Analyse des algorithmes

Algorithme Glouton:

On a réalisé des tests pour des tailles de 1000, 5000, 10000, 50000 et 100000 villes, et on a fait des simulations sur 50 listes de points, sauf pour les deux derniers où 25 ont été utilisés.

- Puissance

On suppose une complexité polynomiale pour cet algorithme au vu de la croissance des temps d'exécution. On effectue donc un test de puissance dans ce sens et on trouve effectivement une droite, ce qui confirme l'hypothèse. La pente de la droite est de 1.991, on a donc un complexité qui croît selon $N^{1.991}$, où N est le nombre de villes.

- Ratio

On fait l'hypothèse - en suivant le test de puissance - que la complexité est polynomiale de degré 1.991. On cherche à déterminer la constante multiplicative. On trouve une constante de 2.19e-09, affinant donc notre approximation. Le ratio de l'approximation et des données est constant ce qui prouve que l'exposant est bien correct.

- Constante

On effectue enfin le test de la constante, utilisant notre approximation précédemment trouvée. L'ordonnée à l'origine est nulle et on trouve une constante multiplicative de 1.012 ce qui affine légèrement l'approximation de la constante multiplicative.

La constante étant très proche de 1, notre hypothèse était donc valide.

- **Conclusion**

On trouve une complexité de l'ordre de N^2 pour l'algorithme glouton, avec une constante multiplicative très faible de l'ordre de 2.10^{-9}.

Algorithme Approximatif:

On a réalisé des tests sur les mêmes exemplaires que l'algorithme glouton. Sauf que l'on a utilisé moins d'exemplaires sur les grands nombres de villes car l'algorithme était plus lent. On a donc utilisé respectivement 50, 50, 50, 25 et 10 listes pour les différentes tailles.

- Puissance

On suppose une complexité polynomiale pour cet algorithme au vu de la croissance des temps d'exécution. On effectue donc un test de puissance dans ce sens et on trouve effectivement une droite, ce qui confirme l'hypothèse. La pente de la droite est de 2.010, on a donc un complexité qui croît selon $N^{2.010}$, où N est le nombre de villes.

- Ratio

On fait l'hypothèse - en suivant le test de puissance - que la complexité est polynomiale de degré 2.010. On cherche à déterminer la constante multiplicative. On trouve une constante de 5.17e-09, affinant donc notre approximation. Le ratio de l'approximation et des données est constant ce qui prouve que l'exposant est bien correct.

- Constante

On effectue enfin le test de la constante, utilisant notre approximation précédemment trouvée. L'ordonnée à l'origine est nulle et on trouve une constante multiplicative de 1.042 ce qui affine légèrement l'approximation de la constante multiplicative.

La constante étant très proche de 1, notre hypothèse était donc valide.

- **Conclusion**

On trouve une complexité de l'ordre de N^2 comme pour l'algorithme glouton, mais avec une constante environ 2.5 fois plus élevée.

<u>Algorithme de Programmation Dynamique:</u>

On a réalisé des tests sur des exemplaires de taille 5 à 20, l'exemplaire de taille 25 remplit la RAM. Pour chaque taille on a fait des moyennes sur 50 exécutions.

- Puissance

On a supposé une croissance exponentielle, on a donc réalisé un test de puissance modifié en ayant x en échelle linéaire et y en échelle logarithmique. On trouve une droite, ce qui confirme notre hypothèse. On utilise un logarithme népérien, donc en prenant l'exponentielle de notre coefficient, on trouve une base de 2.518. Ainsi notre algorithme a une complexité exponentielle en 2.518^N.

- Ratio

On fait l'hypothèse - en suivant le test de puissance - que la complexité est exponentielle. On cherche à déterminer une constante multiplicative. On trouve une constante de 1.21e-07, affinant donc notre approximation. Le ratio de l'approximation et des données est constant ce qui prouve que l'exposant est bien correct.

- Constante

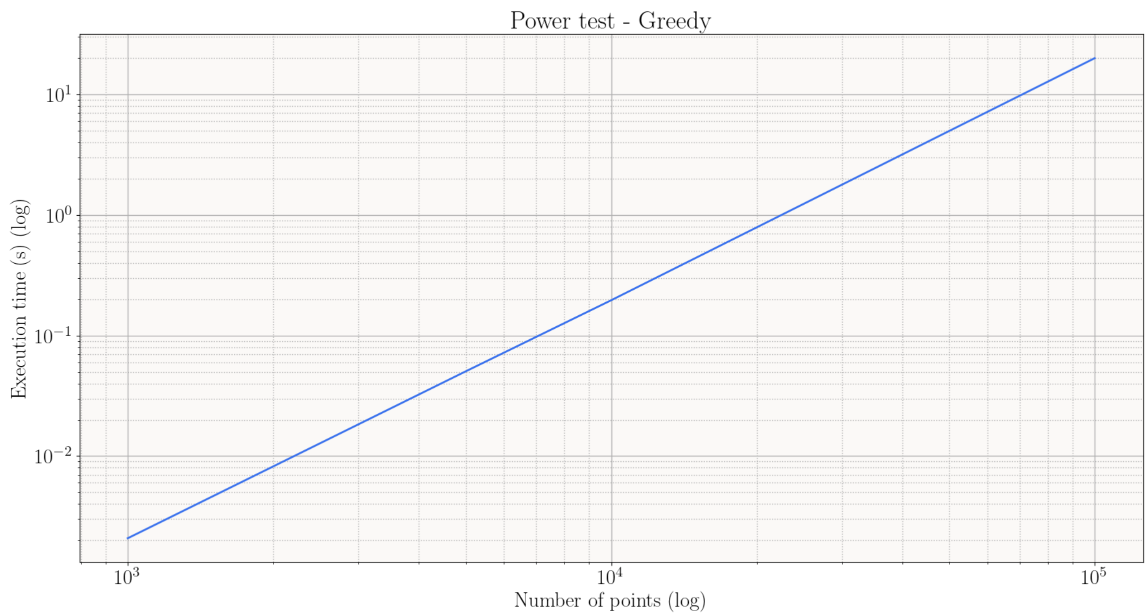
On effectue enfin le test de la constante, utilisant notre approximation précédemment trouvée. L'ordonnée à l'origine est nulle et on trouve une constante multiplicative de 1.031 ce qui affine légèrement l'approximation de la constante multiplicative.

La constante étant très proche de 1, notre hypothèse était donc valide.

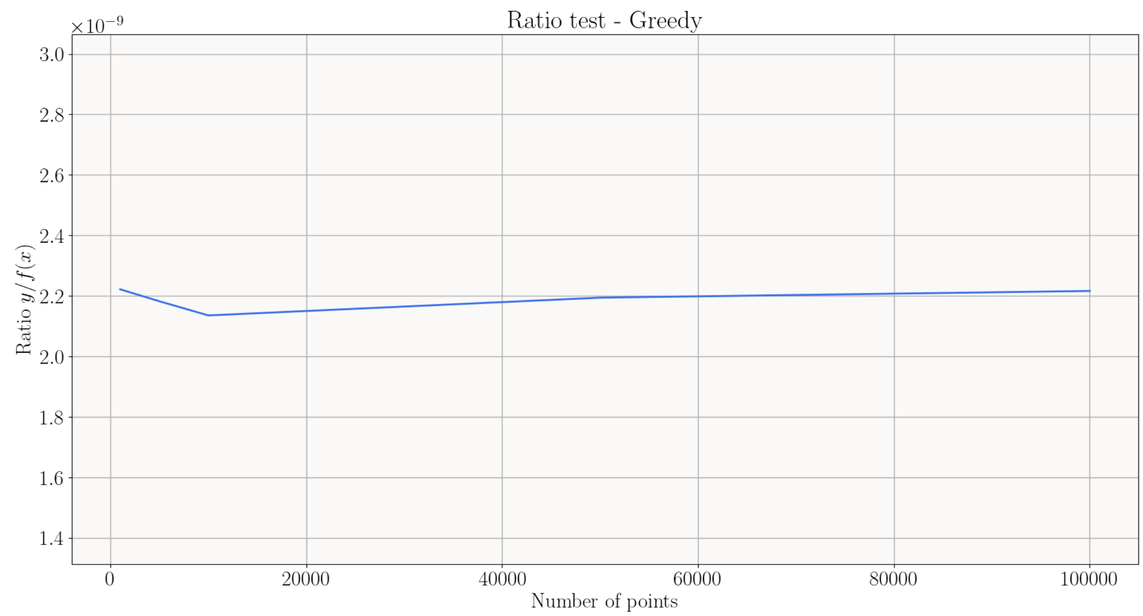
- **Conclusion**

On a une complexité exponentielle, ce qui était attendu au vu de la structure de l'algorithme. La constante a beau être faible - bien qu'encore plus élevée que celles des deux autres algorithmes - la croissance exponentielle rend les exemplaires de taille élevée insolvables.

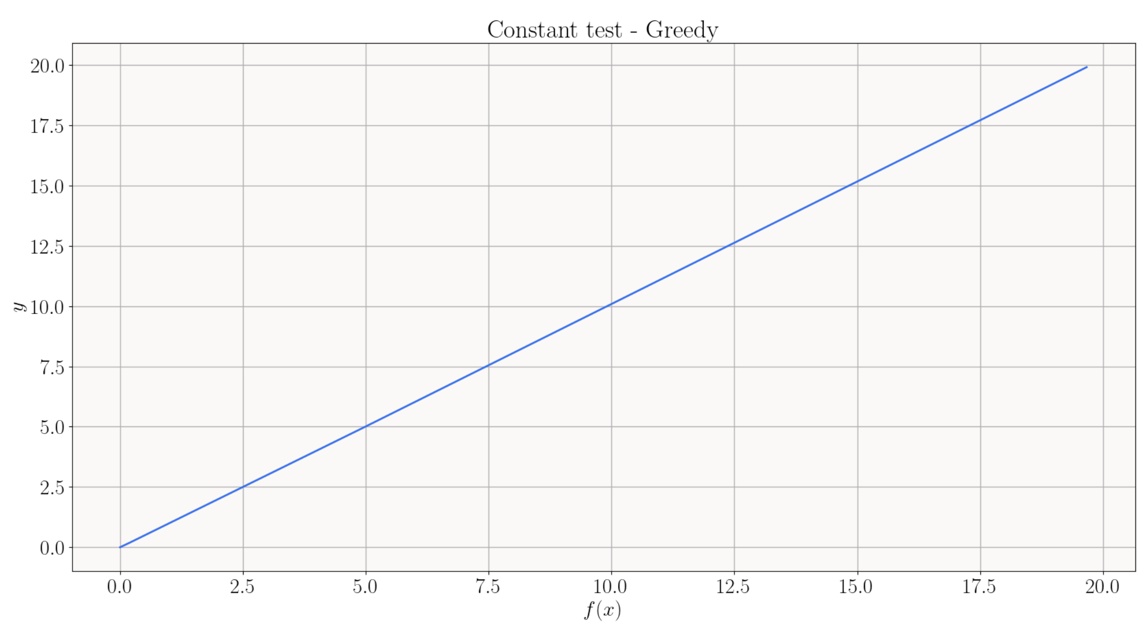
== Graphiques en annexe dans les pages suivantes ==



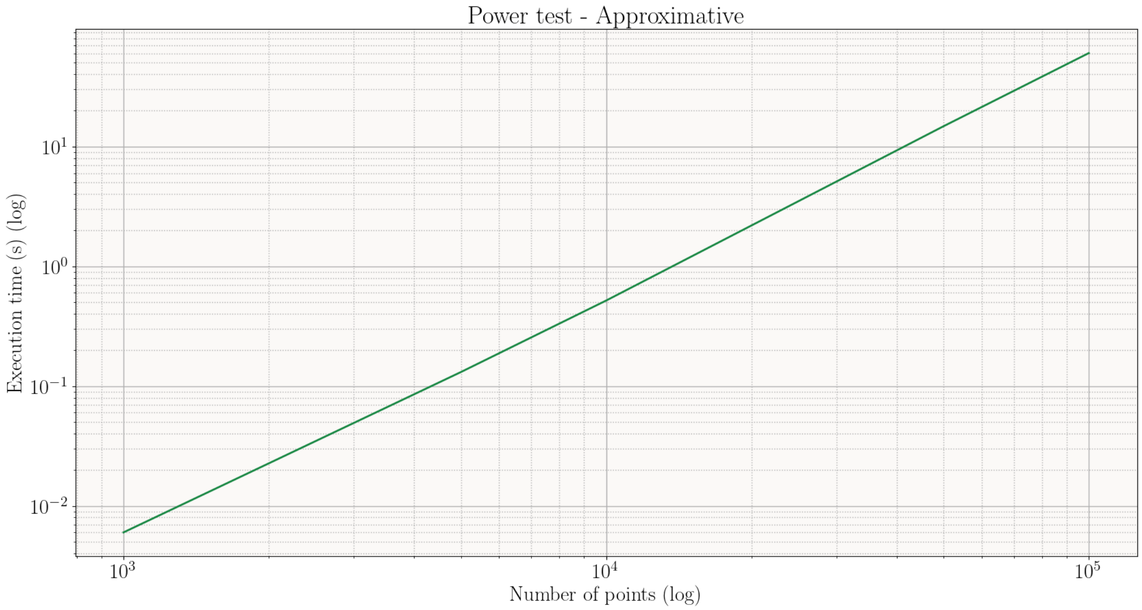
Power test - Greedy
Execution time (s) (log)
Number of points (log)
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{3}
10^{4}
10^{5}



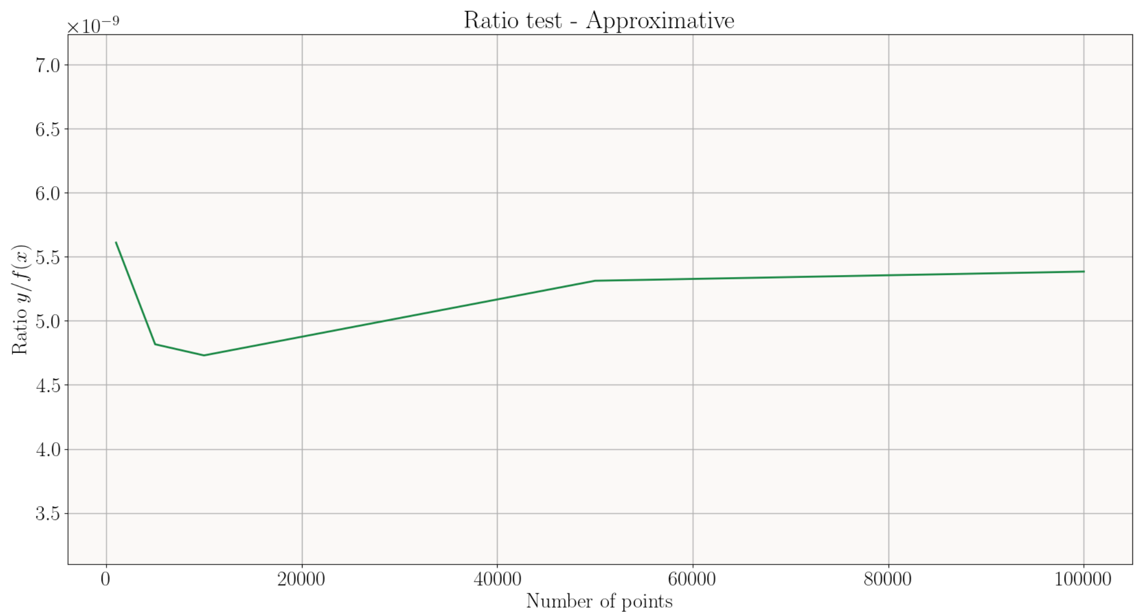
Ratio test - Greedy
$\times 10^{-9}$
Ratio $y/f(x)$
Number of points
3.0
2.8
2.6
2.4
2.2
2.0
1.8
1.6
1.4
0
20000
40000
60000
80000
100000



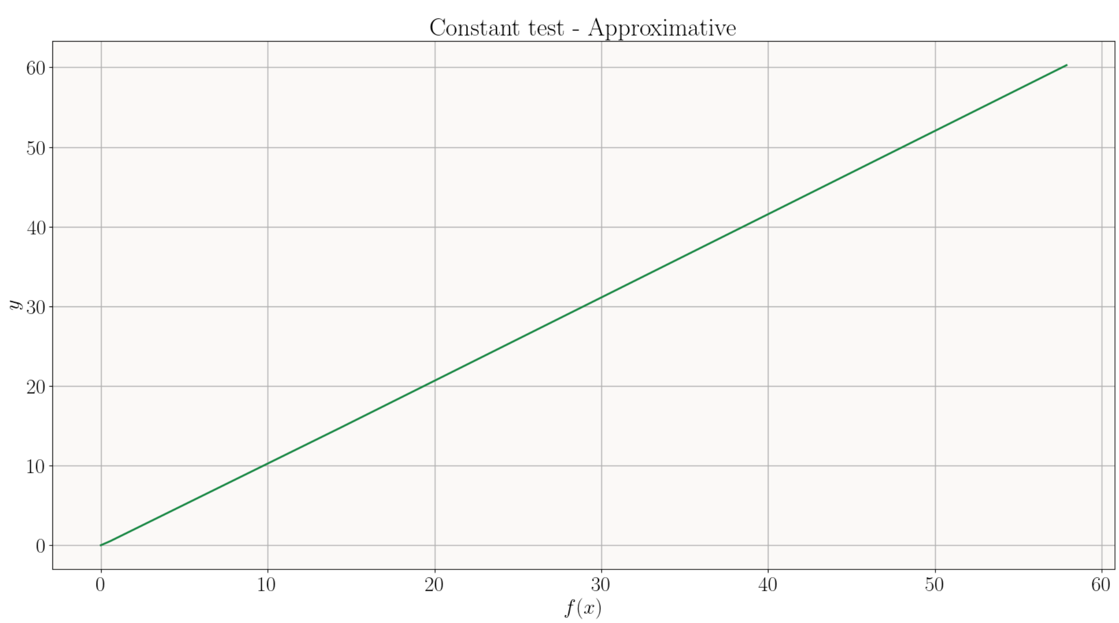
Constant test - Greedy
y
$f(x)$
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0



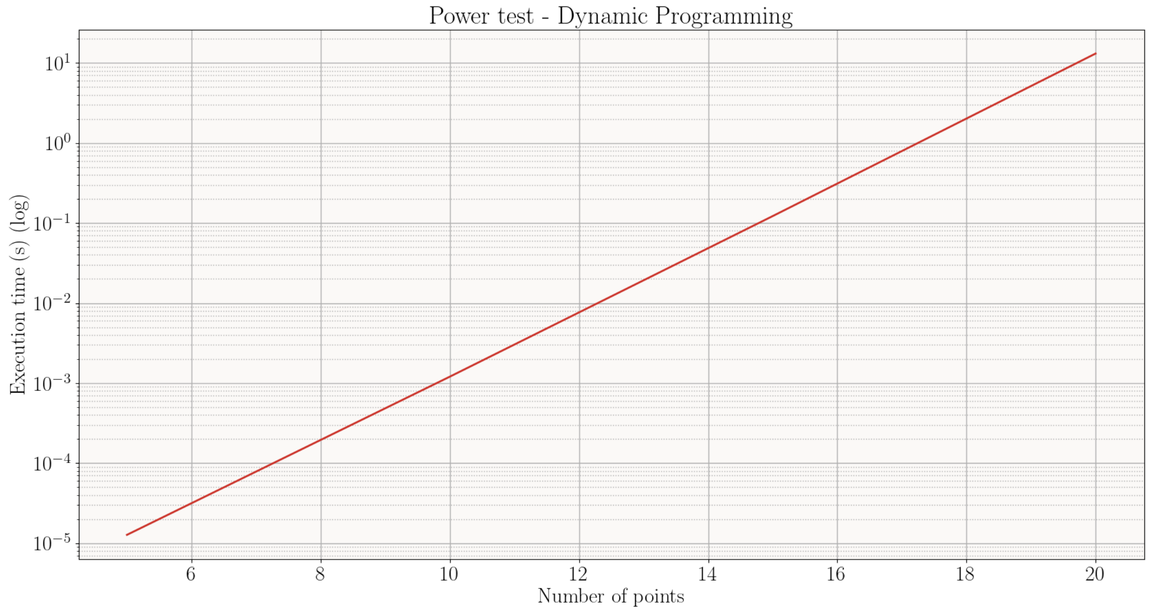
Power test - Approximative
Execution time (s) (log)
Number of points (log)
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{3}
10^{4}
10^{5}



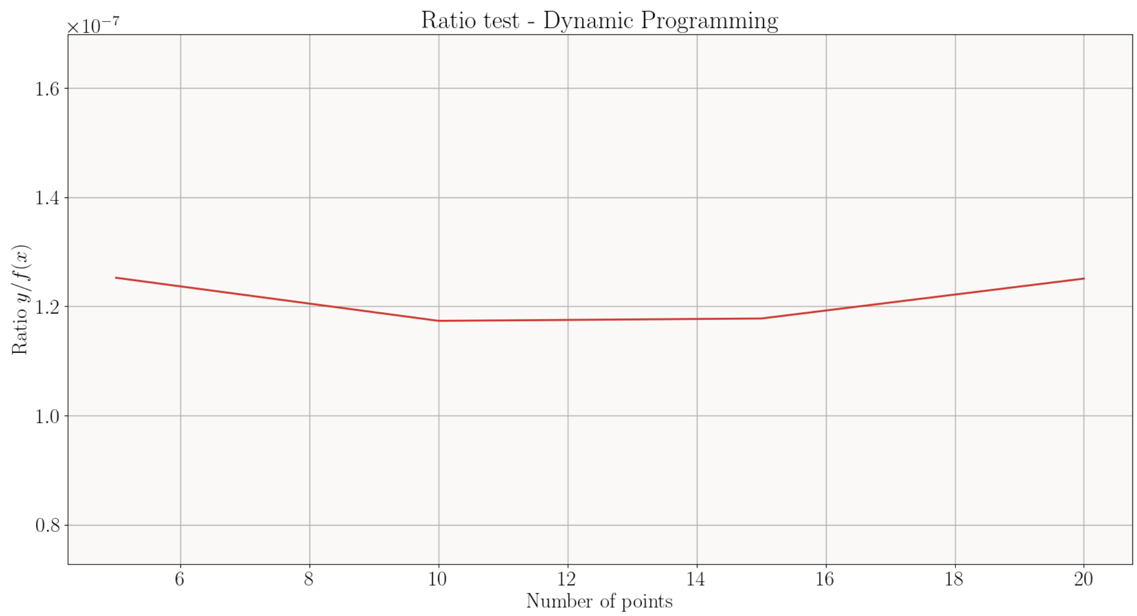
Ratio test - Approximative
$\times 10^{-9}$
Ratio $y/f(x)$
Number of points
7.0
6.5
6.0
5.5
5.0
4.5
4.0
3.5
0
20000
40000
60000
80000
100000



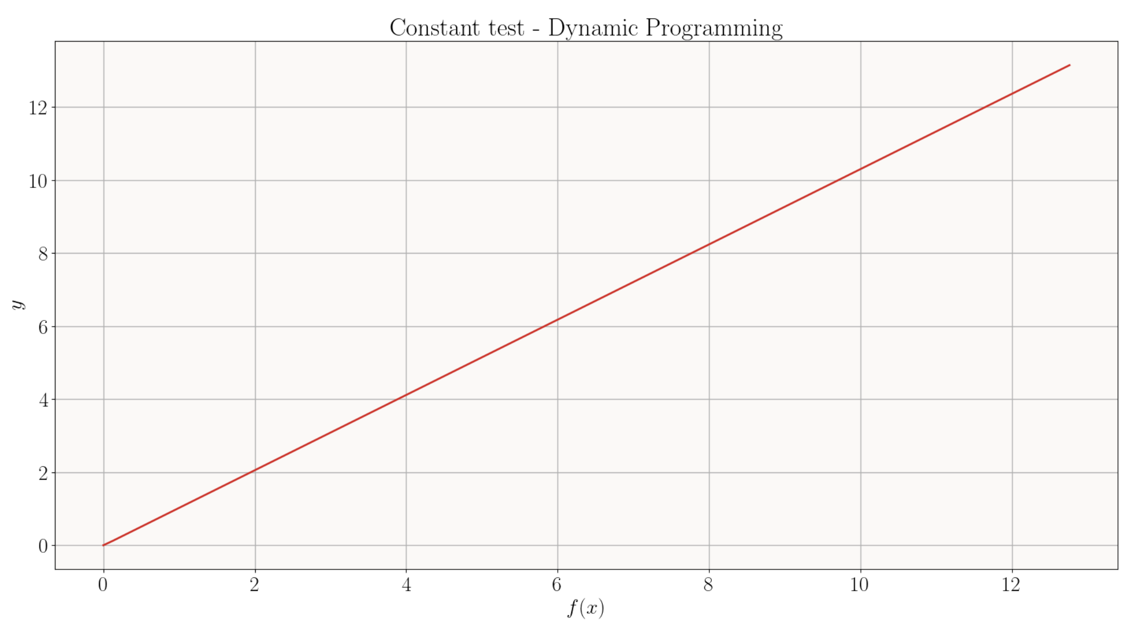
Constant test - Approximative
y
$f(x)$
60
50
40
30
20
10
0
0
10
20
30
40
50
60



Power test - Dynamic Programming
Execution time (s) (log)
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
6
8
10
12
14
16
18
20
Number of points



Ratio test - Dynamic Programming
$\times10^{-7}$
Ratio $y/f(x)$
1.6
1.4
1.2
1.0
0.8
6
8
10
12
14
16
18
20
Number of points



Constant test - Dynamic Programming
y
12
10
8
6
4
2
0
0
2
4
6
8
10
12
$f(x)$