# Assignment 1 - Surface Representations

**Name**: Aman Bhardwaj

**Student ID**: 12333472

## Introduction

This report outlines the implementation and analysis of a dense grid to store and manipulate the Signed Distance Function (SDF) for the given shapes: Circle and Rectangle under periodic and reflective boundary conditions. The sign of the distance indicates whether the point is inside (negative) or outside (positive) the surface. Then, the calculations of the surface normal vector and curvature for any given point in the dense SDF grid is carried out. Further, the tasks involve the changes in the SDF due to advancing of the surface by using various ways of advancing the surface (normal subtraction and Engquist-Osher scheme).

## Task Overview

1) Grid Implementation: It involves developing a dense rectangular grid to store the SDF with variable grid extents and grid spacing. Then reflective and periodic boundary conditions are implemented on the SDF. The two surfaces used in this exercise are Circle (centre, radius) and Rectangle (minimum corner, maximum corner).
2) Calculation of Geometric Variables: It involves the calculation of Normal Vector and the Curvature of any given point from the SDF using the Numerical Derivatives.
3) Advancing the Surface: It involves various ways to advance the surface by simply subtracting a velocity value from every grid point and other by using several steps of the Engquist-Osher scheme. In this, Velocity value, V = 10 is kept constant and applied to Circle (with fixed radius = 10) and Rectangle (side lengths = 5, 20). It is to be analysed what effects these schemes have on the result and how the grid spacing influences the results of each scheme. Finally, using the normal vector and curvature generated earlier, behaviour of the rectangle surface is examined over many time steps when it is moved by the velocity vector function and the curvature used instead of the velocity.

## Results and Analysis

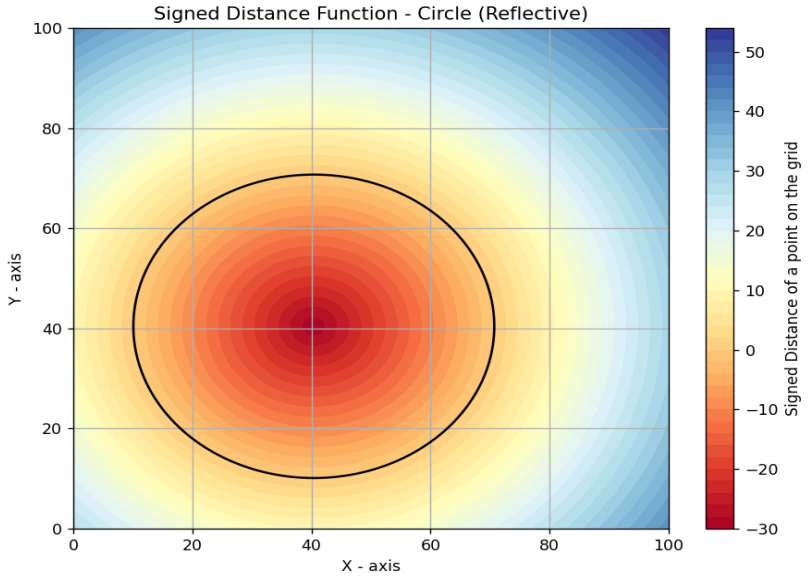
**Task 1**: In this task, the grid is designed to handle different sizes and spacings. Reflective and periodic boundary conditions were implemented using helper functions to map points outside of the grid back inside according to the chosen boundary condition.

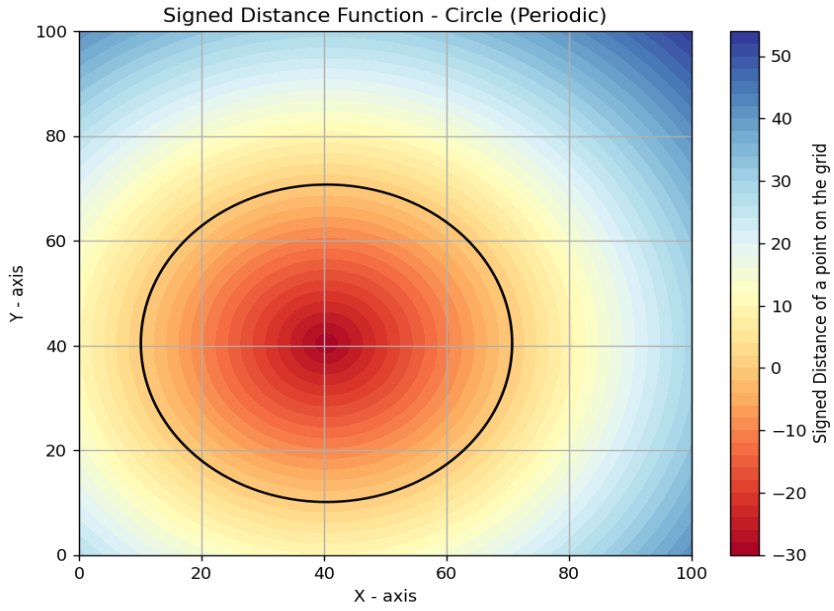
- **Reflective Boundary Condition**: Points outside the grid are mapped inside by reflecting them across the boundary. For example, a point at $x_{max} + d$ is mapped to $x_{max} - (d - \Delta x)$.

- **Periodic Boundary Condition**: Points outside the grid are wrapped around to the opposite side so they appear in the neighbouring grid in the same location as the centre grid. For example, a point at $x_{max} + d$ is mapped to $x_{min} + (d - \Delta x)$.

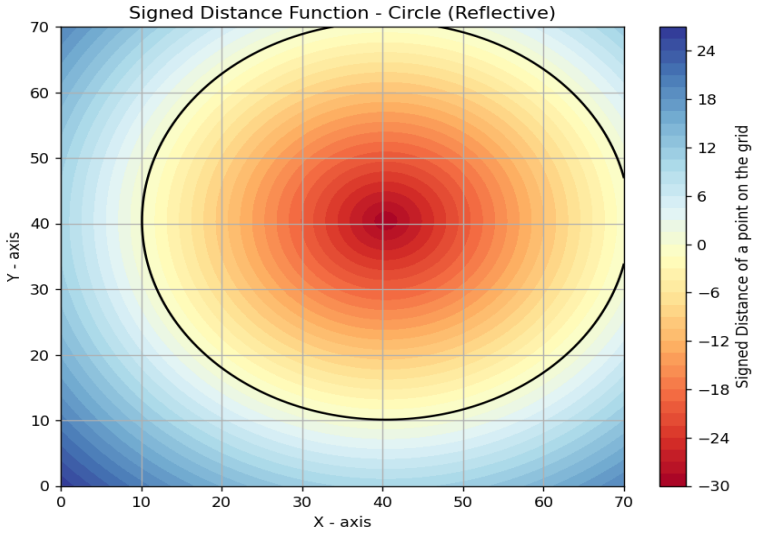
For both circle and rectangle, the SDF is computed and stored in the grid. The SDF value at any grid point represents the shortest distance to the surface, with a negative sign indicating points inside the surface and the positive sign for the points outside the surface.

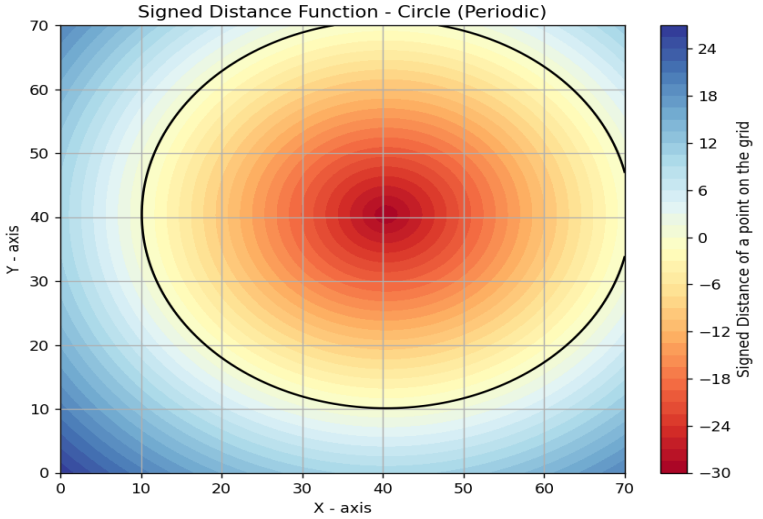
By implementing the code in python, creating a class SDFGrid, I calculated the SDF for both circle and rectangle by making functions to calculate the distance of any point on the grid from these surfaces, then applied the boundary conditions and visualized the results through the command line arguments using the main() function. I used contour plots through matplotlib to visualize the surfaces containing SDF's in which the colours towards the red side represent points inside the surface and those towards the blue side represent point outside the surface. I got the following results for the Circle and the Rectangle:



These figures show the results of both periodic and reflective boundary conditions on SDF of a Circle surface with grid size x= 100, y = 100 whose centre is at (40, 40) and radius is 30 and spacing = 1.

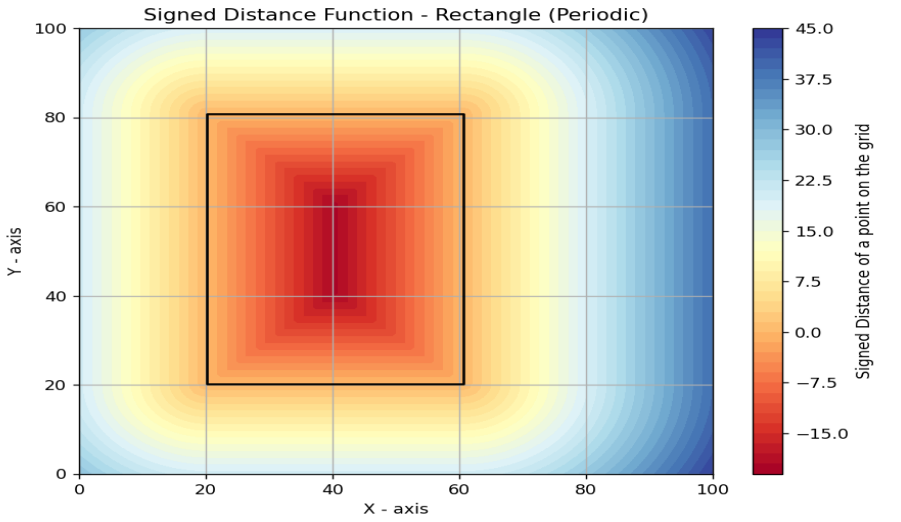


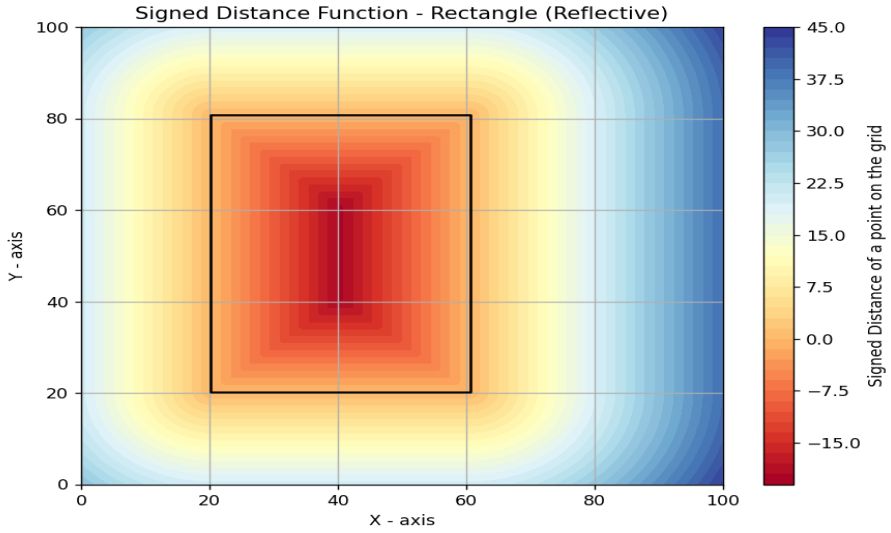




The above figures show the results of both periodic and reflective boundary conditions on SDF of a Circle surface with grid size x= 100, y = 100 whose centre is at (40, 40) and radius is 30 and spacing = 0.7. The negative values show points inside the surface and the positive values represent the points outside the surface. At 0, the point lies on the surface.

So, from these figures, it can be said that higher grid resolution or decreasing the grid spacing by making $\Delta x$ smaller results in more accurate and clear surface representations, as demonstrated by the above figures.

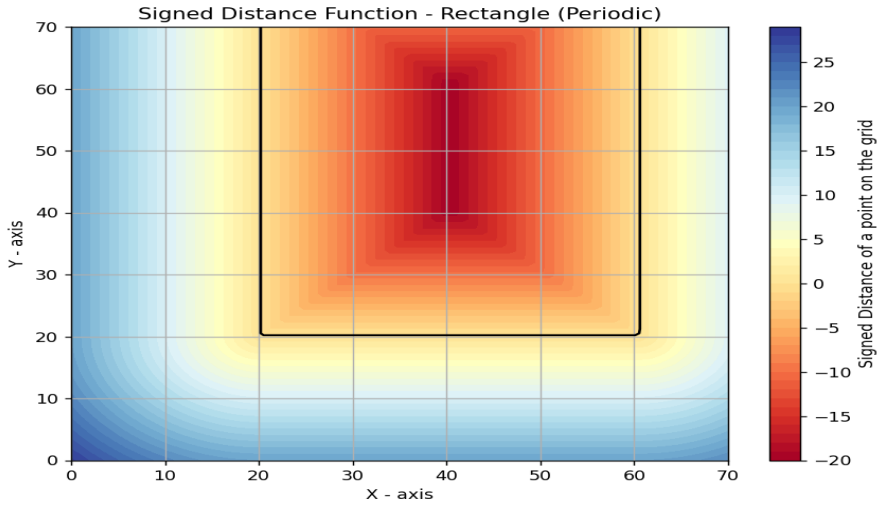
Now, for the rectangle surface, applying the periodic and reflective boundary conditions on the dense SDF grid, with grid size x = 100, y = 100, grid spacing = 1, minimum corner = (20, 20), maximum corner = (60, 80), the results obtained are as follows:

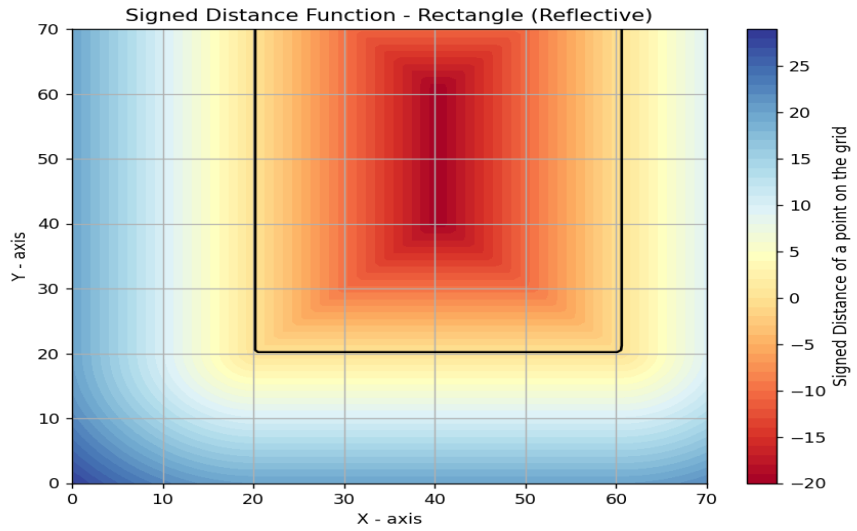



These figures represent the dense rectangular grid containing the SDF of a rectangular surface, where periodic and reflective boundary conditions are applied.

For, the rectangular surface with grid size x = 100, y = 100, grid spacing = 0.7, minimum corner = (20, 20), maximum corner = (60, 80), the results obtained are as follows:






It can be clearly seen from these figures that when the grid spacing is reduced (increased grid resolution so that the grid points are closer), it provides more clear and accurate representation of the rectangular surface as can be seen in the last two figures where I used the spacing as 0.7 keeping the other parameters same.

**Task 2**: Now, for the second task, functions for calculating the normal vector and the curvature of any point on the grid are to be implemented. I used the earlier functions of the SDFGrid class for calculating distance of a point from the surfaces in the grid by importing the SDFGrid class from Task 1. Then, I made functions to compute the numerical derivatives, normal and the curvature and run these calculations on command line through the main() function. The normal function gives the normal vector to these surfaces and the curvature functions computes the curvature for any point on the grid. Here, no boundary conditions are to be used as only calculations for the curvature and normal are to be carried out.

For example, if I give the values for (./grid x-size y-size spacing [Circle | Rectangle] [parameters] x y) as: 100 100 0.7 Rectangle 20 20 60 80 23 27, I get the result as:

**Normal at (23,27): -0.96333i + -0.26832j**

**Curvature at (23,27): 0.23805**

For Circle, using 100 100 0.7 Circle 40 40 30 23 27, I got the results as:

**Normal at (23,27): -0.74966i + -0.66182j**

**Curvature at (23,27): 0.03138**

The above values give the normal vectors and curvatures for circle and rectangle surfaces at a point (23, 27) on the dense rectangular SDF grid.

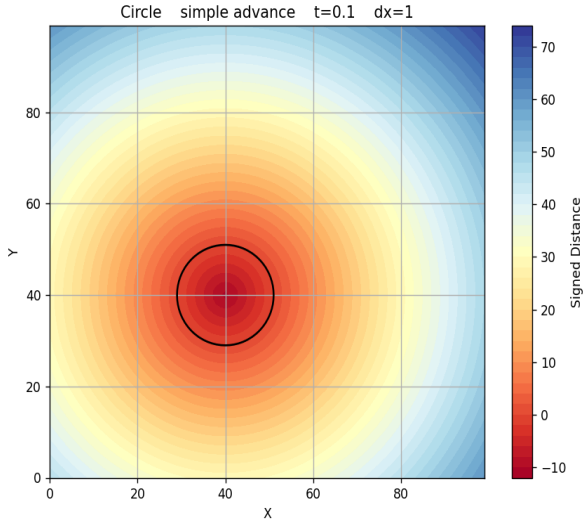
The normal and curvature values are rounded to 5 decimal places.

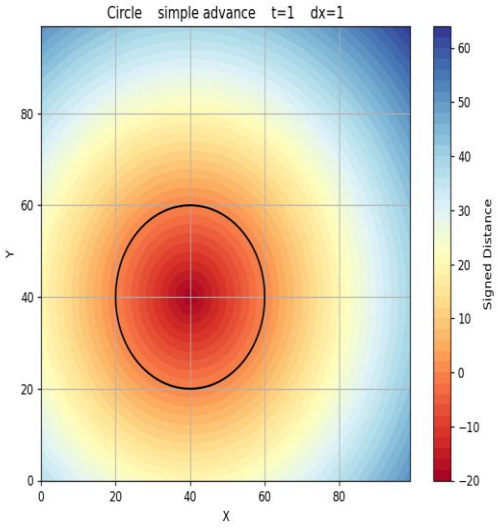
**Task 3**: In this task, different methods of advancing surfaces were analyzed as, Simple Subtraction, which directly subtracts the velocity value from SDF. This method can distort the surfaces as it does not maintain the normalization of the SDF. The other method, Engquist-Osher Scheme, maintains SDF normalization by considering the gradient and ensures stable surface movement. I used and imported the SDFGrid class from Task 2 where numerical derivatives, normal and curvature were calculated and then created functions for simple advance and Engquist-Osher method and a function for comparing the advancements. Further I created functions for Velocity field and using the curvature as velocity.

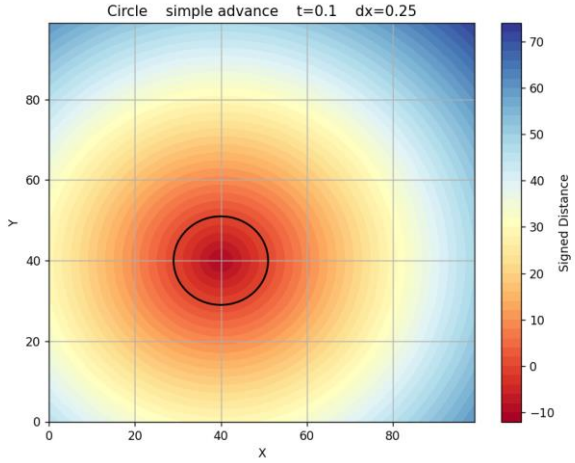
The results were obtained by comparing these methods and using constant positive velocity (V = 10) greater than many grid spacings at the fixed resolutions $\Delta x = (1, 0.25)$ for the times t = (0.1, 1). The radius of circle is taken as 10 and side lengths of the rectangle as 5 and 20. So, the following plots were obtained after using these methods at the given resolutions:

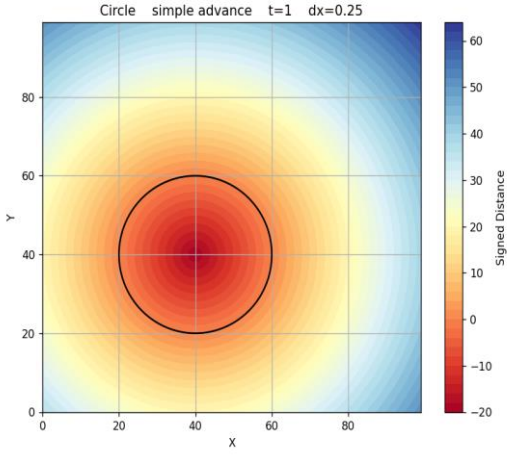
For the circle the command line arguments are given as: ./Grid[x-size(n_x) y-size(n_y)] [Circle / Rectangle] [parameters] which are given as:

100 100 Circle 40 40



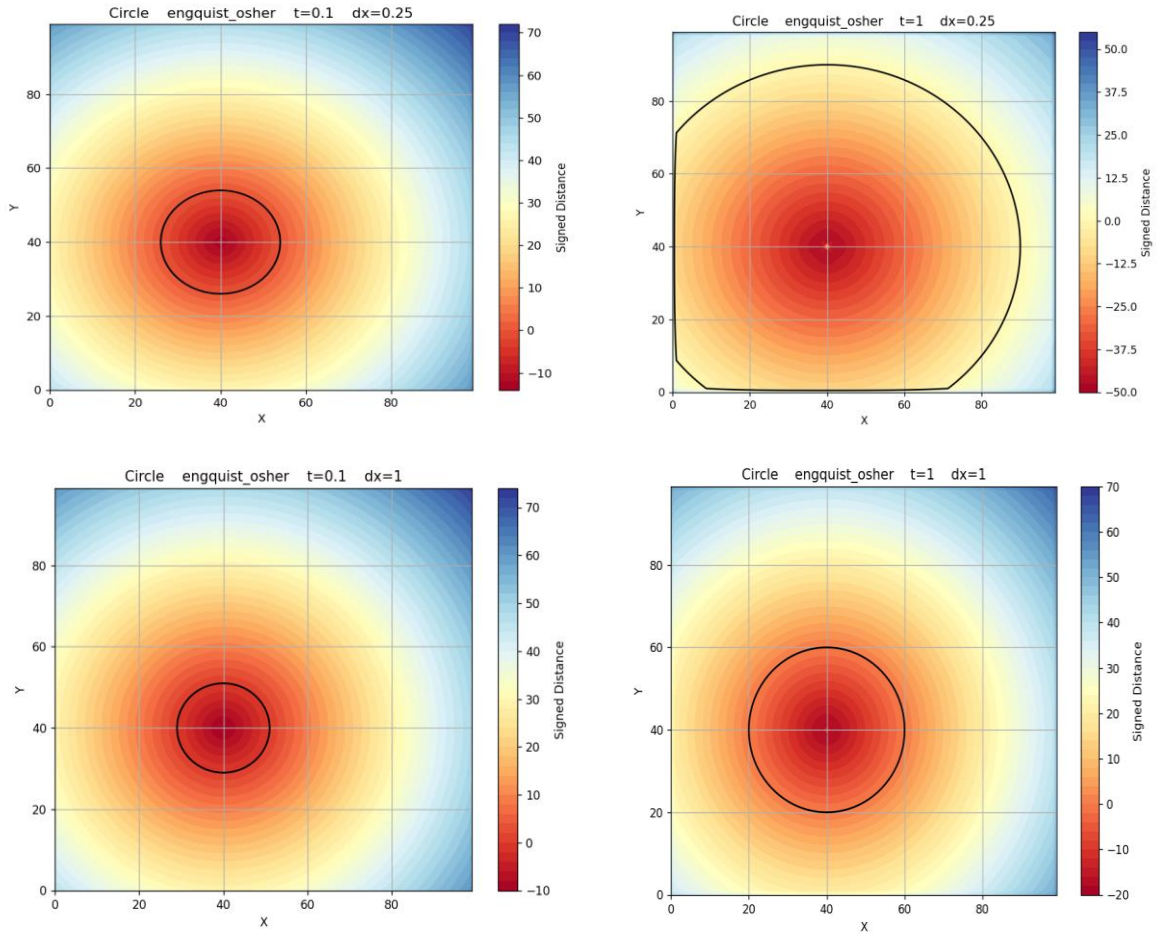



It can be observed from the above figures, that for t = 0.1 with changing the grid spacing, there is no effect on the result for the circle but, for the same spacing but increasing or longer times, from t = 0 to t = 1, the effects are more visible on the SDF of the circle surface using the simple subtraction method as above.

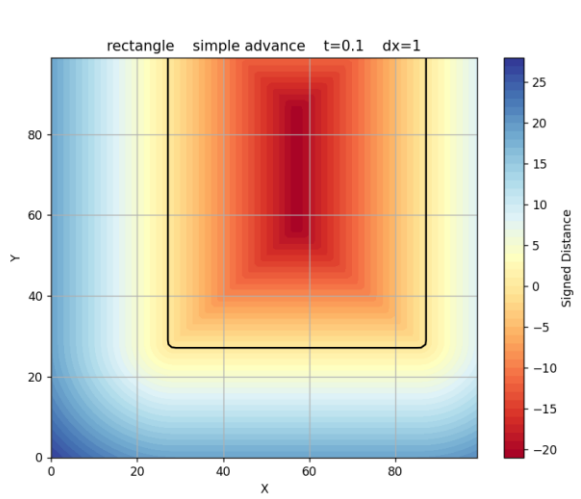
Now, using the Engquist-Osher scheme, following plots are obtained for the same command line arguments (100 100 Circle 40 40):

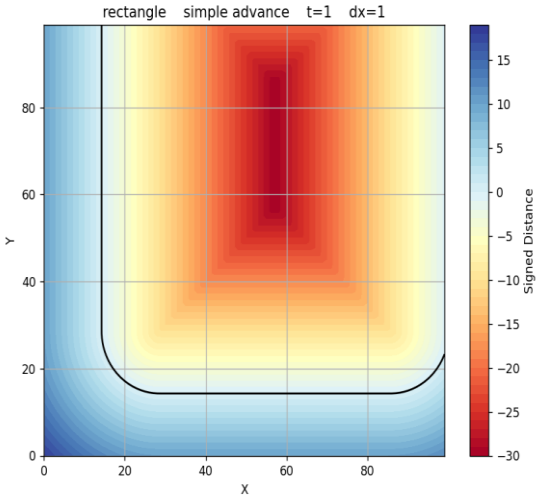


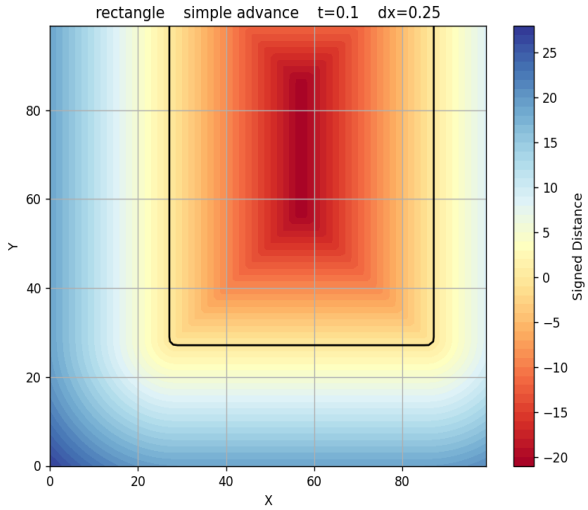

It can be noticed from the above that at t = 0.1and changing the spacing, the advancement is smooth and accurate. Even after longer time t = 1, there are some changes in the surface with lesser spacing containing some distortions on the circle surface, but as the spacing is increased from 0.25 to 1, the results produced are more accurate and precise and the advance is smooth.

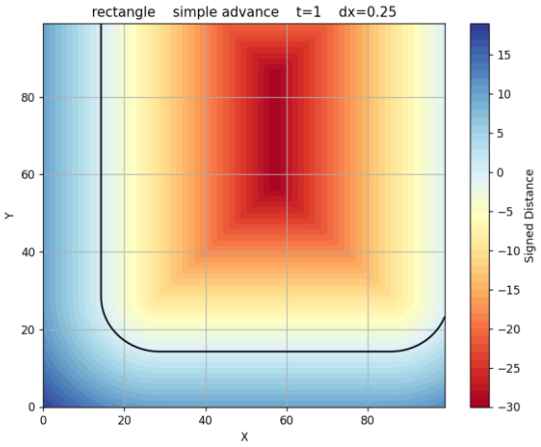
Now, for the rectangle, the command line arguments are given as: ./Grid[x-size(n_x) y-size(n_y)] [Circle / Rectangle] [parameters] which are given as:

100 100 Rectangle 20 20 and the following results were obtained for the two advancement methods:



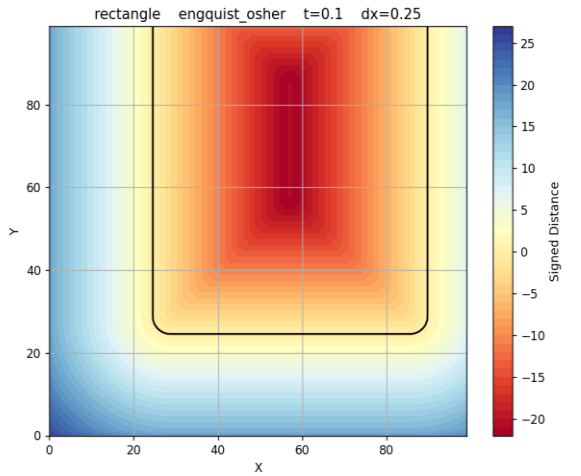



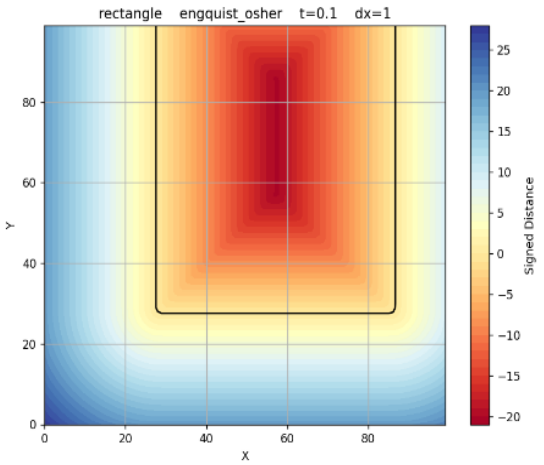



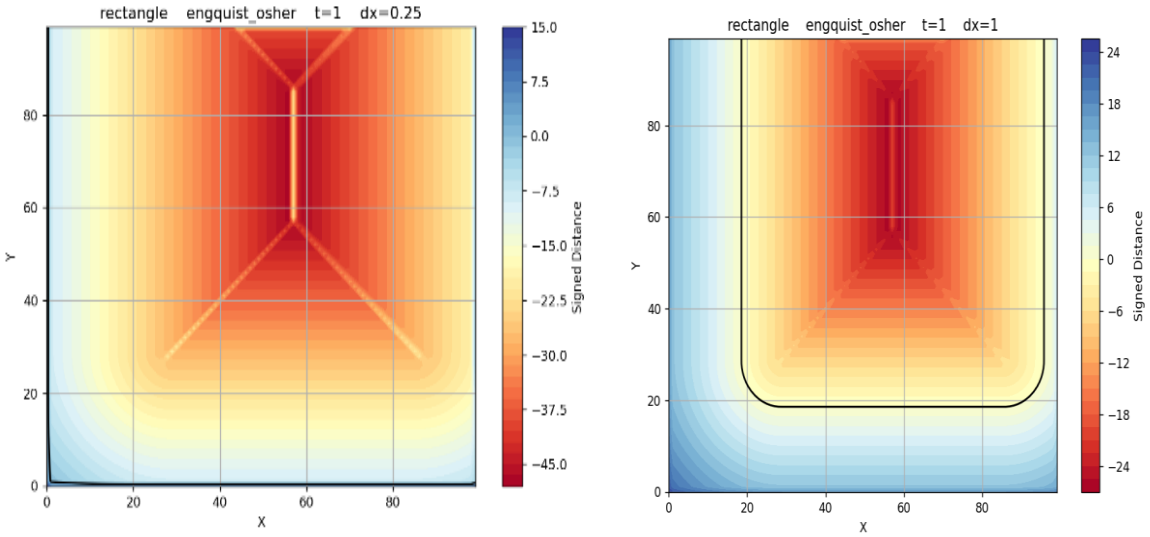




At t = 0.1, in simple advance, there is not much effect on the rectangle, but later at t = 1 and even at lower and higher spacing, distortion effects can be seen on the corners of the rectangle which signifies that at longer times, simple advance results in more finer effects of the surface.

Now, for the Engquist-Osher scheme, following plots are obtained for the same command line arguments (100 100 Rectangle 20 20):



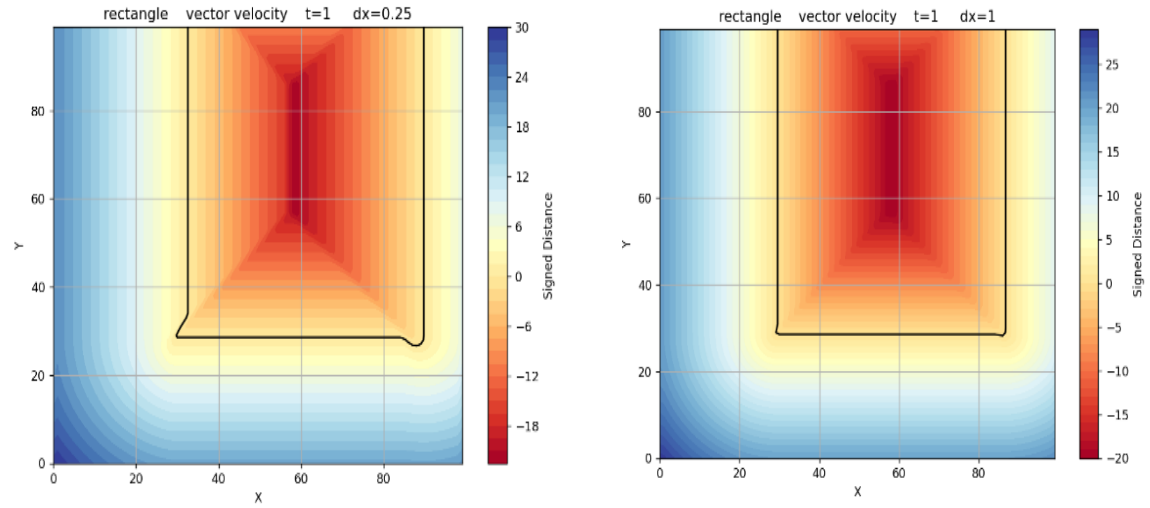





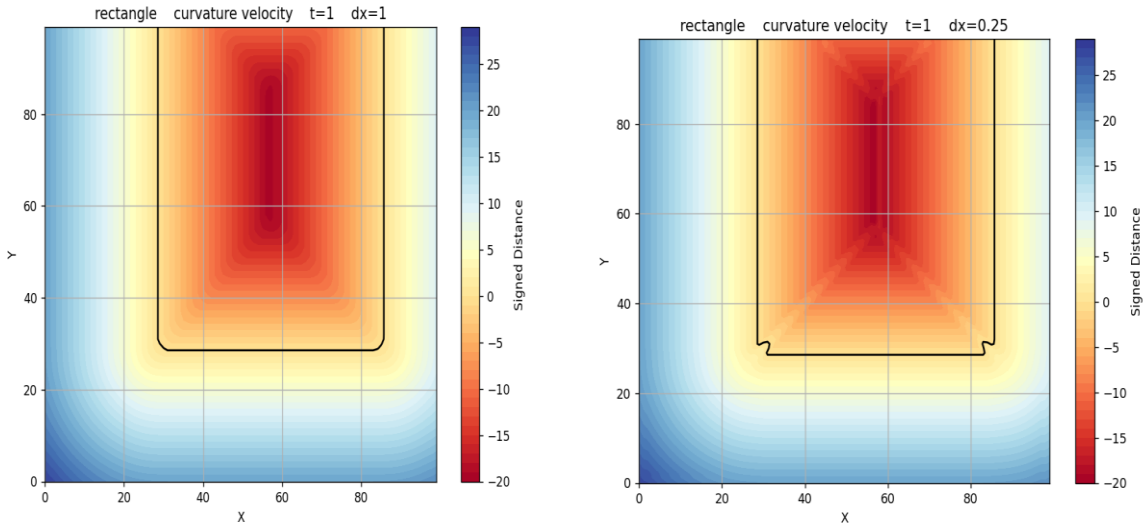

It is noticed from above that at higher times and finer resolution of 0.25, the effects of Engquist-Osher scheme are finer and more accurate and provides fine details of the surface movement and maintaining stability even for longer time steps. Also, rectangle surface shows more finer and accurate effects than the circle.

So, from the above it can be said that of the two methods, the Engquist-Osher scheme provides more accurate and enhanced details of the surface movement at higher resolutions and higher times.

Now, for the behaviour of the Rectangle with Vector Velocity Function and Curvature-Based Velocity, the following plots were obtained:




It can seen that for spacing = 1 there are visible effects as rectangle shifts a little in the corners in the x-direction and these effects are more precise at lower spacing and higher resolution = 0.25 which can be seen in the above left figure.




For curvature used as velocity, at lower resolution = 1, the corners of the rectangle curve and smooth out and at higher resolution = 0.25, the effects are more pronounced on the corners, and they are more distorted.

This curvature as velocity can be used for smoothing of surfaces, optimization of shapes for removing sharp edges and corners over large times from the surfaces.