ROBUST ALGORITHM FOR ESTIMATION OF PARAMETERS IN NON-LINEAR REGRESSION MODELS

J. Tvrdík

University of Ostrava

Abstract

The paper deals with algorithms for estimation of non-linear regression parameters. Stochastic population-based algorithm with competition was implemented and compared with standard gradient algorithm commonly used for least-squares estimates. The results show that this stochastic algorithm found the global minimum in most tasks where gradient algorithm fails. Such population-based algorithms can be used as a tool for estimation of non-linear regression parameters, especially in tasks of higher difficulty level or in tasks when suitable starting values for gradient method are not available.

1 Introduction

In non-linear regression model, we suppose that elements of random vector $\mathbf{Y}$ are expressed as follows

$$Y_i = f(\mathbf{x}_i, \boldsymbol{\beta}) + \varepsilon_i, \quad i = 1, 2, \ldots, n, \tag{1}$$

where $\mathbf{x}_i^T = (x_1, x_2, \ldots, x_k)$ is i-th row of regressors matrix $\mathbf{X}$, $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_d)$ is vector of parameters, f is a given function non-linear in parameters, and ε_i's are independent identically distributed random variables with zero means. Estimation of parameters by the least squares method means to find such estimates of $\boldsymbol{\beta}$ that minimize the residual sum of squares $Q(\boldsymbol{\beta})$ given by the following equation

$$Q(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[Y_i - f(\mathbf{x}_i, \boldsymbol{\beta})\right]^2 \ . \tag{2}$$

The estimation of $\boldsymbol{\beta}$ is the global optimization problem because the objective function $Q(\boldsymbol{\beta})$ need not be unimodal. That is why iterative deterministic algorithms (like Levenberg-Marquardt or Gauss-Newton) used in standard statistical procedures often fail in finding the right solution of the problem. Several statistical packages (SPSS, S-Plus, NCSS, SYSTAT) [12] were tested on higher-level-difficulty tasks collected by NIST[8]. Either failure or significant disagreement with certified values were found in about one half tasks. Iterative deterministic can stop at a local minimum different from the global one in many tasks. Moreover, these algorithms need starting values of parameters and the choice of starting values has substantial influence on convergence of iteration process. Classical evolutionary algorithms [2] can be applied to the global optimization problems, but they are sensitive to the tuning of their controlling parameters. Statisticians and data analysts usually are not experts in such fine art of tuning. Statisticians require a robust algorithm giving reliable estimates at reasonable time consumption without changing default values of its controlling parameters. This paper attempts to propose such an algorithm.

2 Controlled Random Search with Competing Heuristics

Let us consider continuous global optimization problem with box constraints, i.e. for a given objective function $f : D \to \mathcal{R}$, $D \subset \mathcal{R}^d$, the point $\mathbf{x}^*$ is to be found such that $\mathbf{x}^* = \arg\min_{\mathbf{x} \in D} f(\mathbf{x})$. The point $\mathbf{x}^*$ is called the global minimum, D is the search space defined as $D = \prod_{i=1}^{d} [a_i, b_i]$, $a_i < b_i$, $i = 1, 2, \ldots, d$ and the objective function is computable, i.e. there is an efficient algorithm capable to evaluate $f(\mathbf{x})$ with sufficient accuracy in any point $\mathbf{x} \in D$.

Controlled random search (CRS) is a simple stochastic algorithm searching for the global minimum in problems defined above. CRS algorithm was proposed by Price [7] in 1977. There are several modifications of the CRS algorithm which were successfully used in solving the global optimization problems [1]. The CRS algorithm can be written in pseudo-code as follows:

```
1   generate P (population of N points in D at random);
2   find x_max (the point in P with the highest function value);
3   repeat
4     generate a new trial point y ∈ D by a heuristic;
5     if f(y) < f(x_max) then
6       x_max := y;
7       find new x_max;
8     endif
9   until stopping condition;
```

A heuristic used at line 4 is any non-deterministic rule generating a new trial point $\mathbf{y} \in D$. Price used the reflection in simplex known described by Nelder and Mead [6]. The simplex is $d+1$ point from P randomly chosen at each step. However, there are many different heuristics that can be used. Moreover, it is not necessary to use the same heuristic within the whole search process, several heuristics can alternate. Let us have h heuristics at our disposal and at each iteration step a heuristic is chosen at random with the probability $q_i,\ i = 1, 2, \ldots, h$.

The probabilities are changing according to the successfulness of heuristics in preceding steps of searching process. The heuristic is successful in the current step of evolutionary process if it generates such a trial point $\mathbf{y}$ that $f(\mathbf{y}) < f_{\max}$, $f_{\max} = f(\mathbf{x}_{\max})$. Probability q_i can be simply evaluated by frequency of success. Other way is the weighting of success by

$$w_i = \frac{f_{\max} - \max(f(\mathbf{y}), f_{\min})}{f_{\max} - f_{\min}}\,. \tag{3}$$

Thus $w_i \in (0,\ 1]$ and probability q_i is evaluated as

$$q_i = \frac{W_i + w_0}{\sum_{j=1}^{h}(W_j + w_0)}\,, \tag{4}$$

where W_i is sum of w_i in previous search and $w_0 > 0$ is an input parameter of the algorithm. In order to avoid the degeneration of evolutionary process the current values of q_i are reset to their starting values ($q_i = 1/h$) when any probability q_i decreases below a given limit $\delta > 0$. The CRS algorithm with competing heuristics belongs to the class of evolutionary algorithms described in [10] and [11].

Four competing heuristics were used in the implementation of the CRS algorithm for non-linear regression parameter estimate. Three of them are based on randomized reflection in the simplex S ($d+1$ points chosen from P) proposed in [3]. A new trial point $\mathbf{y}$ is generated from the simplex by the relation

$$\mathbf{y} = \mathbf{g} + U(\mathbf{g} - \mathbf{x}_H). \tag{5}$$

where $\mathbf{x}_H = \arg\max_{\mathbf{x} \in S} f(\mathbf{x})$ and $\mathbf{g}$ is the centroid of remaining d points of the simplex S. The multiplication factor U is a random variable distributed uniformly in $[s, \alpha - s)$, where $\alpha > 0$ and s are input parameters, $0 < s < \alpha/2$. All the $d+1$ points of simplex are chosen at random from P in two heuristics, the heuristic denoted REFL1 uses $\alpha = 2$, and $s = 0.5$, the heuristic denoted REFL25 uses $\alpha = 5$, and $s = 1.5$.

In the third heuristic denoted REFLB one point of simplex is the point of P with the minimal function value and remaining d points of simplex S are chosen at random from remaining points of P. Input parameters of the heuristic are set $\alpha = 2$ and $s = 0.5$.

The fourth competing heuristic is based on differential evolution [9]. A point $\mathbf{u}$ is generated according to

$$\mathbf{u} = \mathbf{r}_1 + F(\mathbf{r}_2 - \mathbf{r}_3)\,, \tag{6}$$

where $\mathbf{r}_1, \mathbf{r}_2$ and $\mathbf{r}_3$ are three distinct points taken randomly from P and $F > 0$ is an input parameter. The elements y_j, $j = 1, 2. \ldots, d$ of trial point $\mathbf{y}$ are built up by the crossover of randomly taken $\mathbf{x}$ (not coinciding with the current $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$) and $\mathbf{u}$ using the following rule:

$$y_j = \begin{cases} u_j & \text{if } U_j \le C \quad \text{or} \quad j = l \\ x_j & \text{if } U_j > C \quad \text{and} \quad j \ne l\,, \end{cases} \tag{7}$$

where l is a randomly chosen integer from $\{1, 2, \ldots, d\}$, $U_1, U_2, \ldots, U_d$ are independent random variables uniformly distributed in $[0,\ 1)$, and $C \in [0, 1]$ is an input parameter influencing the number of elements to be exchanged by crossover. Ali and Törn [1] suggested to adapt the value of the scaling factor F within searching process according to the equation

$$F = \begin{cases} \max(F_{\min}, 1 - |\frac{f_{\max}}{f_{\min}}|) & \text{if } |\frac{f_{\max}}{f_{\min}}| < 1 \\ \max(F_{\min}, 1 - |\frac{f_{\min}}{f_{\max}}|) & \text{otherwise,} \end{cases} \tag{8}$$

where $f_{\min}$, $f_{\max}$ are respectively the minimum and maximum function values in the population and $F_{\min}$ is an input parameter ensuring $F \in [F_{\min}, 1)$. The heuristic which uses Eq.(8) for evaluation of F with $F_{\min} = 0.4$ and $C = 0.9$ is denoted DERADP in the following text.

3 Numerical Experiments

One criterion for judging the accuracy of software output is comparison with "certified values" from sources that are known to be reliable. The NIST collection of datasets [8] collection contains 27 non-linear regression datasets (tasks) ordered by level of difficulty (lower – 8 tasks, average – 11 tasks, and higher – 8 tasks). The certified results are reported to 11 decimal places for each dataset. Numerical tests were carried out with all the NIST non-linear regression datasets. The results obtained by tested algorithms were evaluated by number of duplicated digits of the certified results. The number of duplicated digits λ can be calculated via *log relative error* [5] as

$$\lambda = \begin{cases} 0 & \text{if } \frac{|m-c|}{|c|} \ge 1 \\ 11 & \text{if } \frac{|m-c|}{|c|} < 1 \times 10^{-11} \\ -\log_{10}\left(\frac{|m-c|}{|c|}\right) & \text{otherwise}\,, \end{cases} \tag{9}$$

where c denotes the certified value and m denotes the estimated value. According to [8], except in cases where the certified value is essentially zero (for example, as occurs for the three Lanczos problems), a good non-linear least squares procedure should be able to duplicate the certified results to at least 4 or 5 digits. Two values of the number of duplicated digits are reported in our results, λ_Q is the agreement of residual sum of squares, see Eq. (2), and λ_β is the average agreement of estimated parameters $(\beta_1, \beta_2, \ldots, \beta_d)$. As it is seen later, producing correct results on all datasets of higher difficulty does not imply that the procedure will pass all datasets of average or even lower difficulty. Similarly, producing correct results for all datasets in this collection does not imply that the procedure will do the same for other particular dataset.

Two algorithms for estimation of parameters were compared in our numerical experiments. One of them is a modification of Levenberg-Marquardt algorithm implemented in `nlinfit` procedure of Statistical Toolbox [4]. Each dataset contains two starting d-tuples of parameters values for iterative deterministic algorithms of Levenberg-Marquardt type. The results obtained with these starting values are marked `Start1` and `Start2` in the following text. Especially values in `Start2` are very close to the certified values.

Controlling parameters of `nlinfit` were set to their default values, the calling statement in Matlab was

```
[beta,r,J]=nlinfit(X,y_obs,task_name,beta0);
```

The second algorithm is the CRS with four competing heuristics (CRS4CH) described in section 2. It is also implemented in Matlab, its source code is supplied in Appendix A. Because of its stochastic nature (initial population is chosen at random in D, new trial points are also generated non-deterministically), it must be tested in repeated runs. One hundred of repetitions were carried out for each task. Except λ_Q and λ_β two other variables are reported. Time consumption is expressed by average number (ne) of objective function evaluation needed to reach the stopping condition. The reliability RP of the search is measured as percentage of successful searches in which λ_Q exceeds a given value. The stopping condition was defined in all the tasks as

$$R^2_{\max} - R^2_{\min} < 1 \times 10^{-12}\,, \tag{10}$$

where $R^2_{\max}$ and $R^2_{\min}$ are the maximum and the minimum of the determination index in the population P. The determination index R^2 is defined by

$$R^2 = 1 - \frac{Q(\boldsymbol{\beta})}{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}\,. \tag{11}$$

The determination index R^2 specifies the part of total variability of Y which is explained by regression model.

Definitions of search spaces D for the test tasks are given in Appendix B. The tuning parameters of the CRS algorithm with competing heuristics were the same for all the tasks. This setting can be recommended as default for non-linear regression problems:

- population size, $N = 5d^2$,
- parameter w_0 used in Eq. 4, $w_0 = 0.5$,
- limit value δ for the reset of probabilities q_i to initial values, $\delta = 0.04$

4 Results

Overall comparison of both algorithms is shown in Tables 1 to 3 ordered according to level of difficulty. The results obtained by the `nlinfit` procedure can be accepted for all the tasks of lower difficulty level while the CRS4CH failed partly in three tasks where either λ_Q or RP are less than required – see Table 1. The CRS4CH was outperformed by `nlinfit` at lower level of difficulty. In the case of medium level tasks (Table 2) the `nlinfit` crashed in the MGH17 task (error in `nlinfit` at line 201) but the CRS4CH worked well. Both algorithms were not successful in Lanczos1. The CRS4CH did not achieve the required accuracy in other three tasks (Gauss3, Hahn1, Lanczos2). Concerning the higher level difficulty Table 3, `nlinfit` crashed in Boxbod task (error in `nlinfit` at line 201) and in three or even four tasks its performance was not satisfactory in log relative errors of the objective function. On the other hand the CRS4CH performed perfectly in seven tasks and its performance was almost perfect in the last task.

Table 1: Comparison of algorithms – low level of difficulty

			nlinfit				*CRS4CH*				
			Start1		*Start2*					$\lambda_Q > 4$	$\lambda_Q > 0$
task	d	$1 - R^2$	λ_Q	λ_β	λ_Q	λ_β	λ_Q	λ_β	*ne*	*RP*	*RP*
chwirut1	3	2.00E-02	10.8	9.0	10.6	8.9	11.0	6.3	3602	100	100
chwirut2	3	1.40E-02	11.0	9.1	11.0	9.2	11.0	6.1	3562	100	100
danwood	2	5.67E-04	11.0	10.2	11.0	9.9	10.3	6.7	1306	100	100
gauss1	8	3.04E-03	11.0	8.1	11.0	8.5	8.2	4.0	87177	80	80
gauss2	8	3.51E-03	10.6	8.3	10.6	8.5	1.8	1.1	71939	18	18
lanczos3	6	1.51E-09	10.6	8.1	10.6	8.1	3.1	0.8	80615	0	100
misra1a	2	1.84E-05	10.5	10.2	10.4	10.2	8.8	6.0	1807	100	100
misra1b	2	1.12E-05	11.0	10.2	11.0	10.2	8.5	6.1	1543	100	100

Table 2: Comparison of algorithms – medium level of difficulty

			nlinfit				*CRS4CH*				
			Start1		*Start2*					$\lambda_Q > 4$	$\lambda_Q > 0$
task	d	$1 - R^2$	λ_Q	λ_β	λ_Q	λ_β	λ_Q	λ_β	*ne*	*RP*	*RP*
enso	9	4.02E-01	8.4	4.9	8.1	4.7	11.0	5.3	107798	100	100
gauss3	8	3.10E-03	11.0	7.7	11.0	7.7	0.5	0.6	71442	5	5
hahn1	7	1.96E-04	10.6	7.0	10.6	6.0	1.4	1.3	44460	15	15
kirby2	5	2.85E-05	11.0	7.3	10.2	6.6	8.5	5.4	17533	100	100
lanczos1	6	1.34E-26	1.9	10.7	1.9	10.7	0.0	0.8	78809	0	0
lanczos2	6	2.10E-12	10.0	8.7	9.9	8.5	0.3	1.4	78120	0	99
mgh17	5	4.74E-05			11.0	7.6	8.6	5.4	22443	100	100
misra1c	2	6.06E-06	11.0	8.2	11.0	8.2	8.2	6.2	1840	100	100
misra1d	2	8.34E-06	11.0	10.3	11.0	10.3	8.5	6.2	1767	100	100
nelson	3	6.98E-02	10.9	8.9	10.9	9.0	10.9	6.5	7566	100	100
roszman1	4	1.59E-03	11.0	6.4	11.0	6.5	10.4	5.9	10383	100	100

Table 3: Comparison of algorithms – higher level of difficulty

			nlinfit				*CRS4CH*				
			Start1		*Start2*					$\lambda_Q > 4$	$\lambda_Q > 0$
task	d	$1 - R^2$	λ_Q	λ_β	λ_Q	λ_β	λ_Q	λ_β	*ne*	*RP*	*RP*
bennett5	3	1.06E-05	2.0	1.3	2.1	1.4	7.2	4.0	56434	100	100
boxbod	2	1.20E-01			10.1	5.7	10.4	7.2	1023	100	100
eckerle4	3	2.94E-03	0.0	0.0	10.6	7.7	10.4	7.7	3141	100	100
mgh09	4	5.94E-03	3.9	2.1	8.2	4.3	10.9	5.9	17994	100	100
mgh10	3	6.20E-08	0.0	2.0	0.0	2.5	6.1	5.1	30022	100	100
rat42	3	1.73E-03	11.0	7.0	10.9	7.0	10.5	6.7	3567	100	100
rat43	4	8.16E-03	10.0	5.7	11.0	5.9	11.0	6.2	7772	100	100
thurber	7	4.92E-04	7.8	5.0	8.2	5.2	9.5	6.1	43939	99	99

5 Discussion

The results obtained by the CRS4CH does not seem to be reliable enough to claim the robustness of the algorithm. But the results are not as bad as its smaller reliability on some lower- and medium-level-difficulty tasks showed. The failure occurred mainly in Lanczos and Gauss tasks. These tasks are based on generated data and the datasets are constructed as numerical trouble-makers. In addition, Lanczos tasks have very small values of R^2 (see Table 1 and 2). Such small variability unexplained by model does not occur in tasks when experimental data are processed. Thus, the significant frequency of failures was found in the case of Hahn1 dataset only. The cause of this failure will be analyzed later.

Moreover, the results of repeated runs of the CRS4CH even if they are not successful can be used as starting values of deterministic algorithms. In Table 4 there are the estimates obtained by the `nlinfit` when the estimates from one hundred runs of the CRS4CH were used as starting values. Three tuples of starting points were used: the centroid, minimal and maximal corner of the $d-$dimensional box of solution found by the CRS4CH. As can be seen in Table 4, at least one solution of each tasks is useful or even acceptable.

Table 4: Estimates from CRS4CH as starting values for nlinfit

		average		*minimum*		*maximum*	
task	*level*	λ_Q	λ_β	λ_Q	λ_β	λ_Q	λ_β
gauss1	lower	0.0	0.1	0.0	0.2	0.0	9.0
gauss2	lower	0.0	0.6	10.6	0.6	10.6	0.6
lanczos3	lower	10.6	1.1	4.7	1.1	4.8	1.1
gauss3	medium	0.0	0.6	11.0	0.6	0.0	0.3
hahn1	medium	0.0	0.0	0.0	0.0	10.6	3.3
lanczos1	medium	1.9	1.1	0.0	1.2	0.0	1.3
lanczos2	medium	10.3	1.1	0.3	1.2	1.0	1.2

The search process done by the CRS4CH can be characterized by some variables those values are the results of adaptation of the algorithm. Their averages are presented in Table 5, where the relative frequencies (per cent) are given in columns denoted by the names of heuristics. The column *rsucc* contains relative frequency of successful function evaluations (if $f(\mathbf{y}) < f_{\max}$), and *rst1000* is the number of resets of probabilities per 1000 objective function evaluations. As seen in Table 5, the frequencies of the use of heuristics differ among tasks, which shows the ability of the algorithm to adapt the search to the task. However, there is a question if the adaptability is sufficient. Comparison of two pairs of tasks (number of estimated parameters is the same in each pair) is shown in Figure 1 and 2. As is seen in Figure 1, for two tasks with different exploitation of heuristics within the search process the right values of estimates were found with perfect reliability. In this case the algorithm approved its adaptability. On other hand, we can see in Figure 2 that similar distribution of heuristics exploitation can give the result very different in the reliability of estimates. In this case the adaptability of the algorithm is not sufficient.

Table 5: Average values of search characteristics

		Relative frequencies of the use of heuristics					
level	*task*	REFL1	REFL25	REFLB	DEADP	rst1000	rsucc
lower	chwirut1	26.1	2.4	36.6	34.9	16.2	52.3
lower	chwirut2	26.4	2.5	37.1	34.0	16.5	52.3
lower	danwood	42.5	5.0	2.1	50.4	20.2	43.2
lower	gauss1	25.7	0.8	24.6	48.9	9.6	36.4
lower	gauss2	34.0	0.8	37.2	28.0	13.3	40.1
lower	lanczos3	37.7	1.2	46.2	14.9	12.3	33.8
lower	misra1a	42.2	5.3	3.1	49.4	18.6	44.2
lower	misra1b	44.1	6.7	3.2	46.0	17.6	41.6
medium	enso	28.7	0.5	30.4	40.3	9.6	29.8
medium	gauss3	28.9	0.7	31.0	39.4	11.5	37.0
medium	hahn1	32.3	1.2	36.0	30.4	12.3	39.1
medium	kirby2	33.8	1.1	38.2	26.9	13.3	38.6
medium	lanczos1	37.9	1.2	46.2	14.8	11.9	32.7
medium	lanczos2	38.1	1.2	46.0	14.8	12.1	33.2
medium	mgh17	34.0	1.0	39.8	25.3	13.2	38.3
medium	misra1c	46.5	7.4	3.8	42.2	15.3	45.2
medium	misra1d	45.8	6.8	4.5	42.9	14.5	45.5
medium	nelson	28.2	2.5	30.2	39.1	11.6	44.6
medium	roszman1	29.4	1.3	34.1	35.2	15.9	41.4
higher	bennett5	31.3	2.6	40.4	25.8	11.2	39.6
higher	boxbod	31.5	4.0	4.7	59.7	21.9	50.1
higher	eckerle4	22.7	1.7	27.8	47.8	18.8	50.3
higher	mgh09	29.3	1.5	35.7	33.6	13.6	41.1
higher	mgh10	58.5	1.5	23.2	16.8	7.4	47.7
higher	rat42	25.1	2.0	31.9	41.1	15.9	44.2
higher	rat43	27.8	1.0	33.3	37.8	15.9	41.3
higher	thurber	38.0	0.7	41.0	20.4	11.9	35.7

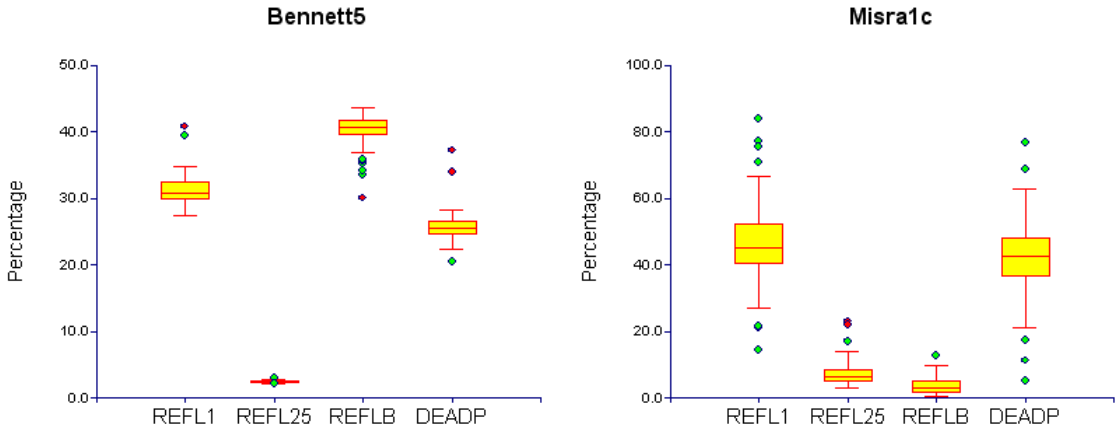

Figure 1: Different distributions of heuristics exploitation - both performed well

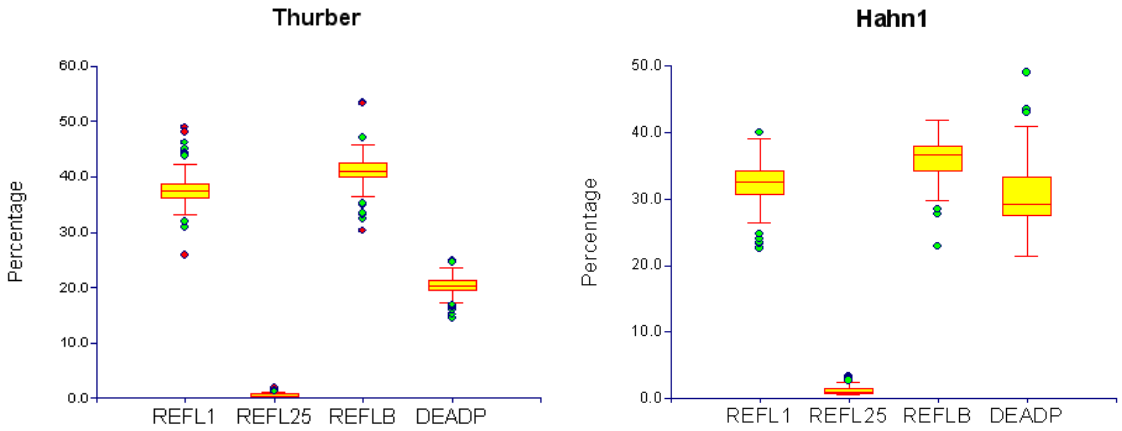

Figure 2: Similar distributions of heuristics exploitation - performation different

Simple look at relationship of variables influencing the search process give us their correlation matrix in Table 6. Correlation coefficients greater than 0.40 in absolute value are emphasized. The most surprising is high positive correlation between *ne* and the frequency of REFLB in spite of the fact that REFLB plays a role of local minimizer. It can be caused by more frequent exploitation of REFLB in tasks of higher dimension of the search space. The significant negative correlation between *ne* and *rsucc* is also surprising but the reason is likely the same as above. Another look at the dependency of most important variables (i.e. reliability expressed

Table 6: Correlation matrix

	d	λ_Q	*ne*	REFL1	REFL25	REFLB	DEADP	*rst1000*	*rsucc*
d	1.000	-0.489	0.820	-0.189	-0.633	0.549	-0.352	-0.399	-0.642
λ_Q	-0.489	1.000	-0.529	-0.160	0.216	-0.316	0.463	0.242	0.390
ne	0.820	-0.529	1.000	-0.047	-0.480	0.452	-0.391	-0.469	-0.730
REFL1	-0.189	-0.160	-0.047	1.000	0.210	-0.448	-0.401	-0.214	0.050
REFL25	-0.633	0.216	-0.480	0.210	1.000	-0.681	0.419	0.269	0.277
REFLB	0.549	-0.316	0.452	-0.448	-0.681	1.000	-0.632	-0.272	-0.379
DEADP	-0.352	0.463	-0.391	-0.401	0.419	-0.632	1.000	0.451	0.341
rst1000	-0.399	0.242	-0.469	-0.214	0.269	-0.272	0.451	1.000	0.430
rsucc	-0.642	0.390	-0.730	0.050	0.277	-0.379	0.341	0.430	1.000

by λ_Q and time consumption *ne*) can be seen in Table 7 where the results of linear regression are shown. The regressors are selected by stepwise procedure among variables in Table 5. The results of the regression indicate that both λ_Q and *ne* are influenced by the number of resets (dependent on input parameter δ) and by the choice of heuristics to competition. The next research of more reliable and less time consuming algorithm for non-linear parameter estimate should go this way.

Table 7: Regression analysis

	dependent: ne, $R^2 = 0.75$					*dependent:* λ, $R^2 = 0.39$			
	b(i)	sb(i)	T	p		b(i)	sb(i)	T	p
Intercept	56491	3754.0	15.0	0.0000	Intercept	8.754	0.426	20.5	0.0000
d	9686	211.9	45.7	0.0000	d	-1.047	0.037	-28.0	0.0000
REFL1	195	31.2	6.2	0.0000	DEADP	0.143	0.006	22.8	0.0000
relsucc	-1611	66.0	-24.4	0.0000	REFL25	-0.428	0.038	-11.2	0.0000
rst1000	-542	75.3	-7.2	0.0000	REFLB	0.028	0.006	4.3	0.0000
					rst1000	-0.082	0.013	-6.2	0.0000

6 Conclusions

The CRS4CH algorithm outperforms the `nlinfit` procedure in the tasks of higher difficulty, but the presented version of the algorithm is not satisfactory successful in some task of medium- and lower-difficulty. Next research of its behaviour is needed. In spite of this fact, the present version of CRS4CH algorithm can serve as an alternative tool for estimation of parameters in non-linear regression models. The implementation of the algorithm in Matlab is presented in Appendix A and it is also available at author's web site. Advantage of the algorithm is that it does not need the specification of starting values like deterministic iterative algorithms commonly used in non-linear parameter estimates.

References

[1] Ali, M.M., Törn, A.: Population set based global optimization algorithms: Some modifications and numerical studies, Computers and Operations Research **31** (2004) 1703 – 1725.

[2] Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, New York (1996)

[3] Křivý, I., Tvrdík, J.: The Controlled Random Search Algorithm in Optimizing Regression Models. Comput. Statist. and Data Anal. **20** (1995) 229 – 234.

[4] MATLAB, version 7.1.0, The MathWorks, Inc. (2005).

[5] McCullough, B.D., Wilson, B.: On the accuracy of statistical procedures in Microsoft Excel 2003. Comput. Statist. and Data Anal. **49** (2005) 1244 – 1252.

[6] Nelder, J.A., Mead, R.: A Simplex Method for Function Minimization. Computer J. **7** (1964) 308 – 313.

[7] Price, W. L.: A Controlled Random Search Procedure for Global Optimization. Computer J. **20** (1977) 367 – 370.

[8] Statistical Reference Datasets. Nonlinear regression. NIST Information Technology Laboratory. http://www.itl.nist.gov/div898/strd/. December 1, 2001.

[9] Storn, R., Price, K.: Differential evolution - a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. J. Global Optimization **11** (1997) 341 – 359.

[10] Tvrdík, J., Křivý, I., Mišík, L.: Evolutionary Algorithm with Competing Heuristics. In: Ošmera, P. (ed.): MENDEL 2001, 7th International Conference on Soft Computing. Technical University, Brno (2001) 58 – 64.

[11] Tvrdík, J., Mišík, L., Křivý, I.: Competing Heuristics in Evolutionary Algorithms. In: Sinčák, P. et al. (eds.): Intelligent Technologies – Theory and Applications. IOS Press, Amsterdam (2002) 159 – 165.

[12] Tvrdík, J., Křivý, I.: Comparison of algorithms for nonlinear regression estimates. In: Antoch, J. (ed.): COMPSTAT 2004. Physica-Verlag (2004) 1917 – 1924.

This work was supported by the grant 201/05/0284 of the Czech Grant Agency and by the institutional research MSM 6198898701 solved in the Institute for Research and Applications of Fuzzy Modeling, University of Ostrava.

Josef Tvrdík
Department of Computer Science, University of Ostrava
30. dubna 22, 701 03 Ostrava
tvrdik@osu.cz
www.albert.osu.cz/tvrdik

A Source Code in MATLAB

```
function [x_star, fn_star, R2, func_evals, rsuc, nrst, ri]=...
    nlr_crs4ch(mod_name, y_obs, X, a, b,  N, my_eps, max_evals, delta, w0, int_disp)
%
% competing heuristics for non-linear regression (simple version)
% Tvrdik, October 2004
% input parameters
% obligatory:
%   mod_name    non-lin model (M file)
%   y_obs       dependent (column vector)
%   X           matrix of regressors
%   a, b        row vectors, limits of search space, a < b (box constraints)
%   N           size of population
%   my_eps      small positive value for stopping criterion
%   max_evals   max. evals per one dimension of search space
%   delta       if any probability qi is less than delta, reset of qi
%   w0          positive number (w0=0.5 recomended) for success evaluation
%
% facultative:
%   int_disp    positive integer, number of function evaluation for
%               displaying current results
%
% output:
%   x_star          global minimum found by search
%                   (estimates of parameters)
%
%   fn_star         the minimal function value found by CRS4CH
%                   (residual sum of squares)
%
%   R2              R squared, index of determination
%
%   func_evals      number of func_evals for reaching stopping condition
%
%   rsuc            relative frequency (per cent) of successful trial points y
%
%   ri              relative frequency (per cent) of success of each heuristic
%                   (row vector with 4 elements]
%
%   nrst            number of resets
%
h=4;  % number of   heuristics in competition
d=length(a);
% initialization
P=zeros(N,d);
for i=1:N
    P(i,:)=a+(b-a).*rand(1,d);
end
for i=1:N
    P(i,d+1)= rss(mod_name,P(i,1:d),y_obs,X);
end  % 0-th generation initialized
```

```
[fmax, indmax]=max(P(:,d+1));
[fmin, indmin]=min(P(:,d+1));
func_evals=N; success=0;   ni=zeros(1,h);  nrst=0; wi=zeros(1,h)+w0;
tss=(y_obs-mean(y_obs))'*(y_obs-mean(y_obs));
tss_myeps=tss*my_eps;
while (fmax-fmin > tss_myeps) & (func_evals < d*max_evals)  % main loop
    [hh p_min]=roulete(wi);
    if p_min<delta
        wi=zeros(1,h)+w0;
        nrst=nrst+1;
    end  %reset
    switch hh % number of selected heuristic
        case 1
            y=refl_rwd(P,[2 0.5]);
        case 2
            y=refl_rwd(P,[5 1.5]);
        case 3
            y=refl_bestrwd(P, 2, 0.5, indmin, P(indmin,1:d));
        case 4
            y=decrs_radp(P,[0.4 0.9 fmin fmax]);
    end
    y=zrcad(y,a,b);     % perturbation
    fy=rss(mod_name,y,y_obs,X);
    func_evals=func_evals+1;
    if fy < fmax  % trial point y is good for renewing population
        P(indmax,:)=  [y fy];
        success=success+1;
        ni(hh)=ni(hh)+1;
        if fmax-fmin <= eps    w=w0;
        else w=(fmax-max([fy fmin]))/(fmax-fmin); % weight of increase, <= 1
        end
        wi(hh)=wi(hh)+w;  %  qi will be changed
        [fmax, indmax]=max(P(:,d+1));
        [fmin, indmin]=min(P(:,d+1));
    end
    if nargin==11 & mod(func_evals,int_disp)==0 disp(P(indmin,:)); end
end   % main loop - end
x_star=P(indmin,1:d);
fn_star=fmin;
R2=1-fmin/tss;
rsuc=100*success/func_evals;
ri=100*ni/success;
%
```

```
function y=refl_rwd(P,alpha);
% randomized shifted  - worst (with highest f) point of simplex is reflected
% see  Tvrdik 2004
shift_d=alpha(2); % shift_d < alpha(1)
alpha=alpha(1);
N=length(P(:,1));  d=length(P(1,:))-1;
vyb=nahvyb(N,d+1);  % simplex S chosen at random from P
S=P(vyb,:);
[x,indx]=max(S(:,d+1)); % worst point is reflected
x=S(indx,1:d);
S(indx,:)=[];
S(:,d+1)=[];
g=mean(S);
y=g+(g-x)*(shift_d +(alpha-2*shift_d)*rand(1));
% randomized shifted reflection
% expected value = alpha/2


% random sample, k of N without repetiotion
function nahv=nahvyb(N,k);
opora=1:N;
nahv=zeros(1,k);
for i=1:k
    index=1+fix(rand(1)*length(opora));
    nahv(i)=opora(index);
    opora(index)=[];
end
%
```

```
function y=refl_bestrwd(P,alpha,shift_d, indmin, xmin);
% randomized, shifted  - worst (with highest f) point of simplex is reflected
% centroid g computed also from xmin
% see Ali and Torn 2004
N=length(P(:,1));  d=length(P(1,:))-1;
vyb=nahvyb_expt(N,d,indmin);  % d points chosen at random from P except indmin
S=P(vyb,:);
[x,indx]=max(S(:,d+1)); % worst point is reflected
x=S(indx,1:d);
S(indx,:)=[];
S(:,d+1)=[];
g=(xmin + sum(S))/d;
y=g+(g-x)*(shift_d +(alpha-2*shift_d)*rand(1));    % randomized reflection


% random sample, k of N without repetition,
% numbers given in vector expt are not included
%
function vyb=nahvyb_expt(N,k,expt);
opora=1:N;
if nargin==3 opora(expt)=[]; end
vyb=zeros(1,k);
for i=1:k
    index=1+fix(rand(1)*length(opora));
    vyb(i)=opora(index);
    opora(index)=[];
end


% heuristic - de_rand binomial
% F is adapted, see Ali and Torn 2004
function y=decrs_radp(P,vecpar);
Fmin=vecpar(1); CR=vecpar(2); fmin=vecpar(3); fmax=vecpar(4);
N=length(P(:,1)); d=length(P(1,:))-1;
vyb=nahvyb(N,4); % four random points
r1=P(vyb(1),1:d);
r2=P(vyb(2),1:d);
r3=P(vyb(3),1:d);
y=P(vyb(4),1:d);
pom1=Fmin; pom2=1; pom3=1;
if abs(fmin)>0 pom2=abs(fmax/fmin); end
if pom2<1  pom1=1-pom2;
elseif abs(fmax)>0 pom1=1-abs(fmin/fmax);
end
F=max([Fmin, pom1]);
v=r1+F*(r2-r3);
change=find(rand(1,d)<CR);
if length(change)==0 % at least one element is changed
    change=1+fix(d*rand(1));
end
y(change)=v(change);
```

```
function [res, p_min]=roulete(cutpoints)
%
% returns an integer from [1, length(cutpoints)] with probability proportional
% to cutpoints(i)/ summa cutpoints
%
h =length(cutpoints);
ss=sum(cutpoints);
p_min=min(cutpoints)/ss;
cp(1)=cutpoints(1);
for i=2:h
    cp(i)=cp(i-1)+cutpoints(i);
end
cp=cp/ss;
res=1+fix(sum(cp<rand(1)));


function result=rss(mod_name, b, y, X);
%
%   RSS pro odhady b
%
y_hat= feval(mod_name,b, X);
% size(y_hat)
result=(y-y_hat)'*(y-y_hat);


% zrcadleni, Perturbation y into <a,b>
function result=zrcad(y,a,b)
zrc=find(y<a|y>b);
for i=zrc
    while (y(i)<a(i)|y(i)>b(i))
        if y(i)>b(i)
            y(i)=2*b(i)-y(i);
        elseif y(i)<a(i)
            y(i)=2*a(i)-y(i);
        end
    end
end
result=y;
```

B Search Spaces of NIST Tasks

Table 8: Search spaces – Part 1

task	enso		gauss1		gauss2		gauss3	
i	a_i	b_i	a_i	b_i	a_i	b_i	a_i	b_i
1	0	20	0	1000	0	1000	0	1000
2	-100	100	0	1	0	0.5	0	1
3	-100	100	0	1000	0	1000	0	1000
4	-100	100	0	1000	0	300	0	1000
5	-100	100	0.0001	100	0.5	1000	0.0001	100
6	-100	100	0	1000	0	1000	0	1000
7	-100	100	0	1000	0	300	0	1000
8	-100	100	0.0001	100	0.5	1000	0.0001	100
9	-100	100						

Table 9: Search spaces – Part 2

task	hahn1		thurber		lanczos1		lanczos2		lanczos3	
i	a_i	b_i	a_i	b_i	a_i	b_i	a_i	b_i	a_i	b_i
1	0	100	0	10000	0	100	0	100	0	100
2	-1	1	0	5000	0	10	0	10	0	10
3	-0.1	0.1	0	5000	0	100	0	100	0	100
4	-0.1	0.1	0	1000	0	10	0	10	0	10
5	-0.1	0.1	0	10	0	100	0	100	0	100
6	-0.1	0.1	0	10	0	10	0	10	0	10
7	-0.1	0.1	0	10						

Table 10: Search spaces – Part 3

task	kirby2		mgh09		mgh10		mgh17		rozsman1	
i	a_i	b_i	a_i	b_i	a_i	b_i	a_i	b_i	a_i	b_i
1	0	10	0	100	0	100	0	10	0	100
2	-1	0	0	100	0	1000000	0	10	-10	10
3	0	1	0	100	0	100000	-20	-0.001	-5000	5000
4	-1	0	0.01	100			0	1	-450	0
5	0	1					0	1		

Table 11: Search spaces – Part 4

task	misra1a		misra1b		misra1c		misra1d	
i	a_i	b_i	a_i	b_i	a_i	b_i	a_i	b_i
1	-10000	10000	0	2000	0	10000	0	10000
2	-0.2	0.2	0	0.1	0	1	0	1

Table 12: Search spaces – Part 5

task	chwirut1		chwirut2		nelson		danwood		boxbod	
i	a_i	b_i	a_i	b_i	a_i	b_i	a_i	b_i	a_i	b_i
1	0	10	0	10	1	10	0	100	1	1000
2	0	1000	0	1000	0	1	1	10	0.1	2
3	0	100	0	100	-1	0				

Table 13: Search spaces – Part 6

task	bennet5		eckerle4		rat42		rat43	
i	a_i	b_i	a_i	b_i	a_i	b_i	a_i	b_i
1	-5000	-1000	0	10	0	1000	0	1000
2	0	500	1	10	0	10	0	100
3	0.1	10	400	500	0	1	0	1
4							0.1	10