Part I

Appendix of Functions for Evolution Strategies

es2( )

```
function [xs_arch,best] = es2(seed,mu,lambda,strat_rc, ctrl_rc, cool,sigmult)
%INPUT
%seed          - random seed number
%mu            - number of parents
%lambda        - number of offspring
%NB            - schwefel recommends mu:lambda = 1:7
%strat_rc      - strategy variable recombination type e.g. bi, gi, gd, bd
%control_rc    -control variable recombination;
% see recombine function for more information
% cool         - temperature cooling rate
%sigmult       - initial multiplier for variances (strategy variables)
%OUTPUT
%xs_arch       -archive of best solutions
%best          -list of best function values as time progresses



%HouseKeeping
%--------------------------------------------------------------
rng(seed);
%number of function evaluations
noEvalf=0;
% archived variables
xs_arch=[]; fs_arch=[];
%--------------------------------------------------------------


% Initialise Strategy parameters
%--------------------------------------------------------------
% initialised based on Scwefel
% rotation angle init. in [0,2pi] for each child
alpha = 2*pi*rand(lambda,1);
% variances initialised in [2-0],[1-0] for each child
sigma =[2*rand(lambda,1),rand(lambda,1)]*sigmult;
% variables to control mutation of strategy parameters
tau   = 1/(sqrt(mu*lambda*sqrt(mu*lambda)));
beta  = .0873;
%--------------------------------------------------------------


% Generate Initial Parent  Population
%--------------------------------------------------------------
% Population randomly generated in [-1,1]*[-2,2]
% Vaues Stored in x, y
x=4*rand(lambda,1)-2;
y=2*rand(lambda,1)-1;
% Generation  number for parents
gen=1;
%Fit values of x,y for the camel-hump function
%See function  at end of text for extra details
fitxy = fit(x,y);
%store the best solutions found for current generation
best  = nan(floor(1000/lambda),2);
best(gen,:)=[noEvalf,min(fitxy)];
%--------------------------------------------------------------



%MAIN LOOP:
%Evolution cycle involves: selection, Mutate, recombination, Mutate, Assessment
while noEvalf <1000



    %Housekeeping/ Plotting work
    %--------------------------------------------------------------
    % Compute the best sol or a given generation recursively
    if gen>1
        best(gen,:)=[noEvalf,min(best(gen-1,2),min(fitxy))];
    end
    % Update the archive using the replace funciton
    [xs_arch, fs_arch]=replace([x,y],fitxy,xs_arch,fs_arch);
    %--------------------------------------------------------------

```

```matlab

    %1.SELECTION: select the mu best parents
    % must select their x, y values, rotation matrix and variance (sd)
    %------------------------------------------------------------
    [~,index]=sort(fitxy);
    index=index(1:mu);
    x=x(index); y=y(index); sigma=sigma(index,:); alpha=alpha(index);
    %------------------------------------------------------------



    %2.Mutate strategy parameters: variances and rotation angles
    %------------------------------------------------------------
    for i=1:mu
        % cool the rotation hyperparameter
        tau=tau*cool;
        % aleph are random numbers to mutate variances and rotations
        alephi=randn(1,2)*[1,0;0,0.5];
        alephij=randn(1);
        sigma(i,:)=sigma(i,:).*exp(+tau*alephi);
        % cool the actual value of the variance
        sigma(i,:) =sigma(i,:).*cool;
        % Don't cool the rotation angle
        alpha(i)=alpha(i)+beta*alephij;
    end;
    %------------------------------------------------------------



    %3.Recombine strategy parameters and offspring:
    %---------------------------------------------------------------
    %recombine variances according to strat_rc, which gives type of recomb
    [z1,z2]=recombine(sigma(:,1),sigma(:,2),lambda,strat_rc); sigma=[z1,z2];
    %recombine rotation angles according to strat_rc
    [alpha,~]=recombine(alpha,alpha,lambda,strat_rc);
    %recombine variances according to control_rc, which gives type of recomb
    [x,y]=recombine(x,y,lambda,control_rc);
    %--------------------------------------------------------------



    %4.Mutate population's control (x,y) variables
    %------------------------------------------------------------
    % Build rotation matrix
    for i=1:lambda
       R=                   [ cos(alpha(i)),      -sin(alpha(i));
                              sin(alpha(i)) ,      cos(alpha(i)) ];
        % sampling from distribution induced by rotation matrix and variances

        % Create mutations
        % Apply rotation matrix to variances, and sample from distribution
        % sampling is performed by transforming a 2-d standard normal using
        % rotation matrices and variances
        mutate=R*(diag(sigma(i,:)))*randn(2,1);

        % mutate the 25 strong population
        x(i)=x(i)+mutate(1);
        y(i)=y(i)+mutate(2);
    end

    %5. Assess New population for fitness
    %------------------------------------------------------------
    fitxy = fit(x,y);
    %------------------------------------------------------------

    % End of loop: Increase generation number of parents
    gen=gen+1;

end

  function z= fit(x,y)
    %fit values of x,y under camel function with penalty, inc counter

    %Find penalty for out of bound vars
    %-----------------------------------------------------
    % degree to which x,y out of bounds
    outx = max(abs(x)-2,0);
    outy = max(abs(y)-1,0);
    % penalty exponent, as time increases penalty harsher
    k=0.5;
    % penalty weight: harsher for y because bound smaller
    w=1./[2,1];
    %penalty term
    penalty=(gen^k)*[outx,outy]*w';
    %-----------------------------------------------------

    %Return camel function + penalty
    %-----------------------------------------------------
```

```
    z=camel(x,y)+penalty;
    % increment counter for function evaluations
    noEvalf=noEvalf+size(x);
  end


end
```

recombine( )

```
function [x_out,y_out]= recombine(x,y,num,type)
    % function recombines according to discrete intermediate etc etc
    % Creates num new vectors in [x-out,y_out] from [x,y]
    % according to recombination type specified in type.
    % EXPLANATION
    % Discrete     - form new features from only 2 specific vectors
    % Global       - combine features from any 2 vectors
    % Intermediate - continuously combine parameters
    % Discrete     - pick either one feature or another


        % z is a temprary sore for the recombined variables
        len=length(x); z=nan(num,2);

        %
        if len==1
            x_out=repmat(x,num,1); y_out=repmat(y,num,1); return;
        end
        switch type
          % GLOBAL INTERMEDIATE:
          case 'gi'
            % iterate over features
            for j=1:num
                %pick any 2 vectors to recombine
                pick=randperm(len);
                weight=rand(1);
                z(j,1) = weight*x(pick(1),1)+(1-weight)*x(pick(2),1);
                weight=rand(1); pick=randperm(len);
                z(j,2)=weight*y(pick(1),1)+(1-weight)*y(pick(2),1);
            end
          % GLOBAL DISCRETE:
          case'gd'
              % iterate over features
            for j=1:num
                % pick 2 vectors to combine
                pick=randperm(len);
                z(j,1) = x(pick(1),1);
                pick=randperm(len);
                z(j,2)=y(pick(2),1);
            end
         % BIVARIATE DISCRETE:
         case 'bd'
            % pick 2 vectors to combine
            pick=randperm(len);
            % iterate over features
            for j=1:num
                weight=randi(2);
                z(j,1) = x(pick(weight),1);
                weight=randi(2);
                z(j,2)=y(pick(weight),1);
            end
         %BIVARIATE INTERMIEDIATE:
         case 'bi'
            % pick 2 vectors to combine
            pick=randperm(len);
            % iterate over features
            for j=1:num
                weight=rand(1);
                z(j,1) = weight*x(pick(1),1)+(1-weight)*x(pick(2),1);
                weight=rand(1);
                z(j,2)=weight*y(pick(1),1)+(1-weight)*y(pick(2),1);
            end
            otherwise
             print('error: no such recombinationoption')
        end
        x_out=z(:,1); y_out=z(:,2);
 end
```

Part II

Appendix of Functions for Simulated Annealing

anneal( )

```
function [xs_arch,best]=anneal(seed,type_pro, type_en,type_cool,beta_c, T0, inc_alpha, dec_alpha, restart)
%INPUT
%type_pro  - type of proposal distrivbution 'gauss', '...
%type_en   - type of energy function    '
%type_cool - type of cooling function 'geom', '...
%T0        - initial temperature
%inc-alpha - increase factor for proposal distributions
%dec-alpha - decrease factor for proposal distribution
%restart   - restart type- 'none', 'minima', 'random'
%-----------------------------------------------------------------

% Initialisation:
%-----------------------------------------------------------------
%Meta variables.
% set time to 1
t=1;
% set number of function evaluations to 0
noEvalf=0;
% best stores the best function value found so far
best=nan(1000,1);
% alpha is the initial multiplier for proposal dist, like a step size
alpha=1;
% f_prev is previous function value of an accepted point
f_prev=1000;
% f_curr  is the function value of the current accepted point
f_curr;
% f_ eval is the value of a the current function evaluation, whether
% accepted or not.
f_eval;
% number of restarts is 0
noRes=0;

%First Points.
%-----------------------------------------------------------------
% Archive variables, xs is for the [x,y] values and fs is function value
xs_arch=[]; fs_arch=[];
% initialise the first x point in [-2,2]*[-1,1]
x=[4*rand()-2,2*rand()-1];
% compute it's energy, which calls the came function
e_x=energy(x);
% update function_pointer values
f_prev=f_curr;
f_curr=f_eval;



    %Function to restart x values to escape local minima
    %-------------------------------------------------------------
    function restarts()
        noRes=noRes+1;
        %Reset initial temperature and turn back the clock
        T0=temp0; t=1;
        %step size/ prop dist. factor also reset to 1
        alpha=1;
        % perform either a random restart or restart from a minima:

        if ~strcmp(restart,'none')
            if strcmp(restart,'minima')
                % decrease initial temperature slight from what it was, so
                % each restart has alittle less energy
                T0=T0*(0.7)^noRes;
            % Random restarts:
            elseif strcmp(restart,'random')
                %randomly place [x,y] in [-2,2]*[-1,1]
                x=[4*rand()-2,2*rand()-1];
                % compute energy associated with x
                e_x=energy(x);
                % Update function holders
                f_prev=f_curr;
                f_curr=f_eval;
            end
        end
    end
```

```matlab
    %------------------------------------------------------------




% Main Loop: Cool, Propose, Archive
%----------------------------------------------------------------
while noEvalf<1000
    % update best solutions found so far
    best(t)=min(f_eval,best(max(t-1,1)));

    %1. Find the temperature at the current time
    %------------------------------------------------------------
    temp=cool_t(t-1);

    %2. Propose a new point from proposal distribution
    %------------------------------------------------------------
    y= proposal(x);
    e_y=energy(y);

    %3. Update the archive with the new point y
    %------------------------------------------------------------
    [xs_arch, fs_arch]=replace([y(1),y(2)],f_eval,xs_arch,fs_arch)

    %4. Accept new point with probability P(v)
    %------------------------------------------------------------
    %- rand generates a unfirom random number in [0,1]
    u=rand();
    % v is the ratio of energies exponentiated by inverse temperature
    % -explained in coursework
    v=min(1,(e_y/e_x)^(1/temp));
    if u <= v
            % 4.1 Point has been accepted
            %----------------------------------------------------
            % update x and f_prec and f_curr
            x=y; e_x=e_y;
            f_prev=f_curr;
            f_curr=f_eval;
            % We increase the step size/prop. dist factor after success
            alpha=inc_alpha*alpha;
    else

            %4.2 Point not accepted
            %-----------------------------------------------------
            % decrease step size
            alpha=dec_alpha*alpha;

             %if sufficiently converged and option to restart => restart
            if abs(f_eval-f_prev) <0.0001 & ~strcmp(restart,'none')
                restarts()
            end;

    end
    % loop over, increase time by 1
    t=t+1;
end


function t_out=cool_t(t)
% cooling function - input is time t
% beta_c is cooling rate
% T0 is initial temperature
        switch type_cool
            case 'geom'
                t_out=T0*beta_c^t;
            case 'inverse-linear'
                t_out=T0/(2+beta_c*t);
            case 'logarithmic'
                t_out=T0/log(beta_c*t+exp(1));
            case 'inverse-expo'
                t_out=T0*exp(1-beta_c*t);
        end
end

function y=proposal(x)
% input x is current accepted point
% sample from proposal distributions centred on x
% type_pro gives type of proposal distribution
% alpha is a parameter to control how far from x is the
% point we sample

       switch type_pro
         case 'gauss'
                dx=randn(1,2)*diag([2,1]);
                dx=dx.*alpha;
         case 'uniform'
                dx=3*(rand(1,2)*diag([2,1])-[1,0.5]);
                dx=dx.*alpha;
```

```
         case 'logistic'
                pd = makedist('Logistic','mu',0,'sigma',alpha);
                dx1=pd.random(); dx2=pd.random();
                dx=[dx1,dx2]*diag([2,1]);
        end
        y=x+dx;
        alpha;

end


% energy functions
function e= energy(x)
% camel is the camel-hump function
% x is a new point for which we want to calculate energy
% We also update the counter for function evaluations
% We want energy to be decreasing in f
% type_en gives the different types of functions

        f=camel(x(1),x(2));
        f_eval=f;
        switch type_en
            case 'cubic'
                f=f^3;
            case 'squared'
                f=sign(f)*f^2;
            case 'root'
                f=sign(f)*(abs(f))^(1/2);
            case 'normal'
                % energy just is -f
                f=f;
            case 'atan'
                f=atan(f);
        end
        e=exp(-f);
        noEvalf=noEvalf+1;
end

end

% Ploting Functions: Stage 1, gather parameters
% s i struct with parameters for points of interest
s=struct();
% parameters varied
strat={'gi','gd','bd','bi'};
ctrl ={'gi','gd','bd','bi'};

for i=1:length(strat)
    for j=1:length(ctrl)
        v0=0;l=0;
        st=strat{i};
        ct=ctrl{j};
        s(i,j).strat=st;
        s(i,j).ctrl=ct;
        % Iterate over 50 different seed values and average archive/best f
        for seed=1:50
            [xs_arch,v]=es2(seed,5,15,st,ct);
            v0=v+v0;
            l=l+length(xs_arch);
        end;
        s(i,j).v=v0/50;
        s(i,j).len=l/50;
        s(i,j).leg=(['ctrl= ',ct]);
    end
end
```

Part III

Function used for Both Parts

camel( )

```
% camel function
function f = camel(x,y)
    f =  (4-2.1*(x.^2)+(x.^4)/3).*(x.^2) + x.*y+(4*(y.^2)-4).*(y.^2);
end


 % function to plot the contours of camel function
 function plotR(x1,y1,a)
    % Args: x1 and y1 are vectors, a is a colour
    % Plot contours
    %-------------------------------------------------------
    x = linspace(-10,10); y = linspace(-10,10);
    [xx,yy] = meshgrid(x,y); ff = camel(xx,yy);
    levels = -300:1:300;
    LW = 'linewidth'; FS = 'fontsize'; MS = 'markersize';
    contour(x,y,ff,LW,1.2), colorbar
    axis([-2 2 -2 2]), axis square, hold on
    % Plot points
    %-------------------------------------------------------
    scatter(x1,y1,a)
 end
```

plot( )

```
% function to plot the contours of camel function
 function plotR(x1,y1,a)
    % Args: x1 and y1 are vectors, a is a colour
    % Plot contours
    %-------------------------------------------------------
    x = linspace(-10,10); y = linspace(-10,10);
    [xx,yy] = meshgrid(x,y); ff = camel(xx,yy);
    levels = -300:1:300;
    LW = 'linewidth'; FS = 'fontsize'; MS = 'markersize';
    contour(x,y,ff,LW,1.2), colorbar
    axis([-2 2 -2 2]), axis square, hold on
    % Plot points
    %-------------------------------------------------------
    scatter(x1,y1,a)
 end
```

replace( )

```
function [xs,fs] = replace(xnew,fnew,xs,fs)
% Updates the old archive xs fswith new values
% xnew - a column of vectors to be input
% fnew - function value at xnew
% xs   - old archive, to be updated
% fs   - function  values for old archive, to update

%Archive Params- as in lecture notes
%dmin  - threshold for points to be considered far
%l=max - length of archive
%d_sim - threshold for points to be considered close
d_min=0.5; l=40; d_sim=0.1;

% iterate through all values of archive
for z =1:size(xnew,1)
    % x and f are archived values
    x=xnew(z,:); f=fnew(z,:);

    % If archive is empty update it
    %-----------------------------------------------------
    if (isempty(xs))
        xs=[xs;x]; fs=[fs;f];
        return;
    end
    %-----------------------------------------------------

    %Preprocessing: finding min distance of x to archive vals
    %-----------------------------------------------------
    %distance is distance of archived values from xnew
    vector= xs-repmat(x,size(xs,1),1);
    distance=sqrt(sum(vector.*vector,2));
    %dist_near is nearest x distance of archive from xnew
    [dist_near, near]=min(distance);
    %minimum and maximum values of the archive
    [fmin, imin]=min(fs);
    [fmax, imax]=max(fs);
    %-----------------------------------------------------

    %Main comparisons.
    %-----------------------------------------------------
    % xnew is not near any points and archive not full- so update
    if  (length(xs) < l & dist_near > d_min)
        xs=[xs;x]; fs=[fs;f];
    % xnew is better than the worst point archived, and not near any points
    %so replace with worst value in archice
    elseif size(xs,1)==l && dist_near > d_min & (f<fmax)
        xs(imax,:)=x; fs(imax,:)=f;
    % xnew is best value found or better than the xs value its near to
    % so replace the nearest value
    elseif (f<fmin) | (dist_near < d_sim & f<fs(near))
        xs(near,:)=x; fs(near,:)=f;
    end
end

 end
```

Example Script to Plot Figures Used in CW

```
% Ploting Functions: Stage 1, gather parameters
% s i struct with parameters for points of interest
s=struct();
% parameters varied
strat={'gi','gd','bd','bi'};
ctrl ={'gi','gd','bd','bi'};

for i=1:length(strat)
    for j=1:length(ctrl)
        v0=0;l=0;
        st=strat{i};
        ct=ctrl{j};
        s(i,j).strat=st;
        s(i,j).ctrl=ct;
        % Iterate over 50 different seed values and average archive/best f
        for seed=1:50
            [xs_arch,v]=es2(seed,5,15,st,ct);
            v0=v+v0;
            l=l+length(xs_arch);
        end;
        s(i,j).v=v0/50;
        s(i,j).len=l/50;
        s(i,j).leg=(['ctrl= ',ct]);
    end
end
% Ploting Functions: Stage 2, plot parameters

for i=1:length(strat)
for j=1:length(ctrl)
    % plots ave. best f found over 1000 f-evals in fig 1
            figure(1)
            hold on
            subplot(2,2,i)
            plot(s(i,j).v(:,1),s(i,j).v(:,2));
            axis([0 1000 -1.0317 -0.9])
            legend(s(i,:).leg);
            title(['strat= ',s(i,j).strat])

            % plots ave. archive in fig 2
            figure(2)
            hold on
            subplot(2,2,i)
            scatter(j,s(i,j).len,100,'diamond','filled')
            axis([1 length(ctrl) 0 40])
            legend(s(i,:).leg);
            title(['strat= ',s(i,j).strat])

end

figure(1); hold on;
%title('min f(x) found so far, averaged over 50 runs ')
xlabel('function evals.'); ylabel(' min f(x)');

figure(2)
hold on
%title('size of archive on termination, averaged over 50 runs')
set(gca,'xtick',[]); ylabel('length');
end
```