Jesse Wynn
ME 575
Homework 5
March 08, 2018

Simulated Annealing

Algorithm Parameters:

```
N = 50
Ps = 0.6
Pf = 1e-04
delta = 1.25
num_perturbations = 3 (per cycle)

global minimum: x* = [0; 0]
f(x*) = 0
```

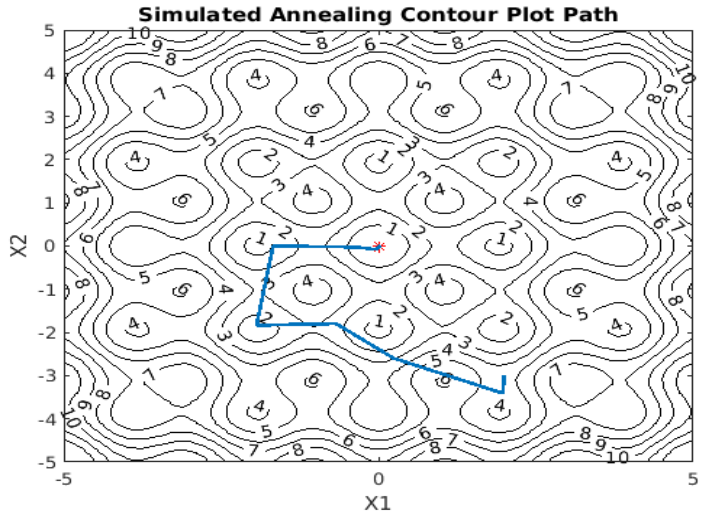
Discussion of Parameters:

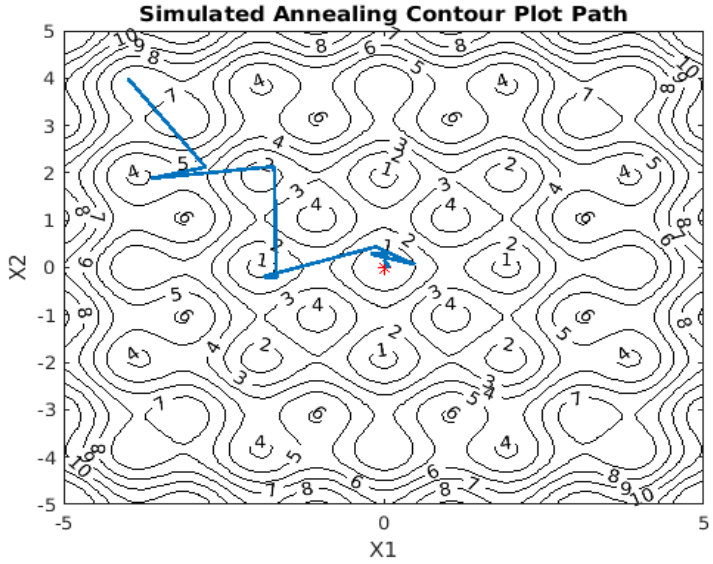
- The parameter N = 50 was chosen because it was shown to be enough cycles to consistently reach the optimum. Initially I started with N = 100 but realized that similar performance with much fewer function calls could be achieved with N = 50.
- Ps was chosen to be 0.6. This means that in the beginning, there was a 60% chance of picking a worse set of x values. This was good because in the beginning we need to get out of the local min that we're closest to, and work towards a global min.
- Pf was chosen to be 1e-04. This means that at the end, there is a 1 in 10,000 chance that we will move to a worse state—this is what we want when we think we've arrived near the optimum.
- The parameter delta was chosen to be 1.25. Based on the contour plot, it looks like the approximate distance between local minimus is a little more than 1.0 on average and so delta of 1.25 was chosen to give the algorithm a good chance of getting out of a local minimum 'rut'. Also, after we were 80% through our cooling cycles, I set delta to 10% of it's original value. This is so that when we're close to the optimum, the algorithm can do a better job of 'honing' in on the true optimum.
- The number of perturbations per cooling cycle was chosen to be 3. This was based on the text where it said that this parameter should be at least equal to the number of design variables (2 in this case).

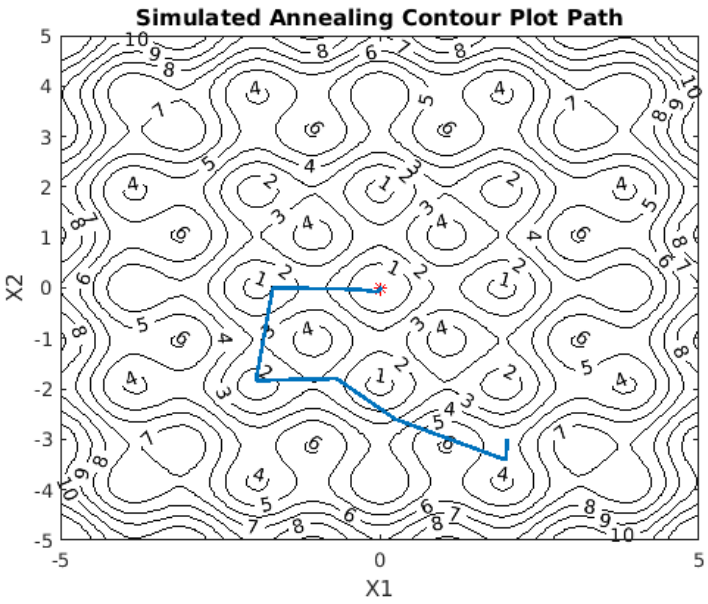
Performance:

In a series of 20 runs from different places in the design space, **the simulated annealing algorithm was successful in arriving within 0.1 of the true global optimum 90 to 95% of the time**. To be honest, this is better performance than I would have initially expected given the relative simplicity of the algorithm. I believe that more fiddling with parameters could possibly eek out a little more performance but for now I am satisfied. On average the algorithm required **approximately 215 function evaluations**.

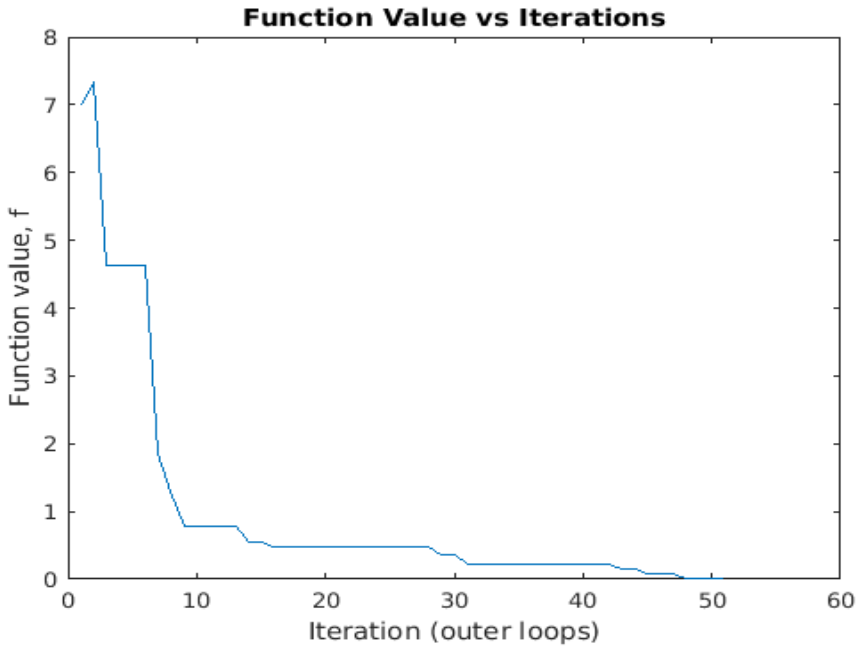
Contour plots of path from three different starting poins:







Example Function vs Iterations Plot:



MATLAB Code:

```
clc
clear
close all

% tuning params
N = 50;
Ps = 0.6;
Pf = 1e-4;
delta = 1.25;
num_inner_loops = 3;

global nfun;
nfun = 0;

% arrays to hold data
dE_vals = [];
f_vals = [];
% T_vals = [];
T_vals = zeros(1,100000);
x_states = zeros(100000,2);

% initial x value
x0 = [4, 3]';
x_states(1,:) = x0;

% compute starting and final temperatures
Ts = -1/log(Ps);
Tf = -1/log(Pf);

% compute F
F = (Tf/Ts)^(1/(N - 1));

% initialization
f = get_f(x0);
f_vals(1) = f;
count = 0;
x = x0;
T = Ts;
incount = 0;
% while we haven't reached Tf...
while T > Tf

    for i=1:num_inner_loops

        incount = incount + 1;
        T_vals(incount) = T;

        % perturb the current x
        if count > 0.8*N
            xp = x + randn(2,1)*delta*0.1;  % if we're close to the end, make delta very small
        else
            xp = x + randn(2,1)*delta;  % the regular perturbation
        end
```

```
    % saturate xp s.t. -5 < xp < 5
    xp = saturate(xp);

    % compute f at the perturbed x
    fp = get_f(xp);

    % if fp is lower than our current f, then accept it and move on
    if fp < f
        % accept the perturbed design
        x = xp;
        f = get_f(x);

        % compute dE and add to the dE_vals array
        dE = abs(fp - f);
        dE_vals(length(dE_vals) + 1) = dE;
    else
        % compute dE
        dE = abs(fp - f);

        % check to see if we have more than one value to average
        if isempty(dE_vals)
            dE_avg = dE;
        else
            % compute the average dE
            dE_avg = sum(dE_vals)/length(dE_vals);
        end

        % compute probability of selecting a worse desing
        P = exp(-dE/(dE_avg*T));

        % see if random number is less than P
        rnum = rand(1);
        if rnum < P
            % accept the perturbed design
            x = xp;
            f = get_f(x);

            % add the 'accepted' dE value to the array
            dE_vals(length(dE_vals) + 1) = dE;
        else
            % don't accept and we keep the current design
            x = x;
            % f = get_f(x);
        end

    end

end

% decrease the temperature
T = F * T;

% increment the outer loop counter
count = count + 1;

% store function values from each loop
f_vals(length(f_vals) + 1) = get_f(x);
% T_vals(length(T_vals) + 1) = T;
```

```
    x_states(count + 1,:) = x';

end

iters = 1:length(f_vals);
x_states = x_states(1:length(f_vals),:);

figure(1), clf
plot(iters, f_vals)
xlabel('Iteration (outer loops)')
ylabel('Function value, f')
title('Function Value vs Iterations')

iters = 1:incount;
T_vals = T_vals(1:incount);
figure(2), clf
plot(iters, T_vals)
xlabel('Iteration (inner loops)')
ylabel('Temperature, T')
title('Temperature vs Iterations')

% call the contour plotter
plot_contours(x_states);

x
f = get_f(x)

if abs(x(1)) < 0.1 && abs(x(2)) < 0.1
    disp('Success')
else
    disp('Failed to converge to global optimum.')
end

nfun

function fval = get_f(x)
global nfun
fval = 2 + 0.2*(x(1)^2) + 0.2*(x(2)^2) - cos(pi*x(1)) - cos(pi*x(2));
nfun = nfun + 1;
end

function x_sat = saturate(x)
if x(1) > 5
    x(1) = 5;
elseif x(1) < -5
    x(1) = -5;
else
    x(1) = x(1);
end
if x(2) > 5
    x(2) = 5;
elseif x(2) < -5
    x(2) = -5;
else
    x(2) = x(2);
end
x_sat = x;
end
```