

Code TIPE

Nils Malmberg : n°196

1 Modules

```
[1]: import matplotlib.pyplot as plt
import numpy as np
```

2 Modélisation de la houle : sous la forme $\text{Acos}(\omega t)$

Modélisation de plusieurs houles (échelle de Beaufort)

Sources :

https://fr.wikipedia.org/wiki/%C3%89chelle_de_Beaufort

http://wikhydro.developpement-durable.gouv.fr/index.php/Houle_al%C3%A9atoire

Formule de Yoshimi Goda :

$$S_{max} = 11,25 \left(\frac{2\pi\omega_p U}{g} \right)^{-2,5}$$

Houle sous la forme : $h(t) = A_v \cos(\omega_v t)$

```
[2]: g = 9.81 #champ de pesanteur m.s^-2

Bft = np.array([1,2,3,4,5,6,7,8,9,10,11]) #tableau avec les indices de l'échelle
      ↳ de Beaufort

lignes = Bft.shape[0] #nombre de lignes de Bft

V = np.array([ np.round((np.sqrt((Bft[k]**3)*9)/3.6),2) for k in range(lignes)])
      ↳ #vitesse du vent en km/h

Av = np.array([0.1,0.35,0.75,1.5,2.5,3.5,4.75,6.5,8.5,10.75,13.75]) #Amplitude
      ↳ des vagues (données par l'échelle de Beaufort) en m

Pulsations = np.array([ np.round((9.81/(2*np.pi*V[k]))*((Av[k]/11.25)**(-1/2.
      ↳ ↳ 5))),2) for k in range(lignes)]) #pulsation en rad/s
```

```

Periodes = np.array([ np.round((2*np.pi)/Pulsations[k],2) for k in
    ↳range(lignes)]) #Periode des vagues en s

Vv = np.array([np.round(g*Periodes[k]/2*np.pi,2) for k in range(lignes)])
    ↳#Vitesse des vagues en m/s

tps = np.linspace(0, 120, 160) # temps en s

#Bft_complet [[Beaufort], [vitesse vent m/s], [amplitude vagues m], [vitesse
    ↳vagues m/s]
Bft_complet = np.c_[Bft, V, Av, Pulsations, Periodes, Vv] # matrice

print(Bft_complet)

```

```

[[1.00000e+00 8.30000e-01 1.00000e-01 1.24400e+01 5.10000e-01 7.86000e+00]
 [2.00000e+00 2.36000e+00 3.50000e-01 2.65000e+00 2.37000e+00 3.65200e+01]
 [3.00000e+00 4.33000e+00 7.50000e-01 1.07000e+00 5.87000e+00 9.04500e+01]
 [4.00000e+00 6.67000e+00 1.50000e+00 5.20000e-01 1.20800e+01 1.86150e+02]
 [5.00000e+00 9.32000e+00 2.50000e+00 3.10000e-01 2.02700e+01 3.12350e+02]
 [6.00000e+00 1.22500e+01 3.50000e+00 2.00000e-01 3.14200e+01 4.84170e+02]
 [7.00000e+00 1.54300e+01 4.75000e+00 1.40000e-01 4.48800e+01 6.91580e+02]
 [8.00000e+00 1.88600e+01 6.50000e+00 1.00000e-01 6.28300e+01 9.68180e+02]
 [9.00000e+00 2.25000e+01 8.50000e+00 8.00000e-02 7.85400e+01 1.21026e+03]
 [1.00000e+01 2.63500e+01 1.07500e+01 6.00000e-02 1.04720e+02 1.61368e+03]
 [1.10000e+01 3.04000e+01 1.37500e+01 5.00000e-02 1.25660e+02 1.93636e+03]]

```

```

[3]: def mod_houle2(Beaufort,temps):
    """fonction donnant une modélisation sinusoïdale de la houle grâce à
    ↳l'échelle de l'échelle de Beaufort"""
    Avague = Bft_complet[Beaufort-1][2]
    omegav = Bft_complet[Beaufort-1][3]
    #print(Avague)
    #print(omegav)
    return(Avague*np.cos(omegav*np.array(temps)))

```

```

[4]: t = np.linspace(0,120,10000)

plt.figure(figsize=(40,30))
plt.suptitle("Position d'un point matériel sur la vague (en m) en fonction du
    ↳temps (en s)",fontsize=50)

for i in range(1,lignes-1) :
    plt.subplot(4,3,i)
    houle2 = np.array([mod_houle2(i+1,X) for X in t])

    plt.plot(t,houle2, 'b', label="%d" %(i+1),linewidth=4)

```

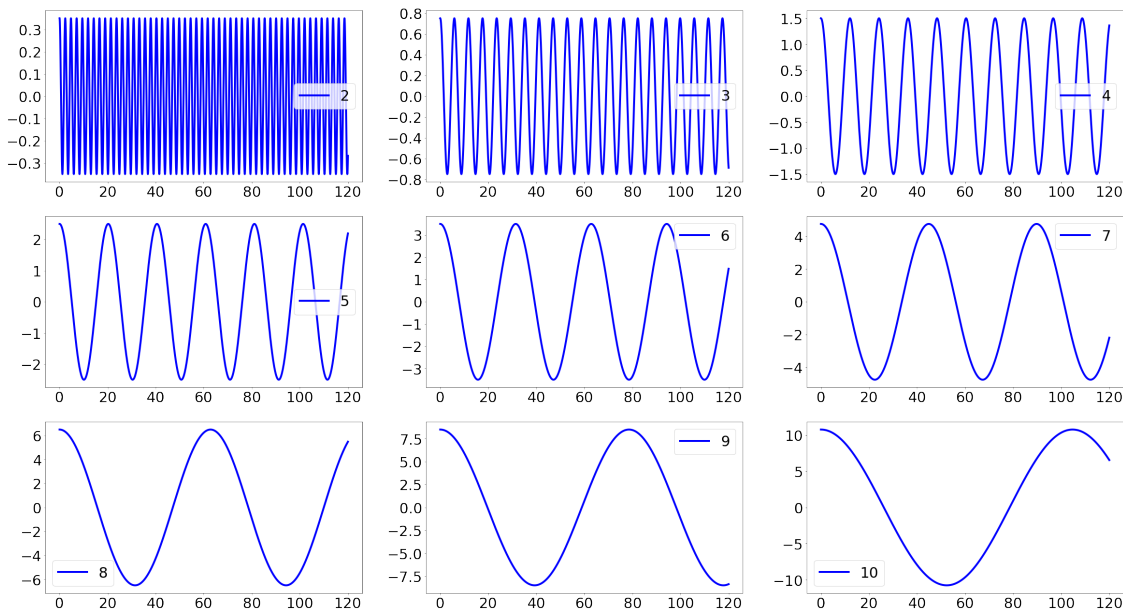
```
plt.xticks(fontsize=30)
plt.yticks(fontsize=30)

plt.legend(fontsize=30)

#plt.xlabel("temps en seconde")
#plt.ylabel("position d'un point matériel sur la vague en mètre")

plt.show()
```

Position d'un point matériel sur la vague (en m) en fonction du temps (en s)



3 Résolution numérique de l'équation différentielle du Searev

$$\ddot{\theta} + \frac{\omega_0}{Q} \dot{\theta} + \omega_0^2 \theta = \omega_0^2 h(t)$$

$$\text{Avec : } \omega_0^2 = \frac{1}{J_\Delta} \left(mg + 2k \frac{(X' - l_0)}{X'} \left[aR - \frac{d^2 R^2}{X'^2} + \frac{d^2 R^2}{X'(X' - l_0)} \right] \right)$$

$$\text{et } Q = \frac{J_\Delta}{I_\alpha^2} \omega_0$$

```
[5]: alpha = 4*(10**6) # coefficient de frottement
m = 400*(10**3) # masse en kg
g = 9.81 # accélération de pesanteur en m.s**-2
k = 1000 # constante de raideur de la seringue en N.m**-1
a = 10 # distance entre ... en m
```

```

R = 9 # rayon de la roue en m
d = 7 # distance entre ... en m
l = 1.35 # distance entre Delta et le centre d'inertie en m
Jdelta = m*(l**2) # somme(masses à différents points * distance entre ce point
→et le centre d'inertie**2) en kg.m**2
Xprime = np.sqrt(R**2 + a**2 + d**2 - 2*a*R) # en m
l0 = 3.23 # longueur à vide des ressorts en m

w0carre = (1/Jdelta)*(m*g + (2*k*(Xprime -l0)/Xprime)*(a*R - (((d*R)**2)/
→Xprime**2) + (((d*R)**2)/Xprime*(Xprime-l0))))
Q = (Jdelta/(alpha*(l**2)))*(np.sqrt(w0carre))

```

3.1 Tracé de la solution de l'équation différentielle, solution déterminée à la main

Rappel : $\ddot{\theta} + \frac{\omega_0}{Q}\dot{\theta} + \omega_0^2\theta = \omega_0^2 A_v \cos(\omega_v t)$

Avec : $\omega_0^2 = \frac{1}{J_\Delta} \left(mg + 2k \frac{(X' - l_0)}{X'} \left[aR - \frac{d^2 R^2}{X'^2} + \frac{d^2 R^2}{X'(X' - l_0)} \right] \right)$ et $Q = \frac{J_\Delta}{l^2 \alpha} \omega_0$

Solution : $\theta(t) = \frac{A_v}{\sqrt{(1-u^2)^2 + (\frac{u}{Q})^2}} \cos(\omega t + \varphi)$ avec $u = \frac{\omega}{\omega_0}$ et $\tan(\varphi) = \frac{u}{Q(u^2-1)}$

et $\dot{\theta}(t) = -\omega \frac{A_v}{\sqrt{(1-u^2)^2 + (\frac{u}{Q})^2}} \sin(\omega t + \varphi)$

```

[6]: def solution_eq_diff1(tps, Ampv, omegav):
    u = omegav/np.sqrt(w0carre)
    tanphi = u/Q*(-1+u**2)
    return (Ampv/np.sqrt((1-u**2)**2 + (u/Q)**2))*np.cos(omegav*tps + np.
→arctan(tanphi))

```

```

[7]: def derivee_solution_eq_diff1(tps, Ampv, omegav):
    u = omegav/np.sqrt(w0carre)
    tanphi = u/Q*(-1+u**2)
    return -(omegav*Ampv)/np.sqrt((1-u**2)**2 + (u/Q)**2))*np.sin(omegav*tps +
→np.arctan(tanphi))

```

```

[8]: tps = np.linspace(0, 120, 1000)

plt.figure(figsize=(40,30))
plt.suptitle(" Angle (rad) et vitesse de rotation (rad/s) en fonction du temps
→(s)", fontsize=50)

for i in range(1,lignes-1) :
    E = i+1
    table_theta = np.array([solution_eq_diff1(X,Av[i],Pulsations[i]) for X in
→tps ],dtype=float)
    table_omega = np.array([derivee_solution_eq_diff1(X,Av[i],Pulsations[i]) for
→X in tps ],dtype=float)

```

```

plt.subplot(4,3, i)
plt.plot(tps, table_theta, 'b', label='$\Theta(t) \sim \text{rad}$',linewidth=4)
plt.plot(tps, table_omega, 'r', label='$\omega(t) \sim \text{rad.s}^{-1}$',linewidth=4)

plt.xticks(fontsize=30)
plt.yticks(fontsize=30)

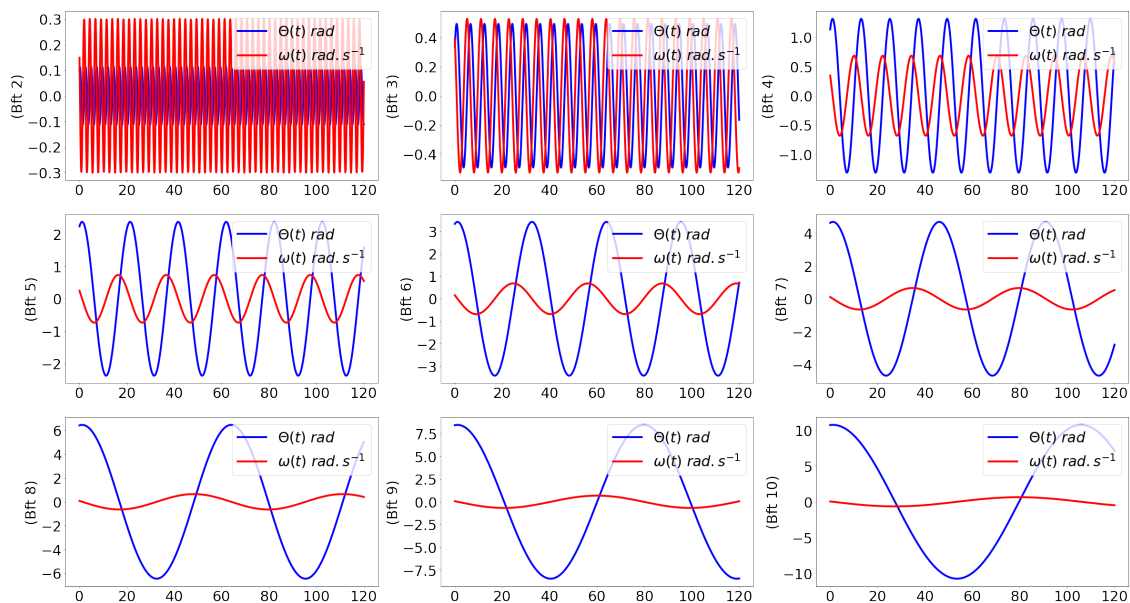
#plt.title('Echelle de Beaufort %i' %E)
#plt.xlabel('Temps en s')

plt.legend(loc = 'upper right',fontsize=30)
plt.ylabel('(Bft %i)' %E,fontsize=30)

plt.show()

```

Angle (rad) et vitesse de rotation (rad/s) en fonction du temps (s)



Ces courbes semblent cohérentes mais on observe aussi des pistes d'amélioration du système. En effet, à partir du numéro 4-5 sur l'échelle de Beaufort, il y a une perte importante au niveau de $\omega(t)$.

Solution : un frein électronique s'enclanchant afin de réguler et permettre d'augmenter $\omega(t)$?

4 Puissance générée en kW

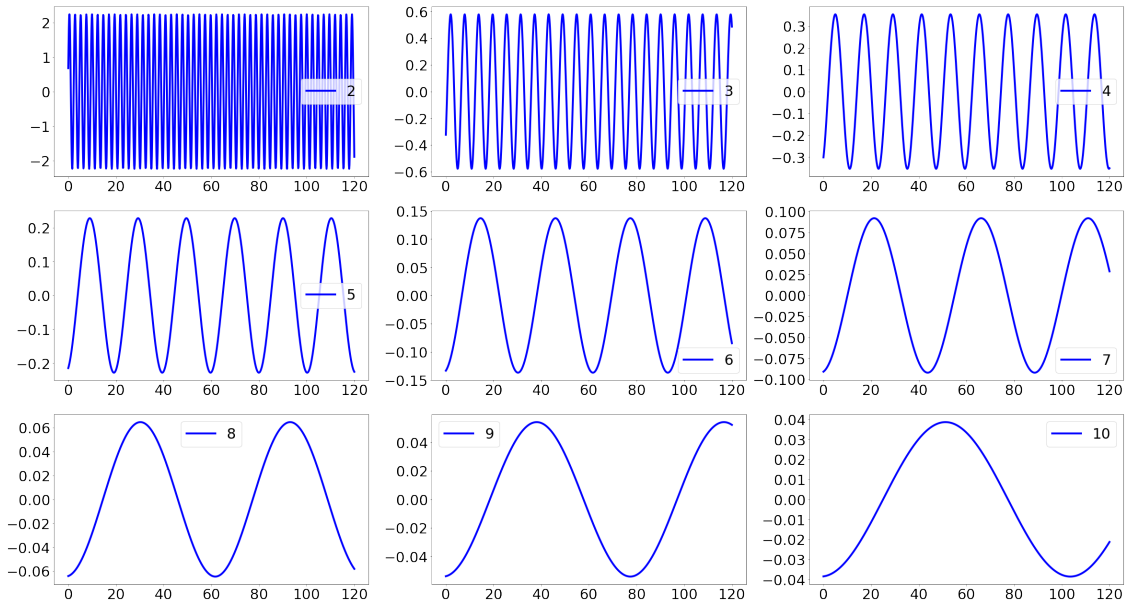
4.1 Calcul de l'accélération

$$\ddot{\theta} = \omega_0^2 A_v \cos(\omega_v t) - \frac{\omega_0}{Q} \dot{\theta} - \omega_0^2 \theta$$

```
[9]: def mod_houle3(tps,Ampv,omegav):  
      return Ampv*np.cos(omegav*np.array(tps))  
  
def derivee_seconde_solution_eq_diff1(tps,Ampv,omegav):  
    return w0carre*mod_houle3(tps,Ampv,omegav)-(np.sqrt(w0carre)/  
→Q)*derivee_solution_eq_diff1(tps, Ampv, omegev)-w0carre*solution_eq_diff1(tps,   
→Ampv, omegev)
```

```
[10]: tps = np.linspace(0, 120, 1000)  
  
plt.figure(figsize=(40,30))  
plt.suptitle('Accélération en fonction du temps',fontsize=50)  
  
for i in range(1,lignes-1) :  
    E = i+1  
    table_acceleration = np.  
→array([derivee_seconde_solution_eq_diff1(X,Av[i],Pulsations[i]) for X in tps_  
→],dtype=float)  
  
    plt.subplot(4,3, i)  
    plt.plot(tps, table_acceleration, 'b', label='%i' %E,linewidth=4)  
  
    plt.xticks(fontsize=30)  
    plt.yticks(fontsize=30)  
  
    plt.legend(fontsize=30)  
  
plt.show()
```

Accélération en fonction du temps



4.2 Expression et calcul de la puissance

$$P = C\dot{\theta} = J_{\Delta}\ddot{\theta}$$

```
[11]: tps = np.linspace(0, 120, 160)

plt.figure(figsize=(40,30))
plt.suptitle('Puissance produit en kW en fonction du temps en s',fontsize=50)

for i in range(1,lignes-1) :
    E = i+1
    table_puissance = np.
    →array([Jdelta*derivee_solution_eq_diff1(X,Av[i],Pulsations[i])*_
    →derivee_seconde_solution_eq_diff1(X,Av[i],Pulsations[i]) for X in tps_
    →],dtype=float)

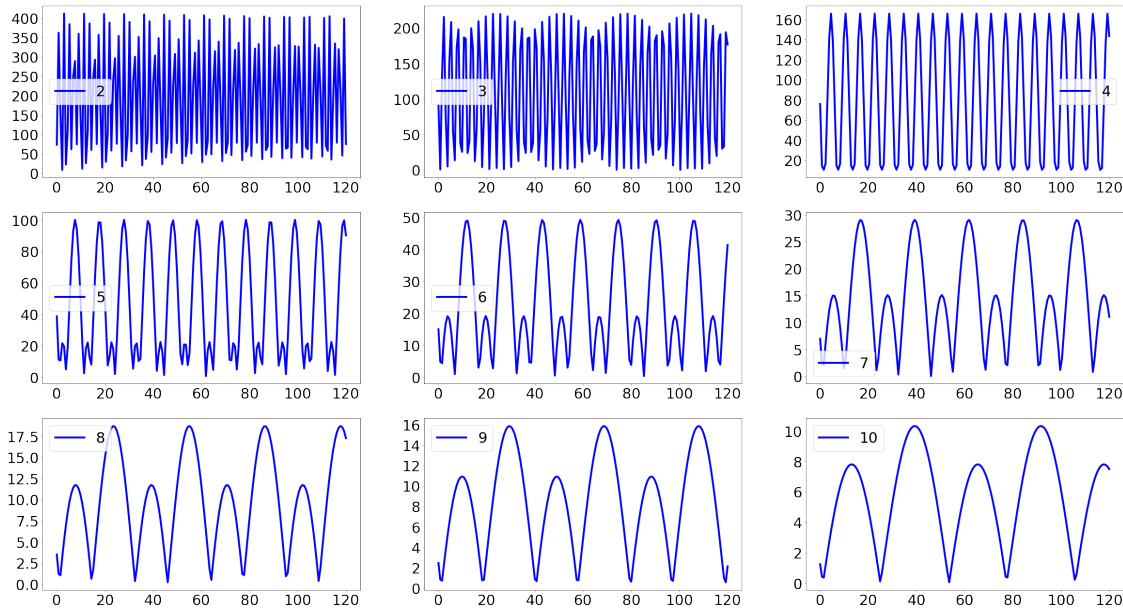
    plt.subplot(4,3, i)
    plt.plot(tps, abs(table_puissance)/1000, 'b', label='%i' %E,linewidth=4)

    plt.xticks(fontsize=30)
    plt.yticks(fontsize=30)

    plt.legend(fontsize=30)
```

```
plt.show()
```

Puissance produit en kW en fonction du temps en s



5 Rendement

5.1 Puissance en kW/m

$H_{1/3}$ est la hauteur significative des vagues définie comme la hauteur moyenne du tiers des vagues les plus grandes.

Si on observait les vagues en temps réel à un endroit donné, on observerait des amplitudes de vagues différentes. Ces différents relevés pourront être tracés dans un graphique qui donne la fréquence d'une hauteur donnée. A partir de ce graphique (on pourrait tracer la courbe de densité de la distribution des hauteurs des vagues), on sépare le graphique en 3 parts contenant le même nombre d'observations. On garde le tiers contenant les valeurs les plus grandes. Parmi ce tiers on prend la valeur moyenne.

Dans notre cas, nous avons étudié des vagues théoriques toutes identiques. La hauteur significative est donc confondue avec la simple hauteur.

“il est possible de calculer la puissance transportée par mètre de front d'onde, c'est-à-dire par mètre perpendiculaire à la direction de propagation des vagues. Exprimée en kW/m, elle est donnée par : $P \approx 0,4TH_{1/3}^2$ ” car $\frac{1}{2} \times \frac{\rho g^2}{32\pi} \approx 0.4$

Sources : pdf :

Les vagues qui animent la surface des océans du globe

5.1.1 Puissance maximale que la houle peut produire "idéalement" en kW/m

```
[12]: #Puissance maximale que l'on peut récupérer
def puissance_transportee(periode,amplitude):
    return 0.4*periode*(amplitude**2)

[28]: #On trace : la puissance maximale que peut produire la houle en kW/m en fonction
      ↳ de l'échelle de Beaufort

power_tab = []

for j in range(lignes) :
    power_tab.append(puissance_transportee(Periodes[j],Av[j]))

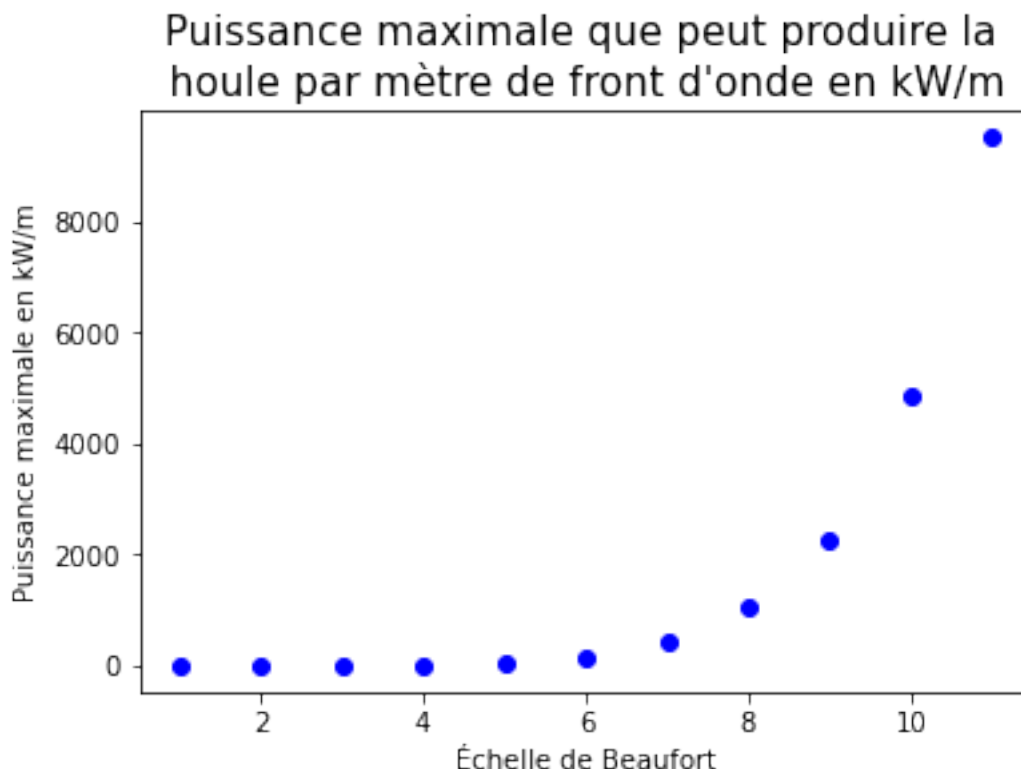
plt.scatter(Bft,power_tab,c = 'blue')

plt.xlabel("Échelle de Beaufort")
plt.ylabel("Puissance maximale en kW/m")
plt.title("Puissance maximale que peut produire la \n houle par mètre de front,
      ↳ d\'onde en kW/m ",fontsize = 15)

plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

plt.show()

print(power_tab)
```



```
[0.0020400000000000006, 0.11613, 1.32075, 10.872000000000002,
50.675000000000004, 153.95800000000003, 405.04200000000003, 1061.827,
2269.8060000000005, 4840.682000000001, 9503.0375]
```

5.1.2 Puissance générée par la modèle en kW/m

On connaît la puissance générée en 120s (courbes précédentes). On doit donc déterminer le nombre de mètre de front d'onde parcourue durant ces 120 secondes.

Pour toute fonction $f : [a, b] \rightarrow \mathbb{R}$ de classe C^1 , on note :

$$L(f) = \int_a^b \sqrt{1 + (f'(t))^2} dt$$

une expression intégrale de la longueur de la courbe représentative de f .

(Source : CCP Maths 1 PSI 2014)

```
[14]: def int_trapeze(f,a,b,n,Bft):
    h = (b-a)/n
    S = (f(a,Av[Bft-1],Pulsations[Bft-1])+f(b,Av[Bft-1],Pulsations[Bft-1]))/2
    for k in range(1,n):
        S += f(a+k*h,Av[Bft-1],Pulsations[Bft-1])
    return h*S
```

Houle sous la forme :

$$h(t) = A_v \cos(\omega_v t)$$

$$h'(t) = -A_v \omega_v \sin(\omega_v t)$$

```
[15]: def fct_sous_int(t,Ampv,omegav):
        return np.sqrt(1+(-Ampv*omegav*np.sin(omegav*t))**2)
```

```
[16]: def L(f,a,b,n,Bft):
        return int_trapeze(f,a,b,n,Bft)
```

```
[17]: #Test
Longueur = [L(fct_sous_int,0,120,10000,i) for i in Bft]

print(Longueur)
```

```
[157.81524159312144, 142.64241369347934, 137.50389338559611, 136.66960787544718,
136.55774980757187, 133.77675213487518, 132.04159540278354, 132.2550954237203,
132.6259213533725, 130.88623925537345, 133.70524751775017]
```

```
[18]: puissance_par_m = []

for i in range(lignes):
    table_puissance = np.
    →array([(Jdelta*derivee_solution_eq_diff1(X,Av[i],Pulsations[i])*
    →derivee_seconde_solution_eq_diff1(X,Av[i],Pulsations[i])) for X in tps
    →],dtype=float)
    puissance_par_m.append((sum(abs(table_puissance)/1000)/len(table_puissance))/
    →Longueur[i])

print(puissance_par_m)
```

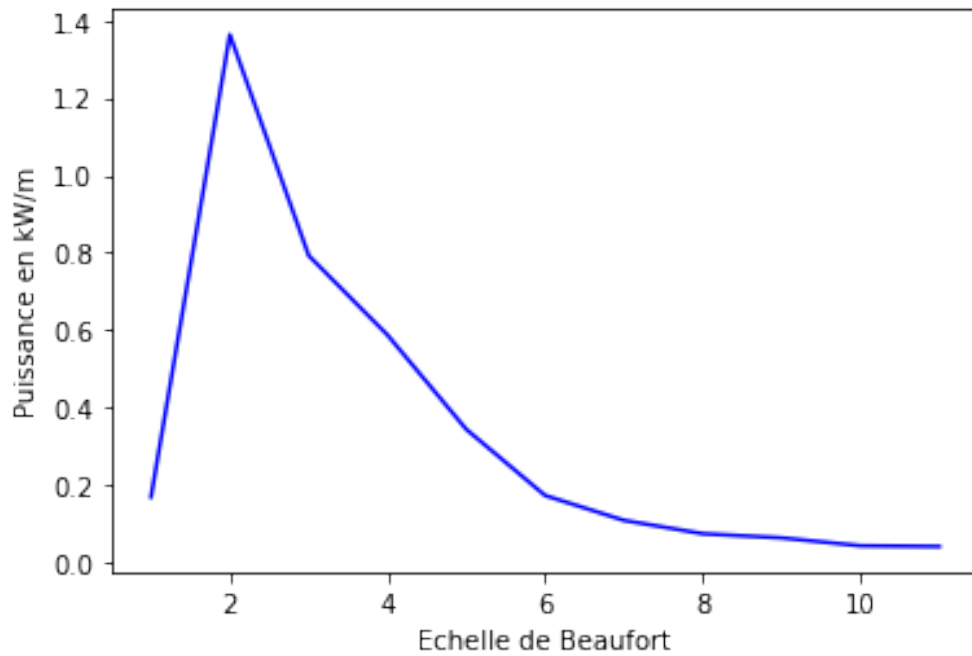
```
[0.1695519122795534, 1.3630950450364938, 0.7928980077928319, 0.5877077242637871,
0.3441935762268787, 0.17345773055659083, 0.10925280999016969,
0.07482766261017477, 0.06408173227064648, 0.04370322210945701,
0.041825846924582764]
```

```
[30]: plt.plot(Bft,puissance_par_m,'b')

plt.xlabel("Echelle de Beaufort")
plt.ylabel("Puissance en kW/m")

plt.title("Puissance en kW/m en fonction de l'échelle de Beaufort" , fontsize
    →=15)
plt.show()
```

Puissance en kW/m en fonction de l'échelle de Beaufort



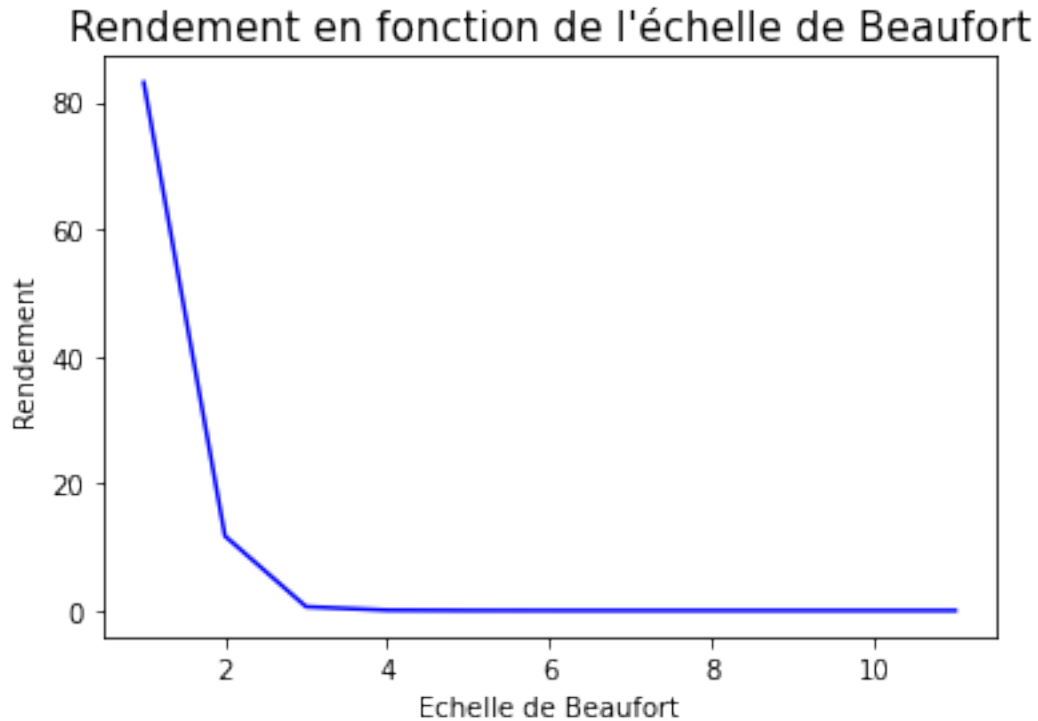
5.2 Calcul du rendement :

$$\eta = \frac{\text{Puissance utile}}{\text{Puissance fournie}}$$

```
[31]: plt.plot(Bft,np.array(puissance_par_m)/np.array(power_tab), 'b')

plt.xlabel("Echelle de Beaufort")
plt.ylabel("Rendement")

plt.title("Rendement en fonction de l'échelle de Beaufort" , fontsize = 15)
plt.show()
```



Problème : rendements > 1

Erreur non résolue à ce jour.

6 La partie suivante s'est avérée inutile car au début je pensais que $\theta(t)$ appartenait à l'intervalle $[-\frac{\pi}{2}; +\frac{\pi}{2}]$ car la roue ne faisait pas de tours complets or cela est faux car elle peut le faire.

```
[21]: def solution_eq_diff(tps, Ampv, omegav):
    u = omegav/np.sqrt(w0carre)
    tanphi = u/Q*(-1+u**2)
    var = (Ampv/np.sqrt((1-u**2)**2 + (u/Q)**2))*np.cos(omegav*tps + np.
    →arctan(tanphi))
    if var >= np.pi/2 :
        return(np.pi/2)
    elif var <= -np.pi/2 :
        return(-np.pi/2)
    else :
        return var
```

```
[22]: def derivee_solution_eq_diff(tps, Ampv, omegav):
    u = omegav/np.sqrt(w0carre)
```

```

    tanphi = u/Q*(-1+u**2)
    var = (Ampv/np.sqrt((1-u**2)**2 + (u/Q)**2))*np.cos(omegav*tps + np.
→arctan(tanphi))
    if var >= np.pi/2 :
        return 0
    elif var <= -np.pi/2 :
        return 0
    else :
        return -(omegav*Ampv)/np.sqrt((1-u**2)**2 + (u/Q)**2))*np.
→sin(omegav*tps + np.arctan(tanphi))

```

```

[23]: tps = np.linspace(0, 120, 160)

plt.figure(figsize=(40,30))
plt.suptitle(" Angle (rad) et vitesse de rotation (rad/s) en fonction du temps,
→(s)", fontsize=50)

for i in range(1,lignes-1) :
    E = i+1
    table_theta = np.array([solution_eq_diff(X,Av[i],Pulsations[i]) for X in tps,
→],dtype=float)
    table_omega = np.array([derivee_solution_eq_diff(X,Av[i],Pulsations[i]) for
→X in tps ],dtype=float)

    plt.subplot(4,3, i)
    plt.plot(tps, table_theta,'b', label='$\Theta(t)\sim\text{rad}$',linewidth=4)
    plt.plot(tps, table_omega,'r', label='$\omega(t)\sim\text{rad.s}^{-1}$',linewidth=4)

    #plt.title('Echelle de Beaufort %i' %E)
    #plt.xlabel('Temps en s')
    plt.ylabel('(Bft %i)' %E,fontsize=30)

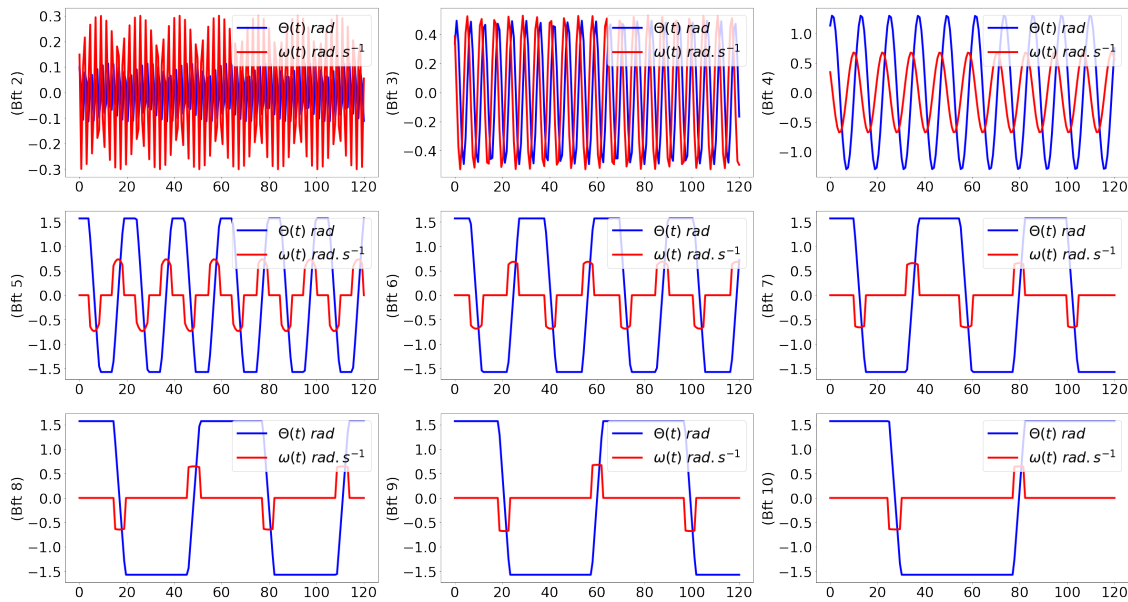
    plt.xticks(fontsize=30)
    plt.yticks(fontsize=30)

    plt.legend(loc = 'upper right',fontsize=30)

plt.show()

```

Angle (rad) et vitesse de rotation (rad/s) en fonction du temps (s)



```
[24]: def mod_houle3(tps,Ampv,omegav):
        return(Ampv*np.cos(omegav*np.array(tps)))

def derivee_seconde_solution_eq_diff(tps,Ampv,omegav):
    return w0carre*mod_houle3(tps,Ampv,omegav)-(np.sqrt(w0carre)/
    ↳Q)*derivee_solution_eq_diff(tps, Ampv, omeagav)-w0carre*solution_eq_diff(tps,
    ↳Ampv, omeagav)
```

```
[25]: tps = np.linspace(0, 120, 160)

plt.figure(figsize=(40,30))
plt.suptitle('Puissance produit en kW en fonction du temps en s',fontsize=50)

for i in range(1,lignes-1) :
    E = i+1
    table_puissance = np.
    ↳array([Jdelta*derivee_solution_eq_diff(X,Av[i],Pulsations[i])*
    ↳derivee_seconde_solution_eq_diff(X,Av[i],Pulsations[i]) for X in tps
    ↳],dtype=float)

    plt.subplot(4,3, i)
    plt.plot(tps, abs(table_puissance)/1000,'b', label='%i' %E,linewidth=4)

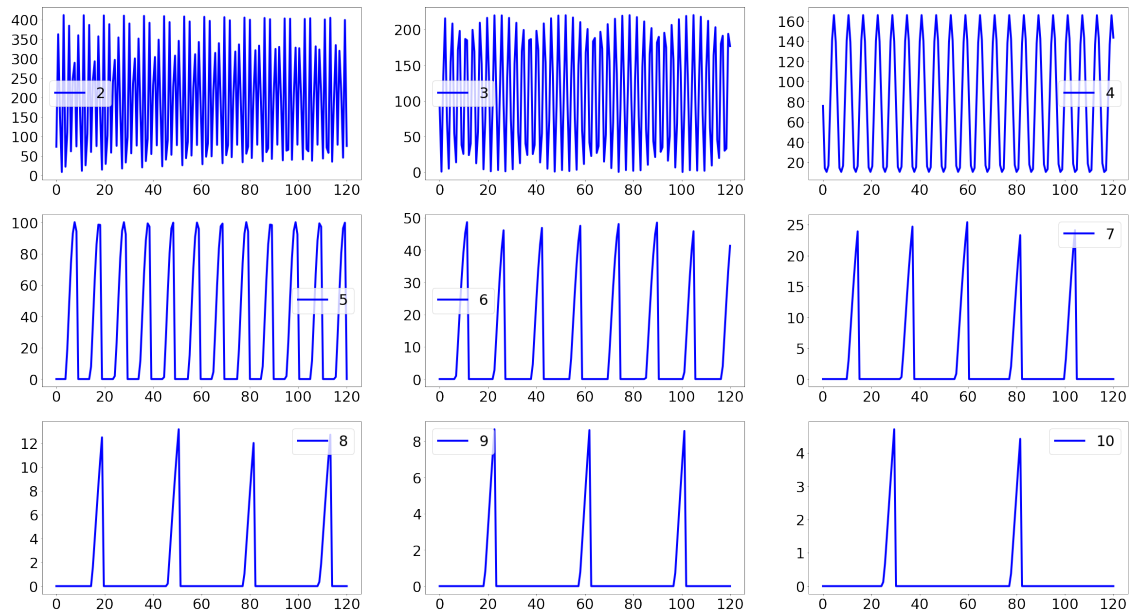
plt.xticks(fontsize=30)
```

```
plt.yticks(fontsize=30)

plt.legend(fontsize=30)

plt.show()
```

Puissance produit en kW en fonction du temps en s



Code python mesures.py

```
01 | # Module utile
02 |
03 | import serial
04 |
05 | # Préparatifs
06 |
07 | arduino_port = "COM3" # port USB
08 | baud = 9600 # nombre d'informations transmis par seconde
09 | fileName = "C:/Users/nilsm/Documents/CPGE/TIPE/Mesures_prototype/Données
en txt/R2 S1.txt" # emplacement et nom du fichier .txt à créer
10 | sample = 1000 # nombre de mesures a prendre
11 | print_labels = False
12 |
13 | # Connection à l'arduino
14 |
15 | ser = serial.Serial(arduino_port, baud)
16 | print("Connected to Arduino port:" + arduino_port)
17 |
18 | # Création du fichier .txt
19 |
20 | file = open(fileName, "a")
21 | print("Created file")
22 |
23 | line = 0 # On initialise le numéro de la ligne
24 |
25 | # Collection des données
26 |
27 | while line <= sample:
28 |     if print_labels:
29 |         if line == 0:
30 |             print("Printing Column Headers") # Affiche les en-têtes des
colonnes
31 |         else:
32 |             print("Line " + str(line) + ": writing...")
33 |             getData = str(ser.readline()) # collecte les valeurs
34 |             Data = getData[2:][: -5] # nouveau tableau sans les caractères
inutiles
35 |             print(Data) # affiche les valeurs collectées
36 |
37 | # Ecriture sur le .txt
38 |
39 |     file = open(fileName, "a") # ouvre le fichier .txt
40 |
41 |     file.write(Data + "\n") # écrit les valeurs \n pour retour à la ligne
42 |     line += 1 # on incrémente de 1 la ligne
43 |
44 | print("Data collection complete!")
45 |
46 | file.close() # fermeture du .txt
```

```

// initialisation des variables stockant les données des ports A0 et A1 + le temps

int sensor1 = A0;
int sensor2 = A1;
float MS;

// on donne une légende aux variables

String datalabel1 = "Tension1(V)";
String datalabel2 = "Tension2(V)";
String datalabel3 = "Temps(s)";
bool label = true; //

int data1, data2;

int freq = 1000; // enregistre les valeurs toutes les millisecondes

void setup(){
    Serial.begin(9600);
    pinMode(sensor1, INPUT);
    pinMode(sensor2, INPUT);
}

void loop(){

    MS = millis();

    // en dessous affiche les entêtes des colonnes du moniteur série
    while(label){
        Serial.print(datalabel3);
        Serial.print(",");
        Serial.print(datalabel1);
        Serial.print(",");
        Serial.print(datalabel2);
        Serial.println();
        label = false;
    }

    // data1 et 2 lisent les valeurs de sensor1 et 2
    data1 = analogRead(sensor1);
    data2 = analogRead(sensor2);

    Serial.print(MS*0.001); // MS en milliseconde => conversion en seconde
    Serial.print(",");
    //data1 et 2 entre 0 et 1023.0 tq 0 ~ 0V et 1023.0 ~ 5V
    Serial.print(data1*5.0 / 1023.0);
    Serial.print(",");
    Serial.println(data2*5.0 / 1023.0);

}

```