Imprimir
Note: You can choose to save the module's content as a PDF document, instead of printing.
To do so, click on the 'Print' link above. In the window that appears, select the PDF
printer as the printer of choice, then click 'Print' and enter any additional information needed.

Interpolación (polinómica, splines, PCHIP...). Ajuste de curvas

Introducción

- **Interpolación** es el proceso de definir una función que tome unos valores específicos para unos puntos dados. Hay muchos métodos de interpolación pero nosotros vamos a centrarnos en la interpolación polinómica y la definida por intervalos, donde en cada intervalo usaremos funciones lineales o cúbicas.
- En muchos casos (experimentales) la cantidad de datos es muy grande y además llevan errores de medida. Cuando nos ocurre esto, puede ser mejor mejor **ajustar** una curva a los datos aunque no pase por todos ellos. Nosotros nos centraremos en el proceso de ajuste de curvas por **mínimos cuadrados**.

Interpolación

Supongamos dados n puntos en el plano, (x_k, y_k), $k = 1 \ldots, n$, en los que los x_k son distintos y para fijar ideas supondremos también que los puntos están ordenados por primera coordenada, es decir $x_k < x_{k+1}$ para todo k.

No queremos extendernos aquí con las fórmulas matemáticas ya que calcularemos las interpolaciones utilizando el asistente MATLAB. En cualquier caso, dichas fórmulas pueden consultarse por ejemplo en el capítulo de interpolación del libro de Moler y que se puede consultar aquí.

1) Interpolación polinómica

Puede demostrarse que existe un único polinomio $P(x)$ de grado estrictamente menor que n que pasa por todos los puntos dados, es decir, $P(x_k) = y_k$ para todo $k = 1, \ldots, k$. La interpolación polinómica consiste en calcular dicho polinomio.

2) Interpolación lineal a trozos

Consiste en considerar en cada intervalo $[x_k, x_{k+1}]$ la recta (función lineal) que une los puntos (x_k, y_k) y (x_{k+1}, y_{k+1}). A no ser que los puntos estén alineados obtendremos una poligonal.

3) Interpolación cúbica a trozos

Este es uno de los métodos de interpolación más usados en aplicaciones. Consiste en utilizar funciones cúbicas en cada intervalo, esto permite que queden varios grados de libertad a la hora de concretar las funciones. Estos grados de libertad son utilizados para que la curva de cada intervalo "pegue" bien con la del intervalo siguiente. Este concepto de que "pegue bien" debe entenderse matemáticamente imponiendo condiciones a las derivadas (primera y segunda) de las curvas en los extremos de los intervalos. De esta manera so obtienen curvas "suaves (smooth)".

3.1) Interpolación cúbica a trozos que preserva la forma (pchip)

Consiste en utilizar polinomios cúbicos de Hermite en cada intervalo. El acrónimo pchip viene de "piecewise cubic Hermite interpolating polynomial". La idea es considerar las pendientes en cada subintervalo de manera que no "sobrepasen" los valores de los datos; esto tiene un precio ya que se pierde la continuidad de la derivada en los extremos de los intervalos. La gran ventaja "preservan" la forma de los datos.

3.2) Interpolación cúbica a trozos por cerchas (splines)

Consiste en utilizar polinomios cúbicos en cada intervalo imponiendo la condición de que la derivada en los extremos sea continua. Esto hace que la curva interpolante sea muy "suave (smooth)". En contraposición con la interpolación anterior, los splines no suelen conservar la forma de los puntos originales.

4) Ilustración de los distintos métodos de interpolación

A continuación incluimos un vídeo ilustrando los distintos tipos de interpolación explicados. El vídeo es una grabación del escritorio de una sesión de MATLAB en la que se ha usado la aplicación **interpgui**. Dicha aplicación forma parte del texto de Moler "Numerical computing with Matlab" y puede descargarla si lo desea, instalarla en su sistema y hacer más pruebas. Para ello bájese el fichero comprimido ncm.zip con todas las aplicaciones matlab del libro de texto de Moler; descomprímalo en una carpeta y ponga dicha carpeta como la de trabajo de su sesión matlab. Una vez hecho, ejecute la aplicación interpgui en la línea de comandos.

Interpolando puntos con Matlab

Como ya debe saber, Matlab es un asistente matemático orientado al cálculo numérico y no al analítico. Por esta razón, la filosofía del programa cuando interpola es "Dado un conjunto de puntos del plano (x_k, y_k) y un conjunto de valores x_l, obtener las correspondientes imágenes y_l utilizando un método de interpolación concreto y que, por supuesto, interpole los puntos originales". A nivel de usuario, Matlab no muestra la función interpolante pero si da los correspondientes valores y_l. Dicha función se guarda y es usada por Matlab en un objeto llamado estructura (más o menos complejo dependiendo del método utilizado), y cuyo uso no vamos a abordar en este curso.

La orden interp1

Este comando tiene la sintaxis

```
Yq = interp1(X,Y,Xq,METHOD)
```

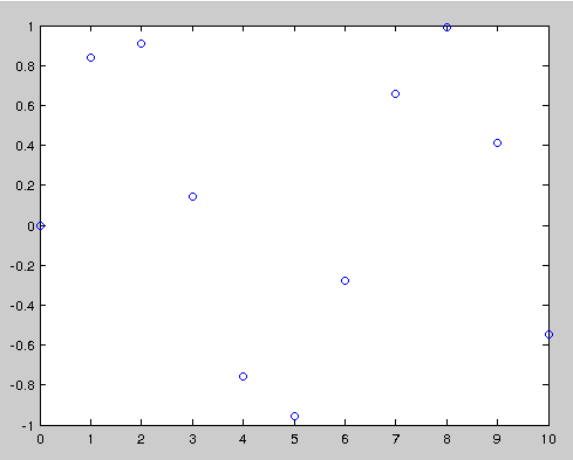
donde X e Y son, respectivamente, los vectores columna con los valores de las coordenadas x e y de los puntos dados. El vector Xq son las coordenadas x para los que queremos conocer su valor valor por interpolación, dichos valores son el output de la orden que quedan guardados en Yq. Por último METHOD es una cadena de texto que indica el método de interpolación a usar, siendo

'linear' - linear interpolation
'spline' - piecewise cubic spline interpolation (SPLINE)
'pchip' - shape-preserving piecewise cubic interpolation

Como puede observar no está la interpolación polinómica. Dicho polinomio lo obtendremos en la siguiente sección de ajuste de curvas ya que, en definitiva, consiste en ajustar un polinomio de grado $n-1$ a n puntos dados. Veamos un ejemplo de su uso.

Supongamos que tenemos los siguientes valores que definen 11 puntos

```
X=[0:10]'; Y=sin(X); plot(X,Y,'o');
```



Queremos utilizarlos para calcular las imágenes de los siguientes valores

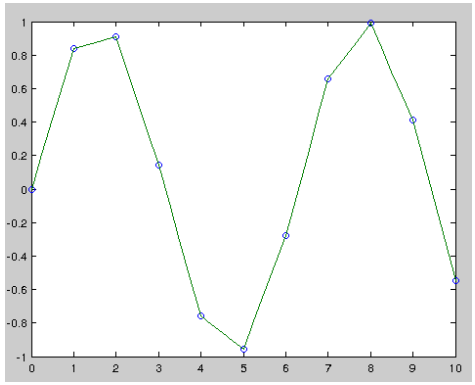
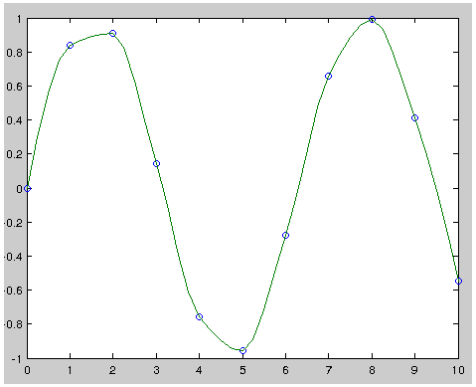
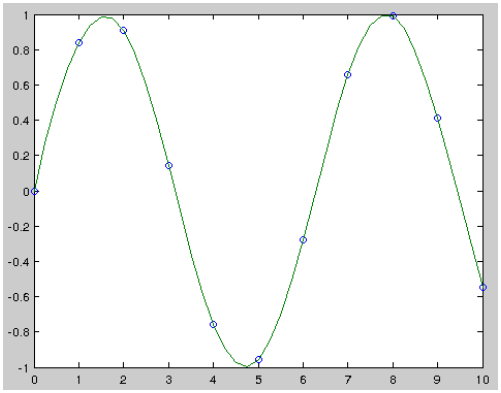
```
Xq=[0:0.25:10]';
```

Utilizaremos interpolación lineal y cúbica por trozos, con splines y también con polinomios cúbicos de Hermite. La orden es básicamente la misma cambiando la opción en la que se indica el método de interpolación.

```
YqLINEAL=interp1(X,Y,Xq,'linear');
YqPCHIP=interp1(X,Y,Xq,'pchip');
YqSPLINE=interp1(X,Y,Xq,'spline');
```

Podemos representar los datos dados y los interpolados conjuntamente en gráficos mediante las órdenes

```
plot(X,Y,'o',Xq,YqLINEAL);
plot(X,Y,'o',Xq,YqPCHIP);
plot(X,Y,'o',Xq,YqSPLINE);
```

Ajuste de curvas

Modelo de ajuste por mínimos cuadrados

Supongamos que se disponen de m observaciones que dependen de una variable independiente t, es decir, disponemos de m datos (t_i, y_i). Si el número de datos es grande o cuando los datos son medidas experimentales (con errores), la función interpolante que pasa por todos ellos calculada como se ha visto anteriormente suele ser de poca utilidad. En estos casos es habitual buscar una función o curva que seguramente no pase por los datos dados pero que esté "suficientemente" cerca de ellos. El término "suficientemente cerca" puede tener distintas interpretaciones y nosotros solamente estudiaremos un método conocido como ajuste por **mínimos cuadarados (least squares)**.

Construyamos pues el modelo de ajuste: Disponemos de las observaciones

$$y_i = y(t_i), i = 1, \ldots, m$$

Buscamos una función $y(x)$ que sea una combinación lineal de n funciones básicas (lineamente independientes), es decir

$$y(t) = b_1 f_1(t) + \cdots + b_n f_n(t)$$

donde los coeficientes b_i deben ser determinados. La situación ideal es que dicha función pasara por todos los puntos dados, lo que implicaría que el sistema de m ecuaciones con n incógnitas

$$Xb = y$$

sería compatible; donde B es el vector columna de coeficientes b_i, y es el vector columna de datos y_i y X es la matriz en la que

$$x_{i,j} = f_j(t_i)$$

Sabemos cómo resolver sistemas lineales en Matlab, luego

```
b=X\y
```

nos dará los valores b_i que nos servirán para obtener la curva buscada $y(t)$. Es habitual que el sistema tenga más ecuaciones que incógnitas ya que cada ecuación proviene de una observación. Este hecho, junto con la seguridad de que las observaciones experimentales siempre contienen errores nos lleva a que el sistema que hay que resolver sea incompatible. En dicho caso, la orden anterior nos cálcula "**la mejor solución**" en el sentido de los mínimos cuadrados. Esto es, minimiza la suma los errores cuadráticos entre el dato observado y el valor calculado mediante $y(t)$.

Este es el modelo general, los ajustes por modelos particulares son aquellos en los que se desea que $y(t)$ tenga una forma concreta. Por ejemplo

- **Ajuste lineal (recta de regresión):** La función buscada es lineal, es decir, una recta con expresión
$$y(t) = b_1 + b_2 t$$
- **Ajuste polinómico de grado n:** La función buscada es un polinomio de grado n, es decir,
$$y(t) = b_1 + b_2 t^2 + \cdots + b_n t^n$$
- Etc

Observe que las funciones pueden ser lineales o no pero que las incógnitas buscadas aparecen linealmente en $y(t)$. Cuando las funciones también involucran incógnitas de manera no lineal el problema es mucho más complicado y se resuelve mediante un proceso híbrido de mínimos cuadrados y de minimización de funciones no lineales que pueden tener restricciones o no. No estudiaremos estas situaciones que tienen una complejidad mayor.

Ejemplos

Se disponen de los datos de población (en millones de personas) de los Estados Unidos de América del siglo pasado por décadas. Los datos son

```
t=[1900:10:2000]';
y=[75.995 91.972 105.711 123.203 131.669 150.697 179.323 203.212 226.505 249.633 281.422]';
```

1) Ajuste de los datos mediante un modelo lineal $y(t) = b_1 + b_2 t$

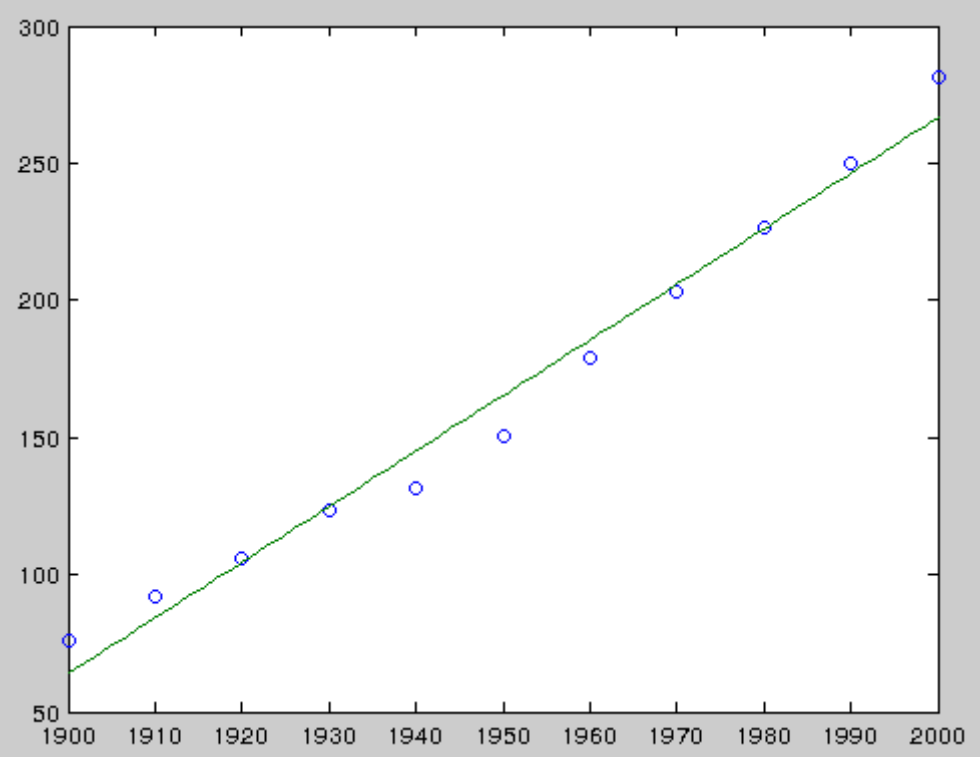
```
X=[t.^0 t.^1];
b=X\y;
```

Si inspeccionamos los valores de b observaremos que la recta buscada es

\[y(t) = - 3783,9 + 2 t \]

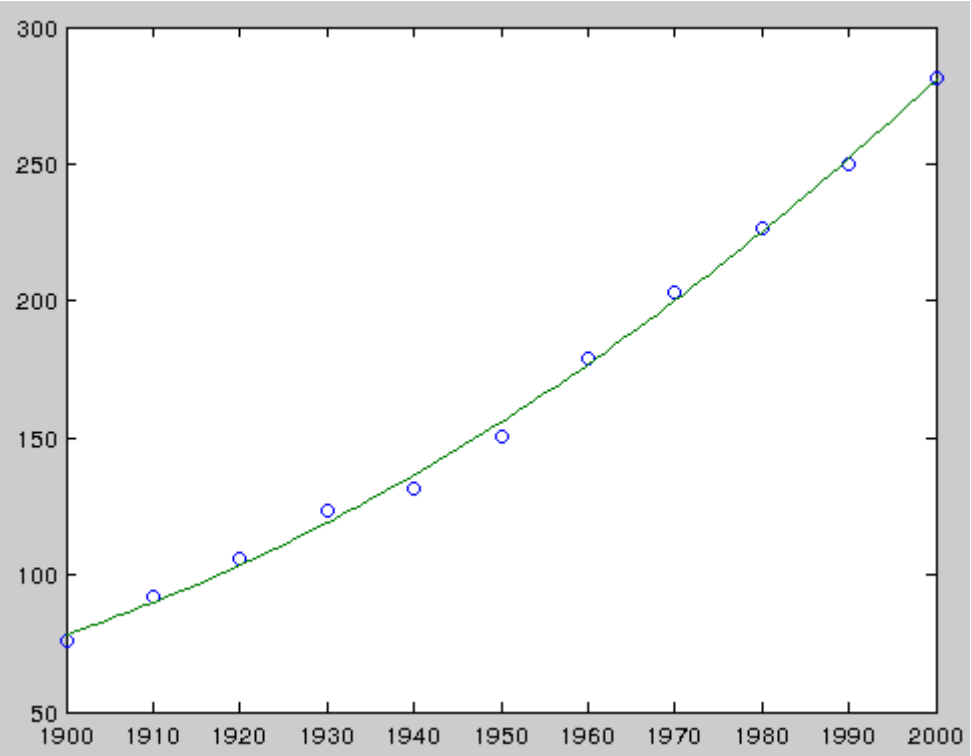
Si queremos representar dicha función y los datos originales no hay más que construir una tabla de valores. Lo haremos en una sola línea de manera muy compacta. A estas alturas no debe suponerle ningún problema la interpretación de dicha orden. Si no la entiende represente la función como se mostró en temas anteriores.

```
tq=[1900:1:2000]'; yq=(b'*[tq.^0 tq.^1]')'; plot(t,y,'o',tq,yq);
```



2) Ajuste los datos mediante un modelo cuadrático (parábola de regresión), es decir, $y(t) = b_1 + b_2 t + b_3 t^2$. No explicaremos los pasos ya que son parfecidos a los anteriores.

```
X=[t.^0 t.^1 t.^2];
b=X\y;
tq=[1900:1:2000]'; yq=(b'*[tq.^0 tq.^1 tq.^2]')'; plot(t,y,'o',tq,yq);
```



3) **Ejercicio)** Ajuste los datos mediante un modelo cúbico, es decir, un polinomio de grado 3. Cuando lo haga obtendrá un mensaje de error debido a que las potencias de números muy grandes deben ser pasadas a números de coma flotante y por el redondeo se obtiene una matriz con algunas columnas linealmente dependientes. Esto se puede arreglar escalando los datos (vea el capítulo del libro de Moler) y también puede usar la orden **polyfit** que mostraremos más adelante.

4) Ajuste **Log-linear**. Supongamos que queremos ajustar la curva

$$y(t) = K\exp(\lambda t)$$

a los datos. Tomando logaritmos tendremos que

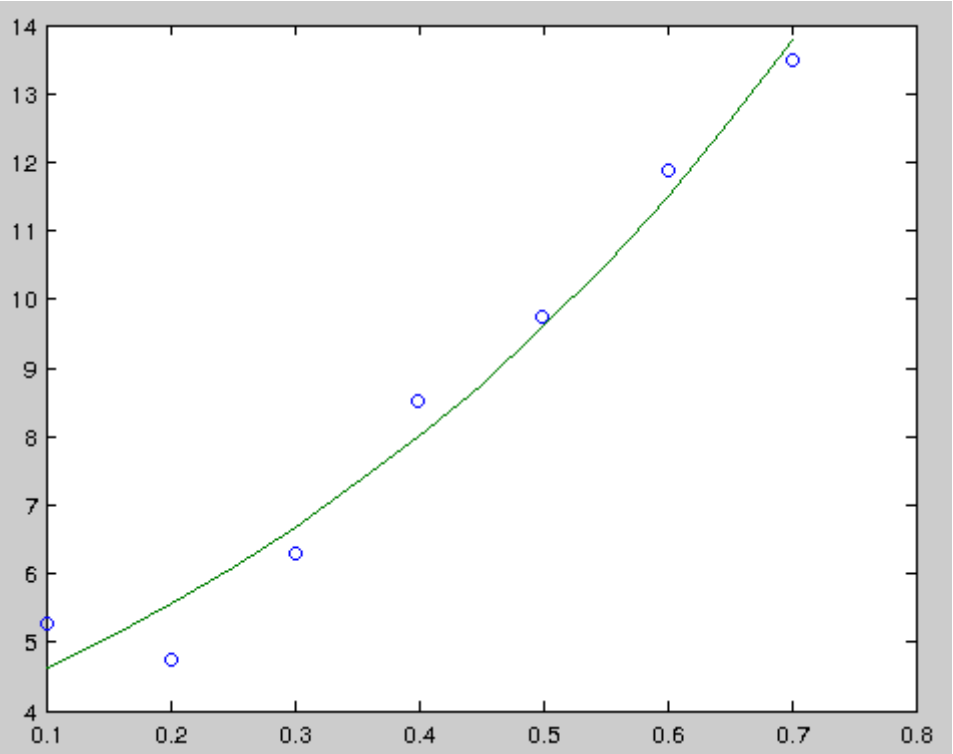
$$\log y(t) = \log K + \lambda t$$

. Llamando $b_1 = \log K, b_2 = \lambda$, tendremos que resolver el sistema

$$\log y(t) = Xb$$

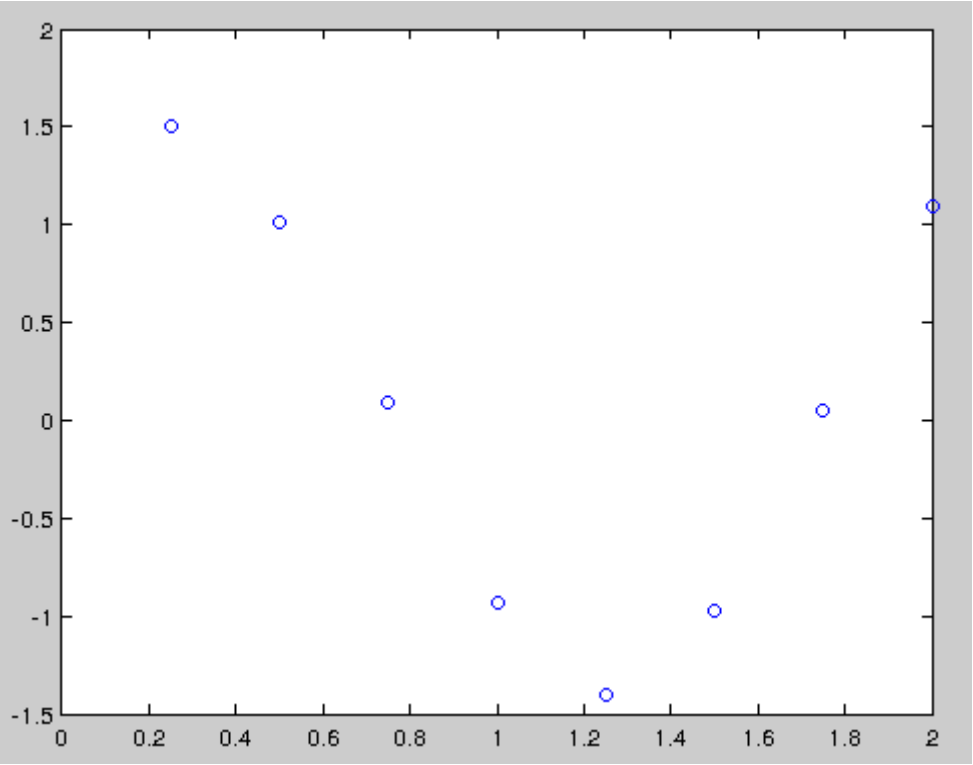
y posteriormente deshacer el logaritmo tomando exponenciales. Veamos un ejemplo.

```
t=[0.1:0.1:0.7]';
y=[5.2648 4.7592 6.3099 8.5081 9.7393 11.8793 13.4771]';
X=[t.^0 t.^1];
b=X\log(y);
tq=[0.1:0.05:0.7]'; yq=exp(b'*[tq.^0 tq.^1]')'; plot(t,y,'o',tq,yq);
```



5) **Ejercicio)** Ajuste los datos

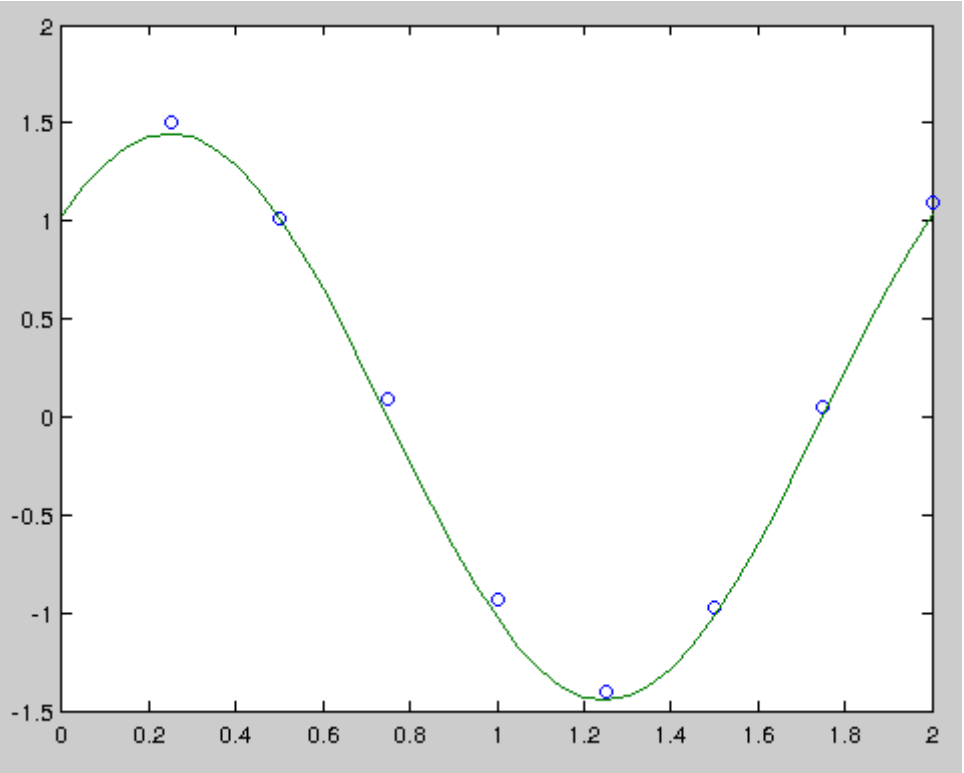
```
t=[0:0.25:2]';
y=[1.0815 1.5048 1.0127 0.0913 -0.9368 -1.4045 -0.9722 0.0547 1.0958]';
plot(t,y,'o');
```



mediante la función

$$y(t) = b_1 \sin \pi t + b_2 \cos \pi t$$

Cuando dibuje la curva de ajuste y los puntos debería obtener un gráfico similar al siguiente.



La orden polyfit de Matlab para ajustar e interpolar polinómicamente

---

Ajustes

En el caso en el que deseemos ajustar puntos por funciones polinómicas, podemos utiizar la orden de MatLab **polyfit.** Esta orden toma tiene la sintasis

```
P = polyfit(X,Y,N)
```

donde X e Y son los vectores columna de las primeras y segundas coordenadas de los puntos a interpolar y N es el grado del polinomio de interpolación que se debe usar. El resultado es un vector que contiene los coeficientes del polinomio de ajuste en orden descendiente de potencias. Dicho vector de coeficientes puede ser utilizado para evaluar polinomios en puntos concretos por medio de la orden **polyval** y cuya sintaxis es

```
Yq = polyval(P,Xq)
```

donde Xq es un vector con los puntos donde se quiere evaluar el polinomio y P es el vector de coeficientes que hemos obtenido antes.

Veamos el ejemplo de la población de Estados Unidos del siglo pasado utilizando esta orden.

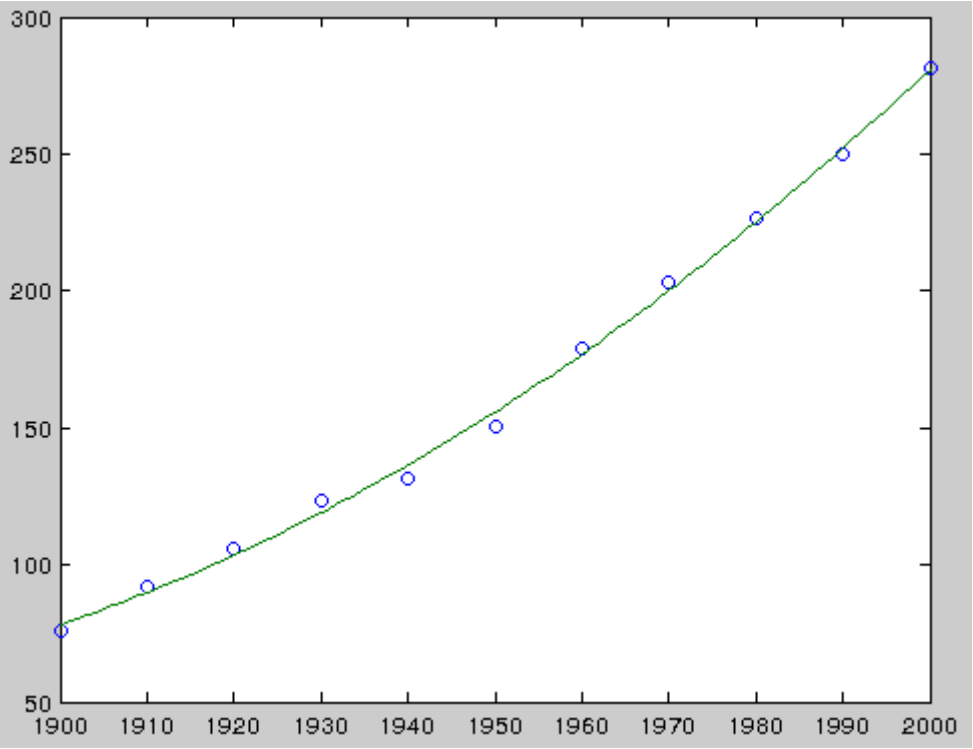
```
t=[1900:10:2000]';
y=[75.995 91.972 105.711 123.203 131.669 150.697 179.323 203.212 226.505 249.633 281.422]';
P=polyfit(t,y,2);
Warning: Polynomial is badly conditioned. Add points with distinct X
         values, reduce the degree of the polynomial, or try centering
         and scaling as described in HELP POLYFIT.
 In polyfit at 76
```

Observe que se produce un mensaje de error debido a que los valores de la variable independiente t son excesivamente grandes. Aún así Matlab nos calcula los coeficientes obteniendo los valores

```
1.0e+04*(0.0000 -0.0035 3.2294)
```

Representar gráficamente el polinomio es bien sencillo.

```
tq=[1900:1:2000]'; yq=polyval(P,tq); plot(t,y,'o',tq,yq);
```



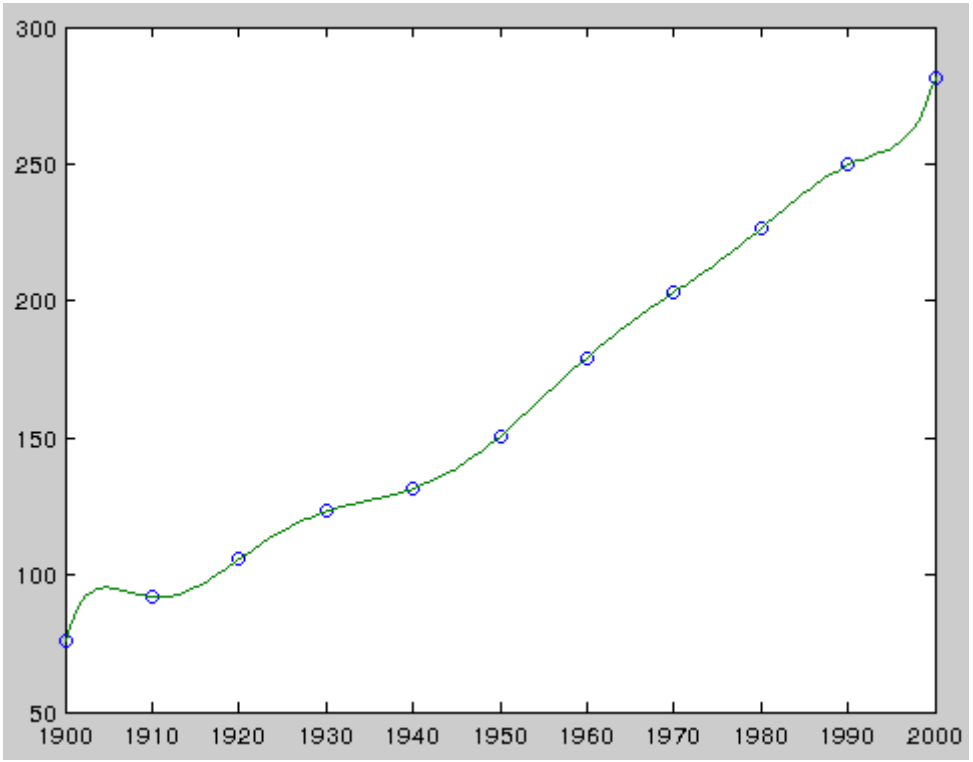
Interpolación polinómica

Observará que todavía no hemos dado un método para calcular el polinomio de interpolación que pasa por un conjunto de puntos dados. Ahora es sencillo si tenemos en cuenta que consiste en ajustar un polinomio de exactamente un grado menos que número de datos tenemos. Hagamos el mismo ejemplo anterior pero escalando los datos porque al tratarse de 11 datos, tendremos que usar un polinomio de grado 10 y elevar a la potencia décima números del orden de 2000 nos producirá grandes errores. Se impone una transformación de los datos mediante una traslación y una homotecia.

```
t=[1900:10:2000]';
y=[75.995 91.972 105.711 123.203 131.669 150.697 179.323 203.212 226.505 249.633 281.422]';
s=(t-1950)/50;
```

Observe que ahora los datos de la variable s han quedado en el intervalo $[-1,1]$. El resto es sencillo pero para representar los datos de manera conjunta debemos deshacer la transformación, es decir, $t=50s+1950$. También hemos automatizado el proceso usando `length(s)-1` para indicar el grado del polinomio de interpolación (número de datos-1).

```
P=polyfit(s,y,length(s)-1);
sq=[-1:0.01:1]'; yq=polyval(P,sq); plot(t,y,'o',50*sq+1950,yq);
```



Actividades

Dados los datos

```
t = (0:.1:2)';
y = [5.8955 3.5639 2.5173 1.9790 1.8990 1.3938 1.1359 ...
       1.0096 1.0343 0.8435 0.6856 0.6100 0.5392 0.3946 ...
       0.3903 0.5474 0.3459 0.1370 0.2211 0.1704 0.2636]';
```

Se pide que haga los gráficos que muestren:

1. Interpolación lineal de los datos
2. Interpolación a trozos con polinomios cúbicos de Hermite
3. Interpolación a trozos con polinomios splines cúbicos
4. Interpolación polinómica
5. Ajuste lineal de los datos

6. Ajuste cuadrático de los datos
7. Ajuste log-lineal de los datos

Observe los resultados e indique de manera justificada cuál (o cuáles) de los 7 ejercicios propuestos cree usted que no tiene sentido utilizar para cálculos posteriores.

Bibliografía

Los capítulos 3 y 5 del libro de Moler. Dichos capítulos disponen de licencia libre para fines educativos por lo que puede descargarlos si quiere.

Cleve B. Moler, Numerical Computing with MATLAB.

Capítulo 3 Interpolación

Capítulo 5 Ajuste de curvas

Al final de cada capítulo también hay una bibliografía recomendada que se puede consultar si se tiene alguna duda o se desea ampliar algún tema.

Como siempre la consulta a Wikipedia es conveniente, sobre todo en su versión inglesa ya que tiene los temas más desarrollados.