PDoublePop: An implementation of parallel genetic algorithm for function optimization

Ioannis G. Tsoulos[(1)]*, Alexandros Tzallas[(1)], Dimitris Tsalikakis[(2)]

[(1)]Department of Communications, Informatics and Management, Technological Educational Institute of Epirus, Greece
[(2)]University of Western Macedonia, Greece

Abstract

A software for the implementation of parallel genetic algorithms is presented in this article. The underlying genetic algorithm is aimed to locate the global minimum of a multidimensional function inside a rectangular hyperbox. The proposed software named PDoublePop implements a client - server model for parallel genetic algorithms with advanced features for the local genetic algorithms such as: an enhanced stopping rule, an advanced mutation scheme and periodical application of a local search procedure. The user may code the objective function either in C++ or in Fortran77. The method is tested on a series of well - known test functions and the results are reported.

PACS:02.60.-x ; 02.60.Pn ; 07.05.Kf; 02.70.Lq; 07.05.Mh

PROGRAM SUMMARY

Title of program: PDoublePop
Catalogue identifier:

Program available from: CPC Program Library, Queen's University of Belfast, N. Ireland.

Computer for which the program is designed and others on which it has been tested: The tool has been tested on Linux and FreeBSD. The tool is designed to be portable in all systems running the GNU C++ compiler, with Open MPI or LAM MPI.

Installation: Technological Educational Institute of Epirus, Greece.

Programming language used: GNU-C++, GNU-C, GNU Fortran - 77, MPI.

*Correspondig author. Email: itsoulos@teiep.gr

Memory required to execute with typical data: 200KB.

No. of bits in a word: 64

No. of processors used: many

Has the code been vectorised or parallelized?: Yes.

No. of bytes in distributed program,including test data etc.: 100 Kbytes.

Distribution format: gzipped tar file.

Keywords: Global optimization, stochastic methods, genetic algorithms, parallel programming.

Nature of physical problem: A series of problems in science and engineering usually can be formulated as a problem of minimizing a function of many variables. The so called local optimization techniques are frequently trapped in local minima, that it sub optimal solutions. For that reason researchers should use more advanced methods that aim to estimate the global minimum of the function.

Typical running time: Depending on the objective function.

LONG WRITE UP

1 Introduction

The frequently arised problem of discovering the global minimum of a multi - dimensional function can be formulated as

$$x^* = \arg\min_{x \in S} f(x) \tag{1}$$

Where $S \subset R^n$ is given by:

$$S = [a_1, b_1] \otimes [a_2, b_2] \otimes \ldots [a_n, b_n] \tag{2}$$

The abobe problem appears in a series of cases in physics [1, 2], chemistry [3, 4], economics [5] etc. The naturraly inspired method of genetic algorithms [6] is one of the most common method to tackle the problem of global minium. Genetic algorithms have been used in a many fields [7, 8, 9] and they have many advantages such as:

- Adaptation in every problem
- Requirement for the objective function only and not for gradient functions
- They can be parallelized easily

Although, genetic algorithms have many advantages they require many generations and a lot of function calls to finish. Hence, the parallelization of the process was mandatory. During the past years many researches have published articles and software using parallel genetic algorithms in many cases such as aerodynamic optimization [10], steel structure optimization [11], brain images [12] etc. The most frequent model for parallel genetic algorithms is the server - client model, such as the so called Island model [13, 14], where clients run a a local genetic algorithm and the server collects information from the clients and occasionally distributes information to them, such as the best discovered minimum. The proposed software named PDoublePop implements a client - server model for parallel genetic algorithms with advanced features for the local genetic algorithms such as: an enhanced stopping rule, an advanced mutation scheme and periodical application of a local search procedure. These new features have been proposed also in a recent work [15] and they are general enough to be adapted in any genetic algorithm. The proposed software has been written entirely in ansi C++ using MPI libraries and it is independed of the used MPI variant: it has been tested also in Lam MPI [16] and in Open Mpi [17] frameworks.

The rest of this article has as follows: in section 2 a detailed description of the method is given, in section 3 some experiments that outline the usability of the software are listed and in section 4 full documentation of the software is provided.

2 Method description

2.1 Server side algorithm

The algorithm executed on server machine gathers periodically from clients the discovered minimums. Upon termination of all clients, the server outputs the best discovered minimum (denoted by (x^*, y^*)). This algorithm has as follows.

1. **Set** N the number of clients.
2. **If** all clients have finished **then**
   (a) **Report** (x^*, y^*) as the global minimum.
   (b) **Terminate**
3. **EndIf**
4. **For** $i = 1 \ldots N$ **Do**
   (a) **Obtain** the mimum (x_i, y_i) from the client i
   (b) **If** $y_i < y^*$ **Then** $(x^*, y^*) = (x_i, y_i)$
5. **EndFor**
6. **Goto** 2

2.2 Client side algorithm

On each client a genetic algorithm as described in [15] is applied with the steps listed below.

1. **Parameter initialization**
   (a) **Set** iter=0, where iter is the current number of generations
   (b) **Set** N_c as the number of chromosomes in the population.
   (c) **Initialize** chromosomes $X_i, i = 1 \ldots N_c$
   (d) **Set** ITERMAX the maximum number of allowed generations.
   (e) **Set** p_c as the selection rate and p_m the mutation rate (positive values less than 1.0)
   (f) **Set** $f_l = \infty$, the best discovered local minimum
2. **Check** the termination rule. The rule is based on the observation of the variance of the best discovered value f_l. At every generation the variance $\sigma^{(\text{iter})}$ of f_l is computed. If genetic algorithm fails to find any new minimum for a number of generations, then the algorithm should terminate. Hence, the stopping rule is stated as

$$|f_h - f_l| \leq e \text{ OR } \sigma^{(\text{iter})} \leq \frac{\sigma^{(\text{last})}}{2} \quad \text{OR iter>ITERMAX} \tag{3}$$

   Where last is the generation where the best value f_l was firstly recorded. **If** equation 3 is true **then Goto** step 8.
3. **Calculate** the corresponding fitness value $f_i = f(X_i), i = 1 \ldots N_c$
4. **Genetic Operators**
   (a) **Apply** the process of crossover, where $M \leq N_c$ parents are selected from the population. The selection of every couple x, y is performed using tournement selection. Having selected the parents, the offsprings $\tilde{x}$ and $\tilde{y}$ are produced according to:

$$\begin{aligned} \tilde{x_i} &= a_i x_i + (1 - a_i) y_i \\ \tilde{y_i} &= a_i y_i + (1 - a_i) x_i \end{aligned} \tag{4}$$

   where a_i are random numbers in $[-0.5, 1.5]$ [18].
   (b) **Mutate** the offsprings produced during crossover with probability p_m. Suppose that the element i of a given chromosome x_i. The new element x_i' is calculated an equation borrowed from the popular PSO optimization method [19]:

$$x_i' = c_1 r_1 \left(x_i^b - x_i\right) + c_2 r_2 \left(x_l^b - x_i\right) \tag{5}$$

where $c_1$and c_2 are two positive constants (acceleration coefficients), r_1 and r_2 are random numbers in the range [0,1] and the vector x^b is a copy of the best so far position of chromosome x (i.e. the position with the lowest function value).

(c) **Replace** the m worst chromosomes in the population with the previously generated offsprings.

5. **Set** iter=iter+1
6. **Obtain** the best value in the population, denoted as f_l . **If** there was improvement Send (x_l, f_l) to Server machine
7. **Goto** step 3
8. **Apply** the local search procedure $\mathcal{LS}()$ to the best chromosome x_l. The local search procedure used was a BFGS variant due to Powell [20].
9. **Obtain** $(x_l, f_l) = \mathcal{LS}(x_l)$
10. **Send** (x_l, f_l) to Server machine.

3 Experiments

In order to measure the efficiency and the effectiveness of the proposed software a comparison is maid against one of the most known software GAlib [21]. More specific a parallel improvement of Galib named GAlib-mpi where used that utilizes MPI for parallelization. In the following the experimental test functions are listed as well as the results from the application of both GAlib and PDoublePop to the mentioned test functions.

3.1 Test functions

Rastrigin

$f(x) = x_1^2 + x_2^2 - \cos(18x_1) - \cos(18x_2)$, $x \in [-1, 1]^2$ with 49 local minima and global minimum $f^* = -2.0$.

Griewank2

$f(x) = 1 + \frac{1}{200}\sum_{i=1}^{2} x_i^2 - \prod_{i=1}^{2} \frac{\cos(x_i)}{\sqrt{(i)}}$, $x \in [-100, 100]^2$ with 529 local minima and global minimum $f^* = 0.0$

Gkls

$f(x) = \text{Gkls}(x, n, w)$, is a function with w local minima, described in [22], $x \in [-1, 1]^n$, $n \in [2, 100]$. In our experiments we use $n = 3$ and $w = 50$.

Test2N

$$f(x) = \frac{1}{2}\sum_{i=1}^{n} x_i^4 - 16x_i^2 + 5x_i$$

with $x \in [-5, 5]^n$. The function has 2^n local minima in the specified range. In our experiments we used the cases of $n = 5, 6, 7$.

Elp16

The function is given by the equation

$$f(x) = \sum_{i=1}^{n} i x_i^3$$

withe $x \in [-n, n]^n$. The value of global minimum is 0.0 and in our experiments we have used $n = 16$

Lennard Jones potential

The molecular conformation corresponding to the global minimum of the energy of N atoms interacting via the Lennard-Jones potential is determined for the case of N = 6 and N=7 atoms (denoted by POTENTIAL6 and POTENTIAL7). The value of global minimum for POTENTIAL5 is -12.712062 and -16.505384 for POTENTIAL7.

Tersoff potential

The binding energy for Tersoff potentials[23] is used as test function here. We used the potential for 3 atoms named Tersoff4 in the relevant experiments with global minimum -5.33

3.2 Experimental results

In table 1 we list the results from the application of GAlib and PDoublePop to the above test functions. The cells denote average number of generations. The figures in parentheses denote the fraction of runs that located the global minimum and were not trapped in one of the local minima. Absence of this number denotes that the global minimum has been recovered in every single run. The experiments were performed 30 times using different seed for the random generator each times and averages were taken. In every experiment the number of chromosomes was set to 200 and the maximum number of allowed generations was set to 2000. The stopping rule used was the same in all the algorithms. The column FUNCTION denotes the function name, the column GAlib denotes the average number of generations for GAlib algorithm and PDoublePop denotes the average number of generations for the proposed software.

The proposed method seems to require a signigicantly lower number of generations than GAlib. Also, the method managed to find the global minimum in all cases, as deduced from the fraction of runs that succeeded in finding the global minimum.

4 Software documentation

4.1 Distribution

The package is distributed in a tar.gz file named `PDoublePop.tar.gz` and under UNIX systems the user must issue the following commands to extract the associated files:

1. gunzip `PDoublePop.tar.gz`
2. tar xfv `PDoublePop.tar`

These steps create a directory named `PDoublePop` with the following contents:

1. **bin**: A directory which is initially empty. After compilation of the package, it will contain the executable **make_doublepop**
2. **doc**: This directory contains the documentation of the package (this file) in different formats: A LyX file, A LaTeX file and a PostScript file.
3. **examples**: A directory that contains some test functions.
4. **include**: A directory which contains the header files for all the classes of the package.
5. **src**: A directory containing the source files of the package.
6. **Makefile**: The input file to the `make` utility in order to build the tool. Usually, the user does not need to change this file.
7. **Makefile.inc**: The file that contains some configuration parameters, such as the name of the C++ compiler etc. The user must edit and change this file before installation.

4.2 Installation

The following steps are required in order to build the tool:

1. Uncompress the tool as described in the previous section.
2. `cd PDoublePop`
3. Edit the file `Makefile.inc` and change (if needed) the five configuration parameters.
4. Type `make`.

The five parameters in `Makefile.inc` are the following:

1. **CXX**: It is the most important parameter. It specifies the name of the C++ compiler. In most systems running the GNU C++ compiler this parameter must be set to mpiCC.

2. **CC**: If the user written programs are in C, set this parameter to the name of the C compiler. Usually, for the GNU compiler suite, this parameter is set to mpicc.

3. **F77**: If the user written programs are in Fortran 77, set this parameter to the name of the Fortran 77 compiler. For the GNU compiler suite a usual value for this parameter is mpif77.

4. **F77FLAGS**: The compiler GNU FORTRAN 77 (g77) appends an underscore to the name of all subroutines and functions after the compilation of a Fortran source file. In order to prevent this from happening we can pass some flags to the compiler. Normally, this parameter must be set to -fno-underscoring.

5. **ROOTDIR**: Is the location of the PDoublePop directory. It is critical for the system that this parameter is set correctly. In most systems, it is the only parameter which must be changed.

6. **MAKE**: The name of the `make` utility. The default value is make and on some systems (such as FreeBSD) it should change to `gmake`.

4.3 User written subprograms

The user should code the objective function using any of the following programming languages: C, C++, Fortran77. The C++ files should have the following command before any function in the file

```
extern "C" {
```

and the line

```
}
```

after them. The user should supply the following functions:

1. **getdimension**(): It is an integer function which returns the dimension of the objective function.

2. **getleftmargin**(left): It is a subroutine (or a void function in C) which fills the double precision array left with the left margins of the objective function.

3. **getrightmargin**(right): Is is a subrourine (or a void function in C) which fills the double precision array right with the right margins of the objective function.

4. **funmin**(x): It is a double precision function which returns the value of the objective function evaluated at point x.

5. **granal**(x,g): It is a subroutine (or a void function in C) which returns in a double precision array g the gradient of the objective function at point x.

4.4 The utility make_doublepop

After the compilation of the package, the executable `make_doublepop` will be placed in the subdirectory `bin` in the distribution directory. This program creates the final executable and it takes the following command line parameters:

1. `-h`: Prints a help screen and terminates.

2. `-p` **filename**: The **filename** parameter specifies the name of the file containing the objective function. The utility checks the suffix of the file and it uses the appropriate compiler. If this suffix is .cc or .c++ or .CC or .cpp, then it invokes the C++ compiler. If the suffix is .f or .F or .for then it invokes the Fortran 77 compiler. Finally, if the suffix is .c it invokes the C compiler.

3. `-o` **filename**: The **filename** parameter specifies the name of the final executable. The default value for this parameter is `doublepop`.

4.5 The program doublepop

The final executable `doublepop` has the following command line parameters:

1. `-h`:The program prints a help and it terminates.

2. `-c count`: The integer parameter `count` specifies the number of chromosomes for the genetic algorithm. The default value for this parameter is 100.

3. -g **gens**: The integer parameter **gens** determines the maximum number of generations allowed for the genetic algorithm. The default value is 200.

4. `-s srate`: The double parameter `srate` specifies the selection rate used in the genetic algorithm. The default value for this parameter is 0.10 (10%).

5. `-m mrate`: The double parameter `mrate` specifies the mutation rate used in the genetic algorithm. The default value for this parameter is 0.05 (5%).

6. -l **flag**: The integer parameter **flag** determines if local search will be applied every few generations to the best chromosome. The value of **flag**=1 denotes that this option is enabled and the value of **flag**=0 denoted that this option is disabled. the default value is 0 (disabled).

7. -p **method**: The integer parameter **method** with default value 0 accepts 3 values
   (a) 0: Simple printing method. In this case the values of best chromosome, best fitness and average number of generations are printed in free format.
   (b) 1: Csv printing method. In this case the values of best chromosome, best fitness and average number of generations are printed in csv format.
   (c) 2: Json printing method. In this case the values of best chromosome, best fitness and average number of generations are printed in json format.
8. `-r seed`: The integer parameter `seed` specifies the seed for the random number generator. It can assume any integer value.

4.6 A working example

4.6.1 Test function

Consider the rastrigin function

$$f(x_1, x_2) = x_1^2 + x_2^2 - \cos(18x_1) - \cos(18x_2) \;\; x \in [-1, 1]^2$$

The function has 49 local minima in that range with the global minimum located in $x^* = (0, 0)$, $f(x^*) = 0$. The plot of the function for the particular range is given in figure 1. The implementation of the function in ANSI C++ is given in figure 2 and in Fortran77 in figure 3. The corresponding files under `examples` subdirectory are `rastrigin.cc` and `rastrigin.f`. Change to subdirectory `examples` and issue the following command for the first file

```
../bin/make_doublepop -p rastrigin.cc
```

and for the second file

```
../bin/make_doublepop -p rastrigin.f
```

In both cases the software `make_doublepop` responds with

```
RUN ./doublepop IN ORDER TO RUN THE PROBLEM
```

4.6.2 Ssh preparation

In order to work with the parallel framework MPI between two computer we should ensure that we can login without password credentials. Let us that we have the following nodes: `node1` with IP 192.168.1.5 and `node2` with IP 192.168.1.10 In both nodes we have installed Linux or FreeBSD. The first node will be the server node and the second the client node. Under `node1` we execute the following commands:

```
ssh-keygen
ssh-copy-id -i ~/.ssh/id_rsa.pub 192.168.1.10
```

Now we can login to node2 issuing the following command

```
ssh 192.168.1.10
```

Now we should compile `PDoublePop` in all nodes and put the test executable `doublepop` in the same directory. In our case we use the subdirectory `/home/user/PDoublePop/examples/` where `user` is the name of the user in Unix environment.

4.6.3 The case of Lam MPI

Let us say that we have again the nodes node1 and node2 with the above IPs. We create the file hostfile under home subdirectory with the following contents

```
192.168.1.5
192.168.1.10
```

We start the MPI environment with the following command

```
lamboot -v -d /home/user/hostfile
```

Now we can run the example with the command

```
mpirun -np 3 /home/user/PDoublePop/examples/doublepop  -p 1
```

The output of the above command will be something like

```
0.0000 ,-0.0000 ,-2,42.00
```

If we want the output in JSON format then we issue the following

```
mpirun -np 3 /home/user/PDoublePop/examples/doublepop -p 2
```

And the output will be

```
{"x":[0.0000 ,-0.0000 ],"y":-2,"generations":42.00}
```

4.6.4 The case of OPEN MPI

For the case of Open MPI we start the program with just one command:

```
mpirun -hostfile /home/user/hostfile -np 3 /home/user/PDoublePop/examples/doublepop
```

The output will be the same as mentioned in Lam MPI case.

References

[1] Patrice Ogou Yapo, Hoshin Vijai Gupta, Soroosh Sorooshian, Multi-objective global optimization for hydrologic models, Journal of Hydrology 204, pp. 83-97, 1998.

[2] Q. Duan, S. Sorooshian, V. Gupta, Effective and efficient global optimization for conceptual rainfall-runoff models, Water Resources Research **28**, pp. 1015-1031 , 1992.

[3] David J. Wales, Harold A. Scheraga, Global Optimization of Clusters, Crystals, and Biomolecules, Science **27**, pp. 1368-1372, 1999.

[4] P.M. Pardalos, D. Shalloway, G. Xue, Optimization methods for computing global minima of nonconvex potential energy functions, Journal of Global Optimization **4**, pp. 117-133, 1994.

[5] Zwe-Lee Gaing, Particle swarm optimization to solving the economic dispatch considering the generator constraints, IEEE Transactions on **18** Power Systems, pp. 1187-1195, 2003.

[6] David E. Goldberg , John H. Holland, Genetic Algorithms and Machine Learning, Machine Learning 3, pp. 95-99, 1988.

[7] J.J. Grefenstette, R. Gopal, B. J. Rosmaita, D. Van Gucht, Genetic Algorithms for the Traveling Salesman Problem, In: Proceedings of the 1st International Conference on Genetic Algorithms, pp. 160 - 168, Lawrence Erlbaum Associates, 1985.

[8] P. Kaelo, M.M. Ali, Integrated crossover rules in real coded genetic algorithms, European Journal of Operational Research **176**, pp. 60-76, 2007.

[9] T. Prasad, N. Park, Multiobjective Genetic Algorithms for Design of Water Distribution Networks, J. Water Resour. Plann. Manage. **130**, pp. 73-82, 2004.

[10] D. J. Doorly , J. Peiró, Supervised Parallel Genetic Algorithms in Aerodynamic Optimisation, Artificial Neural Nets and Genetic Algorithms, pp. 229-233, 1997.

[11] Kamal C. Sarma, Hojjat Adeli, Bilevel Parallel Genetic Algorithms for Optimization of Large Steel Structures, Computer-Aided Civil and Infrastructure Engineering **16**, pp. 295-304, 2001.

[12] Yong Fan, Tianzi Jiang, Evans, D.J., Volumetric segmentation of brain images using parallel genetic algorithms, IEEE Transactions on Medical Imaging **21**, pp. 904-909, 2002.

[13] Arthur L. Corcoran, Roger L. Wainwright, A parallel island model genetic algorithm for the multiprocessor scheduling problem, SAC '94 Proceedings of the 1994 ACM symposium on Applied computing, pp. 483-487, 1994.

[14] Darrell Whitley , Soraya Rana, Robert B. Heckendorn, Island model genetic algorithms and linearly separable problems, Evolutionary Computing Volume 1305 of the series Lecture Notes in Computer Science, pp 109-125, 2005.

[15] I.G. Tsoulos, Modifications of real code genetic algorithm for global optimization, Applied Mathematics and Computation **203**, pp. 598-607, 2008.

[16] Burns Greg, Raja Daoud, James Vaigl, LAM: An open cluster environment for MPI, Proceedings of supercomputing symposium 94, 1994.

[17] Richard L. Graham , Timothy S. Woodall , Jeffrey M. Squyres, Open MPI: A Flexible High Performance MPI, Parallel Processing and Applied Mathematics Volume 3911 of the series Lecture Notes in Computer Science, pp 228-239, 2006.

[18] Z. Michaelewicz, Genetic Algorithms + Data Structures = Evolution Programs. Springer - Verlag, Berlin, 1996.

[19] J. Kennedy and R.C. Eberhart, The particle swarm: social adaptation in information processing systems, in: D. Corne, M. Dorigo and F. Glover (eds.), New ideas in Optimization, McGraw-Hill, Cambridge, UK, pp. 11-32, 1999.

[20] M.J.D Powell , A Tolerant Algorithm for Linearly Constrained Optimization Calculations, Mathematical Programming **45**, pp 547.

[21] M. WALL GAlib: A C++ library of genetic algorithm components. Mechanical Engineering Department, Massachusetts Institute of Technology 87: pp.54, 1996.

[22] M. Gaviano , D.E. Ksasov, D. Lera Y.D. Sergeyev, Software for generation of classes of test functions with known local and global minima for global optimization, ACM Trans. Math. Softw. **29**, pp. 469-480, 2003.

[23] J. Tersoff, New empirical approach for the structure and energy of covalent systems, Physical Review B 37, pp. 6991-7000, 1998.

Table 1: Average number of generations from GALib and the proposed method

FUNCTION	GALib	PDoublePop
Rastrigin	1608.27(0.97)	19.42
Griewank2	257.13(0.67)	20.34
Gkls350	287.53	24.97
Test2N5	840.83(0.87)	43.78
Test2N6	1067.77(0.80)	55.19
Test2N7	986.33(0.60)	74.78
Elp16	1061.73	286.99
Potential6	863.37(0.70)	418.87
Potential7	339.03	224.73
Tersoff3	323.80(0.60)	62.77

Figure 1: Plot figure of rastrigin function.

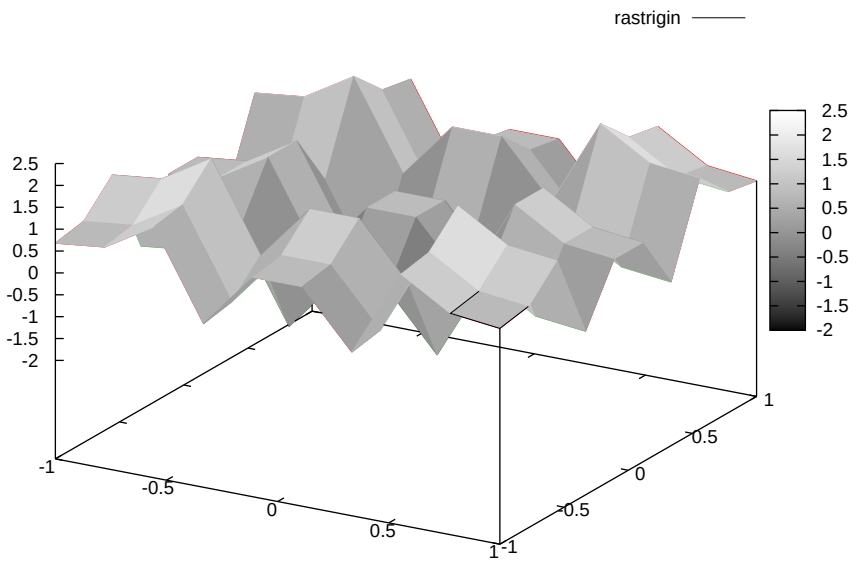

Figure 2: Implementation of rastrigin function in C++.

```
# include <math.h>
extern "C"
{
int       getdimension()
{
          return 2;
}
void      getleftmargin(double *x)
{
          x[0]=-1;
          x[1]=-1;
}

void      getrightmargin(double *x)
{
          x[0]=1;
          x[1]=1;
}
double    funmin(double *x)
{
    return x[0]*x[0]+x[1]*x[1]-cos(18.0*x[0])-cos(18.0*x[1]);
}

void      granal(double *x,double *g)
{
        g[0]=2.0*x[0]+18.0*sin(18.0*x[0]);
        g[1]=2.0*x[1]+18.0*sin(18.0*x[1]);
}
}
```

Figure 3: Implementation of rastrigin function in Fortran77.

```
integer function getdimension()
        getdimension=2
end

subroutine getleftmargin(x)
double precision x(2)
x(1)=-1.0
x(2)=-1.0
end

subroutine getrightmargin(x)
double precision x(2)
x(1)= 1.0
x(2)= 1.0
end

double precision function funmin(x)
double precision x(2)
funmin=x(1)**2+x(2)**2-cos(18*x(1))-cos(18*x(2))
end

subroutine granal(x,g)
double precision x(2)
double precision g(2)
g(1)=2*x(1)+18*sin(18.0*x(1))
g(2)=2*x(2)+18*sin(18.0*x(2))
end
```