Mini rapport

Présentation des résultats

Exemple de 10 tâches et l'exécution pour chaque algorithme

Dans cette partie nous allons tester et comparer nos fonctions sur 10 tâches. Les vraies valeurs des durées de tâches sont tirées selon une distribution de Pareto avec un paramètre alpha égal à 1.1. Les valeurs des prédictions sont obtenues selon une distribution normale autour des vraies valeurs de durées et tâches, avec un paramètre sigma de 1.5.

Nous avons résumé les résultats en fin de section dans un tableau récapitulatif.

Entrée [209]:

```
nbTaches = 10
alpha = 1.1
sigma = 1.5

vrai = [round(np.random.pareto(alpha), 2) for i in range(nbTaches)]
bruit = np.random.randn(nbTaches)*sigma
pred = [round(vrai[i] + bruit[i], 2) for i in range(nbTaches)]

# On vérifie qu'aucune prédiction n'est négative :
for i in range(len(pred)):
    if pred[i]<0:
        pred[i] = 0

print("Vrai = ", vrai)
print("\nPred = ", pred)

Cout =[]
```

```
Vrai =  [0.15, 0.46, 0.17, 0.81, 0.45, 0.41, 0.1, 0.77, 0.05, 0.04]

Pred =  [0, 1.17, 0, 4.49, 0, 2.45, 0, 0, 0, 0]
```

Entrée [210]:

```
# Execution pour Prediction :
dateFin = [0 for i in range(nbTaches)]
print("prediction(", pred, ",", vrai, ", l=dateFin) : ")
res = prediction(pred, vrai, l=dateFin)
print("L'ordonnancement obtenu est : ", res[0])
print("de coût ", res[1])
Cout.append(res[1])
print("Les dates de fins de tâches sont : ", dateFin)
```

```
prediction( [0, 1.17, 0, 4.49, 0, 2.45, 0, 0, 0, 0] , [0.15, 0.46, 0.17, 0.8
1, 0.45, 0.41, 0.1, 0.77, 0.05, 0.04] , l=dateFin) :
L'ordonnancement obtenu est :  [0.15, 0.17, 0.45, 0.1, 0.77, 0.05, 0.04, 0.4
6, 0.41, 0.81]
de coût  15.370000000000001
Les dates de fins de tâches sont :  [0.15, 2.1900000000000004, 0.32, 3.41000
00000000006, 0.77, 2.6000000000000005, 0.87, 1.6400000000000001, 1.690000000
0000002, 1.7300000000000002]
```

Entrée [211]:

```
# Execution pour Spt :
print("spt(",vrai,") : ")
res = spt(vrai)
print("L'ordonnancement obtenu est : ", res[0])
print("de coût ", res[1])
Cout.append(res[1])
```

```
spt( [0.15, 0.46, 0.17, 0.81, 0.45, 0.41, 0.1, 0.77, 0.05, 0.04] ) :
L'ordonnancement obtenu est :  [0.04, 0.05, 0.1, 0.15, 0.17, 0.41, 0.45, 0.4
6, 0.77, 0.81]
de coût  11.299999999999999
```

Entrée [212]:

```
# Execution pour Round-Robin :
dateFin = [0 for i in range(nbTaches)]
print("round_robin(",vrai,", l=dateFin) : ")
Cout.append(round_robin(vrai, l=dateFin))
print("L'ordonnancement obtenu a un coût de ", Cout[-1])
print("Les dates de fins de tâches sont : ", dateFin)
```

```
round_robin( [0.15, 0.46, 0.17, 0.81, 0.45, 0.41, 0.1, 0.77, 0.05, 0.04] , l
=dateFin) :
L'ordonnancement obtenu a un coût de  19.19
Les dates de fins de tâches sont :  [1.24, 2.75, 1.36, 3.41, 2.72, 2.56, 0.8
900000000000001, 3.37, 0.49000000000000005, 0.4]
```

Entrée [213]:

```
# Execution pour Pred-Round-Robin :
lamb = 1/2
print("pred_round_robin(", pred, ",", vrai, ", ", lamb, ") : ")
Cout.append(pred_round_robin(pred, vrai, lamb))
print("L'ordonnancement obtenu a un coût de ", Cout[-1])
```

```
pred_round_robin( [0, 1.17, 0, 4.49, 0, 2.45, 0, 0, 0, 0] , [0.15, 0.46, 0.1
7, 0.81, 0.45, 0.41, 0.1, 0.77, 0.05, 0.04] ,  0.5 ) :
L'ordonnancement obtenu a un coût de  25.6
```

Entrée [214]:

```
# Execution pour notre version de Pred-Round-Robin :
lamb = 1/2
print("pred_round_robin2(", pred, ",", vrai, ", ", lamb, ") : ")
Cout.append( pred_round_robin2(pred, vrai, lamb))
print("L'ordonnancement obtenu a un coût de ", Cout[-1])
```

```
pred_round_robin2( [0, 1.17, 0, 4.49, 0, 2.45, 0, 0, 0, 0] , [0.15, 0.46, 0.
17, 0.81, 0.45, 0.41, 0.1, 0.77, 0.05, 0.04] ,  0.5 ) :
L'ordonnancement obtenu a un coût de  16.505515151515155
```

Entrée [215]:

```
# Execution pour Prediction partie B :
dateFin = [0 for i in range(nbTaches)]
arr = [int(np.random.randint(0, int(nbTaches * min(vrai))+1)) for i in range(nbTaches)]
print("predictionB(", pred, ",", vrai, ",", arr, ", l=dateFin) : ")
Cout.append(predictionB(pred, vrai, arr, l=dateFin))
print("L'ordonnancement obtenu a un coût de ", Cout[-1])
print("Les dates de fins de tâches sont : ", dateFin)
```

```
predictionB( [0, 1.17, 0, 4.49, 0, 2.45, 0, 0, 0, 0] , [0.15, 0.46, 0.17, 0.
81, 0.45, 0.41, 0.1, 0.77, 0.05, 0.04] , [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] , l=
dateFin) :
L'ordonnancement obtenu a un coût de  15.370000000000001
Les dates de fins de tâches sont :  [0.15, 2.1900000000000004, 0.32, 3.41000
00000000006, 0.77, 2.6000000000000005, 0.87, 1.6400000000000001, 1.690000000
0000002, 1.7300000000000002]
```

Entrée [216]:

```
# Execution pour Spt partie B :
print("sptB(",vrai, ",", arr, ") : ")
Cout.append(sptB(vrai, arr))
print("L'ordonnancement obtenu a un coût de ", Cout[-1])
```

```
sptB( [0.15, 0.46, 0.17, 0.81, 0.45, 0.41, 0.1, 0.77, 0.05, 0.04] , [0, 0,
0, 0, 0, 0, 0, 0, 0, 0] ) :
L'ordonnancement obtenu a un coût de  11.299999999999999
```

Entrée [217]:

```
# Execution pour Round-Robin partie B :
dateFin = [0 for i in range(nbTaches)]
print("round_robinB(",vrai, ",", arr, ", l=dateFin) : ")
Cout.append(round_robinB(vrai, arr, l=dateFin))
print("L'ordonnancement obtenu a un coût de ", Cout[-1])
print("Les dates de fins de tâches sont : ", dateFin)
```

```
round_robinB( [0.15, 0.46, 0.17, 0.81, 0.45, 0.41, 0.1, 0.77, 0.05, 0.04] ,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0] , l=dateFin) :
L'ordonnancement obtenu a un coût de  19.19
Les dates de fins de tâches sont :  [1.24, 2.75, 1.36, 3.41, 2.72, 2.56, 0.8
900000000000001, 3.37, 0.49000000000000005, 0.4]
```

Entrée [218]:

```
# Execution pour Pred-Round-Robin partie B :
lamb = 1/2
print("pred_round_robinB(", pred, ",", vrai, ",", arr, ", ", lamb, ") : ")
Cout.append(pred_round_robinB(pred, vrai, arr, lamb))
print("L'ordonnancement obtenu a un coût de ", Cout[-1])
```

```
pred_round_robinB( [0, 1.17, 0, 4.49, 0, 2.45, 0, 0, 0, 0] , [0.15, 0.46, 0.
17, 0.81, 0.45, 0.41, 0.1, 0.77, 0.05, 0.04] , [0, 0, 0, 0, 0, 0, 0, 0, 0,
0] ,  0.5 ) :
L'ordonnancement obtenu a un coût de  25.6
```

Entrée [219]:

```
# Execution pour notre version de Pred-Round-Robin partie B :
lamb = 1/2
print("pred_round_robinB2(", pred, ",", vrai, ",", arr, ", ", lamb, ") : ")
Cout.append(pred_round_robinB2(pred, vrai, arr, lamb))
print("L'ordonnancement obtenu a un coût de ", Cout[-1])
```

```
pred_round_robinB2( [0, 1.17, 0, 4.49, 0, 2.45, 0, 0, 0, 0] , [0.15, 0.46,
0.17, 0.81, 0.45, 0.41, 0.1, 0.77, 0.05, 0.04] , [0, 0, 0, 0, 0, 0, 0, 0, 0,
0] ,  0.5 ) :
L'ordonnancement obtenu a un coût de  16.505515151515155
```

Entrée [220]:

```
print("En résumé, voici les coûts trouvés avec chaque algorithme pour la partie A : ")
print("Pour PREDICTION \t    : ", Cout[0])
print("Pour SPT \t \t    : ", Cout[1])
print("Pour ROUND-ROBIN \t    : ", Cout[2])
print("Pour PRED-ROUND-ROBIN \t    : ", Cout[3])
print("Pour notre PRED-ROUND-ROBIN : ", Cout[4])

print("\nEt voici les coûts trouvés avec chaque algorithme pour la partie B :")
print("Pour PREDICTION \t    : ", Cout[5])
print("Pour SPT \t \t    : ", Cout[6])
print("Pour ROUND-ROBIN \t    : ", Cout[7])
print("Pour PRED-ROUND-ROBIN \t    : ", Cout[8])
print("Pour notre PRED-ROUND-ROBIN : ", Cout[9])
```

```
En résumé, voici les coûts trouvés avec chaque algorithme pour la partie A :
Pour PREDICTION             :  15.370000000000001
Pour SPT                    :  11.299999999999999
Pour ROUND-ROBIN            :  19.19
Pour PRED-ROUND-ROBIN       :  25.6
Pour notre PRED-ROUND-ROBIN :  16.505515151515155

Et voici les coûts trouvés avec chaque algorithme pour la partie B :
Pour PREDICTION             :  15.370000000000001
Pour SPT                    :  11.299999999999999
Pour ROUND-ROBIN            :  19.19
Pour PRED-ROUND-ROBIN       :  25.6
Pour notre PRED-ROUND-ROBIN :  16.505515151515155
```

À première vue, l'algorithme prediction semble être le meilleur (après SPT naturellement). Mais nous verrons plus tard que cette performance dépend beaucoup du paramètre arbitraire sigma, à l'oeuvre dans la génération aléatoire des prédictions.

Les choix pour les données et les résultats de simulations commentés

Dans cette partie nous avons fait des simulations pour comparer les différentes fonctions d'ordonnancement. Les vraies valeurs des durées de tâches sont tirées selon une distribution de Pareto avec un paramètre alpha égal à 1.1. Les valeurs des prédictions sont obtenues selon une distribution normale autour des vraies valeurs de durées et tâches, avec un paramètre sigma que l'on fera varier. Nous testons pour 50 tâches.

Partie A

Dans cette partie, on considère que toutes les tâches peuvent débuter au temps 0.

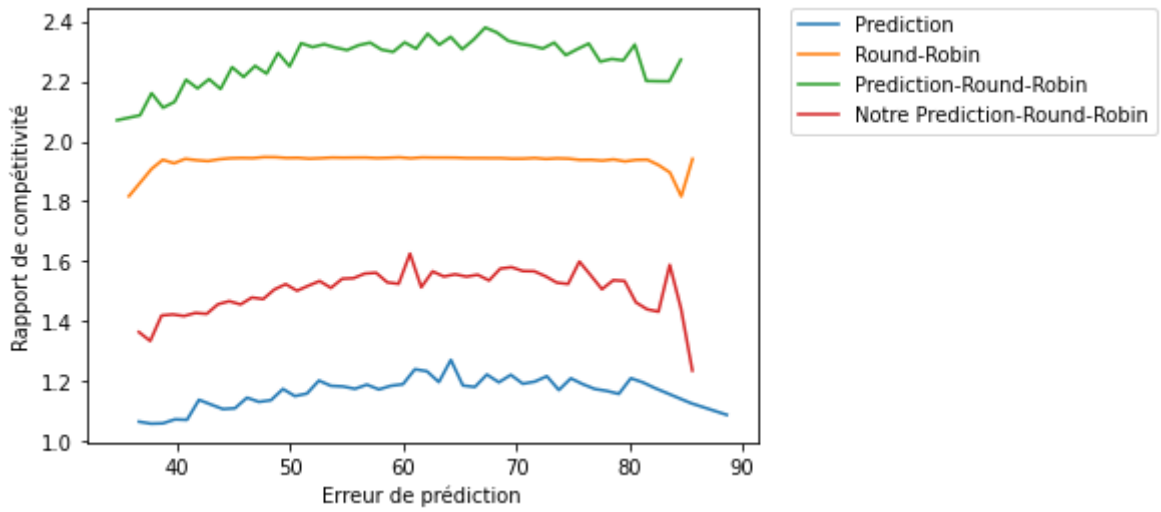
Dans un premier temps, on fixe un paramètre sigma à 1.5 pour la qualité des prédictions.

Entrée [221]:

```
## Pour la partie A
S = 1.5
resA = []
resA.append(simulation2(spt, f=prediction, sigma=S, legende="Prediction"))
#resA.append(simulation2(spt, f=prediction, sigma=5, legende="Prediction sigma 5"))
resA.append(simulation2(spt, f=round_robin, sigma=S, legende="Round-Robin"))
resA.append(simulation2(spt, f=pred_round_robin, sigma=S, legende = "Prediction-Round-Robin
resA.append(simulation2(spt, f=pred_round_robin2, sigma=S, legende = "Notre Prediction-Roun
```

Entrée [222]:

```
## Superposons les graphes
### Partie A
show_multiple_sims(resA)
```



Ce graphique présente les maximas des rapports de compétitivité pour chaque fonction implémentée en fonction des erreurs de prédiction. On observe que pour toutes les fonctions sauf round robin, plus l'erreur de prédiction est grande, plus le rapport de compétitivité l'est aussi, avec une légère baisse à partir des erreurs de prédictions égales à 60. Note : cette baisse n'est probablement pas significative et doit dépendre de la manière dont les données sont générées.

On constate que Prediction obtient les meilleurs résultats, bien qu'il faille prendre du recul puisque les prédictions sont générées aléatoirement avec des paramètres fixés arbitrairement et qu'ici les prédictions sont très bonnes (très proches de la réalité). Ensuite, la meilleure fonction est notre Prediction-round-robin, suivie de round-robin et enfin prediction-round-robin.

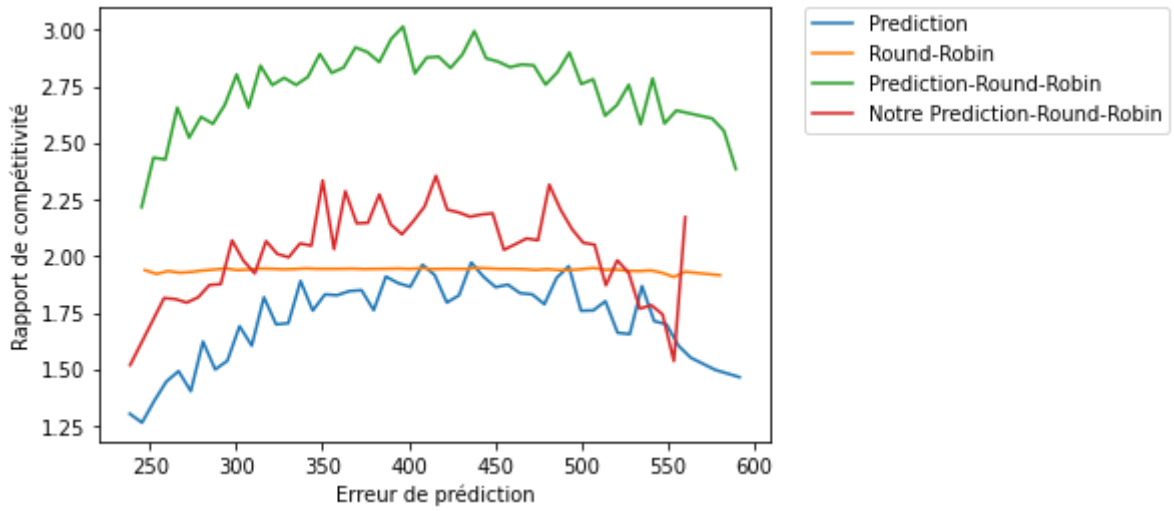
À présent nous étudions l'impact d'une qualité médiocre de prédiction sur les rapports de compétitivité des fonctions.

Entrée [223]:

```
## Pour la partie A
S = 10
resA2 = []
resA2.append(simulation2(spt, f=prediction, sigma=S, legende="Prediction"))
#resA2.append(simulation2(spt, f=prediction, sigma=5, legende="Prediction sigma 5"))
resA2.append(simulation2(spt, f=round_robin, sigma=S, legende="Round-Robin"))
resA2.append(simulation2(spt, f=pred_round_robin, sigma=S, legende = "Prediction-Round-Robi
resA2.append(simulation2(spt, f=pred_round_robin2, sigma=S, legende = "Notre Prediction-Rou
```

Entrée [224]:

```
## Superposons les graphes
### Partie A
show_multiple_sims(resA2)
```



En augmentant sigma, et donc en diminuant la qualité des prédictions, on observe une dégradation du rapport de compétitivité de toutes les fonctions utilisant la prédiction (ce dernier augmente par rapport au dernier graphique, s'éloignant ainsi de 1). La fonction Round-Robin, elle, reste non-impactée et semble être un meilleur choix quand une prédiction de qualité ne peut être pas obtenue, nous assurant un rapport de compétitivité inférieur ou égal à 2 (nous rappelons que les rapports de compétitivité choisis ici sont les pires trouvés pour les classes d'erreur de prédiction, voir la fonction simulation2 pour plus de détails).

Partie B

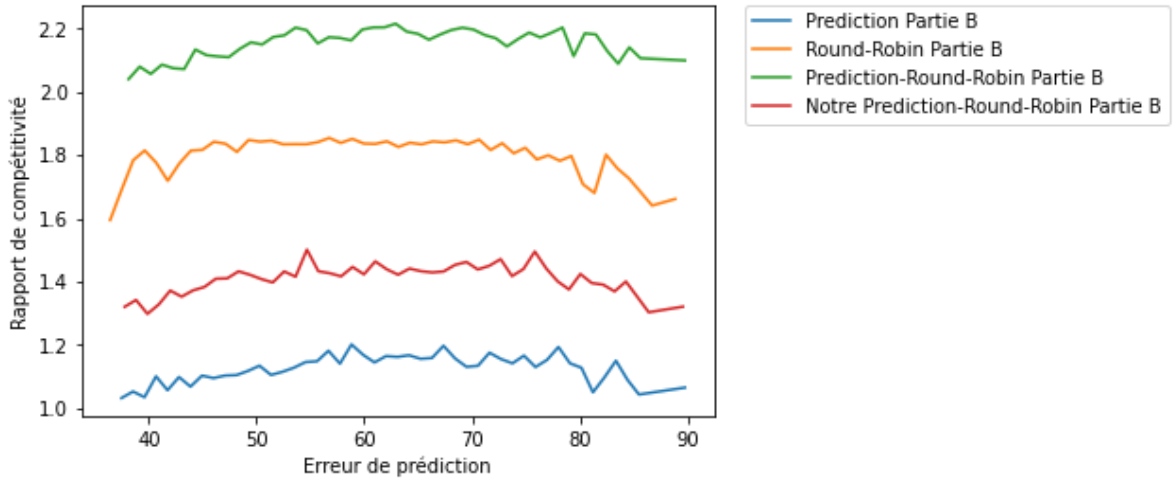
Dans cette partie, nous considèrons que toutes les tâches n'arrivent pas au même moment et donc qu'il faut les ordonnancer à mesure qu'elles arrivent. Les dates d'arrivée des tâches sont générées aléatoirement selon une distribution equiprobable bornée entre 0 et la plus petite durée de tâche multipliée par le nombre de tâches.

Entrée [225]:

```
## Pour la partie B
S = 1.5

resB = []
resB.append(simulation2(sptB, f=predictionB, partie="B", sigma=S, legende="Prediction Parti
#resB.append(simulation2(sptB, f=predictionB, partie="B", sigma=5, legende="Prediction Part
resB.append(simulation2(sptB, f=round_robinB, partie="B", sigma=S, legende="Round-Robin Par
resB.append(simulation2(sptB, f=pred_round_robinB, partie="B", sigma=S, legende="Prediction
resB.append(simulation2(sptB, f=pred_round_robinB2, partie="B", sigma=S, legende="Notre Pre
```

Entrée [226]:

```
## Superposons les graphes
### Partie B
show_multiple_sims(resB)
```



Ce graphique présente les maximums de rapport de compétitivité pour chaque fonction implémentée en fonction des erreurs de prédiction. Dans cette partie les tâches ont également une date d'arrivée.

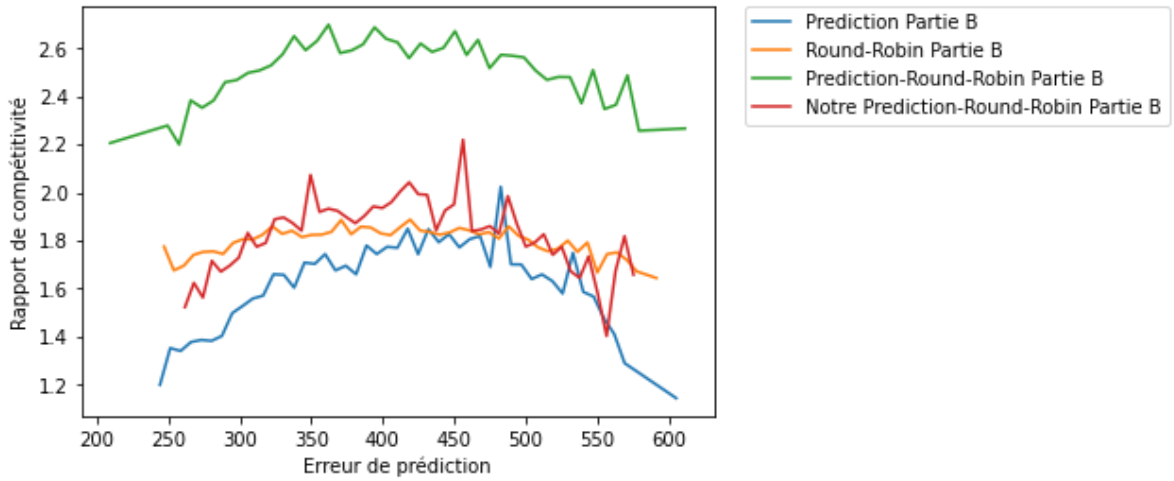
L'analyse est la même que pour la partie A, à la différence qu'ici, la fonction Round-Robin n'est plus aussi constante qu'avant. Elle montre toutefois une efficacité similaire à celle qu'elle avait dans la partie A.

Entrée [227]:

```
## Pour la partie B
S = 10

resB2 = []
resB2.append(simulation2(sptB, f=predictionB, partie="B", sigma=S, legende="Prediction Part
#resB2.append(simulation2(sptB, f=predictionB, partie="B", sigma=5, legende="Prediction Par
resB2.append(simulation2(sptB, f=round_robinB, partie="B", sigma=S, legende="Round-Robin Pa
resB2.append(simulation2(sptB, f=pred_round_robinB, partie="B", sigma=S, legende="Predictio
resB2.append(simulation2(sptB, f=pred_round_robinB2, partie="B", sigma=S, legende="Notre Pr
```

Entrée [228]:

```
## Superposons les graphes
### Partie B avec un grand sigma
show_multiple_sims(resB2)
```



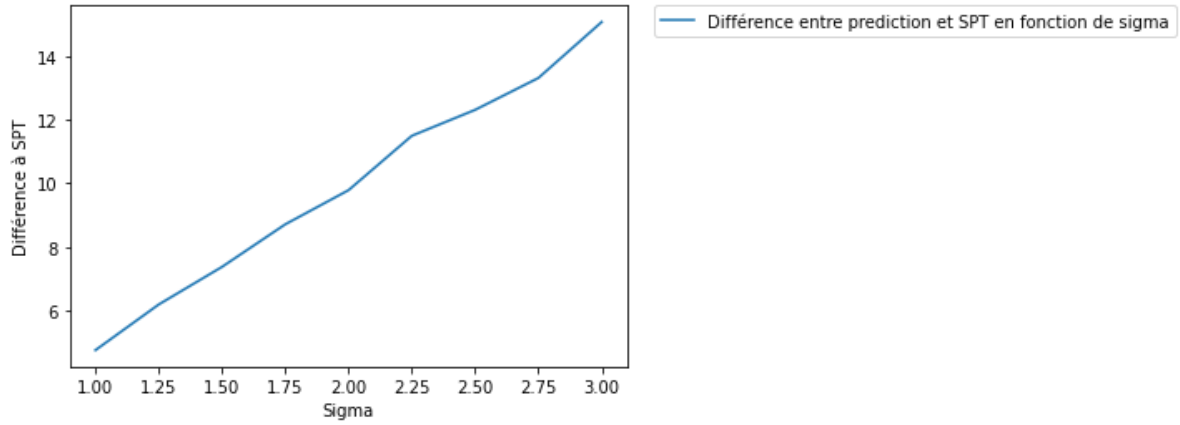
Ici avec une prédiction dégradée, on observe le même comportement que dans la partie A (sans les dates d'arrivée).

Variabilité de la qualité de prédiction

Dans cette partie, nous allons faire varier le paramètre sigma qui entre en jeu pour la génération des dates prédites, suivant une distribution normale.

Entrée [229]:

```
sigmas = [1+i*0.25 for i in range(int(2/0.25)+1)]
nbTaches = 10
Y = []
for s in sigmas:
    subY = []
    for _ in range(1000):

        vrai = [np.random.pareto(alpha)+1 for i in range(nbTaches)]
        bruit = np.random.randn(nbTaches)*s
        pred = [round(vrai[i] + bruit[i], 2) for i in range(nbTaches)]

        subY.append(abs(spt(vrai)[1] - prediction(pred, vrai)[1]))

    Y.append(np.mean(subY))

plt.plot(sigmas, Y, label="Différence entre prediction et SPT en fonction de sigma")
#plt.plot(X, Y, label=legende)

plt.xlabel('Sigma')
plt.ylabel('Différence à SPT')
#plt.legend(loc='best')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.show()
```



Comme attendu, plus le sigma augmente, plus la différence entre le résultat de prediction et la solution optimale est grande, et ce de manière linéaire.

Conclusion

Nos résultats montrent que Round-Robin présente expérimentalement un rapport de compétitivité borné à 2-OPT. L'utilité des fonctions de prédiction dépend directement de la qualité des prédictions, et n'ont pas d'intérêt si cette dernière est médiocre. L'association de Prediction et de Round-Robin n'a pas montré d'avantage sur les autres fonctions et performe même moins bien que celles-ci.