function [ff] = EnvelopeCost(c,data)

X = data;

% alpha = 2300; % moderate bandwidth constraint：适度的带宽约束/惩罚因子

alpha = fix(c(1)); % moderate bandwidth constraint：适度的带宽约束/惩罚因子

tau = 0; % noise-tolerance (no strict fidelity enforcement)：噪声容限（没有严格的保真度执行）

K = fix(c(2)); % modes：分解的模态数

% K = 10; % modes：分解的模态数

DC = 0; % no DC part imposed：无直流部分

init = 1; % initialize omegas uniformly ：omegas的均匀初始化

tol = 1e-7;

%--------------- Run actual VMD code:数据进行vmd分解---------------------------

[u, u\_hat, omega] = VMD(X, alpha, tau, K, DC, init, tol);

for i = 1:K

xx= abs(hilbert(u(i,:))); %最小包络熵计算公式！

xxx = xx/sum(xx);

ssum=0;

for ii = 1:size(xxx,2)

bb = xxx(1,ii)\*log(xxx(1,ii));

ssum=ssum+bb;

end

fitness(i,:) = -ssum;

end

[ff] = min(fitness);

end

% Particle Swarm Optimization

function [gBestScore,gBest,cg\_curve]=PSO(noP,Max\_iter,lb,ub,dim,fobj,da)

%PSO Infotmation

Vmax=ones(1,dim).\*(ub-lb).\*0.15; %速度最大值

w=0.8;

c1=1.2;

c2=1.2;

% Initializations

iter=Max\_iter;

vel=zeros(noP,dim);

pBestScore=zeros(noP);

pBest=zeros(noP,dim);

cg\_curve=zeros(1,iter);

% Random initialization for agents.

pos=initialization(noP,dim,ub,lb);

for i=1:noP

pBestScore(i)=inf;

end

% Initialize gBestScore for a minimization problem

gBestScore=inf;

for l=1:iter

% Return back the particles that go beyond the boundaries of the search

% space

for i=1:size(pos,1)

Flag4ub=pos(i,:)>ub;

Flag4lb=pos(i,:)<lb;

pos(i,:)=(pos(i,:).\*(~(Flag4ub+Flag4lb)))+ub.\*Flag4ub+lb.\*Flag4lb;

end

for i=1:size(pos,1)

%Calculate objective function for each particle

fitness= fobj(pos(i,:)',da);

if(pBestScore(i)>fitness)

pBestScore(i)=fitness;

pBest(i,:)=pos(i,:);

end

if(gBestScore>fitness)

gBestScore=fitness;

gBest=pos(i,:);

end

end

%Update the W of PSO

%Update the Velocity and Position of particles

for i=1:size(pos,1)

for j=1:size(pos,2)

vel(i,j)=w\*vel(i,j)+c1\*rand()\*(pBest(i,j)-pos(i,j))+c2\*rand()\*(gBest(j)-pos(i,j));

if(vel(i,j)>Vmax(1,j))

vel(i,j)=Vmax(1,j);

end

if(vel(i,j)<-Vmax(1,j))

vel(i,j)=-Vmax(1,j);

end

pos(i,j)=pos(i,j)+vel(i,j);

end

end

cg\_curve(l)=gBestScore;

disp(['PSO: At iteration ', num2str(l), ' ,the best fitness is ', num2str(gBestScore)])

disp(['第',num2str(l),'次寻优的最佳位置为：[',num2str(fix(gBest)),']'])

end

end

%% 淘个代码 %%

% 2023/06/15 %

%微信公众号搜索：淘个代码，获取更多免费代码

%%

function [ff] = SampleCost(c,data)

X = data;

% alpha = 2300; % moderate bandwidth constraint：适度的带宽约束/惩罚因子

alpha = fix(c(1)); % moderate bandwidth constraint：适度的带宽约束/惩罚因子

tau = 0; % noise-tolerance (no strict fidelity enforcement)：噪声容限（没有严格的保真度执行）

K = fix(c(2)); % modes：分解的模态数

% K = 10; % modes：分解的模态数

DC = 0; % no DC part imposed：无直流部分

init = 1; % initialize omegas uniformly ：omegas的均匀初始化

tol = 1e-7;

%--------------- Run actual VMD code:数据进行vmd分解---------------------------

[u, u\_hat, omega] = VMD(X, alpha, tau, K, DC, init, tol);

dim = 2; % dim：嵌入维数(一般取1或者2)

tau = 1; %下采样延迟时间（在默认值为1的情况下，用户可以忽略此项）

for i = 1:K

x=u(i,:);%

r = 0.2\*std(x); % r：相似容限( 通常取0.1\*Std(data)~0.25\*Std(data) )

fitness(i,:) = SampleEntropy( dim, r, x, tau );

end

[ff] = min(fitness);

end

%% 淘个代码 %%

% 2023/06/25 %

%微信公众号搜索：淘个代码，获取更多免费代码

%%

%% 样本熵函数

function sampEn = SampleEntropy( dim, r, data, tau )

% 注意：这个样本熵函数是在Kijoon Lee的基础上做的修改

% 样本熵算法的提出者：Richman J s，Moorman J R. Physiological time-seriesanalysis using approximate entropy and sample entropy[J. American Journal of Physiology Heart &. Circula-tory Physiology，2000，278(6):2039-2049.

% 计算给定时间序列数据的样本熵

% 样本熵在概念上类似于近似熵，但有以下区别：

% 1）样本熵不计算自匹配，通过在最后一步取对数，避免了可能出现的log(0)问题；

% 2）样本熵不像近似熵那样依赖数据的长度。

% dim：嵌入维数(一般取1或者2)

% r：相似容限( 通常取0.1\*Std(data)~0.25\*Std(data) )

% data：时间序列数据，data须为1xN的矩阵

% tau：下采样延迟时间（在默认值为1的情况下，用户可以忽略此项）

if nargin < 4, tau = 1; end

if tau > 1, data = downsample(data, tau); end

N = length(data);

result = zeros(1,2);

for m = dim:dim+1

Bi = zeros(N-m+1,1);

dataMat = zeros(N-m+1,m);

% 设置数据矩阵，构造成m维的矢量

for i = 1:N-m+1

dataMat(i,:) = data(1,i:i+m-1);

end

% 利用距离计算相似模式数

for j = 1:N-m+1

% 计算切比雪夫距离，不包括自匹配情况

dist = max(abs(dataMat - repmat(dataMat(j,:),N-m+1,1)),[],2);

% 统计dist小于等于r的数目

D = (dist <= r);

% 不包括自匹配情况

Bi(j,1) = (sum(D)-1)/(N-m);

end

% 求所有Bi的均值

result(m-dim+1) = sum(Bi)/(N-m+1);

end

% 计算得到的样本熵值

sampEn = -log(result(2)/result(1));

end

%% 淘个代码 %%

% 2023/06/12 %

%微信公众号搜索：淘个代码，获取更多免费代码

%%

clc

clear

strr = [pwd,'\\*.m'];

namelist = dir(strr);

len = length(namelist);

if exist([pwd,'\newfile\'])~=0

rmdir([pwd,'\newfile\'],'s') %创建新文件所在的文件夹

end

if exist([pwd,'\newfile\'])==0

mkdir([pwd,'\newfile\']) %创建新文件所在的文件夹

end

newpath = [pwd,'\newfile\'];

fid=fopen([newpath,'newtxt.txt'],'w');

for i = 1:len

file\_name{i}=namelist(i).name;

fpn= fileread(file\_name{i}); %打开文档

cc = [newpath,file\_name{i}];

fprintf(fid,'%s\n',fpn);

end

fclose(fid);

%% 淘个代码 %%

function [u, u\_hat, omega] = VMD(signal, alpha, tau, K, DC, init, tol)

% Variational Mode Decomposition

% Authors: Konstantin Dragomiretskiy and Dominique Zosso

% zosso@math.ucla.edu --- http://www.math.ucla.edu/~zosso

% Initial release 2013-12-12 (c) 2013

%

% Input and Parameters:

% ---------------------

% signal - the time domain signal (1D) to be decomposed

% alpha - the balancing parameter of the data-fidelity constraint

% tau - time-step of the dual ascent ( pick 0 for noise-slack )

% K - the number of modes to be recovered

% DC - true if the first mode is put and kept at DC (0-freq)

% init - 0 = all omegas start at 0

% 1 = all omegas start uniformly distributed

% 2 = all omegas initialized randomly

% tol - tolerance of convergence criterion; typically around 1e-6

%

% Output:

% -------

% u - the collection of decomposed modes

% u\_hat - spectra of the modes

% omega - estimated mode center-frequencies

%

% When using this code, please do cite our paper:

% -----------------------------------------------

% K. Dragomiretskiy, D. Zosso, Variational Mode Decomposition, IEEE Trans.

% on Signal Processing (in press)

% please check here for update reference:

% http://dx.doi.org/10.1109/TSP.2013.2288675

%---------- Preparations

% Period and sampling frequency of input signal

save\_T = length(signal);

fs = 1/save\_T;

% extend the signal by mirroring

T = save\_T;

f\_mirror(1:T/2) = signal(T/2:-1:1);

f\_mirror(T/2+1:3\*T/2) = signal;

f\_mirror(3\*T/2+1:2\*T) = signal(T:-1:T/2+1);

f = f\_mirror;

% Time Domain 0 to T (of mirrored signal)

T = length(f);

t = (1:T)/T;

% Spectral Domain discretization

freqs = t-0.5-1/T;

% Maximum number of iterations (if not converged yet, then it won't anyway)

N = 500;

% For future generalizations: individual alpha for each mode

Alpha = alpha\*ones(1,K);

% Construct and center f\_hat

f\_hat = fftshift((fft(f)));

f\_hat\_plus = f\_hat;

f\_hat\_plus(1:T/2) = 0;

% matrix keeping track of every iterant // could be discarded for mem

u\_hat\_plus = zeros(N, length(freqs), K);

% Initialization of omega\_k

omega\_plus = zeros(N, K);

switch init

case 1

for i = 1:K

omega\_plus(1,i) = (0.5/K)\*(i-1);

end

case 2

omega\_plus(1,:) = sort(exp(log(fs) + (log(0.5)-log(fs))\*rand(1,K)));

otherwise

omega\_plus(1,:) = 0;

end

% if DC mode imposed, set its omega to 0

if DC

omega\_plus(1,1) = 0;

end

% start with empty dual variables

lambda\_hat = zeros(N, length(freqs));

% other inits

uDiff = tol+eps; % update step

n = 1; % loop counter

sum\_uk = 0; % accumulator

% ----------- Main loop for iterative updates

while ( uDiff > tol && n < N ) % not converged and below iterations limit

% update first mode accumulator

k = 1;

sum\_uk = u\_hat\_plus(n,:,K) + sum\_uk - u\_hat\_plus(n,:,1);

% update spectrum of first mode through Wiener filter of residuals

u\_hat\_plus(n+1,:,k) = (f\_hat\_plus - sum\_uk - lambda\_hat(n,:)/2)./(1+Alpha(1,k)\*(freqs - omega\_plus(n,k)).^2);

% update first omega if not held at 0

if ~DC

omega\_plus(n+1,k) = (freqs(T/2+1:T)\*(abs(u\_hat\_plus(n+1, T/2+1:T, k)).^2)')/sum(abs(u\_hat\_plus(n+1,T/2+1:T,k)).^2);

end

% update of any other mode

for k=2:K

% accumulator

sum\_uk = u\_hat\_plus(n+1,:,k-1) + sum\_uk - u\_hat\_plus(n,:,k);

% mode spectrum

u\_hat\_plus(n+1,:,k) = (f\_hat\_plus - sum\_uk - lambda\_hat(n,:)/2)./(1+Alpha(1,k)\*(freqs - omega\_plus(n,k)).^2);

% center frequencies

omega\_plus(n+1,k) = (freqs(T/2+1:T)\*(abs(u\_hat\_plus(n+1, T/2+1:T, k)).^2)')/sum(abs(u\_hat\_plus(n+1,T/2+1:T,k)).^2);

end

% Dual ascent

lambda\_hat(n+1,:) = lambda\_hat(n,:) + tau\*(sum(u\_hat\_plus(n+1,:,:),3) - f\_hat\_plus);

% loop counter

n = n+1;

% converged yet?

uDiff = eps;

for i=1:K

uDiff = uDiff + 1/T\*(u\_hat\_plus(n,:,i)-u\_hat\_plus(n-1,:,i))\*conj((u\_hat\_plus(n,:,i)-u\_hat\_plus(n-1,:,i)))';

end

uDiff = abs(uDiff);

end

%------ Postprocessing and cleanup

% discard empty space if converged early

N = min(N,n);

omega = omega\_plus(1:N,:);

% Signal reconstruction

u\_hat = zeros(T, K);

u\_hat((T/2+1):T,:) = squeeze(u\_hat\_plus(N,(T/2+1):T,:));

u\_hat((T/2+1):-1:2,:) = squeeze(conj(u\_hat\_plus(N,(T/2+1):T,:)));

u\_hat(1,:) = conj(u\_hat(end,:));

u = zeros(K,length(t));

for k = 1:K

u(k,:)=real(ifft(ifftshift(u\_hat(:,k))));

end

% remove mirror part

u = u(:,T/4+1:3\*T/4);

% recompute spectrum

clear u\_hat;

for k = 1:K

u\_hat(:,k)=fftshift(fft(u(k,:)))';

end

end

%% 淘个代码 %%

% 2023/06/25 %

%微信公众号搜索：淘个代码，获取更多免费代码

%%

%\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_%

% Grey Wold Optimizer (GWO) source codes version 1.0 %

% %

% Developed in MATLAB R2011b(7.13) %

% %

% Author and programmer: Seyedali Mirjalili %

% %

% e-Mail: ali.mirjalili@gmail.com %

% seyedali.mirjalili@griffithuni.edu.au %

% %

% Homepage: http://www.alimirjalili.com %

% %

% Main paper: S. Mirjalili, S. M. Mirjalili, A. Lewis %

% Grey Wolf Optimizer, Advances in Engineering %

% Software , in press, %

% DOI: 10.1016/j.advengsoft.2013.12.007 %

% %

%\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_%

% This function initialize the first population of search agents

function Positions=initialization(SearchAgents\_no,dim,ub,lb)

Boundary\_no= size(ub,2); % numnber of boundaries

% If the boundaries of all variables are equal and user enter a signle

% number for both ub and lb

if Boundary\_no==1

Positions=rand(SearchAgents\_no,dim).\*(ub-lb)+lb;

end

% If each variable has a different lb and ub

if Boundary\_no>1

for i=1:dim

ub\_i=ub(i);

lb\_i=lb(i);

Positions(:,i)=rand(SearchAgents\_no,1).\*(ub\_i-lb\_i)+lb\_i;

end

end

%% 淘个代码 %%

% 2023/06/25 %

%微信公众号搜索：淘个代码，获取更多免费代码

%% 以最小包络熵或最小样本熵为目标函数，采用PSO算法优化VMD，求取VMD最佳的两个参数

clear

clc

addpath(genpath(pwd))

load 105.mat

D=2; % 优化变量数目

lb = [100 3];

ub = [2500 10];

dim = 2;

Max\_iter=40; % 最大迭代数目

SearchAgents\_no=15; % 种群规模

xz = 1; %xz=1 or 2, 选择1，以最小包络熵为适应度函数，选择2，以最小样本熵为适应度函数。

if xz == 1

fobj=@EnvelopeCost;

else

fobj=@SampleCost;

end

da = X105\_DE\_time(6001:7000); %这里选取105的DEtime数据

[gBestScore,gBest,cg\_curve]=PSO(SearchAgents\_no,Max\_iter,lb,ub,dim,fobj,da);

%画适应度函数图

figure

plot(cg\_curve,'Color',[0.9 0.5 0.1],'Marker','>','LineStyle','--','linewidth',1);

title('Objective space')

xlabel('Iteration');

ylabel('Best score obtained so far');

legend('PSO优化VMD')

display(['The best solution obtained by PSO is : ', num2str(round(gBest))]); %输出最佳位置

display(['The best optimal value of the objective funciton found by pso is : ', num2str(gBestScore)]); %输出最佳适应度值