Particle Swarm Optimisation

Contents

Summary

The scope of this project is to use the given data (Conroy 1999) that exhibits the small island effect and derive a suitable mathematical equation and its respective parameters such that it reflects the relationship between the number of species against the natural logarithm of area. We have found that the following model fits the data best:

Define the following:

- X is given by $ln(Area)$
- $f(X) = 4 + \frac{a(X+b)}{\pi}(\tan^{-1}(a(X-c)) + \frac{\pi}{2})$

Further employing the use of Particle Swarm Optimisation, we were able to find the parameters such that the Least Squares Errors was the minimum. This indicated that the proposed model, with the found parameters are the best candidate for the provided data.

Model Construction

To meet the requirements, the model must fulfill the following criterions:

I. There should be 2-4 parameters
II. The model should be non-linear
III. There exists a horizontal asymptote for negative values of X
IV. There exists a slant asymptote for positive values of X

The difficulty is to find a nonlinear expression with two asymptotes. After considering exponential, polynomial and trigonometrical models, we decided to follow through with a trigonometrical model involving tan^{-1}. tan^{-1} was specifically chosen because tan^{-1} has two asymptotes, $y = -\frac{\pi}{2}$ and $y = \frac{\pi}{2}$. As we can easily see, $(tan^{-1}(X) + \frac{\pi}{2}) * Linear(X)$ will give a linear function as $X \to \infty$ and 0 as $X \to -\infty$. Based on this idea, we arrived at equation (1).

$$f(X) = e + \frac{a(X+b)}{\pi}(\tan^{-1}(d(X-c)) + \frac{\pi}{2}) \tag{1}$$

It can be easily verified that $f(X) \to e$ as $X \to -\infty$, while $f(X) \to a(X+b)+e$ as $X \to \infty$. To simplify this model, notice that a denotes the slope of the second asymptote and d denotes the rate of change at the 'turning point'. As we expected, the steeper the second asymptote, the sharper the 'turning point' is. As a result, we may assume $d = a$.

As we subsequently found out, if we do not set a to be a constant value, the best fit curve will just be a straight line which is not desired. To make the 'turning point' appear within the domain of the data set, we must deliberately fix a. After some adjustments, we set $e = 4$. Then we arrive at equation (2).

$$f(X) = 4 + \frac{a(X+b)}{\pi}(\tan^{-1}(a(X-c)) + \frac{\pi}{2}) \tag{2}$$

To show that our function is continuously differentiable, we only need to should that tan^{-1} is continuously differentiable which is trivial.

$$\frac{d}{dX} tan^{-1}(X) = \frac{1}{1+X^2}$$

This ensures $f(X)$ to be continuously differentiable.

Theory: Particle Swarm Optimisation

Particle Swarm Optimisation (PSO) is a stochastic optimisation technique originally intended for emulating the social behaviour of bird flocks or fish schools. This method works by iteratively trying to improve a candidate solution within a given measure of quality. This problem is solved by having a large number of particles move around within a defined space, also known as a *search space*. Each particle is affected by its best known position, current position and velocity and these factors are further constrained by the proposed model and its search space. The movement of the swarm of particles is expected to converge towards the best solution over multiple iterations.

To determine our search space, we must consider our proposed model and determine the domain of all the 3 parameters a, b, c. Since $f(X)$ is defined as the number of species, $f(X) \geq 0$, $\forall X \in \mathbb{R}_0^+$

As discussed earlier, $f(X) \to a(X+b) + 4$ as $X \to \infty$. It is easily understood that a must be positive since the asymptote should be increasing as $X \to \infty$. We also notice that the term $tan^{-1}(X-c)$ is actually a translation of $tan^{-1}(X)$ of $+c$ units in the direction of the X-axis. The turning point of the function should be on the right side of y-axis, so c must be positive. There is no constrain on the domain of parameter b.

Hence, we have found that 2 parameters a, c must be positive. We limit our search space within the domain $(0, 10]^2 \subset \mathbb{R}^2$ for a, c and $[-10, 10] \subset \mathbb{R}$ for b. To ensure that the found parameters have the best conformity to the data provided, we must ensure that the Least Squares Error (LSE) is minimised. To do that, we depend on the following equation:

$$E(a, b, c) = \sum_{i=1}^{n} (f(X) - f(X_i, a, b, c))^2$$

where n represents the number of particles.

Curve Fitting: Implementing Optimisation Algorithm

1 Generate Particles

Before the implementation of the PSO algorithm, we must first generate particles to move around the search space. Generating a large number of particles is better as each particle emulates a body within a swarm and a larger swarm translates to a greater probability that the LSE is minimised and conformity between the equation and data is increased. For this, we will set the number of particles involved to be $n = 100$. (Appendix C)

2 Assign Particle Position, Velocity & Calculate LSE

To start the algorithm, each particle must have a position and velocity assigned to it. To do this, we must generate a random number for each of the parameters and velocity constituents using a uniform distribution. The number generated must additionally be contained within the search space. To do this, we make use of the built-in function `rand`. (Appendix D)

After that, we need to ensure that all 100 particles have their position and velocity generated, therefore we need to populate the rest of the matrix by looping (repeating) step 2. After that, we calculate the LSE for each of the particles according to the previously mentioned equation: (Appendix E)

$$E(a, b, c) = \sum_{i=1}^{n} (f(X) - f(X_i, a, b, c))^2$$

3 Initialise Best Current & Global Position Vectors

Within the particle position and velocity matrix, we now need to search for the row with the lowest LSE using the function `min` and tentatively assign the row values to the current best and global best vectors. (Appendix F)

④ Generate New Particle Velocity & Update Particle Position

To emulate the movement of a swarm, we have to calculate new velocities for the movement of each particle. To generate the new velocity of the particles, we make use of the following equation: (Appendix G)

$$\overline{v_{new}} = \omega\overline{v_{old}} + c_1\overline{\psi_1}(\overline{p_{best}} - \overline{p_{current}}) + c_2\overline{\psi_2}(\overline{g_{best}} - \overline{p_{current}})$$

- $\overline{v}$ represents the particle velocity with respect to its parameters a, b, c
- ω represents the inertia weight used to control the velocity.
- c_1 is a constant and represents the cognitive scaling parameter.
- c_2 is another constant and represents the social scaling parameter.
  *Typically, $c_1 = c_2$
- ψ_1, ψ_2 are column vectors with randomly generated entries from the uniform distribution between $[0, 1]$
- $\overline{p_{current}}$ represents the position vector of the particle at the current position.
- $\overline{p_{best}}$ represents the position vector of the particle at the best position.
- $\overline{g_{best}}$ represents the position vector of all particles at the best position.

For the above-mentioned equation, we require the particles to converge towards a local minimum after multiple (1000) loops. After some trials, we decided to use the values $\omega = 0.8$, $c_1 = \ c_2 = \ 1.4$.

Using the new velocity generated by the previously stated equation, we need to update the position of the particle by adding the old position and its newly generated velocity. This can also be written into the following equation: (Appendix G)

$$\overline{p_{new}} = \overline{v_{new}} + \overline{p_{old}}$$

⑤ Run the Algorithm by Looping Multiple Times

We repeat the algorithm 1000 times to ensure that we can find the best possible solutions of the particles with the least LSE. This will also lead us to obtaining the best parameters that are suitable for the model (Appendix G).

Considerations

① Position Bounds

During the updating of the new velocity and position of the particles, we must ensure that the position and velocity are bounded within the domain of the parameters. For example, considering the parameter b, since it has been defined that $b \in [-10, 10] \subset \mathbb{R}$, the position bounds are $-10 \leq p_b \leq 10$ and the velocity bounds are $-20 \leq v_b \leq 20$.

2 Universal Minimum

It must be considered that the first round of PSO algorithm may not provide the solution to the global minimum due to instances where there may be more than 1 local minimum. In such cases, the particles will converge towards that local minimum and other local minimums, which may be a global minimum is severely neglected. Therefore, to provide a workaround to this problem, it is suggested that we re-run the algorithm multiple times from scratch to sufficiently ensure that the results are essentially optimised with the lowest possible LSE and stored into the universal best vector. For this, we run PSO 10 times. (Appendix H)

Results

After implementing the PSO algorithm, we were able to obtain the universal best equation with the following parameters:

$$a = 0.924794 \quad b = 1.009460 \quad c = 5.328689$$

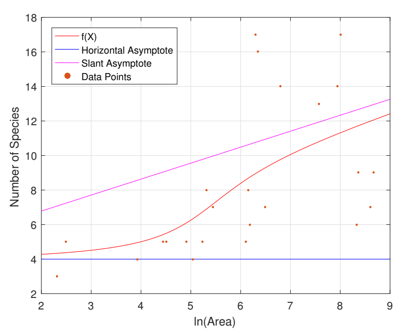
Substituting the obtained parameters into our proposed model gives us the following equation:

$$f(X) = 4 + \frac{0.924794(X + 1.009460)}{\pi}(\tan^{-1}(0.924794(X - 5.328689)) + \frac{\pi}{2})$$

As well as the following LSE:

$$E(a, b, c) = \sum_{i=1}^{n}(f(X) - f(X_i, a, b, c))^2 = 290.313288$$

Using the found parameters and the proposed model, we are able to visualise the curve and its relationship to the data by plotting a graph. Furthermore, the 2 asymptotes further emphasises the shape of the curve for $X \to -\infty$ and $X \to \infty$. A bigger image can be found within Appendix L:



Plot of sample data with the best fit curve and asymptotes.

Appendix

A. Provided Sample Data
B. Data Processing
C. Defining Variables To Be Used
D. Generate Random Position & Velocity Function
E. Initialising Matrices, Particles Starting Position & Velocity
F. Finding and Assigning Best Value
G. Updating Particle Position & Velocity over 1000 loops
H. Ensure Universal Minimum
I. Plotting of Data Points & Graphs
J. Output of Current Code
K. Complete code with Sub-functions
L. Plot Graphic

A. Provided Sample Data

Area (km^2)	$ln(Area)$ (km^2)	Number of Species
3024.2	8.0144	17
547.1	6.3046	17
569	6.3439	16
889.3	6.7904	14
2822	7.9452	14
1933.1	7.5669	13
4309.7	8.3686	9
5777.5	8.6617	9
472.3	6.1576	8
658.2	6.4895	7
5448.9	8.6032	7
202.6	5.3112	8
232.9	5.4506	7
4162.6	8.3339	6
482.2	6.1784	6
446.5	6.1014	5
85.2	4.4450	5
12.1	2.4932	5
135	4.9053	5
189	5.2417	5
91.1	4.5120	5
155.3	5.0454	4
50.6	3.9240	4
10.1	2.3125	3

B. Data Processing

```
%Import data and filter out numbers
[num,txt,raw]=xlsread('data.xlsx');

%Take ln of the Area
num(:,3)=log(num(:,1));

xdata=num(:,3); % Equate x-axis to Ln(Area)
ydata=num(:,2); % Equate y-axis to No. of Species
```

C. Defining Variables To Be Used

```
psorun=10; % Define no. of times PSO repeats from the
    beginning
particlenum = 100; % Define no. of particles to be
    used
repeatnum = 1000; % Define no. of iterations the table
     should update
```

D. Generate Random Position & Velocity Function

```
function [a,b,c,va,vb,vc]=rndgen()

a=10*rand();
va=10*rand();
b=-10+(10+10)*rand();
vb=-10+(10+10)*rand();
c=10*rand();
vc=10*rand();

end
```

E. Initialising Matrices, Particles Starting Position & Velocity

```
% PSO Table:
% [a,b,c,va,vb,vc,lse]
for i=1:particlenum
    [a,b,c,va,vb,vc]=rndgen(); % Send to function to
        generate numbers
    pgen(i,:)=[a,b,c,va,vb,vc,0]; %Populate every row
        of PSO table
    pgen(i,7)=sum((ydata-(4+(pgen(i,1).*(xdata+pgen(i
        ,2))).*(1/pi)...
        .*(atan(pgen(i,1).*(xdata-pgen(i,3)))+(pi/2))
            )).^2);
end
```

F. Finding and Assigning Best Value

```
findbestval=(find(min(pgen(:,7))==pgen(:,7)));

% Current and global best matrix initialisation:
% [bestlse(a),bestlse(b),bestlse(c),bestlse(a,b,c)]
cbest=[pgen(findbestval,1:3),pgen(findbestval,7)];
gbest=cbest;
```

G. Updating Particle Position & Velocity over 1000 loops

```
wght=0.8;
lf=1.4;
% Updating values
for i=1:repeatnum
    for j=1:particlenum
        % Generate New Velocity
        pgen(j,4:6)=wght*pgen(j,4:6)+lf*rand(1,3)...
            .*(cbest(1:3)-pgen(j,1:3))+lf*rand(1,3)...
            .*(gbest(1:3)-pgen(j,1:3));

        % Generate New Position
        pgen(j,1:3)=pgen(j,1:3)+pgen(j,4:6);

        % Position Bounds Check
        if pgen(j,1)<0||pgen(j,1)>10||pgen(j,2)<-10||
            pgen(j,2)>10||...
```

```
                pgen(j,3)<0||pgen(j,3)>10
            [a,b,c,va,vb,vc]=rndgen();
            pgen(j,1)=a;
            pgen(j,2)=b;
            pgen(j,3)=c;
            pgen(j,4)=va;
            pgen(j,5)=vb;
            pgen(j,6)=vc;
        end

        % LSE Error
        pgen(j,7)=sum((ydata-(4+(pgen(j,1).*(xdata+pgen(j,2)))...
            .*(1/pi).*(atan(pgen(j,1).*(xdata-pgen(j,3)))+(pi/2)))))...
            .^2);
        end

        % Find row with minimum LSE and store in current best matrix
        findbestval=(find(min(pgen(:,7))==pgen(:,7)));
        cbest=[pgen(findbestval,1:3),pgen(findbestval,7)];

        % Check whether current particle LSE is lower than global particle LSE
        if cbest(4)<gbest(4)
            gbest=cbest;
        end

    end
end
```

H. Ensure Universal Minimum

```
% Run PSO multiple times
for k=1:psorun

    <Code from Updating Particle Position & Velocity over 1000 loops>

    % Initialise universal matrix if in first loop
    if k==1
        ggbest=gbest;
```

```
    else % Otherwise check whether global LSE <
        universal LSE
        if gbest(4)<ggbest(4)
            ggbest=gbest;
        end
    end
end
```

I. Plotting of Data Points & Graphs

```
% Plotting of graph
x=[0:0.01:10]; %Define region to plot
x1 = 4+(ggbest(1).*(x+ggbest(2))).*(1/pi).*(atan(
    ggbest(1).*(x-ggbest(3)))+(pi/2));
plot(x,x1,'r');
grid on
hold on

% Horizontal asymptote
fplot(4,[0,10],'b')

% Slant asymptote
slantasymp=4+(ggbest(1).*(x+ggbest(2)));
plot(x,slantasymp,'m');

% Scatter plot of data
scatter(xdata,ydata,5,'filled');

% Define axes
axis([2 9 2 18]);

% Define legend & location
lgd = legend('f(X)','Horizontal Asymptote','Slant
    Asymptote','Data Points');
lgd.Location='northwest';

% Define all graph labels
xlabel('ln(Area)');
ylabel('Number of Species');
```

J. Output of Current Code

```
fprintf('PSO has found the following parameters to be the best:\n\n');
fprintf('   a: %f\n   b: %f\n   c: %f\n LSE: %f\n\n'...
    ,ggbest(1),ggbest(2),ggbest(3),ggbest(4));
```

```
PSO has found the following parameters to be the best:

   a: 0.924794
   b: 1.009460
   c: 5.328689
 LSE: 290.313288
```

K. Complete code with Subfunctions

```
%Import data and filter out numbers
[num,txt,raw]=xlsread('data.xlsx');

%Take ln of the Area
num(:,3)=log(num(:,1));

xdata=num(:,3); % Equate x-axis to Ln(Area)
ydata=num(:,2); % Equate y-axis to Number of Species

% Particle Swarm Optimisation
% Initialise variables

% Initialise tables of variables
psorun=10; % Define no. of times PSO repeats from the beginning
particlenum = 100; % Define no. of particles to be used
repeatnum = 1000; % Define no. of iterations the table should update

pgen=zeros(particlenum,7);

% Run PSO multiple times
for k=1:psorun

    % PSO Table:
    % [a,b,c,va,vb,vc,lse]
    for i=1:particlenum
```

```
        [a,b,c,va,vb,vc]=rndgen(); % Send to function
            to generate numbers
        pgen(i,:)=[a,b,c,va,vb,vc,0]; %Populate every
            row of PSO table
        pgen(i,7)=sum((ydata-(4+(pgen(i,1).*(xdata+
            pgen(i,2))).*(1/pi)...
        .*(atan(pgen(i,1).*(xdata-pgen(i,3)))+(pi/2)))
            ).^2); %LSE Error
    end

    findbestval=(find(min(pgen(:,7))==pgen(:,7)));

    % Current and global best matrix initialisation:
    % [bestsse(a),bestsse(b),bestsse(c),bestsse]
    cbest=[pgen(findbestval,1:3),pgen(findbestval,7)];
    gbest=cbest;

    wght=0.8; % Weight
    lf=1.4; % Learning factor
    % Updating values
    for i=1:repeatnum
        for j=1:particlenum
            % Generate New Velocity
            pgen(j,4:6)=wght*pgen(j,4:6)...
                +lf*rand(1,3).*(cbest(1:3)-pgen(j,1:3)
                    )...
                +lf*rand(1,3).*(gbest(1:3)-pgen(j,1:3)
                    );

            % Generate New Position
            pgen(j,1:3)=pgen(j,1:3)+pgen(j,4:6);

            % Position Bounds Check
            if pgen(j,1)<0||pgen(j,1)>10||pgen(j,2)
                <0||pgen(j,2)>10||...
                    pgen(j,3)<-10||pgen(j,3)>10
                [a,b,c,va,vb,vc]=rndgen();
                pgen(j,1)=a;
                pgen(j,2)=b;
                pgen(j,3)=c;
                pgen(j,4)=va;
                pgen(j,5)=vb;
                pgen(j,6)=vc;
            end

            %LSE Error
```

```
            pgen(j,7)=sum((ydata-(4+(pgen(j,1).*(xdata+pgen(j,2))).*(1/pi)...
            .*(atan(pgen(j,1).*(xdata-pgen(j,3)))+(pi/2)))).^2);
        end

        % Find row with minimum LSE and store in current best matrix
        findbestval=(find(min(pgen(:,7))==pgen(:,7)));
        cbest=[pgen(findbestval,1:3),pgen(findbestval,7)];

        %Check whether current particle LSE is lower than global particle LSE
        if cbest(4)<gbest(4)
            gbest=cbest;
        end
    end

    % Initialise universal matrix if in first loop
    if k==1
        ggbest=gbest;
    else % Otherwise check whether global LSE < universal LSE
        if gbest(4)<ggbest(4)
            ggbest=gbest;
        end
    end
end

% Plotting of graph
x=[0:0.01:10]; %Define region to plot
x1 = 4+(ggbest(1).*(x+ggbest(2))).*(1/pi).*(atan(ggbest(1).*(x-ggbest(3)))+(pi/2));
plot(x,x1,'r');
grid on
hold on

% Horizontal asymptote
fplot(4,[0,10],'b');

% Slant asymptote
slantasymp=4+(ggbest(1).*(x+ggbest(2)));
plot(x,slantasymp,'m');

% Scatter plot of data
```

```
scatter(xdata,ydata,5,'filled');

% Define axes
axis([2 9 2 18]);

% Define legend & location
lgd = legend('f(X)','Horizontal Asymptote','Slant
    Asymptote','Data Points');
lgd.Location='northwest';

% Define all graph labels
xlabel('ln(Area)');
ylabel('Number of Species');

fprintf('PSO has found the following parameters to be
    the best:\n\n');
fprintf('    a: %f\n    b: %f\n    c: %f\n LSE: %f\n\n'...
    ,ggbest(1),ggbest(2),ggbest(3),ggbest(4))
```

```
function [a,b,c,va,vb,vc]=rndgen()

a=10*rand();
va=10*rand();
b=-10+(10+10)*rand();
vb=-10+(10+10)*rand();
c=10*rand();
vc=10*rand();

end
```

```
PSO has found the following parameters to be the best:

   a: 0.924817
   b: 1.009375
   c: 5.328718
 LSE: 290.313288
```

L. Plot Graphic