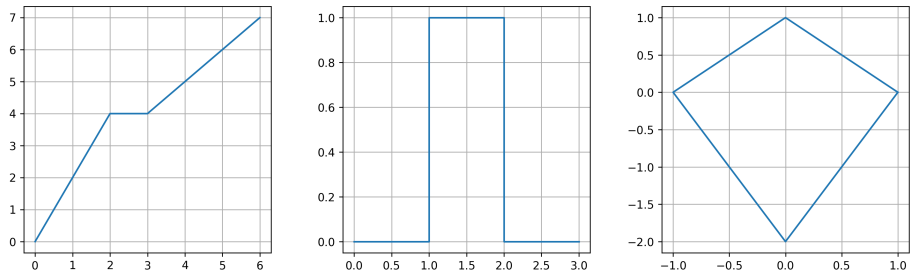
Ejercicios de Python — Capítulo 3

Curu y Fede

martes 6 octubre, 2020

1. Gráficos

1. Generar las figuras indicadas a continuación.



2. Generar, en cada caso, las gráficas de las funciones $f(x)$ indicadas a continuación, en el dominio $D_f = [0, 1]$.

   a) $f(x) = 3x^2 + 5$
   b) $f(x) = 2^x - 1$
   c) $f(x) = \text{sen}(4\pi x)$
   d) $f(x) = x^5 - x^4 + x^3 - x^2 - 2x + 2$

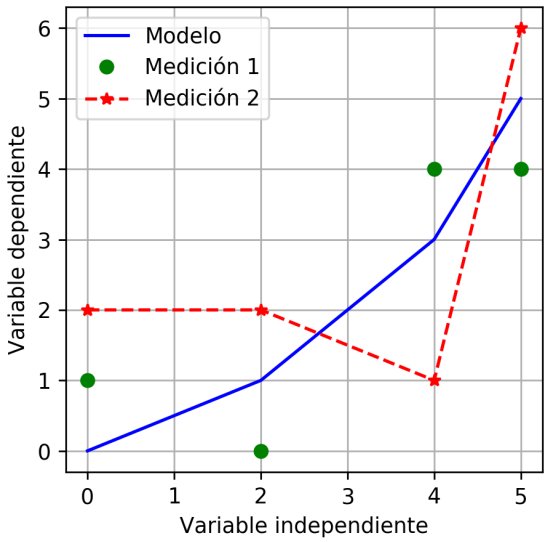
3. Graficar las curvas paramétricas dadas a continuación.

   a) $x(t) = t^3, y(t) = t^2$
   b) $x(t) = \text{sen}(\pi t), y(t) = \text{sen}(2\pi t)$
   c) $x(t) = \Big(\text{sen}(\pi t)\Big)^3, y(t) = \Big(1 - \cos(\pi t)\Big) \cdot \cos(\pi t)$

   Tomar, en todos los casos, $-1 \leq t \leq 1$.

   Ayudita: no importa cómo uno genere sus vectores `x` e `y`, la función siempre se ejecuta con ellos como argumento, esto es, `plt.plot(x,y)`

4. Generar la siguiente figura.



5. Implementar la función `stereo_plot(audio)` que grafica, en dos subgráficas diferentes (`subplots`), ubicadas en una matriz de 1×2, los canales izquierdo y derecho de un audio estéreo, dado como matriz de dos columnas. Observar que esta función sólo debe hacer el gráfico, o sea, no es necesario que *devuelva* nada. El eje x debe correponder al tiempo en segundos de la muestra. En el gráfico izquierdo, el tiempo debe estar dado en segundos, en el gráfico derecho, en minutos. Los dos gráficos deben contener:

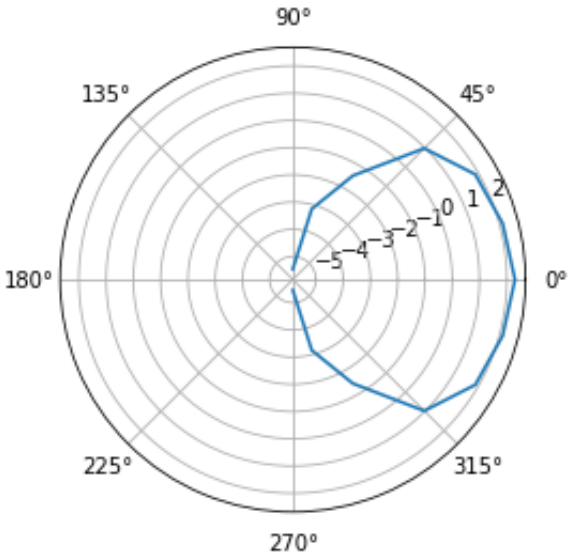
   a) Nombre de los dos ejes

   b) Título

   c) Una leyenda indicando qué es el conjunto graficado con el marcador que se haya elegido

   d) Grilla

6. Existen muchos tipos de gráficos contenidos en `matplotlib` además de la gráfica de un vector contra otro. Un gráfico de interés es el llamado **gráfico polar**, que grafica una amplitud por cada valor de un ángulo, según se ve en la figura:



Este gráfico puede realizarse utilizando la función `plt.polar`, que recibe un vector / lista con ángulos **en radianes** y un vector / lista con amplitudes para esos ángulos.

Para este problema, se suministran seis archivos en formato `.txt` (ver el archivo comprimido `Directividad.7z`), que contienen, para cada frecuencia de interés, la amplitud registrada para una señal emitida por un parlante rotado $0, 15, 30, 45, 60, 75$ o $90°$. Cada archivo corresponde a un ángulo de rotación. Se pide generar el gráfico polar mostrado como ejemplo, a partir de la amplitud registrada para cada rotación en la frecuencia de 1000 Hz. El procedimiento sugerido es:

a) Levantar la data de cada archivo con la función `np.loadtxt`, que recibe cada archivo y lo convierte en una matriz. Notar que por defecto sólo recibirá números, y los títulos de las columnas no lo son. El argumento opcional `skiprows` puede ser de utilidad. A veces estas mediciones se hacen con pasos de hasta 0.5 grados, con lo cual sería deseable no escribir el nombre de cada archivo para levantarlo. Notar que los archivos siguen un estándar en cuanto a cómo se nombraron.

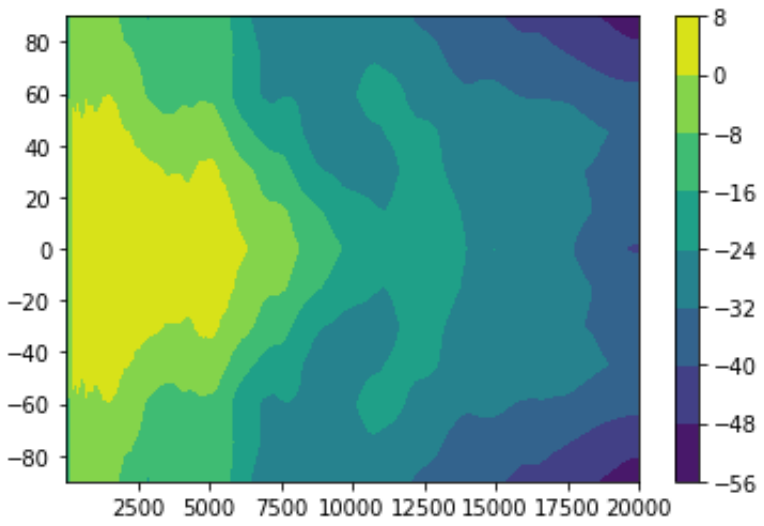
b) Se puede o no almacenar la matriz correspondiente a cada archivo en una lista, pero lo importante es, de cada matriz / archivo, levantar el valor correspondiente a la *magnitud* para un frecuencia de 1 kHz.

c) Puede realizarse el gráfico con las amplitudes adquiridas, pero

para llegar a la figura del ejemplo debe asumirse simetría, esto es, la amplitud en α grados es igual a la amplitud en $-\alpha$ grados.

Pueden leer más sobre esta función en la documentación oficial (clic para ver).

7. Seguiremos otro poco con los ejemplos del ejercicio anterior. La realidad es que el gráfico polar está muy bien pero tenemos muchísimas frecuencias relevadas, y si bien podríamos tomar un diagrama polar para unas cuantas frecuencias, sería lindo poder representar magnitud, frecuencia y ángulo de rotación en un único gráfico. Felizmente, las funciones `plt.contourf, plt.contour` y `plt.pcolormesh` permiten hacer un gráfico muy completo de esta situación. Un gráfico hecho con `contourf` se ve a continuación:



Para realizar este gráfico, necesitamos una matriz de magnitudes. Como en este caso se utilizaron 13 angulos de rotación (7 naturales, 6 por simetría) y 806 frecuencias, neecesitamos una matriz `magnitudes` de 13×806. El procedimiento para usar la función `contourf` es el siguiente:

a) Levantar de cualquiera de los archivos de datos la primera columna, y almacenarla en un vector llamado `frecuencias`. Construir un vector `angulos` con los distintos angulos de rotación (pueden usar o no simetría, pero es interesante implementarlo con simetría de la forma menos tediosa posible).

b) Construir dos matrices, `X` e `Y`, mediante el comando `X, Y = np.meshgrid(frecuencias, angulos)`. Tómense un momento para entender qué son `X` e `Y`.

c) Contruir la matriz `magnitudes`, que debe tener el mismo tamaño que `X` e `Y`, conteniendo una fila por cada valor de rotación y una columna por cada valor de frecuencia.

d) La función se invoca como `plt.contourf(X, Y, magnitudes)`. Tiene muchos argumentos opcionales, como el mapa de colores, coeficiente de transparencia, cantidad de colores a utilizar en el mapa, etc. La paleta de colores referenciada al nivel se incluye con la sentencia `plt.colorbar()`

Pueden leer más sobre esta función en la documentación oficial (clic para ver).

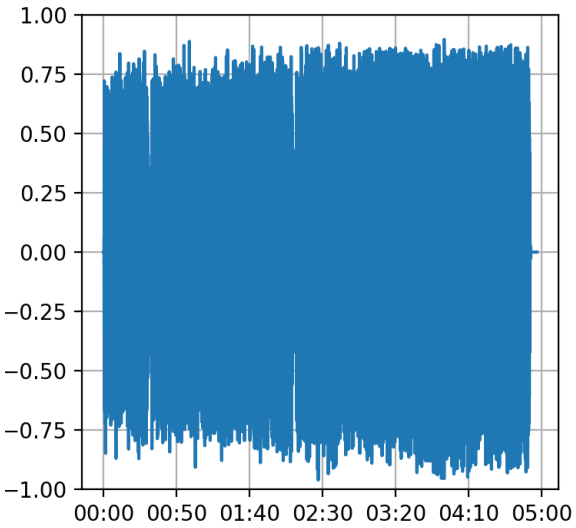
8. *(Opcional, para loquitxs de Excel).* Si hicieron el último ejercicio, posiblemente tuvieron un trabajo relativamente tedioso viendo cómo extraer todos los datos del archivo `.txt`. Una forma muy habitual de manipular datos en Python es utilizando Excel u otro software de planillas de cálculo. Felizmente, existe una extensa librería de análisis de datos que permite traducir archivos en formato `.xlsx` a datos soportados por Python. Esta es la librería `pandas`, que si bien tiene muchísimas facilidades, suele utilizarse, inicialmente, de esta manera:

```
import pandas as pd
df = pd.read_excel('mi_archivo.xlsx')
```

La variable `df` contiene un objeto muy especial, pues no es una lista, ni una tupla, ni un vector de `numpy`, sino un objeto muy poderoso de `pandas` llamado `DataFrame`. Lo bueno del `DataFrame` es que, si empezamos la tabla en la celda A1 de la planilla, cada columna de nuestra tabla puede invocarse como `col = df['nombre_de_columna']`. Así, se les da una tabla que contiene 7 columnas, correspondientes a cada ángulo de rotación, y una fila por cada frecuencia.

Repetir el gráfico realizado con `contourf` utilizando `pandas` y la planilla de Excel suministrada (`mediciones.xlsx`).

9. Implementar la función `time_plot(audio)` que recibe una señal de audio y la grafica, mostrando el `tiempo` en el eje x, formateado en minutos y segundos como se muestra en la gráfica de ejemplo que figura a continuación.



Sugerencia: una forma rápida de hacer esto es con la función de alto orden `FuncFormatter`, perteneciente al módulo `matplotlib.ticker`. La siguiente línea de código permite modificar los valores del eje x de un gráfico ya construido con `plt.plot`.

```
plt.gca().xaxis.set_major_formatter(FuncFormatter(codigo_tiempo))
```

La función `codigo_tiempo(valor, posicion)` toma cada `valor` en el eje x que le pasa `FuncFormatter`, y lo transforma en un `str` con la estructura `'MM:SS'`, donde `'MM'` son los minutos transcurridos, y `'SS'`, los segundos. Dicho de otra manera, `codigo_tiempo(valor, posicion)` es una función que debemos definir al principio de nuestro código, y debe devolver un `str` a partir del argumento numérico `valor` (la `posicion` no se usa).

2. Operaciones numéricas

1. Se mostró cómo calcular el área encerrada por un conjunto de datos aproximándola mediante el método de los rectángulos. También se hizo aproximando mediante trapecios, y el resultado era mejor. Finalmente, se calculó el **área acumulada** de una función sólo con la regla de los rectángulos (que en ese caso implementamos con `cumsum`) para aproximar la primitiva. Se pide:

a) Implementar la función `cumtrapz(datos_x, datos_y)`, que calcula el área acumulada por una función y aproxima la primitiva de la función utilizando la regla de los trapecios.

b) Implementar la función `area_desde_hasta`, que toma como argumento un vector, junto a dos muestras `m_inicial` y `m_final` y aproxima el cálculo del área encerrada entre esas dos muestras (inclusive).

c) Implementar la función `area_desde_hasta_2`, que toma como argumentos una variable independiente y una variable dependiente, junto a dos valores a y b y aproxima el cálculo del área:

$$\int_a^b f(x)dx$$

OJO: Al calcular el área con sumas, uno suma desde una *muestra* hasta la otra, pero a y b no son muestras, sino valores que asume la variable independiente `datos_x` en ciertas muestras. Deben obtener esos valores de muestras y sumar desde allí. Si la variable independiente nunca asume los valores solicitados a y b, la función debe informarlo.

2. Se tiene el siguiente conjunto de mediciones:

```
datos_x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
datos_y = [9.53647, 151.323, 2223.38, 16441.1, 78212.4,
           280053, 823687, 2.09733e+06, 4.78319e+06,
           1.00003e+07]
```

Dados dos vectores de `numpy`, `u` y `v`, una forma de estimar "qué tan diferentes son" es con el **error cuadrático medio**, calculado como `np.mean((u-v)**2)`.

a) Utilizando un ajuste polinómico por cuadrados mínimos (implementado por la función `np.polyfit`) hallar qué polinomio de grado entre 1 y 7 inclusive estima mejor el conjunto de datos suministrado. Para ello, construya los coeficientes del polinomio de estimación para todos los grados entre 1 y 10, evalúelos en el conjunto `datos_x` con `np.polyval` y, para cada polinomio resultante, calcule el error cuadrático medio entre el polinomio y el vector `datos_y`. Grafique el polinomio con mejor estimación junto a los datos estimados.

b) Repita el experimento para un polinomios de ajuste de grado 49.

c) Ahora, evalúe el polinomio que haya obtenido como mejor estimación en el conjunto obtenido mediante `np.linspace(1, 10, 1000)`, grafíquelo y analice si su respuesta era realmente la mejor estimación posible. ¿Cómo se explica lo que ocurre?

3. Se mostró cómo resolver ecuaciones diferenciales que no podríamos resolver analíticamente usando la poderosa función `solve_ivp` del módulo `scipy.integrate`. Utilizar ese método para resolver las siguientes ecuaciones diferenciales.

   a) $\frac{dy}{dt} = at^2 + bt + k; y_0 = 6$ (evalúan el polinomio con `np.polyval`)

   b) $\frac{dy_0}{dt} = 6e^t + 2, \frac{dy_1}{dt} = 5y_0^2 + 3t$. Usar $y_0 = y_1 = 0$.
   Recuerden que, para resolver dos ecuaciones relacionadas, deben enviar a `solve_ivp` un vector con las dos ecuaciones, según se mostró en los ejemplos del libro.

   c) $\frac{d^3y}{dt^3} = 3$
   Recuerden una ecuación de orden 2 pude realizarse mediante un cambio de variable y llevándolo a dos ecuaciones relacionadas. Esta ecuación de la física, llamada **ecuación de tirón (*jerk*) constante**, va a requerir dos cambios de variable y convertirlo a tres ecuaciones, que se cargan como vector a `solve_ivp`. Usar $y_0 = y_1 = y_2 = 0$, o sea, posición, velocidad y aceleración iniciales iguales a cero.

   Sugerencia: crear primero una función de tres variables `(t, y, c)`, de forma tal que `c` sea un vector, e incluya las constantes del problema como variable. Luego, crear otra función que evalúe ese vector `c` en los valores deseados, para evaluar después en `solve_ivp`.

4. *(Más difícil, basado en una historia real)* El doctor Pablo Bolcatto es un físico santafesino que ha elaborado un modelo físico simplificado del canto de una chicharra. En este modelo, una chicharra es una estructura mecánica excitada por las vibraciones que emite desde su abdomen (membrana timbal) y sus costillas (largas y cortas). De acuerdo a este modelo, la vibración $x(t)$ del cuerpo puede modelarse como:

$$\frac{d^2y}{dt^2} + 2\gamma\frac{dy}{dt} + (2\pi f_0)^2 y = F(t)$$

   Aquí, γ es el coeficiente de amortiguamiento y f_0 es la frecuencia de resonancia de la chicharra. Además, la fuerza que excita al insecto de

compone de tres señales, que vienen de la membrana timbal y cada grupo de costillas, siendo su ecuación:

$$F(t) = \sum_{i=1}^{3} \alpha_i \exp\left(-\frac{\cos^2(\pi f_i)}{\Delta_i^2}\right)$$

Aquí, α_i es la amplitud de cada pulso, f_i es la frecuencia de excitación de cada órgano y Δ_i es el ancho de pulso de la excitación. Entonces:

a) Analizar mediante un gráfico la señal $\alpha \exp\left(-\cos^2(\pi f)/\Delta^2\right)$ para $\alpha = 10^9, f = 3000, \Delta = 1/20$.

b) Resolver la ecuación diferencial del doctor Bolcatto para los valores $\alpha_1 = \alpha_2 = \alpha_3 = 10^9, \Delta_1 = \Delta_2 = \Delta_3 = 1/20, \gamma = 300, f_0 = 3350, f_1 = 200, f_2 = 3250, f_3 = 2970$. Utilizar condiciones iniciales `[0.001, 0]`. Graficar la señal $y(t)$ y reproducirla utilizando `sd.play()`. Notar que este es un modelo de las vibraciones del insecto y no de su canto, y no debería sonar para nada agradable. Además, ojo a la amplitud, puede salir enorme de la simulación y romperles los oídos, parlantes y demás.

c) Graficar el espectro de la señal $y(t)$ utilizando la archimegasúperimportante función `np.fft.rfft`, junto a una versión suavizada de la misma. Pueden suavizarla con un filtro de media móvil, un filtro Savitzky-Golay o cualquier otro que encuentren en la web. Usen la ventana del tamaño que les guste.

5. El D_{50} es un parámetro acústico utilizado para analizar la *definición* del sonido en un recinto acústico. Se relaciona con el porcentaje de palabras inteligibles (o sea, entendibles) en la sala, y como tal se calcula como **qué porcentaje de la energía de la señal se da en los primeros** 50 **milisegundos de la señal**. Su definición matemática es:

$$D_{50} = 100 \cdot \frac{\int_0^{0.05\ \mathrm{s}} p^2(t)dt}{\int_0^{\infty} p^2(t)dt}\ [\,\%]$$

Aquí, $p(t)$ es una respuesta al impulso medida en el lugar donde queremos analizar la definición sonora.

Implementar la función `D50(impulso, fs)`, que recibe una respuesta al impulso con su frecuencia de muestreo medida en una sala y devuelve, para ella, el valor de D_{50} calculado con la fórmula mencionada.

6. El C_{80} es un parámetro similar al D_{50}, que representa **el cociente entre la energía ocurrida en los primeros** 80 **ms. de señal y la energía ocurrida después**. Supuestamente, esto nos da una medida de cuanta energía se da de forma temprana y cuánta en una bola de sonido en la que ya no se distingue mucho cuándo termina un sonido y empieza otro. Se calcula así:

$$C_{80} = 10 \log_{10} \left(\frac{\int_0^{0.08\ \text{s}} p^2(t)dt}{\int_{0.08}^{\infty} p^2(t)dt} \right) \ [dB]$$

   Implementar la función `C80(impulso, fs)`, que recibe un impulso con su frecuencia de muestreo y calcula el C_{80}.

3. Ejercicio proyecto

En este ejercicio, se pide implementar la función `TR(impulso, fs)`. Esta función calcula el **tiempo de reverberación** de una sala a través de un impulso. El tiempo de reverberación se define como el tiempo que el impulso tarda, desde el pico de máximo nivel, en caer 60 dB. Como en general hay bastante ruido de fondo, es difícil llegar a tener 60 decibeles de diferencia, y por eso el `TR` se estima calculando una caída de 20 dB y multiplicándola por 3 (en cuyo caso, se le dice T_{20}) o calculando una caída de 30 dB y multiplicándola por 2 (que se le dice T_{30}). Además, como queremos estar seguros de que la fuente se apagó, en vez de calcular la caída desde el máximo (0 dB) hasta los -20 dB, se calcula desde -5 dB hasta los -25 dB (o -5 a -35, obvio). Implementaremos, por ahora, el TR global (sin considerar distintas frecuencias).

Entonces, para calcular el tiempo de reverberación, se propone el siguiente orden de trabajo:

1. Levantar la respuesta al impulso con `sf.read`. Como está llena de fluctuaciones muy rápidas, se le deben aplicar una o varias (preferentemente todas) técnicas de suavizado. Se suelen aplicar, en este orden:

   a) Módulo de la transformada de Hilbert, se implementa en `Python` mediante `np.abs(scipy.signal.hilbert(x))`.

   b) Un filtro pasabajos, p. ej., media móvil o Savitzky-Golay.

c) La integral de Schroeder, que puede calcularse como:

$$E(t) = 10 \log_{10} \frac{\int_t^\infty p^2(t)dt}{\int_0^\infty p^2(t)dt}$$

Aquí $p(t)$ es la señal resultante de aplicar los dos procesos anteriores al impulso.

Pueden probar aplicar algunos de esos métodos y otros no. Al final pasen la señal a dB haciendo:

$$E(t) = 20 \log_{10} \left(\frac{p(t)}{\text{máx}\,[p(t)]} \right)$$

OJO: antes de tomar el logaritmo, deben reemplazar los ceros por el mínimo valor distinto de 0, así el logaritmo no explota. Recordar la función `a_dbfs` de la guía 2.

2. Una vez obtenida una señal suavizada $E(t)$, esta debería caer en forma bastante lineal, pues es el logaritmo de una caída exponencial. Entonces, lo que se hace es *ajustar la caída con cuadrados mínimos, con una recta, o sea, un polinomio de grado 1* (**VER OBSERVACIÓN!!!**). Sobre esa recta que queda buscamos la caída de -5 dB, la de -25 dB (o lo que sea) y calculamos el tiempo de reverberación.

OBSERVACIÓN: Esta obervación es súper importante.Si la señal que recibimos es un impulso recortado a mano que termina cuando sólo queda ruido de fondo, digamos que en principio está todo bien. Sin embargo, si yo tengo un impulso con algo de ruido de fondo al final, cuando se apagó el balazo, la detección automática del punto donde terminó el impulso y quedó sólo el ruido es *un problema de verdad*. Necesitamos ese punto, pues ahí es hasta donde hacemos la integral de Schroeder. Para hallar ese punto, el método más usado es el aplicado por Lundeby en el artículo *Uncertainties of Measurements in Room Acoustics*, pero les contamos dos opciones más, como por ejemplo tomar ventanitas y calcular el valor RMS, y ver a partir de cual, yendo de atrás hacia adelante, el valor RMS es estrictamente creciente. Otro posible es el de analizar el ajuste de cuadrados mínimos con varias rectas, y quedarse con la que tiene la menor diferencia (esto es, el menor error cuadrático medio) contra la señal resultante.

Toda esta información está un poquito más desarrollada en esta formidable respuesta en StackExchange (clic para ver).