Mateusz Wszeborowski 165562

Model of terrain using RBFs

Report

Problem

The main problem is when given a series of x,y,z points we need to interpolate the surface - find the value (z) of the points in between. This might be done using radial basis functions. Those types of functions only depend on the distance from the center. These means that they are symmetrical in a circle shape.

Used RBFs

During this project several RBFs were used. They are presented in a table below.

name	formula
Gaussian	$e^{-(\beta * r)^2}$
Multiquadric	$\sqrt{1 + (\beta r)^2}$
Inverse quadric	$\frac{1}{1 + (\beta r)^2}$
Polynomial	$r^2 + \beta^2$
This plate spline	$r^2 log(r)$

Moreover one local RBF was used.

$$r' = r / r_0$$
$$CP\ C^6 = 0,\ r' > 1$$
$$CP\ C^6 = (1 - r')^8\ (32(r')^3 + 25(r')^2 + 8r'),\ r' \leq 1$$

Overview

The goal of this project is to implement RBF interpolation. 2 separate datasets will be evaluated. First one will be a reconstruction of the lecture example and second one will introduce another function. Moreover standard RBFs will be compared to local RBFs.

Artificial data - lecture example

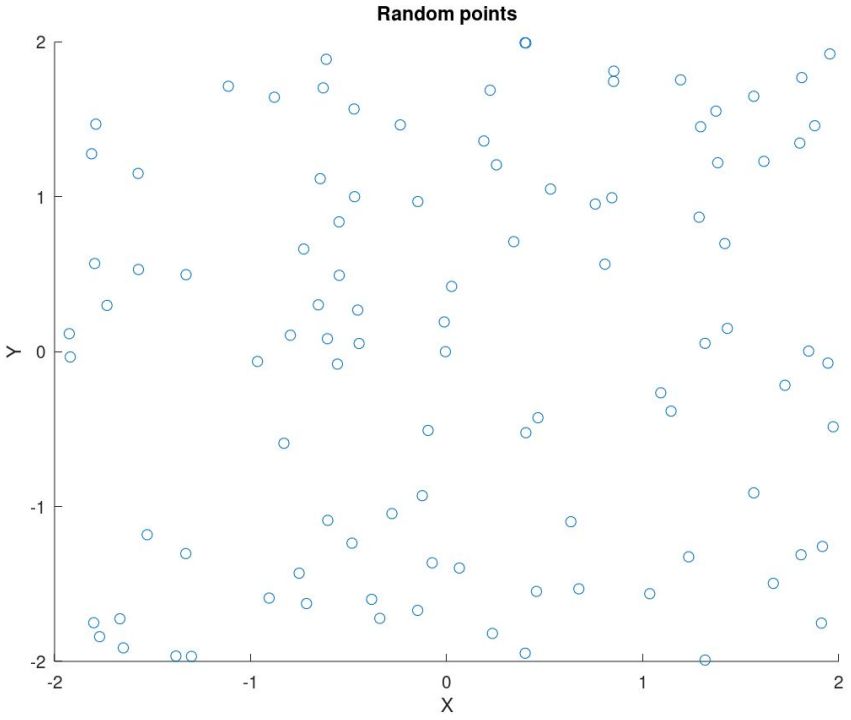
The first experiment was to perform interpolation on a known function.

$$f(x, y) = sin(\pi * x) * sin(\frac{\pi}{2} * y) * e^{-\frac{x^2 + y^2}{6}}$$

The domain was: $x \in <-2, 2>, y \in <-2, 2>$ each with step 0.2.

Steps

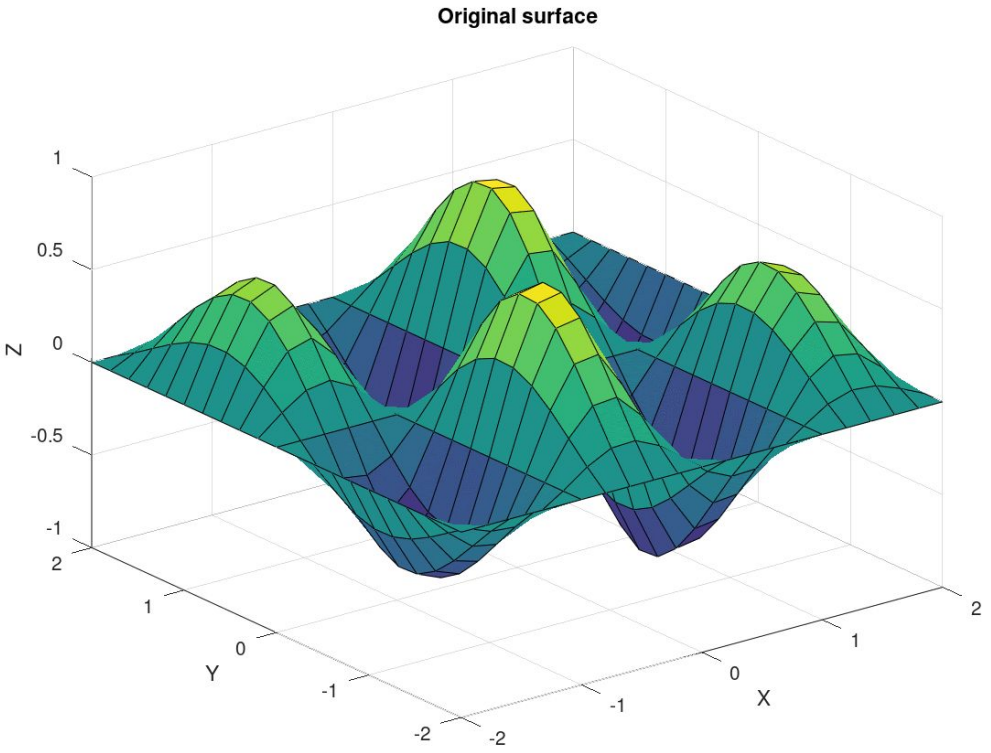
First step was to generate 100 random points from this domain.

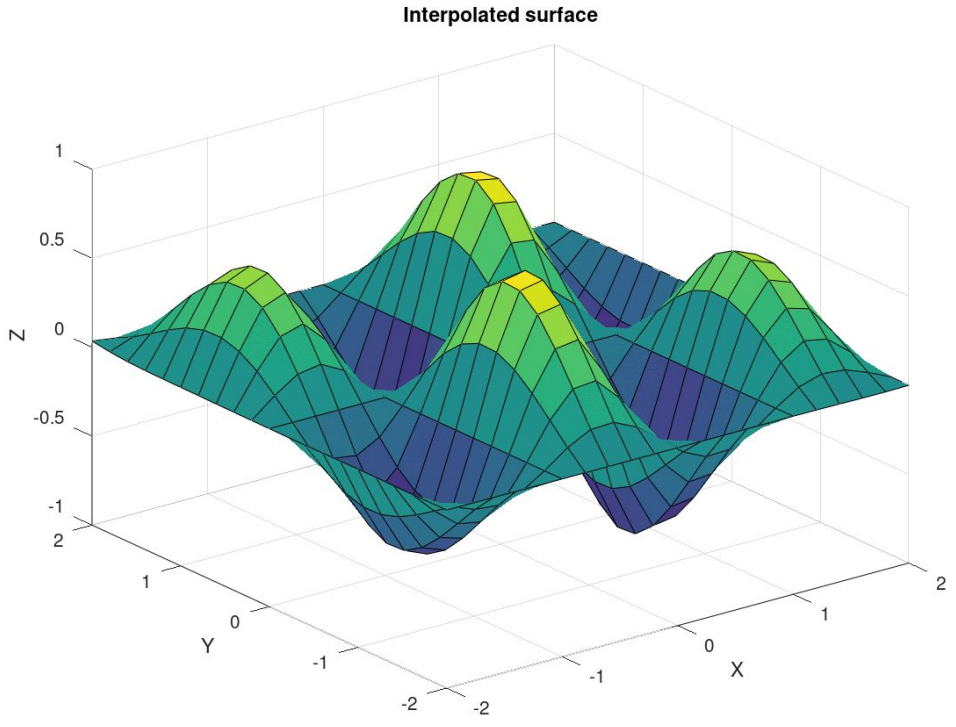


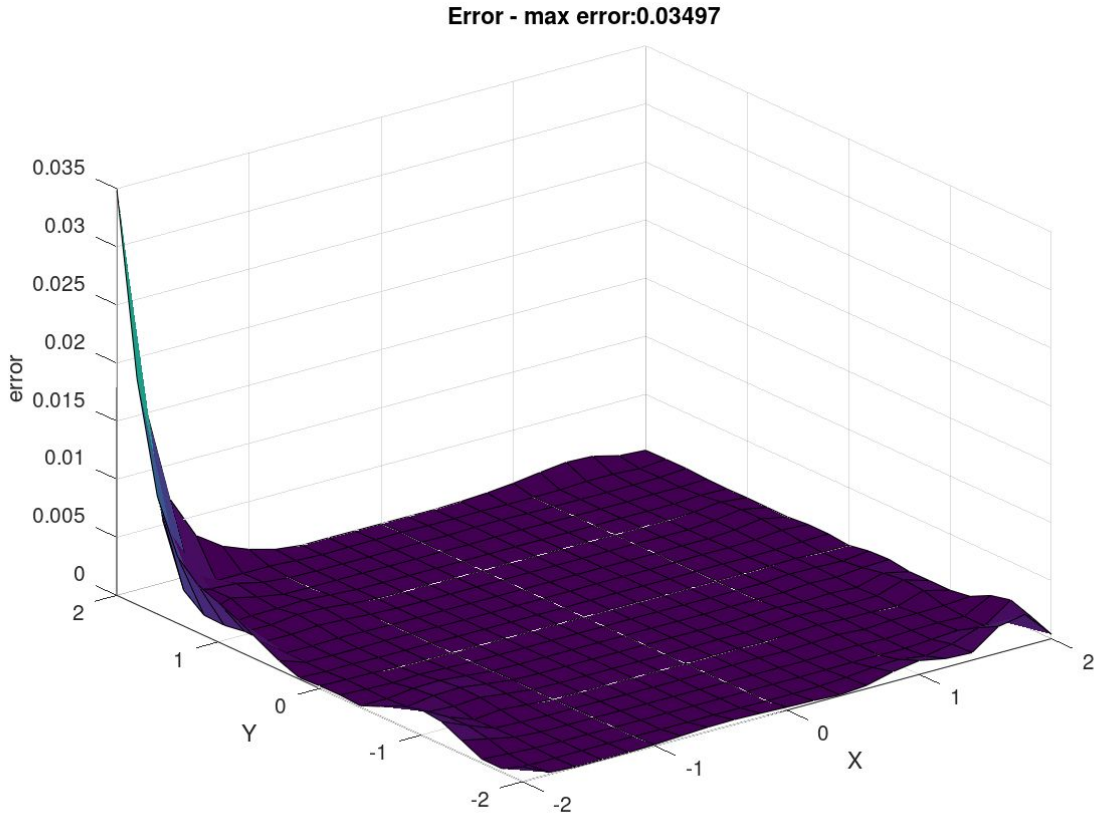
Next, those were used to solve interpolation linear system. Gaussian RBF was used with $\beta = 0.75$. Variant with additional polynomial component was used.

The last step was to interpolate the mesh for the rest of the points from the domain.

Results


Original surface
1
0.5
0
-0.5
-1
Z
2
1
0
-1
-2
Y
-2
-1
0
1
2
X


Interpolated surface
1
0.5
0
-0.5
-1
Z
2
1
0
-1
-2
Y
-2
-1
0
1
2
X



Results credibility

We can easily say how good this solution is since we know the original function from which the generated points had their value.

Times

Solving the equation took 0.9 seconds and evaluating the interpolation for the rest of the points took 2 seconds.

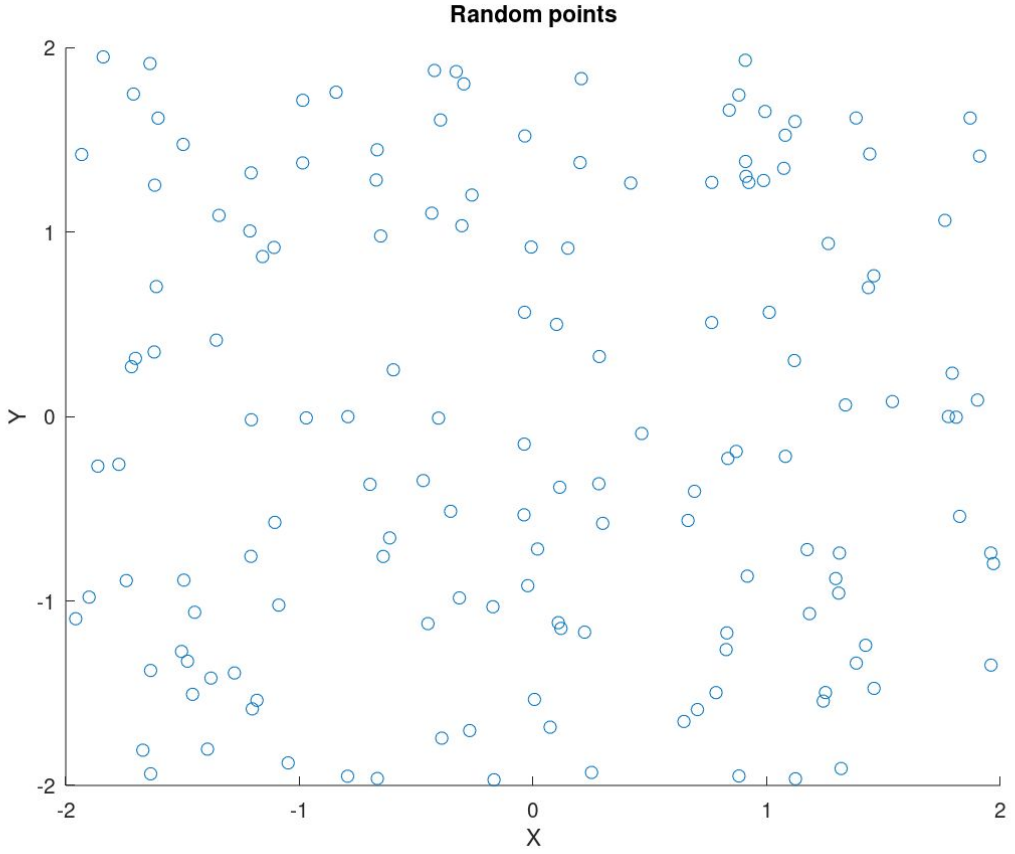
Artificial data

$$f(x,\ y)\ =\ cos(abs(x)\ +\ abs(y))\ *\ (abs(x)\ +\ abs(y))$$

The domain was: $x\ \varepsilon <-2,\ 2>,\ y\ \varepsilon <-2,\ 2>$ each with step 0.2.

Sample points

150 sample points were generated.



Grid search was introduced to find best parameters among:

- rbf functions: "gaussian", "multiquadric", "inverse-quadric", "polynomial", "thin-plate-spline"
- β : 0.5, 0.75, 1, 1.25, 1.5, 2

Rbf function	Beta	Max error	Mean error	Time for solving equation	Time for evaluating for the rest of the points
Gaussian	0.5	57.44	0.92145	0.52637	2.2014
Gaussian	0.75	112.75	1.4546	0.52743	2.1284
Gaussian	1	17.629	0.37744	0.50232	2.1274
Gaussian	1.25	1.756	0.10167	0.50318	2.1781
Gaussian	1.5	0.50363	0.050975	0.50198	2.1822
Gaussian	2	0.60557	0.065497	0.50683	2.2503
multiquadric	0.5	2.9233	0.094222	0.55656	2.4161
multiquadric	0.75	0.42717	0.041464	0.58552	2.1413
multiquadric	1	0.60514	0.037956	0.50009	2.1423

multiquadric	1.25	0.84564	0.038722	0.50062	2.114
multiquadric	1.5	0.9749	0.039854	0.50122	2.1165
multiquadric	2	1.0914	0.042422	0.50202	2.1495
inverse-quadric	0.5	1.1818	0.066901	0.52316	2.0976
inverse-quadric	0.75	0.50591	0.038444	0.50796	2.1036
inverse-quadric	1	0.62583	0.035984	0.5022	2.1289
inverse-quadric	1.25	0.4943	0.03633	0.50343	2.0975
inverse-quadric	1.5	0.36279	0.040811	0.50057	2.1091
inverse-quadric	2	0.53676	0.063044	0.50037	2.1383
polynomial	0.5	2.1563	0.42675	0.54988	2.1253
polynomial	0.75	2.6912	0.37719	0.51953	2.1887
polynomial	1	2.941	0.39769	0.52282	2.1191
polynomial	1.25	2.7228	0.3777	0.54279	2.3894
polynomial	1.5	2.6101	0.36745	0.562	2.1864
polynomial	2	3.0526	0.42393	0.52293	2.1243
thin-plate-spline	0.5	1.4561	0.057303	0.52048	2.1069
thin-plate-spline	0.75	1.4561	0.057303	0.50138	2.1309
thin-plate-spline	1	1.4561	0.057303	0.50289	2.2331
thin-plate-spline	1.25	1.4561	0.057303	0.5017	2.1022
thin-plate-spline	1.5	1.4561	0.057303	0.50301	2.1492
thin-plate-spline	2	1.4561	0.057303	0.50164	2.2203

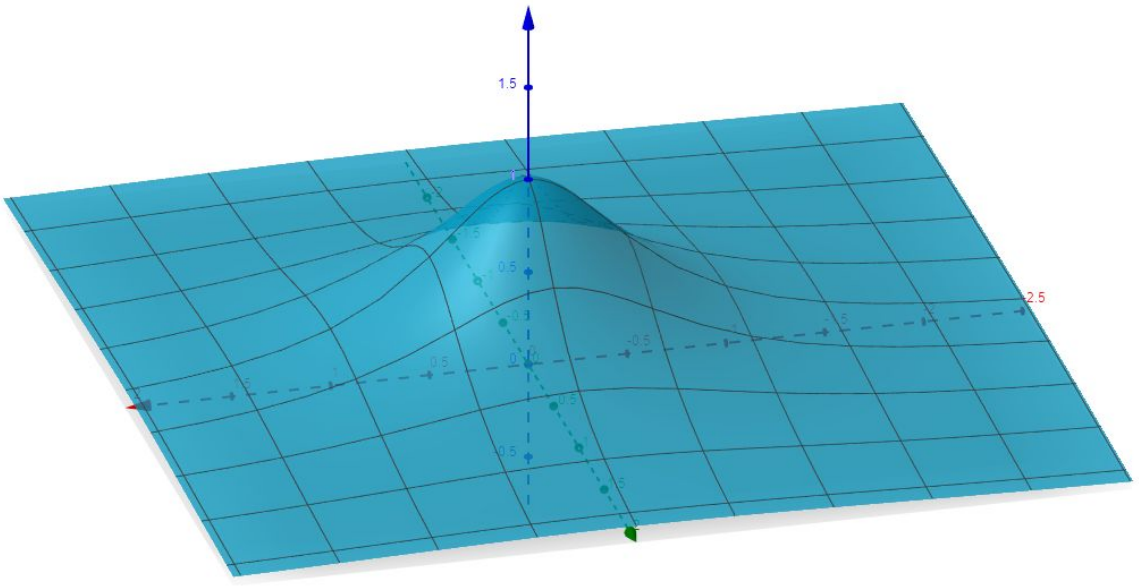
Best parameters (according to the sum of max and mean error):

- rbf function: inverse-quadric
- beta: 1.5

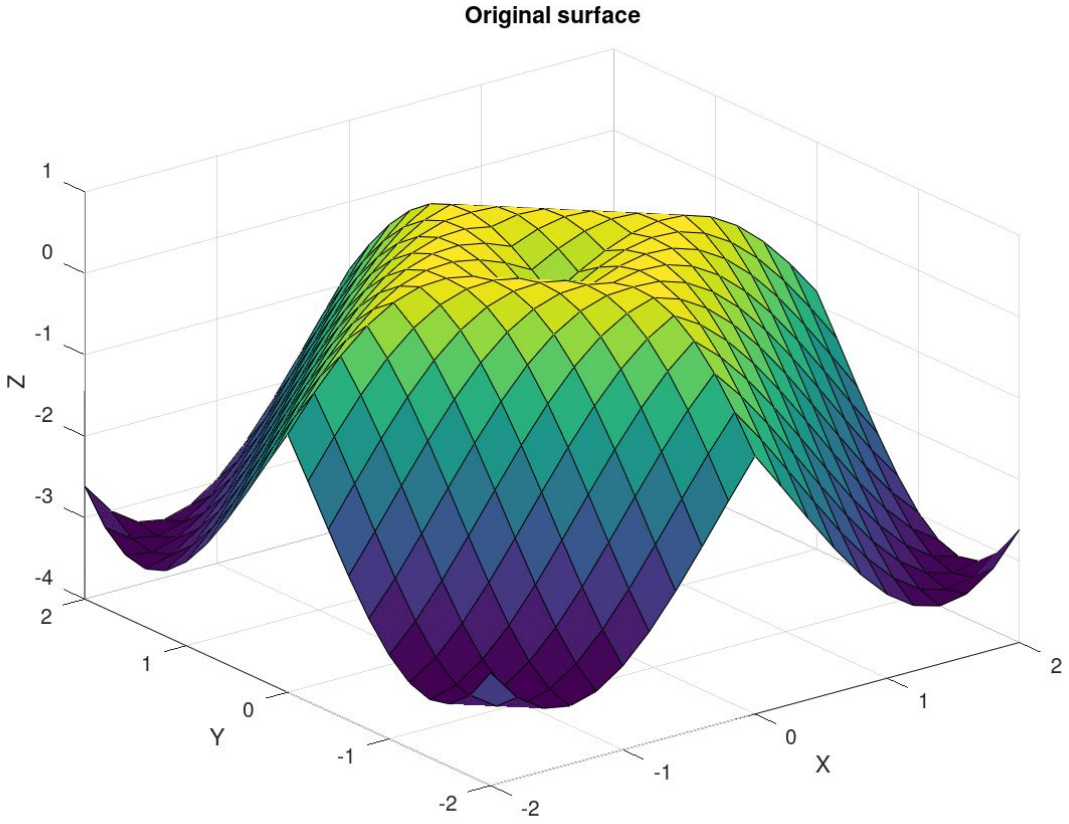
This function is plotted below.

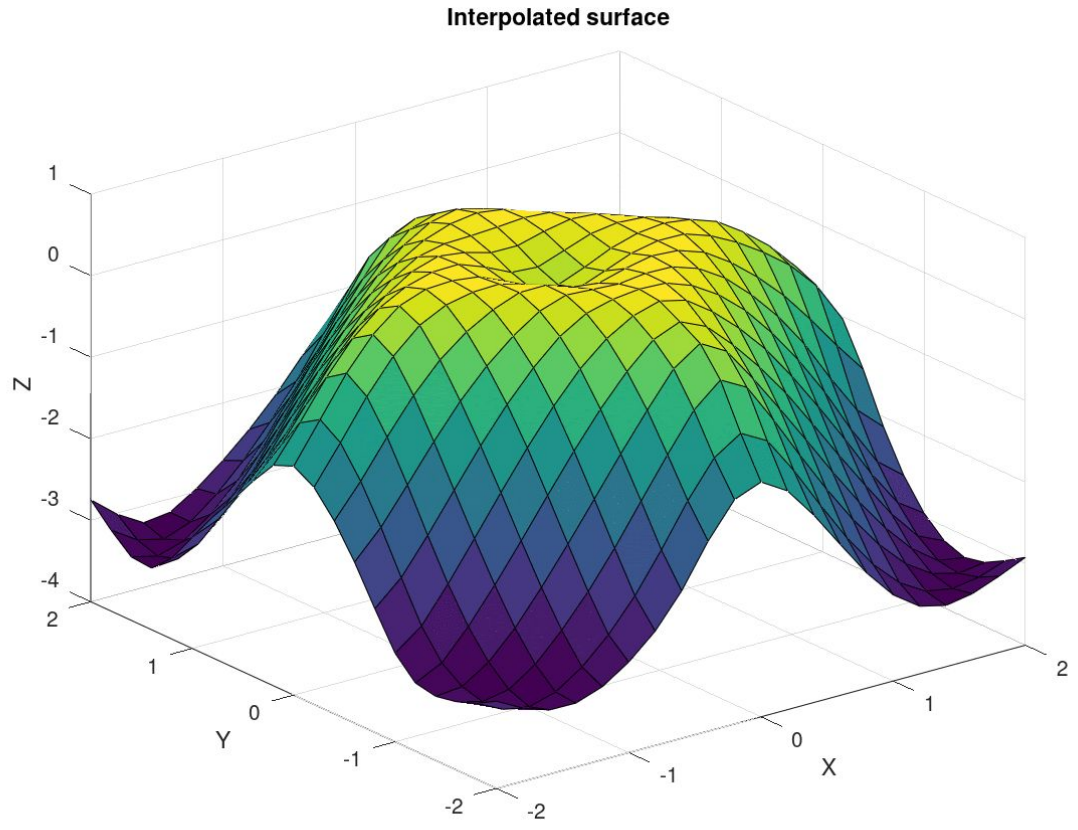
Observing the results we can see that some functions gave very high max errors - biggest errors come from a gaussian function.

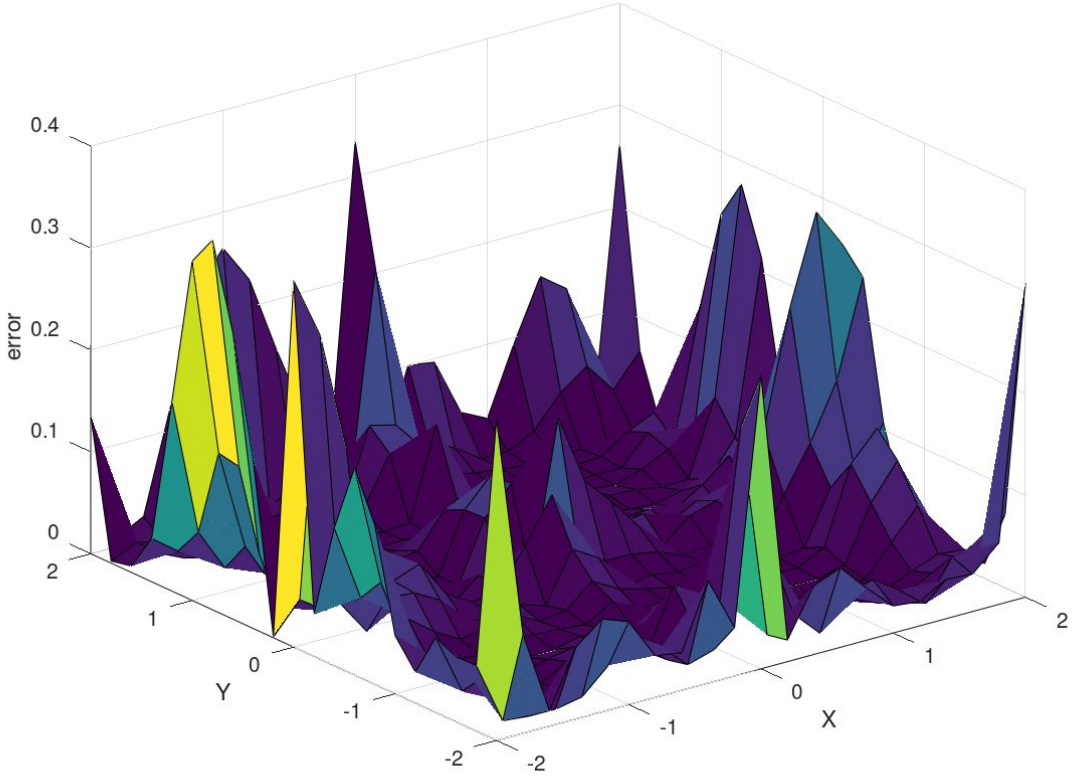
We can also see that picked functions do not influence the times. This is an expected result.



Inverse quadric with $\beta = 1.25$




Interpolated surface
Z
1
0
-1
-2
-3
-4
Y
2
1
0
-1
-2
X
-2
-1
0
1
2


Error -max error:0.36279 -mean error:0.040811
error
0.4
0.3
0.2
0.1
0
Y
2
1
0
-1
-2
X
-2
-1
0
1
2

Analyzing the graphs we can see that although the mean error is pretty low the max error can get pretty heigh. This might be due to irregularity of the function. We can see that the original function has some sharp edges that are flatten during interpolation. The borders can also get significant errors. This might be because of lack of sample points.

Local Basis Functions

Last experiment was about trying to repeat the second experiment but with local basis functions - that equal 0 if $r > r_0$.

During experiments r_0 was set to 1.5 and RBF was set to $CP\ C^6$.

The main goal of this part is to compare standard RBFs and local RBFs. The metrics will include:

- time needed to solve the equation
- time needed to evaluate the function for the rest of the points
- mean error
- max error

The artificial function that was used to create sample points is the same that was used during the first experiment:

$$f(x, y) = sin(\pi * x) * sin(\frac{\pi}{2} * y) * e^{-\frac{x^2 + y^2}{6}}$$

The solution was tested on K = 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000 sample points.

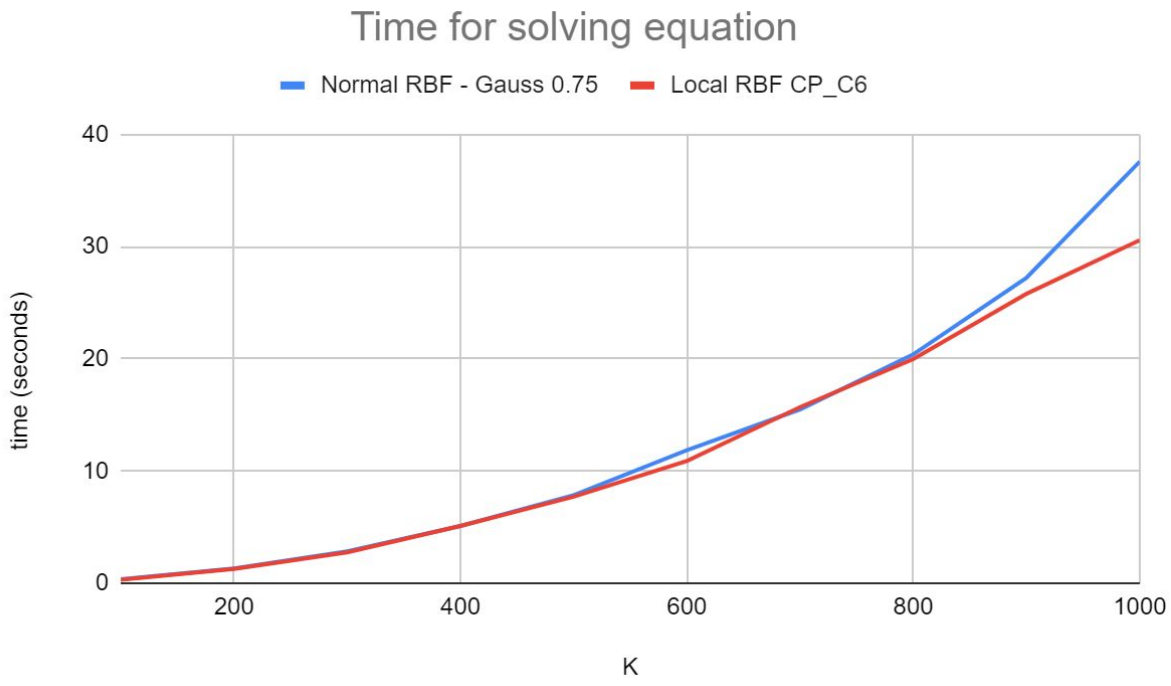
The results are shown in a table below.

	Normal RBF - Gauss 0.75				Local RBF CP_C6			
K	time solve	time eval	error mean	error max	time solve	time eval	error mean	error max
100	0,34282	1,8973	0,00029845	0,02163	0,30519	1,9098	0,04082	0,28538
200	1,304	4,0681	9,36E-06	0,0011897	1,2651	3,8557	0,0071144	0,071398
300	2,8303	5,7182	6,14E-07	6,28E-05	2,75	6,1541	0,0039205	0,073244
400	5,0952	7,7355	3,40E-07	6,78E-05	5,1	7,7031	0,0023683	0,089564
500	7,8461	9,5952	6,21E-08	3,04E-06	7,7134	9,6617	0,001323	0,040013

							2	
600	11,846	11,824	4,16E-08	2,38E-06	10,899	11,805	0,000777 68	0,027665
700	15,485	13,35	1,30E-08	1,07E-06	15,701	13,791	0,000607 77	0,01796
800	20,396	16,852	3,27E-08	1,88E-06	19,975	15,587	0,000433 73	0,013375
900	27,212	17,522	1,42E-08	1,40E-06	25,79	17,265	0,000470 04	0,029379
1000	37,58	21,355	1,87E-08	1,43E-06	30,57	20,334	0,000303 96	0,011618

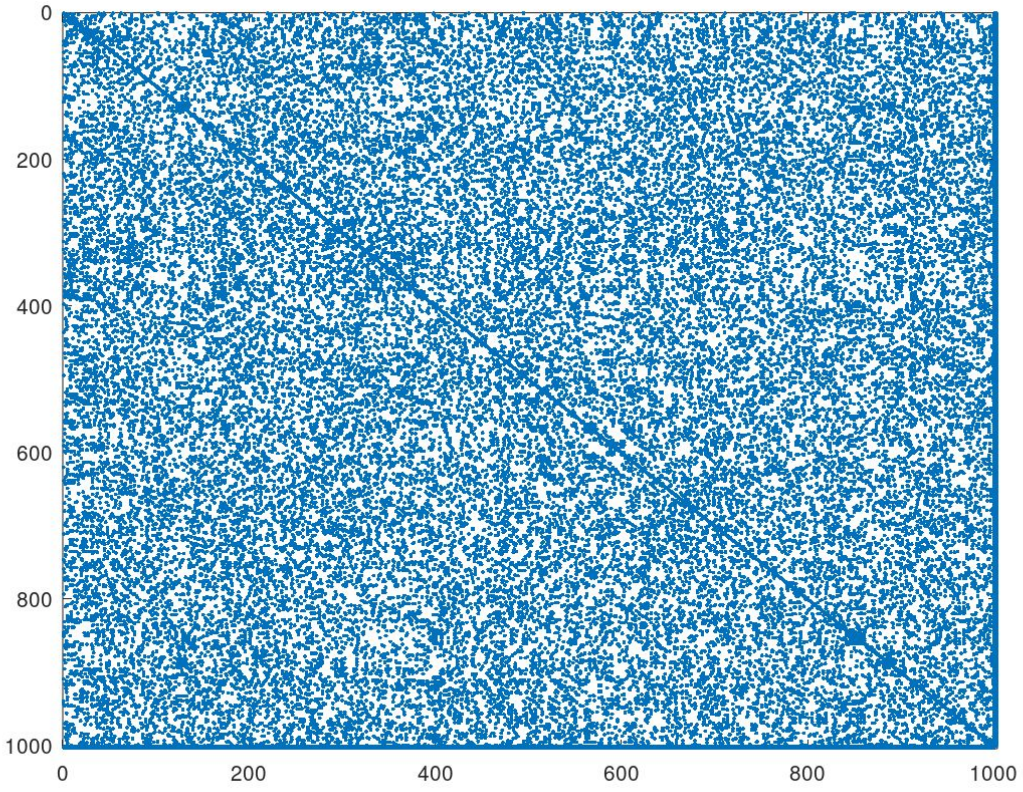
We can also see the results on the charts below.

Time for solving the equation



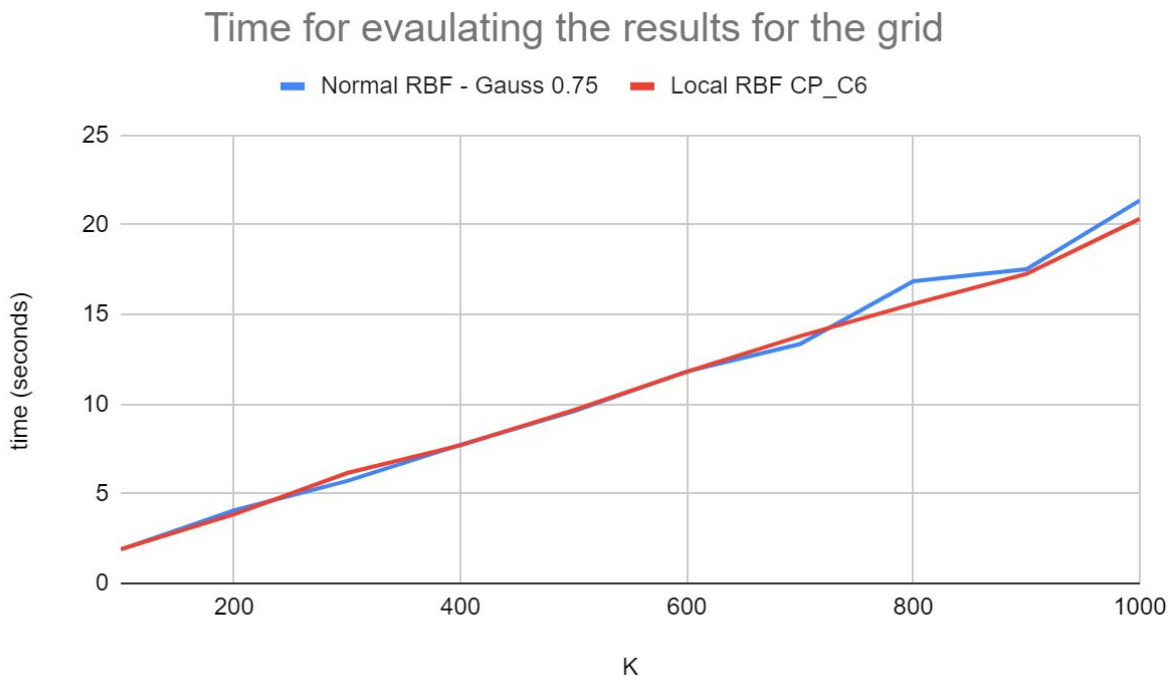
We can see that for a small number of points the results are almost identical, but as the number of points start to rise the local rbfs sparse matrices tend to solve the equation faster.

In the picture below we can see a spy of the matrix used to solve the equation.



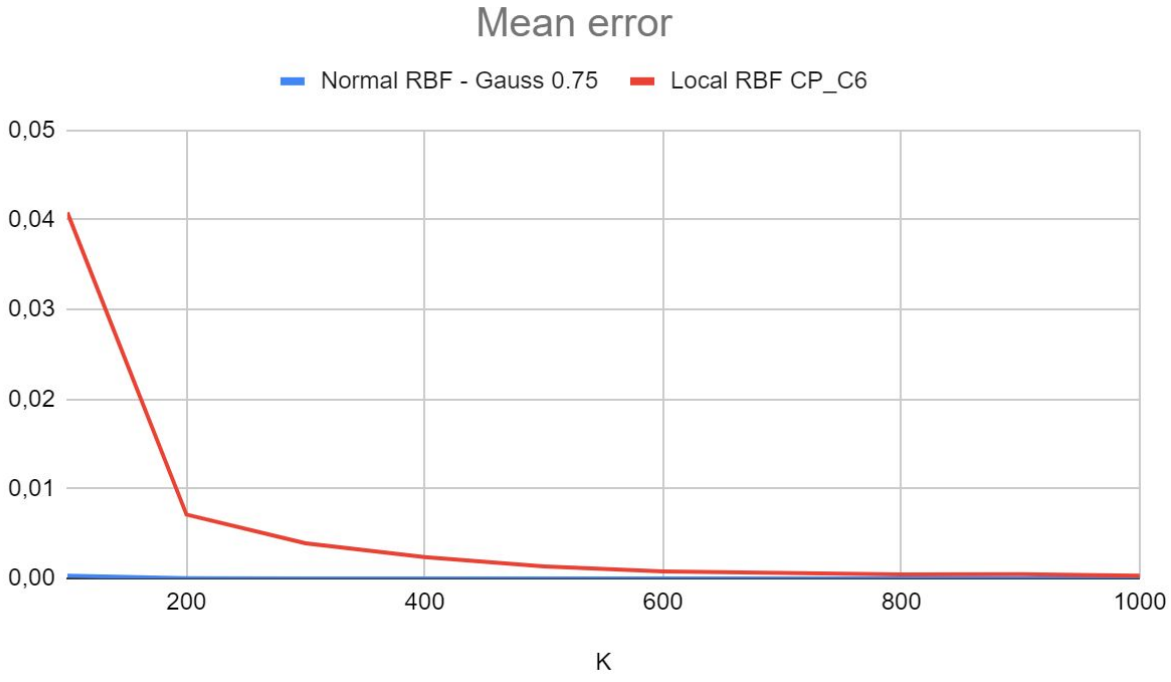
Spy of a matrix

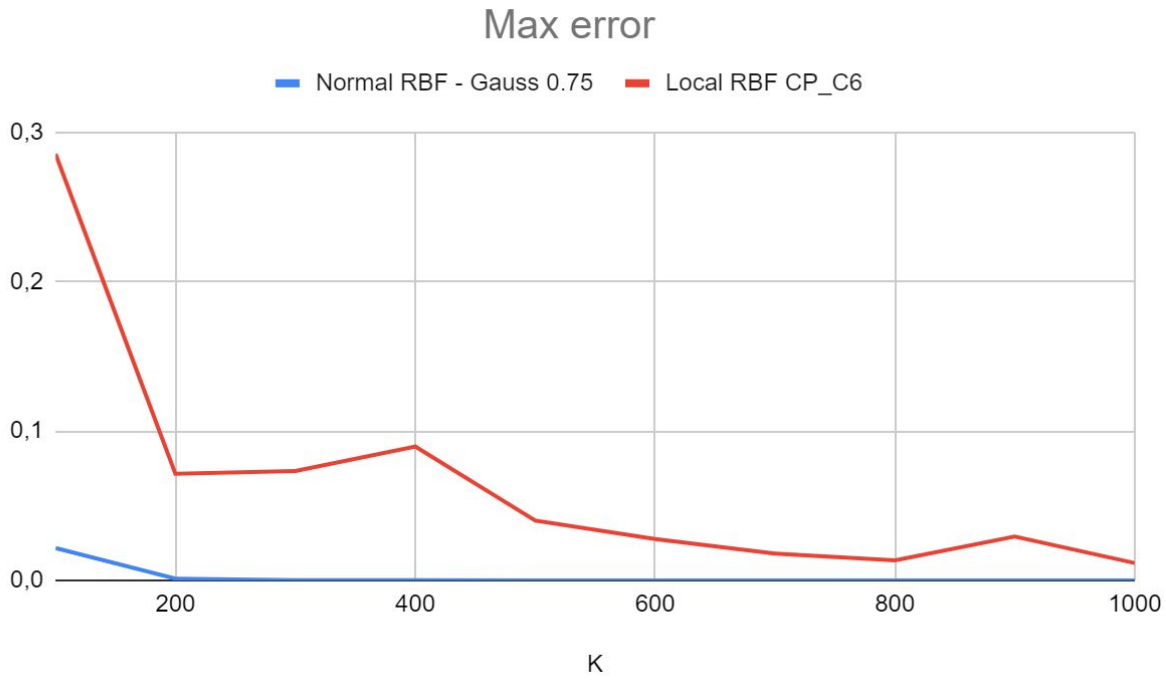
Time for evaluating the results for the rest of the points



As we can see these times are statistically identical and this is an expected result.

Errors





We can see on these charts that local RBFs is sacrificing accuracy for better times. It is worth noting that for a small number of points the error is prette high for local RBFs but is almost deniable in standard RBFs.

Performance of the code

Almost all the operations used in the code are performed using matrices. This should be significantly faster than using loops.

Summary

To sum up, we can see that RBFs can be used for interpolation. The might give pretty good results quite fast and with lack of complexity.