

LINGO 教程完整版



LINGO 是用来求解线性和非线性优化问题的简易工具。LINGO 内置了一种建立最优化模型的语言，可以简便地表达大规模问题，利用 LINGO 高效的求解器可快速求解并分析结果。

1 快速入门

当你在 windows 下开始运行 LINGO 系统时，会得到类似下面的一个窗口：



外层是主框架窗口，包含了所有菜单命令和工具条，其它所有的窗口将被包含在主窗口之下。在主窗口内的标题为 LINGO Model - LINGO1 的窗口是 LINGO 的默认模型窗口，建立的模型都都要在该窗口内编码实现。下面举两个例子。

例 1.1 如何在 LINGO 中求解如下的 LP 问题：

$$\begin{aligned}
 \min \quad & 2x_1 + 3x_2 \\
 \text{s.t.} \quad & \\
 & x_1 + x_2 \geq 350 \\
 & x_1 \geq 100 \\
 & 2x_1 + x_2 \leq 600 \\
 & x_1, x_2 \geq 0
 \end{aligned}$$


在模型窗口中输入如下代码：

```
min=2*x1+3*x2;
```

```
x1+x2>=350;
```

```
x1>=100;
```

```
2*x1+x2<=600;
```

然后点击工具条上的按钮  即可。

例 1.2 使用 LINGO 软件计算 6 个发点 8 个收点的最小费用运输问题。产销单位运价如下表。

单位 销地 运价 产地	B1	B2	B3	B4	B5	B6	B7	B8	产量
A1	6	2	6	7	4	2	5	9	60
A2	4	9	5	3	8	5	8	2	55
A3	5	2	1	9	7	4	3	3	51
A4	7	6	7	3	9	2	7	1	43
A5	2	3	9	5	7	2	6	5	41
A6	5	5	2	2	8	1	4	3	52
销量	35	37	22	32	41	32	43	38	

使用 LINGO 软件，编制程序如下：

```
model:
```

```
!6 发点 8 收点运输问题;
```

```
sets:

    warehouses/wh1..wh6/: capacity;

    vendors/v1..v8/: demand;

    links(warehouses,vendors): cost, volume;

endsets

!目标函数;

    min=@sum(links: cost*volume);

!需求约束;

    @for(vendors(J):

        @sum(warehouses(I): volume(I,J))=demand(J));

!产量约束;

    @for(warehouses(I):

        @sum(vendors(J): volume(I,J))<=capacity(I));

!这里是数据;

data:

    capacity=60 55 51 43 41 52;

    demand=35 37 22 32 41 32 43 38;

    cost=6 2 6 7 4 2 9 5

           4 9 5 3 8 5 8 2

           5 2 1 9 7 4 3 3


           7 6 7 3 9 2 7 1
```

```
2 3 9 5 7 2 6 5
```

```
5 5 2 2 8 1 4 3;
```

```
enddata
```

```
end
```

然后点击工具条上的按钮 即可。

为了能够使用 LINGO 的强大功能，接着第二节的学习吧。

2 LINGO 中的集

对实际问题建模的时候，总会遇到一群或多群相联系的对象，比如工厂、消费者群体、交通工具和雇工等等。LINGO 允许把这些相联系的对象聚合成集（sets）。一旦把对象聚合成集，就可以利用集来最大限度的发挥 LINGO 建模语言的优势。

现在我们将深入介绍如何创建集，并用数据初始化集的属性。学完本节后，你对基于建模技术的集如何引入模型会有一个基本的理解。

2.1 为什么使用集

集是 LINGO 建模语言的基础，是程序设计最强有力的基本构件。借助于集，能够用一个单一的、长的、简明的复合公式表示一系列相似的约束，从而可以快速方便地表达规模较大的模型。

2.2 什么是集

集是一群相联系的对象，这些对象也称为集的成员。一个集可能是一系列产品、卡车或雇员。每个集成员可能有一个或多个与之有关联的特征，我们把这些特征称为属性。属性值可以预先给定，也可以

是未知的，有待于 LINGO 求解。例如，产品集中的每个产品可以有一个价格属性；卡车集中的每辆卡车可以有一个牵引力属性；雇员集中的每位雇员可以有一个薪水属性，也可以有一个生日属性等等。

LINGO 有两种类型的集：原始集(primitive set)和派生集(derived set)。

一个原始集是由一些最基本的对象组成的。

一个派生集是用一个或多个其它集来定义的，也就是说，它的成员来自于其它已存在的集。

2.3 模型的集部分

集部分是 LINGO 模型的一个可选部分。在 LINGO 模型中使用集之前，必须在集部分事先定义。集部分以关键字“sets:”开始，以“endsets”结束。一个模型可以没有集部分，或有一个简单的集部分，或多个集部分。一个集部分可以放置于模型的任何地方，但是一个集及其属性在模型约束中被引用之前必须定义了它们。

2.3.1 定义原始集

为了定义一个原始集，必须详细声明：

- 集的名字
- 可选，集的成员
- 可选，集成员的属性

定义一个原始集，用下面的语法：

```
setname[/member_list/][:attribute_list];
```

注意：用“[]”表示该部分内容可选。下同，不再赘述。

Setname 是你选择的来标记集的名字，最好具有较强的可读性。集名字必须严格符合标准命名规则：以拉丁字母或下划线（_）为首字符，其后由拉丁字母（A—Z）、下划线、阿拉伯数字（0, 1, …, 9）组成的总长度不超过 32 个字符的字符串，且不区分大小写。

注意：该命名规则同样适用于集成员名和属性名等的命名。

Member_list 是集成员列表。如果集成员放在集定义中，那么对它们可采取显式罗列和隐式罗列两种方式。如果集成员不放在集定义中，那么可以在随后的数据部分定义它们。

① 当显式罗列成员时，必须为每个成员输入一个不同的名字，中间用空格或逗号隔开，允许混合使用。

例 2.1 可以定义一个名为 students 的原始集，它具有成员 John、Jill、Rose 和 Mike，属性有 sex 和 age：

```
sets:
    students/John  Jill, Rose  Mike/: sex, age;
endsets
```

② 当隐式罗列成员时，不必罗列出每个集成员。可采用如下语法：

```
setname/member1..memberN/[: attribute_list];
```

这里的 member1 是集的第一个成员名，memberN 是集的最末一个成员名。LINGO 将自动产生中间的所有成员名。LINGO 也接受一些特定的首成员名和末成员名，用于创建一些特殊的集。列表如下：

隐式成员列表格式	示例	所产生集成员
1..n	1..5	1, 2, 3, 4, 5

StringM..StringN	Car2..car14	Car2, Car3, Car4, ..., Car14
DayM..DayN	Mon..Fri	Mon, Tue, Wed, Thu, Fri
MonthM..MonthN	Oct..Jan	Oct, Nov, Dec, Jan
MonthYearM..MonthYearN	Oct2001..Jan2002	Oct2001, Nov2001, Dec2001, Jan2002

③ 集成员不放在集定义中，而在随后的数据部分来定义。

例 2.2

!集部分;;

sets:

students:sex, age;

endsets

!数据部分;

data:

students, sex, age= John 1 16

Jill 0 14

Rose 0 17

Mike 1 13;

enddata

注意：开头用感叹号（!），末尾用分号（;）表示注释，可跨多行。

在集部分只定义了一个集 students，并未指定成员。在数据部分罗列了集成员 John、Jill、Rose 和 Mike，并对属性 sex 和 age 分别给出了值。

集成员无论用何种字符标记，它的索引都是从 1 开始连续计数。

在 `attribute_list` 可以指定一个或多个集成员的属性，属性之间必须用逗号隔开。

可以把集、集成员和集属性同 C 语言中的结构体作个类比。如下图：

集	←→	结构体
集成员	←→	结构体的域
集属性	←→	结构体实例

LINGO 内置的建模语言是一种描述性语言，用它可以描述现实世界中的一些问题，然后再借助于 LINGO 求解器求解。因此，集属性的值一旦在模型中被确定，就不可能再更改。在 LINGO 中，只有在初始部分中给出的集属性值在以后的求解中可更改。这与前面并不矛盾，初始部分是 LINGO 求解器的需要，并不是描述问题所必须的。

2.3.2 定义派生集

为了定义一个派生集，必须详细声明：

- 集的名字
- 父集的名字
- 可选，集成员
- 可选，集成员的属性

可用下面的语法定义一个派生集：

```
setname(parent_set_list) [/member_list/] [:attribute_list  
];
```

`setname` 是集的名字。`parent_set_list` 是已定义的集的列表，

多个时必须用逗号隔开。如果没有指定成员列表，那么 LINGO 会自动创建父集成员的所有组合作为派生集的成员。派生集的父集既可以是原始集，也可以是其它的派生集。

例 2.3

```
sets:
    product/A B/;
    machine/M N/;
    week/1..2/;
    allowed(product,machine,week):x;
endsets
```

LINGO 生成了三个父集的所有组合共八组作为 allowed 集的成员。列表如下：

编号	成员
1	
(A, M, 1)	
2	2
	(A, M
, 2)	
3	3
	(A, N
, 1)	
4	4

	(A, N
, 2)	
5	5
	(B, M
, 1)	
6	6
	(B, M
, 2)	
7	7
	(B, N
, 1)	
8	8
	(B, N
, 2)	

成员列表被忽略时，派生集成员由父集成员所有的组合构成，这样的派生集成为稠密集。如果限制派生集的成员，使它成为父集成员所有组合构成的集合的一个子集，这样的派生集成为稀疏集。同原始集一样，派生集成员的声明也可以放在数据部分。一个派生集的成员列表有两种方式生成：①显式罗列；②设置成员资格过滤器。当采用方式①时，必须显式罗列出所有要包含在派生集中的成员，并且罗列的每个成员必须属于稠密集。使用前面的例子，显式罗列派生集的成员：

```
allowed(product, machine, week) / A M 1, A N 2, B N 1 /;
```

如果需要生成一个大的、稀疏的集，那么显式罗列就很讨厌。幸运的是许多稀疏集的成员都满足一些条件以和非成员相区分。我们可以把这些逻辑条件看作过滤器，在 LINGO 生成派生集的成员时把使逻辑条件为假的成员从稠密集中过滤掉。

例 2.4

```
sets:
```

```
!学生集：性别属性 sex，1 表示男性，0 表示女性；年龄属性 age.    ;
```

```
students/John, Jill, Rose, Mike/:sex, age;
```

```
!男学生和女学生的联系集：友好程度属性 friend，[0，1]之间的数。    ;
```

```
linkmf(students, students) | sex(&1) #eq# 1 #and# sex(&2) #eq# 0: friend;
```

```
!男学生和女学生的友好程度大于 0.5 的集；
```

```
linkmf2(linkmf) | friend(&1, &2) #ge# 0.5 : x;
```

```
endsets
```

```
data:
```

```
sex, age = 1 16
```

```
0 14
```

```
0 17
```

```
0 13;
```

```
friend = 0.3 0.5 0.6;  
  
enddata
```

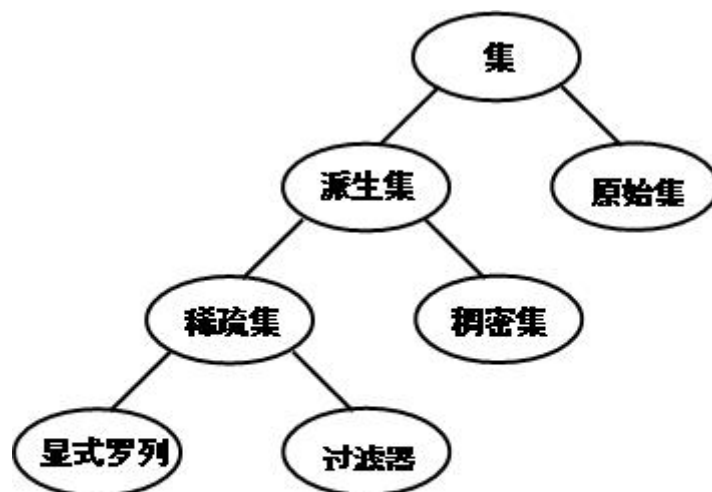
用竖线 (|) 来标记一个成员资格过滤器的开始。#eq#是逻辑运算符，用来判断是否“相等”，可参考 § 4. &1 可看作派生集的第 1 个原始父集的索引，它取遍该原始父集的所有成员；&2 可看作派生集的第 2 个原始父集的索引，它取遍该原始父集的所有成员；&3, &4, ……，以此类推。注意如果派生集 B 的父集是另外的派生集 A，那么上面所说的原始父集是集 A 向前回溯到最终的原始集，其顺序保持不变，并且派生集 A 的过滤器对派生集 B 仍然有效。因此，派生集的索引个数是最终原始父集的个数，索引的取值是从原始父集到当前派生集所作限制的总和。

总的来说，LINGO 可识别的集只有两种类型：原始集和派生集。

在一个模型中，原始集是基本的对象，不能再被拆分成更小的组分。原始集可以由显式罗列和隐式罗列两种方式来定义。当用显式罗列方式时，需在集成员列表中逐个输入每个成员。当用隐式罗列方式时，只需在集成员列表中输入首成员和末成员，而中间的成员由 LINGO 产生。

另一方面，派生集是由其它的集来创建。这些集被称为该派生集的父集（原始集或其它的派生集）。一个派生集既可以是稀疏的，也可以是稠密的。稠密集包含了父集成员的所有组合（有时也称为父集的笛卡尔乘积）。稀疏集仅包含了父集的笛卡尔乘积的一个子集，可通过显式罗列和成员资格过滤器这两种方式来定义。显式罗列方法就

是逐个罗列稀疏集的成员。成员资格过滤器方法通过使用稀疏集成员必须满足的逻辑条件从稠密集成员中过滤出稀疏集的成员。不同集类型的关系见下图。



LINGO 集类型

3 模型的数据部分和初始部分

在处理模型的数据时，需要为集指派一些成员并且在 LINGO 求解模型之前为集的某些属性指定值。为此，LINGO 为用户提供了两个可选部分：输入集成员和数据的数据部分（Data Section）和为决策变量设置初始值的初始部分（Init Section）。

3.1 模型的数据部分

3.1.1 数据部分入门

数据部分提供了模型相对静止部分和数据分离的可能性。显然，这对模型的维护和维数的缩放非常便利。

数据部分以关键字“data:”开始，以关键字“enddata”结束。在这里，可以指定集成员、集的属性。其语法如下：

```
object_list = value_list;
```

对象列 (object_list) 包含要指定值的属性名、要设置集成员的集名，用逗号或空格隔开。一个对象列中至多有一个集名，而属性名可以有任意多。如果对象列中有多个属性名，那么它们的类型必须一致。如果对象列中有一个集名，那么对象列中所有的属性的类型就是这个集。

数值列 (value_list) 包含要分配给对象列中的对象的值，用逗号或空格隔开。注意属性值的个数必须等于集成员的个数。看下面的例子。

例 3.1

```
sets:  
    set1/A, B, C/: X, Y;  
endsets  
  
data:  
    X=1, 2, 3;  
    Y=4, 5, 6;  
enddata
```

在集 set1 中定义了两个属性 X 和 Y。X 的三个值是 1、2 和 3，Y 的三个值是 4、5 和 6。也可采用如下例子中的复合数据声明 (data statement) 实现同样的功能。

例 3.2

```
sets:
```

```
set1/A, B, C/: X, Y;  
  
endsets  
  
data:  
  
    X, Y=1 4  
           2 5  
           3 6;  
  
enddata
```

看到这个例子，可能会认为 X 被指定了 1、4 和 2 三个值，因为它们是数值列中前三个，而正确的答案是 1、2 和 3。假设对象列有 n 个对象，LINGO 在为对象指定值时，首先在 n 个对象的第 1 个索引处依次分配数值列中的前 n 个对象，然后在 n 个对象的第 2 个索引处依次分配数值列中紧接着的 n 个对象，……，以此类推。

模型的所有数据——属性值和集成员——被单独放在数据部分，这可能是最规范的数据输入方式。

3.1.2 参数

在数据部分也可以指定一些标量变量（scalar variables）。当一个标量变量在数据部分确定时，称之为参数。看一例，假设模型中用利率 8.5% 作为一个参数，就可以象下面一样输入一个利率作为参数。

例 3.3

```
data:  
  
    interest_rate = .085;
```



```
enddata
```

也可以同时指定多个参数。

例 3.4

```
data:
```

```
    interest_rate, inflation_rate = .085 .03;
```

```
enddata
```

3.1.3 实时数据处理

在某些情况，对于模型中的某些数据并不是定值。譬如模型中有一个通货膨胀率的参数，我们想在 2%至 6%范围内，对不同的值求解模型，来观察模型的结果对通货膨胀的依赖有多么敏感。我们把这种情况称为实时数据处理（what if analysis）。LINGO 有一个特征可方便地做到这件事。

在本该放数的地方输入一个问号（?）。

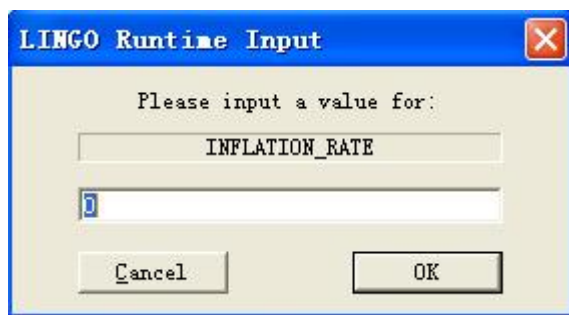
例 3.5

```
data:
```

```
    interest_rate, inflation_rate = .085    ?;
```

```
enddata
```

每一次求解模型时，LINGO 都会提示为参数 `inflation_rate` 输入一个值。在 WINDOWS 操作系统下，将会接收到一个类似下面的对话框：



直接输入一个值再点击 OK 按钮，LINGO 就会把输入的值指定给 `inflation_rate`，然后继续求解模型。

除了参数之外，也可以实时输入集的属性值，但不允许实时输入集成员名。

3.1.4 指定属性为一个值

可以在数据声明的右边输入一个值来把所有的成员的该属性指定为一个值。看下面的例子。

例 3.6

```
sets:
    days /MO, TU, WE, TH, FR, SA, SU/:needs;
endsets
data:
    needs = 20;
enddata
```

LINGO 将用 20 指定 `days` 集的所有成员的 `needs` 属性。对于多个属性的情形，见下例。

例 3.7

```
sets:
```

```
days /MO, TU, WE, TH, FR, SA, SU/:needs, cost;  
  
endsets  
  
data:  
  
    needs cost = 20 100;  
  
enddata
```

3.1.5 数据部分的未知数值

有时只想为一个集的部分成员的某个属性指定值，而让其余成员的该属性保持未知，以便让 LINGO 去求出它们的最优值。在数据声明中输入两个相连的逗号表示该位置对应的集成员的属性值未知。两个逗号间可以有空格。

例 3.8

```
sets:  
  
    years/1..5/: capacity;  
  
endsets  
  
data:  
  
    capacity = , 34, 20, , ;  
  
enddata
```

属性 capacity 的第 2 个和第 3 个值分别为 34 和 20，其余的未知。

3.2 模型的初始部分

初始部分是 LINGO 提供的另一个可选部分。在初始部分中，可以输入初始声明（initialization statement），和数据部分中的数据

声明相同。对实际问题的建模时，初始部分并不起到描述模型的作用，在初始部分输入的值仅被 LINGO 求解器当作初始点来用，并且仅仅对非线性模型有用。和数据部分指定变量的值不同，LINGO 求解器可以自由改变初始部分初始化的变量的值。

一个初始部分以“init:”开始，以“endinit”结束。初始部分的初始声明规则和数据部分的数据声明规则相同。也就是说，我们可以在声明的左边同时初始化多个集属性，可以把集属性初始化为一个值，可以用问号实现实时数据处理，还可以用逗号指定未知数值。

例 3.9

```
init:
    X, Y = 0, .1;
endinit
Y=@log(X);
X^2+Y^2<=1;
```

好的初始点会减少模型的求解时间。在这一节中，我们仅带大家接触了一些基本的数据输入和初始化概念，不过现在你应该可以轻松的为自己的模型加入原始数据和初始部分啦。

4 LINGO 函数

有了前几节的基础知识，再加上本节的内容，你就能够借助于 LINGO 建立并求解复杂的优化模型了。

LINGO 有 9 种类型的函数：

1. 基本运算符：包括算术运算符、逻辑运算符和关系运

算符

2. 数学函数：三角函数和常规的数学函数
3. 金融函数：LINGO 提供的两种金融函数
4. 概率函数：LINGO 提供了大量概率相关的函数
5. 变量界定函数：这类函数用来定义变量的取值范围
6. 集操作函数：这类函数为对集的操作提供帮助
7. 集循环函数：遍历集的元素，执行一定的操作的函数
8. 数据输入输出函数：这类函数允许模型和外部数据源

相联系，进行数据的输入输出

9. 辅助函数：各种杂类函数

4.1 基本运算符

这些运算符是非常基本的，甚至可以不认为它们是一类函数。事实上，在 LINGO 中它们是非常重要的。

4.1.1 算术运算符

算术运算符是针对数值进行操作的。LINGO 提供了 5 种二元运算符：

^ 乘方

* 乘

/ 除

+ 加

- 减

LINGO 唯一的一元算术运算符是取反函数“-”。

这些运算符的优先级由高到底为：

高 - （取反）

^

* /

低 + -

运算符的运算次序为从左到右按优先级高低来执行。运算的次序可以用圆括号“（）”来改变。

例 4.1 算术运算符示例。

$2 - 5 / 3$, $(2 + 4) / 5$ 等等。

4.1.2 逻辑运算符

在 LINGO 中，逻辑运算符主要用于集循环函数的条件表达式中，来控制在函数中哪些集成员被包含，哪些被排斥。在创建稀疏集时用在成员资格过滤器中。

LINGO 具有 9 种逻辑运算符：

#not# 否定该操作数的逻辑值，#not# 是一个一元运算符

#eq# 若两个运算数相等，则为 true；否则为 flase

#ne# 若两个运算符不相等，则为 true；否则为 flase

#gt# 若左边的运算符严格大于右边的运算符，则为 true；否则为 flase

#ge# 若左边的运算符大于或等于右边的运算符，则为 true；否则为 flase

#lt# 若左边的运算符严格小于右边的运算符，则为 true；

否则为 false

#le# 若左边的运算符小于或等于右边的运算符，则为 true；否则为 false

#and# 仅当两个参数都为 true 时，结果为 true，否则为 false

#or# 仅当两个参数都为 false 时，结果为 false；否则为 true

这些运算符的优先级由高到低为：

高 #not#

#eq# #ne# #gt# #ge# #lt# #le#

低 #and# #or#

例 4.2 逻辑运算符示例

$2 \#gt\# 3 \#and\# 4 \#gt\# 2$ ，其结果为假（0）。

4.1.3 关系运算符

在 LINGO 中，关系运算符主要是被用在模型中，来指定一个表达式的左边是否等于、小于等于、或者大于等于右边，形成模型的一个约束条件。关系运算符与逻辑运算符#eq#、#le#、#ge#截然不同，前者是模型中该关系运算符所指定关系的为真描述，而后者仅仅判断一个该关系是否被满足：满足为真，不满足为假。

LINGO 有三种关系运算符：“=”、“<=”和“>=”。LINGO 中还能用“<”表示小于等于关系，“>”表示大于等于关系。LINGO 并不支持严格小于和严格大于关系运算符。然而，如果需要严格小于和严格大于关系，比如让 A 严格小于 B：

$A < B$,

那么可以把它变成如下的小于等于表达式:

$A + \varepsilon \leq B$,

这里 ε 是一个小的正数, 它的值依赖于模型中 A 小于 B 多少才算不等。

下面给出以上三类操作符的优先级:

高 #not# - (取反)

^

* /

+ -

#eq# #ne# #gt# #ge# #lt# #le#

#and# #or#

低 <= = >=

4.2 数学函数

LINGO 提供了大量的标准数学函数:

@abs(x) 返回 x 的绝对值

@sin(x) 返回 x 的正弦值, x 采用弧

度制

@cos(x) 返回 x 的余弦值

@tan(x) 返回 x 的正切值

@exp(x) 返回常数 e 的 x 次方

@log(x) 返回 x 的自然对数

$@lgm(x)$ 返回 x 的 gamma 函数的自然对数

$@sign(x)$ 如果 $x < 0$ 返回 -1；否则，返回 1

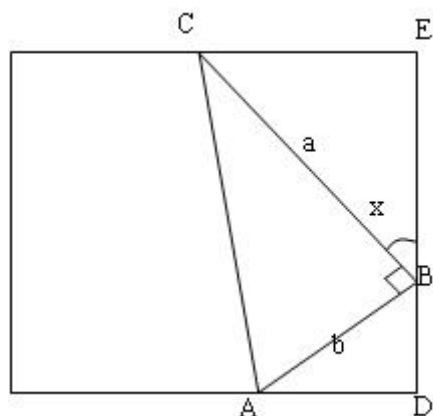
$@floor(x)$ 返回 x 的整数部分。当 $x \geq 0$ 时，返回不超过 x 的最大整数；当 $x < 0$ 时，返回不低于 x 的最大整数。

$@smax(x_1, x_2, \dots, x_n)$ 返回 x_1, x_2, \dots, x_n 中的最大值

$@smin(x_1, x_2, \dots, x_n)$ 返回 x_1, x_2, \dots, x_n 中的最小值

例 4.3 给定一个直角三角形，求包含该三角形的最小正方形。

解：如图所示。



求最小的正方形就相当于求如下的最优化问题：

LINGO 代码如下：

```
model:
```

```
sets:
```

```
    object/1..3/: f;
```

```
endsets
```

```
data:
```

```
a, b = 3, 4; !两个直角边长, 修改很方便;  
  
enddata  
  
f(1) = a * @sin(x);  
  
f(2) = b * @cos(x);  
  
f(3) = a * @cos(x) + b * @sin(x);  
  
min = @smax(f(1), f(2), f(3));  
  
@bnd(0, x, 1.57);  
  
end
```

在上面的代码中用到了函数@bnd, 详情请见 4.5 节。

4.3 金融函数

目前 LINGO 提供了两个金融函数。

1. @fpa(I, n)

返回如下情形的净现值: 单位时段利率为 I , 连续 n 个时段支付, 每个时段支付单位费用。若每个时段支付 x 单位的费用, 则净现值可用 x 乘以@fpa(I, n)算得。@fpa 的计算公式为

$$\sum_{k=1}^n \frac{1}{(1+I)^k} = \frac{1-(1+I)^{-n}}{I}。$$

净现值就是在一定时期内为了获得一定收益在该时期初所支付的实际费用。

例 4.4 贷款买房问题 贷款金额 50000 元, 贷款年利率 5.31%, 采取分期付款方式 (每年年末还固定金额, 直至还清)。问拟贷款 10 年, 每年需偿还多少元?

LINGO 代码如下：

$50000 = x * @fpa(.0531, 10);$

答案是 $x=6573.069$ 元。

2. $@fp1(I, n)$

返回如下情形的净现值：单位时段利率为 I ，第 n 个时段支付单位费用。 $@fp1(I, n)$ 的计算公式为

$$(1+I)^{-n}$$

细心的读者可以发现这两个函数间的关系：

$$@fpa(I, n) = \sum_{k=1}^n @fp1(I, k)。$$

4.4 概率函数

1. $@pbn(p, n, x)$

二项分布的累积分布函数。当 n 和（或） x 不是整数时，用线性插值法进行计算。

2. $@pcx(n, x)$

自由度为 n 的 χ^2 分布的累积分布函数。

3. $@peb(a, x)$

当到达负荷为 a ，服务系统有 x 个服务器且允许无穷排队时的 Erlang 繁忙概率。

4. $@pel(a, x)$

当到达负荷为 a ，服务系统有 x 个服务器且不允许排队时的 Erlang 繁忙概率。

5. @pfd(n, d, x)

自由度为 n 和 d 的 F 分布的累积分布函数。

6. @pfs(a, x, c)

当负荷上限为 a ，顾客数为 c ，平行服务器数量为 x 时，有限源的 Poisson 服务系统的等待或返修顾客数的期望值。 a 是顾客数乘以平均服务时间，再除以平均返修时间。当 c 和（或） x 不是整数时，采用线性插值进行计算。

7. @phg(pop, g, n, x)

超几何 (Hypergeometric) 分布的累积分布函数。 pop 表示产品总数， g 是正品数。从所有产品中任意取出 n ($n \leq pop$) 件。 pop , g , n 和 x 都可以是非整数，这时采用线性插值进行计算。

8. @ppl(a, x)

Poisson 分布的线性损失函数，即返回 $\max(0, z-x)$ 的期望值，其中随机变量 z 服从均值为 a 的 Poisson 分布。

9. @pps(a, x)

均值为 a 的 Poisson 分布的累积分布函数。当 x 不是整数时，采用线性插值进行计算。

10. @psl(x)

单位正态线性损失函数，即返回 $\max(0, z-x)$ 的期望值，其中随机变量 z 服从标准正态分布。

11. @psn(x)

标准正态分布的累积分布函数。

12. @ptd(n, x)

自由度为 n 的 t 分布的累积分布函数。

13. @qrand(seed)

产生服从 $(0, 1)$ 区间的拟随机数。@qrand 只允许在模型的数据部分使用，它将用拟随机数填满集属性。通常，声明一个 $m \times n$ 的二维表， m 表示运行实验的次数， n 表示每次实验所需的随机数的个数。在行内，随机数是独立分布的；在行间，随机数是非常均匀的。这些随机数是用“分层取样”的方法产生的。

例 4.5

```
model:
data:
    M=4; N=2; seed=1234567;
enddata
sets:
    rows/1..M/;
    cols/1..N/;
    table(rows,cols): x;
endsets
data:
    X=@qrand(seed);
enddata
end
```

如果没有为函数指定种子，那么 LINGO 将用系统时间构造种子。

14. @rand(seed)

返回 0 和 1 间的伪随机数，依赖于指定的种子。典型用法是 $U(I+1)=@rand(U(I))$ 。注意如果 seed 不变，那么产生的随机数也不变。

例 4.6 利用@rand 产生 15 个标准正态分布的随机数和自由度为 2 的 t 分布的随机数。

```
model:
```

```
!产生一系列正态分布和 t 分布的随机数;
```

```
sets:
```

```
    series/1..15/: u, znorm, zt;
```

```
endsets
```

```
!第一个均匀分布随机数是任意的;
```

```
u( 1) = @rand( .1234);
```

```
!产生其余的均匀分布的随机数;
```

```
@for(series( I) | I #GT# 1:
```

```
    u( I) = @rand( u( I - 1))
```

```
);
```

```
@for( series( I):
```

!正态分布随机数;

@psn(znorm(I)) = u(I);

!和自由度为 2 的 t 分布随机数;

@ptd(2, zt(I)) = u(I);

!ZNORM 和 ZT 可以是负数;

@free(znorm(I)); @free(zt(I));

);

end

4.5 变量界定函数

变量界定函数实现对变量取值范围的附加限制, 共 4 种:

@bin(x) 限制 x 为 0 或 1

@bnd(L, x, U) 限制 $L \leq x \leq U$

@free(x) 取消对变量 x 的默认下界为 0 的限制,

即 x 可以取任意实数

@gin(x) 限制 x 为整数

在默认情况下, LINGO 规定变量是非负的, 也就是说下界为 0, 上界为 $+\infty$ 。@free 取消了默认的下界为 0 的限制, 使变量也可以取负值。@bnd 用于设定一个变量的上下界, 它也可以取消默认下界为 0 的约束。

4.6 集操作函数

LINGO 提供了几个函数帮助处理集。

1、@in(set_name, primitive_index_1 [, primitive_index_2, ...])

如果元素在指定集中，返回 1；否则返回 0。

例 4.7 全集为 I，B 是 I 的一个子集，C 是 B 的补集。

```
sets:
```

```
    I/x1..x4/;
```

```
    B(I)/x2/;
```

```
    C(I) | #not#@in(B, &1) ;;
```

```
endsets
```

2. @index([set_name,] primitive_set_element)

该函数返回在集 set_name 中原始集成员 primitive_set_element 的索引。如果 set_name 被忽略，那么 LINGO 将返回与 primitive_set_element 匹配的第一个原始集成员的索引。如果找不到，则产生一个错误。

例 4.8 如何确定集成员 (B, Y) 属于派生集 S3。

```
sets:
```

```
    S1/A B C/;
```

```
    S2/X Y Z/;
```

```
    S3(S1, S2)/A X, A Z, B Y, C X/;
```

```
endsets
```

```
X=@in(S3, @index(S1, B), @index(S2, Y));
```

看下面的例子，表明有时为 @index 指定集是必要的。

例 4.9

```
sets:
```



```
girls/debble, sue, alice/;  
  
boys/bob, joe, sue, fred/;  
  
endsets
```

```
I1=@index(sue);
```

```
I2=@index(boys, sue);
```

I1 的值是 2，I2 的值是 3。我们建议在使用@index 函数时最好指定集。

3. @wrap(index, limit)

该函数返回 $j = \text{index} - k * \text{limit}$ ，其中 k 是一个整数，取适当值保证 j 落在区间 $[1, \text{limit}]$ 内。该函数相当于 index 模 limit 再加 1。该函数在循环、多阶段计划编制中特别有用。

4. @size(set_name)

该函数返回集 set_name 的成员个数。在模型中明确给出集大小时最好使用该函数。它的使用使模型更加数据中立，集大小改变时也更易维护。

4.7 集循环函数

集循环函数遍历整个集进行操作。其语法为

```
@function(setname[(set_index_list)[|conditional_qualifier]]:  
  
expression_list);
```

@function 相应于下面罗列的四个集循环函数之一；setname 是要遍历的集；set_index_list 是集索引列表；

`conditional_qualifier` 是用来限制集循环函数的范围，当集循环函数遍历集的每个成员时，LINGO 都要对 `conditional_qualifier` 进行评价，若结果为真，则对该成员执行 `@function` 操作，否则跳过，继续执行下一次循环。`expression_list` 是被应用到每个集成员的表达式列表，当用的是 `@for` 函数时，`expression_list` 可以包含多个表达式，其间用分号隔开。这些表达式将被作为约束加到模型中。当使用其余的三个集循环函数时，`expression_list` 只能有一个表达式。如果省略 `set_index_list`，那么在 `expression_list` 中引用的所有属性的类型都是 `setname` 集。

1. @for

该函数用来产生对集成员的约束。基于建模语言的标量需要显式输入每个约束，不过 `@for` 函数允许只输入一个约束，然后 LINGO 自动产生每个集成员的约束。

例 4.10 产生序列 {1, 4, 9, 16, 25}

```
model:

sets:

    number/1..5/:x;

endsets

@for(number(I): x(I)=I^2);

end
```

2. @sum

该函数返回遍历指定的集成员的一个表达式的和。

例 4.11 求向量[5, 1, 3, 4, 6, 10]前 5 个数的和。

```
model:

data:

    N=6;

enddata

sets:

    number/1..N/:x;

endsets

data:

    x = 5 1 3 4 6 10;

enddata

s=@sum(number(I) | I #le# 5: x);

end
```

3. @min 和@max

返回指定的集成员的一个表达式的最小值或最大值。

例 4.12 求向量[5, 1, 3, 4, 6, 10]前 5 个数的最小值，后 3 个数的最大值。

```
model:

data:

    N=6;

enddata

sets:
```

```
number/1..N/:x;

endsets

data:

    x = 5 1 3 4 6 10;

enddata

minv=@min(number(I) | I #le# 5: x);

maxv=@max(number(I) | I #ge# N-2: x);

end
```

下面看一个稍微复杂一点儿的例子。

例 4.13 职员时序安排模型 一项工作一周 7 天都需要有人(比如护士工作)，每天(周一至周日)所需的最少职员数为 20、16、13、16、19、14 和 12，并要求每个职员一周连续工作 5 天，试求每周所需最少职员数，并给出安排。注意这里我们考虑稳定后的情况。

```
model:

sets:

    days/mon..sun/: required,start;

endsets

data:

    !每天所需的最少职员数;

    required = 20 16 13 16 19 14 12;

enddata

!最小化每周所需职员数;
```

```

min=@sum(days: start);

@for(days(J):

    @sum(days(I) | I #le# 5:

        start(@wrap(J+I+2, 7))) >= required(J));

end

```

5 综合举例

例 5.1 求解非线性方程组

$$\begin{cases} x^2 + y^2 = 2 \\ 2x^2 + x + y^2 - y = 4 \end{cases}$$

其 LINGO 代码如下:

```

model:

    x^2+y^2=2;

    2*x^2+x+y^2-y=4;

end

```

计算的部分结果为

Variable		Value
	X	1.221330
	Y	0.7129758
Row		Slack or Surplus
	1	-0.1914895E-04
	2	-0.1892746E-04

例 5.2 装配线平衡模型 一条装配线含有一系列的工作站，在

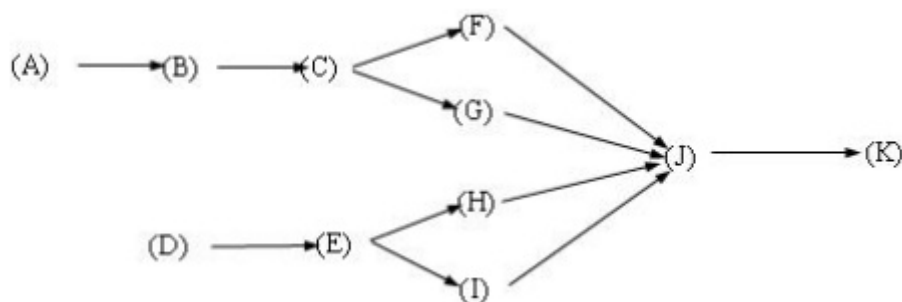
最终产品的加工过程中每个工作站执行一种或几种特定的任务。装配线周期是指所有工作站完成分配给它们各自的任务所化费时间中的最大值。平衡装配线的目标是为每个工作站分配加工任务，尽可能使每个工作站执行相同数量的任务，其最终标准是装配线周期最短。不适当的平衡装配线将会产生瓶颈——有较少任务的工作站将被迫等待其前面分配了较多任务的工作站。

问题会因为众多任务间存在优先关系而变得更复杂，任务的分配必须服从这种优先关系。

这个模型的目标是最小化装配线周期。有 2 类约束：

- ① 要保证每件任务只能也必须分配至一个工作站来加工；
- ② 要保证满足任务间的所有优先关系。

例 有 11 件任务（A—K）分配到 4 个工作站（1—4），任务的优先次序如下图。



每件任务所花费的时间如下表。

任务	A	B	C	D	E	F	G	H	I	J	K
时间	45	11	9	50	15	12	12	12	12	8	9

MODEL:

!装配线平衡模型;

SETS:

!任务集合，有一个完成时间属性 T;

TASK/ A B C D E F G H I J K/: T;

!任务之间的优先关系集合 (A 必须完成才能开始 B, 等等);

PRED(TASK, TASK)/ A,B B,C C,F C,G F,J G,J
J,K D,E E,H E,I H,J I,J /;

! 工作站集合;

STATION/1..4/;

TXS(TASK, STATION): X;

! X 是派生集合 TXS 的一个属性。如果 $X(I, K) = 1$, 则表示第 I 个任务

指派给第 K 个工作站完成;

ENDSETS

DATA:

! 任 务
A B C D E F G H I J K 的完成时
间估计如下;

T
= 45 11 9 50 15 12 12 12 12 8 9;

ENDDATA

! 当任务超过 15 个时，模型的求解将变得很慢;

!每一个作业必须指派到一个工作站，即满足约束①;

@FOR(TASK(I): @SUM(STATION(K): X(I, K)) = 1);

!对于每一个存在优先关系的作业来说, 前者对应的工作站 I 必须小于后

者对应的工作站 J, 即满足约束②;

```
@FOR( PRED( I, J): @SUM( STATION( K): K * X( J, K) - K * X( I, K)) >= 0);
```

!对于每一个工作站来说, 其花费时间必须不大于装配线周期;

```
@FOR( STATION( K):
```

```
@SUM( TXS( I, K): T( I) * X( I, K)) <= CYCTIME);
```

!目标函数是最小化转配线周期;

```
MIN = CYCTIME;
```

!指定 X(I, J) 为 0/1 变量;

```
@FOR( TXS: @BIN( X));
```

```
END
```

例 5.3 旅行售货员问题 (又称货郎担问题, Traveling Salesman Problem)

有一个推销员, 从城市 1 出发, 要遍访城市 2, 3, ..., n 各一次, 最后返回城市 1。已知从城市 i 到 j 的旅费为 c_{ij} , 问他应按怎样的次序访问这些城市, 使得总旅费最少?

可以用多种方法把 TSP 表示成整数规划模型。这里介绍的一种建立模型的方法, 是把该问题的每个解 (不一定是最优的) 看作是一次“巡回”。

在下述意义下，引入一些 0-1 整数变量：

$$x_{ij} = \begin{cases} 1, & \text{从 } i \text{ 到 } j, i \neq j \\ 0, & \text{其他} \end{cases}$$

其目标只是使

$$\sum_{i,j=1}^n c_{ij} x_{ij}$$

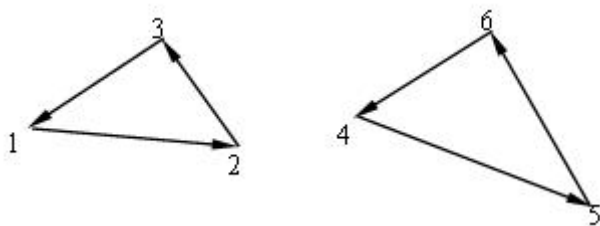
为最小。

这里有两个明显的必须满足的条件：访问城市 i 后必须要有一个即将访问的确切城市；访问城市 j 前必须要有一个刚刚访问过的确切城市。用下面的两组约束分别实现上面的两个条件

$$\sum_{j=1}^n x_{ij} = 1, \quad i = 1, 2, \dots, n \quad \sum_{i=1}^n x_{ij} = 1, \quad j = 1, 2, \dots, n$$

到此我们得到了一个模型，它是一个指派问题的整数规划模型。

但以上两个条件对于 TSP 来说并不充分，仅仅是必要条件。例如：



以上两个条件都满足，但它显然不是 TSP 的解，它存在两个子巡回。

这里，我们将叙述一种在原模型上附加充分的约束条件以避免产生子巡回的方法。把额外变量 附加到问题中。可把这些变量看作是连续的（最然这些变量在最优解中取普通的整数值）。现在附加下面形式的约束条件

$$u_i - u_j + n x_{ij} \leq n-1, \quad 2 \leq i \neq j \leq n.$$

为了证明该约束条件有预期的效果，必须证明：（1）任何含子巡回的路线都不满足该约束条件；（2）全部巡回都满足该约束条件。

首先证明（1），用反证法。假设还存在子巡回，也就是说至少有两个子巡回。那么至少存在一个子巡回中不含城市 1。把该子巡回记为 $i_1 i_2 \cdots i_k i_1$ ，则必有

$$u_{i_1} - u_{i_2} + n \leq n-1$$

$$u_{i_2} - u_{i_3} + n \leq n-1$$

...

$$u_{i_k} - u_{i_1} + n \leq n-1$$

把这 k 个式子相加，有

$$n \leq n-1, \quad \text{矛盾!}$$

故假设不正确，结论（1）得证。

下面证明（2），采用构造法。对于任意的总巡回，可取访问城市 i 的顺序数，取值范围为。

因此，。下面来证明总巡回满足该约束条件。

（i）总巡回上的边

$$\begin{cases} u_{i_1} - u_{i_2} + n = n-1 \leq n-1 \\ u_{i_2} - u_{i_3} + n = n-1 \leq n-1 \\ \cdots \\ u_{i_{n-2}} - u_{i_{n-1}} + n = n-1 \leq n-1 \end{cases}$$

（ii）非总巡回上的边

$$\begin{cases} u_{i_r} - u_j \leq n-2 \leq n-1, & r=1, 2, \cdots, n-2, \quad j \in \{2, 3, \cdots, n\} - \{i_r, i_{r+1}\} \\ u_{i_{n-1}} - u_j \leq n-2 \leq n-1, & j \in \{2, 3, \cdots, n\} - \{i_r\} \end{cases}$$

从而结论 (2) 得证。

这样我们把 TSP 转化成了一个混合整数线性规划问题。

$$\left\{ \begin{array}{l} \min \quad z = \sum_{i,j=1}^n c_{ij} x_{ij} \\ s.t. \quad \sum_{i=1}^n x_{ij} = 1, \quad j = 1, 2, \dots, n \\ \sum_{j=1}^n x_{ij} = 1, \quad i = 1, 2, \dots, n \\ u_i - u_j + n x_{ij} \leq n - 1, \quad 2 \leq i \neq j \leq n \\ x_{ij} = 0, 1, \quad i, j = 1, 2, \dots, n \\ u_i \geq 0, \quad i = 2, 3, \dots, n \end{array} \right.$$

显然，当城市个数较大（大于 30）时，该混合整数线性规划问题的规模会很大，从而给求解带来很大问题。TSP 已被证明是 NP 难问题，目前还没有发现多项式时间的算法。对于小规模问题，我们求解这个混合整数线性规划问题的方式还是有效的。

TSP 是一个重要的组合优化问题，除了有直观的应用外，许多其它看似无联系的优化问题也可转化为 TSP。例如：

问题 1 现需在一台机器上加工 n 个零件（如烧瓷器），这些零件可按任意先后顺序在机器上加工。我们希望加工完成所有零件的总时间尽可能少。由于加工工艺的要求，加工零件 j 时机器必须处于相应状态 s_j （如炉温）。设起始未加工任何零件时机器处于状态 s_0 ，且当所有零件加工完成后需恢复到 s_0 状态。已知从状态 s_i 调整到状态 s_j 需要时间 c_{ij} 。零件 j 本身加工时间为 p_j 。为方便起见，引入一个虚零件 0，其加工时间为 0，要求状态为 s_0 ，则 $\{0, 1, 2, \dots, n\}$ 的一个圈置换 π 就表示对所有零件的一个加工顺序，在此置换下，

完成所有加工所需要的总时间为

$$\sum_{i=0}^n (c_{i\pi(i)} + p_{\pi(i)}) = \sum_{i=0}^n c_{i\pi(i)} + \sum_{j=0}^n p_j.$$

由于

$$\sum_{j=0}^n p_j$$

是一个常数，故该零件的加工顺序问题变成 TSP。

!旅行售货员问题;

model:

sets:

city / 1.. 5/: u;

link(city, city):

dist, ! 距离矩阵;

x;

endsets

n = @size(city);

data: !距离矩阵，它并不需要是对称的;

dist = @qrand(1); !随机产生，这里可改为你要解决的问题的数据;

题的数据;

enddata

!目标函数;

min = @sum(link: dist * x);

```
@FOR( city( K):  
    !进入城市 K;  
    @sum( city( I) | I #ne# K: x( I, K)) = 1;  
    !离开城市 K;  
    @sum( city( J) | J #ne# K: x( K, J)) = 1;  
);  
!保证不出现子圈;  
@for(city(I) | I #gt# 1:  
    @for( city( J) | J#gt#1 #and# I #ne# J:  
        u(I)-u(J)+n*x(I, J)<=n-1);  
);  
!限制 u 的范围以加速模型的求解, 保证所加限制并不排除掉  
TSP 问题的最优解;  
@for(city(I) | I #gt# 1: u(I)<=n-2 );  
!定义 X 为 0\1 变量;  
@for( link: @bin( x));  
end
```