Convex Hull Algorithm Evaluation

Blake Gillian
2024-04-07

Introduction

Background

Jarvis' March and Graham's Scan are algorithms used to compute the convex hull of a given set of points.

Following from sections 1a and 1b, both algorithms were implemented in two dimensions. Thus, the aim of this report is to evaluate their actual and expected performance under a range of test conditions, and to compare the efficacy of their respective approaches.

Experimental Design

The basic operation of 'a comparison between the angles of points' was chosen. To evaluate the performance of each algorithm, the number of basic operations made was measured in three different conditions, at three different sizes. For each size and condition, the average of three trails was taken.

The input sizes of 4, 16 and 256 were chosen to demonstrate the changing cost of each algorithm, with 4 being the minimum number of points to require a convex hull algorithm, and 256 representing a problem of significant size.

The three different conditions were: points distributed randomly, points distributed randomly on a circle, and points distributed randomly within a four point hull. These conditions were chosen to demonstrate the best, worst and average case performance of each algorithm.

In the Jarvis' March code, note that two comparisons between points occur for each n in the case that a given point is not counterclockwise, or already in the convex hull. This comparison could have been avoided by sorting the points by distance before evaluating their angle, such that the closest points are evaluated first. Thus, the second comparison is not a necessity for the Jarvis' March algorithm, and only a feature of this specific implementation. For this reason, and to preserve the O(nh) asymptotic complexity, this second operation was not included in the results.

Hypotheses

It is expected that since Jarvis' March is O(nh), where h is the number of points on the hull, it will perform better than Graham's scan (which is O($n \log_2(n)$), as dictated by quick sort) on hulls of a size smaller than $\log_2(n)$.

Experimental Results

Jarvis' March Averages

	Small (4)	Medium (16)	Large (256)
Random Distribution	14	116.67	4010.67
Random on a Circle	14	256	57940
Random in a Hull	14	64	1109.33

Graham's Scan Averages

	Small (4)	Medium (16)	Large (256)
Random Distribution	2.33	41.67	2191.66
Random on a Circle	2.67	39.33	2210.67
Random in a Hull	3	47	2210.67

Discussion

Choice of Data Structure

The Graham's Scan algorithm was implemented with two integer arrays of length n, being the number of points evaluated. This data structure was chosen in consideration of the quick sort algorithm, allowing for non-linear traversal of the list items and in place sorting. While algorithms such as merge sort have better worst-case asymptotic complexities, their implementation in practical applications is usually slower than quick sort. For this reason, if the points lists given were expected to already be sorted by polar coordinates in some cases, a linked list would have been used to store the points.

Additionally, stacks are used to temporarily store point indices as set out by the Graham's scan algorithm. The stack was implemented with an integer array of length n, being the maximum number of points stored at one time (i.e. if all points given form a convex hull). The space allocated to the stack could have potentially been reduced with the use of a linked list, as in the average case it is rare for all n elements be in use. However, whilst this may reduce the average spacial complexity of the implementation, operations on linked lists are more complicated and thus require a greater number of basic operations (assignment statements).

The Jarvis' March algorithm was also implemented with two integer arrays of length n. The use of linked lists would not have made any significant difference to the program, except in the cost of transferring the given problem into another data structure.

Both implementations of Jarvis' March and Graham's Scan store the finalized convex hull as a linked list. Since each set of points has only one convex hull, the cost of this decision is the same for both algorithms. Whilst another data structure (such as a two dimensional array, or queue) may have been used, the doubly linked list was chosen for its ease of implementation and versatility (i.e. in the direction of traversal).

The spacial complexity of both algorithms is heavily dependent on the method of implementation. However, because Graham's Scan uses a stack, in addition to a data structure to store points, it will generally require more space. Furthermore, with the requirement to sort points before processing, Graham's Scan may require another O(n) for working memory (such as with Merge Sort[2]). Consequently, Jarvis' March generally requires less memory than Graham's Scan, and a smaller variety of data structures to be implemented.

Experimental Evaluation

The number of basic operations made by Jarvis' March approximates the O(nh) of its time complexity. Accordingly, for large values of n, its cost is proportional to the number of points in the hull, and thus heavily dependent on the condition of the input. This is seen through the 'Large' column, wherein a small hull (of 4 points) grows proportionally to n, whereas a large hull (where $h = n$) approximates $n^2 = 256^2 = 65536 \approx 57940$. The average case occupies a space between both extremes, and grows according to the expected number of vertices of a random convex hull (this is O($n^{1/3}$) for a disc and O($\log(n)$) for a square, formed from n points[1]). Intuitively, it can be understood that random points placed on a disk would have a greater chance of being a part of a convex hull than those placed on a square - or conversely, that points arranged on the perimeter of a circle have to move a greater distance from the perimeter to exit the convex hull, than those on the perimeter of a square. Practically, for $n \approx 2000$, $h = 0.02 * n$ in circles, and $h = 0.006 * n$ in squares[3].

By contrast, the number of operations used by Graham's Scan consistently approximates $n \log_2(n)$. For instance, $(2210.67)/(\log(256) * 256) = 1.079 \approx 1$. This relationship is independent to the distribution of points, and occurs at all input sizes (though grows closer as $n \to \infty$). As mentioned in the *Experimental Design* section, this asymptotic relationship is entirely dependent on the sorting algorithm used to implement Graham's Scan. Due to the quick sort algorithm, the average number of basic operations is O($n \log(n)$) on a random set of points.

Note that in the 'Random Distribution' case, whilst Jarvis' March's Large 4011 operations is higher than Graham's Scan's 2192, its growth in relation to the Small and Medium cases is smaller. The Large dataset is thus not fully indicative of the asymptotic growth of each algorithm, and it should be expected that for datasets of an even greater size, Jarvis' March would outperform Graham's scan for points in a random distribution in a square.

CONCLUSIONS

Whilst the performance of Jarvis' March is heavily dependent on the condition of its input, Graham's scan consistently performs according to the complexity of the sorting algorithm used. Accordingly, the asymptotic growth of Jarvis' Scan is significantly worse than Graham's Scan when the convex hull of a given set of points is large in relation to the number of points, and better when the hull is composed of only a small number of points. With a random set of points, the shape of their distribution (i.e. in a disc, polygon, ball, etc.[1]) determines the efficacy of Jarvis' March - any condition where the expected number of vertices in the convex hull is O($\log_2(n)$) will have the same asymptotic growth as Graham's Scan[3]. Thus, the results substantiate the hypothesis that Jarvis' March will perform better than Graham's scan on hulls of a size smaller than $\log_2(n)$.

For smaller input sizes, the number of basic operations used by Jarvis' March is always higher than Graham's Scan. In this case, and for the specific instance of convex hulls which are $h =$ O($\log_2(n)$) in relation to n, the best performing algorithm is decided by implementation.

BIBLIOGRAPHY

1. Har-Peled, S. (1997). On the Expected Complexity of Random Convex Hulls. [online] Available at: https://sarielhp.org/p/notes/97/rand_hull/rand_hull.pdf [Accessed 7 Apr. 2024].

2. Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C. (2009). Introduction to algorithms. MIT Press.

3. Wicklin, R. (2021). The expected number of points on a convex hull. [online] The DO Loop. Available at: https://blogs.sas.com/content/iml/2021/12/06/expected-number-points-convex-hull.html [Accessed 8 Apr. 2024].

APPENDIX

The following python script was used to generate pseudo-random points that meet the three size and condition criteria.

```python
import random
import math
PI = 3.141592653589;
trials = 1

#RANDOM
print("\nPoints in a square.")
sizes = [4, 16, 256]
for size in sizes: #different input sizes
    for trial in range(0, trials): #number of trials per size.
        print(size)
        for x in range(0, size):
            print(f'{random.randint(-10000,10000)/1000} {random.randint(-10000,10000)/1000}')

print("\nPoints on a circle.")
radius = 10;
for size in sizes: #different input sizes
    for trial in range(0, trials): #number of trials per size.
        print(size)
        for x in range(0, size):
            angle = random.random() * PI * 2

            print(f'{math.sin(angle) * radius:.3f} {math.cos(angle) * radius:.3f}')

print("\nPoints within a 4 point hull.")
edges = [[10.000, 10.000], [10.000, -10.000], [-10.000, 10.000], [-10.000, -10.000]]
remd = 4
times = [random.randint(0, size)]

for size in sizes: #different input sizes
    for trial in range(0, trials): #number of trials per size.
        print(size)
        points = []
        points += edges
        for x in range(0, size-4):
            points.append([random.randint(-10000,10000)/1000, random.randint(-10000,10000)/1000])
        random.shuffle(points)
        for x in points:
            print(f'{x[0]} {x[1]}')
```

A sample of the points generated:

Points in a square.

4 -9.178 -6.371 -5.38 6.69 -5.48 -0.436 2.135 5.263

16 4.999 -1.501 9.66 4.816 2.886 -0.051 0.19 7.083 5.258 5.897 6.462 -4.335 -2.07 -4.596 -1.125 -9.41 -0.509 -5.709 8.604 9.089 -4.544 -6.051 -2.642 -5.801 -7.768 -7.609 -6.116 3.438 -4.448 -3.973 5.267 3.352

256 0.261 2.839 -1.723 -7.952 9.323 6.655 -9.808 -4.19 2.24 -8.145 6.403 -8.907 -5.401 7.197 4.745 2.846 -9.148 7.575 6.819 -7.903 -2.225 -4.539 2.919 -4.817 -6.82 -8.835 3.068 9.895 2.921 5.706 -6.147 5.407 -9.208 -2.518 2.452 1.175 -6.961 -8.421 -9.653 6.839 9.718 5.616 4.952 7.838 6.501 -0.594 -0.153 9.173 -9.393 -2.594 -2.538 -7.785 -0.729 5.594 9.607 2.206 -4.486 3.442 0.899 -3.752 5.954 8.573 -8.187 1.557 -1.776 8.073 -3.613 -3.624 6.732 -7.629 5.62 8.728 -5.278 -5.928 2.53 1.104 8.272 -9.096 -3.373 -5.86 -8.48 8.38 -7.222 5.117 -6.204 2.295 9.157 -0.102 -5.225 2.877 -8.844 4.915 -5.346 9.046 7.663 0.071 -9.109 7.471 -8.425 -4.69 -2.893 -4.285 1.685 7.772 7.058 -0.117 0.034 -5.472 -8.508 1.029 1.162 -3.095 0.475 0.465 3.22 9.932 -6.866 0.125 -2.689 -4.731 -2.832 1.413 -3.414 6.995 -4.508 9.011 -2.683 -7.062 6.145 -3.907 4.981 3.874 -8.514 3.828 3.669 4.649 -4.703 0.661 -6.773 -0.812 4.588 7.755 9.699 -3.132 -1.454 -0.473 -0.902 -3.273 -3.269 6.248 6.899 -8.339 6.329 0.146 -7.628 1.24 5.851 2.358 -6.631 1.517 -8.183 -4.14 6.008 0.39 2.445 2.404 1.157 -6.962 -6.894 -5.344 -9.44 5.996 4.018 5.557 6.045 9.627 7.63 -9.189 3.079 -7.827 4.993 7.723 -2.295 1.115 6.604 -7.833 -0.533 -0.972 -4.91 -2.503 -5.049 -7.014 -3.294 -2.129 -1.752 0.287 -5.666 -6.394 8.21 7.231 -2.903 3.94 -5.991 -7.197 -1.221 9.736 -4.419 9.643 8.007 -7.157 -8.657 5.015 -2.252 0.857 7.491 -5.505 -7.473 9.195 0.155 2.594 9.774 1.362 6.248 -0.59 6.305 -6.305 7.23 5.255 6.329 5.152 -5.283 -8.721 -5.038 -9.559 7.018 -8.412 0.232 -3.019 3.026 -1.77 5.601 3.856 9.786 8.538 -2.59 3.716 -2.418 7.952

8.95 -8.427 7.854 9.002 3.534 5.079 4.515 0.668 2.615 5.051 1.792 0.514 9.974 4.175 -0.078 -7.431 -5.876 7.389 1.703 7.778 8.76 9.616 -5.417 -0.539 1.144 -3.465 3.566 -0.625 3.011 8.805 -5.935 5.307 2.981 9.413 -7.322 1.68 8.431 8.912 -8.155 9.252 -6.6 -7.041 6.573 7.964 7.306 1.951 -0.815 -9.445 8.48 4.851 3.575 -6.646 1.406 -0.579 -0.046 3.604 1.119 -3.615 2.388 -6.799 4.075 9.103 9.614 -2.26 -9.192 -9.61 -3.583 8.505 4.381 3.102 -9.864 9.479 -1.888 3.822 -5.757 -1.988 -4.722 8.881 -8.423 8.725 0.195 3.701 6.18 2.498 -6.562 -5.311 -4.005 2.11 3.388 -2.935 9.74 1.807 4.738 5.936 -2.167 -9.433 5.657 2.491 8.902 -5.766 6.456 -4.728 9.013 9.868 -4.429 -1.258 7.327 -6.133 -1.302 -1.607 6.364 2.361 8.699 9.697 -9.621 -4.614 5.174 1.412 -5.079 8.748 2.415 -9.363 -6.964 -1.767 1.17 8.468 7.871 5.541 6.097 8.715 -0.553 5.569 -3.421 3.358 -7.499 9.102 -4.819 -3.428 -9.161 0.191 8.656 -4.899 -3.014 -4.892 4.75 -0.336 -8.688 6.658 -5.941 -5.149 -3.469 -9.165 -3.696 4.74 2.174 2.687 4.534 8.07 -8.84 3.094 -2.863 6.519 4.142 -0.402 -0.283 -1.518 -4.696 -7.916 0.774 1.467 -1.094 0.276 -6.466 -2.114 2.299 -3.436 7.165 -4.744 4.89 9.603 -0.07 -0.913 2.37 7.457 -8.409 -0.888 -2.124 2.567 -4.541 -0.916 5.704 -2.917 0.073 -9.458 -5.867 -8.336 -2.524 -9.341 -7.288 -6.941 8.812 -9.611 1.036 4.974 -5.392 -3.618 -7.571 -6.569 -2.504 4.167 3.203 -5.239 -7.143 7.98 -0.42 -0.183 7.162 3.765 3.354 -9.178 -5.221 -1.077 -6.261 -5.845 3.018 -2.473 4.229 -6.837 0.716 -7.065 6.178 -5.336 -4.577 2.621 1.48 -0.003 9.133 -0.02 5.112 6.976 9.86 -0.305 6.615 2.864 4.219 -0.286 -8.874 3.051 7.049 -5.961 -4.495 -1.987 6.656 1.338 -2.037 4.12 3.42 4.358 0.474 -1.032 -7.597 6.708 3.857 8.848

Points on a circle.

4 1.386 -9.903 -2.201 9.755 -9.716 2.365 -3.965 -9.180

16 -10.000 -0.015 6.747 -7.381 -1.778 9.841 9.947 1.025 6.103 -7.922 -9.907 1.363 3.595 9.331 -9.238 3.829 2.428 9.701 -4.847 -8.747 9.078 4.195 9.376 3.477 5.700 -8.216 8.920 4.521 -0.819 9.966 4.132 -9.106

256 7.820 6.233 -5.058 8.626 2.853 -9.584 9.514 -3.080 7.191 -6.949 9.663 2.576 3.527 -9.357 7.235 6.903 8.610 5.086 -2.595 -9.657 9.897 -1.429 -8.503 -5.263 -9.298 3.681 9.460 3.241 5.101 -8.601 -8.200 5.724 2.873 -9.578 -2.449 -9.695 -3.966 -9.180 -9.482 3.176 -3.849 9.230 -7.240 6.898 7.581 6.522 -6.355 -7.721 4.680 8.837 9.104 -4.137 7.436 -6.686 8.828 -4.697 -9.526 3.043 -9.051 -4.251 5.620 8.271 -9.793 2.023 6.206 7.841 4.862 8.738 5.780 -8.161 -7.984 -6.021 5.847 8.112 0.528 -9.986 5.103 8.600 9.476 -3.196 4.205 9.073 9.601 -2.797 4.119 9.112 -7.663 -6.424 1.139 -9.935 9.981 0.613 -7.892 6.141 9.201 3.917 -8.475 -5.308 -6.323 7.747 5.954 8.034 -8.792 4.765 3.077 9.515 8.541 -5.201 -9.986 -0.527 -9.983 -0.580 -7.567 6.538 -9.580 2.869 4.583 -8.888 -4.731 8.810 -4.712 8.820 -3.735 -9.276 6.977 7.164 -0.283 9.996 -2.101 9.777 -6.701 7.422 9.308 -3.656 -2.917 9.565 9.874 -1.580 0.186 -9.998 9.947 -1.023 -8.340 5.518 2.733 9.619 -4.820 8.762 -7.998 -6.003 8.209 5.711 8.336 5.524 6.645 7.473 -5.564 8.309 1.558 9.878 -5.040 -8.637 5.976 -8.018 9.997 0.258 5.887 -8.083 7.637 6.455 8.230 5.680 -9.657 -2.596 9.986 -0.524 5.400 -8.417 3.579 -9.337 -3.333 -9.428 5.028 -8.644 -9.704 2.415 -8.836 4.683 -2.475 -9.689 -7.431 6.692 -9.983 -0.588 -0.537 9.986 8.166 5.772 -9.778 2.094 3.373 9.414 -7.410 -6.715 4.676 8.839 3.028 9.531 2.289 9.734 8.703 4.926 -1.042 9.946 -1.331 -9.911 -0.360 9.994 -3.523 9.359 9.950 -1.003 8.716 4.903 9.854 1.703 -7.156 -6.986 -1.142 -9.935 2.707 9.627 -9.184 -3.957 -9.589 -2.839 -9.177 -3.973 9.821 -1.883 -0.112 9.999 -7.963 -6.049 -0.337 -9.994 8.747 4.846 -8.534 5.212 -5.691 -8.223 -6.509 7.592 1.492 9.888 5.219 8.530 2.646 9.644 7.609 -6.489 -9.763 -2.166 -9.481 -3.181 9.997 0.264 -9.003 4.353 8.217 -5.700 -9.601 -2.796 -6.767 -7.363 9.677 2.523 2.850 -9.585 -9.857 1.683 -9.452 -3.264 9.940 -1.091 3.707 9.288 -9.541 -2.996 7.110 -7.032 9.320 3.626 -7.840 6.208 -9.782 2.078 -8.255 5.643 9.987 -0.517 3.225 -9.466 7.142 6.999 -7.583 6.519 -9.069 -4.214 -8.171 5.765 -8.325 -5.540 8.164 -5.775 -4.021 9.156 -9.372 3.487 6.669 -7.452 -4.673 -8.841 -0.471 -9.989 -1.142 9.935 -6.689 7.434 0.988 9.951 4.841 8.750 -6.333 7.739 9.573 2.892 -7.703 -6.377 -9.991 -0.433 -9.454 -3.258 -3.030 9.530 9.085 -4.178 9.322 -3.619 -2.928 -9.562 0.362 9.993 -9.537 -3.009 -6.107 7.918 7.429 -6.694 9.994 -0.353 8.571 -5.152 -1.152 9.933 9.637 -2.670 9.998 -0.184 7.542 6.566 6.495 -7.604 2.544 -9.671 7.049 -7.093 -3.617 9.323 0.725 -9.974 -8.364 -5.481 -6.431 -7.657 -9.927 -1.207 -7.426 6.697 2.335 9.724 -9.991 -0.428 -4.958 8.684 -8.420 5.395 -9.992 0.394 -3.259 9.454 9.666 2.562 9.974 -0.717 3.077 9.515 7.338 6.794 -3.235 9.462 6.801 -7.331 8.973 -4.415 -3.029 9.530 -5.916 8.063 -9.367 -3.500 2.250 9.744 7.848 -6.198 -2.214 -9.752 6.722 7.404 -7.419 -6.705 -9.637 2.669 5.558 -8.313 -0.805 9.968 7.067 7.075 3.261 9.454 -9.980 0.632 7.025 -7.117 -9.432 -3.321 9.444 -3.289 5.970 -8.022 5.652 -8.249 -9.167 3.995 5.448 8.385 9.759 -2.183 -3.971 9.178 6.292 7.773 4.307 9.025 -9.321 3.622 9.291 -3.698 8.768 4.808 -6.697 7.426 7.530 -6.580 8.708 4.916 -9.943 1.070 -9.651 -2.619 9.159 4.015 -0.651 -9.979 3.038 -9.527 -7.906 -6.123 -8.920 4.520 4.080 -9.130 -7.401 6.725 -7.577 6.526 -9.921 -1.253 4.694 8.830 7.138 7.003 5.107 -8.597 -7.874 6.165 8.773 4.799 -9.273 -3.744

Points within a 4 point hull.

4 -10.0 10.0 10.0 10.0 10.0 -10.0 -10.0 -10.0

16 10.0 -10.0 2.559 -5.843 -5.987 -5.065 0.54 -9.543 -10.0 -10.0 -3.824 -1.795 5.604 6.246 -7.64 -8.97 6.83 7.68 -3.237 5.978 8.29 2.759 10.0 10.0 -10.0 10.0 7.738 9.381 7.533 7.859 6.074 5.892

256 -9.192 -7.886 -4.554 -6.357 7.559 -0.811 9.93 -2.047 7.34 9.865 -2.602 -5.869 3.421 4.251 5.052 0.025 1.767 3.62 -3.39 -0.046 -8.341 -1.122 5.464 -7.705 4.485 1.541 2.786 4.195 -4.842 -3.645 -7.435 3.578 -7.489 -6.949 5.549 -8.987 1.795 1.16 -5.031 3.705 -9.841 -5.62 2.048 -0.338 7.538 -7.797 -3.667 3.165 -2.477 4.291 -8.099 -8.224 5.147 0.417 1.931 -2.715 -3.574 -1.11 0.024 -2.249 0.031 6.806 -8.163 5.919 -7.006 5.302 -6.484 -2.991 -6.361 -4.689 -4.348 -1.99 9.302 -9.394 -9.475 7.286 5.604 -5.198 -3.096 -9.234 3.157 0.493 6.447 0.791 -8.682 5.562 -8.876 -0.042 -7.391 9.916 -5.976 2.163 1.927 -4.9 -9.248 3.046 -0.763 -9.065 1.185 -7.704 -7.01 -8.811 -2.165 -5.76 -2.509 8.781 0.285 2.721 4.16 -6.145 -7.608 7.663 -2.429 2.397 3.093 -5.749 -1.75 -2.342 6.945 -8.638 -5.543 -0.289 4.936 6.993 3.149 -0.031 10.0 -10.0 8.311 -9.908 4.679 7.701 8.927 8.86 2.498 4.099 3.374 4.338 8.585 -5.312 -2.305 2.79 -8.443 4.919 1.675 8.864 -0.522 9.458 3.477 -4.464 -9.47 6.331

-0.804 8.274 6.496 6.816 10.0 10.0 -8.019 -3.992 4.424 3.208 1.287 1.332 1.711 6.718 -1.47 -7.03 -6.095 -8.757 3.69 -1.643 9.264 -6.29 -1.984 -7.28 -0.416 1.628 3.54 1.786 7.823 -9.476 -2.933 8.303 0.04 -4.851 -3.153 -1.719 7.014 -8.59 1.589 -2.355 -3.835 0.64 -9.814 -3.704 -0.738 4.542 -0.255 -0.892 1.781 0.476 -2.745 0.879 2.903 8.423 7.206 -8.957 9.759 -9.809 4.794 -9.44 7.019 -7.646 9.941 -6.267 1.278 -6.31 -6.896 4.371 8.345 -3.902 1.545 8.346 -5.746 0.492 1.843 2.669 -8.878 5.216 5.47 0.899 -8.625 -7.638 -0.379 0.313 5.16 -6.045 -3.555 -3.344 -5.163 8.173 0.671 -7.265 1.063 3.078 7.077 5.969 9.026 9.545 -5.501 -9.017 -8.066 -5.099 -9.192 3.569 0.036 6.79 -5.108 -5.13 -9.075 -8.706 -5.716 6.61 -8.344 -3.592 7.973 -1.386 -7.578 -8.373 -2.11 3.244 -3.693 1.301 5.781 2.014 -1.812 2.727 -3.011 -1.091 5.241 -1.391 -1.052 -9.861 -7.772 1.603 9.247 9.723 -1.682 8.928 -5.355 -2.367 -8.245 -3.202 9.285 3.424 -8.647 -7.646 -4.004 -9.281 4.215 -8.2 -1.374 -5.361 -7.26 4.657 -2.85 -5.709 0.445 7.119 5.083 4.466 4.577 6.543 1.249 8.832 -7.77 -2.923 -9.867 -6.818 -0.068 -5.316 1.257 4.792 -4.685 3.91 5.668 1.116 6.582 7.025 4.309 -0.074 -1.508 2.032 -5.721 3.321 -6.574 -2.649 -8.307 -0.521 4.954 -5.497 3.658 -2.604 -10.0 10.0 4.877 -8.803 6.606 9.035 -1.374 -6.387 -5.31 -1.137 7.719 -8.088 8.368 7.876 4.063 -7.333 -10.0 -10.0 -7.556 -3.095 7.766 -6.381 -0.941 -2.796 -9.001 1.407 6.28 3.066 -8.644 -5.471 7.988 5.931 0.715 3.989 -4.414 -5.196 -4.274 -3.17 -6.642 2.913 2.86 1.771 -4.018 -1.914 -2.521 -7.627 -5.432 -8.63 8.157 -7.099 1.997 0.39 5.098 -9.919 -9.997 -5.107 3.837 0.424 2.85 0.11 -2.021 -0.495 0.284 8.178 -0.492 -2.666 -2.645 5.958 4.457 5.886 8.893 1.931 2.997 3.539 -4.641 -1.099 8.789 5.483 2.314 -1.813 -9.657 -5.339 -1.094 7.817 5.418 -2.682 3.109 -2.114 4.339 9.124 -5.837 -4.252 5.684 3.589 6.641 -1.596 -3.443 -7.528 -3.114 8.601 -1.898 -0.599 -2.388 2.546 7.588 1.571 -7.715 7.177 -9.163 -5.413 2.388 1.706 -10.0 8.576 -4.291 0.552 -6.115 -1.746 7.387 -5.508 7.84 9.554 -3.492 -6.288 5.828 -7.487 -3.321 9.802 7.52 8.142 9.245 7.137 4.864 2.115 7.76 2.504 -5.401 1.663 -9.896 8.229 -3.446 1.291 5.06 -4.611 4.032 -1.582 -9.022 8.169 1.83 7.488 -2.716 2.434 -9.52 2.194 5.596 3.419 -9.381 -7.209 -4.211 6.736 -8.823 4.329 9.823 0.239 5.914 1.419 -0.523 2.971