Modified Artificial Cooperative Search Algorithms

By

Namir Clement Shammas

Contents

1-Introduction

The Artificial Cooperative Search (ACS) algorithm was designed by Pinar Civicioglu. His 2013 article "*Artificial cooperative search algorithm for numerical optimization problems*[1]" introduces the algorithm and discusses how it works. The ACS algorithm belongs to the class of evolutionary algorithms for numerical optimization. The algorithm yields fast convergence and highly accurate results. The basic concept of ACS is to work with two baseline populations, (call them A and B, or alpha and beta). These populations contain candidate solutions drawn from a trust region—each dimension in this region is defined by an upper and lower limit. The algorithm uses these populations to define two new ones-the predator and the prey. The mapping between the first two populations and the predator and prey populations uses random steps and a binary matrix. Moreover, the algorithm calculates a fifth population from the values in the predator and prey populations. These calculations are followed by updating the initial two populations A and B. The entire process is repeated for a prefix number of times. The sought results are taken from the best solution in the two populations A and B. To learn more details about the rationale and design of the algorithm, I recommend that you consult the article mentioned above.

The next section presents the detailed original pseudo-code for ACS. The three sections that follow present three variants of ACS that I developed. I will also present MATLAB implementations for all versions of the ACS algorithm. To the best of my knowledge, I have not located any programming language listing for the ACS algorithm. Thankfully, Civicioglu's detailed pseudo-code is very helpful in coding it into MATLAB, Excel VBA, and any other capable programming language. I chose to present the MATLAB code in this article because it is simpler to present and use than the Excel VBA code.

2-Original Pseudo-Code for ACS Algorithm

```
1. Given MaxPop population size, N variables, MaxIter maximum iterations, XLow
   low limits for the variables, XHi upper limit for the variables, and Probab
   (probability of biological interaction in ACS). Rand is the function that returns
   uniformly distributed numbers in the open range (0,1].
2. To minimize Fx(X)
•  GlobMin = infinity
•  For I = 1 to MaxPop
       o  For J = 1 to N
              ▪  A(I,J) = XLow(J) + (XHi(J) – XLow(J)) * Rand
              ▪  B(I,J) = XLow(J) + (XHi(J) – XLow(J)) * Rand
       o  YA(I) = Fx(A(I,:)
       o  YB(I)= Fx(B(I,:)
•  For Iter =1 to MaxIter
       o  REM Selection
```

```
    o If Rand < Rand Then
        ▪ Predictor = A
        ▪ Ypred = YA
        ▪ Key = 1
    o Else
        ▪ Predictor = B
        ▪ Ypred = YB
        ▪ Key = 2
    o If Rand < Rand Then
        ▪ Prey = A
    o Else
        ▪ Prey = B
    o Prey = Permutation of Prey
    o If Rand < Rand Then
        ▪ R = 4 * Rand * (Rand – Rand)
    o Else
        ▪ R = 1/exp(4 * Rand)
    o Fill binary matrix M(MaxPop, N) with 1s.
    o For q = 1 to MaxPop * N
        ▪ If Rand < Prob * Rand Then
            • I = 1 + fix(Rand * MaxPop)
            • J = 1 + fix(Rand * N)
            • M(I, J) = 0
    o If Rand < Prob * Rand Then
        ▪ For I = 1 to MaxPop
            • For J = 1 to N
                o If Rand < Prob * Rand Then
                    ▪ M(I, J) = 1
                o Else
                    ▪ M(I, J) = 0
    o For I = 1 to MaxPop
        ▪ If Sum(M(I,:)) = N Then
            • J = 1 + fix(Rand * N)
            • M(I, J) = 0
    o REM Mutation
    o X = Predator + R * (Prey – Predator)
    o For I = 1 to MaxPop
        ▪ For J = 1 to N
            • REM Crossover
            • If M(I,J) > 0 Then X(I,J) = Predator(I,J)
            • REM Boundary control
            • IF X(I,J) < XLow(J) OR X(I,J) > XHi(J) Then
                o X(I,J) = XLow(J) + (XHi(J) – XLow(J)) * Rand
    o Rem Selection update
    o For I = 1 to MaxPop
        ▪ If Fx(X(I,:)) < Ypred (i) Then
            • Predator(I,:) = X(I,:)
            • Ypred (I) = Fx(X(I,:))
    o If Key = 1 Then
        ▪ A = Predator
        ▪ YA = Ypred
    o Else
        ▪ B  = Predator
        ▪ YB= Ypred
    o Ybest = min(Ypred)
    o Ibest = row of Predator corresponding to Ybest
    o If Ybest < GlobMin Then
        ▪ GlobMin = Ybest
        ▪ GlobMinX = Peditor(Ibest,:)
• Return GlobMin and vector GlobMinX
```

You may have noticed that the above pseudo-code tests the expression Rand < Rand in several places. This expression may seem strange or new to you, since we are used to expressions comparing Rand with a fraction between 0 and 1, usually 0.5. The expression Rand < Rand is normally distributed and has a similar distribution to Rand > 0.5.

Another point, regarding the use of the statement R = 1/exp(4 * Rand). The original ACS pseudo-code uses the statement R = Gamma_pdf(4 * Rand, 1) which seems to suggest that the it was a short form for Gamma_pdf(4 * Rand, 1, 1). This is the case where the distribution's parameters alpha and beta are equal to 1, making the Gamma_pdf(x,1,1,) distribution the same as exp(−x).

Finally, the binary matrix M can be replaced with a Boolean matrix, changing 0 with false and 1 with true.

3-First and Minor Modification

This version uses augmented forms of matrices A and B. These matrices have MaxPop rows and N+1 columns. The last column stores the function values given by the first N columns of that row.

The second change is reversing how binary matrix M is created. This pseudo-code shows that initially, the matrix M is filled with zero. The For q loop assigns 1 to selected members of matrix M. With this change, I noticed a slight improvement in the accuracy of the solutions—at least for the functions I tested. The text in red highlights the changes I made to the original pseudo-code.

1. Given MaxPop population size, N variables, MaxIter maximum iterations, XLow low limits for the variables, XHi upper limit for the variables, and Probab (probability of biological interaction in ACS). Rand is the function that returns uniformly distributed numbers in the range (0,1].
2. To minimize Fx(X)

```
• For I = 1 to MaxPop
    o For J = 1 to N
        ▪ A(I,J) = XLow(J) + (XHi(J) – XLow(J)) * Rand
        ▪ B(I,J) = XLow(J) + (XHi(J) – XLow(J)) * Rand
    o A(I,N+1) = Fx(A(I,:)
    o B(I,N+1) = Fx(B(I,:)
• Find minimum value Amin and Index IdxA for smallest value in column N+1 of A.
• Find minimum value Bmin and Index IdxB for smallest value in column N+1 of B.
• If Amin < BminThen
    o GlobMin = Amin
    o GlobMinX = A(IdxA,1:N)
• Else
    o GlobMin = Bmin
    o GlobMinX = B(IdxB,1:N)
```

```
• For Iter =1 to MaxIter
    o REM Selection
    o If Rand < Rand Then
        ▪ Predictor = A
        ▪ Ypred = A(:,N+1)
        ▪ Key = 1
    o Else
        ▪ Predictor = B
        ▪ Ypred = B(:,N+1)
        ▪ Key = 2
    o If Rand < Rand Then
        ▪ Prey = A
    o Else
        ▪ Prey = B
    o Prey = Permutation of Prey
    o If Rand < Rand Then
        ▪ R = 4 * Rand * (Rand – Rand)
    o Else
        ▪ R = 1/exp(4 * Rand)
    o (Re)Initialize matrix M with MaxPop rows and N columns, storing 0 in each
      element.
    o For q = 1 to MaxPop * N
        ▪ If Rand < Prob * Rand Then
            • I = 1 + fix(Rand * MaxPop)
            • J = 1 + fix(Rand * N)
            • M(I, J) = 1
    o If Rand < Prob * Rand Then
        ▪ For I = 1 to MaxPop
            • For J = 1 to N
                o If Rand < Prob * Rand Then
                    ▪ M(I, J) = 1
                o Else
                    ▪ M(I, J) = 0
    o For I = 1 to MaxPop
        ▪ If Sum(M(I,:)) = N Then
```

```
                • J = 1 + fix(Rand * N)
                • M(I, J) = 0
    o REM Mutation
    o X = Predator + R * (Prey – Predator)
    o For I = 1 to MaxPop
        ▪ For J = 1 to N
                • REM Crossover
                • If M(I,J) > 0 Then X(I,J) = Predator(I,J)
                • REM Boundary control
                • IF X(I,J) < XLow(J) OR X(I,J) > XHi(J) Then
                    o X(I,J) = XLow(J) + (XHi(J) – XLow(J)) * Rand
    o Rem Selection update
    o For I = 1 to MaxPop
        ▪ zFx = Fx(X(I,:))
        ▪ If zFx < Ypred (i) Then
                • Predator(I,:) = X(I,:)
                • Ypred (I) = zFx
    o If Key = 1 Then
        ▪ A(:,1:N) = Predator
        ▪ A(:,N+1) = Ypred
        ▪ Find minimum value Amin and Index IdxA for smallest value in
          column N+1 of A.
        ▪ Ybest = Amin
        ▪ bestX = A(IdxA,1:N)
    o Else
        ▪ B(:,1:N) = Predator
        ▪ B(:,N+1) = Ypred
        ▪ Find minimum value Bmin and Index IdxB for smallest value in
          column N+1 of B.
        ▪ Ybest = Bmin
        ▪ bestX = A(IdxB,1:N)
    o If Ybest < GlobMin Then
        ▪ GlobMin = Ybest
        ▪ GlobMinX = bestX
```

```
• Return GlobMin and vector GlobMinX
```

4-Second and Major Modification

This second version uses the first modifications and employs a new approach for populating the Predator and Prey matrices with values. The new approach compares each row in the sorted matrices A and B to decide which of the row values go to the Predator or Prey matrix. The augmented matrices A and B are first sorted using the values in column N+1 as sorting key values. This new approach cross-distributes the rows of matrices A and B, based on merit (i.e. the function value calculated for each row in both matrices), rather than random chance. Also, this variant does not use the Key variable. The updates for matrices A and B at the end of the main loop is done (and you may be a bit surprised here) by random comparisons. This scheme allows both matrices A and B to have the same probability of being updated. The text in red highlights the new changes I made to the pseudo-code in section 3.

1. Given MaxPop population size, N variables, MaxIter maximum iterations, XLow low limits for the variables, XHi upper limit for the variables, and Probab (probability of biological interaction in ACS). Rand is the function that returns uniformly distributed numbers in the range (0,1].
2. To minimize Fx(X)

```
• For I = 1 to MaxPop
    o For J = 1 to N
        ▪ A(I,J) = XLow(J) + (XHi(J) – XLow(J)) * Rand
        ▪ B(I,J) = XLow(J) + (XHi(J) – XLow(J)) * Rand
    o A(I,N+1) = Fx(A(I,:)
    o B(I,N+1) = Fx(B(I,:)
• Find value Amin and Index IdxA for smallest value in column N+1 of A.
• Find value Bmin and Index IdxB for smallest value in column N+1 of B.
• If Amin < BminThen
    o GlobMin = Amin
    o GlobMinX = A(IdxA,1:N)
• Else
    o GlobMin = Bmin
    o GlobMinX = B(IdxB,1:N)
```

```
• For Iter =1 to MaxIter
    o REM Selection
    o Sort matrix A using column N+1 as sort key values.
    o Sort matrix B using column N+1 as sort key values.
    o For I = 1 to MaxPop
        ▪ If A(I,N+1) < B(I,N+1) Then
            • Predator(I,:) = A(I,:)
            • Ypred(i) = A(I,N+1)
            • Prey(I,:) = B(I, :)
        ▪ Else
            • Predator(I,:) = B(I,:)
            • Ypred(i) = B(I,N+1)
            • Prey(I,:) = A(I, :)
    o Prey = Permutation of Prey
    o If Rand < Rand Then
        ▪ R = 4 * Rand * (Rand – Rand)
    o Else
        ▪ R = 1/exp(4 * Rand)
    o (Re)Initialize matrix M with MaxPop rows and N columns, storing 0 in each
      element.
    o For q = 1 to MaxPop * N
        ▪ If Rand < Prob * Rand Then
            • I = 1 + fix(Rand * MaxPop)
            • J = 1 + fix(Rand * N)
            • M(I, J) = 1
    o If Rand < Prob * Rand Then
        ▪ For I = 1 to MaxPop
            • For J = 1 to N
                o If Rand < Prob * Rand Then
                    ▪ M(I, J) = 1
                o Else
                    ▪ M(I, J) = 0
    o For I = 1 to MaxPop
        ▪ If Sum(M(I,:)) = N Then
```

```
            • J = 1 + fix(Rand * N)
            • M(I, J) = 0
    o REM Mutation
    o X = Predator + R * (Prey – Predator)
    o For I = 1 to MaxPop
        ▪ For J = 1 to N
            • REM Crossover
            • If M(I,J) > 0 Then X(I,J) = Predator(I,J)
            • REM Boundary control
            • IF X(I,J) < XLow(J) OR X(I,J) > XHi(J) Then
                o X(I,J) = XLow(J) + (XHi(J) – XLow(J)) * Rand
    o Rem Selection update
    o For I = 1 to MaxPop
        ▪ zFx = Fx(X(I,:))
        ▪ If zFx < Ypred (i) Then
            • Predator(I,:) = X(I,:)
            • Ypred (I) = zFx
    o If Rand < Rand Then
        ▪ A(:,N+1) = Ypred
        ▪ Find minimum value Amin and Index IdxA for smallest value in
          column N+1 of A.
        ▪ Ybest = Amin
        ▪ bestX = A(IdxA,1:N)
    o Else
        ▪ B(:,N+1) = Ypred
        ▪ Find minimum value Bmin and Index IdxB for smallest value in
          column N+1 of B.
        ▪ Ybest = Bmin
        ▪ bestX = A(IdxB,1:N)
    o If Ybest < GlobMin Then
        ▪ GlobMin = Ybest
        ▪ GlobMinX = bestX
• Return GlobMin and vector GlobMinX
```

The original pseudo-code and the two modifications presented here yield program listings, all of which, maintain superb accuracy and a very good rate of convergence.

5-Third and Major Modification

This third version uses a very different approach in populating the predator and prey matrix. The approach here accentuates the merit basis of the function values in columns N+1 of matrices A and B. This variant merges the augment matrices A and B, creating a bigger matrix, call it Pop. The algorithm then sorts the matrix Pop using the values in column N+1 as the key sort values. Next, the algorithms populates the predator matrix with the leading MaxPop rows in the matrix Pop. Finally, the algorithm populates the prey matrix with the trailing MaxPop rows of matrix Pop.

```
1. Given MaxPop population size, N variables, MaxIter maximum iterations, XLow
   low limits for the variables, XHi upper limit for the variables, and Probab
   (probability of biological interaction in ACS). Rand is the function that returns
   uniformly distributed numbers in the range (0,1].
2. To minimize Fx(X)
• For I = 1 to MaxPop
    o For J = 1 to N
        ▪ A((I,J) = XLow(J) + (XHi(J) – XLow(J)) * Rand
        ▪ B((I,J) = XLow(J) + (XHi(J) – XLow(J)) * Rand
    o A(I,N+1) = Fx(A(I,:)
    o B(I,N+1) = Fx(B(I,:)
• Find value Amin and Index IdxA for smallest value in column N+1 of A.
• Find value Bmin and Index IdxB for smallest value in column N+1 of B.
• If Amin < BminThen
    o GlobMin = Amin
    o GlobMinX = A(IdxA,1:N)
• Else
    o GlobMin = Bmin
    o GlobMinX = B(IdxB,1:N)
• For Iter =1 to MaxIter
    o REM Selection
    o Merge matrices A and B to create matrix Pop
```

```
o Sort matrix Pop using column N+1 as sort key values.
o Predator(1:MaxPop,1:N) = Pop(1:MaxPop,1:N)
o ypred(1:Maxpop) = Pop(1:MaxPop,N+1)
o Prey(1:MaxPop) = Pop(MaxPop+1:2MaxPop,1:N)
o Prey = Permutation of Prey
o If Rand < Rand Then
    ▪ R = 4 * Rand * (Rand – Rand)
o Else
    ▪ R = 1/exp(4 * Rand)
o (Re)Initialize matrix M with MaxPop rows and N columns, storing 0 in each
  element.
o For q = 1 to MaxPop * N
    ▪ If Rand < Prob * Rand Then
        • I = 1 + fix(Rand * MaxPop)
        • J = 1 + fix(Rand * N)
        • M(I, J) = 1
o If Rand < Prob * Rand Then
    ▪ For I = 1 to MaxPop
        • For J = 1 to N
            o If Rand < Prob * Rand Then
                ▪ M(I, J) = 1
            o Else
                ▪ M(I, J) = 0
o For I = 1 to MaxPop
    ▪ If Sum(M(I,:)) = N Then
        • J = 1 + fix(Rand * N)
        • M(I, J) = 0
o REM Mutation
o X = Predator + R * (Prey – Predator)
o For I = 1 to MaxPop
    ▪ For J = 1 to N
        • REM Crossover
        • If M(I,J) > 0 Then X(I,J) = Predator(I,J)
        • REM Boundary control
```

```
                • IF X(I,J) < XLow(J) OR X(I,J) > XHi(J) Then
                    o X(I,J) = XLow(J) + (XHi(J) – XLow(J)) * Rand
    o Rem Selection update
    o For I = 1 to MaxPop
        ▪ zFx = Fx(X(I,:))
        ▪ If zFx < Ypred (i) Then
            • Predator(I,:) = X(I,:)
            • Ypred (I) = zFx
    o If Rand < Rand Then
        ▪ A(:,N+1) = Ypred
        ▪ Find minimum value Amin and Index IdxA for smallest value in
          column N+1 of A.
        ▪ Ybest = Amin
        ▪ bestX = A(IdxA,1:N)
    o Else
        ▪ B(:,N+1) = Ypred
        ▪ Find minimum value Bmin and Index IdxB for smallest value in
          column N+1 of B.
        ▪ Ybest = Bmin
        ▪ bestX = A(IdxB,1:N)
    o If Ybest < GlobMin Then
        ▪ GlobMin = Ybest
        ▪ GlobMinX = bestX
• Return GlobMin and vector GlobMinX
```

The original pseudo-code and the two modifications presented here yield program listings, all of which, maintain superb accuracy and a very good rate of convergence.

6-The MATLAB Test Functions

Before I present the MATLAB code for the pseudo-code in sections 3 and 4, let me present a few test functions. The first test function is stored in file fx1.m:

```
function s = fx1(x)
   s=0;
   n=length(x);
   for i=1:n
       s=s+(x(i)-i)^2;
   end
end
```

The above function has solutions of 1, 2, 3 …, n. The value of the function at the optimum point is zero. The second test function is stored in file fx2.m:

```
function s = fx2(x)
 n = length(x);
 s = 0;
 for i=1:n
   z = i;
   for j=1:3
     z = z + (i+j)/10^j;
   end
   s = s + (x(i) - z)^2;
 end
end
```

The above function has solutions of 1.234, 2.345, 3.456, and so on. The value of the function at the optimum point is zero. The third test function is stored in file fx3.m:

```
function s = fx3(x)
   s=0;
   n=length(x);
   for i=1:n
       s=s+(x(i)-i^2)^2;
   end
end
```

The above function has solutions of 1, 4, 9, 16, …, n^2. The value of the function at the optimum point is zero. The fourth test function is stored in file rosenbrock.m:

```
function s = rosenbrock(x)
  n=length(x);
  s = 0;
  for i=1:n-1
    s = s + (x(i)-1)^2 + 100*(x(i+1) - x(i)^2)^2;
  end
end
```

The above function has solutions of 1 for each variable. The value of the function at the optimum point is zero. The fifth test function is stored in file nle1_4.m:

```
function s = nle1_4(X)
  F = zeros(4,1);
  F(1) = X(2) * X(4) ^ 2 - 32;
  F(2) = 2 * X(2) ^ 2 * X(3) - 24;
  F(3) = X(3) ^ 2 * X(1) - 9;
  F(4) = X(4) * X(1) ^ 2 - 4;
  s = sum(F.^2);
end
```

The above function has solutions of 1, 2, 3, and 4. The value of the function at the optimum point is zero.

When using test functions, whose solutions are known, reporting the error for intermediate and final results is a good thing. One statistics is the square root of the mean sum of error squared:

RMS = $\sum\ (s_i - x_i\ )^2$ (6-1)

Where s_i is the actual solution and x_i is the best obtained solution.

Another version of equation 6-1 uses the square root of the mean sum of relative error squared:

RMRS = $\sum\ ((s_i - x_i)/s_i\ )^2$ (6-2)

Equation 6-2 is a better measure, unless the actual solution is zero. In this case, I suggest using a small positive value. While equation 6-2 is an improvement over equation 6-1, it ignores the effect of the original range for variable *i*. The wider the range, the more effort an algorithm has to make to obtain a very good solution. So the square root of the mean sum of relative range error squared is:

RMRRS = $\sum\ ((s_i - x_i)/r_i\ )^2$ (6-3)

Where r_i is the range for acceptable values of variable *i*. Using a wider range yields smaller RMRRS values to reflect the extra effort required to reach the final solution.

I mention the above equations for your information. Since the ACS algorithm yields excellent results, as you will see in the next sections, the above statistics converge to zero for the various test functions.

7-MATLAB ACS Function For Original Pseudo-Code

The MATLAB listing, which defines function acs0(), implements the pseudo-code in section 2 is:

```
function [GlobMin,GlobMinX] = acs0(fx,XLow,XHi,MaxPop,MaxIter,p)
% Implements the Artificial Cooperative Search (ACS)
Optimization.
```

```
%
% Code by Namir Shammas, February 2018.
%
% INPUT
% =====
% f - handle to the optimized function f(x).
% XLow - the array of lower limits.
% XHi - the array of upper limits. Note: You can supply a
single-element
% array to this parameter. The function will adjust the array
size to match
% that of array XLow and will have all elements in array XHi
equal to the
% single value you supplied.
% MaxPop - the Population size.
% MaxIter - the maximum number of iterations
% p - probability of biological interaction in ACS
%
% OUTPUT
% ======
%
% GlobMin - best value for f(x).
% GlobMinX - best solution vector.
%
% REFERENCE
% =========
% "Artificial cooperative search algorithm for numerical
% optimization problems", by Pinar Civicioglu
% Information Sciences 229 (2013) 58–76
%

  if nargin < 6, p = 0.1; end
  if nargin < 5, MaxIter = 2000; end
  if nargin < 4, MaxPop = 200; end

  n = length(XLow);
  n2 = length(XHi);
  if n2 < n && n2 > 0
    XHi = XHi(1) + zeros(1,n);
  end
  PopAlpha = zeros(MaxPop,n);
  PopBeta = zeros(MaxPop,n);
  YAlpha = zeros(MaxPop,1);
  YBeta = zeros(MaxPop,1);

  for i=1:MaxPop
    PopAlpha(i,:) = XLow + (XHi - XLow) .* rand(1,n);
    PopBeta(i,:) = XLow + (XHi - XLow) .* rand(1,n);
    YAlpha(i) = fx(PopAlpha(i,:));
    YBeta(i) = fx(PopAlpha(i,:));
  end

  GlobMin = 1e+99;

  % ------------------------------------- Main Loop
  for iGen=1:MaxIter
    if rand < rand
      Predator = PopAlpha;
      ypred = YAlpha;
      key = 1;
    else
      Predator = PopBeta;
      ypred = YBeta;
      key = 2;
    end

    if rand < rand
      Prey = PopAlpha(:,1:n);
    else
      Prey = PopBeta(:,1:n);
    end
    Prey = Prey(randperm(MaxPop),1:n);
    if rand < rand
      R = 4 * rand *(rand - rand);
    else
      R = gampdf(4 * rand, 1);
    end

    M = 1 + zeros(MaxPop,n);

    for q=1:MaxPop * n
      if rand < p * rand
        i = 1 + fix(rand * MaxPop);
        j = 1 + fix(rand * n);
        M(i,j) = 0;
      end
    end

    if rand < p * rand
        for i=1:MaxPop
          for j=1:n
            if rand < p * rand
              M(i,j) = 1;
            else
```

```
              M(i,j) = 0;
            end
          end
        end
    end

    for i=1:MaxPop
      if sum(M(i,:)) == n
        j = 1 + fix(rand * n);
        M(i,j) = 0;
      end
    end

    % mutation
    x = Predator + R * (Prey - Predator);
    for i=1:MaxPop
      for j=1:n
        % crossover
        if M(i,j) > 0
          x(i,j) = Predator(i,j);
        end
        if x(i,j) < XLow(j) || x(i,j) > XHi(j)
          x(i,j) = XLow(j) + (XHi(j) - XLow(j)) * rand;
        end
      end
    end

    % selection
    for i=1:MaxPop
      zFx = fx(x(i,:));
      if zFx < ypred(i)
        Predator(i,:) = x(i,:);
        ypred(i) = zFx;
      end
    end

    if key==1
      PopAlpha = Predator;
      YAlpha = ypred;
      [Amin,IdxA] = min(YAlpha);
      ybest = Amin;
      bestX = PopAlpha(IdxA,:);
    else
      PopBeta = Predator;
      YBeta = ypred;
      [Bmin,IdxB] = min(YBeta);
      ybest = Bmin;
      bestX = PopBeta(IdxB,1:n);
    end

    if ybest < GlobMin
      GlobMin = ybest;
      GlobMinX = bestX;
    end
  end % iGen
end
```

The function acs0() has the following input parameters:

- The parameter f is handle to the optimized function f(x).
- The single-row vector parameter XLow which specifies the array of lower limits. The function uses the size of this array to deduce the number of variables.
- The single-row vector parameter XHi which specifies the array of upper limits. You can supply a single-element array to this parameter. The function will adjust the array size to match that of array XLow and will have all elements in array XHi equal to the single value you supplied. So, for example, if you supply the arguments [0 0 0 0] and [5] to parameters XLow and XHi, respectively, the function replaces the original [5], initially stored in XHi, into [5 5 5 5] and stores it back into array XHi.
- The parameter MaxPop is the Population size.
- The parameter MaxIter is the maximum number of iterations
- The parameter p is probability of biological interaction in ACS, Typical values are between 0.1 to 0.5.

The function acs1() yields the following output parameters:

- The parameter GlobMin is best value for f(x).
- The vector GlobMinX represents the best solution vector.

Let’s see how the function acs0() work with the test functions. Here is a sample session for testing function fx1:

```
>> [GlobMin,GlobMinX] = acs0(@fx1,[0 0 0 0],[5 5 5 5], 200,
2000, 0.1)
GlobMin =
     0
GlobMinX =
     1     2     3     4
```

The vector [0 0 0 0] is the argument for parameter XLow. The vector [5 5 5 5] is the argument for parameter XHi. The values 200, 2000, and 0.1 are the arguments for parameters MaxPop, MaxIter, and p, respectively. The results are excellent. Here is a sample session for testing function fx2:

```
>> [GlobMin,GlobMinX] = acs0(@fx2,[0 0 0 0], [5 5 5 5], 200,
2000, 0.1)
GlobMin =
     0
GlobMinX =
    1.2340    2.3450    3.4560    4.5670
```

The results are also excellent. Here is a sample session for testing function fx3:

```
>> [GlobMin,GlobMinX] = acs0(@fx3,[0 0 0 0], [25 25 25 25], 200,
2000, 0.1)
GlobMin =
     0
GlobMinX =
     1     4     9    16
```

The results are, one again, excellent, despite the fact we are using a set of much wider ranges. Here is a sample session for testing function rosenbrock:

```
>> [GlobMin,GlobMinX] = acs0(@rosenbrock,[0 0 0 0], [2.15 2.15
2.15 2.15], 200, 2000, 0.1)
GlobMin =
   3.0851e-13
GlobMinX =
    1.0000    1.0000    1.0000    1.0000
```

The results are excellent with the difficult function rosenbrock. Here is a sample session for testing function nle1_4:

```
>> [GlobMin,GlobMinX] = acs0(@nle1_4,[0 0 0 0], [5 5 5 5], 200,
2000, 0.1)
GlobMin =
   1.2735e-12
GlobMinX =
    1.0000    2.0000    3.0000    4.0000
```

The results are excellent here too.

8-First Variant MATLAB ACS Function

The MATLAB listing, which defines function acs1(), implements the pseudo-code in section 3 is:

```
function [GlobMin,GlobMinX] = acs1(fx,XLow,XHi,MaxPop,MaxIter,p)
```

```
% Implements the Artificial Cooperative Search (ACS)
Optimization.
%
% Code by Namir Shammas, February 2018.
%
% INPUT
% =====
% f - handle to the optimized function f(x).
% XLow - the array of lower limits.
% XHi - the array of upper limits. Note: You can supply a
single-element
% array to this parameter. The function will adjust the array
size to match
% that of array XLow and will have all elements in array XHi
equal to the
% single value you supplied.
% MaxPop - the Population size.
% MaxIter - the maximum number of iterations
% p - probability of biological interaction in ACS
%
% OUTPUT
% ======
%
% GlobMin - best value for f(x).
% GlobMinX - best solution vector.
%
% REFERENCE
% =========
% "Artificial cooperative search algorithm for numerical
% optimization problems", by Pinar Civicioglu
% Information Sciences 229 (2013) 58-76
%

  if nargin < 6, p = 0.1; end
  if nargin < 5, MaxIter = 2000; end
  if nargin < 4, MaxPop = 200; end

  n = length(XLow);
  n2 = length(XHi);
  if n2 < n && n2 > 0
    XHi = XHi(1) + zeros(1,n);
  end
  m = n + 1;
  PopAlpha = zeros(MaxPop,m);
  PopBeta = zeros(MaxPop,m);

  for i=1:MaxPop
```

```
    PopAlpha(i,1:n) = XLow + (XHi - XLow) .* rand(1,n);
    PopBeta(i,1:n) = XLow + (XHi - XLow) .* rand(1,n);
    PopAlpha(i,m) = fx(PopAlpha(i,1:n));
    PopBeta(i,m) = fx(PopAlpha(i,1:n));
  end
  [Amin,IdxA] = min(PopAlpha(:,m));
  [Bmin,IdxB] = min(PopBeta(:,m));

  if Amin < Bmin
    GlobMin = Amin;
    GlobMinX = PopAlpha(IdxA,1:n);
  else
    GlobMin = Bmin;
    GlobMinX = PopBeta(IdxB,1:n);
  end

  % ------------------------------------- Main Loop
  for iGen=1:MaxIter
    if rand < rand
      Predator = PopAlpha(:,1:n);
      ypred = PopAlpha(:,m);
      key = 1;
    else
      Predator = PopBeta(:,1:n);
      ypred = PopBeta(:,m);
      key = 2;
    end

    if rand < rand
      Prey = PopAlpha(:,1:n);
    else
      Prey = PopBeta(:,1:n);
    end
    Prey = Prey(randperm(MaxPop),1:n);
    if rand < rand
      R = 4 * rand *(rand - rand);
    else
      R = 1/exp(4* rand); % same as gampdf(4 * rand, 1);
    end

    M = zeros(MaxPop,n);

    for q=1:MaxPop * n
      if rand < p * rand
        i = 1 + fix(rand * MaxPop);
        j = 1 + fix(rand * n);
        M(i,j) = 1;
      end
    end

    if rand < p * rand
        for i=1:MaxPop
          for j=1:n
            if rand < p * rand
              M(i,j) = 1;
            else
              M(i,j) = 0;
            end
          end
        end
    end

    for i=1:MaxPop
      if sum(M(i,:)) == n
        j = 1 + fix(rand * n);
        M(i,j) = 0;
      end
    end

    % mutation
    x = Predator + R * (Prey - Predator);
    for i=1:MaxPop
      for j=1:n
        % crossover
        if M(i,j) > 0
          x(i,j) = Predator(i,j);
        end
        if x(i,j) < XLow(j) || x(i,j) > XHi(j)
          x(i,j) = XLow(j) + (XHi(j) - XLow(j)) * rand;
        end
      end
    end

    % selection
    for i=1:MaxPop
      zFx = fx(x(i,:));
      if zFx < ypred(i)
        Predator(i,:) = x(i,:);
        ypred(i) = zFx;
      end
    end

    if key==1
      PopAlpha(:,1:n) = Predator(:,1:n);
      PopAlpha(:,m) = ypred;
      [Amin,IdxA] = min(PopAlpha(:,m));
      ybest = Amin;
      bestX = PopAlpha(IdxA,1:n);
    else
      PopBeta(:,1:n) = Predator(:,1:n);
      PopBeta(:,m) = ypred;
      [Bmin,IdxB] = min(PopBeta(:,m));
      ybest = Bmin;
      bestX = PopBeta(IdxB,1:n);
    end

    if ybest < GlobMin
      GlobMin = ybest;
      GlobMinX = bestX;
    end
  end % iGen
end
```

The function acs1() has the same input and output parameters as function asc0().

Let's see how the function acs1() work with the test functions. Here is a sample session for testing function fx1:

```
>> [GlobMin,GlobMinX] = acs1(@fx1,[0 0 0 0],[5 5 5 5], 200,
2000, 0.1)
GlobMin =
     0
GlobMinX =
     1     2     3     4
```

The vector [0 0 0 0] is the argument for parameter XLow. The vector [5 5 5 5] is the argument for parameter XHi. The values 200, 2000, and 0.1 are the arguments for parameters MaxPop, MaxIter, and p, respectively. The results are excellent. Here is a sample session for testing function fx2:

```
>> [GlobMin,GlobMinX] = acs1(@fx2,[0 0 0 0], [5 5 5 5], 200,
2000, 0.1)
GlobMin =
     0
GlobMinX =
    1.2340    2.3450    3.4560    4.5670
```

The results are also excellent. Here is a sample session for testing function fx3:

```
>> [GlobMin,GlobMinX] = acs1(@fx3,[0 0 0 0], [25 25 25 25], 200,
2000, 0.1)
```

```
GlobMin =
     0
GlobMinX =
     1     4     9    16
```

The results are, one again, excellent, despite the fact we are using a set of much wider ranges. Here is a sample session for testing function rosenbrock:

```
>> [GlobMin,GlobMinX] = acs1(@rosenbrock,[0 0 0 0], [2.15 2.15
2.15 2.15], 200, 2000, 0.1)
GlobMin =
   3.0851e-13
GlobMinX =
    1.0000    1.0000    1.0000    1.0000
```

The results are excellent with the difficult function rosenbrock. Here is a sample session for testing function nle1_4:

```
>> [GlobMin,GlobMinX] = acs1(@nle1_4,[0 0 0 0], [5 5 5 5], 200,
2000, 0.1)
GlobMin =
   1.2735e-12
GlobMinX =
    1.0000    2.0000    3.0000    4.0000
```

The results are excellent too here.

9-Second Variant MATLAB ACS Function

The MATLAB listing, which defines function acs2(), implements the pseudo-code in section 4 is:

```
function [GlobMin,GlobMinX] = acs2(fx,XLow,XHi,MaxPop,MaxIter,p)
% Implements the Artificial Cooperative Search (ACS)
Optimization.
%
% Code by Namir Shammas, February 2018.
%
% INPUT
% =====
% f - handle to the optimized function f(x).
% XLow - the array of lower limits.
% XHi - the array of upper limits.
% MaxPop - the Population size.
% MaxIter - the maximum number of iterations
% p - probability of biological interaction in ACS
%
% OUTPUT
% ======
%
% GlobMin - best value for f(x).
% GlobMinX - best solution vector.
%
% REFERENCE
% =========
% "Artificial cooperative search algorithm for numerical
% optimization problems", by Pinar Civicioglu
% Information Sciences 229 (2013) 58-76
%
  if nargin < 6, p = 0.1; end
  if nargin < 5, MaxIter = 2000; end
  if nargin < 4, MaxPop = 200; end

  n = length(XHi);
  m = n + 1;
  PopAlpha = zeros(MaxPop,m);
  PopBeta = zeros(MaxPop,m);
  Prey = zeros(MaxPop,n);
  Predator = zeros(MaxPop,n);
  ypred = zeros(MaxPop,1);

  for i=1:MaxPop
    PopAlpha(i,1:n) = XLow + (XHi - XLow) .* rand(1,n);
    PopBeta(i,1:n) = XLow + (XHi - XLow) .* rand(1,n);
    PopAlpha(i,m) = fx(PopAlpha(i,1:n));
    PopBeta(i,m) = fx(PopAlpha(i,1:n));
  end
  [Amin,IdxA] = min(PopAlpha(:,m));
  [Bmin,IdxB] = min(PopBeta(:,m));

  if Amin < Bmin
    GlobMin = Amin;
    GlobMinX = PopAlpha(IdxA,1:n);
  else
    GlobMin = Bmin;
    GlobMinX = PopBeta(IdxB,1:n);
  end

  % -------------------------------------- Main Loop
  for iGen=1:MaxIter
    PopAlpha = sortrows(PopAlpha,m);
    PopBeta = sortrows(PopBeta,m);
    for i=1:MaxPop
      if PopAlpha(i,m) < PopBeta(i,m)
        Predator(i,1:n) = PopAlpha(i,1:n);
        Prey(i,1:n) = PopBeta(i,1:n);
        ypred(i) = PopAlpha(i,m);
      else
        Predator(i,1:n) = PopBeta(i,1:n);
        Prey(i,1:n) = PopAlpha(i,1:n);
        ypred(i) = PopAlpha(i,m);
      end
    end

    Prey = Prey(randperm(MaxPop),1:n);
    if rand < rand
      R = 4 * rand *(rand - rand);
    else
      R = 1/exp(4* rand); % same as gampdf(4 * rand, 1);
    end

    M = zeros(MaxPop,n);

    for q=1:MaxPop * n
      if rand < p * rand
        i = 1 + fix(rand * MaxPop);
        j = 1 + fix(rand * n);
        M(i,j) = 1;
      end
    end

    if rand < p * rand
        for i=1:MaxPop
          for j=1:n
            if rand < p * rand
              M(i,j) = 1;
            else
              M(i,j) = 0;
            end
          end
        end
    end

    for i=1:MaxPop
      if sum(M(i,:)) == n
        j = 1 + fix(rand * n);
        M(i,j) = 0;
      end
    end

    % mutation
    x = Predator + R * (Prey - Predator);
    for i=1:MaxPop
```

```
      for j=1:n
        % crossover
        if M(i,j) > 0
          x(i,j) = Predator(i,j);
        end
        if x(i,j) < XLow(j) || x(i,j) > XHi(j)
          x(i,j) = XLow(j) + (XHi(j) - XLow(j)) * rand;
        end
      end
    end

    % selection
    for i=1:MaxPop
      zFx = fx(x(i,:));
      if zFx < ypred(i)
        Predator(i,:) = x(i,:);
        ypred(i) = zFx;
      end
    end

    if rand < rand
      PopAlpha(:,1:n) = Predator(:,1:n);
      PopAlpha(:,m) = ypred;
      [Amin,IdxA] = min(PopAlpha(:,m));
      ybest = Amin;
      bestX = PopAlpha(IdxA,1:n);
    else
      PopBeta(:,1:n) = Predator(:,1:n);
      PopBeta(:,m) = ypred;
      [Bmin,IdxB] = min(PopBeta(:,m));
      ybest = Bmin;
      bestX = PopBeta(IdxB,1:n);
    end

    if ybest < GlobMin
      GlobMin = ybest;
      GlobMinX = bestX;
    end
  end % iGen
end
```

The function acs2() has the same input and output parameters as function asc0().

Let's see how the function acs2() work with the test functions. Here is a sample session for testing function fx1:

```
>> [GlobMin,GlobMinX] = acs2(@fx1,[0 0 0 0],[5 5 5 5], 200,
2000, 0.1)
GlobMin =
     0
GlobMinX =
     1     2     3     4
```

The vector [0 0 0 0] is the argument for parameter XLow. The vector [5 5 5 5] is the argument for parameter XHi. The values 200, 2000, and 0.1 are the arguments for parameters MaxPop, MaxIter, and p, respectively. The results are excellent. Here is a sample session for testing function fx2:

```
>> [GlobMin,GlobMinX] = acs2(@fx2,[0 0 0 0], [5 5 5 5], 200,
2000, 0.1)
GlobMin =
     0
GlobMinX =
    1.2340    2.3450    3.4560    4.5670
```

The results are excellent. Here is a sample session for testing function fx3:

```
>> [GlobMin,GlobMinX]=acs2(@fx3,[0 0 0 0], [25 25 25 25], 200,
2000, 0.1)
GlobMin =
     0
GlobMinX =
     1     4     9    16
```

The results are also excellent, despite the fact we are using a set of much wider ranges. Here is a sample session for testing function rosenbrock:

```
>> [GlobMin,GlobMinX] = acs2(@rosenbrock,[0 0 0 0], [2.15 2.15
2.15 2.15], 200, 2000, 0.1)
GlobMin =
   3.0851e-13
GlobMinX =
    1.0000    1.0000    1.0000    1.0000
```

The results are excellent given how difficult function rosenbrock is. Here is a sample session for testing function nle1_4:

```
[GlobMin,GlobMinX] = acs2(@nle1_4, [0 0 0 0], [5 5 5 5], 200,
2000, 0.1)
GlobMin =
     0
GlobMinX =
     1     2     3     4
```

The results are excellent too here.

10-Third Variant MATLAB ACS Function

The MATLAB listing, which defines function acs3(), implements the pseudo-code in section 5 is:

```
function [GlobMin,GlobMinX] = acs3(fx,XLow,XHi,MaxPop,MaxIter,p)
% Implements the Artificial Cooperative Search (ACS)
Optimization.
%
% Code by Namir Shammas, February 2018.
%
% INPUT
% =====
% f - handle to the optimized function f(x).
% XLow - the array of lower limits.
% XHi - the array of upper limits.
% MaxPop - the Population size.
% MaxIter - the maximum number of iterations
% p - probability of biological interaction in ACS
%
% OUTPUT
% ======
%
% GlobMin - best value for f(x).
% GlobMinX - best solution vector.
%
% REFERENCE
% =========
% "Artificial cooperative search algorithm for numerical
% optimization problems", by Pinar Civicioglu
% Information Sciences 229 (2013) 58-76
%
  if nargin < 6, p = 0.1; end
  if nargin < 5, MaxIter = 2000; end
  if nargin < 4, MaxPop = 200; end

  n = length(XHi);
  m = n + 1;
  PopAlpha = zeros(MaxPop,m);
  PopBeta = zeros(MaxPop,m);
  Prey = zeros(MaxPop,n);
  Predator = zeros(MaxPop,n);
  ypred = zeros(MaxPop,1);

  for i=1:MaxPop
    PopAlpha(i,1:n) = XLow + (XHi - XLow) .* rand(1,n);
    PopBeta(i,1:n) = XLow + (XHi - XLow) .* rand(1,n);
    PopAlpha(i,m) = fx(PopAlpha(i,1:n));
    PopBeta(i,m) = fx(PopAlpha(i,1:n));
  end
  [Amin,IdxA] = min(PopAlpha(:,m));
  [Bmin,IdxB] = min(PopBeta(:,m));

  if Amin < Bmin
    GlobMin = Amin;
    GlobMinX = PopAlpha(IdxA,1:n);
  else
    GlobMin = Bmin;
    GlobMinX = PopBeta(IdxB,1:n);
  end

  % -------------------------------------- Main Loop
  for iGen=1:MaxIter
    %PopAlpha = sortrows(PopAlpha,m);
    %PopBeta = sortrows(PopBeta,m);
    Pop = [PopAlpha; PopBeta];
    Pop = sortrows(Pop, m);
    Predator = Pop(1:MaxPop,1:n);
    ypred = Pop(1:MaxPop,m);
    Prey = Pop(MaxPop+1:2*MaxPop,1:n);

    Prey = Prey(randperm(MaxPop),1:n);
    if rand < rand
      R = 4 * rand *(rand - rand);
    else
      R = 1/exp(4* rand); % same as gampdf(4 * rand, 1);
    end

    M = zeros(MaxPop,n);

    for q=1:MaxPop * n
      if rand < p * rand
        i = 1 + fix(rand * MaxPop);
        j = 1 + fix(rand * n);
        M(i,j) = 1;
      end
    end

    if rand < p * rand
        for i=1:MaxPop
          for j=1:n
            if rand < p * rand
              M(i,j) = 1;
            else
              M(i,j) = 0;
            end
          end
        end
    end

    for i=1:MaxPop
      if sum(M(i,:)) == n
        j = 1 + fix(rand * n);
        M(i,j) = 0;
      end
    end

    % mutation
    x = Predator + R * (Prey - Predator);
    for i=1:MaxPop
      for j=1:n
        % crossover
        if M(i,j) > 0
          x(i,j) = Predator(i,j);
        end
        if x(i,j) < XLow(j) || x(i,j) > XHi(j)
          x(i,j) = XLow(j) + (XHi(j) - XLow(j)) * rand;
        end
      end
    end

    % selection
    for i=1:MaxPop
      zFx = fx(x(i,:));
      if zFx < ypred(i)
        Predator(i,:) = x(i,:);
        ypred(i) = zFx;
      end
    end

    if rand < rand
      PopAlpha(:,1:n) = Predator(:,1:n);
      PopAlpha(:,m) = ypred;
      [Amin,IdxA] = min(PopAlpha(:,m));
      ybest = Amin;
      bestX = PopAlpha(IdxA,1:n);
    else
      PopBeta(:,1:n) = Predator(:,1:n);
      PopBeta(:,m) = ypred;
      [Bmin,IdxB] = min(PopBeta(:,m));
      ybest = Bmin;
```

```
      bestX = PopBeta(IdxB,1:n);
    end

    if ybest < GlobMin
      GlobMin = ybest;
      GlobMinX = bestX;
    end
  end % iGen
end
```

The function acs3() has the same input and output parameters as function asc0().

Let’s see how the function acs3() work with the test functions. Here is a sample session for testing function fx1:

```
>> [GlobMin,GlobMinX] = acs3(@fx1,[0 0 0 0],[5 5 5 5], 200,
2000, 0.1)
GlobMin =
     0
GlobMinX =
     1     2     3     4
```

The vector [0 0 0 0] is the argument for parameter XLow. The vector [5 5 5 5] is the argument for parameter XHi. The values 200, 2000, and 0.1 are the arguments for parameters MaxPop, MaxIter, and p, respectively. The results are excellent. Here is a sample session for testing function fx2:

```
>> [GlobMin,GlobMinX] = acs3(@fx2,[0 0 0 0], [5 5 5 5], 200,
2000, 0.1)
GlobMin =
     0
GlobMinX =
    1.2340    2.3450    3.4560    4.5670
```

The results are excellent. Here is a sample session for testing function fx3:

```
>> [GlobMin,GlobMinX]=acs3(@fx3,[0 0 0 0], [25 25 25 25], 200,
2000, 0.1)
GlobMin =
     0
GlobMinX =
     1     4     9    16
```

The results are also excellent, despite the fact we are using a set of much wider ranges. Here is a sample session for testing function rosenbrock:

```
>> [GlobMin,GlobMinX] = acs3(@rosenbrock,[0 0 0 0], [2.15 2.15
2.15 2.15], 200, 2000, 0.1)
```

```
GlobMin =
   3.0851e-13
GlobMinX =
    1.0000    1.0000    1.0000    1.0000
```

The results are excellent given how difficult function rosenbrock is. Here is a sample session for testing function nle1_4:

```
[GlobMin,GlobMinX] = acs3(@nle1_4, [0 0 0 0], [5 5 5 5], 200,
2000, 0.1)
GlobMin =
     0
GlobMinX =
     1     2     3     4
```

The results are excellent too here.

11-Conclusion

The MATLAB functions acs0(), acs1(), asc2(), and asc3() perform excellently. Their results are not affected by how these functions differently select the predator and prey populations and how they update the populations A and B, in each iteration. The different versions of the ACS algorithm range from purely random selection of the predator and prey population, to an intermediate merit-based (i.e. function values) approach, and to a heavy reliance on merit. In all cases, the different versions of ACS performed very well. The reason behind this observation is that the ACS mixes well the predator and prey population to yield good new baseline populations of A and B.

I would like to congratulate Pinar Civicioglu, the algorithm designer, for creating a robust and efficient algorithm. The robustness of the algorithms allows it to support design variations without breaking down. I experimented with other variants and still obtained good, but less accurate, results.

References

1. Pinar Civicioglu, “Artificial cooperative search algorithm for numerical optimization problems”, Information Sciences 229 (2013) 58–76.
2. Ramesh Kumar Selvaraju and Ganapathy Somaskandan, “ACS algorithm tuned ANFIS-based controller for LFC in deregulated environment”, Journal of Applied Research and Technology 15 (2017) 152–166.
3. D. Jayakumar and Dr. D. B. Jabaraj, “Energy Optimization of Condenser water loop in HVAC System using Artificial Co-operative Search (ACS) algorithm”,

International Journal of Engineering and Techniques - Volume 2 Issue 3, May – June 2016.

4. Michel Toulouse, Teodor Gabriel Crainic, and K. Thulasiraman, "Global optimization properties of parallel cooperative search algorithms: A simulation study", Parallel Computing 26 (2000) 91–112.
5. Md. Rakib Hassan, Md. Kamrul Hasan, and M.M.A Hashem, "An Improved ACS Algorithm for the Solutions of Larger TSP Problems", Proceedings of the ICEECE December 22-24, Dhaka, Bangladesh.
6. Ramesh Kumar Selvaraju and Ganapathy Somaskandan, "Design of Artificial Cooperative Search algorithm based Load Frequency Controller for Interconnected Deregulated Power Systems with AC-DC Parallel Tie-lines", Australian Journal of Basic and Applied Sciences, 8(13) August 2014, Pages: 326-338.

Document History

Date	***Version***	***Comments***
02/28/2018	1.0.0	Initial release.