Apêndice

A.1 INFORMAÇÕES BÁSICAS

Uma vez que a otimização é uma ferramenta amplamente utilizada na aplicação da filosofia das redes inteligentes, este apêndice traz programações, em MatLab, dos seguintes algoritmos de otimização:

- Colônia de formigas.
- Recozimento simulado (*Simulated annealing*).
- Algoritmo genético.
- Enxame de partículas.

A intenção é mostrar para o leitor como os procedimentos definidos para cada otimização podem ser programados. É claro que, para problemas mais complexos, as soluções apresentadas aqui não poderão ser utilizadas diretamente. Porém, o que se deseja é que o leitor entenda os mecanismos por trás de cada ferramenta.

A escolha por utilizar o MatLab se dá pelo fato de tal ferramenta ter uma linguagem de programação mais simples e *solvers* de algoritmos de otimização, que podem ser utilizados sem que o usuário tenha que escrever toda a programação ou ainda entender de maneira aprofundada essas técnicas. Quando os nomes das funções terminarem com a letra "f", indica que a otimização foi feita pelo *solver* do MatLab.

É importante que todos os arquivos em que as programações forem salvas estejam na mesma pasta do computador utilizado.

O algoritmo cujo nome é "toma" deve, obrigatoriamente, ser salvo na mesma pasta que todos os outros arquivos. Alguns dos algoritmos chamam a função contida dentro do citado.

Não faz parte desta edição do livro a explicação dos conceitos e procedimentos dos algoritmos de otimização apresentados aqui. Dessa forma, recomenda-se que o leitor procure outras referências para entender as ideias que levaram ao desenvolvimento dos algoritmos.

As programações apresentadas neste apêndice foram feitas para minimizar uma função contínua. Todas elas foram comentadas para que o leitor possa tentar fazer a relação entre os procedimentos teóricos das ferramentas e sua tradução para a linguagem de programação definida.

A.2 FUNÇÃO CUSTO A SER OTIMIZADA

A função que os algoritmos otimizam neste apêndice é apresentada no *survey* publicado por Tomassini (1995).

No estudo, o autor investiga a eficiência do algoritmo genético para a minimização de uma função. Logo na página 7 do documento, o autor traz a seguinte ilustração do gráfico da função (Figura A.1):

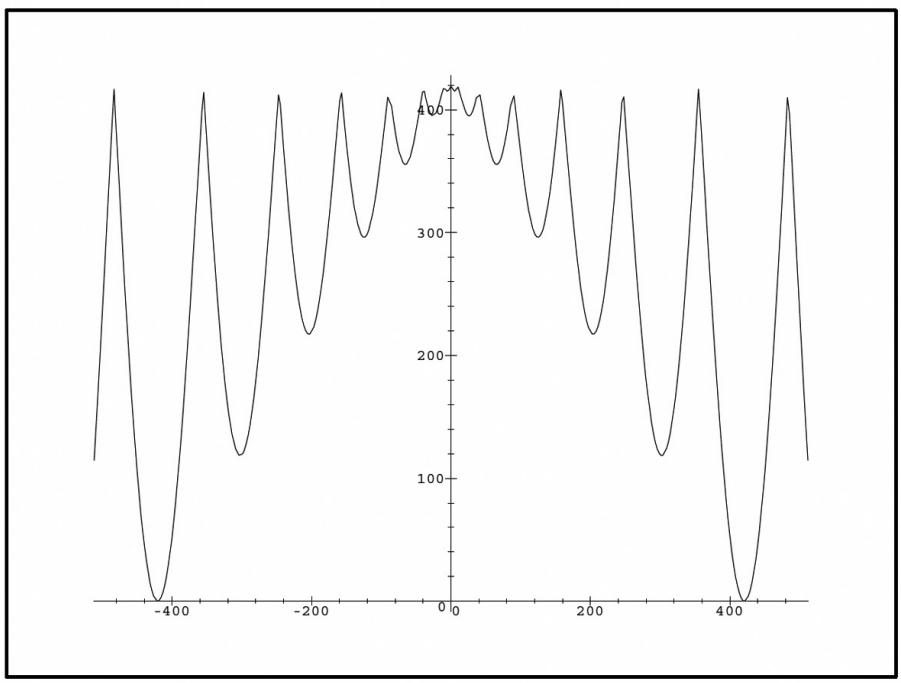

Figura A.1 Gráfico da função custo.
Fonte: Tomassini, 1995.

O documento afirma que o gráfico apresentado na Figura A.1 corresponde à seguinte função matemática (A.1):

$$f(x) = -\left| x\, sin(\sqrt{|x|}) \right| \tag{A.1}$$

O problema, então, é encontrar o valor de "x" que minimiza a função (A.1) para o intervalo de busca [0-512]. Uma vez que a função $f(x)$ é simétrica, não há a necessidade de analisar o intervalo [-512, 0].

No entanto, quando se tenta reproduzir o gráfico da figura A.1 por meio da função descrita pela equação (A.1), não se tem o mesmo resultado. Isso pode ser comprovado pelo algoritmo disponibilizado aqui, cujo nome é "funcao". Esse o motivo pelo qual esse trabalho foi escolhido.

Muitas vezes, quando estamos começando a aprender uma técnica de otimização, testamos a programação com algum estudo já publicado para verificar se o resultado sai como o esperado. Como demonstrado, isso pode ser muito perigoso se não houver atenção máxima para esses detalhes.

A função correta é (A.2):

$$f(x) = -\left| x \sin(\sqrt{|x|}) \right| - 418{,}9829 \tag{A.2}$$

Outro ponto importante a se avaliar é que os algoritmos de otimização muitas vezes não encontram a solução ótima, mas sim a otimizada, ou seja, próxima da ótima. Dessa maneira, em algoritmos como o de recozimento simulado, os valores das soluções encontradas variarão entre simulações.

Recomenda-se que os leitores, após pesquisarem sobre os algoritmos de otimização, testem cada um deles, alterando suas variáveis iniciais para verificar quais afetam o desempenho de cada um.

```
% Nome - funcao
% Desenvolvido por Dr. Raul Vitor Arantes Monteiro / Dezembro de 2023
% Grupo de Pesquisa em Operação de Sistemas Elétricos e Redes Inteligentes
% Departamento de Engenharia Elétrica
% Universidade Federal de Mato Grosso
% Função custo: -(abs(x.*sin(sqrt(abs(x)))))-418.9829

%Verifica a função do artigo Tomassini

function funcao

x = 0:1:512;
n = 1:1:length(x);

%func2 testa a função conforme dada no artigo
%func2 = -(abs(x.*sin(sqrt(abs(x)))));   %descomentar para testar

%func testa a função corrigida do artigo
%func = -(abs(x.*sin(sqrt(abs(x)))))-418.9829;   %descomentar para testar

%plot(x,func2)   %plota a função conforme dada no artigo. Descomentar para usar com func2

%plot(x,func)   %plota a função corrigida. Descomentar para usar com func

end
```

```
% Nome - toma
% Função custo
% Desenvolvido por Dr. Raul Vitor Arantes Monteiro / Dezembro de 2023
% Grupo de Pesquisa em Operação de Sistemas Elétricos e Redes Inteligentes
% Departamento de Engenharia Elétrica
% Universidade Federal de Mato Grosso

function f = Toma(x)

f = -(abs(x.*sin(sqrt(abs(x)))))-418.9829;
end
```

```
% Nome – ACO1
% Copyright (c) 2015, Yarpiz (www.yarpiz.com)
% All rights reserved. Please read the “license.txt” for license terms.
% Project Code: YPEA104
% Project Title: Ant Colony Optimization for Continuous Domains (ACOR)
% Publisher: Yarpiz (www.yarpiz.com)
% Developer: S. Mostapha Kalami Heris (Member of Yarpiz Team)
% Contact Info: sm.kalami@gmail.com, info@yarpiz.com
% Adaptado por Raul Vitor Arantes Monteiro

function ACO1
clc;
clear;
close all;
%% Definição do problema
CostFunction=@(x) toma(x);       % Função custo
nVar=1;             % Número de variáveis de decisão
VarSize=nVar;   % Tamanho da matriz de variáveis
VarMin=0;        % Limite inferior do espaço de busca
VarMax=512;     % Limite superior do espaço de busca

%% Parâmetros do ACOR
MaxIt=50;         % Máximo número de iterações
nPop=200;          % Tamanho da população
nSample=20;       % Tamanho da população extra
q=0.3;                % Fator de intensificação (seleção da pressão)
zeta=0.1;             % Taxa de desvio-distância
%% Inicialização

% Cria estrutura de indivíduos vazia
empty_individual.Position=[];
empty_individual.Cost=[];

% Cria matriz da população
pop=repmat(empty_individual,nPop,1);

% Inicialização dos membros da população
for i=1:1:nPop
```

```
    % Cria soluções aleatórias
    pop(i).Position= (512)*rand;

    % Avalia
    pop(i).Cost=CostFunction(pop(i).Position);

end

% Organiza população em ordem crescente
[~, SortOrder]=sort([pop.Cost]);
pop=pop(SortOrder);

% Atualiza a melhor solução encontrada na população inicial
BestSol=pop(1);

% Arranjo para guardar o melhor custo
BestCost=zeros(MaxIt,1);

% Pesos das soluções
w=1/(sqrt(2*pi)*q*nPop)*exp(-0.5*(((1:nPop)-1)/(q*nPop)).^2);

% Seleção de probabilidades
p=(w/sum(w));

%% ACOR Loop principal
for it=1:MaxIt

    % Média
    s=zeros(nPop,nVar);
    for l=1:nPop
        s(l,:)=pop(l).Position;
        %s1(1,:)=pop(1).Position;
    end

    % Desvios padrão
    sigma=zeros(nPop,nVar);
    for l=1:nPop
```

```
    D=0;
   for rr=1:nPop
       D=D+abs(s(l,:)-s(rr,:));
    end
    sigma(l,:)=zeta*D/(nPop-1);
end

% Cria arranjo para a nova população
newpop=repmat(empty_individual,nSample,1);

for t=1:nSample

    % Inicializa a matriz de posições da nova população
    newpop(t).Position=zeros(VarSize);
    s_auxi=zeros(nPop,nVar);

    % Construção da solução
    for i=1:nVar

        % Seleção de acordo com o kernel Gaussiano
        l=RouletteWheelSelection(p);
        s_auxi(l,i)=s(l,i)+sigma(l,i)*(rand*3);

        %Verifica se não extrapolou os limites máximo e mínimo e corrige
        if s_auxi(l,i)>VarMax
            s_auxi(l,i)=s_auxi(l,i)-VarMax;
        elseif s_auxi(l,i)<-VarMax
            s_auxi(l,i)=s_auxi(l,i)+VarMax;
        else
        end

        % Gera variáveis Gaussianas aleatórias
        newpop(t).Position(i)=s_auxi(l,i);
    end

    % Avaliação
    newpop(t).Cost=CostFunction(newpop(t).Position);
```

```
    end

    % Mescla a população principal gerada com a nova população gerada
    pop=[pop
        newpop]; %#ok

    % Organiza a população em ordem crescente de acordo com as melhores soluções
    [~, SortOrder]=sort([pop.Cost]);
    pop=pop(SortOrder);

    % Deleta membros extras
    pop=pop(1:nPop);
    popauxi(:,it)=pop.Position;

    % Atualiza a melhor solução encontrada
    BestSol=pop(1);

    % Armazena a melhor solução
    BestCost(it)=BestSol.Cost;
    BestPos(it)=BestSol.Position;

    % Mostra as principais informações de acordo com a iteração
    disp(['Iteração ' num2str(it) ': Melhor custo = ' num2str(abs(BestCost(it))) ': Melhor solução =
' num2str(BestPos(it))]);

end

%% Results
figure;
plot(BestCost,'LineWidth',2);
%semilogy(BestCost,'LineWidth',2);
xlabel('Iteration');
ylabel('Best Cost');
grid on;

function j=RouletteWheelSelection(P)
    r=rand;
    C=cumsum(P);
```

```
   j=find(r<=C,1,'first');
end

end

% Nome - sa
% Algoritmo de otimização por recozimento simulado (Simulated Annealing) para funções contínuas
% Desenvolvido por Raul Vitor Arantes Monteiro
% Grupo de Pesquisa em Operação de Sistemas Elétricos e Redes Inteligentes
% Departamento de Engenharia Elétrica
% Universidade Federal de Mato Grosso

function sa

%define temperatura inicial
t=100;

%define a taxa resfriamento de temperatura
decait=0.9;

%determina o número de iterações para cada valor de temperatura
k=500;

%gera as possíveis soluções entre 0 e 512
xsoli=rand(k,1)*512;

%gera soluções vizinhas
xviz=xsoli(:,:)*1.1;
for i=1:k
   if xviz(i,:)>k
   xviz(i,:)=xviz(i,:)-k;
   elseif xviz(i,:)<-k
      xviz(i,:)=xviz(i,:)+k;
   else
   end
end
```

```
%escolhe solução inicial aleatoriamente
posxsoli=ceil(rand(1,1)*k);
xatual=xsoli(posxsoli,1);

iter=1;
j=k;

while t>1e-10

   for ii=1:j
   %escolhe vizinho aleatório

   xviz1(1,1)=zeros;
   posxviz=ceil(rand(1,1)*k);
   xviz1(1,1)=xviz(posxviz,1);

   %cálculo da função
   fatual=-((abs(xatual.*sin(sqrt(abs(xatual)))))-418.9829);
   fviz=-((abs(xviz1.*sin(sqrt(abs(xviz1)))))-418.9829);

   %cálculo de delta
   delta=fviz-fatual;
   xatual;
   %testa as probabilidades
   if delta<=0
      xatual=xviz1;
   end
   probab=exp(-delta/t);
   teste=rand(1,1);
   if delta>0 && probab>teste
      xatual=xviz1;
   end

   end

ffinal(iter)=fatual;
xfinal(iter)=xatual;
t=t*decait;
```

```
disp(['iteração: ' num2str(iter) ': Melhor custo: ' num2str(abs(ffinal(iter))) ': Melhor solução: '
num2str(xfinal(iter))]);

iter=iter+1;

end

figure(1)
lgt=size(ffinal,2);
plot(1:1:lgt,ffinal,'LineWidth',2);
xlabel('Iteração');
ylabel('Melhor custo');
grid on;

end
```

```
% Nome - saf
% Algoritmo de otimização por recozimento simulado (Simulated Annealing) para funções contínuas
% Desenvolvido por Raul Vitor Arantes Monteiro
% Grupo de Pesquisa em Operação de Sistemas Elétricos e Redes Inteligentes
% Departamento de Engenharia Elétrica
% Universidade Federal de Mato Grosso

function saf
clc;
clear;

rng default
FitnessFcn=@toma;
iniTemp = 500;
nIters = 100;
lb=0;
ub=512;
k = 100;
x0 = rand(1,1)*512;
options = optimoptions('simulannealbnd',...
                'AcceptanceFcn',{@acceptancesa},...
                'InitialTemperature',iniTemp,...
                'TemperatureFcn',{@temperaturefast},...
                'AnnealingFcn',{@annealingfast},...
                'ReannealInterval',10,...
                'MaxFunctionEvaluations', k,...
                'MaxIterations', nIters,...
                'MaxFunctionEvaluations', nIters,...
                'MaxStallIterations',nIters,...
                'PlotFcn',{@saplot, @saplotbestf, @saplotbestx, @saplotf},...
                'Display','iter',...
                'DisplayInterval', 1);

[Xbest,Gbest,~,~] = simulannealbnd(FitnessFcn,x0,lb,ub,options);

disp('============================================================')
```

```
disp(['Melhor Custo Global: ' num2str(floor(Gbest)) ' | Melhor Posição Global: ' num2s-
tr(Xbest)]);
disp('============================================================')

end

function stop = saplot(~,optimvalues,~)

stop = false;
  X=(optimvalues.bestx);
  Y=-(optimvalues.bestx);
  A=meshgrid(X,Y);
  plot(A',optimvalues.bestfval,'bx');
  xlabel('Melhores Soluções');
  ylabel('Custo');
  title('Espaço de busca');
  axis([0 550 -100 550]);
  pause(.1);


end
```

```
% Nome - pso
% Algoritmo de otimização por enxame de partículas para funções contínuas
% Desenvolvido por Dr. Raul Vitor Arantes Monteiro / Dezembro de 2023
% Grupo de Pesquisa em Operação de Sistemas Elétricos e Redes Inteligentes
% Departamento de Engenharia Elétrica
% Universidade Federal de Mato Grosso
% Função custo: -(abs((x(j,1+ite).*sin(sqrt(abs(x(j,1+ite)))))))-418.9829

function pso

tic; %começa a contar o tempo
%define espaço de busca
swarm_size=30;
inertia=0.1;
%define número máximo de iterações
max_ite=30;
%define velocidade inicial
v=ones(swarm_size,1);
%espaço de busca
sc_s=512;
%coeficientes de aceleração
c1=2;
c2=2;
%posições iniciais
x =(512)*rand(swarm_size,1);
size(x);
ite=0;
pb=0;
%cria variáveis vazias
pbest=zeros(length(swarm_size));
gbest=zeros(length(swarm_size));
gbest1=zeros(length(swarm_size));
func=zeros(length(swarm_size));
fun=zeros(length(swarm_size));
%%
%Início do algoritmo
```

```
while ite < max_ite
   for j=1:1:swarm_size
      fun(j,1+ite)= -(abs((x(j,1+ite).*sin(sqrt(abs(x(j,1+ite)))))))-418.9829 %função a ser otimiza-
da
%=============================================================
%verifica a melhor posição entre a solução anterior e a atual de cada partícula
%=============================================================
      if ite~=0 && (fun(j,1+ite)>fun(j,ite))
         x(j,1+ite)=x(j,ite);
         fun(j,1+ite)=fun(j,ite);
       else
         x(j,1+ite)=x(j,1+ite);
         fun(j,1+ite)=fun(j,1+ite);
       end
       if ite == 0
         pbest(j,1)=x(j,1);
       else
         p=[fun(j,ite),fun(j,1+ite)];
         [~,pb]=min(p);
       end
       if pb == 2 && ite ~= 0
         pbest(j,1)=x(j,1+ite);
       elseif pb == 1 && ite ~=0
         pbest(j,1)=x(j,ite);
       end
   end
%=============================================================
%verifica a melhor posição global entre todas as partículas
%=============================================================
   if ite==0
      [~,g]=min(fun(:,ite+1));
      gbest=x(g(1,1),1+ite);
   end
   if ite~=0
      g=[fun(:,ite+1),fun(:,ite)];
      [~,gb]=min(g);
      xval=[x(:,ite+1),x(:,ite)];
      gbest_=xval(gb,1);
```

```
    gbest=min(gbest_);
  end

  x_i(:,1)=x(:,ite+1);

  for j=1:1:swarm_size
    rand1=rand*c1; %define um valor aleatório do coeficiente de aceleração
    rand2=rand*c2; %define um valor aleatório do coeficiente de aceleração
    vk(j,1)=(inertia*v(j,1+ite))+((rand1)*(pbest(j,1)-x_i(j,1)))+((rand2)*(gbest-x_i(j,1))); %Ca-
cula a velocidade de cada partícula
    xk(j,1)=vk(j,1)+x_i(j,1); %movimenta a partícula no espaço de busca

    if xk(j,1)<0 || xk(j,1)>512 %restringe movimento da partícula dentro do espaço de busca [0
512]
      xk(j,1)=x_i(j,1);
    end
  end

  ite=ite+1;
  inertia=inertia*(1/ite);
  x(:,1+ite)=xk(:,1);
  v(:,1+ite)=vk(:,1);
%================================================================
%Guarda os valores históricos de gbest e da função custo 'func'
%================================================================
  gbest1(ite)=gbest;
  func(ite)=-(abs((gbest).*sin(sqrt(abs(gbest))))) -418.9829;



%%
%Criação do gráfico interativo
X=pbest;
Y=-pbest;
A=meshgrid(X,Y);
swarm=-(abs(pbest.*sin(sqrt(abs(pbest)))))-418.9829;

%================================================================
 % Desenhando o enxame
```

```
%===============================================================
figure(1)
clf;
plot(A,swarm,'bx');
xlabel('Pbest');
ylabel('Custo');
title('Espaço de busca');
axis([0 550 -100 550]);
pause(.3);

disp(['iteração:' num2str(ite) ': Melhor Custo:' num2str(abs(func(ite))) ': Gbest:' num2str(gbes-
t1(ite))]);

%Fim do algoritmo
end
toc; %termina a contagem do tempo
%% Criação do plot final
x=0:1:512;
fcusto =-((abs(x.*sin(sqrt(abs(x)))))-418.9829);
lx = length(x);

figure (2)
plot(1:1:lx,fcusto,'LineWidth',2);
xlabel('Iteração');
ylabel('Função custo');
grid on;

lgt=size(func,2);
figure(3)
plot(1:1:lgt,func,'LineWidth',2);
%semilogy(BestCost,'LineWidth',2);
xlabel('Iteração');
ylabel('Melhor custo');
grid on;


end
```

```
% Nome - psof
% Algoritmo de otimização por enxame de partículas para funções contínuas
% Desenvolvido por Dr. Raul Vitor Arantes Monteiro / Dezembro de 2023
% Grupo de Pesquisa em Operação de Sistemas Elétricos e Redes Inteligentes
% Departamento de Engenharia Elétrica
% Universidade Federal de Mato Grosso

function psof
clc;
clear;

rng default
FitnessFcn=@toma;
nSwarnSize = 50;
nIters = 100;
nVars=1;
lb=0;
ub=512;

options = optimoptions('particleswarm',...
                'CreationFcn',{@pswcreationuniform},...
                'InertiaRange',[0.1,1],...
                'InitialSwarmSpan',512,...
                'MaxIterations', nIters,...
                'MaxStallIterations',nIters,...
                'PlotFcn',{@pswplotbestf,@pswplot},...
                'SwarmSize',nSwarnSize,...
                'Display','iter');

[Xbest,Gbest,~,~] = particleswarm(FitnessFcn,nVars,lb,ub,options);

disp('============================================================')
disp(['Melhor Custo Global: ' num2str(floor(Gbest)) ' | Melhor Posição Global: ' num2s-
tr(Xbest)]);
disp('============================================================')

end
```

```
function stop = pswplot(optimValues,~)

stop = false;

  X=(optimValues.swarm);
  Y=-(optimValues.swarm);
  A=meshgrid(X,Y);
  plot(A',optimValues.swarmfvals,'bx');
  xlabel('Melhores Soluções');
  ylabel('Custo');
  title('Espaço de busca');
  axis([0 550 -100 550]);
  pause(.1);

end
```

```
% Nome - gaf
% Algoritmo de otimização por algoritmo genético
% Desenvolvido por Dr. Raul Vitor Arantes Monteiro / Dezembro de 2023
% Grupo de Pesquisa em Operação de Sistemas Elétricos e Redes Inteligentes
% Departamento de Engenharia Elétrica
% Universidade Federal de Mato Grosso
% Função custo: -(abs(x.*sin(sqrt(abs(x)))))-418.9829;

function gaf
clc;
clear;
rng default
FitnessFcn=@toma;
nPopSize = 50;
nIters = 100;
nVars=1;
lb=0;
ub=512;
tournamentSize = 2;
nGenes=nVars*8;
nElites = round(nPopSize * 0.2);
options = optimoptions('ga',...
                'CreationFcn',{@gacreationlinearfeasible},...
                'PopulationSize',nPopSize,...
                'InitialPopulationRange',[0;512],...
                'MaxGenerations',nIters,...
                'SelectionFcn',{@selectiontournament,tournamentSize},...
                'MutationFcn',{@mutationadaptfeasible, 0.1},...
                'CrossoverFcn', {@crossovertwopoint},...
                'EliteCount',nElites,...
                'MaxStallGenerations',100,...
                'PlotFcn',{@gaplotbestf,@gaplot},...
                'Display', 'diagnose');

[chromosome, y_fit,~,~,~,~] = ga(FitnessFcn, nVars,[], [], [], [], lb, ub, [],options);

disp('============================================================')
disp(['Melhor Custo: ' num2str(floor(y_fit)) ' | Melhor Indivíduo: ' num2str(chromosome)]);
```

```
disp('============================================================')
end

function [state,options,optchanged] = gaplot(options,state,~)

optchanged = false;

  X=(state.Population(:,1));
  Y=-(state.Population(:,1));
  A=meshgrid(X,Y);
  plot(A',state.Score,'bx');
  xlabel('Melhores Soluções');
  ylabel('Custo');
  title('Espaço de busca');
  axis([0 550 -100 550]);
  pause(.1);

end
```