# Métodos Computacionales 2: Taller 2

# Fourier

Por Santiago Henao Castellanos

Este taller se centra en los conceptos y aplicaciones de la transformada de Fourier.

El taller debe ser presentado como un único archivo `.py`, que debe asumir que todos los datos solicitados se encuentran en la carpeta actual, y debe correr simplemente invocando `python Tarea_2.py`. El código no debe tener ningún `plt.show()`, y tampoco debe interrupir su ejecución o esperar ningún input. Todos los resultados que se pidan guardar (gráficas, videos, etc) deben estar en la misma carpeta del código, y deben ser los mismos que produce el código al ser ejecutado. Si se especifica que debe definir una función, debe usar el nombre asignado. Se penalizará con nota si no se cumplen estas especificaciones.

Si el código alza una excepción, se calificará hasta ese punto.

# 1. Intuición e interpretación (Transformada general)

La siguiente es una función que puede utilizar en este punto para generar sus datos para este punto:

```
def generate_data(tmax,dt,A,freq,noise):
    '''Generates a sin wave of the given amplitude (A) and frequency (freq),
    sampled at times going from t=0 to t=tmax, taking data each dt units of time.
    A random number with the given standard deviation (noise) is added to each data point.
    ----------------
    Returns an array with the times and the measurements of the signal. '''
    ts = np.arange(0,tmax+dt,dt)
    return ts, np.random.normal(loc=A*np.sin(2*np.pi*ts*freq),scale=noise)
```

Para todo este punto se necesitará la función que se pide escribir en la Sección 1.a.

## 1.a. Límite de Nyquist

### 1.a.a. Implementación

Escriba una función que, dados unos arrays de tiempo ($t$), de medición de la señal ($y$), y de frecuencias ($f$), calcule la transformada en dichas frecuencias.

La definición puede ser:

$$F_k = \sum_{i=1}^{N} y_i e^{-2\pi \mathbb{i} f_k t_i}$$

donde $N$ es la longitud de los datos. NOTA: el array de frecuencias no necesariamente es del mismo tamaño que los datos.

La función debe llamarse `Fourier_transform`.

### 1.a.b. Prueba

Genere una señal con la función proporcionada arriba, y grafique su espectro calculado hasta 2.7 veces la frecuencua de Nyquist. Guarde como `1.a.pdf`

## 1.b. Signal-to-noise

La razón señal-a-ruido (SN) se define como la amplitud del fenómeno que nos importa (signal) sobre una medida del ruido del fondo (noise), como lo puede ser la desviación estándar.

Con la función de arriba podemos generar una señal con el $\mathrm{SN}_{\mathrm{time}}$ que queramos, porque sería igual a `A/freq`.

En el dominio de frecuencias, se mide el $\mathrm{SN}_{\mathrm{freq}}$ como la altura del pico principal de la señal dividida sobre la desviación estándar de la parte del espectro que no tiene picos.

Genere muchos conjuntos de datos, cada uno $\mathrm{SN}_{\mathrm{time}}$ diferente (puede ser logarítmicamente distribuídos de 0.01 a 1.0) y calcule las SN de cada uno de esos datos pero en espacio de frecuencias. Grafique $\mathrm{SN}_{\mathrm{freq}}$ vs $\mathrm{SN}_{\mathrm{time}}$.

Encuentre algún modelo para lo que observa. Posiblemente se vea mejor en log-log.

Para pensar: ¿qué variables harían cambiar este comportamiento? (`A`, `tmax`, `freq`, `dt`, ...)

## 1.c. Principio de indeterminación de las ondas

Usando la función para generar datos, muestre cómo cambia el ancho de los picos de la transformada en función de `tmax`.

Para pensar: ¿esto cambia si muevo alguna de las otras variables?

## 1.d. (BONO) Más allá de Nyquist

Modificque la función `generate_data` para que acepte un argumento ***opcional*** llamado `sampling_noise` que meta ruido a los tiempos de muestreo de la señal `ts` *antes* de que se mida la señal, y retorne la señal medida en esos nuevos tiempos "perturbados".

Grafique la transformada hasta 2.7 veces la frecuencia de Nyquist, para varios `sampling_noise`.

Dependiendo de qué tan grande sea ese ruido de muestreo, los picos repetidos de la transformada deberían irse eliminando, quedando la frecuencia real, incluso si ésta es mayor que la de Nyquist.

En bloque neón hay una muestra de cómo podría quedar esta gráfica.

# 2. Ciclos de actividad solar (FFT 1D)

Adjuntos encontrará unos datos `SN_d_tot_V2.0.csv` que corresponden al registro histórico más extenso de manchas solares que pude conseguir. Tanto los datos como su descripción están disponibles públicamente desde el Observatorio Real de Bélgica, pero con el archivo adjunto basta.

## 2.a. Arreglar

Importe los datos. Notará que antes de 1850 hay algunos días que tienen $-1$ manchas. Esto claramente quiere decir que no se tomaron datos (NO quiere decir que no hayan manchas).

Use algún método que no sea de Fourier para reemplazar estos valores faltantes.

Para pensar: ¿por qué no puede simplemente quitarlos?

## 2.b. Filtrado y análisis

- Obtenga el período del ciclo solar en días.
  - BONO: use el truco descrito en clase para encontrar el período con aún más precisión.
  - En cualquier caso, guarde este número como texto en un archivo llamado `2.b.txt`
- Diseñe un filtro pasa bajas para capturar el comportamiento general de los datos, sin tanto ruido.
  - Puede ponerse creativo con la función de filtro.
  - Grafique los datos antes y después de filtrar en `2.b.data.pdf`.
- Halle los máximos locales de la señal filtrada, que debería ser una curva suave.
  - Grafique el número de manchas solares en el máximo contra la fecha en la que ocurre el máximo en `2.b.maxima.pdf`.
  - Se baja si se considera el año como variable categórica.

Para pensar: ¿qué tanto se puede filtrar la señal?

# 3. Filtrando imágenes (FFT 2D)

## 3.a. Desenfoque

Adjunta encontrará una foto del gato Miette. Multiplique la transformada 2D con una imagen del mismo tamaño de una gaussiana para obtener una versión desenfocada de la imagen. Debe hacer esto para cada canal de color de la imagen.

Se recomienda abrir la imagen con `np.array(PIL.Image.open(...))`. Se recomienda tratar la transformada de la imagen con `fftshift`.

Guarde la imagen borrosa como `3.a.jpg`.

## 3.b. Ruido periódico

### 3.b.a. P_a_t_o

Adjunta también encontrará la imagen de un pato, en blanco y negro esta vez. Esta imagen presenta ruido periódico, que puede evidenciar por los picos en la transformada bidimensional.

Elimine estos picos manualmente, poniendo ceros en el array donde están los picos. Realice la transformada inversa para obtener la imagen sin ruido periódico. Guarde la imagen en `3.b.a.jpg`

Para pensar: en vez de ceros, ¿se le ocurre alguna manera mejor de quitar estos picos?

### 3.b.b. G_a_t_o

Haga lo mismo con la imagen del gato que parece que estuviera detrás de unas persianas medio abiertas. Guarde en `3.b.b.png`

Para pensar: ¿se le ocurre alguna manera de detectar estos picos automáticamente?

## 4. Aplicación real: datos con muestreo aleatorio

El archivo de datos `OGLE-LMC-CEP-0001.dat` contiene tres columnas: tiempo, brillo, e incertidumbre en el brillo de una estrella. El tiempo está dado en el número fraccionario de días desde el 9 de Octubre de 1995, por alguna razón.

Los datos fueron tomados en Las Campanas, Chile, y tienen un muestreo de más o menos cada noche. A veces se toman varios datos la misma noche, a veces hay varias noches sin datos, y se toman a horas ligeramente distintas.

Para pensar: ¿podríamos decir que la frecuencia de muestreo es de 1 por día?

Encuentre la frecuencia de la señal.

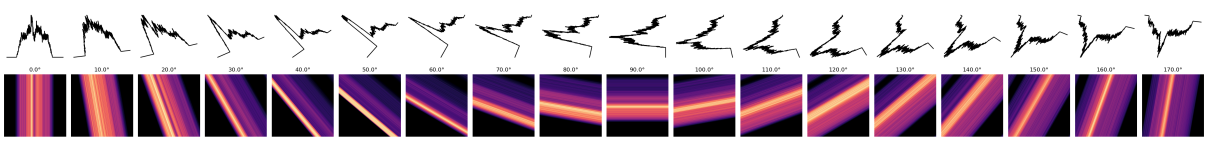
Para comprobar que sea esta realmente la frecuencia de la señal, calcule la *fase* `ϕ = np.mod(f*t,1)`, donde $f$ es la frecuencia de la señal y $t$ es la columna de tiempo.

Grafique el brillo de la estrella en función de $\phi$, guarde en `4.pdf`.

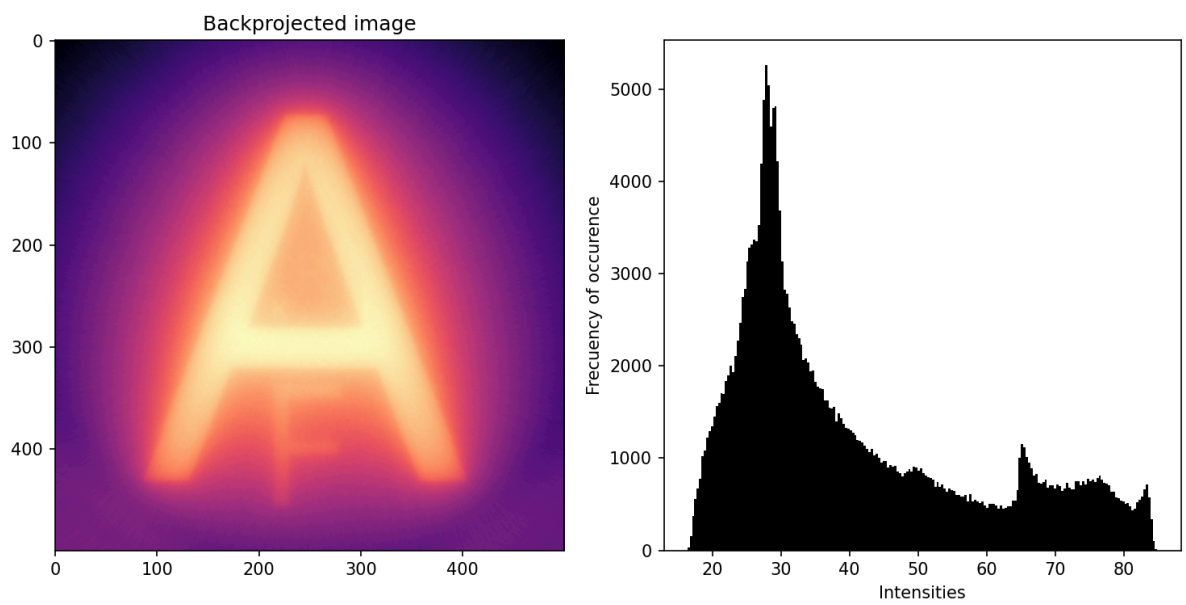
# 5. Aplicación real: Reconstrucción tomográfica filtrada

En bloque neón hay un video que muestra en resumen cómo funciona una tomografía. Mire el video antes de continuar.

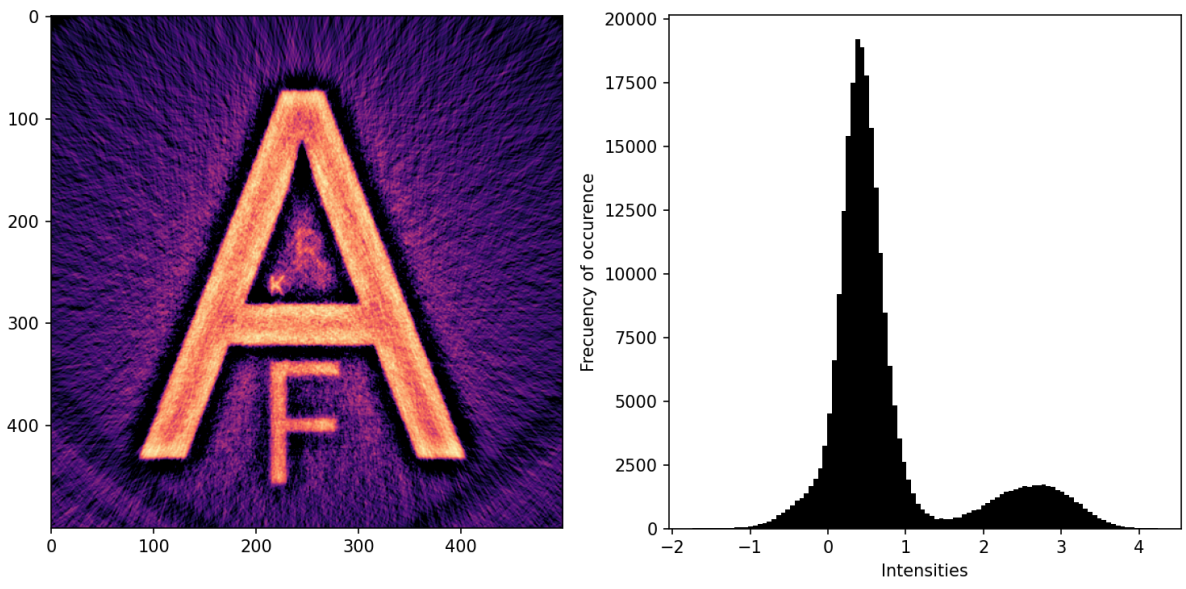
- Para cada ángulo, se toma una imagen 1D de los rayos X atravesando la parte del cuerpo que se quiere observar.
  - El resultado es la una suma de intensidades para ese ángulo, una señal 1D.
- Se hace esto para muchos ángulos y se guardan las proyecciones.
- Luego, cada imagen 1D se convierte en una imagen 2D repitiendo los datos hacia abajo.
  - La imagen resultante se rota, como se muestra en la figura




- Se suman todas estas imágenes, y así se logra una reconstrucción aproximada de la imagen original.




Pero esta imagen es muy borrosa, no se distinguen los detalles. Para mejorar los detalles, diseñe un filtro pasa altas y aplíquelo a cada una de las proyecciones (señales 1D) antes de la suma. Esto producirá algunos artefactos en la imagen, pero mejorará el contraste:




Pista: si tiene la señal 1D en el array `signal`, para formar la imagen ya rotada puede servirse del siguiente código:

```
from scipy import ndimage as ndi
imagen_rotada = ndi.rotate(
    np.tile(signal[:,None],rows).T,
    rotation_angle,
    reshape=False,
    mode="reflect"
)
```

Donde `rotation_angle` es el ángulo de rotación en grados, `rows` es el número de filas (pixeles en y) de la imagen resultante, y los demás argumentos se encargan de conservar el brillo.

Encontrará una carpeta con muchos conjuntos de datos de proyecciones de una tomografía craneal. Use el número de su grupo. Si por algún motivo algún miembro de su grupo tienen sensibilidad a las imágenes anatómicas, use a `skelly.npy`

Guarde la imagen filtrada resultante en `4.png`.