Appendix

MATLAB Code

There are two GAs in this appendix. The first is a binary GA using uniform crossover and tournament selection. The second is a continuous GA using single-point crossover and roulette wheel selection. You are encouraged to try other crossover and mutation operators. The population size and mutation rates are easily changed. Stopping criteria include the maximum number of iterations, maximum number of function calls, and a minimum cost.

The cost function can be changed by placing the name of the MATLAB® function between the single quotes in the line `ff=''`. It returns a column cost vector of costs. Three cost functions are provided here. The first two are the mathematical functions

$$f_1(\mathbf{x}) = \sum_{n=1}^{nvar} x_n \tag{A.1}$$

$$f_2(\mathbf{x}) = 60 + \sum_{n=1}^{nvar} \left[x_n^2 - 10\cos(2\pi x_n)\right] \tag{A.2}$$

Both have a minimum of zero when all $x_n = 0$. The third cost function returns the maximum sidelobe level for an amplitude weighted array factor. Its equation is written as

$$af(u) = \sum_{n=1}^{N} a_n e^{jkx_n u} \tag{A.3}$$

Genetic Algorithms in Electromagnetics, by Randy L. Haupt and Douglas H. Werner

For this function the amplitude weights are assumed symmetric, so *nvar* = *N/2*. This cost function may be tested using thinning and the binary GA or amplitude tapering and the continuous GA.

The MATLAB code is as follows:

```
% bga
%
% This is a typical GA that works with binary
variables
% Uses - uniform crossover
%      - tournament selection
%
% Randy Haupt
% 11/21/05

clear
global funcount
funcount=0;

% Defining cost function
nvar=6;
nbits=3;
Nt=nvar*nbits;
ff='testfun3';

% GA parameters
npop=8; % population size
mutrate=0.2; % mutation rate
el=1; % number of chromosomes not mutated
Nmut=ceil(mutrate*((npop-el)*Nt)); %# mutations
% stopping criteria
maxgen=400; % max # generations
maxfun=2000; % mas # function calls
mincost=-50; % acceptable cost

% initial population
P=round(rand(npop,Nt));

% cost function
cost=feval(ff,P);
[c,in]=min(cost);
tp=P(1,:); tc=cost(1);
P(1,:)=P(in,:); cost(1)=cost(in);
P(in,:)=tp; cost(in)=tc;
minc(1)=min(cost); % best cost in each generation
```

```
for gen=1:maxgen

   % Natural selection
   indx=find(cost<=mean(cost));
   keep=length(indx);
   cost=cost(indx); P=P(indx,:);
   M=npop-keep;

   % Create mating pool using tournament selection
   Ntourn=2;
   for ic=1:M
      rc=ceil(keep*rand(1,Ntourn));
      [c,ci]=min(cost(rc)); % indicies of mother
      ma=rc(ci);
      rc=ceil(keep*rand(1,Ntourn));
      [c,ci]=min(cost(rc)); % indicies of father
      pa=rc(ci);
      % generate mask
      mask=round(rand(1,Nt));
      % crossover
      P(keep+ic,:)=mask.*P(ma,:)+not(mask).*P(pa,:);
   end

   % Mutation
   elP=P(el+1:npop,:);
   elP(ceil((npop-el)*Nt*rand(1,Nmut)))=round(rand(1,
     Nmut));
   P(el+1:npop,:)=elP;

   % cost function
   cost=feval(ff,P);
   [c,in]=min(cost);
   tp=P(1,:); tc=cost(1);
   P(1,:)=P(in,:); cost(1)=cost(in);
   P(in,:)=tp; cost(in)=tc;

   minc(gen+1)=cost(1);
   [gen cost(1)]
   % Convergence check
   if funcount>maxfun | gen>maxgen | minc(gen+1)
     <mincost
      break
   end

end
```

```
% Present results
day=clock;
disp(datestr(datenum(day(1),day(2),day(3),day(4),day(5),day(6)),0))
disp([‘optimized function is ‘ ff])
format short g
disp([‘# variables = ‘ num2str(nvar) ‘ # bits = ‘
  num2str(nbits)])
disp([‘min cost = ‘ num2str(mincost)])
disp([‘best chromosome = ‘ num2str(P(1,:))])

figure(1)
plot([0:gen],minc)
xlabel(‘generation’);ylabel(‘cost’)



% cga
%
% This is a typical GA that works with continuous
variables
% Uses - single point crossover
%      - roulette wheel selection
%
% Randy Haupt
% 11/21/05

clear
global funcount
funcount=0;

% Defining cost function
nvar=10;
ff=’testfun3’;

% GA parameters
npop=8; % population size
mutrate=0.15; % mutation rate
natsel=npop/2; % #chromosomes kept
M=npop-natsel; % #chromosomes discarded
el=1; % number of chromosomes not mutated
Nmut=ceil(mutrate*((npop-el)*nvar)); %# mutations
parents=1:natsel; % indicies of parents
prob=parents/sum(parents); % prob assigned to parents
odds=[0 cumsum(prob)]; Nodds=length(odds); % cum prob
% stopping criteria
```

```
maxgen=500; % max # generations
maxfun=2000; % mas # function calls
mincost=-50; % acceptable cost

% initial population
P=rand(npop,nvar);

% cost function
cost=feval(ff,P);
% Natural selection
[cost ind]=sort(cost);
P=P(ind(1:natsel),:);
cost=cost(1:natsel);

minc(1)=min(cost); % best cost in each generation

for gen=1:maxgen

  % Create mating pool
  for ic=1:2:M
      r=rand;ma=max(find(odds<r)); % indicies of mother
      r=rand;pa=max(find(odds<r)); % indicies of father
      xp=ceil(rand*nvar);          % crossover point
      r=rand;                      % mixing parameter
      xy=P(ma,xp)-P(pa,xp);        % mix from ma and pa
      % generate masks
      mask1=[ones(1,xp) zeros(1,nvar-xp)];
      mask2=not(mask1);
      % crossover
      P(natsel+ic,:)=mask1.*P(ma,:)+mask2.*P(pa,:);
      P(natsel+ic+1,:)=mask2.*P(ma,:)+mask1.*P(pa,:);
      % create single point crossover variable
      P(natsel+ic,xp)=P(natsel+ic,xp)-r*xy;
      P(natsel+ic+1,xp)=P(natsel+ic+1,xp)+r*xy;
  end

  % Mutation
  elP=P(el+1:npop,:);
  elP(ceil((npop-el)*nvar*rand(1,Nmut)))=rand(1,Nmut);
  P(el+1:npop,:)=elP;

  % cost function
  cost=feval(ff,P);
  % Natural selection
  [cost ind]=sort(cost);
```

```
   P=P(ind(1:natsel),:);
   cost=cost(1:natsel);

   minc(gen+1)=cost(1);
   [gen cost(1)]
   % Convergence check
   if funcount>maxfun | gen>maxgen | minc(gen+1)<
     mincost
      break
   end

end

% Present results
day=clock;
disp(datestr(datenum(day(1),day(2),day(3),day(4),day(5),day(6)),0))
disp([‘optimized function is ‘ ff])
format short g
disp([‘ # par = ‘ num2str(nvar)])
disp([‘min cost = ‘ num2str(mincost)])
disp([‘best chromosome = ‘ num2str(P(1,:))])

figure(1)
plot([0:gen],minc)
xlabel(‘generation’);ylabel(‘cost’)

% a test function with one local minima at xn=0
%
% Randy Haupt
% 11/21/05

function sll=testfun1(chrom)
global funcount

[nr,nc]=size(chrom);
funcount=funcount+nr; % keeps counting number of
  function calls
bb=10*(chrom-.5); % transforms chromosome variables

sll(:,1)= bb.^2*ones(nc,1);

% a test function with many local minima at xn=0
%
% Randy Haupt
% 11/21/05
```

```
function sll=testfun2(chrom)
global funcount

[nr,nc]=size(chrom);
funcount=funcount+nr; % keeps counting number of
  function calls
bb=10*(chrom-.5); % transforms chromosome variables

sll(:,1)=10*nc+[bb.^2-10*cos(2*pi*bb)]*ones(nc,1);


% a test function for cga - place a null with phase
only weighting
%
% Randy Haupt
% 11/21/05

function sll=testfun3(chrom)
global funcount

[nr,nc]=size(chrom);
funcount=funcount+nr; % keeps counting number of
  function calls

k=2*pi; % wavenumber
d=0.5; % element spacing
N=2*nc; % number of elements in array
x=(0:(N-1))*d; % element spacing
u=0:2/10/N:1; % u=cos(phi)
Q=exp(j*k*x'*u); % phase

for ic=1:nr
    w=[fliplr(chrom(ic,:)) chrom(ic,:)]; % amplitude
      weights
    af=20*log10(abs(w*Q)).'; % array factor in dB
    af=af-max(af); % normalize array factor
    saf=flipud(sort(af));
    ind=min(find(saf>af));
    sll(ic,1)=saf(ind(1)); % max sidelobe level
end
```