Surf Et Danger De Courant D'Arrachement

MOUACHA BASSOU

N°SCEI:30462
Tipe :Jeux et Sports

Encadré par: Mr. Essahel Said

2023-2024

SOMMAIRE:

Introduction:

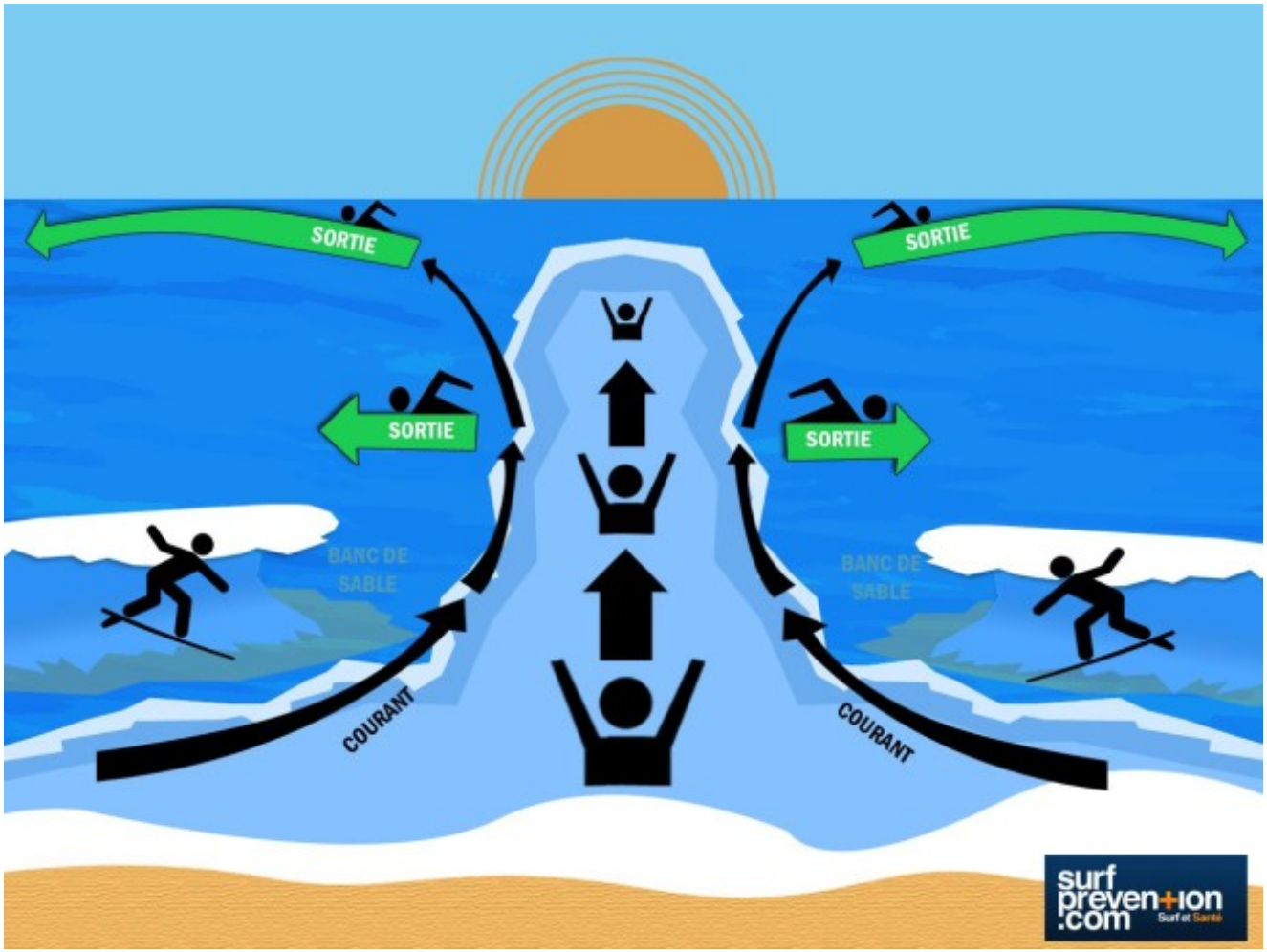


Figure: (1) Illustration des courants d'arrachement et leurs dangers sur les surfeurs

Source:https://blog.surf-prevention.com/2012/07/18/courant-de-baine/

Problématique:

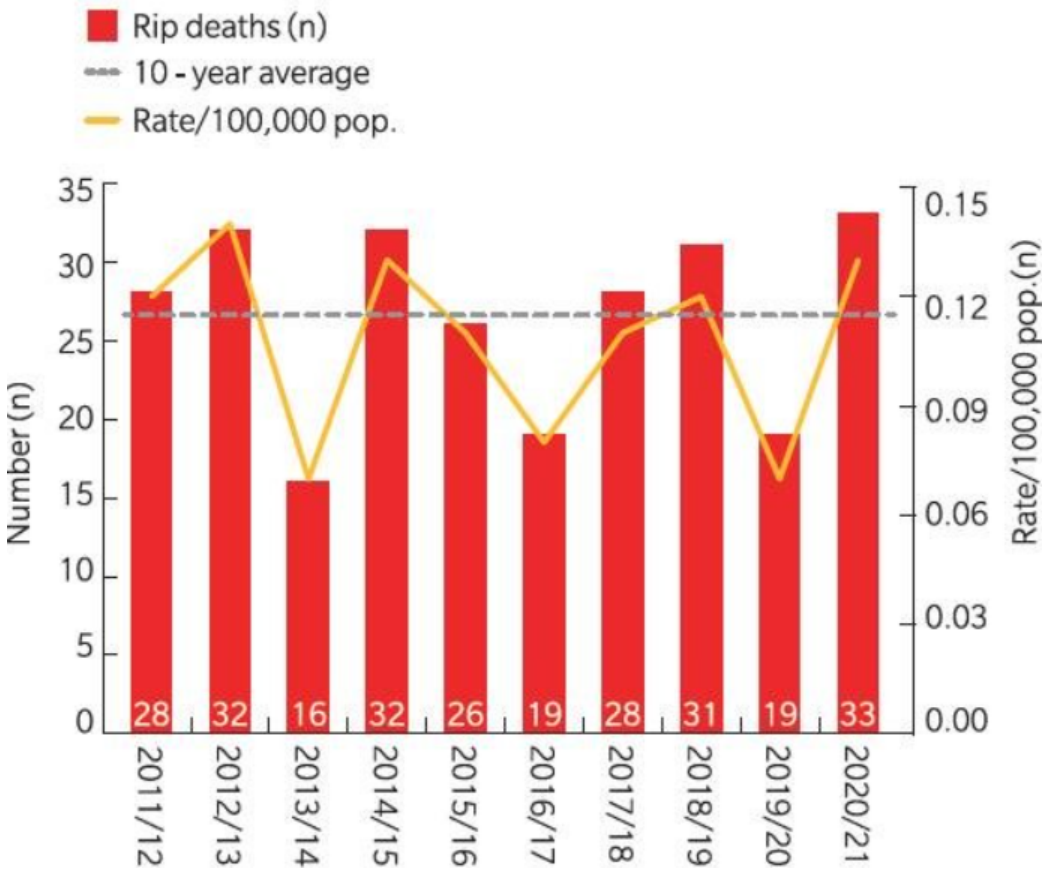


Figure: (2): Les statistiques des décès annuels en Australie à cause des courants d'arrachement

Source:https://www.scienceofthesurf.com/rip-current-safety

Problématique:

Quelle approche stratégique optimale d'un point de vue énergétique pour échapper aux courants d'arrachement, parmi les quatres stratégies d'échappement envisagées:

1. Nager directement vers la rive (S1)
2. Nager perpendiculairement au courant avant de se tourner vers la rive (S2)
3. Nager à un angle de 45 degrés vers la rive (S3)
4. Se laisser porter par le courant avant de nager vers la rive (S4)

Étude théorique:

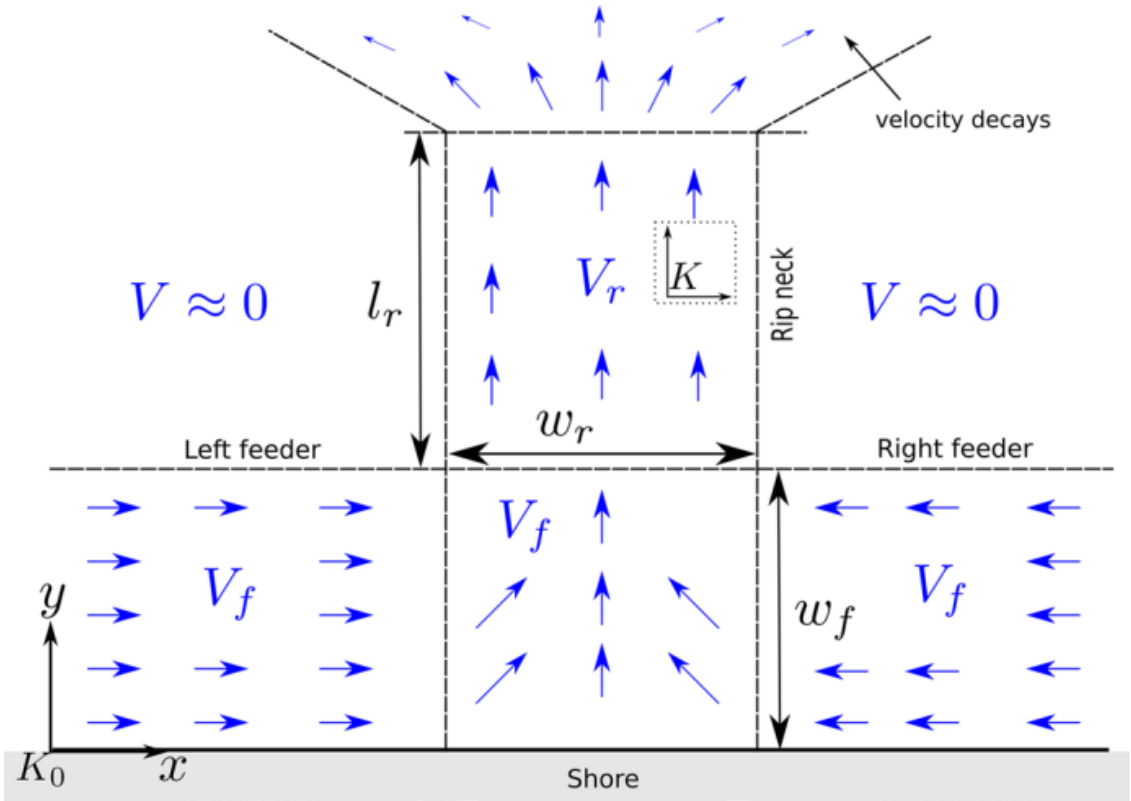


Figure: (3): Le modèle du courant d'arrachement adopté dans l'étude.

Cadre d'étude:

- La structure du courant d'arrachement a été simplifiée (figure 3)
- Nageur comme un objet rigide de géométrie arbitraire

Le Bilan des Forces:

- **La Force Auto – Générée** : $\vec{F_a}$
- **La Force de Traînée (induite par l'écoulement)** :

$$\vec{F_T} = \frac{1}{2}\rho A C_T \vec{V}^2 = \widetilde{C_T}\vec{V}^2$$

avec:

A: une aire caractéristique

$\mathbf{C_T}$: le coefficient de traînée du nageur

Relation Travail (W) - Puissance (P):

Considérons :

$$\vec{v}_{sK} = \vec{v}_s - \vec{v}_f$$

la vitesse du nageur par rapport au référentiel (K)
Avec:

- v_s : La vitesse du nageur par rapport à K_0
- v_f : La vitesse locale d'écoulement

Relation Travail (W) - Puissance (P):

L'expression du travail est:

$$\mathbf{W = \int_{t_i}^{t_f} F_a \cdot v_{sK}\, dt}$$

avec:

- F_a : La force Auto-générée
- v_{sK} :la vitesse de nageur par rapport a référence (K)

Relation Travail (W) - Puissance (P):

Par application du PFD :

$$\vec{F_a} = -\vec{F_T}$$

La nouvelle formule du travail devient :

$$\boxed{\mathbf{W} = \widetilde{\mathbf{C_T}} \int_{\mathbf{t_i}}^{\mathbf{t_f}} \left[(\mathbf{v_{sx}} - \mathbf{v_{fx}})^2 + (\mathbf{v_{sy}} - \mathbf{v_{fy}})^2\right]^{3/2} \mathrm{d}\mathbf{t}} \qquad (1)$$

avec:

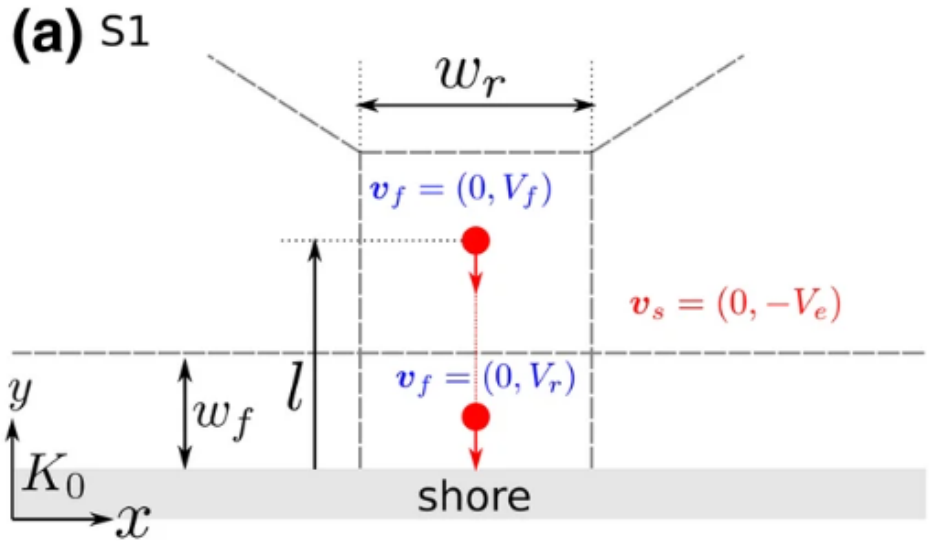
- $\widetilde{C_T} = \frac{1}{2}\rho A C_T$

Alors, La puissance est:

$$\boxed{\mathbf{P} = \frac{\mathbf{W}}{\mathbf{t_f} - \mathbf{t_i}}} \qquad (2)$$

Application:

- **Stratégie 1 (S1):**

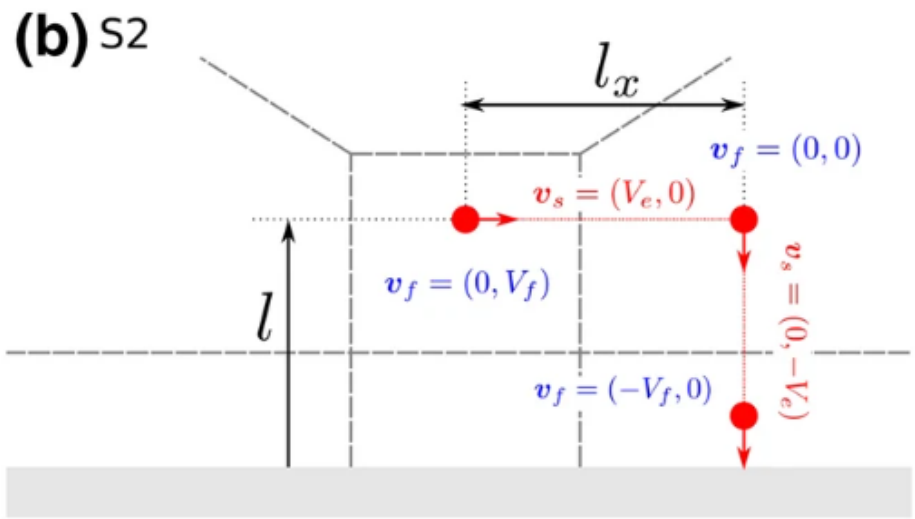


D'après (1) et (2), On a :

$$W_1 = \widetilde{C_T}\left[\frac{(l - w_f)}{V_e}(V_e + V_r)^3 + \frac{w_f}{V_e}(V_e + V_f)^3\right]$$

$$P_1 = \frac{\widetilde{C_T}}{l}\left[(l - w_f)(V_e + V_r)^3 + w_f(V_e + V_f)^3\right]$$

- **Stratégie 2 (S2):**

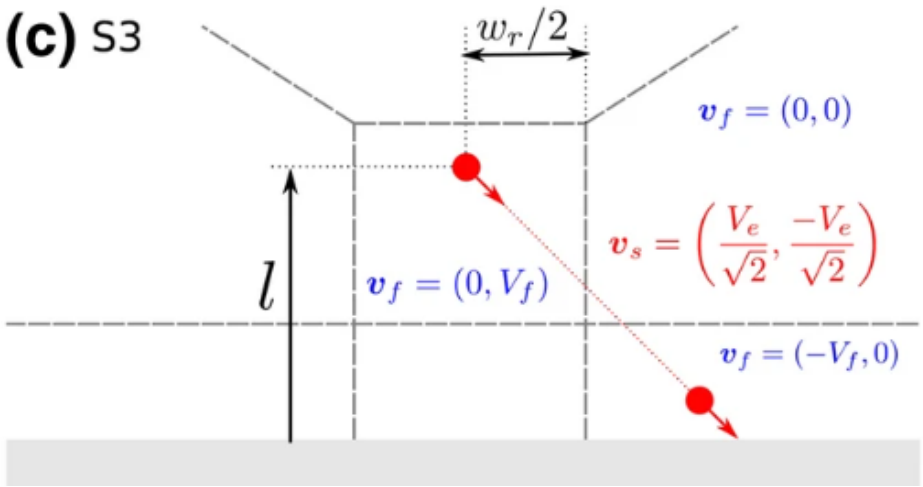


D'après (1) et (2), On a :

$$W_2 = \widetilde{C_T}\left[\frac{w_r}{2V_e}(V_e^2 + V_r^2)^{3/2} + \left(l_x - \frac{w_r}{2} + l - w_f\right)V_e^2 + \frac{w_f}{V_e}(V_e^2 + V_f^2)^{3/2}\right]$$

$$P_2 = \frac{\widetilde{C_T}}{l_x + l}\left[\frac{w_r}{2}(V_e^2 + V_r^2)^{3/2} + \left(l_x - \frac{w_r}{2} + l - w_f\right)V_e^3 + w_f(V_e^2 + V_f^2)^{3/2}\right.$$

- **Stratégie 3 (S3):**

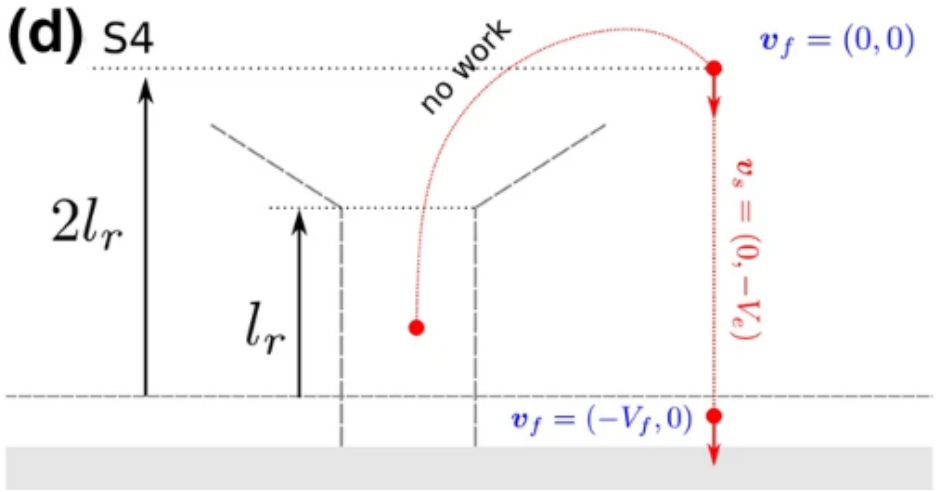


D'après (1) et (2), On a :

$$W_3 = \widetilde{C_T}\left\{\frac{\sqrt{2}w_r}{2V_e}\left[\frac{V_e^2}{2}+\left(\frac{V_e}{\sqrt{2}}+V_r\right)^2\right]^{3/2}+\sqrt{2}\left(l-w_f-\frac{w_r}{2}\right)V_e^2 \quad +\frac{\sqrt{2}w_f}{V_e}\left[\left(\frac{V_e}{\sqrt{2}}+V_f\right)^2+\frac{V_e^2}{2}\right]^{3/2}\right.$$

$$P_3 = \frac{\widetilde{C_T}}{l}\left\{\frac{w_r}{2}\left[\frac{V_e^2}{2}+\left(\frac{V_e}{\sqrt{2}}+V_r\right)^2\right]^{3/2}+\left(l-w_f-\frac{w_r}{2}\right)V_e^3 \quad +w_f\left[\left(\frac{V_e}{\sqrt{2}}+V_f\right)^2+\frac{V_e^2}{2}\right]^{3/2}\right\}$$

- **Stratégie 4 (S4):**



D'après (1) et (2), On a :

$$W_4 = \widetilde{C_T}\left[2l_r V_e^2 + \frac{w_f}{V_e}\left(V_e^2 + V_f^2\right)^{3/2}\right]$$

$$P_4 = \frac{\widetilde{C_T}}{2l_r + w_f}\left[2l_r V_e^3 + w_f\left(V_e^2 + V_f^2\right)^{3/2}\right]$$

Simulation Python:

Objectif:

Générer une base de données à partir de principe de Force Brute

```
ls = [25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 150, 200, 250, 400]
# ls- Distance initiale du nageur par rapport à la côte [m]
Ve = [0.1, 1.0]
# Ve - Vitesse d'echappement du nageur [m]
Vr = [0.2, 2.5]
# Vr - Vitesse du canal du courant d'arrachement [m/s]
Vf = [0.25, 1.0]
# Vf - Vitesse du canal d'alimentation [m/s]
wf = [25, 75]
# wf - Largeur du canal d'alimentation [m]
wr = [10, 50]
# wr - Largeur du canal d'alimentation [m]
lr = [25, 50, 75, 100, 200, 400]
# lr - Distance initiale par rapport à la stratégie 4 [m]
lx = [1, 1.5, 2]
# lx - Distance parcourue parallèlement à la côte dans la stratégie 2 [m]
```

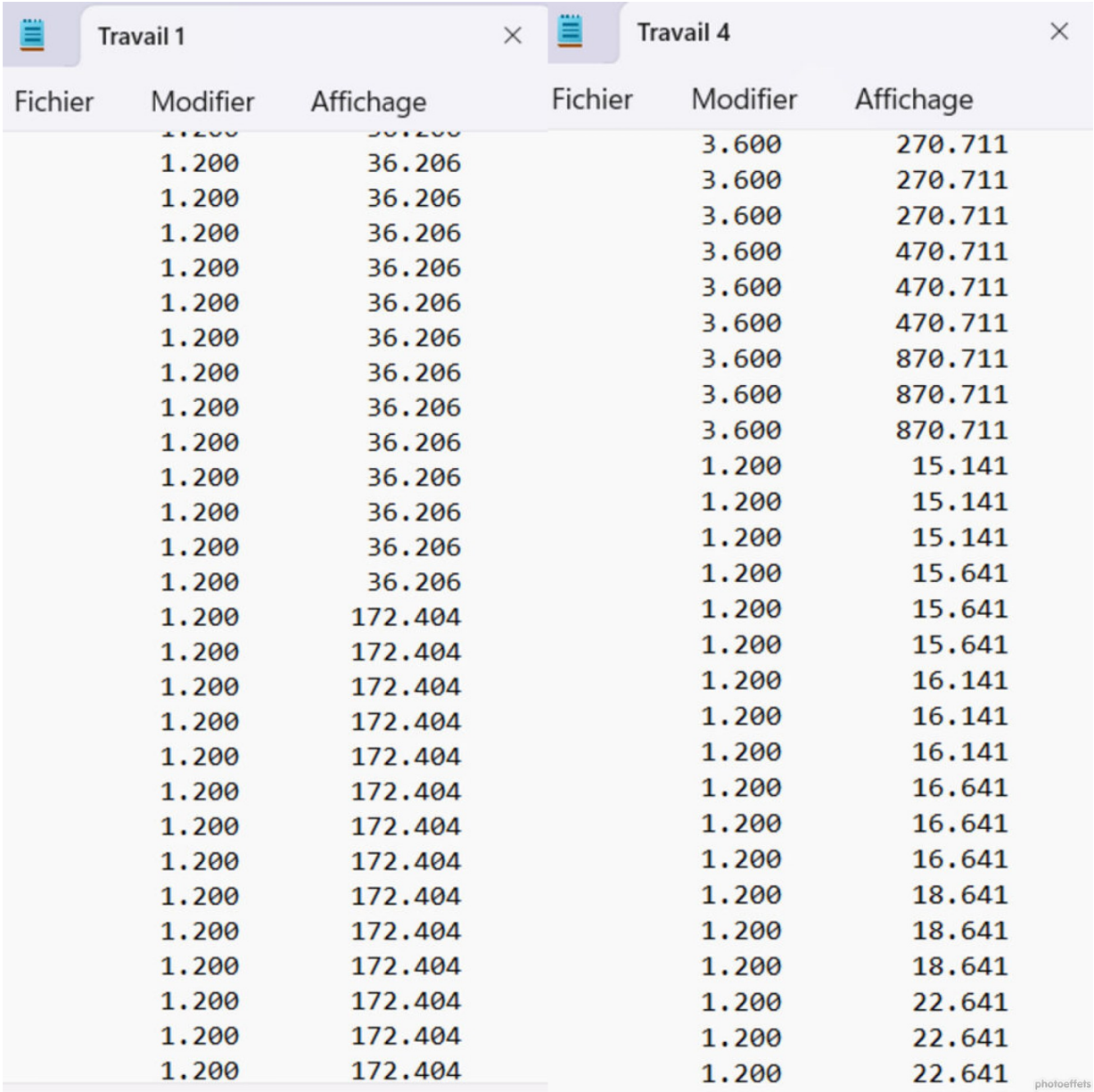
Figure: La gamme de valeurs données à chaque stratégie pour représenter les courants d'arrachement de toutes formes et tailles

```
# Boucles pour passer chaque valeur de la gamme aux fonctions de travail définies ci-dessus
    for i in ls:
    for m in wf:
        for k in Vr:
            for l in Vf:
                for j in Ve:
                    for n in wr:
                        for o in lr:
                            for p in lx:
                                travail1 = travail_strategie1(i, o, j, k, l, m, n)
                                travail2 = travail_strategie2(p * n, i, o, j, k, l, m, n)
                                travail3 = travail_strategie3(i, o, j, k, l, m, n)
                                travail4 = travail_strategie4(o, j, k, l, m, n, i)
                                compteur = compteur + 1
                        # Empiler les résultats dans les matrices
                                newrow1 = np.array([i / m, travail1])
                                mat1 = np.vstack((mat1, newrow1))
                                newrow2 = np.array([i / m, travail2])
                                mat2 = np.vstack((mat2, newrow2))
                                newrow3 = np.array([i / m, travail3])
                                mat3 = np.vstack((mat3, newrow3))
                                newrow4 = np.array([i / m, travail4])
                                mat4 = np.vstack((mat4, newrow4)
                                newrow = np.array([compteur, i, j, k, l, m, n, o, p])
                                mat_pars = np.vstack((mat_pars, newrow))
```

Figure: Générer une base de données à partir de principe de "Force Brute"

```
# Calcul du temps pour atteindre la côte pour chaque stratégie
                travail1_time = i / j
                travail2_time = (p * n + i) / j
                travail3_time = (ma.sqrt(2) * i) / j
                travail4_time = ((2 * o) + m) / j
# Empiler les résultats de la puissance
                newrow1 = np.array([i / m, travail1 / travail1_time])
                mat1_power = np.vstack((mat1_power, newrow1))
                newrow2 = np.array([i / m, travail2 / travail2_time])
                mat2_power = np.vstack((mat2_power, newrow2))
                newrow3 = np.array([i / m, travail3 / travail3_time])
                mat3_power = np.vstack((mat3_power, newrow3))
                newrow4 = np.array([i / m, travail4 / travail4_time])
                mat4_power = np.vstack((mat4_power, newrow4))
```

Figure: Générer une base de données à partir de principe de "Force Brute"



Travail 1

Fichier	Modifier	Affichage
	1.200	36.206
	1.200	36.206
	1.200	36.206
	1.200	36.206
	1.200	36.206
	1.200	36.206
	1.200	36.206
	1.200	36.206
	1.200	36.206
	1.200	36.206
	1.200	36.206
	1.200	36.206
	1.200	36.206
	1.200	172.404
	1.200	172.404
	1.200	172.404
	1.200	172.404
	1.200	172.404
	1.200	172.404
	1.200	172.404
	1.200	172.404
	1.200	172.404
	1.200	172.404
	1.200	172.404
	1.200	172.404
	1.200	172.404

Travail 4

Fichier	Modifier	Affichage
	3.600	270.711
	3.600	270.711
	3.600	270.711
	3.600	470.711
	3.600	470.711
	3.600	470.711
	3.600	870.711
	3.600	870.711
	3.600	870.711
	1.200	15.141
	1.200	15.141
	1.200	15.141
	1.200	15.641
	1.200	15.641
	1.200	15.641
	1.200	16.141
	1.200	16.141
	1.200	16.141
	1.200	16.641
	1.200	16.641
	1.200	16.641
	1.200	18.641
	1.200	18.641
	1.200	18.641
	1.200	22.641
	1.200	22.641
	1.200	22.641

Figure:
Une base de données de travail de la stratégie 1 (W1) et la stratégie 4 (W4)

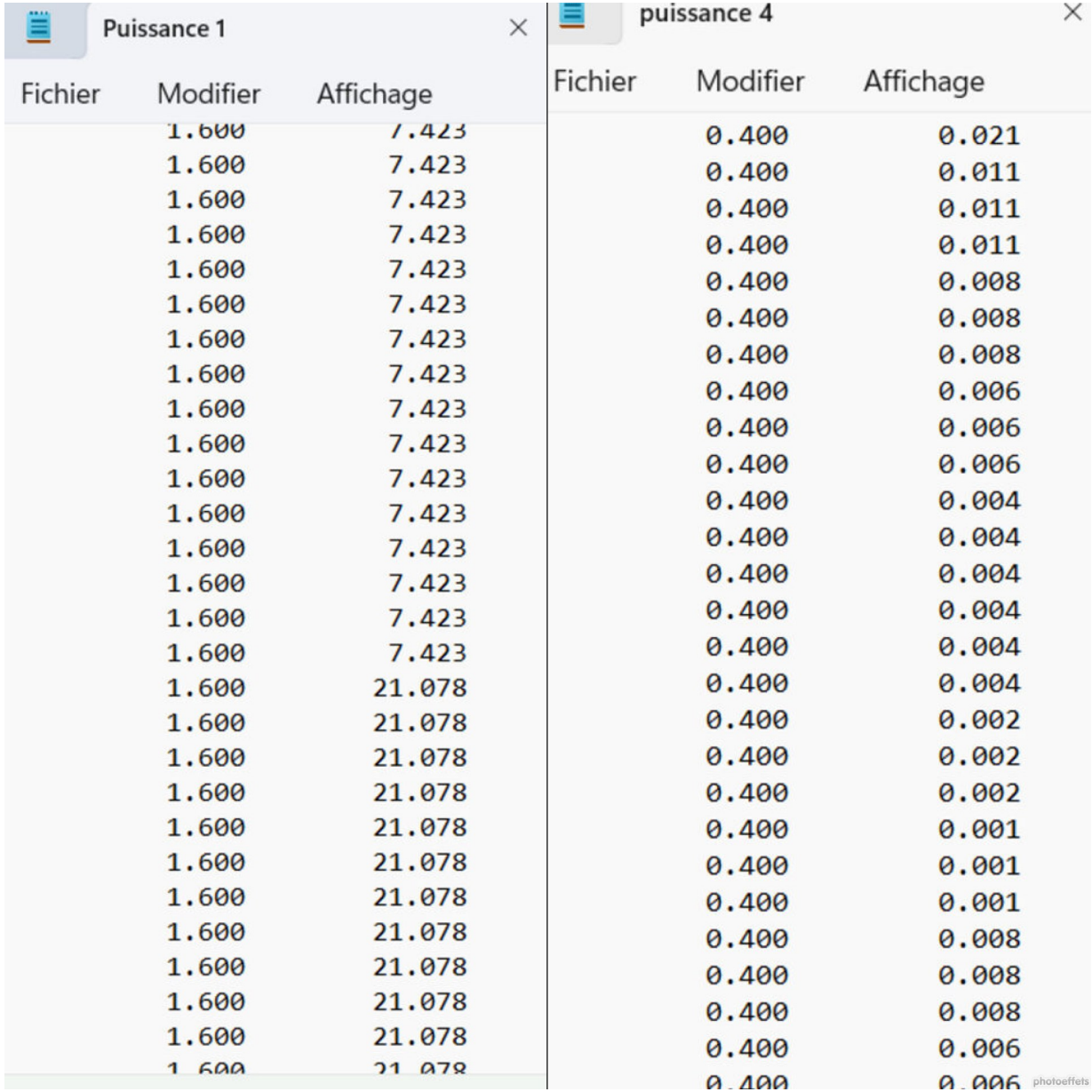


Figure:
Une base de données de puissance de la stratégie 1 (P1) et la stratégie 4 (P4)

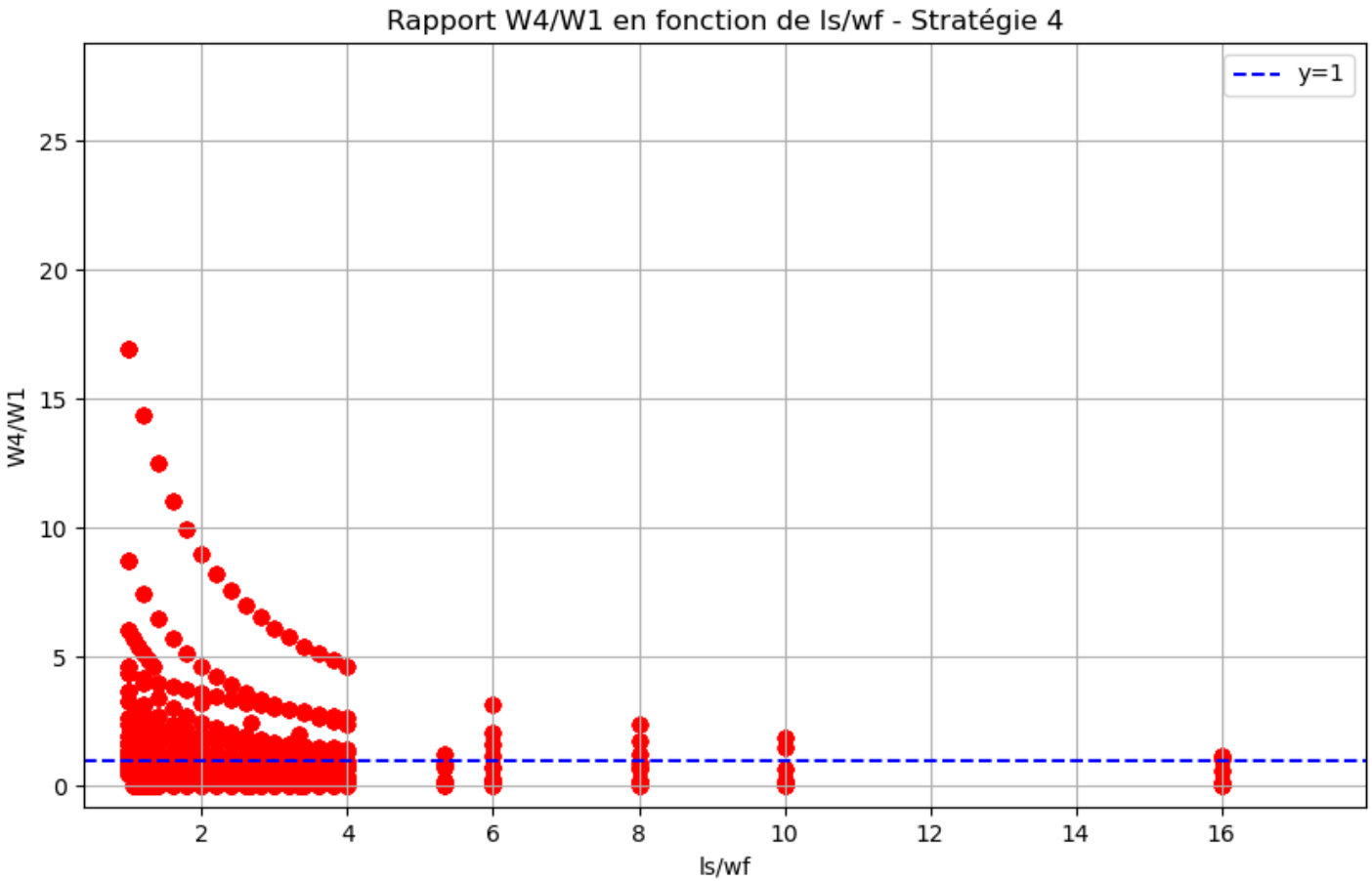


Figure: Le Rapport $\frac{W4}{W1}$ en fonction de $\frac{ls}{wf}$

Analyse:

```
>>> (executing file "Simulation.py")
Pourcentage de points en dessous de y=1 pour la Stratégie 4 : 92.11%
```

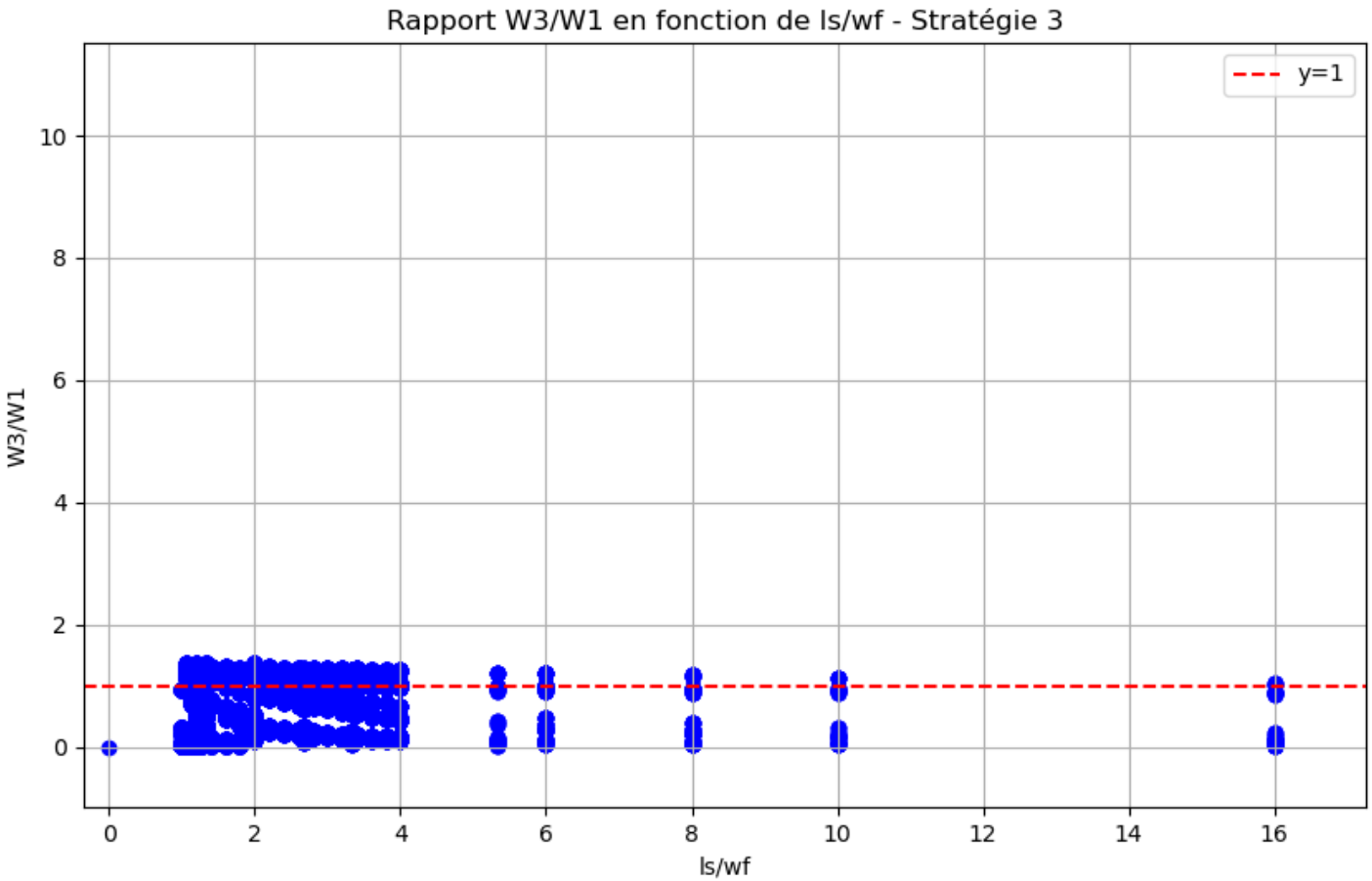


Figure: Le Rapport $\frac{W3}{W1}$ en fonction de $\frac{ls}{wf}$

Analyse:

```
>>> (executing file "Simulation.py")
Pourcentage de points en dessous de y=1 pour la Stratégie 3 : 82.06%
```

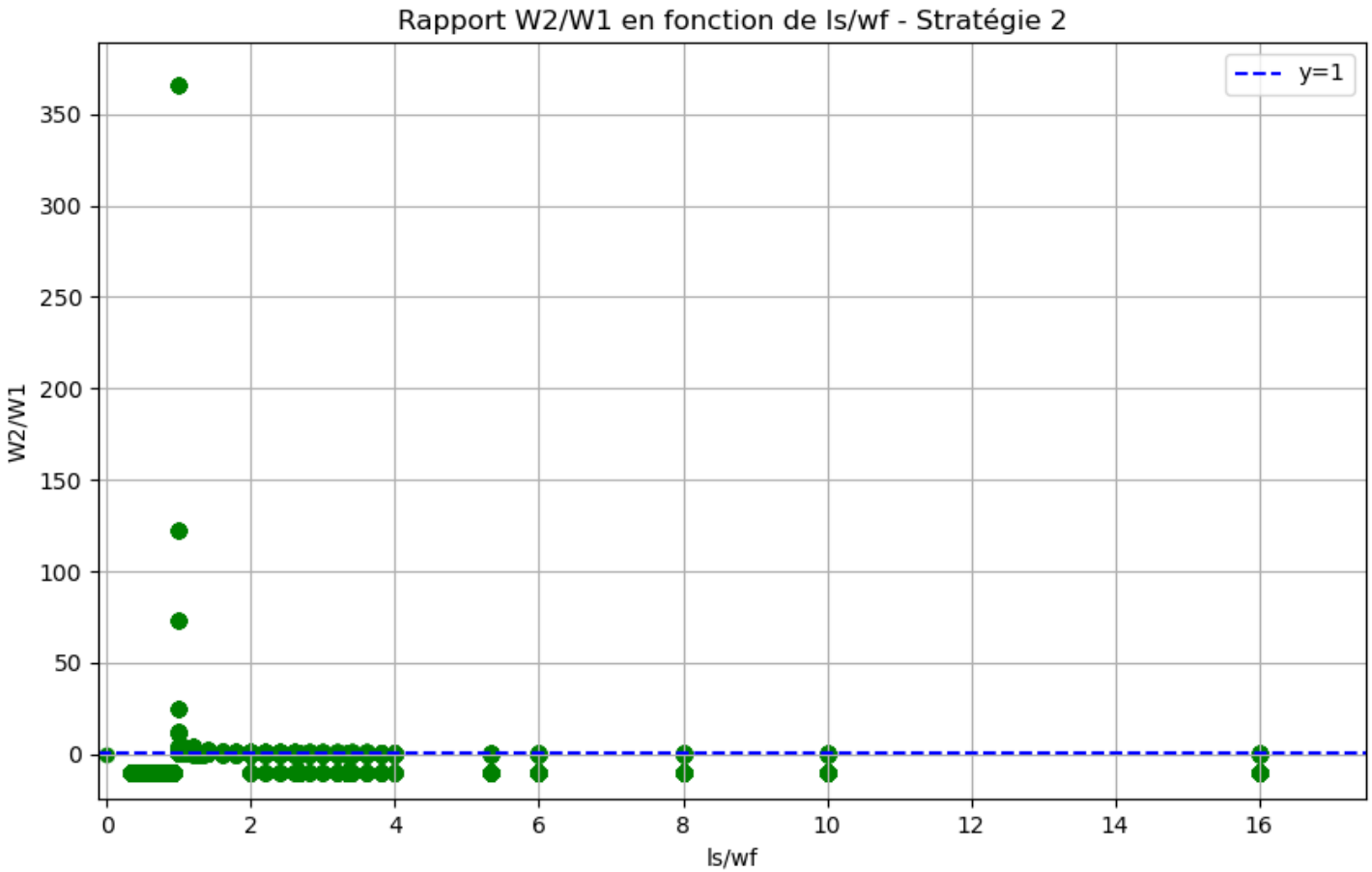


Figure: Le Rapport $\frac{W2}{W1}$ en fonction de $\frac{ls}{wf}$

Analyse:

```
>>> (executing file "Simulation.py")
Pourcentage de points en dessous de y=1 pour la strategie 2: 88.91%
```

Résultat:

le rapport $\frac{Wi}{W1}$:	**Stratégie 2**	**Stratégie 3**	**Stratégie 4**
le pourcentage %	**88,91 %**	**82,06%**	**92,11%**

La meilleure stratégie est la stratégie 4: se laisser porter par le courant avant de nager vers la rive (S4)

Conclusion:

limites du modèle:

- Ne tient pas compte du temps nécessaire pour atteindre la sécurité.
- Risque de panique chez les nageurs piégés.
- la combinaison entre le temps et l'énergie.

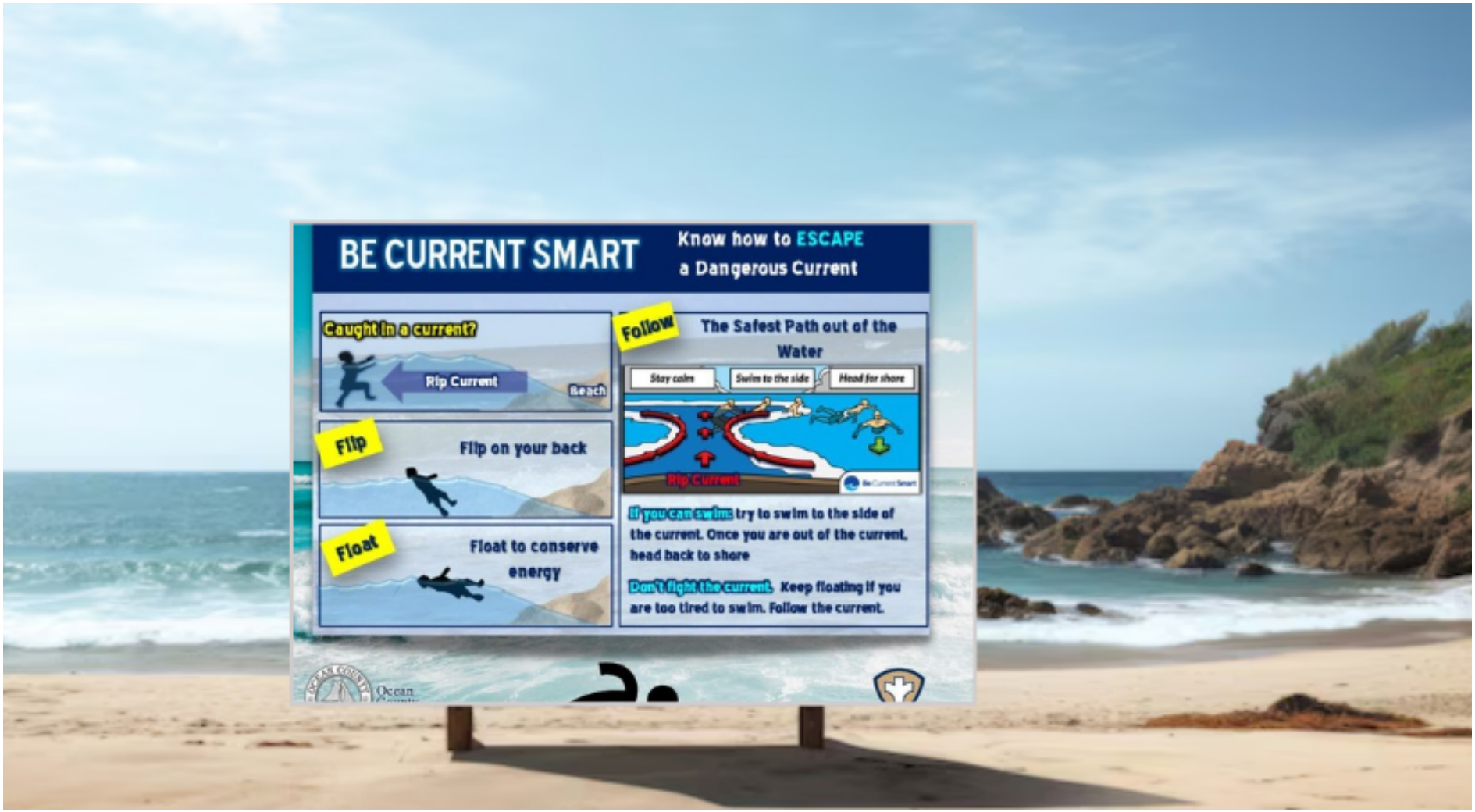


Figure: Panneau à l'entrée de la plage : Conseils pour échapper à un courant d'arrachement

Merci De Votre Attention !

Annexe:

la démonstration de (1): V dans ce cas, elle est considérée comme la vitesse du nageur par rapport à l'écoulement, c'est-à-dire:
$V \equiv \|v_{sK}\| = \|v_s - v_f\|$. On obtient alors :
$\mathrm{F}.v_{sK} = -F_T \cdot v_{sK} = \|F_T\| \|v_s - v_f\| = \widetilde{C_T} \|v_s - v_f\|^3$
Les vecteurs vs et vf sont bidimensionnels et ont des composantes vs = (vsx, vsy) et vf = (vfx, vfy) dans le plan cartésien xy, mesurées à partir de K0.

Travail et Puissance S1:

$$W_1 = \widetilde{C_D} \int_{t_i}^{t_1} (V_e + V_r)^3 \, \mathrm{d}t + \widetilde{C_D} \int_{t_1}^{t_e} (V_e + V_f)^3 \, \mathrm{d}t$$

$$= \widetilde{C_D} \left[(V_e + V_r)^3 (t_1 - t_i) + (V_e + V_f)^3 (t_e - t_1) \right].$$

Or:

$$(\mathrm{t}_1 - t_i) = \frac{(l - w_f)}{V_e} \quad (\mathrm{t}_e - t_1) = \frac{w_f}{V_e}$$

$$W_1 = \widetilde{C_D} \left[\frac{(l - w_f)}{V_e} (V_e + V_r)^3 + \frac{w_f}{V_e} (V_e + V_f)^3 \right].$$

$$P_1 = \frac{\widetilde{C_D}}{l} \left[(l - w_f) (V_e + V_r)^3 + w_f (V_e + V_f)^3 \right].$$

Travail et Puissance S2:

$$W_2 = \widetilde{C_D}\left[\int_{t_i}^{t_1}(V_e^2+V_r^2)^{3/2}\mathrm{d}t+\int_{t_1}^{t_2}V_e^3\mathrm{d}t + \int_{t_2}^{t_3}V_e^3\mathrm{d}t+\int_{t_3}^{t_e}(V_e^2+V_f^2)^{3/2}\mathrm{d}t\right],$$

Or:

$(\mathrm{t}_1 - t_i) = w_r/(2V_e)$;

$(\mathrm{t}_2 - t_1) = (l_x - \frac{w_r}{2})/V_e$;

$(\mathrm{t}_3 - t_2) = (l - w_f)/V_e; (t_e - t_3) = w_f/V_e$;

On obtient :

$$W_2 = \widetilde{C_D} \left[\frac{w_r}{2V_e}(V_e^2 + V_r^2)^{3/2} + \left(l_x - \frac{w_r}{2} + l - w_f \right) V_e^2 + \frac{w_f}{V_e}(V_e^2 + V_f^2)^{3/2} \right].$$

et Puissance:

$$P_2 = \frac{\widetilde{C_D}}{l_x + l} \left[\frac{w_r}{2}(V_e^2 + V_r^2)^{3/2} + \left(l_x - \frac{w_r}{2} + l - w_f \right) V_e^3 + w_f(V_e^2 + V_f^2)^{3/2} \right].$$

Code Python:

```python
# Stratégie 1 - Nager directement vers la côte contre le courant d'arrachement
def travail_strategie1(l, lr, Ve, Vr, Vf, wf, wr):
    """
    Une fonction qui retourne le travail effectué par un nageur s'il choisit d'échapper
    à un courant d'arrachement via la stratégie 1
    """
    W1 = ((Ve+Vr)**3)*((l-wf)/Ve)  # Calcule le travail pour la section nageant dans le canal du courant d'arrachement
    W2 = ((Ve+Vf)**3)*(wf/Ve)      # Calcule le travail pour la section nageant dans le canal d'alimentation
    if l - wf < 0 or l > wf + lr:  # Limites inférieures et supérieures de la distance initiale
        return -1
    else:
        return W1 + W2             # Retourne la valeur totale du travail effectué pour la stratégie 1
```

Figure: Travail W1

Code Python:

```python
# Stratégie 2 - Nager parallèlement au courant d'arrachement sur une distance lx pour échapper au canal du courant d'arrachement, puis tourner de 90 degrés pour nager directement vers la côte
def travail_strategie2(lx, l, lr, Ve, Vr, Vf, wf, wr):
    """
    Une fonction qui retourne le travail effectué par un nageur s'il choisit d'échapper
    à un courant d'arrachement via la stratégie 2
    """
    W1 = ((Ve**2 + Vr**2)**1.5)*(wr/(2*Ve))  # Calcule le travail pour la section nageant à l'intérieur du canal du courant d'arrachement
    W2 = ((Ve**2)*(lx-(wr/2)))               # Calcule le travail pour la section à l'extérieur du courant d'arrachement nageant parallèlement à la côte
    W3 = ((Ve**2)*(l-wf))                    # Calcule le travail pour la section à l'extérieur du courant d'arrachement, mais nageant directement vers la côte
    W4 = ((Ve**2 + Vf**2)**1.5)*(wf/Ve)      # Calcule le travail pour la section dans le canal d'alimentation nageant directement vers la côte
    if l - wf < 0 or l > wf + lr:           # Limites inférieures et supérieures de la distance initiale
        return 10
    else:
        return W1+W2+W3+W4                  # Retourne la valeur totale du travail effectué pour la stratégie 2
```

Figure: Travail W2

Code Python:

```python
def travail_strategie3(l, lr, Ve, Vr, Vf, wf, wr):
    """
    Une fonction qui retourne le travail effectué par un nageur s'il choisit d'échapper
    à un courant d'arrachement via la stratégie 3
    """
    a = Ve**2/2                          # Premier terme dans l'équation du travail effectué en nageant dans le canal du courant d'arrachement
    b = (Ve*ma.sqrt(2))/2 + Vr           # Deuxième terme dans l'équation du travail effectué en nageant dans le canal du courant d'arrachement
    c = (wr*ma.sqrt(2))/(2*Ve)           # Troisième terme dans l'équation du travail effectué en nageant dans le canal du courant d'arrachement
    W1= (((a + b**2)**1.5)*c)            # Calcule le travail effectué pour la section nageant dans le canal du courant d'arrachement
    d = Ve**2                            # Premier terme dans l'équation du travail effectué à l'extérieur du canal du courant d'arrachement
    e = (2*(l-wf-(wr/2)))/ma.sqrt(2)     # Deuxième terme dans l'équation du travail effectué à l'extérieur du canal du courant d'arrachement en
    nageant parallèlement à la côte
    W2 = (d*e)                           # Calcule le travail effectué pour la section nageant à l'extérieur du canal du courant d'arrachement
    f = (Ve*ma.sqrt(2))/2 + Vf           # Premier terme dans l'équation du travail effectué en nageant dans le canal d'alimentation
    g = Ve**2/2                          # Deuxième terme dans l'équation du travail effectué en nageant dans le canal d'alimentation
    h = (2*wf)/(Ve*ma.sqrt(2))           # Troisième terme dans l'équation du travail effectué en nageant dans le canal d'alimentation
    W3 = ((((f**2)+g)**1.5)*h)           # Calcule le travail effectué pour la section nageant dans le canal d'alimentation
    if l < wf + 0.5*wr or l > wf + lr:   # Relation pour s'assurer que le nageur traverse les trois sections du modèle
        return 10
    else:
        return W1+W2+W3                  # Retourne la valeur totale du travail effectué pour la stratégie 3
```

Figure: Travail W3

Code Python:

```python
def travail_strategie4(lr, Ve, Vr, Vf, wf, wr, l):
    """
    Une fonction qui retourne le travail effectué par un nageur s'il choisit d'échapper
    à un courant d'arrachement via la stratégie 4
    (l n'est pas utilisé dans la formule mais est nécessaire pour la comparaison avec la
stratégie 1)
    """
    W4 = ((2*Ve*Ve*lr) + (Ve*Ve + Vf*Vf)**1.5*(wf/Ve))  # Tous les termes dans l'équation
du travail effectué pour la stratégie 4
    if l < wf or l > wf + lr:
        return 10
    else:
        return W4                    # Retourne la valeur totale du travail effectué pour
la stratégie 4
```

Figure: Travail W4

Code Python:

```python
ls = [25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 150, 200, 250, 400]
# ls- Distance initiale du nageur par rapport à la côte [m]
Ve = [0.1, 1.0]
# Ve - Vitesse d'echappement du nageur [m]
Vr = [0.2, 2.5]
# Vr - Vitesse du canal du courant d'arrachement [m/s]
Vf = [0.25, 1.0]
# Vf - Vitesse du canal d'alimentation [m/s]
wf = [25, 75]
# wf - Largeur du canal d'alimentation [m]
wr = [10, 50]
# wr - Largeur du canal d'alimentation [m]
lr = [25, 50, 75, 100, 200, 400]
# lr - Distance initiale par rapport à la stratégie 4 [m]
lx = [1, 1.5, 2]
# lx - Distance parcourue parallèlement à la côte dans la stratégie 2 [m]
```

Figure: les valeurs des données

Code Python:

```
# Boucles pour passer chaque valeur de la gamme aux fonctions de travail définies ci-
dessus
for i in ls:
    for m in wf:
        for k in Vr:
            for l in Vf:
                for j in Ve:
                    for n in wr:
                        for o in lr:
                            for p in lx:
                                travail1 = travail_strategie1(i, o, j, k, l, m, n)
                                travail2 = travail_strategie2(p * n, i, o, j, k, l, m, n)
                                travail3 = travail_strategie3(i, o, j, k, l, m, n)
                                travail4 = travail_strategie4(o, j, k, l, m, n, i)

                                # Calculer les rapports Wi/W1 et les stocker dans les
matrices appropriées
                                ratio2 = travail2 / travail1
                                ratio3 = travail3 / travail1
                                ratio4 = travail4 / travail1

                                # Empiler les résultats dans les matrices de rapports
                                newrow2 = np.array([i / m, ratio2])
                                mat_ratio2 = np.vstack((mat_ratio2, newrow2))
                                newrow3 = np.array([i / m, ratio3])
                                mat_ratio3 = np.vstack((mat_ratio3, newrow3))
                                newrow4 = np.array([i / m, ratio4])
                                mat_ratio4 = np.vstack((mat_ratio4, newrow4))
```

Figure: Le Rapport $\frac{Wi}{W1}$

Code Python:

```python
# Générer les fichiers pour les rapports Wi/W1
fichier = open('ratio2.dat', 'w')
for ligne in mat_ratio2:
    for colonne in ligne:
        fichier.write('%14.3f' % colonne)
    fichier.write('\n')
fichier.close()

fichier = open('ratio3.dat', 'w')
for ligne in mat_ratio3:
    for colonne in ligne:
        fichier.write('%14.3f' % colonne)
    fichier.write('\n')
fichier.close()

fichier = open('ratio4.dat', 'w')
for ligne in mat_ratio4:
    for colonne in ligne:
        fichier.write('%14.3f' % colonne)
    fichier.write('\n')
fichier.close()
import matplotlib.pyplot as plt
```

Figure: Générer les fichiers pour les rapports $\frac{Wi}{W1}$

Code Python:

```python
import matplotlib.pyplot as plt
# Charger les données des fichiers de rapports
data_ratio2 = np.loadtxt('ratio2.dat')
data_ratio3 = np.loadtxt('ratio3.dat')
data_ratio4 = np.loadtxt('ratio4.dat')
import matplotlib.pyplot as plt
# Charger les données du fichier de rapports pour la Stratégie 4
data_ratio4 = np.loadtxt('ratio4.dat')
# Extraire les colonnes des données
ls_wf_ratio4 = data_ratio4[:, 0]
ratio4_values = data_ratio4[:, 1]
# Tracer le graphique pour la Stratégie 4
plt.figure(figsize=(10, 6))
plt.scatter(ls_wf_ratio4, ratio4_values, color='red', marker='o')
plt.title('Rapport W4/W1 en fonction de ls/wf - Stratégie 4')
plt.xlabel('ls/wf')
plt.ylabel('W4/W1')
plt.grid(True)
plt.show()
```

Figure: Générer les figures de la simulation