VOANHKHA OPTIMIZATION TOOLBOX HISTORY LOG

**08 Aug 2017**  
Finish UNCONSTRAINED OPTIMIZATION (exact line search, backtracking line search methods, together with gradient descent, steepest descent, and Newton main algorithm).

|  |
| --- |
| %% Gradient Descent  clear all  f = 'exp(x(1)+3\*x(2)-0.1) + exp(x(1)-3\*x(2)-0.1) + exp(-x(1)-0.1);';  [xf1\_all, f1\_all] = Descent\_Kha(f, [-1; 1] ,'gradient' ,'exact');  [xf2\_all, f2\_all] = Descent\_Kha(f, [-1; 1] ,'gradient' ,'backtrack');  p\_opt = 2.559266696658216;  err1 = f1\_all - p\_opt;  err2 = f2\_all - p\_opt;  figure; plot(log10(err1)); hold on; plot(log10(err2)); |

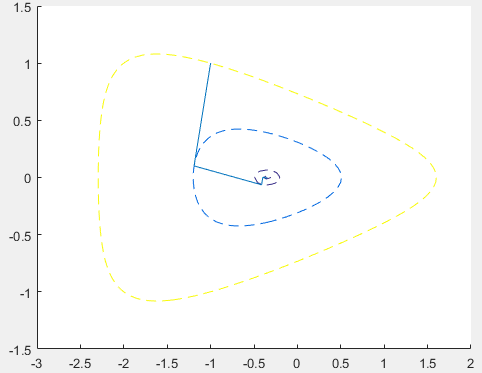
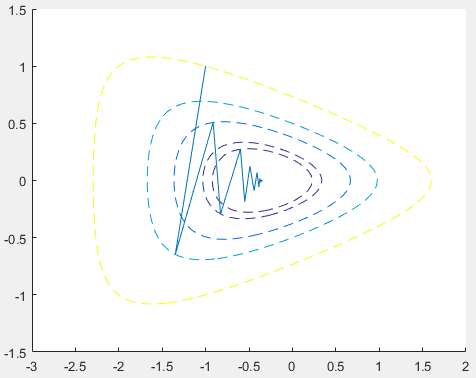
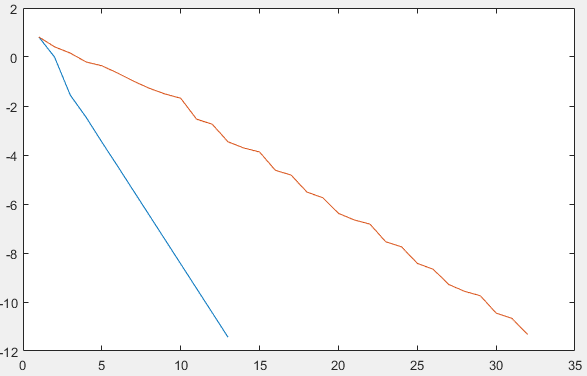


Fig 9.3 p.471 Fig 9.5 p.472

**Fig 9.4 p.471

|  |
| --- |
| %% Steepest Descent  [xf3\_all, f3\_all] = Descent\_Kha(f, [-1; 1] ,'steepest' ,'backtrack',[2 0; 0 8]);  [xf4\_all, f4\_all] = Descent\_Kha(f, [-1; 1] ,'steepest' ,'backtrack', [8 0;0 2]);  err3 = f3\_all - p\_opt;  err4 = f4\_all - p\_opt;  figure; plot(log10(err3)); hold on; plot(log10(err4)); |

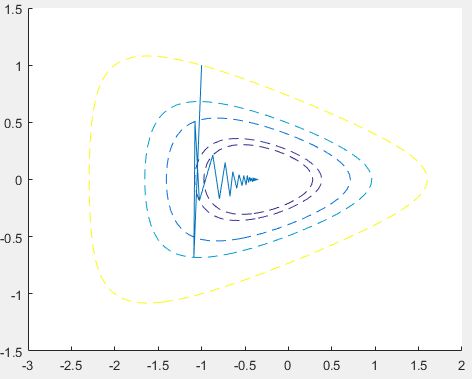
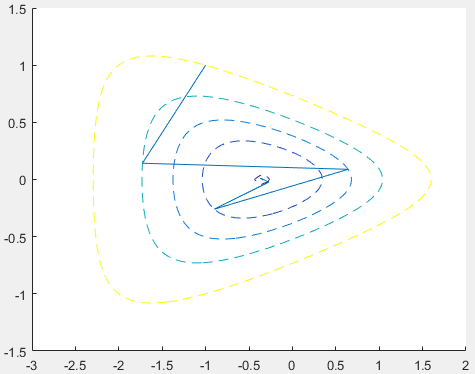


Fig 9.11 p.481 Fig 9.12 p.482

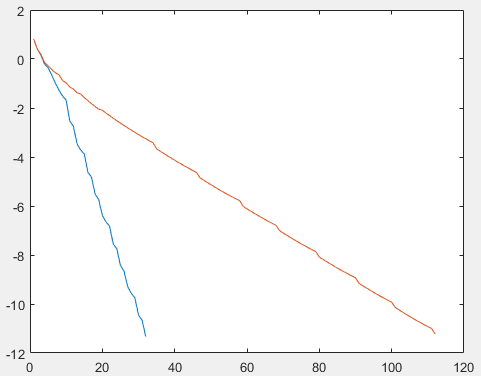
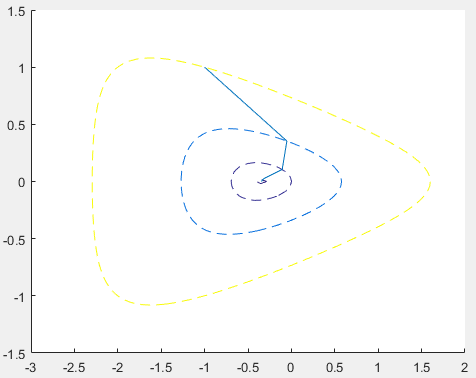
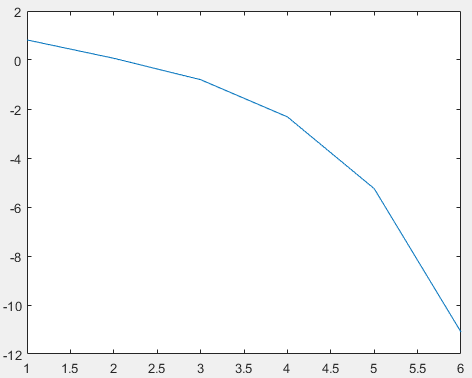


Fig 9.13 p.482

|  |
| --- |
| %% Newton, change epsilon to 1e-15 instead of 1e-10 for more accuracy  [xf5\_all, f5\_all] = Descent\_Kha(f, [-1; 1] ,'newton' ,'backtrack');  err5 = f5\_all - p\_opt;  figure; plot(log10(err5)); |

  
 Fig 9.19 p.481 Fig 9.20 p.493

FULL ALGORITHM

Main Algorithm

|  |
| --- |
| %% UNCONSTRAINED OPTIMISATION  % Version 1.0  % Author: Kha Vo (voanhkha@yahoo.com)    %% Instruction  % Example:  % f1 = 'exp(x(1)+3\*x(2)-0.1) + exp(x(1)-3\*x(2)-0.1) + exp(-x(1)-0.1);';  % [xf1\_all, f1\_all] = Descent\_Kha(f, [-1; 1] ,'gradient' ,'exact');    % FUNCTION EXAMPLES  % f = '1/2\*(x(1)^2 + 10\*x(2)^2);' % for Figure 9.2  % f = 'exp(x(1)+3\*x(2)-0.1) + exp(x(1)-3\*x(2)-0.1) + exp(-x(1)-0.1);' % for Figure 9.3  % f = 'norm(A\*x - b)^2;    % METHOD  % opt\_method = 'gradient' or 'steepest' or 'newton'  % linesearch\_method = 'exact' or 'backtrack'  % P = [8 0; 0 2]; % only for steepest descent    function [x\_all, f\_all] = Descent\_Kha(f, x0, opt\_method, linesearch\_method, P\_steepest)  %% TOLERANCE  epsilon = 1e-10;    %% PLOT OPTIONS  plotflag = 1;  n\_contours = 5; % number of first contours plotted  plot\_range = [-3 2 -1.5 1.5]; %[-3 3 -3 3] [-5 5 -12 12]    %% MAIN PROGRAM  if nargin < 5  P\_steepest = eye(size(x0));  end  if strcmp(opt\_method,'steepest')  P\_inv = inv(P\_steepest);  end    x = sym('x', [size(x0) 1]);  eval(strcat('f = ', f));  grad\_f = gradient(f,x); % calculate gradient of f as a function of x  hess\_f = hessian(f,x);  x\_value = x0; % initial point x0  f\_value = double(subs(f, x, x0)); % initial f(x0)  grad\_value = double(subs(grad\_f, x, x0));  x\_all = []; % store all values of x after each iteration of index k  f\_all = []; % store all values of f(x)      k = 0; % iteration count    while 1    switch opt\_method  case 'gradient'  Delta\_x = -grad\_value;  cri = norm(grad\_value)^2;  case 'steepest'  Delta\_x = -P\_inv\*grad\_value;  cri = norm(grad\_value)^2;  case 'newton'  hess\_value = double(subs(hess\_f, x, x\_value));  Delta\_x = -(hess\_value\grad\_value); % equivalent to -inv(hess\_value)\*grad\_value  lambda\_sq = grad\_value'\*(hess\_value\grad\_value);  cri = lambda\_sq/2;  end      % ------ Begin Stopping Criterion ----------%  if cri <= epsilon  break;  end  % -------- End Stopping Criterion ------------%  x\_all = [x\_all, x\_value];  f\_all = [f\_all, f\_value];  k = k + 1;    switch linesearch\_method  case 'backtrack'  t = BackTrack\_LineSearch(f, grad\_f, x\_value, Delta\_x);  case 'exact'  t = Exact\_LineSearch(f, grad\_f, x\_value, Delta\_x);  end    x\_value = x\_value + t\*Delta\_x;  f\_value = double(subs(f, x, x\_value));  grad\_value = double(subs(grad\_f, x, x\_value));    end        if plotflag == 1  figure  hold on  fcontour(f,plot\_range,'LevelList', f\_all(1:n\_contours), 'LineStyle', '--')  line(x\_all(1,:) , x\_all(2,:));  end |

Line search methods

|  |
| --- |
| function t = BackTrack\_LineSearch(f, grad\_f, x\_current, Delta\_x)  % Author: Kha Vo (voanhkha@yahoo.com)  % Using backtracking line search mentioned in Chapter 9, Convex  % Optimization book, Boyd.  alpha = 0.1;  beta = 0.7;  t = 1;  stopflag = 0;  x = sym('x', [size(x\_current) 1]);  while stopflag ~= 1;  if (double(subs(f, x, x\_current + t\*Delta\_x)) > double(subs(f, x, x\_current))...  + alpha\*t\*double(subs(grad\_f, x, x\_current))'\*Delta\_x)  t = beta\*t;  else stopflag = 1;  end  end  function t = Exact\_LineSearch(f, grad\_f, x\_value, Delta\_x)  % Author: Kha Vo (voanhkha@yahoo.com)  % Using Golden Section Method as mentioned in Appendix C,  % Nonlinear Programming book, Bertsekas.  tolerance = 1e-5;  tau = (3-sqrt(5))/2;  t = 1;  stopflag = 0;  x = sym('x', [size(x\_value) 1]);    % Determine first two ends (a1,a2) by searching for f'(a1)<0 and f'(a2)>0  a2 = 0;  dev\_a2 = -1;  while dev\_a2 < 0  a1 = a2;  a2 = a2 + 0.5;  dev\_a2 = double(subs(grad\_f, x, x\_value + a2\*Delta\_x))'\*Delta\_x;  end    % Golden Section Method  while stopflag ~= 1;  if a2-a1 > tolerance  b1 = a1 + tau\*(a2 - a1);  b2 = a2 - tau\*(a2 - a1);  ga1 = double(subs(f, x, x\_value + a1\*Delta\_x));  ga2 = double(subs(f, x, x\_value + a2\*Delta\_x));  gb1 = double(subs(f, x, x\_value + b1\*Delta\_x));  gb2 = double(subs(f, x, x\_value + b2\*Delta\_x));  if gb1 < gb2  if ga1 <= gb1  a2 = b1;  else  a2 = b2;  end  elseif gb1 > gb2  if gb2 >= ga2  a1 = b2;  else  a1 = b1;  end  else  a1 = b1; a2 = b2;  end  else stopflag = 1;  end    t = (a2+a1)/2;  end |