Parameter Identification of Dynamical Systems with Spectral Richness PSO algorithm

User manual

Version 1.0.0

—

May 2, 2022

Contents

List of Code Listings

1 Introduction

The present document describes the process of identification of parameters for a dynamical system with the PSO algorithm and a modified version called Spectral Richness PSO (SR-PSO) algorithm on MATLAB software. A brief description from the parameter identification problem and the issues related to the excitation signal on classical examples are included. The given description of the identification problem is used to propose the optimization problem that the PSO solves. The description includes the coding of the PSO algorithms on Matlab, the ways to implement the simulation of the dynamical systems in order to use the estimated parameters to compute a simulated output that is used on a performance index to determinate if the estimates produce a behavior equivalent to the original system to be identified. Two ways to simulate the dynamical system are included, the first one use the SIMULINK environment of MATLAB to evaluate the estimated parameters and the second one consists on implement the dynamical model as code using the Euler integration method in order to decrease the computing cost. The estimation of the Spectral Richness from a signal is included and the way this property of the excitation applied to the system is used to compute the local and global weights used on the PSO algorithm. The way to execute the PSO algorithm given on MATLAB is included step by step. A troubleshooting section includes several hints to solve some implementation problems that could produce errors to the user.

2 Parameter identification of dynamical systems

The mathematical model from a dynamical system use differential equations to describe a time varying phenomenon that depends from the initial conditions.

The model is expressed as follows:

$$\begin{aligned} \dot{\mathbf{x}} &= f(\mathbf{x}, t, u, \theta) \\ y &= g(\mathbf{x}, t, u) \end{aligned} \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n$ are the states of the system, $y \in \mathbb{R}$ is the measurable output, t is the time, $u \in \mathbb{R}$ is a input and $\theta \in \mathbb{R}^m$ are the parameters.

The vector θ contains the parameters of the model, these define the behavior of the model and for the case of real systems usually represents physical properties that could be mechanical, electrical, thermal, chemical, etc. In some cases, the parameters could not have a direct physical interpretation since several models implements complex expressions for the description of the behavior like dead-zones, hysteresis, etc.

The main idea used on the parameter estimation is the implementation from a virtual model:

$$\begin{aligned} \dot{\mathbf{x}}_\mathbf{m} &= f(t, \mathbf{x}_\mathbf{m}, u_m, \hat{\theta}) \\ y_m &= g(t, \mathbf{x}_\mathbf{m}, u_m) \end{aligned} \tag{2}$$

being $\mathbf{x_m} \in \mathbb{R}^n$, $y_m \in \mathbb{R}$ and $u_m \in \mathbb{R}$ respectively the state, the output and the input of model. Where the input u_m have the same structure that the one used on the original system, the only difference is that $\mathbf{x}$ is substituted by $\mathbf{x_m}$. The $\hat{\theta}$ is a vector of estimated parameters. In order to generate an accurate simulation of the system or to design a control law, a estimation of the parameter vector $\hat{\theta} \in \mathbb{R}^m$ must be computed such $\hat{\theta}$ are equal or at least have similar values to θ.

The estimation of the parameters could be computed with several techniques of the Automatic Control area like the Least-Square algorithm, Gradient algorithm, Algebraic methods, etc.

2.1 Optimization problem

Since the main problem is to find a vector $\hat{\theta}$ equal or similar to θ, the principal idea used to solve this problem is make a comparison between the outputs of the both systems. Since both systems have the same mathematical structure, is expected that if y_m is close to y the parameters of the two models have similar values.

Such the optimization problem could be define as:

$$\begin{aligned} \min \quad & J(\hat{\theta}_\mathbf{i}(\mathbf{k})) \\ \text{subject to:} \quad & \Omega_s := \left\{ \hat{\theta}_\mathbf{i}(\mathbf{k}) | \, \underline{\theta}_r \leq \hat{\theta}_{ir} \leq \bar{\theta}_r, r = 1, \dots, m \right\}. \end{aligned} \tag{3}$$

The performance function J is defined as:

$$J(\hat{\theta}_i(k)) = \int_0^T |e_m(\tau)| d\tau; \; \hat{\theta}_\mathbf{i}(\mathbf{k}) \in \Omega_s \subset \mathbb{R}^m \tag{4}$$

where the error model $e_m = y - y_m$.

2.2 Classical issues related to the Persistent Excitation

Many of the classical and novel algorithms on the Automatic Control area for the parameter identification are based on the rewritten of the model into a regression form given as:

$$z = \theta^\top \phi \tag{5}$$

where $\phi \in \mathbb{R}^m$ is a regressor vector and z is the output of the system. The convergence of $\hat{\theta}$ to θ on this class of algorithms depends that a Persistent Excitation (PE) condition be accomplished. This conditions is defined as follows:

$$\sigma_2 I \geq \int_{t_0}^{t_0+\delta} \phi(\tau)\phi^\top(\tau)d\tau \geq \sigma_1 I \quad \forall \quad t_0 > 0 \tag{6}$$

where σ_1, σ_2 and δ are positive constants and I the identity matrix. This expression implies the integral over the time of $\phi(\tau)\phi^\top$ is positive definite.

A correlation between the PE condition and a property from the excitation signal applied to the system called Spectral Richness has been defined previously on the literature.

Let $r \in \mathbb{R}$ be a stationary signal, then r is called *Sufficiently Rich of order* η if its Spectral Measure $S_r(\omega)$ has a support of at least η points. If the *Sufficiently Rich of order* is fulfilled the PE condition is also accomplished, this implies the convergence of the vector $\hat{\theta}$ to θ could be assured.

2.3 Spectral Richness numerical estimation

The Spectral Richness (SR) from a signal r is defined as the cardinality $\chi = \left|\Sigma_r(\omega)\right|$. However, since the Spectral Measure $S_r(\omega)$ analytical calculation could only be performed for few and simple cases, a numerical estimation could be obtained through the use of the Discrete Fast Fourier Transform (FFT).

The FFT only provides the spectrum from the signal on the interval $\Psi = [0, f_s]$, being f_s the sampling frequency used for the acquisition from the signal r. Such the set of spectral measures is defined as:

$$\Sigma_r(\omega) = \{\Sigma_{r1}(\omega_1), \ldots, \Sigma_{rm}(\omega_m)\}, \omega_i \in \Psi \tag{7}$$

Since exists several potential excitation signals r, the normalized spectral measure is defined as:

$$\bar{\Sigma}_r(\omega) = \{\bar{\Sigma}_{r1}(\omega_1), \ldots, \bar{\Sigma}_{rm}(\omega_m)\}, \omega_i \in \Psi \tag{8}$$

$$\bar{\Sigma}_{ri}(\omega_i) = \frac{\Sigma_{ri}(\omega_i)}{\max\limits_{\omega_i \in \Psi}(\Sigma_{ri}(\omega_i))}. \tag{9}$$

In order to discard the residual spectral lines that are produced by the FFT errors produced by the numerical implementation, a end set $\tilde{\Sigma}_r(\omega)$ is defined

as:

$$\tilde{\Sigma}_r(\omega) = \{\tilde{\Sigma}_{r1}(\omega_1), \ldots, \tilde{\Sigma}_{r\chi}(\omega_\chi)\}, \omega_i \in \Psi \quad (10)$$

$$\tilde{\Sigma}_{ri}(\omega_i) = \begin{cases} 0 & \text{if} \quad \bar{\Sigma}_{ri}(\omega_i) < \kappa \\ 1 & \text{if} \quad \bar{\Sigma}_{ri}(\omega_i) \geq \kappa \end{cases} \quad (11)$$

where the threshold κ is defined as:

$$\kappa = 1 - \pi Z \cot(\pi Z) \quad (12)$$

with $Z = \psi/N$ and N is the number of samples of r.

3 PSO algorithm

The PSO algorithm is written as:

$$\begin{aligned} \nu_i(k+1) &= \nu_{\mathbf{i}}(k) + \phi_1(k)p\left(\mathbf{p}_{\mathbf{b},\mathbf{i}}(k) - \theta_{\mathbf{i}}(k)\right) \\ &\quad + \phi_2(k)l\left(\mathbf{g}_{\mathbf{b}}(k) - \theta_{\mathbf{i}}(k)\right) \\ \theta_{\mathbf{i}}(k+1) &= \theta_{\mathbf{i}}(k) + \nu_{\mathbf{i}}(k+1). \end{aligned} \quad (13)$$

The vectors $\theta_{\mathbf{i}}(k), \nu_{\mathbf{i}}(k) \in \mathbb{R}^m$ are the position and the velocity of the particle i, and $\theta_{\mathbf{i}}(k)$ corresponds to a solution for the optimization problem. Moreover, the initial condition $\theta_{\mathbf{i}}(0) \in \Omega_s$ is set at random values.

The term $\mathbf{p}_{\mathbf{b},\mathbf{i}}(k)$, also known as pbest, is called the best evaluated solution obtained by the particle i when it produces the lowest value on the performance index $J(\mathbf{p}_{\mathbf{b},\mathbf{i}}(k))$:

$$\mathbf{p}_{\mathbf{b},\mathbf{i}}(k) = {}_{s\in[1,\ldots,k]} \left\{J\left(\theta_{\mathbf{i}}(s)\right)\right\}. \quad (14)$$

Analogously, the term $\mathbf{g}_{\mathbf{b}}$, also known as gbest, corresponds to the best evaluated solution over the swarm defined as:

$$\mathbf{g}_{\mathbf{b}}(k) = {}_{g\in[1,\ldots,\rho]} \left\{J\left(\theta_{\mathbf{g}}(k)\right)\right\}. \quad (15)$$

3.1 Code implementation

The main program to execute the PSO algorithm is contained on the file **PSO.m**, this one allows the estimation of parameters from a dynamical system. On this file the particular case of the thermoelectric model from a cooler is given, for example purposes and to allow the user the test of different class of models the data from the model to be identified is generated

by a numerical simulation given by a function contained on the **Dynamical_simulation_nonideal.m** file. The numerical implementation is shown on Listing 1, on this script must be noticed that a input perturbation term given by a sine function is included to simulate the effects from an air wave on the ambient and a emulation of the measurement noise based on a random vector is also included in order to increase the similarity to the data that is obtained on a experimental test.

```
function Y = Dynamical_simulation_nonideal(r,t,y0,theta)
    k=theta(1);
    z=theta(2);
    p1=theta(3);
    p2=theta(4);
    Cf=(k*p1*p2)/z;
    ts=t(2)-t(1);
    N=length(t);
    x1=zeros(N,1);
    x2=zeros(N,1);
    x1(1)=y0(1);
    x2(1)=y0(2);
%% Implementation of the input perturbation
    rm=r+0.01*sin(t);
%% Evaluation of the dynamical system using the Euler method
    for i=2:N
        x1(i)=x1(i-1)+ts*x2(i-1);
        if i==2
            dr=(rm(i-1)-0)/ts;
        else
            dr=(rm(i-1)-rm(i-2))/ts;
        end
        Din=-x2(i-1)*(p1+p2)-x1(i-1)*p1*p2+Cf*(-z*rm(i-1)-dr);
        x2(i)=x2(i-1)+ts*Din;
    end
%% Definition of the output from the dynamical system with
%  measurement noise
    Y=x1+rand(N,1)*0.005;
end
```

Listing 1: Simulation from a dynamical system to provide data for the test of the PSO algorithm given on the file **Dynamical_simulation_nonideal.m**

On the Listing 2 the initial section of the PSO is included, on this one could be seen that the reference signal, and initial conditions are included. On the

particular case given on the repository the set of parameters *theta* used for the generation of the data from the dynamical simulation, in the case that experimental data is used only is required that the output *y* is loaded as a vector and saved with the name *Data*. The initialization values from the PSO algorithm are included, if other model is going to be implemented the variable *PN* must be change to the number of parameters to be identified on the new model and the dimension and values for constraints bounds *LB* and *UB* must be adjusted to correspond with the constraints for each parameter from the new model.

```
close all; clear all; clc;
load('References.mat')
r=White_noise; %Selection the reference to be used
y0=[0,0]; %Implementation of the initial conditions
theta=[2.5466,0.1375,0.0115,0.1379]; %Set real parameters
Data=Dynamical_simulation_nonideal(r,t,y0,theta); %Simulation
PN=4;                       %Number of parameters
NP=30;                      %Number of particles on the swarm
K_max=200;                  %Maximum number of iterations
LB=[0,0,0,0];               %Lower constraint boundaries
UB=[8,3,3,3];               %Upper constraint boundaries
alpha=0.5;                  %Stop criteria
C1=1.5;                          %Weigth for Global Solution
C2=1.5;                          %Weigth for Local Solution
Swarm=zeros(NP,PN);
for i=1:PN
    A=rand(NP,1)*(UB(i)-LB(i))+LB(i);
    Swarm(:,i)=A;
end
Vel_1=zeros(NP,K_max);
Vel_2=zeros(NP,K_max);
Vel_3=zeros(NP,K_max);
Vel_4=zeros(NP,K_max);
Save_pos=zeros(K_max,PN);
Save_eva=zeros(K_max,1);
Evaluation=zeros(NP,K_max);
```

Listing 2: Initialization of the algorithm PSO on the file **PSO.m**

The evaluation from the first iteration of the algorithm is given on Listing 3. On this section the way the PSO evaluate the particles on the swarm is given by a function contained on **Evaluation_swarm.m**, the selection of the lbest position and local best evaluation use the values of the swarm since

a previous iteration does not exists. The selection of the gbest solution is performed taking the best solution from the *Evaluation* vector and saving the position of the particle and their evaluation. The vectors *Save_pos* and *Save_eva* are used to record the changes of the gbest solution over the time. The estimation of the particle velocities given by the equation (13) is computed by a function contained on **Velocity_estimation.m** file. In order to assure that the new positions from the particles are contained on the feasible set a function given on the file **Boundary_limit.m** that limit the decreases the velocity values until the obtained position of the particle is constrained by the boundaries. At final, the position of the particle is updated using a function given on the file **Update_pos.m** based on the expression (13).

```
Ite=1;                    %Number of iteration
Evaluation(:,Ite)=Evaluation_swarm(r,t,y0,Swarm,Data,NP)';
LBP=Swarm;
LBE=Evaluation(:,Ite);
[Sort_list,Index]=sort(Evaluation(:,Ite));
GBP=Swarm(Index(1),:);
GBE=Evaluation(Index(1),Ite);
Save_pos(Ite,:)=GBP;
Save_eva(Ite)=GBE;
Vel=Velocity_estimation(Swarm,Vel_1,Vel_2,Vel_3,Vel_4,NP,...
GBP,LBP,C1,C2,Ite);
Vel_lim=Boundary_limit(Swarm,Vel,LB,UB,NP,PN);
Vel_1(:,Ite+1)=Vel_lim(:,1);
Vel_2(:,Ite+1)=Vel_lim(:,2);
Vel_3(:,Ite+1)=Vel_lim(:,3);
Vel_4(:,Ite+1)=Vel_lim(:,4);
Swarm=Update_Pos(Swarm,Vel_1,Vel_2,Vel_3,Vel_4,NP,PN,Ite);
```

Listing 3: First iteration from the PSO algorithm on the file **PSO.m**

The while cycle that is used to implement of the particle swarm is shown on Listing 4. On this, a while cycle is repeated until the number of iterations reach the value of *K_max* implying the acceptable maximum number of iterations is reached or the gbest solution reach a value lower to *alpha*, both values are defined for the user as be shown on Listing 2. For the update of the particles all the previously applied functions has been used in order, the only exception is the selection of the lbest value that requires the use of a function defined on the file **Update_LBP.m**. Once the while cycle is broken the gbest solution computed is displayed.

```
while 1
    Ite=Ite+1;
    Evaluation(:,Ite)=Evaluation_swarm(r,t,y0,Swarm,Data,NP)';
    [LBP,LBE] = Update_LBP(LBP,LBE,Swarm,Evaluation,Ite,NP);
    [Sort_list,Index]=sort(Evaluation(:,Ite));
    if Evaluation(Index(1),Ite)<=GBE
        GBP=Swarm(Index(1),:);
        GBE=Evaluation(Index(1),Ite);
    end
    Save_pos(Ite,:)=GBP;
    Save_eva(Ite,:)=GBE;
    Vel=Velocity_estimation(Swarm,Vel_1,Vel_2,Vel_3,Vel_4,NP,...
    GBP,LBP,C1,C2,Ite);
    Vel_lim=Boundary_limit(Swarm,Vel,LB,UB,NP,PN);
    Vel_1(:,Ite+1)=Vel_lim(:,1);
    Vel_2(:,Ite+1)=Vel_lim(:,2);
    Vel_3(:,Ite+1)=Vel_lim(:,3);
    Vel_4(:,Ite+1)=Vel_lim(:,4);
    Swarm=Update_Pos(Swarm,Vel_1,Vel_2,Vel_3,Vel_4,NP,PN,Ite);

    if Ite>K_max
        break;
    end
    if GBE<alpha
        break;
    end
end
disp('The best estimated parameters are :')
GBP
disp('With a performance function (J) value of:')
GBE
```

Listing 4: Computing of swarm changes and display from the computed result from the PSO algorithm on the file **PSO.m**

3.1.1 Independent functions

As be mentioned several independent functions has been defined in order to code a clear defined PSO algorithm, each of these functions implements a equation related to the update of the particles or to a mechanism used to compare between solutions or assure the solutions fulfill the bounds given by the user. These functions are described on the following paragraphs.

The function EVALUATION_SWARM() is used to use the parameters computed as position of the particles on the PSO on the dynamical system simulation to compute the performance function. The functions is contained on the file **Evaluation_swarm.m**, the contain of this file is shown on Listing 5. Could be noticed that each particle of the swarm is evaluated on the for cycle, each set of parameters are test on a simulation of the dynamical model, on this particular case is given by a function DYNAMICAL_SIMULATION_IDEAL(). Must be noticed that the simulation must have as input the reference signal *r* applied to the original system, the time vector *t*, the initial conditions that exists during the experimentation and the set of parameters that corresponds to the position of the particle. Notice that the applied performance function corresponds to the one given on (4) that is the integral of the absolute value from the error, the user could replace their own performance function on this section from the code.

```
function Y = Evaluation_swarm(r,t,y0,Swarm,Data,NP)
Evaluation=zeros(NP,1);
for i=1:NP
    Ev_num=Dynamical_simulation_ideal(r,t,y0,Swarm(i,:));
    J=sum(abs(Data-Ev_num));
    Evaluation(i,1)=J;
end
Y=Evaluation;
end
```

Listing 5: Implementation of the particles as parameters on the dynamical model to generate a output and the use of this output to compute the performance function on the file **Evaluation_swarm.m**

The function VELOCITY_ESTIMATION() that computes the velocity from the particles based on the expression (13) is contained on the file **Velocity_estimation.m**. Is noticed that each component of velocity for each particle is computed and saved during the for cycle. the required inputs to this function are the position of the particles, the velocities that the particles have on the previous iteration, the number of particles, the gbest solutions, the set of lbest solutions, the terms C1 and C2 that are the given weights and the number of iteration.

```
function Y = Velocity_estimation(Swarm,Vel_1,Vel_2,Vel_3...
                                 ,Vel_4,NP,GBP,LBP,C1,C2,Ite)
    V1=zeros(NP,1);
    V2=zeros(NP,1);
    V3=zeros(NP,1);
    V4=zeros(NP,1);
    for i=1:NP
        V1(i)=Vel_1(i,Ite)+C1*rand*(GBP(1)-Swarm(i,1))...
            +C2*rand*(LBP(i,1)-Swarm(i,1));
        V2(i)=Vel_2(i,Ite)+C1*rand*(GBP(2)-Swarm(i,2))...
            +C2*rand*(LBP(i,2)-Swarm(i,2));
        V3(i)=Vel_3(i,Ite)+C1*rand*(GBP(3)-Swarm(i,3))...
            +C2*rand*(LBP(i,3)-Swarm(i,3));
        V4(i)=Vel_4(i,Ite)+C1*rand*(GBP(4)-Swarm(i,4))...
            +C2*rand*(LBP(i,4)-Swarm(i,4));
    end
    Y=[V1,V2,V3,V4];
end
```

Listing 6: Implementation of the particles as parameters on the dynamical model to generate a output and the use of this output to compute the performance function on the file **Velocity_estimation.m**

The function BOUNDARY_LIMIT() is used to assure the new positions of the particles fulfills the conditions that define the feasible set of solutions mentioned on (3) is contained on the file **Boundary_limit.m**. For this particular case a mechanism based on the limitation of the velocity is coded as be shown on Listing 7. Could be noticed that a vector for the test of the particles on the swarm is defined, each particle is tested using a while cycle in order to evaluate with conditional expressions if the solution remains on the set Ω_s that divides to the half the velocity of the particle until the resulting solution satisfies the conditionals. On this case, only constraint boundaries are included, but in case the optimization problem includes other class of equality or inequality constraints these must be implemented into this while cycle. Once the conditionals are satisfied for all the particles, the resulting velocities are sent as output.

```
function Y = Boundary_limit(Swarm,Vel,LB,UB,NP,PN)
    Swarm_test=zeros(NP,PN);  %Swarm for testing solutions
    Vel_Lim=zeros(NP,PN);
    for i=1:NP
        Check=1;
        while 1
            Swarm_test(i,1)=Swarm(i,1)+Vel(i,1);
            Swarm_test(i,2)=Swarm(i,2)+Vel(i,2);
            Swarm_test(i,3)=Swarm(i,3)+Vel(i,3);
            Swarm_test(i,4)=Swarm(i,4)+Vel(i,4);
            if (Swarm_test(i,1)>=LB(1))&&(Swarm_test(i,1)<=UB(1))&&...
               (Swarm_test(i,2)>=LB(2))&&(Swarm_test(i,2)<=UB(2))&&...
               (Swarm_test(i,3)>=LB(3))&&(Swarm_test(i,3)<=UB(3))&&...
               (Swarm_test(i,4)>=LB(4))&&(Swarm_test(i,4)<=UB(4))
                Check=0;
            end
            if Check==0
                Vel_Lim(i,:)=Vel(i,:);
                break;
            end
            Vel(i,:)=Vel(i,:)/2;
        end
    end
    Y=Vel_Lim;
end
```

Listing 7: Implementation of the mechanism used to constraint the position of the particles to the given boundaries coded on the file **Boundary_limit.m**

The function UPDATE_POS() is used to update the position of the particles using the previously limited velocities using the expression given on (13) and is coded on the file **Update_Pos.m** that is shown on the Listing 8. On this, is noted that each particle position is updated using the velocity given using a for cycle.

```
function Y = Update_Pos(Swarm,Vel_1,Vel_2,Vel_3,Vel_4,NP,PN,Ite)
    New_Swarm=zeros(NP,PN);
    for i=1:NP
        New_Swarm(i,1)=Swarm(i,1)+Vel_1(i,Ite+1);
        New_Swarm(i,2)=Swarm(i,2)+Vel_2(i,Ite+1);
        New_Swarm(i,3)=Swarm(i,3)+Vel_3(i,Ite+1);
        New_Swarm(i,4)=Swarm(i,4)+Vel_4(i,Ite+1);
    end
    Y=New_Swarm;
end
```

Listing 8: Implementation of the mechanism used to constraint the position of the particles to the given boundaries coded on the file **Update_Pos.m**

Finally, the UPDATE_LBP() function is used to compare the lbest solution that has been computed during the previous iteration with their corresponding particle on the updated swarm in order to update this parameter. This function is coded on the file **Update_LBP.m** and their content is show on the Listing 9, is noticed that this code just compare the lbest solution from each particle with the evaluation of the new particle, if the new particle generate a best results then replace the previous value of the lbest, on the other case the lbest remain their previous value.

```
function [X,Y] = Update_LBP(LBP,LBE,Swarm,Evaluation,Ite,NP)
    for i=1:NP
        if LBE(i)>Evaluation(i,Ite)
            LBP(i,:)=Swarm(i,:);
            LBE(i)=Evaluation(i,Ite);
        end
    end
    X=LBP;
    Y=LBE;
end
```

Listing 9: Implementation of the mechanism used to constraint the position of the particles to the given boundaries coded on the file **Update_LBP.m**

3.2 Dynamical system simulation schemes

The implementation of the dynamical simulation schemes is used on the function EVALUATION_SWARM() as be described previously, its necessary that

a dynamical simulation of the system be implemented in order to generate the output y_m that is used to compute the performance function J.

Exists two main methods to implement the simulation from the dynamical system. The first one is manually coding the system of equations and implementing a numerical integration method. This case is coded on the file **Dynamical_simulation_ideal.m**, the code is given on the Listing 10 where a Euler numerical integration method is implemented. The Euler method is selected by their easy implementation and low computational cost, the user could replace this method with another integration method. However, this method must have a **fixed step** because is required that the output vector computed have the same length than the original data vector.

```
function Y = Dynamical_simulation_ideal(r,t,y0,theta)
    k=theta(1);
    z=theta(2);
    p1=theta(3);
    p2=theta(4);
    Cf=(k*p1*p2)/z;
    ts=t(2)-t(1);
    N=length(t);
    x1=zeros(N,1);
    x2=zeros(N,1);
    x1(1)=y0(1);
    x2(1)=y0(2);
    for i=2:N
        x1(i)=x1(i-1)+ts*x2(i-1);
        if i==2
            dr=(r(i-1)-0)/ts;
        else
            dr=(r(i-1)-r(i-2))/ts;
        end
        Din=-x2(i-1)*(p1+p2)-x1(i-1)*p1*p2+Cf*(-z*r(i-1)-dr);
        x2(i)=x2(i-1)+ts*Din;
    end
    Y=x1;
end
```

Listing 10: Simulation from a dynamic system using the Euler numerical integration method coded on the file **Dynamical_simulation_ideal.m**

The second method relies on SIMULINK to generate the output y_m, this method provides simplicity to the implementation but increases the computa-

tional burden since the environment of SIMULINK requires more resources than a numerical integration method manually coded. The implemented scheme is shown on Figure 1, must be noticed that corresponds to the mathematical model expressed previously. However, in this case is directly expressed as transfer function which implies this method facilitate the implementation of the dynamical system simulation since is not necessary code the integration method or convert some representations like a transfer function into state space.

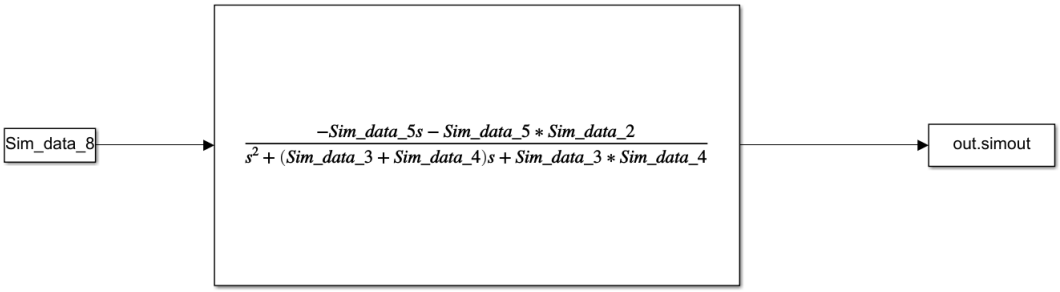


Figure 1: Simulink model implemented on SIMULINK, the model applied corresponds to a transfer function. The model is found on the file **Simulink_ideal_S.slx**.

The way this SIMULINK model is applied to the previous code is by the substitution of DYNAMICAL_SIMULATION_IDEAL() that simulates the model using Eulet method for the function DYNAMICAL_SIMULATION_SIMULINK() that is coded on **Dynamical_simulation_simulink.m**. The coding of this function is shown on Listing 11, on this could be noticed the parameters of the transfer function, the simulation maximum time, the sampling time and the reference signal merged with the time as a timeseries vector are save on a external file. These parameters and the signals are going to be loaded from the SIMULINK environment with a Initial Function Callback from the Model Explorer as be shown on Figure 2 in order to avoid problems related to the way the variables are obtained by SIMULINK. The settings for the simulation on the SIMULINK environment are defined is shown on the Figure 3, notice the sampling time, maximum simulation time are based on the data read from the file and the integration method is of fixed step. The model is simulated using the function SIM() that execute the SIMULINK model, the output must be processed until only the vector of values of y_m is given.

```
function Y = Dynamical_simulation_ideal(r,t,y0,theta)
    Sim_data_1=theta(1); %k
    Sim_data_2=theta(2); %z
    Sim_data_3=theta(3);%p1
    Sim_data_4=theta(4);%p2
    Sim_data_5=(Sim_data_1*Sim_data_3*Sim_data_4)/Sim_data_2;%Cf
    Sim_data_6=max(t);%T_max
    Sim_data_7=t(2)-t(1);%ts
    Sim_data_8=timeseries(r,t);%Ref
    save('Simu_data','Sim_data_1','Sim_data_2','Sim_data_3',...
    'Sim_data_4','Sim_data_5','Sim_data_6','Sim_data_7','Sim_data_8');
    S=sim('Simulation_ideal_S.slx');
    S_timeseries=S.simout;
    S_rela=squeeze(S_timeseries);
%% Definition of the output from the dynamical system
    Y=S_rela;
end
```

Listing 11: Simulation from a dynamic system using a model implemented on SIMULINK. The code is given on the file **Dynamical_simulation_simulink.m**

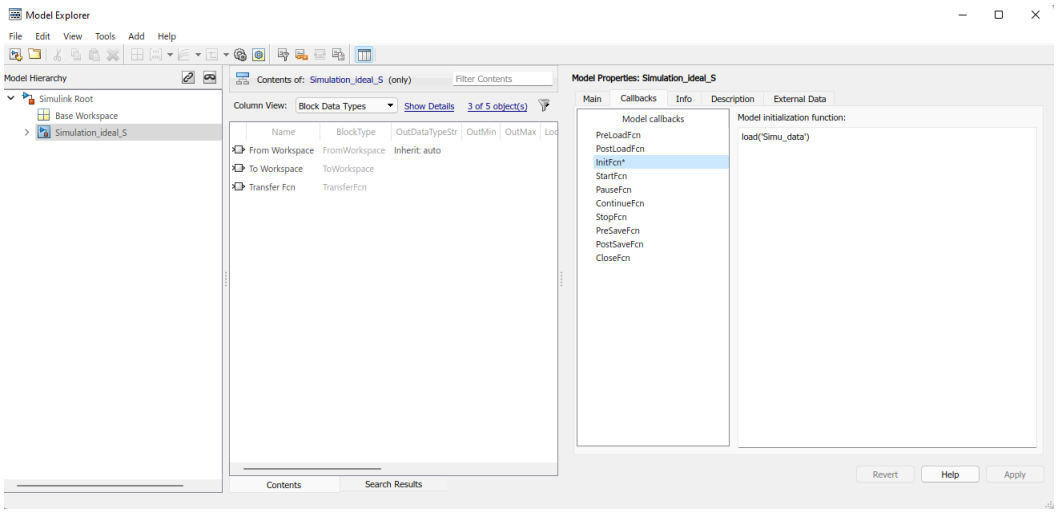


Figure 2: Read of the simulation parameters given by the PSO algorithm. The model is found on the file **Simulink_ideal_S.slx**.

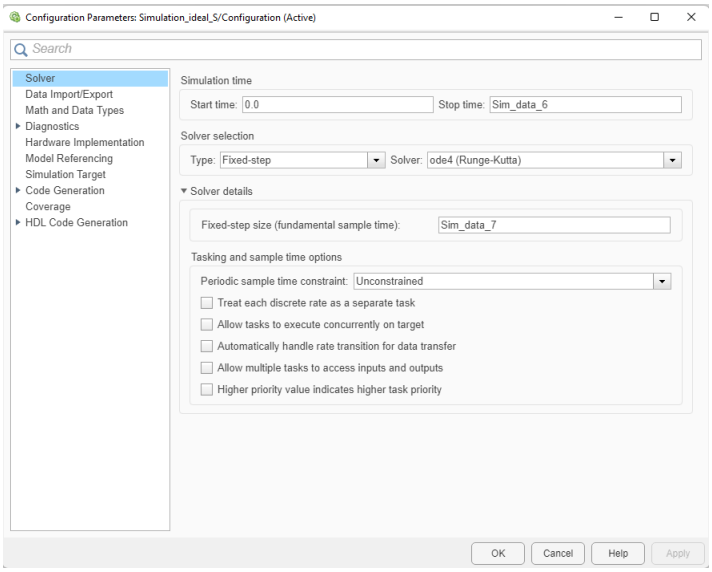


Figure 3: Setting of the simulation parameters based on the data given by the PSO algorithm. The model is found on the file **Simulink_ideal_S.slx**.

4 Spectral Richness PSO algorithm

The Spectral Richness PSO (SR-PSO) algorithm is an algorithm developed on XXXX. This algorithm is focused on decreasing the variability of the solutions computed by a classical PSO algorithm, this could be reached by the modification of the weights given to the gbest solution and the lbest solution on the computing of new positions for the particles. When the gbest weight *I* is increased the variability of the solutions is decreased, however, this implies the solutions could be trapped in local minimums.

Since the relationship between the weights tends to produce changes between the solutions, is desired that on the initial iterations of the algorithm the search of solutions on the space is performed, but when the iterations increase the gbest solution must be increased and the lbest solutions must be decreased to avoid the variability of the solutions. However, the relationship of the change between the weights could be defined based on the kind of excitation applied to the dynamical system measured with the Spectral Richness χ.

4.1 Estimation of Spectral Richness from a signal

The mathematical estimation of the Spectral Richness depends from the use of the Discrete Fast Fourier Transform to compute the spectral lines from the

reference signal that is applied to the dynamical system as excitation. The coding is performed using MATLAB as could be seen on Listing 12, is noticed that the process given on Subsection 2.3 is performed using the function FFT given by the environment. If the user desire to implement this part of the algorithm in other environment, must be checked that an equivalent functions is available or code their own version from the FFT algorithm.

```
function Y = Spectral_Richness(r,t)
ts=t(2)-t(1);
fs=1/ts;
%% Fast fourier transformation using the fft function
L=max(size(r));
Y = fft(r);
P2 = abs(Y/L);
P1 = P2(1:round(L/2+1));
P1(2:end-1) = 2*P1(2:end-1);
%% Normalization from the frequency components
P1=P1/max(P1);
%% Estimation of the threshold kappa
Z=(fs)/length(r);
Kap=1-pi*Z*cot(pi*Z);
%% Evaluation of spectral lines to compute chi
chi=0;
for i=1:length(P1)
    if P1(i)>Kap
        chi=chi+1;
    end
end
    Y=chi;
end
```

Listing 12: Compute from the Spectral Richness value χ implementing the threshold κ to discard spectral lines that corresponds to numerical errors from the FFT. The code is given on the file **Spectral_Richness.m**

4.2 Local and global weights for the particles

In order to modify the performance of the solution search from the PSO algorithm, a pair of dynamical terms based on the value of the Spectral Richness χ and the iteration number k is proposed. The main idea is to decrease the search to enhance the convergence of the parameters during the last iterations by the modification from the weights terms p and l. However,

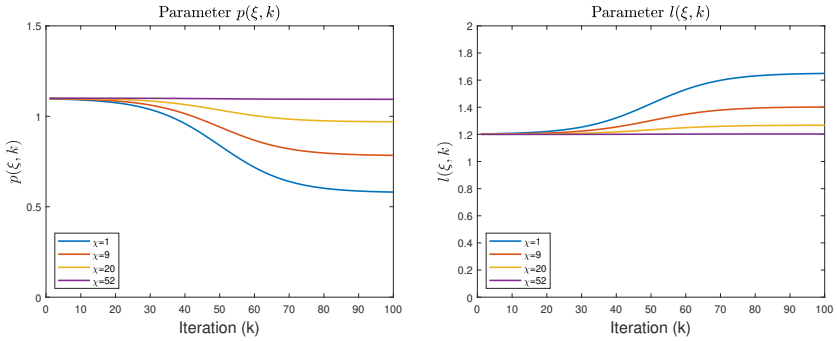
this is only required when the system has been excited with a signal that posses a low value of χ for this reason the decrease of the search is performed only on these cases.

The mathematical expression that describe the change of the weights is given by the following expressions:

$$p(\chi,k) = \bar{p} + \left(\frac{\bar{p}}{(1+e^{-\chi})}\right)\left(\frac{1}{1+e^{-w_\chi(k-0.5k_{\max})}}\right) + \frac{-\bar{p}p_c}{1+e^{-w_\chi(k-0.5k_{\max})}}. \tag{16}$$

$$I(\chi,k) = \frac{\bar{I}e^{-\chi}}{1+e^{-w_\chi(k-0.5k_{\max})}} + I_{\min}. \tag{17}$$

The way this terms evolves is shown on Figure 4, where the change of the weights change over the number of iterations. Four cases are shown, to make an example from the effect that χ generate on p and I that modifies the way the PSO algorithm search the solutions.



(a) Performance of the weight $p(\chi,k)$ as the number of iterations k increases.

(b) Performance of the weight $I(\chi,k)$ as the number of iterations k increases.

Figure 4: Evolution of the weights of the SR-PSO algorithm when the excitation signals applied to the dynamical system have different values of Spectral Richness χ.

The implementation from the term $p(\chi,k)$ is given on the Listing 13 and for the case of $I(\chi,k)$ is given on the Listing 14. For the case of $p(\chi,k)$ must be noticed that the terms that corresponds to fractions are separated in order to facilitate the user to understand the expression applied. The same is performed for the case of $I(\chi,k)$, on this case the expression is easily recognized.

```
function Y = P_SR(Pb,chi,omc,k,kmax,pc)
    N1=Pb;
    D1=1+exp(-chi);
    F1=N1/D1;
    N2=1;
    D2=1+exp(-omc*(k-0.5*kmax));
    F2=N2/D2;
    N3=-Pb*pc;
    D3=1+exp(-omc*(k-0.5*kmax));
    F3=N3/D3;
    Y=Pb+(F1*F2)+F3;
end
```

Listing 13: Implementation of the compute of $p(k, \chi)$ for the SR-PSO algorithm. The code is given on the file **P_SR.m**

```
function Y = L_SR(Lb,chi,omc,k,kmax,Lmin)
    N=Lb*exp(-chi);
    D=1+exp(-omc*(k-0.5*kmax));
    F=N/D;
    Y=Lmin+F;
end
```

Listing 14: Implementation of the compute of $l(k, \chi)$ for the SR-PSO algorithm. The code is given on the file **L_SR.m**