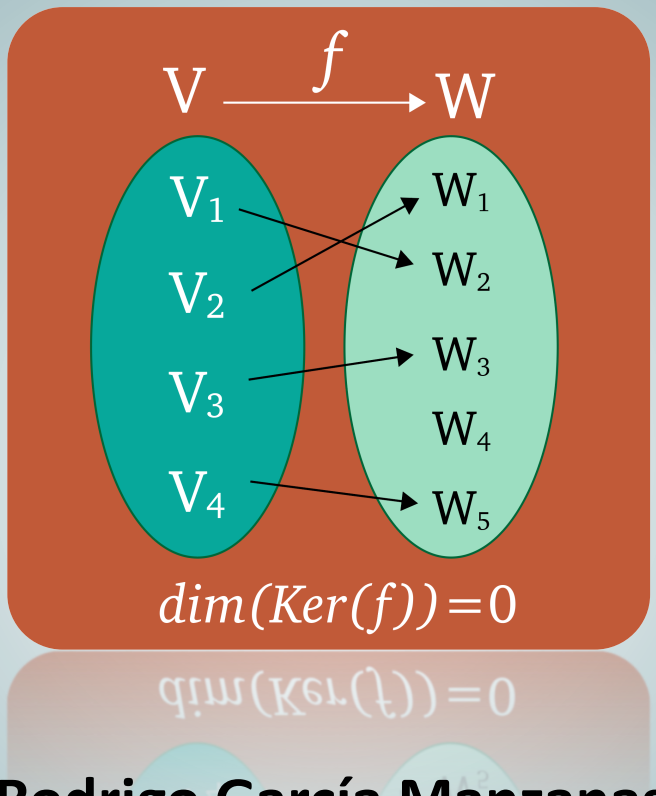
Álgebra

Práctica 9. Aproximación de funciones, resolución de sistemas incompatibles y ajuste a una nube de puntos



Rodrigo García Manzanas
Neila Campos González
Ana Casanueva Vicente

Departamento de Matemática Aplicada y Ciencias de la Computación

Grado en Ingeniería Química

G320: Álgebra

Práctica 9: Aproximación de funciones, resolución de sistemas incompatibles y ajuste a una nube de puntos

Rodrigo García Manzanas (rodrigo.manzanas@unican.es)

Objetivos

- Aproximar una función trascendente por un polinomio
- Resolución (aproximada) de sistemas incompatibles por mínimos cuadrados
- Ajuste a una nube de puntos

Aproximación de funciones

En ocasiones puede ser útil sustituir funciones trascendentes (trigonométricas, exponenciales, logaritmos) por polinomios, que son más manejables. Para ello, necesitaremos encontrar aquel polinomio que mejor se aproxime (más se parezca) a la función trascendente de la que nos interese "deshacernos". Si queremos utilizar un polinomio de grado *n*, la aproximación que buscamos será la proyección ortogonal de $c_1x^n + c_2x^{n-1} + ... + c_nx + c_{n+1}$ sobre el subespacio $\mathbb{P}_n$. Por tanto, antes de nada, necesitaremos una base ortogonal de $\mathbb{P}_n$. Para ello, podemos partir de la base canónica de este subespacio $(\{1, x, ..., x^n\})$ y aplicar el método de Gram-Schmidt.

Veamos cómo resolver en MATLAB el ejemplo de la página 21 de los apuntes, en el que se pide aproximar la función e^x por un polinomio de grado menor o igual a 2:

```
syms x real  % definimos como simbólica la variable x
f = exp(x);  % función a aproximar
```

Aplicamos Gram-Schmidt para hallar una base ortogonal de $\mathbb{P}_2$ partiendo de la canónica $(\{1, x, x^2\})$:

```
v1 = sym(1);  % primer "vector" de la base canónica.
% NOTA: A pesar de ser un número, en este caso tenemos que expresarlo como simbólico,
% pues la función que nos permite calcular integrales (int) sólo
% admite argumentos de entrada simbólicos
v2 = x;  % segundo "vector" de la base canónica
v3 = x^2;  % tercer "vector" de la base canónica
```

```
%% Construimos una base ortogonal mediante Gram-Schmidt
u1 = v1;  % primer vector de la nueva base ortogonal

proy_u1_v2 = u1*int(u1*v2, 0, 1)/int(u1*u1, 0, 1);  % proyección ortogonal de v2
% sobre u1
u2 = v2 - proy_u1_v2;  % segundo vector de la nueva base ortogonal

proy_u1_v3 = u1*int(u1*v3, 0, 1)/int(u1*u1, 0, 1);  % proyección ortogonal de v3
% sobre u1
proy_u2_v3 = u2*int(u2*v3, 0, 1)/int(u2*u2, 0, 1);  % proyección ortogonal de v3
% sobre u2
u3 = v3 - proy_u1_v3 - proy_u2_v3;  % tercer vector de la nueva base ortogonal

% Podemos comprobar que los tres vectores de la base de P2 que he
% obtenido {u1, u3, u3} son ortogonales entres sí:
int(u1*u2, 0, 1)  % u1 y u2 son ortogonales
```

```
ans = 0
```

```
int(u1*u3, 0, 1)  % u1 y u3 son ortogonales
```

```
ans = 0
```

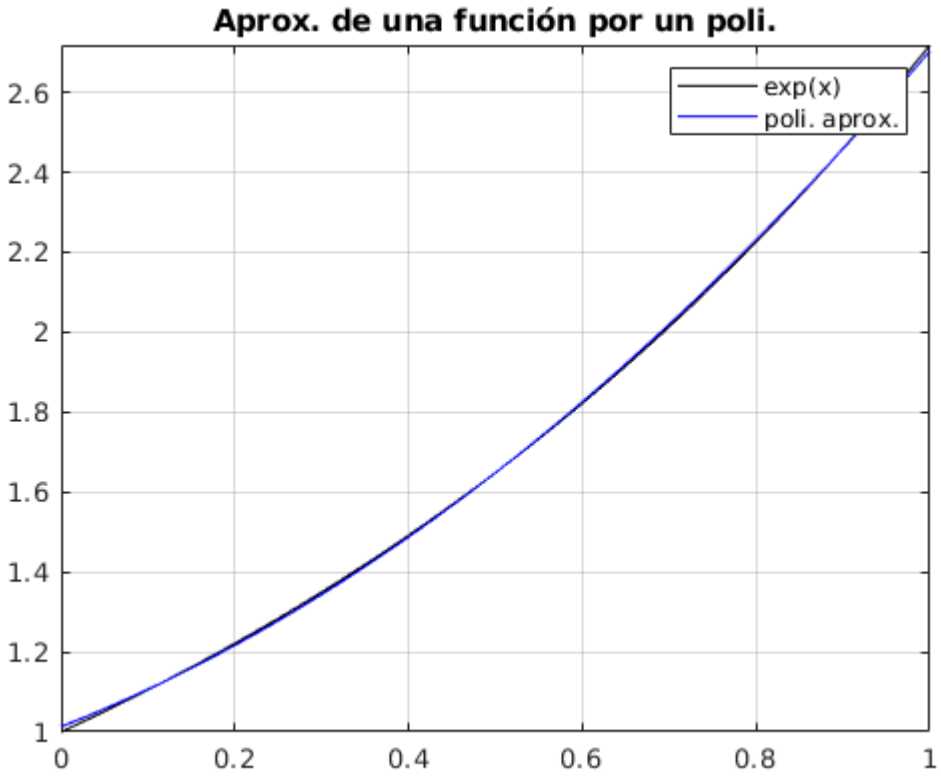
```
int(u2*u3, 0, 1)  % u2 y u3 son ortogonales
```

```
ans = 0
```

```
%% Ahora ya puedo aplicar la formula de la proyección
proy_u1_f = u1*int(u1*f, 0, 1)/int(u1*u1, 0, 1);  % proy. ortogonal de f sobre u1
proy_u2_f = u2*int(u2*f, 0, 1)/int(u2*u2, 0, 1);  % proy. ortogonal de f sobre u2
proy_u3_f = u3*int(u3*f, 0, 1)/int(u3*u3, 0, 1);  % proy. ortogonal de f sobre u3
proy_f = proy_u1_f + proy_u2_f + proy_u3_f;  % polinomio aproximador que buscaba
```

Para visualizar cuán buena es la aproximación que he obtenido, puedo dibujar ambas funciones en un mismo gráfico. Para ello usaremos la función *fplot*, que permite plotear (dibujar) funciones definidas como simbólicas:

```
figure  % abro una nueva ventana gráfica

fplot(f, [0, 1], 'k')  % dibujo f (la función a aproximar: exp(x))
% en el intervalo [0, 1], en color negro (argumento opcional 'k': blacK)
hold on  % orden para indicarle a MATLAB que quiero seguir dibujando en la
% misma ventana gráfica que está activa
fplot(proy_f, [0, 1], 'b')  % dibujo proy_f (nuestro polinomio aproximado)
% en el intervalo [0, 1], en color azul (argumento opcional 'b': Blue)
legend('exp(x)', 'poli. aprox.')  % añado una leyenda al gráfico.
% Hay que escribir los textos de la leyenda en el mismo orden en el
% que se han ido generando los gráficos
title('Aprox. de una función por un poli.')  % añado un título al gráfico
grid on  % añade una rejilla en el fondo (mejora la visualización)
```



Vemos en el gráfico que el polinomio que hemos hallado proporciona una aproximación muy buena a e^x en el intervalo $[0,1]$. Por ejemplo, veamos el valor que toman ambas funciones en el punto $x = 0.5$:

```
double(subs(f, 0.5))  % 1.6487
```

```
ans = 1.6487
```

```
double(subs(proy_f, 0.5))  % 1.6483
```

```
ans = 1.6483
```

Si quisiéramos dar una medida del error "total" que se comete con nuestra aproximación en todo el intervalo $[0,1]$ podríamos sumar (integrar) la porción de área que queda encerrada entre la curva negra y la azul. Normalmente, para evitar que errores positivos (*proy_f* > *f*) y negativos (*proy_f* < *f*) puedan cancelarse, lo que podría dar una idea errónea del error cometido, se suele calcular dicho error como $|proy\_f - f|^2$, al que se le denomina *error cuadrático*. En nuestro caso, sobre todo el intervalo $[0,1]$, tendríamos:

```
err = double(int((proy_f-f)^2, 0, 1))
```

```
err = 2.7835e-05
```

Vemos que el error cometido es despreciable.

Resolución (aproximada) de sistemas incompatibles por mínimos cuadrados

Imaginemos el siguiente sistema de ecuaciones lineales:

$$\begin{cases} x - y = 1 \\ y + z = 0 \\ x + y + 2z = -1 \end{cases}$$

Es fácil comprobar que es incompatible (no tiene solución):

```
A = [1 -1 0; 0 1 1; 1 1 2];  % matriz de coeficientes
b = [1 0 -1]';  % vector de términos independientes
Aamp = [A b];  % matriz ampliada
[rank(A) rank(Aamp)]  % S.I.
```

```
ans = 1×2
     2     3
```

Hemos visto que en estas circunstancias siempre es posible obtener una solución aproximada por mínimos cuadrados. Se trataría de sustituir el vector de términos indpendientes $\vec{b}$, para el cual no existe solución, por otro vector $\vec{c}$ para el cual el sistema sí tenga solución/soluciones. Este vector $\vec{c}$ será la proyección de $\vec{b}$ en el subespacio generado por las columnas de *A*. Por tanto, para encontrar nuestro $\vec{c}$ tendremos que ver en primer lugar si las columnas de *A* son L.I.:

```
[Ared, cp] = rref(A)  % en este caso, sólo la primera y segunda columnas de A
```

```
Ared = 3×3
     1     0     1
     0     1     1
     0     0     0
cp = 1×2
     1     2
```

```
% son L.I. La tercera es la suma de la primera y la segunda
```

Ahora ya estamos en condiciones de calcular la matriz de proyección sobre el subespacio generado por las columnas de *A*, que nos permitirá encontrar el vector $\vec{c}$ para el cual el sistema pasa a ser compatible.

```
P = A(:,cp)*inv(A(:,cp)'*A(:,cp))*A(:,cp)';  % matriz de proyección
c = P*b;  % vector para el cual el sistema pasa a ser compatible. Es
% la proyección de b sobre el subespacio generado por las columnas de A
[rank(A), rank([A c])]  % S.C.I.
```

```
ans = 1×2
     2     2
```

Nuestro nuevo sistema (el que resulta de sustituir $\vec{b}$ por $\vec{c}$) es compatible indeterminado (infinitas soluciones). Ya sabemos cómo resolver este tipo de sistemas:

```
syms x y z real
X = [x y z]';  % vector de incógnitas simbólicas
sol = solve(A*X == c, [x y]);  % resuelvo en las incógnitas principales (columnas
% pivotales de A)
sol_mc = [sol.x sol.y z]  % solución aproximada por mínimos cuadrados
```

```
sol_mc =
```

$$\left(-z \quad -z-\frac{2}{3} \quad z\right)$$

Siempre es recomendable comprobar que la solución que hemos hallado es correcta:

```
% Compruebo que así sea para un par de valores del parámetro z
A*subs(sol_mc, z, 0)' - c  % para z=0
```

```
ans =
```

$$\begin{pmatrix}0\\0\\0\end{pmatrix}$$

```
A*subs(sol_mc, z, 2.365)' - c   % para z=2.365
```

```
ans =
```

$$\begin{pmatrix}0\\0\\0\end{pmatrix}$$

Y en cualquier caso el error cuadrático cometido sería:

```
norm(c-b)^2  % error cuadrático
```

```
ans = 0.6667
```

Si las columnas de la matriz de coeficientes del sistema incompatible que trato de resolver son L.I., el proceso se simplifica, puesto que una vez que hallemos nuestro $\vec{c}$, nos quedará un S.C.D con una única solución. Por ejemplo, para resolver el sistema que se nos plantea en el problema 11, haríamos lo siguiente:

```
A = [3 1; 1 -1; 1 3];  % matriz de coeficientes
b = [4 3 7]';  % vector de términos independientes
[rank(A) rank([A b])]  % S.I.
```

```
ans = 1×2
     2     3
```

```
% Las columnas de A son L.I. (rg(A)=2), por lo que podremos
% calcular directamente la matriz de proyección sobre el subespacio
% generado por sus columnas
P = A*inv(A'*A)*A';  % matriz de proyección
c = P*b;  % proyección de b sobre el subespacio generado por las columnas de A
[rank(A), rank([A c])]  % S.C.D.
```

```
ans = 1×2
     2     2
```

Sabemos que podemos resolver los S.C.D.. con *linsolve*:

```
format rat
sol_mc = linsolve(A, c)  % solución única (de mínimos cuadrados)
```

```
sol_mc = 2×1
      11/8
      11/8
```

```
format short
A*sol_mc - c  % compruebo que la solución es correcta
```

```
ans = 3×1
10^-14 ×
   -0.2665
   -0.0888
   -0.0888
```

```
format rat
norm(c-b)^2  % calculo el error cuadrático cometido
```

```
ans =
      27/2
```

De hecho, una vez sabemos que las columnas de la matriz de coeficientes son L.I. no haría falta ni siquiera pasar por la matriz de proyección, puesto que sabemos que la solución de mínimos cuadrados puede calcularse directamente como (pag. 26 de los apuntes):

```
sol_mc = inv(A'*A)*A'*b  % solución de mínimos cuadrados
```

```
sol_mc = 2×1
      11/8
      11/8
```

Como último ejemplo, vamos a resolver el sistema que nos dan en la pág. 26 de los apuntes:

$$\begin{cases} -x+4y = 15 \\ y = 6 \\ -y = 4 \\ x-2y = -5 \end{cases}$$

```
A = [-1 4; 0 1; 0 -1; 1 -2];  % matriz de coeficientes
b = [15 6 4 -5]';  % vector de términos independientes
[rank(A) rank([A b])]  % S.I.
```

```
ans = 1×2
      2
```

```
% Como las columnas de A son L.I. (rg(A)=2), podemos calcular
% directamente la solución de mínimos cuadrados como:
sol_mc = inv(A'*A)*A'*b  % solución (única) de mínimos cuadrados
```

```
sol_mc = 2×1
      -1
       3
```

Acabamos de hallar que la solución de mínimos cuadrados es $x = -1, y = 3$. Esto quiere decir que el vector $\vec{c}$ se puede expresar como combinación lineal de las columnas de *A*, siendo los coeficientes de dicha C.L. -1 y 3. ¿Ves la relación entre este ejemplo y el problema 10?

Ajuste a una nube de puntos

Hemos visto que dada una nube de puntos, se puede ajustar por un tipo de función predefinida (por ejemplo una recta, una parábola, una exponencial...). Para ello habría que resolver el sistema que resulte de forzar a que nuestra función aproximadora/interpoladora pase por todos los puntos de la nube. Previsiblemente, este sistema será incompatible (no tendrá solución). Sin embargo, podremos buscar una solución aproximada por mínimos cuadrados. A modo de ejemplo, veamos cómo resolver el problema 12:

```
x0 = [2 5 7];  % coordenada x de los puntos de la nube
y0 = [6 7.9 8.5];  % coordenada y de los puntos de la nube
```

Para obtener un ajuste lineal (es decir, una recta $y = ax + b$), habrá que resolver el sistema

$$\begin{cases} 2a + b = 6 \\ 5a + b = 7.9 \\ 7a + b = 8.5 \end{cases}$$

en el que las incógnitas serán *a* y *b* (coeficientes de la recta de ajuste). Lo resolvemos:

```
A = [x0', ones(1, length(x0))'];  % matriz de coeficientes
b = y0';  % vector de términos independientes
[rank(A) rank([A b])]  % S.C.I.
```

```
ans = 1×2
     2
```

Las columnas de *A* son L.I. ($rg(A) = 2$), por lo que pudo calcular directamente la solución de mínimos cuadrados como:

```
sol_mc = inv(A'*A)*A'*b  % solución de mínimos cuadrados
```

```
sol_mc = 2×1
    97/190
   483/95
```

El primer (segundo) elemento de esta solución será el coeficiente *a* (*b*) de mi ajuste lineal. Por tanto, el polinomio aproximador/interpolador que busco será $y = 0.5105x + 5.0842$, es decir:

```
syms x real  % defino la variable independiente, x, como simbólica
pol1 = sum(sol_mc'.*[x 1])  % este es el polinomio aproximador
```

```
pol1 =
```

$$\frac{97\,x}{190}+\frac{483}{95}$$

```
% de grado 1 (recta) que busco
% NOTA: Para calcular pol1 he multiplicado primero sol_mc por [x, 1]
% elemento a elemento, y a continuación he sumado todos los elementos con sum
```

Si en lugar de un polinomio aproximador de grado1 (una recta), quisiera uno de grado 2 (una parábola $y = ax^2 + bx + c$), tendría que resolver el siguiente sistema:

$$\begin{cases} 2^2a + 2b + c &= 6 \\ 5^2a + 5b + c &= 7.9 \\ 7^2a + 7b + c &= 8.5 \end{cases}$$

Donde mis incógnitas serán *a*, *b* y *c* (coeficientes de la curva de ajuste). Lo resolvemos:

```
A = [x0.^2', x0', ones(1, length(x0))'];  % matriz de coeficientes (el
% simbolo ".^" se utiliza para realizar la potencia elemento a elemento)
b = y0';  % vector de términos independientes
[rank(A) rank([A b])]  % S.C.D.
```

```
ans = 1×2
     3
```

Vemos que en este caso, el sistema es compatible determinado (solución única). Esto ocurre porque tenemos únicamente tres puntos en nuestra nube, por lo que, el número de ecuaciones y el de incógnitas se igualan. *En general, si tenemos una nube con n puntos, siempre podremos encontrar un polinomio de grado* $n-1$ *que pase por todos ellos.* En nuestro caso, la parábola que buscamos vendrá dada por la solución (única) del sistema que hemos planteado.

```
sol = linsolve(A, b)  % S.C.D., podemos resolverlo con linsolve
```

```
sol = 3×1
    -1/15
    11/10
    61/15
```

```
syms x real  % defino la variable independiente, x, como simbólica
pol2 = sum(sol'.*[x^2 x 1])  % este el polinomio aproximador de
```

```
pol2 =
```

$$-\frac{x^2}{15}+\frac{11\,x}{10}+\frac{61}{15}$$

```
% grado 2 (parábola) que busco
```

Por último, si quisiéramos un ajuste exponencial del tipo $y = Ae^{Bx}$, podríamos tomar logaritmos a ambos lados de la expresión, quedándonos $ln(y) = ln(A) + Bx$, que se convierte en un polinomio de grado 1. Se trataría por tanto de resolver el siguiente sistema:

$$\begin{cases} 2B + ln(A) = ln(6) \\ 5B + ln(A) = ln(7.9) \\ 7B + ln(A) = ln(8.5) \end{cases}$$

Donde mis incógnitas serán *B* y $ln(A)$ (coeficientes de la curva de ajuste). Lo resolvemos:

```
A = [x0', ones(1, length(x0))'];  % matriz de coeficientes
b = log(y0');  % vector de términos independientes (en MATLAB, el
% logaritmo neperiano se calcula con la función log)
[rank(A) rank([A b])]  % S.C.I.
```

```
ans = 1×2
     2
```

Las columnas de *A* son L.I. ($rg(A) = 2$), por lo que pudo calcular directamente la solución de mínimos cuadrados como:

```
format short
sol_mc = inv(A'*A)*A'*b  % solución de mínimos cuadrados
```

```
sol_mc = 2×1
    0.0714
    1.6664
```

Por tanto, nuestra *B* sería 0.0714 y la *A* sería $e^{1.664}$. La función exponencial aproximadora que buscábamos será entonces:

```
B = sol_mc(1);
A = exp(sol_mc(2));
syms x real  % defino la variable independiente, x, como simbólica
fexp = A*exp(B*x)  % este la función exponencial aproximadora que busco
```

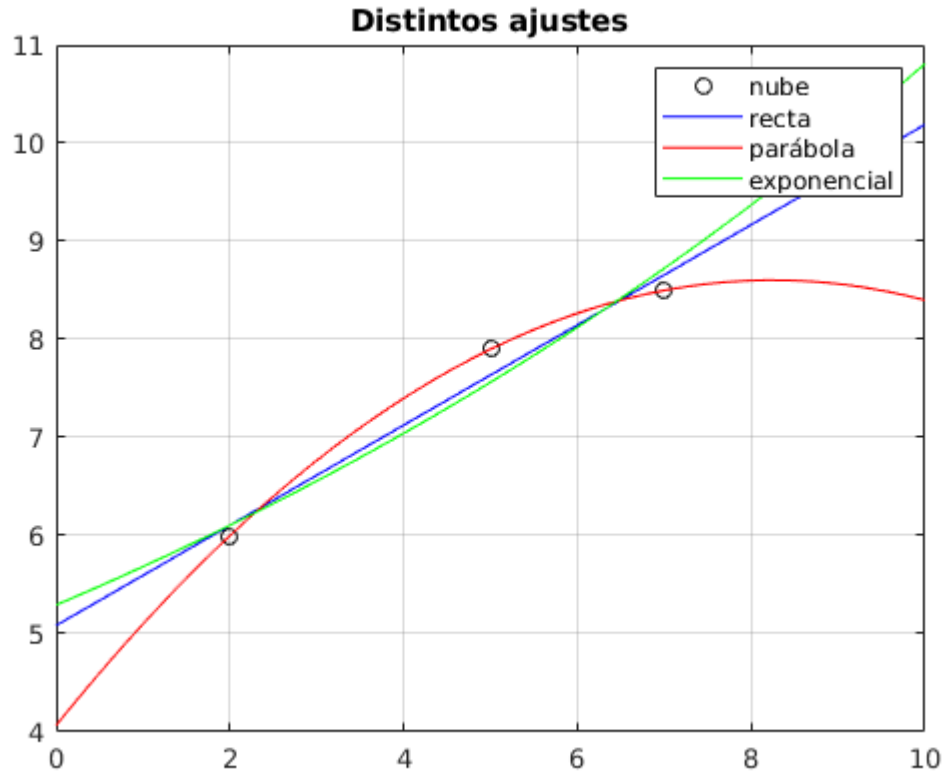
```
fexp =
```

$$\frac{2979609021702515\, e^{\frac{1286251827900453\, x}{18014398509481984}}}{562949953421312}$$

Una vez he obtenido los tres ajustes que buscaba, puedo representar todos ellos en un mismo gráfico junto a la nube de puntos inicial. Para representar gráficamente una nube de puntos se utiilza la función *plot.* En cambio, recuerda que para representar funciones cuya expresión es simbólica (como nuestro *pol1*, *pol2* y *fexp*) se utiliza la función *fplot*:

```
figure  % abro una nueva ventana gráfica

plot(x0, y0, 'ok')  % nube de puntos. El argumento opcional 'ob' indica
% que quiero representar círculos (o) negros (k: blacK)
```

```
hold on  % para seguir dibujando en la misma ventana gráfica
fplot(pol1, [0, 10] , 'color', 'b')  % recta, en azul (b: Blue)

hold on  % sigo dibujando en la misma ventana gráfica
fplot(pol2, [0, 10] , 'color', 'r')  % parábola, en rojo (r: Red)

hold on  % sigo dibujando en la misma ventana gráfica
fplot(fexp, [0, 10] , 'color', 'g')  % exponencial, en verde (g: Green)

legend('nube', 'recta', 'parábola', 'exponencial')  % añado una leyenda. Los
% textos tienen que ir en el mismo orden que las representaciones
% que he ido haciendo
title('Distintos ajustes')  % añado un título
grid on  % dibuja una rejilla en al fondo (facilita la visualización)
```



Podríamos calcular el error cuadrático cometido en cada una de las tres aproximaciones:

```
norm(y0 - double(subs(pol1, x, x0)))^2  % error cuadrático cometido con la recta
```

```
ans = 0.1053
```

```
norm(y0 - double(subs(pol2, x, x0)))^2  % error cuadrático cometido con la parábola
```

```
ans = 0
```

```
norm(y0 - double(subs(fexp, x, x0)))^2  % error cuadrático cometido con la función
```

```
ans = 0.1747
```

```
% exponencial
```

El valor aproximado de la intensidad para un voltaje de $7V$ sería, según los distintos ajustes que hemos obtenido, el siguiente:

```
double(subs(pol1, x, 7))  % intensidad estimada por la recta
```

```
ans = 8.6579
```

```
double(subs(pol2, x, 7))   % intensidad estimada por la parábola
```

```
ans = 8.5000
```

```
double(subs(fexp, x, 7))   % intensidad estimada por la función exponencial
```

```
ans = 8.7248
```

Para el caso de ajustes polinómicos, MATLAB dispone de la función *polyfit,* que devuelve directamente los coeficientes del polinomio interpolador. Podemos utilizarla para comprobar que el *pol1* y *pol2* que hemos obtenido son correctos:

```
% En la función polyfit, El tercer argumento de entrada hace referencia
% al grado del polinomio interpolador que queremos considerar
polyfit(x0, y0, 1)  % coeficientes de la recta de ajuste
```

```
ans = 1×2
    0.5105    5.0842
```

```
polyfit(x0, y0, 2)  % coeficientes de la parábola de ajuste
```

```
ans = 1×3
   -0.0667    1.1000    4.0667
```

Ejercicios propuestos

Ejercicio 1:

Aproxima la función $f(x) = sen(x)$ por otra del tipo $g(x) = ax + b$ (con *a* y *b* coeficientes $\in \mathbb{R}$) en el intervalo $[-\pi/2, \pi/2]$, considerando el producto escalar $f \cdot g = \int_{-\pi/2}^{\pi/2} f(x)g(x)dx$

Ejercicio 2

Comprueba que los siguientes sistemas son incompatibles y busca una solución de mínimos cuadrados. Calcula el error cuadrático cometido.

a) $\begin{cases} x-2y+4z &= -1 \\ x-y+z &= -4 \\ x &= -3 \\ x+y+z &= -2 \\ x+2y+4z &= 0 \end{cases}$ *b)* $\begin{cases} 3x+y+8z &= 4 \\ x-y+4z &= 3 \\ x+3y &= 7 \end{cases}$

Ejercicio 3

Dado el siguiente conjunto de puntos en el plano $\mathbb{R}^2$: $\{(-3,10),(-2,8),(-1,7),(0,6),(1,4),(2,5),(3,6)\}$.

- *a)* Busca, resolviendo los sistemas necesarios, los polinomios de orden 2, 4 y 6 que mejor se ajusten a la nube. Comprueba con *polyfit* que los ajustes que has obtenido son correctos
- *b)* Representa en un mismo gráfico la nube y los tres polinomios interpoladores
- *c)* Calcula el error cuadrático cometido con cada uno de los tres polinimos interpoladores

Ejercicio 4

Utiliza la función *polyfit* para encontrar un ajuste de tipo potencial ($y = ax^b$) a la siguientes datos: $\{(10,1.06),(20,1.33),(30,1.52),(40,1.68),(50,1.81),(60,1.91),(70,2.01),(80,2.11)\}$. Representa la nube y la curva de ajuste, y calcula el error cuadrático cometido.