

# Best Optimized Rational Heteronomial Model Selection

by  
Namir Clement Shammas

## Contents

1/Introduction.....	2
2/Using Optimization Alorithms.....	3
3/ The Excel Sheet .....	3
4/The Random Search Optimization.....	6
5/Nested Random Search Optimization.....	13
6/Particle Swarm Optimization.....	20
7/The Cuckoo Search Algorithm (CSA).....	32
8/Hunter Prey Optimization (HPO) Algorithm.....	39
9/The Differential Evolution (DE) Optimization.....	44
10/The Enhanced Vibrating Particles System.....	50
11/The Ant Colony Optimization (ACO) .....	54
12/Arithmetic Optimization Algorithm (AOA). .....	60
Conclusion .....	64
Appendix A .....	64
The Particle Swarm Optimization (PSO).....	64
The Cuckoo Search Optimization (CSO) Function .....	67
The Hunter Prey Optimization (HPO) Function.....	71
The Differential Evolution (DE) Optimization Function .....	73
The Vibrating Particles System (EVPS) Optimization Function.....	76
The Ant Colony Optimization (ACO) Algorithm Function.....	79
The Arithmetic Optimization Algorithm (AOA) Function.....	82

## 1/Introduction

This study expands on a previous three-part study for selecting the best rational heteronomial. Here is the general form of the rational heteronomials I will handle:

$$Y^{py} = (a + b_1 * X_1^{px11} + b_2 * X_2^{px12} + b_3 * X_1^{px13} * X_2^{px14}) / (1 + c_1 * X_1^{px211} + c_2 * X_2^{px22} + c_3 * X_1^{px23} * X_2^{px24}) \quad (1)$$

In this study, I use optimization algorithms (instead of nested loops) to find the best powers to which variables are raised in a rational heteronomial. This approach eliminates the nested loops, shortens the program code, and makes it more flexible. That's the **good news**! The **bad news** is that the optimization methods I will be using still require defining the ranges (a.k.a. trust regions) for the different powers, as well as appropriate maximum number of iterations and search population size. These parameters cannot be easily guessed ahead of time. Trial and error may be needed. For each set of powers, we perform multiple linearized regression to calculate the values of regression coefficients  $a$ ,  $b_i$ , and  $c_j$ . The multiple linearized model is:

$$Y^{py} = a + b_1 * X_1^{px11} + b_2 * X_2^{px12} + b_3 * X_1^{px13} * X_2^{px14} - c_1 * Y^{py} * X_1^{px211} - c_2 * Y^{py} * X_2^{px22} - c_3 * Y^{py} * X_1^{px23} * X_2^{px24} \quad (1b)$$

Notice that the terms in the denominator of equation (1) are now multiplied by  $Y^{py}$  and these terms are also **subtracted** from the terms in the numerator of equation (1).

☛ In this study I will be using the values for the dependent variable without any transformations, by default. The main reason is to avoid runtime errors in calculations. The model in equation (1) is not easy to optimize when using transformations for the dependent variable. Predicting values for the dependent variable may generate complex values leading to runtime errors. My initial suspicion, which turns out to be true, is that there are local minimums that compete with the global minimum. The code that I provide will allow you to further experiment with transformations of the dependent variable. **Do so at your own risk!**

## 2/Using Optimization Algorithms

In this study I use optimization algorithms to find the best powers of equation (1) instead of employing nested loops. This approach simplifies the code. Of course, I will be using optimization functions stored in separate code files. The code for the optimization functions is listed in Appendix A. Discussing how each optimization algorithm works is beyond the scope of this study (and to prevent the study's page count from getting out of hand). I encourage you to search the Internet to learn how each algorithm works. Keep in mind that the performance and efficiency of the various optimization algorithms will vary. The optimization functions used are:

- Random search optimization.
- Nested random search optimization.
- The particle swarm optimization (PSO) algorithm.
- Cuckoo search optimization (CSO).
- Hunter prey optimization (HPO).
- Differential evolution (DE) optimization.
- Vibrating particles system (EVPS) optimization.
- Ant colony optimization (ACO) algorithm.
- Arithmetic optimization algorithm (AOA).

☛ Since all the optimization algorithms in this study use random numbers, the results will always vary. And as such, the quality of the results may well range between very good and somewhat poor! The quality of the results is affected by the ranges of the powers you supply in the Excel files (see the next section). Thus, you may need to redo the calculations using multiple different sets of ranges. This process will require patience!

## 3/ The Excel Sheet

The study uses Excel workbooks to provide input and output data. These files provide the data for the curve fitting, the ranges for the optimized variables, and the selection of the variables for the numerator and denominator parts of the rational heteronomial.

Any Excel workbook used in this study has the following sheets:

- The **Data** sheet has the following columns (see Figure 3.1):

- Column A contains the values for the dependent variable Y. The column has the header Y in the first row. The values for the dependent variable appear in row 2 and below.
- Column B and above contain the tabulated values for the independent variables. The columns have the headers X1, X2, and so on in the first row. The values for the independent variables appear in row 2 and below.
- The **Transf** sheet contains information for the transformation ranges. It has the following columns (see Figures 3.2):
  - Column A has the header Y in row 1. Rows 3, and 4 contain the minimum and maximum power used with variable Y. The content of row 2 is ignored.
  - Column B has the header X1 in row 1. Rows 2, 3, and 4 contain the variable selection index, and minimum and maximum power used with variable X1. A positive variable index value indicates that the variable is in the numerator part of the rational heteronomial. A negative variable index value indicates that the variable is in the denominator part of the rational heteronomial.
  - Column C and beyond have values like those in column B for the other independent variables. The number of columns populated in B and beyond represents the total number of terms. The number of columns with positive variable indices represents the number of terms in the numerator part. The number of columns with negative variable indices represents the number of terms in the denominator part. To make it easier to read the results in sheet **Results**, separate the variables with positive variable indices from the ones with negative indices.
- Sheet **Results** is the placeholder for the regression results. Row 2 contains the powers for the dependent variable and the dependent variables. Row 3 contains the regression coefficients.
- Sheet **Project** contains a copy of the input data, the calculated values of the dependent variable (based on the best model), and percentage errors,

The data in Figure 3.1 (which will be used with the various optimization algorithms) are based on the following equation:

$$Y = (5 + X_1^2 + X_2^{2.5} + X_3^2 + X_4^{1.5}) / (1 + X_1^{1.25} + X_2^{1.3} + X_3^{1.15} + X_4^{1.15}) \quad (2)$$

Y	X1	X2	X3	X4
8.2997697	15	14	7	55
9.7403605	34	12	3	56
9.8171256	42	7	20	81
10.652665	18	14	18	29
11.571577	29	7	39	57
12.840639	42	13	25	46
13.503839	49	10	43	89
14.193163	69	2	12	98
14.564704	51	6	38	54
14.571504	13	6	56	72
15.068463	63	14	6	73
16.529971	45	9	49	44
16.644062	80	5	26	99
17.438801	48	14	34	19
17.668513	65	19	18	62
17.786412	69	4	44	67
17.957852	19	23	43	63
18.892333	40	4	77	99
19.163513	42	4	74	85
19.191481	63	15	14	22
19.281362	37	26	42	73
19.745319	39	29	27	82
20.024114	52	8	42	3
20.31761	32	22	65	78
20.813814	43	30	40	92
21.526468	80	17	1	49
21.664196	4	8	84	98
21.766129	47	20	52	29
22.077494	56	21	9	14
22.097747	33	19	75	70

*Figure 3.1. A partial view of the first 30 data points in sheet **Data** in the data1\_44xxx.xlsx Excel workbooks.*

I purposely use error-free data in the sheet **Data**. This approach allows me to assess the quality of the results using different power ranges and different optimization algorithms. A very good optimization algorithm (which locates the global minimum) will obtain results that match very closely the powers and coefficients in equation (2).

Y	X1	X2	X3	X4	YX1	YX2	YX3	YX4
0	1	2	3	4	-1	-2	-3	-4
1	1	1	1	1	1	1	1	1
1	3	3	3	3	3	3	3	3

*Figure 3.2. The **Transf** sheet in the data1\_44xxx.xlsx Excel workbooks.*

## 4/The Random Search Optimization

I will start with the least efficient and most free-to-search algorithm. The random search algorithm is very easy to code and is relatively fast. Thus, you can perform a high number of iterations, like a million! The test program file test\_FlexRand1.m contains the following code that tests the data in Excel workbook data1\_44FlexRand1. Here is the code for the test program test\_FlexRand1.m:

```
clc
close all
clear

outFile = "bestrathetFlexRand1_";
% You may comment the next statement
delete(strcat(outFile,"*.txt"));

dt = datetime('now','TimeZone','local','Format','y-MMM-d HH_mm_ss');
dtstr = string(dt);
XlFile = strcat(pwd,'\Data1_44FlexRand1.xlsx');
outFile = strcat(outFile,dtstr,".txt");

[bestMd11, bestPwrs1] = bestrathetFlexRand(XlFile, outFile);

clear GLOBAL
fprintf("Done!\n");
```

The test program performs a test using Excel file Data1\_44FlexRand1.xlsx and echoes the console output to diary file bestrathetFlexRand1\_2024-Dec-10 15\_45\_08.txt.

The test program calls function `besttrathetFlexRand()` listed next:

```
function [bestMdl, bestPwrs] = ...
    besttrathetFlexRand(XlFile, outFile, MaxIters, StallItersToStop)

    global mdl
    global xdata
    global ydata
    global xVarIdx

    warning("off")

    diary(outFile)

    if nargin < 4, StallItersToStop = 100000; end
    if nargin < 3, MaxIters = 1000000; end

    dt = datetime('now','TimeZone','local','Format','y-MMM-d HH_mm_ss');
    dtstr = string(dt);

    fprintf("Please wait ...\n\n");
    fprintf("%s\n", dtstr);
    fprintf("Program besttrathetFlexMatlabPS01\n\n");
    fprintf("Excel file used is %s\n", XlFile);
    fprintf("Diary file used is %s\n", outFile);
    xyData = readmatrix(XlFile,'Sheet','Data');
    [maxrows,nVars] = size(xyData);
    ydata = xyData(:,1);
    xdata = xyData(:,2:end);
    xyTransf = readmatrix(XlFile,'Sheet','Transf');
    [~,nVars] = size(xyTransf);
    nXvars = nVars - 1;
    xVarIdx = xyTransf(1,2:end);
    Lb = xyTransf(2,:);
    Ub = xyTransf(3,:);

    bestSSE = 1e99;
    lastIter = 0;
    for iter=1:MaxIters
        pop = Lb + (Ub - Lb).*rand(1,nVars);
        dummy= myFit(pop);
        if mdl.SSE < bestSSE
            gBestX = pop;
            bestSSE = mdl.SSE;
            fprintf("Iter: %d, ", iter);
            fprintf("Best Fx =%e, X=[" , bestSSE);
            fprintf("%f, ", gBestX(1:nVars-1));
            fprintf("%f]\n", gBestX(nVars));
            lastIter = iter;
        end
        if (iter - lastIter) > StallItersToStop, break; end
    end
    fprintf("\nResults (powers/coefficients)\n")
    bestPwrs = gBestX
```

```

fprintf("Regression model\n")
dummy = myFit(gBestX);
format long
AdjR2 = mdl.Rsquared.Adjusted;
fprintf("Adj R-sqr = %14.8f\n", AdjR2);
writematrix(AdjR2, XlFile, 'Sheet', 'R2', 'Range', 'A2:A2');
bestMdl = mdl

res = zeros(2,nVars+1);
res(1,1)= gBestX(1);
res(1,3:end) = gBestX(2:end);
res(2,2:end) = mdl.Coefficients(:,1);
writematrix(res(1,1), XlFile, 'Sheet', 'Results', 'Range', 'A2:A2');
writematrix(res(1,3:end), XlFile, 'Sheet', 'Results', 'Range', 'C2:Z2');
writematrix(res(2,2:end), XlFile, 'Sheet', 'Results', 'Range', 'B3:Z3');

% now do the projections
ycalc = project(res);
percErr = 100.*(ycalc-ydata)./ydata;
ProjMat = [ydata xdata ycalc percErr];
writematrix(ProjMat, XlFile, 'Sheet', 'Project', 'Range', strcat("A2:Z",
num2str(length(ydata))));

format short
diary off

beepit(4)
end

function beepit(nTimes)
    for i=1:nTimes
        beep;
        pause(1);
    end
end

function r = myFit(coeff)
    global mdl
    global xdata
    global ydata
    global xVarIdx

    if coeff(1) >= 0
        y = ydata.^coeff(1);
    else
        y = 1./ydata.^abs(coeff(1));
    end
    X = [];
    for i = 1:length(xVarIdx)
        j = abs(xVarIdx(i));
        if coeff(i+1) >= 0
            x = xdata(:,j).^coeff(i+1);
        else
            x = 1./xdata(:,j).^abs(coeff(i+1));
        end
        if xVarIdx(i) < 0
            x = -1.*y.*x;

```



```

        end
        X = [X x];
    end
    mdl = fitlm(X,y,'RobustOpts','on');
    r = mdl.SSE;
    if isnan(r)
        r = 1e99;
    end
end

function ycalc = project(res)
    global xdata
    global ydata
    global xVarIdx

    n = length(ydata);
    yPwrInv = 1/res(1,1);
    totTerms = length(xVarIdx);
    ycalc = zeros(n,1);
    for i=1:n
        try
            sum1 = res(2,2);
            sum2 = 1;
            for j=1:totTerms
                p = j+2;
                if xVarIdx(j) >=0
                    k = xVarIdx(j);
                    sum1 = sum1 + res(2,p)*xdata(i,k)^res(1,p);
                else
                    k = abs(xVarIdx(j));
                    sum2 = sum2 + res(2,p)*xdata(i,k)^res(1,p);
                end
            end
            ycalc(i) = (sum1 / sum2)^yPwrInv;
        catch ME
            ycalc(i) = Inf;
        end
    end
end
end

```

The above function performs the following tasks:

- Display the names of the Excel and diary files used.
- Read the regression data, transformation data to set the ranges for the optimized powers.
- Perform a random search for the best powers used in the different terms. This task optimizes the power parameters in the function myFit(). This function calculates the sum of errors squared. The variable gBetsX stores the best powers. The variable bestSSE stores the least sum of errors squared.
- Write the values for the best regression model in sheet **Results**.

- Write the input data, the calculated predicted values for the dependent variable, and the percentage errors in sheet **Project**. This task invokes the local function `project()` to calculate the projected values for predicted values for the dependent variable. Thus, sheet **Project** provides data for cross-validation.

Here are the contents of the diary file `besttrathetFlexRand1_2024-Dec-10 15_45_08.txt`:

```
Please wait ...

2024-Dec-10 15_45_08
Function besttrathetFlexRand

Excel file used is C:\MATLAB\bestOptimumRatHet\Data1_44FlexRand1.xlsx
Diary file used is besttrathetFlexRand1_2024-Dec-10 15_45_08.txt
Iter: 1, Best Fx =6.539237e+03, X=[1.000000, 1.071630, 1.442458, 2.345461,
2.870664, 1.465317, 1.502739, 2.910969, 1.508382]
Iter: 2, Best Fx =5.409693e+03, X=[1.000000, 1.207539, 1.174060, 2.015212,
1.415890, 1.403829, 1.632180, 1.543630, 1.139634]
Iter: 4, Best Fx =4.816426e+03, X=[1.000000, 2.502090, 2.203053, 2.587680,
1.055534, 2.046672, 2.337696, 1.881905, 2.642849]
Iter: 6, Best Fx =2.583263e+03, X=[1.000000, 1.889964, 2.114374, 1.855658,
2.337866, 1.371668, 2.922634, 1.323729, 1.023643]
Iter: 403, Best Fx =2.473388e+03, X=[1.000000, 2.757263, 2.899241, 1.433438,
2.076382, 1.126331, 1.038378, 1.552183, 1.763725]
Iter: 689, Best Fx =2.361136e+03, X=[1.000000, 1.734713, 2.252422, 2.550561,
1.290080, 1.289946, 2.425340, 1.632973, 1.027872]
Iter: 1297, Best Fx =2.189612e+03, X=[1.000000, 2.185901, 2.247976, 2.826123,
2.591902, 1.105952, 2.423721, 1.354468, 1.190333]
Iter: 1499, Best Fx =1.670909e+03, X=[1.000000, 1.149419, 2.701538, 2.958855,
2.277469, 1.170426, 1.000809, 1.493447, 1.172925]
Iter: 9490, Best Fx =1.531770e+03, X=[1.000000, 1.817314, 2.883592, 1.864292,
2.142077, 1.378060, 1.003385, 1.362209, 1.256433]
Iter: 14415, Best Fx =1.398820e+03, X=[1.000000, 2.527744, 2.198522,
2.825706, 1.513629, 1.228206, 2.094443, 1.488528, 1.111695]
Iter: 19234, Best Fx =1.244825e+03, X=[1.000000, 2.813724, 2.311992,
2.349229, 1.517286, 1.535952, 2.682040, 1.272504, 1.124190]
Iter: 28456, Best Fx =1.186872e+03, X=[1.000000, 1.945708, 2.868865,
1.875950, 1.298072, 1.211363, 1.012906, 1.236734, 1.418553]
Iter: 35067, Best Fx =9.938279e+02, X=[1.000000, 2.239195, 2.319515,
2.283218, 2.441977, 1.274612, 2.024785, 1.294065, 1.167914]

Results (powers/coefficients)

bestPwrs =

    1.0000    2.2392    2.3195    2.2832    2.4420    1.2746    2.0248
1.2941    1.1679

Regression model
Adj R-sqr =    0.99776646

bestMdl =
```

Linear regression model (robust fit):

$$y \sim 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8$$

Estimated Coefficients:

	Estimate	SE	tStat	pValue
(Intercept)	18.0993103521591	0.404513347397325	44.7434193917497	9.85579503616332e-119
x1	0.00156861634508431	3.79974122830064e-05	41.2821887290954	2.82680840473415e-111
x2	0.0102500083756599	0.00012873959823051	79.6181479245193	4.51809698069492e-175
x3	0.00145999140064393	3.18759169194891e-05	45.8023342303066	6.33975503819303e-121
x4	-4.07595192652753e-05	9.56911883208332e-06	-4.2594851187987	2.93927953007333e-05
x5	0.00425115878618873	6.73174136281363e-05	63.1509524366501	6.22296333783989e-152
x6	7.64060279397448e-05	1.91225561222225e-06	39.9559700342322	2.72258922775523e-108
x7	0.00255424261778418	4.69409045156657e-05	54.414005101494	2.28564162121057e-137
x8	0.00409593665362559	7.20900361597969e-05	56.8169593443734	1.45177904736554e-141

Number of observations: 250, Error degrees of freedom: 241

Root Mean Squared Error: 2.03

R-squared: 0.998, Adjusted R-Squared: 0.998

F-statistic vs. constant model: 1.39e+04, p-value = 2.13e-316

As expected, the above output is not too close to the correct data. For example, the powers appear as [1.0000, 2.2392, 2.3195, 2.2832, 2.4420, 1.2746, 2.0248, 1.2941, 1.1679] compared to the actual powers of [1, 2, 2.5, 2, 1.5, 1.25, 1.3, 1.15, 1.15]. The two sets of regression coefficients are not very close to each other.

Figure 4.1 shows the sheet **Results** in file Data1\_44FlexRand1.xlsx.

Y	Intercept	X1	X2	X3	X4	YX1	YX2	YX3	YX4
1		2.239195196	2.319515326	2.283218184	2.441976615	1.27461242	2.024784868	1.294065138	1.167913572
	18.09931035	0.001568616	0.010250008	0.001459991	-4.07595E-05	0.004251159	7.6406E-05	0.002554243	0.004095937

*Figure 4.1 The sheet **Results** in file Data1\_44FlexRand1.xlsx.*

Figure 4.2 shows the first 31 rows of sheet **Project** in file Data1\_44FlexRand1.xlsx.

Y	X1	X2	X3	X4	Ycalc	%Err
8.29976972	15	14	7	55	14.0709819	69.5346061
9.74036055	34	12	3	56	13.3990977	37.56264587
9.81712564	42	7	20	81	10.9073737	11.10557277
10.6526654	18	14	18	29	16.4483769	54.40620893
11.5715774	29	7	39	57	13.282619	14.78658947
12.8406389	42	13	25	46	15.0359879	17.09688295
13.5038389	49	10	43	89	12.9643137	-3.995346747
14.1931634	69	2	12	98	12.6059285	-11.18309481
14.5647036	51	6	38	54	14.6070574	0.290797703
14.5715035	13	6	56	72	14.7081272	0.937608347
15.0684634	63	14	6	73	15.3210797	1.676457074
16.5299714	45	9	49	44	16.5519607	0.133026466
16.6440619	80	5	26	99	14.6308188	-12.09586448
17.4388009	48	14	34	19	18.3995466	5.509241365
17.6685133	65	19	18	62	18.1618521	2.79219224
17.7864124	69	4	44	67	16.2108071	-8.858476968
17.9578521	19	23	43	63	19.6851992	9.618896223
18.8923329	40	4	77	99	16.7083302	-11.56025936
19.1635126	42	4	74	85	17.2443955	-10.01443284
19.1914811	63	15	14	22	19.6237539	2.252420372
19.281362	37	26	42	73	20.1912264	4.718880764
19.7453187	39	29	27	82	20.7123526	4.897534976
20.0241135	52	8	42	3	18.891401	-5.656742558
20.3176101	32	22	65	78	20.404231	0.426334491
20.8138144	43	30	40	92	21.023198	1.00598406
21.5264679	80	17	1	49	21.0273211	-2.318758624
21.6641961	4	8	84	98	19.5691876	-9.670372978
21.7661287	47	20	52	29	22.0446601	1.279655161
22.0774939	56	21	9	14	22.840009	3.453811983

Figure 4.2. The first 31 rows in sheet **Project** of file *Data1\_44FlexRand1.xlsx*.

Table 4.2 shows errors that seem to somewhat agree with the adjusted coefficient of determination of 0.99776646.

## 5/Nested Random Search Optimization

This section looks at an enhanced random search optimization. The method covered performs an additional nested random search when an improved optimum point is located. The nested search generates random numbers close to that optimum point in hope to find a better optimum point. The test program file test\_FlexNestedRand1.m contains the following code that tests the data in Excel workbook data1\_44FlexNestedRand1. Here is the code for the test program test\_FlexRand1.m:

```
clc
close all
clear

outFile = "bestrathetFlexNestedRand1_";
% You may comment the next statement
delete(strcat(outFile,"*.txt"));

dt = datetime('now','TimeZone','local','Format','y-MMM-d HH_mm_ss');
dtstr = string(dt);
XlFile = strcat(pwd,'\Data1_44FlexNestedRand1.xlsx');
outFile = strcat(outFile,dtstr,".txt");

[bestMdl1, bestPwrs1] = bestrathetFlexNestedRand(XlFile, outFile);

clear GLOBAL
fprintf("Done!\n");
```

The test program performs a test using Excel file Data1\_44FlexRand1.xlsx and echoes the console output to the diary file bestrathetFlexNestedRand1\_2024-Dec-10 15\_56\_21.txt.

The test program calls function bestrathetFlexNestedRand() listed next:

```
function [bestMdl, bestPwrs] = ...
    bestrathetFlexNestedRand(XlFile, outFile, MaxIters, StallItersToStop)

    global mdl
    global xdata
    global ydata
    global xVarIdx

    warning("off")

    diary(outFile)

    if nargin < 4, StallItersToStop = 100000; end
    if nargin < 3, MaxIters = 1000000; end

    dt = datetime('now','TimeZone','local','Format','y-MMM-d HH_mm_ss');
    dtstr = string(dt);
```

```

fprintf("Please wait ...\n\n");
fprintf("%s\n", dtstr);
fprintf("Function bestrathetFlexNestedRand\n\n");
fprintf("Excel file used is %s\n", XlFile);
fprintf("Diary file used is %s\n", outFile);
xyData = readmatrix(XlFile, 'Sheet', 'Data');
ydata = xyData(:,1);
xdata = xyData(:,2:end);
xyTransf = readmatrix(XlFile, 'Sheet', 'Transf');
[~,nVars] = size(xyTransf);
nXvars = nVars - 1;
xVarIdx = xyTransf(1,2:end);
Lb = xyTransf(2,:);
Ub = xyTransf(3,:);
Lb2 = Lb;
Ub2 = Ub;
MaxIters2 = fix(MaxIters/10);
bestSSE = 1e99;
lastIter = 0;
for iter=1:MaxIters
    pop = Lb + (Ub - Lb).*rand(1,nVars);
    dummy= myFit(pop);
    if mdl.SSE < bestSSE
        gBestX = pop;
        bestSSE = mdl.SSE;
        for i=1:nVars
            if gBestX(i) > 0
                Ub2(i)=1.15*gBestX(i);
                Lb2(i)=0.85*gBestX(i);
            elseif gBestX(i) < 0
                Ub2(i)=0.85*gBestX(i);
                Lb2(i)=1.15*gBestX(i);
            else
                Ub2(i)= -0.15;
                Lb2(i)= 0.15;
            end
            if Lb2(i) < Lb(i), Lb2(i) = Lb(i); end
            if Ub2(i) < Ub(i), Ub2(i) = Ub(i); end
        end
        for iter2=1:MaxIters2
            pop = Lb2 + (Ub2 - Lb2).*rand(1,nVars);
            dummy= myFit(pop);
            if mdl.SSE < bestSSE
                gBestX = pop;
                bestSSE = mdl.SSE;
            end
        end
        fprintf("Iter: %d, ", iter);
        fprintf("Best Fx =%e, X=[" , bestSSE);
        fprintf("%f, ", gBestX(1:nVars-1));
        fprintf("%f]\n", gBestX(nVars));
        lastIter = iter;
    end
    if (iter - lastIter) > StallItersToStop, break; end
end
fprintf("\nResults (powers/coefficients)\n")

```

```

bestPwrs = gBestX
fprintf("Regression model\n")
dummy = myFit(gBestX);
format long
AdjR2 = mdl.Rsquared.Adjusted;
fprintf("Adj R-sqr = %14.8f\n", AdjR2);
writematrix(AdjR2, XlFile, 'Sheet', 'R2', 'Range', 'A2:A2');
bestMdl = mdl

res = zeros(2,nVars+1);
res(1,1)= gBestX(1);
res(1,3:end) = gBestX(2:end);
res(2,2:end) = mdl.Coefficients{:,1};
writematrix(res(1,1), XlFile, 'Sheet', 'Results', 'Range', 'A2:A2');
writematrix(res(1,3:end), XlFile, 'Sheet', 'Results', 'Range', 'C2:Z2');
writematrix(res(2,2:end), XlFile, 'Sheet', 'Results', 'Range', 'B3:Z3');

% now do the projections
ycalc = project(res);
percErr = 100.*(ycalc-ydata)./ydata;
ProjMat = [ydata xdata ycalc percErr];
writematrix(ProjMat, XlFile, 'Sheet', 'Project', 'Range', strcat("A2:Z",
num2str(length(ydata))));

format short
diary off

beepit(4)
end

function beepit(nTimes)
    for i=1:nTimes
        beep;
        pause(1);
    end
end

function r = myFit(coeff)
    global mdl
    global xdata
    global ydata
    global xVarIdx

    if coeff(1) >= 0
        y = ydata.^coeff(1);
    else
        y = 1./ydata.^abs(coeff(1));
    end
    X = [];
    for i = 1:length(xVarIdx)
        j = abs(xVarIdx(i));
        if coeff(i+1) >= 0
            x = xdata(:,j).^coeff(i+1);
        else
            x = 1./xdata(:,j).^abs(coeff(i+1));
        end
        if xVarIdx(i) < 0

```

```

        x = -1.*y.*x;
    end
    X = [X x];
end
mdl = fitlm(X,y,'RobustOpts','on');
r = mdl.SSE;
if isnan(r)
    r = 1e99;
end
end
end

function ycalc = project(res)
    global xdata
    global ydata
    global xVarIdx

    n = length(ydata);
    yPwrInv = 1/res(1,1);
    totTerms = length(xVarIdx);
    ycalc = zeros(n,1);
    for i=1:n
        try
            sum1 = res(2,2);
            sum2 = 1;
            for j=1:totTerms
                p = j+2;
                if xVarIdx(j) >=0
                    k = xVarIdx(j);
                    sum1 = sum1 + res(2,p)*xdata(i,k)^res(1,p);
                else
                    k = abs(xVarIdx(j));
                    sum2 = sum2 + res(2,p)*xdata(i,k)^res(1,p);
                end
            end
            ycalc(i) = (sum1 / sum2)^yPwrInv;
        catch ME
            ycalc(i) = Inf;
        end
    end
end
end

```

The code of the above function is an enhanced version of that of the simple random search optimization. The above function has additional code to perform a nested random search. This part uses the ranges in arrays Lb2 and Ub2 which are calculated to be +/-15% of the ranges in array gBestX. If the nested random search finds a better optimum point, it updates the variables that store the information for the best optimum point.

Here are the contents of the diary file bestrathetFlexNestedRand1\_2024-Dec-10 15\_56\_21.txt:

Please wait ...



```

2024-Dec-10 15_56_21
Function bestrathetFlexNestedRand

Excel file used is C:\MATLAB\bestOptimumRatHet\Data1_44FlexNestedRand1.xlsx
Diary file used is bestrathetFlexNestedRand1_2024-Dec-10 15_56_21.txt
Iter: 1, Best Fx =4.016666e+03, X=[1.000946, 2.319897, 2.383941, 2.632755,
2.891285, 1.344926, 2.884891, 1.805025, 1.911415]
Iter: 28, Best Fx =1.912531e+03, X=[1.010943, 2.394475, 2.830252, 2.279593,
2.203852, 1.553123, 1.032448, 1.183913, 1.428052]
Iter: 2393, Best Fx =1.216857e+03, X=[1.002870, 2.524209, 2.199331, 2.586233,
2.820301, 1.378786, 2.725085, 1.310817, 1.029509]
Iter: 18205, Best Fx =9.830566e+02, X=[1.000000, 1.802886, 2.548722,
1.972209, 1.053724, 1.144748, 1.039898, 1.033744, 1.183437]

Results (powers/coefficients)

bestPwrs =

    1.0000    1.8029    2.5487    1.9722    1.0537    1.1447    1.0399
    1.0337    1.1834

Regression model
Adj R-sqr =    0.99778179

bestMdl =

Linear regression model (robust fit):
    y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8

Estimated Coefficients:

```

	Estimate	SE	tStat	pValue
(Intercept)	16.0235972673208	0.573619933921678	27.9341709026254	1.57592368976555e-77
x1	-0.013910581490907	0.000349235637389128	-39.8315063001644	5.23306681642024e-108
x2	-0.00488784954760914	0.000111050092370527	-44.0148174870533	3.35344539174112e-117
x3	-0.00569710162004006	0.000169115558642316	-33.6876255844064	3.86298689329218e-93
x4	-0.101297178510363	0.00633163015136206	-15.9985937410719	9.54416589720746e-40
x5	-0.00949514579833356	0.000218815434281587	-43.3934006049799	7.0413302076916e-116
x6	-0.0279932947891609	0.000466544617547531	-60.0013240669506	6.75743081883034e-147
x7	-0.00940776448550864	0.000235888171914138	-39.8823069811784	4.00731116707526e-108
x8	-0.00524355521849931	0.000123336793606439	-42.5141197948699	5.54255010249497e-114

```

Number of observations: 250, Error degrees of freedom: 241
Root Mean Squared Error: 2.02
R-squared: 0.998, Adjusted R-Squared: 0.998
F-statistic vs. constant model: 1.4e+04, p-value = 9.29e-317

```

As expected, the above output is not too close to the correct data. For example, the powers appear as [1.0000, 1.8029, 2.5487, 1.9722, 1.0537, 1.1447, 1.0399, 1.0337, 1.1834] compared to the actual powers of [1, 2, 2.5, 2, 1.5, 1.25, 1.3, 1.15, 1.15]. The two sets of regression coefficients are not very close to each other. The adjusted coefficient of determination for the nested random search is *slightly* better than that of the regular random search.

Figure 5.1 shows the sheet **Results** in file Data1\_44FlexNestedRand1.xlsx.

Y	Intercept	X1	X2	X3	X4	YX1	YX2	YX3	YX4
1		1.80288629	2.548721835	1.972208724	1.053724422	1.144747565	1.039898485	1.033744096	1.1834372
	16.0235973	-0.013910581	-0.00488785	-0.005697102	-0.101297179	-0.009495146	-0.027993295	-0.009407764	-0.005243555

*Figure 5.1 The sheet **Results** in file Data1\_44FlexNestedRand1.xlsx.*

Figure 5.2 shows the first 31 rows of sheet **Project** in file Data1\_44FlexNestedRand1.xlsx.

Y	X1	X2	X3	X4	Ycalc	%Err
8.29976972	15	14	7	55	-9.23731889	-211.2960866
9.74036055	34	12	3	56	3.339153091	-65.71838308
9.81712564	42	7	20	81	8.433183528	-14.09722318
10.6526654	18	14	18	29	-25.4779786	-339.1699875
11.5715774	29	7	39	57	8.108652661	-29.92612499
12.8406389	42	13	25	46	9.647420233	-24.86806681
13.5038389	49	10	43	89	13.46854058	-0.261394733
14.1931634	69	2	12	98	16.5808806	16.82300929
14.5647036	51	6	38	54	14.90065983	2.306646492
14.5715035	13	6	56	72	13.94363118	-4.308905831
15.0684634	63	14	6	73	15.39832235	2.18906824
16.5299714	45	9	49	44	16.27422631	-1.547160155
16.6440619	80	5	26	99	18.51775019	11.25739775
17.4388009	48	14	34	19	14.67083025	-15.87248279
17.6685133	65	19	18	62	17.55016548	-0.669823142
17.7864124	69	4	44	67	20.11046698	13.06646094
17.9578521	19	23	43	63	15.95740237	-11.13969362
18.8923329	40	4	77	99	20.80412659	10.11941551
19.1635126	42	4	74	85	21.35968616	11.4601829
19.1914811	63	15	14	22	18.68115446	-2.659130843
19.281362	37	26	42	73	18.37967668	-4.676460517
19.7453187	39	29	27	82	19.17942273	-2.865975596
20.0241135	52	8	42	3	19.8073991	-1.082267324
20.3176101	32	22	65	78	19.58558886	-3.602890321
20.8138144	43	30	40	92	20.37654262	-2.100872712
21.5264679	80	17	1	49	22.88981724	6.333362907
21.6641961	4	8	84	98	23.85006984	10.08979839
21.7661287	47	20	52	29	20.62432016	-5.245804236
22.0774939	56	21	9	14	20.85442619	-5.539884757
22.0977472	33	19	75	70	21.78892796	-1.397514486

Figure 5.2. The first 31 rows of sheet **Project** in file  
*Data1\_44FlexNestedRand1.xlsx*.

Table 5.2 shows errors that seem to somewhat agree with the adjusted coefficient of determination of 0.99778179.

## 6/Particle Swarm Optimization

This section looks at a popular evolutionary optimization algorithm, the particle swarm optimization (PSO). The test program test\_FlexPSO1.m has the following code.

```
clc
close all
clear

outFile = "besttrathetFlexPSO1_";
% You may comment the next statement
delete(strcat(outFile,"*.txt"));

dt = datetime('now','TimeZone','local','Format','y-MMM-d HH_mm_ss');
dtstr = string(dt);
XlFile = strcat(pwd,'\data1_44FlexPSO1.xlsx');
outFile = strcat("besttrathetFlexMatlabPSO1_",dtstr,".txt");
MaxPop = 1000;
MaxIters = 2000;
[bestMdl1, bestPwrs2] = ...
    besttrathetFlexPSO(XlFile, outFile, MaxPop,MaxIters, fix(MaxIters/10));

clear GLOBAL
fprintf("Done!\n");
```

The test program performs a test using Excel file data1\_44FlexPSO1.xlsx and echoes the console output to diary file besttrathetFlexPSO1\_2024-Dec-11 08\_08\_18.txt.

☛ The program works with a population of 1000 members and uses 2000 iterations. These numbers are high compared to those used with common test functions. We need the proverbial big guns here because the optimization of rational heteronomials is a difficult problem. Using a population of 20 members and 100 iterations will cause the optimization process to fall flat on its face! This study uses high populations and iterations with all the evolutionary optimization algorithms.

The test program calls function besttrathetFlexPSO1() that is listed next:

```
function [bestMdl, bestPwrs] = ...
    besttrathetFlexPSO(XlFile, outFile, MaxPop, MaxIters, itersToReset)

    global mdl
    global xdata
```

```

global ydata
global xVarIdx

warning("off")

diary(outFile)

if nargin < 5, itersToReset = 75; end
if nargin < 4, MaxPop = 500; end
if nargin < 3, MaxIters = 5000; end

dt = datetime('now','TimeZone','local','Format','y-MMM-d HH_mm_ss');
dtstr = string(dt);

fprintf("Please wait ...\n\n");
fprintf("%s\n", dtstr);
fprintf("Function bestrathetFlexPSO\n\n");
fprintf("Excel file used is %s\n", XlFile);
fprintf("Diary file used is %s\n", outFile);
xyData = readmatrix(XlFile,'Sheet','Data');
[maxrows,nVars] = size(xyData);
ydata = xyData(:,1);
xdata = xyData(:,2:end);
xyTransf = readmatrix(XlFile,'Sheet','Transf');
[~,nVars] = size(xyTransf);
nXvars = nVars - 1;
xVarIdx = xyTransf(1,2:end);
Lb = xyTransf(2,:);
Ub = xyTransf(3,:);

gBestX = pso(@myFit,Lb,Ub,MaxPop,MaxIters,itersToReset)
fprintf("\nResults (powers/coefficients)\n")
bestPwrs = gBestX
fprintf("Regression model\n")
dummy = myFit(gBestX);
format long
AdjR2 = mdl.Rsquared.Adjusted;
fprintf("Adj R-sqr = %14.8f\n", AdjR2);
writematrix(AdjR2, XlFile, 'Sheet', 'R2', 'Range', 'A2:A2');
bestMdl = mdl

res = zeros(2,nVars+1);
res(1,1)= gBestX(1);
res(1,3:end) = gBestX(2:end);
res(2,2:end) = mdl.Coefficients{:,1};
writematrix(res(1,1), XlFile, 'Sheet', 'Results', 'Range', 'A2:A2');
writematrix(res(1,3:end), XlFile, 'Sheet', 'Results', 'Range', 'C2:Z2');
writematrix(res(2,2:end), XlFile, 'Sheet', 'Results', 'Range', 'B3:Z3');

% now do the projections
ycalc = project(res);
percErr = 100.*(ycalc-ydata)./ydata;
ProjMat = [ydata xdata ycalc percErr];
writematrix(ProjMat, XlFile, 'Sheet', 'Project', 'Range', strcat("A2:Z",
num2str(length(ydata))));

format short

```

```

diary off

beepit(4)
end

function beepit(nTimes)
    for i=1:nTimes
        beep;
        pause(1);
    end
end

function r = myFit(coeff)
    global mdl
    global xdata
    global ydata
    global xVarIdx

    if coeff(1) >= 0
        y = ydata.^coeff(1);
    else
        y = 1./ydata.^abs(coeff(1));
    end
    X = [];
    for i = 1:length(xVarIdx)
        j = abs(xVarIdx(i));
        if coeff(i+1) >= 0
            x = xdata(:,j).^coeff(i+1);
        else
            x = 1./xdata(:,j).^abs(coeff(i+1));
        end
        if xVarIdx(i) < 0
            x = -1.*y.*x;
        end
        X = [X x];
    end
    mdl = fitlm(X,y,'RobustOpts','on');
    r = mdl.SSE;
    if isnan(r)
        r = 1e99;
    end
end

function ycalc = project(res)
    global xdata
    global ydata
    global xVarIdx

    n = length(ydata);
    yPwrInv = 1/res(1,1);
    totTerms = length(xVarIdx);
    ycalc = zeros(n,1);
    for i=1:n
        try
            sum1 = res(2,2);
            sum2 = 1;

```

```

    for j=1:totTerms
        p = j+2;
        if xVarIdx(j) >=0
            k = xVarIdx(j);
            sum1 = sum1 + res(2,p)*xdata(i,k)^res(1,p);
        else
            k = abs(xVarIdx(j));
            sum2 = sum2 + res(2,p)*xdata(i,k)^res(1,p);
        end
    end
    ycalc(i) = (sum1 / sum2)^yPwrInv;
catch ME
    ycalc(i) = Inf;
end
end
end

```

The PSO test program is like the random search test program. The main difference is that the PSO test program calls function `pso()` to optimize the powers for the variables. The function `pso()` resides in file `pso.m`.

The test yields the following console output found in file `besttrathetFlexPSO1_2024-Dec-11 08_08_18.txt`. I have trimmed the output to remove lines with somewhat repetitive information.

```

Please wait ...

2024-Dec-11 08_08_18
Function besttrathetFlexPSO

Excel file used is C:\MATLAB\bestOptimumRatHet\data1_44FlexPSO1.xlsx
Diary file used is besttrathetFlexPSO1_2024-Dec-11 08_08_18.txt
Best Fx = 2.112555e+03, Best X = [1.000000, 1.695438, 2.944302, 2.672105,
2.423611, 1.429799, 1.030895, 1.420087, 2.072937]
Iter = 1, Best Fx = 1.274686e+03 *, Best X = [1.000000, 2.179603, 2.993057,
2.653246, 2.442529, 1.268741, 1.006461, 1.219854, 1.511717]
Iter = 6, Best Fx = 7.364696e+02 *, Best X = [1.000000, 2.325766, 2.378654,
2.245297, 1.695739, 1.300730, 2.028855, 1.248955, 1.116967]
Iter = 104, Best Fx = 6.573708e+02 *, Best X = [1.000000, 2.237281, 2.456043,
2.059648, 1.554839, 1.298743, 1.829190, 1.196510, 1.120345]
Iter = 116, Best Fx = 6.206870e+02 *, Best X = [1.000000, 2.227263, 2.478245,
2.049134, 1.379878, 1.297834, 1.702756, 1.196413, 1.137149]
Iter = 121, Best Fx = 6.107601e+02 *, Best X = [1.000000, 2.228532, 2.477613,
2.049412, 1.513249, 1.294040, 1.696299, 1.196558, 1.140586]
Iter = 128, Best Fx = 5.880199e+02 *, Best X = [1.000000, 2.178480, 2.476578,
2.049440, 1.757292, 1.289315, 1.661365, 1.195267, 1.137961]
Iter = 129, Best Fx = 5.737518e+02 *, Best X = [1.000000, 2.151940, 2.476910,
2.049354, 1.868912, 1.283683, 1.637778, 1.194478, 1.134349]
Iter = 131, Best Fx = 5.487941e+02 *, Best X = [1.000000, 2.117415, 2.477451,
2.074135, 2.014104, 1.266969, 1.638550, 1.195966, 1.146886]
Iter = 132, Best Fx = 5.438582e+02 *, Best X = [1.000000, 2.106597, 2.477546,
2.080299, 2.359994, 1.259308, 1.608552, 1.197037, 1.150796]

```

```

Iter = 135, Best Fx = 5.220468e+02 *, Best X = [1.000000, 2.100082, 2.489929,
2.081553, 2.355801, 1.263853, 1.619196, 1.189013, 1.142016]
Iter = 140, Best Fx = 5.129162e+02 *, Best X = [1.000000, 2.096804, 2.490329,
2.072636, 2.426366, 1.263886, 1.617651, 1.187880, 1.142673]
Iter = 146, Best Fx = 5.121274e+02 *, Best X = [1.000000, 2.094704, 2.490266,
2.072457, 2.419336, 1.263889, 1.617596, 1.188522, 1.142589]
Iter = 147, Best Fx = 5.089042e+02 *, Best X = [1.000000, 2.090437, 2.490742,
2.070322, 2.166649, 1.263959, 1.598957, 1.188751, 1.143339]
Iter = 155, Best Fx = 5.043666e+02 *, Best X = [1.000000, 2.090680, 2.486674,
2.059983, 2.206999, 1.264185, 1.570074, 1.189087, 1.145242]

```

.....

```

2.000031, 1.500300, 1.249919, 1.300150, 1.150026, 1.150040]
Iter = 245, Best Fx = 1.902524e+00, Best X = [1.000000, 1.999833, 2.500032,
2.000021, 1.499965, 1.249924, 1.300066, 1.150020, 1.150035]
Iter = 246, Best Fx = 1.812842e+00, Best X = [1.000000, 1.999857, 2.500033,
1.999966, 1.499682, 1.249926, 1.300130, 1.150024, 1.149985]
Iter = 247, Best Fx = 1.279848e+00, Best X = [1.000000, 1.999876, 2.500031,
1.999995, 1.500001, 1.249941, 1.300109, 1.150022, 1.150022]
Iter = 247, Best Fx = 1.238225e+00, Best X = [1.000000, 1.999886, 2.500050,
1.999977, 1.500187, 1.249947, 1.300129, 1.150017, 1.150024]
Iter = 247, Best Fx = 1.188170e+00, Best X = [1.000000, 1.999875, 2.500009,
2.000031, 1.500259, 1.249965, 1.300070, 1.150015, 1.150040]
Iter = 248, Best Fx = 4.185127e-01, Best X = [1.000000, 1.999901, 2.499994,
2.000022, 1.499978, 1.249977, 1.299973, 1.150002, 1.150013]
Iter = 249, Best Fx = 2.749174e-01, Best X = [1.000000, 1.999904, 2.499989,
2.000017, 1.499982, 1.249973, 1.299955, 1.149998, 1.150004]
Iter = 251, Best Fx = 1.846272e-01 *, Best X = [1.000000, 1.999943, 2.499989,
2.000018, 1.500004, 1.249977, 1.299961, 1.149996, 1.150002]
Iter = 252, Best Fx = 1.265295e-01, Best X = [1.000000, 1.999974, 2.499995,
2.000020, 1.499957, 1.249983, 1.299976, 1.150008, 1.150006]
Iter = 254, Best Fx = 1.153379e-01, Best X = [1.000000, 1.999994, 2.499994,
2.000020, 1.499910, 1.249992, 1.299977, 1.150010, 1.150007]
Iter = 254, Best Fx = 6.000512e-02, Best X = [1.000000, 1.999998, 2.499993,
2.000019, 1.499908, 1.249993, 1.299976, 1.150009, 1.149998]
Iter = 254, Best Fx = 3.758920e-02, Best X = [1.000000, 2.000003, 2.499991,
2.000021, 1.499915, 1.249999, 1.299977, 1.150009, 1.149999]
Iter = 255, Best Fx = 3.263656e-02, Best X = [1.000000, 2.000005, 2.499991,
2.000019, 1.499907, 1.250001, 1.299978, 1.150009, 1.149995]
Iter = 257, Best Fx = 2.434924e-02, Best X = [1.000000, 2.000004, 2.499992,
2.000012, 1.499873, 1.250000, 1.299981, 1.150004, 1.149996]
Iter = 257, Best Fx = 2.314813e-02, Best X = [1.000000, 2.000002, 2.499993,
2.000012, 1.499863, 1.250000, 1.299979, 1.150005, 1.149994]
Iter = 258, Best Fx = 9.548155e-03, Best X = [1.000000, 2.000000, 2.499998,
2.000013, 1.499933, 1.250001, 1.299999, 1.150005, 1.149997]
Iter = 258, Best Fx = 8.621883e-03, Best X = [1.000000, 2.000002, 2.500005,
2.000006, 1.499934, 1.250002, 1.300009, 1.150004, 1.149998]
Iter = 260, Best Fx = 7.417345e-03, Best X = [1.000000, 2.000001, 2.500003,
2.000002, 1.499930, 1.250001, 1.300006, 1.150004, 1.149997]
Iter = 260, Best Fx = 4.957786e-03, Best X = [1.000000, 2.000004, 2.500000,
2.000004, 1.499965, 1.250001, 1.300005, 1.150003, 1.149999]

```

.....

```

Iter = 311, Best Fx = 3.909894e-12, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]

```



```

Iter = 311, Best Fx = 3.087704e-12, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 312, Best Fx = 2.606027e-12, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 312, Best Fx = 2.157658e-12, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 313, Best Fx = 2.053524e-12, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 314, Best Fx = 2.025492e-12, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 314, Best Fx = 1.911259e-12, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 314, Best Fx = 1.764952e-12, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 315, Best Fx = 1.559596e-12, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 315, Best Fx = 1.495582e-12, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 315, Best Fx = 1.038469e-12, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 316, Best Fx = 9.198642e-13, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 316, Best Fx = 7.846390e-13, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 317, Best Fx = 7.729563e-13, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 317, Best Fx = 6.265476e-13, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 317, Best Fx = 5.013022e-13, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 318, Best Fx = 4.275246e-13, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]

```

.....

```

Iter = 357, Best Fx = 2.092097e-19, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 357, Best Fx = 1.990361e-19, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 359, Best Fx = 1.984863e-19, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 360, Best Fx = 1.950011e-19, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 360, Best Fx = 1.931805e-19, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 360, Best Fx = 1.923973e-19, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 362, Best Fx = 1.907244e-19, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 365, Best Fx = 1.884167e-19, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 365, Best Fx = 1.844018e-19, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 373, Best Fx = 1.836606e-19, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 380, Best Fx = 1.813005e-19, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]

```

```

Iter = 380, Best Fx = 1.771178e-19, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 407, Best Fx = 1.768634e-19, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 417, Best Fx = 1.765823e-19, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 440, Best Fx = 1.762938e-19, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 553, Best Fx = 1.740537e-19, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 605, Best Fx = 1.723624e-19, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter = 606, Best Fx = 1.694529e-19, Best X = [1.000000, 2.000000, 2.500000,
2.000000, 1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
----- Reset population -----
AT iteration 807
----- Reset population -----
AT iteration 1008
----- Reset population -----
AT iteration 1209
----- Reset population -----
AT iteration 1410
----- Reset population -----
AT iteration 1611
----- Reset population -----
AT iteration 1812
random Search Success Count = 190

Results (powers/coefficients)

bestPwrs =

      1.0000      2.0000      2.5000      2.0000      1.5000      1.2500      1.3000
1.1500      1.1500

Regression model
Adj R-sqr =      1.000000000

bestMdl =

Linear regression model (robust fit):
      y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8

Estimated Coefficients:

```

	Estimate	SE	tStat	pValue
(Intercept)	5.00000000001799	8.25459829317715e-12	605722994921.6	0
x1	0.999999999999863	2.2775108674235e-13	4390758412192.09	0
x2	0.999999999999844	2.27601120651459e-13	4393651477363.38	0
x3	0.999999999999899	2.27544106210538e-13	4394752369787.33	0
x4	0.999999999999356	2.31554053317286e-13	4318646059842.93	0
x5	0.999999999999845	2.27637349618306e-13	4392952218414.98	0
x6	0.999999999999852	2.28102252929136e-13	4383998786327.28	0
x7	0.999999999999987	2.27623495170133e-13	4393219598233.64	0
x8	0.999999999999814	2.27694778938139e-13	4391844225253.39	0

Number of observations: 250, Error degrees of freedom: 241  
 Root Mean Squared Error: 2.65e-11  
 R-squared: 1, Adjusted R-Squared: 1  
 F-statistic vs. constant model: 8e+25, p-value = 0.

I was pleased that the above output found the correct powers and regression coefficients. For example, the powers of [1, 2, 2.5, 2, 1.5, 1.25, 1.3, 1.15, 1.15] are the correct ones. The intercept of the numerator part is very close to 5 and all the coefficients of the various terms are very close to 1. Thus, the PSO algorithm was able to locate the global minimum for the given set of power ranges (see Figure 3.2).

Figures 6.1 and 6.2 show the sheets **Results** and **Project**. The results in the latter confirm the success of the particle swarm optimization.

Y	Intercept	X1	X2	X3	X4	YX1	YX2	YX3	YX4
1		2	2.5	2	1.5	1.25	1.3	1.15	1.15
	5	1	1	1	1	1	1	1	1

Figure 6.1. The sheet **Results** in the Excel file *data1\_44FlexPSO1.xlsx*.

Y	X1	X2	X3	X4	Ycalc	%Err
8.29976972	15	14	7	55	8.29976972	-6.20672E-13
9.74036055	34	12	3	56	9.74036055	-3.64741E-13
9.81712564	42	7	20	81	9.81712564	5.42834E-14
10.6526654	18	14	18	29	10.6526654	2.16778E-13
11.5715774	29	7	39	57	11.5715774	1.22808E-13
12.8406389	42	13	25	46	12.8406389	5.53355E-14
13.5038389	49	10	43	89	13.5038389	1.57853E-13
14.1931634	69	2	12	98	14.1931634	3.75467E-14
14.5647036	51	6	38	54	14.5647036	8.53742E-14
14.5715035	13	6	56	72	14.5715035	9.7525E-14
15.0684634	63	14	6	73	15.0684634	-1.6504E-13
16.5299714	45	9	49	44	16.5299714	1.50448E-13
16.6440619	80	5	26	99	16.6440619	0
17.4388009	48	14	34	19	17.4388009	3.66704E-13
17.6685133	65	19	18	62	17.6685133	-1.00538E-13
17.7864124	69	4	44	67	17.7864124	3.99486E-14
17.9578521	19	23	43	63	17.9578521	7.91345E-14
18.8923329	40	4	77	99	18.8923329	1.1283E-13
19.1635126	42	4	74	85	19.1635126	7.41558E-14
19.1914811	63	15	14	22	19.1914811	1.11072E-13
19.281362	37	26	42	73	19.281362	7.37025E-14
19.7453187	39	29	27	82	19.7453187	8.99634E-14
20.0241135	52	8	42	3	20.0241135	6.03234E-13
20.3176101	32	22	65	78	20.3176101	5.24577E-14
20.8138144	43	30	40	92	20.8138144	1.53621E-13
21.5264679	80	17	1	49	21.5264679	-1.98047E-13
21.6641961	4	8	84	98	21.6641961	-1.6399E-14
21.7661287	47	20	52	29	21.7661287	2.44833E-13
22.0774939	56	21	9	14	22.0774939	2.4138E-13
22.0977472	33	19	75	70	22.0977472	-3.21545E-14

Figure 6.2. The first 31 rows of sheet **Project** in the Excel file *data1\_44FlexPSO1.xlsx*.

The success of the above test program also depends on the ranges of power as shown in Figure 3.2. To illustrate the point, look at the ranges of power in Figure 6.3 found in file data1\_44FlexPSO2.xlsx

Y	X1	X2	X3	X4	YX1	YX2	YX3	YX4
0	1	2	3	4	-1	-2	-3	-4
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
3	3	3	3	3	3	3	3	3

Figure 6.3. The sheet **Transf** in the Excel file data1\_44FlexPSO2.xlsx.

Figure 6.3 shows that the lower limit for the powers of all the variables is set to 0.5 (down from 1).

The code for the updated test file test\_FlexPSO2.m is:

```
clc
close all
clear

outFile = "besttrathetFlexPSO2_";
% You may comment the next statement
delete(strcat(outFile, "*.txt"));

dt = datetime('now', 'TimeZone', 'local', 'Format', 'y-MMM-d HH_mm_ss');
dtstr = string(dt);
XlFile = strcat(pwd, '\data1_44FlexPSO2.xlsx');
outFile = strcat(outFile, dtstr, ".txt");
MaxPop = 1000;
MaxIters = 5000;
[bestMdl1, bestPwrs2] = ...
    besttrathetFlexPSO(XlFile, outFile, MaxPop, MaxIters, fix(MaxIters/10));

clear GLOBAL
fprintf("Done!\n");
```

Notice that the above test file uses 5000 iterations, up from the 2000 iterations used in the first test file.

The trailing part of the diary file besttrathetFlexPSO2\_2024-Dec-11 16\_44\_51.txt is:

```
Please wait ...

2024-Dec-11 16_44_51
Function besttrathetFlexPSO

Excel file used is
I:\DropBox\Dropbox\MATLAB\bestOptimumRatHet\data1_44FlexPSO2.xlsx
```

```

Diary file used is bestrathetFlexPSO2_2024-Dec-11 16_44_51.txt
Best Fx = 6.057445e+00, Best X = [0.506976, 0.882382, 1.835402, 1.776584,
1.463040, 1.276491, 0.508889, 2.538106, 2.096046]
Iter = 1, Best Fx = 5.250041e+00 *, Best X = [0.501284, 0.515789, 1.991420,
1.774408, 1.516679, 0.672131, 0.502751, 2.494768, 1.817034]

.....

Iter = 3826, Best Fx = 4.403249e+00, Best X = [0.500000, 2.385499, 2.194078,
2.237868, 1.310602, 2.022212, 0.500002, 2.172244, 1.496422]
----- Reset population -----
AT iteration 4327
----- Reset population -----
AT iteration 4828
random Search Success Count = 133

Results (powers/coefficients)

bestPwrs =

      0.5000      2.3855      2.1941      2.2379      1.3106      2.0222      0.5000
2.1722      1.4964

Regression model
Adj R-sqr =      0.99760550

bestMdl =

Linear regression model (robust fit):
      y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8

Estimated Coefficients:

```

	Estimate	SE	tStat	pValue
(Intercept)	2.6440174911784	0.0490321001580477	53.9242145993298	1.71335507438048e-136
x1	-5.62447959594774e-06	1.86936289286958e-06	-3.00876818374941	0.00290142558695042
x2	-0.000149504357943481	7.3106673620631e-06	-20.4501655648152	1.40934414299381e-54
x3	-2.0484447917533e-07	4.12067543580317e-06	-0.0497113840598813	0.960393558913482
x4	-0.00189501646927662	0.000212507291809046	-8.91741856547413	1.2172038618745e-16
x5	-3.13770164645626e-06	1.25057958825313e-06	-2.50899796856523	0.0127648737907842
x6	-0.107464523804327	0.00137162791763469	-78.3481601844647	1.88614989816968e-173
x7	-1.43151364669361e-07	6.70911577534283e-07	-0.21336845191354	0.831219973886061
x8	-7.97981518093765e-05	1.14181087420636e-05	-6.98873636711876	2.6995154823626e-11

```

Number of observations: 250, Error degrees of freedom: 241
Root Mean Squared Error: 0.135
R-squared: 0.998, Adjusted R-Squared: 0.998
F-statistic vs. constant model: 1.3e+04, p-value = 9.33e-313

```

The above output shows that the test program does not yield a perfect fit for the error-free data. Figure 6.4 shows the first 31 rows in sheet **Results**. The figure shows that the percentage errors are significantly higher than those in Figure 6.2. This time the PSO algorithm zoomed in on a local minimum.

Y	X1	X2	X3	X4	Ycalc	%Err
8.29976972	15	14	7	55	15.5699651	87.59514556
9.74036055	34	12	3	56	14.0328418	44.06901786
9.81712564	42	7	20	81	9.30937776	-5.172062588
10.6526654	18	14	18	29	17.331295	62.6944462
11.5715774	29	7	39	57	10.8586034	-6.161424056
12.8406389	42	13	25	46	15.2762209	18.9677637
13.5038389	49	10	43	89	10.3009749	-23.7181738
14.1931634	69	2	12	98	5.27395211	-62.84160226
14.5647036	51	6	38	54	10.1282932	-30.46001135
14.5715035	13	6	56	72	9.48878649	-34.88121209
15.0684634	63	14	6	73	13.4158391	-10.96743726
16.5299714	45	9	49	44	12.6873355	-23.24647676
16.6440619	80	5	26	99	6.33878974	-61.9156083
17.4388009	48	14	34	19	17.6071243	0.96522341
17.6685133	65	19	18	62	17.5730122	-0.540515507
17.7864124	69	4	44	67	7.8725537	-55.73838307
17.9578521	19	23	43	63	21.5475287	19.98945447
18.8923329	40	4	77	99	6.73813196	-64.3340397
19.1635126	42	4	74	85	7.46321036	-61.05510233
19.1914811	63	15	14	22	17.7555595	-7.482078068
19.281362	37	26	42	73	22.7438417	17.95765133
19.7453187	39	29	27	82	24.1175139	22.14294556
20.0241135	52	8	42	3	13.8336034	-30.91527698
20.3176101	32	22	65	78	19.2123109	-5.440104134
20.8138144	43	30	40	92	23.5766555	13.27407423
21.5264679	80	17	1	49	16.5088914	-23.30887025
21.6641961	4	8	84	98	9.00894629	-58.41550624
21.7661287	47	20	52	29	21.7183429	-0.219541708
22.0774939	56	21	9	14	23.3710851	5.859320671
22.0977472	33	19	75	70	17.7914542	-19.4874752

Figure 6.4. The first 31 rows in sheet Results in file data1\_44FlexPSO2.xlsx.

☛ The lesson learned here is “***Choose your power ranges carefully!***” Since most readers do not know ahead of time the best power ranges, they must experiment with these values to get the best results.

## 7/The Cuckoo Search Algorithm (CSA)

This section looks at another popular evolutionary optimization algorithm, the Cuckoo search algorithm (CSA). The test program test\_FlexCuckoo1.m has the following code.

```
clc
close all
clear

optimName = "cuckoo_searchEx4";

outFile = "bestrathetFlexCuckoo1_";
% You may comment the next statement
delete(strcat(outFile,"*.txt"));

dt = datetime('now','TimeZone','local','Format','y-MMM-d HH_mm_ss');
dtstr = string(dt);
XlFile = strcat(pwd,'\Data1_44FlexCuckoo1.xlsx');
outFile = strcat(outFile,dtstr,".txt");

MaxPop = 1000;
MaxIters = 2000;
[bestMdl1, bestPwrs1] = ...
    bestrathetFlexGenericOptim(XlFile, outFile, ...
    @cuckoo_searchEx4,MaxPop,MaxIters,optimName);

clear GLOBAL
fprintf("Done!\n");
```

The test program uses the generic optimization helper function bestrathetFlexGenericOptim(). The first parameter for calling this function is the handle of function cuckoo\_searchEx4() which performs the Cuckoo search. The test program performs a test using Excel file data1\_44FlexCuckoo1.xlsx and echoes the console output to diary file bestrathetFlexCuckoo1\_2024-Dec-11 08\_11\_40.txt.

The test program calls function bestrathetFlexGenericOptim() is listed next:

```
function [bestMdl, bestPwrs] = ...
    bestrathetFlexGenericOptim(XlFile, outFile, optimFx, MaxPop, MaxIters,
    optimName)

    global mdl
    global xdata
    global ydata
```



```

global xVarIdx

warning("off")

diary(outFile)

if nargin < 5, itersToReset = 75; end
if nargin < 4, MaxPop = 500; end
if nargin < 3, MaxIters = 5000; end

dt = datetime('now','TimeZone','local','Format','y-MMM-d HH_mm_ss');
dtstr = string(dt);
fprintf("Using optimizer %s\n", optimName)
fprintf("Please wait ... \n\n");
fprintf("%s\n", dtstr);
fprintf("Function bestrathetFlexGenericOptim\n\n");
fprintf("Excel file used is %s\n", XlFile);
fprintf("Diary file used is %s\n", outFile);
xyData = readmatrix(XlFile,'Sheet','Data');
[maxrows,~] = size(xyData);
ydata = xyData(:,1);
xdata = xyData(:,2:end);
xyTransf = readmatrix(XlFile,'Sheet','Transf');
[~,nVars] = size(xyTransf);
nXvars = nVars - 1;
xVarIdx = xyTransf(1,2:end);
Lb = xyTransf(2,:);
Ub = xyTransf(3,:);

gBestX = optimFx(@myFit,Lb,Ub,MaxPop,MaxIters);
fprintf("\nResults (powers/coefficients)\n")
bestPwrs = gBestX
fprintf("Regression model\n")
dummy = myFit(gBestX);
format long
AdjR2 = mdl.Rsquared.Adjusted;
fprintf("Adj R-sqr = %14.8f\n", AdjR2);
writematrix(AdjR2, XlFile, 'Sheet', 'R2', 'Range', 'A2:A2');
bestMdl = mdl

res = zeros(2,nVars+1);
res(1,1)= gBestX(1);
res(1,3:end) = gBestX(2:end);
res(2,2:end) = mdl.Coefficients{:,1};
writematrix(res(1,1), XlFile, 'Sheet', 'Results', 'Range', 'A2:A2');
writematrix(res(1,3:end), XlFile, 'Sheet', 'Results', 'Range', 'C2:Z2');
writematrix(res(2,2:end), XlFile, 'Sheet', 'Results', 'Range', 'B3:Z3');

% now do the projections
ycalc = project(res);
percErr = 100.*(ycalc-ydata)./ydata;
ProjMat = [ydata xdata ycalc percErr];
writematrix(ProjMat, XlFile, 'Sheet', 'Project', 'Range', strcat("A2:Z",
num2str(length(ydata))));

format short
diary off

```

```

beep on
for i=1:3
    beep;
    pause(1);
end
end

function r = myFit(coeff)
    global mdl
    global xdata
    global ydata
    global xVarIdx

    if coeff(1) >= 0
        y = ydata.^coeff(1);
    else
        y = 1./ydata.^abs(coeff(1));
    end
    X = [];
    for i = 1:length(xVarIdx)
        j = abs(xVarIdx(i));
        if coeff(i+1) >= 0
            x = xdata(:,j).^coeff(i+1);
        else
            x = 1./xdata(:,j).^abs(coeff(i+1));
        end
        if xVarIdx(i) < 0
            x = -1.*y.*x;
        end
        X = [X x];
    end
    mdl = fitlm(X,y,'RobustOpts','on');
    r = mdl.SSE;
    if isnan(r)
        r = 1e99;
    end
end

function ycalc = project(res)
    global xdata
    global ydata
    global xVarIdx

    n = length(ydata);
    yPwrInv = 1/res(1,1);
    totTerms = length(xVarIdx);
    ycalc = zeros(n,1);
    for i=1:n
        try
            sum1 = res(2,2);
            sum2 = 1;
            for j=1:totTerms
                p = j+2;
                if xVarIdx(j) >=0
                    k = xVarIdx(j);
                    sum1 = sum1 + res(2,p)*xdata(i,k)^res(1,p);
                else

```

```

        k = abs(xVarIdx(j));
        sum2 = sum2 + res(2,p)*xdata(i,k)^res(1,p);
    end
end
ycalc(i) = (sum1 / sum2)^yPwrInv;
catch ME
    ycalc(i) = Inf;
end
end
end
end

```

The above function has the parameter `optimFx` which is the handle for the optimization function that has the following call template:

```
[bestX,bestFx]= optimFx(fx,Lb,Ub,MaxPop,MaxIters)
```

The above template shows the minimum set of parameters needed. An optimization function can have more parameters but must assign default values to these extra parameters. The function `cuckoo_searchEx4()` is an example. It has an additional parameter, `Toler`, and assigns a default value of `1e-15` to that parameter.

The test yields the following console output found in file `besttrathetFlexCuckoo1_2024-Dec-11 08_11_40.txt`. I have trimmed the output to remove lines with somewhat repetitive information.

```

Using optimizer cuckoo_searchEx4
Please wait ...

2024-Dec-11 08_11_40
Function besttrathetFlexGenericOptim

Excel file used is
I:\DropBox\Dropbox\MATLAB\bestOptimumRatHet\Data1_44FlexCuckoo1.xlsx
Diary file used is besttrathetFlexCuckoo1_2024-Dec-11 08_11_40.txt
Iter: 3, Fx = 2.742066e+04, X=[1.000000, 1.839957, 2.943564, 2.973274,
2.720017, 1.778693, 1.900723, 1.492374, 2.568237]
Iter: 5, Fx = 1.845850e+04, X=[1.000000, 1.267914, 2.605924, 2.039180,
2.857297, 2.601789, 1.833014, 1.741340, 1.448485]
Iter: 7, Fx = 1.220701e+04, X=[1.000000, 2.120113, 2.254100, 1.133743,
2.284577, 2.657120, 1.781065, 1.875001, 1.173423]
Iter: 10, Fx = 1.078309e+04, X=[1.000000, 2.638802, 2.438195, 1.306155,
2.317556, 2.011657, 2.948219, 2.298760, 2.604288]
Iter: 11, Fx = 1.078309e+04, X=[1.000000, 2.638802, 2.438195, 1.306155,
2.317556, 2.011657, 2.948219, 2.298760, 2.604288]
Iter: 12, Fx = 1.076301e+04, X=[1.000000, 2.646608, 2.438669, 1.307408,
2.317747, 2.009469, 2.951092, 2.301062, 2.596052]

.....

Iter: 747, Fx = 3.173973e-14, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]

```

```

Iter: 749, Fx = 3.173973e-14, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter: 750, Fx = 3.173973e-14, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter: 752, Fx = 1.767894e-14, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter: 753, Fx = 1.650738e-14, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]

Results (powers/coefficients)

bestPwrs =

      1.0000      2.0000      2.5000      2.0000      1.5000      1.2500      1.3000
1.1500      1.1500

Regression model
Adj R-sqr =      1.000000000

bestMdl =

Linear regression model (robust fit):
      y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8

Estimated Coefficients:

```

	Estimate	SE	tStat	pValue
(Intercept)	5.00000000415569	5.87565611800101e-10	8509687945.89151	0
x1	0.99999999902987	1.62114135499475e-11	61684935543.8362	0
x2	0.999999999898035	1.62007389034678e-11	61725579669.9483	0
x3	0.999999999885697	1.61966805925323e-11	61741045899.6618	0
x4	0.999999999781863	1.64821102315685e-11	60671842727.1856	0
x5	0.999999999896898	1.62033176958165e-11	61715755913.0674	0
x6	0.999999999900281	1.62364096996952e-11	61589970836.9057	0
x7	0.999999999888917	1.62023315305182e-11	61719512281.6335	0
x8	0.999999999889544	1.62074055376796e-11	61700189926.4324	0

```

Number of observations: 250, Error degrees of freedom: 241
Root Mean Squared Error: 1.89e-09
R-squared: 1, Adjusted R-Squared: 1
F-statistic vs. constant model: 1.58e+22, p-value = 0

```

I was pleased that the above output found the correct powers and regression coefficients. For example, the powers of [1, 2, 2.5, 2, 1.5, 1.25, 1.3, 1.15, 1.15] are the correct ones. The intercept of the numerator part is very close to 5 and all the coefficients of the various terms are very close to 1. The Cuckoo search algorithm was able to find the global minimum based on the power ranges in Figure 3.2.

Figures 7.1 and 7.2 show the sheets **Results** and **Project**. The results in the latter confirm the success of the Cuckoo search algorithm.

Y	Intercept	X1	X2	X3	X4	YX1	YX2	YX3	YX4
1		2	2.5	2	1.5	1.25	1.3	1.15	1.15
	5	1	1	1	1	1	1	1	1

Figure 7.1. The sheet **Results** in the Excel file *data1\_44FlexCuckoo1.xlsx*.

Y	X1	X2	X3	X4	Ycalc	%Err
8.29976972	15	14	7	55	8.29976972	-1.24948E-10
9.74036055	34	12	3	56	9.74036055	-4.37143E-11
9.81712564	42	7	20	81	9.81712564	-2.03744E-11
10.6526654	18	14	18	29	10.6526654	-1.67586E-11
11.5715774	29	7	39	57	11.5715774	-1.0431E-10
12.8406389	42	13	25	46	12.8406389	-3.4308E-11
13.5038389	49	10	43	89	13.5038389	-1.24047E-11
14.1931634	69	2	12	98	14.1931634	2.57821E-11
14.5647036	51	6	38	54	14.5647036	-4.80657E-11
14.5715035	13	6	56	72	14.5715035	-9.64644E-11
15.0684634	63	14	6	73	15.0684634	-1.0091E-11
16.5299714	45	9	49	44	16.5299714	-4.21039E-11
16.6440619	80	5	26	99	16.6440619	-8.965E-13
17.4388009	48	14	34	19	17.4388009	2.83992E-11
17.6685133	65	19	18	62	17.6685133	-1.35324E-11
17.7864124	69	4	44	67	17.7864124	-3.66329E-11
17.9578521	19	23	43	63	17.9578521	-4.88655E-11
18.8923329	40	4	77	99	18.8923329	1.85982E-11
19.1635126	42	4	74	85	19.1635126	-5.07967E-12
19.1914811	63	15	14	22	19.1914811	2.94155E-11
19.281362	37	26	42	73	19.281362	-1.29164E-11
19.7453187	39	29	27	82	19.7453187	1.02918E-11
20.0241135	52	8	42	3	20.0241135	6.00218E-11
20.3176101	32	22	65	78	20.3176101	-1.59296E-11
20.8138144	43	30	40	92	20.8138144	1.62326E-11
21.5264679	80	17	1	49	21.5264679	-2.87498E-11
21.6641961	4	8	84	98	21.6641961	5.85445E-12
21.7661287	47	20	52	29	21.7661287	-3.39502E-12
22.0774939	56	21	9	14	22.0774939	5.80117E-11
22.0977472	33	19	75	70	22.0977472	-1.63667E-11

Figure 7.2. The first 31 rows of sheet **Project** in the Excel file *data1\_44FlexCuckool.xlsx*.

## 8/Hunter Prey Optimization (HPO) Algorithm

This section looks at an optimization method that is very good but not very popular. It is the Hunter Prey Optimization (HPO) algorithm. The test program `test_FlexHPO1.m` has the following code.

```
clc
close all
clear

optimName = "Hunter Prey Optimization (HPO)";

outFile = "bestrathetFlexHPO1_";
% You may comment the next statement
delete(strcat(outFile,"*.txt"));

dt = datetime('now','TimeZone','local','Format','y-MMM-d HH_mm_ss');
dtstr = string(dt);
XlFile = strcat(pwd,'\Data1_44FlexHPO1.xlsx');
outFile = strcat(outFile,dtstr,".txt");

MaxPop = 1000;
MaxIters = 1000;
[bestMdl1, bestPwrs1] = ...
    bestrathetFlexGenericOptim(XlFile, outFile, ...
        @HPO,MaxPop,MaxIters,optimName);

clear GLOBAL
fprintf("Done!\n");
```

The test program uses the generic optimization helper function `bestrathetFlexGenericOptim()`. The first parameter for calling this function is the handle of function `HPO()` which performs the HPO method. The test program performs a test using Excel file `data1_44FlexHPO1.xlsx` and echoes the console output to diary file `bestrathetFlexHPO1_2024-Dec-12 11_14_56.txt`.

The test yields the following console output found in file `bestrathetFlexCuckoo1_2024-Dec-11 08_11_40.txt`. I have trimmed the output to remove lines with somewhat repetitive information.

```
Using optimizer Hunter Prey Optimization (HPO)
Please wait ...

2024-Dec-12 11_14_56
Function bestrathetFlexGenericOptim

Excel file used is C:\MATLAB\bestOptimumRatHet\Data1_44FlexHPO1.xlsx
Diary file used is bestrathetFlexHPO1_2024-Dec-12 11_14_56.txt
Iter: 1, Fx = 1.975713e+03, X=[1.000000, 2.843151, 1.991765, 2.586008,
2.949144, 1.285499, 2.542031, 1.188961, 1.173721]
```

```

Iter: 1, Fx = 1.782621e+03, X=[1.000000, 2.657871, 2.036811, 1.908456,
1.417799, 1.068109, 2.349690, 1.067121, 1.009869]
Iter: 1, Fx = 1.755255e+03, X=[1.000000, 2.529389, 2.078064, 1.597397,
1.226399, 1.160397, 2.782344, 1.125575, 1.169767]
Iter: 1, Fx = 1.608169e+03, X=[1.000000, 2.579905, 2.138768, 2.139511,
1.592523, 1.144488, 2.582005, 1.277909, 1.171423]
Iter: 1, Fx = 1.539305e+03, X=[1.000000, 2.679471, 2.221745, 2.273487,
1.588416, 1.212694, 2.559167, 1.516717, 1.115171]
Iter: 1, Fx = 1.372545e+03, X=[1.000000, 2.406380, 2.610637, 2.330401,
1.903213, 1.366327, 2.671451, 1.302986, 1.165513]
Iter: 1, Fx = 1.155929e+03, X=[1.000000, 2.409401, 2.344092, 2.599592,
1.714225, 1.352561, 2.674718, 1.436790, 1.159204]
Iter: 1, Fx = 1.111508e+03, X=[1.000000, 2.420930, 2.363876, 2.594549,
1.530003, 1.331754, 2.307395, 1.425524, 1.150383]
Iter: 2, Fx = 9.886745e+02, X=[1.000000, 2.516890, 2.345913, 1.970293,
1.256522, 1.285422, 2.273216, 1.283284, 1.084698]

.....

Iter: 250, Fx = 2.390509e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter: 252, Fx = 2.352848e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter: 258, Fx = 2.345961e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter: 265, Fx = 2.331410e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter: 294, Fx = 2.308174e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
----- New population at iter 344
----- New population at iter 394
----- New population at iter 444
----- New population at iter 494
----- New population at iter 544
----- New population at iter 594
Iter: 632, Fx = 2.297745e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter: 665, Fx = 2.297606e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter: 666, Fx = 2.192500e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter: 666, Fx = 2.184589e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter: 667, Fx = 2.176355e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter: 668, Fx = 2.162232e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter: 675, Fx = 2.132638e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter: 675, Fx = 2.128838e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter: 675, Fx = 2.085127e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter: 693, Fx = 2.065913e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
----- New population at iter 743

```



```

Iter: 778, Fx = 2.056869e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
----- New population at iter 828
----- New population at iter 878
----- New population at iter 928
----- New population at iter 978

Results (powers/coefficients)

bestPwrs =

      1.0000      2.0000      2.5000      2.0000      1.5000      1.2500      1.3000
1.1500      1.1500

Regression model
Adj R-sqr =      1.000000000

bestMdl =

Linear regression model (robust fit):
      y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8

Estimated Coefficients:
              Estimate              SE              tStat              pValue
-----
(Intercept)  4.99999999996493  9.09441388101794e-12  549788041910.105      0
x1            1.000000000000515  2.50922282481877e-13  3985297718935.63      0
x2            1.000000000000522  2.50757058972502e-13  3987923626568.31      0
x3            1.000000000000527  2.50694243932384e-13  3988922858057.28      0
x4            1.000000000000519  2.55112159539594e-13  3919844517838.39      0
x5            1.000000000000517  2.50796973842636e-13  3987288940067.63      0
x6            1.000000000000535  2.51309176008451e-13  3979162304729.07      0
x7            1.000000000000524  2.50781709855073e-13  3987531628933.95      0
x8            1.000000000000517  2.50860245971089e-13  3986283263552.33      0

Number of observations: 250, Error degrees of freedom: 241
Root Mean Squared Error: 2.92e-11
R-squared: 1, Adjusted R-Squared: 1
F-statistic vs. constant model: 6.59e+25, p-value = 0

```

Again, I was pleased that the above output found the correct powers and regression coefficients. For example, the powers of [1, 2, 2.5, 2, 1.5, 1.25, 1.3, 1.15, 1.15] are the correct ones. The intercept of the numerator part is very close to 5 and all the coefficients of the various terms are very close to 1. The hunter prey algorithm was able to find the global minimum based on the power ranges in Figure 3.2.

The HPO algorithm achieved a very good convergence toward the global minimum at under 300 iterations. Additional iterations further refined the results.

Figures 8.1 and 8.2 show the sheets **Results** and **Project**. The results in the latter confirm the success of the HPO algorithm.

Y	Intercept	X1	X2	X3	X4	YX1	YX2	YX3	YX4
1		2	2.5	2	1.5	1.25	1.3	1.15	1.15
	5	1	1	1	1	1	1	1	1

*Figure 8.1. The sheet **Results** in the Excel file data1\_44FlexHPO1.xlsx.*

Y	X1	X2	X3	X4	Ycalc	%Err
8.299769719	15	14	7	55	8.2997697	-1.7336E-12
9.74036055	34	12	3	56	9.7403605	-1.44073E-12
9.817125639	42	7	20	81	9.8171256	-8.32346E-13
10.65266544	18	14	18	29	10.652665	-1.06721E-12
11.57157737	29	7	39	57	11.571577	-1.68861E-13
12.84063889	42	13	25	46	12.840639	-4.98019E-13
13.50383891	49	10	43	89	13.503839	-2.49935E-13
14.1931634	69	2	12	98	14.193163	-2.50312E-13
14.5647036	51	6	38	54	14.564704	-1.09767E-13
14.57150355	13	6	56	72	14.571504	2.92575E-13
15.0684634	63	14	6	73	15.068463	-5.06909E-13
16.52997144	45	9	49	44	16.529971	-2.14926E-14
16.64406194	80	5	26	99	16.644062	-1.28071E-13
17.43880092	48	14	34	19	17.438801	-1.42607E-13
17.66851327	65	19	18	62	17.668513	-3.41829E-13
17.78641235	69	4	44	67	17.786412	7.98973E-14
17.95785207	19	23	43	63	17.957852	2.1762E-13
18.89233293	40	4	77	99	18.892333	5.64152E-14
19.16351257	42	4	74	85	19.163513	9.26947E-14
19.19148106	63	15	14	22	19.191481	-3.14703E-13
19.28136196	37	26	42	73	19.281362	1.10554E-13
19.74531874	39	29	27	82	19.745319	1.25949E-13
20.02411354	52	8	42	3	20.024114	2.66133E-13
20.31761007	32	22	65	78	20.31761	0
20.81381437	43	30	40	92	20.813814	1.02414E-13
21.5264679	80	17	1	49	21.526468	-2.64063E-13
21.66419612	4	8	84	98	21.664196	1.6399E-14
21.76612866	47	20	52	29	21.766129	9.79333E-14
22.07749391	56	21	9	14	22.077494	-1.93104E-13
22.09774717	33	19	75	70	22.097747	-6.43091E-14

*Figure 8.2. The first 31 rows of sheet **Project** in the Excel file **data1\_44FlexHPO1.xlsx**.*

## 9/The Differential Evolution (DE) Optimization

This section looks at another popular optimization algorithm, the Differential Evolution (DE) method. The test program test\_FlexDE1.m has the following code.

```
clc
close all
clear

optimName = "Differential evolution";

outFile = "bestrathetFlexDE1_";
% You may comment the next statement
delete(strcat(outFile,"*.txt"));

dt = datetime('now','TimeZone','local','Format','y-MMM-d HH_mm_ss');
dtstr = string(dt);
XlFile = strcat(pwd,'\Data1_44FlexDE1.xlsx');
outFile = strcat(outFile,dtstr,".txt");

MaxPop = 1000;
MaxIters = 5000;
[bestMdl1, bestPwrs1] = ...
    bestrathetFlexGenericOptim(XlFile, outFile, ...
        @de,MaxPop,MaxIters,optimName);

clear GLOBAL
fprintf("Done!\n");
```

The test program uses the generic optimization helper function `bestrathetFlexGenericOptim()`. The first parameter for calling this function is the handle of function `de()` which performs the differential evolution method. The test program performs a test using Excel file `data1_44FlexDE1.xlsx` and echoes the console output to diary file `bestrathetFlexDE1_2024-Dec-13 09_08_03.txt`.

The test yields the following console output found in file `bestrathetFlexDE1_2024-Dec-13 09_08_03.txt`. I have trimmed the output to remove lines with somewhat repetitive information.

```
Using optimizer Differential evolution
Please wait ...

2024-Dec-13 09_08_05
Function bestrathetFlexGenericOptim

Excel file used is C:\MATLAB\bestOptimumRatHet\Data1_44FlexDE1.xlsx
Diary file used is bestrathetFlexDE1_2024-Dec-13 09_08_03.txt
```

```

1: Best fx = 2.112555e+03, X=[1.000000, 1.695438, 2.944302, 2.672105,
2.423611, 1.429799, 1.030895, 1.420087, 2.072937]
2: Best fx = 1.196974e+03, X=[1.000000, 2.109121, 2.861606, 1.398906,
2.050639, 1.253478, 1.118673, 1.000000, 1.374348]
4: Best fx = 1.004036e+03, X=[1.000000, 1.000000, 2.917550, 3.000000,
2.195309, 1.000000, 1.000000, 1.000000, 1.572331]
5: Best fx = 1.001673e+03, X=[1.000000, 1.000000, 2.917550, 3.000000,
2.195309, 1.014074, 1.000000, 1.000000, 1.572331]
12: Best fx = 9.202406e+02, X=[1.000000, 1.000000, 2.860179, 3.000000,
1.144660, 1.000000, 1.000000, 1.476364, 1.000000]
16: Best fx = 8.790934e+02, X=[1.000000, 1.460728, 3.000000, 2.709279,
1.000000, 1.161590, 1.000000, 1.448820, 1.000000]
17: Best fx = 6.719882e+02, X=[1.000000, 1.146236, 2.805941, 2.138322,
1.000000, 1.023011, 1.000000, 1.000000, 1.141802]
29: Best fx = 6.205762e+02, X=[1.000000, 1.804547, 2.691962, 1.400943,
1.000000, 1.214752, 1.000000, 1.000000, 1.131880]
35: Best fx = 5.875214e+02, X=[1.000000, 1.591271, 2.691962, 1.733423,
1.000000, 1.214752, 1.000000, 1.000000, 1.131880]
41: Best fx = 5.687469e+02, X=[1.000000, 1.567222, 2.693449, 1.984693,
1.033340, 1.177960, 1.000000, 1.188079, 1.139712]
51: Best fx = 4.895493e+02, X=[1.000000, 2.072412, 2.492578, 1.912826,
1.113113, 1.287645, 1.000000, 1.056790, 1.126040]
67: Best fx = 4.620225e+02, X=[1.000000, 2.072412, 2.492578, 1.901054,
1.113113, 1.287645, 1.000000, 1.115315, 1.126040]
94: Best fx = 4.415438e+02, X=[1.000000, 1.691262, 2.527658, 1.872859,
1.000000, 1.181700, 1.000000, 1.096448, 1.096220]
102: Best fx = 4.290948e+02, X=[1.000000, 1.813992, 2.484189, 1.886907,
1.000000, 1.193842, 1.000000, 1.050768, 1.104194]
117: Best fx = 4.216650e+02, X=[1.000000, 1.703993, 2.526360, 1.917917,
1.000000, 1.193498, 1.000000, 1.100432, 1.098765]
118: Best fx = 4.130369e+02, X=[1.000000, 1.703993, 2.526360, 1.917917,
1.000000, 1.193498, 1.000000, 1.100432, 1.110042]
120: Best fx = 4.120028e+02, X=[1.000000, 1.774098, 2.505819, 1.874872,
1.002943, 1.199866, 1.000101, 1.080434, 1.095624]
132: Best fx = 4.010323e+02, X=[1.000000, 1.813992, 2.484189, 1.886907,
1.000000, 1.223232, 1.000000, 1.050768, 1.127623]
135: Best fx = 3.938845e+02, X=[1.000000, 1.774098, 2.505819, 1.874872,
1.000000, 1.199866, 1.000000, 1.080434, 1.105404]
148: Best fx = 3.876416e+02, X=[1.000000, 1.798935, 2.504965, 1.958956,
1.005030, 1.203112, 1.000023, 1.105999, 1.103098]
160: Best fx = 3.645172e+02, X=[1.000000, 1.834116, 2.491389, 1.979816,
1.051271, 1.228027, 1.000000, 1.097058, 1.123633]
171: Best fx = 3.490129e+02, X=[1.000000, 1.869794, 2.491389, 1.963983,
1.051271, 1.228027, 1.000000, 1.097058, 1.123633]
193: Best fx = 3.472727e+02, X=[1.000000, 1.869794, 2.491389, 1.963983,
1.000000, 1.228027, 1.000000, 1.097058, 1.123633]
195: Best fx = 3.469072e+02, X=[1.000000, 1.869794, 2.491389, 1.963983,
1.000468, 1.228027, 1.000000, 1.108591, 1.123633]
198: Best fx = 3.465892e+02, X=[1.000000, 1.923838, 2.474002, 2.027538,
1.000000, 1.238475, 1.000000, 1.149539, 1.123870]
219: Best fx = 3.462562e+02, X=[1.000000, 1.869794, 2.491389, 1.954604,
1.000468, 1.228027, 1.000000, 1.108591, 1.123633]
230: Best fx = 3.462171e+02, X=[1.000000, 1.986269, 2.470596, 1.940830,
1.000000, 1.254831, 1.000000, 1.105513, 1.113513]
232: Best fx = 3.388101e+02, X=[1.000000, 1.911445, 2.470566, 1.929075,
1.041361, 1.239703, 1.000000, 1.098175, 1.116580]

```

```

234: Best fx = 3.273231e+02, X=[1.000000, 1.866144, 2.452834, 1.994519,
1.000000, 1.216995, 1.000000, 1.127984, 1.110394]
245: Best fx = 3.240657e+02, X=[1.000000, 1.866144, 2.452834, 1.994519,
1.000000, 1.216995, 1.000000, 1.127984, 1.117810]
246: Best fx = 3.232957e+02, X=[1.000000, 1.944531, 2.470415, 1.957212,
1.000000, 1.243506, 1.000000, 1.118272, 1.123151]
260: Best fx = 3.213905e+02, X=[1.000000, 1.946918, 2.458376, 1.961234,
1.000000, 1.243212, 1.000000, 1.108641, 1.118811]
268: Best fx = 3.204080e+02, X=[1.000000, 1.893754, 2.463204, 1.984575,
1.000760, 1.220247, 1.000000, 1.127700, 1.116580]
282: Best fx = 3.192263e+02, X=[1.000000, 1.875086, 2.452834, 1.994519,
1.000000, 1.216995, 1.000000, 1.127984, 1.117810]
286: Best fx = 3.184723e+02, X=[1.000000, 1.893754, 2.450177, 1.984575,
1.000535, 1.220247, 1.000000, 1.127700, 1.116580]
288: Best fx = 3.155709e+02, X=[1.000000, 1.925003, 2.450177, 1.984575,
1.003951, 1.226174, 1.000000, 1.127700, 1.116580]
319: Best fx = 3.141321e+02, X=[1.000000, 1.894287, 2.452126, 1.988283,
1.000000, 1.221035, 1.000000, 1.125973, 1.116852]
337: Best fx = 3.141190e+02, X=[1.000000, 1.894287, 2.452126, 1.988283,
1.000000, 1.221035, 1.000000, 1.125973, 1.116568]
378: Best fx = 3.129676e+02, X=[1.000000, 1.931894, 2.453950, 1.990468,
1.000000, 1.227376, 1.000001, 1.131002, 1.120259]
379: Best fx = 3.129675e+02, X=[1.000000, 1.931894, 2.453950, 1.990468,
1.000000, 1.227376, 1.000000, 1.131002, 1.120259]
406: Best fx = 3.123378e+02, X=[1.000000, 1.927647, 2.451917, 1.979343,
1.018490, 1.229206, 1.000000, 1.128531, 1.118382]
409: Best fx = 3.121265e+02, X=[1.000000, 1.927647, 2.451917, 1.980191,
1.018490, 1.229206, 1.000000, 1.128531, 1.118382]
466: Best fx = 3.120644e+02, X=[1.000000, 1.927647, 2.451917, 1.980191,
1.019330, 1.229206, 1.000000, 1.128531, 1.118382]
513: Best fx = 3.114008e+02, X=[1.000000, 1.935014, 2.453683, 1.983334,
1.026105, 1.232449, 1.000000, 1.130807, 1.121513]
609: Best fx = 3.110380e+02, X=[1.000000, 1.934077, 2.453683, 1.983334,
1.026105, 1.232449, 1.000000, 1.130807, 1.121513]
619: Best fx = 3.107614e+02, X=[1.000000, 1.934077, 2.453683, 1.983334,
1.027453, 1.232449, 1.000000, 1.130807, 1.121513]
707: Best fx = 3.105877e+02, X=[1.000000, 1.936204, 2.454328, 1.980953,
1.054533, 1.232468, 1.000000, 1.131044, 1.123602]
738: Best fx = 3.105721e+02, X=[1.000000, 1.936204, 2.454328, 1.980953,
1.054533, 1.232468, 1.000000, 1.132074, 1.123602]
896: Best fx = 3.104758e+02, X=[1.000000, 1.934077, 2.453683, 1.983334,
1.034553, 1.232449, 1.000000, 1.130807, 1.121513]
900: Best fx = 3.103064e+02, X=[1.000000, 1.934077, 2.453683, 1.983334,
1.030204, 1.232449, 1.000000, 1.130807, 1.121513]
909: Best fx = 3.101385e+02, X=[1.000000, 1.934077, 2.453683, 1.983334,
1.040198, 1.232449, 1.000000, 1.130807, 1.122599]
996: Best fx = 3.098513e+02, X=[1.000000, 1.934077, 2.453683, 1.983334,
1.042660, 1.232449, 1.000000, 1.130807, 1.122599]
1076: Best fx = 3.098194e+02, X=[1.000000, 1.933328, 2.453396, 1.983469,
1.052083, 1.231871, 1.000000, 1.131195, 1.122969]
1144: Best fx = 3.098151e+02, X=[1.000000, 1.935143, 2.453683, 1.983334,
1.042660, 1.232449, 1.000000, 1.130807, 1.122510]
1154: Best fx = 3.097563e+02, X=[1.000000, 1.933328, 2.453396, 1.983469,
1.052083, 1.231871, 1.000000, 1.131195, 1.123177]
1169: Best fx = 3.096513e+02, X=[1.000000, 1.935143, 2.453683, 1.983334,
1.044987, 1.232449, 1.000000, 1.130807, 1.122510]

```

```

1234: Best fx = 3.094527e+02, X=[1.000000, 1.934378, 2.453683, 1.983334,
1.044987, 1.232449, 1.000000, 1.130807, 1.122510]
----- New population at iter 1484
----- New population at iter 1734
----- New population at iter 1984
----- New population at iter 2234
----- New population at iter 2484
----- New population at iter 2734
----- New population at iter 2984
----- New population at iter 3234
----- New population at iter 3484
----- New population at iter 3734
----- New population at iter 3984
----- New population at iter 4234
----- New population at iter 4484
----- New population at iter 4734
----- New population at iter 4984

Results (powers/coefficients)

bestPwrs =

      1.0000      1.9344      2.4537      1.9833      1.0450      1.2324      1.0000
1.1308      1.1225

Regression model
Adj R-sqr =      0.99929788

bestMdl =

Linear regression model (robust fit):
      y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8

Estimated Coefficients:
              Estimate              SE              tStat              pValue
-----
(Intercept)      15.1623532331381      0.326956411892552      46.3742342453921      4.31613374786415e-122
x1      -0.0144898874008123      0.000178984976673235      -80.9558861873965      9.41008220151673e-177
x2      -0.0135003895850575      0.000167362205251949      -80.66570086558      2.16828298103531e-176
x3      -0.0112862066743431      0.000151824737901518      -74.3370733276945      3.61620352641959e-168
x4      -0.127193755799467      0.00370412012156074      -34.3384532966694      8.47305671561111e-95
x5      -0.0117030801595222      0.000144427440516704      -81.0308630939744      7.58791790402717e-177
x6      -0.0523868198296077      0.000540601359992184      -96.9047133554475      5.17303513790037e-195
x7      -0.0116443553491795      0.000147912271242774      -78.7247417090035      6.20157890301531e-174
x8      -0.0125680721896249      0.000154955030398335      -81.1078682461408      6.08424842644509e-177

Number of observations: 250, Error degrees of freedom: 241
Root Mean Squared Error: 1.13
R-squared: 0.999, Adjusted R-Squared: 0.999
F-statistic vs. constant model: 4.43e+04, p-value = 0

```

The results of the test program seem to guide the DE algorithm to find a local minimum. This is somewhat disappointing as my aim is to use error-free data to have an optimization algorithm find the global minimum. The DE algorithm has failed to do that.

Figures 9.1 and 9.2 show the sheets **Results** and **Project..**

Y	Intercept	X1	X2	X3	X4	YX1	YX2	YX3	YX4
1		1.934377635	2.453683398	1.983334486	1.044987395	1.232449273	1	1.130807384	1.122510257
	15.16235323	-0.014489887	-0.01350039	-0.011286207	-0.127193756	-0.01170308	-0.05238682	-0.011644355	-0.012568072

*Figure 9.1. The sheet **Results** in the Excel file data1\_44FlexDE1.xlsx.*



Y	X1	X2	X3	X4	Ycalc	%Err
8.299769719	15	14	7	55	4.0401938	-51.32161573
9.74036055	34	12	3	56	7.4025468	-24.00130578
9.817125639	42	7	20	81	8.8636776	-9.71208998
10.65266544	18	14	18	29	5.2472134	-50.74271847
11.57157737	29	7	39	57	10.432424	-9.844412683
12.84063889	42	13	25	46	11.609466	-9.58809568
13.50383891	49	10	43	89	13.269924	-1.732214055
14.1931634	69	2	12	98	14.933829	5.218465835
14.5647036	51	6	38	54	14.727656	1.118820262
14.57150355	13	6	56	72	14.487899	-0.573751384
15.0684634	63	14	6	73	14.848082	-1.462535564
16.52997144	45	9	49	44	16.57306	0.260666932
16.64406194	80	5	26	99	17.279139	3.815635452
17.43880092	48	14	34	19	16.570592	-4.978604773
17.66851327	65	19	18	62	17.470402	-1.12126533
17.78641235	69	4	44	67	18.8677	6.079289483
17.95785207	19	23	43	63	17.274938	-3.802869483
18.89233293	40	4	77	99	19.892837	5.295819165
19.16351257	42	4	74	85	20.337918	6.128339077
19.19148106	63	15	14	22	18.869212	-1.67922951
19.28136196	37	26	42	73	18.976516	-1.581041219
19.74531874	39	29	27	82	19.522735	-1.127273044
20.02411354	52	8	42	3	20.397188	1.863127209
20.31761007	32	22	65	78	20.096869	-1.086450298
20.81381437	43	30	40	92	20.671199	-0.685196647
21.5264679	80	17	1	49	21.778585	1.171193646
21.66419612	4	8	84	98	22.583716	4.244420659
21.76612866	47	20	52	29	21.483936	-1.296476454
22.07749391	56	21	9	14	21.505225	-2.592093689
22.09774717	33	19	75	70	22.091759	-0.027099105

*Figure 9.2. The first 31 rows of sheet **Project** in the Excel file data1\_44FlexDE1.xlsx.*

## 10/The Enhanced Vibrating Particles System

This section looks at another not so popular optimization algorithm, the Enhanced Vibrating Particles System method. The test program test\_FlexEVPS1.m has the following code.

```
clc
close allclc
close all
clear

optimName = "Enhanced VIBRATING PARTICLES SYSTEM";

outFile = "bestrathetFlexEVSP1_";
% You may comment the next statement
delete(strcat(outFile,"*.txt"));

dt = datetime('now','TimeZone','local','Format','y-MMM-d HH_mm_ss');
dtstr = string(dt);
XlFile = strcat(pwd,'\data1_44FlexEVPS1.xlsx');
outFile = strcat(outFile,dtstr,".txt");

MaxPop = 5000;
MaxIters = 5000;
[bestMdl1, bestPwrs1] = ...
    bestrathetFlexGenericOptim(XlFile, outFile, ...
    @evpsFx,MaxPop,MaxIters,optimName);

clear GLOBAL
fprintf("Done!\n");
```

The test program uses the generic optimization helper function bestrathetFlexGenericOptim(). The first parameter for calling this function is the handle of function evpsFx() which performs the differential evolution method. The test program performs a test using Excel file data1\_44FlexEVPS1.xlsx and echoes the console output to diary file bestrathetFlexEVPS1\_2024-Dec-14 17\_27\_41.txt.

The test yields the following console output found in file bestrathetFlexEVPS1\_2024-Dec-13 14\_40\_44.txt. I have trimmed the output to remove lines with somewhat repetitive information.

```
Using optimizer Enhanced VIBRATING PARTICLES SYSTEM
Please wait ...

2024-Dec-14 17_27_41
```

Function bestrathetFlexGenericOptim

Excel file used is C:\MATLAB\bestOptimumRatHet\data1\_44FlexEVPS1.xlsx

Diary file used is bestrathetFlexEVPS1\_2024-Dec-14 17\_27\_41.txt

Iter: 1, Fx = 2.758018e+03, X=[1.000000, 2.736902, 2.593505, 2.146775, 2.460373, 1.628756, 2.161719, 1.183694, 1.347580]  
 Iter: 2, Fx = 1.405661e+03, X=[1.000000, 2.430731, 2.344936, 2.126510, 2.266502, 1.468217, 2.706119, 1.193810, 1.157513]  
 Iter: 4, Fx = 1.393843e+03, X=[1.000000, 2.503873, 2.114491, 2.011667, 2.273900, 1.230341, 2.278715, 1.189600, 1.072796]  
 Iter: 5, Fx = 1.351438e+03, X=[1.000000, 2.681169, 2.133881, 2.373374, 1.337646, 1.373947, 2.816198, 1.456653, 1.242443]  
 Iter: 6, Fx = 1.231882e+03, X=[1.000000, 2.126909, 2.537299, 1.889210, 2.310861, 1.239568, 2.025508, 1.228999, 1.196694]  
 Iter: 7, Fx = 1.210114e+03, X=[1.000000, 2.922571, 2.431686, 2.068994, 1.685296, 1.394072, 2.605368, 1.283029, 1.156931]  
 Iter: 8, Fx = 1.140046e+03, X=[1.000000, 2.362010, 2.218646, 2.486962, 1.739471, 1.230341, 2.025141, 1.361157, 1.143847]  
 Iter: 9, Fx = 9.571863e+02, X=[1.000000, 2.060886, 2.427098, 2.213249, 2.696294, 1.238552, 1.994459, 1.273999, 1.202589]  
 Iter: 10, Fx = 9.410586e+02, X=[1.000000, 2.206832, 2.360461, 2.215357, 1.520781, 1.274234, 2.704888, 1.323125, 1.178332]  
 Iter: 13, Fx = 8.113641e+02, X=[1.000000, 2.102181, 2.344218, 1.998186, 1.120190, 1.269960, 1.961380, 1.135553, 1.076849]  
 Iter: 16, Fx = 7.900961e+02, X=[1.000000, 2.166559, 2.346308, 1.990589, 1.116418, 1.278118, 2.146233, 1.156937, 1.146145]  
 Iter: 19, Fx = 7.022263e+02, X=[1.000000, 2.185409, 2.481876, 2.012190, 1.842277, 1.274395, 1.846729, 1.225639, 1.107440]

.....

Iter: 1266, Fx = 3.082987e+02, X=[1.000000, 1.923370, 2.454919, 1.987407, 1.178514, 1.230863, 1.000000, 1.129552, 1.134533]  
 Iter: 1290, Fx = 3.082987e+02, X=[1.000000, 1.923370, 2.454919, 1.987407, 1.178514, 1.230863, 1.000000, 1.129552, 1.134533]  
 Iter: 1300, Fx = 3.082987e+02, X=[1.000000, 1.923370, 2.454919, 1.987407, 1.178514, 1.230863, 1.000000, 1.129552, 1.134533]  
 Iter: 1365, Fx = 3.082987e+02, X=[1.000000, 1.923370, 2.454919, 1.987407, 1.178514, 1.230863, 1.000000, 1.129552, 1.134533]  
 Iter: 1449, Fx = 3.082987e+02, X=[1.000000, 1.923370, 2.454919, 1.987407, 1.178514, 1.230863, 1.000000, 1.129552, 1.134533]  
 Iter: 1494, Fx = 3.082987e+02, X=[1.000000, 1.923370, 2.454919, 1.987407, 1.178514, 1.230863, 1.000000, 1.129552, 1.134533]  
 Iter: 1552, Fx = 3.082987e+02, X=[1.000000, 1.923370, 2.454919, 1.987407, 1.178514, 1.230863, 1.000000, 1.129552, 1.134533]  
 Iter: 1582, Fx = 3.082987e+02, X=[1.000000, 1.923370, 2.454919, 1.987407, 1.178514, 1.230863, 1.000000, 1.129552, 1.134533]  
 Iter: 1601, Fx = 3.082987e+02, X=[1.000000, 1.923370, 2.454919, 1.987407, 1.178514, 1.230863, 1.000000, 1.129552, 1.134533]  
 ----- New population at iter 1701  
 ----- New population at iter 1801  
 ----- New population at iter 1901

Results (powers/coefficients)

bestPwrs =

```

Columns 1 through 7
    1.0000    1.9234    2.4549    1.9874    1.1785    1.2309    1.0000

Columns 8 through 9
    1.1296    1.1345

Regression model
Adj R-sqr =      0.99930083

bestMdl =

Linear regression model (robust fit):
    y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8

Estimated Coefficients:
              Estimate              SE              tStat              pValue
-----
(Intercept)    14.8247216092148      0.314523519976172      47.1339046769471      1.26659386063461e-123
x1             -0.0151289137786852      0.00018658883778353     -81.0815585669541      6.56097058839601e-177
x2             -0.0133206767428669      0.000164663259730301    -80.8964717732698      1.11614395688453e-176
x3             -0.0109680128257782      0.000147369853380772    -74.4250779529462      2.75147426544429e-168
x4             -0.0680493874032878      0.00196109566059022     -34.6996777213852      1.03817685868841e-95
x5             -0.0117000177127135      0.000144043331760498    -81.2256809788822      4.34136267424055e-177
x6             -0.0521123565190364      0.000535301046491948     -97.35149381932      1.75484534141241e-195
x7             -0.0116048996525816      0.000147084842493596    -78.8993580564691      3.70878685040575e-174
x8             -0.0117861980475934      0.000145208201273604    -81.1675783063078      5.12733670623092e-177

Number of observations: 250, Error degrees of freedom: 241
Root Mean Squared Error: 1.13
R-squared: 0.999, Adjusted R-Squared: 0.999
F-statistic vs. constant model: 4.45e+04, p-value = 0

```

The results of the test program seem to guide the EVPS algorithm to find a local minimum. This is somewhat disappointing as my aim is to use error-free data to have an optimization algorithm find the global minimum. The EVPS algorithm has failed to do that.

Figures 10.1 and 10.2 show the sheets **Results** and **Project**..

Y	Intercept	X1	X2	X3	X4	YX1	YX2	YX3	YX4
1		1.923370195	2.45491946	1.98740661	1.178513928	1.230863034	1	1.12955242	1.134533382
	14.82472161	-0.015128914	-0.013320677	-0.010968013	-0.068049387	-0.011700018	-0.052112357	-0.0116049	-0.011786198

Figure 10.1. The sheet **Results** in the Excel file *data1\_44FlexEVPS1.xlsx*.

Y	X1	X2	X3	X4	Ycalc	%Err
8.299769719	15	14	7	55	3.763469	-54.65574055
9.74036055	34	12	3	56	7.291338	-25.14304211
9.817125639	42	7	20	81	8.899146	-9.350803561
10.65266544	18	14	18	29	4.893575	-54.06243715
11.57157737	29	7	39	57	10.31225	-10.88296339
12.84063889	42	13	25	46	11.52151	-10.2730747
13.50383891	49	10	43	89	13.3239	-1.332476892
14.1931634	69	2	12	98	15.06504	6.142899392
14.5647036	51	6	38	54	14.68728	0.841621643
14.57150355	13	6	56	72	14.40777	-1.123636738
15.0684634	63	14	6	73	14.88056	-1.247000615
16.52997144	45	9	49	44	16.50833	-0.130930501
16.64406194	80	5	26	99	17.3749	4.390994445
17.43880092	48	14	34	19	16.55582	-5.063335673
17.66851327	65	19	18	62	17.46627	-1.144671667
17.78641235	69	4	44	67	18.87875	6.141393234
17.95785207	19	23	43	63	17.18675	-4.293930119
18.89233293	40	4	77	99	19.97758	5.744392047
19.16351257	42	4	74	85	20.38186	6.357655721
19.19148106	63	15	14	22	18.88271	-1.608890266
19.28136196	37	26	42	73	18.95832	-1.675416011
19.74531874	39	29	27	82	19.53968	-1.041450523
20.02411354	52	8	42	3	20.5811	2.781563511
20.31761007	32	22	65	78	20.0851	-1.144397099
20.81381437	43	30	40	92	20.71007	-0.498420151
21.5264679	80	17	1	49	21.77343	1.14724906
21.66419612	4	8	84	98	22.63981	4.503367195
21.76612866	47	20	52	29	21.43856	-1.504955373
22.07749391	56	21	9	14	21.56227	-2.333687783
22.09774717	33	19	75	70	22.06866	-0.131634179

*Figure 10.2. The first 31 rows of sheet **Project** in the Excel file **data1\_44FlexEVPS1.xlsx**.*

Figure 10.3 compares the results of the DE and EVPS algorithms. You can see that both algorithms have zoomed in at the same local minimum by evidence of the closeness of the powers and coefficients of the results. In experimenting with the power ranges, I have discovered if I use 0.5 as the lower range of powers for the independent variables, these algorithms zoom in on yet another local minimum.

Alorithm	Y	Intercept	X1	X2	X3	X4	YX1	YX2	YX3	YX4
EVPS	1		1.923370195	2.45491946	1.98740661	1.178513928	1.230863034	1	1.12955242	1.134533382
		14.82472161	-0.015128914	-0.013320677	-0.010968013	-0.068049387	-0.011700018	-0.052112357	-0.0116049	-0.011786198
DE	1		1.934377635	2.453683398	1.983334486	1.044987395	1.232449273	1	1.130807384	1.122510257
		15.16235323	-0.014489887	-0.01350039	-0.011286207	-0.127193756	-0.01170308	-0.05238682	-0.011644355	-0.012568072

*Figure 10.3. Comparing the results of the DE and EVPS algorithm.*

## 11/The Ant Colony Optimization (ACO)

This section looks at another popular optimization algorithm, the Ant Colony Optimization algorithm. The test program test\_FlexACO1.m has the following code.

```
clc
close all
clear

optimName = "Ant Colony Optimization";

outFile = "besttrathetFlexACO1_";
% You may comment the next statement
delete(strcat(outFile,"*.txt"));

dt = datetime('now','TimeZone','local','Format','y-MMM-d HH_mm_ss');
dtstr = string(dt);
XlFile = strcat(pwd,'\Data1_44FlexACO1.xlsx');
outFile = strcat(outFile,dtstr,".txt");

MaxPop = 500;
MaxIters = 2000;
[bestMdl1, bestPwrs1] = ...
    bestrathetFlexGenericOptim(XlFile, outFile, ...
    @aco,MaxPop,MaxIters,optimName);

clear GLOBAL
fprintf("Done!\n");
```

The test program uses the generic optimization helper function `bestrathetFlexGenericOptim()`. The first parameter for calling this function is the handle of function `evpsFx()` which performs the differential evolution method. The test program performs a test using Excel file `data1_44FlexACO1.xlsx` and echoes the console output to diary file `bestrathetFlexACO1_2024-Dec-13 08_40_20.txt`.

The test yields the following console output found in file `bestrathetFlexACO1_2024-Dec-13 08_40_20.txt`. I have trimmed the output to remove lines with somewhat repetitive information.

```
Using optimizer Ant Colony Optimization
Please wait ...

2024-Dec-13 08_40_20
Function bestrathetFlexGenericOptim

Excel file used is
C:\DropBox\Dropbox\MATLAB\bestOptimumRatHet\Data1_44FlexACO1.xlsx
Diary file used is bestrathetFlexACO1_2024-Dec-13_08_40_20.txt
Iter 0, Fx = 2.211814e+03, X=[1.000000, 2.605931, 2.998956, 2.961956,
1.254074, 1.464480, 1.047265, 2.214865, 1.221619]
Iter 1, Fx = 2.114980e+03, X=[1.000000, 2.610365, 1.963749, 2.469052,
1.214691, 1.219510, 2.922778, 1.101344, 1.407881]
Iter 2, Fx = 1.585481e+03, X=[1.000000, 1.658919, 2.302772, 1.967872,
1.102516, 1.138185, 2.462689, 1.222997, 1.246461]
Iter 4, Fx = 9.200404e+02, X=[1.000000, 2.527078, 2.424148, 2.127835,
1.162022, 1.308600, 2.430003, 1.274069, 1.098100]
Iter 5, Fx = 8.568707e+02, X=[1.000000, 1.817560, 2.854629, 1.987812,
1.407470, 1.288282, 1.087768, 1.213039, 1.225168]
Iter 7, Fx = 7.441781e+02, X=[1.000000, 2.270784, 2.481321, 2.074773,
2.065561, 1.302270, 1.946025, 1.211445, 1.166519]
Iter 15, Fx = 6.224636e+02, X=[1.000000, 2.170004, 2.478068, 2.082279,
1.001865, 1.276413, 1.833609, 1.190839, 1.123716]
Iter 24, Fx = 6.080516e+02, X=[1.000000, 2.175712, 2.489960, 2.098034,
2.584816, 1.280863, 1.798876, 1.218607, 1.136351]

.....

Iter 25, Fx = 6.056235e+02, X=[1.000000, 2.207345, 2.469692, 2.092293,
Iter 444, Fx = 1.801918e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter 452, Fx = 1.777533e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter 481, Fx = 1.764355e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter 482, Fx = 1.742642e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]
Iter 484, Fx = 1.691495e-19, X=[1.000000, 2.000000, 2.500000, 2.000000,
1.500000, 1.250000, 1.300000, 1.150000, 1.150000]

Results (powers/coefficients)
```

```

bestPwrs =

    Columns 1 through 8

    1.0000    2.0000    2.5000    2.0000    1.5000    1.2500    1.3000
1.1500

    Column 9

    1.1500

Regression model
Adj R-sqr =      1.00000000

bestMdl =

Linear regression model (robust fit):
    y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8

Estimated Coefficients:

```

	Estimate	SE	tStat	pValue
(Intercept)	4.9999999999222	8.24720434505201e-12	606266049778.677	0
x1	1.00000000000181	2.27547081694683e-13	4394694902497.53	0
x2	1.00000000000181	2.27397249933953e-13	4397590561417.31	0
x3	1.00000000000186	2.27340286562968e-13	4398692440835.29	0
x4	1.00000000000144	2.31346641812229e-13	4322517898544.12	0
x5	1.00000000000018	2.27433446449178e-13	4396890675555.33	0
x6	1.00000000000184	2.27897933328974e-13	4387929216358.14	0
x7	1.00000000000183	2.27419604410946e-13	4397158295090.67	0
x8	1.00000000000178	2.27490824327454e-13	4395781689033.61	0

```

Number of observations: 250, Error degrees of freedom: 241
Root Mean Squared Error: 2.65e-11
R-squared: 1, Adjusted R-Squared: 1
F-statistic vs. constant model: 8.01e+25, p-value = 0

```

The results of the test program show that the Ant Colony Optimization algorithm was able to find the global minimum, based on the power ranges in Figure 3.2. The powers of [1, 2, 2.5, 2, 1.5, 1.25, 1.3, 1.15, 1.15] are the correct ones. The intercept of the numerator part is very close to 5 and all the coefficients of the various terms are very close to 1.

Figures 11.1 and 11.2 show the sheets **Results** and **Project**.



Y	Intercept	X1	X2	X3	X4	YX1	YX2	YX3	YX4
1		2	2.5	2	1.5	1.25	1.3	1.15	1.15
	5	1	1	1	1	1	1	1	1

Figure 11.1. The sheet **Results** in the Excel file *data1\_44FlexACO1.xlsx*.

Y	X1	X2	X3	X4	Ycalc	%Err
8.299769719	15	14	7	55	8.2997697	-1.56238E-12
9.74036055	34	12	3	56	9.7403605	-9.84802E-13
9.817125639	42	7	20	81	9.8171256	-2.89512E-13
10.65266544	18	14	18	29	10.652665	-8.00411E-13
11.57157737	29	7	39	57	11.571577	-1.68861E-13
12.84063889	42	13	25	46	12.840639	-3.04345E-13
13.50383891	49	10	43	89	13.503839	2.63089E-14
14.1931634	69	2	12	98	14.193163	3.75467E-14
14.5647036	51	6	38	54	14.564704	-7.31779E-14
14.57150355	13	6	56	72	14.571504	8.53344E-14
15.0684634	63	14	6	73	15.068463	-2.59349E-13
16.52997144	45	9	49	44	16.529971	-2.14926E-14
16.64406194	80	5	26	99	16.644062	2.13452E-14
17.43880092	48	14	34	19	17.438801	1.22235E-13
17.66851327	65	19	18	62	17.668513	-2.21184E-13
17.78641235	69	4	44	67	17.786412	1.99743E-14
17.95785207	19	23	43	63	17.957852	-3.95672E-14
18.89233293	40	4	77	99	18.892333	1.88051E-13
19.16351257	42	4	74	85	19.163513	1.11234E-13
19.19148106	63	15	14	22	19.191481	-5.55358E-14
19.28136196	37	26	42	73	19.281362	0
19.74531874	39	29	27	82	19.745319	3.59854E-14
20.02411354	52	8	42	3	20.024114	4.43554E-13
20.31761007	32	22	65	78	20.31761	0
20.81381437	43	30	40	92	20.813814	8.53451E-14
21.5264679	80	17	1	49	21.526468	-2.47559E-13
21.66419612	4	8	84	98	21.664196	9.83941E-14
21.76612866	47	20	52	29	21.766129	1.14255E-13
22.07749391	56	21	9	14	22.077494	3.2184E-14
22.09774717	33	19	75	70	22.097747	-4.82318E-14

*Figure 11.2. The first 31 rows of sheet **Project** in the Excel file data1\_44FlexACO1.xlsx.*

## 12/Arithmetic Optimization Algorithm (AOA).

This section looks at another not so popular optimization algorithm, the Arithmetic Optimization Algorithm (AOA) method. The test program test\_FlexAOA1.m has the following code.

```
clc
close all
clear

optimName = "Arithmetic Optimization Algorithm";

outFile = "besttrathetFlexAOA1_";
% You may comment the next statement
delete(strcat(outFile,"*.txt"));

dt = datetime('now','TimeZone','local','Format','y-MMM-d HH_mm_ss');
dtstr = string(dt);
XlFile = strcat(pwd,'\Data1_44FlexAOA1.xlsx');
outFile = strcat(outFile,dtstr,".txt");

MaxPop = 1000;
MaxIters = 5000;
[bestMdl1, bestPwrs1] = ...
    bestrathetFlexGenericOptim(XlFile, outFile, ...
    @AOAfx,MaxPop,MaxIters,optimName);

clear GLOBAL
fprintf("Done!\n");
```

The test program uses the generic optimization helper function bestrathetFlexGenericOptim. The first parameter for calling this function is the handle of function AOAfx() which performs the differential evolution method. The test program performs a test using Excel file data1\_44FlexAOA1.xlsx and echoes the console output to diary file bestrathetFlexAOA1\_2024-Dec-13 19\_18\_15.txt.

The test yields the following console output found in file bestrathetFlexAOA1\_2024-Dec-13 19\_18\_15.txt.

```
Using optimizer Arithmetic Optimization Algorithm
Please wait ...

2024-Dec-13 19_18_15
Function bestrathetFlexGenericOptim

Excel file used is
I:\DropBox\Dropbox\MATLAB\bestOptimumRatHet\Data1_44FlexAOA1.xlsx
Diary file used is bestrathetFlexAOA1_2024-Dec-13 19_18_15.txt
```

```

Itrs: 11, Fx = 1.574221e+03, X=[1.000000, 2.585664, 2.327150, 1.769776,
2.052924, 1.404817, 2.997958, 1.080757, 1.264544]
Itrs: 45, Fx = 1.464046e+03, X=[1.000000, 1.692211, 2.901867, 2.905168,
1.281485, 1.178238, 1.135255, 1.496712, 1.142010]
Itrs: 90, Fx = 1.189294e+03, X=[1.000000, 2.600172, 2.181167, 2.178927,
1.290613, 1.186631, 2.320764, 1.192532, 1.150144]
Itrs: 582, Fx = 9.025339e+02, X=[1.000000, 2.041487, 2.739852, 1.724784,
1.239909, 1.139747, 1.013381, 1.155357, 1.151676]
Itrs: 1464, Fx = 7.598838e+02, X=[1.000000, 1.416403, 2.736385, 1.711175,
1.643486, 1.075846, 1.026514, 1.083210, 1.331540]
Itrs: 1674, Fx = 7.298943e+02, X=[1.000000, 2.177793, 2.449476, 2.185635,
1.423448, 1.275149, 1.959086, 1.182086, 1.111501]

Results (powers/coefficients)

bestPwrs =

      1.0000      2.1778      2.4495      2.1856      1.4234      1.2751      1.9591
1.1821      1.1115

Regression model
Adj R-sqr =      0.99833668

bestMdl =

Linear regression model (robust fit):
      y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8

Estimated Coefficients:

```

	Estimate	SE	tStat	pValue
(Intercept)	21.3200203423601	0.392169712516008	54.3642705235425	2.8024366440379e-137
x1	0.0028418014184246	5.13943638744301e-05	55.2940284535453	6.36575089107275e-139
x2	0.00784074937520921	8.98741467341895e-05	87.2414332722278	2.53904958582507e-184
x3	0.00290320460085813	4.86928252040322e-05	59.6228415314401	2.82136704422723e-146
x4	-0.0034358074103377	0.000974746089206582	-3.524822975319	0.000507007847901353
x5	0.00575809006812991	7.65106600894232e-05	75.2586641051069	2.09745155445014e-169
x6	0.00018944104737198	3.27497339349858e-06	57.8450645577931	2.58598721352975e-143
x7	0.0056431189276811	8.28772778200002e-05	68.0900613065175	2.13445540289488e-159
x8	0.00729352648390559	0.000107973766156284	67.5490606982143	1.32681595860399e-158

```

Number of observations: 250, Error degrees of freedom: 241
Root Mean Squared Error: 1.74
R-squared: 0.998, Adjusted R-Squared: 0.998
F-statistic vs. constant model: 1.87e+04, p-value = 0

```

The results of the test program seem to guide the AOA to find a local minimum. This is somewhat disappointing as my aim is to use error-free data to have an optimization algorithm find the global minimum. The AOA algorithm has failed to do that.

Figures 12.1 and 12.2 show the sheets **Results** and **Project**..

Y	Intercept	X1	X2	X3	X4	YX1	YX2	YX3	YX4
1		2.177793484	2.449476055	2.185634821	1.423447596	1.275149014	1.959085538	1.182086457	1.1115012
	21.32002034	0.002841801	0.007840749	0.002903205	-0.003435807	0.00575809	0.000189441	0.005643119	0.007293526

*Figure 12.1. The sheet **Results** in the Excel file data1\_44FlexAOA1.xlsx.*

Y	X1	X2	X3	X4	Ycalc	%Err
8.29976972	15	14	7	55	13.9885383	68.54128174
9.74036055	34	12	3	56	13.5772687	39.39184909
9.81712564	42	7	20	81	11.329366	15.40410558
10.6526654	18	14	18	29	16.6896277	56.67090815
11.5715774	29	7	39	57	13.6252637	17.74767807
12.8406389	42	13	25	46	15.2693323	18.91411625
13.5038389	49	10	43	89	13.5301854	0.195103384
14.1931634	69	2	12	98	13.5477519	-4.547340915
14.5647036	51	6	38	54	15.175108	4.190983763
14.5715035	13	6	56	72	15.1851826	4.211500906
15.0684634	63	14	6	73	15.7848906	4.754480956
16.5299714	45	9	49	44	17.1041431	3.473518816
16.6440619	80	5	26	99	15.5917765	-6.322287149
17.4388009	48	14	34	19	19.0676218	9.340211167
17.6685133	65	19	18	62	18.3929248	4.100013865
17.7864124	69	4	44	67	16.963827	-4.624796588
17.9578521	19	23	43	63	19.4790237	8.470788205
18.8923329	40	4	77	99	17.4873251	-7.436920486
19.1635126	42	4	74	85	17.9250487	-6.46261423
19.1914811	63	15	14	22	20.3783132	6.184161325
19.281362	37	26	42	73	20.0169985	3.815272815
19.7453187	39	29	27	82	20.4849949	3.746083738
20.0241135	52	8	42	3	20.210509	0.930855222
20.3176101	32	22	65	78	20.4591259	0.696517936
20.8138144	43	30	40	92	20.9505734	0.657058782
21.5264679	80	17	1	49	21.6863842	0.742882125
21.6641961	4	8	84	98	20.2917379	-6.335144752
21.7661287	47	20	52	29	22.3905597	2.868820103
22.0774939	56	21	9	14	23.5754854	6.785151965
22.0977472	33	19	75	70	21.9726785	-0.5659791

*Figure 12.2. The first 31 rows of sheet **Project** in the Excel file *data1\_44FlexAOA1.xlsx*.*

## Conclusion

The particle swarm optimization, cuckoo search, hunter prey search, and ant colony optimization were methods that succeeded in finding the global minimum.

## Appendix A

This appendix lists the MATLAB code for the optimization functions used in the earlier code. The subsections list the code and briefly discuss the input and output parameters. I have tweaked the original code to streamline the input and output parameters and adapt the functions to handle iteration stalling. For details on how each function works I encourage you to search the Internet for the details of these algorithms. As I post all this material free of charge on my web site, I do not financially benefit.

### The Particle Swarm Optimization (PSO)

The MATLAB code for the function pso() is:

```
function [gBestX,gBestCost] = pso(fx,Lb,Ub,MaxPop,MaxIters,itsToReset)
% PSO implements particle swarm optimization.
%
% INPUT
% =====
% fx - handle of optimized function.
% Lb - array of low bound values.
% Ub - array of upper bound values.
% MaxPop - maximum population of swarm.
% MaxIters - maximum number of iterations
% itsToReset - number of stalled iterations that lead to data reset.
%
% OUTPUT
% =====
% gBestX - array of best solutions.
% gBestCost - best optimized function value.
%
% Example
% =====
%
% >> [gBestX,gBestCost] = pso(@fx2,[0 0 0 0],[5 5 5 5],50,500)
%
%
nVar = length(Lb);
template.position = zeros(1,nVar);
template.velocity = zeros(1,nVar);
```



```

template.cost = 0;
template.best_position = zeros(1,nVar);
template.best_cost = 1e+99;

c1 = 1.5;
c2 = 2;
w = 1;
wdamp = 0.995;
gBestCost = 1e+99;
gBestX = zeros(1,nVar);

% Initizialize population
for i=1:MaxPop
    pop(i) = template;
    for j=1:nVar
        pop(i).position(j) = Lb(j) + (Ub(j) - Lb(j))*rand;
        pop(i).velocity(j) = 0;
    end

    pop(i).cost = fx(pop(i).position);

    if pop(i).cost < pop(i).best_cost
        pop(i).best_cost = pop(i).cost;
        pop(i).best_position = pop(i).position;
    end

    if pop(i).best_cost < gBestCost
        gBestCost = pop(i).best_cost;
        gBestX = pop(i).best_position;
    end
end

fprintf('Best Fx = %e, ', gBestCost);
fprintf('Best X =[');
fprintf('%f, ', gBestX(1:nVar-1));
fprintf(' %f]\n', gBestX(nVar));
randSearchSuccessCount = 0;
lastIter = 0;
% pso loop
for iter = 1:MaxIters
    if gBestCost == 0, return; end
    if (iter - lastIter) > itersToReset
        fprintf("----- Reset population -----
-- \n")
        fprintf("AT iteration %d\n", iter);
        lastIter = iter;
        for i=1:MaxPop
            pop(i) = template;
            for j=1:nVar
                pop(i).position(j) = Lb(j) + (Ub(j) - Lb(j))*rand;
                pop(i).velocity(j) = 0;
            end

            pop(i).cost = fx(pop(i).position);

            if pop(i).cost < pop(i).best_cost

```

```

        pop(i).best_cost = pop(i).cost;
        pop(i).best_position = pop(i).position;
    end

    if pop(i).best_cost < gBestCost
        gBestCost = pop(i).best_cost;
        gBestX = pop(i).best_position;
    end
end
end

for i = 1:MaxPop
    for j = 1:nVar
        pop(i).velocity(j) = w * pop(i).velocity(j) + ...
            c1 * rand * (pop(i).best_position(j) - pop(i).position(j)) + ...
            c2 * rand * (gBestX(j) - pop(i).position(j));
        pop(i).position(j) = pop(i).position(j) + pop(i).velocity(j);
        if pop(i).position(j) < Lb(j) || pop(i).position(j) > Ub(j)
            pop(i).position(j) = Lb(j) + (Ub(j) - Lb(j)) * rand;
        end
    end
end

pop(i).cost = fx(pop(i).position);

if pop(i).cost < pop(i).best_cost
    pop(i).best_cost = pop(i).cost;
    pop(i).best_position = pop(i).position;

    if pop(i).best_cost < gBestCost
        lastIter = iter;
        gBestCost = pop(i).best_cost;
        km = 25;
        Lb2 = Lb;
        Ub2 = Ub;
        for j = 1:nVar
            gBestX(j) = pop(i).best_position(j);
            if gBestX(j) > 0
                Lb2(j) = 0.85*gBestX(j);
                Ub2(j) = 1.15*gBestX(j);
            elseif gBestX(j) < 0
                Lb2(j) = 1.15*gBestX(j);
                Ub2(j) = 0.85*gBestX(j);
            else
                Lb2(j) = -0.05;
                Ub2(j) = 0.05;
            end
            if Lb2(j) < Lb(j), Lb2(j) = Lb(j); end
            if Ub2(j) > Ub(j), Ub2(j) = Ub(j); end
        end
        s = ' ';
        for i2=1:km
            for i1=1:nVar
                rpop = gBestX;
                rpop(i1) = Lb2(i1) + (Ub2(i1) - Lb2(i1)) * rand;
                zfx = fx(rpop);
                if zfx < gBestCost
                    gBestCost = zfx;

```

```

        gBestX = rpop;
        s = '*';
        randSearchSuccessCount = randSearchSuccessCount + 1;
    end
end
end
fprintf('Iter = %i, ', iter);
if ~strcmp(s, '*')
    fprintf('Best Fx = %e, ', gBestCost);
else
    fprintf('Best Fx = %e %s, ', gBestCost, s);
end
fprintf('Best X = [');
fprintf('%f, ', gBestX(1:nVar-1));
fprintf(' %f]\n', gBestX(nVar));

    end
end
end
w = w * wdamp;
end
fprintf("random Search Success Count = %d\n", randSearchSuccessCount)
end

```

The pso function has the following input parameters:

- The parameter fx is the handle of the function to optimize.
- The parameters Lb and Ub define the lower and upper limits of the power ranges, respectively.
- The parameter MaxPop specifies the maximum population size.
- The parameter MaxIters defines the maximum number of iterations.
- The parameter itersToReset specifies the number of consecutive iterations that fail to obtain a better guess for the optimum. This condition causes the function to create a new set of population members.

The function has the following output parameters:

- The parameter gBestX is the array of optimum powers for the terms in the rational heteronomial.
- The parameter gBestCost is the value of the best guess for the optimized function.

## The Cuckoo Search Optimization (CSO) Function

The function cuckoo\_searchEx4() implements the CSO method.

```

% -----
% Cuckoo Search (CS) algorithm by Xin-She Yang and Suash Deb      %
% Programmed by Xin-She Yang at Cambridge University              %
% Programming dates: Nov 2008 to June 2009                        %

```

```

% Last revised: Dec 2009 (simplified version for demo only) %
% -----
% Papers -- Citation Details:
% 1) X.-S. Yang, S. Deb, Cuckoo search via Levy flights,
% in: Proc. of World Congress on Nature & Biologically Inspired
% Computing (NaBIC 2009), December 2009, India,
% IEEE Publications, USA, pp. 210-214 (2009).
% http://arxiv.org/PS_cache/arxiv/pdf/1003/1003.1594v1.pdf
% 2) X.-S. Yang, S. Deb, Engineering optimization by cuckoo search,
% Int. J. Mathematical Modelling and Numerical Optimisation,
% Vol. 1, No. 4, 330-343 (2010).
% http://arxiv.org/PS_cache/arxiv/pdf/1005/1005.2908v2.pdf
% -----
% This demo program only implements a standard version of %
% Cuckoo Search (CS), as the Levy flights and generation of %
% new solutions may use slightly different methods. %
% The pseudo code was given sequentially (select a cuckoo etc), %
% but the implementation here uses Matlab's vector capability, %
% which results in neater/better codes and shorter running time. %
% This implementation is different and more efficient than the %
% the demo code provided in the book by %
% "Yang X. S., Nature-Inspired Metaheuristic Algorithms, %
% 2nd Edition, Luniver Press, (2010). " %
% -----
%
% ===== %
% Notes: %
% Different implementations may lead to slightly different %
% behavior and/or results, but there is nothing wrong with it, %
% as this is the nature of random walks and all metaheuristics. %
% -----
%
% -----
% Note: Code altered by Namir Shammas. This version uses an augmented
% version of the matrix nest, where the last column has the function value
% for the data in that row. Also, the code updates the second half of the
% population in matrix nest as long as parameter pa is less than a random
% number. The values of optimization parameters vary between 0.1 and 0.5,
% based on the main iterations loop counter.
%
% -----
%
% Note: Very good algorithm

function [bestX,bestFx]=cuckoo_searchEx4(fx,Lb,Ub,MaxPop,MaxIters,Toler)
    if nargin < 6, Toler = 1e-15; end
    % Number of nests (or different solutions)
    n=MaxPop;

    %% Simple bounds of the search domain
    nVars = length(Lb);
    m = nVars + 1;
    nest = zeros(n,m); % additional column for function values
    % Random initial solutions
    for i=1:n
        nest(i,1:nVars)=Lb+(Ub-Lb).*rand(1,nVars);
        nest(i,m) = fx(nest(i,1:nVars));
    
```

```

end

% sort nests, such that best nest is at index 1
nest = sortrows(nest,m);

%% Starting iterations
for iter=1:MaxIters
    pa = 0.5 - 0.4*(iter - 1)/(MaxIters - 1);

    % Generate new solutions (but keep the current best)
    new_nest=get_cuckoos(nest,Lb,Ub);
    [nest]=get_best_nest(fx,nest,new_nest,iter);

    % Discovery and randomization
    new_nest=empty_nests(fx,nest,Lb,Ub,pa) ;

    % Evaluate this set of solutions
    [nest]=get_best_nest(fx,nest,new_nest,iter);

    if abs(nest(1,m)) <= Toler
        break;
    end
end %% End of iterations
bestX = nest(1, 1:nVars);
bestFx = nest(1, m);
end

%% ----- All subfunctions are list below -----
%% Get cuckoos by random walk
function nest=get_cuckoos(nest,Lb,Ub)
    % Levy flights
    [n, m] = size(nest);
    nVars = m - 1;
    % Levy exponent and coefficient
    % For details, see equation (2.21), Page 16 (chapter 2) of the book
    % X. S. Yang, Nature-Inspired Metaheuristic Algorithms, 2nd Edition,
    Luniver Press, (2010).
    beta=3/2;
    sigma=(gamma(1+beta)*sin(pi*beta/2)/(gamma((1+beta)/2)*beta*2^((beta-
1)/2)))^(1/beta);

    for j=1:n
        s=nest(j,1:nVars);
        % This is a simple way of implementing Levy flights
        % For standard random walks, use step=1;
        %% Levy flights by Mantegna's algorithm
        u=randn(size(s))*sigma;
        v=randn(size(s));
        step=u./abs(v).^(1/beta);

        % In the next equation, the difference factor (s-best) means that
        % when the solution is the best solution, it remains unchanged.
        stepsize=0.01*step.*(s-nest(1,1:nVars));
        % Here the factor 0.01 comes from the fact that L/100 should be the
        typical
        % step size of walks/flights where L is the typical lengthscale;
        % otherwise, Levy flights may become too aggressive/efficient,

```

```

        % which makes new solutions (even) jump out side of the design domain
        % (and thus wasting evaluations).
        % Now the actual random walks or flights
        s=s+stepsize.*randn(size(s));
        % Apply simple bounds/limits
        nest(j,1:nVars) = simplebounds(s,Lb,Ub);
    end
end

%% Find the current best nest
function [nest]=get_best_nest(fx,nest,newnest,iter)
% Note: newnest is a matrix n by nVars, while nest is a matrix n by nVars+1.
% Evaluating all new solutions
[n, m] = size(nest);
nVars = m - 1;
bestFx = nest(1,m);
for j=2:n
    fnew=fx(newnest(j,1:nVars));
    if fnew<=nest(j,m)
        nest(j,m) = fnew;
        nest(j,1:nVars) = newnest(j,1:nVars);
    end
end
end

nest = sortrows(nest, m);
if bestFx > nest(1,m)
    fprintf("Iter: %d, Fx = %e, X=[" , iter, nest(j,m));
    fprintf("%f, ", nest(j,1:nVars-1));
    fprintf("%f]\n", nest(j,nVars));
end
end

%% Replace some nests by constructing new solutions/nests
function new_nest=empty_nests(fx,nest,Lb,Ub,pa)
% The lower population is replaced
[n, m] = size(nest);
nVars = m - 1;
nby2 = fix(n/2);
for i=1:nby2
    i1 = 1 + fix(nby2*rand);
    i2 = i1;
    while i1==i2
        i2 = 1 + fix(nby2*rand);
    end
    stepsize = rand*(nest(i1+nby2,1:nVars)-nest(i2+nby2,1:nVars));
    if rand>pa
        new_nest(i,1:nVars)=nest(i,1:nVars)+stepsize;
        new_nest(i,1:nVars)=simplebounds(new_nest(i,1:nVars),Lb,Ub);
        new_nest(i, m) = fx(new_nest(i,1:nVars));
    else
        new_nest(i,:) = nest(i+nby2,:);
    end
    new_nest(i,:) = simplebounds(new_nest(i,:),Lb,Ub);
end
% new_nest = [nest(1:nby2,:) new_nest(1:nby2,:)];
new_nest = [nest(1:nby2,:); new_nest(1:nby2,:)];

```

```

end

% Application of simple constraints
function s=simplebounds(s,Lb,Ub)
    nVars = length(s)-1;
    for i=1:nVars
        if s(i) < Lb(i) || s(i) > Ub(i)
            s(i) = Lb(i) + (Ub(i) - Lb(i))*rand;
        end
    end
end
end

```

The cuckoo\_searchEx4 function has the following input parameters:

- The parameter fx is the handle of the function to optimize.
- The parameters Lb and Ub define the lower and upper limits of the power ranges, respectively.
- The parameter MaxPop specifies the maximum population size.
- The parameter MaxIters defines the maximum number of iterations.
- The parameter Toler specifies the minimum function value that causes the optimization process to stop. The default value for this parameter is 1e-15.

The function has the following output parameters:

- The parameter bestX is the array of optimum powers for the terms in the rational heteronomial.
- The parameter bestFx is the value of the best guess for the optimized function.

## The Hunter Prey Optimization (HPO) Function

The function HPO() implements the hunter prey optimization.

```

% Developed in MATLAB R2017b
% Source codes demo version 1.0
%
% Author, inventor and programmer: Iraj Naruei and Farshid Keynia,
% e-Mail: irajnaruei@iauk.ac.ir , irajnaruei@yahoo.com
%
% Co-author and Advisor: Farshid Keynia
%
% e-Mail: fkeynia@gmail.com
%
% Co-authors: Amir Sabbagh Molahoseyni
%
% e-Mail: sabbagh@iauk.ac.ir
%
% You can find the HPO article at
% Hunter?rey optimization: algorithm and applications
% https://doi.org/10.1007/s00500-021-06401-0

```

```

%
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Very good function!!

function [bestX,bestFx] = HPO(fx,Lb,Ub,MaxPop,MaxIters,B)
% Constriction Coeficient
if nargin < 6, B = 0.1; end
nVars = length(Lb);

%% Initialization
HPpos=rand(MaxPop,nVars).*(Ub-Lb)+Lb;
HPposFitness = zeros(MaxPop,1);
% for i=1:MaxPop
%     HPposFitness(i)=inf;
% end
% Evaluate
for i=1:MaxPop
    HPposFitness(i)=fx(HPpos(i,:));
end
% NFE = MaxPop;
[~,indx] = min(HPposFitness);
%
bestX = HPpos(indx,:); % bestX HPO
bestFx =HPposFitness(indx);
lastIter = 0;
MaxStall = fix(MaxIters/20);
%% HPO Main Loop
for iter = 1:MaxIters
    if (iter - lastIter) >= MaxStall
        lastIter = iter;
        fprintf("----- New population at iter %d\n",
iter);
        HPpos=rand(MaxPop,nVars).*(Ub-Lb)+Lb;
        for i=1:MaxPop
            HPposFitness(i)=fx(HPpos(i,:));
        end
        [~,indx] = min(HPposFitness);
        %
        bestX2 = HPpos(indx,:); % bestX HPO
        bestFx2 =HPposFitness(indx);
        if bestFx2 < bestFx
            bestFx = bestFx2;
            bestX = bestX2;
        end
    end
    c = 1 - (iter-1)*((0.98)/MaxIters); % Update C Parameter
    kbest=round(MaxPop*c); % Update kbest
    for i = 1:MaxPop
        r1=rand(1,nVars) < c;
        r2=rand;
        r3=rand(1,nVars);
        idx=(r1==0);
        z=r2.*idx+r3.*~idx;
        if rand < B
            xi=mean(HPpos);
            dist = pdist2(xi,HPpos);

```



```

        [~,idxsortdist]=sort(dist);
        SI=HPpos(idxsortdist(kbest),:);
        HPpos(i,:)=HPpos(i,:)+0.5*((2*(c)*z.*SI-HPpos(i,:))+(2*(1-c)*z.*xi-
HPpos(i,:)));
    else
        for j=1:nVars
            rr=-1+2*z(j);
            HPpos(i,j)= 2*z(j)*cos(2*pi*rr)*(bestX(j)-HPpos(i,j))+bestX(j);
        end
    end
    % HPpos(i,:) = min(max(HPpos(i,:),Lb),Ub);
    for j=1:nVars
        if HPpos(i,j) < Lb(j) || HPpos(i,j) > Ub(j)
            HPpos(i,j) = Lb(j) + (Ub(j) - Lb(j))*rand;
        end
    end
    % Evaluation
    HPposFitness(i) = fx(HPpos(i,:));
    % Update bestX
    if HPposFitness(i) < bestFx
        lastIter = iter;
        bestX = HPpos(i,:);
        bestFx = HPposFitness(i);
        fprintf("Iter: %d, Fx = %e, X=[" , iter, bestFx);
        fprintf("%f, ", bestX(1:nVars-1));
        fprintf("%f]\n", bestX(nVars));
    end
end
end
end
end

```

The HPO function has the following input parameters:

- The parameter `fx` is the handle of the function to optimize.
- The parameters `Lb` and `Ub` define the lower and upper limits of the power ranges, respectively.
- The parameter `MaxPop` specifies the maximum population size.
- The parameter `MaxIters` defines the maximum number of iterations.
- The parameter `B` is an operational parameter with a default value of 0.1.

The function has the following output parameters:

- The parameter `bestX` is the array of optimum powers for the terms in the rational heteronomial.
- The parameter `bestFx` is the value of the best guess for the optimized function.

## The Differential Evolution (DE) Optimization Function

The function `de()` implements the differential evolution optimization.

```

%
% Copyright (c) 2015, Yarpiz (www.yarpiz.com)
% All rights reserved. Please read the "license.txt" for license terms.
%
% Project Code: YPEA107
% Project Title: Implementation of Differential Evolution (DE) in MATLAB
% Publisher: Yarpiz (www.yarpiz.com)
%
% Developer: S. Mostapha Kalami Heris (Member of Yarpiz Team)
%
% Contact Info: sm.kalami@gmail.com, info@yarpiz.com
%

% Note: Implementation and algorithm are excellent! Function fx9 requires
% more population and max iters

function [bestX,bestFx] = de(fx,XLow,XHi,MaxPop,MaxIters)

%% Problem Definition
nVar = length(XLow);
VarSize=[1 nVar]; % Decision Variables Matrix Size

%% DE Parameters

beta_min=0.2; % Lower Bound of Scaling Factor
beta_max=0.8; % Upper Bound of Scaling Factor
pCR=0.2; % Crossover Probability

%% Initialization

empty_individual.Position=[];
empty_individual.Cost=[];

BestSol.Cost=inf;
lastBesSol = BestSol.Cost;
pop=repmat(empty_individual,MaxPop,1);

for i=1:MaxPop

    pop(i).Position=unifrnd(XLow,XHi,VarSize);

    pop(i).Cost=fx(pop(i).Position);

    if pop(i).Cost<BestSol.Cost
        BestSol=pop(i);
    end

end

BestCost=zeros(MaxIters,1);

%% DE Main Loop
bestFx = 1e+99;
lastIter = 0;
MaxStall = fix(MaxIters/20);
for it=1:MaxIters

```

```

if (it - lastIter) >= MaxStall
    fprintf("----- New population at iter %d\n", it);
    lastIter = it;
    for i=1:MaxPop
        pop(i).Position=unifrnd(XLow,XHi,VarSize);
        pop(i).Cost=fx(pop(i).Position);
        if pop(i).Cost < BestSol.Cost
            BestSol=pop(i);
        end
    end
end

for i=1:MaxPop

    x=pop(i).Position;

    A=randperm(MaxPop);

    A(A==i)=[];

    a=A(1);
    b=A(2);
    c=A(3);

    % Mutation
    %beta=unifrnd(beta_min,beta_max);
    beta=unifrnd(beta_min,beta_max,VarSize);
    y=pop(a).Position+beta.*(pop(b).Position-pop(c).Position);
    y = max(y, XLow);
    y = min(y, XHi);

    % Crossover
    z=zeros(size(x));
    j0=randi([1 numel(x)]);
    for j=1:numel(x)
        if j==j0 || rand<=pCR
            z(j)=y(j);
        else
            z(j)=x(j);
        end
    end
end

NewSol.Position=z;
NewSol.Cost=fx(NewSol.Position);

if NewSol.Cost<pop(i).Cost
    pop(i)=NewSol;

    if pop(i).Cost<BestSol.Cost
        BestSol=pop(i);
    end
end

end

% Update Best Cost
BestCost(it)=BestSol.Cost;

```

```

% Show Iteration Information
if lastBesSol > BestSol.Cost
    lastIter = it;
    fprintf("%i: Best fx = %e, X=[", it, BestSol.Cost);
    fprintf("%f, ", BestSol.Position(1:nVar-1));
    fprintf("%f]\n", BestSol.Position(nVar));
    lastBesSol = BestSol.Cost;
%    disp(['Iteration ' num2str(it) ': Best Cost = '
num2str(BestCost(it))]);
end
end

bestX = BestSol.Position;
bestFx = BestSol.Cost;
end

```

The function `de()` has the following input parameters:

- The parameter `fx` is the handle of the function to optimize.
- The parameters `Lb` and `Ub` define the lower and upper limits of the power ranges, respectively.
- The parameter `MaxPop` specifies the maximum population size.
- The parameter `MaxIters` defines the maximum number of iterations.

The function has the following output parameters:

- The parameter `bestX` is the array of optimum powers for the terms in the rational heteronomial.
- The parameter `bestFx` is the value of the best guess for the optimized function.

## The Vibrating Particles System (EVPS) Optimization Function

The function `evps()` implements an enhanced version of the vibrating particles system optimization.

```

% The EVPS code in MATLAB
% Enhanced VIBRATING PARTICLES SYSTEM - EVPS
function [bestX, bestFx] = evpsFx(fx,Lb,Ub,popSize,maxIters)

% Example
% [bestX, bestFx] = evpsFx(@fx1,[0 0 0 0],5*[5 5 5 5],50,500)
% Initializing variables
MaxStall = fix(maxIters/20);
nVar = length(Lb); % Number of optimization variables
alpha=0.05;
w1=0.3;
w2=0.3;
w3=1-w1-w2;
p=0.2; % With the probability of (1-p) the effect of BP
% is ignored in updating

```

```

PAR=0.1;
HMCR=0.95;
Memorysize=4;
result=zeros(popSize,nVar+1);
Memory=zeros(Memorysize,nVar+1);
neighbor=0.1; % Parameters for handling the side constraints
% Initializing particles
position = zeros(popSize,nVar);
for i=1:popSize
    position(i,:) = Lb +rand(1,nVar).*(Ub-Lb);
end
% Search
agentCost=zeros(popSize,3); % Array of agent costs
HBV=zeros(popSize,nVar+2); % Historically best matrix
fxArr = zeros(popSize,1);
bestFx = 1e99;
lastIter = 0;
for iter=1:maxIters
    if (iter-lastIter) >= MaxStall
        fprintf("----- New population at iter %d\n", iter);
        lastIter = iter;
        for i=1:popSize
            position(i,:) = Lb +rand(1,nVar).*(Ub-Lb);
            fxArr(i) = fx(position(i,:));
        end
        [~,indx] = min(fxArr);
        bestFx2 = fxArr(indx);
        if bestFx2 < bestFx
            bestFx = bestFx2;
            bestX = position(indx,:);
        end
    end

    % Evaluating and storing
    for m=1:popSize
        % Evaluating the objective function for each particle
        penalizedWeight=fx(position(m,:));
        agentCost(m,1)=penalizedWeight;
        agentCost(m,2)=m;
        agentCost(m,3)=penalizedWeight; % was weight;
    end
    sortedAgentCost=sortrows(agentCost);
    for m=1:popSize
        if iter==1 || agentCost(m,1)<HBV(m,1)
            HBV(m,1)=agentCost(m,1);
            HBV(m,2)=agentCost(m,3);
            for n=1:nVar
                HBV(m,n+2)=position(m,n);
            end
        end
    end
    sortedHBV=sortrows(HBV);
    result(iter,1)=sortedHBV(1,1);
    result(iter,2:nVar+1)=sortedHBV(1,3:nVar+2);
    if iter==1
        for i=1:Memorysize
            Memory(i,1)=sortedHBV(i,1);

```

```

        Memory(i,2:nVar+1)=sortedHBV(i,3:nVar+2);
    end
end
if Memory(Memorysize,1)>sortedHBV(1,1)
    Memory(Memorysize,1)=sortedHBV(1,1);
    Memory(Memorysize,2:nVar+1)=sortedHBV(1,3:nVar+2);
end
Memory=sortrows(Memory);
% Updating particle positions
D=(iter/maxIters)^(-alpha);
for m=1:popSize
    temp1=ceil(unifrnd(1,popSize/2,1));
    temp2=ceil(unifrnd(popSize/2,popSize,1));
    temp11=round(unifrnd(1,Memorysize,1));
    if p<rand
        w3=0;
        w2=1-w1;
    end

    for n=1:nVar
        if rand<w3
            A(m,n)=(-1)^(ceil(randn()))*(position(sortedAgentCost(temp2,2),n)-
position(m,n));
            position(m,n)=D*A(m,n)*rand+position(sortedAgentCost(temp2,2),n);
        elseif rand<w2
            A(m,n)=(-1)^(ceil(randn()))*(position(sortedAgentCost(temp1,2),n)-
position(m,n));
            position(m,n)=D*A(m,n)*rand+position(sortedAgentCost(temp1,2),n);
        else
            A(m,n)=(-1)^(ceil(randn()))*(Memory(temp11,1+n)-position(m,n));
            position(m,n)=D*A(m,n)*rand+Memory(temp11,1+n);
        end
    end
    end
    w2=0.3;w3=1-w1-w2;
end
% Handling the side constraints
for m=1:popSize
    for n=1:nVar
        if position(m,n)<Lb(n) || position(m,n)>Ub(n)
            temp1=rand;temp2=rand;temp3=ceil(rand*popSize);
            if temp1<=HMCR && temp2<=(1-PAR)
                position(m,n)=sortedHBV(temp3,2+n);
            elseif temp1<=HMCR && temp2>(1-PAR)
                position(m,n)=sortedHBV(temp3,2+n)+neighbor;
                if position(m,n)>Ub(n)
                    position(m,n)=sortedHBV(temp3,2+n)-2*neighbor;
                end
            else
                position(m,n)=Lb(n)+(rand*(Ub(n)-Lb(n)));
            end
        end
    end
end
end
if bestFx > sortedHBV(1,1)
    bestFx = sortedHBV(1,1);
    bestX = sortedHBV(1,3:end);
    lastIter = iter;

```

```

        fprintf("Iter: %d, Fx = %e, X=", iter, bestFx)
        fprintf("%f, ", bestX(1:nVar-1));
        fprintf("%f]\n", bestX(nVar));
    end
end
end

```

The function `evps()` has the following input parameters:

- The parameter `fx` is the handle of the function to optimize.
- The parameters `Lb` and `Ub` define the lower and upper limits of the power ranges, respectively.
- The parameter `MaxPop` specifies the maximum population size.
- The parameter `MaxIters` defines the maximum number of iterations.

The function has the following output parameters:

- The parameter `bestX` is the array of optimum powers for the terms in the rational heteronomial.
- The parameter `bestFx` is the value of the best guess for the optimized function.

## The Ant Colony Optimization (ACO) Algorithm Function

The function `aco()` implements the ant colony optimization.

```

%
% Copyright (c) 2015, Yarpiz (www.yarpiz.com)
% All rights reserved. Please read the "license.txt" for license terms.
%
% Project Code: YPEA104
% Project Title: Ant Colony Optimization for Continuous Domains (ACOR)
% Publisher: Yarpiz (www.yarpiz.com)
%
% Developer: S. Mostapha Kalami Heris (Member of Yarpiz Team)
%
% Contact Info: sm.kalami@gmail.com, info@yarpiz.com
%
function [bestX, bestFx]=aco(fx,Lb,Ub,MaxPop,MaxIters)
%% Problem Definition

nVars=length(Lb); % Number of Decision Variables
%% ACOR Parameters
MaxStall = fix(MaxIters/20);
lastIter = 0;
nSample=4*MaxPop; % Sample Size
q=0.5; % Intensification Factor (Selection Pressure)
zeta=1; % Deviation-Distance Ratio
%% Initialization
% Create Empty Individual Structure
empty_individual.Position=[];

```

```

empty_individual.Cost=[];
% Create Population Matrix
pop= repmat(empty_individual,MaxPop,1);
bestFx = 1e99;
% Initialize Population Members
for i=1:MaxPop

    % Create Random Solution
    for j=1:nVars
        pop(i).Position(j) = Lb(j) + (Ub(j)-Lb(j))*rand;
    end

    % Evaluation
    pop(i).Cost=fx(pop(i).Position);
end
% Sort Population
 [~, SortOrder]=sort([pop.Cost]);
pop=pop(SortOrder);
% Update Best Solution Ever Found
bestFx = pop(1).Cost;
bestX = pop(1).Position;
fprintf("Iter 0, Fx = %e, X=[" , bestFx);
fprintf("%f, ", bestX(1:nVars-1));
fprintf("%f]\n", bestX(nVars));

% Solution Weights
w=1/(sqrt(2*pi)*q*MaxPop)*exp(-0.5*((1:MaxPop)-1)/(q*MaxPop)).^2);
% Selection Probabilities
p=w/sum(w);

%% ACOR Main Loop
for iter=1:MaxIters
    if (iter - lastIter) > MaxStall
        lastIter = iter;
        for i=1:MaxPop
            % Create Random Solution
            for j=1:nVars
                pop(i).Position(j) = Lb(j) + (Ub(j)-Lb(j))*rand;
            end

            % Evaluation
            pop(i).Cost=fx(pop(i).Position);
        end
        % Sort Population
        [~, SortOrder]=sort([pop.Cost]);
        pop=pop(SortOrder);
        % Update Best Solution Ever Found
        if bestFx > pop(1).Cost
            bestFx = pop(1).Cost;
            bestX = pop(1).Position;
        end
    end

    % Means
    s=zeros(MaxPop,nVars);
    for l=1:MaxPop
        s(l,:)=pop(l).Position;
    end
end

```



```

end

% Standard Deviations
sigma=zeros(MaxPop,nVars);
for l=1:MaxPop
    D=0;
    for r=1:MaxPop
        D=D+abs(s(l,:)-s(r,:));
    end
    sigma(l,:)=zeta*D/(MaxPop-1);
end

% Create New Population Array
newpop= repmat(empty_individual,nSample,1);
for t=1:nSample

    % Initialize Position Matrix
    for j=1:nVars
        newpop(t).Position(j) = Lb(j) + (Ub(j)-Lb(j))*rand;
    end

    % Solution Construction
    for i=1:nVars

        % Select Gaussian Kernel
        l=RouletteWheelSelection(p);

        % Generate Gaussian Random Variable
        newpop(t).Position(i)=s(l,i)+sigma(l,i)*randn;
        if newpop(t).Position(i) < Lb(i) || ...
            newpop(t).Position(i) > Ub(i)
            newpop(t).Position(i) = Lb(i) + (Ub(i)-Lb(i))*rand;
        end
    end

    % Evaluation
    newpop(t).Cost=fx(newpop(t).Position);

end

% Merge Main Population (Archive) and New Population (Samples)
pop=[pop
    newpop]; %#ok

% Sort Population
[~, SortOrder]=sort([pop.Cost]);
pop=pop(SortOrder);

% Delete Extra Members
pop=pop(1:MaxPop);
if bestFx > pop(1).Cost
    lastIter = iter;
    bestFx = pop(1).Cost;
    bestX = pop(1).Position;
    % Show Iteration Information
    fprintf("Iter %d, Fx = %e, X=[" , iter, bestFx);
    fprintf("%f, ", bestX(1:nVars-1));

```

```

        fprintf("%f]\n", bestX(nVars));
        % disp(['Iteration ' num2str(iter) ': Best Cost = '
num2str(BestCost(iter))]);
    end
end

```

The function `aco()` has the following input parameters:

- The parameter `fx` is the handle of the function to optimize.
- The parameters `Lb` and `Ub` define the lower and upper limits of the power ranges, respectively.
- The parameter `MaxPop` specifies the maximum population size.
- The parameter `MaxIters` defines the maximum number of iterations.

The function has the following output parameters:

- The parameter `bestX` is the array of optimum powers for the terms in the rational heteronomial.
- The parameter `bestFx` is the value of the best guess for the optimized function.

## The Arithmetic Optimization Algorithm (AOA) Function

The function `AOAfx()` implements the arithmetic optimization algorithm.

```

function [bestX,bestFx]=AOAfx(fx,Lb,Ub,MaxPop,MaxIters)

    nVars = length(Lb);
    bestX = Lb + (Ub-Lb)/2;
    bestFx = fx(bestX);
    lastIter = 0;
    MaxStall = fix(MaxIters/20);

    %Initialize the positions of solution
    X = zeros(MaxPop, nVars);
    for i=1:MaxPop
        X(i,:) = Lb + (Ub - Lb).*rand(1,nVars);
    end
    Xnew=X;
    Ffun=zeros(MaxPop);% (fitness values)
    Ffun_new=zeros(MaxPop);% (fitness values)

    MOP_Max=1;
    MOP_Min=0.2;
    Alpha=5;
    Mu=0.499;

    for i=1:MaxPop
        Ffun(i)=fx(X(i,:)); %Calculate the fitness values of solutions
        if Ffun(i) < bestFx
            bestFx=Ffun(i);

```

```

    bestX=X(i,:);
end
end

for iter=1:MaxIters %Main loop

    if (iter - lastIter) > MaxStall
        lastIter = iter;
        for i=1:MaxPop
            X(i,:) = Lb + (Ub - Lb).*rand(1,nVars);
            Ffun(i)=fx(X(i,:));
            if Ffun(i) < bestFx
                bestFx=Ffun(i);
                bestX=X(i,:);
            end
        end
    end
end

MOP=1-((iter)^(1/Alpha)/(MaxIters)^(1/Alpha)); % Probability Ratio
MOA=MOP_Min+iter*((MOP_Max-MOP_Min)/MaxIters); %Accelerated function

%Update the Position of solutions
for i=1:MaxPop % if each of the Ub and Lb has a just value
    for j=1:nVars
        r1=rand();
        if r1 < MOA
            r2=rand();
            if r2 > 0.5
                Xnew(i,j)=bestX(1,j)/(MOP+eps)*((Ub(j)-Lb(j))*Mu+Lb(j));
            else
                Xnew(i,j)=bestX(1,j)*MOP*((Ub(j)-Lb(j))*Mu+Lb(j));
            end
        else
            r3=rand();
            if r3 > 0.5
                Xnew(i,j)=bestX(1,j)-MOP*((Ub(j)-Lb(j))*Mu+Lb(j));
            else
                Xnew(i,j)=bestX(1,j)+MOP*((Ub(j)-Lb(j))*Mu+Lb(j));
            end
        end
    end
end

for j=1:nVars
    if Xnew(i,j) < Lb(j) || Xnew(i,j) > Ub(j)
        Xnew(i,j) = Lb(j) + (Ub(j) - Lb(j))*rand;
    end
end

Ffun_new(i)=fx(Xnew(i,:)); % calculate Fitness function
if Ffun_new(i) < Ffun(i)
    X(i,:)=Xnew(i,:);
    Ffun(i)=Ffun_new(i);
end

if Ffun(i) < bestFx
    lastIter = iter;
    bestFx=Ffun(i);
end

```

```
        bestX=X(i,:);  
        fprintf("Iters: %d, Fx = %e, X=[", iter, bestFx);  
        fprintf("%f, ", bestX(1:nVars-1));  
        fprintf("%f]\n", bestX(nVars));  
    end  
end  
end  
end
```

The function AOAfx() has the following input parameters:

- The parameter fx is the handle of the function to optimize.
- The parameters Lb and Ub define the lower and upper limits of the power ranges, respectively.
- The parameter MaxPop specifies the maximum population size.
- The parameter MaxIters defines the maximum number of iterations.

The function has the following output parameters:

- The parameter bestX is the array of optimum powers for the terms in the rational heteronomial.
- The parameter bestFx is the value of the best guess for the optimized function.