CCB-informatique

```
import math
%pylab inline
```

Question 1

```
def init_T(Tmax, dt):
    T = [0]
    t = 0
    while t < Tmax:
        t = t + dt # on vient de s'assurer que le dernier element
        # ajouté est < Tmax
        T.append(t)
    return T
```

```
init_T(1.3, 0.2) # hou la jolie erreur de float
```

```
[0, 0.2, 0.4, 0.6000000000000001, 0.8, 1.0, 1.2, 1.4]
```

```
init_T(1.2, 0.2)
```

```
[0, 0.2, 0.4, 0.6000000000000001, 0.8, 1.0, 1.2]
```

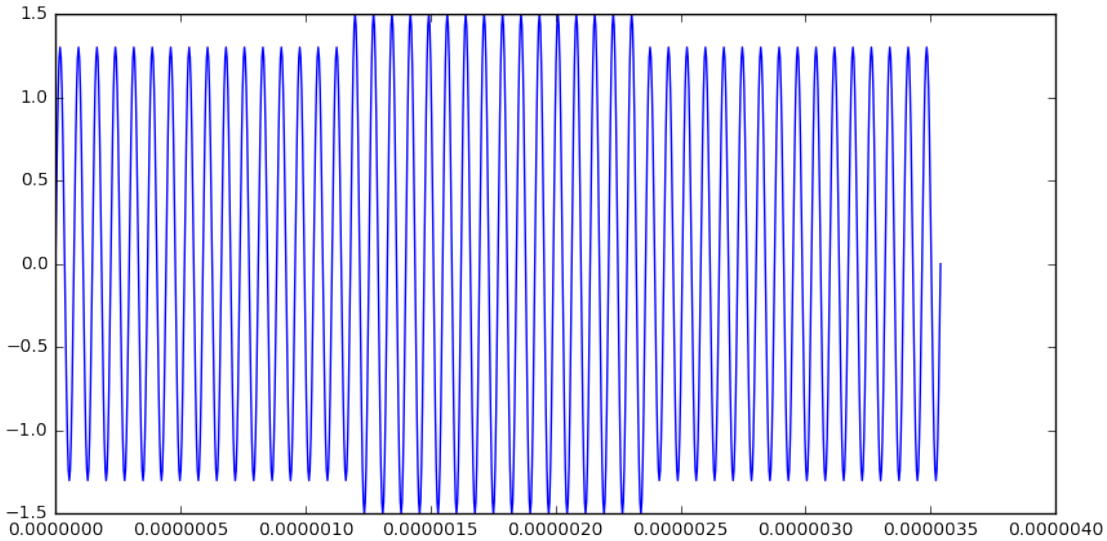
Question 2

```
def e0(t, f):
    if t <= 16/f:
        return 1.3
    if t <= 32/f:
        return 1.5
    return 1.3

def init_E(T, f):
    E = []
    for i in range(len(T)):
        E.append(e0(T[i], f) * math.sin(2 * math.pi * f * T[i]))
    return E

# version 2
```

```
def init_T2(T, f):
    E = []
    for t in T:
        E.append(e0(t, f) * math.sin(2 * math.pi * f * t))
    return E
#version 3
def init_T3(T, f):
    return [e0(t, f) * math.sin(2 * math.pi * f * t) for t in T]
```

```
f = 13.56e6
T = init_T(48/f, 1/(20*f))
E = init_E(T, f)
```

```
plt.xticks(np.linspace(0, 48/f, 7))
plt.figure(figsize=(10,5))
plt.plot(T, E)
```



Question 3

On a $\frac{ds}{dt}(t_i) \approx \frac{s(t_{i+1})-s(t_i)}{\Delta t}$.
Ainsi, grâce à l'équation différentielle de l'énoncé (appliquée en t_i), $s(t_{i+1}) = (1 - \frac{\Delta t}{\tau})s(t_i)$
On vérifiera sur la valeur $s(t_{i+1})$ obtenue si la diode change d'état.

Question 4

Ici il faudrait tester la valeur de $\frac{ds}{dt}(t_{i+1})$ pour savoir si la diode va changer d'état, ou encore connaître $s(t_{i+2})$ que l'on a pas avec la méthode précédente.

On prendra donc $s(t_{i+1}) = e(t_{i+1})$ et pour vérifier si la diode est bloquée on utilisera $\frac{\mathrm{d}s}{\mathrm{d}t}(t_{i+1}) \approx \frac{s(t_{i+1})-s(t_i)}{\Delta t}$ et on verifiera si cette quantité est supérieure à $-\frac{1}{\tau}s(t_{i+1})$.

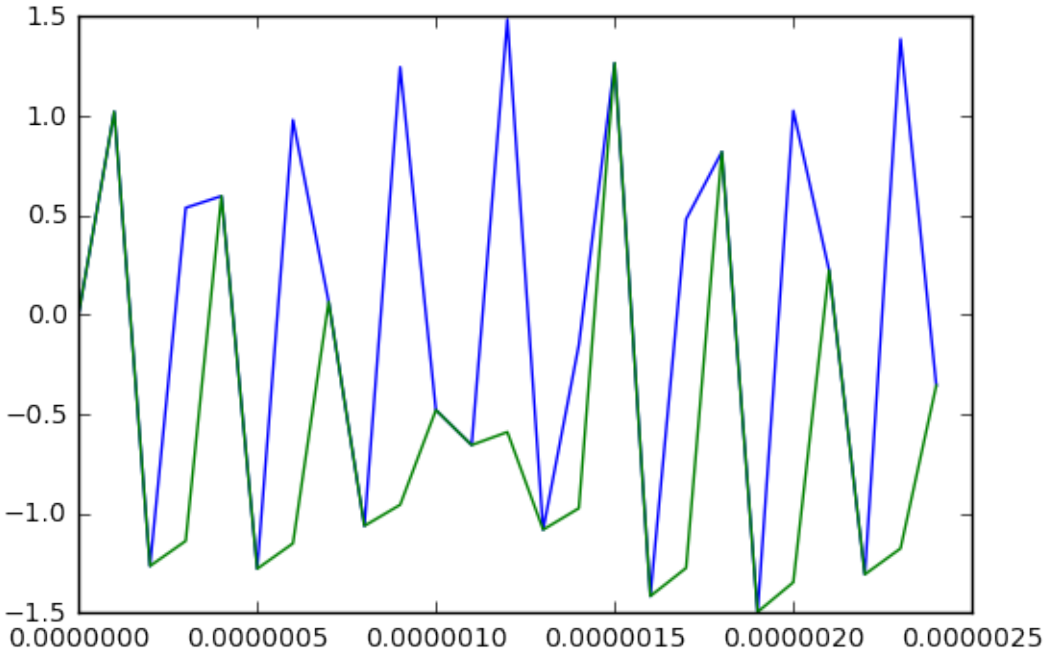
Question 5

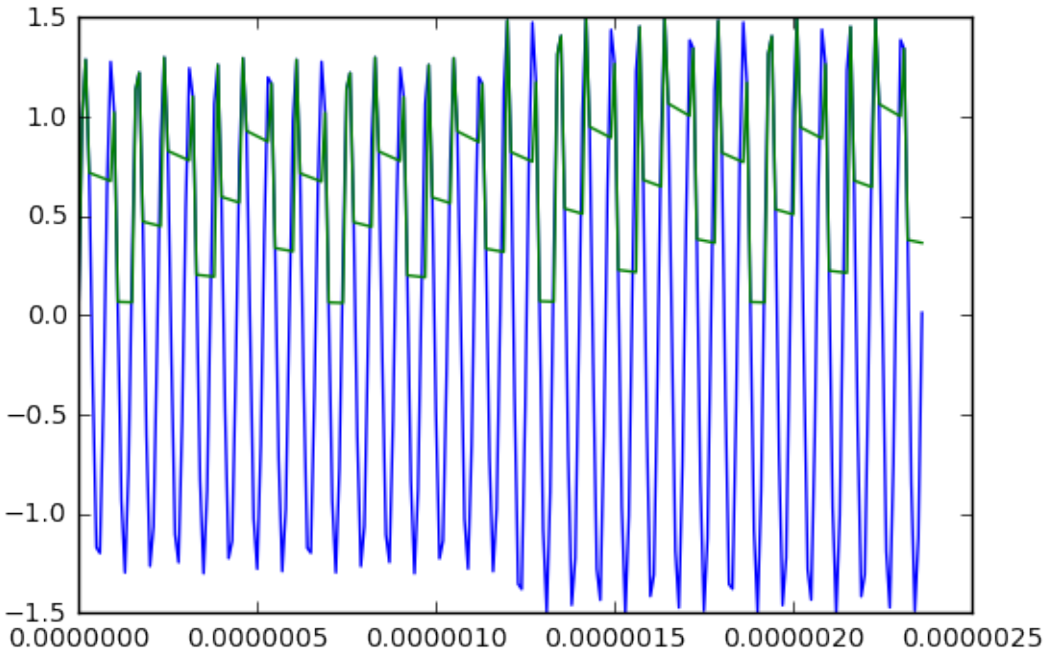
```
    def solve(T, E, tau):
        s = E[0]
        S = [s]
        passante = True
        n = len(T)
        for i in range(n-1):
            # on calcule s = s(t_{i + 1}) et on change l'etat de la diode
            if passante:
                s = E[i+1]
                b = (s - S[i]) / (T[i+1] - T[i]) + s / tau  # euler arrière
                if b <= 0:
                    passante = False
            else:
                s = (1 - (T[i+1] - T[i]) / tau) * S[i]  # euler explicite
                if s <= E[i+1]:
                    passante = True

            S.append(s)
        return S
```

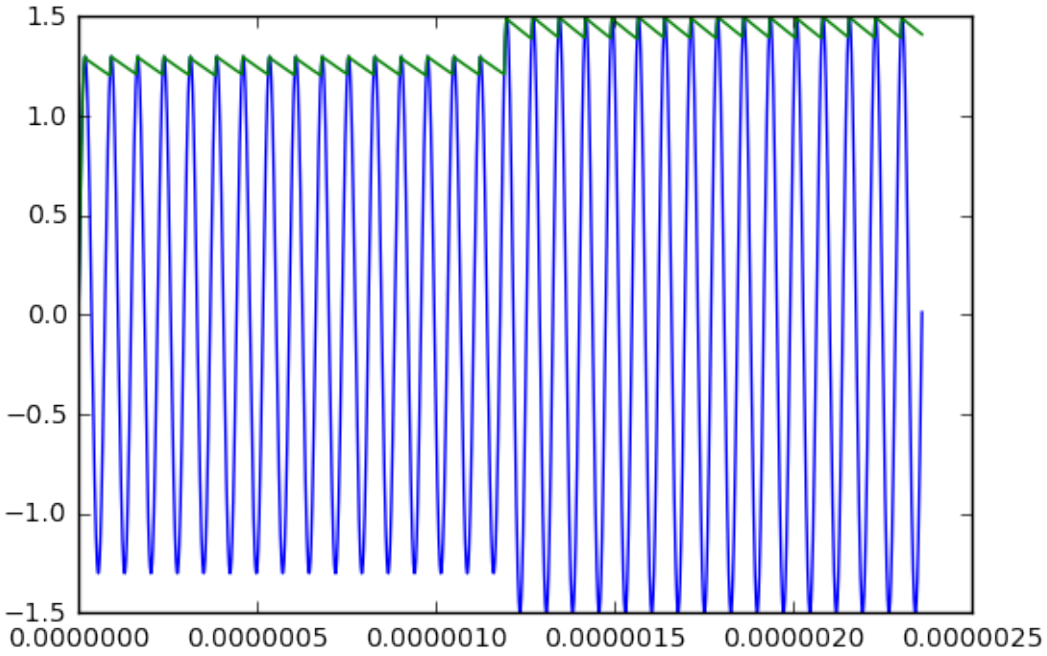
Testons cela

```
    def teste(liste_pas):
        tau = 1e-6
        f = 13.56e6
        Tmax = 32/f
        for pas in liste_pas:
            T = init_T(Tmax, pas)
            E = init_E(T, f)
            S = solve(T, E, tau)
            plt.plot(T, E)
            plt.plot(T, S)
            plt.show()
```

```
    teste([100e-9, 10e-9, 1e-9])
```


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.0000000
0.0000005
0.0000010
0.0000015
0.0000020
0.0000025


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.0000000
0.0000005
0.0000010
0.0000015
0.0000020
0.0000025



Question 6 et 7 et 8

6 La réponse est assez claire sur les graphiques précédents. On distingue les figures par la longueurs des segments de droite, qui est égale au pas de temps.

7 On repère un "saut" à chaque changement d'état : la sortie devient égale à l'entrée.

8 Il faut que le pas de temps soit d'un ordre de grandeur plus petit que la prériode pour que la valeur de sortie soit toujours proche de la valeur maximale courante et que l'on puisse distinguer les bits 0 et 1.

On peut distinguer un seuil seulement pour la dernière figure (seuil = 1.4V, toujours au dessus quand le bit est 1, toujorus en dessous quand il vaut 0). Pour les deux autres figures, certaines valeurs de sortie pendant la transmission de 0 sont au dessus de valeurs obtenues pendant la transmission du 1.

Question 9

5 s'écrit 101 en binaire, son bit de parité est donc 0
16 s'écrit 10000, son bit de parité est 1
37 s'écrit 100101, son bit de parité est 1

Question 10

```
def parite(bits):
    return sum(bits) % 2

# un petit rappel. On obtient la liste des bits en partant
```

```
    # du bit de poid faible
    def binaire(n):
        L = []
        while n != 0:
            L.append(n % 2)
            n = n // 2
        return L
```

```
    binaire(5), binaire(16), binaire(37)
```

```
([1, 0, 1], [0, 0, 0, 0, 1], [1, 0, 1, 0, 0, 1])
```

```
    parite(binaire(5)), parite(binaire(16)), parite(binaire(37))
```

```
(0, 1, 1)
```

Question 11

On veut par exemple transmettre la donnée 00 avec le bit de parité 0, mais on obtient 101 (une erreursur la donnée, une erreur sur le bit de parité), mais la vérification est correcte.

Même si une erreur est detectée, on ne peut pas savoir sur quel bit elle porte, ni si c'est sur 1 bit ou plus généralement sur un nombnre impair de bits.

Question 12

```
    def encode_hamming(donnee):
        d1, d2, d3, d4 = donnee
        p1 = parite([d1, d2, d3])
        p2 = parite([d1, d3, d4])
        p3 = parite([d2, d3, d4])
        return [p1, p2, d1, p3, d2, d3, d4]
```

```
    message = encode_hamming([0,1,1,1])
    message
```

```
[0, 0, 0, 1, 1, 1, 1]
```

Question 13

```
    def decode_hamming(message):
        c1 = parite(message[3:]) # attention aux indices
        c2 = parite(message[1:3] + message[5:7])
        c3 = parite([message[2*i] for i in range(4)])
        pos = c3 + 2*c2 + 4*c1
        if pos != 0:
            print("Une erreur s'est produit au bit", pos)
            message[pos - 1] = 1 - message[pos - 1]  # astuce pour changer un bit
        return [message[2]] + message[4:]
```

```
    # version2
    def decode_hamming(message):
        m1, m2, m3, m4, m5, m6, m7 = message
        # et on traduit l'énoncé
```

```
    decode_hamming(message)
```

```
[0, 1, 1, 1]
```

```
    message[2] = 1
    decode_hamming(message)
```

```
Une erreur s'est produit au 3 bit


[0, 1, 1, 1]
```

Question 14

```
    encode_hamming([1, 0, 1, 1])
```

```
[0, 1, 1, 0, 0, 1, 1]
```

```
    decode_hamming([1, 0, 1, 0, 0, 1, 1]) # cas des 2 premiers bits de la liste
```

```
Une erreur s'est produit au 3 bit


[0, 0, 1, 1]
```

On observe que dans le cas de plusieurs erreurs, la correction n'est pas fiable

Question 15

On peut penser à ajouter une 8ème bit de parité. La question précédente fait penser (mais est-ce vrai ?) qu'une double erreur va être detectée par nos 3 bits de contrôle.

Dans le cas d'une simple erreur, le bit de parité devient faux, dans le cas d'une double erreur il reste correct, mais l'erreur est detectée par nos bits de contrôle.

Question 16

id_titre est un entier, zones une liste de 2 entiers et date_fin une liste de 3 entiers

```
    ### pour le test
    lignes = '''\
    49987654
    1, 3, 2015-08-31
    2014-10-29, 08:34:15, 4568
    2014-10-28, 20:21:48, 365
```

```
2014-10-28, 18:47:54, 987
'''
lignes = lignes.split('\n')
```

```
passages = []

for i in range(2, 5):
    passage = lignes[i].rstrip('\n').split(',')
    date = passage[0].split('-')
    heure = passage[1].split(':')
    identifiant = passage[2]

    donnees = [
        int(date[0]),
        int(date[1]),
        int(date[2]),
        int(heure[0]),
        int(heure[1]),
        int(heure[2]),
        int(identifiant)]
    passages.append(donnees)
```

```
passages
```

```
[[2014, 10, 29, 8, 34, 15, 4568],
 [2014, 10, 28, 20, 21, 48, 365],
 [2014, 10, 28, 18, 47, 54, 987]]
```

Question 18

```
# NDT : étrange changement de convention pour les noms
# de fonctions...
def estAvant(date1, date2):
    annee = date1[0] - date2[0]
    mois = date1[1] - date2[1]
    jour = date1[2] - date2[2]
    # on s'inspire de la question suivante, en abusant sur le nombre de jour par mois
    return (annee * 31*12 + mois*31 + jour) <= 0

# version 2
def estAvant2(date1, date2):
    annee = date1[0] - date2[0]
    mois = date1[1] - date2[1]
    jour = date1[2] - date2[2]
```

```
        return annee <= 0 or (annee == 0 and mois <= 0) or (annee == 0 and mois == 0 and jour
```

Question 19

```
    def nbSecondesEntre(heure1, heure2):
        h = (heure1[0] - heure2[0]) * 60 * 60
        m = (heure1[1] - heure2[1]) * 60
        s = (heure1[2] - heure2[2])
        return h + m + s
```

```
    nbSecondesEntre([8, 30, 0], [9, 0, 12])
```

```
-1812
```

Question 20

```
    def testPassage():
        """
        Toutes les données dont on a besoin ont déjà été définies.
        """
        if id_titre in Liste_noire:
            print("Titre refusé")
            return False
        if Zone < zones[0] or Zone > zone[1]:
            print("Non valide dans cette zone")
            return False

        aujourdhui = Maintenant[:3]
        heure = Maintenant[3:]
        if estAvant(date_fin, aujourdhui):
            print("Titre expiré")
            return False
        for passage in passages:
            if passage[6] == Id_Point and passage[0:3] == aujourdhui:
                diff = nbSecondesEntre(heure, passage[3:6])
                if diff < 450:
                    print("Titre déjà validé")
                    return False
        return True
```

Question 21

Suivant le programme de base de données utilisé, il faudrai vérifier le format des dates...

```
    SELECT date, heure FROM passages
    JOIN points ON passages.id_point = points.id
```

```
    WHERE points.ligne = 1
    AND date >= "2014-07-01" AND date <= "2014-08-31"
```

Question 22

```
    SELECT count(*) FROM passages
    JOIN points ON passages.id_point = points.id
    JOIN titres ON passages.id_titre = titres.id
    WHERE date = "2014-12-31"
    AND (zone < zone_min OR zone > zone_max)
```