Introducción a la Programación Científica

Octave

Es un **entorno integrado** diseñado para el **cálculo científico** y la **visualización de datos**. Al que Python, podemos **interactuar con el intérprete** de Octave, como se muestra en el Ejemplo 1. Notemos varias cosas. Primero, podemos **evaluar expresiones** matemáticas, usando el intérprete como una calculadora avanzada, con una sintaxis y semántica muy similar a Python. El intérprete guarda el resultado de nuestras expresiones en una variable especial llamada ans. También podemos definir nuestras **variables** y darle valores directamente (sin usar ans). Estos valores pueden ser **escalares** (un valor atómico), **vectores** o **matrices**. Se operan estos valores cuando matemáticamente está definida la operación. Por último, notemos que existen operadores matemáticos **elemento a elemento** para vectores o matrices del mismo tamaño.

Ejemplo 1

```
octave:1> 2 ** 8 + (5 - 4) / 2 * 5
ans =  258.50
octave:2> ans / 4
ans =  64.625
octave:3> escalar = 5
escalar =  5
octave:4> vector = [1:7]
vector =
   1   2   3   4   5   6   7
octave:5> matriz1 = [1 2 3; 4 5 6]
matriz1 =
   1   2   3
   4   5   6
octave:6> matriz2 = [1 0 0; 0 1 0]
matriz2 =
   1   0   0
   0   1   0
octave:7> escalar * vector
ans =
    5   10   15   20   25   30   35
octave:8> escalar * matriz1
ans =
    5   10   15
   20   25   30
octave:9> matriz1 + matriz2
ans =
   2   2   3
   4   6   6
octave:10> matriz1 * matriz2
error: operator *: nonconformant arguments (op1 is 2x3, op2 is 2x3)
octave:10> matriz1 .* matriz2
ans =
   1   0   0
   0   5   0
octave:11>
```

Octave también provee muchas **funciones nativas**, así como algunas **constantes** de uso común en cálculos matemáticos. Podemos consultar la documentación que se instala junto al entorno para usarlas.

Pregunta 1

Con tu grupo de trabajo, responde la primera pregunta de la actividad.

Pero dijimos que el entorno Octave también está diseñado para **visualizar datos**. Esta es una gran diferencia con el lenguaje Python, puesto que Octave provee muchas utilidades para **graficar** en dos y tres dimensiones. El Ejemplo 2 muestra cómo podemos ver gráficamente el comportamiento de un polinomio en un rango de valores, específicamente del polinomio $(x+1)^7 = x^7 + 7x^6 + 21x^5 + 35x^4 + 35x^3 + 21x^2 + 7x + 1$. La Figura 1 muestra el resultado entregado por Octave.

Ejemplo 2

```
octave:1> coefs = [1 7 21 35 35 21 7 1];
octave:2> x = -2:0.05:0;
octave:3> length(x)
ans =  41
octave:4> y = polyval(coefs, x);
octave:5> plot(x, y)
octave:6> hold on
octave:7> plot(x, y, 'o.r')
octave:8> title 'Polinomio grado 7'
octave:9>
```

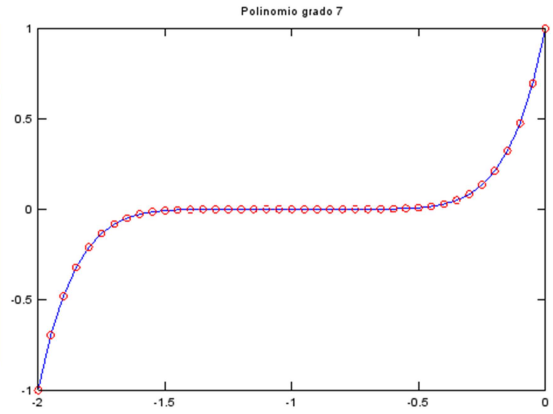


Figura 1

Podemos aprender varias cosas en el Ejemplo 2. La primera sentencia define un vector, que en este caso contiene los coeficientes del polinomio $(x+1)^7$, que es almacenado en la variable `coefs`. Como la sentencia termina en un **punto y coma** (`;`), el intérprete de Octave no muestra en pantalla el resultado de la operación. La segunda sentencia también define un vector, con valores para la variable x del polinomio, que se asigna a la variable `x`. Pero en este caso se usa el **operador dos puntos** (`:`) con tres parámetros para indicar que genere los valores del -2 al 0 con espacios de 0.05. Recordemos que si bien una función puede estar definida para todos los reales, los computadores modernos tienen una **arquitectura digital**, y por lo tanto sólo podremos computarla para **un número finito** de puntos. Valores flotantes separados por 0.05, o 41 puntos en [0, -2], parecen ser lo suficientemente "continuo" como para **apreciar el comportamiento** del polinomio.

La función `length()`, tercera sentencia, nos permite conocer el largo de un vector. Notemos que para ver el resultado, no debemos finalizar la sentencia con un punto y coma. En la cuarta sentencia, la función `plyval()` evalúa el polinomio definido por los coeficientes en `coefs`, en los valores contenidos en el vector `x`. EL resultado es almacenado en la variable `y`.

La función `plot(x, y)` en la quinta sentencia es la que solicita al intérprete de Octave graficar en dos dimensiones los puntos definidos por los vectores `x` e `y`. En la configuración *by default*, Octave abre una ventana, grafica los puntos (x_i, y_i) y la línea que extrapola esos puntos. Notemos que `x` e `y` deben tener el mismo largo para esto. El resultado es la línea azul en el gráfico de la Figura 1. La sentencia 7 también grafica los puntos definidos por los vectores `x` e `y`, pero esta vez solicitamos que sólo se marquen los puntos con círculos rellenos de color rojo (`'o.r'`). Para que esta sentencia no genere un nuevo gráfico, sino que aparezca sobre la gráfica anterior, usamos la sentencia `hold on` (sentencia 6). La última sentencia agrega un título al gráfico generado.

Al igual que en Python, podemos crear un **programa** con un editor de texto y luego indicar a Octave que lo **interprete**. El Ejemplo 3 muestra el listado del archivo `ejemploSeno.m`. En este programa podemos ver que el signo "gato" (*sharp/hash*) también convierte una línea en un **comentario** que es ignorado por el intérprete. Pero en Octave, también podemos crear un **bloque de comentarios** usando los signos `#{` y `#}` para abrirlo y cerrarlo respectivamente.

Ejemplo 3: Programa `ejemploSeno.m`

```
#{
 Programa: ejemploSeno.m
 Objetivo: ejemplificar el "ploteo" de la funcion seno
 Autor: Coordinacion
#}

# Grafica la funcion en el intervalo [0, 8pi] (en verde)
angulo = [0:pi/12:8*pi];
seno = sin(angulo);
plot(angulo, seno, 'g');

# Coloca titulos y etiquetas (labels) de los ejes
title 'Funcion trigonometrica seno'
xlabel('Angulo (radianes)')
ylabel('Seno')

# Obtiene objeto "ejes" y modifica los "ticks" y
# las etiquetas del eje X
ejes = gca();
xTicks = [pi/2:pi:8*pi];
xTickLabels = [ "pi/2";  "3pi/2";  "5pi/2";  "7pi/2";
                "9pi/2"; "11pi/2"; "13pi/2"; "15pi/2"];
set(ejes, 'xlim', [0, 8 * pi])
set(ejes, 'xtick', xTicks)
set(ejes, 'xticklabel', xTickLabels)
```

El Ejemplo 3 también muestra cómo obtener acceso al **objeto** que maneja los **ejes del gráfico**, el que luego es modificado a nuestra conveniencia cambiando algunas de sus **propiedades** con la función `set()`. Este tipo de objetos se llaman ***handlers*** en Octave y los hay de varios tipos para los diferentes objetos gráficos. Por ejemplo, podemos tener acceso a la **figura** completa con la función `gcf()`, lo que nos permite cambiar propiedades acerca de su tamaño, orientación, márgenes, colores, tipos de puntos, etc. que no hayamos hecho en la llamada a la función `plot()`.

Ejercicio propuesto

Interpreta paso a paso el programa del Ejemplo 3. Asegúrate de entender qué hace cada uno y los efectos que tiene sobre el gráfico que resulta. Apóyate en los manuales de Octave que se instalan con el intérprete. Son muchas las propiedades que pueden cambiarse para "personalizar" nuestros gráficos tanto como podríamos requerir.

¿Por qué Octave?

Hay varias razones para que aprendamos a utilizar el Octave. Primero, utiliza un lenguaje de **muy alto nivel**, que permite concentrarnos en el problema que estemos resolviendo, más que en los detalles de cómo implementar los objetos que tenemos que utilizar.

Segundo, está **orientado** a ser utilizado por **ingenieros** y provee muchas funcionalidades nativas para ello. Además, hay un creciente número de **bibliotecas con funciones especializadas** que extienden el lenguaje para manejar, por ejemplo, semales de sonido, video, imágenes médicas, etc.

De hecho, Octave es **altamente compatible** (pero no absolutamente compatible) con MatLab®, un entorno profesional muy usado en la industria y en las universidades, y que seguramente **usaremos en nuestro futuro académico y profesional**. Pero a diferencia de MatLab®, Octave es **distribuido libremente** bajo licencia GNU/GLP. En palabras sencillas, esto significa que **podemos usarlo** para cualquier propósito, **podemos adaptarlo** a nuestras necesidades, **podemos compartirlo** con otros, pero si compartimos una versión de Octave que hemos cambiado, debemos hacerlo bajo licencia GNU/GLP (o sea, no podemos "cobrar" por agregar una funcionalidad a Octave).

Pregunta 2

Responde la segunda pregunta de la actividad con la ayuda de tu grupo de trabajo.

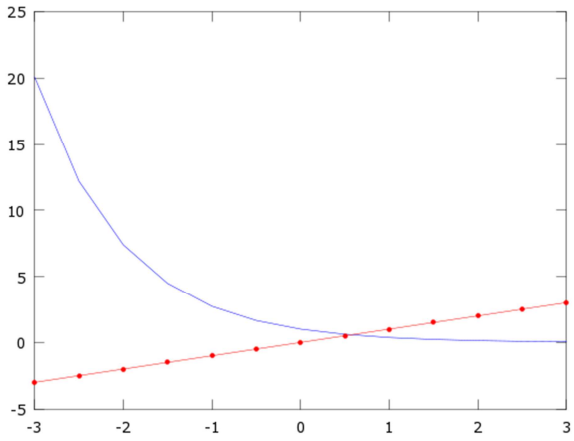
¿Para qué sirve Octave a los científicos e ingenieros?

Si bien estamos acostumbrados a "resolver" problemas en Cálculo, Álgebra y Física, la verdad que esos problemas están diseñados para medir conocimientos y habilidades de los estudiantes. Los **problemas del mundo real** son normalmente **más complejos** e involucran demasiadas variables como para resolverlos con manipulación simbólica. También hay problemas matemáticos que **no admiten** una solución analítica (como las llamadas ecuaciones trascendentales). Por ejemplo, qué valores de x satisfacen:

$$x = e^{-x}$$

Parece que los métodos que conocemos nos fallan en este caso, de hecho, ¿existe tal valor de x? Podemos graficar las expresiones por separado para evaluar su comportamiento.

```
octave:1> x = -3:0.5:3;
octave:2> plot(x, exp(-x))
octave:3> hold on
octave:4> plot(x, x, 'r')
octave:5> plot(x, x, 'o.r')
```

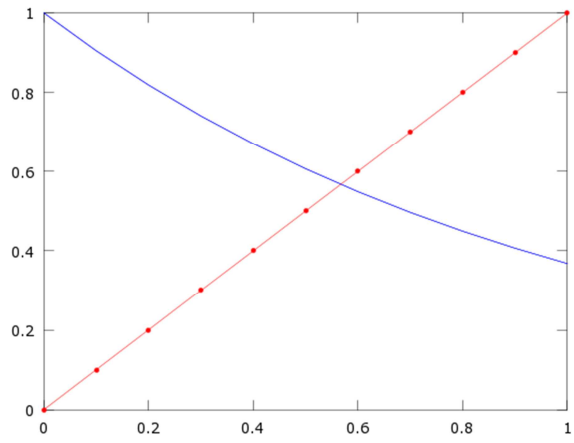


En la figura que resulta, podemos ver que existe al menos un punto en donde la ecuación se cumple. "Al ojo", ese punto parece ser $x = 0{,}5$. Comprobando:

```
octave:6> exp(-0,5)
ans = 0.60653
```

Luego $0{,}5$ es una **solución aproximada** a nuestro problema. Pero podemos refinarla para conseguir un **valor más preciso**.

```
octave:7> x = 0:0.1:1;
octave:8> hold off
octave:9> plot(x, exp(-x))
octave:10> hold on
octave:11> plot(x, x, 'r')
octave:12> plot(x, x, 'o.r')
```

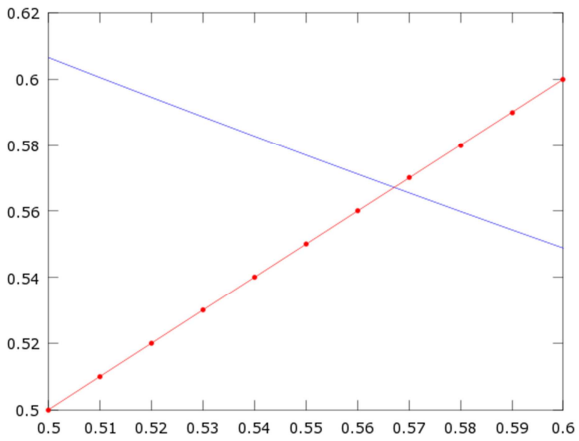


Así, usando nuestro buen ojo, una **solución más aproximada** podría ser $x = 0{,}56$.

```
octave:13> exp(-0,56)
ans = 0.57121
```

Pero podríamos ser más precisos refinando nuestra solución todavía más:

```
octave:14> x = 0.5:0.01:0.6;
octave:15> hold off
octave:16> plot(x, exp(-x))
octave:17> hold on
octave:18> plot(x, x, 'r')
octave:19> plot(x, x, 'o.r')
octave:20>
octave:20> exp(-0,567)
ans = 0.56722
```



Hemos obtenido una mejor solución aproximada, ya que las tres primeras **cifras significativas** de $x = 0{,}567$ coinciden con $e^{-x} = 0{,}56722$.

¿Pero es $x = 0{,}567$ el valor que satisface la ecuación $x = e^{-x}$? Un matemático fundamentalista respondería enfáticamente **"NO"**. De hecho, si seguimos con el mismo procedimiento y con mayor número de **cifras significativas**:

```
octave:145> format long
octave:146>
octave:146> exp(-0.5671432904)
ans= 0.56143290415333
octave:147>
```

Parece satisfacer de **mejor forma** la ecuación. ¿Pero es $x = 0{,}5671432904$ la solución que buscamos? Nuevamente la respuesta rigurosa es **"NO"**. Pero en el **mundo real**, y especialmente en la **ingeniería**, los valores 0,5, 0,56, 0,567 y 0,5671432904 pueden todas ser perfectamente **soluciones adecuadas** en nuestra aplicación. Cuánta precisión se requiere es una restricción del problema que se está enfrentando. Si se trata del largo de un perno para un puente que se está construyendo y x está en metros, entonces x=0,5671 podría ser una solución aceptable. Si x está en milímetros, entonces 0,56 podría ser más que suficiente. Lo importante es que **el puente no se caiga** en el próximo terremoto.

Este es el costo que debemos pagar al usar **computadores** y **métodos numéricos** para resolver nuestros problemas matemáticos: un cierto grado de **error es inevitable**. Formalmente, aparecen varios **tipos de errores**:

- Cuando a un fenómeno en el mundo real se le fuerza a obedecer un modelo matemático, se introduce un **error debido al modelado** (e_m), que explica la diferencia entre la solución s entregada por el modelo y la solución al problema real s_r.

- Como vimos, un modelo matemático podría no tener solución explícita con manipulación simbólica, y tendríamos que recurrir a un **método de resolución numérico**, lo que introduce **errores de aproximación** (e_a).
- Los modelos matemáticos a menudo involucran un número infinito de operaciones aritméticas, que obviamente no podemos computar, y tenemos que usar un **problema numérico finito** que aproxime el modelo matemático, dando origen a un **error de truncamiento** (e_t) entre s y la solución del problema numérico s_n.

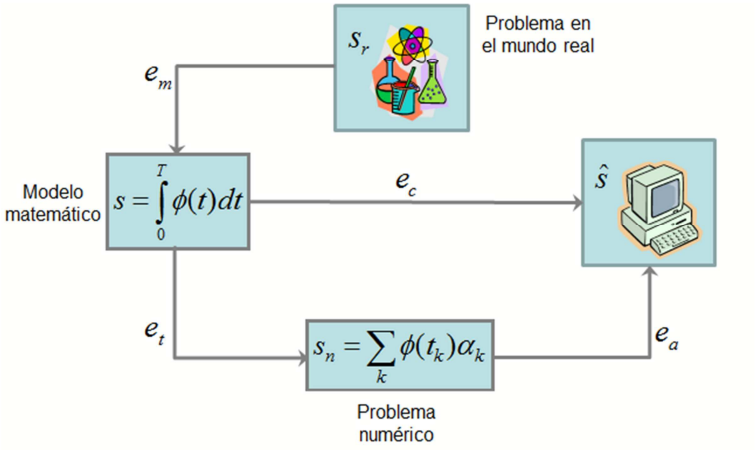


Figura 2. Fuentes de errores al resolver numéricamente problemas matemáticos.

La Figura 2 muestra un esquema con estas fuentes de errores. Considerando estos errores, aparece el **error computacional** $e_c = e_t + e_a$. Es común considerar dos tipos de errores computacionales. Primero está el **error computacional absoluto**, que es la diferencia entre la solución del modelo matemático s y la solución $\hat{s}$ obtenida al final del modelo numérico, es decir: $e_c^{\text{abs}} = |s - \hat{s}|$. Por otro lado está el **error computacional relativo**, que depende del tamaño de la solución (sólo cuando $s \neq 0$), y que se calcula como $e_c^{\text{rel}} = \frac{|s-\hat{s}|}{|s|}$.

Es muy importante tener en cuenta que estos **errores son inevitables**. Es primordial entonces preocuparnos de ellos y **controlar su propagación**. Los **métodos numéricos** (o análisis numérico, o **cálculo numérico**) es una rama de la matemática que estudia técnicas y algoritmos para resolver problemas numéricamente manteniendo algún tipo de control sobre la propagación del error. Por ahora, nos bastará ver de dónde vienen algunos de estos errores, pero eso en la próxima clase.

Pregunta 3

Responde ahora, junto a tu grupo de trabajo, la tercera pregunta de la actividad.