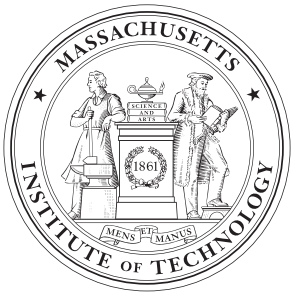
16.323 Optimal Control

PROBLEM SET 1

Due: Thursday, February 20, 2014



Author:
Daniel WIESE

Instructor:
Prof. Steven HALL

BFGS Solver

A BFGS solver was coded, including a line search routine. A script was used which called the BFGS function `bfgs.m`, which takes as an input a handle for the desired function and the initial starting point guess for the minimum. This function then calls the line search routine `linesearch.m` as necessary, where the Wolfe conditions are used to determine when the line search should stop.

(1) Rosenbrock Function

In this part, the Rosenbrock function was used to test and debug the functions introduced above.

$$f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$$

Figure 1 shows the search history as a line connecting points in the x_1, x_2 plane, over the region $-2 \leq x_1 \leq 2$, $-1 \leq x_2 \leq 3$ with the contours of the Rosenbrock function shown. Figure 2 shows the convergence history, the distance of each point x_k in the BFGS iteration from the optimal solution x^*, where the minimum for the Rosenbrock function can be found by inspection as $x_1 = 1$, $x_2 = 1$, with $f(1, 1) = 0$. About 50 iterations and 15,000 function calls were required, depending on how c_1 and c_2 were picked for the Wolfe conditions. The built in MATLAB function `fminunc.m` used 34 iterations and 150 function calls. The difference is most likely due to the bracketing portion of the line search algorithm. Bracketing requires using an arbitrary Δ to move along the search direction to bracket a minimum, before using bisection to more accurately find the location of the minimum. The bracketing procedure depended strongly on how Δ was chosen, and by not choosing it in a good way, many function calls were required to bracket the minimum for each iteration. This part of the code can definitely be improved before solving future problems.

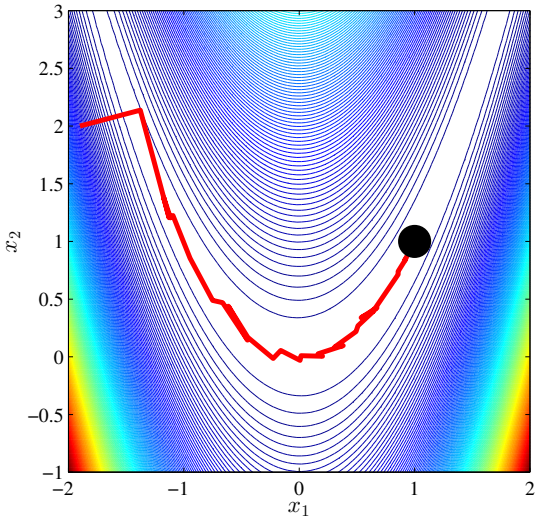


Figure 1: Search history

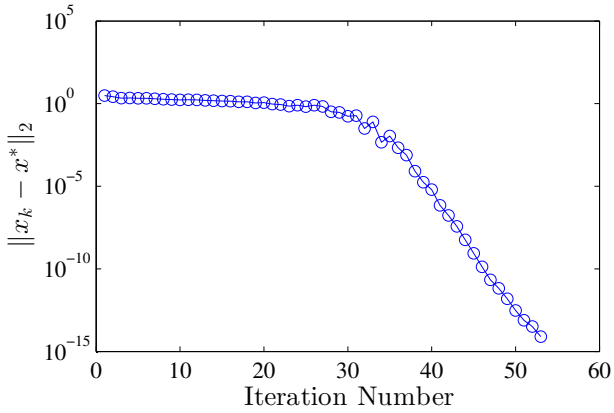


Figure 2: Convergence history

(2) Second Objective Function

In this problem, the following objective function was used instead of the Rosenbrock function.

$$f(x_1, x_2) = x_1^4 + 100x_2^4$$

The BFGS code was used, with the appropriate function handle passed to find the minimum. Figure 3 shows the search history plotted in the x_1, x_2 plane. Figure 4 shows the convergence history, the distance of each point x_k in the BFGS iteration from the optimal solution x^*. The optimal solution for this objective function can be found by inspection as $x_1 = 0$, $x_2 = 0$ with $f(0, 0) = 0$. From Figure 4 we can see that the convergence is linear. Additionally, this is expected due to the objective function having 4th power.

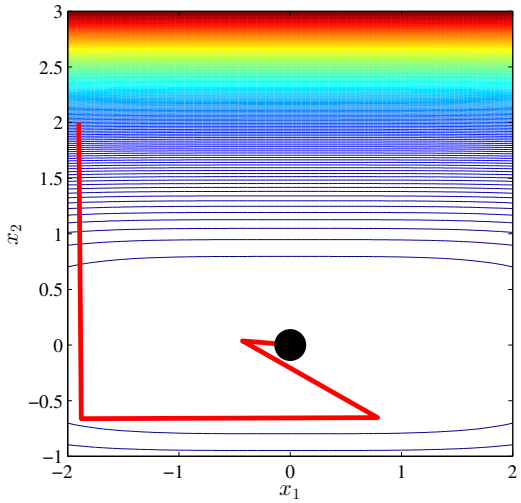


Figure 3: Search history

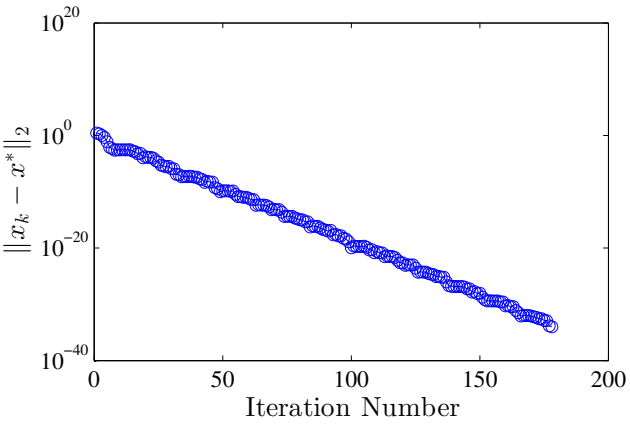


Figure 4: Search history

(3) Multiple Minima

For the following function

$$f(x_1, x_2) = 2x_1^2 - 1.05x_1^4 + \frac{1}{6}x_1^6 - x_1x_2 + x_2^2$$

From Figures 5-7 we can see the presence of three minima in the region $-3 \leq x_1 \leq 3$, $-3 \leq x_2 \leq 3$. The minima values are listed in the caption below each figure. There are two stationary points corresponding to the saddle points which can clearly be seen in Figures 5-7. The saddle points are at $(x_1, x_2) = (1.0706, 0.5353)$ and $(-1.0706, -0.5353)$ as evaluated analytically and solved the roots of a fifth order polynomial using MATLAB.

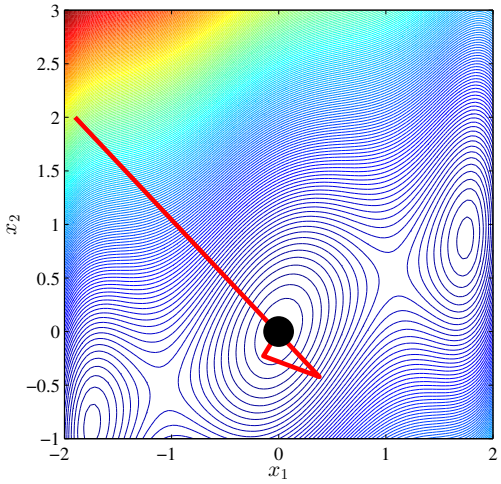


Figure 5: Search history starting from initial condition of $x_1 = -1.9$, $x_2 = 2$, converging to the minimum at x_1

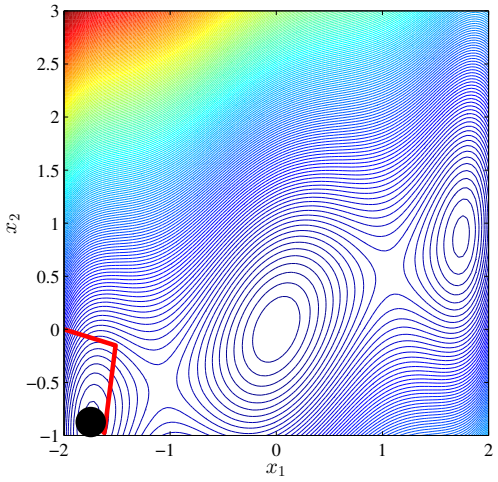


Figure 6: Search history starting from initial condition of $x_1 = -2$, $x_2 = 0$, converging to the minimum at $x_1 = -1.7476$, $x_2 = -0.8738$ where $f(-1.7476, -0.8738) = 0.2986$.

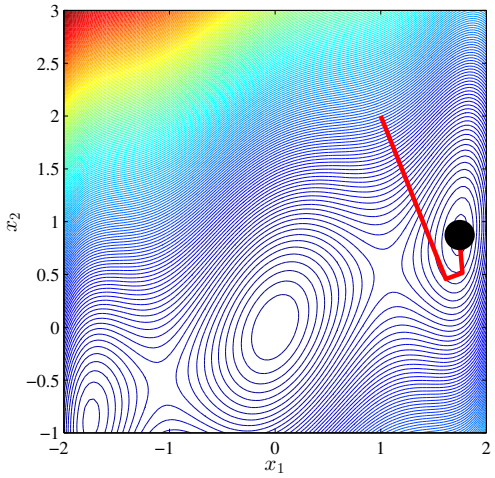


Figure 7: Search history starting from initial condition of $x_1 = 1$, $x_2 = 2$, converging to the minimum at $x_1 = 1.7476$, $x_2 = 0.8738$ where $f(1.7476, 0.8738) = 0.2986$.

Constrained Minimization

(4) Rectangle in Ellipse

For this problem we can solve it two ways. The first method is substitution, where the equality constraint is used to solve for one variable in terms of the other, and then this is substituted into the objective function. The objective function then depends on only one variable, and this function can be maximized by differentiating and setting the derivative to zero. Substitution works well for linear constraints, but is hard to generalize for larger systems and nonlinear constraints. So, The other way is with Lagrange multipliers. The problem is

$$\begin{aligned} \text{maximize} \quad & P(x,y) = 4(x+y) \\ \text{subject to} \quad & \frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \end{aligned}$$

Method 1: Substitution Solving for x from the constraint we have

$$x = \sqrt{a^2\left(1 - \frac{y^2}{b^2}\right)}$$

Plugging this into P we have

$$P = 4\left[a^2\left(1 - \frac{y^2}{b^2}\right)\right]^{\frac{1}{2}} + 4y$$

Differentiating

$$\frac{dP}{dy} = 2\left[a^2\left(1 - \frac{y^2}{b^2}\right)\right]^{-\frac{1}{2}}\left(2\frac{a^2}{b^2}y + 4\right) + 4$$

$\frac{dP}{dy} = 0$ implies

$$\boxed{y = \frac{b^2}{\sqrt{a^2+b^2}} \qquad \text{and} \qquad x = \frac{a^2}{\sqrt{a^2+b^2}}}$$

Method 2: Lagrange Multipliers

$$L(x,y,\lambda) = P(x,y) + \lambda c(x,y)$$

where

$$c(x,y) = \frac{x^2}{a^2} + \frac{y^2}{b^2} - 1 = 0$$

Evaluate

$$\frac{\partial L}{\partial x} = 4 + \frac{2\lambda}{a^2}x = 0$$
$$\frac{\partial L}{\partial y} = 4 + \frac{2\lambda}{b^2}y = 0$$
$$\frac{\partial L}{\partial \lambda} = \frac{x^2}{a^2} + \frac{y^2}{b^2} - 1 = 0$$

Solving this system of three equations and three unknowns we arrive at the same result as using the substitution method.

(5) Parallelpiped in Ellipsoid

$$\begin{aligned} &\text{maximize} && V(x,y,z) = 8xyz \\ &\text{subject to} && \frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} - 1 = 0 \end{aligned}$$

For this problem use Lagrange multipliers

$$L(x,y,z,\lambda) = 8xyz + \lambda\left(\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} - 1\right)$$

Evaluate

$$\frac{\partial L}{\partial x} = 8yz + \frac{2\lambda}{a^2}x = 0$$
$$\frac{\partial L}{\partial y} = 8xz + \frac{2\lambda}{b^2}y = 0$$
$$\frac{\partial L}{\partial z} = 8xy + \frac{2\lambda}{c^2}z = 0$$
$$\frac{\partial L}{\partial \lambda} = \frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} - 1 = 0$$

Solving the first three equations for λ and equating, we have

$$-\frac{4yza^2}{x} = -\frac{4xzb^2}{y} = -\frac{4xyc^2}{z}$$

giving

$$x^2 = \frac{a^2}{b^2}y^2 \qquad z^2 = \frac{c^2}{b^2}y^2 \qquad z^2 = \frac{c^2}{a^2}x^2$$

Plugging these into the last equation we obtain

$$\boxed{x = \frac{a}{\sqrt{3}} \qquad \text{and} \qquad y = \frac{b}{\sqrt{3}} \qquad \text{and} \qquad z = \frac{c}{\sqrt{3}}}$$

(6) Inequality Constraints

To solve an optimization problem with inequality constraints, we essentially only need to take into account the active constraints. To do this, we multiply the partial derivatives of the Lagrangian with respect to each multiplier by the corresponding multiplier, as shown below. We then consider all the different cases where $\lambda_i = 0$ indicating the constraint is inactive and $\lambda_j \neq 0$ indicating the constraint is active. By multiplying by the extra factor of λ, the equations will still be solvable regardless of whether each λ_i is zero or not, either because the constraint is inactive, or because the constraint is satisfied.

$$\begin{aligned} f(x,y) &= x^2 + y^2 - 6xy - 4x - 5y \\ c_1(x,y) &= y + (x-2)^2 - 4 \leq 0 \\ c_2(x,y) &= 1 - x - y \leq 0 \end{aligned}$$

Form the Lagrangian $L(x, y, \lambda_1, \lambda_2) = f(x, y) + \lambda_1 c_1(x, y) - \lambda_2 c_2(x, y)$ giving

$$L(x, y, \lambda_1, \lambda_2) = x^2 + y^2 - 6xy - 4x - 5y + \lambda_1 \left[y + (x-2)^2 - 4\right] + \lambda_2 \left[1 - x - y\right]$$

Form necessary conditions

$$\begin{aligned} \frac{\partial L}{\partial x} &= 2x - 6y - 4 + 2\lambda_1(x-2) - \lambda_2 = 0 \\ \frac{\partial L}{\partial y} &= 2y - 6x - 5 + \lambda_1 - \lambda_2 = 0 \\ \lambda_1 \frac{\partial L}{\partial \lambda_1} &= \lambda_1 \left[y + (x-2)^2 - 4\right] = 0 \\ \lambda_2 \frac{\partial L}{\partial \lambda_2} &= \lambda_2(1 - x - y) = 0 \end{aligned}$$

Now we need to consider the various options in terms of which constraints are active/inactive.

Both constraints inactive: $\boldsymbol{\lambda_1 = \lambda_2 = 0}$ Substituting these values of λ into the above equations we get

$$\begin{aligned} 2x - 6y - 4 &= 0 \\ 2y - 6x - 5 &= 0 \end{aligned}$$

which gives the solution $y = -\frac{17}{16}$, $x = \frac{5}{8}$. Plugging these values back into the constraints, we can see that the constraints are not satisfied, so this is not a valid option.

Inactive: $\lambda_1 = 0$, active: $\lambda_2 \neq 0$ Next we try making λ_2 active, giving the following equations.

$$\begin{aligned} 2x - 6y - 4 - \lambda_2 &= 0 \\ 2y - 6x - 5 - \lambda_2 &= 0 \\ \lambda_2(1 - x - y) &= 0 \end{aligned}$$

From the third equation we have $x = 1 - y$ and substituting that into the first two equations we have

$$\begin{aligned} 2(1 - y) - 6y - 4 - \lambda_2 &= 0 \\ 2y - 6(1 - y) - 5 - \lambda_2 &= 0 \end{aligned}$$

which can be solved yielding $y = \frac{9}{16}$, $x = \frac{7}{16}$ and $\lambda_2 = -\frac{13}{2}$. Checking the constraints, we can see that they are both satisfied. So this point is a valid solution, but we aren't yet sure whether this is actually the minimum until we try the other combinations of active and inactive constraints. Evaluating the function here, we have $f\left(\frac{7}{16}, \frac{9}{16}\right) = -5.53$.

Active: $\lambda_1 \neq 0$, inactive: $\lambda_2 = 0$

$$\begin{aligned} 2x - 6y - 4 + 2\lambda_1(x - 2) &= 0 \\ 2y - 6x - 5 + \lambda_1 &= 0 \\ \lambda_1 \left[y + (x - 2)^2 - 4\right] &= 0 \end{aligned}$$

From the third equation we can solve $y = 4 - (x - 2)^2$ and plug into the first two giving

$$\begin{aligned} 2x - 6\left[4 - (x - 2)^2\right] - 4 + 2\lambda_1(x - 2) &= 0 \\ 2\left[4 - (x - 2)^2\right] - 6x - 5 + \lambda_1 &= 0 \end{aligned}$$

From the second equation solve for λ_1 as

$$\lambda_1 = 6x + 5 - 2\left[4 - (x - 2)^2\right]$$

Then substitute this into the first equation

$$2x - 6\left[4 - (x - 2)^2\right] - 4 + 2\left(6x + 5 - 2\left[4 - (x - 2)^2\right]\right)(x - 2) = 0$$

and simplify

$$
\begin{aligned}
2x - 6\left[4 - (x-2)^2\right] - 4 + 2\left(6x + 5 - 2\left[4 - (x-2)^2\right]\right)(x-2) &= 0 \\
2x - 6\left[4 - x^2 + 4x - 4\right] - 4 + 2\left(6x + 5 - 8 + 2x^2 - 8x + 8\right)(x-2) &= 0 \\
2x + 6x^2 - 24x - 4 + (-4x + 10 + 4x^2)(x-2) &= 0 \\
6x^2 - 22x - 4 - 4x^2 + 10x + 4x^3 + 8x - 20 - 8x^2 &= 0 \\
4x^3 - 6x^2 - 4x - 24 &= 0
\end{aligned}
$$

Solving this equation numerically we have $x = 2.6962$, $y = 3.5153$, $f(2.6962, 3.5153) = -65.6023$ and $\lambda_1 = 14.1469$.

Both active: $\boldsymbol{\lambda_1 \neq 0}$, $\boldsymbol{\lambda_2 \neq 0}$ In this case we use the fourth and fifth equations to find the equation $x^2 - 5x + 1 = 0$, and $y = 1 - x$. Solving these equations we get four different possible solutions, none of which is a minimum.

Minimum

After checking all of the cases for the constraints being active or inactive, we found the minimum is given by the following

$$
\boxed{
\begin{aligned}
x &= 2.6962 \\
y &= 3.5153 \\
f &= -65.6023 \\
\lambda_1 &= 14.1469
\end{aligned}
}
$$

Sensitivity

At the minimum found above, we can rewrite the constraint $c_1(x, y) \leq 0$ as $\bar{c}_1(x, y) \leq C$ where $C = 4$. If C is changed, the change in the minimum is given by $\frac{\partial f}{\partial C} = -\lambda_1$. So, if $C = 4.1$ then we have $\frac{\partial f}{\partial C} = -14.1469$. To compute the total change we compute

$$
\begin{aligned}
\Delta f &= \frac{\partial f}{\partial c_1}\Delta c_1 \\
&= -\lambda_1 \Delta c_1 \\
&= -14.1469(0.1) \\
&- 1.41469
\end{aligned}
$$

So, if the cost function is changed as shown, then the minimum that can be achieved becomes smaller. So the new minimum would be

$$
\boxed{f(x, y) = -67.02}
$$

Writing a Script to Confirm Results

A script was written which uses the MATLAB function `fmincon.m` to solve the constrained minimization, yielding the same results as determined above. The code can be found in the appendix.

(7) Bryson Paper Summary

The calculus of variations is the foundation upon which optimal control theory was built. The calculus of variations deals with optimizing a function $J(y)$ over the set of admissible functions $y(x)$. The calculus of variations started with Pierre de Fermat. Galileo Galilei posed two problems which were later solved using CV.John Bernoulli and Leibniz used Fermat's ideas to solve one of Galileo's problems. Isaac Newton is credited with inventing CV in 1685. Leonard Euler provided the beginnings of a real theory of CV, and Lagrange invented the method of Lagrange multipliers that we use today. During the early to mid 1800s, Gustav Jacob Jacobi and William Rowan Hamilton were working on what would later become the Hamilton-Jacobi-Bellman equation, developed by Bellman in the mid 1900s. Also during the mid 1800s, Karl Wilhelm Theodor Weierstrass was also working on CV on a more rigorous basis, on which Bellman and Pontryagin built in the 1900s.

In 1960s, this theory had led to optimal control as it is known at the undergraduate level today. Kalman used a quadratic function to penalize state variables and control inputs, and used CV to derive an optimal control law which turned out to be linear constant gain feedback of the state variables, which was later renamed by Athans as LQR control. Furthermore, this idea was extended to the estimator problem, giving rise to the LQE, where the expected value of the integral quadratic performance index in the presence of Gaussian white noise. Combining LQE and LQR resulted in the LQG compensator, which is widely used today. Also around this time nonlinear programming was also being developed, whose successful application was crucially dependent on the invention of the digital computer. Dynamic programming was an extension of the work of Hamilton and Jacobi, also being developed in the 1950s, and the Euler-Lagrange equations of CV can also be derived using dynamic programming.

This theory was finally able to be applied to practical applications using numerical solutions and the computer, such as optimal trajectory generation for fighter jets, as well as for developing logic used in the Apollo 11 moon landing in 1969. Much work at this time by people such as Bryson and Ross was done to develop numerical algorithms. Also at this time, Kalman and McFarlane were some of the people working with Riccati equations to solve the LQ optimal control problem, resulting in the elegant, efficient method we know today. During the 1970s much focus was turned toward robustification of the optimal control methods, and development of H_∞ control theory.

Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dan Wiese
% 16.323 - HW#1
% master_hw1_v2.m
%--------------------------------------------------------------------------
% Master script
% This script defines the function handles for the different minimization
% problems, and contains the desired starting point. The BFGS solver bfgs_v1.m
% is called by passing it a function handle and initial guess. The state
% history, number of iterations, and number of function calls are returned and
% plotted.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
close all
clc;
%Make function handle for the desired function
fhand=@functionpart1_rosen;
% fhand=@functionpart2_objective;
% fhand=@functionpart3_multiple;
%Initial starting point
x0=[-1.9,2]';
% x0=[-2,0]';
%Call the BFGS solver which will return the points leading up to the minimum
[x,ni,nf]=bfgs_v1(fhand,x0);
disp(nf)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT THE RESULTS: STATE HISTORIES
%Define plot limits
x1min=-2;
x1max=2;
x2min=-1;
x2max=3;
%Generate linspace over which to plot
x1=linspace(x1min,x1max,100);
x2=linspace(x2min,x2max,100);
%Evaluate the function value over plot space
for ii=1:length(x1)
    for jj=1:length(x2)
        [func(ii,jj),¬]=fhand([x1(ii),x2(jj)]);
    end
end
%Make plot vector from path
% xplot=[x0'; x];
xplot=x;

figure(1)
contour(x1,x2,func',200)
hold on
plot(xplot(:,1),xplot(:,2),'linewidth',3,'color',[1 0 0])
plot(xplot(end,1),xplot(end,2),'.','MarkerSize',60,'Color',[0,0,0]);
xlim([x1min, x1max]);
```

```
ylim([x2min, x2max]);
xlabel('$x_{1}$','interpreter','latex','FontSize',14)
ylabel('$x_{2}$','interpreter','latex','FontSize',14)
set(gca,'fontsize',12);
set(gca,'fontname','times');
set(gcf,'Units','pixels');
set(gcf,'PaperUnits','inches','PaperPosition',[0 0 5 5]);
set(gcf,'PaperPositionMode','manual')
set(gcf,'InvertHardCopy','off');
set(gcf,'color',[1 1 1])
daspect([1 1 1]);
% print('-depsc','../Figures/part1_rosen_state.eps');
% print('-depsc','../Figures/part2_objective_state.eps');
% print('-depsc','../Figures/part3_multiple_state_ic2.eps');

%% PLOT THE RESULTS: CONVERGENCE HISTORIES

xstarrosen=[1,1];
for jj=1:length(x)
    errorrosen(jj)=norm(x(jj,:)-xstarrosen,2);
end

xstarobjective2=[0,0];
for jj=1:length(x)
    errorobjective2(jj)=norm(x(jj,:)-xstarobjective2,2);
end

% figure(2)
% semilogy(errorrosen,'o-')
% xlabel('Iteration Number','interpreter','latex','FontSize',14)
% ylabel('$\|x_{k}-x^{*}\|_{2}$','interpreter','latex','FontSize',14)
% set(gca,'fontsize',12);
% set(gca,'fontname','times');
% set(gcf,'Units','pixels');
% set(gcf,'PaperUnits','inches','PaperPosition',[0 0 5 3]);
% set(gcf,'PaperPositionMode','manual')
% set(gcf,'InvertHardCopy','off');
% set(gcf,'color',[1 1 1])
% print('-depsc','../Figures/part1_rosen_convergence.eps');

% figure(2)
% semilogy(errorobjective2,'o-')
% xlabel('Iteration Number','interpreter','latex','FontSize',14)
% ylabel('$\|x_{k}-x^{*}\|_{2}$','interpreter','latex','FontSize',14)
% set(gca,'fontsize',12);
% set(gca,'fontname','times');
% set(gcf,'Units','pixels');
% set(gcf,'PaperUnits','inches','PaperPosition',[0 0 5 3]);
% set(gcf,'PaperPositionMode','manual')
% set(gcf,'InvertHardCopy','off');
% set(gcf,'color',[1 1 1])
% print('-depsc','../Figures/part2_objective2_convergence.eps');

% figure(3)
% surf(x1,x2,func')
% hold on
```

```
% plot(x(:,1),x(:,2))

%% EVAULUATE SOME STUFF

% [theminimimum,¬]=fhand(x(end,:)');
% [X,FVAL,exitflag,output]=fminunc(fhand,x0)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dan Wiese
% 16.323 - HW#1
% bfgs.m
%-------------------------------------------------------------------------------
% BFGS solver
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [xpath,NI,NF]=bfgs_v1(fhand,x0)
%(1) Set initial guess for Hessian and initialize stuff
[n,temp]=size(x0);
H0=eye(n,n);
Hk=H0;
xk=x0;
xpath=[x0'];
gtol=10^-20;
normg=1;
NI=0;
NF=0;
global ni nf
ni=0;
nf=0;

while normg>gtol
    [¬,gk]=fhand(xk);
%      nf=nf+1;

    %(2) At each k, set search direction
    dk=-inv(Hk)*gk;
    dk=dk/norm(dk,n);

    %(3) Do a line search along search direction
    [alphak,xk1,¬,gk1,nfinc]=linesearch_v7(fhand,xk,dk);
%      nf=nf+1;

    %(4) Set sk change in x and yk change in gradient
    sk=alphak*dk;
    yk=gk1-gk;

    %(5) Update our guess for the Hessian
    Hk1=Hk+((yk*yk')/(yk'*sk))-((Hk*sk*sk'*Hk)/(sk'*Hk*sk));

    %Set everything to updated values
    Hk=Hk1;
    xk=xk1;
    gk=gk1;
```

```

    %Calculate norm of gradient
    normg=norm(gk,2);
    disp(normg)

    %Store current iteration of x value
    xpath=[xpath; xk(:)'];

    %Incremement iteration counter ni and output that as fval
    NI=NI+1;
end
NF=nf;
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dan Wiese
% 16.323 - HW#1
% linesearch_v7.m
%-------------------------------------------------------------------------------
% Linesearch
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [alpha,xk1,fk1,gk1,NFinc]=linesearch_v7(fhand,xk,dk)
NFinc=0;
% global nf
% nf=nf+1;
%-------------------------------------------------------------------------------
%Preliminary stuff and constants
c1=0.00001;
c2=0.9;
Δ=10;
wolfecountermax=10;
endnow=0;
wolfe=0;
jj=1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
while endnow==0 && wolfe==0;
    %BRACKETING
    %Preallocate stuff for bracketing procedure
    clearvars a fxk
    [fxk,¬]=fhand(xk);
    a(1)=0;
    Δ=Δ⋆1⋆abs(rand^(1/2));
%     disp(Δ)
    foundbracket=0;
    fa=fxk;
    ii=1;
    %Go along alpha taking Delta steps until a bracket is found
    while foundbracket==0;
    Delta=Δ⋆2^(ii-1);
    a(ii+1)=a(ii)+Delta;
    [fap1,¬]=fhand(xk+a(ii+1)⋆dk);
        if fap1≤fa
            foundbracket=0;
        else
```

```
            if ii==1
                lambdal=a(ii);
                lambdau=a(ii+1);
            else
                lambdal=a(ii-1);
                lambdau=a(ii+1);
            end
            foundbracket=1;
        end
    ii=ii+1;
    fa=fap1;
    end
%     nf=nf+1;
    %-------------------------------------------------------------------------
    %BISECTION SEARCH
    %Pick a point in middle of bracket
    alpha=(lambdal+lambdau)/2;
    %Preallocate for bisection search
    wolfecounter=0;
    armijo=0;
    curvature=0;
    % Iterate the bisection until wolfe conditions are met, or it reaches max
    % tries
    while armijo==0 && curvature==0 && wolfecounter<wolfecountermax;
    %Evaluate the gradient at the middle point
    [¬,gk]=fhand(xk+alpha*dk);
        %If slope is positive, minimum is left of the middle point
        if gk'*dk>0
            lambdau=alpha;
            alphanew=(lambdal+lambdau)/2;
        %If slope is negative, minimum is right of the middle point
        elseif gk'*dk<0
            lambdal=alpha;
            alphanew=(lambdal+lambdau)/2;
        %If slope is zero, we have found stationary point
        elseif gk'*dk==0
            alphanew=(lambdal+lambdau)/2;
            endnow=1;
        end
        [fkpadk,gkpadk]=fhand(xk+alphanew*dk);
        alpha=alphanew;
        %Armijo Condition
        if fkpadk>fxk+c1*alpha*dk'*gk;
            armijo=0;
        else
            armijo=1;
        end
        %Curvature Condition
        if c2*dk'*gk>dk'*gkpadk
            curvature=0;
        else
            curvature=1;
        end
    wolfecounter=wolfecounter+1;
    end
%     nf=nf+1;
```

```
    if armijo==1 && curvature==1
        wolfe=1;
    else
        wolfe=0;
    end
    jj=jj+1;
end

% disp(armijo)
% disp(curvature)
%-------------------------------------------------------------------------------
fk1=fkpadk;
gk1=gkpadk;
xk1=xk+alpha*dk;
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dan Wiese
% 16.323 - HW#1
%-------------------------------------------------------------------------------
% Rosenbrock function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[f,g]=functionpart1_rosen(x)

global nf
nf=nf+1;

x1=x(1);
x2=x(2);

if nargout==1
    f=(1-x1)^2+100*(x2-x1^2)^2;
else if nargout==2
    f=(1-x1)^2+100*(x2-x1^2)^2;
    dfdx1=-2*(1-x1)-400*(x2-x1^2)*x1;
    dfdx2=200*(x2-x1^2);
    g=[dfdx1, dfdx2]';
else
    error('how many output arguements are you using?')
end

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dan Wiese
% 16.323 - HW#1
%-------------------------------------------------------------------------------
% Rosenbrock function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[f,g]=functionpart2(x)

global nf
nf=nf+1;
```

```
x1=x(1);
x2=x(2);

if nargout==1
    f=x1^4+100*x2^4;
else if nargout==2
    f=x1^4+100*x2^4;
    dfdx1=4*x1^3;
    dfdx2=400*x2^3;
    g=[dfdx1, dfdx2]';
else
    error('how many output arguements are you using?')
end

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dan Wiese
% 16.323 - HW#1
%-------------------------------------------------------------------------------
% Function given in problem 3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[f,g]=functionpart3_multiple(x)

global nf
nf=nf+1;

x1=x(1);
x2=x(2);

if nargout==1
    f=2*x1^2-1.05*x1^4+(1/6)*x1^6-x1*x2+x2^2;
else if nargout==2
    f=2*x1^2-1.05*x1^4+(1/6)*x1^6-x1*x2+x2^2;
    dfdx1=4*x1-4.2*x1^3+x1^5-x2;
    dfdx2=-x1+2*x2;
    g=[dfdx1, dfdx2]';
else
    error('how many output arguements are you using?')
end

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dan Wiese
% 16.323 - HW#1
% problem6_master.m
%-------------------------------------------------------------------------------
% Problem 6 Master Script
% This script sets up the various constraints for a constrained minimization
% problem, defines an initial guess, and then runs fmincon.m to solve the
problem. The objective function and inequality constraints are stored in
```

```
% separate files and called using function handles.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
close all
clc;

fhand=@problem6_function;
conhand=@problem6_constraints;

A=[];
B=[];
Aeq=[];
Beq=[];
LB=[];
UB=[];

X0=[3,3];
[X,FVAL,EXITFLAG,OUTPUT]=fmincon(fhand,X0,A,B,Aeq,Beq,LB,UB,conhand)

% X=[7/16,9/16];
% [f,¬]=fhand(X)
% bb=roots([4 -6 -4 -24])
% x=bb(1);
% y=4-(x-2)^2;
% X=[x y];
% [f,¬]=fhand(X)
% lambda1=5+6*x-2*y
% xx=(5+sqrt(21))/2
% X1=[4.7912,-3.7913];
% X2=[4.7912,0.7913];
% X3=[0.2087,-3.7913];
% X4=[0.2087,0.7913];
% [f,¬]=fhand(X4)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dan Wiese
% 16.323 - HW#1
% problem6_function.m
%-------------------------------------------------------------------------------
% This is the objective function for the minimization problem with inequality
% constraints.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function f=problem6_function(X)
x=X(1);
y=X(2);
f=x^2+y^2-6*x*y-4*x-5*y;
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dan Wiese
```

```
% 16.323 - HW#1
% problem6_constraints.m
%-------------------------------------------------------------------------
% This function has the inequality constraints.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [C,Ceq]=problem6_constraints(X)
x=X(1);
y=X(2);
c1=y+(x-2)^(2)-4.1;
c2=1-x-y;
C=[c1,c2];
Ceq=[];
end
```