MVF - Multivariate Test Functions Library in C for Unconstrained Global Optimization

Ernesto P. Adorio
Department of Mathematics
U.P. Diliman
ernesto.adorio@gmail.com
eadorio@yahoo.com

Revised January 14, 2005

Abstract

Mvf.c is a library of multidimensional functions written in C for unconstrained global optimization or with simple box constraints. This article describes the functions currently in the library, the supporting random number functions and instructions in creating Python bindings using the SWIG interface generator. Updated March 5, 2004.

Contents

1 MVF, a library of test functions in C **3**

2 Test functions **3**

2.1 Ackley function . . . 3
2.2 Beale function . . . 3
2.3 Bohachevsky functions . . . 3
2.4 Booth function . . . 4
2.5 Box-Betts exponential quadratic sum . . . 4
2.6 Branin function . . . 4
2.7 Camel functions . . . 4
2.8 Chichinadze function . . . 4
2.9 Cola function . . . 5
2.10 Colville function . . . 5
2.11 Corana function . . . 5
2.12 Easom function . . . 5
2.13 Egg holder function . . . 6
2.14 Exp2 function . . . 6
2.15 Gear function . . . 6
2.16 Goldstein-Price function . . . 6
2.17 Griewank function . . . 6
2.18 Hansen function . . . 6
2.19 Hartman function . . . 7
2.20 Himmelblau function . . . 7
2.21 Hyperellipsoid function . . . 7
2.22 Kowalik function . . . 7
2.23 Holzman functions . . . 8
2.24 Hosaki function . . . 8

2.25 Katsuuras function 8
2.26 Langerman function 8
2.27 Lennard-Jones potential function 9
2.28 Leon function 9
2.29 Levy functions 9
2.30 Matyas function 10
2.31 MaxMod function 10
2.32 McCormick function 10
2.33 Michalewitz function 10
2.34 Multimod function 10
2.35 Neumaier test functions 10
2.36 Odd square function 11
2.37 Paviani function 11
2.38 Plateau function 11
2.39 Powell function 12
2.40 Price function 12
2.41 Quartic with noise functions 12
2.42 Rana function 12
2.43 Rastrigin function 12
2.44 Rosenbrock saddle functions 13
2.45 Schaffer functions 13
2.46 Schubert function 13
2.47 Schwefel functions 13
2.48 Shekel foxholes functions 14
2.49 Sphere (harmonic) functions 16
2.50 Step function 16
2.51 Strectched V function 16
2.52 SumSquares function 16
2.53 Trecanni function 16
2.54 Trefethen4 function 17
2.55 Watson function 17
2.56 Xor function 17
2.57 Zettl function 17
2.58 Zimmerman function 17

A Source codes **19**
A.1 mvf.h header file 20
A.2 MVF.c 21

B Random number routines **51**
B.1 rnd.h header file for random number generator 51
B.2 rnd.c, basic random number routines 52

C Creating Python bindings with SWIG **55**

1 MVF, a library of test functions in C

A source library of multivariate test functions for global nonlinear optimization is an absolutely necessity for testing nonlinear global optimization algorithms. Most of the source libraries available for downloading from the WWW are written in Fortran, Matlab [BH] or `C++`. This MVF library can be freely used for educational and research purposes and the sources are included in this article. The current version is MVF C library Ver 0.1 and we would appreciate any constructive criticisms and bug reports from users of this C library.

All continuous multivariate functions for nonlinear global optimization follow the following prototype

```
double mvffuncname(int n, double x[])
```

where n is the number of variables, mvffuncname is the name of the function and x is an n dimensional input vector or array. The MVF library consist of a header file `mvf.h` and a single C source file `mvf.c` of the test functions. We describe in alphabetical order the functions which are presently in the MVF library. Most of the functions are also described in R. Iwaarden's thesis [I] who has also written a C++ library (though we are unable to download the C++ codes). The WWW is also an excellent resource for additional information on the test fucntions. In the description of the test functions, x^* indicates the global minimum point and f^* indicates the global minimum value.

We also provide instructions in the appendices to create a Python module from the C library using the SWIG interface generator.

We will add more to this function library and will include graphs and plots in future versions of this article. For further references to test function collections, try the the site

`http://www.mat.univie.ac.at/ neum/glopt/test.html#test_satis`

2 Test functions

2.1 Ackley function

Ackley function **mvfAckley** is computed as

$$f(n,x) = -20e^{-0.2\sqrt{\frac{1}{n}\sum_{i=0}^{n-1} x_i^2}} - e^{\frac{1}{n}\sum_{i=0}^{n-1}\cos(2\pi x_i)} + 20 + e$$

with domain $|x_i| \leq 30$. It has a global minimum 0 at $x_i = 0$.

2.2 Beale function

The two-dimensional **mvfBeale** function computes

$$f(n,x) = (1.5 - x_0 + x_0x_1)^2 + (2.25 - x_0 + x_0x_1^2)^2 + (2.625 - x_0 + x_0x_1^3)^2$$

with domain $-4.5 \leq x_i \leq 4.5$. The global minimum is 0 at the point $(3, 0.5)$.

2.3 Bohachevsky functions

These two-dimensional functions all have a global minimum 0 at the zeroth vector. The domain is $|x_i| \leq 50.0$. The first function **mvfBohachevsky1** is defined as

$$f(n,x) = x_o^2 + 2.0x_1^2 - 0.3\cos(3\pi x_0) - 0.4\cos(4\pi x_1) + 0.7$$

while the second function **mvfBohachevsky2** is defined as

$$f(n,x) = x_0^2 + 2.0x_1^2 - 0.3\cos(3\pi x_0) cos(4\pi x_1) + 0.3$$

2.4 Booth function

The two-dimensional function **mvfBooth** function computes

$$f(n,x) = (x_0 + 2x_1 - 7)^2 + (2x_0 + x_1 - 5)^2$$

domain $-10 \le x_i \le 10$. The global minimum is 0 athe point $(1,3)$.

2.5 Box-Betts exponential quadratic sum

The 3-dimensional function **mvfBoxBetts** computes the formula

$$f(n,x) = \sum_{i=0}^{n-1} g(x_i)^2$$

where $g(x) = exp(-0.1(i+1)x_0) - exp(-0.1(i+1)x_1) - (exp(-0.1\,(i+1)) - exp(-(i+1)))x_2$. This function has a global minimum 0 at the point $(1,10,1)$.

2.6 Branin function

Branin's function **mvfBranin** computes the value

$$f(n,x) = \left(x_1 - \frac{5.1x_0^2}{4\pi^2} + \frac{5x_0}{\pi} - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_0) + 10$$

with domain $-5 < x_0 \le 10, 0 \le x_1 \le 15$. It has a global minimum function value of 0.398 at the points $(-3.142, 12.275), (3.142, 2.275), (9.425, 2.425)$. **mvfBranin2** computes the value

$$f(n,x) = (1.0{-}2.0{*}x[1]{+}sin(4.0{*}\pi{*}x[1])/20.0{-}x[0])^2{+}(x[1]{-}sin(2.0{*}\pi{*}x[0])/2.0)^2;$$

with test domain $|x_i| = 10$. The global minimum is 0 at the point $(0.402357, 0.287408)$.

2.7 Camel functions

The 2-dimensional three-hump camel function **mvfCamel3** computes

$$f(n,x) = 2x_0^2 - 1.05x_0^4 + \frac{1}{6}x_0^6 + x_0x_1x_1^2$$

with domain $-5 \le x_i \le 5$. The global minimum is 0 at the point $(0,0)$.

The two-dimensional six-hump camel function **mvfCamel6** computes the value

$$f(n,x) = (4 - 2.1x_0^2 + \frac{1}{3}x_0^4)x_0^2 + x_0x_1 + (-4 + 4x_1^2)x_2^2$$

with domain $|xi| < 5$. It has a global minimum function value of -1.0316285 at the points $(0.08983, -0.7126)$ and $(-0.08983, 0.7126)$.

2.8 Chichinadze function

The 2-dimensional function **mvfChichinadze** computes

$$x_0^2 - 12x_0 + 11 + 10cos(\frac{\pi}{2}x_0) + 8\sin(5\pi x_0) - \frac{1}{\sqrt{5}}\exp\left(-\frac{(x_1 - 1/2)^2}{2}\right)$$

with domain $-30 \le x_0 \le 30, -10 \le x_1 \le 10$. The global minimum is 43.3159 at (5.90133, 0.5).

2.9 Cola function

The 17-dimensional function **mvfCola** computes indirectly the formula $f(n,u)$ by setting $x_0 = y_0, x_1 = u_0, x_i = u_{2(i-2)}, y_i = u_{2(i-2)+1}$

$$f(n,u) = h(x,y) = \sum_{j<i} (r_{i,j} - d_{i,j})^2$$

where $r_{i,j}$ is given by

$$r_{i,j} = [(x_i - x_j)^2 + (y_i - y_j)^2]^{1/2}$$

and d is a symmetric matrix given by

$$d = \begin{pmatrix} \dots \\ 1.27 & \dots \\ 1.69 & 1.43 & \dots \\ 2.04 & 2.35 & 2.43 & \dots \\ 3.09 & 3.18 & 3.26 & 2.85 & \dots \\ 3.20 & 3.22 & 3.27 & 2.88 & 1.55 & \dots \\ 2.86 & 2.56 & 2.58 & 2.59 & 3.12 & 3.06 & \dots \\ 3.17 & 3.18 & 3.18 & 3.12 & 1.31 & 1.64 & 3.00 & \dots \\ 3.21 & 3.18 & 3.18 & 3.17 & 1.70 & 1.36 & 2.95 & 1.32 & \dots \\ 2.38 & 2.31 & 2.42 & 1.94 & 2.85 & 2.81 & 2.56 & 2.91 & 2.97 & \dots \end{pmatrix}$$

This function has bounds $0 \le x_0 \le 4$ and $-4 \le x_i \le 4$ for $i = 1 \dots n-1$. It has a global minimum of 11.7464.

2.10 Colville function

The 4-dimensional function **mvfColville** is related to Rosenbrock's function. It is computed using the formula

$$f(n,x) = 100(x_0 - x_1^2)^2 + (1-x_0)^2 + 90(x_3 - x_2^2)^2 + (1-x_2)^2 + \\ 10.1((x_1-1)^2 + (x_3-1)^2) + 19.8(x_1-1)(x_3-1)$$

with domain $-10 \le x_i \le 10$. The global minimum 0 is at the point $(1,1,1,1)$.

2.11 Corana function

The 4-dimensional function **mvfCorana** has the global minimum $f* = 0$ and has domain $|x_i| \le 100$. It is computed as follows

$$f(n,x) = \sum_{i=0}^{3} \begin{matrix} 0.15(z_i - 0.05 \quad sgn(z_i))^2 d_i & \text{if } |x_i - z_i| < 0.05 \\ d_i x_i^2 & \text{otherwise} \end{matrix}$$

where $z_i = \lfloor |x_i/0.2| + 0.49999 \rfloor sgn(x_i) 0.2$ and $d = (1, 1000, 10, 100)$.

2.12 Easom function

The function **mvfEason** is a two-dimensional function with domain $|x_i| \le 100$ and with global minimum -1 at (π, π).

$$f(n,x) = -\cos(x_0)\cos(x_1)\exp(-(x_0 - \pi)^2 - (x_1 - \pi)^2.$$

2.13 Egg holder function

The function **mvfEggholder** has a domain $|x_i| < 512$.

$$f(n,x) = \sum_{i=0}^{n-2} -(x_{i+1}+47.0)\sin\sqrt{|x_{i+1}+x_i*0.5+47.0|}+\sin\sqrt{|x_i-(x_{i+1}+47.0)|}(-x_i);$$

2.14 Exp2 function

The 2-dimensional function **mvfExp2** computes

$$f(n,x) = \sum_{i=0}^{9}\left(e^{-ix_0/10} - 5e^{-ix_1/10} - e^{-i/10} + 5e^{-i}\right)^2$$

with domain $0 \le x_i \le 20$. It has a global minimum of 0 at the point $(1, 10)$.

2.15 Gear function

The 4-dimensional function **mvfGear** has the formula [CDG99]

$$f(n,x) = \{1.0/6.931 - \lfloor x_0\rfloor * \lfloor x_1\rfloor/(\lfloor x_2\rfloor * \lfloor x_3\rfloor)\}^2$$

with domain $12 \le x_1 \le 60$. It achieves global minimum value of 2.7e-12 at the point $x* = (16, 19, 43, 49)$ where x_0, x_1 and x_2, x_3 may be permuted.

2.16 Goldstein-Price function

The function `mvfGoldsteinPrice` returns the value

$$\begin{aligned} f(n,x) = {} & [1 + (x_0 + x_1 + 1)^2(19 - 14x_0 + 3x_0^2 - 14x_1 + 6x_0x_1 + 3x_1^2)] \\ & \times[30 + (2x_0 - 3x_1)^2(18 - 32x_0 + 12x_0^2 + 48x_1 - 36x_0x_1 + 27x_1^2)] \end{aligned}$$

with domain $|x_i| \le 2$. The minimum is 3 at the point $(0, -1)$.

2.17 Griewank function

The two-dimensional Griewank function **mvfGriewank2** computes

$$f(n,x) = \frac{(x_0^2 + x_1^2)}{200} - cos(x_0) * cos(x_1/\sqrt{2}) + 1$$

with test domain $|x_i| \le 100$ while the 10-dimensional Griewank[G] function **mvfGriewank10** computes the value

$$f(n,x) = \frac{1}{4000}\sum_{i=0}^{n-1}(x_i - 100)^2 - \prod_{i=0}^{n-1}\cos\left(\frac{x_i - 100}{\sqrt{i+1}}\right) + 1$$

with test domain $|x_i| \le 600$. The global minimum for both functions is 0 at $x_i = 0$.

2.18 Hansen function

This 2-dimensional function **mvfHansen** has the formula

$$f(n,x) = \sum_{i=0}^{4}(i+1)\cos(ix_0 + i + 1)\sum_{j=0}^{4}(j+1)\cos((j+2)x_1 + j + 1)$$

which has domain $|x_i| \le 10$. The global minimum is -176.54.

2.19 Hartman function

There are two special values of $n = 3$ and $n = 6$. Both `mvfHartman3` and `mvfHartman6` are computed by the same formula

$$f(n,x) = -\sum_{i=0}^{3} c_i \exp\left(-\sum_{j=0}^{n-1} A_{i,j}(x_j - p_{i,j})^2\right)$$

using different A and p matrices. The domain is $0 \le x_j \le 1, j \in [0, 1, \dots, n-1]$.

For $n = 3$, we have $A = \begin{pmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{pmatrix}$, $c = (1, 1.2, 3, 3.2)^t$ and

$p = \begin{pmatrix} 0.3689 & 0.1170 & 0.2673 \\ 0.4699 & 0.4387 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.03815 & 0.5743 & 0.8828 \end{pmatrix}$ and the global minimum is $f^\star - 3.86$ at $x^\star = (0.114, 0.556, 0.852)$.

For $n = 6$, we have $A = \begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & ,0.05 & 10 & 0.1 & 14 \end{pmatrix}$, $c = (1, 1.2, 3, 3.2)^t$,

$p = \begin{pmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{pmatrix}$ and the global minimum is -3.32 at the point $(0.201, 0.150, 0.477, 0.275, 0.311, 0.657)$.

2.20 Himmelblau function

The two-dimensional function **mvfHimmelblau** has domain $x_i \in [-6, 6]$ and possesses a global minimum $f* = 0$ at $x^\star = (3, 2)$.

$$f(n,x) = (x_0^2 + x_1 - 11)^2 + (x_0 + x_2^2 - 7)^2$$

2.21 Hyperellipsoid function

This n-dimensional function **mvfHyperellipsoid** with domain $|x_i| \le 0$ is defined as

$$\sum_{i=0}^{n-1} i^i + x_i^2.$$

2.22 Kowalik function

The number of variables n in Kowalik's function **mvfKowalik** is 4.

$$f(n,x) = \sum_{i=0}^{10} \left\{a_i - (x_0(b_i^2 + b_i x_1)/(b_i^2 + b_i x_2 + x_3)\right\}^2$$

where a and b are defined as $a = (0.1957, 0.1947, 0.1735, 0.1600, 0.0844, 0.0627, 0.0456, 0.0342, 0.0323, 0.0235, 0.0246)$ with $b = (4, 2, 1, 1/2, 1/4, 1/6, 1/8, 1/10, 1/12, 1/14, 1/16)$. The Kowalik function has domain $|x_i| < 5$.

Global minimum is $f^\star = 0.00030748610$ at $x^\star = (0.192833, 0.190836, 0.123117, 0.135766)$.

2.23 Holzman functions

The 3-dimensional function **mvfHolzman1** function computes the value **mvfHolzman1**

$$f(n,x) = \sum_{i=0}^{98} \left(-0.1(i+1) + exp(\frac{1}{x_0}(u_i - x_1)^{x_2}) \right)$$

where

$$u_i = 25 + +(-50 \log(0.01(i+1)))^{\frac{2}{3}}$$

with domain $0.1 \le x_0 \le 100, 0 \le x_1 \le 25.6, 0 \le x_2 \le 5$ The global minimum is 0 at the point $(50, 25, 1.5)$. This is a very expensive function to compute.

mvfHolzman2 The $n-$dimensional function **mvfHolzman2** computes

$$f(n,x) = \sum_{i=0}^{n-1} i x_i^4$$

with domain $-10 \le x_i \le 10$. This has a global minimum of 0 at the point $x_i = 0$.

2.24 Hosaki function

This 2-dimensional multivariate function **mvfHosaki** computes the value

$$f(n,x) = (1 - 8x_0 + 7x_0^2 - \frac{7}{3}x_0^3 + \frac{1}{4}x_0^4)x_1^2 e^{-x_1}$$

which has a global minimum -2.3458 at the point $(4, 2)$.

2.25 Katsuuras function

The function **mvfKatsuuras** has the global minimum $f* = 0$ at $x_i* = 0$ and is calculated as

$$f(n,x) = \prod_{i=0}^{n-1} \left(1 + (i+1) \sum_{k=1}^{d} round(2^k x_i) 2^{-k} \right)$$

where $n = 10$ and $d = 32$ for testing.

2.26 Langerman function

Langerman function **mvfLangerman** computes the value

$$f(n,x) = \sum_{i=0}^{29} c_i \exp(-\frac{(x - A_i)^t (x - A_i)}{\pi}) \cos((x - A_i)^t (x - A_i)\pi)$$

with domain $0 \le x_j \le 10, j \in [0, n-1]$. The matrix A and column vector c are identical for Shekel's foxhole `mvfFoxhole` function except that $c_2 = 1.5$ for Langerman's function. Langerman's function has a global minimum value of -1.4.

2.27 Lennard-Jones potential function

The Lennard-Jones potential function **mvfLennardJones** for a cluster of $N = n/3$ atoms in three dimensional space is given by

$$f(n,x) = \sum_{i=0}^{N-2} \sum_{j>i}^{N-1} \frac{1}{r_{ij}^{12}} - \frac{1}{r_{ij}^{6}}$$

where $r_{i,j}$ is the distance between atoms labeled i and j given by

$$r_{i,j} = \sqrt{(x_{3i} - x_{3j})^2 + (x_{3i+1} - x_{3j+1})^2 + (x_{3i+2} - x_{3j+2})^2}$$

The variables are the coordinates of the atoms. Tables of the putative minimum potential and the corresponding configuration of the atoms are listed in the Cambridge database with url `http://brian.ch.cam.ac.uk/CCD.html`

2.28 Leon function

Related to Rosenbrocks function, this function is computed as

$$f(n,x) = 100(x_1 - x_0^3)^2 + (x_0 - 1.0)^2$$

and which has a global minimum at $f^\star = 0$ at $x^\star = (1,1)$. The test domain is $|x_i| <= 10$.

2.29 Levy functions

The $n-$dimensional Levy function **mvfLevy** computes the value

$$f(n,x) = \sin^2(3\pi x_0) + \sum_{i=0}^{n-2} (x_i - 1)^2 \left(1 + \sin^2(3\pi x_{i+1})\right) + (x_{n-1} - 1)(1 + sin^2(2\pi x_{n-1}))$$

For $n = 4$, the global minimum is -21.502356 at the point $(1, 1, 1, -9.752356)$ while for $n = 5, 6, 7$, the global minimum is -11.504403 at the point $(1, \ldots, 1, -4.754402)$.

The 2-dimensional Levy function **mvfLevy3** computes the vlaue

$$f(n,x) = \sum_{i=0}^{4} (i+1)\cos((i+2)x_0 + i + 1) \sum_{j=0}^{4} (j+1)\cos((j+2)x_1 + j + 1)$$

while the Levy function **mvfLevy5** adds extra terms to mvfLevy3:

$$f(n,x) = mvfLevy3(n,x) + (x_0 + 1.42513)^2 + (x_1 + 0.80032)^2$$

According to Iwaarden, the global minimum [I] for **mvfLevy3** is -176.542 at the point $(-1.30671, -1.42513)$ and the global minimum for **mvfLevy5** is -176.138 at the point $(-1.30685, -1.42485)$.

(A test run indicates $f^* = 136.623088$ for **mvfLevy3** and we are looking for the corrected formula). The test domain for **mvfLevy3** and **mvfLevy5** is $|x_i| <= 10$.

The n-dimensional function **mvfLevy8** is given by

$$f(n,x) = sin^2(\pi * y_0) + \sum_{i=0}^{n-2} (y_i - 1)^2 (1 + 10 sin^2(\pi y_i + 1)) + (y_{n-1} - 1)^2 (1 + \sin^2(2\pi x_{n-1}))$$

where $y_i = 1 + \frac{x_i - 1}{4}$. The test domain is $|x_i| \leq 10$ and the global minimum is 0 at $x_i = 1$.

2.30 Matyas function

The function **mvfMatyas** computes

$$f(n,x) = 0.26(x_0^2 + x_1^2) - 0.48x_0x_1$$

with domain $-10 \leq x_i \leq 10$. The global minimum is 0 at the point (0,0).

2.31 MaxMod function

The n-dimensional function **mvfMaxmod** function computes

$$f(n,x) = \max(|x_i|)$$

with domain $-10 \leq x_i \leq 10$. The global minimum is 0 at the point $x_i = 0$.

2.32 McCormick function

The 2-dimensional function **mvfMccormick** computes

$$f(n,x) = sin(x_0 + x_1) + (x_1 - x_1)^2 - 1.5x_0 + 2.5x_1 + 1$$

where the domain is $-1.5 \leq x_0 \leq 4, -3 \leq x_1 \leq 4$. The global minimum is -1.9133 at the point $(-0.547, -1.54719)$. (Iwaarden [I] has $x* = (0,0)$ and $f(x*) = 0$).

2.33 Michalewitz function

The function **mvfMichalewicz** computes the value

$$f(n,x) = -\sum_{i=0}^{n} \left(\sin(x_i) \sin^{20}(\frac{ix_i^2}{\pi}) \right)$$

with domain $0 \leq x_i \leq \pi$. It has a global minimum value of $-0.966n$.

2.34 Multimod function

The n-dimensional function **mvfMultimod** function computes

$$f(n,x) = \sum_{i=0}^{n-1} |x_i| \prod_{i=0}^{n-1} |x_i|$$

with domain $-10 \leq x_i \leq 10$. The global minimum is 0 at the point $x_i = 0$.

2.35 Neumaier test functions

The following functions were introduced by Neumaier for testing global optimization software. The url `http://www.mat.univie.ac.at/ neum/glopt/my_problems.html` should be consulted for further information.

The function **mvfNeumaierPerm** is computed as

$$PERM(n, beta) : f(n,x) = \sum_{k=0}^{n-1} (\sum_{i=0}^{n-1} [(i+1)^{k+1} + \beta][(x_i/(i+1))^{k+1} - 1])^2$$

with bounds $x_i \in [-n, n]$. The global minimum is at $x_i = i + 1$ with $f* = 0$. Suggested values for tesing are $(n, \beta) = (4, 50), (4, 0.5), (10, 10^9), (10, 10^7)$.

The second Neumaier test function is **mvfNeumaierPerm0** calculated as

$$f(n,x) = \sum_{k=0}^{n-1} (sum_{i=0}^{n-1}[i+1+beta][(x_i)^{k+1} - (1/(i+1))^(k+1)])^2$$

with suggested bounds $x_i \in [-1,1]$ The global minimum is at $x_i = 1/i$ with $f* = 0$". Test cases suggested are $(n, beta) = (4,10), (10,100)$. The parameter β is a non-negative value.

The third Neumaier test function **mvfNeumaierPowersum** is computed as

$$f(n,x) = \sum_{k=0}^{n-1} ([\sum_{i=0}^{n-1} x_i^{k+1}] - b_k)^2$$

If the $b_k = \sum_{i=0}^{n-1} z_i^{k+1}$ for a target solution z, then the global minimum $f* = 0$. Suggested test instance for $b = (8,18,44,114)$ with $x_i \in [0,4]$. This has the solution $(1,2,2,3)$.

The fourth Neumaier test function **mvfNeumaierTrid** is calculated as

$$f(n,x) = [\sum_{i=0}^{n-1} (x_i - 1)^2] - [\sum_{i=1}^{n-1} x_i x_{i-1}]$$

A suitable domain would be $x_i \in [-n^2, n^2]$. The function has a global minimum at $x_i = i(n+1-i)$ with $f* = -n(n+4)(n-1)/6$.

2.36 Odd square function

This function **mvfOddSquare** has domain $|x_i| \leq 5\pi$ and the dimension usually is 5 or 10. Let $d_2^2 = ||x-a||_2^2$ and $d_\infty^2 = ||x-a||_\infty^2$.

$$f(n,x) = -exp(-cos(d_\infty^2)/(2\pi)) * cos(\pi d_\infty^2) * (1 + cd_2^2)/(d1 + 0.01))$$

The constant c is set at 0.2 and the vector $a = (1, 1.3, 0.8, -0.4, -1.3, 1.6, -0.2, -0.6, 0.5, 1.4)$.

2.37 Paviani function

This 10-dimensional function **mvfPaviani** is given by

$$f(n,x) = \sum_{i=0}^{n-1} \left\{ \ln^2(x_i - 2.0) + \ln^2(10 - x_i) \right\} - \left(\prod_{i=0}^{n-1} x_i \right)^{0.2}$$

with domain $2.0001 \leq x_i \leq 9.9999$. The global minimum is -45.7784 at the point $x_i = 9.340266$.

2.38 Plateau function

The 5 dimensional function [I2] **mvfPlateau** computes

$$f(n,x) = 30 + \sum_{i=0}^{4} \lfloor x_i \rfloor$$

with domain $|x_i| \leq 5.12$. This has a global minimum 30 at $x_i = 0$.

2.39 Powell function

The n dimensional function mvfPowell, where n is divisible by 4, computes

$$f(n,x) = \sum_{i=1}^{n/4} (x_{4j-4}+10x_{4j-3})^2+5(x_{4j-2}-x_{4j-1})^2+(x_{4j-3}-2x_{4j-2})^4+10(x_{4j-4}-x_{4j-1})^4$$

with domain $-4 \leq x_i \leq -5$. The global minimum is 0 at $(3,-1,0,1,\ldots,3,-1,0,1)$.

2.40 Price function

This 2-dimensional function **mvfPrice** computes

$$f(n,x) = (2x_0^3 x_1 - x_1^3)^2 + (6x_0 - x_1^2 + x_1)^2$$

with domain $-10 \leq x_i \leq 10$. The global minimum is 0 at the points $(0,0)$, $(2,4)$ and $(1.464,-2.506)$.

2.41 Quartic with noise functions

The function `mvfQuarticNoiseZ` returns the value

$$f(n,x) = \sum_{i=0}^{n-1} {x_i}^4 + N(0,1)$$

where $N(0,1)$ is a normal random variate with zero mean and standard deviation 1. The related **mvfQuarticNoiseU** a uniform random variate in the interval $[0,1]$.

2.42 Rana function

The function **mvfRana** has domain $|x_i| < 500$.

$$f(n,x) = \sum_{i=0}^{n-2} (x_{i+1}+1.0)\cos(t_2)\sin(t_1) + \cos(t_1)\sin(t_2) * x_i;$$

where $t_1 = \sqrt{|x_{i+1}+x_i+1.0|}$ and $t_2 = \sqrt{|x_{i+1}-x_i+1.0|}$.

2.43 Rastrigin function

Rastrigin function **mvfRastrigin** is computed as

$$f(n,x) = \sum_{i=0}^{n-1} \left\{ x_i^2 - 10 * \cos(2\pi x_i) + 10 \right\}$$

while the 2-dimensional function **mvfRastrigin2** is

$$f(n,x) = x_0 * x_0 + x_1 * x_1 - cos(12 * x[0]) - cos(18 * x_1);$$

The test domain is $|x_1| \leq 5.12$. Both functions have a global minimum of 0 at the zero vector $x_i = 0$.

2.44 Rosenbrock saddle functions

The basic 2-dimensional Rosenbrock **mvfRosenbrock** function computes

$$f(n,x) = 100(x_1 - x_0^2)^2 + (1.0 - x_0)^2$$

with domain $|x_i| \leq 10$. The global minimum is 0 at $x_i = -1.0$. The first extended Rosenbrock function **mvfRosenbrockExt1** computes

$$f(n,x) = \sum_{i=0}^{n/2-1} 100(x_{2i+1} - x_{2i}^2)^2 + (1 - x_{2i})^2$$

and the second extended Rosenbrock function **mvfRoenbrockExt2** computes

$$f(n,x) = \sum_{i=1}^{n-1} 100(x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2.$$

Both functions have test domain $|x_i \leq 10$. The global minimum is 0 at $x_i = 1$.

2.45 Schaffer functions

There are two 2-dimensional Schaffer functions in the mvf library. Both have the same global minimum $f* = 0$ at $x_i = 0$. The first schaffer function **mvf-Schaffer1** computes

$$f(n,x) = 0.5 + \frac{\{sin(\sqrt{x_0^2 + x_1^2})\}^2 - 0.5}{\{1.0 + 0.001(x_0^2 + x_1^2)\}^2}$$

and it has domain $|x_i| \leq 100$. The second function **mvfSchaffer2** computes

$$f(n,x) = (x_0^2 + x_1^2)^{1/4}(50(x_0^2 + x_1^2)^{0.1} + 1)$$

2.46 Schubert function

Function **mvfSchubert** computes the value

$$f(n,x) = \sum_{i=0}^{n-1} \sum_{j=0}^{j<5} (j+1) * \sin((j+2)x_i + (j+1))$$

with domain $|x_i| \leq 10.0$. It has global minimum -24.062499 at some points $(-6.774576, -6.774576), \ldots, (5.791794, 5.791794)$.

2.47 Schwefel functions

The function **mvfSchwefel1_2** computes

$$f(n,x) = \sum_{i=0}^{n-1} \left\{ \sum_{j=0}^{j<i} x_i \right\}^2$$

with domain $|x_i| < 10$. The global minimum function value is 0 at $x_i = 0$. The function **mvfSchwefel2_21** computes the value

$$f(n,x) = max(|x_i|)$$

with domain $|x_i| < 10$. The global minimum is 0 at $x_i = 0$. The function **mvfSchwefel2_22** computes the value

$$f(n,x) = \sum_{i=0}^{n-1} |x_i| + \prod_{i=0}^{n-1} |x_i|$$

with domain $|x_i| < 10$ and the global minimum function value 0 occurs at $x_i = 0$. The function **mvfSchwefel2_26** returns the value

$$f(n,x) = -\sum_{i=0}^{n-1} x_i sin(\sqrt{x_i})$$

with domain $|x_i| < 500$ and the global minimum is -12569.5 at $x_i = 420.9687$ for $n = 3$.

The 3-dimensional function **mvfSchwefel3_2** computes

$$f(n,x) = \sum_{i=1}^{n-1} (x_0 - x_i)^2 + (1 - x_i)^2;$$

with domain $|x_i| < 10$. The global minimum function value is 0 at the point (1,1,1).

2.48 Shekel foxholes functions

The two-dimensional Shekel's function **mvfShekel2** uses the formula

$$\frac{1}{f(n,z)} = \frac{1}{500} + \sum_{j=0}^{24} \frac{1}{c_j + \sum_{i=0}^{1} (x_i - A_{i,j})^6}$$

with domain $|x_i| <= 65.536$. The global minimum is 1 at the point $(-32, 32)$. The matrix A is given by

$$A = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \dots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \dots & 32 & 32 & 32 \end{pmatrix}$$

Shekel 4-dimensional foxhole is a 4-dimensional function which computes the value of

$$f(n,x) = \sum_{i=0}^{m} \frac{1}{(x - A_i)^t (x - A_i) + c_i}$$

with domain $0 \leq x_i \leq 10$. The most common values for m are 5,7 and 10, giving rise to the corresponding functions `mvfShekel4_5`, `mvfShekel4_7`, and `mvfShekel4_10`. The (10×4) matrix A is given by

$$A = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{bmatrix}$$

The column vector c is given by

$c = (0.1, 0.2, 0.2, 0.4, 0.4, 0.6, 0.3, 0.7, 0.5, 0.5)^t$

The **Shekel4** function has the following behavior for $m = 5, 7, 10$:

m	Global Minimum	Points
5	-10.1532	(4.00004, 4.00013, 4.00004, 4.00013)
7	-10.403	(4.00057,4.00069,3.99949,3.99961)
10	-10.5364	(4.00075,4.00059,3.99966,3.99951)

Shekel's 10-dimensional foxhole function **mvfShekel10** returns the value

$$f(n,x) = -\sum_{i=0}^{29} \frac{1}{(x-A_i)^t(x-A_i)+c_i}$$

where A_i is the ith row of a (30×10) matrix and c_i is the $(i+1)$th element of a 30-rowed column vector c. This function has domain $0 \leq x_j \leq 10, j \in [0, n-1]$.

The matrix A is given by

$$A = \begin{bmatrix}
9.681 & 0.667 & 4.783 & 9.095 & 3.517 & 9.325 & 6.544 & 0.211 & 5.122 & 2.020 \\
9.400 & 2.041 & 3.788 & 7.931 & 2.882 & 2.672 & 3.568 & 1.284 & 7.033 & 7.374 \\
8.025 & 9.152 & 5.114 & 7.621 & 4.564 & 4.711 & 2.996 & 6.126 & 0.734 & 4.982 \\
2.196 & 0.415 & 5.649 & 6.979 & 9.510 & 9.166 & 6.304 & 6.054 & 9.377 & 1.426 \\
8.074 & 8.777 & 3.467 & 1.863 & 6.708 & 6.349 & 4.534 & 0.276 & 7.633 & 1.567 \\
7.650 & 5.658 & 0.720 & 2.764 & 3.278 & 5.283 & 7.474 & 6.274 & 1.409 & 8.208 \\
1.256 & 3.605 & 8.623 & 6.905 & 0.584 & 8.133 & 6.071 & 6.888 & 4.187 & 5.448 \\
8.314 & 2.261 & 4.224 & 1.781 & 4.124 & 0.932 & 8.129 & 8.658 & 1.208 & 5.762 \\
0.226 & 8.858 & 1.420 & 0.945 & 1.622 & 4.698 & 6.228 & 9.096 & 0.972 & 7.637 \\
305 & 2.228 & 1.242 & 5.928 & 9.133 & 1.826 & 4.060 & 5.204 & 8.713 & 8.247 \\
0.652 & 7.027 & 0.508 & 4.876 & 8.807 & 4.632 & 5.808 & 6.937 & 3.291 & 7.016 \\
2.699 & 3.516 & 5.874 & 4.119 & 4.461 & 7.496 & 8.817 & 0.690 & 6.593 & 9.789 \\
8.327 & 3.897 & 2.017 & 9.570 & 9.825 & 1.150 & 1.395 & 3.885 & 6.354 & 0.109 \\
2.132 & 7.006 & 7.136 & 2.641 & 1.882 & 5.943 & 7.273 & 7.691 & 2.880 & 0.564 \\
4.707 & 5.579 & 4.080 & 0.581 & 9.698 & 8.542 & 8.077 & 8.515 & 9.231 & 4.670 \\
8.304 & 7.559 & 8.567 & 0.322 & 7.128 & 8.392 & 1.472 & 8.524 & 2.277 & 7.826 \\
8.632 & 4.409 & 4.832 & 5.768 & 7.050 & 6.715 & 1.711 & 4.323 & 4.405 & 4.591 \\
4.887 & 9.112 & 0.170 & 8.967 & 9.693 & 9.867 & 7.508 & 7.770 & 8.382 & 6.740 \\
2.440 & 6.686 & 4.299 & 1.007 & 7.008 & 1.427 & 9.398 & 8.480 & 9.950 & 1.675 \\
6.306 & 8.583 & 6.084 & 1.138 & 4.350 & 3.134 & 7.853 & 6.061 & 7.457 & 2.258 \\
0.652 & 2.343 & 1.370 & 0.821 & 1.310 & 1.063 & 0.689 & 8.819 & 8.833 & 9.070 \\
5.558 & 1.272 & 5.756 & 9.857 & 2.279 & 2.764 & 1.284 & 1.677 & 1.244 & 1.234 \\
3.352 & 7.549 & 9.817 & 9.437 & 8.687 & 4.167 & 2.570 & 6.540 & 0.228 & 0.027 \\
8.798 & 0.880 & 2.370 & 0.168 & 1.701 & 3.680 & 1.231 & 2.390 & 2.499 & 0.064 \\
1.460 & 8.057 & 1.336 & 7.217 & 7.914 & 3.615 & 9.981 & 9.198 & 5.292 & 1.224 \\
0.432 & 8.645 & 8.774 & 0.249 & 8.081 & 7.461 & 4.416 & 0.652 & 4.002 & 4.644 \\
0.679 & 2.800 & 5.523 & 3.049 & 2.968 & 7.225 & 6.730 & 4.199 & 9.614 & 9.229 \\
4.263 & 1.074 & 7.286 & 5.599 & 8.291 & 5.200 & 9.214 & 8.272 & 4.398 & 4.506 \\
9.496 & 4.830 & 3.150 & 8.270 & 5.079 & 1.231 & 5.731 & 9.494 & 1.883 & 9.732 \\
4.138 & 2.562 & 2.532 & 9.661 & 5.611 & 5.500 & 6.886 & 2.341 & 9.699 & 6.500
\end{bmatrix}$$

The column vector c is given by

$c = (0.806, 0.517, 0.10, 0.908, 0.965, 0.669, 0.524, 0.902, 0.531, 0.876, 0.462,$
$0.491, 0.463, 0.714, 0.352, 0.869, 0.813, 0.811, 0.828, 0.964, 0.789,$
$0.360, 0.369, 0.992, 0.332, 0.817, 0.632, 0.883, 0.608, 0.326)^t$

Shekel function consists of many foxholes of varying depths surrounded by relatively flat surfaces. Most algorithms become trapped in first foxhole they fall into.

2.49 Sphere (harmonic) functions

The function `mvfSphere` computes the value

$$f(n,x) = \sum_{i=0}^{n-1} x_i^2$$

in the domain $|x| = 100$. It has a global minimum $f^\star = 0$ at $x_i = 0$. The sphere function is non-linear, convex, unimodal and symmetric.

Somewhat related is the **mvfSphere2** function which is computed as

$$f(n,x) = \sum_{i=0}^{n-1} \{ \sum_{j=0}^{j<=i} x_j \}^2$$

and has the same global minimum and point.

2.50 Step function

The function `mvfStep` computes the value

$$f(n,x) = \sum_{i=0}^{n-1} \left(\lfloor x_i \rfloor + 0.5 \right)^2$$

in the domain $|x| = 100$. It has a global minimum 0 at $x_i = 0.5$. The presence of many flat plateus and steep ridges presents difficulties for algorithms based on gradient information.

Another step function **mvfStep2** is given by

$$f(n,x) = 6n + \sum_{i=0}^{n-1} \lfloor x_i \rfloor$$

where the domain is $|x_i| \leq 5.12$. The global minimum point is at $x_i = 0$.

2.51 Strectched V function

The function **mvfStretchedV** has domain $|x_i| = 10.0$.

$$f(n,x) = \sum_{i=0}^{n-2} t^{1/4} \sin 50.0 t^{0.1} + 1.0^2$$

where $t = x_{i+1}^2 + x_i^2$.

2.52 SumSquares function

The function **mvfSumSquares** computes

$$f(n,x) = \sum_{i=0}^{n-1} i x_i^2$$

with domain $-10 \leq x_i \leq 10$. The global minimum is 0 at $x_i = 0$.

2.53 Trecanni function

The 2-dimensional function **mvfTrecanni** computes

$$f(n,x) = x_0^4 + 4x_0^3 + 4x_0^2 + x_1^2$$

with domain $-5 \leq x_i \leq 5$. The global minimum is 0 at the points $(0,0)$ and $(-2,0)$.

2.54 Trefethen4 function

The 2-dimensional function **mvfTrefethen4** appeared as an global minimization problem in an international contest for obtaining 100 accurate digits.

$$
\begin{aligned}
f(n,x) =& exp(sin(50.0*x[0])) + sin(60.0*exp(x[1])) + sin(70.0*sin(x[0])) \\
& + sin(sin(80*x[1])) - sin(10.0*(x[0]+x[1])) \\
& + 1.0/4.0*(x[0]*x[0]+x[1]*x[1])
\end{aligned}
$$

The domain is $-6.5 < x[0] < 6.5, -4.5 < x[0] < 4.5$ and the global minimum is attained at $(-0.0244031, 0.2106124)$ with value -3.30686865.

2.55 Watson function

This 6-dimensional **mvfWatson** function is computed from the formula

$$
\sum_{i=0}^{29}\{\sum_{j=0}^{4}((j-1)a_i^j x_{j+1}) - \left[\sum_{j=0}^{5} a_i^j x_{j+1}\right]^2 - 1\}^2 + x_0^2
$$

where the coefficient $a_i = i/29.0$. This function has a global minimum $f^\star = 0.002288$ at the point $x^\star = (-0.0158, 1.012, -0.2329, 1.260, -1.513, 0.9928)$. The test domain is $|x_i| \leq 10$.

2.56 Xor function

The 9-dimensional **mvfXor** function computes

$$
\begin{aligned}
f(n,x) = & 1/(1+exp(-x[6]/(1+exp(-x[0]-x[1]-x[4])) \\
& - x[7]/(1+exp(-x[2]-x[3]-x[5])) - x[8]))^2 \\
& + 1/sqr(1+exp(-x[6]/(1+exp(-x[4])) - x[7]/(1+exp(-x[5])) - x[8])) \\
& + 1/sqr(1-1/(1+exp(x[6]/(1+exp(-x[0]-x[4]))) - x[7]/(1+exp(-x[3]-x[5])) - x[8])) \\
& + 1/sqr(1-1/(1+exp(x[6]/(1+exp(-x[1]-x[4]))) - x[7]/(1+exp(-x[3]-x[5])) - x[8]))
\end{aligned}
$$

2.57 Zettl function

The two dimensional function **mvfZettl** function computes

$$
f(n,x) = (x_0^2 + x_1^2 - 2x_0)^2 + 0.25x_0
$$

which has the global minimum $f^\star = -0.003791$ at $x^\star = (-0.02990, 0)$. Use a test domain $|x_i| <= 10.0$.

2.58 Zimmerman function

The two dimensional function **mvfZimmerman** has domain $0 \leq x_i \leq 100$. Let us first define the following functions: $Zh1(x) = 9 - x[0] - x[1]; Zh2(x) = (x_0-3)^2 + (x_1-2.0)^2 - 16; Zh3(x) = x_0x_1 - 14$ and $Zp(t) = 100(1+t)$ where x is a vector and t is a scalar. Then the function is computed as

$$
\begin{aligned}
f(n,x) = \max\{& Zh1(x), Zp(Zh2(x))sgn(Zh2(x)), Zp(Zh3(x))sgn(Zh3(x)), \\
& Zp(-x[0])sgn(x[0]), Zp(-x[1])sgn(x[1])\}
\end{aligned}
$$

References

[BH] Bjorkman, Mattias Kenneth Holmstrom, "Global optimization using the Direct algorithm in Matlab", URL:http://www.ima.mdh.se/tom

[CDG99] Corne, Dorigo Glover, "New Ideas in Optimization", McGraw-Hill, 1999

[DS] L. C. W. Dixon and G. P. Szego, Eds., "Towards Global Optimization 2", Amsterdam, North Holland, 1978.

[DTU] `URL:http://www.imm.dtu.dk/_km/GlobOpt/testex/testproblems.html`

[GP] A. A. Goldstein I. F. Price, "On descent from local minima", Math. Comput. 25 (July 1971).

[G] A. O. Griewank, "Generalized Descent for global optimization", Journal of optimization theory and applications, Vol 34, No. 1, pp. 11-39, 1981

[H] J. K. Hartman, Tech. Rept. NP55HH72051A, Naval Postgraduate School, Monterey, CA, 1972

[I] Ronald John Iwaarden, "An improved unconstrained global optimization algorithm", Ph.D. Thesis, University of Colorado, Denver, 1996

[I2] L. Ingber, "Simulated Annealing, Practice Vs. Theory", Journal Math. Computing and Modelling, V18,N11, D1993, 29-57

[P1] W. L. Price, "Global optimization by Controlled Random Search", Journal of Optimization Theory and Applications 40 (1983), 333-348

[P2] W. L. Price, "Global optimization Algorithms for a CAD Workstation", Journal of Optimization Theory and Applications 55 (1987), 133-146

[S] D. P. Solomatine, "Application of Global Optimization to Models Calibration", 1995

[MN] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator", ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3-30 1998

A Source codes

Most of the global optimization functions in this article were described in R. Iwaarden's thesis [I] for testing his improved unconstrained optimization algorithm. The de Jong basic function test suite consists of the following optimization functions: 3-dimensional sphere, 2-dimensional rosenbrock saddle function, 5-dimensional step2 function, 30-dimensional quartic with noise function and 2-dimensional shekel foxhole function.

The Dixon-Szego test suite consists of 10 functions: branin, 6-hump camel, goldstein-price, hartman3, hartman6, rosenbrock. shekel4_5. shekel4_7, shekel4_10 and shubert. The first ICEO contest on evolutionary optimization has the following functions included in the test suite: sphere, griewank, shekel, michalewicz and langerman. The WWW site in Reference [DTU] has a list of 14 test functions with formulas for the gradient for some of the functions.

A.1 mvf.h header file

```
#ifndef _MVF_H
  #define _MVF_H

double sqr (double x );
double dmax (int n ,double * x );
double sgn (double x );
double dist2 (int n ,double * x ,double * y );
double distInf (int n ,double * x ,double * y );
double prod (int n ,double * x );

double mvfAckley (int n ,double * x );
double mvfBeale (int n ,double * x );
double mvfBohachevsky1 (int n ,double * x );
double mvfBohachevsky2 (int n ,double * x );
double mvfBooth (int n ,double * x );
double mvfBoxBetts (int n ,double * x );
double mvfBranin (int n ,double * x );
double mvfBranin2 (int n ,double * x );
double mvfCamel3 (int n ,double * x );
double mvfCamel6 (int n ,double * x );
double mvfChichinadze (int n ,double * x );
double mvfCola (int n ,double * x );
double mvfColville (int n ,double * x );
double mvfCorana (int n ,double * x );
double mvfEasom (int n ,double * x );
double mvfEggholder (int n ,double * x );
double mvfExp2 (int n ,double * x );
double mvfFraudensteinRoth (int n ,double * x );
double mvfGear (int n ,double * x);
double mvfGeneralizedRosenbrock (int n ,double * x );
double mvfGoldsteinPrice (int n ,double * x );
double mvfGriewank (int n ,double * x );
double mvfHansen (int n ,double * x );
double mvfHartman3 (int n ,double * x );
double mvfHartman6 (int n ,double * x );
double mvfHimmelblau (int n ,double * x );
double mvfHolzman1 (int n ,double * x );
double mvfHolzman2 (int n ,double * x );
double mvfHosaki (int n ,double * x );
double mvfHyperellipsoid (int n ,double * x );
double mvfKatsuuras (int n ,double * x );
double mvfKowalik (int n ,double * x );
double mvfLangerman (int n ,double * x );
double mvfLennardJones (int n ,double * x );
double mvfLeon(int n, double *x);
double mvfLevy (int n ,double * x );
double mvfMatyas (int n ,double * x );
double mvfMaxmod(int n, double *x);
double mvfMcCormick (int n ,double * x );
double mvfMichalewitz (int n ,double * x );
double mvfMultimod(int n, double *x);
void setNeumaierPerm (double beta );
void setNeumaierPerm0 (double beta );
double mvfNeumaierPerm (int n, double *x );
double mvfNeumaierPerm0 (int n, double *x );
double mvfNeumaierPowersum (int n, double *x );
double mvfNeumaierTrid (int n, double *x );
double mvfOddsquare (int n ,double * x );
double mvfPaviani (int n ,double * x );
double mvfPlateau (int n ,double * x );
double mvfPowell (int n ,double * x );
double mvfQuarticNoiseU (int n ,double * x );
double mvfQuarticNoiseZ (int n ,double * x );
double mvfRana (int n ,double * x );
double mvfRastrigin (int n ,double * x );
double mvfRastrigin2 (int n ,double * x );
double mvfRosenbrock (int n ,double * x );
double mvfSchaffer1 (int n ,double * x );
double mvfSchaffer2 (int n ,double * x );
double mvfSchwefel1_2 (int n ,double * x );
double mvfSchwefel2_21 (int n ,double * x );
double mvfSchwefel2_22 (int n ,double * x );
double mvfSchwefel2_26 (int n ,double * x );
double mvfShekel2 (int n ,double * x );
double mvfShekelSub4 (int m ,double * x );
double mvfShekel4_5 (int n ,double * x );
double mvfShekel4_7 (int n ,double * x );
double mvfShekel4_10 (int n ,double * x );
double mvfShekel10 (int n ,double * x );
double mvfShubert (int n ,double * x );
double mvfShubert2 (int n ,double * x );
double mvfShubert3 (int n ,double * x );
double mvfSphere (int n ,double * x );
double mvfSphere2(int n, double *x );
double mvfStep (int n ,double * x );
double mvfStretchedV (int n ,double * x );
double mvfSumSquares (int n ,double * x );
double mvfTrecanni (int n ,double * x );
double mvfTrefethen4 (int n ,double * x );
double mvfWatson(int n, double *x);
double mvfXor (int n, double *x);
double mvfZettl(int n, double *x);

double _Zh1 (double * x );
double _Zh2 (double * x );
double _Zh3 (double * x );
double _Zp (double x );
double mvfZimmerman (int n ,double * x );
#endif
```

A.2 MVF.c

The code listings for the functions are in the ”mvf.c” file. Feel free to use them for research purposes.

```
/*
```

```
-   File    mvf.c
-   Original compiled by Ernie.
-   Revisions:
-     2003.01.30  Ver 0.1
-     2004.03.05  Added EggHolder, StretchedV, Rana functions
-                 from http://www.ft.utb.cz/people/zelinka/soma/
-     2004.04.19  http://www.mat.univie.ac.at/~neum/glopt/janka/funcs.html
-     Bohachevsky1, Bohachevsky2, Easom
-     Hyperellipsoid, Neumaier2, Neumaier3,
-     Oddsquare, Schaffer1, Schaffer2, Zimmerman
-  LennardJones
-         Corana
-         Xor
*/

#include <stdio.h>
#include <math.h>

#include "rnd.h"
#include "mvf.h"

// ============================================== //

double sqr( double x )
{
  return x*x;
};

double dmax(int n, double *x)
{
int i;

double m = x[0];
for (i = 1; i < n; i++) {
if (x[i] > m) m = x[i];
}
return m;
}

double sgn(double x)
{
if (x < 0.0) return -1.0;
return 1.0;
}

double dist2(int n, double *x, double *y)
{
register int i;
double s, t;

s = 0.0;
for (i = 1; i < n; i++) {
t  = (x[i]-y[i]);
s += t * t;
}
return s;
}

double distInf(int n, double *x, double *y)
{
register int i;
double t,m;

m = fabs(x[0] - y[0]);
for (i = 1; i < n; i++) {
t  = fabs(x[i]-y[i]);
if (t > m) {
m = t;
}
}
return m;
}


double prod(int n, double *x)
{
  int i;
  double p;

  p = 1.0;
  for (i = 0; i < n; i++) {
    p *= x[i];
  }
  return -p;
}

double mvfAckley(int n, double *x)
/*
-Dimension: n arbitrary
-       Domain:   | x_i | <= 30.0
-       Minimum 0 at x_i = 0.0
*/
{
int i;
double t, s1, s2;

s1 = s2 = 0.0;

for (i = 0; i < n; i++) {
    t  = x[i];
    s1 += t * t;
    s2 += cos(2.0*M_PI * x[i]);
}
return -20.0 * exp(-0.2 * sqrt(s1/n)) - exp(s2 / n) +
         20.0 + M_E;
}
```

```
double mvfBeale(int n, double *x)
/*
Dimension: 2
Domains:  | x_i | le 4.5
Minimum: 0 at (3, 0.5)
*/
{
   return pow(1.5   - x[0] + x[0]*x[1], 2)  +
     pow(2.25 - x[0] + x[0] * x[1]*x[1], 2) +
     pow(2.625 - x[0] + x[0] * pow(x[1], 3), 2);
}

double mvfBohachevsky1(int n, double *x)
/*
- Dimension: 2
- Global minimum: 0 at (0,0)
- Domain: |x[i]| <= 50.0
*/
{
return  x[0] * x[0] + 2.0 * x[1]*x[1]
  - 0.3 * cos(3.0 * M_PI * x[0])
  - 0.4 * cos(4.0 * M_PI * x[1])
  + 0.7;
}

double mvfBohachevsky2(int n, double *x)
/*
- Dimension: 2
- Global minimum: 0 at (0,0)
- Domain: |x[i]| <= 50.0
*/
{
return x[0] * x[0] + 2.0 * x[1]*x[1]
  - 0.3 * cos(3.0 * M_PI * x[0]) * cos(4.0 * M_PI * x[1])
  + 0.3;
}


double mvfBooth(int n, double *x)
/*
Dimension: 2
Domain: |x_i| <=  10
Minimum: 0 at (1,3)
*/
{
   return pow(x[0] + 2*x[1] - 7, 2) +
          pow(2 * x[0] + x[1] - 5, 2);
}

double mvfBoxBetts(int n, double * x)
/*
Dimension: 3
Domain:  [0.9, 1.2], [9, 11.2], [0.9, 1.2]
Minimum: 0 at  (1, 10, 1)
```

```
*/
{
  int i;
  double x0=x[0],x1=x[1],x2=x[2] ;
  double sum=0.0 ;

  for (i=1; i<=10 ; i++)
    sum+=pow(exp(-0.1*i*x0)-exp(-0.1*i*x1)-
            (exp(-0.1*i)-exp(-1.0*i))*x2,2.0);
  return sum ;
}

double mvfBranin(int n, double *x)
/*
-   n = 2;
-   Domain:  -5.0 < x[0] <= 10.0,   0 <= x[1] <= 15
-   Minimum: 0.397889 at (-3.142, 12.275),
                         (3.142, 2.275),
                         (9.425, 2.425)
*/
{
double s, x0;

x0 = x[0];
s  = x[1] -
      ( 5.1/(4.0*M_PI*M_PI) * x0 - 5.0/M_PI) * x0 - 6.0;
  return s*s + 10*(1.0 - 1.0/(8.0*M_PI)) * cos(x0) + 10.0;
}


double mvfBranin2(int n, double *x)
{
  return pow(1.0-2.0*x[1]+sin(4.0*M_PI*x[1])/20.0-x[0],2)+
        pow(x[1]-sin(2.0*M_PI*x[0])/2.0,2);
}


double mvfCamel3(int n, double *x)
/*
-  Dimension: 2
-  Domain: | x[i] | <= 5
-  Minimum: 0 at (0,0)
*/
{
  double x02 = x[0] * x[0];
  double x04 = x02 * x02;
  double x06 = x04 * x02;

  return 2*x02-1.05*x04 +x06/6.0+x[0]*x[1]*x[1]*x[1];
}

double mvfCamel6(int n, double *x)
/*
Dimension: 2
```

```
*/
{
  double x1=x[0],x2=x[1] ;

  return 4.0*x1*x1-0.21E1*pow(x1,4.0)
         + pow(x1,6.0)/3+x1*x2-4.0*x2*x2
         + 4.0*pow(x2,4.0);
}

double mvfChichinadze(int n, double *x)
/*
-  Dimension: 2
-  Domain: x[0] <= 30, x[1] <= 10
-  f* = 43.3159
-  x* = (5.90133, 0.5)
*/
{
    return x[0]*x[0]-12 *x[0]+11+10*cos(M_PI/2.0*x[0])
      + 8 * sin(5 * M_PI * x[0])
      - exp ( -(x[1] - 0.5) * 0.5) / sqrt(5.0);
}

static  double dis[46] = {
  1.27,
  1.69,1.43,
  2.04,2.35,2.43,
  3.09,3.18,3.26,2.85,
  3.20,3.22,3.27,2.88,1.55,
  2.86,2.56,2.58,2.59,3.12,3.06,
  3.17,3.18,3.18,3.12,1.31,1.64,3.00,
  3.21,3.18,3.18,3.17,1.70,1.36,2.95,1.32,
  2.38,2.31,2.42,1.94,2.85,2.81,2.56,2.91,2.97
};

double mvfCola( int n, double * x )
{
  double sum = 0.0, temp;
  int i, j, t, k = 1;

  double mt[20] = {0, 0, 0, 0};

  for( i = 4; i < 20; i++)
    mt[i] = x[i-3];

  for( i = 1; i < 10; i++)
    for( j = 0; j < i; j++) {
      temp = 0.0;
      for( t = 0; t < 2; t++ )
        temp += sqr( mt[i*2+t]-mt[j*2+t] );
      sum += sqr( dis[k-1] - sqrt(temp) );
      k++;
  }
  return sum;
}
```

```
double mvfColville(int n, double *x)
/*
Dimension: 4
Domain: | x[i] <= 10
Global minimum: 0 at (1,1,1,1)
*/
{
  return 100 * pow(x[0]-x[1]*x[1], 2)
     + pow(1-x[0], 2)
     + 90 * pow(x[3] -x[2]*x[2], 2)
   + pow(1-x[2], 2)
   + 10.1 * (pow(x[1] -1, 2) +  pow(x[3] - 1, 2))
   + 19.8 * (x[1] - 1)*(x[3] - 1);
}


double mvfCorana(int n, double *x)
/*
- Dimension 4
- Domain: | x[i] | < 1000
*/
{
int i;
double s;
double z;
double d[] = {1, 1000, 10, 100};

s = 0.0;
for (i = 0; i < 4; i++) {
z = floor( fabs(x[i]/0.2) + 0.49999) * sgn(x[i]) * 0.2;
if (fabs(x[i] - z) < 0.05) {
s += 0.15 * sqr(z - 0.05 * sgn(z)) * d[i];
} else {
s += d[i] * x[i]*x[i];
}
};
return s;
}

double mvfEasom(int n, double *x)
/*
-Dim 2
-   Global minimum -1 at (pi, pi)
-   Domain: | x[i] | <= 100
*/
{
return -cos(x[0])
 * cos(x[1]) *
 exp(- sqr(x[0] - M_PI) -
       sqr(x[1] - M_PI));
}

double mvfEggholder(int n, double *x)
```

```
/*
- Dimension: n
- Domain:  | x_i | < 512
*/
{
  int i;
  double sum;

  sum = 0.0;
  for (i = 0; i < n-1; i++) {
    sum += -(x[i+1] + 47.0) * sin(sqrt(fabs(x[i+1] + x[i] * 0.5 + 47.0))) +
                            sin(sqrt(fabs(x[i] - (x[i+1] + 47.0)))) * (-x[i]);
  }
  return sum;
}


double mvfExp2(int n, double *x)
/*
Dimension: 2
Domain:  0 <= x[i] <= 20
Global minimum: 0 at (1, 10)
*/
{
  register int i;
  double   sum = 0.0;
  double   t;

  for (i = 0; i < 10; i ++) {
    t = exp(-i * x[0] / 10.0)
        - 5 * exp(- i * x[1] * 10) - exp(-i / 10.0)
        + 5 * exp(-i);
    sum += t * t;
  }
  return sum;
}

double mvfFraudensteinRoth(int n, double *x)
/*
- Dimension 2
*/
{
  return sqr(-13.0 + x[0] + ((5 - x[1]) * x[1] - 2) * x[1])
       + sqr(-29.0 + x[0] + ((x[1] + + 1.0) * x[1] - 14) * x[1]);
}

double mvfGear(int n, double *x)
{
        double t;

        /* n == 4 always */
        t = 1.0 / 6.931 - floor(x[0]) * floor(x[1]) / (floor(x[2]) * floor(x[3]));
        return t * t;
}
```

```
double mvfGeneralizedRosenbrock(int n, double * x)
{
  double s = 0.0;
  int    i;

  for( i = 1; i < n; i++ )
    s += 100.0*pow(x[i]-x[i-1]*x[i-1],2)
         + pow(x[i-1]-1.0,2);
   return s;
}

double mvfGoldsteinPrice(int n, double * x)
/*
-Dimension: n = 2 always
-  Domain:  | x_i | <= 2
-  Minimum value 3 at (0, -1)
*/
{

  return (1.0+pow(x[0]+x[1]+1.0,2)*
    (19.0-14.0*x[0]+3.0*x[0]*x[0]-14.0*x[1]
     +6.0*x[0]*x[1]+3.0*x[1]*x[1]))*
    (30.0+pow(2.0*x[0]-3.0*x[1],2)*
    (18.0-32.0*x[0]+12.0*x[0]*x[0]+48.0*x[1]
     -36.0*x[0]*x[1]+27.0*x[1]*x[1]));
}

double mvfGriewank(int n, double * x)
{
  int i;
  double sum=0 ;
  double prod=1 ;

  for (i=0 ; i<10 ; i++) {
    sum+=x[i]*x[i];
    prod*=cos(x[i]/sqrt((double)(i+1))) ;
  }
  return sum/4000.0-prod + 1 ;
}

double mvfHansen(int n, double * x)
{
  return (cos(1.0)+2.0*cos(x[0]+2.0)
    +3.0*cos(2.0*x[0]+3.0)+4.0*cos(3.0*x[0]+4.0)
    +5.0*cos(4.0*x[0]+5.0))*(cos(2.0*x[1]+1.0)
    +2.0*cos(3.0*x[1]+2.0)
    +3.0*cos(4.0*x[1]+3.0)
    +4.0*cos(5.0*x[1]+4.0)
    +5.0*cos(6.0*x[1]+5.0));
}

double mvfHartman3(int n, double *x)
{
```

```
int    i, j;
double s, t, t1;


static double a[4][3] = {
{  3, 10, 30},
{0.1, 10, 35},
{  3, 10, 30},
{0.1, 10, 35}
};
double c[] = { 1, 1.2, 3, 3.2 };
double p[4][3] = {
{0.3689, 0.1170, 0.2673},
{0.4699, 0.4387, 0.7470},
{0.1091, 0.8732, 0.5547},
{0.03815,0.5743, 0.8828}
};
n = 3; // forced check
s = 0.0;
for (i = 0; i < 4; i++) {
t = 0.0;
for (j = 0; j < n; j++) {
t1 =  x[j]-p[i][j];
t  += a[i][j] * (t1*t1);
}
s += c[i] * exp(-t);
}
return -s;
}

double mvfHartman6(int n, double *x)
/*
-  Harman_6 (m = 4)
-  Domain:  0 <= x_i <= 1
-  Minimum value -3.32  at (0.201, 0.150, 0.477, 0.275, 0.311, 0.657)
*/
{
int    i, j;
double s, t, t1;
static double a[4][6] = {
{10,  3,  17,   3.5,  1.7,  8.0},
{0.05,  10, 17, 0.1, 8,  14},
{3, 3.5, 1.7, 10, 17, 8},
{17, 8, 0.05, 10, 0.1, 14}
};

double c[] = {1, 1.2, 3, 3.2};
double p[4][6] = {
{0.1312, 0.1696, 0.5569, 0.0124, 0.8283, 0.5886},
{0.2329, 0.4135, 0.8307, 0.3736, 0.1004, 0.9991},
{0.2348, 0.1415, 0.3522, 0.2883, 0.3047, 0.6650},
{0.4047, 0.8828, 0.8732, 0.5743, 0.1091, 0.0381}
};
```

```
n = 6; // forced check
s = 0.0;
for (i = 0; i < 4; i++) {
t = 0.0;
for (j = 0; j < n; j++) {
t1 = x[j]-p[i][j];
t += a[i][j] * (t1*t1);
}
s += c[i] * exp(-t);
};
return -s;
}

double mvfHimmelblau(int n, double *x)
/*
-  Dimension: 2
-  Domain: | x[i] | < 5.0
-  Global minimum: 0 at (3,2).
*/
{
return sqr(x[0]*x[0]+x[1]-11.0) + sqr(x[0]+x[1]*x[1]-7);
}

double mvfHolzman1(int n, double *x)
/*
Dimension: 3
Domain: 0.1 <= x[0]  <= 100.0
        0 <= x[1] <= 25.6
          0 <= x[2] <= 5
Global minimum: 0 at (50, 25, 1.5)
*/
{
  int    i;
  double sum = 0.0;
  double ui;

  for (i = 0; i < 100; i++) {
    ui = 25 + pow(-50.0*log(0.01*(i+1)),2.0/3.0);
    sum += -0.1 * (i+1)
           +exp(1/ x[0] * pow(ui-x[1], x[2]));
  }
  return sum;
}

double mvfHolzman2(int n, double *x)
/*
Dimension: n
Domain: | x[i] | <= 10
Global minimum: 0 at x[i] = 0
*/
{
  int i;
  double sum = 0.0;
  for (i = 0; i < n; i++) {
    sum += i * pow(x[i] , 4);
  }
  return sum;
}

double mvfHosaki(int n, double *x)
/*
-n = 2
*/
{
return (1  + x[0] *
       (-8 + x[0] *
     (7  + x[0] *
     (-7.0/3.0 + x[0] *1.0/4.0
     )))) *
       x[1]*x[1] * exp(-x[1]);
}

double mvfHyperellipsoid(int n, double *x)
/*
-n = 30
-Domain: |x| <= 1.0
*/
{
int i;
double s  = 0.0;

for (i = 0; i < n; i++) {
s += i * i + x[i]*x[i];
}
return s;
}


double mvfKatsuuras(int n, double *x)
/*
= Dimension: n (10)
- Domain: | x[i] | <= 1000
- Global minimum 1.0 at 0 vector.
*/
{
int i, k;
int d = 32;
double prod;
double s;
double pow2;

prod = 1.0;
for (i = 0; i < n; i++) {
s = 0.0;
for (k = 1; k <= d; k++) {
pow2 = pow(2, k);
s += round(pow2 * x[i]) / pow2;
}
prod *= 1.0 + (i+1) * s;
}
return prod;
}

double mvfKowalik(int n, double *x)
/*
-n = 4
-Domain:  | x_i | < 5.0
-Global minimum is 0.0 at x_i = 0.00
*/
{
  static double b[] = {
    4.0, 2.0, 1.0, 1/2.0,1/4.0,1/6.0,1/8.0,
    1/10.0,1/12.0,1/14.0, 1/16.0 };

  static double a[] =  {
0.1957, 0.1947, 0.1735, 0.1600, 0.0844,
0.0627, 0.0456, 0.0342, 0.0323, 0.0235,
0.0246 };

int    i;
double sum;
double  bb, t;

sum = 0.0;
for (i = 0; i < 11; i++) {
  bb = b[i] * b[i];
  t = a[i]-(x[0]*(bb+b[i]*x[1])/(bb+b[i]*x[2]+x[3]));
sum += t * t;
}
return sum;
}

double afox10[30][10] = {
  { 9.681,0.667,4.783, 9.095, 3.517, 9.325, 6.544, 0.211, 5.122, 2.020},
  {9.400, 2.041, 3.788, 7.931, 2.882, 2.672, 3.568, 1.284, 7.033, 7.374},
  {8.025, 9.152, 5.114, 7.621, 4.564, 4.711, 2.996, 6.126, 0.734, 4.982},
  {2.196, 0.415, 5.649, 6.979, 9.510, 9.166, 6.304, 6.054, 9.377, 1.426},
  {8.074, 8.777, 3.467, 1.863, 6.708, 6.349, 4.534, 0.276, 7.633, 1.567},
  {7.650, 5.658, 0.720, 2.764, 3.278, 5.283, 7.474, 6.274, 1.409, 8.208},
  {1.256, 3.605, 8.623, 6.905, 0.584, 8.133, 6.071, 6.888, 4.187, 5.448},
  {8.314, 2.261, 4.224, 1.781, 4.124, 0.932, 8.129, 8.658, 1.208, 5.762},
  {0.226, 8.858, 1.420, 0.945, 1.622, 4.698, 6.228, 9.096, 0.972, 7.637},
  {7.305, 2.228, 1.242, 5.928, 9.133, 1.826, 4.060, 5.204, 8.713, 8.247},
  {0.652, 7.027, 0.508, 4.876, 8.807, 4.632, 5.808, 6.937, 3.291, 7.016},
  {2.699, 3.516, 5.874, 4.119, 4.461, 7.496, 8.817, 0.690, 6.593, 9.789},
  {8.327, 3.897, 2.017, 9.570, 9.825, 1.150, 1.395, 3.885, 6.354, 0.109},
  {2.132, 7.006, 7.136, 2.641, 1.882, 5.943, 7.273, 7.691, 2.880, 0.564},
  {4.707, 5.579, 4.080, 0.581, 9.698, 8.542, 8.077, 8.515, 9.231, 4.670},
  {8.304, 7.559, 8.567, 0.322, 7.128, 8.392, 1.472, 8.524, 2.277, 7.826},
  {8.632, 4.409, 4.832, 5.768, 7.050, 6.715, 1.711, 4.323, 4.405, 4.591},
  {4.887, 9.112, 0.170, 8.967, 9.693, 9.867, 7.508, 7.770, 8.382, 6.740},
  {2.440, 6.686, 4.299, 1.007, 7.008, 1.427, 9.398, 8.480, 9.950, 1.675},
  {6.306, 8.583, 6.084, 1.138, 4.350, 3.134, 7.853, 6.061, 7.457, 2.258},
  {0.652, .343, 1.370, 0.821, 1.310, 1.063, 0.689, 8.819, 8.833, 9.070},
  {5.558, 1.272, 5.756, 9.857, 2.279, 2.764, 1.284, 1.677, 1.244, 1.234},
  {3.352, 7.549, 9.817, 9.437, 8.687, 4.167, 2.570, 6.540, 0.228, 0.027},
  {8.798, 0.880, 2.370, 0.168, 1.701, 3.680, 1.231, 2.390, 2.499, 0.064},
  {1.460, 8.057, 1.336, 7.217, 7.914, 3.615, 9.981, 9.198, 5.292, 1.224},
  {0.432, 8.645, 8.774, 0.249, 8.081, 7.461, 4.416, 0.652, 4.002, 4.644},
  {0.679, 2.800, 5.523, 3.049, 2.968, 7.225, 6.730, 4.199, 9.614, 9.229},
  {4.263, 1.074, 7.286, 5.599, 8.291, 5.200, 9.214, 8.272, 4.398, 4.506},
  {9.496, 4.830, 3.150, 8.270, 5.079, 1.231, 5.731, 9.494, 1.883, 9.732},
  { 4.138, 2.562, 2.532, 9.661, 5.611, 5.500, 6.886, 2.341, 9.699, 6.500}
};



double cfox10[] = {
0.806,0.517,1.5,0.908,0.965,
0.669,0.524,0.902,0.531,0.876,
0.462,0.491,0.463,0.714,0.352,
0.869,0.813,0.811,0.828,0.964,
0.789,0.360,0.369,0.992,0.332,
0.817,0.632,0.883,0.608,0.326};

double mvfLangerman(int n, double *x)
{
int  i;
double  Sum, dist;

cfox10[2] = 1.5;
Sum = 0;
for ( i = 0; i < 5; i++ ){
      dist = dist2(n, x, afox10[i]);
  Sum -= cfox10[i] * (exp(-dist) / M_PI) * cos(M_PI * dist);
}
return Sum;
}

double mvfLennardJones(int n, double *x)
{
int i,j, i3, j3;
double z, dz, dzp, E;
int natoms;

natoms = n / 3;
for (i = 0; i < natoms; i++) {
i3 = i * 3;
for (j = i + 1; j < natoms; j++) {
dz = 0.0;
z   = x[i3] - x[j3=j*3];
dz  += z*z;
z   = x[i3+1] - x[j3+1];
dz  += z*z;
z   = x[i3+2] - x[j3+2];
dz  += z*z;

if (dz < 1.0e-6) { /* one pair of atoms too near ? */
return 1.0e30;
}
else {
dzp = 1.0 / (dz * dz * dz);
E += (dzp - 2.0) * dzp;
}
}
}
return E;
}

double mvfLeon(int n, double *x)
/*
See mvfRosenbrock for information.
*/
{
   double a = x[1]-x[0]*x[0]*x[0];
   double b = 1.0-x[0] ;

   return 100.0*a*a + b*b ;
}

double mvfLevy(int n, double * x)
/*
-  Global minimum
-    for n=4,  fmin    = -21.502356 at (1,1,1,-9.752356 )
-    for n=5,6,7, fmin = -11.504403 at (1,\dots,1,-4.754402 )
*/
{
  int    i;
  double sum=0.0;

  for (i=0 ; i<=n-2 ; i++)
    sum+=pow(x[i]-1,2.0)*(1+pow(sin(3*M_PI*x[i+1]),2.0));

  return pow(sin(3*M_PI*x[0]),2.0) + sum +
         (x[n-1]-1)*(1+pow(sin(2*M_PI*x[n-1]),2.0));
}

double mvfMatyas(int n, double *x)
     /*
       Dimension: 2
       Domain: |x[i] <= 10
       Global minimum: 0 at x[i] = 0
      */
{
  return 0.26 * (x[0]*x[0] + x[1]*x[1]) - 0.48 * x[0] * x[1];
}

double mvfMaxmod(int n, double *x)
/*
```

```
  Domain: |x[i] <= 10
  Global minimum: 0 at x[i] = 0
*/
{
  int i;
  double t = x[0];
  double u;

  for (i = 1; i < n; i++) {
    u = fabs(x[i]);
    if (u < t) t = u;
  }
  return u;
}

double mvfMcCormick(int n, double * x)
/*
  Dimension: 2
  Domain: -1.5 <= x[0] <= 4
               -3   <= x[1] <= 4
  Global minimum: -1.9133 at (-0.547, -1.54719).
*/
{
  return sin(x[0]+x[1])
        +pow(x[0]-x[1],2.0)-1.5*x[0]+2.5*x[1]+1.0 ;
}

double mvfMichalewitz(int n, double *x)
{
double   u;
int     i;

u=0;
for (i=0;i<n;i++) {
u = u + sin(x[i])
* pow(sin((i+1)*x[i]*x[i]/M_PI), 2.0 * 10.0);
}
return(-u);
}

double mvfMultimod(int n, double *x)
{
  int i;
  double t, s, p;

  s = p = fabs(x[0]);
  for (i = 1; i < n; i++) {
     t = fabs(x[1]);
     s += t;
     p *= t;
  }
  return s + p;
}
```

```
/*  Neumaier test functions */
double PERM_BETA = 50;
double PERM0_BETA = 10; /* Non-negative */

void setNeumaierPerm( double beta)
{
  PERM_BETA = beta;
}

void setNeumaierPerm0(double beta)
{
  PERM0_BETA = beta;
}

double mvfNeumaierPerm(int n, double x[])
/*
  Global minimum x[i] = (i+1) with f* = 0.
  Domain : x[i] in [-n,n] for i=0,...,n-1.
  suggested beta values: (n,beta) = (4,50), (4,0.5), (10,10^9), (10,10^7)
*/
{
  int i, k;
  double sum = 0.0;

  for (k = 0; k < n; k++) {
    for (i = 0; i < n; i++) {
       sum += (pow(i+1, k+1) + PERM_BETA) * sqr(pow(x[i]/(i+1), (k+1)) -1);
    }
  }
  return sum;
}


double mvfNeumaierPerm0(int n, double x[])
/*
 Domain: x_i in [-1,1] for i=0,...,n-1.
 Global minimu f* = 0 at x[i] = 1/(i+1)

 Suggested test values (n, beta) = (4,10) (10,100)
*/
{
  int i, k;
  double sum = 0.0;

  for (k = 0; k < n; k++) {
    for (i = 0; i < n; i++) {
      sum += ((i+1)+PERM0_BETA) * sqr(pow(x[i+1], k)- pow(1/(i+1), k));
    }
  }
  return sum;
}


double _POWERSUM_B[20] = {8,18,44,114};
double mvfNeumaierPowersum(int n, double x[])
/*
  Global minimum f* = 0.
  Domain:  for default values _POWERSUM_B, x[i] in [0,4].
           and x* = (1,2,2,3).
*/
{
  int i, k;
  double s1, sum;

  sum = 0.0;
  for (k = 0; k < n; k++) {
     s1 = 0.0;
     for (i = 0; i < n; i++) {
        s1 += pow(x[i], k+1);
     }
     sum += sqr(s1 - _POWERSUM_B[k]);
  }
  return sum;
}


double mvfNeumaierTrid(int n, double x[])
/*
  Suitable bounds for a bound constrained version would be [-n^2,n^2] for each component
  The solution is x_i=i(n+1-i) with f(x)=-n(n+4)(n-1)/6.
*/
{
  int i;
  double s1, s2;

  s1 = 0.0;
  for (i = 0; i < n; i++) {
    s1 += sqr(x[i] - 1);
    if (i) s2 += x[i] * x[i-1];
  }
  return s1 - s2;
}


double mvfOddsquare(int n, double *x)
/*
- Dimension: n = 5, 10
- Domain: | x[i] | <= 5 *pi
*/
{
double c = 0.2;
double a[] = {1, 1.3, 0.8, -0.4, -1.3, 1.6, -0.2, -0.6, 0.5, 1.4};
double d1 = dist2(n, x, a);
double d2 = sqr(distInf(n, x, a));

return -exp(-cos(d2) / (2*M_PI)) * cos( M_PI * d2) * (1.0 + c * (d1) / (d1 + 0.01));
}
```

```
double mvfPaviani(int n, double *x)
{
  double a,b,sum=0.0, mul=1.0 ;
  int i;

  for (i=0 ; i<n ; i++) {
    a=log(x[i]-2.0) ; b=log(10.0-x[i]) ;
    sum+= a*a + b*b ;
    mul*= x[i] ;
  }
  return sum - pow(mul,0.2) ;
}

double mvfPlateau(int n, double *x)
/*
Dimension 5
Bounds:   |x[i]| < 5.12 for i = 0 to 4
Reference:
   L. Ingber, "Simulated Annealing, Practice Vs. Theory",
              Journal Math. Computing and Modelling, V18,N11,
              D1993, 29-57
*/
{
  int i;
  double sum = 0.0;

  for (i = 0; i < 5; i++) {
    sum += floor(x[i]);
  }
  return 30.0 + sum;
}

double mvfPowell(int n, double *x)
{
  int j;
  double sum;

  sum = 0.0;
  for (j = 0; j < n/4; j++) {
sum +=  sqr(x[4*j-3] + 10 * x[4*j-2])
          + 5 * sqr(x[4*j-1] - x[4*j])
          + pow(x[4*j-2] - 2 * x[4*j-1], 4)
  + 10 * pow(x[4*j - 3] - x[4*j], 4);
  }
  return sum;
}

double mvfQuarticNoiseU(int n, double *x)
/*
-Domain: | x_i | < 1.28
-Global minimum is 0.0 at x_i = 0.00
*/
{
```

```
int i;
double sum, t;

sum = 0.0;
for (i = 0; i < n; i++) {
t = x[i];
t = t*t;
sum += t*t + rndUniform(0.0, 1.0);
}
return sum;
}

double mvfQuarticNoiseZ(int n, double *x)
/*
-Test optimization function. Range is | x_i | < 1.28
-Global minimum is 0.0 at x_i = 0.00
*/
{
  int i;
double sum, t;

sum = 0.0;
for (i = 0; i < n; i++) {
t = x[i];
t = t*t;
sum += (i*t*t + rndZ());
}
return sum;
}


double mvfRana(int n, double *x)
/*
-   Domain = | x_i | < 500
*/
{
  int i;
  double t1, t2, sum;

  sum = 0.0;
  for (i = 0; i < n-1; i++) {
    t1 = sqrt(fabs(x[i+1] + x[i] + 1.0));
    t2 = sqrt(fabs(x[i+1] - x[i] + 1.0));
    sum += (x[i+1] + 1.0) * cos(t2) * sin(t1) + cos(t1) * sin(t2) * x[i];
  }
  return sum;
}


double mvfRastrigin(int n, double *x)
/*
-   Domain = | x_i | < 5.12
-   Global minimum is 0.0 at x_i = 0.00
*/
```

```
{
int    i;
double sum, t;

if (n > 20) {
n = 20;
}

sum = 0.0;
for (i = 0; i < n; i++) {
t   = x[i];
sum += t*t - cos(2.0 * M_PI * x[i]);
}
return sum + n*10;
}

double mvfRastrigin2(int n, double *x)
/*
-   n = 2 always
-   Domain =  | x_i | < 5.12
-   Global minimum is 0.0 at x_i = 0.00
*/
{
double t;

t = x[0];
return 2.0 + t*t - cos(18 * t) - cos(18 * x[1]);
}

double mvfRosenbrock(int n, double * x)
{
   double a = x[1]-x[0]*x[0] ;
   double b = 1.0-x[0] ;

   return 100.0*a*a + b*b ;
}

double mvfSchaffer1(int n, double *x)
/*
- Dimension 2
- Global minimum 0 at (0)
- Domain: |x[i]| <= 100
*/
{
return 0.5 + (sqr(sin(sqrt(sqr(x[0]) + sqr(x[1])))) - 0.5) / sqr(1.0 + 0.001 * sqr(x[0])
}

double mvfSchaffer2(int n, double *x)
/*
- Dimension 2
- Global minimum = 0 at (0).
*/
{
return pow(x[0]*x[0] + x[1]*x[1], 0.25)
```

```
  * (50.0 * pow(x[0]*x[0] + x[1]*x[1], 0.1) + 1.0);


}

double mvfSchwefel1_2(int n, double *x)
/*
-n = 100 for testing
-Domain:  | x_i  | < 10.00
-Global minimum is 0.0 at x_i = 0.00
*/
{
  int i, j;
  double sum, sum1;

  sum = 0.0;
  for (i = 0; i < n; i++) {
    sum1 = 0.0;
    for (j = 0; j < i; j++) {
      sum1 += x[i];
    }
    sum += sum1 * sum1;
  }
  return sum;
}

double mvfSchwefel2_21(int n, double *x)
/*
-n = 100 for testing
- Domain:  | x_i  | < 10.000
-Global minimum is 0.0 at x_i = 0.00
*/
{
  int    i;
  double s,t;

  s = fabs(x[0]);
  for (i = 1; i < n; i++) {
 t = fabs(x[i]);
 if (t > s) {
  s = t;
 }
  }
  return s;
}

double mvfSchwefel2_22(int n, double *x)
/*
-n = 30 for testing
-Domain:  | x_i  | < 10.00
-Global minimum is 0.0 at x_i = 0.00
*/
{
  int i;
  double sum = 0.0;
  double prod = 1.0;

  for (i = 0; i < n; i++) {
    sum  += fabs(x[i]);
    prod *= fabs(x[i]);
  }
  return sum + prod;
}

double mvfSchwefel2_26(int n, double *x)
/*
-   Domain is | x_i | < 500
-   Global minimum at fmin = -122569.5 at  x_i = 420.9687
*/
{
int    i;
double sum;

sum = 0.0;
for (i = 0; i < n; i++) {
sum += x[i] * sin(sqrt(x[i]));
}
return - sum;
}

double mvfShekel2(int n, double *x)
/*
-   n  = 2 always
-   Domain:   | x_i | <= 65.536
-   Minimum 1 at x = (-32, -32)
*/
{
int    j;
double s, t0, t1;

static double a[2][25] = {
  {-32,-16,0,16,32,-32,-16,0,16,32,-32,-16,0,
   16,32,-32,-16,0,16,32,-32,-16,0,16,32},
  {-32,-32,-32,-32,-32,-16,-16,-16,-16,-16,0,
   0,0,0,0,16,16,16,16,16,32,32,32,32,32}
};

s = 0.0;
for (j = 0; j < 25; j++) {
t0 = x[0] - a[0][j];
t1 = x[1] - a[1][j];
t0 = (t0 * t0 * t0);
t0 *= t0 * t0;
t1 = (t1 * t1 * t1);
t1 = t1 * t1;
s  += 1.0/ ((double) j + t0 + t1);
}
return 1.0 / (1.0/500.0  + s);
```

```
}

static double afox4[10][4]={ { 4,4,4,4 } ,
                         { 1,1,1,1 } ,
                         { 8,8,8,8 } ,
                         { 6,6,6,6 } ,
                         { 3,7,3,7 } ,
                         { 2,9,2,9 } ,
                         { 5,5,3,3 } ,
                         { 8,1,8,1 } ,
                         { 6,2,6,2 } ,
                         {7,3.6,7,3.6} };

static double cfox4[10]= {.1,.2,.2,.4,.4,.6,.3,.7,.5,.5 };

double mvfShekelSub4(int m, double *x)
{
  double R=0.0, S;
  int i,j;

  for(i=0;i<m;i++) {
    S=0;
    for (j=0;j<4;j++) S+=pow(x[j]-afox4[i][j],2);
    R-=1/(S+cfox4[i]);
  }
  return R;
}

double mvfShekel4_5(int n, double * x)
/*
-   Domain   0 <= x_j <=  10
-   Global minimum fmin = -10.1532,
-     at X = (4.00004, 4.00013, 4.00004, 4.00013)
-   Number of local optimizers = 5
*/
{
  return mvfShekelSub4(5, x);
}

double mvfShekel4_7(int n, double * x)
/*
-   Domain   0 <= x_j <=  10
-Global minimum fmin = -10/4029 at
-      (4.00057,4.00069,3.99949,3.99961) \
-   Number of local minimum = 7
*/
{
  return mvfShekelSub4(7, x);
}

double mvfShekel4_10(int n, double * x)
/*
-   Domain   0 <= x_j <=  10
-   Global minimum fmin = -10.5364 at
-    (4.00075,4.00059,3.99966,3.99951)
-   Number of local minimum = 10
*/
{
  return mvfShekelSub4(10, x);
}

double mvfShekel10(int n, double *x)
{
int    i;
double sum;

cfox10[2] = 0.10;
sum  = 0.0;
n    = 10; /* force checked */
for (i = 0; i < 30; i++) {
sum += 1.0/(dist2(n, x, afox10[i]) + cfox10[i]);
}
return sum;
}

double mvfShubert(int n, double * x)
/*
-   Domain  |x| <= 10.0
-   Number of local minimum = 400
-   Global minimum fmin = -24.062499 at the ff. points
-    (-6.774576, -6.774576), ..., (5.791794, 5.791794)
*/
{
   return -sin(2.0*x[0]+1.0)
          -2.0*sin(3.0*x[0]+2.0)
          -3.0*sin(4.0*x[0]+3.0)
          -4.0*sin(5.0*x[0]+4.0)
          -5.0*sin(6.0*x[0]+5.0)
          -sin(2.0*x[1]+1.0)
          -2.0*sin(3.0*x[1]+2.0)
          -3.0*sin(4.0*x[1]+3.0)
          -4.0*sin(5.0*x[1]+4.0)
          -5.0*sin(6.0*x[1]+5.0);
}

double mvfShubert2(int n, double *x)
{
   int i;
   double  s1, s2;

   s1 = s2 = 0.0;
   for (i = 0; i < 5; i++) {
    s1 += (i+1) * cos((i+2)* x[0] + i+1);
    s2 += (i+1) * cos((i+2)* x[1] + i+1);
  }
   return s1 + s2;
}
```

```
double  mvfShubert3(int n, double *x)
/*
-   Domain  |x| <= 10.0
-   Number of local minimum = 400
-   Global minimum fmin = -24.062499 at the ff. points
-    (-6.774576, -6.774576), ..., (5.791794, 5.791794)
*/
{
  int i, j;
  double  s;

  n = 2;  // force checked
  s = 0.0;

  for (i = 0; i < n; i++) {
    for (j = 0; j < 5; j++) {
      s += (j+1)*sin((j+2)*x[i] + (j + 1));
    }
  }
  return -s;
}

double mvfSphere(int n, double *x)
/*
- Domain: | x[i] | < 10
- Global minimum: 0 at x[i] = 0
*/
{
  int i;
  double sum;

  sum = 0.0;
  for (i = 0; i < n; i++) {
    sum += x[i]*x[i];
  }
  return sum;
}


double mvfSphere2(int n, double *x)
/*
- Domain: | x[i] | < 10
- Global minimum: 0 at x[i] = 0
*/
{
  int i, j;
  double s1, s2;

  s1 = 0.0;
  s2 = 0.0;
  for (i = 0; i < n; i++) {
    s2 += x[i];
    s1 += s2 * s2;
  }
  return s1;
}


double mvfStep(int n, double *x)
/*
-   n           In: number of variables, 30
-   x           In: point of evaluation
-   Domain: | x_i  | < 100.0
-   Global minimum is 0.0 at x_i = 0.00
*/
{
    int i;
    double y;
    double sum;

    sum = 0.0;
    for (i = 0; i < n; i++) {
      y = floor(x[i] + 0.5);
      sum += y*y;
    }
    return sum;
}


double mvfStretchedV(int n, double *x)
/*
- Dimension:
- Domain:  | x_i | <= 10.0
*/
{
  int i = 0;
  double sum;
  double t;

  sum = 0.0;
  for (i = 0; i < n - 1; i++) {
    t = x[i+1]*x[i+1] + x[i]*x[i];
    sum += pow(t, 0.25) * pow(sin(50.0 * pow(t, 0.1) + 1.0), 2);
  }
  return sum;
}

double mvfSumSquares(int n, double *x)
/*
Dimension: n
Domain: | x[i] | <= 10
Global minimum: 0 at x[i] = 0
*/
{
  int i;
  double sum = 0;
  for (i = 0; i < n; i++) {
sum += i * x[i]*x[i];
  }
  return sum;
}

double mvfTrecanni(int n, double *x)
/*
  Dimension: 2
Domain: | x[i] | <= 5
Global minimum: 0 at (0,0) and (-2,0)
*/
{
    return ((x[0]  +  4) * x[0] + 4)  * x[0] * x[0];
}


double mvfTrefethen4(int n, double *x)
/*
- Dimension: 2
- Domain: -6.5 < x[0] < 6.5,
-         -4.5 < x[0] < 4.5
-
- Global minimum: -3.30686865  at
-         (-0.0244031, 0.2106124)
*/
{
    return  exp(sin(50.0 * x[0])) +
            sin(60.0 * exp(x[1])) +
            sin(70.0 * sin(x[0])) +
            sin(sin(80*x[1])) -
            sin(10.0 * (x[0] + x[1])) +
            1.0/4.0 * (x[0]*x[0] + x[1] * x[1]);
}

double mvfXor(int n, double* x)
/*
- Dimension 9
-
*/
{
  return    1/ sqr(1 + exp(- x[6] / (1+exp(-x[0]-x[1]-x[4])) - x[7] / (1+exp(-x[2]-x[3]-
    + 1/ sqr(1 + exp(-x[6]/(1 + exp(-x[4])) - x[7]/(1+exp(-x[5])) - x[8]))
    + 1/ sqr(1 - 1/(1+exp(x[6]/(1 + exp(-x[0]-x[4])))  - x[7]/(1 + exp(-x[3] - x[5])) -
    + 1/ sqr(1 - 1/(1+exp(x[6]/(1 + exp(-x[1]-x[4])))  - x[7]/(1 + exp(-x[3] - x[5])) -
}

double mvfWatson(int n, double *x)
{
  int    i, j;
  double a;
  double s;
  double t, s1, s2;
  s += x[0] * x[0];

  for (i = 0; i < 30; i++) {
    a = i / 29.0;
    s1 = 0.0;
    s2 = 0.0;
    for (j = 0; j < 5; j++) {
        s1 += (j+1)* pow(a, j) * x[j+1];
    }
    for (j = 0; j < 6; j ++) {
        s2 += pow(a, j) * x[j];
    }
    t = (s1 - s2 * s2 - 1);
    s += t * t;
  };
  return s;
}


double mvfZettl(int n, double *x)
{
  double t = (x[0] * x[0] + x[1] * x[1] - 2 * x[0]);
  return t * t   + 0.25 * x[0];
}

double _Zh1(double *x)
{
return 9.0 - x[0] - x[1];
}

double _Zh2(double *x)
{
return sqr(x[0] - 3.0) + sqr(x[1] - 2.0) - 16.0;
}

double _Zh3(double *x)
{
return x[0] * x[1] - 14.0;
}

double _Zp(double x)
{
return 100.0 * (1 + x);
}

double mvfZimmerman(int n, double *x)
/*
- Dimension: 2
- Global minimum  at
- Domain: 0 <= x[i] <= 100.0
*/
{
double px[] = {
_Zh1(x),
_Zp(_Zh2(x)) * sgn(_Zh2(x)),
                _Zp(_Zh3(x)) * sgn(_Zh3(x)),
_Zp(-x[0]) * sgn(x[0]),
_Zp(-x[1]) * sgn(x[1])
  };
return dmax(5, px);
}
```

B Random number routines

The MVF library requires some statistical functions which are found in the file `rnd.c`. The routines uses the Mersenne Twister random number generator developed in reference [MN]. For the latest versions, consult the web site

`http://www.math.keio.ac.jp/ matumoto/emt.html`

- rndSeed(seed)
  Seeds the random number generator.
- rnd()
  Returns a uniform random number in $[0, 1]$.
- rnd0(N)
  Returns a random integer in $[0, N - 1]$.
- rndInt(a, b)
  Returns a random integer in $[a, b]$.
- rndUniform(a, b)
  Returns a uniform random number in the interval $[a, b]$.
- rndZ()
  Returns a standardized normal random variate with mean 0 and variance 1.
- rndNormal(mu, std)
  Returns a normal random variate with given mu and standard deviation std.

B.1 rnd.h header file for random number generator

```
#ifndef _RND_H
   #define _RND_H

void     rndSeed(unsigned long seed);
double   rnd(void);
int      rnd0(int N);
int      rndInt(int a, int b);
double   rndUniform(double a, double b);
double   rndZ(void);
double   rndNormal(double mu, double std);
#endif
```

B.2 rnd.c, basic random number routines

```
/*
File           rnd.c

Remarks.
  This file needs updating with the latest versions of
  the Mersenne Twister.
*/

#include <math.h>
#include <signal.h>
#include <string.h>
#include <time.h>

#include <sys/time.h>
#include <unistd.h>

#include <stdlib.h>
#include <stdio.h>

#include "rnd.h"

/* ********** Start of Mersenne Twister **********************
This section contains code for the Mersenne Twister
random number generator. Please refer to the web site

  http://www.math.keio.ac.jp/~matumoto/emt.html

for the latest versions and more information. The authors of
Mersenne Twister have permitted the use of MT for commercial
purposes.

   http://www.math.keio.ac.jp/~matumoto/eartistic.html
*/

/* Period parameters */
#define _N 624
#define _M 397
#define MATRIX_A 0x9908b0df   /* constant vector a */
#define UPPER_MASK 0x80000000 /* most significant w-r bits */
#define LOWER_MASK 0x7fffffff /* least significant r bits */

/* Tempering parameters */
#define TEMPERING_MASK_B 0x9d2c5680
#define TEMPERING_MASK_C 0xefc60000
#define TEMPERING_SHIFT_U(y)  (y >> 11)
#define TEMPERING_SHIFT_S(y)  (y << 7)
#define TEMPERING_SHIFT_T(y)  (y << 15)
#define TEMPERING_SHIFT_L(y)  (y >> 18)

static unsigned long  mt[_N]; /* the array for the state vector  */
static int mti=_N+1; /* mti==N+1 means mt[N] is not initialized */
```

```
/* Initializing the array with a seed */
void rndSeed(unsigned long seed)
{
int i;

for (i=0;i<_N;i++) {
mt[i] = seed & 0xffff0000;
seed = 69069 * seed + 1;
mt[i] |= (seed & 0xffff0000) >> 16;
seed = 69069 * seed + 1;
}
mti = _N;
}

void lsgenrand(unsigned long seed_array[])
/* the length of seed_array[] must be at least N */
{
int i;

for (i=0;i<_N;i++)
mt[i] = seed_array[i];
mti=_N;
}

double rnd(void)
{
unsigned long y;
static unsigned long mag01[2]={
0x0, MATRIX_A };
/* mag01[x] = x * MATRIX_A  for x=0,1 */

if (mti >= _N) { /* generate N words at one time */
int kk;

if (mti == _N+1)   /* if sgenrand() has not been called, */
rndSeed(4357); /* a default initial seed is used   */

for (kk=0;kk<_N-_M;kk++) {
y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
mt[kk] = mt[kk+_M] ^ (y >> 1) ^ mag01[y & 0x1];
}
for (;kk<_N-1;kk++) {
y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
mt[kk] = mt[kk+(_M-_N)] ^ (y >> 1) ^ mag01[y & 0x1];
}
y = (mt[_N-1]&UPPER_MASK)|(mt[0]&LOWER_MASK);
mt[_N-1] = mt[_M-1] ^ (y >> 1) ^ mag01[y & 0x1];

mti = 0;
}

y = mt[mti++];
y ^= TEMPERING_SHIFT_U(y);
y ^= TEMPERING_SHIFT_S(y) & TEMPERING_MASK_B;
y ^= TEMPERING_SHIFT_T(y) & TEMPERING_MASK_C;
y ^= TEMPERING_SHIFT_L(y);

return ( ((double)y + 1.0) * 2.3283064359965952e-10 );
/* reals: (0,1)-interval */
/* return y; */ /* for integer generation */
}
/* ****** End of Mersenne Twister  */




int rnd0(int N)
/* returns a random integer from 0 to N-1 */
{
return (int) (rnd() * N);
}

int rndInt(int a, int b)
{
return (int) (rnd() * (b - a + 1)) + a;
}

double rndUniform(double a, double b)
{
return a + rnd() * (b - a);
}

double rndZ(void)
{
  /* Unoptimized version */
return sqrt(-2.0*log(rnd())) * sin(2*M_PI*rnd());
}

double rndNormal(double mu, double std)
{
  return  rndZ() * std + mu;
}
```

C Creating Python bindings with SWIG

Swig is an interface generator which allows python code to call C routines and vice-versa. Here are the steps in making a shared dll library in Linux and accessing the C functions from the Python prompt.

1. Write the interface file `mvf.i`

   The declarations of the interface file defines mapping between python data types and C datatypes. Here is the content of the interface file.

   ```
   %module mvf

   %{
     #include "mvf.h"
     #include "rnd.h"
   %}
   %include "carrays.i"

   %array_class(double, doubleArray);

   %include mvf.h
   %include rnd.h
   ```

   Create the wrapper files by issuing the command

   `swig -python mvf.i`

   The output of this command is to create the files `mvf.py` and `mvf_wrap.c`.

2. Create the shared dll.

   Compile the file mvf.c together with the support wrapper files and rnd.c. Be sure that underscores are prepended to the shared file name. For convenience, we create a shell script file.

   ```
   swig -python mvf.i
   gcc -Wall -fpic -c mvf.c rnd.c mvf_wrap.c -I/usr/local/include/python2.3
   gcc -Wall -shared mvf_wrap.o mvf.o rnd.o -lm -O3 -o _mvf.so
   ```

3. Import the module in Python

   From the command line interpreter, you can now access the shared functions.

   ```
   [toto@adorio mvf]$ python
   Python 2.3.3 (#1, Feb  4 2004, 13:34:29)
   [GCC 3.3.1 (Mandrake Linux 9.2 3.3.1-2mdk)] on linux2
   Type "help", "copyright", "credits" or "license" for more information.
   >>> import mvf
   >>> dir(mvf)
   >>> dir(mvf)
   ['_Zh1', '_Zh2', '_Zh3', '_Zp', '__builtins__', '__doc__', '__file__',
   '__name__', '_mvf','_newclass', '_object', '_swig_getattr', '_swig_setattr',
   'charArray', 'charArrayPtr', 'charArray_frompointer', 'dist2', 'distInf',
   'dmax', 'doubleArray', 'doubleArrayPtr','doubleArray_frompointer',
   'floatArray', 'floatArrayPtr', 'floatArray_frompointer','intArray',
   'intArrayPtr', 'intArray_frompointer', 'mvfAckley', 'mvfBeale',
   'mvfBohachevsky1', 'mvfBohachevsky2', 'mvfBooth', 'mvfBoxBetts',
   'mvfBranin', 'mvfBranin2', 'mvfCamel3', 'mvfCamel6', 'mvfChichinadze',
   ```

```
'mvfCola', 'mvfColville', 'mvfCorana', 'mvfEasom', 'mvfEggholder',
'mvfExp2', 'mvfFraudensteinRoth', 'mvfGear', 'mvfGeneralizedRosenbrock',
'mvfGoldsteinPrice', 'mvfGriewank', 'mvfHansen', 'mvfHartman3',
'mvfHartman6', 'mvfHimmelblau', 'mvfHolzman1', 'mvfHolzman2',
'mvfHosaki', 'mvfHyperellipsoid', 'mvfKatsuuras', 'mvfKowalik',
'mvfLangerman', 'mvfLennardJones', 'mvfLevy', 'mvfMatyas', 'mvfMcCormick',
'mvfMichalewitz', 'mvfNeumaierPerm', 'mvfNeumaierPerm0',
'mvfNeumaierPowersum', 'mvfNeumaierTrid', 'mvfOddsquare', 'mvfPaviani',
'mvfPlateau', 'mvfPowell', 'mvfQuarticNoiseU', 'mvfQuarticNoiseZ',
'mvfRana', 'mvfRastrigin', 'mvfRastrigin2', 'mvfRosenbrock', 'mvfSchaffer1',
'mvfSchaffer2', 'mvfSchwefel1_2', 'mvfSchwefel2_21', 'mvfSchwefel2_22',
'mvfSchwefel2_26', 'mvfShekel10', 'mvfShekel2', 'mvfShekel4_10',
'mvfShekel4_5', 'mvfShekel4_7', 'mvfShekelSub4', 'mvfShubert',
'mvfShubert2', 'mvfShubert3', 'mvfSphere', 'mvfStep', 'mvfStretchedV',
'mvfSumSquares', 'mvfTrecanni', 'mvfTrefethen4', 'mvfXor', 'mvfZimmerman',
'prod', 'rnd', 'rnd0', 'rndInt', 'rndNormal', 'rndSeed', 'rndUniform',
'rndZ', 'setNeumaierPerm', 'setNeumaierPerm0', 'sgn', 'sqr', 'types']
>>> a = mvf.doubleArray(2)
>>> a[0] = 1; a[1] = 2
>>> mvf.mvfSphere(2, a)
5.0
>>>
```

The names for the C functions are prefixed with `'mvf'`. In Python, this can be a nuisance for touch typists, so we can import the module using `'from mvf import *'` instead of `'import mvf'` as we had done in the above example.