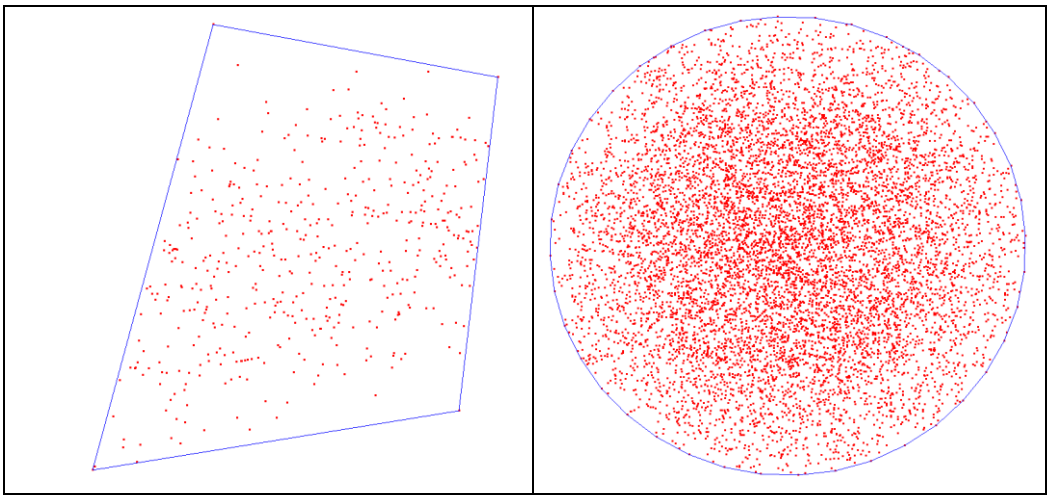
COSC262 Algorithms
Assignment: Convex Hulls

Max. Marks 20

Due: 5pm, 3 June 2016



1. Outline

In this assignment, you will implement methods for computing the convex hulls of two-dimensional points, and validate the algorithms using several test cases. You will then perform an experimental analysis of their time complexity using point datasets, and discuss your findings in a report.

2. Datasets

You are given two data sets: "Set_A" and "Set_B". Each set contains eight data files (each with suffix "`.dat`") containing the coordinates of two-dimensional points. The first line of a data file will contain a single integer number n ($3 \leq n \leq 30000$), specifying the number of points in that file. This will be followed by n lines, each line containing two integers representing the x and y coordinates of a point. The data contained in the files satisfy the following properties:

- Every point has integer coordinates in the range [1, 800]
- There are no duplicate (conincident) points
- Every file has at least three non-collinear points. The data in every file should therefore produce a non-degenerate convex hull.
- All files are editable text files.

The data size n is also included as the numeric value in the file name. The files in "Set_A" (`A_500.dat`, `A_30000.dat` etc.) contain points that generate convex hulls with only a few (typically, 3 or 4) vertices, as shown in the first figure above. Files in "Set_B" contain points that generate convex hulls with larger number of vertices (as in the second figure given above). The data sets are designed in such a way that *none of the edges of the generated convex hulls will contain more than two points.*

3. Program Structure

One of the main tasks of this assignment is the development of a Python program containing the implementations of the following convex hull algorithms:

i. The **Gift Wrap** algorithm based on the method outlined on Slides 1.1: 38-40

ii. The **Graham-Scan** algorithm as outlined on Slides 1.1: 41-46.

iii. A third algorithm for computing convex hulls: This could be any other non-naïve method for computing the convex hull of a set of points (eg., insertion hull, quick hull), or a suggested improvement of the method in (i) or (ii).

A template for your Python program is provided in the file `convexhull.py`. Please do not change the file name or the names of functions included in the file. You may import modules and define additional functions as necessary. Please also use short comment lines to annotate important steps in the program.

4. Algorithm Validation

The functions that implement the convex hull algorithms in your program should produce a list containing the *indices* of convex hull vertices, with each index denoting the position of a vertex in the input list of points (not the sorted list). The first point in the input list will have an index 0, the second point index 1, and so on.

The output list generated by the giftwrap and Grahamscan algorithms should begin with the index of the lowermost point of the convex hull, with subsequent indices referring to points ordered in the anticlockwise direction along the convex hull. If there are two points with minimum *y*-value, select the rightmost point as the starting point.

The expected output corresponding to a given data file can be found in a file with the same name with extension "`.out`" (eg. `A_500.out`). Note that the files contain a sequence of integers giving the indices of the convex hull vertices starting from the lowermost point. The giftwrap and Graham-scan algorithms should produce a list of these indices in exactly the same order. The third algorithm should also generate the required indices of hull vertices, but the order need not be the same as specified in the `.out` file.

To help you visualize the distribution of points in each data file and the shapes of the corresponding convex hulls, the plots of the points and the convex hulls are provided in the file `DataPlots.pdf`.

5. Algorithm Analysis

After validating the algorithms for generating convex hulls, you should perform an experimental analysis of their complexity with respect to input size *n*, by recording the time taken by each of the three methods. The supplied data sets (Set_A and Set_B) contain files with the number of points varying from 10 to 30000. When estimating the time taken by a method, please run the program at least three times for each data file, and

take the average time. For each of the three algorithms, you should generate a graph containing two plots: one for Set_A, and the other for Set_B, showing the variation of time taken with input size. Use the graphs to compare the performance of each algorithm for the two types of data, and also to compare the performance between the three methods. Provide the graphs and a brief outline of your analysis in your report (see next section). Please also try to answer the following questions in your report.

a). How did the results vary for each algorithm? Please give a brief explanation of the trends seen in each graph, unexpected variations if any, and similarities to (or deviations from) the theoretical measures of complexity.

b). Which one of the three algorithms gave the best result in performance analysis? Why did that algorithm perform better than the others?

6. Report (2-4 pages)

Prepare a report detailing your work. The report should include the following sections:

1. Algorithm implementation: In this section, you may discuss any specific data structures or Python functions that you found useful for your implementation. You may also discuss problems encountered while implementing or testing the algorithms, and how you could solve them. Please also give a description of the third algorithm ("`amethod`").
2. Algorithm analysis: Provide the results of the comparative analysis in the form of graphs as described in the previous section, and give your interpretation of the results. If you have proposed an improvement of either the giftwrap algorithm or the Graham-scan algorithm in "`amethod`", describe how the suggested improvement could be seen in experimental results.
3. References Required only if you found any source of information or resource particularly useful for your understanding of the algorithms, program development, or analysis.

7. Marking Scheme

The submitted programs will be tested with our input files which will be different from the ones provided. The points in the data files used for testing will also satisfy all the conditions outlined in Section 2. The distribution of marks among different components of the assignment will be as follows:

Report : 6 marks.
Code - design and correctness of algorithms: 12 marks.
Code – structure, readability, use of proper data structures/functions : 2 marks

8. Assignment Submission

Assignment due date: **3 June 2016**; Drop-dead date: 10th June 2016.

Please submit the source files (`convexhull.py` and any other supplementary files) and the report (in WORD or PDF format), using the assignment submission link on Learn (`learn.canterbury.ac.nz`).

9. Miscellaneous

1. This is not a group project. Your report must represent your own individual work. In particular, students are not permitted to share program source code in any way.

2. Please check regularly on Learn system forums for spec updates and clarifications.

3. Standard departmental regulations regarding dishonest practices and late submissions (1 week “drop dead” with 15% penalty) apply.