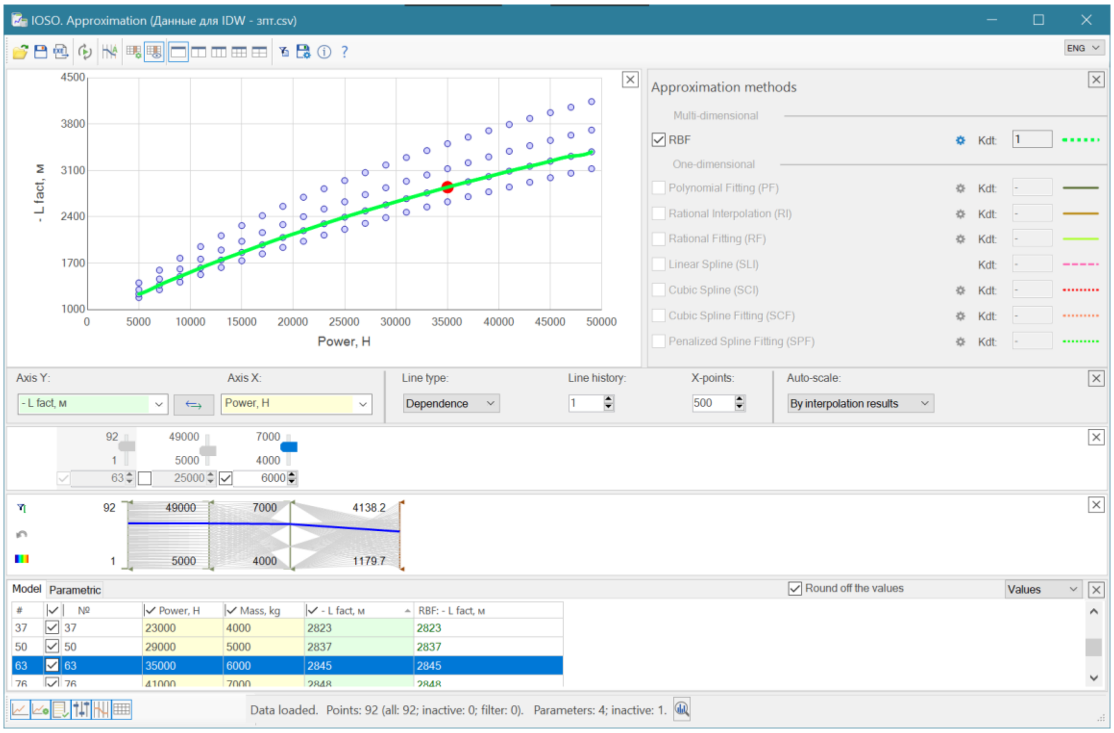
IOSO Approximation software v.1.0

User Guide

Contents

IOSO Approximation 1.0 Software

The «IOSO Approximation» software allows you to interpolate data and create an approximation model as an executable file (* .exe). Interpolation is a method of constructing new data points within the range of a discrete set of known data points in the mathematical field of numerical analysis.



Approximation methods

Multi-dimensional

	Method	Kdt
✓	RBF	1

One-dimensional

	Method	Kdt
	Polynomial Fitting (PF)	-
	Rational Interpolation (RI)	-
	Rational Fitting (RF)	-
	Linear Spline (SLI)	-
	Cubic Spline (SCI)	-
	Cubic Spline Fitting (SCF)	-
	Penalized Spline Fitting (SPF)	-

Axis Y: - L fact, м
Axis X: Power, H
Line type: Dependence
Line history: 1
X-points: 500
Auto-scale: By interpolation results

92 / 1: 63
49000 / 5000: ☐ 25000
7000 / 4000: ✓ 6000

Model | Parametric

✓ Round off the values
Values

#	✓	№	✓ Power, H	✓ Mass, kg	✓ - L fact, м	RBF: - L fact, м
37	✓	37	23000	4000	2823	2823
50	✓	50	29000	5000	2837	2837
63	✓	63	35000	6000	2845	2845
76	✓	76	41000	7000	2848	2848

Data loaded. Points: 92 (all: 92; inactive: 0; filter: 0). Parameters: 4; inactive: 1.

The “IOSO Approximation” software allows the user to create fit functions using powerful univariate and multivariate fitting methods

One-dimensional approximation methods:

- Polynomial Interpolation (PI)
- Polynomial Fitting (PF)
- Rational Interpolation (RI)
- Rational Fitting (RF)
- Linear Spline (SLI)
- Cubic Spline (SCI)
- Catmull-Rom Spline (SCRI)
- Akima Spline (SAI)
- Cubic Spline Fitting (SCF)
- Hermite Spline Fitting (SHF)
- Penalized Spline Fitting (SPF)

Multi-dimensional approximation methods:

- RBF - Radial basis function interpolation;
- IDW Inverse distance weighting interpolation.

.

The initial Data for the construction of the approximation function can be:

- protocols of projects of the IOSO optimization software;
- arrays of data in text format files, including CSV files;
- data entered into the IOSO Approximation table**.**

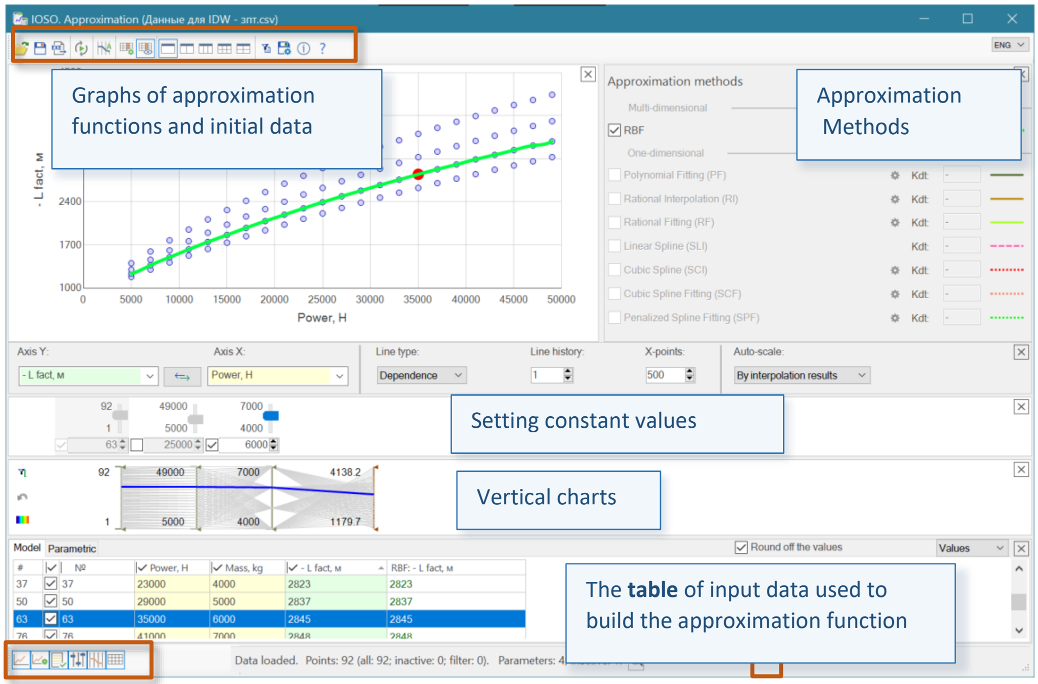
The constructed approximation function can be saved as an executable model (exe file) and used later as surrogate models in problems of modeling systems or objects. These models can be used in the IOSO optimization software to replace complex resource-intensive computational models when solving optimization problems with multidisciplinary projects and the possibility of automated creation, connection and training of approximation models on new data is implemented.

IOSO Approximation software developed by SIGMA Technology (https://www.iosotech.com).

Please send suggestions and comments on the software application to company@iosotech.com with a note in the subject line of IOSO.

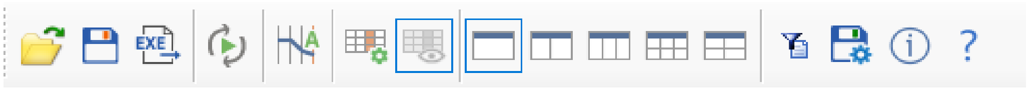
IOSO Approximation Software Interfaces

GUI of IOSO Approximation software. Main interface.



Use this interface to create and present an approximate model using different tabular graphics.

The top toolbar contains commands:



- open the file. Different types of files are available to open:

- **file with data** in .csv or text format. For the text format, values must be written in split columns: a comma dot or tabulation;

- previously saved **approximation project file** (*.apr);

- **IOSO Optimization project file** (* .opm, * .popm). Data from the protocols of parametric studies, from the protocol of the current optimization problem, or from the general protocol can be taken from this file. From this data, data can be selected for the selected models (models).

- save the approximation project (*.apr).

– save the generated approximation function as an executable file (* exe). This executable file can be used as an independent computational model in the IOSO optimization software or for other purposes.

– create an approximation function. Not active when the "Rebuild automatically" mode is enabled in the section of approximation methods;

- display the values of the approximation function in the vertical graph window;

- setting the used parameters (columns) in the table to build the approximation function;

- show columns with unused parameters in the table or not;

- selecting the number of graphs displayed in the Graphics window

-- customizing the User Interface and used approximation methods;

- - filter for the data used to build the fit function. The settings of this filter are synchronized with the filter on vertical charts;

- IOSO Approximation software information;

- software help.

Commands for displaying software interfaces located on the bottom toolbar:

and .

These commands are used for visualization and work with the software with low-resolution displays.

- enable / disable the "Charts" window;

- enable / disable the "Chart settings" panel;

- enable / disable the window " Approximation Methods";

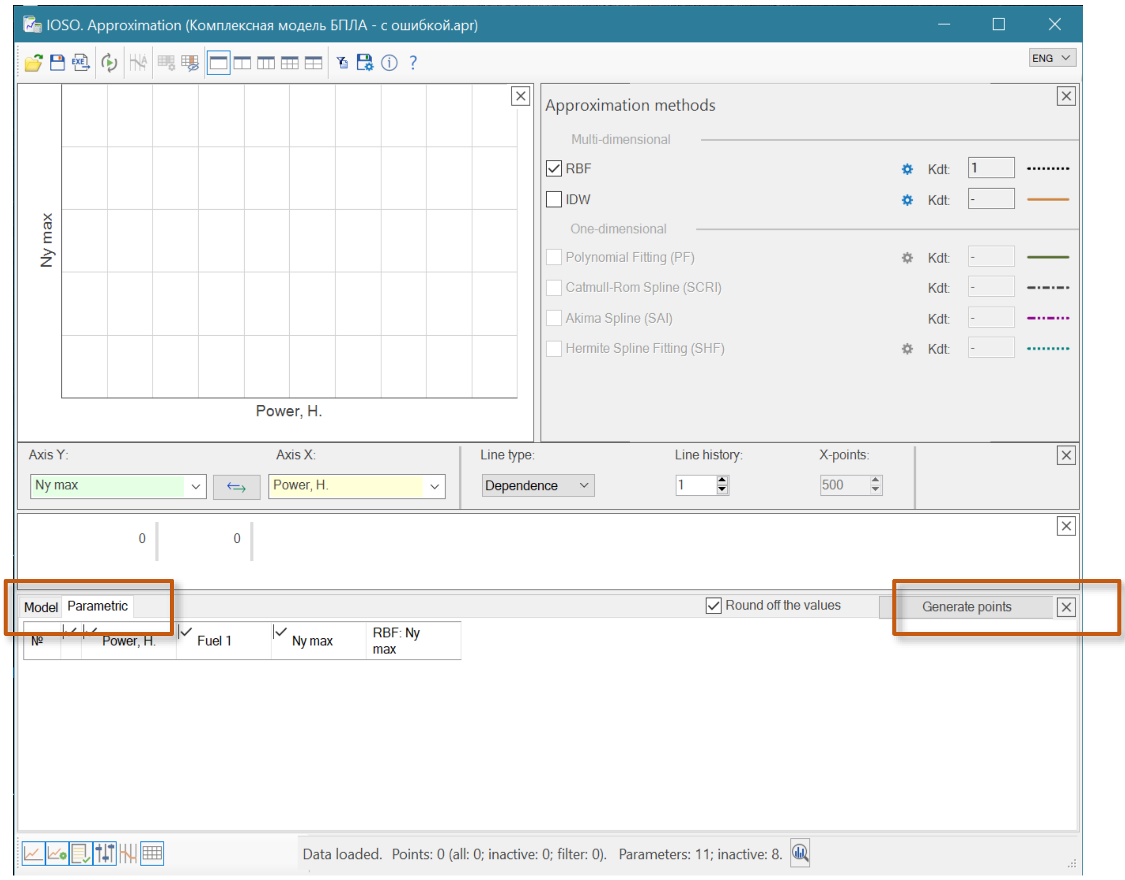
enable / disable the "Setting constant values" window. Setting constant values is used to evaluate the behavior of the multivariate approximation function when changing the selected parameters (input values);

enable / disable the "Vertical charts" window;

enable / disable the "Table" window;

- Statistical information on the array of input data.

You can perform parametric studies of the created approximation model using the Parameters interface. Switching to this mode is performed on the tab (see Fig. Below)



IOSO. Approximation (Комплексная модель БПЛА - с ошибкой.apr)

Approximation methods

Multi-dimensional

✓ RBF	Kdt:	1
IDW	Kdt:	-

One-dimensional

Polynomial Fitting (PF)	Kdt:	-
Catmull-Rom Spline (SCRI)	Kdt:	-
Akima Spline (SAI)	Kdt:	-
Hermite Spline Fitting (SHF)	Kdt:	-

Axis Y: Ny max
Axis X: Power, H.
Line type: Dependence
Line history: 1
X-points: 500

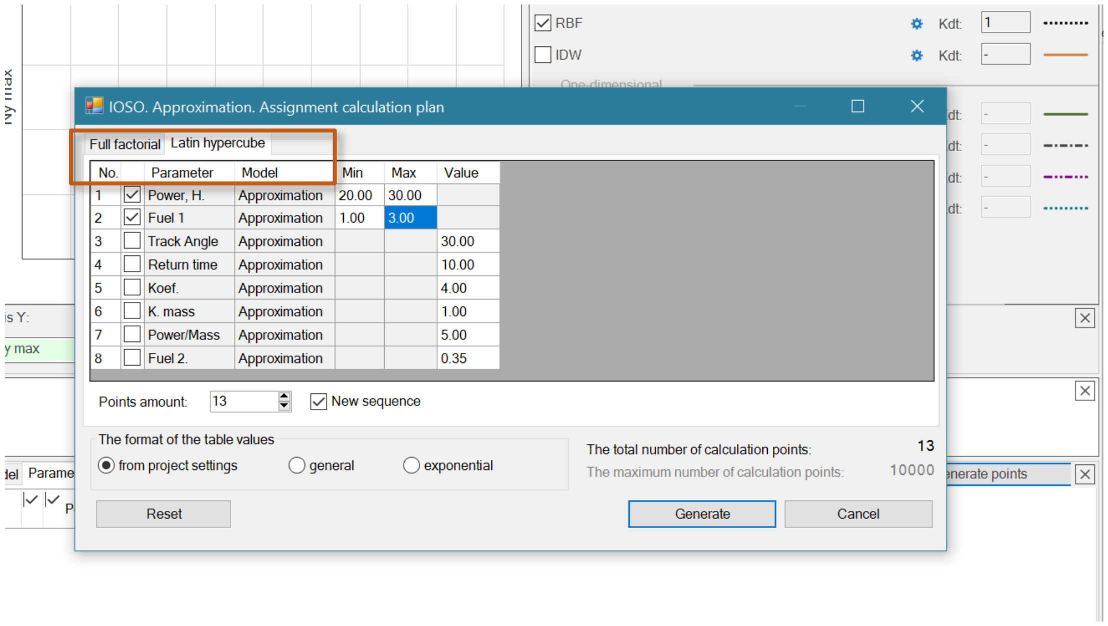
Round off the values ✓

№	Power, H.	Fuel 1	Ny max	RBF: Ny max

Data loaded. Points: 0 (all: 0; inactive: 0; filter: 0). Parameters: 11; inactive: 8.

You create a calculation point pattern using the calculation plan creation form. The following data array modes are available:

- full-factor experiment;
- Latin hypercube.



No.		Parameter	Model	Min	Max	Value
1	✓	Power, H.	Approximation	20.00	30.00	
2	✓	Fuel 1	Approximation	1.00	3.00	
3		Track Angle	Approximation			30.00
4		Return time	Approximation			10.00
5		Koef.	Approximation			4.00
6		K. mass	Approximation			1.00
7		Power/Mass	Approximation			5.00
8		Fuel 2.	Approximation			0.35

Points amount: 13 ✓ New sequence

The format of the table values
[✓ from project settings] general exponential

The total number of calculation points: 13
The maximum number of calculation points: 10000

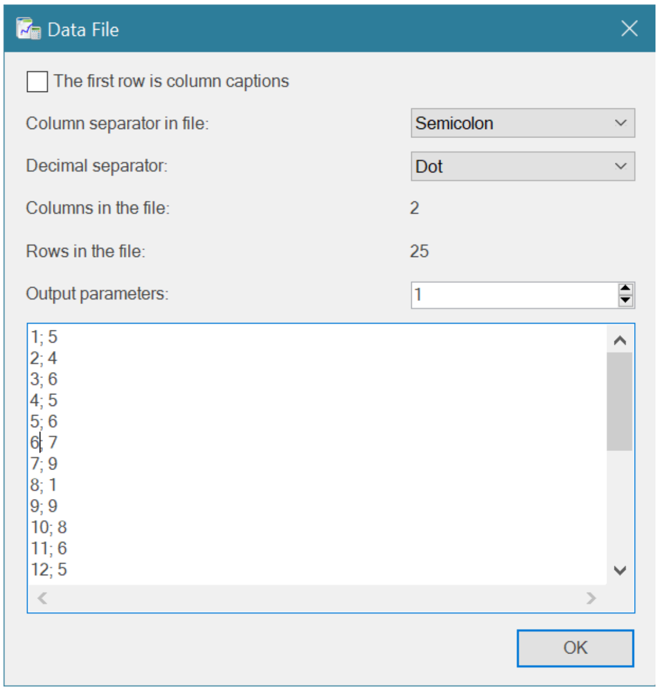
Creating an approximation model step by step

Step 1. Loading initial data

To load an initial data, use the "Open file" command of the top toolbar .

The initial Data for the construction of the approximation function can be:
- Protocols of projects of the IOSO optimization software;
- arrays of data in text format files, including CSV files;
- data entered into the IOSO Approximation table.

When loading a data array in the "Output parameters" position, you must specify the number of columns with function values. It is assumed that the first columns contain the function arguments, the last columns contain the function values. The number of function values can be 1 or more.


Data File

☐ The first row is column captions

Column separator in file: Semicolon

Decimal separator: Dot

Columns in the file: 2

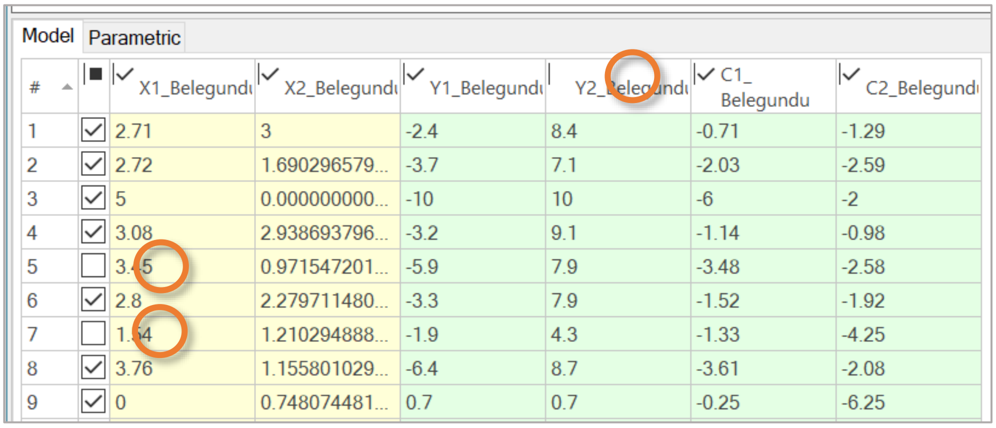
Rows in the file: 25

Output parameters: 1

```
1; 5
2; 4
3; 6
4; 5
5; 6
6; 7
7; 9
8; 1
9; 9
10; 8
11; 6
12; 5
```

The data is loaded into a table and displayed on graphs. The function arguments are displayed in the table on a yellow background - (in this figure there are 2 of them), the function values - on a green background - (in the figure there are 4 of them).

It is possible to use the Check box to select data (function and function arguments) for which it is necessary to construct approximation functions. The figure shows that for the "Y2_Belegundu" function, the approximation function is not built and the values in lines 5,7 will not be used to build the approximation functions.



#	■	✓ X1_Belegund	✓ X2_Belegund	✓ Y1_Belegund	Y2_Belegund	✓ C1_ Belegundu	✓ C2_Belegund
1	✓	2.71	3	-2.4	8.4	-0.71	-1.29
2	✓	2.72	1.690296579...	-3.7	7.1	-2.03	-2.59
3	✓	5	0.000000000...	-10	10	-6	-2
4	✓	3.08	2.938693796...	-3.2	9.1	-1.14	-0.98
5	☐	3.45	0.971547201...	-5.9	7.9	-3.48	-2.58
6	✓	2.8	2.279711480...	-3.3	7.9	-1.52	-1.92
7	☐	1.54	1.210294888...	-1.9	4.3	-1.33	-4.25
8	✓	3.76	1.155801029...	-6.4	8.7	-3.61	-2.08
9	✓	0	0.748074481...	0.7	0.7	-0.25	-6.25

The button "Statistics about the array of initial data" allows you to show the following information about the array of initial data:

- Minimum and maximum parameter value;

- mid-arithmetic step value between the parameters ("Avg. step");

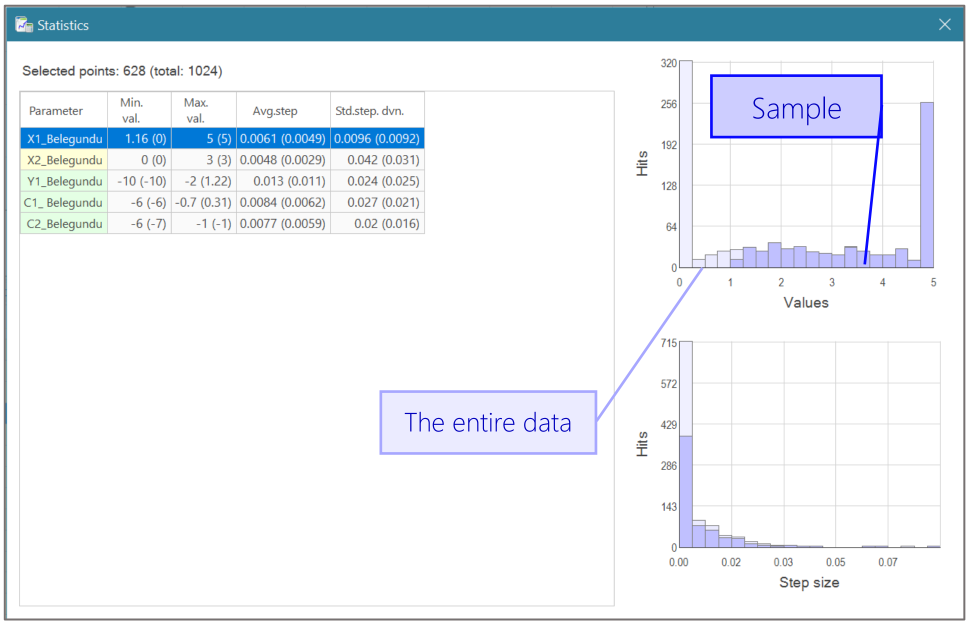
- root-mean-square deviation for parameter intervals ("Std. step dvn"). To create an approximation function, it is recommended to use data arrays with a uniform distribution of parameters in the studied range;

- graphical distribution - a histogram of the number of values in 20 equal segments of the studied range;

- histogram of the number of parameter step values in 20 equal segments of the available parameter step range.

The table contains values for the current and the entire selection. The values of the entire array are indicated in parentheses.

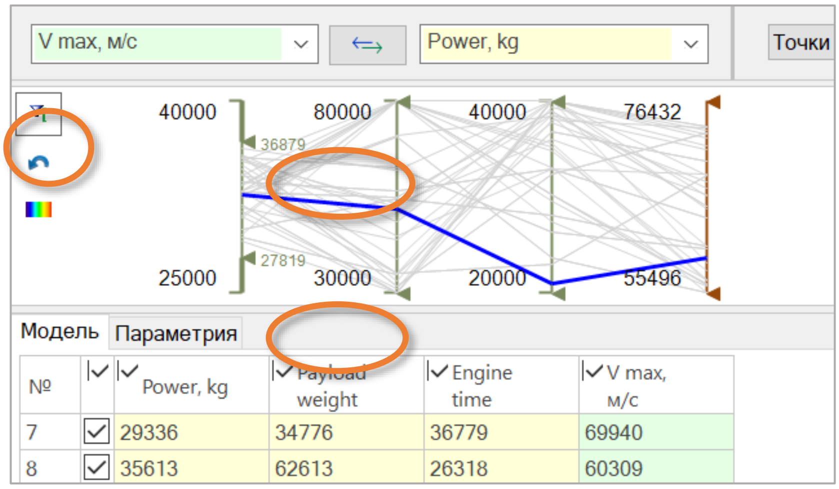
The values of the entire data set are shown in light color on the histogram, the values of the current sample are shown in dark color on the histogram.



Selected points: 628 (total: 1024)

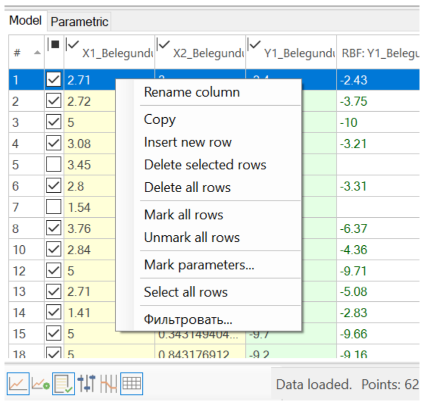
Parameter	Min. val.	Max. val.	Avg.step	Std.step. dvn.
X1_Belegundu	1.16 (0)	5 (5)	0.0061 (0.0049)	0.0096 (0.0092)
X2_Belegundu	0 (0)	3 (3)	0.0048 (0.0029)	0.042 (0.031)
Y1_Belegundu	-10 (-10)	-2 (1.22)	0.013 (0.011)	0.024 (0.025)
C1_ Belegundu	-6 (-6)	-0.7 (0.31)	0.0084 (0.0062)	0.027 (0.021)
C2_Belegundu	-6 (-7)	-1 (-1)	0.0077 (0.0059)	0.02 (0.016)

The filter in the "vertical charts" window is also used to filter data. The filter is opened by pressing the icon . the range of parameters in the selection changes when moving the indices on the vertical scale of the parameter.



№	✓	✓ Power, kg	✓ Payload weight	✓ Engine time	✓ V max, м/с
7	✓	29336	34776	36779	69940
8	✓	35613	62613	26318	60309

it is possible to "manually" generate an array of data in the table. The table editing menu is invoked by the right mouse button.



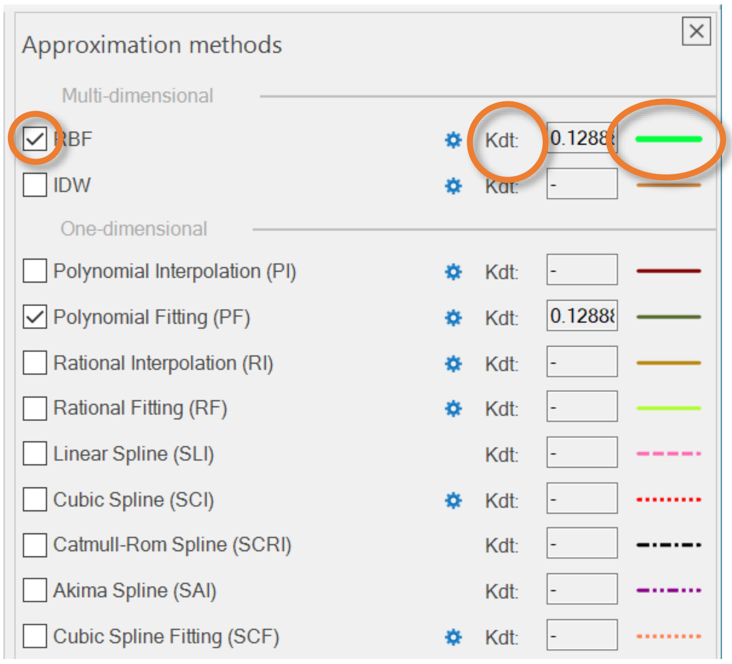
Model | Parametric

#	■	X1_Belegundı	X2_Belegundı	Y1_Belegundı	RBF: Y1_Belegu
1	✓	2.71	2		-2.43
2	✓	2.72			-3.75
3	✓	5			-10
4	✓	3.08			-3.21
5		3.45			
6	✓	2.8			-3.31
7		1.54			
8	✓	3.76			-6.37
10	✓	2.84			-4.36
12	✓	5			-9.71
13	✓	2.71			-5.08
14	✓	1.41			-2.83
15	✓	5	0.343149404...	-9.7	-9.66
18	✓	5	0.843176912	-9.2	-9.16

Data loaded. Points: 62

Step 2. Building an approximation function

To build an approximation function, you must select the method / methods of approximation and the function will be built automatically.



Approximation methods

Multi-dimensional		
✓	RBF	Kdt: 0.1288
☐	IDW	Kdt: -
One-dimensional		
☐	Polynomial Interpolation (PI)	Kdt: -
✓	Polynomial Fitting (PF)	Kdt: 0.1288
☐	Rational Interpolation (RI)	Kdt: -
☐	Rational Fitting (RF)	Kdt: -
☐	Linear Spline (SLI)	Kdt: -
☐	Cubic Spline (SCI)	Kdt: -
☐	Catmull-Rom Spline (SCRI)	Kdt: -
☐	Akima Spline (SAI)	Kdt: -
☐	Cubic Spline Fitting (SCF)	Kdt: -

The color of the fit function can be changed by double-clicking the pattern of the fit function line.

⚙ - approximation method settings.

Multidimensional approximation methods

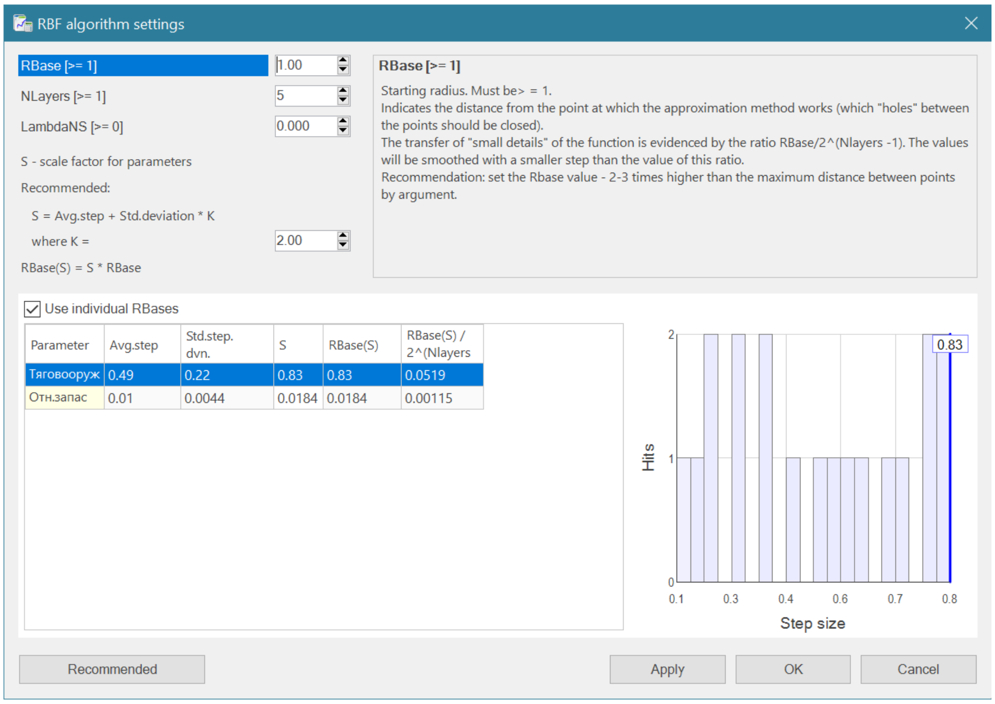
- RBF - Radial basis function interpolation;
- IDW Inverse distance weighting interpolation.

RBF - Radial basis function approximation

RBF - Radial basis function approximation. Hierarchical algorithm.
Parameters:

- **RBase** - initial radius. Must be > 0.
- **NLayers** - number of layers in the model. Must be > 0, recommended value to start with - about 5.
- **LambdaNS** - must be >= 0, nonlinearity penalty coefficient, negative values are not allowed.

This parameter adds controllable smoothing to the problem, which may reduce noise.
Specification of non->=0, nonlinearity penalty coefficient, negative values are not allowed.
This parameter adds controllable smoothing to the problem, which may reduce noise.
Specification of non-zero **LambdaNS** means that in addition to fitting error solver will also minimize LambdaNS*|S''(x)|^2 (appropriately generalized to multiple dimensions).
Specification of exactly zero value means that no penalty is added (we do not even evaluate matrix of second derivatives which is necessary for smoothing).
Calculation of nonlinearity penalty is costly - it results in several-fold increase of model construction time.
Evaluation time remains the same.
Optimal LambdaNS is problem-dependent and requires trial and error. Good value to start from is 1e-5...1e-6, which corresponds to slightly noticeable smoothing of the function.
Value 1e-2 usually means that quite heavy smoothing is applied.


RBF algorithm settings

RBase [>= 1]	1.00
NLayers [>= 1]	5
LambdaNS [>= 0]	0.000

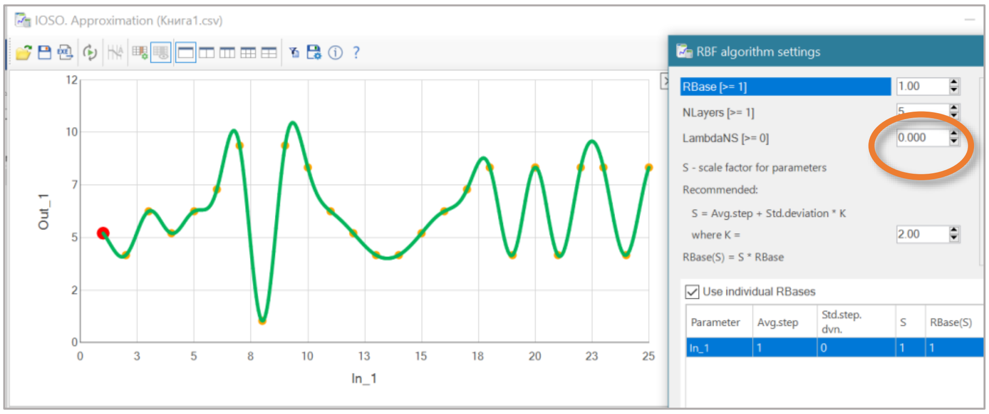
S - scale factor for parameters
Recommended:
S = Avg.step + Std.deviation * K
where K = 2.00
RBase(S) = S * RBase

RBase [>= 1]
Starting radius. Must be> = 1.
Indicates the distance from the point at which the approximation method works (which "holes" between the points should be closed).
The transfer of "small details" of the function is evidenced by the ratio RBase/2^(Nlayers -1). The values will be smoothed with a smaller step than the value of this ratio.
Recommendation: set the Rbase value - 2-3 times higher than the maximum distance between points by argument.

[✓] Use individual RBases

Parameter	Avg.step	Std.step. dvn.	S	RBase(S)	RBase(S) / 2^(Nlayers
Тяговооруж	0.49	0.22	0.83	0.83	0.0519
Отн.запас	0.01	0.0044	0.0184	0.0184	0.00115

RBF algorithm has controlled smoothing with non-linearity penalties.



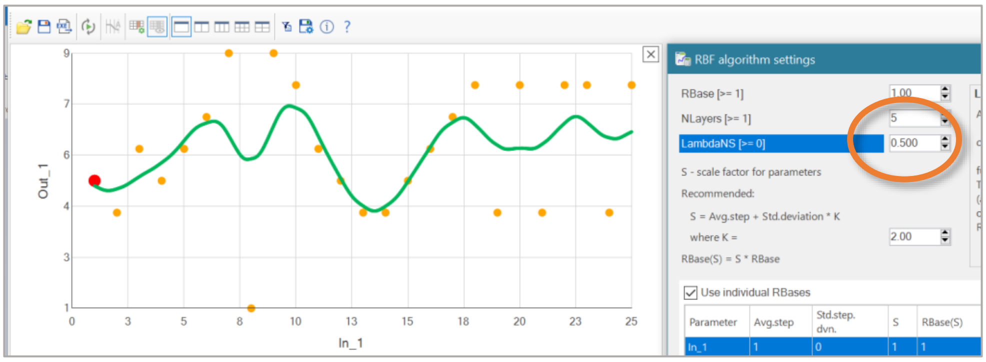
RBase [>= 1]	1.00
NLayers [>= 1]	5
LambdaNS [>= 0]	0.000

S - scale factor for parameters
Recommended:
S = Avg.step + Std.deviation * K
where K = 2.00
RBase(S) = S * RBase

☑ Use individual RBases

Parameter	Avg.step	Std.step. dvn.	S	RBase(S)
In_1	1	0	1	1

Penalty for non-linearity - LambdaNS = *0.*



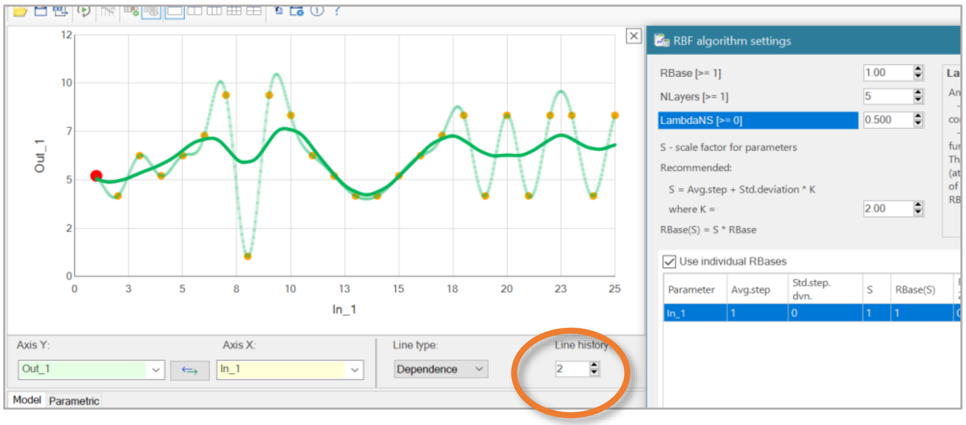
RBase [>= 1]	1.00
NLayers [>= 1]	5
LambdaNS [>= 0]	0.500

S - scale factor for parameters
Recommended:
S = Avg.step + Std.deviation * K
where K = 2.00
RBase(S) = S * RBase

☑ Use individual RBases

Parameter	Avg.step	Std.step. dvn.	S	RBase(S)
In_1	1	0	1	1

Penalty for non-linearity - LambdaNS = *0.5*.



RBase [>= 1]	1.00
NLayers [>= 1]	5
LambdaNS [>= 0]	0.500

S - scale factor for parameters
Recommended:
S = Avg.step + Std.deviation * K
where K = 2.00
RBase(S) = S * RBase

☑ Use individual RBases

Parameter	Avg.step	Std.step. dvn.	S	RBase(S)
In_1	1	0	1	1

Axis Y:	Axis X:	Line type:	Line history
Out_1	In_1	Dependence	2

The parameter " *Lines history*" allows you to track the history of changes - the previous charts are shown.

Introduction to RBF's

This article is intended for readers who needs introductory course on basic concepts of RBF interpolation.

Problem types

Interpolation problems can be separated into two main classes: problems with spatial variables (XYZ) and problems with mixed, spatial/temporal variables (XY-T). In the problems of the first kind all variables have same dimensions (spatial) and scale. In the problems of the second kind one variable (time) has special meaning, dimension (temporal) and scale. Fundamental limitation of all RBF methods is that they can successfully solve only problems of the first kind.

For example, consider an interpolation problem for a scalar grid in 2D space. There is a set of sensors which measure field magnitude with particular frequency. Average distance between sensors is 1m, average measurement interval is 1ms. Problem of the first kind is to build RBF model which described field state at some moment of time, using <u>only</u> measurements performed at the same time. Problem of the second kind is to describe evolution of the field *in time*, having a set of time series (measurements performed by sensors at different moments of time).

If you try to solve second problem, you will see that spatial and temporal variables have different scale. RBF is a basis function with radial symmetry, it has same span in all dimensions. Thus, either basis function radius will be equal to 1.0 - ideal for spatial interpolation, but too large for interpolation in time, or it will be equal to 0.001 - ideal for temporal interpolation, but too small to build good spatial model.

In theory, problem can be rescaled. However, optimal scaling coefficients are problem-specific, and there is not out-of-box solution for problems with mixed variables. This limitation is an intrinsic property of the RBF models. Many other interpolation methods, like multidimensional polynomials or splines, are invariant with respect to scaling of the variables. Such methods correctly work when different variables have different scale.

Basis functions

RBF model has following form: $f(x)=SUM(w_i \cdot \varphi(|x-c_i|))$. Here φ is a basis function, w_i are weight coefficients, c_i are interpolation centers. Interpolation centers usually coincide with original (input) grid, basis function is usually chosen from the list below, and weights are calculated as solution of the linear system. Following basis functions are used:

- *$f(r)=exp(-r^2/R0^2)$* � Gaussian basis function. The main benefit of this function is that it is compact - it is small at distances about *3·R0* from the center. At distances equal to *6·R0* or larger this function can be considered zero. As result, we have to solve linear system with sparse matrix. However, significant drawback is that we have one parameter to tune - *R0*. Too small *R0* will make model sharp and non-smooth, and too large one will result in system with ill-conditioned matrix.
- *$f(r)=sqrt(1+(r/R0)^2)$* � multiquadric basis function. Non-compact basis function, which is non-zero at any point. It is very important drawback - we have to solve systems with dense matrices. Another drawback - again, we have to tune *R0*.
- *$f(r)=r^{2k}·ln(r)$, $f(r)=r^{2k+1}$* � polyharmonic basis functions. Non-compact basis functions without tunable parameters. This is the most important advantage of polyharmonic basis. However, non-compactness, its main drawback, very often makes this basis unsuitable for medium and large scale problems. Linear systems with large dense matrices are hard to solve.

Implementation of RBF functions uses Gaussian basis. Compactness of the basis functions gives us high performance. We have to tune scale parameter *R0*, but from our point of view this drawback is insignificant when compared with ability to solve problems with tens and hundreds of thousands points.

Choosing good radius

<u>What does model radius mean?</u>

If we choose Gaussian basis function, we have to choose its tunable parameter - radius. Radius *R0* sets scale at which basis function is significantly non-zero. Gaussian basis function has its maximum at *r=0* (equal 1), it is approximately 0.36 at *r=R0*, it is approximately 0.02 at *r=2·R0*, it is approximately 0.0001 at *r=3·R0*. At distance *r=6·R0* or larger Gaussian basis function is indistinguishable from zero.

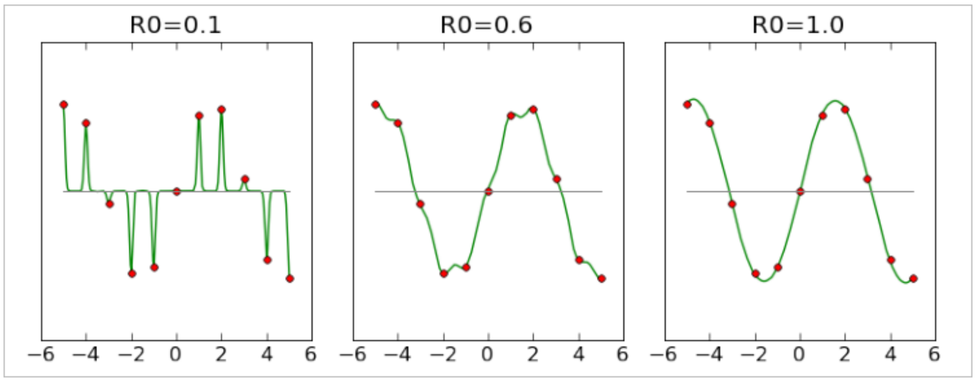
Another meaning of this parameter is a size of the "spheres of influence" of the points with known function values. Model value at *x* is determined by function values at nearby nodes, mostly by ones at distance *R0* or closer.

One more meaning of this parameter is a typical size of the "gap" in the grid, which can be "repaired" by RBF model. Consider evaluation of the RBF model at some point *x*, where function value is unknown If there are exists several nodes at distance *R0* or closer, model will reproduce original function with good precision. Best results are achieved when *x* is located between nodes, in the inner part of the interpolation area. However, if nearest nodes are located at distances about *2·R0*, then *x* won't be covered by their basis functions, and model quality at *x* will be low.

The larger radius we choose, the better will be RBF model. However, large radius usually means large condition number of the system, large error in the coefficients, slow model construction and evaluation. Smaller radius means small error in the coefficients, models which are quick to build and to evaluate. However, radius which is too small leads to rapid degradation of the model quality.

1D example

Consider 1-dimensional RBF interpolation problem. We have equidistant grid of 11 points with known function values, with unit step (red markers). We built three RBF models with different radii, and plotted them at charts below. Lets compare quality of these models.



First model was built with radius 0.1, ten times smaller than distance between points. You may see that it passes *exactly* through all points, but its behavior *between* them makes it useless. Second model has *R0=0.6* and it is significantly better than the first one. However, there is some degree of non-smoothness, sometimes it is clearly biased toward zero ("default value" for RBF models). Third model, which has *R0=1* (equal to distance between points) gives almost perfect result.

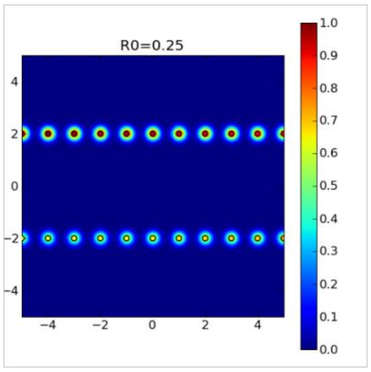
Note #1
We have not considered interpolation with larger radii, because charts which correspond to *R0 > 1* are visually indistinguishable from one built with *R0=1*.

From charts above we can conclude that radius should be at least equal to the average distance between points d_{avg}, and for safety reasons it is better to have radius which is several times larger than d_{avg}. Smaller radii lead to rapid degradation of the model. However, it is important to remember that large radius means larger condition number. Straightforward attempt to solve linear system will face problems with precision even at quite moderate *R=3*.

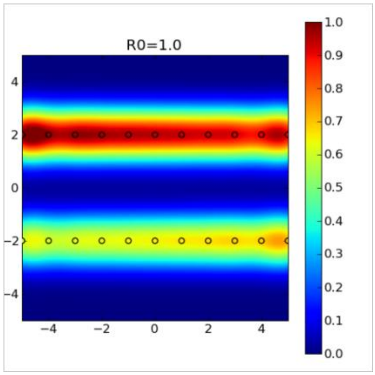
2D example

Above we've considered quite simple example - one-dimension, equidistant points. It was very simple to choose appropriate radius. Below we consider more complex example: points are located in 2D space at two nearby lines - *y=-2* and *y=+2*. At each of these lines we have equidistant grid with unit step. Distance between lines is four times larger than the grid step, and it complicates radius selection.

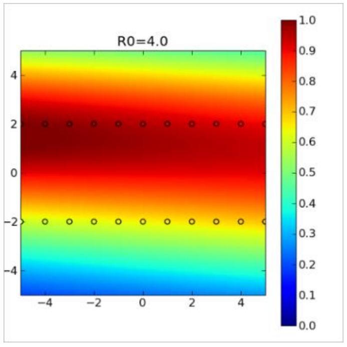
We've started from obviously insufficient radius *R0=0.25*. On the chart below we've plotted model values at each point of the 2D space. You may see that each point (black circle) has "sphere of influence" - colored bubble, which dissolves in the blue background (corresponds to zero, "default value" of the model, which is returned when there is no nearby nodes). However, these spheres do not overlap, model is almost zero in the gaps between them. This chart is a 2D analog of the first of 1-dimensional charts above.



With *R0=1* we got new chart (below). "Spheres of influence" grew larger and overlap with each other (red/yellow/green lines at the top and bottom). We can conclude that this model is a good description of the function as long as we stay near *y=-2* or *y=+2*. However, between these two lines there is a part of space which is not covered by the basic functions. At these points, marked by blue color, our model will return zero value.



Third chart was built with *R0=4* (which is equal to distance between *y=-2* and *y=+2*). This model is many times better than previous ones, because it reproduces function both at *y=-2*/*y=+2* and between them. You may see that basis functions covers space between any pair of points.



Conclusions

From the first example we can conclude that radius should be *at least* equal to average distance between nodes. Smaller values will give us sharp model which does not reproduce function behavior between nodes. However, average distance is only a lower bound - such radius selection rule works only when space is uniformly filled with nodes. In case there exists non-uniformity, some complex pattern in the distribution of nodes, then "average distance to the nearest neighbor" may be too small for good radius, which was shown by the second example. In the general case radius *R0* should so large that for any point where we want to evaluate model there will be several nodes at distance *R0* (or closer) around evaluation point.

Polynomial term

We can add linear term to the "basic" RBF model considered above: $f(x)=SUM(w_i \cdot \varphi(|x-c_i|)) + a^T \cdot x+b$. Presence of the linear term helps to reproduce global behavior of the function. Instead of linear term we can use polynomial term of the higher degree. From the other side, sometimes better option is to use constant term.

Model construction

After choice of the basis function (and, optionally, polynomial term) we have to build model, i.e. to calculate weight coefficients. There exists several approaches to the solution of this problem:

- "straightforward" approach � solution of the linear system using dense solver. It works with any basis function, but we have to live with low performance (solution time grows with number of points *N* as $O(N^3)$. Thus, several thousand of points is a limit for this approach.
- for compact (Gaussian) basis functions we can use sparse linear solver, either direct or iterative one. In this case matrix of the system will be sparse (the more compact basis function we choose, the more sparse matrix we will get). Because convergence of the iterative solver depends on the system conditioning, it will easily solve problems with small radius (approximately equal to the distance till the nearest neighbor), but will have difficulties with larger radii. From the other side, it does not need additional memory. Direct solver can solve problems with larger radii (up to 2-3 distances), but needs additional memory to store LU or Cholesky decomposition of the system.
- in case non-compact basis functions (polyharmony functions or multiquadric) are used, linear system will be dense. Direct dense solver will show better results that iterative one, but there exists complex strategy which allows to achieve good performance with iterative solver. In order to accelerate convergence, it is possible to use special kind of preconditioning - approximate cardinal basis functions (ACBF) preconditioning. In this case we compose several non-local basis functions to build another, local one (cardinal basis function). It would be great to have perfect local basis with unit condition number, but in reality, we have to limit ourselves by some approximate solution (that's why it is called "approximate"). In the new basis our problem can be solved within 5-10 iterations of GMRES solver. Usually, solution itself needs only minor fraction of the total processing time, and most time is spent in the construction of new basis.
- ACBF-preconditioning, which was described above, can be used for compact basis functions too. In this case compact functions become "more compact", which allows us to use iterative solvers with quite large radius, equal to 2 distances till the nearest neighbor.
- another solution which works only with compact basis functions is to separate original dataset into several smaller datasets. RBF construction problem is solved separately for these smaller parts, and then we compose partial models into larger, global one. Model composition is imperfect and we have artifacts near boundaries, so we have to solve interpolation problem one

more time - now to correct errors of the first model. As result, we got iterative algorithms, which repeats "separate-solve-compose-correct" loop several times before achieving desired accuracy.

- algorithms described above share one common trait - they solve original problem without any modifications. However, by slightly changing problem statement we can get good interpolant with many times lower performance penalty. For example, instead of N Gaussian functions with same radii we can use more functions with different radii. Interpolation problem can be solved in several steps, by building hierarchy of finer and finer model with decreasing radii. Such approach has several names: hierarchical RBF, multiscale RBF, multilevel RBF or multilayer RBF. Its main benefit - high performance, which achieved due to use of hierarchy of easy to build models.

Model evaluation

After construction of the RBF model we have to solve one more problem: to evaluate model at given point *x*. Depending on the specific choice of the basic functions (compact or non-compact), two approaches to the evaluation are possible:

- summation of all N basis function, "default" solution which can be used with any basis function. And it is the only solution which is possible for non-compact basis functions.
- summation of *nearby* basis functions, which are non-zero at the evaluation point. This option can be used only with compact basis functions, and its main benefit is high performance. Such solution requires data structures which support efficient spatial queries, like kd-trees.

Resume

Above we've studied RBF interpolation, basic concepts and subtle tricks, which are used to achieve better precision or performance. One particular implementation of the idea above is an algorithm, which:

- uses compact basis functions.
- can solve both interpolation and fitting/smoothing problems.

IDW Inverse distance weighting: interpolation.

IDW Inverse distance weighting: interpolation/fitting with improved Shepard-like algorithm.
Parameters:

- Search Radius - Initial search radius, > 0 is required. A model value is obtained by "smart" averaging of the dataset points within search radius.
- Layers Number - number of layers in the model.

By default, 16 layers is built, which is enough for most cases.
The more layers you have, the finer details can be reproduced with IDW model. The less layers you have, the less memory and CPU time is consumed by the model. Memory consumption grows linearly with layers count, running time grows sub-linearly. The default number of layers (16) allows you to reproduce details at distance down to 1/65536 of the search radius. You will rarely need to change it.

When comparing RBF and IDW, RBF has the properties of an "elastic metal sheet" and IDW has the "inelastic rubber sheet." Method IDW weak point - sag function at the edges of the array if the array has insufficient number of points - "inelastic sheet" at the edges sag, IDW method lacks support from surrounding points.

One-dimensional approximation methods.

- Polynomial Interpolation (PI)
- Polynomial Fitting (PF)
- Rational Interpolation (RI)
- Rational Fitting (RF)
- Linear Spline (SLI)
- Cubic Spline (SCI)
- Catmull-Rom Spline (SCRI)
- Akima Spline (SAI)
- Cubic Spline Fitting (SCF)
- Hermite Spline Fitting (SHF)
- Penalized Spline Fitting (SPF)

Polynomial interpolation.

Polynomial interpolation is the most known one-dimensional interpolation method. Its advantages lie in its simplicity of realization and the good quality of interpolants obtained from it. However, it has several disadvantages (some of them will be considered later) and is lately hard-pressed by alternative interpolation methods: splines and rational functions. But, in spite of that, polynomial interpolation is still one of the main tools of numerical analysis.

Parameters:
Polynom– polynomial variants. It can take one of the values:

- General - Lagrange polynomial.
- Equidistant - Lagrange polynomial: generation of the model on equidistant grid.

Polynomial fitting.

Polynomial functions fitting.
Parameters:

- Number of basis functions - must be greater than or equal to 1.

Rational interpolation.

The rational interpolation is one of the most difficult methods of interpolation. Its advantages are the high accuracy and absence of the problems which are typical for polynomial interpolation. At the same time, these methods have several weaknesses: for example, we can always find an interpolating polynomial for any set of points, but not all set of points have an interpolating rational function. Poles are also a big problem. They tend to appear unexpectedly. However, the last methods can solve most of the problems typical for the first rational interpolation methods. To make this clear, it is worth considering the evolution of rational interpolation algorithms.

The rational interpolation (i.e., interpolation by rational functions) consists of the representation of a given function as the quotient of two polynomials:

$$f(x) = \frac{p_0 + p_1 x + \ldots + p_L x^L}{q_0 + q_1 x + \ldots + q^N x^N}$$

Parallel with the spline interpolation , the rational interpolation is an alternative for the polynomial interpolation. The main disadvantage of the polynomial interpolation is that it is unstable on the most common grid - equidistant grid. If we are free to choose the grid, we can solve the problem by choosing the Chebyshev nodes. Otherwise (or if we just need a less node-dependent algorithm) the rational interpolation can be an appropriate alternative for the polynomial interpolation.

Rational functions interpolation.
Parameters:

- D - order of the interpolation scheme, 0 <= D <= N - 1.
  N - number of nodes, N > 0.

If you are unsure which D to choose, use a smaller value D (3 to 5).

Rational fitting.

Rational functions fitting.
Parameters:

- Number of basis functions - must be greater than or equal to 2.

Linear spline.

The linear spline is just a piecewise linear function. The linear splines have low precision, it should also be noted that they do not even provide first derivative continuity. However, in some cases, piecewise linear approximation could be better than higher degree approximation. For example, the linear spline keeps the monotony of a set of points..

Cubic spline.

All splines considered on this page are cubic splines - they are all piecewise cubic functions. However, if someone says "cubic spline", they usually mean a special cubic spline with continuous first and second derivatives. The cubic spline is given by the function values in the nodes and derivative values on the edges of the interpolation interval (either of the first or second derivatives).

- If the exact values of the first derivative in both boundaries are known, such spline is called *clamped spline*, or *spline with exact boundary conditions*. This spline has interpolation error $O(h^4)$.
- If the value of the first (or second) derivative is unknown, we can set the so-called natural boundary conditions $S''(A)=0$, $S''(B)=0$. Thus, we get a natural spline. The natural spline has interpolation error $O(h^2)$. The closer to the boundary nodes the more the error becomes. In the inner nodes the interpolation accuracy is much better.
- One more boundary condition which we can use when boundary derivatives are unknown is the "parabolically terminated spline". In this case, the boundary interval is represented as the second (instead of the third) degree polynomial (for inner intervals, third-degree polynomials are still used). In a number of cases this provides better accuracy than natural boundary conditions.
- We can also set periodic boundary conditions (this kind of conditions is used to model periodic functions).

At last, we can combine different types of boundary conditions for different boundaries. It does make sense if we have only partial information about the function behavior at the boundaries (e.g., we know the left boundary derivative, and have no information about the right boundary derivative).

Parameters:

- Monotone – values: True or False. Monotone cubic interpolation is a variant of cubic spline that preserves monotonicity of the data being interpolated.

Catmull-Rom spline.

Catmull-Rom spline is a Hermite spline whose derivatives are chosen to be

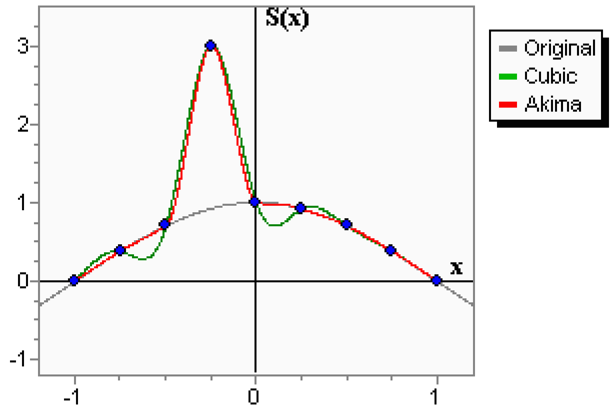
$$S'(x_i) = \frac{y_{i+1} - y_{i-1}}{x_{i+1} - x_{i-1}}$$

Catmull-Rom spline is continuous up to the first derivative; second derivative is discontinuous. It is local: spline values depend only on four function values (two on the left of x, two on the right). It supports two kinds of boundary conditions:

Parabolically terminated spline. In this case, the boundary interval is represented as the second (instead of the third) degree polynomial (for inner intervals, third-degree polynomials are still used). In a number of cases this provides better accuracy than natural boundary conditions.

Periodic boundary conditions (this kind of conditions is used to model periodic functions).

Akima spline.



The Akima spline is a special spline which is stable to the outliers. The disadvantage of cubic splines is that they could oscillate in the neighborhood of an outlier. On the graph you can see a set of points having one outlier. The cubic spline with boundary conditions is green-colored. On the intervals which are next to the outlier, the spline noticeably deviates from the given function - because of the outlier. Akima spline is red-colored. We can see that in contrast to the cubic spline, the Akima spline is less affected by the outliers.
An important property of the Akima spline is its locality - function values in $[x_i, x_{i+1}]$ depends on $f_{i-2}, f_{i-1}, f_i, f_{i+1}, f_{i+2}, f_{i+3}$ only. The second property which should be taken into account is the non-linearity of the Akima spline interpolation - the result of interpolation of the sum of two functions doesn't equal the sum of the interpolations schemes constructed on the basis of the given functions. No less than 5 points are required to construct the Akima spline. In the inner area (i.e. between x_2 and x_{N-3} when the index goes from 0 to N-1) the interpolation error has order $O(h^2)$.

Cubic spline fitting.

Least squares fitting by cubic spline.
Parameters:

- Number of basis functions – must be greater than or equal to 4.

Hermit spline fitting.

The cubic Hermite spline is a third-degree spline, whose derivative has given values in nodes. For each node not only the function value is given, but its first derivative value too. Hermite's cubic spline has a continuous first derivative, but its second derivative is discontinuous. The interpolation accuracy is much better than in the piecewise linear case.
Parameters:

- Number of basis functions – must be greater than or equal to 4.

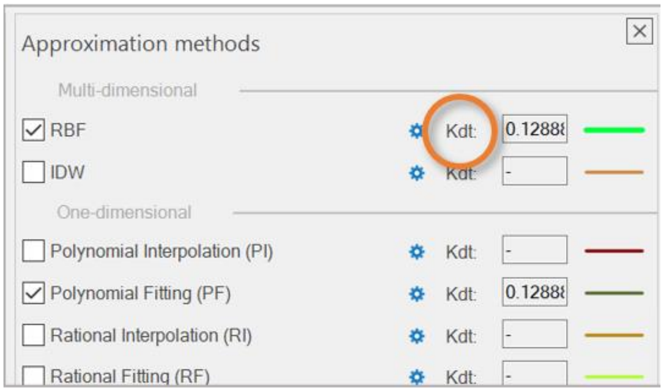
Penalized Spline Fitting.

Fitting by smoothing (penalized) cubic spline.
Approximates N scattered points (some of X[] may be equal to each other) by cubic spline with M nodes at equidistant grid spanning interval [min(x,xc),max(x,xc)].
Parameters:

- Number of basis functions – must be greater than or equal to 4.
- Regularization constant – must be greater than or equal to 0.

It penalizes nonlinearity in the regression spline.
Possible values to start from are 0.00001, 0.1, 1

The determination factor is used to estimate the quality of the approximation function- **Kdt**



Approximation methods

Multi-dimensional

✓ RBF	Kdt:	0.1288
☐ IDW	Kdt:	-

One-dimensional

☐ Polynomial Interpolation (PI)	Kdt:	-
✓ Polynomial Fitting (PF)	Kdt:	0.1288
☐ Rational Interpolation (RI)	Kdt:	-
☐ Rational Fitting (RF)	Kdt:	-

If there is no association of the approximation function with the data array, the determination coefficient is zero, and when the functional relationship is complete, it is one, that is, it determines which fraction of the variation of the characteristic Y is taken into account in the approximation model and is due to the influence of factors on it.

The determination factor (R2) is defined as follows:

$$\boxed{R^2 = \frac{\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} = 1 - \frac{\sum_{i=1}^{n} e_i^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}.}$$

Step 3. Saving the approximation model

Save the approximation project using the Save command- .

To save the created approximation function as an executable (* exe), use the "Build Approximation Model" command- **.**

This executable can be used as an independent calculation model in the IOSO optimization software or for other purposes