Programació de Simulacions i d'Instruments de Mesura. Exercicis de programació en Python.

Grau de Física. Curs 2021-22.

Artur Carnicer, artur.carnicer@ub.edu

Actualitzat: 17 de novembre de 2021, 9:58

Índex

Tasca 1

Per començar.

1.1 Consideracions sobre l'assignatura.

- L'assignatura es divideix en dos mòduls disjunts de 3 ECTS cadascun: Programació de Simulacions en Python i Programació d'Instruments de Mesura (Labview).
- El mètode que seguirem en aquesta part es basa en l'aprenentatge basat en problemes / projectes. Totes les sessions són pràctiques i, estrictament, no existeixen classes de teoria. Tota la informació necessària per desenvolupar els exercicis es donarà sobre la marxa.
- Els exercicis es faran a classe i per tant, l'assistència és imprescindible. A més, és necessari dedicar dues hores de treball personal per cada hora de classe de mitjana, d'acord amb la filosofia del crèdit ECTS. Es recomana treballar en equip.
- Caldrà presentar diverses tasques al llarg del curs. La data d'entrega s'indicarà a classe i amb una setmana d'antelació. Caldrà presentar-les dins el termini indicat i han de fer correctament tot el que es demana. Es rebran comentaris del professor en cas que sigui necessari.
- Es treballarà sempre amb accés a internet. Això inclou els exàmens i les proves d'avaluació continuada.

A més, tingueu present que

- aquesta és una assignatura de física. Sovint caldrà aplicar coneixements previs i sentit comú.
- En programació, els errors són inevitables. Cal aprendre a enfrontar-se amb ells: són una excel·lent oportunitat d'aprendre coses noves. Cal acostumar-se a fer servir la documentació i els recursos de cerca que ofereix Internet.
- Un codi pot no fer el que un vol; però un codi que no s'executa és inacceptable.
- Escriviu la següent ordre a la consola Python: `import this`. Analitzeu el que diu: programar bonic i amb bon estil fa que el codi sigui més fàcil de llegir i de corregir en cas d'error. I possiblement més eficient i ràpid.

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
```

```
In the face of ambiguity, refuse the temptation to guess.
There should be one|and preferably only one|obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than right now.[a]
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea|let's do more of those!

Tim Peters, PEP 20 -- The Zen of Python
```

1.2 Anaconda. Instal·lació.

Durant el curs farem servir l'entorn de programació (Integrated Development Environment, IDE) Spyder. Aquest és programa molt potent que incorpora un editor, la consola IPython i un conjunt d'eines que faciliten la tasca de programació. Anaconda és una distribució de Python científic que us permet descarregar un únic fitxer instal·lable que inclou la darrera versió de Python, totes les llibreries científiques que necessitarem i l'Spyder.

Baixeu-vos el paquet des de `https://www.anaconda.com/products/individual`. Anaconda és codi obert i lliure de cost. Trobareu versions per Windows, Mac i Linux. Tret que l'ordinador on l'instal·leu sigui molt (molt!) antic, escolliu sempre la versió de 64-bits.

Per evitar conflictes de versions és imprescindible que tots fem servir la mateixa. A més si teniu instal·lada una versió anterior de Python, Jupyter notebooks o productes similars, cal que les elimineu primer per evitar que el sistema es torni inestable. L'instal·lador d'Anaconda no dóna cap mena de problema; si l'Anaconda no us funciona, probablement el problema està al vostre ordinador.

1.3 Canvis en les preferències de l'editor.

1. Quan arranqueu per primer cop (pot trigar una estona) trobareu una finestra com la que es mostra a la Fig. 1.1[1]. Es mostraran tres panells: l'editor, la consola IPython i el d'informació (en particular, ens interessa el que diu 'explorador de variables').

2. Si comencem un nou projecte, el primer que s'ha de fer és canviar el nom de l'arxiu (no treballeu mai en un fitxer temporal). Doneu-li un nom descriptiu. Important: guardeu-lo en un directori de treball que us sigui fàcil de trobar (és possible que el directori que us ofereix el programa no sigui el lloc adequat). Per compatibilitat amb tots els sistemes operatius, no feu servir espais ni caràcters propis del català o castellà: accents, dièresi, ç, ñ, l·l. Eviteu també símbols com @, #, {, etcètera.

3. Quan estigueu en condicions de fer córrer el programa, premeu el botó *play* (icona amb un triangle verd). Us hauria d'aparèixer una finestra com la de la Fig. 1.2. Assegureu-vos que el codi (i) s'executarà en una consola dedicada, (ii) que s'eliminarà el contingut de la memòria abans de l'execució i (iii) que tots els fitxers que necessiti o generi el vostre codi es troben al directori on heu guardat el fitxer. Finalment, (iv) assegureu-vos que està marcada l'opció que demana si aquest diàleg ha de ser visible la primera vegada que es fa córrer el codi.

4. Per tal que aquestes opcions siguin permanents i no calgui modificar-les cada cop, aneu al menú *Eines* i seleccioneu *Preferències*. A la finestra (Fig. 1.3) seleccioneu l'opció *Run* i marqueu les opcions indicades abans.

[1] L'aspecte de les finestres pot canviar lleugerament segons versions i idioma.

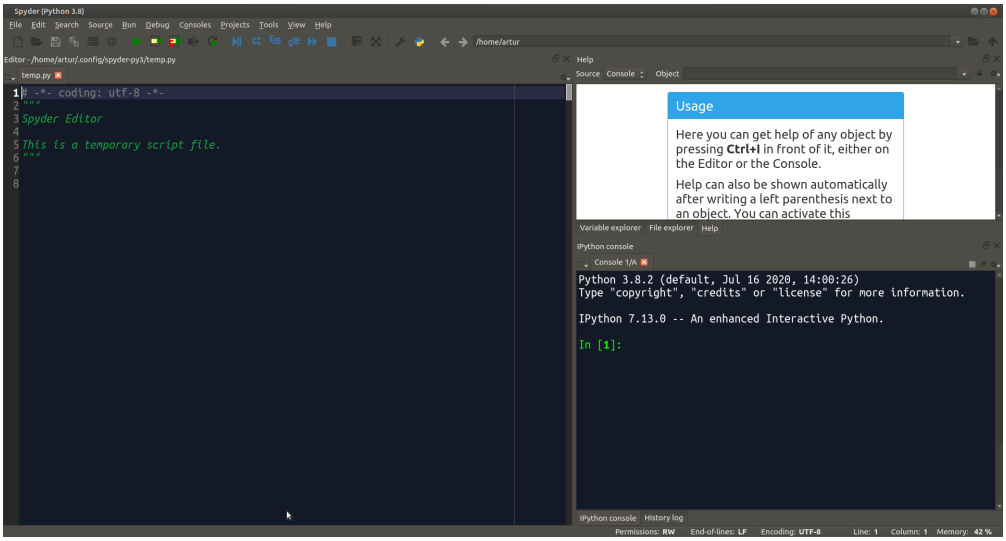


Figura 1.1:

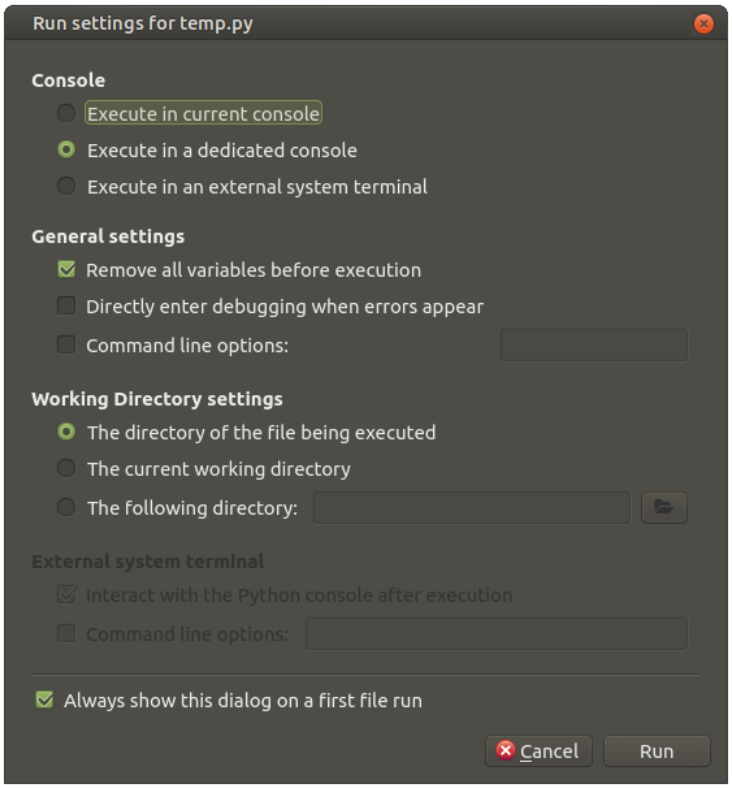


Figura 1.2:

5. **Opcional**: Sense sortir de la finestra de preferències, seleccioneu on s'indica *IPython console*, i aneu a la solapa *Gràfics* (Fig. 1.4). On posa *Backend* seleccioneu *Automàtic*. S'ha de dir que això va a gustos: Si deixem l'opció per defecte *Inline* els gràfics apareixeran a l'interior de la consola IPython, però si seleccionem *Automàtic* els gràfics es mostraran en finestres independents.

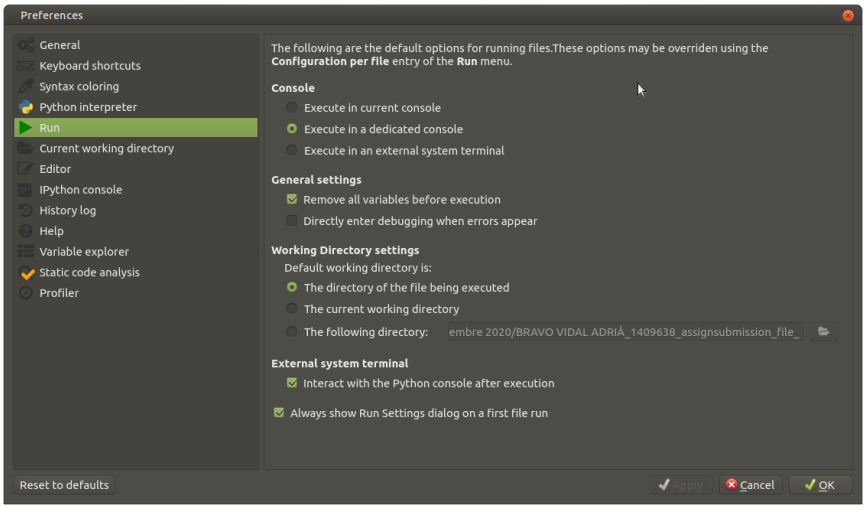


Figura 1.3:

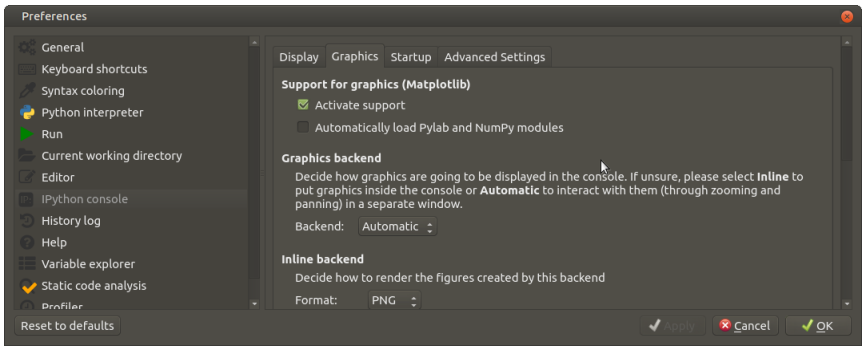


Figura 1.4:

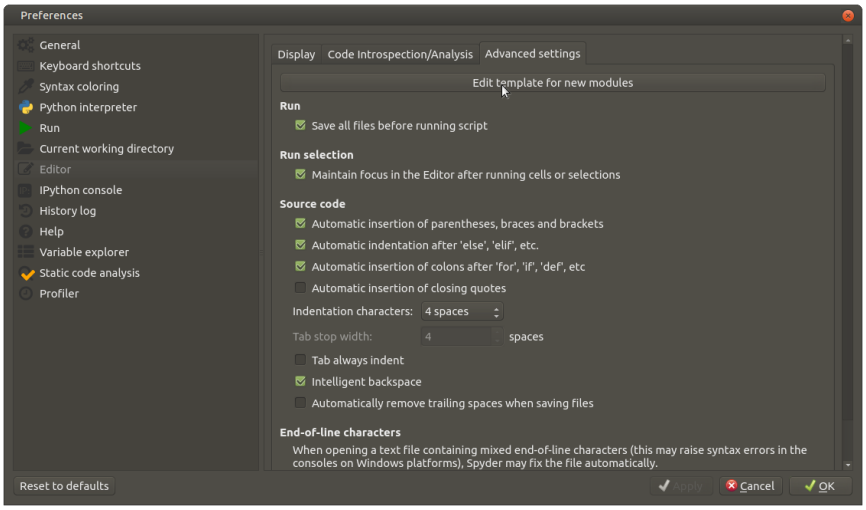


Figura 1.5:

6. **Opcional**: Aneu a l'opció *Editor* > *opcions avançades* (Fig. 1.5) i marqueu el botó Template. Editeu el fitxer: tot els que poseu aquí apareixerà en els nous scripts que genereu. Això és pràctic per incloure els mòduls que fem servir sovint (e.g. `numpy`, `matplotlib`, `scipy`. De moment podeu ignorar aquest pas i tornar aquí més endavant.

7. **Opcional (però recomanable)**: Finalment, activeu l'anàlisi automàtica d'estil. D'aquesta manera, l'editor us indicarà com millorar la llegibilitat del vostre codi d'acord amb les normes del PEP8 [2]. Aneu a *Editor* > *Introspecció del codi* i marqueu l'opció *Anàlisi de codi en temps real* [3].

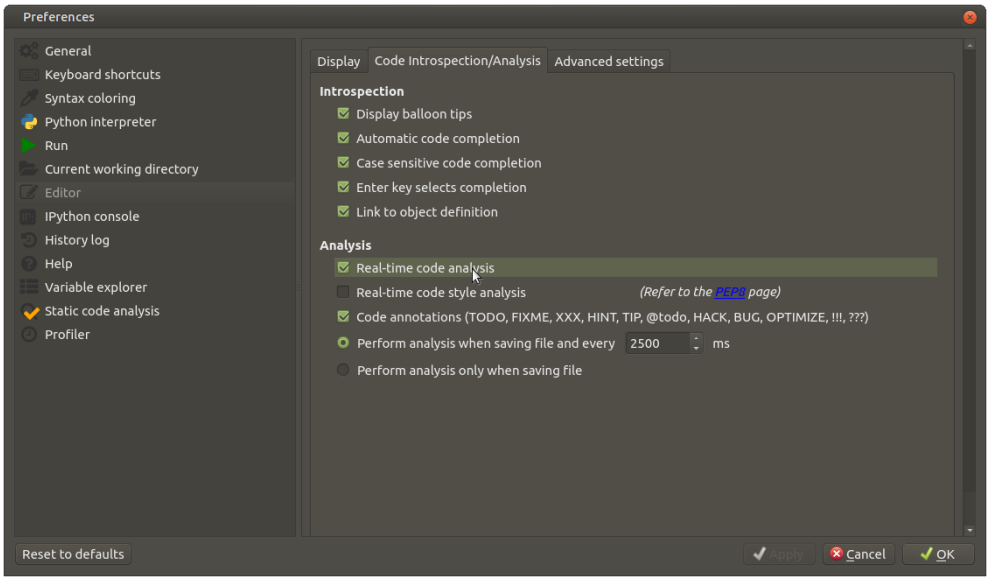


Figura 1.6:

1.4 Presentació de tasques.

1. Per cadascuna de les tasques haureu de presentar el codi que heu generat. Aquest ha de funcionar de forma autònoma sense intervenció del professor: escollir els valors adequats de les variables forma part del disseny de la simulació.

2. Cal que el codi sigui llegible, d'acord amb el que s'ha comentat abans. Comenteu el codi en aquelles parts que no resultin òbvies. Penseu que podreu disposar d'aquests materials durant els exercicis d'avaluació i en conseqüència és important que la informació que generada sigui de qualitat.

3. El codi ha de córrer sense errors. Aquells que no s'executin no seran considerats.

4. El codi es presentarà mitjançant el Campus Virtual de l'assignatura, abans de la data d'entrega. En cap cas es consideraran trameses passada l'hora límit. Absteniu-vos d'utilitzar el correu electrònic.

5. Anomeneu el vostre script *cognom1_cognom2_nom.py* (o bé *cognom_nom.py* si no disposeu de segon cognom). **No feu servir espais ni accents**. Pugeu-lo al campus virtual.

6. La presentació de la tasca suposa l'assoliment d'una fita que té un cert pes en la nota final. Es valoraran de la següent manera:

[2] PEP = Python Enhancement Proposals
[3] PEP 8 – Style Guide for Python Code `https://www.python.org/dev/peps/pep-0008/`

- 1: pràctica correcta
- 0.5: Resolució correcta, però incompleix algun detall, apareixen avisos d'errors no fatals en l'execució, etcètera.
- 0: codi incomplet (falten apartats), no entregada o errors d'execució.

Nota important: aquesta tasca no s'ha de presentar

Tasca 2

Conceptes de Python científic: *Pi-thon.*

Objectiu	Revisió de conceptes. Tècniques bàsiques de càlcul i representació.

Funcions, ordres i constants que s'introdueixen	`import matplotlib.pyplot as plt`
	`plt.figure`, `plt.plot`, `plt.subplot`, `plt.semilogy`, `plt.grid`, `plt.savefig`
	`import numpy as np`
	`np.linspace`, `np.arange`, `np.loadtxt`, `np.savetxt`, `np.array`, `np.append`
	`np.zeros`, `np.ones`, `np.empty`
	`np.Inf`, `np.NaN`, `np.pi`, `np.e`
	`import scipy.integrate as integrate`,
	`integrate.quad`, `integrate.simpson`
	`import numpy.random as random`, `random.rand`

Tècniques	Bucles `for` i `while`. Condicions (`if / else`) Llistes i array. Diferències entre els dos Gràfiques Funcions Funcions com a paràmetre de funcions Generació d'arxius de dades Càlcul de sèries i integrals sense paràmetres lliures

En aquesta pràctica calcularem el valor de π de diverses maneres. A partir dels exercicis introduirem diverses instruccions d'ús freqüent en Python.

1. Calculeu π mitjançant la fórmula Bailey-Borwein-Plouffe, `https://en.wikipedia.org/wiki/Bailey-Borwein-Plouffe_formula`. Itereu els termes de la successió mitjançant un bucle `while`.

   - Feu una gràfica de l'error en funció del nombre d'iteracions (compareu el valor obtingut en cada iteració amb `np.pi`). Atureu el càlcul quan l'error sigui prou petit. Feu servir `plt.figure`, `plt.plot`, etcètera. Us pot ser útil recordar com s'escriuen en Python els números en notació científica: e.g. $3000 =$ `3e+3`, $0.025 =$ `2.5e-2`. Feu servir escala logarítmica per a l'eix y. Decoreu la gràfica adequadament: graella, llegenda, tituleu els eixos, etcètera. Noteu que es poden posar caràcters grecs, i símbols matemàtics fent servir codis LaTeX.
   - Guardeu els valors de la gràfica en un arxiu de text pla. Considereu les ordres `np.loadtxt` i `np.savetxt`. Finalment, guardeu la imatge com un fitxer `.png` amb `plt.savefig`.
   - Opcional: podeu guardar taules de dades en un arxiu `.xlsx` fent servir aquesta recepta de codi:

     ```
     import pandas as pd
     df  = pd.DataFrame(array_2d)
     df.to_excel('arxiuexcel.xlsx', header=False, index=False)
     ```

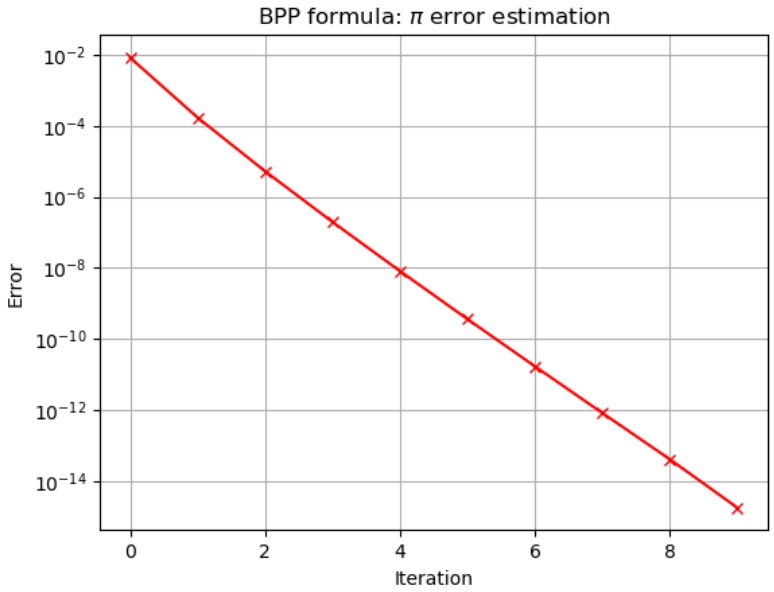


Figura 2.1: Evolució de l'error

2. Calculeu π a partir de les integrals següents:

$$\int_{-\infty}^{\infty} \exp(-x^2)\, dx = \sqrt{\pi} \tag{2.1}$$

$$\int_{-\infty}^{\infty} \frac{1}{1+x^2}\, dx = \pi \tag{2.2}$$

   Estudieu la informació que trobareu a `https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html`. Feu servir `integrate.quad` i/o `integrate.simpson`. Quan cal fer servir un o altre? Recordeu que per definir els valor de les abscisses on calcular la integral podeu fer servir `numpy.linspace` o `np.arange`. Noteu que Python disposa de valors predefinits (veieu `https://www.numpy.org/devdocs/reference/constants.html`). En aquest exercici us pot ser d'utilitat `np.Inf`.

3. Considereu un quadrat de costat L i el cercle inscrit en aquest quadrat. Genereu punts a l'interior del quadrat amb coordenades aleatòries. Determineu π a partir del quocient entre el número de vegades que els punts cauen al cercle i el nombre total de punts calculats: prenent números a l'atzar, genereu les coordenades de punts a l'interior del quadrat. Feu servir `numpy.random.rand`. Determineu si aquests punts es troben també a l'interior del cercle. Mostreu i guardeu la gràfica de l'error en funció del nombre d'iteracions. Modifiqueu la figura de l'apartat anterior i mostreu les dues figures de l'error en una única finestra. Feu servir `plt.subplot`.

4. `np.Inf` i `np.NaN`: Proveu de calcular `np.Inf - np.Inf`, `0 * np.Inf` o `np.Inf / np.Inf`. Què vol dir `np.NaN`? Poseu a la línia d'ordres `np.NaN == np.NaN` o `np.NaN != np.NaN`. Té sentit?

5. Estudieu el significat dels sencers de 8, 16, 32 i 64 bits i les seves funcions associades: `np.int8()`, `np.uint8()`, `np.int16()`, `np.uint16()`, `np.int32()`, `np.uint32()`, `np.int64()`, `np.uint64()`. Analitzeu les implicacions dels diferents tipus de sencers llegint aquest article publicat a *Wired*: `https://www.wired.com/2014/12/gangnam-style-youtube-math/`.

Nota important: aquesta tasca no s'ha de presentar

Tasca 3

Representació de funcions i més: l'equació de Debye.

Objectiu	Més sobre representació de funcions Resolució d'equacions Regressió

Funcions i ordres que s'introdueixen	`np.real`, `np.imag`, `np.argmax`, `.max()`, `1j`
	`plt.semilogx`, `plt.legend`, `plt.xlim`
	`scipy.optimize.fsolve`
	`scipy.stats.linregress`

Tècniques	Manipulació nombres complexos Màxim (i mínim) d'un *array* Més sobre funcions que criden a funcions: equacions no lineals Regressió lineal i correlació

L'equació de Debye, descriu el comportament de la permitivitat dielèctrica ϵ per a certs materials com l'aigua en funció de la freqüència i de la temperatura[1]:

$$\varepsilon(\nu, T) = \varepsilon_\infty(T) + \frac{\varepsilon_0(T) - \varepsilon_\infty(T)}{1 - i\,2\pi\nu\,\tau(T)} \tag{3.1}$$

on ν és la freqüència de l'ona electromagnètica ($\omega = 2\pi\nu$), i és la unitat imaginària, ϵ_0 i ϵ_∞ són les permitivitats estàtiques i a freqüències òptiques respectivament, i τ és el temps de relaxació rotacional. La dependència d'aquests tres darrers paràmetres en funció de la temperatura està descrita per les següents funcions:

$$\begin{aligned} \varepsilon_0(T) &= a_1 - b_1 T + c_1 T^2 - d_1 T^3 \\ \varepsilon_\infty(T) &= \varepsilon_0(T) - a_2 \exp(-b_2 T) \\ \tau(T) &= c_2 \exp\left(\tfrac{d_2}{T + T_0}\right). \end{aligned} \tag{3.2}$$

Els paràmetres es determinen experimentalment. En el cas que ens ocupa, aquests prenen els següents valors: $a_1 = 87.9$, $b_1 = 0.404$, $c_1 = 9.59\text{e-}4$, $d_1 = 1.33\text{e-}6$, $a_2 = 80.7$, $b_2 = 4.42\text{e-}3$, $c_2 = 1.37\text{e-}13$, $d_2 = 651$, $T_0 = 133$. Tots els paràmetres estan en unitats en SI excepte la temperatura que es dóna en °C. Aquest model és vàlid per temperatures i freqüències als intervals $0 \leq T \leq 100$ °C i 0.1 GHz $\leq \nu \leq 1000$ GHz [2].

1. Noteu que ϵ és una funció a valors complexos, $\epsilon = \epsilon_r + \epsilon_i i$. Mostreu les parts reals (ϵ_r) i imaginària (ϵ_i) en funció de la freqüència (0.1 GHz $\leq \nu \leq$ 1000 GHz) per a T = 0, 20, 40, 60, 80 i 100 °C. Mostreu

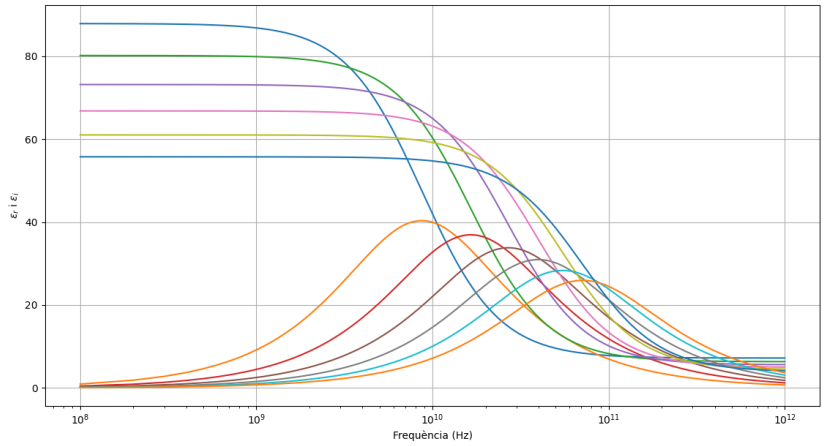


Figura 3.1: Permitivitat dielèctrica en funció de la freqüència

les 12 corbes en una única figura (**figura 1**). Feu servir eix logarítmic per a les freqüències. Decoreu la figura: poseu una graella, i etiqueteu els eixos.

2. Escriviu per consola els valors de les freqüències ν per a les quals $\epsilon_i(\nu, T)$ passa per un màxim (T = 0, 20, 40, 60, 80 i 100 °C).

3. Observeu que a l'interval $10^{10} - 10^{11}$ Hz les corbes ϵ_r i ϵ_i per a cada temperatura, es tallen. Determineu el valor de la freqüència ν_T per a la qual es verifica $\epsilon_r(\nu_T, T = 0) = \epsilon_i(\nu_T, T = 0)$. Feu el mateix per a tots els valor de T (T = 0, 20, 40, 60, 80 i 100 °C) i escriviu per consola els parells de valors següents:

Freqüència	T
$\nu_T(T = 0)$	0
$\nu_T(T = 20)$	20
$\nu_T(T = 40)$	40
$\nu_T(T = 60)$	60
$\nu_T(T = 80)$	80
$\nu_T(T = 100)$	10

Taula 3.1:

Feu una representació (**figura 2**) de les temperatures en funció de les freqüències de tall. Decoreu-la: poseu una graella, i etiqueteu els eixos. Nota: podeu fer servir la funció `scipy.optimize.fsolve` per resoldre les equacions. **Cal destacar que, malgrat que la permeabilitat és funció de ν i T, des del punt de vista pràctic pot ser convenient implementar-les en Python fent-les dependre només de ν.**

4. Els punts que heu obtingut a l'apartat anterior segueixen una llei aproximadament lineal. Utilitzant la funció `scipy.stats.linregress`, calculeu la recta de regressió. Indiqueu per consola el pendent, el terme independent i el coeficient de correlació. Mostreu la recta superposada als punts de la taula 3.1.

[1] Adaptat de l'examen de gener de 2020

[2] Exercici basat en els resultats publicats a Andryieuski, A., Kuznetsova, S., Zhukovsky, S. et al., "Water: Promising Opportunities For Tunable All-dielectric Electromagnetic Metamaterials", Sci Rep **5**, 13535 (2015).

Presentació: el codi ha de generar dues figures.

Tasca 4

Equacions diferencials ordinàries: pèndols i camps centrals

Objectiu	Resolució de problemes descrits per equacions diferencials ordinàries

Funcions i ordres que s'introdueixen	`scipy.integrate.odeint`
	Sub-arrays: l'operador `b:e:s`
	`scipy.special.eval_hermite`

Tècniques	Integració d'un conjunt d'equacions diferencials de primer ordre

En aquest projecte mostrarem tècniques de resolució d'equacions diferencial ordinàries que aplicarem a diversos problemes de física: (i) el pèndol forçat i esmorteït, (ii) la resolució de l'equació d'Schrödinger per l'oscil·lador harmònic quàntic, (iii) l'oscil·lador acoblat i (iv) el moviment de cossos sotmesos a una força gravitatòria.

4.1 El pèndol esmorteït i forçat.

La segona llei de Newton del moviment és

$$\ddot{\theta} = -\frac{g}{L}\sin\theta + \frac{-b\dot{\theta} + A\cos\Omega t}{mL^2} \tag{4.1}$$

on g l'acceleració de la gravetat, θ, L i m són l'angle, la longitud i la massa del pèndul, respectivament, b el coeficient de fricció, A i Ω l'amplitud i la freqüència de la força externa i t és el temps. A més, cal tenir present que $\dot{\theta} = \omega$. L'ordre Python que permet resoldre equacions diferencials és:

```
res = scipy.integrate.odeint(equdif, ci, t).
```

Aquesta funció s'utilitza per calcular sistemes d'equacions diferencials de primer ordre. Tanmateix, en el cas que ens ocupa el que tenim és una única equació diferencial de segon ordre. Per treballar amb aquesta funció cal convertir una equació de segon ordre en un sistema d'equacions diferencials de primer ordre. La primera de les variables, `equdif`, [1] és una referència a la funció que conté la descripció del sistema d'equacions (no cal indicar les variables); `ci` és un vector amb les condicions inicials i `t` és un *array* amb els punts on s'avalua la solució (la variable respecte a la qual es deriva, el temps en el nostre cas). La funció `odeint` retorna `res`, un *array* 2D que conté el resultat.

Considerem inicialment el cas del pèndol simple. Aquest segueix l'equació diferencial

$$\dot{\omega} = -\frac{g}{L}\theta\,, \tag{4.2}$$

[1] Els noms de les funcions i de les variables utilitzades en el guió poden ser unes altres. S'han utilitzat aquestes només per il·lustrar l'exemple.

que escriurem com un sistema de dues equacions diferencials de primer ordre amb incògnites θ i ω:

$$\begin{aligned} \dot{\omega} &= -\frac{g}{L}\theta \\ \dot{\theta} &= \omega \end{aligned} \tag{4.3}$$

Fixeu-vos que la segona és una equació trivial però necessària. La part més delicada és com s'escriuen les equacions anteriors en la funció `equdif` que serà cridada per `odeint`. Per exemple, aquesta funció s'ha de declarar així:

```
def equdif(y,t):
"""
y[0]=velocitat
y[1]=angle
"""
    return -g * y[1] / L, y[0]
```

La utilització de la funció `scipy.integrate.odeint(equdif, ci, t)` ens obliga a ser molt curosos a l'hora d'escriure `equdif`. La variable `y` és una estructura indexada que tindrà tantes components com equacions integrem. Tenim, però, llibertat a l'hora de decidir a quina variable física assignem cada component. Estudieu la casuística que es presenta a la secció 4.5.

`res` és un *array* bidimensional amb tantes files com numero de punts tingui la variable `t` i tantes columnes com equacions (dues en el cas que estem estudiant). Podem seleccionar individualment tots els valors de cada columna fent servir l'operador `:`. Així, l'*array* unidimensional `res[:, 0]` contindrà el resultat d'integrar la primera de les equacions del return(i.e. `-g * y[1]/L`, la velocitat angular) mentre que `res[:, 1]` ens donarà la informació d'integrar `y[0]` (l'angle). Reproduïu l'exemple per familiaritzar-vos amb el format. Obtingueu els diagrames $\omega(t)$ i $\theta(t)$. Utilitzeu l'ordre `subplot` per organitzar els resultats en una única figura (**figura 1**). [2].

Generalitzeu el problema a un oscil·lador esmorteït i forçat descrit per l'equació diferencial de l'equació 4.1. Mostreu els diagrames $\omega(t)$ i $\theta(t)$ d'un pèndol sotmès a fregament i una força externa:

- Considereu primer el cas $b = 0$ i $A = 0$. Doneu valors raonables per a m i L i les condicions inicials.
- Repetiu el punt anterior considerant $b \neq 0$ i $A = 0$.
- Considereu ara un cas hipotètic en que $A = 1.35$, $m = 1$, $g = 1$, $L = 1$, $b = 0.5$ i $\Omega = 0.666$. Interpreteu els valors que obteniu. Tenen sentit?

Mostreu tres finestres, una per cada cas considerat (**figures 2 a 4**).

4.2 Condicions inicials definides en punts arbitraris.

L'equació de Schrödinger per a l'oscil·lador harmònic

$$-\frac{\hbar^2}{2m}\Psi"(x) + \frac{1}{2}m\omega^2 x^2 \Psi(x) = E\Psi(x) \tag{4.4}$$

on $\omega^2 = k/m$, m és la massa de la partícula, k la constant elàstica i $E = (n + \frac{1}{2})\hbar\omega$ els nivells d'energia ($n = 0, 1, 2, \ldots$), està relacionada amb la següent equació diferencial

$$\Psi"(x) + \left((2n + 1) - Bx^2\right)\Psi(x) = 0. \tag{4.5}$$

que té per solució analítica

$$\Psi_n(x) = \left(2^n n!\sqrt{\pi}\right)^{-1/2} \exp(-x^2/2) H_n(x) \tag{4.6}$$

[2] Podeu trobar més informació sobre `odeint` a la pàgina corresponent del manual [1]

on $H_n(x)$ són els polinomis d'Hermite amb la normalització utilitzada a Física. Resoleu l'equació diferencial 4.5 per $x \in [-5, 5]$, pels casos $B = 1$, $n = 0, 2, 4$. Feu servir les condicions inicials següents:

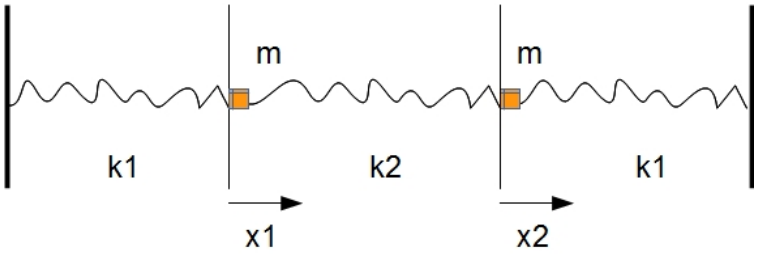
- $\Psi_0(0) = 0.751126, \Psi_0'(0) = 0$
- $\Psi_2(0) = -0.531125, \Psi_2'(0) = 0$
- $\Psi_4(0) = 0.459969, \Psi_4'(0) = 0$

Genereu una finestra amb dos subplots (**figura 5**). En el primer, mostreu les funcions d'Hermite per $n = 0, 2, 4$ [Eq. (4.6)]. En el segon, mostreu $\Psi(x)$ calculant l'equació diferencial [Eq. (4.5)]. Escaleu l'eix x entre -5 i 5 i l'eix y entre -0.6 i 0.8. Nota: la funció `scipy.special.eval_hermite` us pot resultar d'utilitat.

4.3 Oscil·ladors acoblats.

Considereu l'equació diferencial d'un sistema acoblat com l'indicat a la figura:

$$m\ddot{x_1} + (k_1 + k_2)x_1 - k_2x_2 = 0 \tag{4.7}$$
$$m\ddot{x_2} + (k_1 + k_2)x_2 - k_2x_1 = 0 \tag{4.8}$$



Resoleu l'equació diferencial per $m = 1$, $k_1 = 10$ i $k_2 = 0.5$ (unitats SI), amb les condicions inicials $x_1(0) = 1$, $x_2(0) = 0$, $v_1(0) = 0$, $v_2(0) = 0$. Feu variar el temps entre 0 i 40 s.

1. Genereu una gràfica que mostri l'evolució de $x_1(t)$ i $x_2(t)$. Poseu una etiqueta a l'eix 'x' que digui 'Temps (s)' i una a l'eix 'y' que digui 'Posició objecte'. Mostreu una graella superposada al gràfic.
2. Genereu una gràfica que mostri l'evolució de $v_1(t)$ i $v_2(t)$. Poseu una etiqueta a l'eix 'x' que digui 'Temps (s)' i una a l'eix 'y' que digui 'Velocitat objecte'. Mostreu una graella superposada al gràfic.
3. Mostreu les dues gràfiques en una finestra (**figura 6**).

4.4 Moviment d'un cos en un camp gravitatori.

En aquest cas es tracta de resoldre l'equació

$$\vec{F} = -G\frac{Mm}{r^2}\hat{r} \tag{4.9}$$

que en coordenades cartesianes s'escriu

$$\ddot{x} = -\frac{GMx}{(x^2 + y^2)^{3/2}} \tag{4.10}$$
$$\ddot{y} = -\frac{GMy}{(x^2 + y^2)^{3/2}} \tag{4.11}$$

Les variables en aquest problema seran x, y, v_x i v_y. Transcriviu el sistema d'equacions diferencials anterior. Busqueu les dades que necessiteu a Internet. Feu els càlculs per la Terra i el Cometa Halley al voltant del Sol. Mostreu $x(t)$, $y(t)$, $|v|(t)$ i $y(x)$. Mostreu les quatre gràfiques agrupades adequadament en una figura. Feu servir una finestra per la Terra i una pel Halley (**figures 7 i 8**). Per aquest darrer cas, feu servir les següents dades [2]:

```
G = 6.67e-11 (SI)
M_⊙ = 1.9891e30 Kg
au = 1.49598e11 m
aphelion = 35.082 * au
vel_ap = 0.869e3 m/s
T = 3.15576e7 * 75.32 s
```

Presentació: el codi ha de generar 8 figures.

4.5 Addenda: importància de l'ordre de les variables en el disseny de l'equació diferencial

En el codi que trobareu a continuació es consideren totes les combinacions possibles a l'hora de plantejar l'equació diferencial del pèndol simple. Algunes són correctes i d'altres no. Estudieu els casos i extraieu conclusions.

```
#!/usr/bin/env python3
#-*- coding: utf-8 -*-

import numpy as np
import scipy.integrate as spi
import matplotlib.pyplot as plt

def equdif1(y, t):
    # y[0] omega
    # y[1] theta
    # return omegaprima, thetaprima
    return -g * y[1] / L, y[0]

def equdif2(y, t):
    # y[0] theta
    # y[1] omega
    # return thetaprima, omegaprima
    return y[1], -g * y[0] / L

def equdif3(y, t):
    # y[0] omega
    # y[1] theta
    # return thetaprima, omegaprima
    return y[0], -g * y[1] / L

def equdif4(y, t):
    # y[0] theta
    # y[1] omega
    # return omegaprima, thetaprima,
```

```
    return -g * y[0] / L, y[1]

g = 9.8
L = 1
punts = 6000  # > 1e3 punts
deltat = 10
t = np.linspace(1, deltat, punts)

ci1 = [0, 0.01*np.pi]
ci2 = [0.01*np.pi, 0]

res1 = spi.odeint(equdif1, ci1, t)
res2 = spi.odeint(equdif2, ci2, t)
res3 = spi.odeint(equdif3, ci1, t)
res4 = spi.odeint(equdif4, ci2, t)

plt.figure()
plt.subplot(221)
plt.plot(res1[:, 0])
plt.plot(res1[:, 1])
plt.ylim([-0.1, 0.1])
plt.subplot(222)
plt.plot(res2[:, 0])
plt.plot(res2[:, 1])
plt.ylim([-0.1, 0.1])
plt.subplot(223)
plt.plot(res3[:, 0])
plt.plot(res3[:, 1])
plt.ylim([-0.1, 0.1])
plt.subplot(224)
plt.plot(res4[:, 0])
plt.plot(res4[:, 1])
plt.ylim([-0.1, 0.1])
```

Tasca 5

Arrays 2D: representació dels conjunts de Mandelbrot i Julia.

Objectiu	Familiaritzar-se amb l'ús d'estructures bidimensionals Utilitzar operacions lògiques amb estructures indexades

Funcions i ordres que s'introdueixen	`numpy.meshgrid`, `numpy.abs`
	`matplotlib.pyplot.imshow`,
	`matplotlib.pyplot.imsave`
	`imageio.mimsave`
	Operacions lògiques amb arrays

Tècniques	Visualitzar i guardar matrius en fals color Generar arxius dinàmics (`.gif`) Generar dues figures simultàniament Representació de funcions de dues variables.

El Conjunt de Mandelbrot (CM) es defineix com el conjunt de punts $c \in \mathbb{C}$, pels quals la successió

$$z_{n+1}(c) = z_n^2(c) + c \quad \text{amb} \quad z_0 = 0 \quad \forall c \tag{5.1}$$

convergeix. Es pot demostrar que aquesta sèrie convergeix en c si $\|z_n(c)\| < 2$. El CM es visualitza com una imatge binària: Si el valor considerat de c pertany al CM, se li assigna el valor `TRUE` (1), o `FALSE` (0) si no hi pertany. Per exemple, la Figura 5.1 mostra el CMs calculats per diversos valor d'n. L'objectiu d'aquesta pràctica és desenvolupar el codi per generar la representació gràfica del CM.

5.1 Indicacions per desenvolupar el codi.

1. Considereu inicialment que els valors que pot prendre c es troben a l'interval `-2 - 1.5j` i `1 + 1.5j`. Aquests valors es proposen perquè sabem que la fractal quedarà centrat. Més endavant podreu canviar aquests valors per verificar el caràcter fractal del CM. Genereu l'array complex de dimensió MxN que contindrà els valors de c. Les dimensions de c són una variable de disseny i les heu d'escollir vosaltres. Fixeu-vos que els valors de M i N condicionen el detall amb què visualitzareu el CM així com el nombre d'iteracions que haureu de fer fins que no observeu cap canvi al CM.

   Per generar c podeu utilitzar la funció `np.meshgrid()`. Estudieu el que diu la documentació d'aquesta funció per tal d'entendre el seu funcionament [1]. Construïu c fent `c = X + 1j * Y`, on `X` i `Y` surten de fer `X, Y = np.meshgrid(np.linspace(...), np.linspace(...))` .

2. Genereu l'array 2D `CM` que contindrà la representació del conjunt. Podeu reservar memòria fent servir `np.zeros()`.

[1] Consulteu la secció 5.2 per saber més sobre `np.meshgrid()`

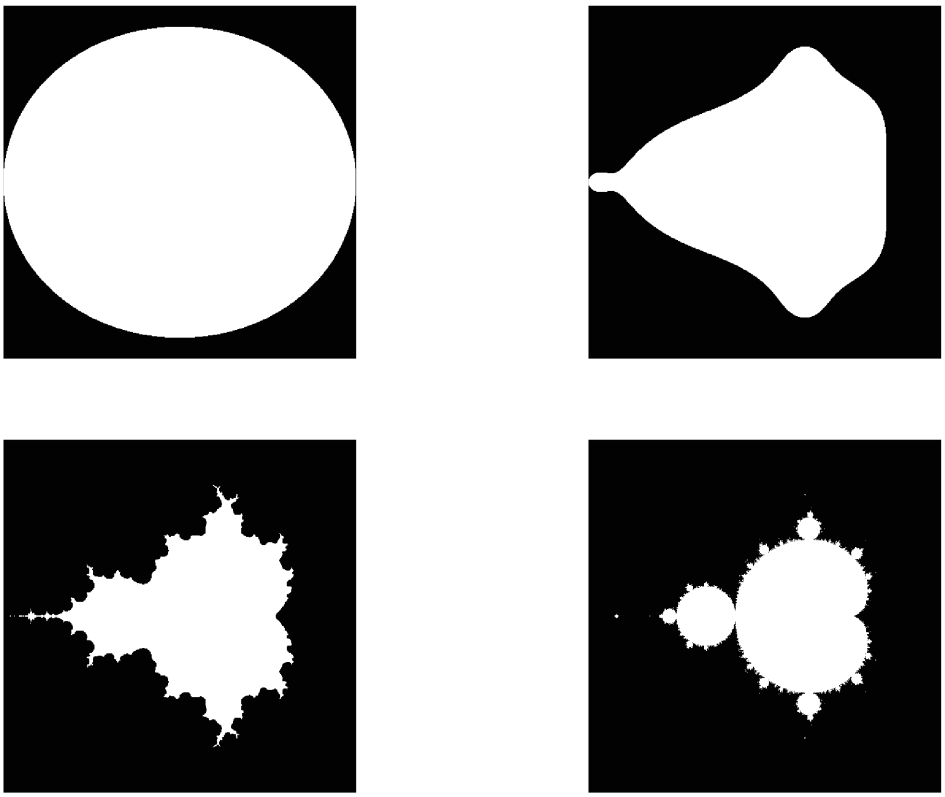


Figura 5.1: CM determinats segons el nombre d'iteracions

3. Feu un bucle i calculeu el valor de z_{n+1} per cada iteració n; utilitzeu `z = z**2 + c` i apliqueu el criteri de convergència: si aquest es verifica en el punt considerat, CM val 1, altrament aquest serà 0 (feu, per exemple, `CM = np.abs(z) < 2`).

4. Representeu la matriu CM obtinguda amb `plt.imshow(CM, cmap='gray')`. Genereu una figura que contingui el CM per a diverses iteracions (**figura 1**).

5. Observeu els avisos que apareixen a la línia d'ordres. Python ens indica que no és capaç de calcular `z=z**2+c` en alguns punts, ja que se supera el màxim valor possible. Per saber quin és aquest valor, feu:

   ```
   In[1] import sys
   In[2] sys.float_info.max
   ```

   Pyhton marca aquests valors com `NaN` però, d'altra banda, `plt.imshow` mostra els NaN com si fossin zero. A conseqüència d'això el resultat de mostrar el CM sembla que sigui correcte, però en realitat generem dos errors que es compensen. Penseu que podríeu fer per evitar l'aparició dels `NaN`s.

6. Pels valors c fora del CM, compteu el nombre d'iteracions necessàries per verificar que no es compleix la condició de convergència en el punt c. Aquest concepte rep el nom de velocitat de convergència; denominarem VC l'array 2D corresponent. Representeu VC amb `plt.imshow(VC, cmap='jet')` [2]. Mostreu una figura amb les VC després de diverses iteracions (**figura 2**) [3].

[2] Podeu consultar els mapes de color disponibles a `http://matplotlib.org/examples/color/colormaps_reference.html`.

[3] És possible que la informació continguda a la secció 5.3 us sigui d'utilitat aquí.

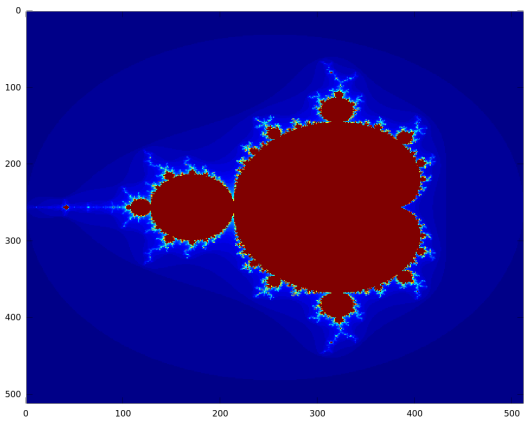


Figura 5.2: CM acolorit representat la velocitat de convergència

7. Implementeu fractals del conjunt de Julia. Per exemple $z_{n+1}(c) = z_n^2(c) - 0.7269 + 0.1889i$, amb $z_0(c) = c$. A `https://en.wikipedia.org/wiki/Julia_set`, trobareu molts exemples interessants. Genereu dues noves figures (**figures 3 i 4**) que continguin el CM per diferents nombres d'iteracions i la VC final.

8. OPCIONAL: Si guardeu en fitxer les imatges generades en cada iteració (amb `plt.imsave`) podreu fusionar-les després en un fitxer `.gif` i visualitzar la seqüència.[4].

Presentació: el codi ha de generar 4 figures. No poden apareixer errors en la consola d'ordres. Important: aquest script NO ha de guardar imatges ni generar l'arxiu .gif.

5.2 Recepta pràctica 1: Més sobre `np.meshgrid` i representació de funcions de dos variables.

La funció `np.meshgrid` és molt útil per generar funcions de dues dimensions amb molt poc codi. Veiem un exemple: representem la funció $z = x^2 + y^2$ al voltant del punt (0,0).

```
#!/usr/bin/env python3
#-*- coding: utf-8 -*-

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

x = np.linspace(-5, 5, 10)
y = np.linspace(-5, 5, 10)
X, Y = np.meshgrid(x, y)
```

[4]Consulteu `https://stackoverflow.com/questions/753190/programmatically-generate-video-or-animated-gif-in-python/10376713` per veure un exemple d'utilització de `imageio.mimsave`. Fixeu-vos que la solució que es proposa és pseudocodi i cal interpretar-lo: és a dir, dona la idea, però no es pot utilitzar directament

```
z = X**2 + Y**2

fig = plt.figure()
ax = plt.gca(projection='3d')
ax.plot_surface(X, Y, z)
```

Es generarà una finestra interactiva (permet canviar el punt de vista) que mostrarà la funció $z = x^2 + y^2$ en 3D.

5.3 Recepta pràctica 2: Com omplir dues figures simultàniament.

Normalment, quan volem mostrar informació en una finestra, invoquem una ordre `plt.figure` i a continuació donem totes les ordres gràfiques de forma seqüencial. En aquesta tasca pot resultar necessari obrir i omplir simultàniament dues finestres (MC i VC). El codi d'exemple que es mostra a continuació suggereix com fer-ho.

```
#!/usr/bin/env python3
#-*- coding: utf-8 -*-

import numpy as np
import matplotlib.pyplot as plt
import numpy.random as rnd

M1 = rnd.random([256, 256])
M2 = rnd.random([256, 256])

fig1 = plt.figure(1, figsize=(6,6))
fig2 = plt.figure(2, figsize=(6,6))
fig1.suptitle('Cas 1', fontsize=12)
fig2.suptitle('Cas 2', fontsize=12)

ax1 = fig1.add_subplot(221, xlabel='Eix x', ylabel=' Eix y', title='Subplot Fig1')
cf = ax1.imshow(M1, cmap = 'gray')
fig1.colorbar(cf, ax=ax1, shrink=.8)

ax2 = fig2.add_subplot(221, xlabel='Eix x', ylabel=' Eix y', title='Subplot Fig2')
cf = ax2.imshow(M2, cmap = 'gray')
fig2.colorbar(cf, ax=ax2, shrink=.8)

ax3=fig1.add_subplot(212)
ax3.plot(M1[128, :])
ax3.axis('on')

ax4=fig2.add_subplot(212)
ax4.plot(M2[128, :])
ax4.axis('on')
```

Tasca 6

Matrius i *arrays* 3D: sistemes de múltiples capes dielèctriques.

Objectiu	Familiaritzar-se amb l'ús d'estructures multidimensionals Utilitzar funcions l'àlgebra lineal.
Funcions i ordres que s'introdueixen	`numpy.dot` `numpy.linalg.matrix_power`
Tècniques	Generació d'estructures tridimensionals Selecció de *sub-arrays* amb l'operador `b:e:s`

Normalment, els sistemes òptics estan recoberts per una pel·lícula dielèctrica per evitar reflexos. Treballar amb elements recoberts és especialment important en sistemes formats per múltiples components. En cada superfície que travessa el feix es produeix una reflexió que fa que la llum total a la sortida de l'instrument sigui una fracció de la llum incident. Els recobriments consisteixen en una pila de materials dielèctrics de gruix molt prim que es dipositen sobre la superfície del component òptic. Habitualment s'alternen capes d'índex de refracció alt i petit que tenen un gruix de $\lambda/4$, tal com indica la figura. Un exemple senzill de recobriment és el format per una única capa de gruix $d = \lambda_0/4n$ (λ_0 és longitud d'ona en el buit) feta amb un material d'índex n que verifica $n = \sqrt{n_s}$ sent n_s l'índex del substrat.

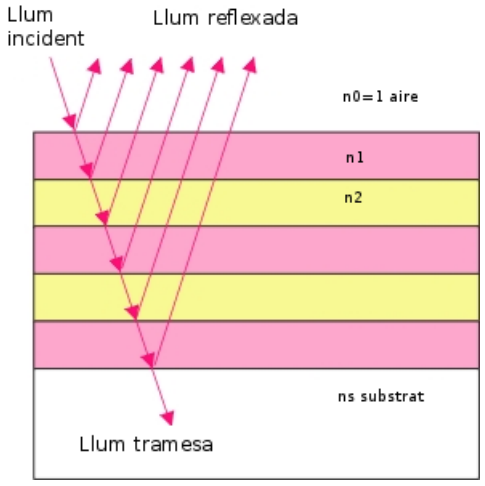


Figura 6.1: Sistema multicapes

En aquest projecte es proposa calcular la reflectància $R(\lambda)$ i la transmitància $T(\lambda)$ d'un sistema de capes primes que alternen materials d'índex de refracció alt i baix. Treballarem amb dos materials habituals en

tecnologia de capa prima: el diòxid de titani (TiO_2) amb índex $n_d = 2.48$ i el fluorur de magnesi MgF_2 amb índex $n_d = 1.37$ [1]. El substrat serà el vidre BK7, amb $n_d = 1.52$.

6.1 Fonamentació Física.

Assumint incidència normal, es pot demostrar (veieu per exemple [3, 4]) que cada capa es pot caracteritzar per la següent matriu

$$M = \begin{pmatrix} \cos\left(\frac{2\pi}{\lambda} n(\lambda) d\right) & \frac{i}{n(\lambda)} \sin\left(\frac{2\pi}{\lambda} n(\lambda) d\right) \\ i n(\lambda) \sin\left(\frac{2\pi}{\lambda} n(\lambda) d\right) & \cos\left(\frac{2\pi}{\lambda} n(\lambda) d\right) \end{pmatrix} \tag{6.1}$$

on d i $n(\lambda)$ són el gruix de la làmina i l'índex de refracció respectivament, i $i = \sqrt{-1}$. El gruix de les capes es determina a partir de la longitud d'ona λ_r (longitud d'ona de referència, per a la qual es dissenya la capa) per la qual es dissenya la capa (l'heu d'escollir vosaltres). Tal com s'ha comentat abans, els sistemes multicapes es produeixen apilant parells de capes que alternen un índex de refracció alt i un altre de baix. D'aquesta manera els gruixos de les capes queden determinats fent $d_1 = \dfrac{\lambda_r}{4n_1(\lambda_r)}$ i $d_2 = \dfrac{\lambda_r}{4n_2(\lambda_r)}$. Per un sistema de N capes, la matriu de transferència total es pot calcular com

$$M_T = M_1 M_2 \ldots M_{N-1} M_N \,. \tag{6.2}$$

Aleshores introduïm B i C

$$\begin{pmatrix} B \\ C \end{pmatrix} = M_T \begin{pmatrix} n_0 \\ n_s \end{pmatrix} \tag{6.3}$$

on $n_0 = 1$ i n_s són els índexs de l'aire i del substrat (vidre), respectivament. Finalment, R i T es calculen a partir de

$$R(\lambda) = \left| \frac{n_0 B - C}{n_0 B + C} \right|^2 \tag{6.4}$$

$$T(\lambda) = 1 - R(\lambda) \tag{6.5}$$

6.2 Tasques.

1. Implementeu una funció que retorni l'índex de refracció per cada longitud d'ona pels materials considerats (TiO_2, MgF_2 i BK7). Trobareu les relacions de dispersió corresponents a la pàgina web indicada al peu[2]. Fixeu-vos que és possible accedir al codi Python que descriu $n(\lambda)$ per cada material (no cal introduir-lo a mà). Representeu $n(\lambda)$ per als tres materials considerats (**figura 1**). El rang de longituds d'ona s'ha d'ajustar a l'interval on les formules es poden aplicar.

2. Implementeu una funció que calculi la matriu M per a cada capa i longitud d'ona.

3. Simuleu un sistema format per dues capes (TiO_2 i MgF_2, gruix $\lambda/4$ cada una) sobre vidre d'index n_s=1.52: determineu els gruixos de les capes (d_1 i d_2) i representeu $R(\lambda)$ i $T(\lambda)$. Afecta l'ordre de les capes al resultat?

4. Simuleu un sistema format per dues capes (TiO_2 i MgF_2, gruix $\lambda/4$ cada una) sobre vidre BK7. Representeu $R(\lambda)$ i $T(\lambda)$.

5. Incrementeu el nombre de blocs de dues capes alternes de TiO_2 i MgF_2 (fins a 4 o 5 dobles capes). Estudieu l'efecte sobre $R(\lambda)$ i $T(\lambda)$. Representeu-les gràficament (**figura 2**). Verifiqueu que aquest

[1] n_d=n(587 nm)
[2] `https://www.refractiveindex.info`

sistema actua com un filtre antireflectant per un rang important de longituds d'ona (vegeu exemple Fig. 6.2).

Finalment, tingueu present que els operadors $*$ i $**$ no són adequats per fer operacions amb matrius, ja que aquests fan el càlcul component a component. Per calcular M cal utilitzar el producte de matrius convencional. Les ordres `numpy.dot` i `numpy.linalg.matrix_power` us poden ser d'utilitat.

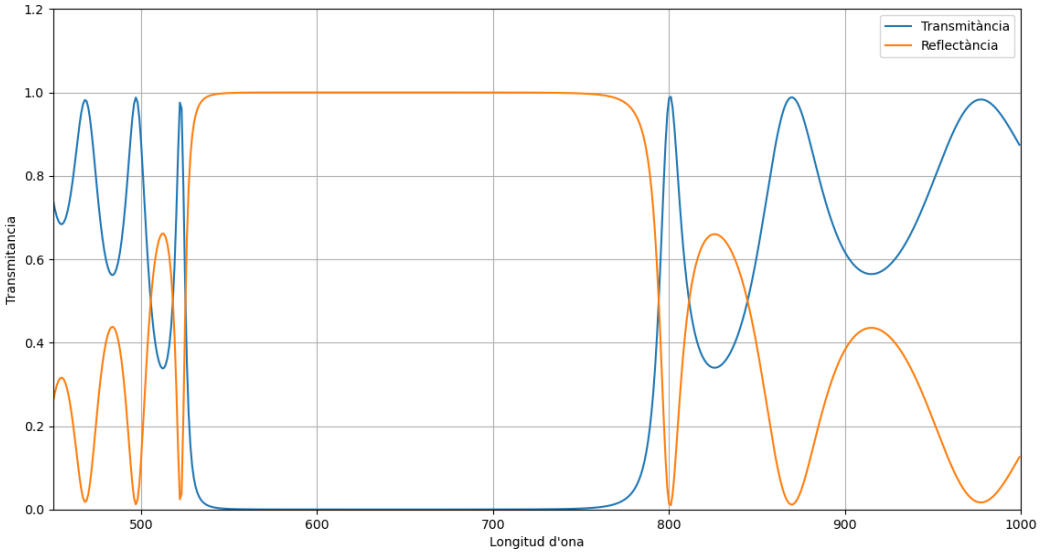


Figura 6.2: Simulació d'una làmina antireflectant de 10 capes)

Presentació: el codi ha de generar 2 figures. La primera (figura 1) ha de contenir $n(\lambda)$ pels tres materials considerats i la segona (figura 2) ha de mostrar $R(\lambda)$ i $T(\lambda)$ per un sistema multicapes.

Tasca 7

Transformades de Fourier: processament d'un senyal d'àudio.

Objectiu	Filtratge en freqüències d'un senyal Processament d'àudio

Funcions i ordres que s'introdueixen	`scipy.io.wavfile.read`, `scipy.io.wavfile.write`
	`numpy.fft.fft`, `numpy.fft.ifft`
	`numpy.fft.fftshift`, `numpy.fft.ifftshift`
	`scipy.signal.square`, `scipy.signal.sawtooth`

Tècniques	Transformades de Fourier en una dimensió Teorema del mostreig de Nyquist-Shannon Filtres de baixa freqüència

En aquest projecte es farà un filtratge de freqüències d'un arxiu d'àudio. Aquests arxius contenen la informació de l'ona sonora en funció del temps. Com que es tracta d'un format digital, la informació no és continua sinó que ha estat mostrejada. En el nostre cas particular, l'ona ha estat discretitzada a un ritme de 44100 mostres (*samples*) per segon. El concepte *nombre de mostres per segon* es denomina freqüència de mostreig. D'altra banda, els valors extrems que pot prendre l'ona tenen una profunditat de 16 bits. Recordeu que un sencer amb signe de 16 bits pot prendre valors en l'interval [-32768, 32767]. Aquest rang de valors es denomina amplada de banda (*bandwidth*). Aquesta combinació de valors (44100 HZ i 16 bits) és el que habitualment es coneix com a so de qualitat CD. Treballarem amb arxius no comprimits en format `.wav`. Python disposa del parell d'ordres (`scipy.io.wavfile.read` i `scipy.io.wavfile.write`) per carregar i guardar aquests arxius. En particular, la primera obre l'arxiu i carrega la seva informació en un *array*. En cada una de les seves components s'emmagatzema una mostra de l'ona. La longitud d'aquest *array* serà el nombre de mostres per segon per la durada. Equivalentment, la longitud del senyal en segons (L) serà el nombre de mostres de l'arxiu (N) per la durada de cada una d'elles (T), $L = NT$.

L'objectiu d'aquesta tasca és la manipulació i processament d'una pista d'àudio. En particular, farem servir un filtre de baixa freqüència (passa baixes) per eliminar el timbre que identifica l'instrument i convertir-lo en un so pur. Recordeu que una ona de so $g(t)$ pot ser descrita com a superposició d'ones de diferents freqüències (harmònics). Fer un filtratge de baixa freqüència significa eliminar les contribucions dels harmònics que oscil·len més de pressa i que són els que porten la informació dels detalls més subtils del so. La transformada de Fourier (TF) és l'eina matemàtica que permet fer aquesta descomposició.

$$G(f) = \mathrm{FT}[g(t)] = \int_{-\infty}^{\infty} g(t) \exp(-i2\pi f t) dt \tag{7.1}$$

on $G(f)$ és la distribució transformada de $g(t)$ i f és la freqüència [1]. $G(f)$ és una funció amb valors complexos, però el seu mòdul $G(f)$ informa del pes relatiu de l'ona que oscil·la amb freqüència f i que contribueix a descriure $g(t)$.

[1] Atenció: no confondre la freqüència de l'ona f amb la freqüència de mostreig $1/T$ (nombre de mostres per segon).

Python disposa d'una implementació d'aquest operador mitjançant l'ordre `numpy.fft.fft`. La transformada inversa es calcula mitjançant `numpy.fft.ifft`. Aquestes funcions tenen com argument el senyal d'àudio $g(t)$ i retornen un *array* amb el valor de la transformada. Les freqüències corresponents f s'han de calcular de forma independent i verifiquen

$$f \in \left[-\frac{1}{2T}, \frac{1}{2T}\right] = \left[-\frac{N}{2L}, \frac{N}{2L}\right] \tag{7.2}$$

on $f_t = 1/2T$ és la freqüència de tall o de Nyquist. Aquest resultat es coneix com el Teorema del Mostreig, enunciat per Shannon el 1949. Fixeu-vos que aquest teorema lliga la freqüència de l'ona amb la freqüència de mostreig que cal fer servir per descriure-la correctament. Si volem tenir informació de tot l'espectre audible (de 20 a 20000 Hz), actuarem de la següent manera: prenem $L = 1$s i com que la freqüència de mostreig és de 44100 Hz, aleshores $f_t = 22050$ Hz. Tenint present que l'espectre audible varia entre 20 i 20000 Hz, seleccionar una freqüència de mostreig de 44100 Hz és adequat. Per més informació podeu consultar [5].

Noteu que la funció `numpy.fft.fft` retorna les freqüències ordenades de 0 a f_t i de $-f_t$ a 0. Per visualitzar la transformada en l'ordre habitual $-f_t$ a f_t, necessitareu fer servir la funció `numpy.fft.fftshift`.

En aquest projecte desenvoluparem diverses tasques. D'una banda analitzarem conceptes relatius a l'anàlisi en freqüència que es poden estudiar amb la transformada de Fourier. Després, aplicarem aquests conceptes a l'estudi d'un senyal d'àudio i implementarem un filtratge de baixa freqüència.

7.1 Transformada de Fourier d'un senyal periòdic.

Genereu un senyal periòdic de freqüència 5 Hz amb l'ajut de les funcions `scipy.signal.square` i `scipy.signal.sawtooth`. Utilitzeu el teorema de Shannon per determinar l'interval de digitalització adequat si la longitud del senyal és 1 s. Analitzeu els resultats. Noteu que $G(f)$ és una magnitud complexa. Mostreu el mòdul de la transformada (feu servir `numpy.abs`). Aquesta gràfica us dóna informació del pes dels diferents harmònics en què es descompon l'ona.

Filtreu (elimineu) totes les contribucions d'alta freqüència superiors a 5 Hz i convertiu-les en senyals sinusoidals. En aquest cas el filtre serà un *array* d'igual nombre de components que $G(f)$ (o que $g(t)$). La informació que s'escriu en aquest *array* serà una funció rectangle d'amplada suficient l (rect(f/l)) que només deixi passar les contribucions de $G(t)$ per sota de 5 Hz.

Calculeu la transformada inversa de $G(t)$rect(f/l) i recupereu la versió filtrada de $g(t)$, $g_f(t)$. Les figures corresponents a l'ona quadrada i la dent de serra (**figures 1 i 2**, respectivament), han de mostrar quatre subplots: (i) $g(t)$, (ii) $|G(f)|$, (iii) $|G(t)|$rect(f/l) i (iv) $g_f(t)$ [2].

7.2 Creació d'un senyal d'àudio.

Genereu una ona sinusoidal de freqüència 440 Hz, digitalitzada a 44100 Hz i amb una durada d'uns pocs segons. Guardeu aquesta ona fent servir l'ordre `scipy.io.wavfile.write`. Escolteu-la. Heu generat un diapasó.

7.3 Filtratge de baixes freqüències.

1. Disposeu del fitxer `piano-la3.wav` al Campus Virtual[3]. Feu la transformada de Fourier d'aquest senyal. Fixeu-vos que la quantitat de mostres del fitxer és molt superior a la resolució de la pantalla. Quin problema representa això a l'hora de representar el fitxer?

2. Considereu un *sub-array* de 44100 dades (L=1), partir de $t = 0$. Calculeu i mostreu la transformada.

[2] Farem servir el mòdul de la transformada $|G(f)|$ a efectes de visualització. Els càlculs s'han de fer sempre amb tota la informació complexa de la transformada $G(f)$

[3] Fitxer generat amb l'ajut del paquet PySynth `https://github.com/mdoege/PySynth`

3. Elimineu la informació dels harmònics que superin la freqüència fonamental.
4. Calculeu la transformada inversa i guardeu el resultat en format `.wav`. Mostreu en un gràfic l'ona original, la filtrada i els espectres de la transformada de Fourier abans i després d'aplicar el filtre (**figura 3** - veure exemple a la Fig. 7.1).
5. Escolteu el fitxer original i el processat. Quines diferències aprecieu?

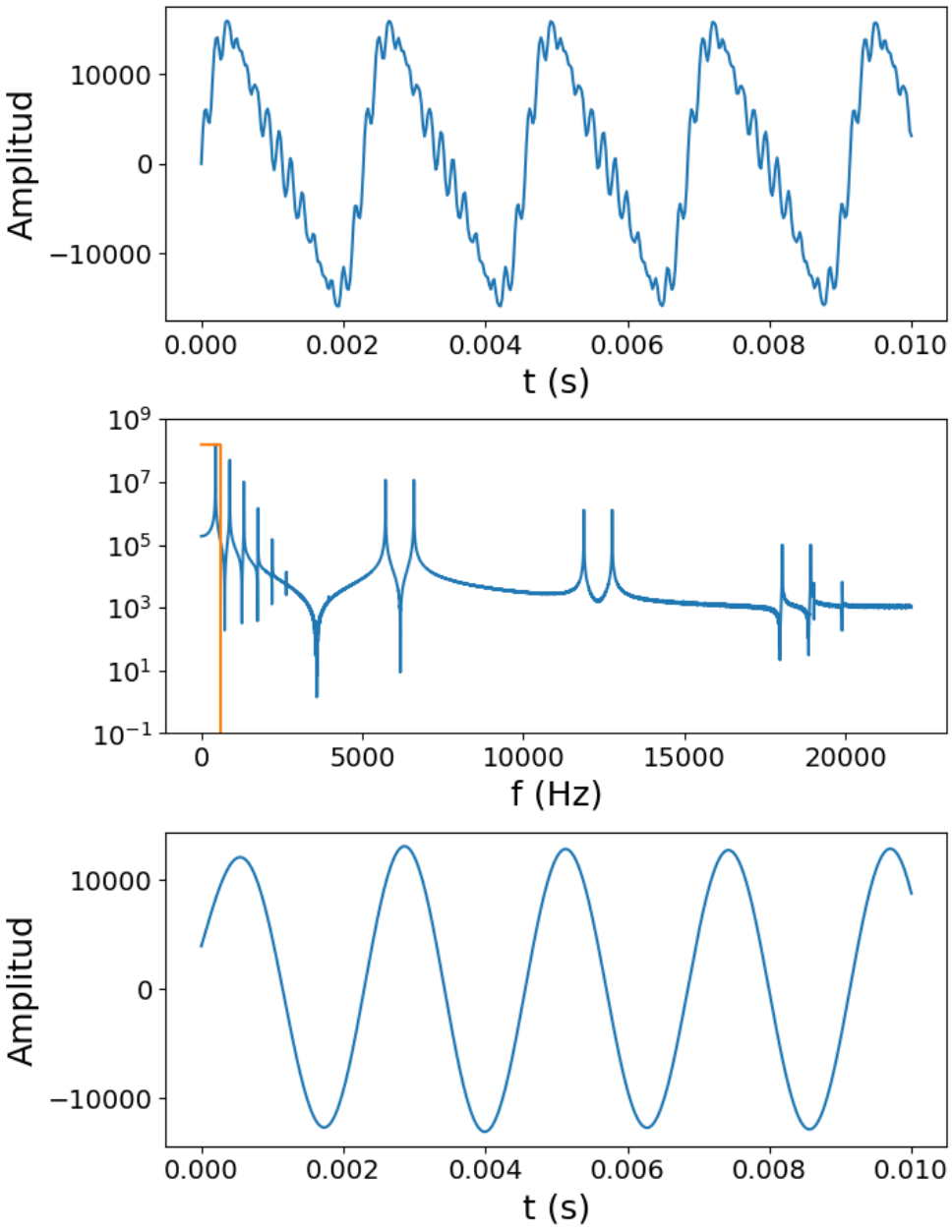


Figura 7.1: Efecte del filtratge de baixes freqüències sobre la forma de l'ona generada per un sintetitzador que simula la nota La3 (440 Hz) executada per un piano. El filtre actua a 600 Hz (línia taronja). Observeu la pèrdua d'harmònics després del filtratge

Presentació: el codi ha de generar 3 figures. No cal pujar l'arxiu d'àudio. Assegureu-vos que el vostre codi busca l'arxiu d'àudio en el directori de treball. No li canvieu el nom.

Bibliografia

[1] Tutorial scipy.integrate `http://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html`. [Online; accedit el 30 de setembre de 2016].

[2] J. Rahe. Komet Halley `https://ui.adsabs.harvard.edu/abs/1982S%26W....21...21R/abstract`, 1982. [Online; accedit 17 de setembre de 2018].

[3] H.A. Macleod. *Thin-film optical filters*. Taylor & Francis, 2001.

[4] James C. Wyant. Multilayer fims `http://wyant.optics.arizona.edu/ThinFilms/multilayerfilms.pdf`. [Online; accedit el 22 d'octubre de 2012].

[5] Wikipedia. Sampling (signal processing) — wikipedia, the free encyclopedia, 2017. [Online; accedit 13 de setembre de 2017].