Notas de la clase de 13 abril, 2016

- Sobre el problema 2 de la tarea 9: aquí está una versión *no vectorizada* de `root_newton(n,a,E)`, similar a la función `root2(x)` que aparece en las notas de la clase de 6 de abril:

```
function x= root_newton(n,a,E)
   x=a;
   while abs(x^n-a)>E
      x=x+(x^n-a)/(n*x^(n-1));
   end
end
```

  Si queremos vectorizar esta función (en x) – por ejemplo, para poder graficarla facilmente – el reto es modificar la condición `abs(x^n-a)>E`. Aquí está una opción

```
function x=root_newton(n,a,E)
   x=a;
   while sum(abs(x.^n-a)>E)>0
      x=x-(x.^n-a)./(n*x.^(n-1));
   end
end
```

  De este modo, el ciclo `while` sigue corriendo hasta que *todas* las entradas $x(i)$ de x satisfacen que $|x(i)^n - a(i)| < E$.

- Otra manera de vectorizar `root_newton(n,a,E)` es

```
function x=root_newton(n,a,E)
   x=a;
   for i=1:length(x)
   while abs(x(i)^n-a(i))>E
      x(i)=x(i)-(x(i)^n-a(i))/(n*x(i)^(n-1));
   end
end
```

  Así cada $x(i)$ se determina por separado. Pero creo que la vectorización anterior es más eficiente (en Matlab código vectorizado no solo es más elegante, sino también ma's "eficiente", ie más rápido).

- Ya que tenemos el `root_newton(n,a,E)` vectorizado, es más fácil hacer las gráficas del problema 3:

```
x=0:.01:1;
E=10^-6;
hold on
for n=1:5
    plot(x,root_newton(n,x,E))
end
plot([0 1],[0 0],[0 0], [0 1]) % ejes de coordenadas
title('Graficas de la n-esima raiz de x usando el metodo de Newton')
xlabel('0<x<1');
ylabel('n-esima raiz de x');
legend('n=1','n=2', 'n=3', 'n=4', 'n=5','Location','southeast')
```

- Ecuaciones diferenciales. Consideramos la ecuación diferencial de *primer orden* $y'(x) = 2y(x)$ (una derivada), con la condición inicial $y(0) = 3$. Es fácil checar que $y(x) = 3e^{2x}$ es una solución (en cursos más avanzados se muestra que una ecuación diferencial de 1er orden con condición inicial tiene una única solución). Matlab puede resolver toda ecuación de la forma $y'(x) = F(x, y(x))$ con una condición inicial $y(x_0) = y_0$, numéricamente (ie sin dar fórmula explicita para $y(x)$). Lo ilustramos con $y' = 2y, y(0) = 3$, en el rango $0 \leq x \leq 10$. Primero se define la $F(x, y)$, en un archivo separado:

  ```
  function dy=F(x,y)
  dy=2*y;
  end
  ```

  y luego se usa en el comando `ode45` para resolver la ecuación $y' = 2y$ en el rango $[0, 10]$ y graficar la solución

  ```
  [t, y]=ode45(@F,[0 1],3);
  plot(t,y)
  ylim([0 50])
  ```

  Se puede hacer lo mismo tambien en un solo archivo

  ```
  F=@(x,y) 2*y
  [t, y]=ode45(F,[0 1],3);
  plot(t,y)
  ylim([0 50])
  ```

  Nota con cuidado el uso de `@` en estos dos ejemplos.
- Aquí está otro ejemplo. Resolvemos $y' = \sin(xy), y(0) = -1$, en en rango $0 \leq x \leq 1$.

  ```
  F=@(x,y) sin(x*y)
  [x, y]=ode45(F,[-5 5], 1);
  plot(x,y)
  ylim([0,5])
  ```

  A diferencia del ejemplo anterior, esta ecuación es imposible resolver explicitamente (como la mayoria de las ecuaciones diferenciales).
- Si la gráfica sale "poligonal" se puede agregarle más puntos

  ```
  F=@(x,y) sin(x*y)
  [x, y]=ode45(F,-5:.1:5, 1);
  plot(x,y)
  ylim([0,5])
  ```

- Ecuación de 2ndo orden. La ecuacion de Newton, $F = ma$ (fuerza=masa por acceleraccion), da una ecuación diferencial de 2ndo orden (dos derivadas). Por ejemplo, una masa $m = 1$ que se mueve a lo largo del eje de y bajo la influencia de una fuerza $F = -y$ (la ley de Hooke), satisface la ecuación $y''(t) = -y(t)$. La solución general es $y(t) = C_1 \cos t + C_2 \sin(t)$. Para determinar una solución tenemos que agregar dos condiciones iniciales. Por ejemplo, $y(0) = 2, y'(0) = 1$, da la solución $y = 2\cos t + \sin(t)$. Para resolver $y''(t) = -y(t)$, $y(0) = 2, y'(0) = 1$, en Matlab tenemos que convertirlo primero a un sistema de dos ecuaciones de 1er orden, agregando a y' como una función incognita nueva. O sea, definimos $y_1(t) = y(t)$, $y_2(t) = y'(t)$ y resolvemos el sistema

$$\begin{pmatrix} y_1' \\ y_2' \end{pmatrix} = \begin{pmatrix} y_2 \\ -y_1 \end{pmatrix}, \quad y_1(0) = 2, y_2(0) = 1.$$

  Esto se hace así:

```
F=@(t,y) [y(2);-y(1)]
[t, y]=ode45(F,[0 10*pi], [2 1]);
plot(t,y(:,1))
```

Nota que el valor de F tiene que ser una columna, por lo que usamos `[y(2);-y(1)]` en lugar de `[y(2), -y(1)]`.