1130-EMARO-MSA-2008 - 2022L

EXERCISES ON OPTIMIZATION TECHNIQUES

Nwachukwu Anthony
Doctoral School nr.3
Warsaw University of Technology

Abstract

This material contains the exercises on optimization techniques for 1130-EMARO-MSA-2008-2022L. AMPL and MATLAB were used for both linear and non-linear programming problems.

Contents

Listings

Chapter 1

Course Organization

This exercise class is a compliment of the main lecture. There will be 2 projects in this exercise part of the course, one on linear programming with AMPL and the other on non-linear programming with MATLAB. Each of them is 20 marks. During the classes there will be ungraded exercises to help the student practice.

Chapter 2

AMPL and Linear Programming

AMPL is an algebraic modeling language used for linear and nonlinear optimization problems and was developed at Bell Laboratories [1]. The AMPL system supports the entire optimization modeling lifecycle — formulation, testing, deployment, and maintenance, AMPL can provide the head start you need to successfully implement large-scale optimization projects. It can be embedded into C++, C#, Java, MATLAB, Python, and R application using AMPL APIs.

2.1 INSTALLATION

To install ampl in your PC, follow these steps:

1. Go to *www.ampl.com*
2. Navigate to Download a Free Demo
3. Based on your OS, proceed to *AMPL IDE download for Windows*, *AMPL IDE download for Linux*, *AMPL IDE download for macOS* or their equivalent command line downloads.
4. Extract the tar files *amplide.mswin64* for windows users, *ampl.linux64* for linux users, *amplide.macosx64* for mac users or their equivalent 32 bits files
5. To run the IDE for windows, inside your AMPL folder, double-click the amplide folder icon to open that folder, and then double-click the amplide.exe file icon to start the AMPL IDE application

6. To run the command line for linux, cd into your AMPL directory, and type *./ampl*, then proceed to run ampl commands
7. For mac users and a more detailed instructions check the official website *https://ampl.com/try-ampl/download-a-free-demo/*

NOTE:

For more information on obtaining AMPL, check out [2]

2.2 ENVIRONMENT AND BASIC SYNTAX

2.2.1 Basic Files

The basic files necessary to run an AMPL code are:

1. **.mod** - used to declare the elements of the models: variables, objective, constraints and data (sets and parameters).
2. **.dat** - used to define the data for the model.
3. **.run** - where variable configurations are defined, "scripting constructs," such as reading tables or data bases.

2.2.2 AMPL IDE

The AMPL IDE is divided into 3 sections:

1. **Current Directory** - aka current folder. It shows the current folder you are working on. You can copy the path to your AMPL code file and paste it in the search path.
2. **Console** - aka command window. AMPL code can be directly executed here or an AMPL *.run* file can be called here
3. **Editor** - AMPL files can be written here and saved. It could be *.dat*, *.mod*, *.run* or any other files

NOTE:

For detailed information on integrating AMPL into other application like Python, MATLAB, JAVA and more via AMPL API, check out [3]

https://www.youtube.com/watch?v=jJHo88QRkUM https://www.youtube.com/watch?v=ZN1cqSyD8pg https://www.youtube.com/watch?v=HNpBRpaPmGk

2.2.3 Syntax

1. Variable: var VariableName;
2. Objective: minimize or maximize ObjectiveName: . . . ;
3. Constraint: subject to RestrictionName: . . . ;

2.2.4 Remarks

1. Every line instruction must be terminated with ";".
2. Line comments are preceded by the symbol "#".
3. Block commands are enclosed by the symbols "//. . . //".
4. AMPL is "case-sensitive".
5. Variable names must be unique.

2.3 BASIC EXAMPLE

We consider a simple linear programming example

$$\begin{aligned} \max z = & 250x_1 + 180x_2 \\ s.t \quad & 4x_1 + x_2 \leq 6 \\ & 3x_1 + 2x_2 \leq 7 \\ & x_i \geq 0 \ (i = 1,2) \end{aligned} \tag{2.1}$$

The above problem can be modelled in AMPL thus:

```
# DECISION VARIABLES
var x1 >= 0;
var x2 >= 0;

# OBJECTIVE FUNCTION
maximize z: 250*x1 + 180*x2;

# CONSTRAINTS
s.t. M1: 4*x1 + x2 <= 6;
s.t. M2: 3*x1 + 2*x2 <= 7;
```

Listing 2.1: basic.mod

```
# RESET AMPL ENVIRONMENT
reset;

# LOAD MODEL
model basic.mod;

# ASSIGN SOLVER
option solver cplex;

# SOLVE
solve;

# DISPLAY RESULTS
display x1, x2, z;
```

Listing 2.2: basic.run

```
ampl: include basic.run
```

Listing 2.3: Console

2.4 SEPARATING MODEL AND DATA

We now consider a more complex problem. Linear programming can be represented in a standard form thus,

$$\begin{aligned} max(or\,min) \quad z &= c_1x_1 + \ldots + c_nx_n \\ s.t \quad & a_{11} + \ldots + a_{1n}x_n \quad \geq (=, or \leq) \quad b_1 \\ & a_{21} + \ldots + a_{2n}x_n \quad \geq (=, or \leq) \quad b_2 \\ & \ldots \\ & a_{m1} + \ldots + a_{mn}x_n \quad \geq (=, or \leq) \quad b_m \\ & x_j \quad \geq \quad (=, or \leq) \quad 0, \quad j = 1, \ldots, n \end{aligned} \tag{2.2}$$

Problem 2.2 can be written in compact form thus,

$$\begin{aligned} max(or\,min) \quad z &= \sum_{j=1}^{n} c_j x_j \\ s.t \quad & \sum_{j=1}^{n} a_{ij}x_j \quad \geq (=, or \leq) \quad b_i \\ & x_j \quad \geq \quad (=, or \leq) \quad 0, \quad j = 1, \ldots, n \end{aligned} \tag{2.3}$$

Consider this linear programming problem lpp,

$$\begin{aligned} \max z = & 20x_1 + 10x_2 + 15x_3 + 45x_4 \\ s.t \quad & 150x_1 + 235x_2 + 125x_3 + 380x_4 \geq 435 \\ & 5x_1 + x_2 \geq 6 \\ & 3x_1 + 5x_2 + 5x_3 + 7x_4 \geq 10 \\ & 2x_1 + 3x_2 + 4x_3 + 9x_4 \geq 5 \\ & x_j \geq 0 \ (j = 1, 2, 3, 4) \end{aligned} \tag{2.4}$$

The above problem can be modelled in AMPL thus:

```
param n;
param m;
set J := {1..n}; #set of decision variables
set I := {1..m}; #set of constraints

param C {J}; #objective function coefficients
param A {I,J}; #constraint coefficients matrix
param B {J}; #right hand sides of constraints

var X {J} >= 0 ; #decision variables

minimize z: sum {j in J} C[j] * X[j];

s.t. Constraint {i in I}:
    sum {j in J} A[i,j] * X[j] >= B[i];
#include complex.run
```

Listing 2.4: complex.mod

```
data;
param n := 4;
param m := 4;

param C :=
    1 20
    2 10
    3 15
    4 45;

param A: 1  2   3   4 :=
        1   150 235 125 380
        2   5   1   0   0
        3   3   5   5   7
        4   2   3   4   9;

param   B:=
        1   435
        2   6
        3   10
        4   5;
```

Listing 2.5: complex.dat

```
# RESET AMPL ENVIRONMENT
reset;

# LOAD MODEL
model complex.mod;

# LOAD DATA
model complex.dat;

# DISPLAY PROBLEM FORMULATION
expand z, Constraint;

# ASSIGN SOLVER
option solver cplex;

# SOLVE
solve;

# DISPLAY RESULTS
display X, z;
```

Listing 2.6: complex.run

```
ampl: include basic.run
```

Listing 2.7: Console

2.5 INTEGER AND MIXED INTEGER PROGRAMMING

In AMPL variables are considered real numbers by default. In order to consider integer or binary variables we need to specify the keyword *integer* or *binary* in the declaration of the variables immediately before the ending semi-colon. Consider problem 2.4, replace the variables requirement from real to integers as shown below,

$$\begin{aligned} \max z = & 20x_1 + 10x_2 + 15x_3 + 45x_4 \\ s.t \quad & 150x_1 + 235x_2 + 125x_3 + 380x_4 \geq 435 \\ & 5x_1 + x_2 \geq 6 \\ & 3x_1 + 5x_2 + 5x_3 + 7x_4 \geq 10 \\ & 2x_1 + 3x_2 + 4x_3 + 9x_4 \geq 5 \\ & x_j \quad integer \ (j = 1,2,3,4) \end{aligned} \tag{2.5}$$

The above problem can be modelled in AMPL thus:

```
param n;
param m;
set J := {1..n} #set of decision variables
set I := {1..m} #set of constraints

param C {J}; #objective function coefficients
param A {I,J}; #constraint coefficients matrix
param B {J}; #right hand sides of constraints

var X {J} >= 0 integer; #decision variables (change integer to binary
   for binary variables)

minimize z: sum {j in J} C[j] * X[j];

s.t. Constraint {i in I}:
    sum {j in J} A[i,j] * X[j] >= B[i];
```

Listing 2.8: complex-integer.mod

```
data;
param n := 4;
param m := 4;

param C :=
    1 20
    2 10
    3 15
    4 45;

param A: 1  2   3   4 :=
```

```
        1   150 235 125 380
        2   5   1   0   0
        3   3   5   5   7
        4   2   3   4   9;

param   B:=
        1   435
        2   6
        3   10
        4   5;
```

Listing 2.9: complex.dat

```
# RESET AMPL ENVIRONMENT
reset;

# LOAD MODEL
model complex.mod;

# LOAD DATA
model complex.dat;

# DISPLAY PROBLEM FORMULATION
expand z, Constraint;

# ASSIGN SOLVER
option solver cplex;

# SOLVE
solve;

# DISPLAY RESULTS
display X, z;
```

Listing 2.10: complex.run

```
ampl: include basic.run
```

Listing 2.11: Console

For detailed and more examples consult the material from AMPL [4]

Chapter 3

MATLAB and Linear Programming

3.1 INTRODUCTION

3.1.1 Origin of Matlab

The name MATLAB stands for MATrix LABoratory. MATLAB was written originally to provide easy access to matrix software developed by the LINPACK (linear system package) and EISPACK (Eigen system package) projects. The software package has been commercially available since 1984 and is now considered a standard tool in most universities and industries worldwide. It has powerful built-in routines that enable a very wide variety of computations.

3.1.2 Applications of Matlab

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, simulation and programming environment.

3.1.3 Programming

MATLAB is a modern programming language environment. It has sophisticated data structures, contains built-in editing and debugging tools, and supports object-oriented program-

ming. These factors make MATLAB an excellent tool for teaching and research.

3.1.4 Computation

MATLAB has many advantages compared to conventional computer languages (e.g., C, FORTRAN) for solving technical problems. MATLAB is an interactive system whose basic data element is an array that does not require dimensioning.

3.1.5 Visualization

It also has easy to use graphics commands that make the visualization of results immediately available.

3.1.6 Simulation and other toolboxes

Specific applications are collected in packages referred to as toolbox. There are toolboxes for signal processing, symbolic computation, control theory, simulation, optimization, and several other fields of applied science and engineering.

3.2 Environment and Basic Syntax

3.2.1 Desktop Basics

When you start MATLAB, the desktop appears in its default layout as seen in figure 3.1.

The desktop includes these panels:

1. Current Folder — Access your files.
2. Command Window — Enter commands at the command line, indicated by the prompt (»).
3. Workspace — Explore data that you create or import from files.

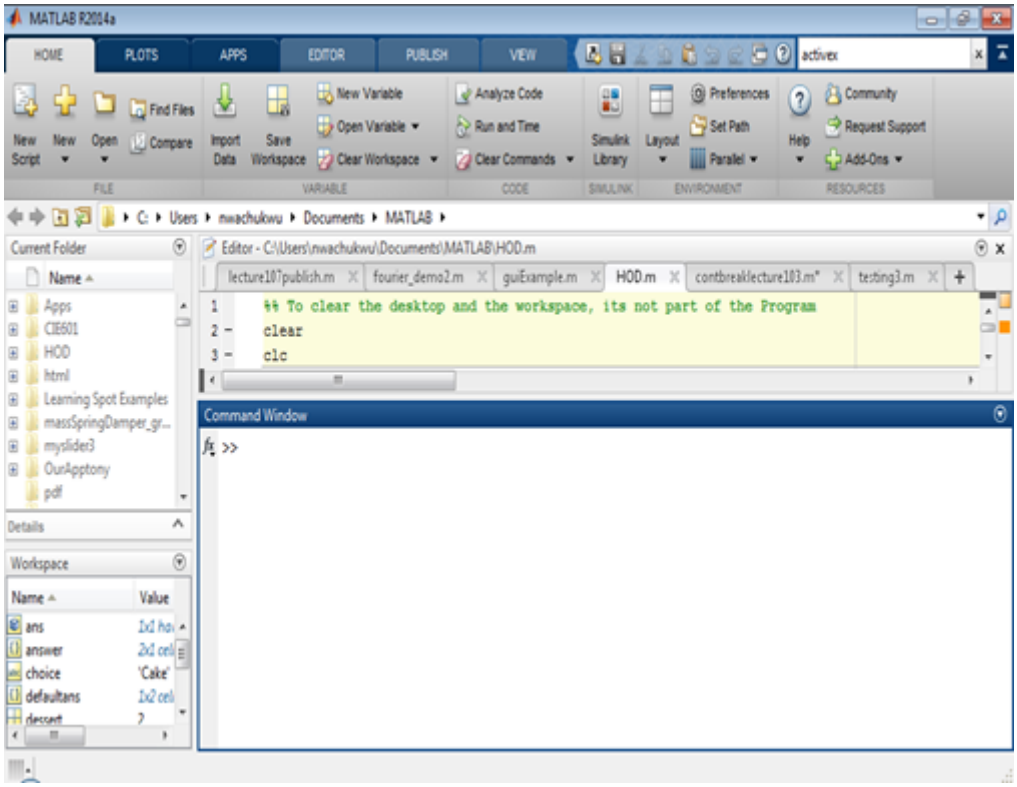

Figure 3.1: MATLAB Desktop

As you work in MATLAB, you issue commands that create variables and call functions. For example, create a variable named “x” by typing this statement at the command line:

```
» x = 23
```

Create more variables. When you end a command with semicolon (;), the answer will not be displayed but it will be stored in the workspace. You can use it for computation. Example is shown below.

On the Matlab desktop you see several other icons like Plots, Apps, (?) etc. These are short cuts to several Matlab functions. You will get to use some of them as we proceed with the lectures.

```
» a = 34, b = 54; c = 67;
a =
34
» t = a + b
t =
88
```

3.2.2 Finding Help

To get detailed information regarding a command you know, use **help**, while if you don't know the command but don't but have a clue on what it is, use **lookfor**

```
» help format
» lookfor differentiation
```

3.2.3 Controlling The Appearance Of Floating Point Number

Format determines how numbers are displayed on your command window. The default setting is format short. To get detailed information regarding the various types of formats available in Matlab, run the help command as shown below

```
» help format
```

The following commands and more were displayed: format short, format long, format bank, format rat, format shorte, format longe, format shortg, format longg, format shorteng, format longeng, format hex, format +, format compact and format loose. Let's examine the first four. Others are left for you to try out.

```
» format short
» pi
ans =
3.1416
» format long
» pi
ans =
3.141592653589793
» format bank
» pi
ans =
3.14
» format rat
» pi
ans =
355/113
```

format short: Scaled fixed point format with 5 digits.
format long: Scaled fixed point format with 15 digits for double and 7 digits for single.
format bank: This displays the answer in 2 decimal places.
format rat: This displays the answer in numerator/denominator style.

Note that format does not affect how MATLAB computations are done. Computations on float variables namely single or double, are done in appropriate floating point precision, no matter how those variables are displayed.

3.2.4 Variable Precision Arithmetic

R = vpa(e, n) numerically evaluates the element e using variable precision floating point arithmetic with n decimal digit accuracy.

NB: It is advisable to use this instead R = vpa('e', n). The former will first compute e to the default precision then use the result generated to compute R. To compute R directly without approximation error, enclose e in quotation. Example

```
» vpa(pi+1,18)
ans =
4.14159265358979312
» vpa('pi+1',18)
ans =
4.14159265358979324
» x = pi+1; vpa(x,18)
ans =
4.14159265358979312
» y = sym('pi+1'); vpa(y,18)
ans =
4.14159265358979324
```

Note the following:
vpa(pi+1,18) gives different result from vpa('pi+1',18).
vpa(pi+1,18) first computes pi+1 then uses the approximated result to compute the vpa. This is seen in x = pi+1; vpa(x,18).
vpa('pi+1',18) goes ahead to compute vpa using the exact value of pi+1, then finds the vpa. This is evident in y = sym('pi+1'); vpa(y,18).
sym('a') is a command used in evaluating the exact value of a (that is without approximation).

3.2.5 Managing The Workspace And Command Window

Cleaning the workspace

To clear the variables defined in the workspace, you use this command clear or clear all.

```
» clear x z
» clear
» clear all
```

Cleaning the Command window To clear everything on the command window, just type this command clc

```
» clc
```

Aborting a Running Command

If you issue out a command which refused to end or taking time to complete and you wish to end it, you have to hold the Ctrl and the C keys on your keyboard. This will stop further computation.

```
» x = 3, while x > 2, x=x+1, end
```

NB: The command in the example above will never end. To stop it, hold the Ctrl and C keys of your keyboard down at the same time (that is Ctrl + C).

Line continuation Suppose while writing command you decided to move to the next line maybe to allow neat printing. You will have to use the line continuation command so that Matlab will be able to treat the second line as continuation of the first. The command is ...

```
» x = [3, 4, 5; ...
   6, 3, 4; 8, 2, 4]
x =
   3   4   5
   6   3   4
   8   2   4
```

Viewing contents of the workspace You can view the contents of the workspace using whos.

```
» whos
```

Saving and Loading Workspace Variables Workspace variables do not persist after you exit MATLAB. Saving preserves the workspace in your current working folder in a compressed file with a .mat extension, called a MAT-file. Save your data for later use with the save command,

```
» save filename.mat
```

Restore data from a MAT-file into the workspace using load.

```
» load filename.mat
```

3.3 BASIC MATHEMATICAL OPERATIONS

3.3.1 Predefined Matlab Constants

Figure 3.2 shows the constants you will be working with in Matlab during your computations.

`pi`	The π number, $\pi = 3.14159\ldots$
`i,j`	The imaginary unit i, $\sqrt{-1}$
`Inf`	The infinity, ∞
`NaN`	Not a number

Figure 3.2: Predefined Constants

3.3.2 Elementary Functions in Matlab

Table 3.3 shows the list of some of the functions that are already defined in Matlab.

`cos(x)`	Cosine	`abs(x)`	Absolute value
`sin(x)`	Sine	`sign(x)`	Signum function
`tan(x)`	Tangent	`max(x)`	Maximum value
`acos(x)`	Arc cosine	`min(x)`	Minimum value
`asin(x)`	Arc sine	`ceil(x)`	Round towards $+\infty$
`atan(x)`	Arc tangent	`floor(x)`	Round towards $-\infty$
`exp(x)`	Exponential	`round(x)`	Round to nearest integer
`sqrt(x)`	Square root	`rem(x)`	Remainder after division
`log(x)`	Natural logarithm	`angle(x)`	Phase angle
`log10(x)`	Common logarithm	`conj(x)`	Complex conjugate

Figure 3.3: Elementary Functions

3.3.3 Arithmetic Operators and Special Characters in Matlab

Table 3.4 shows the arithmetic operators and special characters you will need in computations. You need to understand that Matlab obeys the law of BODMAS, that is Bracket first, followed by Of, then Division and Multiplication, finally Addition and Subtraction.

Character	Description
+	Addition
−	Subtraction
*	Multiplication (scalar and array)
/	Division (right)
^	Power or exponentiation
:	Colon; creates vectors with equally spaced elements
;	Semi-colon; suppresses display; ends row in array
,	Comma; separates array subscripts
...	Continuation of lines
%	Percent; denotes a comment; specifies output format
'	Single quote; creates string; specifies matrix transpose
=	Assignment operator
( )	Parentheses; encloses elements of arrays and input arguments
[ ]	Brackets; encloses matrix elements and output arguments

Figure 3.4: Arithmetic Operators and Special Characters

Examples

$$\frac{34cos^{-1}\pi}{105+3} - \frac{7^3}{2} \times \frac{|-52|}{3!} \tag{3.1}$$

```
» 34 * acos(pi) / (105 + 3) – 7 Λ  3 / 2 * abs(-52) / factorial(3)
ans =
-1.4863e+03 + 5.7030e-01i
```

By default, Matlab computes trigonometry functions in radian. If you wish to compute in degree, you will have to add **d**, that immediately after the trig function, **acosd(pi)**

$$e^{-\infty} + \sqrt{49} - \sin 30 + \ln 100 - \log 100 \tag{3.2}$$

```
» exp(-inf) + sqrt(49) – sind(30) + log(100) – log10(1000)
ans =
8.1052
```

3.4 ENTERING MATRICES AND VECTORS

A matrix is an array of numbers. To type a matrix into MATLAB you must

1. begin with a square bracket, [
2. separate elements in a row with spaces or commas (,)
3. use a semicolon (;) to separate rows
4. end the matrix with another square bracket,].

```
» A = [1 2 3; 0 5 6; 7 8 9]
A =
   1    2    3
   0    5    6
   7    8    9

» syms a b c d e g f

» B = [a b c; d e 6; g -f 0]
B =
   [ a,    b,    c]
   [ d,    e,    6]
   [ g,    -f,    0]

» b = [5, 8, 3]
b =
   5    8    3

» c = [4; -7; 0]
c =
   4
   -7
   0
```

3.5 CREATE 2-D PLOTS

Figure 3.5 shows some plots one can make using MATLAB

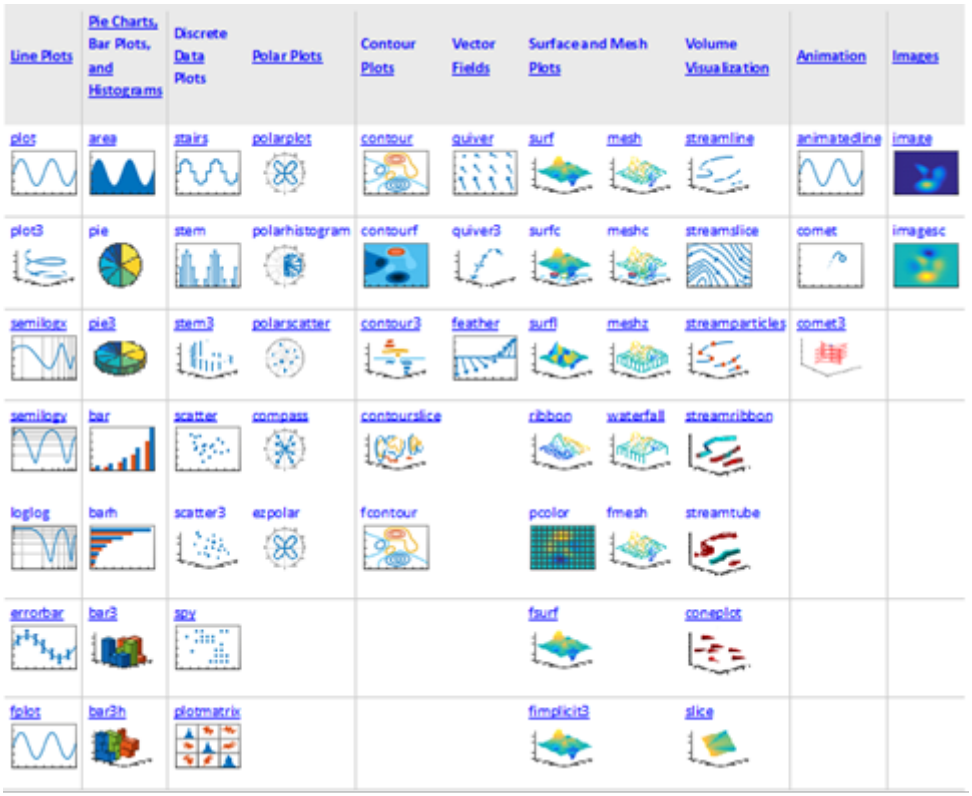

Figure 3.5: MATLAB Plots

3.5.1 Creating a 2-D Line Graph

The example below and figure 3.6 show how to create a simple line graph. Use the linspace function to define x as a vector of 100 linearly spaced values between 0 and 2π. Use figure command to create a new figure and plot command for plotting the function.

```
» x = linspace(0,2*pi,100);
» y = sin(x);
» plot(x,y)
```

Let's demonstrate how to plot more than one line by passing multiple x, y, and z to the plot function as shown in figure 3.7

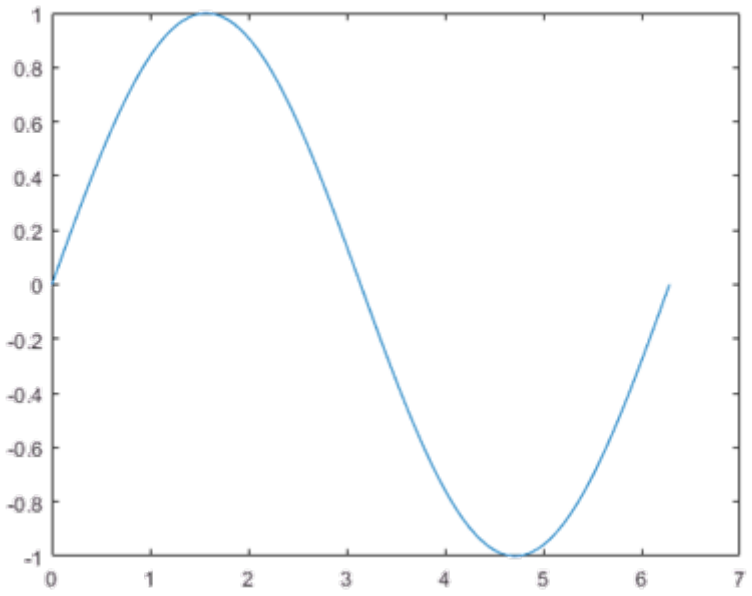

Figure 3.6: 2D Plot

```
» x = linspace(0,2*pi,100);
» y = [0.9093 -0.7568 -0.2794 0.9894 -0.5440];
» z = [-0.4161 -0.6536 0.9602 -0.1455 -0.8391];
» plot(x, y, x, z)
```

3.5.2 Specify Line Style, Color, and Markers

```
» x = linspace(0,2*pi,25);
» y1 = sin(x);
» y2 = sin(x-pi/4);
» figure
» plot(x,y1,'–go',x,y2,':r*')
```

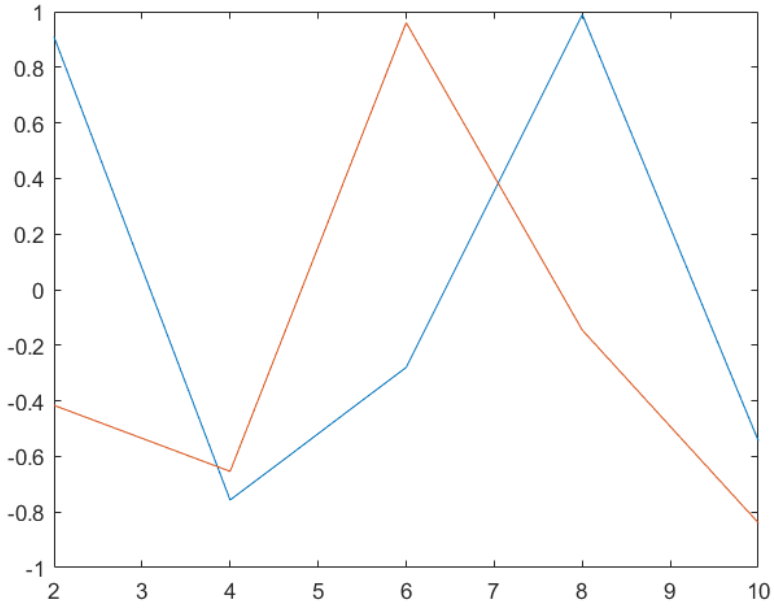

Figure 3.7: Multiple Plots

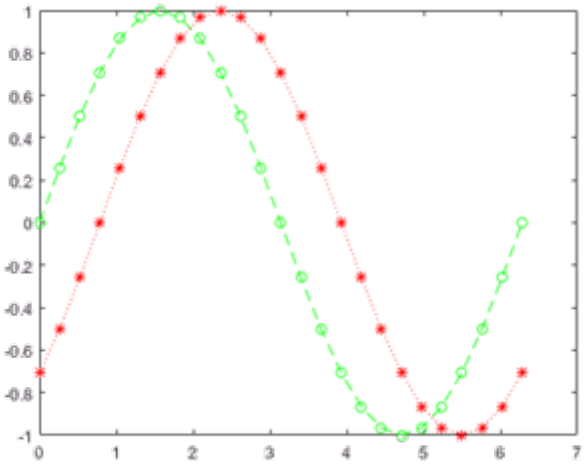

Figure 3.8: Specify Line Style, Color, and Markers

3.5.3 Adding Title, Axis Labels, And Legend

```
» title('Graph of Sine and Cosine Between -2\pi and 2 \pi')
» xlabel('-2 \pi < x < 2 \pi')
» ylabel('sine and cosine values')
» legend('y = sin(x)','y = cos(x)')
```

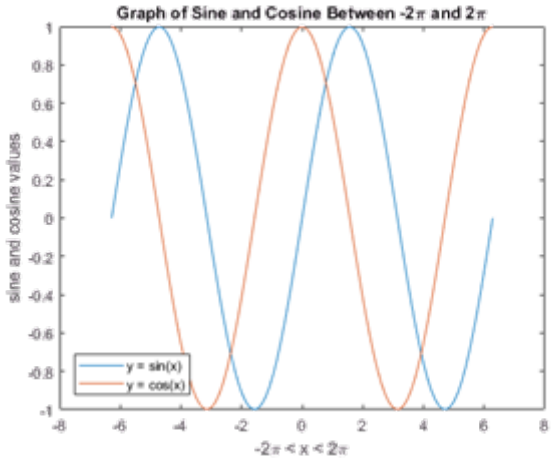

Figure 3.9: Adding Title, Axis Labels, And Legend

3.6 LINEAR PROGRAMMING

Consider a linear programming problem of this type,

$$\min_x \; f^T x \quad s.t \quad \begin{cases} A.x \leq b \\ A_{eq}.x = b_{eq} \\ l_b \leq x \leq u_b \end{cases} \tag{3.3}$$

f, x, b, b_{eq}, l_b and u_b are vectors, and A and A_{eq} are matrices. MATLAB provides a function, *linprog* that solves problems of type 3.3. The syntax is,

$$\begin{gathered} x = linprog(f, A, b) \\ [x, fval, exitflag, output] = linprog(f, A, b, Aeq, beq, lb, ub, options) \end{gathered} \tag{3.4}$$

Exit flag 1 implies function converged to a solution x while 2 implies no feasible point was found.The available algorithms are 'dual-simplex' (default), 'interior-point-legacy', and 'interior-point'. For full description of the various exit flags, options which include different available algorithms check out [5]. Consider this linear programming problem lpp,

$$\begin{aligned} \min z = & 20x_1 + 10x_2 + 15x_3 + 45x_4 \\ s.t \quad & 150x_1 + 235x_2 + 125x_3 + 380x_4 \geq 435 \\ & 5x_1 + x_2 \geq 6 \\ & 3x_1 + 5x_2 + 5x_3 + 7x_4 \geq 10 \\ & 2x_1 + 3x_2 + 4x_3 + 9x_4 \geq 5 \\ & x_j \geq 0 \ (j = 1,2,3,4) \end{aligned} \tag{3.5}$$

Problem 3.5 can be solved in MATLAB thus

```
% clear workspace and command window
clear
clc

% enter data
f = [20, 10, 15, 45];
A = -[150, 235, 125, 380; 5, 1, 0, 0; 3, 5, 5, 7; 2, 3, 4, 9];
b = -[435, 6, 10, 5];
Aeq = [];
beq = [];
lb = [0, 0, 0, 0]; %or zeros(length(f),1);
ub = [];

% specify options
options = optimoptions('linprog','Algorithm','dual-simplex', 'Display',
    'iter');

% solve the problem
[x,fval, flag, output] = linprog(f,A,b,Aeq,beq,lb,ub,options)
```

Listing 3.1: lppmatlab.m

3.7 MIXED-INTEGER PROGRAMMING

Consider an integer linear programming problem of this type,

$$\min_x f^T x \quad s.t \quad \begin{cases} x(intcon) \text{ are integers} \\ A.x \leq b \\ A_{eq}.x = b_{eq} \\ l_b \leq x \leq u_b \end{cases} \tag{3.6}$$

f, x, b, b_{eq}, l_b and u_b are vectors, and A and A_{eq} are matrices. MATLAB provides a function, *intlinprog* that solves problems of type 3.6. The syntax is,

$$\begin{gathered} x = intlinprog(f, intcon, A, b) \\ [x, fval, exitflag, output] = intlinprog(f, A, b, Aeq, beq, lb, ub, x0, options) \end{gathered} \tag{3.7}$$

The available algorithms are 'dual-simplex' and 'primal-simplex'. For full description of the various exit flags, options which include different available algorithms check out [6].

3.7.1 Integer Problem

Consider this integer linear programming problem,

$$\begin{aligned} \max z = & 20x_1 + 10x_2 + 15x_3 + 45x_4 \\ s.t \quad & 150x_1 + 235x_2 + 125x_3 + 380x_4 \geq 435 \\ & 5x_1 + x_2 \geq 6 \\ & 3x_1 + 5x_2 + 5x_3 + 7x_4 \geq 10 \\ & 2x_1 + 3x_2 + 4x_3 + 9x_4 \geq 5 \\ & x_j \quad integer \ (j = 1, 2, 3, 4) \end{aligned} \tag{3.8}$$

Problem 3.8 can be solved in MATLAB thus

```
% clear workspace and command window
clear
clc

% enter data
f = [20, 10, 15, 45];
A = -[150, 235, 125, 380; 5, 1, 0, 0; 3, 5, 5, 7; 2, 3, 4, 9];
b = -[435, 6, 10, 5];
Aeq = [];
beq = [];
lb = [0, 0, 0, 0]; %or zeros(length(f),1);
ub = [];
intcon = [1,2,3,4];
x0=zeros(length(f),1);

% specify options
options = optimoptions('intlinprog','RootLPAlgorithm','dual-simplex', '
    Display','iter');

% solve the problem
[x,fval, flag, output] = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub,x0,
    options)
```

Listing 3.2: lpp_int_matlab.m

3.7.2 Mixed Integer, Binary Problem

Consider yet another this integer linear programming problem but this time we want to force x to be binary,

$$\begin{aligned} \min z = & 20x_1 + 10x_2 + 15x_3 + 45x_4 \\ s.t \quad & 150x_1 + 235x_2 + 125x_3 + 380x_4 \geq 435 \\ & 5x_1 + x_2 \geq 6 \\ & 3x_1 + 5x_2 + 5x_3 + 7x_4 \geq 10 \\ & 2x_1 + 3x_2 + 4x_3 + 9x_4 \geq 5 \\ & x_j \geq 0 \ (j = 1,3,4) \\ & x_2 \quad binary \end{aligned} \tag{3.9}$$

Problem 3.9 can be solved in MATLAB thus

```
% clear workspace and command window
clear
clc

% enter data
f = [20, 10, 15, 45];
A = -[150, 235, 125, 380; 5, 1, 0, 0; 3, 5, 5, 7; 2, 3, 4, 9];
b = -[435, 6, 10, 5];
Aeq = [];
beq = [];
lb = [0, 0, 0, 0]; %or zeros(length(f),1);
ub = [inf, 1, inf, inf]; % enforces x(2) is binary
intcon = 2;
x0=zeros(length(f),1);

% specify options
options = optimoptions('intlinprog','RootLPAlgorithm','dual-simplex', '
    Display','iter');

% solve the problem
[x,fval, flag, output] = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub,x0,
    options)
```

Listing 3.3: lpp_binary_matlab.m

Chapter 4

The Simplex Method

When given a LP problem, you will have to take it through the processes below in order to solve the problem,

1. standard form
2. slack variable
3. simplex algorithm

4.1 STANDARD FORM

For a LP program to be in standard form, the following conditions must be satisfied,

1. Objective function must be maximization
   E.g.
   $$\min 4x_1 + x_2 + x_3 \quad \rightarrow \quad \max \ -4x_1 - x_2 - x_3$$
2. Variables must be non-negative
   E.g.
   $$x_1 - x_2 = 6$$
   $$x_1 + 2x_3 \geq 24$$

$$x_2 \geq 0$$

$$x_3 \leq 0$$

$$x_1 \quad unbounded$$

The variable x_3 is non-positive. To met the constraint of all variables being non-negative:

$$x_3 = x_3' - x_3'' \leq 0$$

$$x_1 = x_1' - x_1''$$

$$x_1', x_1'', x_3', x_3'' \geq 0$$

Following the previous example and adjusting these changes:

$$\max \ -4(x_1' - x_1'') - x_2 - (x_3' - x_3'')$$

$$= -4x_1' + 4x_1'' - x_2 - x_3' + x_3''$$

$$x_1' - x_1'' x_2 = 6$$

$$x_1' - x_1'' + 2x_3' - 2x_3'' \geq 24$$

$$x_2 \geq 0$$

$$x_1', x_1'', x_3', x_3'' \geq 0$$

3. Equality in constraints should be converted to 2 inequality constraints
   E.g.

   $$2x_1 + x_2 = 5$$

   would be converted to:

   $$2x_1 + x_2 \leq 5$$

   $$2x_1 + x_2 \geq 5$$

4. Put inequalities to correct form; functions ≤ constraints

   $$\mathrm{a}x_1 + \mathrm{b}x_2 \leq \mathrm{c}$$

E.g.

$$\min x_1 + 3x_2 + x_3$$

$$\text{s.t. } x_1 - x_3 = 6$$

$$2x_1 + x_2 \geq 24$$

$$x_2 \geq 0$$

$$x_3 \leq 0$$

Following the steps that have been described:

1) $\max -x_1 - 3x_2 - x_3$

2) $x_3 = x_3' - x_3''$ and $x_1 = x_1' - x_1''$

3) $x_1 - x_3 \geq 6$ and $x_1 - x_3 \leq 6$

Putting all together:

$$\max -x_1' + x_1'' - 3x_2 - x_3' + x_3''$$

$$\text{s.t. } x_1' - x_1'' - x_3' + x_3'' \leq 6$$

$$\text{s.t. } x_1' - x_1'' - x_3' + x_3'' \leq -6$$

$$-2x_1 - x_2 \leq 24$$

4.2 SLACK FORM

For inequality constraints of the form,

$$\mathrm{a}x_1 + \mathrm{b}x_2 \leq \mathrm{c}$$

we introduce a slack variable s thus,

$$\mathrm{s} = \mathrm{c} - \left(\mathrm{a}x_1 + \mathrm{b}x_2\right)$$

E.g. in standard form we have:

$$\max 3x + 2y$$

$$x + y \leq 7$$

$$2x - y \leq 2$$

$$x, y \geq 0$$

In slack form we have:

$$z = 3x + 2y$$

$$s_1 = 7 - (x + y)$$

$$s_2 = 2 - (2x - y)$$

z	Objective function variables
x, y	non-basic variables
s1, s2	basic variables

4.3 SIMPLEX ALGORITHM

The following steps should be done continuously while there are no positive coefficient in the objective function.

1. Check if there is a non-negative coefficient in the objective function. If yes, continue to 2, else stop.
2. Set all non-basic variables to zero
3. Select a non-basic variable with a positive coefficient in the objective function
4. Increase the variable as much as possible without increasing non-negative constraints
5. Find the tightest constraint
6. Switch roles of basic and non-basic variables

4.4 EXAMPLE

$$\begin{aligned} \max z = & 6x_1 + 5x_2 + 4x_3 \\ s.t \quad & 2x_1 + 1x_2 + 1x_3 \leq 18 \\ & x_1 + 2x_2 + 2x_3 \leq 30 \\ & 2x_1 + 2x_2 + 2x_3 \leq 24 \\ & x_j \geq 0 \ (j = 1,2,3) \end{aligned} \tag{4.1}$$

Introducing slack variables we obtain

$$\begin{aligned} z &= 0 + 6x_1 + 5x_2 + 4x_3 \\ s_1 &= 18 - 2x_1 - x_2 - x_3 \\ s_2 &= 30 - x_1 - 2x_2 - 2x_3 \\ s_3 &= 24 - 2x_1 - 2x_2 - 2x_3 \\ x_1, x_2, x_3, s_1, s_2, s_3 &>= 0 \end{aligned} \tag{4.2}$$

Pivot 1

1. We select x_1 since it's coefficient is positive in the objective function
2. x_1 can be increased to maximum of $9, 30, 12$
3. s_1 has the tightest constraints
4. Rearrange s_1 such that $x_1 = 9 - 0.5s_1 - 0.5x_2 - 0.5x_3$. By substitution and simplification we obtain

$$\begin{aligned} z &= 54 - 3s_1 + 2x_2 + x_3 \\ x_1 &= 9 - 0.5s_1 - 0.5x_2 - 0.5x_3 \\ s_2 &= 21 + 0.5s_1 - 1.5x_2 - 1.5x_3 \\ s_3 &= 6 + s_1 - x_2 - x_3 \\ x_1, x_2, x_3, s_1, s_2, s_3 &>= 0 \end{aligned} \tag{4.3}$$

Pivot 2

1. We select x_2 since it's coefficient is positive in the objective function
2. x_2 can be increased to maximum of $18, 14, 6$
3. s_3 has the tightest constraints
4. Rearrange s_3 such that $x_2 = 6 + s_1 - s_3 - x_3$. By substitution and simplification we obtain

$$\begin{aligned} z &= 66 - s_1 - 2s_3 - x_3 \\ x_1 &= 6 - s_1 + 0.5s_3 \\ s_2 &= 12 - s_1 - 1.5s_3 \\ x_2 &= 6 + s_1 - s_3 - x_3 \\ x_1, x_2, x_3, s_1, s_2, s_3 &>= 0 \end{aligned} \tag{4.4}$$

Since there is no positive non-basic variables in the objective function, we can not optimize the function any further.

The solution is

$$(x_1, x_2, x_3, s_1, s_2, s_3, Z) = (6, 6, 0, 0, 12, 0, 66) \tag{4.5}$$

4.5 EXAMPLE

$$\begin{aligned} \max z = & 3x_1 + 2x_2 \\ s.t \quad & x_1 + x_2 \leq 7 \\ & 2x_1 - x_2 \leq 2 \\ & x_j \geq 0 \ (j = 1, 2) \end{aligned} \tag{4.6}$$

Introducing slack variables we obtain

$$\begin{aligned} z &= 0 + 3x_1 + 2x_2 \\ s_1 &= 7 - x_1 - x_2 \\ s_2 &= 2 - 2x_1 + x_2 \\ x_1, x_2, s_1, s_2 &>= 0 \end{aligned} \tag{4.7}$$

Pivot 1

1. We select x_1 since it's coefficient is positive in the objective function
2. x_1 can be increased to maximum of $7,1$
3. s_2 has the tightest constraints
4. Rearrange s_1 such that $x_1 = 1 - 0.5s_2 + 0.5x_2$. By substitution and simplification we obtain

$$\begin{aligned} z &= 3 - 1.5s_2 + 3.5x_2 \\ s_1 &= 6 + 0.5s_2 - 1.5x_2 \\ x_1 &= 1 - 0.5s_2 + 0.5x_2 \\ x_1, x_2, s_1, s_2 &>= 0 \end{aligned} \tag{4.8}$$

Pivot 2

1. We select x_2 since it's coefficient is positive in the objective function
2. x_2 can be increased to maximum of $4,-2$
3. s_1 has the tightest constraints
4. Rearrange s_1 such that $x_2 = 4 - 0.67s_1 + 0.33x_2$. By substitution and simplification we obtain

$$\begin{aligned} z &= 17 - 2.33s_1 - 0.33x_2 \\ x_2 &= 4 - 0.67s_1 + 0.33x_2 \\ x_1 &= 3 - 0.33s_1 - 0.33x_2 \\ x_1, x_2, s_1, s_2 &>= 0 \end{aligned} \tag{4.9}$$

Since there is no positive non-basic variables in the objective function, we can not optimize the function any further.

The solution is

$$(x_1, x_2, s_1, s_2, Z) = (3, 4, 0, 0, 17) \tag{4.10}$$

Chapter 5

MATLAB and Non-Linear Programming

5.1 Quadratic Programming

Consider a quadratic programming problem of this type,

$$\min_x \frac{1}{2}x^T Hx + f^T x \quad s.t \quad \begin{cases} A.x \leq b \\ A_{eq}.x = b_{eq} \\ l_b \leq x \leq u_b \end{cases} \tag{5.1}$$

f, x, b, b_{eq}, l_b and u_b are vectors, and A and A_{eq} are matrices. MATLAB provides a function, *quadprog* that solves problems of type 5.1. The syntax is,

$$\begin{aligned} x &= quadprog(H, f, A, b) \\ x &= quadprog(H, f, A, b, Aeq, beq) \\ x &= quadprog(H, f, A, b, Aeq, beq, lb, ub) \\ x &= quadprog(H, f, A, b, Aeq, beq, lb, ub, x0) \\ x &= quadprog(H, f, A, b, Aeq, beq, lb, ub, options) \\ [x, fval, exitflag, output] &= quadprog(\_\_) \end{aligned} \tag{5.2}$$

Exit flag 1 implies function converged to a solution x while 2 implies no feasible point was found.The available algorithms are 'interior-point-convex' (default), 'trust-region-reflective', and 'active-set'. For full description of the various exit flags, options which include different available algorithms check out [7].

5.1.1 Example: QPP

Consider this quadratic programming problem qpp,

$$\begin{aligned} \min z = & \frac{1}{2}x_1^2 + x_2^2 - x_1x_2 - 2x_1 - 6x_2 \\ s.t \quad & x_1 + x_2 \leq 2 \\ & -x_1 + 2x_2 \leq 2 \\ & 2x_1 + x_2 \leq 3 \\ & x_j \geq 0 \ (j = 1,2) \end{aligned} \tag{5.3}$$

The quadratic syntax is

$$\frac{1}{2}x^T Hx + f^T x$$

where

$$H = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}$$

$$f = \begin{bmatrix} -2 \\ -6 \end{bmatrix}$$

subject to linear constraints.

Problem 5.3 can be solved in MATLAB thus

```
% clear workspace and command window
clear
clc

% enter data
H = [1 -1; -1 2];
f = [-2; -6];
A = [1 1; -1 2; 2 1];
```

```
b = [2; 2; 3];
Aeq = [];
beq = [];
lb = [0, 0]; %or zeros(length(f),1);
ub = [];
X0 = zeros(length(f),1);

% specify options
options = optimoptions('quadprog','Algorithm','active-set', 'Display','
    iter');

% solve the problem
[x,fval, flag, output] = quadprog(H,f,A,b,Aeq,beq,lb,ub,X0,options);
```

Listing 5.1: qppmatlab.m

5.2 NON-LINEAR PROGRAMMING

Consider a more general non-linear programming problem of this type,

$$\min_x f(x) \quad s.t \begin{cases} c(x) \leq 0 \\ ceq(x) = 0 \\ A.x \leq b \\ A_{eq}.x = b_{eq} \\ l_b \leq x \leq u_b \end{cases} \tag{5.4}$$

x, b, b_{eq}, l_b and u_b are vectors, A and A_{eq} are matrices, $c(x)$ and $ceq(x)$ are functions that return vectors, and $f(x)$ is a function that returns a scalar. $f(x), c(x)$, and $ceq(x)$ can be nonlinear functions. MATLAB provides a function, *fmincon* that solves problems of type 5.4. The syntax is,

$$
\begin{aligned}
x &= fmincon(fun, x0, A, b)\\
x &= fmincon(fun, x0, A, b, Aeq, beq)\\
x &= fmincon(fun, x0, A, b, Aeq, beq, lb, ub)\\
x &= fmincon(fun, x0, A, b, Aeq, beq, lb, ub, nonlcon)\\
x &= fmincon(fun, x0, A, b, Aeq, beq, lb, ub, nonlcon, options)\\
x &= fmincon(problem)\\
[x, fval] &= fmincon(\_\_\_)\\
[x, fval, exitflag, output] &= fmincon(\_\_\_)\\
[x, fval, exitflag, outputgrad, hessian] &= fmincon(\_\_\_)
\end{aligned}
\tag{5.5}
$$

Exit flag 1 implies function converged to a solution x while 2 implies no feasible point was found. lambda is a structure with fields containing the Lagrange multipliers at the solution x, grad is the gradient of fun at the solution x while hessian is the Hessian of fun at the solution x. The available algorithms are 'interior-point' (default), 'trust-region-reflective', 'sqp', 'sqp-legacy' (optimoptions only) and 'active-set'. For full description of the various exit flags, options which include different available algorithms check out [8].

5.2.1 Example: Simple NLPP

Consider this non-linear programming problem nlpp,

$$
\begin{aligned}
\min z = & 100(x_2 - x_1^2)^2 + (1 - x_1)^2\\
s.t \quad & x_1 + 2x_2 \leq 1\\
& 2x_1 + x_2 = 1
\end{aligned}
\tag{5.6}
$$

Find the minimum value starting from the point $[0.5, 0]$

Problem 5.6 can be solved in MATLAB thus

```
% clear workspace and command window
```

```
clear
clc

% enter data
fun = @(x)100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
x0 = [0.5,0];
A = [1,2];
b = 1;
Aeq = [2,1];
beq = 1;
lb=[];
ub=[];
NONLCON=[];
options = optimoptions('fmincon', 'Display','iter');
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,NONLCON,options)
```

Listing 5.2: nlppmatlab.m

5.2.2 Example: Ridge Regressor Problem

Ridge Regressor is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the resulting statistical model. The problem is

$$\min_x \left\{ \frac{1}{2} \left|\left| Ax - b \right|\right|_2^2 + \lambda \left|\left| x \right|\right|_2^2 \right\} \tag{5.7}$$

where $x \in R^n$, $b \in R^m$ is the target values, $A \in R^{m \times n}$ is the matrix of observations given, and λ is the regularization parameter.

Problem 5.7 can be solved in MATLAB thus

```
clear
clc
rng(0)

%Read data
trainpth ='./data/train_epi_r.csv';
testpth = './data/test_epi_r.csv';
train = readtable(trainpth);
test = readtable(testpth);
```

```
10
11 %Get A and b
12 train = train{1:3000,:}; X_train = train(:,1:end-1);
13 test = test{1:50,:}; X_test = test(:,1:end-1);
14 y_train = train(:,end);
15 y_test = test(:,end);
16
17 A = X_train;
18 b = y_train;
19
20 % Define minimization parameters
21 x0 = rand(size(A,2),1);
22 lambda = 1;
23
24 fun = @(x) objective(A,x,b,lambda);
25 options = optimoptions('fminunc', 'Display','iter');
26
27 % Minimize
28 [x,fval] = fminunc(fun,x0,options);
29
30 %Error and Prediction
31 train_err = error(A,b,x)
32 test_err = error(X_test,y_test,x)
33
34
35 function xVal = objective(A,x,b,lambda)
36     xVal = (1/2)*norm(A*x - b)^2 + (lambda/2)*norm(x)^2;
37 end
38
39
40 function rmse = error(A,b,x)
41     rmse = sqrt(mean((b - predict(A,x)).^2));
42 end
43
44 function p = predict(A,x)
45     p = A*x;
46 end
```

Listing 5.3: ridge.m

5.2.3 Example: Support Vector Machine Classification Problem

The support vector machine (SVM) is a machine learning model used for classification and regression purposes. It can be used for both linear and non-linear problems. Basically what it does is create a hyperplane to separate the data into classes. The objective is to minimize the cost function,

$$\begin{aligned} \min_{x,b} \quad & \frac{||x||_2^2}{2m} \\ s.t \quad & \mathbf{1} - b \circ (Ax + a) \leq 0 \end{aligned} \tag{5.8}$$

where $x \in R^n$, $b \in R^m$ is the target values, $A \in R^{m \times n}$ is the matrix of observations given, a is the intercept (or bias) to be determined alongside the weights x and λ is the regularization parameter. The Hadamard operator $\circ$ represents elements-by-elements operations.

Problem 5.7 can be solved in MATLAB thus

```
clear all
clc
rng(0)

%Read data
trainpth ='./data/train_epi_r.csv';
testpth = './data/test_epi_r.csv';
train = readtable(trainpth);
test = readtable(testpth);

%Get A and b
train = train{1:end,:}; X_train = train(:,1:end-1);
test = test{1:end,:}; X_test = test(:,1:end-1);
y_train = train(:,end);
y_test = test(:,end);

%Convert numerical response to categorical
y_train(y_train<=4.0)=0;
y_train(y_train>4.0)=1;

y_test(y_test<=4.0)=0;
y_test(y_test>4.0)=1;

% Define minimization parameters
```

```
A = X_train;
b = y_train;
x0 = rand(size(A,2)+1,1);
lambda = 1;

fun = @(x) objective(x,b);
nonlcon = @(x) constraints(A,b,x);
options = optimoptions('fmincon', 'Display','iter', 'Algorithm',...
    'interior-point');

% Minimize
[x,fval] = fmincon(fun,x0,[],[],[],[],[],[],nonlcon,options);

%Error and Prediction
train_err = accuracy(A,b,x)
test_err = accuracy(X_test,y_test,x)


function xVal = objective(x,b)
    m = length(b);m=1;
    xVal = (1/(2*m))*norm(x(1:end-1))^2;
end

function [c,ceq] = constraints(A,b,x)
    c = 1 - b.*(A*x(1:end-1) + x(end));
    ceq = [];
end


function acc = accuracy(A,b,x)
    acc = ( sum(predict(A,x) == double(reshape(b,[],1))) / length(b) )
   * 100;
end

function p = predict(A,x)
    p = sign(A*x(1:end-1) + x(end));
end
```

Listing 5.4: svm.m

Chapter 6

Support Vector Machine

Support Vector Machine (SVM) was developed in the year 1963 by Vladimir N. Vapnik and Alexey Ya. Chervonenkis. Bernhard Boser, Isabelle Guyon and Vladmir Vapnik suggested applying kernel trick to maximum-margin hyperplanes [9] in order to obtain nonlinear classifiers in the year 1992. The currently used soft margin was proposed in 1993 by Corinna Cortes and Vapnik, and was published in 1995 [10]

SVM builds a hyperplane or many hyperplanes in a high- or infinite- dimensional space. The hyperplane having the greatest distance to the nearest training example of any class yileds a good separation [11]. SVM is used for regression, classifications, detecting outliers and many other applications [11].

6.1 HYPERPLANE

Given training examples of n points of the form,

$$(z_1, y_1), ..., (z_n, y_n),$$

where $y_i \in \{-1, 1\}$ and $z_i \in R^m$. We seek for maximum-margin hyperplane separating z_i having $y_i = 1$ from z_i having $y_i = -1$. A hyperplane can be written as the set of points z satisfying

$$w^T \bar{z} - b = 0, \tag{6.1}$$

where w is the normal vector to the hyperplane.

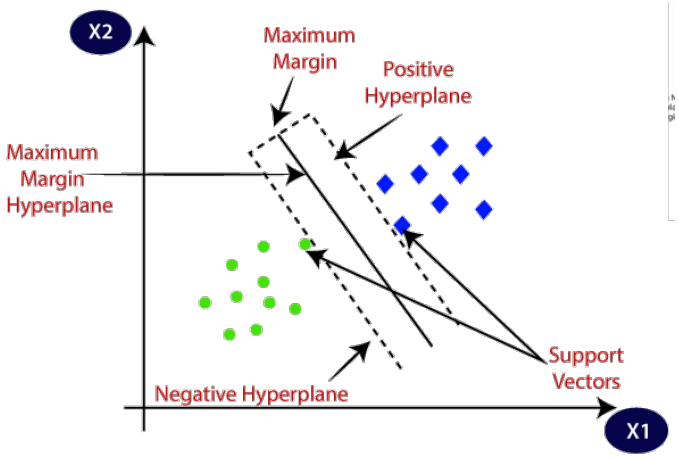

Figure 6.1: Support Vector Machine Hyperplanes [12].

6.2 HARD-MARGIN

Suppose we are able to separate the data linearly, then we can get two parallel hyperplanes that separates the two classes of data in such a way that their distance apart is maximized. We call the area bounded by these hyperplanes, margin and the hyperplane lying halfway we call maximum-margin.

To keep data points away from the margin or on the boundary, the following constraints are introduced,

$$w^T \bar{z}_i + b \geq 1, \quad if \quad y_i = 1, \quad i = 1, ..., n \tag{6.2}$$

$$w^T \bar{z}_i + b \leq -1, \quad if \quad y_i = -1, \quad i = 1, ..., n \tag{6.3}$$

More compactly, 6.2 and 6.3 can be written as

$$y_i(w^T \bar{z}_i + b) \geq 1, \qquad i = 1, ..., n \tag{6.4}$$

Equation 6.4 can be rewritten as

$$-y_i(b + w^T \bar{z}_i) + 1 \leq 0, \qquad i = 1, ..., n \tag{6.5}$$

Our goal is to find vector w and b that maximizes the width of the margins, i.e the points lying on the margins of the hyperplanes of $y = -1$ and $y = 1$ such as z_- and z_+. Hence width is defined as,

$$\begin{aligned} width &= (z_+ - z_-), \quad \text{the difference vector} \\ &= (z_+ - z_-) \cdot \frac{\bar{w}}{||w||}, \quad \text{the unit vector (since } w \text{ is normal)} \end{aligned} \tag{6.6}$$

For the samples that lie on the margin, we have

$$y_i(b + w^T \bar{z}_i) = 1, \qquad i = 1, ..., n \tag{6.7}$$

For z_+ we have $y = 1$, hence

$$w^T z_+ = 1 - b \tag{6.8}$$

For z_- we have $y = -1$, hence

$$w^T z_- = 1 + b \tag{6.9}$$

Substituting 6.8 and 6.9 into 6.6, we have

$$width = [1 - b + (1 + b)] \cdot \frac{1}{||w||} = \frac{2}{||w||} \tag{6.10}$$

By maximizing width we mean,

$$\max \frac{2}{||w||} \quad \Longrightarrow \quad \max \frac{1}{||w||} \quad \Longrightarrow \quad \min ||w|| \quad \Longrightarrow \quad \min \frac{1}{2}||w||^2 \tag{6.11}$$

We now define the optimization problem as

$$\begin{aligned} \min_{w} \quad & \frac{1}{2}\|w\|^2 \\ s.t \quad & -y_i\left(b + w^T \bar{z}_i\right) + 1 \leq 0, \qquad i = 1, ..., n \end{aligned} \tag{6.12}$$

By setting $w_{m+1} = b$ the constraint of problem 6.12 can be re-written as

$$\begin{aligned} & -y_i\left(b + w^T \bar{z}_i\right) + 1 \leq 0, && i = 1, ..., n \\ \Longrightarrow \quad & -y_i\left(w_{n+1} + w^T \bar{z}_i\right) \leq -1, && i = 1, ..., n \\ \Longrightarrow \quad & -y_i\left(w^T \left[\bar{z}_i, 1\right]\right) \leq -1, \quad w \in R^{n+1} && i = 1, ..., n \\ \Longrightarrow \quad & -y_i\left(w^T \left[\bar{z}_i, 1\right]\right) \leq -1, && i = 1, ..., n \\ \Longrightarrow \quad & -diag(y)[Z, \bar{1}]w^T \leq -\bar{1} && \bar{1} \in R^n = (1, ..., 1)^T \end{aligned} \tag{6.13}$$

By setting the following,

$$\begin{aligned} & f = \bar{0}, && \bar{0} \in R^n = (0, ..., 0)^T \\ & H = \begin{bmatrix} \bar{1} & 0 \\ 0 & 0 \end{bmatrix} \\ & A = -diag(y)\begin{bmatrix} Z, & \bar{1} \end{bmatrix} \\ & h = \bar{1} \end{aligned} \tag{6.14}$$

problem 6.12 can be re-written as,

$$\begin{aligned} \min_{w} \quad & \frac{1}{2} w^T H w + f^T w \\ s.t \quad & Aw \leq h, \qquad i = 1, ..., n \end{aligned} \tag{6.15}$$

Remember that $b = w_{m+1}$

6.3 SOFT-MARGIN

At times, problem 6.12 may not have a solution; it may not be possible to obtain a hyperplane that exactly separate all points with the hard margin $\frac{2}{||w||}$. For this reason, it is neccessary to solve a relaxed version of 6.12 with a soft margin,

$$\begin{aligned} \min_{w} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\zeta_i \\ s.t \quad & -y_i\left(b + w^T\bar{z}_i\right) + 1 \leq \zeta_i, \qquad i = 1,...,n \\ & \zeta_i \geq 0, \qquad i = 1,...,n \end{aligned} \tag{6.16}$$

In order to solve problem 6.16 by quadratic method, we need to do the following transformations. We first let $w = [w_1,...,w_m,b,\zeta_1,...,\zeta_n]$, hence $w \in R^{m+1+n}$. From the constraints of problem 6.16 we obtain,

$$-\left[diag(y)\left[Z, \quad \bar{1}\right], \quad I\right]w^T \leq -\bar{1} \qquad\qquad \bar{1} \in R^n = (1,...,1)^T \tag{6.17}$$

$$\begin{aligned} & \zeta_i \geq 0, \qquad i = 1,...,n \\ \implies & w_i \geq 0 \qquad i = m+2,...,m+n+1 \\ \implies & w \geq \left[-\bar{\infty}_{1\times(m+1)} \quad \bar{0}_{1\times n}\right]^T \end{aligned} \tag{6.18}$$

where I is an $n \times n$ identity matrix. From the objective function of problem 6.16 we obtain,

$$\frac{1}{2}w^T\begin{bmatrix} I_{m\times m} & \bar{0}_{m\times(n+1)} \\ \bar{0}_{(n+1)\times m} & \bar{0}_{(n+1)+\times(n+1)} \end{bmatrix}w + \left[\bar{0}_{1\times(m+1)} \quad \frac{c}{n}\bar{1}_{1\times n}\right]w \tag{6.19}$$

To simplify the optimization problem, we make the following substitutions,

$$
\begin{aligned}
H &= \begin{bmatrix} I_{m\times m} & \bar{0}_{m\times(n+1)} \\ \bar{0}_{(n+1)\times m} & \bar{0}_{(n+1)\times(n+1)} \end{bmatrix} \\
f &= \begin{bmatrix} \bar{0}_{1\times(m+1)} & \frac{c}{n}\bar{1}_{1\times n} \end{bmatrix} \\
lb &= \begin{bmatrix} -\bar{\infty}_{1\times(m+1)} & \bar{0}_{1\times n} \end{bmatrix}^T \\
A &= -\begin{bmatrix} diag(y)\begin{bmatrix} Z, & \bar{1} \end{bmatrix}, & I \end{bmatrix} \\
h &= -\bar{1}_{n\times 1}
\end{aligned}
\tag{6.20}
$$

Finally, 6.16 can be written as,

$$
\begin{aligned}
\min_{w} \quad & \frac{1}{2} w^T H w + f^T w \\
s.t \quad & Aw \leq h, \qquad i = 1, ..., n \\
& w \geq lb
\end{aligned}
\tag{6.21}
$$

6.4 THE DUAL PROBLEM

The Lagrangian of 6.12 can be written as

$$
L = \frac{1}{2}||w||^2 + \sum_{i=1}^{n} \alpha_i \left[y_i \left(b - w^T \bar{z}_i \right) + 1 \right] \tag{6.22}
$$

where $\alpha_i \geq 0$ is Lagrange multiplier. By applying KKT conditions we have,

$$
\begin{aligned}
& \frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i \bar{z}_i = 0 \\
& \implies w = \sum_{i=1}^{n} \alpha_i y_i \bar{z}_i
\end{aligned}
\tag{6.23}
$$

and

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{6.24}$$

Substituting 6.23 and 6.24 into 6.22 we obtain

$$\begin{aligned} L &= \frac{1}{2}\Big(\sum_{i=1}^{n} \alpha_i y_i \bar{z}_i\Big)\Big(\sum_{j=1}^{n} \alpha_j y_j \bar{z}_j\Big) + \Big(b\sum_{i=1}^{n} \alpha_i y_i\Big) - \sum_{i=1}^{n} \alpha_i y_i \bar{z}_i\Big(\sum_{j=1}^{n} \alpha_j y_j \bar{z}_j\Big) + \sum_{i=1}^{n} \alpha_i \\ &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (\bar{z}_i)^T \bar{z}_j \end{aligned} \tag{6.25}$$

We will now maximize 6.25

$$\max_{[\alpha_1,\ldots\alpha_n]} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i y_i K(z_i, z_j) y_j \alpha_j, \quad s.t \quad \begin{cases} \sum_{i=1}^{n} \alpha_i y_i = 0, \\ \\ 0 \le \alpha_i \le C, \qquad i = 1, \ldots, n, \end{cases} \tag{6.26}$$

where $K(z_i, z_j) = \phi(z_i)\phi(z_j)$ is a kernel function. Writing problem 6.26 in matrix form we obtain,

$$\min_{z \in R^n} \quad \frac{1}{2}\alpha^T H \alpha - f^T \alpha \quad s.t \quad \begin{cases} y^T \alpha = 0, \\ 0 \le \alpha \le C, \end{cases} \tag{6.27}$$

where $H = diag(y)K(z,z)diag(y)$ and $f = \bar{1}_{m\times 1}$

Problem 6.27 is in quadratic form.

6.5 DECISION RULE

To classify unknown data point $\bar{u}$ after obtaining our model, we employ the following decision rule.

Algorithm: Decision Rule

if $\bar{z} \cdot \bar{w} + b \geq 0$ **then**
  $y = +1$
else
  $y = -1$
end

For primal model a straight forward substitution works, but for dual model, the optimal primal values w can be calculated from expression 6.23,

$$\hat{w} = \sum_{i=1}^{n} \hat{\alpha}_i y_i \bar{z}_i, \tag{6.28}$$

while he offset b is recovered by finding an z_i on the margin's boundary $\big($i.e., we use only the indices having $\alpha \,!= 0\big)$ that solves,

$$y_i\big(\hat{w}^T z_i - \hat{b}\big) = 1 \tag{6.29}$$

6.6 KERNEL FUNCTIONS

Most real world data are not linearly separable hence the need for nonlinear classification. In order to learn a nonlinear classification rule corresponding to the linear classification rule for the transformed data points ϕz_i, we define a kernel function K satisfying

$$K(z_i, z_j) = \phi z_i \cdot \phi z_j \tag{6.30}$$

Some popular kernels are,

1. Linear $\quad K(z, y) = z^T y$
2. Polynomial homogeneous $\quad K(z, y) = (z^T y)^d$
3. Polynomial nonhomogeneous $\quad K(z, y) = (z^T y + 1)^d$
4. Gaussian radial basis function $\quad K(z, y) = \exp(-\gamma ||z - y||^2), \quad \gamma > 0$

5. $K(z,y) = \exp \frac{-||z-y||}{\sigma}, \quad \sigma > 0$

Chapter 7

AMPL and Non-Linear Programming

Bibliography

[1] Robert Fourer, David M. Gay, and Brian W. Kernighan. AMPL: A mathematical programming language. In *Algorithms and Model Formulations in Mathematical Programming*, pages 150–151. Springer Berlin Heidelberg, 1989.

[2] AMPL. Ampl options for students, 2019. Accessed: 2022-03-20.

[3] AMPL. Ampl apis introduction, 2019. Accessed: 2022-03-20.

[4] Robert Fourer, David M. Gay, and Brian W. Kernighan. A modeling language for mathematical programming. *Management Science*, 36(5):519–554, May 1990.

[5] MATLAB. linprog: Solve linear programming problems, 2022. Accessed: 2022-03-27.

[6] MATLAB. intlinprog: Solve mixed-integer linear programming (milp), 2022. Accessed: 2022-03-27.

[7] MATLAB. quadprog: Solve quadratic programming problems, 2022. Accessed: 2022-04-25.

[8] MATLAB. fmincon: Solve nonlinear programming problems, 2022. Accessed: 2022-04-25.

[9] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory - COLT '92*. ACM Press, 1992.

[10] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.

[11] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Series in Statistics. Springer, 2 edition, 2009.

[12] Java T Point. Support vector machine algorithm, 2021. [Online; accessed June 01, 2021].