Programación Científica aplicada a la Ingeniera

Buscando ceros

En la primera clase de Programación Científica debimos buscar las raíces del polinomio $\frac{1}{56}(x^5 - 12x^4 + 35x^3 + 25x^2 - 168x + 112)$. Estas raíces nos indicarían donde una población de bacterias, muy especiales, se mantendría constante. Con las herramientas de Octave que conocíamos, la solución pasaba por **graficar iterativamente** el polinomio hasta encontrar valores adecuados.

Pero este proceso es el mismo para buscar las raíces de cualquier polinomio, por ha sido automatizado en algunas herramientas en Octave. Por ejemplo, podemos encontrar las raíces del polinomio anterior como se muestra en el Ejemplo 1.

Ejemplo 1

```
> coefs = [1 -12 35 25 -168 112] / 56;
> roots(coefs)
ans =
   6.77980
   4.31772
  -2.06728
   2.07996
   0.88980
>
```

Listo, tenemos las cinco raíces del polinomio. La función `roots()` determina las raíces de un polinomio usando un método basado en los vectores propios de la matriz compañera del polinomio [1, 2]. Aunque el método es bastante robusto, suele ser lento y a veces entrega resultados equivocados. En el Ejemplo 2, vemos que la función no resuelve bien el polinomio $(x + 1)^7 = x^7 + 7x^6 + 21x^5 + 35x^4 + 35x^3 + 21x^2 + 7x + 1$.

Ejemplo 2

```
> coefs = [1 7 21 35 35 21 7 1];
> roots(coefs)
ans =
  -1.00898 + 0.00000i
  -1.00563 + 0.00701i
  -1.00563 - 0.00701i
  -0.99802 + 0.00880i
  -0.99802 - 0.00880i
  -0.99186 + 0.00393i
  -0.99186 - 0.00393i
>
```

Podemos ver que la función nos devuelve números complejos y no la raíz $x = -1$ con multiplicidad siete. Aunque existen métodos alternativos (como los métodos de Bairstow, Leguerre, de Newton-Hörner, de Müller, etc.), los métodos numéricos para el cálculo de las **raíces de un polinomio** son fuertemente afectados por errores de redondeo cuando hay raíces con **multiplicidad mayor que uno** [1, 3].

Pero Octave también provee herramientas más poderosas que no están limitadas a los polinomios. En la primera clase también vimos un ejemplo con visualización iterativa para determinar el valor x que cumple la relación $x = e^{-x}$. Eso también puede hacerse buscando dónde la expresión $x - e^{-x}$ es cero, con la función `fzero()`, como se muestra en el Ejemplo 3.

Ejemplo 3

```
> expresion = inline('x - e ** -x', 'x')
expresion =

f(x) = x - e ** -x

> fzero(expresion, [0, 1])
ans =  0.56714
>
```

En el Ejemplo 3, usamos la función `inline()` para declarar una expresión en función de la variable x. Luego usamos la función `fzero()` para que Octave busque **el valor donde la expresión se hace cero** en el intervalo [0, 1]. Esta función implementa un método numérico conocido como de Dekker-Brent [3], que combina varios métodos más clásicos (como el método de la bisección, que es bastante seguro, y el método de la secante, que es converge al cero más rápidamente), lo que lo hace bastante robusto.

Para que esta búsqueda tenga buenos resultados, es esencial elegir bien el **segundo parámetro**. Éste puede ser un valor escalar desde **donde comenzar la búsqueda** del cero, o un **intervalo que contiene el cero**.

El **primer parámetro** indica **la función** cuyos cero se está buscando, y puede ser el es el nombre de una función *inline*, como en el Ejemplo 3, el nombre de una función nativa, el nombre de una función creada por el usuario, e incluso el nombre de un archivo `.m` con el programa que implementa una función.

Consideremos el enunciado del Problema 1 para resumir el uso de esta función.

Problema 1

El Presidente de la República se encuentra preparando el discurso anual y no está seguro si deba mencionar la gestión que su gobierno ha hecho de los ahorros de la Nación. Sabe que el gobernante anterior invirtió 88.5 mil millones de dólares al inicio de su periodo, y que ahora, después de 7 años, esos ahorros valen 164.3 mil millones, mientras que en su periodo se invirtieron 56.8 mil millones de dólares que ahora, luego de dos años, valen 71.6 mil millones. El gabinete tiene la impresión que se está consiguiendo una mejor tasa de interés en este periodo presidencial que en el periodo anterior y que eso debería destacarse en el discurso como una muestra del mejor manejo económico del gobierno actual.

¿Recomienda a Ud. que el Presidente mencione esta gestión en el discurso?

Recuerde que si se invierte una cantidad de dinero v a una tasa de interés r, $0 < r < 1$, entonces en un año la inversión tendrá un valor de $v + r \cdot v$. Esta misma relación se aplica si la inversión se deja por más periodos, pero considerando que el valor de la inversión al final del año anterior es el valor inicial para el nuevo periodo.

Primero tenemos que darnos cuenta que el enunciado del Problema 1 nos presenta una relación recurrente: el valor de una inversión después de n años corresponde al valor de la inversión en el año anterior más el retorno que este valor genera a una tasa de interés r. Es decir:

$$v_n = \begin{cases} v_0 & \text{si } n = 0 \\ v_{n-1} + r \cdot v_{n-1} & \text{si } n > 0 \end{cases}$$

Podemos implementar esta relación recurrente como una **función recursiva** en Octave:

calculaValorInversion.m

```
# Define la funcion que calcula el valor de una inversion
# despues de n periodos a una tasa de interes compuesto r.
# Entrada: v0, valor inicial de la inversion
#          n, numero de periodos para la inversión
#          r, tasa de interés (0 < r < 1)
# Salida: vn, valor de la inversión después de los n periodos
function vn = calculaValorInversion(v0, n, r)
   if(n == 0)
      vn = v0;
   else
      vn = (1 + r) .* calculaValorInversion(v0, n - 1, r);
   end
end
```

Notemos que la función `calculaValorInversion()` utiliza el operador `.*` en vez del operador multiplicación `*`. De esta forma, la función sirve no sólo para valores escalares, sino que también para **parámetros que son vectores**. Esto es típico de la Programación Científica.

Analicemos la sintaxis para definir la función en Octave. Al igual que en Python, podemos identificar un **encabezado** y un **cuerpo** de la función. En encabezado comienza con la palabra reservada `function` (como en Python comenzaban con `def`) que es seguida por la **variable de retorno** a la que se le asigna (=) el valor que resulta de la evaluación de la función. El retorno es **implícito** en este caso: cuando la función **finaliza**, el **valor devuelto** corresponde al **almacenado** en la variable de retorno en ese instante; no se hace a través de una sentencia de retorno explícita. El resto del encabezado sigue la sintaxis que vimos en Python: el nombre de la función, y una lista de parámetros formales. La siguiente línea comienza el cuerpo de la función, que puede tener tantos comandos como sean necesarios, cuyo **final se marca explícitamente** con la palabra reservada `end` (a diferencia de Python, en que el final del cuerpo de un bloque se deduce de los cambios de indentación).

Pero para resolver el Problema 1 no podemos usar directamente esta función, ya que nuestra incógnita es **la tasa de interés** r y no el valor final de la inversión. Necesitamos entonces usar esta función en una que nos permita **buscar los valores de** r que llevan los valores finales de la inversión dados en el enunciado. Definamos entonces la función `diferencia1()` que calcule la diferencia entre el valor actual de la inversión del gobierno anterior y el valor que tendría esta inversión a una cierta tasa de interés r:

```
> source('C:\Users\usach\Desktop\calculaValorInversion.m')
>
> function tasa = diferencia1(r)
>    tasa = 164300000000 - calculaValorInversion(88500000000, 7, r);
> end
>
```

Ahora, la tasa buscada corresponde al valor r que hace cero la función `diferencia1()`. Evaluemos la función con diferentes valores para r para ver cómo se comporta. Para hacer más fácil la visualización de los resultados, podemos usar la función `printf()` para escribir salidas a pantalla con formato. El primer parámetro es un **string formato**, como los que usamos en Python, seguidos del **número necesario de valores** para llenar cada uno de los **marcadores** en el string formato. Los marcadores siguen la misma simbología que hemos visto, por ejemplo `%s` para strings, `%d` o `%i` para valores enteros y `%f` para valores no enteros. También podemos agregar **modificadores** para escribir un valor con un ancho determinado y generar columnas. Evaluemos entonces la función con valores para r entre cero y uno, recordando que `diferencia1()` funciona tanto con un valor escalar como con un valor vectorial para su parámetro.

```
> tasas = 0:0.1:1;
>
> difs = diferencia1(tasas);
>
> for i = 1:length(tasas)
>   printf('%.1f: %15i\n', tasas(i), difs(i))
> end
0.0:     75800000000
0.1:     -8161463350
0.2:   -152811500800
0.3:   -391024375450
0.4:   -768609510400
0.5:  -1347805468750
0.6:  -2211353785600
0.7:  -3467197256050
0.8:  -5253847283200
0.9:  -7746464890150
1.0: -11163700000000
>
```

Podemos ver que la función tiene un cero entre los valores 0.0 y 0.1. Usando esta información, podemos ahora pedir a Octave que busque el cero de la función:

```
> fzero('diferencia1', [0 0.1])
ans =  0.092408
>
```

Luego, la inversión del gobierno anterior obtuvo retornos a una tasa de interés anual de 9.2408%. Repitamos el procedimiento, esta vez para la inversión del gobierno actual:

```
> function tasa = diferencia2(r)
>    71600000000 - calculaValorInversion(56800000000, 2, r);
> end
> difs = diferencia2(tasas);
> for i = 1:length(tasas)
>   printf('%.1f: %15i\n', tasas(i), difs(i))
> end
0.0:     14800000000
0.1:      2872000000
0.2:    -10192000000
0.3:    -24392000000
0.4:    -39728000000
0.5:    -56200000000
0.6:    -73808000000
0.7:    -92552000000
0.8:   -112432000000
0.9:   -133448000000
1.0:   -155600000000
>
> fzero('diferencia2', [0.1 0.2])
ans =  0.12275
>
> printf('%i\n', calculaValorInversion(88500000000, 7, ans) - 164300000000)
34731156333
>
```

Es decir, el gobierno actual está obteniendo retornos con una tasa del 12.275% anual. El Presidente incluso podría jactarse que su gobierno, a estas alturas, podría haber obtenido 34.7 mil millones de dólares más para el país que su antecesor.

Pregunta 1

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

Resolviendo sistemas de ecuaciones

Octave también permite **resolver sistemas de ecuaciones numéricamente**. Esto es muy bueno puesto que **sistemas de ecuaciones lineales** aparecen en frecuentemente en todas las ramas de la ingeniería, incluyendo aplicaciones de estructuras estáticas, sólidos elásticos, transferencia de calor, difusión de gases y líquidos, campos electromagnéticos, circuitos eléctricos, etc. "Sistema de ecuaciones lineales" aquí significa un **conjunto de n ecuaciones de primer grado** definidas sobre un cuerpo algebraico cuya solución corresponde a los valores de las variables que satisfacen simultáneamente todas las ecuaciones. Es decir, estas aplicaciones generan ecuaciones de la forma:

$$\boldsymbol{Ax = b}$$

donde $\boldsymbol{A}$ corresponde a una **matriz cuadrada** de $n \times n$ de coeficientes que **caracteriza a un sistema**, $\boldsymbol{b}$ es un **vector columna dado** con la **entrada externa** al sistema, y $\boldsymbol{x}$ también es un **vector columna** con la solución desconocida que representa la **respuesta** del sistema. Esta notación compacta representa el sistema de ecuaciones lineales:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \equiv \begin{array}{c} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{array}$$

Usaremos el Problema 2 para ejemplificar cómo podemos usar Octave para resolver este tipo de sistemas de ecuaciones. Como siempre, existen **muchas formas de abordar el problema**. Tratemos primero de formularlo, aplicando abstracción de problemas, de forma conveniente para ser resuelto con Programación científica.

Llamaremos al vector $(x_1, x_2, x_3, x_4, x_5)$ a la **proporción** de barras de tipo A, B, C, D y E respectivamente que deberían usarse en la aleación solicitada. Así, el porcentaje de aluminio en la aleación sería: $30{,}0\ x_1 + 22{,}0\ x_2 + 29{,}0\ x_3 + 23{,}0\ x_4 + 34{,}0\ x_5$, mientras que el porcentaje de berilio sería $1{,}0\ x_1 + 2{,}0\ x_2 + 2{,}0\ x_3 + 1{,}7\ x_4 + 2{,}0\ x_5$. Con este mismo razonamiento, estas relaciones pueden obtenerse para los otros metales, generando el siguiente sistema de ecuaciones:

$$\begin{array}{rcrcrcrcrcr} 30{,}0\,x_1 & + & 22{,}0\,x_2 & + & 29{,}0\,x_3 & + & 23{,}0\,x_4 & + & 34{,}0\,x_5 & = & 30{,}68 \\ 1{,}0\,x_1 & + & 2{,}0\,x_2 & + & 2{,}0\,x_3 & + & 1{,}7\,x_4 & + & 2{,}0\,x_5 & = & 1{,}57 \\ \vdots & + & \vdots & + & \vdots & + & \vdots & + & \vdots & = & \vdots \\ 10{,}0\,x_1 & + & 8{,}0\,x_2 & + & 11{,}5\,x_3 & + & 9{,}0\,x_4 & + & 15{,}0\,x_5 & = & 11{,}73 \end{array}$$

Con notación compacta:

$$\begin{bmatrix} 30{,}0 & 22{,}0 & 29{,}0 & 23{,}0 & 34{,}0 \\ 1{,}0 & 2{,}0 & 2{,}0 & 1{,}7 & 2{,}0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 10{,}0 & 8{,}0 & 11{,}5 & 9{,}0 & 15{,}0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} 30{,}68 \\ 1{,}57 \\ \vdots \\ 11{,}73 \end{bmatrix}$$

Problema 2

A la fundición a llegado un pedido de 640 barras de una aleación especial para ser usadas en las vigas del techo de un estadio de fútbol. Cada barra mide 52.3 m y pesa 748.22 Kg, y su composición debe ser: 24,51% hierro, 4,67% carbono, 2,55% silicio, 8,21% cobre, 30,68% aluminio, 2,14% magnesio, 1,57% berilio, 2,87% niquel, 5,41% plomo, 5,66% estaño, 11,73% zinc.

Para responder al pedido, el "metalúrgico" tiene que encontrar la combinación ideal con el siguiente stock de barras estándares de la industria:

Metal	Barra A 130 Kg	Barra B 260 Kg	Barra C 130 Kg	Barra D 130 Kg	Barra E 390 Kg
Aluminio	30.0%	22.0%	29.0%	23.0%	34.0%
Berilio	1.0%	2.0%	2.0%	1.7%	2.0%
Carbono	1.0%	6.0%	7.0%	5.5%	8.0%
Cobre	15.0%	4.0%	5.5%	5.5%	0.0%
Cromo	0.0%	2.2%	0.0%	3.0%	0.0%
Estaño	5.0%	4.5%	5.5%	5.0%	7.0%
Hierro	30.0%	30.0%	23.0%	26.0%	17.0%
Magnesio	1.0%	2.0%	3.0%	1.5%	3.0%
Manganeso	0.0%	4.5%	0.0%	4.9%	0.0%
Molibdeno	0.0%	0.0%	0.0%	0.1%	0.0%
Niquel	1.0%	2.5%	4.5%	2.0%	4.0%
Oro	0.0%	0.1%	0.0%	0.1%	0.0%
Plata	0.0%	1.1%	0.0%	1.0%	0.0%
Platino	0.0%	0.1%	0.0%	0.2%	0.0%
Plomo	5.0%	4.5%	5.5%	5.5%	6.0%
Silicio	1.0%	4.5%	3.5%	4.0%	4.0%
Titanio	0.0%	2.0%	0.0%	2.0%	0.0%
Zinc	10.0%	8.0%	11.5%	9.0%	15.0%

La **solución** de este sistema de ecuaciones lineales nos daría la proporción que tendríamos que usar de cada barra para conseguir la aleación solicitada.

El programa `aleacion.m` muestra una forma de escribir este sistema en Octave. En este programa, los símbolos `...` al final las líneas indican al intérprete de Octave que el comando no ha finalizado y que continúa en la siguiente línea, y el operador `'` traspone una matriz o vector.

aleacion.m

```
propA = [30.0 1.0 1.0 15.0 0.0 5.0 30.0 1.0 0.0 0.0 1.0 0.0 0.0 ...
          0.0 5.0 1.0 0.0 10.0];
propB = [22.0 2.0 6.0  4.0 2.2 4.5 30.0 2.0 4.5 0.0 2.5 0.1 1.1 ...
          0.1 4.5 4.5 2.0  8.0];
propC = [29.0 2.0 7.0  5.5 0.0 5.5 23.0 3.0 0.0 0.0 4.5 0.0 0.0 ...
          0.0 5.5 3.5 0.0 11.5];
propD = [23.0 1.7 5.5  5.5 3.0 5.0 26.0 1.5 4.9 0.1 2.0 0.1 1.0 ...
          0.2 5.5 4.0 2.0  9.0];
propE = [34.0 2.0 8.0  0.0 0.0 7.0 17.0 3.0 0.0 0.0 4.0 0.0 0.0 ...
          0.0 6.0 4.0 0.0 15.0];

A = [porcA; porcB; porcC; porcD; porcE]';
b = [30.68 1.57 4.67 8.21 0.00 5.66 24.51 2.14 0.00 0.00 2.87 0.00 ...
     0.00 0.00 5.41 2.55 0.00 11.73];
```

¿Pero cómo resolvemos el sistema de ecuaciones? Si tuviéramos una ecuación normal, con valores escalares e incógnita x, sabríamos que $Ax = b \implies x = A/b$. En este caso es lo mismo, pero usamos división de matrices:

```
> source('C:\Users\usach\Desktop\aleacion.m')
>
> A \ b
ans =
  4.3000e-001
  -5.5511e-017
  3.2000e-001
  -1.8041e-016
  2.5000e-001
>
```

El **operador ** corresponde a esta división. Este operador es bastante complejo porque aplica muchos diferentes métodos para obtener la solución, dependiendo de las características de la matriz de coeficientes. Si nuestra matriz fuese cuadrada y densa, el operador usaría las típicas técnicas de factorización LU o QR que estudiamos en los cursos de Álgebra; si fuese dispersa, se usan métodos como la factorización de Cholesky o el método de eliminación Gaussiana, dependiendo si la matriz es triangular, o si tiene una diagonal positiva, o si es tridiagonal o si es simétrica o no, etc. Cuando todos los métodos fallan (porque la matriz es singular o casi singular), el operador intenta métodos iterativos que realizan una búsqueda de los valores de $\boldsymbol{x}$ que, por ejemplo, minimizan la suma los cuadrados de los residuos, es decir $\sum_i (A_i \cdot x - b_i)^2$ [2, 3]. Esto también ocurre

en nuestro ejemplo, puesto que nuestra matriz A **no es cuadrada** y por lo tanto no tiene una matriz inversa.

Notemos que la solución que obtenemos, a simple vista, no es muy satisfactoria. Si bien los porcentajes están cerca de la aleación especificada, estaría contaminada con pequeñas cantidades de metales no solicitados (como cromo, manganeso, oro, plata, platino y titanio). Es más, la solución indica que debemos **quitar** ciertas proporciones de barras de tipo B y D desde una mezcla hecha con barras de tipo A, B, C y E. Si bien esto tiene sentido en el **cuerpo de los reales**, no lo tiene en el mundo real.

Pero estas cantidades negativas son muy pequeñas, por lo que podríamos mejorar nuestra solución no usando barras de tipo B y D. En Octave deberíamos escribir:

```
> A = [porcA; porcC; porcE]';
>
> A \ b
ans =
   0.43000
   0.32000
   0.25000
>
```

Lo que nos entrega aleación solicitada. Dejemos como **ejercicio** terminar la solución, calculando los kilos que deben usarse de cada barra, y luego el número de barras que se necesitan.

Pregunta 2

Con tu grupo de trabajo, responde ahora la segunda pregunta de la actividad.

Pero Octave no se limita a los sistemas de ecuaciones lineales, y tiene muchas herramientas para resolver **sistemas de ecuaciones no lineales**, que también aparecen frecuentemente en las aplicaciones de ingeniería. Veamos, nuevamente a través de un ejemplo, cómo podemos resolver estos sistemas de ecuaciones. El Problema 3 nos presenta un desafío en el ámbito de la ingeniería química.

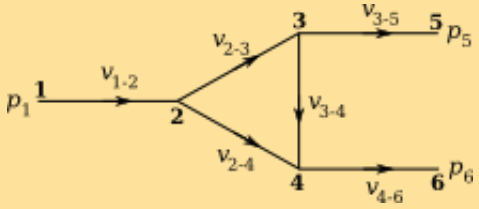
Estos tipos de problemas definen un **conjunto de funciones** no lineales, que debemos convertir en un sistema de ecuaciones no lineales. Para el Problema 3, se tiene inicialmente las siguientes:

$$p_1 - p_2 = \frac{2f\rho l_{12}}{d_{12}^2} v_{12}^2, \quad p_2 - p_3 = \frac{2f\rho l_{23}}{d_{23}^2} v_{23}^2, \quad p_2 - p_4 = \frac{2f\rho l_{24}}{d_{24}^2} v_{24}^2$$

$$p_3 - p_4 = \frac{2f\rho l_{34}}{d_{34}^2} v_{34}^2, \quad p_3 - p_5 = \frac{2f\rho l_{35}}{d_{35}^2} v_{35}^2, \quad p_4 - p_6 = \frac{2f\rho l_{46}}{d_{46}^2} v_{46}^2$$

Problema 3

El laboratorio está implementando una cadena de producción para un nuevo tipo de alcohol. En el proceso de destilación, el fluido pasa por la red de pipetas que muestra la figura.



De la mecánica de fluidos, sabemos que las caídas de presión en cada pipeta cumplen la relación:

$$p_i - p_j = \frac{2f\rho l_{ij}}{d_{ij}^2} v_{ij}^2$$

donde v_{ij}^2 es la velocidad del fluido en la pipeta; f es el factor de fricción, que en esta red se ha simplificado al valor constante 0,005; ρ es la densidad del líquido, en este caso 1.000 Kg m^{-3}; y l_{ij}, d_{ij} son respectivamente el largo y diámetro de la pipeta. La red tiene los siguientes largos y diámetros:

Pipeta:	**1-2**	**2-3**	**2-4**	**3-4**	**3-5**	**4-6**
Largo [m]	2,00	1,80	1,80	1,90	1,50	1,50
Diámetro [m]	0,10	0,08	0,08	0,10	0,09	0,09

Para ir mejorando la instalación, es necesario que se conozcan las velocidades del fluido al interior de cada pipeta y las presiones conseguidas en cada uno de las uniones, numeradas de 1 a 6. Se ha medido la presión a la entrada y la salida de la red: $p_1 = 500$ kPa, $p_5 = 50$ kPa y $p_6 = 100$ kPa. Obviamente, se debe mantener la masa del fluido a través de la red. Así, por ejemplo, la masa que sale de la pipeta 1-2 debe ser igual a la masa total que sale de las pipetas 2-3 y 2-4 (es decir $d_{12}^2 v_{12} = d_{23}^2 v_{23} + d_{24}^2 v_{24}$).

El enunciado del problema nos indica que p_1, p_5 y p_6 son conocidas, dejando **nueve incógnitas**: p_2, p_3, p_4, v_{12}, v_{23}, v_{24}, v_{34}, v_{35} y v_{46}.

Para que el sistema tenga una solución única, necesitamos al menos otras tres funciones que relacionen estas variables. La opción más obvia es usar las ecuaciones de continuidad que igualan las masas entrantes y salientes en cada pipeta (que el enunciado menciona al final). Estas ecuaciones serían:

$$d_{12}^2 v_{12} = d_{23}^2 v_{23} + d_{24}^2 v_{24}$$
$$d_{23}^2 v_{23} = d_{34}^2 v_{34} + d_{35}^2 v_{35}$$
$$d_{46}^2 v_{46} = d_{24}^2 v_{24} + d_{34}^2 v_{34}$$

Con estas ecuaciones completamos las nueve funciones que necesitamos. Para resolver el problema en Octave, igualamos cada ecuación a cero y la definimos como parte de una única función sobre las incógnitas del problema. El programa `flujo.m` muestra los comandos que definen esta función.

flujo.m

```
# Define una funcion que relaciona las variables de una
# red de pipetas que son desconocidas.
# Entrada: X, un vector con un valor inicial para las incognitas
#          X :=[p2 p3 p4 v_12 v_23 v_24 v_34 v_35 v_46]
# Salida: Y, vector con el valor actualizado para las incognitas
#         Y := [p2 p3 p4 v_12 v_23 v_24 v_34 v_35 v_46
function Y = buscaParametrosRed(X)
  P1 = 500;  # kPa
  P5 =  50;  # kPa
  P6 = 100;  # kPa
  friccion = 0.005;
  density = 1000;  # kg/m^3
  #            d_12 d_23 d_24 d_34 d_35 d_46
  diametros = [0.10 0.08 0.08 0.10 0.09 0.09];  # m
  #         l_12 l_23 l_24 l_34 l_35 l_46
  largos = [2.0 1.8  1.8  1.9  1.5 1.5];  # m

  Y(1) = 2 * friccion * density * largos(1) * X(4) ^ 2 / ...
            diametros(1) ^ 2 + X(1) - P1;   # 1-2
  Y(2) = 2 * friccion * density * largos(2) * X(5) ^ 2 / ...
            diametros(2) ^ 2 + X(2) - X(1); # 2-3
  Y(3) = 2 * friccion * density * largos(3) * X(6) ^ 2 / ...
            diametros(3) ^ 2 + X(3) - X(1); # 2-4
  Y(4) = 2 * friccion * density * largos(4) * X(7) ^ 2 / ...
            diametros(4) ^ 2 + X(3) - X(2); # 3-4
  Y(5) = 2 * friccion * density * largos(5) * X(8) ^ 2 / ...
            diametros(5) ^ 2 + P5   - X(2); # 3-5
  Y(6) = 2 * friccion * density * largos(6) * X(9) ^ 2 / ...
            diametros(6) ^ 2 + P6   - X(3); # 4-6

  Y(7) = diametros(1) ^ 2 * X(4) - diametros(2) ^ 2 * X(5) - ...
         diametros(3) ^ 2 * X(6);
  Y(8) = diametros(2) ^ 2 * X(5) - diametros(4) ^ 2 * X(7) - ...
         diametros(5) ^ 2 * X(8);
  Y(9) = diametros(3) ^ 2 * X(6) + diametros(4) ^ 2 * X(7) - ...
         diametros(6) ^ 2 * X(9);
end
```

Notemos que la función `buscaParametrosRed()` recibe como entrada un vector $X = [p_2\ p_3\ p_4\ v_{12}\ v_{23}\ v_{24}\ v_{34}\ v_{35}\ v_{46}]$ con las incógnitas. El programa usa los valores en este vector por medio del **operador de indexación**, con las posiciones que ocupa cada elemento. Así, p_2 es el valor `X(1)`, v_{12} es `X(4)`, etc. Notemos que en Octave los **índices de los elementos en un vector comienzan en uno** y no desde cero como en Python. Similarmente, la función devuelve un vector Y con las mismas incógnitas.

Para obtener una solución usamos el comando `fsolve()`, dando el **nombre de la función** que relaciona las incógnitas y un **valor inicial** para cada una de ellas. Es importante que este vector inicial sea una **estimación inteligente** de la verdadera solución. Después de todo `fsolve()` no hace magia (lo que es válido, en general, para todos los métodos de cálculo numérico). Para el Problema 3, y observando los valores que sí se conocen, podemos suponer que $X_0 = [150\ 150\ 100\ 1\ 1\ 1\ 1\ 1\ 1]$ no debería ser tan descabellado. En Octave escribimos:

```
> X0 = [150 150 100 1 1 1 1 1 1];
> Y = fsolve('buscaParametrosRed', X0);
>
> d_i = [1 2 2 3 3 4];
> d_j = [2 3 4 4 5 6];
> for i = 1:3
>   printf('p%d  = %10.4f\n', i, Y(i))
> end
> for i = 1:6
>   printf('v%d%d = %10.4f\n', d_i(i), d_j(i), Y(i + 3))
> end
p1  =   308.9116
p2  =   145.7554
p3  =   144.0406
v12 =     0.3091
v23 =     0.2409
v24 =     0.2421
v34 =    -0.0300
v35 =     0.2274
v46 =     0.1542
>
```

Podemos ver que obtenemos un vector solución con valores para cada incógnita. Notemos que aquí también obtenemos una solución negativa, pero a diferencia del Problema 1, esta solución sí tiene sentido en este caso: Octave nos está diciendo que el flujo en la pipeta que va del punto 3 al 4 tiene el sentido contrario al marcado en la figura del enunciado.

Pregunta 3

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