Performance of Differential Evolution and Particle Swarm Methods on Some Relatively Harder Multi-modal Benchmark Functions

SK Mishra

North-Eastern Hill University, Shillong (India)

13. October 2006

Online at http://mpra.ub.uni-muenchen.de/1743/
I. Introduction: Global optimization (GO) is concerned with finding the optimum point(s) of a non-convex (multi-modal) function in an m-dimensional space. Although some work was done in the pre-1970’s to develop appropriate methods to find the optimal point(s) of a multi-modal function, the 1970’s evidenced a great fillip in simulation-based optimization research due to the invention of the ‘genetic algorithm’ by John Holland (1975). A number of other methods of global optimization were soon developed. Among them, the ‘Clustering Method’ of Aimo Törn (1978), the “Simulated Annealing Method” of Kirkpatrick and others (1983) and Cerny (1985), “Tabu Search Method” of Fred Glover (1986), the “Ant Colony Algorithm” of Dorigo (1992), the “Particle Swarm Method” of Kennedy and Eberhart (1995) and the “Differential Evolution Method” of Price and Storn (1996) are quite effective and popular. All these methods use the one or the other stochastic process to search the global optima.

II. The Characteristic Features of Stochastic GO Methods: All stochastic search methods of global optimization partake of the probabilistic nature inherent to them. As a result, one cannot obtain certainty in their results, unless they are permitted to go in for indefinitely large search attempts. Larger is the number of attempts, greater is the probability that they would find out the global optimum, but even then it would not reach at the certainty. Secondly, all of them adapt themselves to the surface on which they find the global optimum. The scheme of adaptation is largely based on some guesswork since nobody knows as to the true nature of the problem (environment or surface) and the most suitable scheme of adaptation to fit the given environment. Surfaces may be varied and different for different functions. A particular type of surface may be suited to a particular method while a search in another type of surface may be a difficult proposition for it. Further, each of these methods operates with a number of parameters that may be changed at choice to make it more effective. This choice is often problem oriented and for obvious reasons. A particular choice may be extremely effective in a few cases, but it might be ineffective (or counterproductive) in certain other cases. Additionally, there is a relation of trade-off among those parameters. These features make all these methods a subject of trial and error exercises.

III. The Objectives: Our objective in this paper is to compare the performance of the Differential Evolution (DE) and the Repulsive Particle Swarm (RPS) methods of global optimization. To this end, some relatively difficult test functions have been chosen. Among these test functions, some are new while others are well known in the literature.

IV. Details of the Test Functions used in this Study: The following test (benchmark) functions have been used in this study.
(1) **Perm function #1**: In the domain \( x \in [-4, 4] \), the function has \( f_{\min} = 0 \) for \( x = (1, 2, 3, 4) \). It is specified as

\[
f(x) = \sum_{i=1}^{4} \left( \sum_{l=1}^{4} (i^4 + \beta)((x_i/i)^4 - 1) \right)^2
\]

The value of \( \beta \) (=50) introduces difficulty to optimization. Smaller values of beta raise this difficulty further.

(2) **Perm function #2**: In the domain \( x \in [-1, 1] \), and for a given \( \beta \) (=10), this \( m \)-variable function has \( f_{\min} = 0 \) for \( x_i = (i)^{-1} \); \( i = 1, 2, ..., m \). It is specified as

\[
\sum_{i=1}^{4} \left[ \sum_{l=1}^{4} (i + \beta)((x_j)^4 - (i)^{-1}) \right]^2
\]

Smaller values of beta raise difficulty in optimization.

(3) **Power-sum function**: Defined on four variables in the domain \( x \in [0, 4] \), this function has \( f_{\min} = 0 \) for any permutation of \( x = (1, 2, 2, 3) \). The function is defined as

\[
f(x) = \sum_{i=1}^{4} \left( b_k - \sum_{i=1}^{4} x_i \right)^2 \quad b_k = (8, 18, 44, 114) \text{ for } k = (1, 2, 3, 4)
\]

(4) **Bukin’s functions**: Bukin’s functions are almost fractal (with fine seesaw edges) in the surroundings of their minimal points. Due to this property, they are extremely difficult to optimize by any method of global (or local) optimization and find correct values of decision variables (i.e. \( x_i \) for \( i = 1, 2 \)). In the search domain \( x_i \in [-15, -5], x_2 \in [-3, 3] \) the \( 6^{th} \) Bukin’s function is defined as follows.

\[
f_6(x) = 100 \sqrt{|x_2 - 0.01x_1^2|} + 0.01|x_1 + 10| \quad ; f_{\min}(-10, 1) = 0
\]

(5) **Zero-sum function**: Defined in the domain \( x \in [-10, 10] \) this function (in \( m \geq 2 \)) has \( f(x) = 0 \) if \( \sum_{i=1}^{m} x_i = 0 \). Otherwise \( f(x) = 1 + \left( 10000 \left| \sum_{i=1}^{m} x_i \right| \right)^{0.5} \). This function has innumerably many minima but it is extremely difficult to obtain any of them. Larger is the value of \( m \) (dimension), it becomes more difficult to optimize the function.

(6) **Hougen function**: Hougen function is typical complex test for classical non-linear regression problems. The Hougen-Watson model for reaction kinetics is an example of such non-linear regression problem. The form of the model is

\[
\text{rate} = \frac{\beta_3 x_2 - x_1 / \beta_3}{1 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3}
\]

where the betas are the unknown parameters, \( x = (x_1, x_2, x_3) \) are the explanatory variables and ‘rate’ is the dependent variable. The parameters are estimated via the least squares criterion. That is, the parameters are such that the sum of the squared differences between the observed responses and their fitted values of rate is minimized. The input data given alongside are used.
The best results are obtained by the Rosenbrock-Quasi-Newton method: 
\[ \hat{\beta}_1 = 1.253031; \quad \hat{\beta}_2 = 1.190943; \quad \hat{\beta}_3 = 0.062798; \quad \hat{\beta}_4 = 0.040063; \quad \hat{\beta}_5 = 0.112453. \] 
The sum of squares of deviations \( (S^2) = f_{\text{min}} \) is \( = 0.298900994 \) and the coefficient of correlation \( (R) \) between observed rate and expected rate is \( = 0.99945. \)

(7) **Giunta function**: In the search domain \( x_1, x_2 \in [-1, 1] \) this function is defined as follows and has \( f_{\text{min}}(0.45834, 0.45834) = 0.0602472 \) (Silagadze, 2004), which appears to be incorrect. We get \( f_{\text{min}}(0.46732, 0.46732) = 0.06447 \). 
\[ f(x) = 0.6 + \sum_{i=1}^{2} \left[ \sin(15x_i - 1) + \sin^2(15x_i - 1) + \frac{1}{80} \sin(4(15x_i - 1)) \right]. \]

(8) **DCS function**: The generalized deflected corrugated spring function is an \( m \)-variable function with \( f_{\text{min}}(0.19, 0.1, 0, 0) = 0.3075 \). This function is given as:
\[ f(x) = \frac{1}{10} \sum_{i=1}^{m} (x_i - c_i)^2 - \cos \left[ k \left( \sum_{i=1}^{m} (x_i - c_i)^2 \right)^{0.5} \right]; \quad x \in [-20, 20] \]

(9) **Kowalik or Yao-Liu #15 function**: It is a 4-variable function in the domain \( x \in [-5, 5] \), that has the global minimum \( f_{\text{min}}(0.19, 0.19, 0.12, 0.14) = 0.3075 \). This function is given as:
\[ f(x) = 1000 \sum_{i=1}^{4} \left( a_i - \frac{x_i(b_i^2 + b_i x_i)}{2b_i + b_i x_i + x_i} \right)^2; \quad \text{where } b = \left( \frac{1}{0.25}, \frac{1}{0.5}, \frac{1}{1}, \frac{1}{2}, \frac{1}{4}, \frac{1}{6}, \frac{1}{8}, \frac{1}{10}, \frac{1}{12}, \frac{1}{14}, \frac{1}{16} \right) \] and 
\[ a = (0.1957, 0.1947, 0.1735, 0.1600, 0.0844, 0.0627, 0.0456, 0.0342, 0.0235, 0.0246). \]

(10) **Yao-Liu #7 function**: It is an \( m \)-variable function in the domain \( x \in [-1.28, 1.28] \), that has the global minimum \( f_{\text{min}}(0, 0, ..., 0) = 0 \). This function is given as:
\[ f(x) = \text{rand}[0,1] + \sum_{i=1}^{m} (x_i) \]

(11) **Fletcher-Powell function**: This is an \( m \)-variable function with \( f_{\text{min}}(c_1, c_2, ..., c_m) = 0 \) given as follows:
\[ f(x) = \sum_{i=1}^{m} (A_i - B_i)^2, \]
where 
\[ A_i = \sum_{j=1}^{m} \left[ u_{ij} \sin(c_j) + v_{ij} \cos(c_j) \right]; \quad B_i = \sum_{j=1}^{m} \left[ u_{ij} \sin(x_j) + v_{ij} \cos(x_j) \right]; \quad u_{ij}, v_{ij} = \text{rand}[-100, 100] \]
and \( x, c \in [-\pi, \pi] \). Moreover, \( c \) could be stochastic or fixed. When \( c \) is fixed, it is easier to optimize, but for stochastic \( c \) optimization is quite difficult. One may visualize \( c \) as a vector that has two parts: \( c_1 \) (fixed) and \( r \) (random). Either of them could be zero or both of them could be non-zero.

(12) **New function 1**: This function (with \( f_{\text{min}}(1, 1, ..., 1) = 2 \)) may be defined as follows:
\[ f(x) = (1 + x_m)^{\alpha}; \quad x_m - m - \sum_{i=1}^{m} x_i; \quad x \in [0, 1] \forall i = 1, 2, ..., m \]
This function is not very difficult to optimize. However, its modification that gives us a new function (#2) is considerably difficult.

(13) **New function 2**:: This function (with \( f_{\text{min}}(1, 1, \ldots, 1) = 2 \)) may be defined as follows:

\[
 f(x) = (1+x_m)^{n}; \quad x_m - m - \sum_{i=1}^{m-1} (x_i + x_{i+1}) / 2; \quad x \in [0, 1] \quad \forall \quad i = 1, 2, \ldots, m
\]

Unlike the new function #1, here \( x_m \) of the prior iteration indirectly enters into the posterior iteration. As a result, this function is extremely difficult to optimize.

V. **Some Details of the Particle Swarm Methods used Here**:: In this exercise we have used (modified) Repulsive Particle Swarm method. The Repulsive Particle Swarm method of optimization is a variant of the classical Particle Swarm method (see Wikipedia, [http://en.wikipedia.org/wiki/RPSO](http://en.wikipedia.org/wiki/RPSO)). It is particularly effective in finding out the global optimum in very complex search spaces (although it may be slower on certain types of optimization problems).

In the traditional RPS the future velocity, \( v_{i+1} \) of a particle at position with a recent velocity, \( v_i \), and the position of the particle are calculated by:

\[
 v_{i+1} = \omega v_i + \alpha r_1 (\hat{x}_i - x_i) + \omega \beta r_2 (\hat{x}_h - x_i) + \omega \gamma r_3 z \\
 x_{i+1} = x_i + v_{i+1}
\]

where,

- \( x \) is the position and \( v \) is the velocity of the individual particle. The subscripts \( i \) and \( i+1 \) stand for the recent and the next (future) iterations, respectively.
- \( r_1, r_2, r_3 \) are random numbers, \( \in [0,1] \)
- \( \omega \) is inertia weight, \( \in [0.01, 0.7] \)
- \( \hat{x} \) is the best position of a particle
- \( x_h \) is best position of a randomly chosen other particle from within the swarm
- \( z \) is a random velocity vector
- \( \alpha, \beta, \gamma \) are constants

Occasionally, when the process is caught in a local optimum, some chaotic perturbation in position as well as velocity of some particle(s) may be needed.

The traditional RPS gives little scope of local search to the particles. They are guided by their past experience and the communication received from the others in the swarm. We have modified the traditional RPS method by endowing stronger (wider) local search ability to each particle. Each particle flies in its local surrounding and searches for a better solution. The domain of its search is controlled by a new parameter \( n_{\text{step}} \). This local search has no preference to gradients in any direction and resembles closely to tunneling. This added exploration capability of the particles brings the RPS method closer to what we observe in real life.

Each particle learns from its ‘chosen’ inmates in the swarm. At the one extreme is to learn from the best performer in the entire swarm. This is how the particles in the original PS method learn. However, such learning is not natural. How can we expect the
individuals to know as to the best performer and interact with all others in the swarm? We believe in limited interaction and limited knowledge that any individual can possess and acquire. So, our particles do not know the ‘best’ in the swarm. Nevertheless, they interact with some chosen inmates that belong to the swarm. Now, the issue is: how does the particle choose its inmates? One of the possibilities is that it chooses the inmates closer (at lesser distance) to it. But, since our particle explores the locality by itself, it is likely that it would not benefit much from the inmates closer to it. Other relevant topologies are: (the celebrated) ring topology, ring topology hybridized with random topology, star topology, von Neumann topology, etc.

Now, let us visualize the possibilities of choosing (a predetermined number of) inmates randomly from among the members of the swarm. This is much closer to reality in the human world. When we are exposed to the mass media, we experience this. Alternatively, we may visualize our particles visiting a public place (e.g. railway platform, church, etc) where it (he) meets people coming from different places. Here, geographical distance of an individual from the others is not important. Important is how the experiences of others are communicated to us. There are large many sources of such information, each one being selective in what it broadcasts and each of us selective in what we attend to and, therefore, receive. This selectiveness at both ends transcends the geographical boundaries and each one of us is practically exposed to randomized information. Of course, two individuals may have a few common sources of information. We have used these arguments in the scheme of dissemination of others’ experiences to each individual particle. Presently, we have assumed that each particle chooses a pre-assigned number of inmates (randomly) from among the members of the swarm. However, this number may be randomized to lie between two pre-assigned limits.

VI. Some Details of the Differential Evolution Methods used Here: The differential Evolution method consists of three basic steps: (i) generation of (large enough) population with N individuals \( [x = (x_1, x_2, \ldots, x_m)] \) in the m-dimensional space, randomly distributed over the entire domain of the function in question and evaluation of the individuals of the so generated by finding \( f(x) \); (ii) replacement of this current population by a better fit new population, and (iii) repetition of this replacement until satisfactory results are obtained or certain criteria of termination are met.

The crux of the problem lays in replacement of the current population by a new population that is better fit. Here the meaning of ‘better’ is in the Pareto improvement sense. A set \( S_a \) is better than another set \( S_b \) iff: (i) no \( x_i \in S_a \) is inferior to the corresponding member \( x_i \in S_b \); and (ii) at least one member \( x_k \in S_a \) is better than the corresponding member \( x_k \in S_b \). Thus, every new population is an improvement over the earlier one. To accomplish this, the DE method generates a candidate individual to replace each current individual in the population. The candidate individual is obtained by a crossover of the current individual and three other randomly selected individuals from the current population. The crossover itself is probabilistic in nature. Further, if the candidate individual is better fit than the current individual, it takes the place of the current individual, else the current individual stays and passes into the next iteration.
Algorithmically stated, initially, a population of points (p in d-dimensional space) is generated and evaluated (i.e. f(p) is obtained) for their fitness. Then for each point (p_i) three different points (p_a, p_b and p_c) are randomly chosen from the population. A new point (p_z) is constructed from those three points by adding the weighted difference between two points (w(p_b-p_c)) to the third point (p_a). Then this new point (p_z) is subjected to a crossover with the current point (p_i) with a probability of crossover (c_r), yielding a candidate point, say p_u. This point, p_u, is evaluated and if found better than p_i then it replaces p_i else p_i remains. Thus we obtain a new vector in which all points are either better than or as good as the current points. This new vector is used for the next iteration. This process makes the differential evaluation scheme completely self-organizing.

The crossover scheme (called exponential crossover, as suggested by Kenneth Price in his personal letter to the author) is given below.

The mutant vector is \( \mathbf{v}_{i,g} = \mathbf{x}_{r1,g} + F*(\mathbf{x}_{r2,g} - \mathbf{x}_{r3,g}) \) and the target vector is \( \mathbf{x}_{i,g} \) and the trial vector is \( \mathbf{u}_{i,g} \). The indices r1, r2 and r3 are randomly but different from each other. Uj(0,1) is a uniformly distributed random number between 0 and 1 that is chosen anew for each parameter as needed.

Step 1: Randomly pick a parameter index \( j = j_{rand} \).

Step 2: The trial vector inherits the jth parameter (initially = \( j_{rand} \)) from the mutant vector, i.e., \( u_{j,i,g} = v_{j,i,g} \).

Step 3: Increment \( j \); if \( j = D \) then reset \( j = 0 \).

Step 4: If \( j = j_{rand} \) end crossover; else goto Step 5.

Step 5: If \( c_r \leq U_j(0,1) \), then goto Step 2; else goto Step 6.

Step 6: The trial vector inherits the jth parameter from the target vector, i.e., \( u_{j,i,g} = x_{j,i,g} \).

Step 7: Increment \( j \); if \( j = D \) then reset \( j = 0 \).

Step 8: If \( j = j_{rand} \) end crossover; else goto Step 6.

There could be other schemes (as many as 10 in number) of crossover, including no crossover (probabilistic replacement only, which works better in case of a few functions).

**VII. Specification of Adjustable Parameters:** The RPS as well as the DE method needs some parameters to be specified by the user. In case of the RPS we have fixed the parameters as follows:

- Population size, \( N=100 \); neighbour population, \( NN=50 \); steps for local search, \( NSTEP=11 \); Max no. of iterations permitted, \( ITRN=10000 \); chaotic perturbation allowed, \( NSIGMA=1 \); selection of neighbour : random, \( ITOP=3 \); \( A1=A2=0.5 \); \( A3=5.e-04 \); \( W=.5 \); \( SIGMA=1.e-03 \); \( EPSI=1.d-08 \). Meanings of these parameters are explained in the programs (appended).
In case of the DE, we have used two alternatives: first, the exponential crossover (ncross=1) as suggested by Price, and the second, only probabilistic replacement (but no crossover, ncross=0). We have fixed other parameters as: max number of iterations allowed, Iter = 10000, population size, N = 10 times of the dimension of the function or 100 whichever maximum; pcros = 0.9; scale factor, fact = 0.5, random number seed, iu = 1111 and all random numbers are uniformly distributed between -1000 and 1000; accuracy needed, eps =1.0e-08.

In case of either method, if x in f(x) violates the boundary then it is forcibly brought within the specified limits through replacing it by a random number lying in the given limits of the function concerned.

VIII. Findings and Discussion: Our findings are summarized in tables #1 through #3. The first table presents the results of the DE method when used with the exponential crossover scheme, while the table #2 presents the results of DE with no crossover (only probabilistic replacement). Table #3 presents the results of the RPS method.

A perusal of table #1 suggests that DE (with the exponential crossover scheme) mostly fails to find the optimum. Of course, it succeeds in case of some functions (perm#2, zero-sum) for very small dimension (m), but begins to falter as soon as the dimension is increased. In case of DCS function, it works well up to m (dimension) = 5.

**Table-1: Differential Evolution** (with exponential crossover: Ncoss–1, Cr = 0.9, F = 0.5)

<table>
<thead>
<tr>
<th>Function</th>
<th>M=2</th>
<th>M=4</th>
<th>M=5</th>
<th>M=10</th>
<th>M=20</th>
<th>M=30</th>
<th>M=50</th>
<th>M=100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bukin-6</td>
<td>0.01545</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Giunta</td>
<td>0.06447</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Perm #1</td>
<td>-</td>
<td>2.45035</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Power-Sum</td>
<td>-</td>
<td>0.00752</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Kowalik</td>
<td>-</td>
<td>0.36724</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Hougen</td>
<td>-</td>
<td>-</td>
<td>0.34228</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>New Fn #2</td>
<td>2.05349</td>
<td>2.03479</td>
<td>2.12106</td>
<td>2.57714</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fletcher</td>
<td>0.02288</td>
<td>0.72118</td>
<td>3.13058</td>
<td>12027.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perm #2</td>
<td>0</td>
<td>0.00549</td>
<td>0.07510</td>
<td>0.01005</td>
<td>0.15991</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yao-Liu#7</td>
<td>0.02924</td>
<td>0.02244</td>
<td>0.03338</td>
<td>0.04643</td>
<td>0.19073</td>
<td>0.27375</td>
<td>0.51610</td>
<td></td>
</tr>
<tr>
<td>Zero-Sum</td>
<td>0</td>
<td>1.73194</td>
<td>1.70313</td>
<td>1.26159</td>
<td>1.13372</td>
<td>1.74603</td>
<td>1.70230</td>
<td>1.22196</td>
</tr>
<tr>
<td>DCS</td>
<td>-1</td>
<td>-0.99994</td>
<td>-0.99994</td>
<td>-0.84334</td>
<td>-0.84334</td>
<td>0.40992</td>
<td>0.40992</td>
<td>1.50650</td>
</tr>
</tbody>
</table>

**Table-2: Differential Evolution** (without crossover: Ncoss = 0, Cr = 0.9, F = 0.5)

<table>
<thead>
<tr>
<th>Function</th>
<th>M=2</th>
<th>M=4</th>
<th>M=5</th>
<th>M=10</th>
<th>M=20</th>
<th>M=30</th>
<th>M=50</th>
<th>M=100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bukin-6</td>
<td>0.02132</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Giunta</td>
<td>0.06476</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Perm #1</td>
<td>0.00000</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Power-Sum</td>
<td>0.00000</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Kowalik</td>
<td>0.30745</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Hougen</td>
<td>0.29890</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>New Fn #2</td>
<td>2.03031</td>
<td>2.11842</td>
<td>2.11040</td>
<td>2.57009</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fletcher</td>
<td>0.00862</td>
<td>2.84237</td>
<td>7.09278</td>
<td>261.377</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perm #2</td>
<td>0</td>
<td>0.00009</td>
<td>0.00011</td>
<td>0.07219</td>
<td>13.355</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yao-Liu#7</td>
<td>0.02984</td>
<td>0.00668</td>
<td>0.02467</td>
<td>0.02396</td>
<td>0.21920</td>
<td>0.22435</td>
<td>0.31691</td>
<td></td>
</tr>
<tr>
<td>Zero-Sum</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.32197</td>
<td>1.30459</td>
<td></td>
</tr>
<tr>
<td>DCS</td>
<td>-1</td>
<td>-1</td>
<td>-0.84334</td>
<td>-0.84334</td>
<td>-0.84334</td>
<td>-0.37336</td>
<td>-0.37302</td>
<td>-0.37302</td>
</tr>
</tbody>
</table>
When we use no crossover (only probabilistic replacement) we obtain better results in case of several of the functions under study. Thus, overall, table #2 presents better results than what table #1 does. In case of Perm#1, Perm#2, Zero-sum, Kowalik, Hougen and Power-sum functions the advantage is clear.

Whether crossover or no crossover, DE falters when the optimand function has some element of randomness. This is indicated by the functions: Yao-Liu#7, Fletcher-Powell, and “New function#2”. DE has no problems in optimizing the “New function #1”. But the “New function #2” proves to be a hard nut. However, RPS performs much better for such stochastic functions. When the Fletcher-Powell function is optimized with non-stochastic c vector, DE works fine. But as soon as c is stochastic, it becomes unstable. Thus, it may be observed that an introduction of stochasticity into the decision variables (or simply added to the function as in Yao-Liu#7) interferes with the fundamentals of DE, which works through attainment of better and better (in the sense of Pareto improvement) population at each successive iteration.

<table>
<thead>
<tr>
<th>Function</th>
<th>M=2</th>
<th>M=4</th>
<th>M=5</th>
<th>M=10</th>
<th>M=20</th>
<th>M=30</th>
<th>M=50</th>
<th>M=100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bukin-6</td>
<td>0.75649</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Giunta</td>
<td>0.06447</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Perm #1</td>
<td>-</td>
<td>0.00000</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Power-Sum</td>
<td>-</td>
<td>0.00000</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Kowalik</td>
<td>-</td>
<td>0.30749</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Hougen</td>
<td>-</td>
<td>-</td>
<td>0.42753</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>New Fn #2</td>
<td>2.000000</td>
<td>2.00021</td>
<td>2.00115</td>
<td>2.00644</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fletcher</td>
<td>0.00003</td>
<td>0.09582</td>
<td>0.92413</td>
<td>40.70962</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perm #2</td>
<td>0</td>
<td>0.00006</td>
<td>0.00011</td>
<td>0.00008</td>
<td>1.17420</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yao-Liu#7</td>
<td>0.000000</td>
<td>0.00002</td>
<td>0.00000</td>
<td>0.00003</td>
<td>0.00007</td>
<td>0.00050</td>
<td>0.00060</td>
<td></td>
</tr>
<tr>
<td>Zero-Sum</td>
<td>1.12312</td>
<td>1.14119</td>
<td>1.19259</td>
<td>1.21672</td>
<td>1.44780</td>
<td>1.02749</td>
<td>1.18303</td>
<td>1.56457</td>
</tr>
<tr>
<td>DCS</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-0.84334</td>
<td>-0.84334</td>
<td>-0.37336</td>
<td>-0.37336</td>
<td>11.68769</td>
</tr>
</tbody>
</table>

IX. Conclusion: Ours is a very small sample of test functions that we have used to compare DE (with and without crossover) and RPS methods. So the limitations of our conclusions are obvious. However, if any indication is obtained, those are: (1) for different types of problems, different schemes of crossover (including none) may be suitable or unsuitable, (2) Stochasticity entering into the optimand function may make DE unstable, but RPS may function well.
Bibliography

DEHARD.f  1/18

1:  C MAIN PROGRAM : PROVIDES TO USE REPULSIVE PARTICLE SWARM METHOD
2:  C (SUBROUTINE RPS) AND DIFFERENTIAL EVOLUTION METHOD (DE)
3:  C ---------------------------------------------------------------
4:  C Adjust the parameters suitably in subroutines DE and RPS
5:  C When the program asks for parameters, feed them suitably
6:  C ---------------------------------------------------------------
7:  C PROGRAM DERPS
8:  C IMPLICIT DOUBLE PRECISION (A-H, O-Z)
9:  C COMMON /KF/KF,NFCALL / FUNCTION CODE AND NO. OF FUNCTION CALLS
10:  C CHARACTER *30 METHOD(2)
11:  C CHARACTER *I PROCEED
12:  C DIMENSION X(2,100),KKF(2),MM(2),FMINN(2)
13:  C M IS THE DIMENSION OF THE PROBLEM, KF IS TEST FUNCTION CODE AND
14:  C FMIN IS THE MIN VALUE OF F(X) OBTAINED FROM DE OR RPS
15:  C ---------------------------------------------------------------
16:  WRITE(*,*)'Adjust the parameters suitably in subroutines DE & RPS'
17:  WRITE(*,*)'WARNING '                                 '           '
18:  METHOD(1)=' : DIFFERENTIAL EVALUATION'
19:  METHOD(2)=' : REPULSIVE PARTICLE SWARM'
20:  DO I=1,2
21:  IF(I.EQ.1) THEN
22:  WRITE(*,*)'DIFFERENTIAL EVOLUTION PROGRAM '           '
23:  WRITE(*,*)'TO PROCEED TYPE ANY CHARACTER AND STRIKE ENTER'
24:  READ(*,*) PROCEED
25:  CALL DE(M,X,FMINDE) ! CALLS DE AND RETURNS OPTIMAL X AND FMIN
26:  FMIN=FMINDE
27:  ELSE
28:  WRITE(*,*) '
29:  WRITE(*,*) '           '
30:  WRITE(*,*)'DIFFERENTIAL EVOLUTION PROGRAM '           '
31:  WRITE(*,*)'TO PROCEED TYPE ANY CHARACTER AND STRIKE ENTER'
32:  READ(*,*) PROCEED
33:  CALL RPS(M,X,FMINRPS) ! CALLS RPS AND RETURNS OPTIMAL X AND FMIN
34:  FMIN=FMINRPS
35:  ENDIF
36:  DO J=1,M
37:  XX(I,J)=X(J)
38:  ENDDO
39:  KKF(I)=KF
40:  MM(I)=M
41:  FMINN(I)=FMIN
42:  ENDDO
43:  WRITE(*,*)'FINAL RESULTS '                           '
44:  DO I=1,2
45:  WRITE(*,*)'FUNCTION CODE=',KKF(I),', FMIN=',FMINN(I),', DIM=',MM(I)
46:  WRITE(*,*)'OPTIMAL DECISION VARIABLES : ',METHOD(I)
47:  WRITE(*,*)'(XX(I,J),J=1,M)
48:  WRITE(*,*)'-------------------------------------------------------------------------------------'
49:  ENDDO
50:  WRITE(*,*)'PROGRAM ENDED'
51:  END
52:  C SUBROUTINE DE(M,A,FBEST)
53:  C PROGRAM: "DIFFERENTIAL EVOLUTION ALGORITHM" OF GLOBAL OPTIMIZATION
54:  C THIS METHOD WAS PROPOSED BY R. STORN AND K. PRICE IN 1995. REF --
55:  C "DIFFERENTIAL EVOLUTION - A SIMPLE AND EFFICIENT ADAPTIVE SCHEME
56:  C "  FOR GLOBAL OPTIMIZATION OVER CONTINUOUS SPACES" : TECHNICAL REPORT
58:  C PROGRAM BY SK MISHRA, DEPT. OF ECONOMICS, NEHU, SHILLONG (INDIA)
59:  C ---------------------------------------------------------------
60:  C PROGRAM EVOLDIF
61:  C IMPLICIT DOUBLE PRECISION (A-H, O-Z) ! TYPE DECLARATION
62:  PARAMETER(NMAX=1000,MMAX=100) ! MAXIMUM DIMENSION PARAMETERS
PARAMETER(NCROSS=1) ! CROSS-OVER SCHEME (NCROSS <=0 OR =>1)
PARAMETER(IPRINT=500,EPS=1.D-08) ! FOR WATCHING INTERMEDIATE RESULTS
ITPRINT PRINTS THE INTERMEDIATE RESULTS AFTER EACH IPRINT ITERATION AND 
EPS DETERMINES ACCURACY FOR TERMINATION. IF EPS= 0, ALL ITERATIONS 
WOULDN'T BE UNDERGONE EVEN IF NO IMPROVEMENT IN RESULTS IS THERE.
ULTIMATELY "DID NOT CONVERGE" IS REPORTED.
COMMON /RNDM/IU,IV ! RANDOM NUMBER GENERATION (IU = 4-DIGIT SEED)
COMMON /KF/KF,NFCALL ! FUNCTION CODE AND NO. OF FUNCTION CALLS
CHARACTER *70 FTIT ! TITLE OF THE FUNCTION
THE PROGRAM REQUIRES INPUTS FROM THE USER ON THE FOLLOWING ------
FUNCTION CODE (KF), (2) NO. OF VARIABLES IN THE FUNCTION (M);
N=POPULATION SIZE (SUGGESTED 10 TIMES OF NO. OF VARIABLES, M,
FOR SMALLER PROBLEMS N=100 WORKS VERY WELL);
PCROS = PROB. OF CROSS-OVER (SUGGESTED : ABOUT 0.85 TO .99);
FACT = SCALE (SUGGESTED 0.5 TO .95 OR SO);
ITER = MAXIMUM NUMBER OF ITERATIONS PERMITTED (5000 OR MORE)
RANDOM NUMBER SEED (4 DIGITS INTEGER)
DIMENSION X(NMAX,MMAX),Y(NMAX,MMAX),A(MMAX),FV(NMAX)
DIMENSION IR(3)
------- SELECT THE FUNCTION TO MINIMIZE AND ITS DIMENSION -------
CALL FSELECT(KF,M,FTIT)
SPECIFY OTHER PARAMETERS -------------------------------------------
WRITE(*,*)'POPULATION SIZE [N] AND NO. OF ITERATIONS [ITER] ?'
WRITE(*,*)'SUGGESTED: MAX(100, 10.M); ITER 10000 OR SO'
READ(*,*) N,ITER
WRITE(*,*)'CROSSOVER PROBABILITY [PCROS] AND SCALE [FACT] ?'
WRITE(*,*)'SUGGESTED: PCROS ABOUT 0.9; FACT=.5 OR LARGER BUT <=1'
READ(*,*) PCROS,FACT
WRITE(*,*)'RANDOM NUMBER SEED ?'
WRITE(*,*)'A FOUR-DIGIT POSITIVE ODD INTEGER, SAY, 1171'
READ(*,*) IU
NFCALL=0 ! INITIALIZE COUNTER FOR FUNCTION CALLS
GBEST=1.D30 ! TO BE USED FOR TERMINATION CRITERION
C INITIALIZATION : GENERATE X(N,M) RANDOMLY
DO I=1,N
DO J=1,M
CALL RANDOM(RAND)
X(I,J)=(RAND-.5000)*2000
C RANDOM NUMBERS BETWEEN -RANGE AND +RANGE (BOTH EXCLUSIVE)
ENDDO
DO 100 ITR=1,ITER ! ITERATION BEGINS
C EVALUATE ALL X FOR THE GIVEN FUNCTION
DO I=1,N
DO J=1,M
A(J)=X(I,J)
ENDDO
CALL FUNC(A,M,F)
C STORE FUNCTION VALUES IN FV VECTOR
FV(I)=F
ENDDO
C FIND THE FITTEST (BEST) INDIVIDUAL AT THIS ITERATION
IF(FV(IB).LT.FBEST) THEN
FBEST=FV(IB)
KB=IB
PERSONAL LETTER TO THE AUTHOR (DATED SEPTEMBER 29, 2006)

CROSSOVER SCHEME (EXPONENTIAL) SUGGESTED BY KENNETH PRICE IN HIS

NO CROSS OVER, ONLY REPLACEMENT THAT IS PROBABILISTIC

FROM EACH OTHER ARE IR(1), IR(2) AND IR(3)

THREE RANDOMLY CHOSEN INDIVIDUALS DIFFERENT FROM I AND DIFFERENT

BEST FITNESS VALUE = FBEST : INDIVIDUAL X(KB)

GENERATE OFFSPRINGS

DO I=1,N  ! I LOOP BEGINS

CALL RANDOM(RAND)

IRJ=INT(RAND*M)+1

CHECK THAT THESE THREE INDIVIDUALS ARE DISTINCT AND OTHER THAN I

IF(IR.I.EQ.1.AND.IRJ.NE.I) THEN

IR(IRI)=IRJ

ENDIF

IF(IR.I.EQ.2.AND.IRJ.NE.I.AND.IRJ.NE.IR(1)) THEN

IR(IRI)=IRJ

ENDIF

IF(IR.I.EQ.3.AND.IRJ.NE.I.AND.IRJ.NE.IR(1).AND.IRJ.NE.IR(2)) THEN

IR(IRI)=IRJ

ENDIF

ENDDO  ! IRI LOOP ENDS

DO IX=1,3

IF(IR(IX).LE.0) THEN

GOTO 20  ! IF NOT THEN REGENERATE

ENDIF

ENDDO

THREE RANDOMLY CHOSEN INDIVIDUALS DIFFERENT FROM I AND DIFFERENT

FROM EACH OTHER ARE IR(1), IR(2) AND IR(3)

NO CROSS OVER, ONLY REPLACEMENT THAT IS PROBABILISTIC

IF(NCROSS.LE.0) THEN

DO J=1,M

CALL RANDOM(RAND)

IF(RAND.LE.PCROS) THEN  ! REPLACE IF RAND < PCROS

A(J)=X(IR(1),J)+X(IR(2),J)-X(IR(3),J)*FACT  ! CANDIDATE CHILD

ENDIF

ENDDO  ! J LOOP ENDS

ENDIF

CROSSOVER SCHEME (EXPONENTIAL) SUGGESTED BY KENNETH PRICE IN HIS

PERSONAL LETTER TO THE AUTHOR (DATED SEPTEMBER 29, 2006)

IF(NCROSS.GE.1) THEN

CALL RANDOM(RAND)

1 JR=INT(RAND*M)+1

J=JR

2 A(J)=X(IR(1),J)+FACT*(X(IR(2),J)-X(IR(3),J))

3 J=J+1

4 IF(J.GT.M) J=1

5 IF(J.EQ.JR) GOTO 10

6 A(J)=X(I,J)

7 J=J+1

8 IF(J.GT.M) J=1

9 IF(J.EQ.JR) GOTO 10

10 CONTINUE

ENDIF
CALL FUNC(A,M,F)  ! EVALUATE THE OFFSPRING
IF(F.LT.FV(I)) THEN  ! IF BETTER, REPLACE PARENTS BY THE CHILD
FV(I)=F
DO J=1,M
Y(I,J)=A(J)
ENDO
ENDIF
ENDO  ! I LOOP ENDS
DO I=1,N
DO J=1,M
X(I,J)=Y(I,J)  ! NEW GENERATION IS A MIX OF BETTER PARENTS AND
                        BETTER CHILDREN
ENDDO
ENDDO
IPCOUNT=IPCOUNT+1
IF(IPCOUNT.EQ.IPRINT) THEN
DO J=1,M
A(J)=X(KB,J)
ENDDO
WRITE(('*','(A,2X,F7.7)'),(M=1,M))' FBEST UPTO NOW = ',FBEST
WRITE('*','(A,2X,F7.7)',(M=1,M))' TOTAL NUMBER OF FUNCTION CALLS = ',NFCALL
IF(DABS(FBEST-FBEST).LT.EPS) THEN
WRITE('*','(A,2X,F7.7)',(M=1,M))' COMPUTATION OVER'
RETURN
ELSE
FBEST=FBEST
ENDIF
IPCOUNT=0
ENDIF
! ITERATION ENDS : GO FOR NEXT ITERATION, IF APPLICABLE
WRITE('*','(A,2X,F7.7)',(M=1,M))' DID NOT CONVERGE. REDUCE EPS OR RAISE ITER OR DO BOTH'
WRITE('*','(A,2X,F7.7)',(M=1,M))' INCREASE N, PCROS, OR SCALE FACTOR (FACT)'
RETURN
END
! RANDOM NUMBER GENERATOR (UNIFORM BETWEEN 0 AND 1 - BOTH EXCLUSIVE)
SUBROUTINE RANDOM(RAND1)
DOUBLE PRECISION RAND1
COMMON /RNDM/IU,IV
INTEGER IU,IV
RAND=REAL(RAND1)
IV=IU*65539
IF(IV.LT.0) THEN
IV=IV+2147483647+1
ENDIF
RAND=IV
IU=IV
RAND=RAND*0.4656613E-09
RAND1= (RAND)
RETURN
END

SUBROUTINE FSELECT(KF,M,FTIT)
THE PROGRAM REQUIRES INPUTS FROM THE USER ON THE FOLLOWING ------
FUNCTION CODE (KF), (2) NO. OF VARIABLES IN THE FUNCTION (M);

CHARACTER *70 TIT(100),FTIT
WRITE('*','(A,2X,F7.7)')' RANDOM GENERATOR (UNIFORM BETWEEN 0 AND 1 - BOTH EXCLUSIVE)'
DATA TIT(1)/*KF=1 NEW ZERO-SUM FUNCTION (N#7) M-VARIABLES M=?*/
DATA TIT(2)/*KF=2 PERM FUNCTION #1 (SET BETA) 4-VARIABLES M=4*/
DATA TIT(3)/*KF=3 PERM FUNCTION #2 (SET BETA) M-VARIABLES M=4*/
DATA TIT(4)/*KF=4 POWER-SUM FUNCTION 4-VARIABLES M=4*/
DATA TIT(5)/*KF=5 BUKIN 6TH FUNCTION 2-VARIABLES M=2*/
DATA TIT(6)/*KF=6 DEF. CORRUG SPRING FUNCTION M-VARIABLES M=2*/
DATA TIT(7)/*KF=7 YAO-LIU FUNCTION#7 M-VARIABLES M=7*/
DATA TIT(8) /'KF=8 HOUGEN FUNCTION : 5-VARIABLES M=5'/
DATA TIT(9) /'KF=9 GIUNITA FUNCTION : 2-VARIABLES M=2'/
DATA TIT(10) /'KF=10 KOWALIK FUNCTION : 4-VARIABLES M=4'/
DATA TIT(11) /'KF=11 FLETCHER-POWELL FUNCTION : M-VARIABLES M=?'/
DATA TIT(12) /'KF=12 NEW NFUNCT1 FUNCTION : M-VARIABLES M=?'/
DATA TIT(13) /'KF=13 NEW NFUNCT2 FUNCTION : M-VARIABLES M=?'/
DATA TIT(14) /'KF=14 WOOD FUNCTION : 4-VARIABLES M=4'/
DATA TIT(15) /'KF=15 FENTON-EASON FUNCTION : 2-VARIABLES M=2'/
DATA TIT(16) /'KF=16 TYPICAL NON-LINEAR FUNCTION:M-VARIABLES M=?'/
DATA TIT(17) /'KF=17 WILD FUNCTION : M-VARIABLES M=?'/

C
DO I=1,17
WRITE(*,*)TIT(I)
ENDDO
WRITE(*,*)'FUNCTION CODE [KF] AND NO. OF VARIABLES [M] ?'
READ(*,*)KF,M
FTIT=TIT(KF) ! STORE THE NAME OF THE CHOSEN FUNCTION IN FTIT
RETURN
END

C
======================================== REPULSIVE PARTICLE SWARM ====================
SUBROUTINE RPS(M,BST,FMINIM)
C PROGRAM TO FIND GLOBAL MINIMUM BY REPULSIVE PARTICLE SWARM METHOD
C WRITTEN BY SK MISHRA, DEPT. OF ECONOMICS, NEHU, SHILLONG (INDIA)
C
PARAMETER(N=100,NN=50,MX=100,NSTEP=11,ITRN=100000,NSIGMA=1,ITOP=3)
PARAMETER(NP=500) ! ECHOS RESULTS AT EVERY 500 TH ITERATION
C PARAMETER(N=50,NN=25,MX=100,NSTEP=9,ITRN=10000,NSIGMA=1,ITOP=3)
P PARAMETER (N=100,NN=15,MX=100,NSTEP=9,ITRN=10000,NSIGMA=1,ITOP=3)
C IN CERTAIN CASES THE ONE OR THE OTHER SPECIFICATION WORKS BETTER
C DIFFERENT SPECIFICATIONS OF PARAMETERS MAY SUIT DIFFERENT TYPES
C OF FUNCTIONS OR DIMENSIONS - ONE HAS TO DO SOME TRIAL AND ERROR
C
N = POPULATION SIZE. IN MOST OF THE CASES N=30 IS OK. ITS VALUE
MAY BE INCREASED TO 50 OR 100 TOO. THE PARAMETER NN IS THE SIZE OF
RANDOMLY CHOSEN NEighbours. 15 TO 25 (BUT SUFFICIENTLY LESS THAN
N) IS A GOOD CHOICE. MX IS THE MAXIMAL SIZE OF DECISION VARIABLES.
IN F(X1, X2,...,XM) M SHOULD BE LESS THAN OR EQUAL TO MX. ITRN IS
THE NO. OF ITERATIONS. IT MAY DEPEND ON THE PROBLEM. 200 (AT LEAST)
TO 500 ITERATIONS MAY BE GOOD ENOUGH. BUT FOR FUNCTIONS LIKE
ROSENBROCK OR GRIEWANK OF LARGE SIZE (SAY M=30) IT IS NEEDED THAT
ITRN IS LARGE, SAY 5000 OR EVEN 10000.
SIGMA INTRODUCES PERTURBATION & HELPS THE SEARCH JUMP OUT OF LOCAL
OPTIMA. FOR EXAMPLE : RASTRIGIN FUNCTION OF DIMENSION 30 OR LARGER
NSTEP DOES LOCAL SEARCH BY TUNNELLING AND WORKS WELL BETWEEN 5 AND
15, WHICH IS MUCH ON THE HIGHER SIDE.
ITOP <=1 (RING); ITOP=2 (RING AND RANDOM); ITOP=>3 (RANDOM)
NSIGMA=0 (NO CHAOTIC PERTURBATION);NSIGMA=1 (CHAOTIC PERTURBATION)
NOTE THAT NSIGMA=1 NEED NOT ALWAYS WORK BETTER (OR WORSE)
SUBROUTINE FUNC( ) DEFINES OR CALLS THE FUNCTION TO BE OPTIMIZED.
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
COMMON /RNDM/IV
COMMON /KF,KF,NFCALL
INTEGER IU,IV
CHARACTER *70 FTIT
DIMENSION X(N, MX),V(N, MX),A(MX),VI(MX)
DIMENSION XX(N, MX),F(N),V1(MX),V2(MX),V3(MX),V4(MX),BST(MX)
C A1 A2 AND A3 ARE CONSTANTS AND W IS THE INERTIA WEIGHT.
C OCCASIONALLY, TINKERING WITH THESE VALUES, ESPECIALLY A3, MAY BE
C NEEDED.
DATA A1,A2,A3,W,SIGMA,EPSI / .5D0, .5D0, .5D-04, .5D0, 1.D-03, 1.D-08/
C CALL SUBROUTINE FOR CHOOSING FUNCTION (KF) AND ITS DIMENSION (M)
CALL FSELECT(KF,M,FTIT)
C
GGBEST=1.D30 ! TO BE USED FOR TERMINATION CRITERION
LCOUNT = 0
NFCALL = 0
WRITE(*,*) '4-DIGITS SEED FOR RANDOM NUMBER GENERATION'
WRITE(*,*) 'A FOUR-DIGIT POSITIVE ODD INTEGER, SAY, 1171'
READ(*,*) IU
FMN = -1.0E30
C GENERATE N-SIZE POPULATION OF M-TUPLE PARAMETERS X(I,J) RANDOMLY
DO I=1,N
   DO J=1,M
      CALL RANDOM(RAND)
      X(I,J) = (RAND-0.5D00)*2000
   ENDDO
F(I) = 1.0D30
ENDDO
C INITIALISE VELOCITIES V(I) FOR EACH INDIVIDUAL IN THE POPULATION
DO I=1,N
   DO J=1,M
      CALL RANDOM(RAND)
      V(I,J) = (RAND-0.5D+00)
   ENDDO
ENDDO
DO 100 ITER=1,ITRN
C LET EACH INDIVIDUAL SEARCH FOR THE BEST IN ITS NEIGHBOURHOOD
DO I=1,N
   DO J=1,M
      A(J) = X(I,J)
      VI(J) = V(I,J)
   ENDDO
   CALL LSRCH(A,M,VI,NSTEP,FI)
   IF(FI.LT.F(I)) THEN
      F(I) = FI
      DO IN=1,M
         BST(IN) = A(IN)
      ENDDO
   ENDIF
ENDDO
C F(I) CONTAINS THE LOCAL BEST VALUE OF FUNCTION FOR ITH INDIVIDUAL
C XX(I,J) IS THE M-TUPLE VALUE OF X ASSOCIATED WITH LOCAL BEST F(I)
DO J=1,M
   XX(I,J) = A(J)
ENDDO
C NOW LET EVERY INDIVIDUAL RANDOMLY CONSULT NN(<N) COLLEAGUES AND
C FIND THE BEST AMONG THEM
C ------------------------------------------------------------------
IF(ITOP.GE.3) THEN
C RANDOM TOPOLOGY ******************************************************
C CHOOSE NN COLLEAGUES RANDOMLY AND FIND THE BEST AMONG THEM
BEST = 1.0D30
DO II=1,NN
   CALL RANDOM(RAND)
   NF = INT(RAND*NN)+1
   IF(BEST.GT.F(NF)) THEN
      BEST = F(NF)
      NFBEST = NF
   ENDIF
ENDDO
ENDIF
C----------------------------------------------------------------------
IF(ITOP.EQ.2) THEN
C RING + RANDOM TOPOLOGY *********************************************
C REQUIRES THAT THE SUBROUTINE NEIGHBOR IS TURNED ALIVE
BEST = 1.0D30
CALL NEIGHBOR(I,N,II,I3)
DO II=1,NN
  IF(II.EQ.1) NF=I1
  IF(II.EQ.2) NF=I
  IF(II.EQ.3) NF=I3
  IF(II.GT.3) THEN
    CALL RANDOM(RAND)
    NF=INT(RAND*N)+1
  ENDIF
  IF(BEST.GT.F(NF)) THEN
    BEST=F(NF)
    NFBEST=NF
  ENDIF
ENDIF
ENDDO

C---------------------------------------------------------------

IF(ITOP.LE.1) THEN
C RING TOPOLOGY ****************************************************
C REQUIRES THAT THE SUBROUTINE NEIGHBOR IS TURNED ALIVE
C
BEST=1.0D30
CALL NEIGHBOR(I,N,I1,I3)
DO II=1,3
  IF (II.NE.I) THEN
    II=II+1
  ENDIF
ENDDO
C---------------------------------------------------------------

C IN THE LIGHT OF HIS OWN AND HIS BEST COLLEAGUES EXPERIENCE, THE
C INDIVIDUAL I WILL MODIFY HIS MOVE AS PER THE FOLLOWING CRITERION
C FIRST, ADJUSTMENT BASED ON ONES OWN EXPERIENCE
C AND OWN BEST EXPERIENCE IN THE PAST (XX(I))
DO J=1,M
  CALL RANDOM(RAND)
  V1(J)=A1*RAND*(XX(I,J)-X(I,J))
  IF(RAND.LT.F(I)) THEN
    V2(J)=A2*W*RAND*(XX(NFBEST,J)-X(I,J))
  ENDIF
  CALL RANDOM(RAND)
  V3(J)=A3*RAND*W*RND1
  V3(J)=A3*RAND*W
  CALL RANDOM(RAND)
  V4(J)=W*V(I,J)
  IF(NFBEST.LT.F(I)) THEN
    V2(J)=A2*W*RAND*(XX(NFBEST,J)-X(I,J))
  ENDIF
C---------------------------------------------------------------

C THEN SOME RANDOMNESS AND A CONSTANT A3 CLOSE TO BUT LESS THAN UNITY
CALL RANDOM(RAND)
RND1=RAND
CALL RANDOM(RAND)
V3(J)=A3*RAND*W*RND1
V3(J)=A3*RAND*W
C THEN ON PAST VELOCITY WITH INERTIA WEIGHT W
V4(J)=W*V(I,J)
C FINALLY A SUM OF THEM
V(I,J)= V1(J)+V2(J)+V3(J)+V4(J)
ENDDO
ENDDO

C---------------------------------------------------------------

IF(NSIGMA.EQ.1) THEN
CALL RANDOM(RAND) ! FOR CHAOTIC PERTURBATION
IF(DABS(RAND-.5D00).LT.SIGMA) RANDS=RAND-.5D00
SIGMA CONDITIONED RANDS INTRODUCES CHAOTIC ELEMENT IN TO LOCATION
IN SOME CASES THIS PERTURBATION HAS WORKED VERY EFFECTIVELY WITH
PARAMETER (N=100,NN=15,MX=100,NSTEP=9,ITRN=100000,NSIGMA=1,ITOP=2)
ENDIF
C     -----------------------------------------------------------------
X(I,J)=X(I,J)+V(I,J)*(1.D00+RANDS)
ENDDO
ENDIF
DO I=1,N
  IF(F(I).LT.FMIN) THEN
    FMIN=F(I)
    II=I
  ENDDO
ENDIF
ENDIF
LCOUNT=LCOUNT+1
100 CONTINUE
WRITE(*,'(A)',FMT=10000) 'COMPUTATION OVER:',FTIT
FMINIM=FMIN
RETURN
END
C     -----------------------------------------------------------------
SUBROUTINE LSRCH(A,M,VI,NSTEP,FI)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
COMMON /KFF,KF,NFCALL
COMMON /RNDM/IU,IV
INTEGER IU,IV
DIMENSION A(*),B(100),VI(*)
AMN=1.0D30
DO J=1,NSTEP
  DO JJ=1,M
    B(JJ)=A(JJ)+(J-(NSTEP/2)-1)*VI(JJ)
  ENDDO
  CALL FUNC(B,M,FI)
  IF(FI.LT.AMN) THEN
    AMN=FI
  ENDIF
  DO JJ=1,M
    A(JJ)=B(JJ)
  ENDDO
ENDIF
ENDDO
FI=AMN
RETURN
END
C     -----------------------------------------------------------------
SUBROUTINE NEIGHBOR(I,N,J,K)
  IF(I-1.GE.1 .AND. I.LT.N) THEN
    J=I-1
    K=I+1
  ELSE
    IF(I.GT.1) THEN
      J=I+1
      K=I-1
    ELSE
      J=1
      K=1
    ENDIF
ENDIF

537:     IF(I-1.LT.1) THEN
538:        J=J-1
539:        K=K+1
540:     ENDIF
541:     IF(I.EQ.N) THEN
542:        J=I-1
543:        K=1
544:     ENDIF
545:     ENDIF
546:     RETURN
547:     END
548: C     -----------------------------------------------------------------
549: SUBROUTINE FUNC(X,M,F)
550: C     TEST FUNCTIONS FOR GLOBAL OPTIMIZATION PROGRAM
551: IMPLICIT DOUBLE PRECISION (A-H,O-Z)
552: COMMON /RNDM/IU,IV
553: COMMON /KFF,KF,NFCALL
554: INTEGER IU,IV
555: DIMENSION X(*)
556: NFCALL=NFCALL+1 ! INCREMENT TO NUMBER OF FUNCTION CALLS
557: C     KF IS THE CODE OF THE TEST FUNCTION
558: C     -----------------------------------------------------------------
559: IF(KF.EQ.1) THEN
560: C     ZERO SUM FUNCTION : MIN = 0 AT SUM(X(I))=0
561:    CALL ZEROSUM(M,F,X)
562:    return
563: ENDIF
564: C     -----------------------------------------------------------------
565: IF(KF.EQ.2) THEN
566: C     PERM FUNCTION #1 MIN = 0 AT (1, 2, 3, 4)
567: C     BETA => 0. CHANGE IF NEEDED. SMALLER BETA RAISES DIFFICULTY
568: C     FOR BETA=0, EVERY PERMUTED SOLUTION IS A GLOBAL MINIMUM
569:    CALL PERM1(M,F,X)
570:    return
571: ENDIF
572: C     -----------------------------------------------------------------
573: IF(KF.EQ.3) THEN
574: C     PERM FUNCTION #2 MIN = 0 AT (1/1, 1/2, 1/3, 1/4,..., 1/M)
575: C     BETA => 0. CHANGE IF NEEDED. SMALLER BETA RAISES DIFFICULTY
576: C     FOR BETA=0, EVERY PERMUTED SOLUTION IS A GLOBAL MINIMUM
577:    CALL PERM2(M,F,X)
578:    return
579: ENDIF
580: C     -----------------------------------------------------------------
581: IF(KF.EQ.4) THEN
582: C     POWER SUM FUNCTION; MIN = 0 AT PERM(1,2,2,3) FOR B=(8,18,44,114)
583: C     0 <= X <=4
584:    CALL POWERSUM(M,F,X)
585:    return
586: ENDIF
587: C     -----------------------------------------------------------------
588: IF(KF.EQ.5) THEN
589: C     BUKIN'S 6TH FUNCTION MIN = 0 FOR (-10, 1)
590: C     -15. LE. X(1) .LE. -5 AND -3 .LE. X(2) .LE. 3
591:    CALL BUKING(M,F,X)
592:    return
593: ENDIF
594: C     -----------------------------------------------------------------
595: IF(KF.EQ.6) THEN
596: C     DEFLECTED CORRUGATED SPRING FUNCTION
597: C     MIN VALUE = -1 AT (5, 5, ..., 5) FOR ANY K AND ALPHA=5; M VARIABLE
598:    CALL DCS(M,F,X)
599:    RETURN
600: ENDIF
601: C     -----------------------------------------------------------------
602: IF(KF.EQ.7) THEN
603: C     M =>2
CALL FUNCT7(M,F,X)
RETURN
ENDIF

IF(KF.EQ.8) THEN
HOUGEN FUNCTION 5 VARIABLES : M = 3
CALL HOUGEN(X,M,F)
RETURN
ENDIF

IF(KF.EQ.9) THEN
GIUNTA FUNCTION 2 VARIABLES : M = 2
CALL GIUNTA(M,X,F)
RETURN
ENDIF

C     -----------------------------------------------------------------
IF(KF.EQ.10) THEN
FOR X(1)=0.1928D00; X(2)=0.1905D00; X(3)=0.1230D00; X(4)=0.1356D00
FMIN=0.3075D-03  ; -4 <= X(I) <= 4.
CALL KONALIK(M,X,F)
RETURN
ENDIF

C     -----------------------------------------------------------------
IF(KF.EQ.11) THEN
FLETCHER & POWELL FUNCTION : F MIN f(0,0,...,0 )=0
CALL FLETCHER(M,X,F)
RETURN
ENDIF

C     -----------------------------------------------------------------
IF(KF.EQ.12) THEN
NEW FUNCTION 1 : F MIN f(0,0,...,0 )=0
CALL NFUNCT1(M,X,F)
RETURN
ENDIF

C     -----------------------------------------------------------------
IF(KF.EQ.13) THEN
NEW FUNCTION 2 : F MIN f(0,0,...,0 )=0
CALL NFUNCT2(M,X,F)
RETURN
ENDIF

C     -----------------------------------------------------------------
IF(KF.EQ.14) THEN
WOOD FUNCTION : F MIN : M=4
CALL WOOD(M,X,F)
RETURN
ENDIF

C     -----------------------------------------------------------------
IF(KF.EQ.15) THEN
FENTON & EASON FUNCTION : : M=2
CALL FENTONEASON(M,X,F)
RETURN
ENDIF

C     -----------------------------------------------------------------
IF(KF.EQ.16) THEN
CALL TYPICAL(M,X,F)
RETURN
ENDIF

C     -----------------------------------------------------------------
IF(KF.EQ.17) THEN
WILD FUNCTION : F MIN (-15.8151514,..., -15.8151514)= M * 67.4677348
CALL WILD(M,X,F)
RETURN
ENDIF

C     =================================================================
WRITE(*,*)'FUNCTION NOT DEFINED. PROGRAM ABORTED'
STOP
END
C >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
C FOR BETA=0, EVERY PERMUTED SOLUTION IS A GLOBAL MINIMUM
C BETA => 0. CHANGE IF NEEDED. SMALLER BETA RAISES DIFFICULY
C PERM FUNCTION #2 MIN = 0 AT (1/1, 1/2, 1/3, 1/4,..., 1/M)
C -----------------------------------------------------------------
C FOR BETA=0, EVERY PERMUTED SOLUTION IS A GLOBAL MINIMUM
C BETA => 0. CHANGE IF NEEDED. SMALLER BETA RAISES DIFFICULY
C PERM FUNCTION #1 MIN = 0 AT (1, 2, 3, 4)
C -----------------------------------------------------------------
C ZERO SUM FUNCTION : MIN  = 0 AT SUM(X(I))=0
C >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
C
C DIMENSION X(*)
C SUBROUTINE ZEROSUM(M,F,X)
C IMPLICIT DOUBLE PRECISION (A-H,O-Z)
C COMMON /RNDM/IV,IV
C INTEGER IU,IV
C ZERO SUM FUNCTION : MIN  = 0 AT SUM(X(I))=0
C F=0.D00
C DO I=1,M
C IF(DABS(X(I)).GT.10.D00) THEN
C CALL RANDOM(RAND)
C X(I)=(RAND-0.5D00)*20
C ENDIF
C ENDDO
C SUM=0.D00
C IF(SUM.NE.0.D00) F=1.D00+(10000*DABS(SUM))**0.5
C RETURN
C END

C SUBROUTINE PERM1(M,F,X)
C IMPLICIT DOUBLE PRECISION (A-H,O-Z)
C COMMON /RNDM/IV,IV
C INTEGER IU,IV
C PERM FUNCTION #1 MIN = 0 AT (1, 2, 3, 4)
C C FOR BETA=0, EVERY PERMUTED SOLUTION IS A GLOBAL MINIMUM
C BETA=50.D00
C F=0.D00
C DO I=1,M
C IF(DABS(X(I)).GT.M) THEN
C CALL RANDOM(RAND)
C X(I)=(RAND-0.5D00)*2*M
C ENDIF
C ENDDO
C DO K=1,M
C SUM=0.D00
C DO I=1,M
C SUM=SUM+(I**K+BETA)*((X(I)/I)**K-1.D00)
C ENDDO
C F=F+SUM**2
C ENDDO
C RETURN
C END

C SUBROUTINE PERM2(M,F,X)
C IMPLICIT DOUBLE PRECISION (A-H,O-Z)
C COMMON /RNDM/IV,IV
C INTEGER IU,IV
C DIMENSION X(*)
C PERM FUNCTION #2 MIN = 0 AT (1/1, 1/2, 1/3, 1/4,..., 1/M)
C C FOR BETA=0, EVERY PERMUTED SOLUTION IS A GLOBAL MINIMUM
C BETA=10.D00
C DO I=1,M
C IF(DABS(X(I)).GT.1.D00) THEN
C CALL RANDOM(RAND)
C X(I)=(RAND-1.5D00)**2
C ENDIF
C SGN=X(I)/DABS(X(I))
C ENDDO
C F=0.D00
C DO K=1,M
C SUM=0.D00
C
DO I=1,M
SUM=SUM+(I+BETA)*(X(I)**K-(1.000/I)**K)
ENDDO
F=F+SUM**2
ENDDO
RETURN
END

SUBROUTINE POWERSUM(M,F,X)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
COMMON /RNDM/II,IV
INTEGER IU,IV
DIMENSION X(*)
POWER SUM FUNCTION; MIN = 0 AT PERM(1,2,2,3) FOR B=(8,18,44,114)
0 =< X <=4
F=0.000
DO I=1,M
X(I)=RNDM**4
ENDDO
DO K=1,M
SUM=0.000
DO I=1,M
SUM=SUM+X(I)**K
ENDDO
IF(K.EQ.1) B=8.000
IF(K.EQ.2) B=18.000
IF(K.EQ.3) B=44.000
IF(K.EQ.4) B=114.000
F=F+(SUM-B)**2
ENDDO
RETURN
END

SUBROUTINE BUKIN6(M,F,X)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
COMMON /RNDM/II,IV
DIMENSION X(*)
BUKIN'S 6TH FUNCTION MIN = 0 FOR (-10, 1)
-15. LE. X(1) .LE. -5 AND -3 .LE. X(2) .LE. 3
IF(X(1).LT.-15.000 .OR. X(1).GT.-5.000) THEN
CALL RANDOM(RAND)
X(1)=-(RAND**10+5.000)
ENDIF
IF(DABS(X(2)).GT.3.000) THEN
CALL RANDOM(RAND)
X(2)=(RAND-.5000)**6
ENDIF
F=-100.000*DASQRT(DABS(X(2)-0.0100*X(1)**2))+ 0.0100*DABS(X(1)+10.000)
RETURN
END

SUBROUTINE DCS(M,F,X)
FOR DEFLECTED CORRUGATED SPRING FUNCTION
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
COMMON /RNDM/II,IV
INTEGER IU,IV
DIMENSION X(*)
OPTIMAL VALUES OF (ALL) X ARE ALPHA , AND K IS ONLY FOR SCALING
DATA K,ALPHA/5,5.000/ K AND ALPHA COULD TAKE ON ANY OTHER VALUES
DO I=1,m
IF(DABS(X(I)).GT.20.000) then
MOST OF THE OTHER METHODS DO NOT PERFORM WELL
A(5)=0.217886; SUM OF SQUARES OF DEVIATION = 0.318593458
AND R=0.99945.
QUASI-NEWTON METHOD WITH SUM OF SQUARES OF DEVIATION = 0.298900994
HOUGEN FUNCTION (HOUGEN-WATSON MODEL FOR REACTION KINATICS)
MIN F(0, 0, ..., 0) = 0
---------------------------------------------------------------------
SUBROUTINE FUNCT7(M,F,X)
MIN F(0, 0, ..., 0) = 0
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
COMMON /RNDM/IV,IV
INTEGER IU,IV
DIMENSION X(*)
F=0.D00
DO I=1,M
IF(DABS(X(I)).GT.1.28D00) THEN
CALL RANDOM(RAND)
X(I)=(RAND-0.5D00)*2.56D00
ENDIF
END
DO I=1,M
CALL RANDOM(RAND)
F=F+(I*X(I)**4)
ENDDO
CALL RANDOM(RAND)
F=F+RAND
RETURN
END
---------------------------------------------------------------------
SUBROUTINE HOUGEN(A,M,F)
PARAMETER (N=13,K=3)
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
COMMON /RNDM/IV,IV
INTEGER IU,IV
DIMENSION X(N,K),RATE(N),A(*)
---------------------------------------------------------------------
HOUGEN FUNCTION (HOUGEN-WATSON MODEL FOR REACTION KINATICS)
NO. OF PARAMETERS (A) TO ESTIMATE = 5 = M
BEST RESULTS ARE: \text{FMIN}=0.298900994
A(1)=1.253031; A(2)=1.190943; A(3)=0.062798; A(4)=0.040063
A(5)=0.112453 ARE 2nd BEST ESTIMATES OBTAINED BY ROSENROCK 
QUASI-NEWTON METHOD WITH SUM OF SQUARES OF DEVIATION = 0.298900994
AND R=0.99945.
SECOND BEST RESULTS: \text{fmin}=0.298901 for X(1.25258611. 0.06277558222,
0.400477567, 0.112414812, 1.19137715) given by DE with ncross=0.
THE NEXT BEST RESULTS GIVEN BY HOOKE-JEEVES & QUASI-NEWTON
A(1)=2.475221;A(2)=0.599177; A(3)=0.124172; A(4)=0.083517
A(5)=0.217886; SUM OF SQUARES OF DEVIATION = 0.318593458
R=0.99941
MOST OF THE OTHER METHODS DO NOT PERFORM WELL

DATA X(1,1),X(1,2),X(1,3),RATE(1) / 470,300,10,8.55/
DATA X(2,1),X(2,2),X(2,3),RATE(2) / 285,80,10,1.79/
DATA X(3,1),X(3,2),X(3,3),RATE(3) / 470,300,120,4.82/
DATA X(4,1),X(4,2),X(4,3),RATE(4) / 470,80,120,0.02/
DATA X(5,1),X(5,2),X(5,3),RATE(5) / 470,80,10,2.75/
DATA X(6,1),X(6,2),X(6,3),RATE(6) / 100,190,10,14.39/
DATA X(7,1),X(7,2),X(7,3),RATE(7) / 100,80,65,2.54/
DATA X(8,1),X(8,2),X(8,3),RATE(8) / 470,190,65,4.35/

DEHARD.f
C     DE with ncross=0 SUCCEEDS. RPS SUCCEEDS.
C     RPS IS QUITE ROBUST. DE with ncross=1 FAILS TO FIND MINIMUM, but
C     NOTE: FOR LARGER L AND U LIMITS THIS FUNCTION MISGUIDES DE, BUT ALLOSTERIC ANZYME REACTION AS A NONLINEAR REGRESSIO PROBLEM

C     -----------------------------------------------------------------
C     GIUNTA FUNCTION
C     A(4)*X(I,3))
C     FX=(A(1)*X(I,2)-X(I,3)/A(5))/(1.D00+A(2)*X(I,1)+A(3)*X(I,2)+
C   1 FORMAT(4F8.2)
C     WRITE(*,1)((X(I,J),J=1,K),RATE(I),I=1,N)

DEHARD.f 14/18

&

DATA
DATA
DIMENSION
INTEGER
COMMON
PARAMETER
END

F
FX
D
ENDDO
DATA
DATA
DIMENSION
COMMON
IMPLICIT
SUBROUTINE
END
RETURN
END

C     SUBROUTINE GIUNTA(M,X,F)
C     IMPLICIT DOUBLE PRECISION (A-H,O-Z)
COMMON /RNDM/IV,IV
INTEGER IU,IV
DIMENSION X(*)
GIUNTA FUNCTION
X(I) = -1 TO 1; M=2
DO I=1,M
 IF(DABS(X(I)) .GT. 1.000) THEN
 CALL RANDOM(RAND)
 X(I)=-(RAND-0.5D00)*2.000
 ENDIF
ENDIF
C=15.000/15.D00
f=0.d00
DO i=1,m
 F=f + DSIN(C*X(i)-1.0D0)+DSIN(C*X(i)-1.0D0)**2+
& DSIN(4*(C*X(i)-1.0D0))/50
endo
f=f+0.6d0
RETURN
END

C     SUBROUTINE KONALIK(M,X,F)
PARAMETER(N=11)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
COMMON /RNDM/IV,IV
INTEGER IU,IV
DIMENSION X(*),A(N),BI(N)
DATA (A(I),I=1,N) /0.1957D0, 0.1947D0, 0.1735D0, 0.1600D0, 0.0844D0,
& 0.0627D0, 0.0456D0, 0.0342D0, 0.0323D0, 0.0235D0, 0.0246D0/
DATA (BI(I),I=1,N) /0.2500, 0.5D0, 1.D0, 2.D0, 4.D0, 6.D0, 8.D0,
& 10.D0, 12.D0, 14.D0, 16.D0/
KONALIK, J & MORRISON, JF (1968) : ANALYSIS OF KINETIC DATA FOR ALLOSTERIC ANZYME REACTION AS A NONLINEAR REGRESSIO PROBLEM
MATH. BIOSC. 2: 57-66.
NOTE: FOR LARGER L AND U LIMITS THIS FUNCTION MISGUIDES DE, BUT RPS IS QUITE ROBUST. DE with ncross=1 FAILS TO FIND MINIMUM, but DE with ncross=0 SUCCEEDS. RPS SUCCEEDS.
DO I=1,M

IF(DABS(X(I)).GT.4.D00) THEN
CALL RANDOM(RAND)
X(I)=(RAND-0.5D00)*8
ENDIF
ENDDO
C FOR X(I)=0.1928D00; X(2)=0.1905D00; X(3)=0.1230D00; X(4)=0.1356D00
FMIN=0.3075D-03 ; -4 <= X(I) <= 4.
F=0.D00
DO I=1,N
F1=X(I)*(BI(I)**(-2)+X(2)/BI(I))
F2=BI(I)**(-2)+X(3)/BI(I)+X(4)
F=F+(A(I)-F1/F2)**2
ENDDO
f=f*1000.D00
RETURN
END
-----------------------------------------------
SUBROUTINE NFUNCT1(M,X,F)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
COMMON /KFF/KF,NFCALL
COMMON /RNDM/IU,IV
INTEGER IU,IV
DIMENSION X(*)
C MIN F (1, 1, ..., 1) =2
DO I=1,M
IF(X(I).LT.0.D00 .OR. X(I).GT.1.D00) THEN
CALL RANDOM(RAND)
X(I)=RAND
ENDIF
ENDDO
S=0.D00
DO I=1,M-1
S=S+X(I)
ENDDO
X(M)=(M-S)
F=(1.D00+X(M))*X(M)
RETURN
END
-----------------------------------------------
SUBROUTINE NFUNCT2(M,X,F)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
COMMON /KFF/KF,NFCALL
COMMON /RNDM/IU,IV
INTEGER IU,IV
DIMENSION X(*)
C MIN F (1, 1, ..., 1) =2
DO I=1,M
IF(X(I).LT.0.D00 .OR. X(I).GT.1.D00) THEN
CALL RANDOM(RAND)
X(I)=RAND
ENDIF
ENDDO
S=0.D00
DO I=1,M-1
S=S+(X(I)+X(I+1))/2.D00
ENDDO
X(M)=(M-S)
F=(1.D00+X(M))*X(M)
RETURN
END
-----------------------------------------------
SUBROUTINE FLETHER(M,X,F)
FLETHER-Powell function, M <= 10, else it is very much slow
SOLUTION: MIN F = 0 for X=(C1, C2, C3,...,CM)
PARAMETER(N=10) ! for dimension of different matrices and vectors
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
COMMON /KFF/KF,NFCALL
1006: COMMON /RNDM/IU,IV
1007: INTEGER IU,IV
1008: DIMENSION X(*),A(N,N),B(N,N),AA(N),BB(N),AL(N),C(N),C1(N)
1009: PI=4*DAN(1.000)
1010: C GENERATE A(I,J) AND B(I,J) BETWEEN (-100, 100) RANDOMLY.
1011: C(I) = BETWEEN (-PI, PI) IS EITHER GIVEN OR RANDMELY GENERATED.
1012: C DATA (C(I),I=1,10)/1,1,2,3,-3,-2,-1,0,1,2,3/ ! BETWEEN -PI AND PI
1013: DATA (C1(I),I=1,10)/-3,-3.02,-3.01,1,1.01,1.02,1.03,-.08,.001,3/ ! another example c1 = 0
1014: C DATA (C1(I),I=1,10)/0,0,0,0,0,0,0,0,0,0/ ! another example c1 = 0
1015: NC=1 ! DEFINE NC HERE 0 OR 1 OR 2
1016: C IF NC=0, C1 FROM DATA IS USED (THAT IS FIXED C);
1017: C IF NC=1, C IS MADE FROM C1 BY ADDING RANDM PART - THAT IS C=C1+r
1018: C IF NC=2 THEN C IS PURELY RANDM THAT IS C= 0 + r
1019: C IN ANY CASE C LIES BETWEEN -PI AND PI.
1020: C FIND THE MAX MAGNITUDE ELEMENT IN C1 VECTOR (UPTO M ELEMENTS)
1021: CMAX=DABS(C1(1))
1022: DO J=2,M
1023: IF(DABS(C1(J)),GT,CMAX) CMAX=DABS(C1(J))
1024: END DO
1025: RANGE=PI- CMAX
1026: C-------------------------------------------------------
1027: DO J=1,M
1028: DO I=1,M
1029: CALL RANDOM(RAND)
1030: A(I,J)=(RAND-0.5D00)*200.000
1031: CALL RANDOM(RAND)
1032: B(I,J)=(RAND-0.5D00)*200.000
1033: END DO
1034: IF(NC.EQ.0) AL(J)=C1(J) ! FIXED OR NON-STOCHASTIC C
1035: IF(NC.EQ.1) THEN
1036: CALL RANDOM(RAND)
1037: AL(J)=C1(J)+(RAND-0.5D00)*2*RANGE ! A PART FIXED, OTHER STOCHASTIC
1038: END IF
1039: IF(NC.EQ.2) THEN
1040: CALL RANDOM(RAND)
1041: AL(J)=(RAND-0.5D00)*2*PI ! PURELY STOCHASTIC
1042: END IF
1043: END DO
1044: DO I=1,M
1045: AA(I)=0.000
1046: DO J=1,M
1047: AA(I)=AA(I)+A(I,J)*DSIN(AL(J))+B(I,J)*DCOS(AL(J))
1048: END DO
1049: END DO
1050: END DO
1051: !-------------------------------------------------------
1052: DO I=1,M
1053: IF(DABS(X(I)),GT,PI) THEN
1054: CALL RANDOM(RAND)
1055: X(I)=(RAND-0.5D00)*2*PI
1056: END IF
1057: END DO
1058: DO I=1,M
1059: BB(I)=0.000
1060: DO J=1,M
1061: BB(I)=BB(I)+A(I,J)*DSIN(X(J))+B(I,J)*DCOS(X(J))
1062: END DO
1063: END DO
1064: F=0.000
1065: DO I=1,M
1066: F=F+(AA(I)-BB(I))**2
1067: END DO
1068: RETURN
1069: END
1070: !-------------------------------------------------------
1071: SUBROUTINE FENTONEASON(M,X,F)
1072: IMPLICIT DOUBLE PRECISION (A-H, O-Z)
DIMENSION X(*)

C PENNON & EASON FUNCTION M=2

DO I=1,M
IF(DABS(X(I)) .GT. 100.000) THEN
CALL RANDOM(RAND)
ELSE
X(I)=(RAND-.5D00)*200
ENDIF
ENDDO

F=1.2D00+0.1*X(1)**2 + (0.1D00+0.1*X(2)**2)/X(1)**2 +
& (.1*X(1)**2*X(2)**2+10.000)/((X(1)*X(2))**4)
RETURN
END

-------------------------------------------------------------

SUBROUTINE WOOD(M,X,F)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
COMMON /RNDM/IU,IV
INTEGER IU,IV
DIMENSION X(*)

DO I=1,M
IF(DABS(X(I)) .GT. 5.000) THEN
CALL RANDOM(RAND)
ELSE
X(I)=(RAND-.5D00)*10
ENDIF
ENDDO

F1=100.*(X(2)+X(1)**2)**2 + (1.D0-X(1))**2 + 90.*(X(4)-X(3)**2)**2
F2=(1.D0-X(3))**2 +10.1*((X(2)-1.D0)**2+(X(4)-1.D0)**2)
F3=19.8*(X(2)-1.D0)*(X(4)-1.D0)
F=F1+F2+F3
RETURN
END

-------------------------------------------------------------

SUBROUTINE TYPICAL(M,X,F)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
COMMON /KFF/KF,NFCALL
COMMON /RNDM/IU,IV
INTEGER IU,IV
DIMENSION X(*)

A TYPICAL NONLINEAR MULTI-MODAL FUNCTION FMIN=0

DO I=1,M
IF(DABS(X(I)) .GT. 10.000) THEN
CALL RANDOM(RAND)
ELSE
X(I)=(RAND-.5D00)*20
ENDIF
ENDDO

F=F+DCOS( DABS(X(I))-X(I-1) ) / DABS(X(I-1)+X(I) )
ENDDO

IF 0.001*X(I) IS ADDED TO F, IT BECOMES UNIMODAL

F=F+0.001*X(I)
RETURN
END

-------------------------------------------------------------

SUBROUTINE WILD(M,X,F)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
COMMON /KFF/KF,NFCALL
COMMON /RNDM/IU,IV
INTEGER IU,IV
DIMENSION X(*)

A TYPICAL NONLINEAR MULTI-MODAL FUNCTION FMIN = (-15.8151514,..., -15.8151514)= M * 67.4677348

DO I=1,M
IF(DABS(X(I)) .GT. 50.000) THEN
CALL RANDOM(RAND)
ELSE
X(I)=(RAND-.5D00)*100
ENDIF
END
ENDDO
1141: F=0.D00
1142: DO I=1,M
1143: FX=10*DSIN(0.3*X(I))*DSIN(1.3*X(I)**2) +0.00001*X(I)**4 +0.2*X(I)
1144: F=F+ (FX+80.D00)
1145: ENDDO
1146: RETURN
1147: END