```
>> Lecture 7
>>
>>              -- Optimization
>>
```

"In my opinion, no single design
is apt to be optimal for everyone."
– Donald Norman (1935–)

Contents

Introduction

- Mathematical optimization is to find the optimal selection of feasible solutions with regard to the specific criteria.
- In the simplest case, an optimization problem consists of maximizing or minimizing a real function by systematically choosing input values from within an allowed set and computing the value of the function.
- The generalization of optimization theory and techniques to other formulations comprises a large area of applications.
  - EE: circuit layout, fabrication parameters of transistors...
  - CS: model parameters in machine learning...
  - Fin: optimal portfolio...
  - Economics: tax, wage rate...
  - ...

Optimization Problem in Standard Form (1/3)

- An optimization problem can be represented in the following way:
  - Given a function $f : M \to \mathbb{R}$.
  - (Minimization) Find $x_0 \in M$ such that $f(x_0) \leq f(x)$ for all $x \in M$.
  - (Maximization) Find $x_0 \in M$ such that $f(x_0) \geq f(x)$ for all $x \in M$.
- Many real-world and theoretical problems may be modeled in this general framework.

Optimization Problem in Standard Form (2/3)

- Typically, M is some subset of the Euclidean space $\mathbb{R}^n$, often specified by a set of constraints, equalities or inequalities that the members of M have to satisfy.
- The domain M of f is called the **search space** or the **choice set**, while the elements of M are called **feasible solutions**.
- The function f is called an **objective function**.
  - Aka **loss function**, **cost function**, **utility function**, and **fitness function**.
- A feasible solution that minimizes (or maximizes, if that is the goal) the objective function is called an **optimal solution**.

Optimization Problem in Standard Form (3/3)

- By convention, the standard form of an optimization problem is stated in terms of minimization.
- Convex optimization, a subfield of optimization, studies the problem of minimizing convex functions over convex sets. (You will see later.)
- With recent improvements in computing and in optimization theory, convex minimization is nearly as straightforward as **linear programming**[1].
- Many optimization problems can be reformulated as convex optimization problems.
- For example, the problem of maximizing a concave function f can be re-formulated equivalently as a problem of minimizing the function $-f$, which is convex.

[1] Aka 線性規劃(高二上).

Classification of Optimization Problems

- Finite- vs. infinite-dimensional problems
- Unconstrained vs. constrained problems
- Convex vs. non-convex problems
- Linear vs. non-linear problems
- Continuous vs. discrete problems
- Deterministic vs. stochastic problems

Example

- Consider $f(x) = x^4 - 10.5x^3 + 39x^2 - 59.5x + 30$.

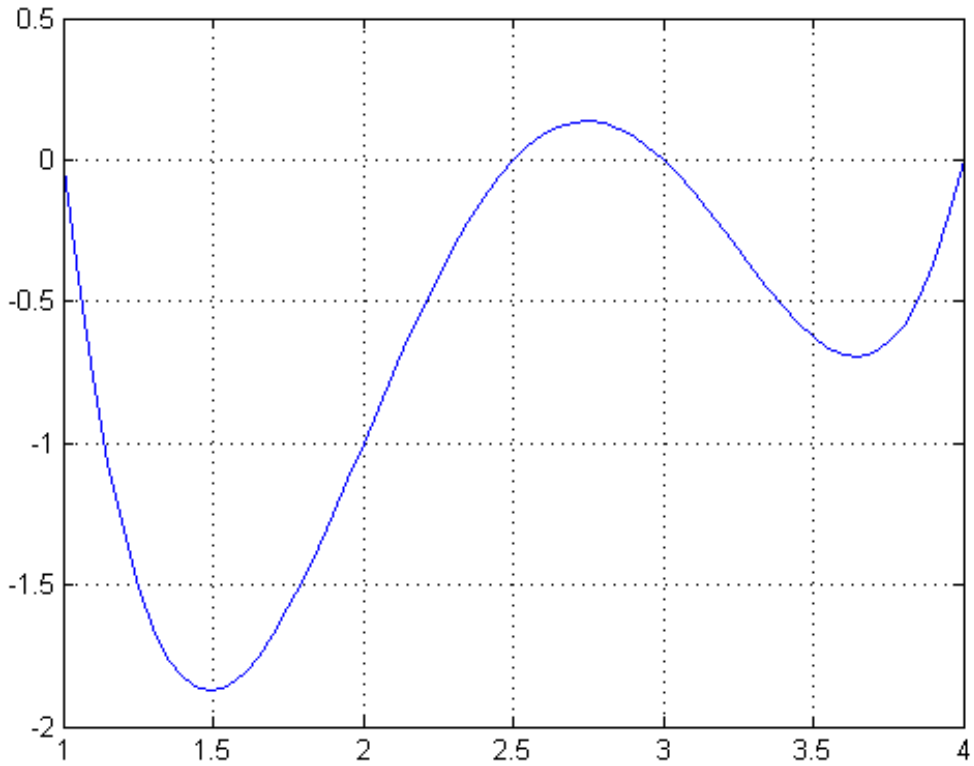
```
>> g=@(x) polyval([1 -10.5 39 -59.5 30], x);
>> x=1:0.05:4;
>> plot(x,g(x)); grid on;
>> [s, fval]=fminunc(g,0) % unconstrained ...
    minimizing g

s =

    1.4878

fval =

   -1.8757
```



- Try: $[s, fval] = \textbf{fminunc}(g, 5)$
- The minimal point returns depending on the initial guess.

Example: Utility Maximization

- A consumer has a budget flush with $w = 10$ and faces prices $p_1 = 1, p_2 = 2$ for *Product 1* and *2*, respectively.
- Let x_1 and x_2 be the weights of two products, and u be the utility function over two goods, given by $u = x_1^{0.8} + x_2^{0.8}$.
- Then, what is the optimal (x_1, x_2)?

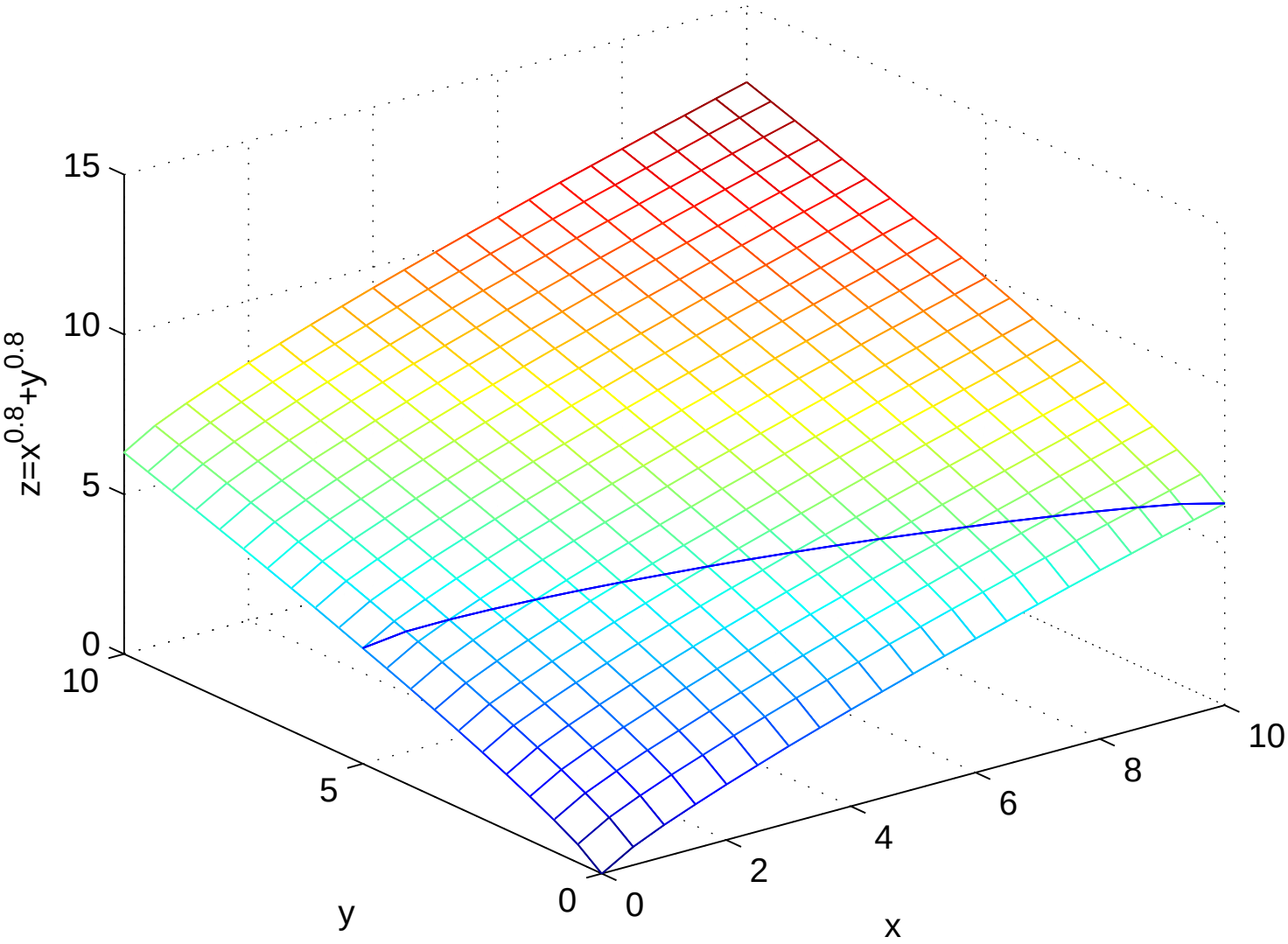
- We can observe the behavior of u first.
  - $\frac{\partial u}{\partial x_i} = 0.8x_i^{-0.2} > 0$ for all $x_i > 0$.
  - $\frac{\partial^2 u}{\partial x_i^2} = -0.16x_i^{-1.2} < 0$ for all $x_i > 0$.
  - So, u increases with a gradual slowdown as x_1 and x_2 increase.
- Formulate the problem into a standard form of optimization:

$$\max_{x_1, x_2} \{u\},$$
$$x_1 p_1 + x_2 p_2 = 10.$$

- Let $A_{eq} = \begin{bmatrix} p_1 & p_2 \end{bmatrix}$, $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and $b_{eq} = 10$. Then the second equation can be $A_{eq} \cdot \vec{x} = b_{eq}$.

mesh (Recap)

```
1 [X,Y]=meshgrid(0:.5:10);
2 u=(X.^0.8+Y.^0.8);
3 LX=0:.5:10;
4 LY=-0.5*LX+5;
5 uu=LX.^0.8+LY.^0.8;
6 mesh(X,Y,u);grid on; hold on;
7 plot3(LX,LY,uu);
```


z=x^0.8+y^0.8
15
10
5
0
10
5
y
0
0
2
4
6
8
10
x

```
>> f=@(x) -(x(1)^0.8+x(2)^0.8);
>> Aeq=[1 2];
>> beq=10;
>> x_0=[8 1]; % initial guess
>> [xx, fval]=fmincon(f,x_0,[],[],Aeq,beq)

xx =

    9.4118 0.2941

fval =

   -6.3865
```

- The maximization problem could be equivalent to minimize $-u$.

Linear Programming Problems

- If the objective function f and the defining functions of M are linear, then the problem you are concern about will be a linear optimization problem.
- A general form of a linear programming problem is given by

$$\begin{array}{lcl} c^\top x & \longrightarrow & min\ (max) \\ s.t. & & \\ Ax & = & a \\ Bx & \leq & b \\ lb \leq & x & \leq ub; \end{array}$$

- That is, $f(x) = c^T x$ and $M = \{x \in \mathbb{R}^n | Ax = a, Bx \leq b, lb \leq x \leq ub\}$.

- Once you have defined the matrices A, B, and the vectors c, a, b, lb and ub, then you can call **linprog** to solve the problem:

$$[x, fval, exitflag, output, lambda] = \textbf{linprog}(c, A, a, B, b, lb, ub, x0, options),$$

where

- c: coefficient vector of the objective
- A: matrix of inequality constraints
- a: right hand side of the inequality constraints
- B or []: matrix of equality constraints, or no constraints
- b or []: right hand side of the equality constraints, or no constraints
- lb, ub or []: lower/upper bounds for x, or no lower/upper bounds
- x_0: initial vector for the algorithm if known; otherwise [].
- *options*: options are set using the **optimset** funciton which determines the details in the algorithm.

- (Continued)
  - x: optimal solution
  - *fval*: optimal value of the objective function
  - *exitflag*: tells whether the algorithm converged or not ($exitflag > 0$ means convergence.)
  - *output*: a struct for number of iterations, algorithm used and PCG iterations (when $LargeScale = on$)
  - *lambda*: a struct containing Lagrange multipliers corresponding to the constraints

About *Optimset*

- The input argument options is a structure, which contains several parameters that you can use with a given Matlab optimization routine. (Try **optimset**('linprog')!!)
- For example,

```
>> options=optimset('ParameterName1',value1,...
'ParameterName2',value2,...)
```

- The following are parameters and their corresponding values which are frequently used with **linprog**:
  - *'LargeScale'*: 'on','off'
  - *'Simplex'*: 'on','off'
  - *'Display'*: 'iter,'final','off'
  - *'Maxiter'*: maximum number of iteration
  - *'TolFun'*: termination tolerance for the objective function
  - *'TolX'*: termination tolerance for the iterates
  - *'Diagnostics'*: 'on' or 'off'

Example 1

- Solve the following linear optimization problem using **linprog**.

$$
\begin{array}{llll}
2x_1+ & 3x_2 & \rightarrow & \max \\
\text{s.t.} & & & \\
x_1+ & 2x_2 & \leq & 8 \\
2x_1+ & x_2 & \leq & 10 \\
 & x_2 & \leq & 3 \\
x_1, & x_2 & \geq & 0
\end{array}
$$

```
c=[-2,-3]';
A=[1,2;2,1;0,1];
a=[8,10,3]';
options=optimset('LargeScale','off');
xsol=linprog(c,A,a,[ ],[ ],[ ],[ ],[ ],options);
```

```
Optimization terminated.

xsol =

    4.0000
    2.0000

>>
```

Example 2

- Solve the following LP using **linprog**:

$$
\begin{array}{c}
c^{\top} x \longrightarrow \max \\
Ax = a \\
Bx \geq b \\
Dx \leq d \\
lb \leq x \leq lu
\end{array}
$$

$$(A|\,a) = \left(\begin{array}{cccccc|c} 1 & 1 & 1 & 1 & 1 & 1 & 10 \\ 5 & 0 & -3 & 0 & 1 & 0 & 15 \end{array}\right), \qquad (B|\,b) = \left(\begin{array}{cccccc|c} 1 & 2 & 3 & 0 & 0 & 0 & 5 \\ 0 & 1 & 2 & 3 & 0 & 0 & 7 \\ 0 & 0 & 1 & 2 & 3 & 0 & 8 \\ 0 & 0 & 0 & 1 & 2 & 3 & 8 \end{array}\right),$$

$$(D|\,d) = \left(\begin{array}{cccccc|c} 3 & 0 & 0 & 0 & -2 & 1 & 5 \\ 0 & 4 & 0 & -2 & 0 & 3 & 7 \end{array}\right), \quad lb = \begin{pmatrix} -2 \\ 0 \\ -1 \\ -1 \\ -5 \\ 1 \end{pmatrix}, lu = \begin{pmatrix} 7 \\ 2 \\ 2 \\ 3 \\ 4 \\ 10 \end{pmatrix}, c = \begin{pmatrix} 1 \\ -2 \\ 3 \\ -4 \\ 5 \\ -6 \end{pmatrix}.$$

```
clear all;
clc

A=[1,1,1,1,1,1;5,0,-3,0,1,0];
a=[10,15]';
B1=[1,2,3,0,0,0;0,1,2,3,0,0;0,0,1,2,3,0;0,0,0,1,2,3];
b1=[5,7,8,8]';
D=[3,0,0,0,-2,1;0,4,0,-2,0,3];
d=[5,7]';
lb=[-2,0,-1,-1,-5,1]';
ub=[7,2,2,3,4,10]';
c=[1,-2,3,-4,5,-6]';
B=[-B1;D]; b=[-b1;d];

[xsol,fval,exitflag,output]=linprog(c,A,a,B,b,lb,ub)
fprintf('%s %s \n', 'Algorithm Used: ...
    ',output.algorithm);
disp('============================');

options=optimset('linprog');
```

```
20 options = optimset(options,'LargeScale','off',...
21 'Simplex','on','Display','iter');
22 [xsol,fval,exitflag]=linprog(c,A,a,B,b,lb,ub,[],options)
23 fprintf('%s %s \n', 'Algorithm Used: ...
       ',output.algorithm);
24 fprintf('%s','Reason for termination:')
25 if (exitflag)
26     fprintf('%s \n',' Convergence.');
27 else
28     fprintf('%s \n',' No convergence.');
29 end
```

Example 3: Approximation of discrete Data by a Curve

- Suppose the measurement of a real process over a 24 hours period be given by the following table with 14 data values:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
t_i	0	3	7	8	9	10	12	14	16	18	19	20	21	23
u_i	3	5	5	4	3	6	7	6	6	11	11	10	8	6

- The values t_i represent time and u_i's are measurements.

- Assuming there is a mathematical connection between the variables t and u, we would like to determine the coefficients $a, b, c, d, e \in \mathbb{R}$ of the function

$$u(t) = at^4 + bt^3 + ct^2 + dt + e,$$

so that the value of the function $u(t_i)$ could best approximate the discrete value u_i at t_i, $i = 1, \ldots, 14$. in the Chebychev sense[2].

[2] http://en.wikipedia.org/wiki/Approximation_theory#Chebyshev_approximation

- Hence, we need to solve the Chebyshev approximation problem, which is written as

$$\max_{i=1,\ldots,14} \left| u_i - \left(a\, t_i^4 + b\, t_i^3 + c\, t_i^2 + d\, t_i + e \right) \right| \rightarrow \min$$
$$\text{s.t. } a, b, c, d, e \in \mathbb{R}$$

- Reformulate it into a linear programming problem:
  - Objective function?
  - Constraints?

Solution to Chebyshev Approximation Problem

- Define the additional variable
  $f := \max_{i=1,\ldots,14} |u_i - (at_i^4 + bt_i^3 + ct_i^2 + dt_i + e)|$.
- Then the problem can be equivalently written as

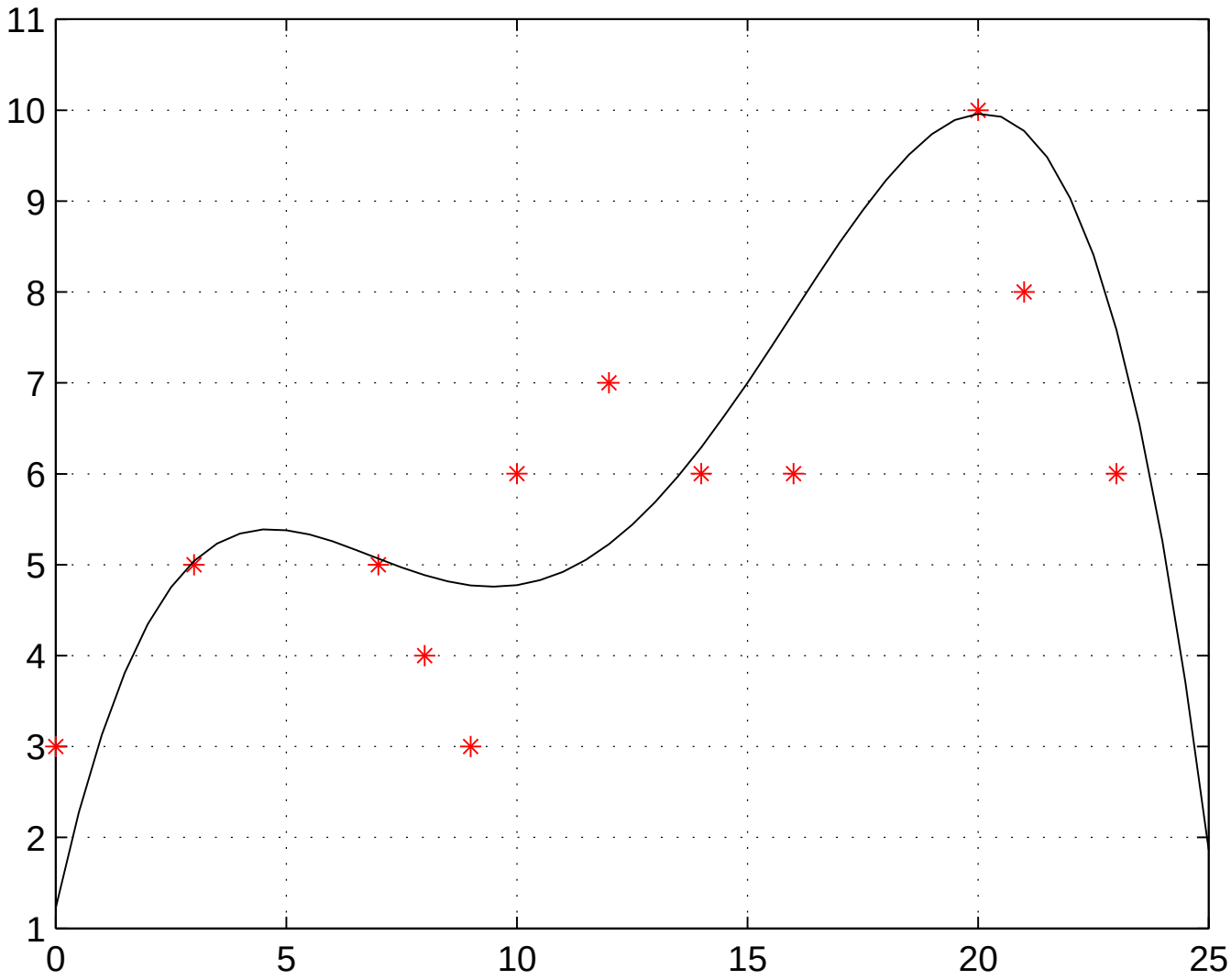
$$\begin{aligned} &\min\{f\}, \\ -(at_i^4 + bt_i^3 + ct_i^2 + dt_i + e) - f &\leq -u_i, \\ (at_i^4 + bt_i^3 + ct_i^2 + dt_i + e) - f &\leq u_i, \end{aligned}$$

  where $i \in \{1, \ldots, 14\}$.
- More specific, $[A]_{28\times 6}[x]_{6\times 1} \leq [u]_{28\times 1}$.
  - Note that $[x]_{6\times 1} = [a, b, c, e, d, f]'$.

```
clear all;
clc

t=[0,3,7,8,9,10,12,14,16,18,19,20,21,23]';
u=[3,5,5,4,3,6,7,6,6,11,11,10,8,6]';
A1=[-t.^4,-t.^3,-t.^2,-t,-ones(14,1),-ones(14,1)];
A2=[t.^4,t.^3,t.^2,t,ones(14,1),-ones(14,1)];
c=zeros(6,1);
c(6)=1; % objective function coefficient (why?)
A=[A1;A2]; % inequality constraint matrix
a=[-u;u]; % right hand side vectro of ineq ...
    constraints
[xsol,fval,exitflag]=linprog(c,A,a);

plot(t,u,'r*'); hold on; grid on;
tt=0:0.5:25;
ut=xsol(1)*(tt.^4)+xsol(2)*(tt.^3)+xsol(3)*(tt.^2)+...
xsol(4)*tt+xsol(5);
plot(tt,ut,'-k','LineWidth',2)
```


11
10
9
8
7
6
5
4
3
2
1
0
5
10
15
20
25

Exercise

- **randi**([*range*], *m*, *n*) generates an *m*-by-*n* random matrix integer values drawn uniformly in *range*.
- Use **randi** as a set of input pairs of the program in Chebyshev Approximation Problem.
  - Let t be a simple sequence like $0 : 1 : m$.
  - Let u be a sequence generated by **randi**.
- See the fitting result.

Integer Programming

- Integer programming problem is a mathematical optimization in which some or all of the variables are restricted to be integers.
  - Sometimes called **integer linear programming** (ILP), in which the objective function and the constraints (other than the integer constraints) are linear.
- Note that integer programming is much harder than linear programming in general. (Why?)

Quadratic Programming Problems

- Quadratic programming is a special type of mathematical optimization problem, which optimizes (minimizing or maximizing) a quadratic function of several variables subject to linear constraints on these variables.
- Let $Q \in \mathbb{R}^{n \times n}$, $A \in \mathbb{R}^{m \times n}$, $B \in l \times n$, $a\mathbb{R}^{m}$, and $b \in \mathbb{R}^{l}$. Then a general form of a quadratic programming problem is given by

$$
\begin{array}{lrcl}
\frac{1}{2} x^T Q x + q^\top x & \rightarrow & \min \\
\text{s.t.} & & \\
Ax & = & a \\
Bx & \leq & b \\
x & \geq & u \\
x & \leq & v
\end{array}
$$

- The general form for calling **quadprog** of the problem is given by

$$[xsol, fval, exitflag, output, lambda] = \mathbf{quadprog}(Q, q, A, a, B, b, lb, ub, x0, options),$$

where

- Q: Hessian of the objective function
- q: Coefficient vector of the linear part of the objective function
- A or []: matrix of inequality constraints, or no inequality constraints
- a or []: right hand side of the inequality constraints, or no inequality constraints
- B or []: matrix of equality constraints
- b or []: right hand side of the equality constraints
- lb, ub or []: lower/upper bounds for x, or no lower/upper bounds
- x_0: initial vector for the algorithm if known; otherwise [].
- *options*: options are set using the **optimset** function which determines the details in the algorithm.

- (Continued)
  - x: optimal solution
  - *fval*: optimal value of the objective function
  - *exitflag*: tells whether the algorithm converged or not ($exitflag > 0$ means convergence.)
  - *output*: a struct for number of iterations, algorithm used and PCG iterations (when $LargeScale = on$)
  - *lambda*: a struct containing Lagrange multipliers corresponding to the constraints

Example 4

- Solve the following quadratic optimization problem using **quadprog**.

$$
\begin{array}{llllll}
x_1^2 + 2x_2^2 + & 2x_1 + & 3x_2 & \rightarrow & \min \\
& \text{s.t.} & & & \\
& x_1 + & 2x_2 & \leq & 8 \\
& 2x_1 + & x_2 & \leq & 10 \\
& & x_2 & \leq & 3 \\
& x_1, & x_2 & \geq & 0
\end{array}
$$

- Try to re-formulate it.

$$Q = \begin{pmatrix} 2 & 0 \\ 0 & 4 \end{pmatrix}, q = \begin{pmatrix} 2 \\ 3 \end{pmatrix}, A = \begin{pmatrix} 1 & 2 \\ 2 & 1 \\ 0 & 1 \end{pmatrix}, a = \begin{pmatrix} 8 \\ 10 \\ 3 \end{pmatrix},$$

$$B = [\,], b = [\,], lb = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, ub = \begin{pmatrix} \infty \\ \infty \end{pmatrix}$$

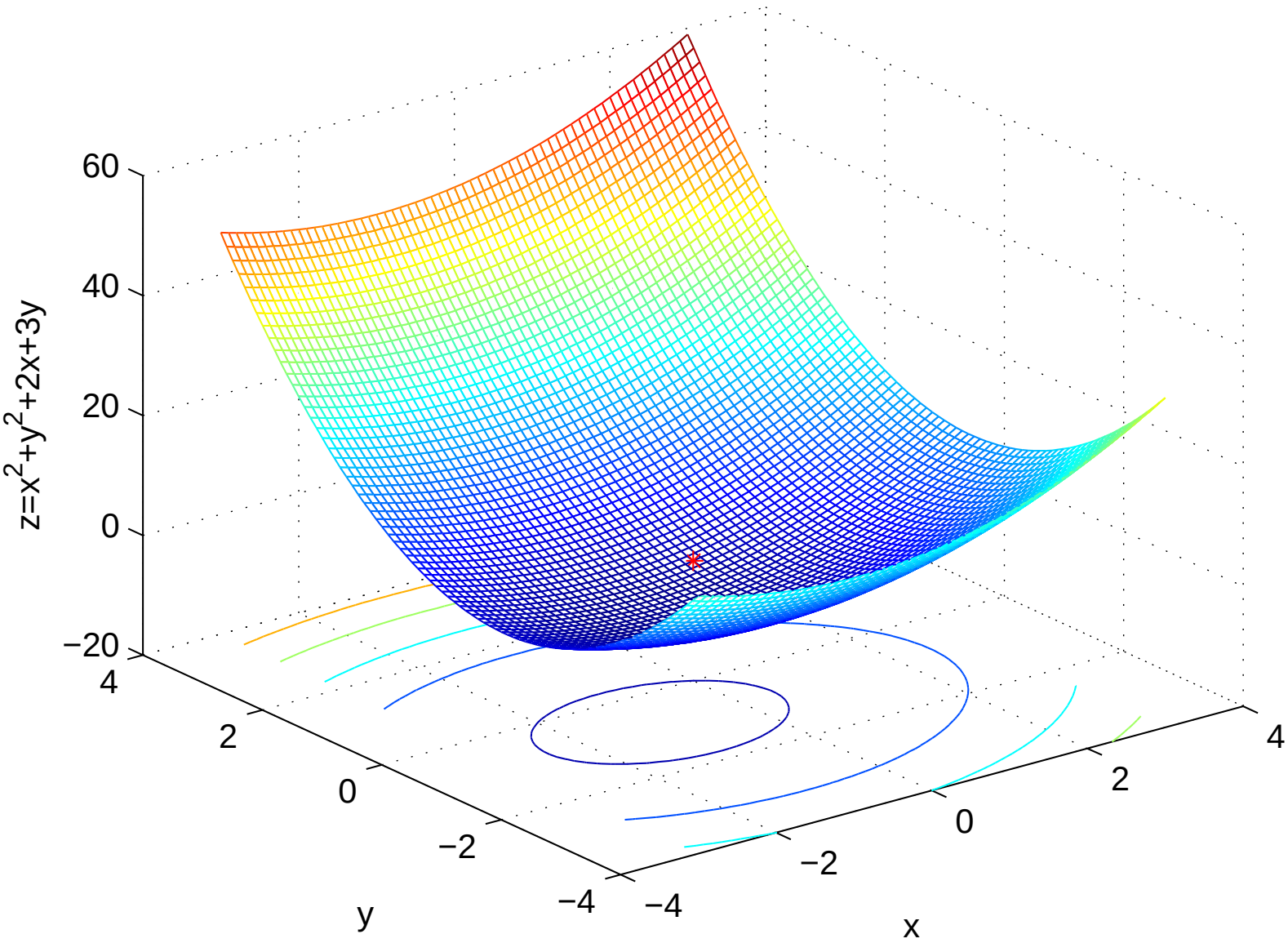
```
clear all;
clc

Q=[2,0;0,4];
q=[2,3]';
A=[1,2;2,1;0,1];
a=[8,10,3]';
lb=[0,0]';
ub=[inf;inf]';

options=optimset('quadprog');
options=optimset('LargeScale','off');
[xsol,fsolve,exitflag,output]=...
quadprog(Q,q,A,a,[],[],lb,ub,[],options)

fprintf('Convergence ');
if exitflag > 0
    fprintf('succeeded.\n');
    xsol
else
```

```
21      fprintf('failed.\n');
22  end
23  fprintf('Algorithm used: %s \n' ,output.algorithm);
24
25  x=-3:0.1:3;
26  y=-4:0.1:4;
27  [X,Y]=meshgrid(x,y);
28  Z=X.^2+2*Y.^2+2*X+3*Y;
29  meshc(X,Y,Z); hold on;
30  plot(xsol(1),xsol(2),'r*');
```


z=x^2+y^2+2x+3y
60
40
20
0
−20
4
2
0
−2
−4
y
−4
−2
0
2
4
x

Example 5

- Solve the following LP using **quadprog**:

$$
\begin{array}{llllllll}
x_1^2 + x_1x_2 + 2{x_2}^2 & +2x_3^2 & +2x_2x_3 & +4x_1+ & 6x_2 & +12x_3 & \rightarrow & \min \\
& \text{s.t.} & & & & & & \\
& x_1 & +x_2 & +x_3 & \geq & 6 & & \\
& -x_1 & -x_2 & +2x_3 & \geq & 2 & & \\
& x_1, & x_2 & x_3 & \geq & 0 & &
\end{array}
$$

$$Q = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 4 & 2 \\ 0 & 2 & 4 \end{pmatrix}, q = \begin{pmatrix} 4 \\ 6 \\ 12 \end{pmatrix}, A = \begin{pmatrix} -1 & -1 & -1 \\ 1 & 1 & -2 \end{pmatrix}, a = \begin{pmatrix} -6 \\ -12 \end{pmatrix},$$

$$B = [\,], b = [\,], lb = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, ub = \begin{pmatrix} \infty \\ \infty \\ \infty \end{pmatrix}$$

```
clear all;
clc

% Initialize
Q=[2,1,0;1,4,2;0,2,4];
q=[4,6,12];
A=[-1,-1,-1;1,2,-2];
a=[-6,-2];
lb=[0;0;0];
ub=[inf;inf;inf];

options=optimset('quadprog'); % choose quadratic ...
    programming
options=optimset('LargeScale','off');

[xsol,fsolve,exitflag,output]=...
quadprog(Q,q,A,a,[],[],lb,ub,[],options);

fprintf('Convergence ');
if exitflag > 0
```

```
20      fprintf('succeeded.\n');
21      xsol
22  else
23      fprintf('failed.\n');
24  end
25  fprintf('Algorithm used: %s \n' ,output.algorithm);
```

```
1   Optimization terminated.
2   Convergence succeeded.
3
4   xsol =
5
6       3.3333
7            0
8       2.6667
9
10  Algorithm used: medium-scale: active-set
```

Curve Fitting

- Curve fitting is the process of constructing a curve, or mathematical function, that has the best fit to a series of data points, possibly subject to constraints.
- Curve fitting requires a parametric model that relates the response data to the predictor data with one or more coefficients.
- The result of the fitting process is an estimate of the model coefficients.

Common Techniques

- Polynomial interpolation
  1. Newton form
  2. Lagrange form
  3. Polynomial splines
- Method of least squares
- You can find more details in curve fitting in the link: http://www.mathcs.emory.edu/~haber/math315/chap4.pdf.

Method of Least Squares

- The first clear and concise exposition of the method of least squares was published by Legendre in 1805.
- In 1809, Gauss published his method of calculating the orbits of celestial bodies.
- The method of least squares is a standard approach to the approximate solution of overdetermined systems, i.e., sets of equations in which there are more equations than unknowns.
- To obtain the coefficient estimates, the least-squares method minimizes the summed square of residuals.

More specific...

- Let $\{y_i\}_{i=1}^n$ be the observed response values and $\{\hat{y}_i\}_{i=1}^n$ be the fitted response values.
- Define the error or residual $e_i = y_i - \hat{y}_i$ for $i = 1, \ldots, n$.
- Then the sum of squares error estimates associated with the data is given by

$$S = \sum_{i=1}^{n} e_i^2. \tag{1}$$

- The common types of least-squares fitting include
  - linear least square,
  - nonlinear least squares.

Error Distributions

- When fitting data that contains random variations, there are two important assumptions that are usually made about the error:
  - The error exists only in the response data, and not in the predictor data.
  - The errors are random and follow a normal distribution with zero mean and constant variance σ^2, given by

$$e_i \sim n(0, \sigma^2).$$

  - $\{e_i\}$ is so-called independent and identically distributed, abbreviated by iid.

Why using normal distribution?

- The normal distribution is one of the probability distributions in which extreme random errors are uncommon[3].
- Statistical results such as confidence and prediction bounds do require normally distributed errors for their validity.
- If the mean of the errors is zero, then the errors are purely random.
  - If not, then it might be that the model is not the right choice for your data, or the errors are not purely random and contain systematic errors.
- A constant variance in the data implies that the "spread" of errors is constant.
  - Data that has the same variance is sometimes said to be of equal quality.

[3] Few outliers.

Linear Least Squares

- A linear model is defined as an equation that is linear in the coefficients.
- Suppose that you have n data points that can be modeled by a 1^{st}-order polynomial, given by

$$y = ax + b.$$

- By (1), $e_i = y_i - (ax_i + b)$.
- Now $S = \sum_{i=1}^{n} (y_i - (ax_i + b))^2$.
- The least-squares fitting process minimizes the summed square of the residuals.
- The coefficient a and b are determined by differentiating S with respect to each parameter, and setting the result equal to zero. (Why?)

- Hence,

$$\frac{\partial S}{\partial a} = -2\sum_{i=1}^{n} x_i(y_i - (ax_i + b)) = 0,$$
$$\frac{\partial S}{\partial b} = -2\sum_{i=1}^{n} (y_i - (ax_i + b)) = 0.$$

- The normal equations are defined as

$$a\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i,$$
$$a\sum_{i=1}^{n} x_i + nb = \sum_{i=1}^{n} y_i.$$

- Solving for a, b, we have

$$a = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2},$$

$$b = \frac{1}{n}(\sum_{i=1}^{n} y_i - a\sum_{i=1}^{n} x_i).$$

- In fact,

$$\begin{bmatrix} \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & n \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} x_i y_i \\ \sum_{i=1}^{n} y_i \end{bmatrix}.$$

Generalized Linear Least Squares

- In matrix form, linear models are given by the formula

$$y = Xb + \epsilon,$$

where y is an n-by-1 vector of responses, X is the n-by-m matrix for the model, b is m-by-1 vector of coefficients, and ϵ is an n-by-1 vector of errors.

- Then the normal equations are given by

$$(X^T X)b = X^T y,$$

where X^T is the transpose of the X.

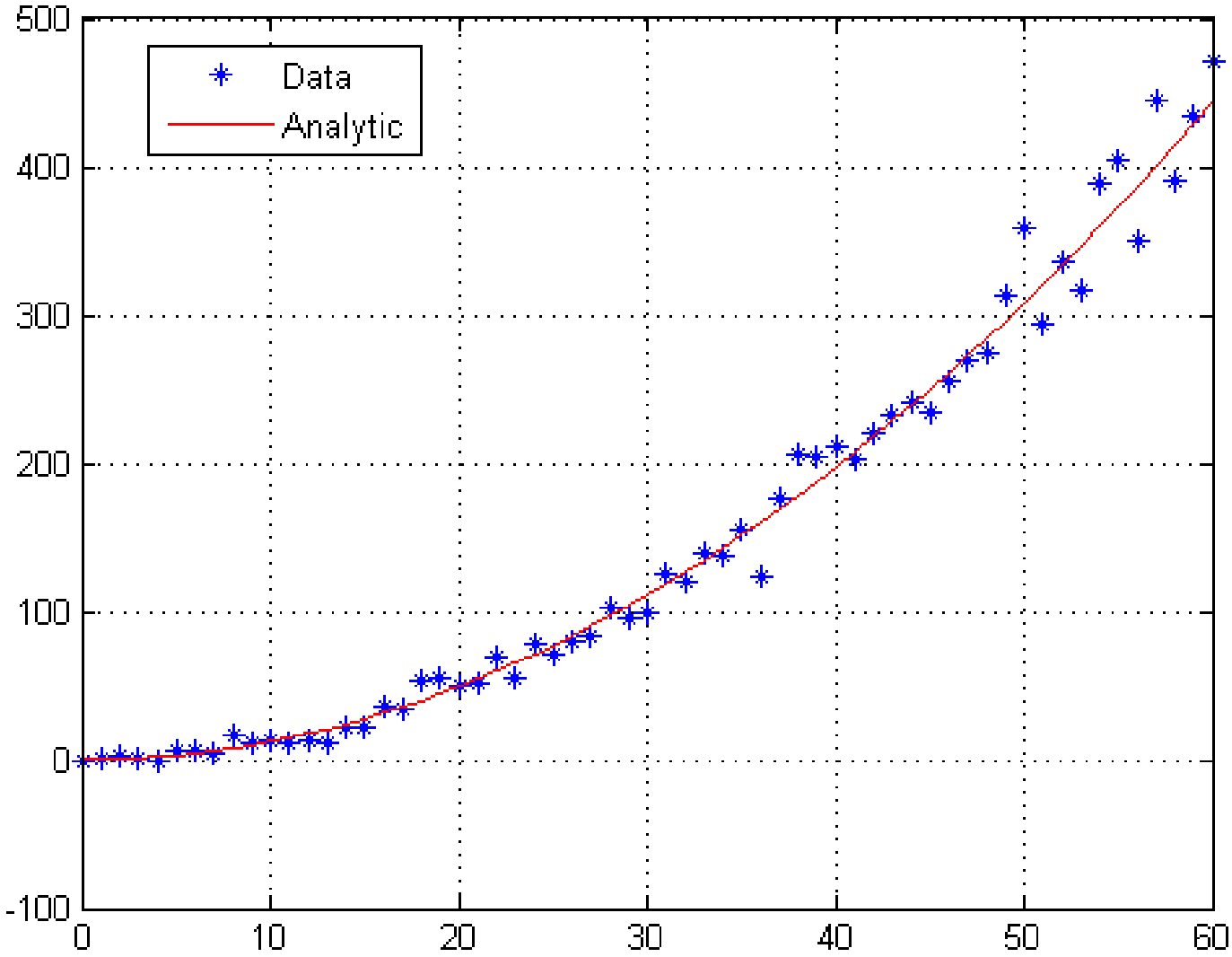
- So, $b = (X^T X)^{-1} X^T y$.

Example: Drag Coefficients

- Let v be the velocity of a moving object and k be a positive constant.
- The drag force due to air resistance is proportional to the square of the velocity, that is, $d = kv^2$.
- In a wind tunnel experiment, the velocity v can be varied by setting the speed of the fan and the drag can be measured directly.

- The following sequence of commands replicates the data one might receive from a wind tunnel:

```
clear all;
clc
% main
v=0:1:60;
d=.1234*v.^2;
dn=d+.4*v.*randn(size(v));
figure(1),plot(v,dn,'*',v,d,'r-'); grid on;
legend('Data','Analytic');
```


Data
Analytic
500
400
300
200
100
0
-100
0
10
20
30
40
50
60

- The unknown coefficient k is to be determined by the method of least squares.
- The formulation could be

$$\begin{cases} v_1^2 k &= dn_1 \\ v_2^2 k &= dn_2 \\ &\vdots \\ v_{61}^2 k &= dn_{61} \end{cases}.$$

- Recall that for any matrix A and vector b with $Ax = b$, $x = A\backslash b$ returns the least square solution.

```
1 >> k=(v.^2)'\dn'
2
3 k =
4
5       0.1239
```

Exercise: Chebyshev Approximation Problem (Revisited)

- Suppose the measurement of a real process over a 24 hours period be given by the following table with 14 data values:

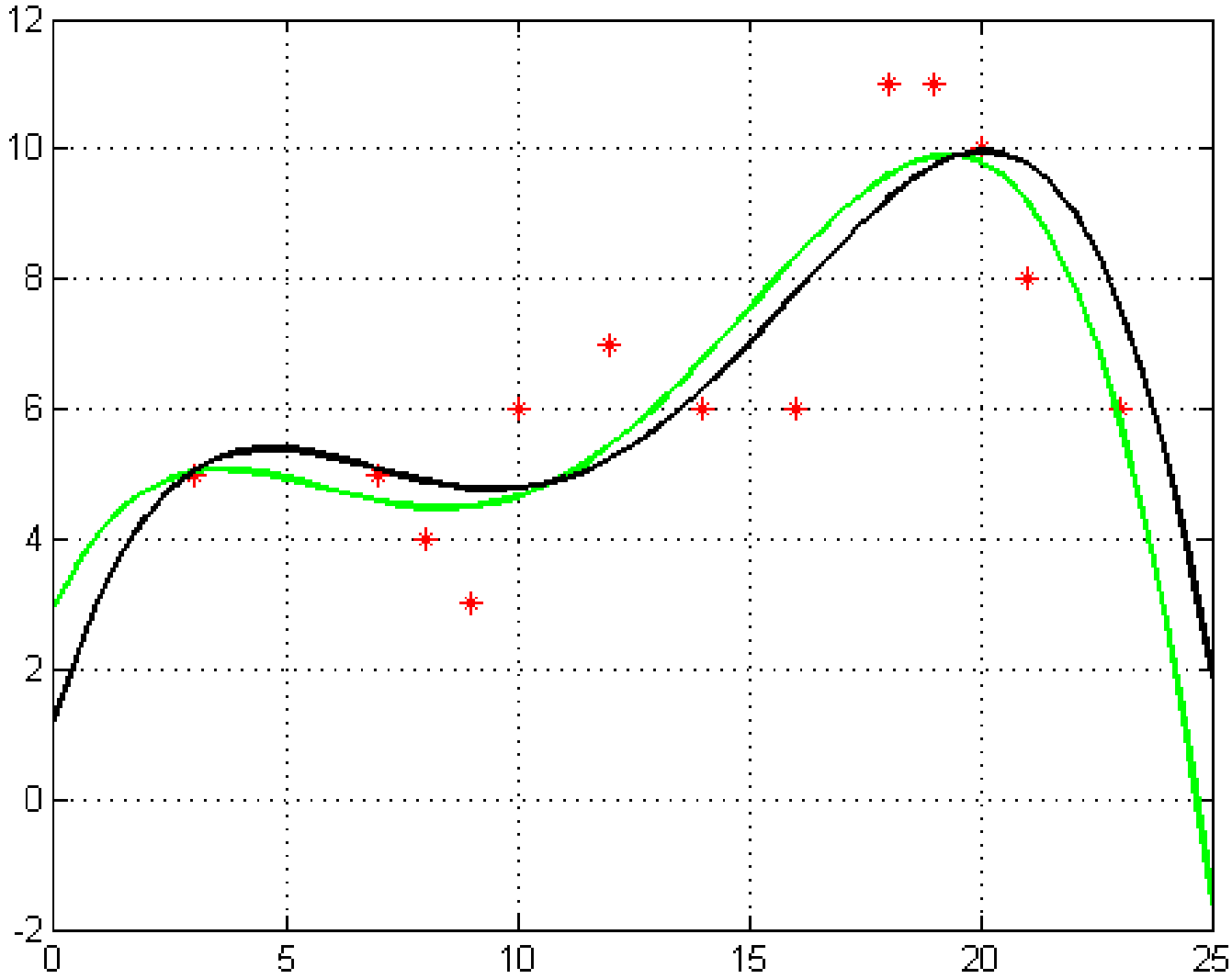
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
t_i	0	3	7	8	9	10	12	14	16	18	19	20	21	23
u_i	3	5	5	4	3	6	7	6	6	11	11	10	8	6

- The values t_i represent time and u_i's are measurements.
- Consider the polynomial $u(t) = at^4 + bt^3 + ct^2 + dt + e$.
- Please determine the coefficients a, b, c, e, d and e, so that the value of the function $u(t_i)$ could best approximate the discrete value u_i at t_i, $i = 1, \ldots, 14$ in the sense of least square error.

```
clear all;
clc
% main
t=[0,3,7,8,9,10,12,14,16,18,19,20,21,23]';
u=[3,5,5,4,3,6,7,6,6,11,11,10,8,6]';
plot(t,u,'r*'); hold on; grid on;
% least squares
X=[t.^4, t.^3, t.^2, t.^1, ones(length(u),1)]; % ...
    basis
b=X\u; % least-squares solution by matlab
tt=[0:0.5:25];
ut1=b(1)*(tt.^4)+b(2)*(tt.^3)+b(3)*(tt.^2)+b(4)*tt+b(5);
plot(tt,ut1,'-g','LineWidth',2)
% chebyshev
A1=[-t.^4,-t.^3,-t.^2,-t,-ones(14,1),-ones(14,1)];
A2=[t.^4,t.^3,t.^2,t,ones(14,1),-ones(14,1)];
c=zeros(6,1);
c(6)=1; % objective function coefficient (why?)
A=[A1;A2]; % inequality constraint matrix
```

```
19 a=[-u;u]; % right hand side vectro of ineq ...
       constraints
20 [xsol,fval,exitflag]=linprog(c,A,a);
21 ut2=xsol(1)*(tt.^4)+xsol(2)*(tt.^3)+xsol(3)*(tt.^2)+...
22 xsol(4)*tt+xsol(5);
23 plot(tt,ut2,'-k','LineWidth',2)
24 % Sum of square errors
25 sum_square_error=[sum((polyval(b,t')-u').^2)...
26 sum((polyval(xsol,t')-u').^2)]
```

```
1 sum_square_error =
2
3         17.596   1.9151e+005
```


12
10
8
6
4
2
0
-2
0
5
10
15
20
25

Linear Least Squares in MATLAB

- $[x, resnorm, residual, exitflag, output]$ =**lsqlin**$(C, d, A, b, \ldots$ $Aeq, beq, lb, ub, x0, options)$ returns a structure output that contains information about the optimization.
  - c: matrix
  - d: vector
  - $Aineq$: matrix for linear inequality constraints
  - $bineq$: vector for linear inequality constraints
  - Aeq: matrix for linear equality constraints
  - beq: vector for linear equality constraints
  - lb, ub: vector of lower/upper bounds
  - $x0$: initial point for x
  - $options$: using the **optimset** funciton which determines the details in the algorithm.

Example

```
>> C = [
     0.9501    0.7620    0.6153    0.4057
     0.2311    0.4564    0.7919    0.9354
     0.6068    0.0185    0.9218    0.9169
     0.4859    0.8214    0.7382    0.4102
     0.8912    0.4447    0.1762    0.8936];
>> d = [
     0.0578
     0.3528
     0.8131
     0.0098
     0.1388];
```

```
>> A =[
    0.2027    0.2721    0.7467    0.4659
    0.1987    0.1988    0.4450    0.4186
    0.6037    0.0152    0.9318    0.8462];
>> b =[
    0.5251
    0.2026
    0.6721];
>> lb = -0.1*ones(4,1);
>> ub = 2*ones(4,1);
>> [x,resnorm,residual,exitflag,output] = ...
                    lsqlin(C,d,A,b,[ ],[ ],lb,ub);
```

Summary: Built-in Functions (1/2)

- Linear and Quadratic Minimization problems:
  - **linprog**
  - **quadprog**
- Nonlinear zero finding (equation solving):
  - **fzero**
  - **fsolve**
- Linear least squares (of matrix problems):
  - **lsqlin**
  - **lsqnonneg**

Summary: Built-in Functions (2/2)

- Nonlinear minimization of functions:
  - **fminbnd**
  - **fmincon**
  - **fminsearch**
  - **fminunc**
  - **fseminf**
- Nonlinear least squares of functions:
  - **lsqcurvefit**
  - **lsqnonlin**
- Nonlinear minimization of multi-objective functions:
  - **fgoalattain**
  - **fminimax**

```
>> Lecture 8
>>
>>              -- Monte Carlo Simulation
>>
```

Contents

Random Variables

- A **random variable** is a function from a sample space Ω into the real numbers $\mathbb{R}$.
- There are lots of examples where the random variables are used.
  - In the experiment of tossing two dice once, a random variable X can be the sum of the numbers.
  - In the experiment of tossing a coin 25 times, a random variable X can be the number of heads in 25 tosses.

Distribution Functions

- With every random variable X, we associate a function called the **cumulative distribution function** of x.
  - Often we call it a **cdf** of X, denoted by $F_X(x) = P_X(X \leq x)$ for all x.
- The function $F_X(x)$ is a cdf if and only if the following three conditions hold:
  1. $\lim_{x \to -\infty} F_X(x) = 0$ and $\lim_{x \to \infty} F_X(x) = 1$.
  2. $F_X(x)$ is a nondecreasing function of x.
  3. $F_X(x)$ is right-continuous; that is, for every number x_0, $\lim_{x \to x_0^+} F_X(x) = F_X(x_0)$.

Random Variable (Revisited)

- A random variable X is continuous if $F_X(x)$ is a continuous function of x.
  - Probability density function (pdf) of X is defined by $p_X(x) = \frac{\partial F_X(x)}{\partial x}$.
- A random variable X is discrete if $F_X(x)$ is a step function of x.
  - Probability mass function (pmf) of X is a similar idea to a pdf, but in the discrete sense.
- Mixtures of both types also exist.

Table of Common Distributions

taken from *Statistical Inference* by Casella and Berger

Discrete Distrbutions

distribution	pmf	mean	variance	mgf/moment
Bernoulli(p)	$p^x(1-p)^{1-x};\ x=0,1;\ p\in(0,1)$	p	$p(1-p)$	$(1-p)+pe^t$
Beta-binomial(n,α,β)	$\binom{n}{x}\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\frac{\Gamma(x+\alpha)\Gamma(n-x+\beta)}{\Gamma(\alpha+\beta+n)}$	$\frac{n\alpha}{\alpha+\beta}$	$\frac{n\alpha\beta}{(\alpha+\beta)^2}$	
Notes: If $X\|P$ is binomial (n,P) and P is beta(α,β), then X is beta-binomial(n,α,β).				
Binomial(n,p)	$\binom{n}{x}p^x(1-p)^{n-x};\ x=1,\ldots,n$	np	$np(1-p)$	$[(1-p)+pe^t]^n$
Discrete Uniform(N)	$\frac{1}{N};\ x=1,\ldots,N$	$\frac{N+1}{2}$	$\frac{(N+1)(N-1)}{12}$	$\frac{1}{N}\sum_{i=1}^{N}e^{it}$
Geometric(p)	$p(1-p)^{x-1};\ p\in(0,1)$	$\frac{1}{p}$	$\frac{1-p}{p^2}$	$\frac{pe^t}{1-(1-p)e^t}$
Note: $Y=X-1$ is negative binomial$(1,p)$. The distribution is *memoryless*: $P(X>s\|X>t)=P(X>s-t)$.				
Hypergeometric(N,M,K)	$\frac{\binom{M}{x}\binom{N-M}{K-x}}{\binom{N}{K}};\ x=1,\ldots,K$	$\frac{KM}{N}$	$\frac{KM}{N}\frac{(N-M)(N-k)}{N(N-1)}$	?
	$M-(N-K)\le x\le M;\ N,M,K>0$			
Negative Binomial(r,p)	$\binom{r+x-1}{x}p^r(1-p)^x;\ p\in(0,1)$	$\frac{r(1-p)}{p}$	$\frac{r(1-p)}{p^2}$	$\left(\frac{p}{1-(1-p)e^t}\right)^r$
	$\binom{y-1}{r-1}p^r(1-p)^{y-r};\ Y=X+r$			
Poisson(λ)	$\frac{e^{-\lambda}\lambda^x}{x!};\ \lambda\ge 0$	λ	λ	$e^{\lambda(e^t-1)}$
Notes: If Y is gamma(α,β), X is Poisson$(\frac{x}{\beta})$, and α is an integer, then $P(X\ge\alpha)=P(Y\le y)$.				

Continuous Distributions

distribution	pdf	mean	variance	mgf/moment
Beta(α,β)	$\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}x^{\alpha-1}(1-x)^{\beta-1};\ x\in(0,1),\ \alpha,\beta>0$	$\frac{\alpha}{\alpha+\beta}$	$\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$	$1+\sum_{k=1}^{\infty}\left(\prod_{r=0}^{k-1}\frac{\alpha+r}{\alpha+\beta+r}\right)\frac{t^k}{k!}$
Cauchy(θ,σ)	$\frac{1}{\pi\sigma}\frac{1}{1+(\frac{x-\theta}{\sigma})^2};\ \sigma>0$	does not exist	does not exist	does not exist
Notes: Special case of Students's t with 1 degree of freedom. Also, if X,Y are iid $N(0,1)$, $\frac{X}{Y}$ is Cauchy				
χ^2_p	$\frac{1}{\Gamma(\frac{p}{2})2^{\frac{p}{2}}}x^{\frac{p}{2}-1}e^{-\frac{x}{2}};\ x>0,\ p\in N$	p	$2p$	$\left(\frac{1}{1-2t}\right)^{\frac{p}{2}},\ t<\frac{1}{2}$
Notes: Gamma$(\frac{p}{2},2)$.				
Double Exponential(μ,σ)	$\frac{1}{2\sigma}e^{-\frac{\lvert x-\mu\rvert}{\sigma}};\ \sigma>0$	μ	$2\sigma^2$	$\frac{e^{\mu t}}{1-(\sigma t)^2}$
Exponential(θ)	$\frac{1}{\theta}e^{-\frac{x}{\theta}};\ x\geq 0,\ \theta>0$	θ	θ^2	$\frac{1}{1-\theta t},\ t<\frac{1}{\theta}$
Notes: Gamma$(1,\theta)$. Memoryless. $Y=X^{\frac{1}{\gamma}}$ is *Weibull*. $Y=\sqrt{\frac{2X}{\beta}}$ is *Rayleigh*. $Y=\alpha-\gamma\log\frac{X}{\beta}$ is *Gumbel*.				
F_{ν_1,ν_2}	$\frac{\Gamma(\frac{\nu_1+\nu_2}{2})}{\Gamma(\frac{\nu_1}{2})\Gamma(\frac{\nu_2}{2})}\left(\frac{\nu_1}{\nu_2}\right)^{\frac{\nu_1}{2}}\frac{x^{\frac{\nu_1}{2}-2}}{\left(1+(\frac{\nu_1}{\nu_2})x\right)^{\frac{\nu_1+\nu_2}{2}}};\ x>0$	$\frac{\nu_2}{\nu_2-2},\ \nu_2>2$	$2(\frac{\nu_2}{\nu_2-2})^2\frac{\nu_1+\nu_2-2}{\nu_1(\nu_2-4)},\ \nu_2>4$	$EX^n=\frac{\Gamma(\frac{\nu_1+2n}{2})\Gamma(\frac{\nu_2-2n}{2})}{\Gamma(\frac{\nu_1}{2})\Gamma(\frac{\nu_2}{2})}\left(\frac{\nu_2}{\nu_1}\right)^n,\ n<\frac{\nu_2}{2}$
Notes: $F_{\nu_1,\nu_2}=\frac{\chi^2_{\nu_1}/\nu_1}{\chi^2_{\nu_2}/\nu_2}$, where the χ^2s are independent. $F_{1,\nu}=t^2_\nu$.				
Gamma(α,β)	$\frac{1}{\Gamma(\alpha)\beta^\alpha}x^{\alpha-1}e^{-\frac{x}{\beta}};\ x>0,\ \alpha,\beta>0$	$\alpha\beta$	$\alpha\beta^2$	$\left(\frac{1}{1-\beta t}\right)^\alpha,\ t<\frac{1}{\beta}$
Notes: Some special cases are exponential $(\alpha=1)$ and χ^2 $(\alpha=\frac{p}{2},\beta=2)$. If $\alpha=\frac{2}{3}$, $Y=\sqrt{\frac{X}{\beta}}$ is *Maxwell*. $Y=\frac{1}{X}$ is *inverted gamma*.				
Logistic(μ,β)	$\frac{1}{\beta}\frac{e^{-\frac{x-\mu}{\beta}}}{\left[1+e^{-\frac{x-\mu}{\beta}}\right]^2};\ \beta>0$	μ	$\frac{\pi^2\beta^2}{3}$	$e^{\mu t}\Gamma(1+\beta t),\ \lvert t\rvert<\frac{1}{\beta}$
Notes: The cdf is $F(x\lvert\mu,\beta)=\frac{1}{1+e^{-\frac{x-\mu}{\beta}}}$.				
Lognormal(μ,σ^2)	$\frac{1}{\sqrt{2\pi}\sigma}\frac{1}{x}e^{-\frac{(\log x-\mu)^2}{2\sigma^2}};\ x>0,\sigma>0$	$e^{\mu+\frac{\sigma^2}{2}}$	$e^{2(\mu+\sigma^2)}-e^{2\mu+\sigma^2}$	$EX^n=e^{n\mu+\frac{n^2\sigma^2}{2}}$
Normal(μ,σ^2)	$\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\mu)^2}{2\sigma^2}};\ \sigma>0$	μ	σ^2	$e^{\mu t+\frac{\sigma^2t^2}{2}}$
Pareto(α,β)	$\frac{\beta\alpha^\beta}{x^{\beta+1}};\ x>\alpha,\ \alpha,\beta>0$	$\frac{\beta\alpha}{\beta-1},\ \beta>1$	$\frac{\beta\alpha^2}{(\beta-1)^2(\beta-2)},\ \beta>2$	does not exist
t_ν	$\frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})}\frac{1}{\sqrt{\nu\pi}}\frac{1}{(1+\frac{x^2}{\nu})^{\frac{\nu+1}{2}}}$	$0,\ \nu>1$	$\frac{\nu}{\nu-2},\nu>2$	$EX^n=\frac{\Gamma(\frac{\nu+1}{2})\Gamma(\nu-\frac{n}{2})}{\sqrt{\pi}\Gamma(\frac{\nu}{2})}\nu^{\frac{n}{2}},\ n$ even
Notes: $t^2_\nu=F_{1,\nu}$.				
Uniform(a,b)	$\frac{1}{b-a},\ a\leq x\leq b$	$\frac{b+a}{2}$	$\frac{(b-a)^2}{12}$	$\frac{e^{bt}-e^{at}}{(b-a)t}$
Notes: If $a=0,\ b=1$, this is special case of beta $(\alpha=\beta=1)$.				
Weibull(γ,β)	$\frac{\gamma}{\beta}x^{\gamma-1}e^{-\frac{x^\gamma}{\beta}};\ x>0,\ \gamma,\beta>0$	$\beta^{\frac{1}{\gamma}}\Gamma(1+\frac{1}{\gamma})$	$\beta^{\frac{2}{\gamma}}\left[\Gamma(1+\frac{2}{\gamma})-\Gamma^2(1+\frac{1}{\gamma})\right]$	$EX^n=\beta^{\frac{n}{\gamma}}\Gamma(1+\frac{n}{\gamma})$
Notes: The mgf only exists for $\gamma\geq 1$.				

Pseudo Random Numbers in MATLAB

- **rand**(N) returns an N-by-N matrix containing pseudo-random values drawn from the continuous uniform distribution on the open interval $(0, 1)$.[4]
- **randi**($imax$, N) returns an N-by-N matrix containing pseudo-random values drawn from the discrete uniform distribution on the open interval $(0, imax)$.
- **randn**(N) returns an N-by-N matrix containing pseudo-random values drawn from the standard normal distribution with zero mean and unit variance.[5]
- **randperm**(n, k) returns a row vector containing k unique integers selected randomly from 1 to n.

[4] Usually denoted by $X \sim u(0, 1)$.
[5] Usually denoted by $X \sim n(0, 1)$.

Exercise

1. Generate n values from a continuous uniform distribution on the interval $[a, b]$ by extending **rand**.
2. Generate n values from a normal distribution with mean mu and standard deviation sig by extending **randn**.

```
1 function rv=rand_general(a,b,n) %[a,b] with n ...
      sampling points
2
3 rv=a+(b-a)*rand(n,1);
```

```
1 function rv=rand_general(mu,sig,n)
2
3 rv = mu + sig*randn(n,1);
```