imaginaria, R es la parte real e I la imaginaria del número complejo.

Después de todas las modificaciones y agregados, y tras renombrarla a *ComplexUndefinied* la estructura adquirió la siguiente apariencia que ejemplificamos en la Fig. 1

[2] IL: Intermediate Language. Lenguaje intermedio al que son compilados las diferentes secuencias de código de alto nivel (VB .Net, C#, J#, etc.) de .Net
[3] Integrated Development Environment

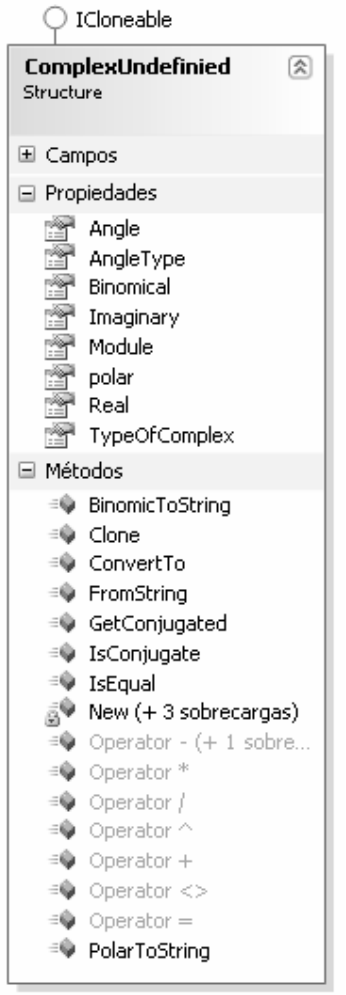

Figura 1: Diagrama de clases de la Estructura *ComplexUndefinied*

Como puede apreciarse, el uso de propiedades permite disponer de otras características de un número complejo, como ser el módulo y su argumento, que se (calculan sin que el usuario se dé cuenta) a partir de la parte real e imaginaria almacenadas en la estructura.

En cuanto a los métodos implementados, existe una gran variedad de los mismos. El método *GetConjugated*, por ejemplo, permite obtener una instancia con el valor conjugado del número complejo examinado. Otro ejemplo es el método *FromString*, que permite establecer los valores internos a partir de una cadena de caracteres.

La creación de una biblioteca para operar con números complejos no es tan difícil después lo visto, por lo que después de cierto tiempo se obtuvo un arsenal completo de clases accesorias que permitan manipular datos de éste y de otros tipos, así como presentar los resultados en la forma deseada.

Para poder obtener funciones a partir de expresiones expresadas en la simbología matemática convencional, se creó una clase con métodos capaces de "parsear"[4] strings[5], además de toda una arquitectura de clases que representan a funciones matemáticas y son administradas en forma centralizada por ésta. Ello permitió que las funciones pudieran ser compuestas, es decir, tener funciones que dependen de otras funciones.

También se tuvo en cuenta la necesidad de prever futuros tipos de funciones que no necesariamente tuviesen como base una expresión matemática habitual, como por ejemplo las funciones expresadas mediante la Transformada de Laplace, que se pueden definir a partir de sus polos y ceros, y las que se les dio soporte en éstas bibliotecas.

Para la visualización de las funciones y como una forma extra de verificar los resultados obtenidos, se crearon complejas bibliotecas de controles a los que se denominó Visualizadores, los que aceptan datos con diferentes formatos, y son capaces de presentar los resultados con el estilo o escala requerida por el investigador.

La versatilidad de .NET Framework junto al entorno de desarrollo permitieron la construcción de herramientas accesorias para el análisis de funciones y las graficas 2D, como por ejemplo clases que realizan el análisis FFT[6], o el caso de una clase para manipular cursores. También se han desarrollado herramientas de Zoom para inter-

[4] Parsing: Mecanismo que permite analizar una secuencia, determinar su estructura y "traducirla"
[5] Cadenas de caracteres
[6] Fast Fourier Transform (Transformada Rápida de Fourier)

actuar cómodamente con las gráficas, y objetos que permiten la búsqueda de raíces reales dentro de un intervalo dado, presentando una gran variedad de opciones de configuración que pueden ser establecidas fácilmente por el usuario.

Finalmente, es preciso destacar que, mediante el uso de las bibliotecas graficas DirectX[7] de Microsoft (las que usualmente se utilizan para juegos de PC de gran velocidad), fue posible obtener en forma más que satisfactoria graficas 3D de funciones de 2 variables y funciones de variable compleja, y estas podían ser manipulas prácticamente en tiempo real por el .usuario

Resultados

La fig. 2 muestra el aspecto de la aplicación principal.

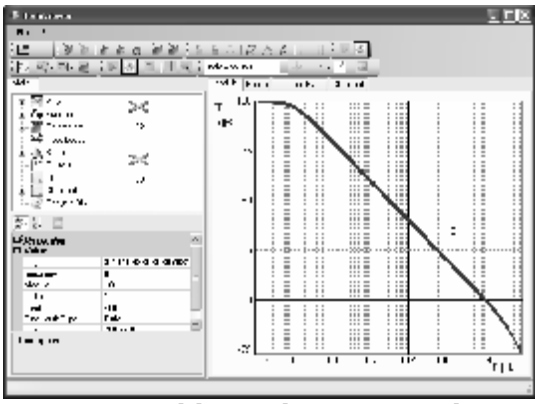

Figura 2: Vista del Formulario correspondiente a la aplicación BodeViewer

Se trata de una típica aplicación de Windows (tipo formulario) que contiene barras de herramientas y menús que permiten el rápido acceso a las herramientas accesorias antes descritas.

El visualizador cuenta con diferentes *tabs* (pestañas) que permiten observar graficas 2D y 3D de las funciones como resultado.

La fig. 3 muestra el detalle de una gráfica de un visualizador de escalas lineales presentando los datos de dos funciones reales, más precisamente un coseno de amplitud unitaria y una parábola cuya expresión es:

$y = 0,01 \cdot x^2 - x + 1$

Dichas funciones se evalúan en el intervalo $(0;10)$:

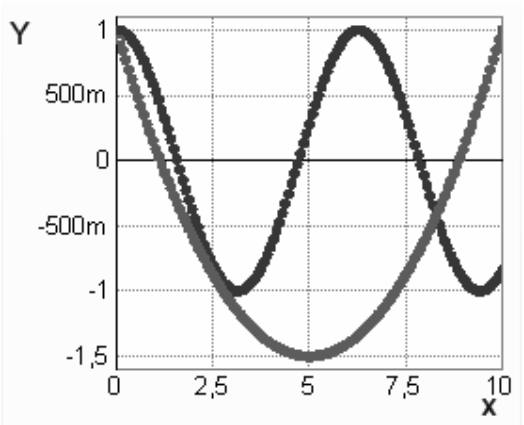

Figura 3: Visualización en 2D de las funciones $y = \cos(x)$ **e** $y = 0,01 \cdot x^2 - x + 1$

Sin embargo es posible especificar al visualizador otro estilo de presentación de sus escalas, como por ejemplo, escalas logarítmicas para ambos ejes. Esto se ejemplifica en la fig. 4, en la que se aprecian las graficas de Bode de modulo de una función Transferencia *A*, la realimentación *B* (usualmente denominado Beta) y la respuesta a lazo cerrado *Af*, definidas como:

$$A = \frac{10^5}{\left(1 + z/10^3\right) \cdot \left(1 + z/10^5\right)}$$

$$B = 10$$

$$Af = \frac{A}{1 + B \cdot A}$$

Donde z es la variable compleja independiente, mientras que *Af* es una función compuesta.

En éste caso en particular es posible observar el pico de realimentación positiva cerca de 1MHz para el circuito que tiene como funciones las mencionadas.

[7] Conjunto de API's desarrolladas por Microsoft para facilitar el desarrollo de juegos en MS Windows

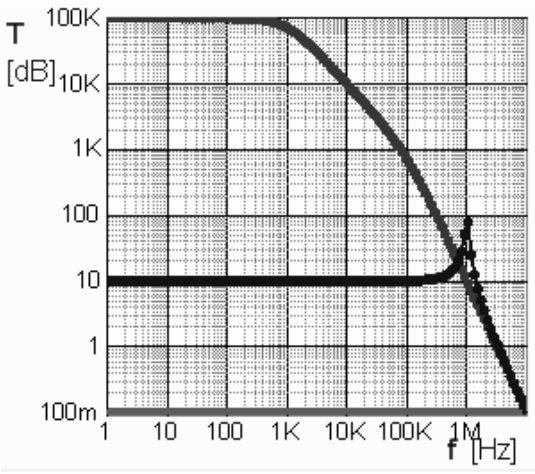

Figura 4: Visualización de funciones con escalas logarítmicas, como por ejemplo un Diagrama de Bode

Por otro lado, tal como se mencionó, es posible obtener el lugar de raíces[8] para el caso de las funciones de Laplace. Por ejemplo, dada la función:

$$A = \frac{10^5}{(1+z/100)\cdot(1+z/1500)\cdot(1+z/4000)}$$

Su diagrama de lugar de Raíces se observa como en la fig. 5

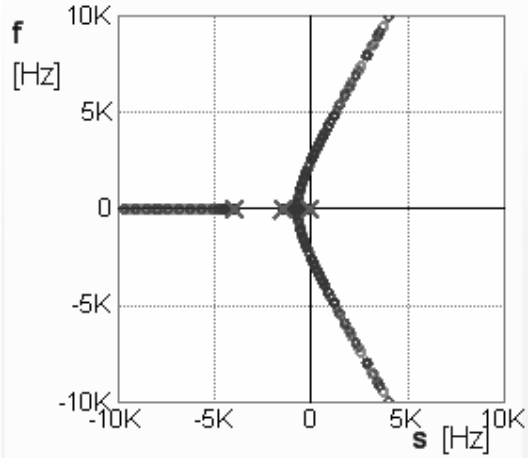

Figura 5: Visualización de un diagrama de lugar de raíces

Si observa la gráfica de la fig. 5, podrá ver la ubicación de los polos representados como cruces, y mediante círculos la trayectoria de éstos para diferentes niveles de realimentación.

En cuanto al análisis de Fourier de funciones, el resultado para un pulso de amplitud y ancho unitarios se aprecia, conjuntamente con su FFT, en las figs. 6 y 7:

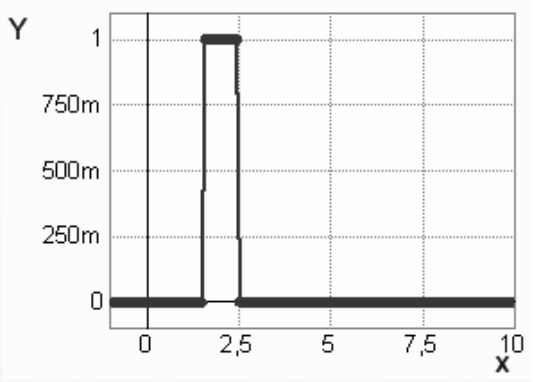

Figura 6: Pulso de amplitud y ancho unitarios

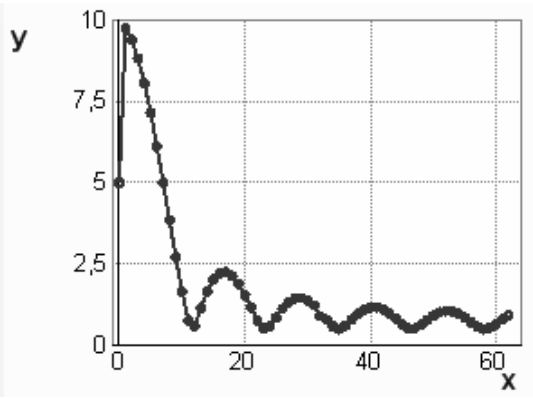

Figura 7: Transformada de Fourier del pulso unitario de la fig. 6

Como último ejemplo de visualización de funciones en 2D, la fig. 8 muestra los gráficos de la herramienta para la búsqueda de raíces reales de una función real dentro de un dado intervalo. En ésta aparece en forma visual el intervalo de búsqueda y los trazos rectos que permiten explicar el método utilizado, que en éste caso es el conocido como el de la *Falsa Posición*. En otra sección de la aplicación ésta informa que a encontrado la raíz en $x = 3{,}141593$.

[8] El lugar de raíces de una función de transferencia H(s) (a lazo abierto) es un diagrama de los lugares donde estarán los polos a lazo cerrado para los diferentes valores de ganancia k del lazo de realimentación.

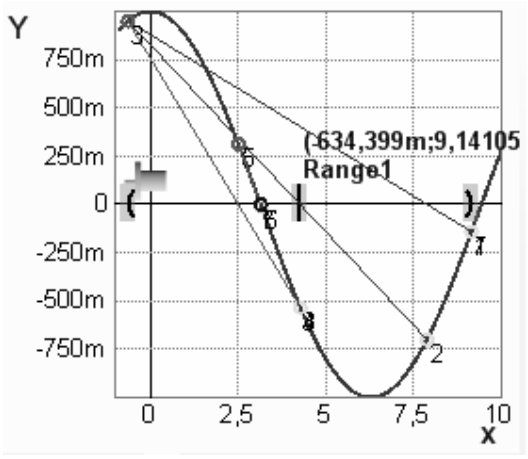

Figura 8: Grafico de la aplicación de búsqueda de raíces

Veamos ahora algunos ejemplos del visualizador 3D. Para el caso de funciones de dos variables, puede observarse en la fig. 9 la grafica de una semiesfera obtenida con la expresión:

$$f(x,y)=\sqrt{40-x^2-y^2}\,.$$

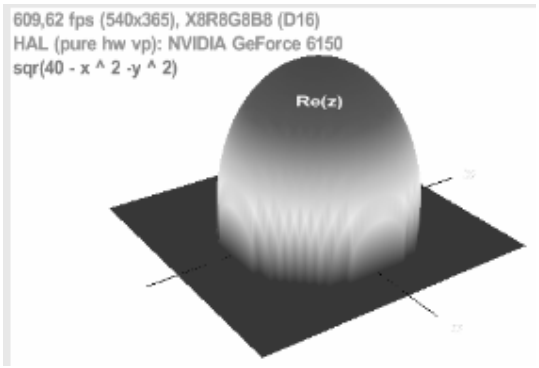

Figura 9: Gráfico en 3D de una semiesfera

Para funciones de variable compleja tomamos primero a un seno con la expresión $g(z)=\sin(0,3\cdot z)$ y mostramos su modulo (fig. 10):

Aquí es posible observar otra de las utilidades de ésta herramienta gráfica, ya que se puede apreciar como los cortes transversales a la superficie dan senos, mientras que los cortes longitudinales presentan a una función tipo coseno hiperbólico. Se puede entender fácilmente la relación matemática entre ambos si recordamos que:

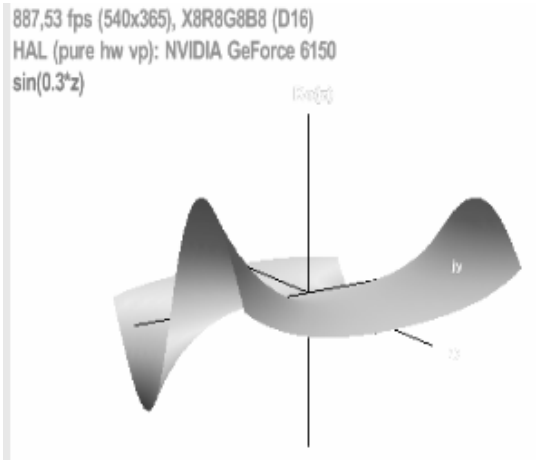

Figura 10: visualización 3D de una función de variable compleja

$$\sin(z)=\frac{e^{i\cdot z}-e^{-i\cdot z}}{2\cdot i}$$

Mientras que el coseno hiperbólico puede escribirse como:

$$\cosh(z)=\frac{e^{z}+e^{-z}}{2}$$

Es fácil ver que éstas expresiones tienen relación una con otra.

Otro ejemplo interesante es el de una función denominada habitualmente sinc, definida para variable compleja como:

$$sinc(z)=\frac{\sin(z)}{z}$$

Al evaluar el módulo de ésta función en el plano complejo se obtiene algo similar a lo mostrado en la fig. 11:

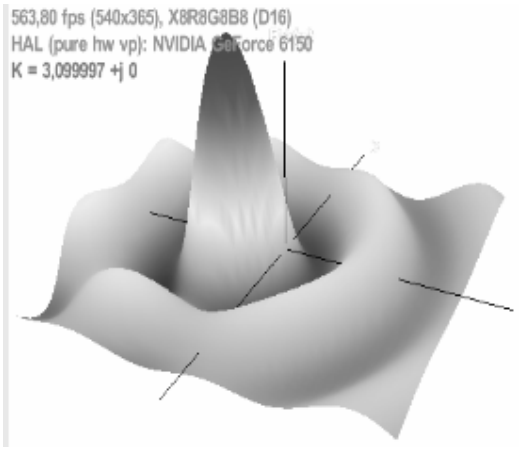

Figura 11: Gráfica de la función sinc(x)

En realidad, la gráfica de la fig. 11 corresponde a una suma de funciones *sinc* desplazadas una de la otra, lo que da un aspecto de interferencia entre ondas. Si bien no es observable en el gráfico estático, la aplicación permite ver en forma dinámica el efecto de la variación del valor de un parámetro cualquiera, dadas las características de animación que posibilita el uso de DirectX.

Discusión

Uno de los puntos más problemáticos al principio fue pasar de programación estructurada a la orientada a objetos. Ello se debió a que, al menos en nuestro caso particular, es habitual que en la especialidad Electrónica se trabaje con problemas para los cuales basta el paradigma de programación estructurada. Pasar entonces de algo a lo que se está muy acostumbrado a un concepto totalmente distinto fue todo un desafió, no por la programación orientada a objetos en sí, sino por el cambio de mentalidad necesario para implementarlo correctamente.

Otro tema a resaltar es la relativa dificultad de trabajar con las API's de DirectX, que no están tan profusamente documentadas como el Framework, lo que sumado al cambio de versiones constantes hace que no resulte fácil escribir código que las utilice.

Conclusión

Por lo visto en las secciones anteriores, nos parece más que evidente que la potencia de diseño de software que nos brinda .NET Framework y su IDE Visual Studio 2005, para el caso particular de las aplicaciones matemáticas, científicas y de ingeniería, que hacen de ésta la elección ideal para cualquier emprendimiento.

Al mismo tiempo, se puede comprobar como la POO brinda las herramientas necesarias para darle la envergadura necesaria a un proyecto; algo que de ninguna manera hubiese sido posible con la programación estructurada antes utilizada.

Agradecimientos
Ing. Osvaldo Pini
Ing. Jorge Sinderman.
Ing. Alejandro Furfaro
Lic. Oscar Noguez

Referencias.

[1] JAGGER, JON y SHARP, JHON, Microsoft Visual C# .NET Step by Step, 2002, Microsoft Press.

[2] Microsoft Visual Basic .NET Lenguaje Reference.

[3] PETROUTSOS, EVENGELOS, Mastering Visual Basic.NET, 2002, Sybex

[4] VETTERLING, WILLIAM T., PRESS, WILLIAM H. y otros, Numerical Recipes, Cambrige University Press.

[5] SÁNCHEZ, JULIO y CANTÓN, MARíA P., DirectX 3D Graphics Programming Bible, 2000 – IDG Books

[6] SCHNEIDER, PHILIP J. y EBERLY, DAVID H., Geometric Tools for Computer Graphics, 2003 – Morgan Kaufmann – Elsevier

Datos de Contacto:
Ing. Gabriel Esquivel. UTN FRBA.
gaesquivel@argentina.com
Ing. Daniel Pellettieri. UTN FRBA.
dnpellet@ciudad.com.ar