TP_impedance_guide_etud

February 9, 2025

1 TP PH106 - mesure d'impédance et caractérisation de composants en Hautes fréquences

Ce TP a pour objectif de mesurer et modéliser l'impédance de composants passifs en hautes fréquences. En particulier vous mettrez en œuvre : • une mesure de paramètre S avec un VNA, • la conversion de cette mesure en impédance et le de-embedding de cette mesure dans le plan du dipôle mesuré, • la modélisation des parasites des composants passifs en hautes fréquences.

1.1 1. Prise en main de l'environnement de travail

Ce TP utilise le language *Python*, sans que vous ayez besoin de le maîtriser. Le fichier actuel est un *Jupyter Notebook* : il permet de méler du texte (écrit avec des balise markdown, si vous voulez utiliser cette syntaxe pour répondre aux questions, rendez vous sur Markdown Syntax) et de l'entrecouper de code *Python*. Par exemple, la cellule suivante importe les bibliothèques de code que nous allons utiliser, puis crée une variable 'instrument' qui permet d'accéder depuis le code vers l'appareil de mesure, le NanoVNA noté 'NVNA' (en utilisant la bibliothèque 'nvna').

NB: si il vous est demandé de choisir un environnement lorsque vous exécutez la première cellule Python, choisir pour ce TP 'nvna-env'

[ ]:
```
import nvna
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
%matplotlib inline

instrument = nvna.NVNA(connect=True)
```

1.1.1 Quelques éléments de syntaxe Python (au cas où)

Vous trouverez ci-dessous quelques éléments de syntaxe Python qui pourraient *éventuellement* vous être utiles.

Considérations générales :

- la valeur π peut être obtenue à l'aide de la commande `np.pi` ;
- le nombre imaginaire i tel que $i^2 = -1$ est noté `1j` en Python ;
- les indices démarrent à 0 en Python ;
- l'accès au contenu d'un dictionnaire `mydict` pour l'entrée `key` s'effectue par la commande `mydict[key]` ;

- les commentaires sont précédés de `#` (sur une ligne) ou encadrés par `"""` (sur plusieurs lignes) ;
- la syntaxe d'une boucle `for` (élémentaire) est la suivante : `python for element in iterable: # do something with element`
- un exemple de chaîne de caractères dynamique est `f"La valeur de a + b est {var_a + var_b}"` où `var_a` et `var_b` sont deux variables (le "f" en tête de chaîne indique d'évaluer les expressions entre accolades) ;
- l'indentation indique les blocs de code (à la manière des `{...}` en C) ;
- les puissances s'exprime par `**` (et non `^`) : `2**3 = 8` ;

Les fonctions mathématiques ou élémentaires dont vous pourriez avoir besoin sont disponibles en utilisant le préfixe `np.` :

- `np.abs` (valeur absolue) ;
- `np.angle` (argument de nombres complexes, en radians)
- `np.mean` (moyenne) ;
- `np.log10` (logarithme décimal) ;
- `np.linspace` (construction d'un vecteur de points équirépartis) ;
- etc.

Les fonctions utilisateur sont définies avec le mot clef 'def', suivi des arguments entre parenthèse suivi de ':'. La ou les valeurs renvoyées par la fonction sont précédées du mot clef 'return'.

Manipulation de tableaux :

- le dernier élément d'un tableau peut être obtenu avec l'indice -1, l'avant-dernier avec l'indice -2, etc. ;
- la sélection des `n` premiers éléments d'un tableau `arr` s'effectue avec la commande `arr[:n]` ;
- la sélection des `n` derniers éléments d'un tableau `arr` s'effectue avec la commande `arr[-n:]` ;
- la sélection des éléments `n_start` à `n_stop` (inclus) dans un tableau `arr` s'effectue avec la commande `arr[n_start:n_stop + 1]`.

Pour obtenir de l'aide sur une fonction `func`, utilisez les commandes `help(func)` ou `func?` dans une cellule de code. Ceci est valable même pour les fonctions personnalisées mises à votre disposition, pour lesquelles un effort de documentation a également fait.

1.1.2 Utilisation du NanoVNA - guide rapide

la bibliothèque nvna a été écrite dans le but de pouvoir utiliser le VNA (ici un NanoVNA) depuis python sans avoir à gérer la communication entre l'ordinateur et l'appareil.

Dans la cellule précédente avec le code *Python*, une variable appellée 'instrument' reçoit une instance d'un object logiciel de type *NVNA*. Vous allez utiliser cet instance comme point d'accès logiciel à l'appareil physique. En python, pour lancer des méthodes spécifiques à un objet, il suffit d'appeler la variable, suivie de '.' puis du nom de la méthode et entre parenthèse les éventuels arguments de la méthode.

Les quelques méthodes qui pourront vous être utiles sont :

- `PORT1_measurement` : lance une mesure et ne renvoie rien
- `get_last_PORT1_Scattering` : renvoie deux tableaux contenant les fréquences de mesure et le paramètre S mesuré,

- `get_last_PORT1_Impedance` : renvoie deux tableaux contenant les fréquences de mesure et l'impédance extraite du paramètre S et ramené dans le plan du DUT si une fonction de de-embedding a été fournie',
- `get_S11` : qui prend en arguments `f_start` la fréquence de départ , `f_stop` la fréquence d'arrivée et `n_points` et qui renvoie deux tableaux contenant les fréquences de mesure et le paramètre S mesuré, attention dans ce cas, il n'y a aucune calibration de prise en compte. Cette méthode change la bande de mesure et ne sera utilisée ici que dans la première mesure en exemple…
- `attach_Scattering2Impedance_converter`: prend en argument une fonction qui calcule l'impédance à partir du paramètre S (voir code à trou proposé)
- `attach_Zdeembed_converter`: prend en argument une fonction qui calcule l'impédance dans le plan du DUT à partir des impédances de calibration (voir code à trou proposé)

A titre d'exemple, pour réaliser votre première mesure dans le texte du TP pour l'activité 2, nous allons faire une mesure de paramètre S puis l'afficher, comme en activité 1 mais sur l'ordinateur cette fois (lisez et complétez)

[ ]:
```
f_start = 5e4   # in Hz
f_stop = 2.5e9    # in Hz
n_points = 201

freq, S11 = instrument.get_S11(f_start, f_stop, n_points)

plt.figure()
plt.plot(freq, np.abs(S11), label='S11')
plt.xlabel('A compléter')
plt.ylabel('A compléter')
plt.semilogy()
plt.legend()
```

Pour être sûr d'avoir bien compris, et en deux/trois phrases : que fait ce code ? (attention aux détails)

1.2 Réponse :

…….

Si vous commencez le TP, vous pouvez sauter le prochain paragraphe, il vous sera utile vers la fin du TP

1.2.1 Code proposé pour l'identification d'un modèle (activité 9)

Il s'agit d'identifier un modèle : de trouver les paramètres d'une fonction mathématique qui se rapproche le plus d'une mesure. Pour nous, l'équation est le module de l'impédance de l'électrode et nous allons utiliser une fonction qui minimisera la grandeur Q définie par :

$$Q = \sum_{k=1}^{N} \left(|Z_{mesure}(f_k)| - |Z_{mod}(f_k)| \right)^2$$

cette fonction est tilée de la bibliothèque 'scipy', et on utilisera directement 'curve_fit'. Vous avez à charge d'écrire la fonction réalisant $|Z_{mod}(f_k)|$. Vous pourrez utiliser le code suivant :

[ ]:
```
f_max = 1e9
f_min = 1e7

def Z_mod(freq, C, L, R):
    return NotImplementedError

freq = NotImplementedError
Z_mesure = NotImplementedError

## identification

imax = freq.searchsorted(f_max, 'right') - 1
imin = freq.searchsorted(f_min, 'right')
freq_ident = freq[imin:imax]
Z_mesure_ident = Z_mesure[imin:imax]
popt, pcov = curve_fit(Z_mod, freq_ident, Z_mesure_ident, p0=[1, 1, 1])
print(popt)

plt.figure()
plt.plot(freq, Z_mesure, label='mesure')
plt.plot(freq, Z_mod(freq, *popt), label='modèle')
plt.loglog()
plt.legend()
```

Vous pouvez remarquer que le code contient des zones à compléter (remplacer le *NotImplementedError*). Vous pouvez également noter les variables `f_max` et `f_min` qui permettent de limiter la bande de fréquence sur laquelle l'algorithme d'idenfification est lancé. Ces paramètres peuvent être intéressant si vous souhaitez mettre de côté une partie de la courbe (si il y a du bruit de mesure, ou si des phénomènes inexpliqués par le modèle sont visibles).

1.3 2. Première mesure et étalonnage dans le plan du DUT

1.3.1 2.1 Passage en impédance

Pour l'activité 3, vous pouvez compléter le code suivant (retirer et remplacer le 'NotImplementedError') :

[ ]:
```
def scattering2impedance(S, z0 = 50.):
    return NotImplementedError

instrument.attach_Scattering2Impedance_converter(scattering2impedance)

input('Brancher la charge Load du kit puis appuyez sur entrée')
instrument.PORT1_measurement()
freq, Z_m = instrument.get_last_PORT1_Impedance()


plt.figure()
```

```
plt.plot(freq, Z_m, label='mesure')
plt.loglog()
plt.legend()
```

1.3.2 2.2 Etalonnage ou de-embedding dans le plan du DUT

à partir d'ici, vous utilisez la carte PCB de mesure

[ ]:
```
# place pour le code de l'activite 4
```

1.4 Réponses activité 4:

Le code à trous suivant vous permet de réaliser l'activité 5:

[ ]:
```
def de_embed(Z_m, Z_short, Z_open, Z_load, z0=50.):
    num = 0
    denom = 1
    return NotImplementedError

instrument.attach_Zdeembed_converter(de_embed)

instrument.PORT1_calibration()
print('PORT1 calibré')


input('Brancher la terminaison Load du kit puis appuyer sur entrée')
instrument.PORT1_measurement()
freq, Z_DUT = instrument.get_last_PORT1_Impedance()

plt.figure()
plt.plot(freq, np.abs(Z_DUT),label='impedance DUT')
plt.loglog()
plt.grid()
plt.legend()
```