数学建模系列

马世拓

数学实验与数学建模

基于 Baltamatica

第 *0* 版

Springer

目录

第一章　数学建模的国产化之路

那么从这一章起我们将开始基于 Baltamatica 的数学建模学习了。大家可能会问，马老师之前讲过基于 MATLAB 的建模，讲过基于 Python 的建模，为什么又要来上这么一个从未听过的软件呢？这里我们是为了向同学们介绍一种国产的全新方案，并且向同学们展示：无论是什么工具做起来其方法论都是相似的。即使就是这样一个刚刚成型的软件，只要我把握住一个学习路径和学习策略，一样是可以进行数学建模实践的。

1.1　什么是数学建模

何谓“数学建模”？我想这个问题其实并没有一个明确的定义。老宗师姜启源先生说，“数学建模就是建立数学模型解决实际问题”，我却总感觉这个定义还是抽象了些。“建立数学模型”，什么是“数学”的“模型”？怎样建这个“模型”？用什么去建这个“模型”？在我刚开始读本科的时候，我也对这个问题感到很疑惑，于是去拜读了很多人的作品，诸如姜启源、谢金星二位先生的《数学模型》、司守奎先生的《数学建模算法与应用》等很多经典教材，隐约是能够觉得，这个“数学”的“模型”本质上还是一些数学知识，而建这个“模型”则不能手算，须用计算机去设计程序。至少姜先生告诉了我数模的本质，司先生告诉了我数模的方法。

可道理和工具都有了，应该如何建立这个“模型”呢？

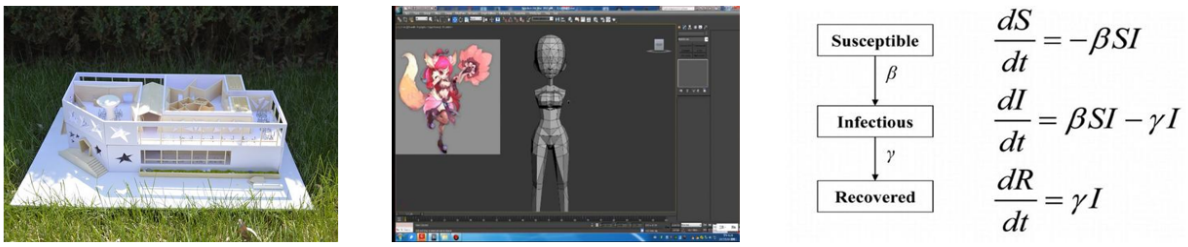


图 1.1: 人类认识世界的三种模型

这个问题我从 2020 年起就在思考，一直思考至今。朋友，请不要被“数学建模”这四个字劝退，它的本质还是解应用题，只是曾经的“小明买糖”变成了如今的“嫦娥探月”。可数学建模若是仅仅作为解题的手段而存在，那我便不会愿意花上四年时间去学它。数学建模作为一种思

想，它是真真切切在工程问题中有所应用的。

我想，一些照本宣科的老师若是来上数学建模这门课，给你们讲的第一个例子大概率是船划水或包饺子。我不愿意照本宣科，也并不打算去介绍一些你们听着简单无聊可在实际工程中却没多大用处的例子。我便以身边最近的一个例子介绍吧：

2020 年初，新冠病毒的到来打乱了我们的正常生活。在新冠肺炎的防治过程中就充满了数学建模的影子：如果读者翻到后面，我会介绍 SEIR 模型，也就是传染病模型，我们把人群分为易感人群、密切接触者、感染者和康复者四类人群，这四类人群的新增、减少都遵循着动力系统的一些规律，我们可以列出一个微分方程组对其进行模拟：

$$\begin{cases} N\frac{ds}{dt} = -N\lambda si \\ N\frac{de}{dt} = N\lambda si - N\delta e \\ N\frac{di}{dt} = N\delta e - N\mu i \\ N\frac{dr}{dt} = N\mu i \\ s(t) + e(t) + i(t) + r(t) = 1 \end{cases} \tag{1.1}$$

读者现在可能并不知道这个方程组是什么意思，没有关系，在后续的学习中我们会一点点解释。而除了解微分方程组以外，我想很多读者还见过“一个教室里面的同学如果封锁了教室传染病会如何传染”这样的动画，而这样的动画可以通过元胞自动机去进行模拟。医生在研发新药、在患者身上进行试验的时候，会收集相应的实验数据，而对这些实验数据的分析、假设检验则又充满了统计学知识的应用……

所以，你看到了，这个过程充满了数学建模。

一个好的模型，它能够准确地反映问题，但又不失简洁性。我历来信奉一条最根本的道义，那就是“大道至简”。什么样的模型可以算得上一个好模型呢？我认为它需要遵循以下要点：

- 形式简洁：模型不至于太冗长，大道至简。
- 精度到位：求解精度符合工程实际的要求。
- 理论创新：在理论层面上进行一些创新。
- 排除干扰：能够排除一些无关紧要的干扰项。
- 可解释性：模型的结果有良好的可解释性。
- 求解方便：模型能够利用各种求解工具进行求解。

数学建模无处不在，在各路研究中都充满了应用。

数学建模的本质就是一种量化研究的思想，在物理与力学中的微分方程与动力系统是数学建模，运筹优化生产安排中的优化模型也是数学建模，股市投资与价格的预测也是数学建模，甚至哪怕一份调查问卷，也可以变成数学建模问题。

数学建模我希望各位不要把它过于神化，也不要过于弱化，它下不保底上不封顶。只要是能够用数学的思想思维，分析问题中的数与量、变与不变、等与不等，这样的一种思维就是数学思维，建立起来的也就是数学模型。而没有任何软件可以说是数学建模的唯一工具，更不等于数学建模本身。编程只是工具，模型与方法论才是灵魂。

1.2 数学建模的软件生态

能够进行数学建模的软件有很多，比如 MATLAB、Python、SPSS、R 等，它们在不同的性能上各有优劣。就目前的情况而言，数学建模的生态最好的是由美国 Mathworks 公司开发的 MATLAB&Simulink 系列产品，而就中美贸易战后，开源社区的 Python 也迅速崛起。SPSS 和 R 在数据分析与统计学习方面有着独特优势，使用尽可能少的操作量来换取尽可能多的数据图表和可视化分析。就优化问题看，早年 IBM 开发的 LINGO 已经风云不再，但仍然有着较为庞大的用户群体，而诸如 CPLEX、Gurobi 等也同样百花齐放。就数学建模的文稿写作而言，不少同学采用了 LaTeX 和微软的 Word 写作，可以说整个生态链已经有非常完备且齐全的工具。

可是，中国人在其中的身影呢?

国产化的建模软件已经有了不小的突破。在多年前文稿写作上，金山的 WPS 就打破了微软 Office 的垄断霸权地位，成功集成了 Word、Excel、PPT 和 PDF 四者于一体。互动协作的文稿编辑同样有飞书、有腾讯文档等，工具那太多了。就运筹优化技术，杉数科技的 COPT 同样是一款性能非常高的优化求解器，在多个子问题的求解上处于 SOTA 水平。数据分析和处理工具当中，也有 SPSSPRO 集成了多种算法，它的底层用 Python 编写，能够实现比 SPSS 更加方便的数据分析和统计建模。那么，有没有一款国产化的编程工具能够像 MATLAB 一样做各种科学计算呢?

伴随着近年中美贸易战的局势激烈，美国政府勒令 Mathworks 公司对 13 所有军工背景的高校停止 MATLAB 软件的供应，这为国人敲响了警钟。二十年前，柳传志在联想切断了硬件生态研发的供应链决定机体硬件技术依托戴尔和 IBM，先站在现有硬件的基础上发展软件是王道。可近几年，随着美国在芯片问题上对中国的制裁，我们发现原来我们硬件问题上欠了这么大一笔账。同样的，当我们兴高采烈地使用 MATLAB 提供的数值计算便利的时候，丝毫没有意识到危险的来临。当美国人的禁令下来的时候我们才发现，原来国产数值计算软件几乎是空白的。

由北京大学重庆大数据研究院的研发团队设计的北太天元数值计算软件（英文名 baltamatica），可以说是国内首款数值计算软件。但是短时间内要想突破 MATLAB 基本上还是很困难的，因为 MATLAB 有几十年的沉淀，无论是用户生态还是开发者社区已经都非常完备，在机械、电气、控制、机器人、医疗和深度学习等领域有着相对完整的工具包，并在欧洲、日本、东南亚、印度等多个国家有着庞大的用户受众。但是 baltamatica 语法几乎和 MATLAB 一样，所以想要学习 baltamatica 的成本其实是很低的，而且 baltamatica 属于是轻量级编程不会存在几十个 G 的占存，只是它少了一些工具包因而开发过程中需要我们自己编写有关函数。

再说 Python，Python 能够得以异军突起，除了各类包加装在原生 Python 上实现了不同的运算功能 (例如 numpy 的矩阵运算、sympy 的符号运算、scipy 的科学运算数值运算、tensorflow、pytorch 的深度学习、sklearn 的机器学习、statsmodels 的统计方法等)，基本已经完成了一般数学建模的完整工具链，用户无需底层开发而是调用上层封装好的函数即可。这也是我为什么第一门数学建模课选用了 Python 作为我的主讲语言，并且我一再强调，在进行 Python 数学建模的时候要学会“站在巨人的肩膀上看问题，不要过分深究底层细节”。但现在我们使用这个新软件的时候，因为它底层几乎都没支持很多功能，这也就要求我们要从底层出发理解每个算法的工作原理。

笔记

1.3 Baltamatica 的安装和基本使用

Baltamatica 的特色在于：第一，它支持中文编程，其变量名是可以用中文的。虽然 MATLAB 也提供了中文安装包，但在编程问题上如果使用中文变量，那么代码可以以一种极近似于中文伪代码的形式运行；第二，它的本质是一个.m 文件编译器，基于 C/C++/Qt 开发，也就是说 MATLAB 文件在 Baltamatica 当中是完全可以运行的（前提是不能使用没开发出来的工具函数），在语法问题上 MATLAB 可以怎么写 Baltamatica 就可以怎么写；第三，Baltamatica 的占存很小，因为是基于 C/C++ 开发所以 GUI 界面还比较简单，最早发布的时候容易有闪退问题、数据的流水线处理不好的问题等等，但后期经过几个版本的迭代和更新这些问题已经得到了比较好的改善。图1.2和图1.3展示了 Baltamatica 和 MATLAB 的界面，大家可以做一个对比。

图 1.2: Baltamatica 的界面

二者的基本布局是类似的。如果要创建文件，则同样新建一个.m 文件即可，在.m 文件中写下文本形式的代码再交给编译器编译。如果在下方命令行窗口中运作，则代码是逐行运行的。例如，如果我们想打印出 hello world，只需要在下面键入：

```
1    disp('hello world');
```

这串字符串就会在命令行打出。

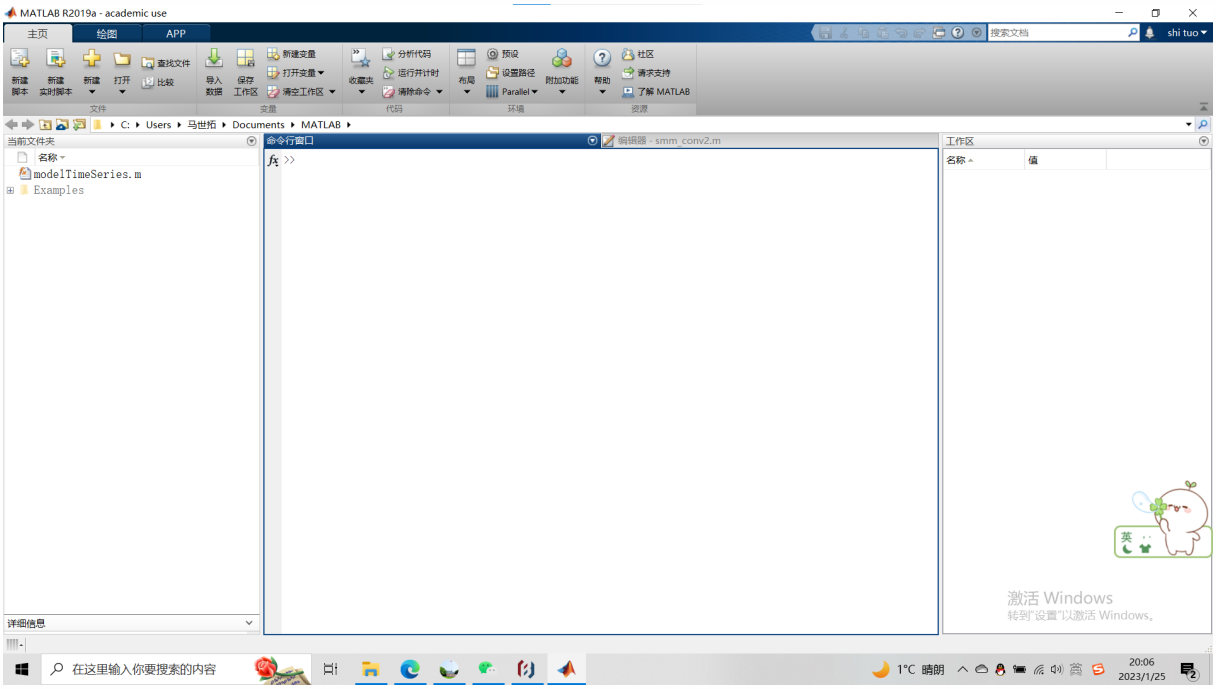


图 1.3: MATLAB 的界面

Baltamatica 的输出其实有两种，一种是以 disp 的形式输出字符串，另一种则是以 C 的 printf 类似的 fprintf 函数输出规格化字符串。获取输入的方式使用 input，括号内可以填写提示信息。但有一个小问题就是在 input 函数中如果执行非法输入比如某个文件 demo.m 在创建进程运行的时候发起了一个 input 请求，而我在 input 当中给出了一个非法输入比如调用一个新的命令 help input 那么创建的新进程和原始的 demo.m 就会相互干扰可能导致意外结果，而在 MATLAB 中则是报错让你再重新输入合法数值。这一点是我认为主创团队仍然需要改进的一个地方。

它的注释同样是用% 表示，然后也支持断点调试，还支持自己开发工具库（只不过需要用 C/C++ 开发我麻了）。下载链接：

windows 版本:

https://disk.pku.edu.cn:443/link/41E48752F9F9352DBD0A1E5383474EB1

ubuntu 20.04 版本

https://disk.pku.edu.cn:443/link/7A99921BEBCABE19EE2C4088C1DB61AC

第二章　Baltamatica 的使用语法

本章我们学习的重点内容是 Baltamatica 的基本语法。大家在学习这门编程语言的时候完全可以参考 MATLAB 的一些教程，但无论学习什么编程语言，语法的基本顺序就像这个样子。我尽可能把我认为一周就能学会的东西尽可能讲得好懂一些，让同学们做完后面的实验就可以尽快上手编程使用。

2.1　数据的存储与数据类型

在开启软件的使用之前，我还是想先给大家普及一些计算机科学中相关的基本常识，加深同学们对于底层数据的理解。

我们都说计算机是一门 01 的科学，因为数据的运算、存储、传输都是以二进制的 0 和 1 比特来进行的。后来也有了量子计算，就是利用量子的不确定性和叠加性把八个状态位合并为一个从而大大加速了运算速度，形成新的计算机体系结构。不管是什么软件、什么编程语言，从最底层对数据的存储与虚拟化模型上来看，它最首要需要解决的问题就是数据的表示问题。我知道你们未必是计算机科班出身，跟你们讲编译原理、计算机系统结构这些东西你们听来可能觉得纯属扯淡，所以我只会尽可能介绍数据在计算机中的组织和表示。

其实数据无非分成了定点数和浮点数表示。定点数呢，在计算机中用的是补码表示法，也就是：当 x 为负数，将二进制的每一位按位取反以后 +1；当 x 为正数，补码为 x 本身。其中，x 的最高位为符号位，当 x 最高位表示 0 时 x 为一个正数，为 1 则表示一个负数。比如我现在给出这样一个定点数 01101.11010，这个定点小数的值应该这样计算：

$$[01101.11010]_{(2)} = +(1*1+1*2^2+1*2^4+1*\frac{1}{2}+1*\frac{1}{2^2}+1*\frac{1}{2^4})_{(10)} = 13.8125_{(10)} \tag{2.1}$$

浮点数我们通常遵循 IEEE754 标准存储。在 IEEE754 的 32 位浮点数中，数字的表示是以一种类似于科学计数法的形式存储的，例如 $-1.1101 \times 2^3 2$。如图2.1在 32bit 的存储模型中，第一位表示符号位，第 2 9 位为阶码，后面的位为尾数。对于阶码而言，它怎么表示比 1 更小的阶和比 1 更大的阶呢？是通过移码操作的，也就是说，将阶码的八位转化为十进制以后，还需要减去 127 才是真正的次方数。而尾数，我们默认它已经有了 1 的整数位，后面的二进制全部处在小数部分。我们来看一个例子：将-0.75 用 IEEE754-32bit 表示，首先符号位为-，0.75 表示为二进制是 0.11 也就是 1.1×2^{-1}，-1=（-126+127），所以阶码为 (01111110)，尾数为 10000000000000000000000，转化过来存储也就是：[10111111010000000000000000000000]。

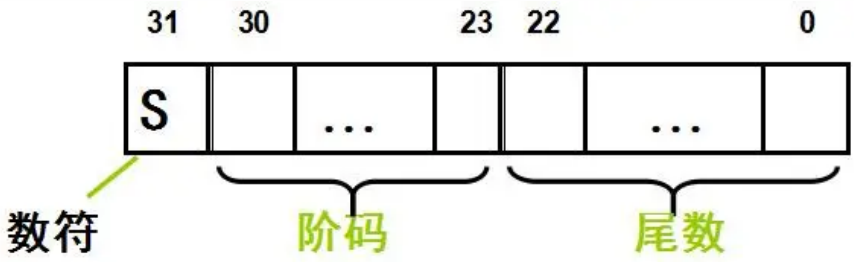


图 2.1: IEEE754-32bit 的存储标准

整数分长整数和短整数，长整数用 64 位存储 (int64)，短整数用 32 位存储 (int32)。而浮点数呢，也分别以 IEEE754-32bit 和 IEEE754-64bit 创建了 single 和 double 两种数据类型。顺带一提，在 Python 当中就不会有这个区分，Python 里面的整数统一是长整数，只在 numpy 等库函数里面可以指定是 np.int64 等。

那么计算机又是如何存储字符和字符串的呢？国际上用计算机表示字符的方式有 ASCII 码，在大体系下又有了 Unicode 编码，UTF-8 编码，能够表示汉字和希俄字符的 GBK 编码、GB18030 编码等，大家都是通过编码将字符转化为对应的数字代号然后用数字表示的方式存储起来的。所以从本质上来讲，对字符与字符串的操作还是对 ASCII 码的操作，也是对数值的操作。

Baltamatica 提供的数据类型包括 logical（布尔型，也就是逻辑型）、int（整数）、single（单精度浮点数，3.4E-38～3.4E+38，有 7 位有效数字）、double（双精度浮点数，1.7E-308～1.7E+308，16 位有效数字）、complex double（复数）、string（字符串）等几种基本类型，在这几种基本类型的组合上拼接向量、矩阵、元胞等复合类型，从而构造出更多的数据类型。而 string 是由一个个 char 组成的，char 为字符类型，但是我不建议大家用 char 类型（因为它底层的异常和中断处理做的不太行可能导致闪退，例如 char 后面跟未赋值变量）而是直接干脆用 string 类。

数据类型之间是可以相互转化的，它存在从高到低的兼容性。例如，我们可以将一个整数扩展为 float32 等。如果我们输入 a=1，在默认情况下，它会被赋值为一个最高精度的 double 类型数。如果我们希望是 float 类型或者 int 类型，可以声明 a=int32(1)。而想要将这个 32 位整数 a 转化为双精度浮点数，可以使用强制类型转化 double(a)。那我如果把 double 转换成 int 类型可不可以呢？原则上是可以的，但是转换的过程中小数点及其以后的部分会被舍弃只保留整数部分。再比如复数类型，如果把 double 或者 int32 转化为复数其实就是将其转化为 double 类型并在后面跟上虚部 0 即可。而复数是无法转化为实数的，只能通过 real 取实部、imag 取虚部。但 double、int、complex 等都是可以转化为字符串类型的。

另外，声明浮点数还可以用科学计数法的形式声明，例如 5.56E10 就可以代表 5.56×10^10。在 Baltamatica 中也预先设置了一些常用数字，比如 pi 表示圆周率，还有 e，inf 等也都是可以用的。顺带一提，IEEE754 当中如果要表示无穷大可以用全 1 阶码和全 0 尾数，表示 0 可以全 0 阶码和全 0 尾数，表示不存在可以用全 1 阶码和非 0 尾数。

练习应用

1. 把下列二进制定点数转化为十进制小数（最高位为符号位）

   (a) 11.10011

   (b) 0111.100010

   (c) 1101001.10110101

2. 写出下面数字的补码

   (a) 31

   (b) -273

   (c) 18456

   (d) 100110010100101

3. 将下面十进制浮点数转化为 IEEE754-32bit 浮点数

   (a) 12.625

   (b) 731.875

2.2 运算符与表达式

本节我们将看到运算符和表达式的运算规则。计算机所做的运算实际上就是三类基本运算：算术运算，逻辑运算和位运算。其中，逻辑运算和算术运算同样是基于位运算进行的。

计算机为什么是基于位运算进行呢？我们知道，在传统晶体半导体逻辑门电路中，bit 的 01 其实也就是数据的关闭和打开。当导线通电，这条通路上就表示为 1；导线未通电则表示为 0。所以，计算机必须以电路的开闭来进行按位运算，即使是多个比特之间的运算其最底层还是按位进行的。

图2.2是我用 Logisim 画的一块 RISC-V 架构的 CPU 示意图，CPU 的 PC 也就是程序控制器按照机器指令的执行流程从存储器中取出一条机器指令后立即跳转到下一条机器指令，指令通过译码器译码以后识别出操作码、地址字段，从地址中通过读写线控制得到操作数，操作码经过逻辑门得到选择操作的选控线比如二进制的定点数加法运算、二进制乘法运算等，将两个操作数输入到 ALU 中根据对应选控线要求进行算术运算（通常两个数都可以是 8 位或者 16 位，我们做 CPU-demo 都是八位八位往里面放），得到结果可以写回存储器，也可以做后续的流水线工作。而 ALU 也就是数字运算的部分，包含了按位逻辑运算（与、或、非、异或）、算术运算（加、减、乘、除）还有位运算（左移，右移，按位取反等）等基本运算。

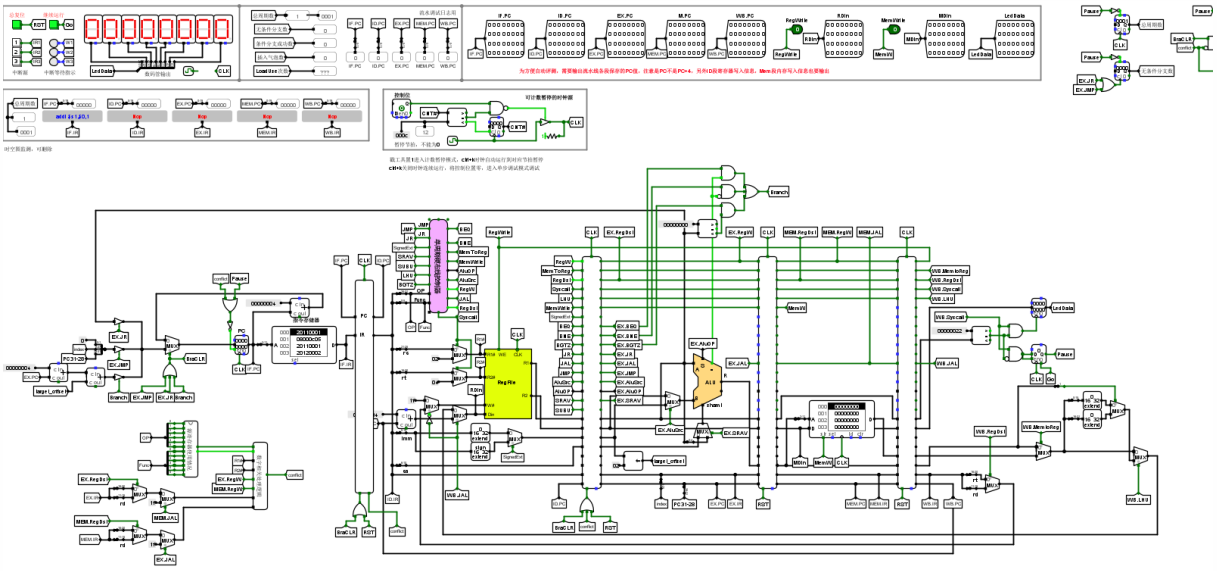


图 2.2: 一个 RISC-V 架构的 CPU

基本的逻辑运算包括与、或、非三种，在此基础上又诞生了异或逻辑结构。逻辑运算的操作对象和操作数其实都是 logical 类型的变量，也就是布尔类型。布尔类型其实是整数的派生，只需要一位就可以表示，就是 0 或 1。我们用 0 代表假而用 1 代表真，那么这几个逻辑函数的真值表如表2.1所示

表 2.1: 不同逻辑函数的真值表

a	b	a and b	a or b	not a	a xor b
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

表示两个布尔量之间操作的符号为 &（与）、|（或）、～（非），异或逻辑其实可以根据前面三个的组合得到。其实对比表2.1我们不难发现，异或运算其实可以抽象为：

$$(a \bigotimes b) = ((\sim a)\&b)|(a\&(\sim b)) \tag{2.2}$$

有兴趣的同学可以列出真值表抽象一番。与或非这三个基本函数都是对单一的布尔变量进行的逻辑判断，而通常我们认为的逻辑判断还有一些等式、不等式以及矩阵和向量的逻辑。这时候我们需要用到的是表达式逻辑语句，也是在这三个基本逻辑的基础上将表达式进一步抽象为布尔变量。常见的包括：>=，<=，>, <, ==, ~=, any, all 八个函数。前四个好理解，那么为什么判定等于是用两个等号呢？这是因为，一个等号其实它不是等号而是赋值，比如 a=3 并不是判断 a 是不是 3，而是计算机找到变量 a 的内存区域并将数字 3 放入（其实像 R 语言里面最早的赋值语句是 <-，用箭头描述这样就好理解一些，不过好像也只有 R 语言是这么写赋值）。那么顾名思义，= 也就代表了不等于。Baltamatica 不像 Python 可以用连续不等号，当我们想要表示 $1 \leq a \leq 3$ 的时候我们不能直接写 1<=a<=3, 而是应该 a>=1 a<=3（如果害怕表达式太多可以给每个表达式加个括号定界一下）。在计算机的底层，a<=1 抽象为汇编指令是“beq a 1”，也就是比较 a-1 和 0 的大小，如果小则跳转，大则下一步。

any 是指在一个数组里面是否存在元素如何如何，all 则是数组中任意一个元素是否都如何如何。比如说，如果我们要找 0 元素和非 0 元素，我们可以直接写：

```
1  a=[1 2 3 4 0 5 1];
2  any(a)
3  all(a)
```

最终得到的输出为：any(a) 是一个 logical 1 也就是真，all(a) 是一个 logical 0 也就是假。最后的量都是逻辑变量。如果要对 a 里面是否存在大于 1 的值进行判断，只需要输入 a>1，它会返回一个与 a 大小相同的数组或矩阵，这个矩阵的每个值都是逻辑量表示满足条件或不满足条件。进一步地，如果想获得满足条件的元素下标，可以使用 find(a>1)，这样就会返回元素的下标了。

逻辑运算可以通过逻辑门来实现，数值运算本质上也是按位进行的逻辑运算。比如说，我们对两个二进制位进行加法运算，也就考虑两根输入线 (0 或 1)，它的运算位和进位如表2.2所示：

表 2.2: 两个布尔变量的加法

a	b	进位 C	计算结果 S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

我们可以看到，两个位进行二进制相加的时候，进位实际上就是 ab，计算结果 C 其实也就是 a xor b。如果使用与门和异或门表示，可以得到两根输入线的半加器电路结构。把多个半加器串联在一起，可以同时进行 8 位、16 位的二进制加法运算，但还有一个问题需要解决，就是低位向高位进位的问题。好在低位的进位要么是 0 要么是 1，我们只需要把该位的计算结果 C_n 与低位的进位 S_n 相异或就可以了，如果发生了进位，那么把这个进位也算到上一个里面去。

在计算机当中，数据通常以补码的形式存储，所以这里的加法通常也是补码二进制加法（减法实际上就是取相反数相加，同样以负数补码的形式存储），也就有了 ALU 中的二进制全加器电路：

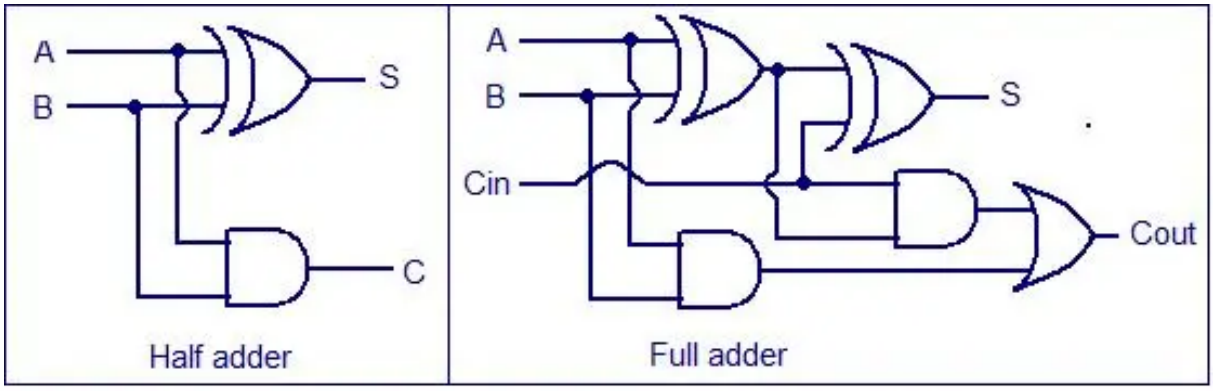


图 2.3: 二进制半加器和全加器

这也就是逻辑运算转化为二进制数字运算的底层逻辑，当然，这属于硬件层面的东西，跟软件设计没有什么关系。不过我是计算机出身，所以谈到不同运算之间联系的时候会刻意思考硬件层面和软件层面之间的联系。而 Baltamatica 对于上层的封装做的是非常好的，因为它本身就是基于 C/C++ 开发，从 C 的层面来讲你就已经看不到硬件的过程了。只需要像 C/C++/Fortran 语言中加减乘除的运算法则去进行就行了。顺带一提，乘法其实也就可以展开为多次加法并进行流水线操作，一种通用的补码二进制乘法算法叫做 Booth 算法。

Baltamatica 提供的数字运算符号包括 +（加法）、-（减法）、*（乘法）、/（除法）、\（反向除法）、^（乘方）等。而在矩阵和向量运算中，还提供了转置符号' 和按位运算的方法，也就是对应位置的元素做加减乘除的方法。具体的我们会在后面讲矩阵语法的时候讲到。算术运算符加、减、乘及乘方与传统意义上的加、减、乘及乘方类似，用法基本相同，而点乘、点乘方等运算有其特殊的一面。点运算是指元素点对点的运算，即矩阵内元素对元素之间的运算。点运算要求参与运算的变量在结构上必须是相似的。

如果想使用其他函数例如对数函数 log、三角函数 sin、cos、tan 等也是可以的，软件中也对这些常见函数进行了内置。对于复数的运算，提供了一些常见函数例如 real（求实部）、imag（求虚部）、abs（求模）、angle（求相位角）、conj（共轭复数）、unwrap（调整相位）、isreal（是否为实矩阵）、cplxpair（将复数矩阵排列为共轭对）等。

练习应用

1. a=1(double),b=2(double) 尝试分析这些代码的结果：

   (a) a&(a>b)

   (b) ((5*a)+3/a)-b^2

   (c) ((a>2)~ $(a-1))|(b == (a >= 1))$

2. 考虑这样一种情况：如果我创建了两个变量 a=1, b=2，现在我想交换两个变量的值，应该如何实现？（如果有兴趣的同学可以把过程封装为函数 swap.m）

3. 思考问题：不同运算符之间的运算优先级应该是怎样的？（可以参考 C 语言的运算优先级顺序）

笔记

2.3 选择语句

本节我们看到程序设计过程中的控制流结构，并学习 if 语句和 switch 语句的用法。其实从上一节的习题 2 中我们不难发现，Baltamatica 的语句是可以逐条运行的。这也是编译过程中的两种方式，即：先编译完成再运行和边编译边逐条运行（解释性语言）。但是在运行的过程中，数据的流动可能不是单向的，而是有分支结构的。这一节的选择语句和下一节的循环语句都属于分支结构，即：数据流可以跳跃式流动或回转式流动。在早期汇编语言中，通常以类似于 goto 的方式到处打断点跳转，极其不易理解（这也是 Dijkstra 发表论文怒斥 goto 就是个祸水的原因之一）。现在我们在写高级语言程序的时候会对这些低层次内容进行包装，形成 if switch else while for break return continue 这几种常见的分支情况。

顺序结构，选择结构和循环结构的数据流如图2.4所示：

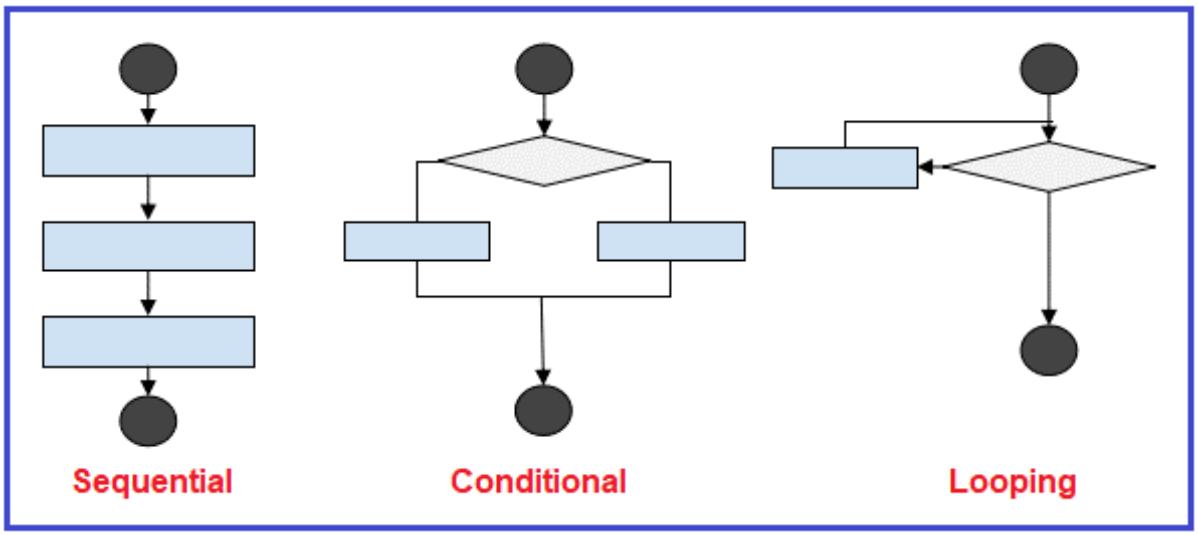


图 2.4: 不同的数据结构

if 语句后面跟的通常是一个判断指令，表示当满足这个条件的时候会执行 if 中的代码部分。一个 if 需要有一个 end 与其配对，这就像括号必须有大括号还得有小括号才能封闭，不然编译的时候是通不过的。这里的 if 不像 C/C++ 需要用括号，也不像 Python 需要用: 和缩进。例如，if 结构的用例应该写作：

```
1    if condition
2        do_some_operations;
3    end
```

我们举一个简单的例子：

```
苹果=input(“妈妈买了几个苹果”); %最开始买回来几个苹果
苹果=苹果-1;
disp(”妈妈先吃了一个”);
苹果=苹果-2;
disp(”爸爸吃了两个”);
if 苹果<3
    苹果=3;
    disp(”发现苹果不够了妈妈又去买了几个，现在一共有三个”);
end
苹果=苹果-3;
disp(”小明吃了三个”);
disp(”大家都吃饱啦”);
```

那么这个程序的分支点就设置在“苹果<3”的这个地方。如果我本来爸爸妈妈吃完以后还有多余的苹果，那么这个分支点就不会被触发，if-end 中间的这一部分代码也就不会被执行。但是苹果不够了的时候就会触发条件，执行 if-end 中间这一段买苹果的部分。

进一步地，我们如果想在不满足条件的情况下执行另一段操作，可以用 if-else-end 语句。注意：else 不需要跟 end，这仨是一起配对的。例如：

```
if condition
    do_some_operations;
else
    do_other_operations;
end
```

同样是上面的案例，稍微修改一下就可以变成：

```
苹果=input(“妈妈买了几个苹果”); %最开始买回来几个苹果
苹果=苹果-1;disp(”妈妈先吃了一个”);
苹果=苹果-2; disp(”爸爸吃了两个”);
if 苹果<3
    disp(”发现苹果不够了妈妈又去买了几个”);
    disp(”小明吃了三个”)
else
    disp(”小明把它们全吃完了”)
end
disp(”大家都吃饱啦”);
```

笔记

使用 if-else 可以划分为两段，那如果划分为多段呢？则可以使用 if-elseif-else-end 配对。这里要注意，elseif 和 C/C++ 里面的 else if 不一样，两个单词是连起来的。比如说，使用方法形如：

```
分数=input("小明考了多少分\n");
if 分数>700
    fprintf("小明可以去清华或者北大\n");
elseif 分数>600
    fprintf("小明可以去一个不错的学校\n");
elseif 分数>400
    fprintf("小明可以去一个普通的学校\n");
else
    fprintf("小明在工地上打起了灰\n");
end
fprintf("大家都有美好的未来\n");
```

如果认为连写的 elseif 不容易理解可以把代码拆开进行 if 结构嵌套，也是可以的。但是此时需要注意，嵌套 if 结构的时候一定要注意 end 配对。我们把上面的例子改写一下也可以变成：

```
分数=input("小明考了多少分\n");
if 分数>700
    fprintf("小明可以去清华或者北大\n");
else
    if 分数>600
    fprintf("小明可以去一个不错的学校\n");
    else
        if 分数>400
        fprintf("小明可以去一个普通的学校\n");
        else
        fprintf("小明在工地上打起了灰\n");
        end
    end
end
fprintf("大家都有美好的未来\n");
```

可能有时候大家会觉得这种嵌套结构有些繁琐，所以为了简化也有了 switch 结构。switch 结构根据变量的取值来选择跳转的窗口点，例如结构：

```
switch 变量或表达式
case 常量表达式1
  语句组1
case 常量表达式2
  语句组2
...
case 常量表达式n
  语句组n
otherwise
  语句组n+1
end
```

注意，这里判断条件的 case 后面跟的都是标量，也就是具体的数值或者字符串，如果是输入不等式判断我建议还是用 if 结构。比如，给出一个案例：

```
switch x
case 0
  fprintf('开启查询服务\n')
case 1
  fprintf('尝试为您添加数据，请输入密码\n')
  passwd=input('输入密码')
case 2
  fprintf('尝试为您删除数据，请输入密码\n')
  passwd=input('输入密码')
otherwise
  fprintf('谢谢使用\n')
end
```

这个案例就是在输入不同 x 按键的时候给出的不同反应。当 x==0 的时候开启查询服务；当 x==1 的时候添加数据，当 x==2 的时候删除数据，否则退出系统。

这一节的习题和下一节一起布置吧。

2.4 循环语句

本节我们将学习循环语句 for 和 while 的用法。循环语句其实就是在未满足条件的时候不断执行循环体内的语句。比如，大家还记得高斯是如何求 1+2+...+100 的吗？是首尾求和乘以项数除以 2。那么这个地方，我们用计算机求解可以使用 while 结构：

```
while 条件表达式
    循环体
end
```

改写出来那就成了：

```
i=1;
s=0;
while i<=100
    s=s+i; %当 i 不到 100 的时候一直加
    i=i+1; %不断增加 i 的值直到打破限制条件
end
disp(s)
```

最终得到的结果也是 5050。当 i<100 满足的时候，循环体内 i=i+1 也就是自增指令会一直执行下去。这里有些同学可能不理解 i=i+1 是什么意思，两边消去 i 难道是证明 0=1？其实这一语句的执行过程应该参考图2.1，ALU 需要先从寄存器中找到变量 i 的地址，从地址中取出 i 存放的数值送到 ALU 的第一个操作数端口，再将一个立即数常数 1 送到 ALU 的第二个操作数端口，选通操作码为 ADD 操作，通过补码二进制加法得到的结果写回原来 i 所在的内存地址中，这样 i 的内容就被成功修改了。

另外，学习过 C/C++/Python 的同学应该还会熟悉，将 while、初始赋值和自增指令合并的 for 语句，会比 while 更加方便。如果将上面的代码改写为 for 语句应该是：

```
s=0;
for i=1:100
    s=s+i; %当 i 不到 100 的时候一直加
end
disp(s)
```

这里下标 i 从 1 出发一直到 100，在默认情况下每次自增 1，运行 100 次。其中，1:100 会生成一个从 1 到 100 的数字组成的数组，如果中间确定步长 2（即 1:2:100）会生成以 1 到 100 范围内从 1 开始每次自增 2 生成的系列数组（一共 50 项）。

而如果我们想在中途打破循环可以使用如下两个命令：

1. **break**：该命令一般用来终止 for 或 while 循环，通常与 if 条件语句结合在一起使用，如果条件满足则利用 break 命令将循环终止。在多层循环嵌套中，break 只终止最内层的循环。例如：

```
s=1;
for i=1:100
i=s+i;
if i>50
        i
        disp('stop');
    %这里就会看到i不会增加到100而是到50就断了
        break;
end
end
```

2. **continue**：该命令通常用在 for 或 while 循环结构中，并与 if 一起使用，其作用是结束本次循环，即跳过其后的循环语句而直接进行下一次是否执行循环的判断。例如：

```
s=1;
for i=1:4
    if i==3
            continue;
    end
    s=s*i;
end
s%这里会看到s并不是1*2*3*4而是1*2*4，因为i==3的时候这一次被pass掉了
```

至此，我们已经接触了几种基本的控制流语句，循环语句和选择语句都是需要 end 与其配对的，并且不需要括号也不需要冒号只需要缩进和 end 配对即可完成编译过程中的识别。下面希望同学们能够主动完成这样几个习题加以练习：

练习应用

假设你要开发一个成绩查询系统，菜单栏目可以显示如下：

```
clc;clear;
fprintf("==========菜单==========\n");
fprintf("||                                  ||\n");
fprintf("||      0. 退出          1. 查询      ||\n");
fprintf("||                                  ||\n");
fprintf("||      2. 新增数据      3. 修改数据  ||\n");
fprintf("||                                  ||\n");
fprintf("||      4. 对总分进行排序            ||\n");
fprintf("||                                  ||\n");
fprintf("||                                  ||\n");
fprintf("==========================================\n");
```

现在需要你完成三个部分的功能：

1. 使用 while 结构不断获取操作符 x 的输入，提示音为“请输入您需要进行的操作”，保留功能 3 和功能 4。

2. 表格的结构为：第一列表示学号（例如 16001），第二列为语文成绩，第三列为数学成绩，第四列为英语成绩，第五列表示总分。现在查询功能是需要输入学号，根据学号查询是否在表中，如果不在则提示” 未查询到该生信息，请重新核对学号后输入”，如果在，则输出学生的各科信息。查询某个学号所在列的方法形如

```
col=find(A(:,1)==id);
```

当 size(col,1) 为 0 的时候说明找不到，不为 0 则返回行号。

3. 功能 2 新增数据，初始状态下置空成绩单表格 A=[] 和数据条目 n，新增数据时输入语文数学英语分数并统计总分。向表格内新增一行的方法形如

```
A=[A;[id,k1,k2,k3,k1+k2+k3]];
```

笔记

2.5 向量与数组

数据在计算机空间内可以连续存储，将数字按照一个方向排列成一个数据集合，在计算机内用一片连续的内存空间存储起来这就构成了数组。Baltamatica 中这个数组还可以表示一个向量。那么，数组这种存储模式有哪些特性呢?

首先，我们需要知道集合类数据类型的存储方式。什么是集合类数据类型呢？就是把若干个类型相同或不同的数据放在一个篮子里面，它们可以一个挨着一个排排坐，也可以散乱分布但是每个人都知道其中至少一个队友的位置。如果所有的元素都是连续分布，存储在一片连续的内存空间当中，这样的存储方式我们称之为顺序存储；而如果元素的存储利用的是计算机内存碎片中的闲置空间，但不同的碎片之间有指针相互指引，我们称这样的存储结构叫链式存储。在一个数组当中，每个元素都只有一个前驱和一个后继节点，所以，它是一个顺序表。

Baltamatica 中的数组类似于 Python 当中的 list，Numpy 当中的 array，C++ 当中的 vector 类型。而数组的数组则构成了矩阵，也就是二维数组。有关于二维数组的存储模型等我们会在后面进行更多的讨论，这一节只需要知道二维数组的索引方式就够了。那么所谓的索引，也就是根据位置下标去查询元素，即：数组的第几个元素。Baltamatica 与常规的程序设计语言 C/C++/Python 不同，它的计数是从 1 开始计数而不是从 0 开始计数，并且它对数组下标的索引是圆括号而非方括号，需要在编程时引起注意。比如，对于某个数组 vec，想要查询它的第三项只需要写 vec(3) 即可。

而对于二维数组而言，它通常以矩阵的形式存储。它是数组的数组，可以将行向量按列拼接，也可以将列向量按行拼接。这也是 Baltamatica 的数组与传统数组不太相同的地方之一——它的数组是有方向性的，如果以; 来分割不同的元素则创建的方向是一个列向量；如果以, 或者空格来分割不同的元素那么创建的方向是一个行向量。行向量和列向量在索引等问题上是一样的，主要的区别体现在向量的数值运算方面，不同方向、不同尺寸的数组向量是无法轻易数值运算的。

那么有同学可能会问，说：如果我把几个长度不同的行向量按列拼接在一起，它会怎么办？实际上，执行的结果会报错。如果希望拼接的结果仍然是一个矩阵，矩阵的列数取这些行向量长度的最大值，行数为向量数，缺失的地方用 0 补齐，则可以创建一个这样尺寸的矩阵或者说二维数组（n*max(len)，其中 len=[length(v1),length(v2),...,length(vn)]），然后进行赋值即可。

向量的运算这个时候点运算就凸显重要性了吧，比如说，如果我希望长度相同的两向量 a 和 b 进行按位运算比如按位加法，我不需要创建新的数组 c 然后 for 循环遍历 a 跟 b，只需要做一次 a+b 即可（加减法还不需要用点，当然加上也是可以的）。但这个时候注意，下面两个式子的结果是不同的：

```
[1;2;3]+[4  5  6]
[1  2  3]+[4  5  6]
```

前者是一个列向量和一个行向量的相加,得到的是一个矩阵,创建的维度是 length(a)*length(b) 这么多。那么，第一行是 b+a(1)，第二行是 b+a(2)，……第 n 行是 b+a(n)，得到的结果是一个矩阵；而后者是尺寸相同的行向量（或列向量）相加，结果是对应位置相加和，即返回 [a(1)+b(1) a(2)+b(2) ... a(n)+b(n)]。

但是从乘法开始，这个地方就有些变味道了。比如说你要是算 [1 2 3]*[4 5 6] 这个地方肯定是报错的，但如果使用点乘 [1 2 3].*[4 5 6] 它就不会报错。因为点乘的方法是反馈新数组 [a(1)*b(1) a(2)*b(2) ... a(n)*b(n)]。而如果我想计算数量积怎么办？那就转置一个。我如果转置前者为列向量去乘后面的行向量，得到的结果将会是矩阵；而行向量乘列向量则会得到一个数量积的值。毕竟，矩阵的乘法叫行列积。除法类似，有按位除法和求逆两种模式。另外，用一个数去乘一个向量或矩阵实际上对它做了一个数乘或者说放缩操作，是点乘普通乘通用的。

矩阵的更多操作还包括求逆 (inv)、求模 (det 求行列式，norm 求范数)、改变尺寸（reshape）、求秩 (rank)、求迹 (trace)、特征值分解 (eig) 等，这里我们重点提一下特征值分解。其实矩阵的分解方法有很多，特征值分解其实就是为了满足方程：

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \tag{2.3}$$

即：

$$|\mathbf{A} - \lambda\mathbf{I}| = 0 \tag{2.4}$$

满足所有方程的解 λ 和对应的向量 x 就是对应的特征值与特征向量。调用函数为 eig。除此以外，还有 LU 分解（lu）、楚列斯基分解（chol）、LDL 分解（ldl）、奇异值分解（svd）等算法可以使用，但多数情况下我们用不着，感兴趣可以 help 一下它的文档查有关功能。有关奇异值分解的问题我们会在后面的章节中另外讨论。

请注意，我不是学数学的，更不是教线性代数的，我不打算抢你们线性代数老师的饭碗抢太多，如果对于矩阵运算的定义和性质还有问题的话，我的建议是你先找一本线代书看看或者听一听线代的课。

最后介绍如何创建特殊的矩阵。常见的特殊矩阵例如上面提到的 I 其实它就是一个单位阵，也就是创建一个 n*n 的方阵，这个方阵对角线上全是 1 而其他位置全是 0。另外，创建一个全 0 的数组可以用 zeros(n) 或者 zeros(m,n)，当只有一个 n 的时候生成的是 n*n 的方阵，而输入 m 和 n 的时候生成的是一个 m*n 的矩阵。特别地，当 m=1，生成一个列向量；n=1，生成一个行向量。同样的，如果希望生成全 1 的矩阵则使用 ones(n) 和 ones(m,n)。如果想要生成一个对角阵，这个对角线上所有的元素都已经确定并存储在向量数组 v 当中，则可以调用 diag(v) 来生成一个 length(v)*length(v) 的对角阵。如果想生成随机数矩阵，可以使用 rand(n) 和 rand(m,n)，形状不用我多说吧？类似，分别是 n*n 的方阵和 m*n 的矩阵，生成的都是 0～1 之间的随机小数。

接下来我们看矩阵和向量的索引里面还需要注意的问题。一维向量的长度可以用 length 函数调用，二维矩阵的尺寸可以用 size 函数调用（当然，一维向量可以看成是特殊的二维矩阵，用 size 也是可以的）。size 后面跟的参数可以有一个或两个，当后面跟的只有一个矩阵的时候返回它的行列值，而跟了一个矩阵还跟了一个 1 或者 2 表示只需要取出行数或列数。而二维数组的索引需要两个下标，比方说 A(3,5) 表示第三行第五列的元素。如果想取出矩阵的某一列则可以使用 A(:,1)，而某一行则可以 A(2,:)。

如果想判断矩阵或者数组中满足某个条件的索引值，直接用 A>0.5 这样的表达式就可以了，它会返回一个尺寸相同的布尔矩阵。find 函数用于寻找满足某个要求的元素下标，例如对于一维数组 v，可以用 find(v<2) 来查找所有小于 2 的元素的索引，但这个索引用于二维数组时则会发生一些变化，这与二维矩阵的存储结构有关，我们后面会再做探讨。如果想判断一个或者一个数组里面的元素是否在另一个数组当中，可以使用 ismember(a,b)，返回值为 [l,I]。其中，l 表示在不在，I 返回对应下标。所以，如果我们想要获得一个数组排序后元素在原来数组里面的位置（参考上一讲当中的功能 4），可以这样实现：

```
b=sort(score);
[l,I]=ismember(b,score);
```

既然这里提到了矩阵的排序函数 sort，那么排序算法的实现也将作为本讲的习题。排序算法有很多，包括选择排序、插入排序、冒泡排序、归并排序、快速排序、希尔排序、堆排序、基数排序、桶排序等多种算法，其中最基本的是冒泡排序和快速排序两类。具体的要求可以参考这一讲的习题。

最后普及一个小概念，叫时间局部性和空间局部性：如果一个变量在程序执行的过程中一段时间内被反复使用多次，称之具备好的时间局部性；如果一组变量在程序存储的过程中存放在一片连续的内存空间，称之具备良好的空间局部性。时间局部性是利用循环结构，例如 while 语句当中 i 反复自增，所以有良好的时间局部性；而空间局部性就比如数组，存放在一片区域里面，所以空间局部性很好。

练习应用

1. 补全上一讲成绩查询系统当中其余的功能，并正确实现输出运行。

2. 冒泡排序实际上是一个元素交换的过程，它的过程也就是：先进行一趟循环，若前面的元素比后面的一个大，则交换两个元素的值。交换遍历过一趟以后再不断进行同样的操作，直到序列不再发生变化则视作排序完成。比如数组 [1 3 2 7 5 4 6]，第一趟因为 3>2 所以 [1 2 3 7 5 4 6]，类似的，交换 7：[1 2 3 5 7 4 6], [1 2 3 5 4 7 6], [1 2 3 5 4 6 7]，第一趟结束。再进行第二趟：[1 2 3 4 5 6 7], [1 2 3 4 5 6 7] 不变了，排序完成。那么参考这个手工实现的过程实现代码并保存到 bubblesort.m 当中。

3. 快速排序是一种分治方法的典型应用，它是以数组的一半为界，将大于它的都放到右边来，小于它的都放到左边来，然后对左右两个子序列再排序。比如同样是 [1 3 2 7 5 4 6]，长度的一半到了 3.5，取 4，所以将小于 7 的放左边大于 7 的放右边，变成 [1 3 2 5 4 6 |7]。7 的位置确定了，左边子序列的长度一半为 3，以 2 为界，小于 2 的放左边大于 2 的放右边，就变成了 [1|2| 3 5 4 6 |7]，那么中间悬而未决的 [3 5 4 6] 取长度的一半 2，小于 5 的放左边大于 5 的放右边就有了 [1|2|3 4|5|6|7]，最后 [3 4] 长度只有 2 可以直接判定出位置就是 [3|4]，最终结果 [1 2 3 4 5 6 7]，排序完成。参考这个手工实现的过程实现代码并保存到 qsort.m 当中。

2.6 函数与匿名表达式

这一节将介绍函数和匿名表达式的写法。之前的一些小节当中，大家听到过我老提函数函数，可能理解的就是函数是 Baltamatica 当中提供的指令。现在，我们可以自己写函数了。

函数的创建方式是 function 指令，通常是一个函数一个.m 文件，到时候要调用就召唤它并传入相应的参数。例如，我们以这样一个函数为例：

```
function [y] = func0(n)
    m = n
    if (m<=1)
        y = 1;
    else
        while m>1
            disp('m=')
            disp(m)
            m = m-1;
            y= m*func0(m);
        end
   y = 2;
    end
end
```

在这样一个函数当中，用 function 和 end 配对表示声明了一个叫 func0 的函数，函数的返回值列表只有一项就是 y，形式参数是 n。那么实际调用的时候放进去的也不一定非得叫 n，比如，我调用的时候用了：

```
    x=100;
    func0(x)
```

这是没有问题的，实际上我传进来的 x 才是实际参数，定义函数的时候我创建的 n 叫形式参数，跟我实际要传的变量的名字没有任何影响。

另外，我们来做一个小实验，写下如下一段代码运行：

```
m=1;
n=0;
disp(”在函数体外”)
disp(m);
disp(n);
```

```
func16()
disp("回到函数体外")
disp(m);
disp(n);
function[y] = func16()
        y =1;
        m = 2; n = 3;
        disp("在函数体内")
        disp(m);
        disp(n);
end
```

你会看到它的输出分别为:

```
在函数体外
m=1
n=1
在函数体内
m=2
n=3
回到函数体外
m=1
n=1
```

诶，这是怎么一回事呢？为什么 m 和 n 在外面定义了以后，我明明在函数体内把它修改了，外面的 m 和 n 没有被关押呢？你可以这样子理解，函数体外面是一个世界，函数体内部是一个封闭的城邦。外面的世界和封闭的城邦中都有两个同名同姓的人，分别叫 m 和 n，外面那对兄弟分别是 1 岁和 1 岁，所以输出了 m=1,n=1；城邦里面的兄弟 m 和 n 分别是 2 岁和 3 岁，所以输出了 m=2,n=3；但当我从城邦里面走出来，问问外面的 mn 兄弟几岁，他们还是会回答 m=1,n=1。那我如何修改函数外面的 m 和 n 呢？只需要在声明 m 和 n 的地方前面加上一句 global m n（全局变量），在函数内和外都设置一下就可以修改了。

计算机在程序运行的时候内存分配是用一个叫堆栈模型的方式存储，堆区以地址增大的方向自底而上增长，而栈区呢则从地址减小的方向自顶而下增长。最底层的是代码区，存放的是.m 文件的源代码文本，编译完成以后基本没什么用途。上面走就是堆栈，函数体的变量都是在栈区分配，函数体执行完成以后函数体内的临时变量就会被销毁掉（又叫出栈）。而在堆区和栈区中间有一大段空闲区域。从空闲区块中可以抽一部分来放所谓全局变量，也就是我这里用的 global。

笔记

在 Java 或者 C++ 当中有静态量 static，跟 global 一样放在这片区域里面。不过 Baltamatica 没有。另外还有一部分常量被放在全局区上面的常量区里面，比如定义的一些常量字符串、常数等（通常不可修改，程序结束以后由操作系统释放），这个区域我们不多做讨论。所以，如果想跨函数引用、修改，以及函数体内外一体化操作，可以将变量设置为 global 变量（毕竟 Python、Java 等高级语言早就不支持指针了）。

return 命令能够使正在运行的函数正常结束并返回调用它的函数或命令行窗口。也就是出现某些状态下能够提前结束函数。比如我们看这个例子：

```
function C=sumAB(A,B)
      [m1,n1]=size(A);
      [m2,n2]=size(B);
      C=zeros(m1,n1);
      if m1 == 0
          disp('input empty');
          C=[];
          return;
      else
          for i=1:m1
              for j=1:n1
                  C(i,j)=A(i,j)+B(i,j);
              end
          end
      end
end
```

当 A 是一个空数组的时候输入是有问题的，我们想提前切断它运行并正常返回，所以使用 return 指令。比如，我可以使用如下的测试用例：

```
>> A=[];
>> B=[3  4];
```

因为 A 是空数组，触发了 m1==0 的条件，所以通过 return 指令返回了。另外大家思考一下，这个函数 sumAB 是想干嘛的呢？可以描述一下它的功能。

讲完这些以后，我们来看递归是什么。古印度有一个古老的故事，梵天创造世界以后将阿修罗囚禁在一个无人之境中，并给了他三根柱子：金柱子，银柱子和铜柱子；并在金柱子上面放了大小不同的 64 块铁盘，要求：一次只能移动一块铁盘，大的铁盘不能摞在小铁盘上面否则铁盘碎裂游戏重来，问需要多久才能将这 64 块铁盘从金柱子移动到银柱子上？

我们这样思考：假设说我们知道 n-1 块铁盘从一根柱子移动到另一根柱子上的时间叫 F(n)，那么，我们不妨先把这 n-1 块铁盘从金柱子移动到铜柱子，再将最底下最大的铁盘移动到银柱子，再把铜柱子上的 n-1 块铁盘移动到银柱子上来，也就是递推公式：

$$F(n) = 2F(n-1) + 1 \tag{2.5}$$

如果只有一块铁盘，那么我只需要移动一次，也就是 F(1)=1。求解有两种思路，一种是从 F(1) 出发顺序往后递推，另一种是先从 n 倒退回 1，然后再从 1 推到 n。这两个思路一个叫递推，一个叫递归。如果把上面的递归方法写成函数，也就是：

```
function [f]=hanoi(n)
    if n==1
        f=1;
    else
        f=2*hanoi(n-1)+1;
    end
end
```

递归方法也可以用于解斐波那契数列等问题，它是非常省代码量的。但节约代码量的代价就是，如果我递归计算斐波那契数列的第 20 项可能程序就会炸。这不仅是 Baltamatica 的问题，使用 Python、MATLAB、Java 等语言的时候都会出现的卡顿现象，因为递归需要大批量重复计算非常浪费时间，又是倒推又是顺推，所以小批量的递归还可以，大批量的递归就会非常卡顿。如果计算斐波那契数列就会发现，f(20)=f(19)+f(18)，那么 f(19) 又得计算 f(18)+f(17)，f(18) 算了两次，不断向下降阶，会发现重复计算的部分越来越多。解决此类问题的一种方式是使用动态规划，即空间换时间，将中间结果利用数组存储起来。

最后给大家介绍函数句柄，我习惯于叫它匿名表达式（因为 Python 里面这么说习惯了）。我们可以用 @ 来声明一个函数，如果将函数用一行匿名表达式书写，以下两种写法是等价的：

```
%写法一
sigmoid=@(x) 1/(1+exp(-x));
%写法二
function y=sigmoid(x)
    y=1/(1+exp(-x));
end
```

在调用的时候都使用 sigmoid(x) 就可以了。

练习应用

1. 将前面成绩查询系统当中的功能都封装为函数再插入到 switch 结构中。注意如果将子函数和 switch 放在同一个文件当中，函数应该写在主框架的后面，这和 C 语言的先声明后引用有些不同。

2. 基于数组和循环结构，试着写出函数并分析时间复杂度：

   (a) 计算排列数 A(m,n)=n*(n-1)*...*(n-m+1)

   (b) 计算组合数 C(m,n)=A(m,n)/A(n,n)

3. 基于递归结构，试着写出函数并分析时间复杂度：

   (a) 计算排列数 A(m,n)=n*(n-1)*...*(n-m+1)

   (b) 计算组合数 C(m,n)=A(m,n)/A(n,n)

笔记

2.7 字符串的处理

本节我们将熟悉一些字符串的基本处理算法。字符串的创建有两种方法，一种是用单引号一种使用双引号。这两种符号在 Python 当中并没有什么明显区别，但在 Baltamatica 当中，使用双引号创建的是一个字符串对象（类似于 C++ 中的 string 类型），而使用单引号创建的是一个字符数组（类似于 C 当中的 char[] 或者说 char* 类型）。二者的使用是有很大区别的，比如，如果我想要分析一个 string 对象的 size，我写：

```
a="this";
b='this';
size(a)
size(b)
```

你就会发现 a 是 1*1 的数组而 b 是一个 1*4 的数组，使用下标索引的时候也可以用 b(2) 把 h 索引出来，但是 a 只能下标一个 a(1) 出来。a 作为 string 类型是一个整体，b 作为 char 类型是一个数组，使用方法是不一样的。如果需要进行索引、查找等操作，我认为用 char 数组表示的方式更加易于理解；如果创建的是字符串矩阵、字符串向量等则更加倾向 string 类型。

常见的操作例如 strcat，可以将两个字符串拼接在一起，对于 string 类和 char 类都有效；使用 length 函数可以求出字符串长度，对 char 类有效；strcmp 可以比较两个字符串是否相等，对于输入的参数是 string 还是 char 没有特殊要求。使用 lower 函数可以将所有字母改成小写，upper 函数将所有字母改成大写等等。

字符串也是可以排序的，对字符串的排序可以是用 char 数组组成的单一字符串，也可以是由 string 类构成的字符串数组。值得注意的是，如果是 char 数组在一维上构成向量（例如 ['this','is','my','cat']）那么返回的是把数组中每个字符串拼接在一起的结果（也就是'thisismycat'）。这个结果也好理解，本质上就是把四个连续的数组都拼在一片连续空间，那么它们的维度也就自然拼接起来了。对于字符串数组的排序通常取每个字符串的首字母排序。比如，我们看下面两个例子：

```
    sort(["this","is","my","cat"])
    sort(['this','is','my','cat'])
```

你就会发现第一个排序的结果为"cat"、"is"、"my"、"this", 而第二句的结果为：'achiimsstty'。这是由于一个对字符串的首字母排序，一个是对字符数组排序，但排序标准都是字符对应的 ASCII 码。

练习应用

1. 对于两个 char 数组 str1='this is my car' 和 str2='is'，设计算法实现 strcmp 的功能（即比较 str1 和 str2 是否完全相同），并将这份代码封装为函数 cmp.m

2. 在问题 1 的基础上，如果我只想比较两个字符串的前 n 位，将新的代码封装为 strncmp.m

3. 对于两个 char 数组 str1 和 str2，若 str1 在 str2 的某个部分出现则称 str1 为 str2 的一个子串。请设计算法，找出 str1 在 str2 中第一次出现在 str2 中的下标是多少。（提示：这个地方如果暴力算法很慢还可以使用一种方法叫 KMP 算法，具体的过程可以参考《数据结构》一书）

4. 在问题 3 的基础上，将函数封装为 strsearch，如果学有余力的同学可以比较一下暴力法和 KMP 算法的复杂度。

5. 在问题 3 的基础上，如果我想把 str1 中所有出现 str2 的部分用一个新的字符串 str3='kk' 替换应该如何实现，并将函数封装为 strrep(str1,str2,str3)

笔记

2.8 元胞，矩阵与表格

本节我们的学习内容是介绍在数组的基础上更深入的元胞、矩阵和表格三种数据类型。这几种数据类型呢可能用的没有那么多和深入，就不多做探究哈，我们暂且介绍一些概念性的东西在里面。

首先元胞 cell 结构，其实从本质上来讲元胞按照数据结构应该说是一个广义表，因为它可以把不同数据类型的元素放到一个集合里面，并且这个集合不易被修改。元胞的创建和索引都使用大括号进行，例如，我们创建一个元胞数组：

```
1 c={12,"university",1+4i}
```

这种方式创建了一个 1*3 的元胞数组，第一项是一个浮点数，第二项是一个字符串，第三项是一个复数。它的使用就比传统的数组矩阵概念都要更广，但它的修改并不容易。我们刚才说到，元胞数组的索引是使用 {} 的，在 MATLAB 当中元胞数组可以通过使用 {} 和 () 两种方式索引，但在 Baltamatica 当中目前还只能支持索引的方法。如果想要修改某个值，可以索引 c{2}=4i 这样子。但如果想删除某一项，传统的数组 vec 可以用 vec(2)=[] 来删除第二项，但在元胞数组中如果 c{2}=[] 只能将这一项设置为空数组，还是会占用第二项的位置和内存区域。所以想要从存储区上删除它仍然是一件很难的事情，这一点我认为也是主创团队需要加以改进的地方之一。

如果想要创造空的元胞数组，可以使用 cell(n) 或者 cell(m,n) 方法。

表格数据通常来讲是以矩阵的形式存放在 Baltamatica 当中，而根据类型又可以分为纯数值的数值矩阵和混合类型的元胞矩阵。在向量与数组一节中我们已经提到过二维数组的索引和矩阵的基本代数运算，我们还提到特殊矩阵的创建、分解和塑形（利用 reshape 函数，例如 reshape(A,3,2) 可以将 A 重塑为三行两列的矩阵），但在使用 find 命令的时候我们留了一个小关子：为什么 find(A>0.5) 的数组下标返回的是一个数而不是横纵两个坐标呢？

事实上，二维甚至更高维度的数组其实都是顺序压缩存储的。它们在内存空间中不会以表格的形式排列而是以线性排列的方式存储表格中的元素数据。只不过，将线性内存中的数据还原回表格，如何根据表格中的行列号去索引定位呢？这涉及到二维坐标和一维坐标的变换问题。在计算机中，二维以及更高维数组的存储有两种方式：行序优先和列序优先。

行序优先的数组存储是将二维数组的每行进行拼接形成的结构。如图2.5所示。如果现在存储的是一个四行五列的矩阵 A(4,5)，起始元素 A(1,1) 在内存中的地址为 0x0074，矩阵的数据类型为 int32 整数（32bit=4B，一个元素的地址得加 4），那么计算元素 A(3,2) 在内存中的地址怎么算呢？A(3,2) 前面已经过了两行，现在来到了第三行，所以是第（2*5+2）个元素，地址 0x0074+4*(2*5+2)=0x0074+0x0030=0x00a4（计算机内地址用 16 进制表示，0x 开头表示 16 进制），这就是它的地址。总结公式：对于 m 行 n 列的矩阵 A，若用 p 表示某个变量的地址，每个元素字长为 D，那么元素 A(i,j) 的地址为：

$$p[A(i,j)] = D \times (m \times (i-1) + j) + p[A(1,1)] \tag{2.6}$$

图2.6是列序优先的示意图, 它是将数组按列纵向拼接。同样对于 A(4,5), 那么如果计算 A(3,2) 的地址,是经过了一列,到第二列的第三项,也就是第 (1*4+3) 个元素,地址 0x0074+4*(1*4+3)=0x0091。所以元素 A(i,j) 的地址为:

$$p[A(i,j)] = D \times (n \times (j-1) + i) + p[A(1,1)] \tag{2.7}$$

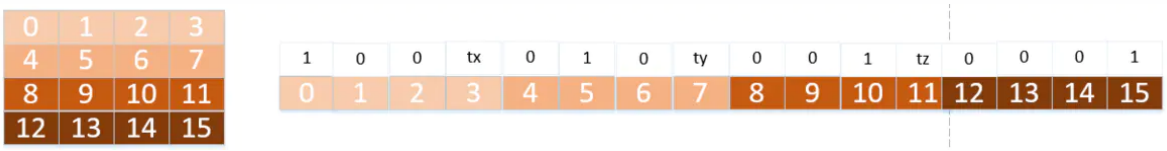


图 2.5: 行序优先存储的示意图

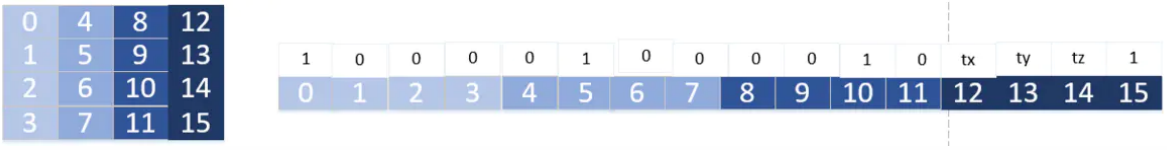


图 2.6: 列序优先存储的示意图

特殊地，如果一个矩阵里面大部分元素都是 0 只有少部分非 0 元素，我们只需要存储非 0 元素的值和下标就够了。这样的存储方式叫做稀疏矩阵，在 Baltamatica 当中提供了 sparse 工具可以用于创建稀疏矩阵。sparse 矩阵可以将普通矩阵以稀疏矩阵的方式存储，或者通过非零元素的下标和对应值还原稀疏矩阵。例如，可以通过下面的方式创建稀疏矩阵:

```
ii = [1, 2, 3, 7, 10];
jj = [1, 2, 3, 3, 10];
vv = [1, 1, 1, 3,  3];
A = sparse(ii, jj, vv);
```

稀疏矩阵通过三元组的方式（存储每个非零元素的横纵坐标和数值），将零元素过滤掉，实现了压缩存储，大大节约了内存空间。

练习应用

1. 上三角矩阵是对于一个 n*n 的矩阵而言，主对角线下方的元素全部为 0，只有对角线上有元素需要存储。那么对于上三角矩阵的压缩存储，若按行序优先且对角线下方元素无需存储，请给出元素 A(i,j) 的地址计算公式（已知首地址为 p，存储 double 类型变量），并用 Baltamatica 实现这个地址变换的函数保存为 delmatadd.m

2. 稀疏矩阵的存储使用三元组形式存储，那么若给定一个矩阵，请写一个函数 mat2spr.m，根据三元组形式将矩阵 A 中的非零元素压缩。返回结果是一个元胞数组，结构为 {非零元素个数，[矩阵行，矩阵列]，[x,y,A;x,y,A;...,x,y,A]}。矩阵 A 的测试用例为 A=[0 1 0 0 0 0; 0 0 0 0 10 0; 0 0 5 0 0 0; 9 0 0 -3 0 0; 0 0 0 0 0 -7];

3. 接问题 2，若给出了问题 2 中的元胞数组，请你设计函数 spr2mat.m 将其还原为一个普通矩阵 A。矩阵 A 的测试用例为 {6,[5,6],[1 2 1;2 5 10;3 3 5;4 1 9;4 4 -3;5 6 -7]}

笔记

2.9 结构体类型

在 C 语言中有结构体，而在 C++/Python/Java 中有面向对象的支持可以将不同的数据元素和操作方法集合为一个类。Baltamatica 本身是一个面向过程编程的工具，所以也会有结构体类型支持类似运算。

struct 其实是一类特殊的数据类型，它将不同类型的数据复合到一个整体中存储。结构体中最基本的要素就是字段和值，如果希望和面向对象当中一样使用一种类似于面向对象的模式那么可以把方法（函数字段）用句柄或者说匿名表达式书写作为特殊的字段调用。

创建一个 struct 类型的变量有两种方法，第一种是直接赋值法：

```
data=struct('x',1:100,'y',sin(1:100),'title','sin(x)');
```

在这种赋值方法中，每个字段用字符串写出字段名称，然后跟上它的取值。取值的类型可以是任意的，单一的整数、浮点数、字符、字符串、布尔变量，或者向量、矩阵、元胞数组，以至于函数句柄，甚至结构体里面嵌套结构体也不是不可能的事情。

另一种方法是间接赋值法，比如：

```
%通过使用圆点表示法添加字段来创建一个结构体
data.x = 1:100;
data.y = sin(data.x);
data.title = 'sin(x)'
```

这里在创建变量的过程中识别到了点号，那么 Baltamatica 的编译器会自动识别出这是一个结构体变量，现在在对它以 x 为名字的字段进行赋值，于是自动开辟一片内存空间存放字段 x。同样的，也就有了对 y 和 title 的赋值。若是每次都对这么多字段进行赋值似乎我们会感觉很繁琐，所以，我们也可以模仿 C++ 当中的构造函数实现对结构体变量的赋值。

练习应用

1. 将前面的学生成绩管理系统用结构体实现，要求：data.id 表示学号，data.k1 表示语文，data.k2 表示数学，data.k3 表示英语，data.score 表示总分，data.score 表示排名。
2. 在问题 1 的基础上新增功能，删除某个学号为 id 的学生的一行成绩。
3. 试着随意写一个结构体，在里面引用函数句柄实现类似于面向对象的方法。

2.10 文件读写与报错排查

本节将为大家介绍文件的读写命令与报错测试。

Baltamatica 加载文件的方式有三个：readmatrix，load 和 csvread。首先说 load，它是读取.mat 文件的方式，.mat 文件也就是将 MATLAB 或者 Baltamatica 的数据保存的文件格式。使用方法形如：load fisheriris 即可。csvread 是一种读取.csv 表格的函数格式，它只能用于读取 csv 格式的电子表格到矩阵中去。而相对比较灵活多变的就是 readmatrix 函数，可以读取 txt、csv 和 xls 或 xlsx 文件，当指定 FileType 为"text" 时可以读取 txt 和 csv 文件；如果指定 FileType 字段为 spreadsheet 则可以读取 xlsx 或 xls 文件。例如：

```
A=readmatrix("test_demo.csv","FileType","text","OutputType","string")
```

这一指令的用法其实就是读取 test_demo.csv，指定文件类型按 text 读取，输出为字符串数组。还可以用 Range 指定单元格范围，Delimiter 指定分隔符等。更多的用法可以试着 help readmatrix 查看文档。

而 Baltamatica 写入文件可以使用 writematrix 函数，调用格式其实和 readmatrix 是一样的。但是当 FileType 字段取"text" 时默认保存数据为 txt 文件，如果想要保存为 csv 需指定目标文件的后缀名为.csv，并且分隔符默认为逗号，如果想要指定分隔符为空格、制表符等可以指定 Delimiter 字段为' '、'
t' 等。写文件的方式有两种，一种是 overwrite（把之前的文件清空重写），还有一种是 append（从老文件最后一行往后追加），这两种模式在字段 WriteMode 当中可以选择。

就报错排查的问题来看，Baltamatica 对这一块的支持是做的不太好的。只有一个警告指令 warning 可以使用，MATLAB 最基本的核心语法之一也就是 try-catch-end 指令没能够实现。如果使用 try-catch-end 指令程序员可以在写程序的时候自行判断错误类型并做出自己的相应处置，即将测试和开发整合到一起的一个编程风格。虽然在编程时使用太多 try-catch 语句会显得你是个新手没有足够 clean 的编程风格，但是这玩意不能没有啊，它还是得有。这是一个很大的弊端。

Baltamatica 实现报错排查的方式（除了一些闪退的错误以外）主要是通过断点调试进行的。这一方法在 Visual Studio、Dev-C++、VSCode、Pycharm、eclipse 和 MATLAB 当中都有所集成，想要开发一个断点调试的功能主要是识别中断、添加断点、逐步运行的过程，不过因为.m 本身就是解释性语言，在编译器的开发过程中按照数据流去逐步运行断点前中后并不是什么太难理解的事情。具体的原理还是参考编译原理和操作系统的内容吧，我就不扯淡如何在 IDEA 里面加断点这种东西了。具体使用 debug，可以点击要设断点的行号，有小圆圈后点击下面那个小虫子就是 debug 模式逐步运行，并且打印出函数栈。

笔记

第三章　基于 Baltamatica 的矩阵运算

本章我们学习的重点内容是 Baltamatica 的矩阵运算。本章会在第二章讲向量与矩阵的基础上给大家讲解矩阵的一些常见线性代数运算，以及一些重要的方法例如奇异值分解、高斯消元等。为了让大家更好理解 Baltamatica 是如何加快底层矩阵计算的运行速度的，我们还安排了一节选学节《大矩阵乘法的并行化算法》可供参考。而最后，我们会讲到一系列综合评价方法。大家在学习综合评价方法的时候始终要抓住两个要点：权重和评分。评价类模型是初次接触数学建模竞赛的同学非常喜爱的一类题型，因为代码量小，原理通俗易懂。这类问题主要是用以解决对于一个目标不同方案之间比较或不同影响因素之间比较的问题。

本质上我们去学习评价模型这一章就是把握两个核心：权重和评分。相对主观一些的评价方法包括层次分析法、模糊评价法等，客观一些的评价方法包括主成分分析、TOPSIS 分析等。这些分析方法相对来说与需要解决的问题和进行评价的人有关，重点往往不是在于代码实现，而是在结果的阐释和合理性评估，以及利用结果去进行决策。但无论如何，我们的核心目的还是为了给不同的决策变量赋权综合获得一个得分，然后得分排序进行评价。

总体来说，任何评价方法的流程步骤都得遵循如图3.1以下步骤：

图 3.1: 评价算法的执行流程

3.1 基本线性代数

大家应该学过线性代数，如果没有学过这里带着大家复习一下。线性代数是一门基础数学科目，基本上所有理工科学生大一的时候都得学线性代数。如果是数学系可能学的就是高等代数了。这门课主要是研究矩阵与向量的数学理论，也会探究线性方程组的解等相关问题。我并不是一个线代老师，这里也不打算抢线代老师的饭碗，这一小节我们仅仅引入线性代数中比较重要的一些定义和计算方法。

我相信大多数同学高中毕业是记得向量这个概念的，但中学阶段我们也仅仅是接触到了三维向量。事实上向量的维数可以是很多维，从代数的意义上你可以认为向量是一个集合，从几何的意义上你又可以认为向量是一个 n 维欧几里得空间中的一个点。

和二维、三维空间中的向量一样，高维空间中的向量同样可以进行加减运算、数量乘运算和数乘运算。但毕竟这是一门应用数学课程，我们不打算把太多精力放在任何一本线代课本里面都能找到的公式上，使用 matlab 举例子恐怕会更加直观。从程序设计的角度来看，如果读者接触过 C 语言应该会了解数组的概念，而在 C++ 语言中 STL 里面已经包含了 vector 类型。

我们可以在 matlab 中创建两个向量。用逗号或者空格隔开的是行向量而用分号隔开的是列向量，通常只有同为行向量或者同为列向量才能做加减和放缩，但数乘则会有趣一些。另外，行向量与列向量之间的转化可以通过转置符号’实现。

数通过集合形成了向量，那向量集合以后又会变成什么呢？如果向量只是沿着同一个方向进行拼接，那么得到的只不过是一个更长一些的向量。但如果是在纵向上做拼接，那么我们或许可以把一个向量排成矩阵。那么矩阵作为向量的集合，自然也保留了向量的一些特性，包括行列相同的矩阵的加减法、数量乘。比较有趣的是矩阵的乘法，它把两个矩阵分别按行、列规约：

$$\mathbf{A}_{(\mathbf{m},\mathbf{n})}\mathbf{B}_{(\mathbf{n},\mathbf{k})} = \begin{pmatrix} a_1 \\ \vdots \\ a_m \end{pmatrix} \begin{pmatrix} b_1 & \dots & b_k \end{pmatrix} = \begin{pmatrix} a_1 b_1 & \dots & a_1 b_k \\ \vdots & \ddots & \vdots \\ a_m b_1 & \cdots & a_m b_k \end{pmatrix}_{(m,k)} \tag{3.1}$$

矩阵排列好以后除了按行可以规约为一群向量的纵向分布，也可以按列规约成一群向量的横向分布。于是才有了上面矩阵乘法的这个计算方法。每一项都是两个同为 n 维的向量数乘。注意，矩阵乘法的维度要求第一个矩阵的列数和第二个矩阵的行数相等，因此要注意维度问题。另外矩阵的乘法没有交换律，AB 不一定等于 BA，甚至可能根本没有 BA。

矩阵运算中，正如前面在向量当中提到的，.* 是元素乘，而矩阵乘法需要让前列和后行相等，所以用 A’*B，数学上记作 ATB。那我们能否类比向量的模，提出“矩阵的模”这样一个概念呢？在线性代数中确实存在这样一个类似的概念，这个概念叫行列式。行列式虽然同样是排成了一个表，但注意，矩阵是一个表格，行列式是一个数，它的值是可以算出来的！有关行列式的计算方法有很多，但最经典最通用的方法是代数余子式展开法。代数余子式的本质就是递归式求解，将行列式 A 中下标为（i,j）的元素所在行和所在列全部去除以后求新的行列式，再乘上对应的符号。

$$A = \sum^{n} a_{ij} A_{ij} \tag{3.2}$$

将 n 阶行列式降低到 n-1 阶，n-1 阶再降低为 n-2 阶，逐层展开到最后二阶，整个行列式求解就可以完成了。不过高阶行列式如果不是特殊行列式计算会有些复杂，我们可以将这个过程交给计算机程序来完成。而递归到最后，这个问题还是二阶行列式的计算。底层计算：

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc \tag{3.3}$$

Baltamatica 当中求行列式可以用 det。这样我们就可以在命令行输出矩阵 A 对应的行列式的值。有了行列式的概念，我们可以用它定义矩阵的逆矩阵计算方法。一个行列数均为 n 的矩阵的逆矩阵满足这样的定义：

$$AA^{-1} = A^{-1}A = I = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix} \tag{3.4}$$

3.2 * 大矩阵乘法的并行化算法

这一节是选学节，主要回顾了大矩阵乘法中的并行化方法，以及如何用 C 语言加速矩阵乘法的运算。为大家介绍北太天元 SDK 开发的流程，但是我这里可能不会放代码出来，大家下去可以试着做一下这个任务。

首先矩阵的乘法应当是如何计算？如果什么都不考虑就做串行化计算，设两个矩阵 A 和 B，大小分别为 M*N 和 N*P，如果 C=A*B，则 C 的大小为 M*P。那么普通的矩阵乘法运算可以写成：

```
C=zeros(M,P);
for i=1:M
    for j=1:P
        for k=1:N
            C(i,j)=C(i,j)+A(i,k)*B(k,j);
        end
    end
end
```

我们仔细看这段代码，就会发现它里面用到了三重循环所以时间复杂度达到了 $O(n^3)$ 量级。这也就是为什么在 Python、MATLAB 和 Baltamatica 当中尽可能多地使用向量运算而非循环。那么，诸如 MATLAB 的向量运算和 Python 的 numpy 之所以能够做到速度更快其归根结底还是在底层也就是 C 开发阶段对底层数据结构和数据流水线进行了优化使其能够并行计算。

在处理大矩阵运算的时候，我们可以利用前面学习矩阵压缩存储公式的部分将外面的两重循环进行整合：

```
C=zeros(M,P);
for z = 1 : M * P
    i = floor(z/P);%整除
    j = mod(z,P); %取余数
    for k = 1:N
        C(i,j)=C(i,j)+A(i,k)*B(k,j);
    end
end
```

为何要这样展开呢？虽然这样展开对 Baltamatica 没有什么明显的时间复杂度上的优化，但是对于底层 C 而言是很有效果的。因为在 C 语言中，如果搭载了 OpenMP/MPI 等并行计算框架，使用 mpigcc 编译的时候可以把第一重循环在机器代码级展开为并行（因为从数据流水上来看一重循环的数据并没有很强的时间局部性）。例如，如果我们现在按照 OpenMP 的框架把这一段改写为 C 语言：

```
#prama omp parallel for num_threads(NUM)
for (z = 0; z < M * P; ++z){
                    i = z / P;
                    j = z % P;
                    C[i][j] = 0;
                    for (k = 0; k < N; ++k){
                            C[i][j] += A[i][k] * B[k][j];
        }
}
```

而在向量计算的方面，我们也讲过，二维矩阵在压缩存储的时候它和一维存储模型之间的地址变换公式。其实，对于 B[k][j] 而言，它每经历一次内循环其实并不是找它相邻的地址，而是隔了一行，所以每次需要计算变换坐标。操作系统会根据这个地址变换检索快表 (TLB)、缓存 (Cache)、内存页表 (Page)，一级一级查找这就会消耗太多时间。如果要使用的下一项就在这一项的隔壁，那检索就不需要经过太多地址变换而是本能性地找下一块就可以了。所以要想加速，应该是 B[j][k]，并且在计算之前把 B 做一个转置:

```
C=zeros(M,P);
B=B';
for z = 1 : M * P
    i = floor(z/P);%整除
    j = mod(z,P); %取余数
    for k = 1:N
        C(i,j)=C(i,j)+A(i,k)*B(j,k);
    end
end
```

我们再考虑一个更大的问题，比如说要计算两个超大矩阵的乘法，这两个矩阵都是巨大，很大，非常大，一个计算机的内存都不够，得用好几台计算机存这个矩阵。矩阵连存储都困难，更不要谈计算了。而在并行与分布式计算的框架下，如果说我用九台计算机进行一个矩阵的乘法运算，首先需要思考：一个乘法我怎么把它拆九步，每台计算机分多少的任务量，每台计算机的中间结果是不是互相需要调用和通信。所以这个地方有一个重要的方法就是 Cannon 算法如图3.2:

将矩阵 A 和 B 分成 p 个方块 $A_{i,j}$ 和 $B_{i,j}(0 \leqslant i,j \leqslant \sqrt{p}-1)$，
每块大小为 $(n/\sqrt{p}) \times (n/\sqrt{p})$，并将它们分配给 $\sqrt{p} \times \sqrt{p}$ 个处理器 $(P_{0,0}, P_{0,1}, \cdots, P_{\sqrt{p}-1,\sqrt{p}-1})$。
开始时处理器 $P_{i,j}$ 存放有块 $A_{i,j}$ 和块 $B_{i,j}$，并负责计算块 $C_{i,j}$，然后算法开始执行：

① 将块 $A_{i,j}(0 \leqslant i,j < \sqrt{p})$ 向左循环移动 i 步；
　将块 $B_{i,j}(0 \leqslant i,j < \sqrt{p})$ 向上循环移动 j 步；
② $P_{i,j}$ 执行乘－加运算；
　然后，将块 $A_{i,j}(0 \leqslant i,j < \sqrt{p})$ 向左循环移动 1 步；
　将块 $B_{i,j}(0 \leqslant i,j < \sqrt{p})$ 向上循环移动 1 步；
③ 重复第②步，在 $P_{i,j}$ 中共执行 $\sqrt{p}$ 次乘－加运算和 $\sqrt{p}$ 次块 $A_{i,j}$ 和 $B_{i,j}$ 的循环单步移位。

图 3.2: Cannon 算法的执行流程

上述的伪代码可能比较难懂，我们举个简单的例子。比如我要将两个矩阵都切分为 9 块，有九台机器构成一个网格化集群，每台机器处理一块矩阵。它会不断移位求和然后得到计算结果。如图3.3所示，每台处理器存储一块 A 和一块 B，当一次迭代完成时，处理器将自己的 A 通过网络通信传给左边的处理器，并把自己的 A 置换为右边处理器传来的 A（如果处理机本身就在最左侧那么把自己的 A 消息通信给最右侧的处理器）；处理器还将自己的 B 通过网络通信传给上边的处理器，并把自己的 B 置换为下边处理器传来的 B（如果处理机本身就在最上侧则把自己的 B 消息通信给最下侧的处理器）。如此循环往复一轮，在每一次通信的过程中，九台处理器内部存储的 A 和 B 相乘，结果发往主服务器拼接为一个大型矩阵 C，然后每一次通信计算得到的结果就向 C 进行累加。

现在给大家一个作业，但这个作业不是必须完成的。这个任务有很大难度，你需要懂 C/C++ 开发。北太天元在安装的时候会有一个 baltam_sdk 的文件夹，里面放的是 SDK 开发文档和数据结构有关的库 bex.h。请参考有关文档，文档中为 Baltamatica 定义了专门的数据结构（虽然 C Project 所需要的体量并没有那么恐怖，即使你只有一个 C 文件去 make 或者 gcc 也都是可以的），通过 gcc 或者 make 等工具你需要将你的 C 项目动态编译出.dll 库文件。你可以认为里面提供的北太数据类型算是某种 STL（这又何尝不是一种 NTR），试着编译这个并行矩阵乘法并将结果保存为 Baltamatica 函数 MyMul(A,B)，使其能够在 Baltamatica 当中正常运行（时间效率不作要求，能跑就是带成功）。

- 第一轮计算：

$$\begin{array}{ccc|ccc} A_{00} & A_{01} & A_{02} & B_{00} & B_{11} & B_{22} \\ A_{11} & A_{12} & A_{10} & B_{10} & B_{21} & B_{02} \\ A_{22} & A_{20} & A_{21} & B_{20} & B_{01} & B_{12} \end{array}$$

- 第二轮计算：

$$\begin{array}{ccc|ccc} A_{01} & A_{02} & A_{00} & B_{10} & B_{21} & B_{02} \\ A_{12} & A_{10} & A_{11} & B_{20} & B_{01} & B_{12} \\ A_{20} & A_{21} & A_{22} & B_{00} & B_{11} & B_{22} \end{array}$$

- 第三轮计算：

$$\begin{array}{ccc|ccc} A_{02} & A_{00} & A_{01} & B_{20} & B_{01} & B_{12} \\ A_{10} & A_{11} & A_{12} & B_{00} & B_{11} & B_{22} \\ A_{21} & A_{22} & A_{20} & B_{10} & B_{21} & B_{02} \end{array}$$

图 3.3: Cannon 算法的执行流程手写版本

3.3 矩阵的奇异值分解

本节我们看到矩阵的奇异值分解算法。特征值分解仅适用于提取方阵特征，但在实际应用中，大部分数据对应的矩阵都不是方阵；矩阵可能是有很多 0 的稀疏矩阵，存储量大且浪费空间，这时就需要提取主要特征；奇异值分解是将任意较复杂的矩阵用更小、更简单的 3 个子矩阵的相乘表示，用这 3 个小矩阵来描述大矩阵重要的特性。

在使用线性代数的地方，基本上都要使用 SVD。SVD 不仅仅应用在 PCA 、图像压缩、数字水印、推荐系统和文章分类、LSA (隐性语义分析)、特征压缩 (或数据降维) 中，在信号分解、信号重构、信号降噪、数据融合、同标识别、目标跟踪、故障检测和神经网络等方面也有很好的应用，是很多机器学习算法的基石。

奇异值分解将矩阵 A(m,n) 分解为 U(m,n)、$\Sigma(m,n)$ 和 V(n,n) 的乘积。其中，Σ 是一个类似于对角矩阵的结构。也就是：

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V} \tag{3.5}$$

这三个矩阵的求解方式如图3.4所示：

矩阵	别称	维度	计算方式	含义
U 矩阵	A 的左奇异矩阵	m 行 m 列	列由 AA^{T} 的特征向量组成，且特征向量为单位向量	包含了有关行的所有信息（代表自己的观点）
Σ 矩阵	A 的奇异值矩阵	m 行 n 列	对角元素来源于 AA^{T} 或 $A^{\mathrm{T}}A$ 的特征值的平方根，并且按降序排列，值越大可以理解为越重要	记录 SVD 过程（是一种日志）
V 矩阵	A 的右奇异矩阵	n 行 n 列	列由 $A^{\mathrm{T}}A$ 的特征向量组成，且特征向量为单位向量	包含了有关列的所有信息（代表自己的特征）

图 3.4: 矩阵的奇异值分解

虽然 Baltamatica 当中已经实现了 svd 函数，我们不妨构造一个自己的 SVD：

```
function [U,Q,V]=qiyi(A)
if rank(A)==0;
    disp('无奇异值分解，重新输入')
end
AV=A'*A;
[x,y]=eig(A);          %求解特征值以及特征向量
[m,n]=size(A);
for j=1:n;
    x(:,j)=x(:,j)/norm(x(:,j),2)      %norm（A,2） 取2范数
end
V=x
AU=A*A';
```

笔记

```
[x1,y1]=eig(A);
[m1,n1]=size(A);
for  j=1:n;
     x1(:,j)=x1(:,j)/norm(x1(:,j),2);
end
U=x1;
[x2,y2]=eig(A'*A);
Q=sqrt(y2);
end
```

应用案例

SVD 被用于推荐系统数据

SVD 在推荐系统当中有着很重要的应用。在一个推荐系统数据集中行代表用户 user，列代表物品 item，其中的值代表用户对物品的打分。基于 SVD 的优势在于：用户的评分数据是稀疏矩阵，可以用 SVD 将原始数据映射到低维空间中，然后计算物品 item 之间的相似度，可以节省计算资源。

整体思路：先找到用户没有评分的物品，然后再经过 SVD“压缩”后的低维空间中，计算未评分物品与其他物品的相似性，得到一个预测打分，再对这些物品的评分从高到低进行排序，返回前 N 个物品推荐给用户。

联系前面与稀疏矩阵、线性代数和矩阵分解的知识，整体推荐的操作流程如下：

- 加载测试数据集；
- 定义计算相似度的方法；
- 通过计算奇异值平方和的百分比累计占比来确定将数据降到多少维才合适，返回需要降到的维度；
- 在已经降维的数据中，基于 SVD 对用户未打分的物品进行评分预测，返回未打分物品的预测评分值；
- 产生前 N 个评分值高的物品，返回物品编号以及预测评分值。

给出测试用例，每一行代表一个用户，每一列代表一个产品，如果矩阵为 0 表示用户未发现此类产品。请基于上述材料和 SVD 分解的知识对这一稀疏矩阵进行奇异值分解并压缩到一个低维度空间，预测每个用户对各个产品的评分并给出最可能推荐的产品。相似度可以用余弦相似度

表示。如果对维度空间压缩感到很困难也可以提前参考后文主成分分析的流程，然后再来进行一些尝试。

$$\cos(u,v) = \frac{uv}{|u||v|} \tag{3.6}$$

```
[[0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 5],
[0, 0, 0, 3, 0, 4, 0, 0, 0, 0, 3],
[0, 0, 0, 0, 4, 0, 0, 1, 0, 4, 0],
[3, 3, 4, 0, 0, 0, 0, 2, 2, 0, 0],
[5, 4, 5, 0, 0, 0, 0, 5, 5, 0, 0],
[0, 0, 0, 0, 5, 0, 1, 0, 0, 5, 0],
[4, 3, 4, 0, 0, 0, 0, 5, 5, 0, 1],
[0, 0, 0, 4, 0, 4, 0, 0, 0, 0, 4],
[0, 0, 0, 2, 0, 2, 5, 0, 0, 1, 2],
[0, 0, 0, 0, 5, 0, 0, 0, 0, 4, 0],
[1, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0]]
```

3.4 解线性方程组

本节我们将学习如何使用矩阵运算解线性方程组的用法。线性方程组的基本形式就例如说：

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \tag{3.7}$$

那么，如果用矩阵形式去写，也可以写作：

$$\mathbf{Ax} = \mathbf{b} \tag{3.8}$$

我们现在暂且讨论这个方程组的系数矩阵 A 是一个方阵，如果方程数比自变量数多，有可能产生解的冲突导致无解；如果自变量的个数比方程多那么有可能有无穷解，是不定方程。对于不定方程其实理应符号运算要比数值运算来的更好，但是 Baltamatica 的符号运算基本停摆了……这个问题我们后面还得说。所以我们这一节就处理方阵问题。

如果这个系数矩阵作为方阵可逆，那么很漂亮，使用矩阵求逆就可以求出方程组的解：

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \tag{3.9}$$

Baltamatica 提供了一个左除运算符“用于求解线性方程组，它使用列主元消去法，使用起来十分方便。对于线性方程组 Ax=b，可以利用左除运算符反斜杠求解，b 左除以 A 可获得线性方程组的数值解 x。

但是矩阵方程组的求解并不一定都需要求逆，有时候系数矩阵也可能求不出逆。因为求逆有着比较大的时间复杂度和空间占用，我们希望把矩阵进行分解来降低各个部分的空间和运行时间，所以又有了基于矩阵分解的线性方程组求解。

一种常用的方法叫 LU 分解法，矩阵的 LU 分解就是将一个 n 阶矩阵表示为一个下三角矩阵和一个上三角矩阵的乘积。线性代数中已经证明，只要方阵是非奇异的，LU 分解总是可以进行的。也就是说，将 A 进行 LU 分解为 $\mathbf{A} = \mathbf{LU}$，原始方程可以等价于求两个退化方程：

$$\begin{cases} \mathbf{Ly} = \mathbf{b} \\ \mathbf{Ux} = \mathbf{y} \end{cases} \tag{3.10}$$

例如：

```
A=[2,1,-5,1;1,-5,0,7;0,2,1,-1;1,6,-1,-4];
b=[13,-9,6,0]';
[L,U]=lu(A);
x=U\(L\b)
```

笔记

可以看到，方程的解等于 [-66.5556,25.6667,-18.7778,26.5556]。不过这也都是数值解了，至少 0.6667 的部分明眼人应该都能看出来其实就是 2/3。符号解准确解的功能开发还得等有关的工作人员开发吧。

迭代法是一种不断用变量的原值推出它的新值的过程，是用计算机解决问题的一种基本方法。在第五章当中我们会看到更多的迭代案例。这里介绍一个用迭代思想解矩阵问题的方法，也就是雅各比迭代。它将一个矩阵 A 分解为一个对角阵 D，上三角阵 U 和下三角阵 L，三者之间的关系为 $\mathbf{A}=\mathbf{D}-\mathbf{U}-\mathbf{L}$。而方程组的求解公式：

$$\mathbf{x}^{(k+1)}=\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\mathbf{x}^{(k)}+\mathbf{D}^{-1}\mathbf{b} \tag{3.11}$$

baltamatica 当中是没有雅各比迭代法的函数的，不过我们可以自己写一个：

```
function [y,n]=jacobi(A,b,x0,ep)
    D=diag(diag(A));
    L=-tril(A,-1);
    U=-triu(A,1);
    B=D\(L+U);
    f=D\b;
    y=B*x0+f;
    n=1;
    while norm(y-x0)>=ep
        x0=y;
        y=B*x0+f;
        n=n+1;
    end
end
```

我们看一下雅各比迭代法相比于 LU 分解法的精度如何，这个方法就是一种时间换空间的方程组计算。为了充分比较精度，可以在测试用例的最开始声明 format long，也就是用长整数、双精度把有效数字都显示出来比较。于是，有：

```
format long
A=[4,-2,-1;-2,4,3;-1,-3,3];
b=[1,5,0]';
[x,n]=jacobi(A,b,[0,0,0]',1.0e-6)
[L,U]=lu(A);
x=U\(L\b)
```

运行结果在小数点后第 7 位才开始有差异，在一般的工程应用中是完全可以得到精度需求的。迭代 35 次以后结果收敛。因为本身我们的误差控制 eps=1.0e-6，如果还想调节精度需求也可

以设置 eps=1.0e-9 等，是可控的。

而当碰上更一般的线性方程组的时候，有一种更通用的方法叫做高斯消元法。高斯消元法是希望通过初等行变换来将系数矩阵及其带常数项的增广矩阵变成阶梯型，然后从阶梯的最底部开始自底而上还原方程的解。比如说，对于一个增广矩阵 B=(A|b)，将矩阵的每一行减去第一行的常数倍使得从第二行往后的每一行首项都为 0；再将第二行起后面每一行减去第二行的常数倍，使得第三行往后每一行第二项都为 0；以此类推……；到最后，矩阵的最后一行变换到只剩下两项：一个系数项，一个常数项，新的矩阵为 G=(P|v)。

新的矩阵 P 为上三角矩阵，v 是常数项矩阵，从最后一项开始解方程，然后逐一回代就可以得到整个方程的解。在我们进行消去回代的过程中如果主元素（位于主对角线上的元素称为主元素）为 0，那么消去过程和回代过程都不能顺利进行。因为 0 乘以任何非 0 常数都为 0，达不到消去其他行的目的；因为回代过程需要主元素做除数，所以主元素不能为 0。

除此之外，即使在进行高斯消去过程中不会出现主元素为 0 的情况，但是出现了主元素比较小（相对于其他元素）的时候，也不能直接使用高斯消去法，因为在回代过程中会放大舍入误差，严重影响解向量结果的精度。这个时候可以使用选主元技术在一定程度上可以避免这种情况。

我们可以编写几个函数来构成高斯消元法：

```
clc
clear all
while 1
    A=input('请输入系数矩阵（方阵A）：');
    b=input('请输入常数向量（列向量b）：');
    [H,L,MARK]=isLegal(A,b);
    if(MARK==1)
        [Ax,bx]=GSelimination(A,b,H);
        [x]=GSback(Ax,bx,H);
        disp('Ax=b的解向量x：')
        disp(x);
        break;
    else
        disp('输入不合法，请重新输入：');
    end
end
```

```
function [H,L,MARK]=isLegal(A,b)
    %判断输入是否合法
    %合法MARK返回1，否则返回0
    [HA,LA]=size(A);    %返回A矩阵的行和列，分别用HA和LA表示
```

笔记

```
    [Hb,Lb]=size(b);      %返回b向量的行（正常来说应该为1）和列，用Hb和Lb表示
    if(HA==LA)
        if(HA==Hb)
            if(Lb==1)
                if(det(A)~=0)
                    MARK=1;H=HA;L=LA;
                else
                    MARK=0;H=0;L=0;
                    disp('该方程组无解！');
                    return;
                end
            else
                MARK=0;H=0;L=0;
                disp('常数向量b输入不合法！');
                return;
            end
        else
            MARK=0;H=0;L=0;
            disp('系数矩阵A和常数向量b行数不相等！');
            return;
        end
    else
        MARK=0;H=0;L=0;
        disp('系数矩阵A不是方阵！');
        return;
    end
end
```

```
function [Ax,bx]=GSelimination(A,b,n)
    %消去过程
    %n表示系数矩阵(方阵)的规模
    %A表示系数矩阵
    %b表示常数矩阵
    for k=1:(n-1)      %要进行n-1次消去过程
        if(A(k,k)~=0)    %主元素不能为0,
            for i=(k+1):n  %循环行
```

```
                c=(-1*A(i,k))/A(k,k); %倍乘因子
                for j=1:n  %单行循环
                    A(i,j)=A(i,j)+c*A(k,j); %更新矩阵元素
                end
                b(i)=b(i)+c*b(k);    %更新常数向量
            end
        else
            disp('主元素出现零, 程序错误!');
            return;
        end
    end
    Ax=A;bx=b;
end
```

```
function [x]=GSback(Ax,bx,n)
    %回代过程
    %Ax为经过消去过程后得到的上三角矩阵
    %bx为经过消去过程后得到的常数向量
    %n表示系数矩阵的规模
    for i=n:-1:1
        if(Ax(i,i)~=0)
            x(i)=bx(i);
            for j=n:-1:i+1
                x(i)=x(i)-Ax(i,j)*x(j);
            end
            x(i)=x(i)/Ax(i,i);
        else
            disp('主元素出现零, 程序错误!');
            return;
        end
    end
    x=x';
end
```

3.5 层次分析法

在现实生活中，有很多地方都需要用到评价。即使是对同一个问题，我有多种选择方案，不同的选择方案之间需要评价；对于同一个方案，我有不同的对象，不同的对象之间需要评价；对于同一个目标，我有不同的评价指标，不同的评价指标需要综合衡量。这些都引出了一个大问题——如何评价 xxxx?

评价类模型指使用比较系统的、规范的方法对于多个指标、多个因素、多个维度、多个个体同时进行评价的方法。不仅仅是一种方法，是一系列方法的总称，即: 对多指标进行一系列有效方法的总称。综合评价是针对研究的对象，建立一个进行测评的指标体系, 利用一定的方法或模型，对搜集的资料进行分析，对被评价的事物作出定量化的总体判断。

一个过程并不是多个指标依次完成，而是利用些特殊的方法将多个指标同时完成。比方说，我在招聘的时候不是先拿第一学历卡一堆人，然后拿机试成绩再卡一堆，再拿面试表现再卡一堆，再根据实习期表现决定，而是把这四个成绩在最后统一折算加权赋分得到绩点排名。评价过程中，根据不同指标的重要性进行加权处理 (在一些评价过程中, 我们可以通过选取任意个评价指标, 通过些手段给予其不同的的权值，得到一个量化得分公式，然后通过这个公式计算不同考察对象的得分从而得到最优解)。评价的结果为根据综合分值大小的单位排序，并据此得到结论。

层次分析法是一种综合评价方法，它是美国运筹学家匹茨堡大学教授萨蒂于 20 世纪 70 年代初提出的一种评价策略。这种策略虽然带有一定主观性，但非常奏效，也是在社会科学研究中经常使用的一类方法。它是一种定性与定量相结合的、系统化的、层次化的分析方法，用于处理复杂的决策问题，比如从多种方案当中选择一种最优的。比如，我们出去玩，有武汉、桂林、广州三个地方可以选，但当我们不知道选什么的时候可以综合多个方面考虑构建层次分析模型。

我们从一个小例子说起，比方说，为了从三种空调：空调 A、空调 B、空调 C，中选购最合适的空调，我们采用层次分析法对我们的需求进行分析与评估，最终完成决策。那么，这个层次分析法的层次，又是什么意思呢?

评价一台空调我们从价格、噪声、功率、寿命四个角度考虑，我们不是先筛选掉价格便宜的，然后再看噪声小的，因为筛到最后可能最后的结果功率贼高贼烧电。我们构建一个分三层的层级模型图如图3.5所示：

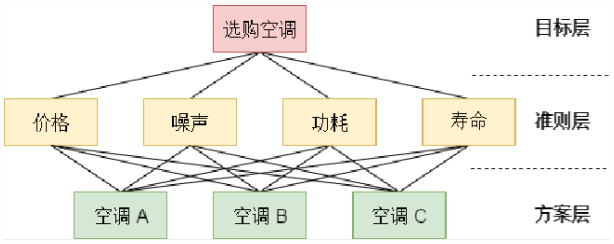


图 3.5: 买空调的层级模型图

层次分析法的模型图通常来讲就像图中描述的那样，分成了目标层、准则层和方案层三个层次。从目标层到准则层有一个 1 对 4 的比较，这是为了综合多个变量形成对目标的总评价；从准则层到方案层有一个 4 对 3 的比较，是为了分别比较不同空调在这 4 个指标上的相对好坏。如果我们有 6 个候选空调那么方案层就变成了 6 个，我们需要比对的也就是 6 对 4。

两个相邻的层次之间是需要构建一个成对比较矩阵的。比方说，我们在目标层和准则层之间就需要构建第一层比较矩阵，这个矩阵的大小是行列均为 4。矩阵的每一项表示因素 i 和因素 j 的相对重要程度。由于对角线上元素都是自己和自己做比较，所以对角线上元素为 1。另外，还有一条重要性质：

$$a_{ij}a_{ji} = 1 \tag{3.12}$$

关于这个矩阵的每一项取值多少，若因素 i 比因素 j 重要，为了描述重要程度，我们用 1-9 中间的整数描述。如果是因素 j 比因素 i 重要，“相对不重要程度”那就用 1-9 的倒数描述即可。这个比较矩阵每一项的确定具有较强的主观性，因为究竟二者重要程度是比较重要还是非常重要也是不同的人可能有不同的理解，但总体来讲是奏效的。一个成对比较矩阵的案例和取值标注如表3.5和3.5所示：

表 3.1: 一个成对比较矩阵的案例

变量	**价格**	**噪声**	**功耗**	**寿命**
价格	1	1/5	1/3	1
噪声	5	1	3	5
功耗	3	1/3	1	3
寿命	1	1/5	1/3	1

表 3.2: 成对比较的取值限制

取值	1	2	3	4	5
相对重要程度	同等重要	介于 1、3 之间	相对重要一些	介于 3、5 之间	相对比较重要
取值	6	7	8	9	
相对重要程度	介于 5、7 之间	相比明显重要	介于 7、9 之间	相比非常重要	

我们不仅需要构造目标层到准则层的一个矩阵，还需要构建准则层到方案层的 4 个矩阵，一共 5 个。这 5 个矩阵我们构造好了以后，层次分析法的步骤如下：

1. 选择指标，构建层次模型。
2. 对目标层到准则层之间和准则层到方案层之间构建比较矩阵。
3. 对每个比较矩阵计算 CR 值检验是否通过 CR 检验，如果没有通过检验需要调整比较矩阵。
4. 求出每个矩阵最大的特征值对应的归一化权重向量。

5. 根据不同矩阵的归一化权向量计算出不同方案的得分进行比较。

什么是 CR 检验？对比较矩阵进行特征值分解，最大的特征值为 λ，那么定义 CI:

$$CI = \frac{\lambda - n}{n - 1} \tag{3.13}$$

另外，经过大批量随机实验得到的统计规律，还可以得到一个 RI 统计值表3.5:

表 3.3: RI 统计值表格

n	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
RI	0	0	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49

得到 RI 和 CI 后，我们计算 CI 和 RI 的比值也就是 CR。查表计算可以得到这个矩阵的 CR 值为 0.0161。通常来讲当 CR 值超过 0.1 我们认为这个矩阵是不合理的，需要被修改被调整。这里由于没有超过这个阈值，我们认为这个比较矩阵通过了一致性检验。于是，剩下的过程我们便可以如法炮制，计算出准则层到方案层的七个矩阵了。

可以用如下代码进行一致性检验:

```
CI=(lambda-n)/(n-1);
RI=[0,0,0.58,0.9,1.12,1.24,1.32,1.41,1.45,1.49,1.52,1.54,1.56,1.58,1.59];
%判断是否通过一致性检验
CR=CI/RI(1,n);
if CR>=0.1
    fprintf('没有通过一致性检验\n');
else
    fprintf('通过一致性检验\n');
end
```

得到一致性检验结果后，我们还需要对最大特征值对应的特征向量进行归一化得到权重向量。为了保证特征值分解出来的特征值都还是实数，这里对每个特征值取实部做一个初步处理（虽然看起来是不是有些草率，但是本来成对比较矩阵的构造就是一个草率的主观过程，所以我在这里取实部做近似是完全没问题的）。归一化的方法为将特征向量除以该向量所有元素之和:

$$x_i = \frac{x_i}{\sum_{i=1}^{n} x_i} \tag{3.14}$$

类似地，我们可以根据文献和建模者自身的评估针对每个评价指标对评价对象构建评价矩阵。将一个成对比较矩阵的 AHP 过程封装为函数，完整函数如下:

```
function Q=AHP(A)
    [V,D]=eig(A);
    lambda=max(max(D));        %找到最大的特征值
    c1=find(D(1,:)==lambda);%找到最大的特征值位置
    feature_vector=V(:,c1); %最大特征值对应的特征向量
    [n,m]=size(A)
    CI=(lambda-n)/(n-1);
    RI=[0,0,0.58,0.9,1.12,1.24,1.32,1.41,1.45,1.49,1.52,1.54,1.56,1.58,1.59];
    %判断是否通过一致性检验
    CR=CI/RI(1,n);
    if CR>=0.1
        fprintf('没有通过一致性检验\n');
        Q=zeros(1,n);
    else
        fprintf('通过一致性检验\n');
        Q=feature_vector/sum(feature_vector);
        Q=real(Q);
    end
end
```

而最终的结果和完整流程是如何做的评价呢？请看下面的案例：

应用案例

不同地区水质的综合评价

某日从三条河流的基站处抽检水样，得到了水质的四项检测指标如表 5.1 所示。请根据提供数据对三条河流的水质进行评价。其中，DO 代表水中溶解氧含量，越大越好；CODMn 表示水中高锰酸盐指数，NH3-N 表示氨氮含量，这两项指标越小越好；PH 值没有量纲，在 6-9 区间内较为合适。

地点名称	pH*	DO	CODMn	NH3-N
四川攀枝花龙洞	7.94	9.47	1.63	0.077
重庆朱沱	8.15	9.00	1.4	0.417
湖北宜昌南津关	8.06	8.45	2.83	0.203

我们构建层次模型图，如图3.6所示：

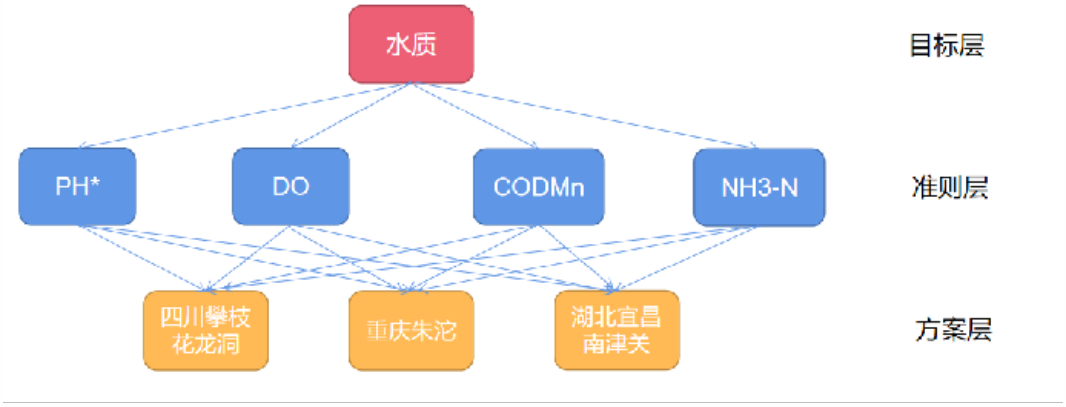


图 3.6: 水质评价的层次模型图

根据文献并查找资料，我们同样得到了三者的成对比较矩阵为 A=[1 1/5 1/3 1; 5 1 3 5; 3 1/3 1 3; 1 1/5 1/3 1];，若对矩阵 A 调用 AHP 函数，这里 n 取 4 的话很容易计算出 CI 的值是 0.0145。而除了 CI，还有一个 RI 值，在不同的 n 的取值下 RI 值也不同。得到 RI 和 CI 后，我们计算 CI 和 RI 的比值也就是 CR。查表计算可以得到这个矩阵的 CR 值为 0.0161。通常来讲当 CR 值超过 0.1 我们认为这个矩阵是不合理的，需要被修改被调整。这里由于没有超过这个阈值，我们认为这个比较矩阵通过了一致性检验。得到一致性检验结果后，我们还需要对最大特征值对应的特征向量进行归一化得到权重向量。归一化的方法为将特征向量除以该向量所有元素之和。

于是，剩下的过程我们便可以如法炮制，计算出准则层到方案层的 4 个矩阵了。接下来我需要比对的是三个地点在四个指标上分别的对应打分，需要得到四个矩阵。这个地方大家可以根据对三个对象的属性之间差异的主观感受去构造成对比较矩阵，因为不同人的感受可能有些差异，可以采取德尔菲法或者平均投票的原则。类似地，我们可以根据文献和建模者自身的评估针对每个评价指标对评价对象构建评价矩阵。计算出目标层到准则层的一个权重向量和四个准则层到方案层的权重向量以后，我们可以列出表格将权重向量进行排布：multirow

变量	**pH***	**DO**	**CODMn**	**NH3-N**	**得分**
	0.0955	0.5596	0.2495	0.0955	
四川攀枝花龙洞	0.4166	0.5396	0.2970	0.6370	0.4767
重庆朱沱	0.3275	0.2970	0.5396	0.1047	0.3421
湖北宜昌南津关	0.2599	0.1634	0.1634	0.2583	0.1817

我们将准则层到方案层得到的四个成对比较矩阵对应的权重向量排列为一个矩阵，矩阵的每一行表示对应的方案，每一列代表评价准则。将这一方案权重矩阵与目标层到准则层的权重向量进行数量积，得到的分数就是最终的评分。

由于四川攀枝花龙洞的得分为 0.4767，重庆朱沱的得分为 0.3421，湖北宜昌南津关的得分为 0.1817，最终我们得到的一个结论是：在评价过程中水中溶解氧含量与钴金属含量占评价体系比重最大，而四川攀枝花龙洞的水质虽然含钴元素比另外两个更高但由于溶解氧更多，NH3-N 的含量更小水体不显富营养化。就整体而言，四川攀枝花龙洞得分高于重庆朱沱和湖北宜昌南津关。

练习应用

现在如果将数据表格更改一下，我新增了两列：

地点名称	pH*	DO	CODMn	NH3-N	鱼类密度	垃圾密度
四川攀枝花龙洞	7.94	9.47	1.63	0.077	6.78	4.94
重庆朱沱	8.15	9.00	1.4	0.417	5.27	4.47
湖北宜昌南津关	8.06	8.45	2.83	0.203	2.50	7.03

我们认为鱼类密度越大水质越好，垃圾密度越小水质越好，请构建新的层次分析模型和评价矩阵对结果进行进一步分析。

3.6 熵权法

熵权法是一种客观赋权方法，基于信息论的理论基础，根据各指标的数据的分散程度，利用信息熵计算并修正得到各指标的熵权，较为客观。相对而言这种数据驱动的方法就回避了上面主观性因素造成的重复修正的影响。

在进行熵权法之前，首先要对指标进行正向化处理。因为指标多种多样，有的指标是越大越好比如成绩是越高越好；有的指标是越小越好比如房贷是越低越好；有的指标呢是在一个区间内最好高了低了都不行比如血压；而有的指标呢有一个最优值，偏差越大它也就越差。数据驱动的模型它不理解这些数字的意义，如果不做任何处理就直接 feed into model 那么很容易出问题，所以第一步就是指标正向化，也就是数据预处理的工作。

对于偏小型指标，它的正向化方式比较简单，可以取相反数；如果指标全部为正数也可以取其倒数。但对于区间型指标，它的规约方法遵循：

$$x_{new} = \begin{cases} 1 - \frac{a-x}{M}, x < a \\ 1, a \leq x \leq b \\ 1 - \frac{x-b}{M}, x > b \end{cases} \tag{3.15}$$

而对于中值型指标，它的正向化操作为：

$$x_{new} = 1 - \frac{|x - x_{best}|}{\max(|x - x_{best}|)} \tag{3.16}$$

在进行正向化完毕以后，所有的指标全部被变换为越大越好。而为了进一步消除量纲这些的影响，我们不妨再进行 min-max 规约来均衡数据的差异。

熵权法是一种为多指标评价提供依据的方法，可以为在此竞争力模型中的多种指标进行赋权。为了客观评价各类指标因素对竞争力模型的权重，本文选取熵权法这一具备多指标综合评价的模型，试图建立各因素与工程类毕业生就业情况的关系，具有更好的客观性。熵权法的主要计算步骤如下：

1. 构建 m 个事物 n 个评价指标的判断矩阵 R,(i=1,2,3,⋯,n；j=1,2,⋯,m)。

2. 将判断矩阵进行归一化处理，得到新的归一化判断矩阵 B。

$$B = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{3.17}$$

3. 熵权法可利用信息熵计算出各指标的权重，从而为多指标评价提供依据。根据信息论中对熵的定义，熵值 e 的计算如下所示：

$$e_j = -\frac{\sum_{i=1}^{n} p_{ij} \ln p_{ij}}{\ln n} \tag{3.18}$$

其中 p 为离散属性中每个类取值的占比。通过上式的熵值，可以评价不同指标的离散程度，一般情况下，信息熵越小，离散程度越大，因子对综合评价的权重就越大。

笔记

4. 计算权重系数，式中代表对于某一个属性 j，第 i 类占样本的比例。n 为属性 j 的取值数量。所以权重系数 w 定义为：

$$w_j = \frac{1 - e_j}{\sum_{i=1}^{m}(1 - e_i)} \tag{3.19}$$

一个熵权法的案例代码如下所示：

```
%指标正向化处理后数据为data1，下面是正向化方法
%区间型指标处理
%%越小越优型处理
index=[3,4,6];%越小越优指标位置
for i=1:length(index)
  data1(:,index(i))=max(data(:,index(i)))-data(:,index(i));
end
%标准化到0-1区间，但最小值为0时对数不存在
data2=(data1-min(data1))./(max(data1)-min(data1));
data2(find(data2==0))=0.001;
%得到信息熵
[m,n]=size(data2);
p=zeros(m,n);
for j=1:n
    p(:,j)=data2(:,j)/sum(data2(:,j));
end
for j=1:n
   E(j)=-1/log(m)*sum(p(:,j).*log(p(:,j)));
end
%计算权重
w=(1-E)/sum(1-E);
w=real(w);
%计算得分
s=data2*w';
Score=100*s/max(s);
disp('不同基站水质分别得分为：')
disp(Score)
```

得到的水质得分和不同指标的权重可以运行的出来。但是要注意一点，熵权法是数据驱动的计算方法，需要有大量数据支撑。同学们下去可以将其改写为一个函数试试看。

3.7 TOPSIS 分析法

TOPSIS 评价法是有限方案多目标决策分析中常用的一种科学方法，其基本思想为，对原始决策方案进行归一化，然后找出最优方案和最劣方案，对每一个决策计算其到最优方案和最劣方案的欧几里得距离，然后再计算相似度。若方案与最优方案相似度越高则越优先。

TOPSIS 方法是基于距离对对象进行评价的策略，相对来说也是比较客观的。这里我们除了介绍 TOPSIS 的方法还会介绍其改进模型。

TOPSIS 的想法就是，通过一定的计算，评估方案系统中任何一个方案距离理想最优解和最劣解的综合距离。如果一个方案距离理想最优解越近，距离最劣解越远，我们就有理由认为这个方案更好。那理想最优解和最劣解又是什么呢？很简单，理想最优解就是该理想最优方案的各指标值都取到系统中评价指标的最优值，最劣解就是该理想最劣方案的各指标值都取到系统中评价指标的最劣值。

在进行 TOPSIS 算法之前，我们同样需要对表格数据进行归一化。最常见的两种归一化操作就是 min-max 归一化和 z-score 归一化。但 TOPSIS 和熵权法有类似的问题：数据不同指标并不是都是越大越好，有的越小越好，有的是最值型，有的是区间范围型，那么对于这样一些指标同样需要进行正向化（参考熵权法当中的内容）。

TOPSIS 分析法的操作步骤如下：

1. 根据归一化得到的决策矩阵（这里我们选取 min-max 归一化）构造规范化权重矩阵 Z.

2. 确定正理想解 Z^+ 和负理想解 Z^-, 其中 J^+,J^- 分别表示效益型指标和成本型指标：

$$\begin{aligned} Z^+ &= \{\max_{j\in J^+} z_{ij}, \min_{j\in J^-} z_{ij}\} \\ Z^- &= \{\min_{j\in J^+} z_{ij}, \max_{j\in J^-} z_{ij}\} \end{aligned} \tag{3.20}$$

3. 计算各评价对象 i（i=1,2,3⋯，402）到正理想解和负理想解的欧几里得距离 D_i:

$$\begin{aligned} D_i^+ &= \sqrt{\sum_{j=1}^{n}(z_{ij}-Z_j^+)^2} \\ D_i^- &= \sqrt{\sum_{j=1}^{n}(z_{ij}-Z_j^-)^2} \end{aligned} \tag{3.21}$$

4. 计算各评价对象的相似度 W_i:

$$W_i = \frac{D_i^-}{D_i^- + D_i^+} \tag{3.22}$$

可以看到，相似度是与负理想解与两理想解距离之和的比值，若占比越大则说明离负理想解越远，越优先选择。

5. 根据 W_i 大小排序可得到结果。

笔记

TOPSIS 分析法的流程图如图3.7所示:

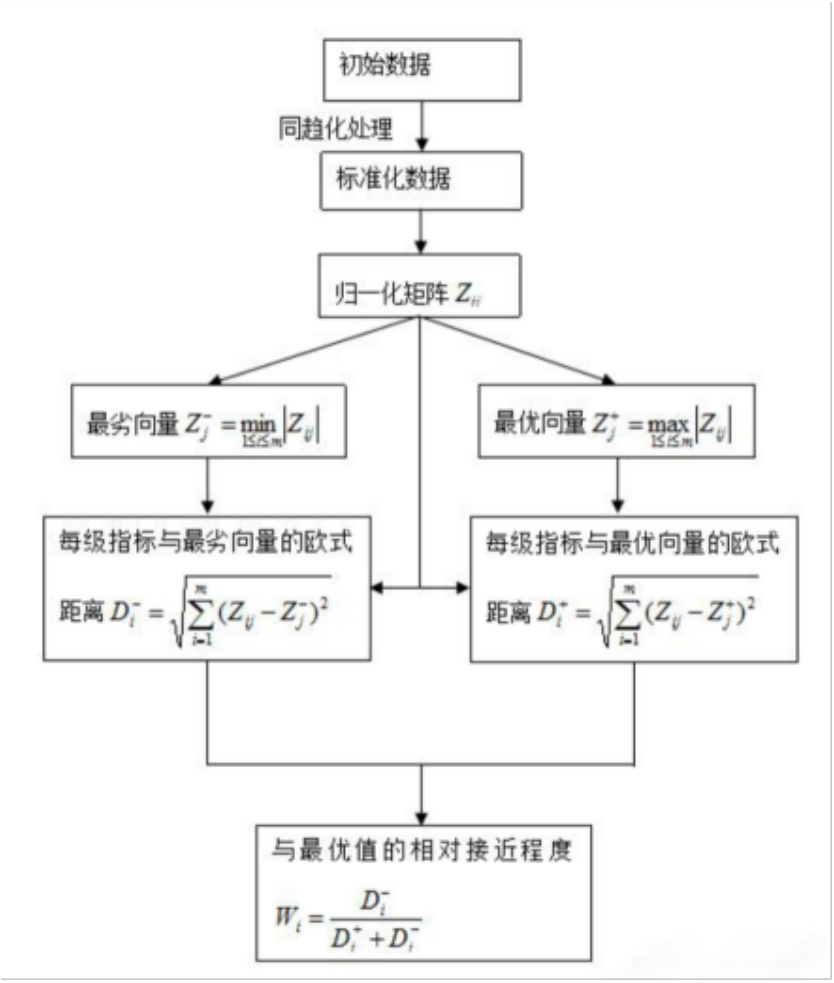


图 3.7: TOPSIS 分析的流程图

将上述过程用代码实现，封装为 TOPSIS.m 函数:

```
function  s=TOPSIS(R)
    %得到最大值和最小值距离
    r_max=max(R);  %每个指标的最大值
    r_min=min(R);  %每个指标的最小值
    d_z = sqrt(sum([(R -repmat(r_max,size(R,1),1)).^2 ],2)) ;  %d+向量
    d_f = sqrt(sum([(R -repmat(r_min,size(R,1),1)).^2 ],2)); %d-向量
    %sum(data,2)对行求和 ，sum(data）默认对列求和
    %得到得分
    s=d_f./(d_z+d_f );
    Score=100*s/max(s);
    for i=1:length(Score)
```

笔记

```
12          fprintf('第%d个投标者百分制评分为：%d\n',i,Score(i));
13      end
14  end
```

在经典 TOPSIS 方法中，计算欧几里得距离时是直接把不同指标的差的平方直接求和的。但事实上，不同指标在评价体系中所占比重可能不同，所以在计算距离的时候应当对不同的指标进行赋权。而这个权值则可通过熵权法或层次分析法获取。权重向量的构建是 TOPSIS 应用的核心，需要尽可能削弱其主观性。通常在解决 TOPSIS 问题的时候我们会有不少数据，所以权重可以通过熵权法这一数据驱动方法获得。于是，距离的计算公式应该变成：

$$\begin{aligned} D_i^+ &= \sqrt{\sum_{j=1}^{n} w_j (z_{ij} - Z_j^+)^2} \\ D_i^- &= \sqrt{\sum_{j=1}^{n} w_j (z_{ij} - Z_j^-)^2} \end{aligned} \tag{3.23}$$

练习应用

现在如果将数据表格更改一下，我收获了更多的水质数据：

地点名称	pH*	DO	CODMn	NH3-N	鱼类密度	垃圾密度
重庆朱沱	8.15	9	1.4	0.42	5.27	4.47
湖北宜昌南津关	8.06	8.45	2.83	0.2	2.5	7.03
湖南岳阳城陵矶	8.05	9.16	3.33	0.29	5.65	5.26
江西九江河西水厂	7.6	7.93	2.07	0.13	5.26	6.39
安徽安庆皖河口	7.39	7.12	2.23	0.2	6.21	7.5
江苏南京林山	7.74	7.29	1.77	0.06	6.44	3.93
四川乐山岷江大桥	7.38	6.51	3.63	0.41	3.17	5.75
四川宜宾凉姜沟	8.32	8.47	1.6	0.14	3.32	6.29
四川泸州沱江二桥	7.69	8.5	2.73	0.28	7.25	8.21
湖北丹江口胡家岭	8.15	9.88	2	0.08	7.22	3.82
湖南长沙新港	6.88	7.59	1.77	0.92	2.94	8.03
湖南岳阳岳阳楼	8	8.15	4.87	0.33	4.68	5.01
湖北武汉宗关	7.94	7.48	3.3	0.13	5.81	4.87
江西南昌滁槎	8.01	7.76	2.67	6.36	4.52	3.22
江西九江蛤蟆石	7.91	7.93	5.47	0.21	7.48	2.4
江苏扬州三江营	8.04	8.34	3.87	0.2	3.73	4.05

请根据前面两节熵权法和 TOPSIS 的代码，对 TOPSIS.m 进行修改，使其变成能够输入权重向量综合的 TOPSIS 改进模型，并利用这个改进函数获得这些地区水质的评分排序，并给予评价。

我给出一个评价语句的参考模板，供你们以后写比赛论文参考：

而通过评分，我们也可以对基站水质进行一个评价。江苏南京林山是最接近最优解的，按百分制计分获得了 *98* 分；其次湖北丹江口胡家岭、四川攀枝花龙洞等风景区的评分也不低，获得了 *78* 分；而到重庆朱沱、四川乐山岷江大桥、湖南长沙新港、江西南昌滁槎等工业区的评分则连 *20* 分都不到。这一结果充分反映，对水资源保护较好的风景名胜区，自然风光美好，生态发展良好的同时也带动了当地经济发展，尤其是以湿地生态为主的江苏南京林山、丹江口胡家岭、攀枝花龙洞等地值得参考和借鉴；而过于发展工业的地区不仅生态环境不够好，产业结构也较为单一，不利于当地绿色经济的发展，需要引起有关部门注意。值得学习的是江西九江河西水厂在水资源治理方面的措施，水厂对水资源的消耗和污染应该是相对比较严重的，但江西九江河西水厂的评分排名还相对比较靠前。

3.8 模糊综合评价法

这一节将介绍模糊综合评价法。身边有很多模糊的概念，例如：轻、重、热、冷、厚、薄、快、慢、大、小、高、低、长、短、贵、贱、强、弱、软、硬、美、丑、稀、稠、锐、钝、深、浅等。我说轻了，多少算轻了？怎么界定？我说一个人体重五十斤算太轻了，有没有考虑他才是个刚上小学的孩子？我说他 180 斤算重了，可是他身高两米二啊。那么模糊（Fuzzy）概念，就是从属于该概念到不属于该概念之间无明显分界线，外延不清楚。模糊概念导致模糊现象。在客观世界中，存在着大量的模糊现象，模糊数学就是用数学方法研究模糊现象。

模糊数学的产生：1965 年，美国伯克利加利福尼亚大学电机工程与计算机科学系教授、自动控制专家 L.A. Zadeh（扎德）发表了文章《模糊集》(Fuzzy Sets，Information and Control, 8, 338-353)，第一次成功滴运用精确的数学方法描述了模糊概念，从而宣告了模糊数学的诞生. 他所引进的模糊集（边界不明显的类）提供了一种分析复杂系统的新方法. 因发展模糊集理论的先驱性工作而获电气与电子工程师学会 (IEEE) 的教育勋章。而模糊综合评定法则是由汪培庄 (北京师范大学数学系) 提出了模糊数学的一种具体应用方法。

模糊综合评价方法是借助模糊数学的隶属度理论把定性评价转化为定量评价，即对受到多种因素制约的事物或对象做出一个总体的评价。模糊综合评价法是一种基于模糊数学的综合评价方法。该综合评价法根据模糊数学的隶属度理论把定性评价转化为定量评价，即用模糊数学对受到多种因素制约的事物或对象做出一个总体的评价。它具有结果清晰，系统性强的特点，能较好地解决模糊的、难以量化的问题，适合各种非确定性问题的解决。

其特点在于评判逐对象进行，对被评价对象有唯一的评价值，不受被评价对象所处对象集合的影响。综合评价的目的是要从对象集中选出优胜对象，因此，最后要将所有对象的评价结果进行排序。评判的意思是指按照给定的条件对事物的优劣、好坏进行评比、判别。综合的意思是指评判条件包含多个因素或多个指标。综合评判就是要对受多个因素影响的事物做出全面评价。

模糊综合评价法需要确定的要素主要是三个：评价指标，程度衡量和隶属度。评价指标又可以叫因素集，通常用 U 表示。比如我们如果想评价一个企业员工，可以从 {政治表现，工作能力，工作态度，工作成绩} 四个方面去做评价。而程度的衡量则是某一项指标的程度好坏，但不同于确定的数值，它往往是一个主观印象评价，或者说是没有一个明确的量化指标的。就比如我说这个员工工作态度认真，认真是有多认真？是每天自愿 996，还是自愿 007，还是仅仅把老板交待的任务完成就可以了？我们似乎看不出来它的一个明确的取值范围。但评分者无论是自评还是他评心里总归是有杆秤，所以这个方法才叫“模糊”。

常用的程度衡量方法就比如李克特五级量表，把人对一个事物的看法模糊为 {很差，差，中等，好，很好} 五个程度。而与前面一些评价方法类似，这些指标也是可以有权重的。当你没有采集到数据的时候，指标权重可以通过层次分析法确定；如果手头有数据，则可以使用熵权法确定。

而隶属度又是什么意思呢？实际上就是一个评分，一项指标对每个程度都分别进行评分。你可以理解为这项指标被预判为某个程度的概率大小，它是一个 0 到 1 之间的数。在实际的调查研

究中，更多的是用德尔菲法也就是向多名相关领域专家发放问卷征求他们的意见以获得模糊评价，或是采取文献分析法从现有文献中找到模糊隶属度。但在短期内如果想要取得模糊隶属度，还可以根据具体的指标去推算，方法和 TOPSIS 中讲到的指标正向化一样。例如，我们如果获得了某种溶液的酸碱度 PH 值，它可能是越偏酸性越好，也可能是越偏碱性越好，也可能是在某个酸碱范围内最好。此时的隶属度计算也有不同的方式。图3.8是不同隶属度的计算方式：

隶属度的计算方法

计算方法	偏小型	中间型	偏大型
梯形法	$u=\begin{cases}1, x<a\\ \frac{b-x}{b-a}, a\le x\le b\\ 0, x>b\end{cases}$	$u=\begin{cases}\frac{x-a}{b-a}, a\le x\le b\\ 1, b\le x\le c\\ \frac{d-x}{d-c}, c\le x\le d\\ 0, x<a, x>d\end{cases}$	$u=\begin{cases}0, x<a\\ \frac{x-a}{b-a}, a\le x\le b\\ 1, x>b\end{cases}$
多项式法	$u=\begin{cases}1, x<a\\ (\frac{b-x}{b-a})^k, a\le x\le b\\ 0, x>b\end{cases}$	$u=\begin{cases}(\frac{x-a}{b-a})^k, a\le x\le b\\ 1, b\le x\le c\\ (\frac{d-x}{d-c})^k, c\le x\le d\\ 0, x<a, x>d\end{cases}$	$u=\begin{cases}0, x<a\\ (\frac{x-a}{b-a})^k, a\le x\le b\\ 1, x>b\end{cases}$
指数法	$u=\begin{cases}1, x\le a\\ e^{-k(x-a)}, x>a\end{cases}$	$u=\begin{cases}e^{-k(a-x)}, x<a\\ 1, a\le x\le b\\ e^{-k(x-b)}, x>b\end{cases}$	$u=\begin{cases}1-e^{-k(x-a)}, x\le a\\ 0, x>a\end{cases}$
正态法	$u=\begin{cases}1, x\le a\\ e^{\frac{(x-a)^2}{\sigma^2}}, x>a\end{cases}$	$u=e^{\frac{(x-a)^2}{\sigma^2}}$	$u=\begin{cases}1-e^{\frac{(x-a)^2}{\sigma^2}}, x\le a\\ 0, x>a\end{cases}$

图 3.8: 不同隶属度的计算方式

对每个指标去计算隶属度可以得到一个隶属度矩阵，再将隶属度矩阵与指标权重相乘能够得到总的隶属度向量。最后，根据不同的隶属度给出评分即可获得对象的总体评分。例如，对于李克特五级量表 {很差，差，中等，好，很好}，分别给出 {100,80,60,30,0} 五个等级的评分，与最终隶属度向量数乘可以得到最终评分值。

设 $U=\{u_1,u_2,\cdots,u_m\}$ 为刻画被评价对象的 m 种评价因素 (评价指标). 其中:m 是评价因素的个数, 有具体的指标体系所决定。为便于权重分配和评议，可以按评价因素的属性将评价因素分成若干类, 把每一类都视为单一评价因素，并称之为第一级评价因素. 第一级评价因素可以设置下属的第二级评价因素, 第二级评价因素又可以设置下属的第三级评价因素, 依此类推，即 $U=U_1\cup U_2\cup\cdots\cup U_s$(有限不交并)，其中 $U_i=\{u_{i1},u_{i2},...,u_{im}\}$,$U_i\cap U_j=\Phi$, 任意 i~=j,i,j=1,2,...,s. 我们称 $\{U_i\}$ 是 U 的一个划分 (或剖分),Ui 称为类 (或块)。设 $V=\{v_1,v_2,\cdots,v_n\}$，是评价者对被评价对象可能做出的各种总的评价结果组成的评语等级的集合。其中：v_j 代表第 j 个评价结果, j=1,2,...,n. n 为总的评价结果数. 一般划分为 3~5 个等级。设 $A=(a_1,a_2,\cdots,a_m)$ 为权重 (权数) 分配模糊矢量，其中 a_i 表示第 i 个因素的权重, 要求 0<a_i,$\sum a_i=1$。A 反映了各因素的重要程度. 在进行模糊综合评价时，权重对最终的评价结果会产生很大的影响，不同的权重有时会得到完全不同的结论.

单独从一个因素出发进行评价，以确定评价对象对评价集合 V 的隶属程度，称为单因素模糊评价 (one-way evaluation)。在构造了等级模糊子集后，就要逐个对被评价对象从每个因素 ui 上

进行量化，也就是确定从单因素来看被评价对象对各等级模糊子集的隶属度，进而得到模糊关系矩阵：

$$R = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ r_{21} & r_{22} & \cdots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m1} & r_{m2} & \cdots & r_{mn} \end{pmatrix} \tag{3.24}$$

其中 r_{ij} 表示某个被评价对象从因素 u_i 来看对等级模糊子集 v_j 的隶属度。一个被评价对象在某个因素 u_i 方面的表现是通过模糊矢量 r_i 来刻画的，r_i 称为单因素评价矩阵，可以看作是因素集 U 和评价集 V 之间的一种模糊关系，即影响因素与评价对象之间的“合理关系”。

利用合适的模糊合成算子将模糊权矢量 A 与模糊关系矩阵 R 合成得到各被评价对象的模糊综合评价结果矢量 B。模糊综合评价的模型为：

$$B = A \circ R = (a_1, a_2, \cdots, a_m) \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ r_{21} & r_{22} & \cdots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m1} & r_{m2} & \cdots & r_{mn} \end{pmatrix} = (b_1, b_2, \cdots, b_n) \tag{3.25}$$

模糊综合评价的结果是被评价对象对各等级模糊子集的隶属度，它一般是一个模糊矢量，而不是一个点值，因而他能提供的信息比其他方法更丰富. 对多个评价对象比较并排序，就需要进一步处理，即计算每个评价对象的综合分值，按大小排序，按序择优. 将综合评价结果 B 转换为综合分值，于是可依其大小进行排序，从而挑选出最优者。实际中最常用的方法是最大隶属度原则，但在某些情况下使用会有些很勉强，损失信息很多，甚至得出不合理的评价结果。提出使用加权平均求隶属等级的方法，对于多个被评事物并可以依据其等级位置进行排序。

模糊数学法应用和研究综合评价方法时，一定要注意可行性和科学性，综合评价方法的思路和步骤。模糊数学法采用模糊数学模型，须先进行单项指标的评价，分别对各单项指标给予透当的权重，应用模糊矩阵复合运算的方法得出综合评价的结果。

模糊评价通过精确的数字手段处理模糊的评价对象，能对蕴藏信息呈现模糊性的资料作出比较科学、合理、贴近实际的量化评价；评价结果是一个矢量，而不是一个点值，包含的信息比较丰富，既可以比较准确的刻画被评价对象，又可以进一步加工，得到参考信息。但模糊综合法计算复杂，对指标权重矢量的确定主观性较强；当指标集 U 较大，即指标集个数凡较大时，在权矢量和为 1 的条件约束下，相对隶属度权系数往往偏小，权矢量与模糊矩阵 R 不匹配，结果会出现超模糊现象，分辨率很差，无法区分谁的隶属度更高，甚至造成评判失败，此时可用分层模糊评估法加以改进。

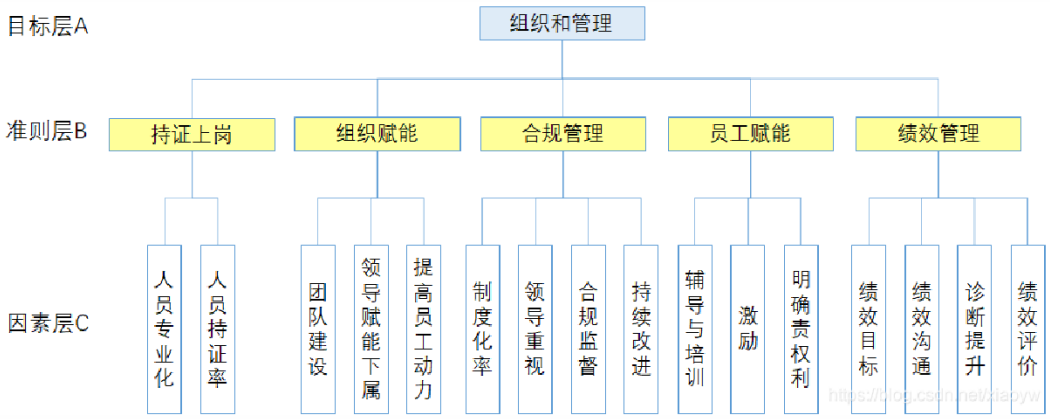
应用案例

企业管理能力的综合评价

假设以企业组织和管理水平评价为例，用模糊综合评价方法给出定量评价。这是专家（或其他统计方式）对评价打分表投票表决结果统计数据，简单的说就是对需要评价的因素（指标）给出主管或客观的“优、良、一般、较差、非常差”评价。企业管理专家通过多方讨论和查阅资料，确定了如下图所示的指标体系，包括五个一级变量和若干个二级变量，并且请了很多专家填写问卷统计每一项分布于李克特五级量表的频率分布如下图所示。

准则层		因素层			单因素模糊评价 V_j 评语集				
准则 U_i	权重	指标集因素 U_{ij}		权重 A_{ij}	优秀	良好	一般	较差	非常差
持证上岗 U_1	A1 权重向量	人员专业化	U_{11}	A11	0.1	0.4	0.3	0.2	0
		人员持证率	U_{12}	A12	0.5	0.3	0.2	0	0
组织赋能 U_2	A2	团队建设	U_{21}	A21	单指标 0.2	0.3	0.2	0.1	0.2
		领导赋能下属	U_{22}	A22	综合评价 0	0.2	0.3	0.3	0.2
		提高员工动力	U_{23}	A23	模糊矩阵 0.2	0.2	0.3	0.2	0.1
合规管理 U_3	A3	制度化率	U_{31}	A31	0.5	0.3	0.2	0	0
		领导重视	U_{32}	A32	0.1	0.3	0.3	0.2	0.1
		合规监督	U_{33}	A33	0.1	0.1	0.4	0.2	0.2
		持续改进	U_{34}	A34	0	0.1	0.3	0.3	0.3
员工赋能 U_4	A4	辅导与培训	U_{41}	A42	0.2	0.3	0.4	0.1	0
		激励	U_{42}	A42	0.1	0.3	0.5	0.1	0
		明确责权利	U_{43}	A43	0.2	0.5	0.3	0	0
绩效管理 U_5	A5	绩效目标	U_{51}	A51	0.3	0.3	0.3	0.1	0
		绩效沟通	U_{52}	A52	0.1	0.3	0.3	0.2	0.1
		诊断提升	U_{53}	A53	0.1	0.1	0.5	0.2	0.1
		绩效评价	U_{54}	A54	0.2	0.3	0.3	0.1	0.1

那么这个指标体系构建如下所示的树形图，我们会发现这就是一个层次分析模型，只不过不需要交叉比较，要比对的只有六个矩阵。



笔记

我们可以通过层次分析法的方式获得各个矩阵的权重系数，从而得到因素层这些指标的归一化权重系数。再通过归一化权重系数和模糊评价矩阵进行乘法可以得到该公司在李克特五级量表中的隶属度。如果对应分值分别为 {优秀：100，良好：80，一般：60，较差：40，非常差：20}，进行数量积可以得到最终评分。

现在，同学们请根据上文对模糊综合评价法的描述，参考我写的这份 Python 代码，写出模糊综合评价的 Baltamatica 代码：

```
import numpy as np
import pandas as pd
import warnings
'''
class AHP是我写的Python版本AHP，需要外部导入。
这里同学们改写为Baltamatica代码的时候使用上一节的AHP.m即可。
'''
def weight():
    # 准则重要性矩阵
    criteria = np.array([[1, 7, 5, 7, 5],
                         [1 / 7, 1, 2, 3, 3],
                         [1 / 5, 1 / 2, 1,  2,  3],
                         [1 / 7, 1 / 3, 1 / 2, 1, 3],
                         [1 / 5, 1 / 3, 1 / 3, 1 / 3, 1]])

    # 对每个准则，方案优劣排序
    b1 = np.array([[1, 5], [1/5, 1]])
    b2 = np.array([[1, 2, 5], [1/2, 1, 2], [1/5, 1/2, 1]])
    b3 = np.array([[1, 5, 6, 8], [1/5, 1 ,2, 7],
    [1/6, 1/2, 1 ,4],[1/8, 1/7, 1/4, 1]])
    b4 = np.array([[1, 3, 4], [1/3, 1, 1], [1/4, 1, 1]])
    b5 = np.array([[1, 4, 5, 5], [1/4, 1, 2, 4],
    [1/5 , 1/2, 1, 2], [1/5,1/4,1/2, 1]])
    b = [b1, b2, b3, b4, b5]
    a,c = AHP(criteria, b).run()
    return a,c
#模糊综合评价法(FCE)，输入准则权重、因素权重
def fuzzy_eval(criteria, eigen):
    #量化评语（优秀、    良好、    一般、    较差、    非常差）
    score = [1 0.8 0.6 0.4 0.2]
```

笔记

```
    df = pd.read_excel('FCE.xlsx')
    print('单因素模糊综合评价：{}\n'.format(df))
    #把单因素评价数据，拆解到5个准则中
    v1 = df.iloc[0:2,:].values
    v2 = df.iloc[2:5,:].values
    v3 = df.iloc[5:9,:].values
    v4 = df.iloc[9:12,:].values
    v5 = df.iloc[12:16,:].values
    vv = [v1,v2,v3,v4,v5]
    val = []
    num = len(eigen)
    for i in range(num):
        v = np.dot(np.array(eigen[i]),vv[i])
        print('准则{}，矩阵积为：{}'.format(i+1,v))
        val.append(v)
    # 目标层
    obj = np.dot(criteria, np.array(val))
    print('目标层模糊综合评价：{}\n'.format(obj))
    #综合评分
    eval = np.dot(np.array(obj),np.array(score).T)
    print('综合评价：{}'.format(eval*100))
if __name__ == '__main__':
    criteria, eigen=weight()
    fuzzy_eval(criteria, eigen)
```

3.9 主成分分析法

这一节将介绍主成分分析法。有时候问题提供的变量是过于精细化的变量，我们想对这些变量去抽象出更高一级的变量去描述数据，同时还能够保持数据的信息尽可能少地丢失，那么这个时候我们就需要用到主成分分析法。

主成分分析的主要目的是希望用较少的变量去解释原来资料中的大部分变异，将原始数据中许多相关性较高的变量转化成彼此相互独立或不相关的变量。通常是选出比原始变量个数少，能解释大部分资料中的变异的几个新变量（也就是主成分）。因此，我们可以知道主成分分析的一般目的是：(1) 数据的降维；(2) 主成分的解释。

主成分分析本质上就是一个线性代数运算。它包括以下步骤：

1. 数据的去中心化：对数据中每个属性减去这一列的均值。这样做的目的在于消除数据平均水平对它的影响。

2. 求协方差矩阵：注意这里需要除 (n-1)。

$$C = \frac{1}{n-1} X^T X \tag{3.26}$$

3. 对协方差矩阵进行特征值分解，虽然实际上在底层做 PCA 工作的时候使用奇异值分解会更为常见一些但特征值分解更容易理解。

$$C = Q \Sigma Q^T \tag{3.27}$$

4. 特征值排序，挑选更大的 k 个特征值，将特征向量组成矩阵 P。

5. 通过线性变换 $Y = PX$ 得到新的主成分，构造出主成分分析的矩阵。

那么根据以上过程，我们可以写出如下 pca 的代码：

```
function [coeff,D,latent,explain,trans]=pca(x)
    z=x-mean(x);%去中心化
    coeff=cov(z);%协方差
    [Q,D]=eig(coeff);%求出协方差矩阵的特征向量、特征根，但其实SVD分解用的更多
    Q=real(Q);
    D=real(D);%防止出现复数解
    d=diag(D);%取出特征根矩阵列向量（提取出每一主成分的贡献率）
    eig1=sort(d,'descend');%将贡献率按从大到小元素排列
    [~,I]=ismember(eig1,d);
    newQ=[];
    for ii = I
```

笔记

```
12            newQ=[newQ  Q(:,ii)];
13        end
14        %Q=fliplr(Q);%依照D重新排列特征向量
15        latent=z*newQ;%得到矩阵B;
16        explain=100*eig1/sum(eig1);%贡献率
17        trans=z\latent;
18    end
```

这个函数的返回值有五个:

1. coeff: X 的协方差矩阵。默认情况下，PCA 将数据居中并使用奇异值分解算法。
2. latent: 构造得到的主成分变量。
3. D: 返回每个主成分方差，即 X 的协方差矩阵的特征值，特征值从大到小进行排序，每一个数据是对应 score 里相应维的贡献率，并由大到小排列。
4. explain: 返回一个向量，其中包含每个主成分方差占总方差的百分比。(每个特征值占比，字面上即每个特征值对系统有多少解释，用百分比表示)。
5. trans: 原始变量和主成分之间的变换矩阵，一种对应关系。

练习应用

1. 请试着更改上面的代码，将特征值分解替换为奇异值分解。
2. 尝试探索 trans 矩阵的含义，并在 heatmap(trans) 的基础上进行解释。
3. 主成分分析会得到多个主成分和对应的特征值，想一想，如果我想综合多个主成分进行综合评价可以进行怎样的更改?

笔记

第四章　Baltamatica 的数据可视化

本章我们学习的内容是 Baltamatica 的数据可视化。数据可视化是认识数据、感知数据最直接的方式，在可视化过程当中要能够做到“以图表意”。Baltamatica 已经为我们提供了不少绘图函数，包括二维绘图和三维绘图，这些绘图函数是基于 OpenGL 开发的图形化库，和 Python 类似，能够导出为 png 文件、svg 文件、pdf 文件等，使用起来还是比较方便的。重点把握可视化的基本原则和方法，以及图像的调整和导出。

4.1　数据可视化的重要意义

古希腊的先哲毕达哥拉斯说“万物皆数”，我私以为这句话说的是没错的。因为纵观整个自然科学，物理学中充满了数据，化学中充满了数据，计算机科学本身也有数据，并且即使是社会学、经济学乃至新闻领域，都充满了量化研究的影子。但他们的数据就是简单的做表格吗？恕我直言，仅仅是做 Excel 表格这种高中生能干的活，大学为什么要开设大数据的专业和相关研究？

因为万物皆数，很多你想象不到的东西都可以转化为数据。

你或许无法想象，有一天唐诗宋词也会被称作“数据”吧？你或许无法想象，许嵩的一首歌也会被称作“数据”吧？你或许也无法想象，有关你女朋友的自拍也被称作为“数据”吧？但事实上，这些都是广义的“数据”。因为数据的目的，是为了描述与传递信息；而信息的载体是多种多样的，人类能够感知与认知的信息，计算机同样有办法处理。所以，真正的数据科学绝非我们传统印象当中这么狭隘，它应当是一门非常有广度、有深度的学科。在我的认知当中（如图4.1），数据科学应当包含多个组成部分：在我看来，数据科学真正应该包括五个主要的学习内容：

- 数据的获取和存储：包括爬虫、软件定义存储、硬件存储有关背景知识等。
- 数据的处理：包括分布式计算、并行计算、数据流等知识，以及 Hadoop、Spark 等大数据框架。
- 数据的分析：包括统计学、数据挖掘与机器学习、计算机视觉、自然语言处理等内容，重在挖掘数据中的模式与知识。
- 数据的管理：现代数据库系统及其架构等内容。
- 数据的应用：数据可视化、数据相关软件的开发、报表分析以及如何将数据挖掘得到的结果还原为实际问题的解决方案。

图 4.1: 数据科学的学习认知

在其中，数据可视化占到了一个很重要的位置。它是对于结果的总结和概括，对于方法的抽象和描述，对于问题的解释和说理。一幅好图有时候能够盖过那一整页的文字，其重要性不可小视。而 Baltamatica 的绘图则是一种科学、朴素取胜的典型数值科技类可视化风格。我们后面也将围绕它重点展开。

数据可视化的基本技法我在以前的课堂里面已经介绍过不少，我习惯于将其抽象为一种能力，因为可视化的图表在问题解释中实在是太重要了。在优化问题的建模中，求解出问题的答案就已经是一个困难的事了，但在大家都能求出答案的情况下，你想比别人更出色，就需要更好地可视化。图4.2是我以前比赛画的一幅图。

可视化之理我可以总结概括为如下几点：

- 图能达意：可视化出来的图首先要能够解决问题，能够清晰地被阅读，并且能够清晰表达作图者想表达的现象、结果等。没有这个一切免谈。
- 风格灵活：合理调节色彩、色调、线条，让图像更美观。这个审美问题怎么能够用文字描述呢？那是学设计和学艺术的人干的事，我这里只能说自己按照自己的审美调节就好。
- 工具灵活：不局限于 MATLAB，能够做出来好图就行。
- 除了画图还要排版：合理摆放图的位置，多用并列式、宫格式排版。

可视化的工具不需要局限，Baltamatica 的可视化其实就非常好看。很多 SCI 期刊是用 MATLAB 做的可视化；Python 的可视化也在科技学术界应用非常广泛。对于统计问题，R、SPSS 等工具也

图 4.2: 竞赛作品

提供了很漂亮的可视化方式。甚至 EXCEL、PPT、PS 等也可以用于可视化。还有一个比较高难度的可视化就是用前端方法，echart.js 等前端框架做的可视化也很漂亮。

数据可视化的本质是一种叙事的艺术。图片有时比文字更能生动直接地将数据的特性、数据的本质、数据人的思考展示在读者面前，但好的可视化一定也是讲究“信、达、雅”的。切尔西·卡尔森说，“对我来说，我总是在设计的最开始提出一个有趣的问题。无论你的数据看起来多么漂亮，如果其潜在问题不能激起你的兴趣，那么没有人愿意将精力花在解读你的作品上。”我们在做可视化的时候，首先一定要想清楚我这个图有没有画的必要，我画这个图目的在哪里？我应该选用怎样的形式表达才最为合适和直观？这个图反映了什么问题？我应该用怎样的工具去绘制它？可视化的工具不局限于 Baltamatica，其他工具比如 python、R、SPSS 甚至 PS 和 PPT 都可以用来进行可视化设计。古人说，“食无定法，烹无定味”，怎样进行可视化最为合适还是应该由读者自己来把握。至少——反映不了趋势变化的数据怎么能用折线图呢你说对吧。

4.2 基本图像的绘制

本节会给大家介绍一些基本平面图像的绘制，包括条形图、折线图、扇形图、散点图在 Baltamatica 中的绘制方法。

Baltamatica 的图像绘制都是基于 OpenGL 开发的图形工具包。最基本的绘制比如条形图 bar，折线图 plot，扇形图 pie 和散点图 scatter 等。条形图 bar 的绘制有两种方式，如果不指定横坐标，可以直接 bar(vec)；如果制定了坐标则可以 bar(x,vec)。例如运行下面的两句代码实现效果也是不同的：

```
1 bar([1 5 4 2])
2 bar([1 5 4 2],[10 21 40 33])
```

效果图如图4.3所示：

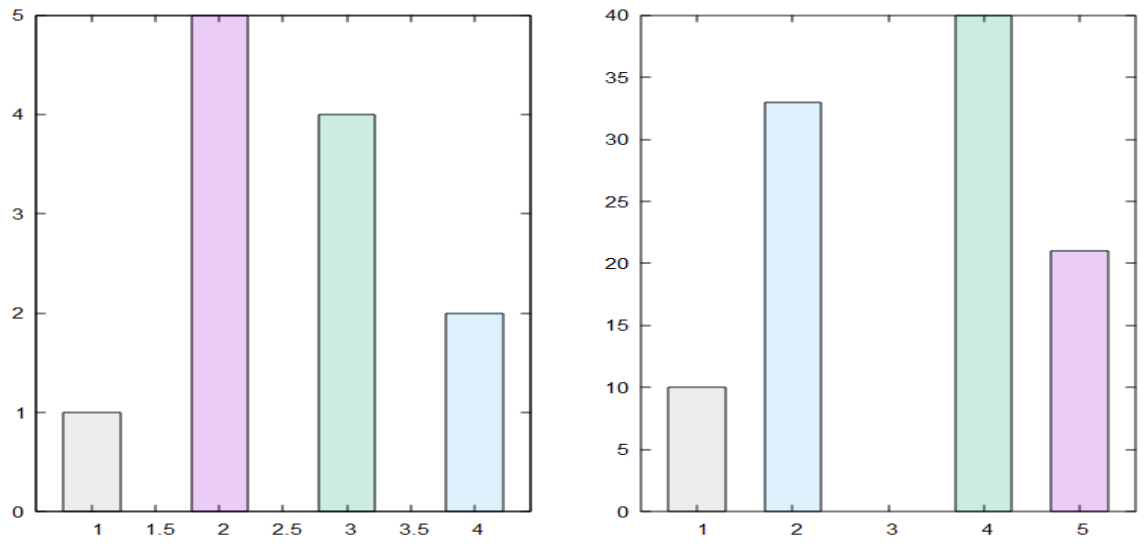


图 4.3: 条形图的效果图

可以看到右侧的条形图指定了坐标后会对对应数据进行排序,而非默认的横坐标为 1:length(vec)。

折线图的绘制则比较简单了，使用的是 plot 指令。和 bar 一样，如果不指定横坐标则指定为 1:n 的等距列表，但如果指定了横坐标则可以绘制曲线。折线图的线宽和线型是可以自己控制的，用’LineWidth’ 字段控制线宽（如果省略则默认为 1），用一些特殊的符号表示线型。例如，用’^’ 表示上三角，’v’ 表示下三角,’o’ 表示圆圈，’.’ 表示点，’x’ 表示叉叉等。通常一条横杠则为实线，两条横杠为虚线。如果想控制线条的颜色，常见的颜色包括’r’ 为红色，’b’ 为蓝色，’y’ 为黄色，’g’ 为绿色，’k’ 为黑色，’c’ 为青色，’m’ 为洋红色等。当然，也可以使用颜色的国际标识代码例如’# xxxxxx’ 或者 RGB 值表示。可以看下面的代码：

```
x=1:0.01:10*pi;
y=x.*sin(x);
plot(x,y,'rx-','LineWidth',1.5)
```

得到的结果如图4.4所示:

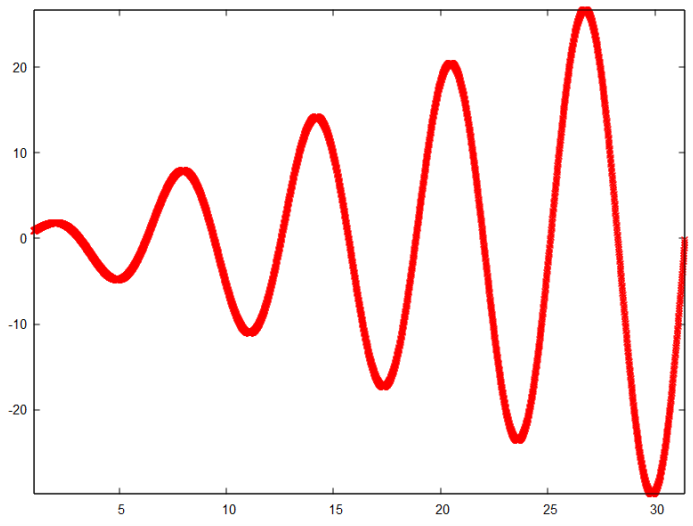


图 4.4: 折线图的效果图

扇形图的绘制更简单,传进去的就是一个向量 vec,函数自身会对 vec 做一次归一化操作,得到每个元素在整体求和中的一个占比然后根据占比绘图。比如如果我们输入 pie([0.20,0.25,0.14,0.05,0.36]) 得到的结果如图4.5所示:

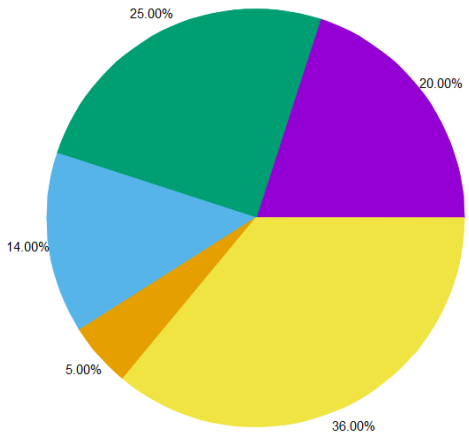


图 4.5: 扇形图的效果图

散点图的做法和折线其实调用模式是类似的，也要输入横纵坐标，也可以控制散点的大小，例如我们看下面这个例子:

```
1 scatter(x,y,20,'r','x','filled')
```

这个字段当中，x 和 y 是散点的横纵坐标，20 是如果用圆形绘制散点的散点大小，'r' 代表颜色，'x' 为形状，filled 表示内部填满不要空心。效果图如图4.6所示：

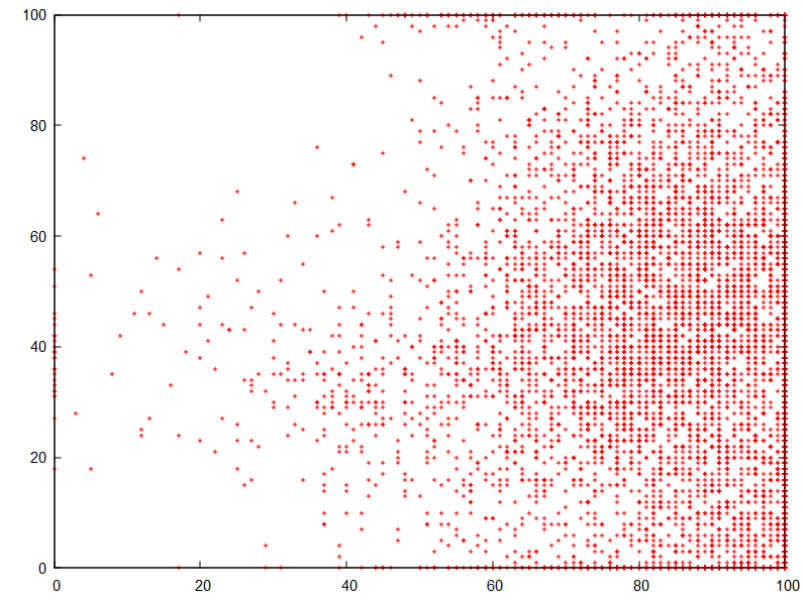


图 4.6: 散点图的效果图

下一讲我们继续看一些高级图形的绘制。目前 Baltamatica 的图形库做的还没有 MATLAB 那么灵活，包括图窗比例、自动缩放、子图刷新等问题还不够到位，还有待主创团队后续开发更多基于 OpenGL 的功能。

4.3 高级图像的绘制

本节会给大家介绍一些高级图像的绘制，包括一些三维曲线、曲面、散点，还有等高线、向量场和热力图在 Baltamatica 中的绘制方法。

首先，给各位先做一个翻版三维曲线和散点图的解说。三维的曲线使用函数叫做 plot3，除了新增一个纵坐标以外没有什么不同；而三维散点使用函数则是 scatter3，可以通过下面一个简单的例子学习三维曲线散点的绘制。我们绘制的是一条螺旋线：

$$\begin{cases} x = \cos\theta \\ y = \sin\theta \\ z = \theta \end{cases} \tag{4.1}$$

效果图如图4.7所示：

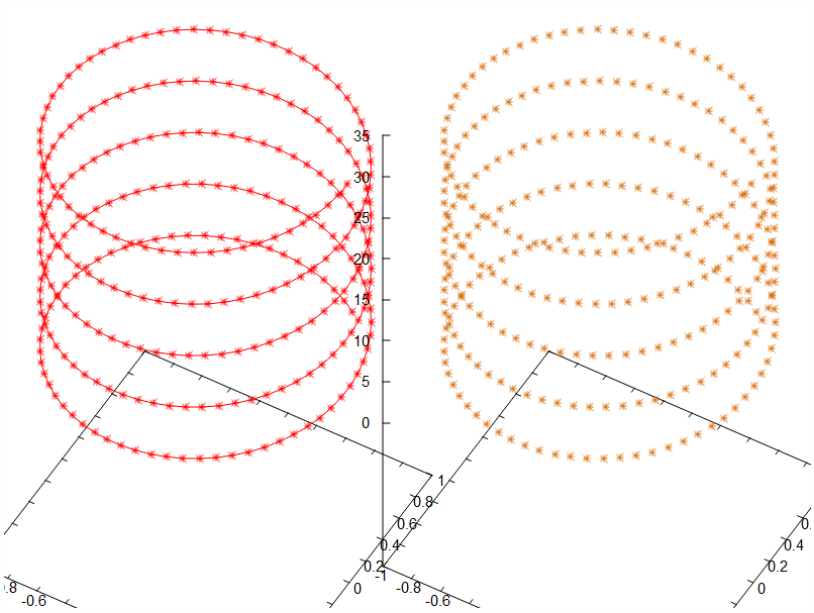


图 4.7: 三维曲线图和三维散点图的效果图

曲面的绘制则麻烦一些。首先，我们知道一个二元函数的值与 xy 两个坐标有关，所以 z 的形状是一个 m*n 的矩阵，x 的形状是长 m 的向量，y 的形状是长 n 的向量，所以先要根据 xy 生成一个网格数据 (meshgrid), 然后再调用 mesh 或者 meshc 函数。例如，如果我们想绘制曲面 $z = x^2 - y^2$ 的图像，可以写出如下代码：

```
x=-1:0.2:2;
y=-1:0.2:2;
[X,Y]=meshgrid(x,y);
z=X.^2-Y.^2;
mesh(X,Y,z)
```

曲面图的形状如图4.8所示。

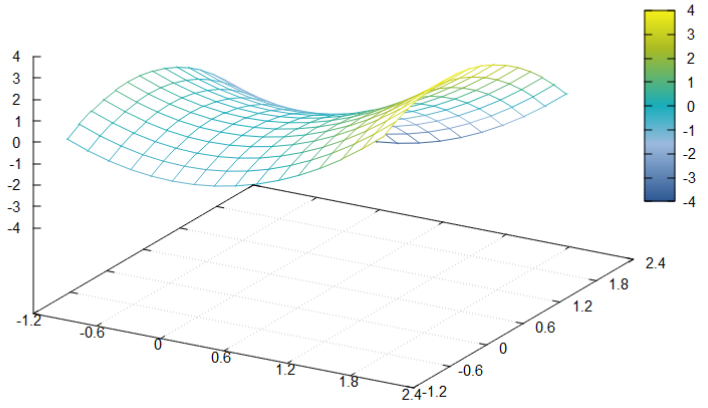


图 4.8: 三维曲面图的效果图

可以看到，这个双曲面是一个马鞍型曲面，绘制不同的 z 值对应颜色也不一样。右侧的调色板也表明了曲面的颜色与值的对应关系。

而从 z 轴自上而下俯视它，还可以绘制出一幅等高线图，使用 contour 指令如图4.9所示：

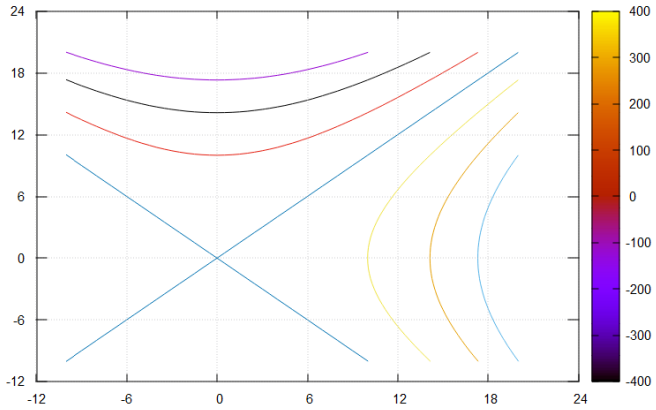


图 4.9: 等高线图的效果图

向量场图的例子用的不多，是 quiver 函数，我们举一个简单的例子看一下效果：

```
x=-1:0.2:2;
y=-1:0.2:2;
u=x.*sin(x);
v=cos(y)+y;
quiver(x,y,u,v);
```

效果图如图4.10所示：

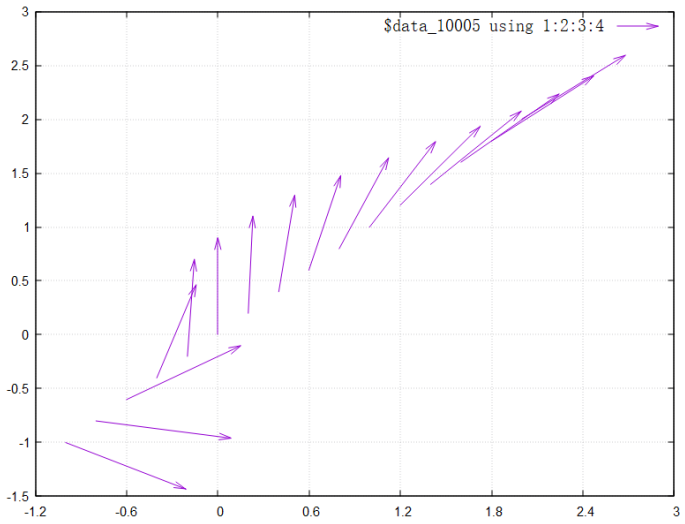


图 4.10: 向量场图的效果图

热力图的绘制是使用 heatmap 指令，用颜色来表示值的相对大小。比如，我们对上一章当中 PCA 代码所生成的 trans 矩阵可以调用热力图来绘制：

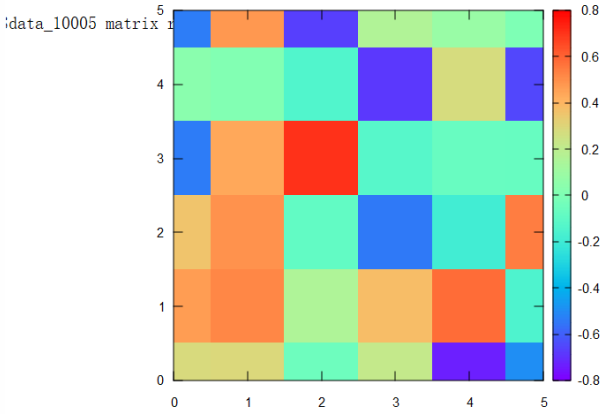


图 4.11: 热力图的效果图

4.4 图像的调整与导出

本节我们将学习如何调整图像的横纵轴、刻度、位置以及排版，还有图像如何导出为 png 图像等美化优化操作。

我们的图像绘制在弹窗中，以坐标图为例，如果想要设置横纵坐标轴的刻度值可以使用 xticks/yticks，而如果想修改每个刻度对应的标签则可以使用 xticklabels/yticklabels。比如，我们可以写下面的代码进行测试：

```
x = 1:10
plot(x)
xticks([0 5 8])
xticklabels(["x = 0","x = 5","x = 8"])
```

这样，刻度只会显示 0 5 8 三个数，并且刻度被更改为 x=0 x=5 x=8。

Baltamatica 里面还没有实现 grid 函数，不过我们可以用 xline/yline 添加辅助线（辅助线默认为虚线）。但是注意一个问题，添加辅助线的时候一定要记得每添加一条辅助线之前使用一句 hold on 声明。hold 指令是为了表示在图窗上直接添加而非刷新式使用。

创建一个图窗可以使用 figure 指令，可以不设置返回值，也可以把图窗命名为 f=figure()。如果认为图窗大小比例不太合适，可以使用 axis 调节。若 axis 当中传入参数为"square"，则绘图区域被重制为一个正方形；如果 axis 当中传入参数为一个列表，列表的返回值应该为 [xmin,xmax,ymin,ymax]。如果想调节图窗或者坐标轴的位置，可以调用 gca、gcf 和 gco 命令对轴、图窗和对象分别进行调节。

由于不同的曲线形状不同、表示的意义和对象不尽相同，可以使用 legend 命令标记不同形状的曲线是什么含义。若要标记坐标轴则可以使用 xlabel/ylabel 命令。如果有地方需要作另外的标记，比如向图窗中添加文本，那么可以使用 text 命令在对应坐标位置添加文本。

Baltamatica 的数据可视化在交互性和可动性问题上做的我觉得还是比较差。但是这个问题不比前面我提到的一些问题，这个可视化问题其实很难办。因为 MATLAB 能画的图种类太多了，而且还可以自动缩放、拖拽、修改属性、添加文本等等，并且三维图像可以做到全景式可视，可平动可旋转三维全景无压力，如果对配色不满意还可以加调色盘等等。但是这些东西用纯纯的 OpenGL 短时间内是很难做到的，这需要很强大的计算机图形学基础（尤其是三维数据的可视化）。但在工业问题上三维图可转可动性和可伸缩性是一个很重要的问题，所以即使任务量再大肯定还是得持续发力的。

第五章　基于 Baltamatica 的数值计算

本章我们学习的重点内容是 Baltamatica 的数值计算。在本科期间可能不少人学过一门课叫做数值分析，那么这里我们主要是讨论几个经典方法的 Baltamatica 代码实现。尤其是对于微分方程数值解和数值积分的问题上，Baltamatica 可能有着很好的性能。

5.1　符号解与数值解

为什么要引入数值解这个概念呢？这是因为现实情况数值解很多是求不出来的。符号变量与符号解的目的是在求微分、积分、极限等代数运算时最终得到的是准确的代数表达式。比如上一节的例子中即使是求定积分，它是一个确切的数值，但 matlab 也没有第一时间把它写成小数。而是将完整的精确形式 12*log(12) 写在工作区。在代数表达式中可能带有参数，但给出的是完整的公式。

数值变量则不同，我的目的是为了求一个满足精度要求的数值。在上一节的最后，我们利用 diff 函数求解了一个多项式的数值近似微分。当我们利用 diff 函数求解时，我们的结果描述的不是一个通项而是对每个 x 而言它的微分是多少的数值。如果利用数值积分去求，最终得到的也并非 12*log(12) 而是一个满足精度的小数。当然，这个精度也是根据工程上的需要来确定的，可能是四位，也可能是六位、八位等。

5.2 牛顿法求方程的根

牛顿法是一种用于函数极值和方程求根的算法，它被广泛用于零点的搜索。搜索方程根的方法有很多，这一节主要介绍区间搜索的分治算法和点搜索的牛顿法。

分治算法其实是一种比较容易理解的粗糙的算法。它基于这样一个事实：在一个区间 [a,b] 中，如果 f(a)f(b)<0 则这个区间里面一定有它的零点。这也是零点判定定理，但是话不能反过来说，如果 [a,b] 里面有 f(x) 的零点它的边界求乘积也不一定小于 0。但是有这么一条判定定理够本了。

我们希望在数值迭代的过程中逐渐缩小搜索范围，所以每次可以比较区间的中点 $f(\frac{b+a}{2})$。如果中点和最小边界乘积为负数，说明这个区间内绝对有零点；如果中点和最大边界乘积为负数，说明这个区间内也必然有零点。虽然还不知零点具体是多少，但是我们的搜索长度是一直变成上一轮的一半。逐步搜索，长度小到一定程度的时候求出来的解也就足够趋向于一个具体值了，误差也比较小了。

我们可以将这一分治策略的代码写在下面：

```
function [tp] = bizero(bound,epsilon,f,print_flag)
        minp=bound(1);
        maxp=bound(2);
        tp=(minp+maxp)/2;
        count=0;
        while abs(f(tp))>epsilon & count<100
                if f(minp)*f(tp)<0
                        maxp=tp;
                elseif f(maxp)*f(tp)<0
                        minp=tp;
                end
            tp=(minp+maxp)/2;
            count=count+1;
        end
        fprintf('二分迭代的近似解 x = %f\n',tp);
    fprintf('迭代次数count = %d\n',count);
end
```

我们可以用以下测试用例来对它的性能进行一次测试：

```
f=@(x) x.^3/3-x;
bizero([1,2],0.0001,f,1)
二分迭代的近似解 x = 1.732056
```

笔记

```
迭代次数 count = 12
```

另一种计算方法是牛顿的切线法。牛顿法原本是用于求解方程的零点搜索问题，但本质上函数的极值也可以抽象为导数的零点问题，所以牛顿法也可以被用于解极值点。牛顿法又有个名字叫切线法，它的原理如图5.1所示：

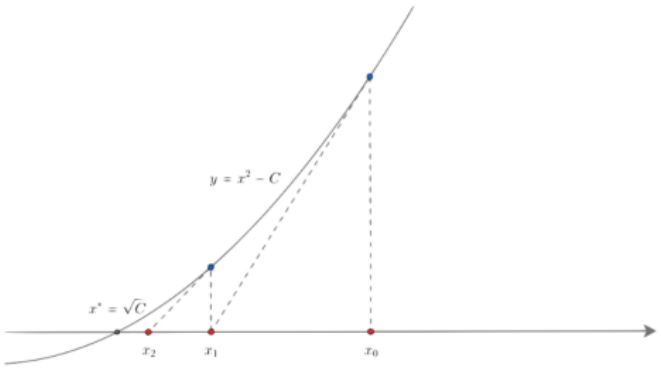


图 5.1: 切线法示意图

假如问题要求函数 $y = x^2 - C$ 的零点，从 x0 开始搜索。首先牛顿法会在 x=x0 处作 y 的切线，求切线与 x 轴的交点 x1，再接着在 x=x1 处作 y 的切线得到切线与 x 轴的交点 x2，如此不断迭代下去，最终这个值会收敛在离 x0 最近的零点处。

根据此，我们可以写出自己的零点求解函数：

```
function [x] = newton(x0,epsilon,f,print_flag)
df=@(x) (f(x+0.001)-f(x))/0.001;
count = 0;
e = 1;
while abs(e) > epsilon
        x1 = x0 - f(x0)/df(x0);%利用vpa函数控制精度
        e = x1 -x0;
        x0 = x1;
        count = count + 1;
        if count>100
            fprintf('牛顿迭代发散。\n');
            break
        end
        if print_flag
            fprintf('已迭代 %d 次，', count);
            fprintf('x为：%f ', x0);
```

```
            fprintf('误差为：%f\n', e);
        end
end
if print_flag
        fprintf('Newton迭代的近似解 x = %f\n',x1);
        fprintf('迭代次数count = %d\n',count);
end
x = x0;
end
```

在写好函数后用如下测试用例测试:

```
f= @(x) x.^3/ 3 - x;
delta = 0;                  % delta的绝对值：|delta|
step = 0.1;                 % delta递增的步长，初始步长为0.1
epsilon = 10^(-6);          % 允许的误差
x0 = 1;
x = newton(x0,epsilon,f,1);
fprintf('x0 = %f\n',x0);
fprintf('迭代结果x = %f\n',x);
```

可以看到结果:

```
已迭代 1 次,x为: 667.444519,误差为: 666.444519
已迭代 2 次,x为: 444.964345,误差为: -222.480174
已迭代 3 次,x为: 296.644728,误差为: -148.319617
已迭代 4 次,x为: 197.765733,误差为: -98.878995
已迭代 5 次,x为: 131.847526,误差为: -65.918206
已迭代 6 次,x为: 87.903741,误差为: -43.943785
已迭代 7 次,x为: 58.610412,误差为: -29.293329
已迭代 8 次,x为: 39.085319,误差为: -19.525093
已迭代 9 次,x为: 26.074280,误差为: -13.011039
已迭代 10 次,x为: 17.408792,误差为: -8.665488
已迭代 11 次,x为: 11.644615,误差为: -5.764177
已迭代 12 次,x为: 7.821084,误差为: -3.823531
已迭代 13 次,x为: 5.301040,误差为: -2.520044
已迭代 14 次,x为: 3.664749,误差为: -1.636291
已迭代 15 次,x为: 2.640016,误差为: -1.024733
已迭代 16 次,x为: 2.055094,误差为: -0.584922
```

```
已迭代 17 次,x为: 1.795263,误差为: -0.259831
已迭代 18 次,x为: 1.735288,误差为: -0.059975
已迭代 19 次,x为: 1.732063,误差为: -0.003226
已迭代 20 次,x为: 1.732051,误差为: -0.000012
已迭代 21 次,x为: 1.732051,误差为: -0.000000
Newton迭代的近似解 x = 1.732051
迭代次数count = 21
x0 = 1.000000
迭代结果x = 1.732051
```

精确度相当之高，相比二分搜索有了不小的提升。

笔记

5.3 欧拉法

欧拉法是一种以差分替代微分来求解微分方程的问题。前面提到过，数值计算微分方程的底层逻辑中一个最基本的思想就是“用差分代替微分”，而欧拉法和龙格库塔法也正是依据此所得到。

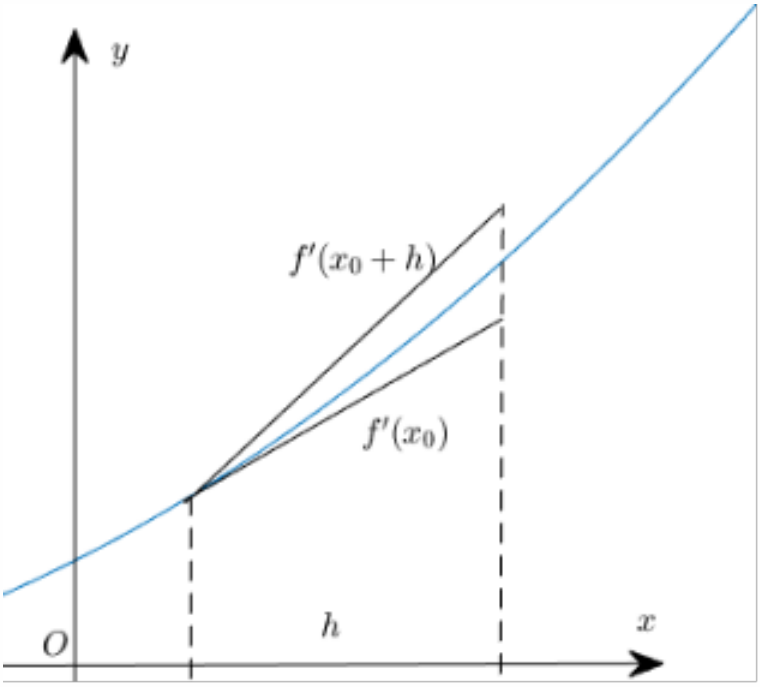


图 5.2: 欧拉法示意图

如图5.2，h 为一个小微元，数值微分计算的要求是希望能够每一次都可以“以直代曲”，在小的时间段上函数值的差分增量能够尽可能逼近微分增量。然而，从图中可以明显看到，如果以 x0 处的导数为斜率，差分增量式比微分增量小的；而以 (x0+h) 处的导数为斜率，差分增量比微分增量又大了。能不能取一个折中的方法呢?

我们可以看到正常情况下的前向欧拉法和后向欧拉法:

$$\begin{cases} f(x_{i+1}) = f(x_i) + \frac{f(x_i)}{x_i}h \\ f(x_{i+1}) = f(x_i) + \frac{f(x_{i+1})}{x_{i+1}}h \end{cases} \tag{5.1}$$

那很自然地，我们会想到用均值代替进行折中，这样误差就会小一些:

$$f(x_{i+1}) = f(x_i) + (\frac{f(x_{i+1})}{x_{i+1}}\frac{f(x_i)}{x_i})\frac{h}{2} \tag{5.2}$$

练习应用

案例：潮汐能 PTO 系统的动力学建模

随着经济和社会的发展，人类面临能源需求和环境污染的双重挑战，发展可再生能源产业已成为世界各国的共识。波浪能作为一种重要的海洋可再生能源，分布广泛，储量丰富，具有可观的应用前景。波浪能装置的能量转换效率是波浪能规模化利用关键问题之一。

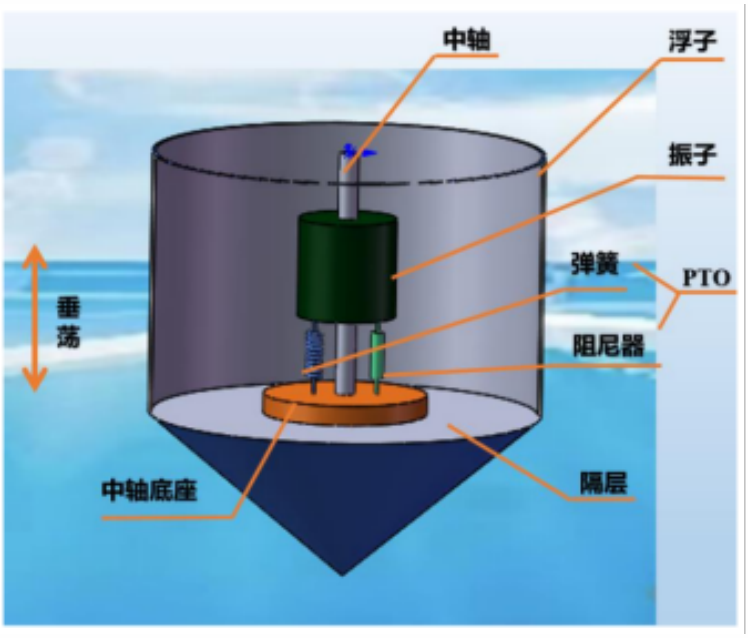


图 5.3: PTO 装置

图5.3为一种波浪能装置示意图，由浮子、振子、中轴以及能量输出系统（PTO，包括弹簧和阻尼器）构成，其中振子、中轴及 PTO 被密封在浮子内部；浮子由质量均匀分布的圆柱壳体和圆锥壳体组成；两壳体连接部分有一个隔层，作为安装中轴的支撑面；振子是穿在中轴上的圆柱体，通过 PTO 系统与中轴底座连接。在波浪的作用下，浮子运动并带动振子运动（参见附件 1 和附件 2），通过两者的相对运动驱动阻尼器做功，并将所做的功作为能量输出。考虑海水是无粘及无旋的，浮子在线性周期微幅波作用下会受到波浪激励力（矩）、附加惯性力（矩）、兴波阻尼力（矩）和静水恢复力（矩）。在分析下面问题时，忽略中轴、底座、隔层及 PTO 的质量和各种摩擦。

中轴底座固定于隔层的中心位置，弹簧和直线阻尼器一端固定在振子上，一端固定在中轴底座上，振子沿中轴做往复运动。直线阻尼器的阻尼力与浮子和振子的相对速度成正比，比例系数为直线阻尼器的阻尼系数。考虑浮子在波浪中只做垂荡运动，建立浮子与振子的运动模型。初始时刻浮子和振子平衡于静水中，波浪频率取 1.4005Hz，这里及以下出现的频率均指圆频率，角度均采用弧度制，分别对以下两种情况计算浮子和振子在波浪激励力 fcoswt（f 为波浪激励力振幅，w 为波浪频率）作用下前 40 个波浪周期内时间间隔为 0.2s 的垂荡位移和速度：(1) 直线阻尼器的

阻尼系数为 10000 N·s/m；(2) 直线阻尼器的阻尼系数与浮子和振子的相对速度的绝对值的幂成正比，其中比例系数取 10000，幂指数取 0.5。给出 10s、20s、40s、60s、100s 时，浮子与振子的垂荡位移和速度。

这个问题也是 2022 年全国大学生数学建模竞赛的题目。我们在开题之前先做一些假设，使我们的问题求解更加方便：

- 假设海平面是水平的（便于计算浮力）
- 不计浮子柱体及锥体的厚度（便于计算转动惯量）
- 振子为密度均匀的圆柱体，忽略中轴的厚度

给一些物理量做一些标记：浮子质量 M0，附加质量 M1，振子质量 m, 浮子柱高 H0，浮子锥高 H1，振子柱高 H，浮子半径 R，振子半径 r，弹簧原长 l, 弹簧刚度 k, 海水密度 ρ，重力加速度 g，波浪激励力振幅 f, 频率 w，垂荡兴波阻尼系数 k3，弹簧长度 x，线性阻尼器系数 β，旋转阻尼器系数 α。

对于浸没在水下的浮子而言，浮子的浮力取决于浸没在水下的体积：

$$V(h)=\begin{cases}\frac{\pi R^2h^3}{3H_1^2}, & 0\le h\le H_1\\ \frac{\pi R^2H_1}{3}+\pi R^2(h-H_1), & H_1\le h\le H_0+H_1\\ \frac{\pi R^2H_1}{3}+\pi R^2H_0, & h\ge H_0+H_1\end{cases} \tag{5.3}$$

我们首先看最初的平衡位置，在平衡状态下的浮子入水高度有关系：浮力 = 浮子 + 振子重量。平衡位置, 弹簧长度也满足胡克定律，就是弹簧的压缩量产生弹力与振子重力平衡。所以，我们可以列出初始状态下：

$$\begin{cases}\rho gV(h)=(m+M_0)g\\ -k(x-l)-mg=0\end{cases} \tag{5.4}$$

可以解得：h=2.8062, x=0.2020。

对于浮子，应用牛顿第二定律，浮子的合外力等于浮力、浮子重力、兴波阻尼力、波浪应激力、阻尼器和弹簧六个部分的力进行合力，也就是：

$$M\frac{dv}{dt}=\rho gV(h)-M_0g+f\cos(\omega t)-k_3v+k(x-l)+\beta(\mu-v) \tag{5.5}$$

并且 dh/dt=-v。而振子的牛顿第二定律就好办了，振子只有一个向下的重力，一个弹力，一个阻尼力，三者的关系：

$$m\frac{du}{dt}=-mg-k(x-l)-\beta(\mu-v) \tag{5.6}$$

并且 dx/dt=u-v。这里根据各个参数的初始条件，可以用改进欧拉法解常微分方程组。浮子的（离开平衡位置的位移）为 (-h+h(0)), 速度 v；振子的（离开平衡位置的位移）为 (-h+x+h(0)-x(0)), （对地）速度 u，那么浮子与振子之间的相对位移为 x(弹簧长度), 相对速度为 (u-v)。我们写出如下求解代码：

```
function  [h,x,v,mu,t]=solution(alpha,beta)   %alpha 幂指数  beta 阻尼系数
global M dt
m=2433;
rho=1025;g=9.8;l=0.5; k=80000;w=1.4005;k3=656.3616;
f=6250; R=1;
T=40*2*pi/w;            %40个周期的时长
n=floor(T/dt)+1;
t=0:dt:(n-1)*dt;          %时间序列
x=zeros(1,n);           %弹簧长度序列
h=x;                    %浮子入水高度序列
mu=x;                   %振子速度序列
v=x;                    %浮子速度序列

%求解常微分方程组
for  i=1:n-1
    tmp1=k*(x(i)-h(i))+beta*(abs(mu(i)-v(i))^alpha)*(mu(i)-v(i));
    tmp2=-rho*g*pi*R^2*h(i)+f*cos(w*t(i))-k3*v(i)+tmp1;
    tmp2=tmp2/M;
    vp=v(i)+tmp2*dt;
    hp=h(i)+v(i)*dt;
    mup=mu(i)-tmp1/m*dt;
    xp=x(i)+mu(i)*dt;

    tmp1=k*(xp-hp)+beta*(abs(mup-vp)^alpha)*(mup-vp);
    tmp2=-rho*g*pi*R^2*hp+f*cos(w*t(i))-k3*vp+tmp1;
    tmp2=tmp2/M;
    vc=v(i)+tmp2*dt;
    hc=h(i)+vp*dt;
    muc=mu(i)-tmp1/m*dt;
    xc=x(i)+mup*dt;

    x(i+1)=(xp+xc)/2;
    v(i+1)=(vp+vc)/2;
    mu(i+1)=(muc+mup)/2;
    h(i+1)=(hp+hc)/2;
end
```

```
37 end
```

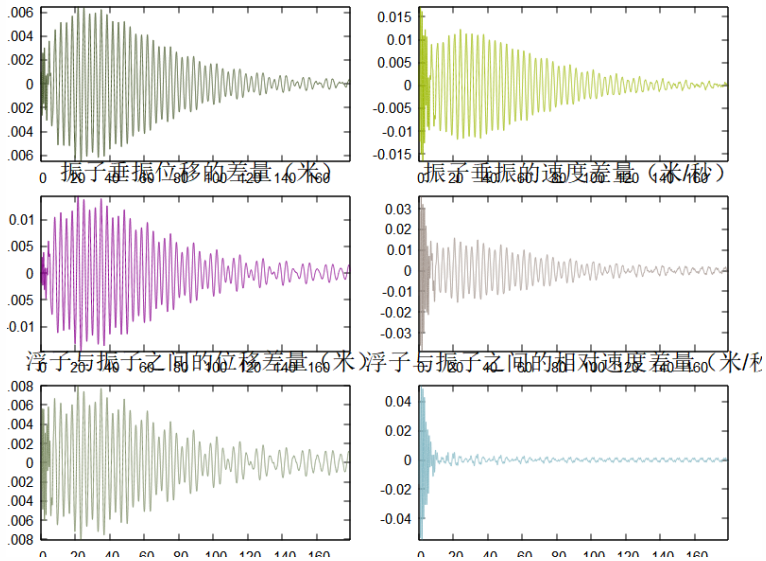
在外部导入参数并可视化:

```
1  global R H0 H1 M dt
2  R=1;H0=3;H1=0.8;M0=4886;M1=1335.535;m=2433;l=0.5;g=9.8;k=80000;beta=10000;rho=
3  % f w 垂荡激励力振幅及圆频率
4  % R M0 H0 H1  浮子半径、质量、柱高、锥高
5  % M1 附加质量
6  % m r,H 振子质量、半径、柱高
7  % k l 弹簧原长及刚度
8  % rho 海水密度
9  % g 重力加速度
10 % k3 垂荡兴波阻尼系数
11 % beta 线性阻尼器阻尼系数
12 % h 浮子离开平衡位置的位移
13 % x 弹簧长度
14 % v 浮子垂振速度
15 % mu 振子垂振速度
16 dt=0.01;                %时间间隔
17 M=M1+M0;
18 %线性阻尼器阻尼力与相对速度成正比
19 [h,x,v,mu,t]=solution(1,beta);
20
21 figure
22 title('阻尼器系数是常数的情形');
23 subplot(3,2,1);
24 plot(t,h);
25 title('浮子垂振相对于平衡位置的位移(米)');
26 subplot(3,2,2);
27 plot(t,v);
28 title('浮子垂振的速度(米/秒)');
29 subplot(3,2,3);
30 plot(t,x);
31 title('振子垂振相对于平衡位置的位移(米)');
32 subplot(3,2,4);
33 plot(t,mu);
```

```
34 title('振子垂振的速度（米/秒）');
35 subplot(3,2,5);
36 plot(t,x-h);
37 title('浮子与振子之间的位移（米）');
38 subplot(3,2,6);
39 plot(t,mu-v);
40 title('浮子与振子之间的相对速度（米/秒）');
41
42 %线性阻尼器阻尼系数与相对速度的平方根成正比
43 [h1,x1,v1,mu1,~]=solution(0.5,beta);
44
45
46 figure
47 title('阻尼器系数是与相对速度绝对值的平方根成正比常数的情形');
48 subplot(3,2,1);
49 plot(t,h);
50 title('浮子垂振相对于平衡位置的位移（米）');
51 subplot(3,2,2);
52 plot(t,v);
53 title('浮子垂振的速度（米/秒）');
54 subplot(3,2,3);
55 plot(t,x);
56 title('振子垂振相对于平衡位置的位移（米）');
57 subplot(3,2,4);
58 plot(t,mu);
59 title('振子垂振的速度（米/秒）');
60 subplot(3,2,5);
61 plot(t,x-h);
62 title('浮子与振子之间的位移（米）');
63 subplot(3,2,6);
64 plot(t,mu-v);
65 title('浮子与振子之间的相对速度（米/秒）');
66
67 %两种阻尼情形下位移与速度的差别（比较）
68 figure
69 subplot(3,2,1);
```

```
plot(t,h-h1);
title('浮子垂振位移差量（米）');
subplot(3,2,2);
plot(t,v-v1);
title('浮子垂振的速度的差量（米/秒）');
subplot(3,2,3);
plot(t,h+x-x(1)-h1-x1+x1(1));
title('振子垂振位移的差量（米）');
subplot(3,2,4);
plot(t,mu-mu1);
title('振子垂振的速度差量（米/秒）');
subplot(3,2,5);
plot(t,x-x1);
title('浮子与振子之间的位移差量（米）');
subplot(3,2,6);
plot(t,mu-v-mu1+v1);
title('浮子与振子之间的相对速度差量（米/秒）');
```

最终解得的速度图像和位移图像如下图所示：



而表格如下表所示：

5* 常数	时间	10 秒	20 秒	40 秒	60 秒	100 秒
	浮子位移	-0.2076	-0.6163	0.2648	-0.3272	-0.0857
	浮子速度	-0.66701	-0.2787	0.2815	-0.5086	-0.6172
	振子位移	-0.2372	-0.6672	0.276	-0.3505	-0.0914
	振子速度	-0.7134	-0.299	0.2969	-0.5431	-0.6579
5* 相对速度绝对值的平方根	时间	10 秒	20 秒	40 秒	60 秒	100 秒
	浮子位移	-0.2026	-0.6103	-0.2699	-0.3235	-0.0843
	浮子速度	-0.667	-0.276	0.2854	-0.5052	-0.6152
	振子位移	-0.231	-0.66	-0.2815	-0.3455	-0.0891
	振子速度	-0.716	-0.3006	0.3012	-0.5407	-0.656

5.4 龙格库塔法

在上一节的基础上，我们考虑能否多一些迭代来得到更加精确的数值模拟呢？于是，龙格库塔法便应运而生了。对前后向导数折中的第一种想法就是把 x0 处的切线斜率和 (x0+h) 处的切线斜率做个平均，以此平均斜率作直线的差分增量就和微分增量能够比较接近了，这是改进欧拉的思想。

第二种想法是以 x0 和 (x0+h) 内部取不同的点的斜率进行迭代平均，这是龙格库塔的思想。经典四阶龙格库塔法的迭代斜率如下：

$$\begin{cases} K_1 = f(x_i, y_i) \\ K_2 = f(x_i + \frac{h}{2}, y_i + \frac{K_1}{2}h) \\ K_3 = f(x_i + \frac{h}{2}, y_i + \frac{K_2}{2}h) \\ K_4 = f(x_i + h, y_i + K_3 h) \\ y_{i+1} = y_i + h(K_1 + 2K_2 + 2K_3 + K_4) \end{cases} \tag{5.7}$$

我们可以编写自己的函数，如下所示：

```
function [x,y]=runge_kutta(ufunc,y0,h,a,b)
n=floor((b-a)/h);          %步数
x(1)=a;                    %时间起点
y(:,1)=y0;                 %赋初值，可以是向量，但是要注意维数
for i=1:n                  %龙格库塔方法进行数值求解
    x(i+1)=x(i)+h;
    k1=ufunc(x(i),y(:,i));
    k2=ufunc(x(i)+h/2,y(:,i)+h*k1/2);
    k3=ufunc(x(i)+h/2,y(:,i)+h*k2/2);
    k4=ufunc(x(i)+h,y(:,i)+h*k3);
```

笔记

```
    y(:,i+1)=y(:,i)+h*(k1+2*k2+2*k3+k4)/6;
end
end
```

但其实现在我们倒也犯不着了，因为新一代的 Baltamatica 已经集成了 ode45 函数，它的底层就是四阶龙格库塔算法。调用格式形如 sol = ode45(odefun,tspan,y0,options);

练习应用

案例：新冠爆发分阶段的 SEIR 模型

自 2020 年来新冠病毒的大流行打破了我们原有的平静生活。在新冠大流行期间，奋斗在一线的科研攻关人员不止有医疗机构的医生、研究员和各互联网大厂开发健康码、行程码的程序员们，高校中的数学领域研究学者和学生们同样发挥了很大作用。英国的帝国理工、中国的西安交大、武汉大学这些学校数学方面的教师、学生们也对新冠病毒的传播进行了数学建模。

模型需要对病毒传播初期、中期和后期的情况有充分的预测，以便于进行疫情防控。新冠病毒传播速度快，感染时间短，能够迅速传染而且变异速度快很难做出疫苗。所以在这个问题当中要考虑易感人群，无症状感染者，感染者三者之间的平衡关系。注意，这个问题中只要把传染和恢复的速度与人员流向搞清楚很容易写代码。

我们可以思考一下，假如我们就考虑人群被分为绿码、黄码和红码，它有这样的一些动力学特性：

传播速度快，规模大，但是可以连续变化。

所有群体（绿码、黄码、红码）在人群中比例总和 =1。

绿码少了多少黄码就多了多少，黄码少了多少红码就会多多少，红码康复多少绿码就会恢复多少，即有出就有进，总体平衡。

SI 模型是最简单的传播模型，把人群分为易感者（S 类）和患病者（I 类）两类，通过 SI 模型可以预测传染病高潮的到来。易感者与患病者有效接触即被感染，变为患病者，无潜伏期、无治愈情况、无免疫力。以一天作为模型的最小时间单元。总人数为 N，不考虑人口的出生与死亡，迁入与迁出，此总人数不变。t 时刻两类人群占总人数的比率分别记为 s(t)、i(t)，两类人群的数量为 S(t)、I(t)。初始时刻 t=0 时，各类人数量所占初始比率为 s0、i0。每个患病者每天有效接触的易感者的平均人数是 λ，即日接触数。

聪明的同学可能已经发现了，没错这就是一个逻辑斯蒂模型。如果你去试着编程解一个逻辑斯蒂模型，它最后必然是一条 S 型曲线，也就是所有的人都会被感染掉。这个结果很显然很扯淡，说明我们肯定有没考虑到的东西。在 SI 模型基础上考虑病愈免疫的康复者（R 类）就得到 SIR 模型，对应疾病被治愈或死亡的状态，感染人群以单位时间传染概率由感染状态转移至移除状态。

用 SIR 模型进行实验，发现走向已经非常接近于一个实际的传染病模拟。但我们认为对于新冠这样的存在潜伏期的疾病，还需要考虑一类人群——高危暴露者，也可以理解为疑似病例。现在假设易感人群变为暴露者的暴露速度为 λ，疑似病例被确诊感染的感染速度为 δ，感染者被治愈的治愈速度为 μ，此时我们得到一种最接近实际情况的模型，也就是 SEIR 模型。如图5.4所示：

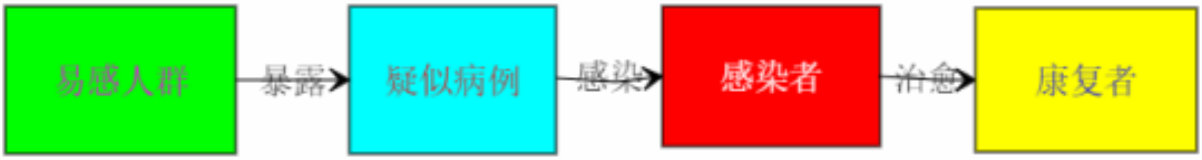


图 5.4: SEIR 模型的示意图

这一模型也可以加入不同的路径图。例如，康复者仍有一定概率变为易感人群因为新冠病毒可能以一定概率发生突变；易感人群也可能通过接种疫苗的方式直接获得免疫变为康复者等。

$$\begin{aligned} N\frac{ds}{dt} &= -N\lambda si \\ N\frac{de}{dt} &= N\lambda si - N\delta e \\ N\frac{di}{dt} &= N\delta e - N\mu i \\ N\frac{dr}{dt} &= N\mu i \\ s(t) + e(t) + i(t) + r(t) &= 1 \end{aligned} \tag{5.8}$$

仿真代码如下：

```
figure;
[t,h] = ode45(@SEIR,[0  300],[0.01  0.98  0.01  0]);  %[初始感染人口占比  初始健康
plot(t,h(:,1),'r');
hold on;
plot(t,h(:,2),'b');
plot(t,h(:,3),'m');
plot(t,h(:,4),'g');
legend('感染人口占比I','健康人口占比S','潜伏人口占比E','治愈人口占比R');
title('SEIR模型')
grid on;

function dy=SEIR(t,x)
beta = 0.1;          %感染率
gamma1 = 0.05;       %潜伏期治愈率
gamma2 = 0.02;       %患者治愈率
alpha = 0.5;         %潜伏期转阳率
```

```
17 dy=[alpha*x(3) - gamma2*x(1);
18       -beta*x(1)*x(2);
19       beta*x(1)*x(2) - (alpha+gamma1)*x(3);
20       gamma1*x(3)+gamma2*x(1)];
21 end
```

它的仿真结果如图所示:

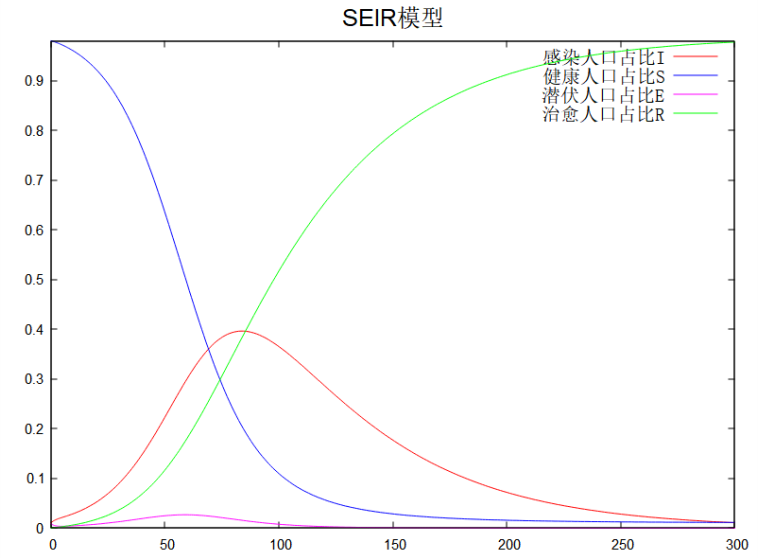


图 5.5: SEIR 模型的仿真

5.5 梯形法求数值积分

梯形法是求解数值积分的一种很方便的方法。我们回忆一下高中的时候学习的定积分如何定义？它其实就是将曲边梯形用若干个长方形面积累加起来：

$$\int_a^b f(x)dx = \lim_{n\to+\infty}\sum_{i=1}^{n} f(a+i\frac{i}{n}(b-a))\frac{b-a}{n} \tag{5.9}$$

但是这种积分的数值分解法只留一个端点必将带来误差。如果曲边梯形用长边代替，面积会偏大；用短边代替，面积就会减小。能不能取一个折中的方案呢？其实参考改进欧拉法就可以得到一个思路：

$$\int_a^b f(x)dx = \lim_{n\to+\infty}\sum_{i=1}^{n} \frac{1}{2}(f(a+\frac{i}{n}(b-a)) + f(a+\frac{i+1}{n}(b-a)))\frac{b-a}{n} \tag{5.10}$$

这就是计算数值积分的梯形公式。我们如果用 Baltamatica 实现如下：

```
function [s] = trap(f,n,a,b)
% 梯形公式：\int L dx= [f(a)+f(b)](b-a)/2
% n为分为n段
    h = (b-a)/n;
    s = 0;
    trape = @(x) (f(x-h) + f(x))*h/2;
    for i = a+h:h:b
        s = s + trape(i);
    end
end
```

前面在讲矩阵的时候就提到过，使用循环会比使用向量化方法更浪费时间。所以我们希望能把 for 循环改写成向量运算，所以我们可以使用如下方法改进：

```
function [s] = trap(f,n,a,b)
    h = (b-a)/n;
    s = h/2 * ( f(a) + f(b) + 2*sum(f(a+h:h:b-h)));
end
```

对梯形公式有更加精确的模拟，即辛普森公式：

$$I_2(f) = \frac{b-a}{6}[f(a) + 4f(\frac{b+a}{2}) + f(b)] \tag{5.11}$$

如果用代码实现可以用下面的代码实现：

```
function [s] = simpson(f,n,a,b)
%Simpson公式：\int L dx= (b-a)/6[f(a) + 4f((a+b)/2) + f(b)]
%n为分为n段
    h = (b-a)/n;
    s = 0;
    sim = @(x) 1/6*(f(x-h)+4*f(x-h/2) +f(x)   );
    for i = a+h:h:b
        s = s + sim(i);
    end
end
```

5.6 偏微分方程的边界条件与数值模拟

本节介绍两个经典的偏微分方程数值模拟的案例。偏微分方程的基本形式类似于常微分方程，但不同的是微分方程中出现的是多元函数和多元函数的偏微分。有时也会以方程组的形式出现。关于偏微分方程的研究是基础数学和计算数学领域比较热门的话题，这里我们不作太多解释，以几个典型的偏微分方程案例带各位熟悉一下偏微分方程。

我们常研究的就是二元函数的二阶偏微分方程，其基本形式为：

$$A\frac{\partial^2 f}{\partial x^2}+2B\frac{\partial^2 f}{\partial x\partial y}+C\frac{\partial^2 f}{\partial y^2}+D\frac{\partial f}{\partial x}+E\frac{\partial f}{\partial y}+Ff=0 \tag{5.12}$$

如果 A、B、C 三个常系数不全为 0，定义判别式 $\Delta=B^2-AC$，当判别式大于 0 称其为双曲线式方程；若判别式等于 0，则称其为抛物线式方程；若判别式小于 0，则称其为椭圆式方程。有关于这几类方程的基本性质与边界条件等内容请感兴趣的读者自行参考偏微分方程领域的书籍，我们的主要目光更多的聚焦在它的应用上。

偏微分方程在物理中尤其是工程物理中应用颇广。例如，广为人知的三维热传导方程：

$$\frac{\partial T}{\partial t}=\alpha\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}+\frac{\partial^2 T}{\partial z^2}\right) \tag{5.13}$$

这就是一个典型的偏微分方程。温度函数 T(x,y,z,t) 在三维空间坐标 (x,y,z) 处的温度 T 随时间 t 的变化与坐标位置的演变关系如上所示。又如，大名鼎鼎的“韦神”韦东奕在流体流速方程中做出的巨大突破——有关纳维·斯托克斯方程的研究：

$$\rho\left(\frac{\partial v}{\partial t}+v\cdot\nabla v\right)=-\nabla p+\mu\nabla^2 v+f \tag{5.14}$$

这一方程更多的是描述流体流速与流体密度、压力、外阻力之间的关系，在机械工程、能源工程等制造领域有着重要应用。

电磁学中也有著名的麦克斯韦方程组，它表示了电场和磁场之间的一个变换关系以及电磁波的生成。参考下面的式子：

$$\begin{cases}\nabla\cdot E=\frac{\rho}{\varepsilon_0}\\ \nabla\times E=-\frac{\partial B}{\partial t}\\ \nabla\cdot B=0\\ \nabla\times B=\frac{j}{c^2\varepsilon_0}+\frac{\partial E}{\partial t}\end{cases} \tag{5.15}$$

其他著名的物理学中的常/偏微分方程（组）有很多，例如麦克斯韦方程组、洛伦兹系统等，这些都是基础数学在物理学和工程中获得应用的典型案例。实际上偏微分方程的求解读者朋友可以参考偏微分方程或者数学物理方法的教程去看看，当然本人本科并不是学数学的所以我不敢多妄议。但事实上偏微分方程更多的是求数值解，因为多数偏微分方程是没有符号解的。

求解数值解更多的是利用差分代替微分的思想，与常微分方程很类似。

应用案例

案例：椭圆方程的模拟

第一个例子是求解一个特殊的椭圆方程：

$$\begin{aligned}&\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=0, x\in[0,a], y=[0,b],\\&u(0,y)=0, u(a,y)=\lambda\sin\frac{3\pi y}{b},\\&u(x,0)=0, u(x,b)=\lambda\sin\frac{2\pi x}{a}\cos\frac{\pi x}{a}\end{aligned}\tag{5.16}$$

在其中 $a=1,b=1,\lambda=1$，这个问题如果读者学习过数学物理方法或许知道，该问题可以用 Jacobbi 迭代的方法解决。对于方程右侧为 0 的椭圆方程，它的一个迭代模式就是遵循下面的式子：

$$u_{ij}=\frac{1}{4}[u_{i,j-1}+u_{i,j+1}+u_{i-1,j}+u_{i+1,j}]\tag{5.17}$$

这个方程是怎么推出来的呢？其实也就是用差分代替差分。我们把二阶偏微分改成二阶差分，x 和 y 的步长都是 h，那么可以改写成：

$$\frac{\frac{u_{i+1,j}-u_{i,j}}{h}-\frac{u_{i,j}-u_{i-1,j}}{h}}{h}+\frac{\frac{u_{i,j+1}-u_{i,j}}{h}-\frac{u_{i,j}-u_{i,j-1}}{h}}{h}=0\tag{5.18}$$

对方程进行变形，也就得到了平均值的迭代结果。

偏微分方程更多的可以参考专门的 PDE 书籍。但无论针对偏微分还是常微分方程，始终把握核心思想都是用差分代替微分。开始将求解区域初始化为 0，设置好边界以后反复迭代直到前后两次迭代结果的误差小于一个很小的阈值（这里不妨设置为 10 的-5 次方）时停止。求解的代码如下：

```
a=1;b=1;h=0.01;
x=0:h:a;
y=0:h:b;
[X,Y]=meshgrid(x,y);
Z=0.5*(1/sinh(2*pi*b/a)*sinh(2*pi*Y/a).*sin(2*pi*X/a)+...
      1/sinh(4*pi*b/a)*sinh(4*pi*Y/a).*sin(4*pi*X/a))+...
      1/sinh(3*pi*a/b)*sinh(3*pi*X/b).*sin(3*pi*Y/b);
%解析结果，用于与差分方程结果比较
figure; meshc(X,Y,Z);  %meshc输出三维等高线图
title('解析法得到的温度场的三维图与等高图')
XL=0;XR=a;
YL=0;YR=b;
N=(XR-XL)/h;
```

笔记

```
M=(YR-YL)/h;
u=zeros(N+1,M+1);
lambda=1;
u(N+1,1:M+1)=lambda*sin(3*pi/b*(0:M)*h);
u(1:N+1,M+1)=lambda*sin(3*pi/a*(0:N)*h).*cos(pi/a*(0:N)*h);
uold=u;
u(2:N,2:M)=u(2:N,2:M)+2; %启动值
while max(max(abs(u-uold)))>=1e-5
        %由于u矩阵为二维，故矩阵最大误差的求解需要使用两重max。也可以换成mean
        uold=u;
        u(2:N,2:M)=0.25*(u(3:N+1,2:M)+u(1:N-1,2:M)+u(2:N,3:M+1)+u(2:N,1:M-1));
end
u_error=mean(mean(u'-Z))
figure; meshc(X,Y,u');
title('差分法得到的温度场的三维图与等高图');
```

最终可以解得其图像:

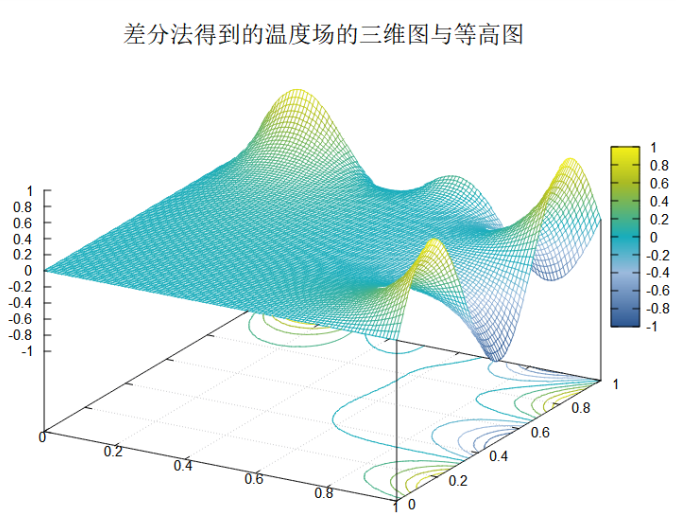


图 5.6: 差分法解椭圆方程的案例

可以看到，数值解和解析解的大体形态差异并不大。经计算，总体的平均误差仅为 0.008，与函数的函数值相比已经是非常小的水平，数值解非常接近解析解。如果我们把方程的右边改写为 x+y，那么迭代过程的方程其实也就变成了:

$$u_{ij} = \frac{1}{4}[u_{i,j-1} + u_{i,j+1} + u_{i-1,j} + u_{i+1,j} - h^2(i+j)] \tag{5.19}$$

现在大家思考，假如边界条件不变，然后要求解的方程变成了：

$$\begin{aligned} &\tfrac{\partial^2 u}{\partial x^2} + \tfrac{\partial^2 u}{\partial y^2} = x + y, x \in [0,a], y = [0,b], \\ &u(0,y) = 0, u(a,y) = \lambda \sin \tfrac{3\pi y}{b}, \\ &u(x,0) = 0, u(x,b) = \lambda \sin \tfrac{2\pi x}{a} \cos \tfrac{\pi x}{a} \end{aligned} \tag{5.20}$$

代码应当如何修改?

案例：热传导方程的模拟

求解一维热传导方程：

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}, u(0,t) = 65, u(x,0) = 20 (x \neq 0), u(1,t) = 37, \alpha = 0.0002 \tag{5.21}$$

同样利用差分代替微分的思想，利用二阶中心差商将微分改为差分可以改成：

$$\frac{u(x_i, t_{j+1})}{d_t} = \alpha \frac{u(x_{i+1}, t_j) - 2u(x_i, t_j) + u(x_{i-1}, t_j)}{d_x^2} \tag{5.22}$$

这样一来，我们便有了如下求解代码：

```
Nx=20;Nt=20;%我们对x轴分成Nx段，有Nx+1个点，对t轴分成Nt段，有Nt+1个点
xd=1;t=100;
dt=1/Nt;dx=xd/Nx;
alpha=0.0002;%热传导系数，可以自行调整，alpha越大表示传热越快
u=zeros(Nx+1,floor(t*Nt)+1);%生成(Nx+1)*(Nt+1)的矩阵
u(end,:)=37;u(:,1)=20;
u(1,:)=65;
for j=1:floor(t*Nt)
    for i=2:Nx
        u(i,j+1)=(1-2*alpha*dt/dx^2)*u(i,j)+(alpha*dt/dx^2)*(u(i+1,j)+u(i-1,j
    end
end
f=@(x,t)u(floor(x*Nx/xd)+1,floor(t*Nt)+1);
figure
X=0:0.05:1;[~,xl]=size(X);
T=0:1:100;[~,Tl]=size(T);
[xx,tt]=meshgrid(X,T);
```

笔记

```
Z=zeros(xl,Tl);
for ii=1:xl
    for jj=1:Tl
        Z(ii,jj)=f(X(ii),T(jj));
    end
end
mesh(xx,tt,Z')
xlabel('与热源之间的距离d')
ylabel('经过的时间t')
zlabel('温度')
```

得到的结果如图5.7所示:

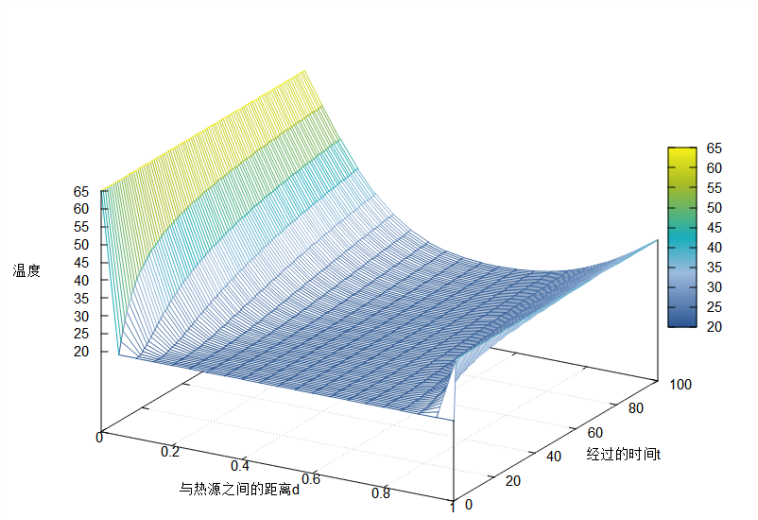


图 5.7: 差分法解椭圆方程的案例

图5.7描绘了与热源不同距离的位置在经过不同的时间 t 温度的变化规律。值得一提的是，这个问题正是由 2018 年全国大学生数学建模竞赛 A 题改编而来。

5.7 C-N 方法介绍

这一节我们简单介绍 C-N 方法。C-N 方法全面叫 Crank-Nilkson 方法，是求解偏微分方程的时候一种精度较高的数值计算方法。最早也是被用于求解热方程，例如上一讲的案例 2 和泊松方程等。

前面我们已经在欧拉法的基础上看到了改进欧拉法，是对于前向欧拉和后向欧拉的一个平均。但我们又觉得这么直接平均精度还是不够，所以又有了龙格库塔法。现在，我们用泰勒公式推断 C-N 方法：

利用泰勒展开对前向欧拉的差分公式和后向欧拉的差分公式沿着中心去进行离散化：

$$\begin{cases} u_{i+1} - u_i = h\frac{\partial u_i}{\partial x_i} + \frac{h^2}{2!}\frac{\partial^2 u_i}{\partial x_i^2} + \frac{h^3}{3!}\frac{\partial^3 u_i}{\partial x_i^3} \\ u_i - u_{i-1} = -h\frac{\partial u_i}{\partial x_i} + \frac{h^2}{2!}\frac{\partial^2 u_i}{\partial x_i^2} - \frac{h^3}{3!}\frac{\partial^3 u_i}{\partial x_i^3} \end{cases} \tag{5.23}$$

对两个式子进行合并处理，可以得到 Crank-Nilkson 的隐式处理格式：

$$\frac{\partial^2 u_i}{\partial x_i^2} = \frac{u_{i+1} + u_{i-1} - 2u_i}{h^2} \tag{5.24}$$

如果 x 和 y 的步长不一样，那么有了二元函数的五点菱形格式：

$$\frac{u_{i+1,j} + u_{i-1,j} - 2u_{i,j}}{h_1^2} + \frac{u_{i,j+1} + u_{i,j-1} - 2u_{i,j}}{h_2^2} = f(i,j) \tag{5.25}$$

显式的 C-N 方法其实思想和改进欧拉是类似的，因为前向欧拉和后向欧拉的效果削弱太多了所以我们期望对二者进行一个平均。所以，显式的 C-N 方法又叫中心差分：

$$\Delta f(x_k) = \frac{1}{2}(f(x_{k+1}) - f(x_{k-1})) \tag{5.26}$$

应用案例

案例：传热方程的 C-N 解法

请使用 Crank-Nilkson 算法解决下面的问题：

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, 0 \le x \le 1, t > 0 u(0,t) = 0, u(1,t) = 0, u(x,0) = \sin \pi x \tag{5.27}$$

给定 h=0.1, k=0.01, 计算到 t=0.1，对比结果。

我们可以写出如下代码：

```
1 % Crank-Nicholson 隐式法 （抛物型偏微分方程）
2 clear
```

笔记

```
%  du      d^2u
% ---- = -------
%  dt      dx^2

% u(t,x_start) =  g_1(t)
% u(t,x_end) =  g_2(t)
% u(t_start,x) =  f(x)
g_1 = @(t) 0;
g_2 = @(t) 0;
f = @(x) sin(pi .* x);
% □x = h  □t = k
k = 0.01;
h = 0.1;
% u j-i t-x k-h
t_start = 0;
t_end = 0.1;
x_start = 0;
x_end = 1;
u = zeros(t_end/k+1,x_end/h+1);
r = k/(h.^2);
n = x_end/h+1;
for j = 1:t_end/k+1
    t = (j-1)*k;
    % Ax = B
    A = zeros(n-2,n-2);
    B = zeros(n-2,1);
    u(j,1) = g_1(t);
    u(j,n) = g_2(t);
    for i = 2:n-1
        x = (i-1) * h;
        if j == 1
            u(j,i) = f(x);
        else
            B(i-1) = r*u(j-1,i-1) + (2-2*r)*u(j-1,i) + r*u(j-1,i+1);
            if i == 2
                A(i-1,1:2) = [2+2*r,-r];
```

```
39                  B(i-1) = B(i-1) + r*u(j,i-1);
40              elseif i == n - 1
41                  A(i-1,end-1:end) = [-r,2+2*r];
42                  B(i-1) = B(i-1) + r*u(j,i+1);
43              else
44                  A(i-1,i-2:i) = [-r,2+2*r,-r];
45              end
46          end
47      end
48      if j ~= 1
49          u(j,2:end-1) = (A\B)';
50      end
51  end
52  surf(x_start:h:x_end,t_start:k:t_end,u)
```

最终解得图像如图5.8所示:

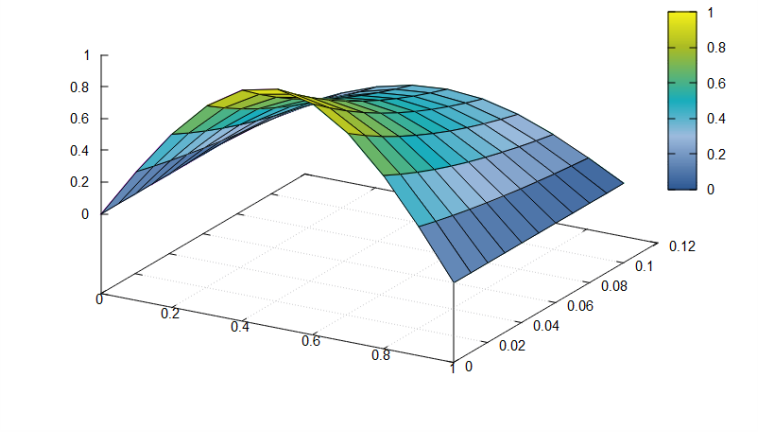


图 5.8: 使用隐式 Crank-Nilkson 法对偏微分方程的数值模拟

第六章　基于 Baltamatica 的运筹优化

本章我们学习的重点内容是 Baltamatica 的运筹优化。优化类问题是数学建模当中考察最频繁的一类问题，它更加考验选手对于模型的理解和架构。但传统的运筹优化我们可以直接调用函数（例如 Python 和 MATLAB 中的一些内置函数），更加复杂的优化问题我们可以用专业求解器求解（例如 COPT、Gurobi 等）。Baltamatica 当中对函数优化的支持并不算太多，所以这需要我们对于运筹学基本优化原理有一个更深的要求。

本章我们会从底层出发，写出一系列的数值函数极值求解算法，包括有约束和无约束等多种类型。

6.1　函数的极值求解与梯度下降法

前面我们接触过梯度的概念，这个概念是基于偏微分的定义来的。事实上，梯度具有这样一个性质：一个多元函数沿着梯度的方向下降是最快的。这个性质又可以通俗的称作“爬山法则”，利用这一准则可以求解函数的极值（主要是局部数值解）。

它的基本原理很简单，从某个起始点开始搜索，在搜索过程中计算当前位置的梯度，每一次迭代遵循如下的迭代公式：

$$x_{t+1} = x_t - \alpha \cdot grad(f) \tag{6.1}$$

当前后两次迭代的函数值之差满足一个很小的阈值（误差的容许范围）时我们认为迭代基本成功。或者从另一个角度，由于极值点的偏导数为 0，那么当梯度的模近似为 0 的时候也可以认为是极值迭代成功。但我个人建议，用函数值之差的判定准则更为准确。

事实上，在后续机器学习中的数值计算往往并不是一个简单的梯度下降去求数值解。因为机器学习中的数据量很大，梯度下降分为随机梯度、批量梯度和小批量梯度三种方法，在此基础上还可以引入动量等方法。我这里不妨提前讲了吧，因为机器学习去做个梯度下降是需要数据驱动的，而机器学习任务的数据量往往又特别大，梯度下降轮次又多。如果每一次迭代都带入所有的数据量作为函数参数，这就叫批量梯度下降，这种方式当然比较稳但是计算量是很大的。而如果每一次迭代都是随机带入某一条数据作为参数，一条条更新就是随机梯度下降，这种方式计算量就比较小但是非常不稳定容易振荡。一个折中的方法就是每次迭代都采取整个数据集的一个子集，这就是小批量梯度下降。图6.1表示了几种梯度下降的示意图：

我们可以写出一个梯度下降的案例：

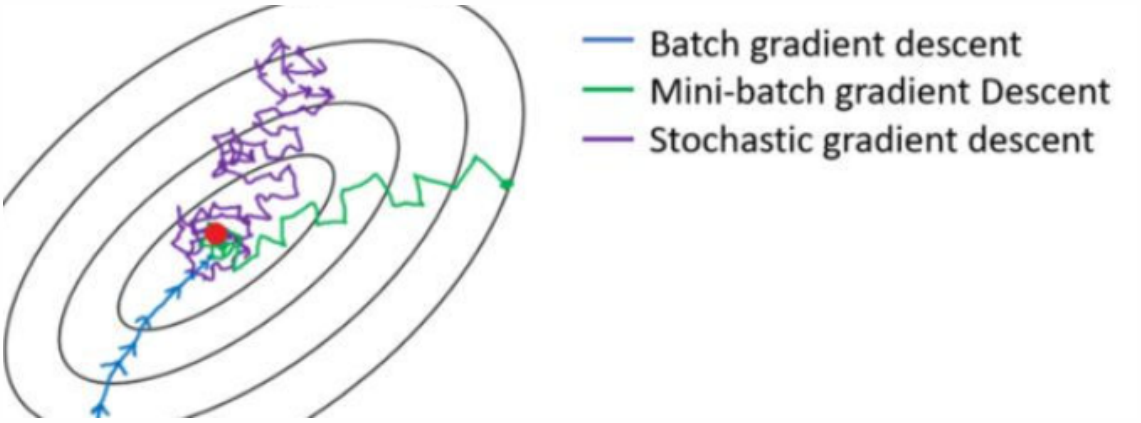


图 6.1: 不同的梯度下降方法

```
%绘制出来最原始的方程的图像，方程为f=x^2+y^2；定义域分别设置成为[-10, 10]，然
[x, y] = meshgrid(-10:0.5:10, -10:0.5:10);
z = x.^2 + y.^2;
mesh(x, y, z);
%分别求x和y的偏导数%
%dx = 2*x;
%dy = 2*y;
%设置x，y的初始值，并且每步骤更新的值都加入到这个记录中来，第一个是初始值，为
record_values = [-9;-9];
%设置每次更新的步长%
step = 0.1;
%设置迭代次数%
count = 20;
%开始迭代%
for i=1:count
    current_x = record_values(1,i);
    current_y = record_values(2,i);
    temp = [current_x - step*2*current_x, current_y - step*2*current_y];
    %把temp附加到record_values后面%
    record_values(:, i+1) = temp;
end
%计算出来点绘制出来这些点按照顺序连线%
hold on;
x_values = record_values(1,:);
y_values = record_values(2,:);
```

笔记

```
z_values = x_values.*x_values + y_values.*y_values;
plot3(x_values, y_values,z_values, 'k--o');
%我们知道当x,y等于0的时候是最小点，标示出来%
plot3(0, 0, 0, 'rp');
```

这个函数 $z = x^2 + y^2$，很显然我们知道极值点就在 (0,0) 这个位置。我们把迭代点的位置可以绘制在图6.2的位置：

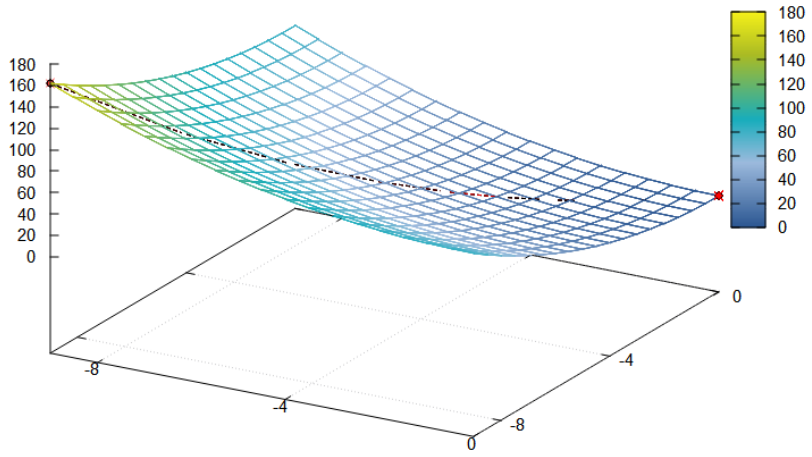


图 6.2: 梯度下降的效果图

练习应用

1. 结合前面讲过的梯度下降的内容，你能否把它改写为一个函数，输入初始值和函数的句柄形式就能对它进行极小值搜索？试着将这一个函数封装为 fmingd(f,x)，能够搜索到局部极小值即可。

2. 在后面学习到线性回归以后，你能否根据线性回归的原理写出最小二乘法的梯度下降模式？

6.2 线性规划与单纯形法

我想各位在中学阶段其实已经是对线性规划有所了解了，不过如果有些学校没学的话也没关系。在这一节当中我们会回顾以前中学接触到的线性规划，补充一些线性代数的知识和 matlab 解线性代数问题的指令，最后引出线性规划的基本形式。数学建模是一门应用数学课程，与基础数学不同，我这里不打算抢读者线性代数老师的饭碗太严重，但我们会尽可能多补充一些基础的常用的有关理论。

线性规划的实际应用有很多，比如说：我们可能见过这样一种问题，去运输一批货物的时候大车能运五箱，小车只能运三箱，但我们大车和小车数量都有限，怎么安排运输方案能够在车辆够用的情况下运费还能最小？如果我们把大车数量记为 x，小车数量记为 y，那么除了 x 和 y 的范围，5x+3y 也有自己的一个范围，算上运费作为优化目标，这不就构成了一个线性规划吗？背景熟悉吧，甚至于有一种小学应用题的恐惧感。

我们大概可以总结出，中学的线性规划通常就两个变量 x 和 y，约束条件三个不等式，最后一个线性的形如 z=ax+by（这里 ab 都是常数）的目标函数。这样的式子我们解方程可以解，画图也可以解，总能在两分钟之内算出正确结果。但在实际情况中，问题真的有这么容易吗？其实不然。同样是拿生产问题做文章，如果我们这里生产的原料不止甲乙丙三种呢（通常在有机化合物合成的时候原料可能有十几种甚至上百种），产出的产品也不止 AB 而是能够产出数十种化合物，还能简单地用高中的方法写吗？

所以我们说，中学的规划存在这样一些局限性：

- 决策变量（如果不好理解暂且称之为自变量吧）往往不止两三个。
- 当变量个数超过三个的时候还能在直角坐标系里面画图吗？不能了。
- 约束条件往往不止三个不等式，不等式可能比变量更多一些。
- 当变量较多的时候，还可能出现方程形式下的约束。
- 中学阶段我们只讨论了线性规划，但如果不等式或者目标函数非线性呢？

这么多情况不知读者朋友会不会被吓到。如果十几个甚至几十个不等式方程组成约束条件，那我草稿纸甚至不知道写不写的下，况且中学阶段没有接触过高维问题。于是，为了以更简单的形式描述更一般的线性规划，我们需要借助一样数学工具——线性代数。我们把所有的方程约束中系数做成系数矩阵 Aeq，等号右边的常数作为列向量 beq；不等式约束中的系数矩阵 A 和不等号右边的常数 b，为了方便起见通常将不等式统一为小于等于；变量 x 在向量 lb 到 ub 之间取值；目标函数的系数向量为 c，那么线性规划的标准形式就如下所示了：

$$\min f = c^T x \quad s.t. \begin{cases} Ax \leq b \\ Aeq \cdot x = beq \\ lb < x < ub \end{cases} \tag{6.2}$$

为了方便 matlab 编程，我们通常将问题统一为函数极小值问题，不等约束统一为小于等于。如果原问题是最大值或者有大于等于，那就乘-1 进行取反即可。如果读者有凸优化理论的背景可能会感到狐疑，说：为什么我看到的标准形式和你这里写的不太一样？是的，经典的凸优化教材会把模型写成另外一种形式：

$$\begin{aligned} &\max f = c^T X \\ s.t. &\begin{cases} A^* \tilde{X} = b^* \\ X \geq 0 \end{cases} \end{aligned} \tag{6.3}$$

我们暂且把这种形式称作规范形式。规范形式求的是函数的极大值，并且把不等关系和等式关系统一为等式关系方便求解。读者朋友可能会有些疑惑，说：不等式怎么可以充当为方程呢？这就是一种数学思想。可能读者朋友可以理解方程是不等式的特例，但不一定理解不等式也可以视作方程的特例，我举个例子。比如对于不等式 2a+3b+c<10，左边比右边小，但是小多少呢？我们把这个差额记作 d，左边如果补上这个差额就可以写作：2a+3b+c+d=10，这样就转化成了等式。这里的 d 被称为松弛变量。包括决策变量的上下界 lb 和 ub 也会被转化为不等关系引入松弛量。

在单纯形法中，我们解决问题通常从理论上都会把问题转换为规范形式来求解，对每一个不等式都引入一个松弛变量去增广我们的原问题。但这些松弛变量不会出现在目标函数当中。

单纯形法其实就有些类似于带入边界点轮换求解的方法，它可以说是在运筹学当中最经典的一类方法啦。基本原理是通过对基向量解轮换，看到底谁是最优解。单纯形法的基本流程如下所示：

1. 确定初始可行基和初始基可行解，并建立初始单纯形表。

2. 在当前表的目标函数对应的行中，若所有非基变量的系数非正，则得到最优解，算法终止；否则进入下一步。

3. 若单纯形表中 1 至 m 列构成单位矩阵，在 j=m+1 至 n 列中，若有某个对应 x_k 的系数列向量 $P_k \leq 0$，则停止计算。否则，转挑选目标函数对应行中系数最大的非基变量作为进基变量。假设 x_k 为进基变量，按规则 $\theta = \min(\frac{b_i}{a_{ik}} | a_{ik} > 0) = \frac{b_u}{a_{uk}}$ 计算，其中 bi 是规范型规划的常数项，a_{ik} 即为在第 i 个约束中变量 k 的系数，可确定 x_u 为出基变量，转下一步。

4. 以 auk 为主元素进行迭代，对 xk 所对应的列向量进行如下变换：$P_k = \begin{pmatrix} a_{1k} \\ a_{2k} \\ \vdots \\ a_{uk} \\ \vdots \\ a_{ml} \end{pmatrix} =$

$$\begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

5. 重复 2-5 步，直到所有检验数非正后终止，得到最优解。

我们可以写出单纯形法的代码：

```
% 求解标准型线性规划:max c*x; s.t. A*x=b; x>=0
% 本函数中的A是单纯初始表，包括:最后一行是初始的检验数，最后一列是资源向量b
% N是初始的基变量的下标
% 输出变量sol是最优解，其中松弛变量（或剩余变量）可能不为0
% 输出变量val是最优目标值，kk是迭代次数
function [sol,val,kk]=slinprog(A,N)
 [mA,nA]=size(A);
 kk=0; % 迭代次数
flag=1;
 while flag
kk=kk+1;
    if A(mA,:)<=0 % 已找到最优解
        flag=0;
        sol=zeros(1,nA-1);
        for i=1:mA-1
            sol(N(i))=A(i,nA);
        end
        val=-A(mA,nA);
    else
        for i=1:nA-1
            if A(mA,i)>0&A(1:mA-1,i)<=0 % 问题有无界解
                disp('have infinite solution!');
                flag=0;
                break;
            end
        end
        if flag % 还不是最优单纯形表，进行转轴运算
```

```
            temp=0;
            for i=1:nA-1
                if A(mA,i)>temp
                    temp=A(mA,i);
                    inb=i; % 进基变量的下标
                end
            end
            sita=zeros(1,mA-1);
            for i=1:mA-1
                if A(i,inb)>0
                    sita(i)=A(i,nA)/A(i,inb);
                end
            end
            temp=inf;
            for i=1:mA-1
                if sita(i)>0&sita(i)<temp
                    temp=sita(i);
                    outb=i; % 出基变量下标
                end
            end
            % 以下更新N
            for i=1:mA-1
                if i==outb
                    N(i)=inb;
                end
            end
            % 以下进行转轴运算
            A(outb,:)=A(outb,:)/A(outb,inb);
            for i=1:mA
                if i~=outb
                    A(i,:)=A(i,:)-A(outb,:)*A(i,inb);
                end
            end
        end
    end
end
```

笔记

怎么测试它呢？例如对于一个线性规划问题：

$$\max 2x_1 + 3x_2$$
$$s.t. \begin{cases} x_1 + 2x_2 \leq 8 \\ 4x_1 \leq 16 \\ 4x_2 \leq 12 \\ x_1, x_2 \geq 0 \end{cases} \tag{6.4}$$

加入松驰变量，化为规范型，得到：

```
A=[1  2  1  0  0  8;
   4  0  0  1  0  16;
   0  4  0  0  1  12;
   2  3  0  0  0  0];
N=[4  3];
[sol,val,kk]=slinprog(A,N)
```

就可以得到最终结果为 (4,2)，最优解为 14。问题的求解还是比较顺利的。

应用案例

在应用案例开始之前，我希望各位能够先完成一个任务：由于 slinprog 函数都是需要手工进行规范形转化就比较繁琐，我们希望有一个函数能够自动进行规范形转化。定义函数 linprog(c,A,b,Aeq,beq,ub,lb)，输入参数分别为目标函数，不等式矩阵，不等式右侧，等式矩阵，等式右侧，下界，上界，请在这个函数中将线性规划标准形换为规范形，最后调用函数 slinprog(As,N) 得到最终结果。

生产决策问题

某厂生产甲、乙两种产品，已知制成一吨产品甲需用资源 A 3 吨，资源 B $4m^3$，制成每吨产品乙需用资源 A 2 吨，资源 B $6m^3$，资源 C 7 个单位。若每吨产品甲和乙的经济价值分别为 7 万元和 5 万元，3 种资源的限制量分别为 80 吨、$220m^3$ 和 230 个单位，试分析应生产这两种产品各多少吨才能使创造的总经济价值最高?

这里可以令生产产品甲的数量为 x_1，生产产品乙的数量为 x_2。很显然，这就是一个二变量的线性规划问题。其中 A 资源限制、B 资源限制和 C 资源限制分别可以用数学语言翻译为：

$$\begin{cases} 3x_1 + 2x_2 \leq 80 \\ 4x_1 + 6x_2 \leq 220 \\ 7x_2 \leq 230 \end{cases} \tag{6.5}$$

随后思考需要最大化的数据量为总体的经济价值 $7x_1 + 5x_2$，然后再考虑变量都是非负数，那么根据题意，代码设置如下：

```
clc
clear
f = [-7;-5];
A = [3  2
     4  6
     0  7];
b = [80;220;230];
lb = zeros(2,1);
[x,f] = linprog(f,A,b,[],[],lb)
```

读者朋友们应该很容易可以通过手算的方式解出问题的答案，那么试试看，你的 linprog 函数能正确工作吗?

笔记

工件生产问题

某车间有两台机床甲和乙，可用于加工 3 种工件。假定这两台机床的可用台时数分别为 600 和 900,3 种工件的数量分别为 400、600 和 500，且已知用两台不同机床加工单位数量的不同工件所需的台时数和加工费用 (如表所示)，问怎样分配机床的加工任务，才能既满足加工工件的需求，又使总加工费用最低?

这里可设在甲机床上加工工件 1、2 和 3 的数量分别为 $x_1x_2x_3$，在乙机床上加工工件 1、2 和 3 的数量分别为 $x_4x_5x_6$，根据 3 种工种的数量限制，则有：$x_1+x_4=400, x_2+x_5=600, x_3+x_6=500$，根据题意：

```
clc
clear
f = [13;9;10;11;12;8];
A = [0.6  1.2  1.1  0  0  0
     0  0  0  0.4  1.2  1.0];
b = [600;900];
Aeq = [1  0  0  1  0  0
       0  1  0  0  1  0
       0  0  1  0  0  1];
beq = [400  600  500];
lb = zeros(6,1);
[x,fval] = linprog(f,A,b,Aeq,beq,lb)
```

求解得在甲机床上加工 500 个工件 2，在乙机床上加工 400 个工件 1、加工 100 个工件 2、加工 500 个工件 3，可在满足条件的情况下使总加工费用最小，最小费用为 14100 元。

厂址选择问题

A、B、C 三地，每地都出产一定数量的产品，也消耗一定数量的原料 (如表6.1所示)，已知制成每吨产品需 3 吨原料，各地之间的距离为：A-B，150km；A-C，100km，B-C，200km。假定每万吨原料运输 1km 的运价是 5000 元，每万吨产品运输 1km 的运价是 6000 元。由于地区条件的差异，在不同地点设厂的生产费用也不同。问究竟在哪些地方设厂，规模多大，才能使总费用最小？另外，由于其他条件限制，在 B 处建厂的规模 (生产的产品数量) 不能超过 6 万吨。

这个问题还是略有一些小复杂的哈，里面其实可以体会一下松弛变量这个思想的应用。方便起见。这里可令 x_{ij} 为由 i 地运到 j 地的原料数量 (万吨)，y_{ij} 为由 i 地运往 j 地的产品数量 (万吨)，i，j = 1,2,3(分别对应 A、B、C 三地)，根据题意，我们各个击破：先看看 A 地，A 的产品生产得靠 BC 运给 A 的产品才能消耗，而一吨 A 得三吨原材料，并且 A 的原材料会流通给 BC，同时

表 6.1: 厂址选择问题的数据

地点	年产原材料/t	年销售产品/t	生产费用/万元
A	20	7	150
B	16	13	120
C	24	0	100

也会收到 BC 流通来的原料，所以有：

$$3y_{11} + 3y_{12} + x_{12} + x_{13} - x_{21} - x_{31} \leq 20 \tag{6.6}$$

类似地如法炮制可以写出 BC。那么再看，AB 两个地方的产品要卖出去吧，销售额限制：

$$y_{11} + y_{12} + y_{13} = 7 \tag{6.7}$$

同样有 AB 两个等式。再加上 B 不能超过 6 万吨这个条件，翻译过来：

$$y_{12} + y_{22} \leq 6 \tag{6.8}$$

最后我们看看目标函数怎么写。目标函数要最小化成本，第一个是运输原材料，1km 运价 0.5 万元，所以原材料的部分就是 $0.5 * (150x_{12} + 150x_{12} + 100x_{13} + 100x_{31} + 200x_{23} + 200_{32}$。那么第二个部分也就是产品运输，1km 运价 0.6 万元，这一部分相对麻烦，有自给自足的和外来产物。自给自足的产品这个好理解，因为 C 是不需要销售产品的，这个部分就是 $150y_{11} + 120y_{22}$。而运输产品这个部分怎么算呢？以 y_{21} 这个部分为例，它需要自我生产销售的同时还有运费 150*0.6=90，所以这个部分应该是 $(120 + 150 * 0.6)y_{21}$，以此类推。所以，完整的模型就应该是：

$$\min 0.5 * (150x_{12} + 150x_{12} + 100x_{13} + 100x_{31} + 200x_{23} + 200_{32}) + (150y_{11} + 120y_{22}) + (120 + 150 * 0.6)y_{21} +$$
$$s.t. \begin{cases} 3y_{11} + 3y_{12} + x_{12} + x_{13} - x_{21} - x_{31} \leq 20 \\ 3y_{21} + 3y_{22} + x_{21} + x_{23} - x_{12} - x_{32} \leq 16 \\ 3y_{31} + 3y_{32} + x_{32} + x_{31} - x_{12} - x_{13} \leq 24 \\ y_{11} + y_{12} + y_{13} = 7 \\ y_{21} + y_{22} + y_{23} = 7 \\ y_{12} + y_{22} \leq 6 \\ x, y \geq 0 \end{cases} \tag{6.9}$$

我们可以写出下面的代码：

```
clc
clear
f = [75,75,50,50,100,100,150,240,210,120,160,220];
```

笔记

```
A = [1 -1 1 -1 0 0 3 3 0 0 0 0
     -1 1 0 0 1 -1 0 0 3 3 0 0
     0 0 -1 1 -1 1 0 0 0 0 3 3
     0 0 0 0 0 0 0 0 1 1 0 0];
b = [21;17;22;6];
Aeq = [0 0 0 0 0 0 1 0 1 0 1 0
            0 0 0 0 0 0 0 1 0 1 0 1];
beq = [6;12];
lb = zeros(12,1);
[x,fval] = linprog(f,A,b,Aeq,beq,lb);
```

可见要使总费用最小，A、B、C 三地的建厂规模分别为 6 万吨、5.667 万吨和 6.333 万吨

6.3 拉格朗日方法与 KKT 条件

从线性到非线性，多元函数可能初来乍到的同学并不一定理解。在高中我们说，函数是从一个集合（定义域）到另一个集合（值域）的一一对应的映射，那现在多元函数？不就变成多个集合到一个集合的映射了嘛？这，怎么也能叫一一对应呢？诶，这个时候你就注意了，多元函数仍然是一个集合到另一个集合的映射，只不过自变量的集合不是数集，而是点集，或者说是 n 维空间里面向量的集合。

如果高中学过导数的同学可能知道，一元函数求极值的一个方法就是求导导数为 0。对于多元函数也类似，如果是无约束就只是给了一个多元函数求极值，只需要针对每个变量分别求导数（我们把这个过程叫做求偏导），所有偏导为 0 解方程组得到的就是极值点的候选解。当然，我们有可能解出来的是 x3=0 的这种极值解，这种情况下我们会通过二阶导数的符号去进一步判定。

当我们碰上了有约束条件下的非线性函数求极值的时候，我们通常使用拉格朗日法。例如，对于广义的含等式条件（先暂时只考虑等式问题）的极值问题：

$$\min f(x) \quad s.t. \begin{cases} C_1(x) = 0 \\ C_2(x) = 0 \\ \vdots \\ C_n(x) = 0 \end{cases} \tag{6.10}$$

我们通过引入 n 个称之为拉格朗日乘子的常数把原问题改写为新的函数：

$$\min L(x, \lambda) = f(x) + \lambda_1 C_1(x) + \lambda_2 C_2(x) + \ldots + \lambda_n C_n(x) \tag{6.11}$$

接下来就像解无约束极值一样对每个 x 和乘子求偏导即可解决问题。而当我们在问题中考虑不等条件，那么这个问题的求解策略其实类似，我们称其为 KKT 条件。问题：

$$\min f(x) \quad s.t. \begin{cases} h(x) = 0 \\ g(x) \le 0 \end{cases} \tag{6.12}$$

我们分别引入两个不同乘子，函数 L 将在 X 取得极值当且仅当：

$$\min L(x, \lambda, \mu) = f(x) + \lambda h(x) + \mu g(x) \quad s.t. \begin{cases} \frac{\partial L}{\partial X} = 0 \\ \lambda \neq 0 \\ \mu \ge 0 \\ \mu g(X) = 0 \\ h(X) = 0 \\ g(X) < 0 \end{cases} \tag{6.13}$$

当然，如果想求的不是一个精确解而是一个近似的数值解，那么我们的方法同样有很多。蒙特卡洛模拟几乎是用的最广泛的一种。

Baltamatica 中提供了 fminbnd 和 fminsearch 两种方法求函数极小值。fminbnd 方法只针对一元函数，它的作用是搜索一元函数在某个区间内的极小值。传递参数除了函数以外还需要给出自变量的搜索区间。例如：

```
>> func=@(x)-log(x)/x;
>> [x fval]=fminbnd(func,0,5)

x =

    2.7183


fval =

   -0.3679
```

我们更多的会解多元函数的极值。解决无约束的多元函数极值问题我们常用函数有 fminsearch 函数：

```
>> newfunc=@(x)(x(1)-1)^2+(x(2)-1)^2;
>> [x fval]=fminsearch(newfunc,[0,0])
```

给一个目标函数，再给它起始搜索的位置（这里我设置的从 [0,0] 开始搜索），fminsearch 只是返回了一个很小很小的数但还并不是 0。这是它的数值解。但一个问题在于它求解的是无约束条件下的极值解。

练习应用

1. 如何基于 fminsearch 函数进行修改，将有等式约束的函数极值用拉格朗日乘子法改写为函数 fminconeq

2. 在问题 1 的基础上，如果增加不等式约束，那么你能否基于 KKT 方法将函数改写为 fmincon？（这一个问题较难，fmincon 的实现使用蒙特卡洛模拟可能更好）

笔记

6.4 蒙特卡洛模拟

蒙特卡洛方法是一种求解规划时常用的方法。它基于一个事实：大量的重复试验下频率可以估计概率，也就是用大规模的候选解模拟出一个近似值逐步逼近精确解。理论上只要实验次数够多精度够细它可以无限逼近精确解。

我想各位上中学的时候还没忘记蒲丰投针估计圆周率的故事吧，如果不幸忘记了，也还记得撒黄豆估计圆周率的方法吧。在一个正方形中画一个内切圆，往正方形内撒一大把黄豆，通过数出圆里面的黄豆和正方形里面的黄豆之比可以估计圆周率的近似值。这一原理也被广泛用于求函数的定积分。图6.3是一个利用蒙特卡洛方法求圆周率的例子，通过统计方形中点的个数和曲线下方点的个数之比，就可以近似模拟扇形面积与方形面积之比。

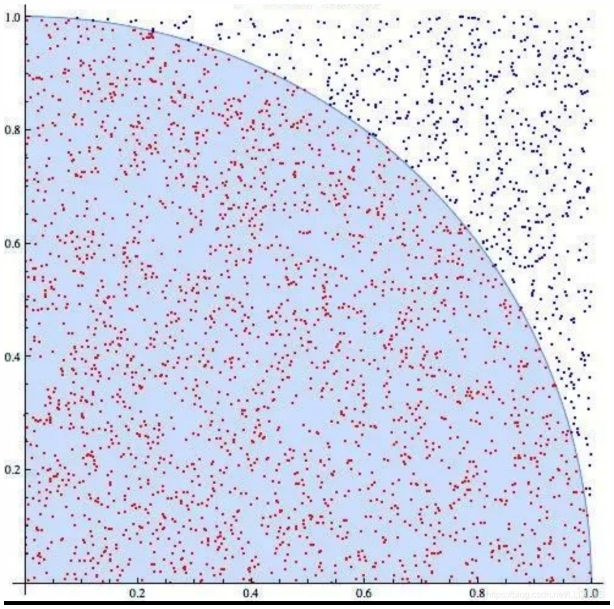


图 6.3: 使用蒙特卡洛方法估计圆周率

蒙特卡洛方法求线性规划的近似最优解我们可以看一个例子：

$$\begin{aligned} &\max f = x_1 + x_2 + 3x_3 + 4x_4 + 2x_5 \\ &s.t. \begin{cases} x_1 + x_2 + x_3 + x_4 + x_5 \leq 400 \\ x_1 + 2x_2 + 2x_3 + x_4 + 6x_5 \leq 800 \\ 2x_1 + x_2 + 6x_3 \leq 200 \\ x_3 + x_4 + 5x_5 \leq 200 \end{cases} \end{aligned} \tag{6.14}$$

对于这个问题，我们可以写出如下代码：

```
function [f,g] = fcons(x)
f=-x(1) + x(2) + 3*x(3) + 4*x(4) + 2*x(5);
% 约束函数
```

笔记

```
g=[sum(x)-400  % sum()函数用于对向量求和，或对矩阵的列求和，即x(1) + x(2) + x
   x(1) + 2*x(2) + 2*x(3) + x(4) + 6*x(5) - 800
   2*x(1) + x(2) + 6*x(3) - 200
   x(3) + x(4) + 5*x(5) - 200];
end
```

进行大量重复试验:

```
% 蒙特卡洛法
p=0;
for i = 1:10^6
    x=randi([0,99], 1, 5); % 产生一行五列的在区间[0, 99]上的随机整数
                           % randi()函数，用于产生一个随机整数矩阵
    [f, g] = fcons(x);  % 调用fcons()函数
    if all(g<=0)     % 使用约束函数进行限制
                     % all()函数 对向量/矩阵所有项进行相同的逻辑比较，并返回bo
        if p < f   % 用p储存上一个的f值，并通过比较来得到最大的f
            x0 = x; % 记录目前达到最大值的x值
            p = f; % 记录目前的最大值
        end
    end
end
x0,p     % 输出x0和p的值
```

它的本质就是在大的范围内进行多次随机采样, 统计在可行域内的样本点谁的函数值最优即可。

应用案例

在进行应用案例之前,请各位思考今天的思考题:将上面的蒙特卡洛代码改写为函数 fmincon,能够求解一般规划问题。等 fmincon 函数写好了，我们来看下面的一个例子。

厂址选择问题-II

某公司有 6 个建筑工地要开工, 每个工地的位置 (用平面坐标系 a,b 表示, 距离单位: 千米) 及水泥日用量 d(吨) 由表 2.5 给出。规划设立两个料场位于 A,B, 日储量各为 20 吨。假设从料场到工地之间均有直线道路相连。试确定料场的位置, 并制定每天的供应计划, 即从 A,B 两料场分别向各工地运送多少吨水泥, 使总的吨千米数最小。数据如表??所示:

表 6.2: 问题使用的数据表

工地	1	2	3	4	5	6
a	1.25	8.75	0.5	5.75	3	7.25
b	1.25	0.75	4.75	5	6.5	7.25
d	3	5	4	7	6	11

这个问题的示意图如图6.4所示:

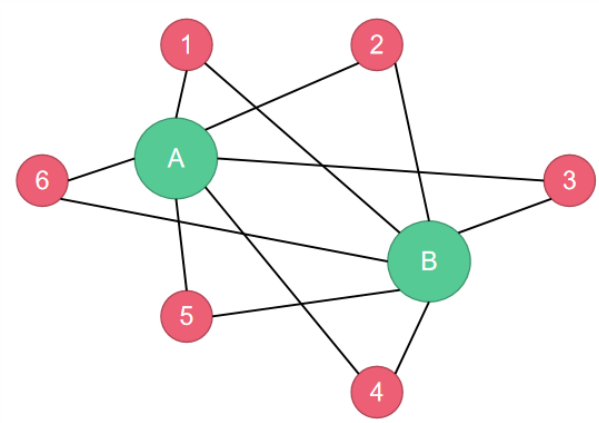


图 6.4: 工地与料场的示意图

我们可以分析这一个问题。我们说，规划问题的核心有三样：决策变量，目标函数和约束条件。决策变量包括哪些？首先两个料场的坐标未知吧，坐标有横纵坐标于是这就有了四个变量；两个料场到六个工地 12 条线路上的运输量也未知吧，于是这就又来了 12 个变量，一共是 16 个。距离可以用欧几里得距离来计算，所以可以写出一个目标函数:

```
1 function  s=tkm(x)
2       s=0;
```

```
    j=4;
    a=[1.25  8.75  0.5  5.75  3  7.25];
    b=[1.25  0.75  4.75  5  6.5  7.25];
    for i=1:6
        s=s+x(j+1)*sqrt((x(1)-a(i))^2+(x(2)-b(i))^2)+x(j+2)*sqrt((x(3)-a(i))^2
        j=j+2;
    end
end
```

这里的 x 是一个 16 维的向量，前四维表示两个料场的坐标，后面 12 个分别表示料场 1 到六个工地的运输量和料场 2 到六个工地的运输量。运输量之间需要满足限制。首先，料场 1 到六个工地的运输量之和与料场 2 到六个工地的运输量之和都不能超过 20 吨，这是存量的限制；其次，每个工地从两个料场获取的运输量之和得等于自己的一个需求量。这里其实如果把每个工地的限制看作一个大于等于也是说得通的，但我们这里为了求解方便把这一部分看作等式约束也是没问题的。

$$\min f(x)=\sum_{i=1}^{6}\sum_{j=1}^{2} m_{ij}\sqrt{(x_j-a_i)^2+(y_j-b_i)^2}$$
$$s.t.\begin{cases}\sum_{i=1}^{6} m_{ij}\le 20, j=1,2\\ \sum_{j=1}^{2} m_{ij}=d_i, i=1,2,\ldots,6\\ m_{ij}\ge 0\\ x_1,x_2,y_1,y_2\ge 0\end{cases} \tag{6.15}$$

这样我们就可以写出等式约束和不等式约束并使用 fmincon 求解：

```
d=[3  5  7  7  6  11];
Aeq=[0  0  0  0  1  0  0  0  0  0  1  0  0  0  0  0;%每一行是每个工地求和
     0  0  0  0  0  1  0  0  0  0  0  1  0  0  0  0;
     0  0  0  0  0  0  1  0  0  0  0  0  1  0  0  0;
     0  0  0  0  0  0  0  1  0  0  0  0  0  1  0  0;
     0  0  0  0  0  0  0  0  1  0  0  0  0  0  1  0;
     0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  1];
beq=d';
A=[0  0  0  0  1  1  1  1  1  1  0  0  0  0  0  0; %对料场1求和
   0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1];%对料场2求和
b=[20  20]';
lb=[0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0];
ub=[inf  inf  inf  inf  inf  inf  inf  inf  inf  inf  inf  inf  inf  inf  inf  inf];
[x fval]=fmincon(@tkm,rand(16,1),A,b,Aeq,beq,lb,ub);
```

笔记

当你去运行这个函数，求得的最小吨千米数为 69.6134，两个料场的坐标分别为 (3,6.5) 和 (0.5,4.75)，求解结束。注意这个地方由于运载量和坐标都是未知的，所以目标函数实际上是一个非线性规划问题。大家还可以思考一下如果把所有的约束都看成不等约束那么结果会怎么变。

6.5 整数规划与分支定界法

在很多实际问题中，变量的取值不仅是有范围的，有时候还有一项重要约束：变量必须取整数。整数这一限制就将连续的空间割裂成了离散的状态空间，虽然说它并不是从无穷到有穷，但求解这类整数规划问题仍然是较为有趣的。一般而言如果不做特殊说明，我们这里解的整数规划是线性规划的整数版。当然，非线性规划的整数版其实原理类似。

离散优化的例子其实也很简单，如果把前面的线性规划或者非线性规划当中加上一条约束：自变量取整数，这个问题就开始有意思了。可能最优解是一个全浮点数，但加上整数约束以后究竟在哪个整数点上取到最优解那还真说不好，你如果用枚举的方式去解那复杂度是成倍上涨。我们想，一定有更快的求解策略。

另一类离散优化的典型例子是匹配问题和组合优化问题，比如有 100 个人匹配 100 项任务，百配百就有五千种匹配模式。而这么多的匹配模式中究竟哪一个是最优解呢？那就得在五千种匹配模式中搜索，变量就是某一个人是否匹配某一项任务，取值只能是 0,1。所以这就是一种离散优化。下面我们也会对这类问题进行介绍。

同单纯形法与蒙特卡洛法之于线性规划，解整数规划的基础原理其实更多。最典型的两种算法就是分支定界法和割平面法。

分支定界法是一种经典的搜索算法，这里把它用在规划当中主要是为了对上下界进行搜索。分支定界本质上是构造一棵搜索树进行上下界搜索，它会把问题的搜索空间组织成一棵树，而分支就是从根出发将原始问题按照整数约束去分支为左子树和右子树，通过不断检查子树的上下界去搜索最优解的过程。

我们举一个例子：求解整数规划

$$\max x_1 + x_2 + x_3$$
$$s.t. \begin{cases} 7x_1 + 8x_2 + 7x_3 \leq 14 \\ x_{1,2,3} = 0, 1 \end{cases} \tag{6.16}$$

首先我们忽略取值 0,1 这个条件，把它就当做一个取值范围 [0,1] 之间的线性规划去做，它肯定是有最优解的，这毋庸置疑。现在，我们从 x3 开始分支，分别按取 0 和取 1 去对原始问题进行划分。如果 x_3 取值为 1，那么原始问题变成了 $7x_1 + 8x_2 \leq 7$ 的条件下最大化 $x_1 + x_2 + 1$；如果取值为 0 那么原始问题变成了 $7x_1 + 8x_2 \leq 14$ 的条件下最大化 $x_1 + x_2$。然后我们分别计算两边的最优解，两边都可能存在最优整数解于是对这两种情况再进行划分。每划分一次我们就会对同一层的子问题求解对应的线性规划（把整数条件换成区间条件）观察谁最小谁可分，到最后遍历完成就得到了最优解。如图**??**所示：

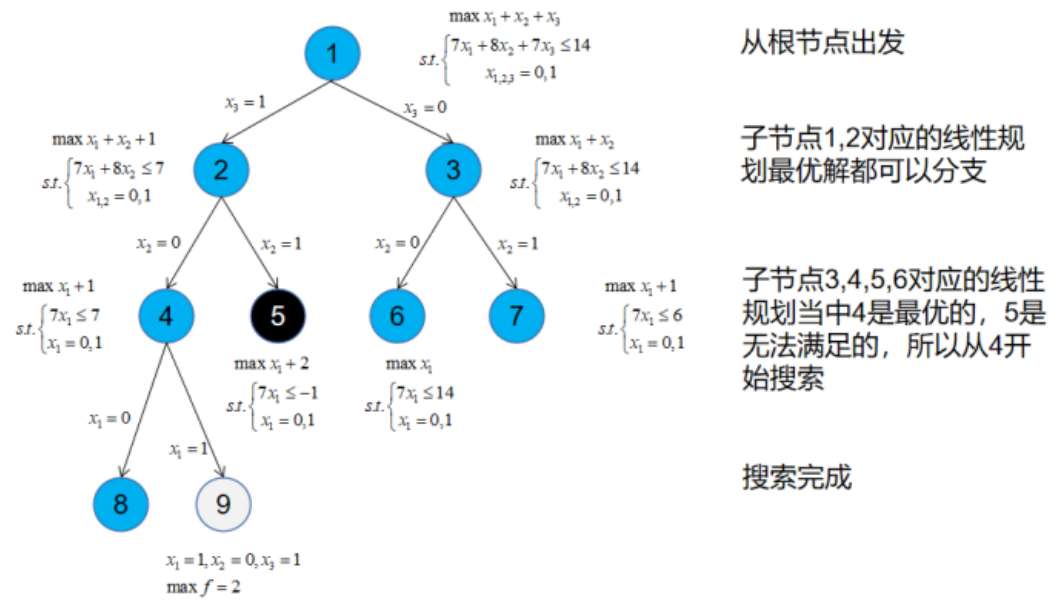


图 6.5: 分支定界法的流程图

我们每经过一层就会更新一个线性规划的最优解（也可以叫松弛解），在根节点我们将它设置为负无穷，而在子节点中只要能够比上一次的最优解更优我们就会更新这个最优解。比如说在第三层，节点 4 的最优解 2 就是最优，而节点 6 的最优解 1 和节点 7 的最优解 1.8571 再怎么分不会比节点 4 更优，所以我们下一步只对节点 4 分支并按照这一分支为子问题定界。

我们可以写出分支定界的代码，这份代码将利用到我们线性规划那一节的思考题中应用的 linprog 函数：

```
%A,b,c分别对应此题的不等式约束系数矩阵，不等式约束常数向量，目标函数系数向量
%Aeq    等式约束系数矩阵，    Beq    等式约束常数向量
%vlb    定义域的下界          vub    定义域的上界
%optXin    每次迭代的最优x    optF    每次迭代最优的f值    iter迭代次数

function [xstar, fxstar, flagOut, iter] = BranchBound1(c,A, b, Aeq, Beq, vlb,
    global optX optVal optFlag;%将最优解定义为全局变量
    iter = iter + 1;
    optX = optXin; optVal = optF;%更新迭代得到的值
    % options = optimoptions("linprog", 'Algorithm', 'interior-point-legacy',

    [x, fit, status] = linprog(c,A, b, Aeq, Beq, vlb, vub, []);
    %status返回算法迭代停止原因
    %status = 1 算法收敛于解x，即x是线性规划的最优解
    if status ~= 1%没有找到最优解，此分支不用继续迭代下去，返回
```

```
16          xstar = x;
17          fxstar = fit;
18          flagOut = status;
19          return;
20      end
21
22      if max(abs(round(x) - x)) > 1e-3%找到的函数最优解仍不是整数解
23          if fit > optVal   %此题求解的是max,此时的函数值大于之前解得的值
24              xstar = x;
25              fxstar = fit;
26              flagOut = -100;
27              return;
28          end
29
30      else%此时解得的函数解为整数解，此分支求解结束，不再继续向下求解，返回
31          if fit > optVal   %此题求解的是max,此时的函数值大于之前解得的值
32              xstar = x;
33              fxstar = fit;
34              flagOut = -101;
35              return;
36          else    %解出的值<之前解得的值，先放入全局变量中暂时存放
37              optVal = fit;
38              optX = x;
39              optFlag = status;
40              xstar = x;
41              fxstar = fit;
42              flagOut = status;
43              return;
44          end
45      end
46      midX = abs(round(x) - x);%得到x对应的小数部分
47      notIntV = find(midX > 1e-3);%得到非整数的x的索引值，find()函数返回非0的索
48      pXidx = notIntV(1);%得到第一个非整数x的下标索引
49      tempVlb = vlb;%临时拷贝一份
50      tempVub = vub;
51      %fix(x) 函数将x中元素零方向取整
```

```
    if vub(pXidx) >= fix(x(pXidx)) + 1%原上界大于此时找到的分界的位置值
        tempVlb(pXidx) = fix(x(pXidx)) + 1;%将这个分界位置值作为新的下界参数传
        [~, ~, ~] = BranchBound1(c,A, b, Aeq, Beq, tempVlb, vub, optX, optVal
    end

    if vlb(pXidx) <= fix(x(pXidx))%原下界小于此时找到的分界的位置值
        tempVub(pXidx) = fix(x(pXidx));%将这个分界位置值作为新的上界参数传入,
        [~, ~, ~] = BranchBound1(c,A, b, Aeq, Beq, vlb, tempVub, optX, optVal
    end
    xstar = optX;
    fxstar = optVal;
    flagOut = optFlag;
end
```

我们可以用如下的测试用例进行测试:

```
clear;
clc;

A = [9 7;7 20];
b = [56 70];
c = [-40,-90];%标准格式是求min，此题为max，需要转换一下

lb = [0; 0];%x值的初始范围下界
ub=[inf;inf];%x值的初始范围上界

optX = [0; 0];%存放最优解的x，初始迭代点(0,0)
optVal = 0;%最优解
[x, fit, exitF, iter] = BranchBound1(c,A, b,[], [], lb, ub, optX, optVal, 0)
```

线性规划在欧几里得空间中的可行域本质上是一个类似于“多边形”的结构，而整数规划的目的本质上是在这个“多边形”中进行修剪找到一个边界都是整数的可行域。而对这个可行域的修剪过程我们又称之为割平面法。当然，除了割平面法以外，还有隐枚举法、蒙特卡洛方法等也都是解整数规划的常用方法。如果想深入了解其原理可以参考运筹学教材中对他们的解释。这里我们主要介绍分支定界的策略，仅仅是了解现代优化工具背后运作的底层逻辑。

6.6 指派问题与匈牙利法

0-1 规划是整数规划中最特殊的一种。它的限制不仅仅是要求变量是整数，而且只能是 0 或 1，故而名曰 0-1 规划。事实上，在数学建模竞赛当中，0-1 规划可以说是最常见的整数规划，是学习的重点。

指派问题又是怎么一回事呢？我们可以看到这样一个例子：

现在有 4 个人 A,B,C,D 可以做四项工作 1,2,3,4，他们每个人只能做一项工作，所需要的时间按照表6.6给出：

表 6.3: 工作安排表

时间	**1**	**2**	**3**	**4**
A	6	7	11	2
B	4	5	9	8
C	3	1	10	4
D	5	9	8	2

我们把四个人与 4 项工作对应的 4*4=16 项安排作为决策变量，变量取值为 0,1，表示某个人是否执行某项工作。一个人只能做一项工作，一项工作也只能一个人来做，比如对 A 而言，A 如果做了 2 就不能做 134，所以 A 匹配的四个变量只能有一个是 1 其余三个都是 0。同样的，对任务 2 而言如果它被安排给了 A 那么 BCD 也都不能去做，所以任务 2 匹配的四个变量也只能一个是 1 其余三个是 0。由此，我们给出定义：

$$\begin{aligned} &\min f(x) = \sum_{i=1}^{4}\sum_{j=1}^{4} x_{ij}T_{ij} \\ &s.t. \begin{cases} \sum\limits_{i=1}^{4} x_{ij} = 1 \\ \sum\limits_{j=1}^{4} x_{ij} = 1 \\ x_{ij} = 0, 1 \end{cases} \end{aligned} \tag{6.17}$$

指派问题和 0-1 规划的解法可以从 matlab 的整数规划函数中进行约束，但还有一种经典的算法可以解 0-1 规划。这个方法被称作匈牙利法。匈牙利法的操作比较有趣，对于时间排布表，首先将每行减去当前行的最小值，然后将每一列的值减去当前列的最小值。接下来一步比较有趣，需要用最少的水平线和竖直线覆盖所有的 0 项。如果线条总数为 4 那么算法停止，给出指派方案；如果少于 4 条那么则计算没有被覆盖的最小值，将没有被覆盖的每行减去最小值，被覆盖的每列加上最小值，然后重新进行覆盖。整个过程如图**??**：

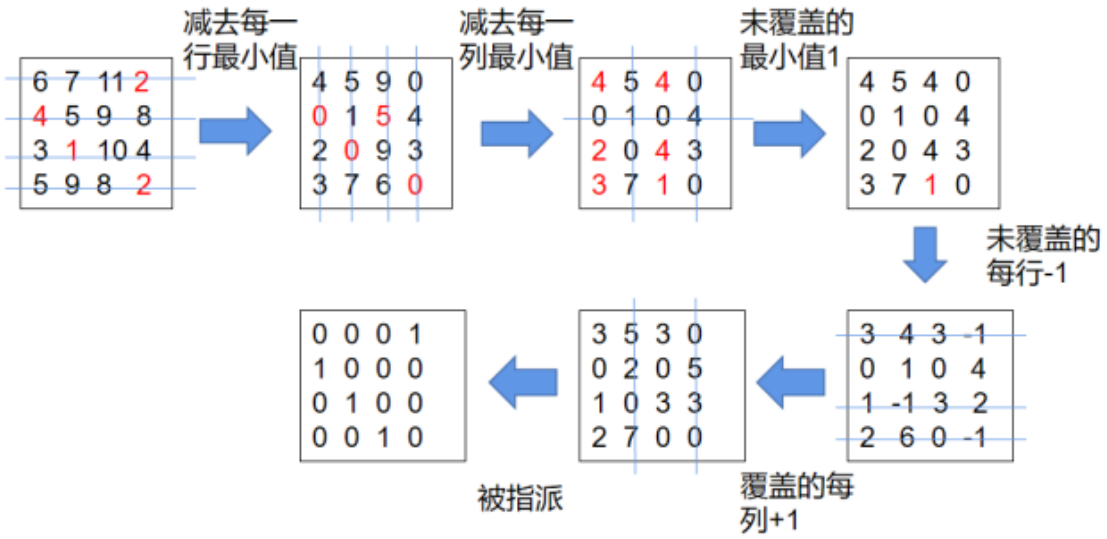


图 6.6: 指派问题的解法流程

练习应用

1. 请根据前面整数规划一节的分支定界法和这一节的匈牙利法，写出解指派矩阵问题的对应代码。此时我们可以规定指派矩阵是一个 n*n 的方阵。

2. 如果这一指派矩阵不再是方阵而是能安排的人比工作任务更多，那么匈牙利法还奏效吗？问题 1 的代码又是否需要做一些修改？请给出简要的思考（可以不用实现）。

6.7 图论中的最短路径问题

自然界中存在的大量复杂系统都可以通过形形色色的网络加以描述。一个典型的复杂网络由许多节点与节点之间的边组成，其中节点用来代表真实系统中不同的实体，而边则用来表示实体间的关系。它是一种基于图论提出的模型，具有相对较为复杂的拓扑结构。复杂网络建模试图解决三个问题：第一，找出可以刻画网络拓扑结构和行为的统计特性，并且给出衡量这些统计特性的方法；第二，构建网络模型以便帮助我们理解这些统计特性背后的真正意义；第三，基于这些统计特性，研究网络中的行为与局部规则。

图论的研究最早起源于欧拉提出的哥尼斯堡七桥问题，也就是能否一笔画走完，自此对图形的研究不再只是单纯考虑其几何关系而是更多地考察拓扑特性。一个网络是由若干个节点通过若干条有向或无向边连接起来用以描述节点间关系的图。在网络中，点的度是指以节点作为顶点的边的数目，即连接该节点的边的数目。若为无向图，入度则为节点作为终点的有向线段数，出度为节点为起点的有向线段数。而网络的度指网络中所有节点度的平均值。度分布 P(k) 指网络中一个任意选择的节点，它的度恰好为 k 的概率。如果边上面加了权值，我们称这个图是一个有权图。

最短路径问题需要基于一幅复杂网络图，分析从节点 i 到节点 j 最短的路径，即使得路径中边的权值之和最小；如果是无权图，那就是需要经历的边条数最少。这个问题当然可以抽象成一个离散优化去做，但是实际上有一些非常经典的算法可以解决这个问题。

Floyd 算法是解决给定的加权图中顶点间的最短路径的一种算法，可以正确处理有向图或负权的最短路径问题，同时也被用于计算有向图的传递闭包。是一种类似于动态规划思想的算法，稠密图效果最佳，边权可正可负。只需要三次循环，但也恰恰是由于它属于循环结构所以 Floyd 算法适合做节点数量小而且稠密的图，能够一次输出所有节点对之间的最短路径。实现 Floyd 算法的代码如下所示：

```
function [dist]=myfloyd(a,start,end)
n=size(a,1);path=zeros(n);
for k=1:n
        for i=1:n
                for j=1:n
                        if a(i,j)>a(i,k)+a(k,j)
                                a(i,j)=a(i,k)+a(k,j);
                                path(i,j)=k;
                        end
                end
        end
end
dist=a(start,end);
```

笔记

```
14 end
```

如果想分析具体的经过路径，可以观察 path 矩阵即可。path 矩阵中第 (i,j) 项是一个节点编号 k，表示从 i 到 j 的最短路径在 j 之前需要先到达节点 k。

Dijkstra 算法是基于贪心思想实现的最短路径算法，首先把起点到所有点的距离存下来找最短的，然后松弛一次再找出最短的，所谓的松弛操作就是，遍历一遍看通过刚刚找到的距离最短的点作为中转站会不会更近，如果更近了就更新距离，这样把所有的点找遍之后就存下了起点到其他所有点的最短距离。Dijkstra 算法是一种典型的单源最短路径算法，用于计算一个节点到其他所有节点的最短路径。主要特点是以起始点为中心向外层层扩展，直到扩展到终点为止。

可以看下面这个例子，构造一个有向图的邻接矩阵并求解从节点 1 出发到每个节点的最短路径长度以及从节点 1 到每个节点的最短路径如下。

```
G = [0,12,inf,inf,inf,16;
     12,0,10,inf,inf,7;
     inf,10,0,3,5,6;
     inf,inf,3,0,4,inf;
     inf,inf,5,4,0,2;
     16,7,6,inf,2,0];
[n,m]=size(G);
startp = 1; %起点
[D,p] = ShortestPath_DIJ(G,startp);
for endp = 1:n %终点
    if startp==endp
        fprintf('%d自身无需路径',startp);
    elseif ~D(endp)
        fprintf('%d %d 之间没有路径', startp,endp);
    else
        fprintf('%d %d 之间的最短路径:\n', startp,endp);
        path = endp;
        while path(end) ~= v0
            path = [path, p(path(end))];
        end
        path = fliplr(path);
        %打印出路径
        for i=1:length(path);
            fprintf(' %d ', path(i));
            if i ~= length(path)
```

笔记

```
                fprintf(" --> ");
            end
        end
    end
    fprintf(" 距离为%d\n",D(endp));
end
function  [D,path] = ShortestPath_DIJ(G,v0)
        [n,m] = size(G);
        D = G(v0,:);
        final = zeros(n,1); %初始化集合用于暂存
    final(v0)=1;
        path = v0*(D ~= inf); %如果j和v0之间的距离inf，则path(j)不可达;
        for i = 2:n
                min_iv = inf;  %min_iv将会存放在第i次选择的节点v的距离
                for w=1:n
            %更新
                        if ~final(w) && D(w) < min_iv
                                    v = w;
                    min_iv = D(w);
                        end
                end
                final(v) = 1;
                for w = 1:n
            %试探性加边
                        if ~final(w) && min_iv+G(v,w)<D(w)
                                D(w) = min_iv + G(v,w); %有动态规划的影子
                                path(w) = v;
                        end
                end
        end
end
%最终输出包括形如1 --> 6 --> 5 --> 4的路径，还有路径长度
```

一般而言，这两种方法都比较适用于无权图或权值非负的图，当图比较稠密时 Dijkstra 算法的计算代价更小一些；节点数量不多时 Floyd 算法更好。不过既然已经有函数可以调用了，学会怎么去用这几个函数也就够了。除了这两种方法，DFS 经过改良也可以用来做无权图的最短路径。此外 A* 算法作为 SLAM 技术中的热潮算法，也被广泛用于最短路径规划。

6.8 图论中的 TSP 与 VRP 问题

TSP 问题的背景源自一个古老的问题，即旅行商难题：假设有一个旅行商人要拜访 n 个城市，他必须选择所要走的路径，路径的限制是每个城市只能拜访一次，而且最后要回到原来出发的城市。路径的选择目标是要求得的路径路程为所有路径之中的最小值。TSP 问题是图论领域一个典型的 NP 难问题，目前进展迅速的想法是使用以进化计算与群体智能方法为代表的启发式方法求解精确解或者近似解。下一节我们就会讲到。

下面以一个简单的例子展开，看看动态规划思想如何求解 TSP 问题。由于动态规划是对 2n-1 的空间进行搜索，搜索空间比较大，这里测试样例仅使用 4 个点组成的有向图进行描述。邻接矩阵为 c=[0 2 4 6;4 0 2 5;8 3 0 1;3 6 9 0]。

```
V=[0 0 0;0 0 1;0 0 2;0 0 3;0 1 2;0 1 3;0 2 3;1 2 3];
%这是方案集合，分别代表{{},{1},{2},{3},{1,2},{1,3},{2,3},{1,2,3}}一共八
c=[0 2 4 6;4 0 2 5;8 3 0 1;3 6 9 0];  %邻接矩阵
n=size(c,1);
z=2^(n-1);
for i = 1:n
    for j = 1:z
        d(i,j)=inf;  %初始化表格，顶点对选择方案
    end
end
y=tsp_dp(d,c,V)
disp('最短路径是：')
y(1,z)  %从顶点0出发到最后{1,2,3}都被添加进来
function y=tsp_dp(d,c,V)
    %动态规划求tsp问题的最优解
    d(:,1)=c(:,1);
    n=size(c,1);
    z=2^(n-1);
    for j = 2:z-1
        for i=2:n
            if ~sum(i-1 == V(j,:)) %i-1不在V中
                for k = 1:3
                    index=tsp_append(V(j,k),V(j,:),V);
                    if (V(j,k) ~=0) && ((c(i,V(j,k)+1) +d(V(j,k)+1,index)) < d
                        d(i,j) = c(i,V(j,k)+1) +d(V(j,k)+1,index);  %更新表格
                    end
```

```
                end
            end
        end
    end
    for k = 1:3
        index=tsp_append(V(j,k),V(j,:),V);
        if V(z,k) ~= 0 && (c(1,V(z,k)+1) + d(V(z,k)+1,index) < d(1,z))
            d(1,z) = c(1,V(z,k)+1) + d(V(z,k)+1,index);
        end
    end
    y=d;
end
function index = tsp_append(stat,array,V)
    %对去掉当前已走过城市后，路径方案的下标检索
for i = 1:length(array)
        if stat == array(i)
            array(i) = 0;
        end
    end
    count1=zeros(length(array)+1,1);
    count2=zeros(length(array)+1,1);
    for i = 1:length(count1)
        count1(i)=sum(array == i-1);
    end
    for j = 1:7
        for i = 1:length(count2)
            count2(i)=sum(V(j,:) == i-1);
        end
        if sum(count1 == count2) == length(count1)
            index = j;
        end
    end
end
```

最终解得在网络图中的最短路径保存在起点对应行的最后一位，也就是 y(1,z) 当中。结果为 8。利用动态规划解网络 TSP 的核心想法就是分支，当我从起点开始可以直达多个其他节点时，对于某一条边，添加到回路圈和不添加到回路圈会对子问题产生怎样的影响。如果添加这一条边

剩下的子问题最短回路会是多长；不添加的话子问题的最短回路又是多长呢？所以在函数中通过 tsp_dp_left 函数进行路径变更路径。当然，把这个问题抽象为 0-1 规划也是可以的。

车辆路径问题最早由 Dantzig 和 Ramser 于 1959 年首次提出，是运筹学中一个经典问题。VRP 问题主要研究物流配送中的车辆路径规划问题，是当今物流行业中的基础问题。在 VRP 问题中，假设有一个供求关系系统，车辆从仓库取货，配送到若干个顾客处。车辆受到载重量的约束，需要组织适当的行车路线，在顾客的需求得到满足的基础上，使代价函数最小。代价函数根据问题不同而不同，常见的有车辆总运行时间最小，车辆总运行路径最短等。基本的问题形式为：假设有 N 辆车，都从原点出发，每辆车访问一些点后回到原点，要求所有的点都要被访问到，求最短的车辆行驶距离或最少需要的车辆数。

在 VRP 问题中，假设有一个供求关系系统，车辆从仓库取货，配送到若干个顾客处。车辆受到载重量的约束，需要组织适当的行车路线，在顾客的需求得到满足的基础上，使代价函数最小。代价函数根据问题不同而不同，常见的有车辆总运行时间最小，车辆总运行路径最短等。

假设现在有多辆运载车运输医疗物资到医院，行驶能力、速度、续航等相同。定义 j,k 为需要接收物资的医院，e 为车辆编号，C 为城市的集合。Djk 为从 j 到 k 的行驶距离，dj 为医院 j 的缺口需求量，而车辆的最大载重量为 Ye。可以列出如下优化模型：

$$
\begin{aligned}
&\min \sum_{e\in E}\sum_{j\in C}\sum_{k\in C} D_{jk}z_{ejk} \\
s.t.\ &\begin{cases}
\sum\limits_{e\in E}\sum\limits_{j\in C} z_{ejk}=1, \forall j,k\in C\backslash\{0\} \\
\sum\limits_{j\in C\backslash\{0\}}\sum\limits_{k\in C\backslash\{0\}}^{k} d_k z_{ejk}\le Y_e, \forall i,e\in E \\
\sum\limits_{j\in C} z_{e0k}=1, \forall e\in E \\
\sum\limits_{j\in C\backslash\{0\}} z_{ejk}-\sum\limits_{k\in C\backslash\{0\}}^{k} z_{ejk}=0, \forall e\in E \\
\sum\limits_{j\in C_i} z_{ejk}=1, \forall e\in E \\
z_{ejk}\in\{0,1\}, \forall e\in E, \forall j,k\in C
\end{cases}
\end{aligned}
\tag{6.18}
$$

本质上这一模型优化的是每辆车在各自回路上的路程和运载量乘积的和。用 z 表示复杂网络上节点 j 到节点 k 的路径上车辆 e 是否运送。本质上这一个成分是离散优化。然后，每一个医院都得有一辆车送；每一辆车送的量不能超载；每辆车最后必须返回配送中心。决策变量 z 是 0-1 变量，但是规模较大，所以与 TSP 类似，它也是用启发式算法解居多。除此以外，还存在多仓库 VRP 等多种问题，有兴趣的读者可以查阅资料了解。

6.9 遗传算法

人类总是能够从自然界获取很多灵感。通过蝙蝠的回声定位，我们发明了雷达；通过鱼的游动，我们发明了潜艇；通过鲨鱼的皮肤，我们发明了潜水服……而遗传算法同样是基于生物原理得到灵感。这个灵感，来自于孟德尔遗传定律和达尔文自然选择学说。

遗传算法是 J.H.Holland 在 1975 年提出，模拟达尔文的遗传选择和自然淘汰的进化过程。这一算法被誉为智能优化算法“根源中的根源”。它被广泛应用于大规模的优化问题，例如非线性规划，离散优化，TSP 问题，VRP 问题，车间调度问题等。

孟德尔在他的遗传学说当中揭示了遗传过程中染色体的一些变化过程：复制，交叉，突变等。而微观的遗传物质的变化影响到了种群在自然界的发展，因为生物的发展与进化主要的过程就是三个：遗传，变异和选择。只有适应环境的竞争力强的生物才能存活下来，不适应者就会消亡。而遗传算法就是借鉴了这一点，通过遗传和变异生成一批候选解，然后在逐代进化的过程中一步步逼近最优解。这里补充几个概念定义：

- 染色体：遗传物质的主要载体，是多个遗传因子的集合。
- 基因：遗传操作的最小单元，以一定排列方式构成染色体。
- 个体：染色体带有特征的实体。
- 种群：多个个体组成群体，进化之初的原始群体被称为初始种群。
- 适应度：用于评价个体适应环境程度的函数值。
- 编码：二进制或十进制去表示研究问题的过程。
- 解码：将二进制或十进制还原为原始问题的过程。
- 选择：以一定概率从种群中选择若干个体的过程，选择的基准方法有很多，常见的有适应度比例法、期望值法、轮盘赌法等。
- 交叉：将两个染色体换组的操作，又称重组。
- 变异：遗传因子按一定概率变化的操作。

遗传算法借鉴了生物学的概念，首先需要对问题进行编码，通常是将函数编码为二进制代码以后，随机产生初始种群作为初始解。随后是遗传算法的核心操作之一——选择，通常选择首先要计算出个体的适应度，根据适应度不同来采取不同选择方法进行选择，常用方法有适应度比例法、期望值法、排位次法、轮盘赌法等。在自然界中，基因的突变与染色体的交叉组合是常见现象，这里也需要在选择以后按照一定的概率发生突变和组合。不断重复上述操作直到收敛，得到的解即最优。遗传算法基本思想如图6.7所示：

其实遗传算法说起来这么复杂，实际上思想本质上还是一个搜索。从一堆可行解里面搜索最优解，没有方向漫无目的的检索叫暴力搜索，有方向的才叫启发式搜索。遗传算法的方向就是进化。

```
1 % function [path,p0]=YCTSP
2 % (d,w,g,by)
```

笔记

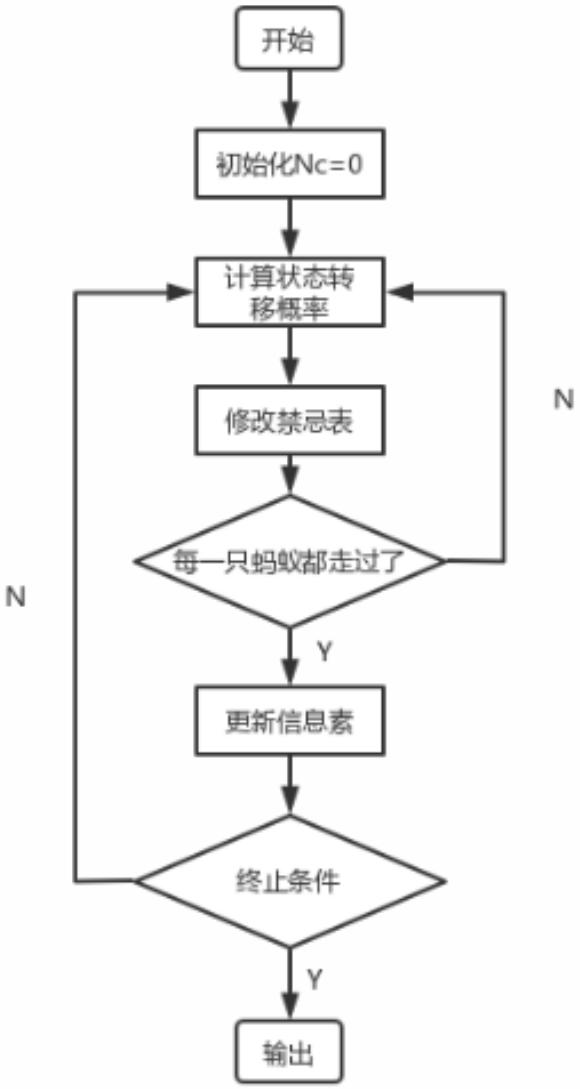


图 6.7: 遗传算法的基本思想

```
tic  %计时开始
clc,clear, close all
sj0=load('city.txt');
n1=length(sj0(:,1));
d=zeros(n1);
for i=1:n1
    for j=1:n1
        d(i,j)=sqrt((sj0(i,1)-sj0(j,1))^2+(sj0(i,2)-sj0(j,2))^2);
    end
end
d=d+d';  w=100;  g=100;  %w为种群的个数，g为进化的代数
```

笔记

```
[N,~]=size(d);
A=zeros(N,w);
B=zeros(1,w);
by=0.1;
TU1=zeros(1,g);
%随机生成子代并计算其总路程：

for j=1:w
    A(:,j)=randperm(N)';
    B(j)=jisuan(d,N,A(:,j));
end
%初始化种群最优解

for i=1:w
    for tt=1:1000
        flag=0;
    for j=1:N-1
        for ol=j+1:N
            D=A(:,i);
            ddd=D(j);
            D(j)=D(ol);
            D(ol)=ddd;
            P=jisuan(d,N,D);
    if B(i)>P
A(:,i)=D;
B(i)=P;
flag=1;
    end
        end
    end
if  flag==0
    break
end
    end
end
p0=min(B);
```

```
cp=find(B==p0);
path=A(:,cp(1));

for k=1:g

for j=1:w
    %自我学习如下:
    if rand>by
        D=A(:,j);
        P=jisuan(d,N,D);
        a=randi([1 N]);
        b=randi([1 N]);
        c=max(a,b);
        e=min(a,b);
        ddd=A(e,j);
        A(e,j)=A(c,j);
        A(c,j)=ddd;
        B(j)=jisuan(d,N,A(:,j));
        if B(j)<p0
            p0=B(j);
            path=A(:,j);
        end
        if B(j)>P
            if rand>by
                A(:,j)=D;
                B(j)=P;
                if B(j)<p0
            p0=B(j);
            path=A(:,j);
                end
            end
        end
    else
        D=A(:,j);
        P=jisuan(d,N,D);
        a=randi([1 N]);
```

```
        b=randi([1 N]);
        c=max(a,b);
        e=min(a,b);
        A(e:c,j)=A(c:-1:e,j);
        B(j)=jisuan(d,N,A(:,j));
        if B(j)<p0
            p0=B(j);
            path=A(:,j);
        end
        if B(j)>P
            if rand>by
                A(:,j)=D;
                B(j)=P;
                if B(j)<p0
                    p0=B(j);
                    path=A(:,j);
                end
            end
        end
    end
    %偷师学艺如下:
    for i=1:w
        D=A(:,j);
        P=jisuan(d,N,D);
        a=randi([1 N]);
        e=find(A(:,j)==A(a,i));
        ddd=A(e,j);
        A(e,j)=A(a,j);
        A(a,j)=ddd;
        B(j)=jisuan(d,N,A(:,j));
        if B(j)<p0
            p0=B(j);
            path=A(:,j);
        end
        if B(j)>P
            if rand>by
```

```
                    A(:,j)=D;
                    B(j)=P;
                    if B(j)<p0
                        p0=B(j);
                        path=A(:,j);
                    end
                end
            end
        end
    end
%追寻第一过程：
for i=1:w
    a=randi([1 N]);
    D=A(:,i);
    P=jisuan(d,N,D);
    e=find(A(:,i)==path(a));
    ddd=A(e,i);
    A(e,i)=A(a,i);
    A(a,i)=ddd;
    if B(i)<p0
        p0=B(i);
        path=A(:,i);
    end
    if B(i)>P
        if rand>by
            A(:,i)=D;
            B(i)=P;
            if B(i)<p0
                p0=B(i);
                path=A(:,i);
            end
        end
    end
end
%超越过程
for tt=1:N
```

```
158     flag=0;
159     for i=1:N-1
160         for j=1:N
161             D=path;
162             ddd=D(j);
163             D(j)=D(o1);
164             D(o1)=ddd;
165             P=jisuan(d,N,D);
166             if p0>P
167                 path=D;
168                 p0=P;
169                 flag=1;
170             end
171         end
172     end
173     if flag==0
174         break
175     end
176 end
177 TU1(k)=p0;
178 end
179 path,p0
180 dpath=[path;path(1)];
181 xx=sj0(dpath,1);yy=sj0(dpath,2);
182 figure(1)
183 plot([1:g],TU1)
184 figure(2)
185 plot(xx,yy,'-o') %画出路径
186 toc
187 function zhi=jisuan(d,N,x)
188 y=x([2:N 1]);
189 zhi=0;
190 for i=1:N
191     zhi=zhi+d(x(i),y(i));
192 end
193 end
```

笔记

6.10 粒子群算法

粒子群 (Particle Swarm Optimization) 算法是 Kennedy 和 Eberhart 在 1995 年提出的算法，它最初是由于研究模拟鸟群捕食的智能行为，被用于连续变量的最优化问题。但后来发现通过二进制编码等方式 PSO 也可以用于解离散优化例如 TSP 等问题。

不知各位是否会注意到，鸟群例如大雁在飞行的时候它们的飞行方向除了受到环境的影响，还会受到其他大雁的影响，从而使群体中每一只大雁的飞行轨迹都整齐划一。而当一只鸟飞离鸟群去寻找栖息地的时候，它不仅要考虑自身运动方向和周围环境，还会从其他优秀的个体的飞行轨迹去模仿学习经验（当然它自己也可能被其它鸟模仿）。这一过程揭示了鸟群运动过程中的两类重要的知识：自我智慧和群体智慧。

现在假设一群鸟在一块有食物的区域内，它们都瞎了都不知道食物在哪里，但知道当前位置与食物的距离。最简单的方法就是搜寻目前离食物最近的鸟的区域。那我现在把这个区域看做是函数的搜索空间，每个鸟被抽象为一个粒子（物理意义上的质点），每个粒子有一个适应度和速度描述飞行方向和距离。粒子通过分析当前最优粒子在解空间中的运动过程去搜索最优解。设微粒群体规模为 N，其中每个微粒在 D 维空间中的坐标位置可表示为 Xi=(xi1,xi2,⋯,xiD)，微粒 i 的速度定义为每次迭代中微粒移动的距离，表示为 Vi=(vi1,vi2,⋯,viD)，Pi 表示微粒 i 所经历的最好位置，Pg 为群体中所有微粒所经过的最好位置，则微粒 i 在第 d 维子空间中的飞行速度 vid 根据下式进行调整：

$$V_{id} = w \cdot V_{id} + c_1 \cdot Rand() \cdot (p_{id} - x_{id}) + c_2 \cdot Rand() \cdot (p_{gd} - x_{id}) \tag{6.19}$$

在这个过程中，每次运动的时间间隔被视作单位 1，那么速度实际上也可以用于描述下一个时间间隔的移动方向和移动距离。

$$x_{id} = x_{id} + V_{id} \tag{6.20}$$

第一项为微粒先前速度乘一个权值进行加速，表示微粒对当前自身速度状态的信任，依据自身的速度进行惯性运动，因此称这个权值为惯性权值第二项为微粒当前位置与自身最优位置之间的距离，为认知部分，表示微粒本身的思考，即微粒的运动来源于自己经验的部分第三项为微粒当前位置与群体最优位置之间的距离，为社会部分，表示微粒间的信息共享与相互合作，即微粒的运动中来源于群体中其他微粒经验的部分粒子群算法基本流程：

1. 初始化: 随机初始化每一微粒的位置和速度。

2. 评估：依据适应度函数计算每个微粒的适应度值，以作为判断每一微粒之好坏。

3. 寻找个体最优解：找出每一微粒到目前为止的搜寻过程中最佳解，这个最佳解称为 Pbest。

4. 寻找群体最优解: 找出所有微粒到目前为止所搜寻到的整体最佳解, 此最佳解称之为 Gbest。

5. 更新每一微粒的速度与位置。

笔记

6. 回到步骤 2 继续执行，直到获得一个令人满意的结果或符合终止条件为止。

粒子群算法的工作流程如图6.8所示：

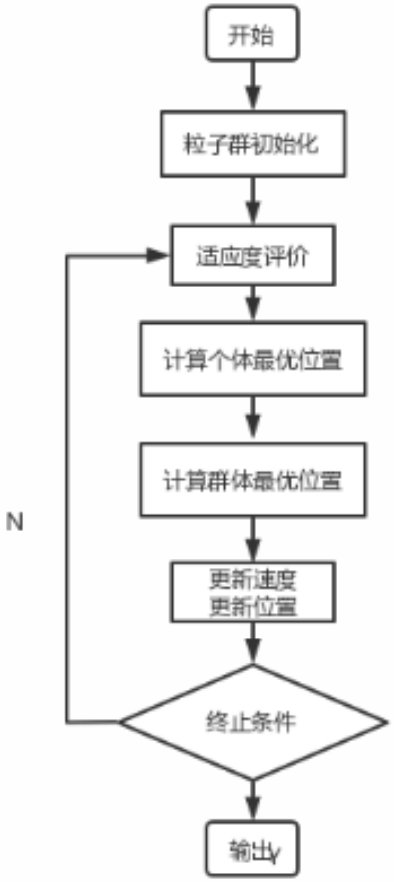


图 6.8: 粒子群算法的计算流程

```
clear; clc
%% 绘制函数的图形
x = -3:0.01:3;
y = 11*sin(x) + 7*cos(5*x);
figure(1)
plot(x,y,'b-')
title('y = 11*sin(x) + 7*cos(5*x)')
hold on  % 不关闭图形，继续在上面画图

%% 粒子群算法中的预设参数（参数的设置不是固定的，可以适当修改）
n = 10; % 粒子数量
narvs = 1; % 变量个数
```

笔记

```
c1 = 2;  % 每个粒子的个体学习因子，也称为个体加速常数
c2 = 2;  % 每个粒子的社会学习因子，也称为社会加速常数
w = 0.9;  % 惯性权重
K = 50;  % 迭代的次数
vmax = 1.2; % 粒子的最大速度
x_lb = -3; % x的下界
x_ub = 3; % x的上界

%% 初始化粒子的位置和速度
x = zeros(n,narvs);
for i = 1: narvs
    x(:,i) = x_lb(i) + (x_ub(i)-x_lb(i))*rand(n,1);     % 随机初始化粒子所在的
end
v = -vmax + 2*vmax .* rand(n,narvs);  % 随机初始化粒子的速度（这里我们设置为[-

%% 计算适应度
fit = zeros(n,1);  % 初始化这n个粒子的适应度全为0
for i = 1:n  % 循环整个粒子群，计算每一个粒子的适应度
    fit(i) = fun1(x(i,:));    % 调用fun1函数来计算适应度（fun1函数为目标函数的
end
pbest = x;   % 初始化这n个粒子迄今为止找到的最佳位置（是一个n*narvs的向量）
ind = find(fit == max(fit), 1);  % 找到适应度最大的那个粒子的下标
gbest = x(ind,:);  % 定义所有粒子迄今为止找到的最佳位置（是一个1*narvs的向量）

%% 在图上标上这n个粒子的位置用于演示
h = scatter(x,fit,80,'*r');  % scatter是绘制二维散点图的函数,80是我设置的散点

%% 迭代K次来更新速度与位置
fitnessbest = ones(K,1);  % 初始化每次迭代得到的最佳的适应度
for d = 1:K  % 开始迭代，一共迭代K次
    for i = 1:n   % 依次更新第i个粒子的速度与位置
        v(i,:) = w*v(i,:) + c1*rand(1)*(pbest(i,:) - x(i,:)) + c2*rand(1)*(gb
% 更新第i个粒子的速度
        % 如果粒子的速度超过了最大速度限制，就对其进行调整
        for j = 1: narvs
            if v(i,j) < -vmax(j)
```

```
                v(i,j) = -vmax(j);
            elseif v(i,j) > vmax(j)
                v(i,j) = vmax(j);
            end
        end
        x(i,:) = x(i,:) + v(i,:); % 更新第i个粒子的位置
        % 如果粒子的位置超出了定义域，就对其进行调整
        for j = 1: narvs
            if x(i,j) < x_lb(j)
                x(i,j) = x_lb(j);
            elseif x(i,j) > x_ub(j)
                x(i,j) = x_ub(j);
            end
        end
        fit(i) = fun1(x(i,:));  % 重新计算第i个粒子的适应度
        if fit(i) > fun1(pbest(i,:))   % 如果第i个粒子的适应度大于这个粒子迄今
            pbest(i,:) = x(i,:);   % 那就更新第i个粒子迄今为止找到的最佳位置
        end
        if  fit(i) > fun1(gbest)  % 如果第i个粒子的适应度大于所有的粒子迄今为
            gbest = pbest(i,:);   % 那就更新所有粒子迄今为止找到的最佳位置
        end
    end
    fitnessbest(d) = fun1(gbest);  % 更新第d次迭代得到的最佳的适应度
    pause(0.1)  % 暂停0.1s
    h.XData = x;  % 更新散点图句柄的x轴的数据（此时粒子的位置在图上发生了变化
    h.YData = fit; % 更新散点图句柄的y轴的数据（此时粒子的位置在图上发生了变化
end

figure(2)
plot(fitnessbest)  % 绘制出每次迭代最佳适应度的变化图
xlabel('迭代次数');
disp('最佳的位置是：'); disp(gbest)
disp('此时最优值是：'); disp(fun1(gbest))
```

6.11 模拟退火算法

模拟退火算法由 Kirkpatrick 等提出，能有效的解决局部最优解问题。它不同于前面基于生物的进化和群智能，它是基于物理学定律提出的方法。模拟退火算法 (Simulated Annealing, SA) 的思想借鉴于固体的退火原理，当固体的温度很高的时候，内能比较大，固体的内部粒子处于快速无序运动，当温度慢慢降低的过程中，固体的内能减小，粒子的慢慢趋于有序，最终，当固体处于常温时，内能达到最小，此时，粒子最为稳定。模拟退火算法便是基于这样的原理设计而成。模拟退火算法来源于晶体冷却的过程，如果固体不处于最低能量状态，给固体加热再冷却，随着温度缓慢下降，固体中的原子按照一定形状排列，形成高密度、低能量的有规则晶体，对应于算法中的全局最优解。而如果温度下降过快，可能导致原子缺少足够的时间排列成晶体的结构，结果产生了具有较高能量的非晶体，这就是局部最优解。因此就可以根据退火的过程，给其在增加一点能量，然后在冷却，如果增加能量，跳出了局部最优解，本次退火就是成功的。模拟退火算法包含两个部分即 Metropolis 准则和退火过程。Metropolis 准则以概率来接受新状态，而不是使用完全确定的规则，称为 Metropolis 准则，计算量较低。从某一个解到新解本质上是衡量其能量变化，若能量向递减的方向跃迁则接受这一次迭代，若能量反而增大，并不是一定拒绝而是以一定的采样概率接受。这一概率值满足 Metropolis 定义：

$$P = e^{-\frac{E(n+1)-E(n)}{T}} \tag{6.21}$$

直接使用 Metropolis 算法可能会导致寻优速度太慢，以至于无法实际使用，为了确保在有限的时间收敛，必须设定控制算法收敛的参数，在上面的公式中，可以调节的参数就是 T，T 如果过大，就会导致退火太快，达到局部最优值就会结束迭代，如果取值较小，则计算时间会增加，实际应用中采用退火温度表，在退火初期采用较大的 T 值，随着退火的进行，逐步降低。模拟退火的过程如图6.9所示：

速度上模拟退火和粒子群都很快，但模拟退火略快一些，比遗传更快，蚁群的速度是最慢的。但粒子群求解大规模函数极值的时候容易碰到边界陷入的情况。模拟退火则相对比较稳定一些。

```
clear; clc
%% 绘制函数的图形
x = -3:0.1:3;
y = 11*sin(x) + 7*cos(5*x);
figure
plot(x,y,'b-')
hold on  % 不关闭图形，继续在上面画图

%% 参数初始化
narvs = 1; % 变量个数
T0 = 100;    % 初始温度
```

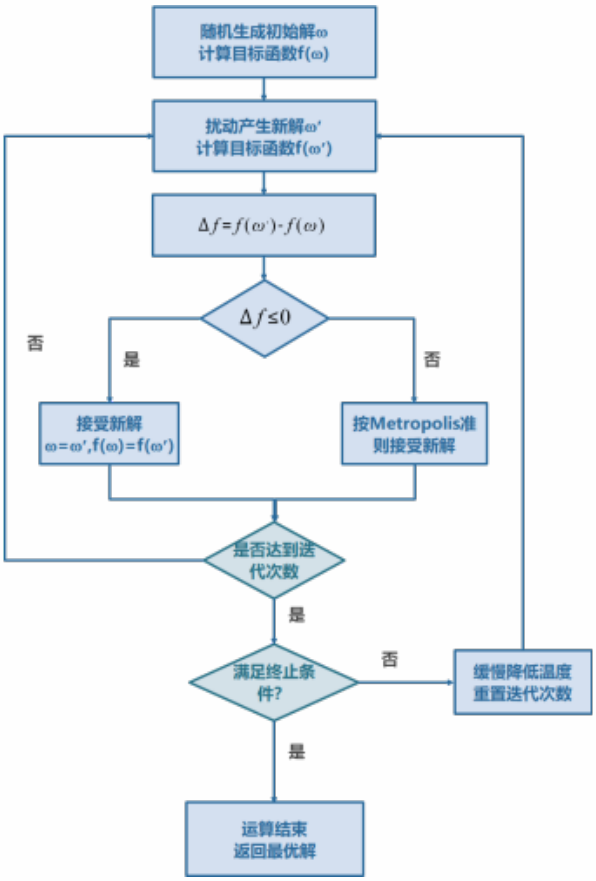


图 6.9: 模拟退火算法的流程图

```
T = T0; % 迭代中温度会发生改变，第一次迭代时温度就是T0
maxgen = 200;  % 最大迭代次数
Lk = 100;  % 每个温度下的迭代次数
alfa = 0.95;  % 温度衰减系数
x_lb = -3; % x的下界
x_ub = 3; % x的上界

%%  随机生成一个初始解
x0 = zeros(1,narvs);
for i = 1: narvs
    x0(i) = x_lb(i) + (x_ub(i)-x_lb(i))*rand(1);
end
y0 = fun41(x0); % 计算当前解的函数值
h = scatter(x0,y0,'*r');  % scatter是绘制二维散点图的函数（这里返回h是为了得到

%% 定义一些保存中间过程的量，方便输出结果和画图
```

```
max_y = y0;      % 初始化找到的最佳的解对应的函数值为y0
MAXY = zeros(maxgen,1); % 记录每一次外层循环结束后找到的max_y（方便画图）

%% 模拟退火过程
for iter = 1 : maxgen  % 外循环，我这里采用的是指定最大迭代次数
    for i = 1 : Lk  % 内循环，在每个温度下开始迭代
        y = randn(1,narvs);  % 生成1行narvs列的N(0,1)随机数
        z = y / sqrt(sum(y.^2)); % 根据新解的产生规则计算z
        x_new = x0 + z*T; % 根据新解的产生规则计算x_new的值
        % 如果这个新解的位置超出了定义域，就对其进行调整
        for j = 1: narvs
            if x_new(j) < x_lb(j)
                r = rand(1);
                x_new(j) = r*x_lb(j)+(1-r)*x0(j);
            elseif x_new(j) > x_ub(j)
                r = rand(1);
                x_new(j) = r*x_ub(j)+(1-r)*x0(j);
            end
        end
        x1 = x_new;     % 将调整后的x_new赋值给新解x1
        y1 = fun41(x1);  % 计算新解的函数值
        if y1 > y0     % 如果新解函数值大于当前解的函数值
            x0 = x1; % 更新当前解为新解
            y0 = y1;
        else
            p = exp(-(y0 - y1)/T); % 根据Metropolis准则计算一个概率
            if rand(1) < p   % 生成一个随机数和这个概率比较，如果该随机数小于
                x0 = x1; % 更新当前解为新解
                y0 = y1;
            end
        end
        % 判断是否要更新找到的最佳的解
        if y0 > max_y  % 如果当前解更好，则对其进行更新
            max_y = y0;  % 更新最大的y
            best_x = x0;  % 更新找到的最好的x
        end
```

```
64      end
65      MAXY(iter) = max_y; % 保存本轮外循环结束后找到的最大的y
66      T = alfa*T;    % 温度下降
67      pause(0.01)  % 暂停一段时间(单位：秒)后再接着画图
68      h.XData = x0;  % 更新散点图句柄的x轴的数据（此时解的位置在图上发生了变化）
69      h.YData = fun41(x0); % 更新散点图句柄的y轴的数据（此时解的位置在图上发生了
70  end
71
72  disp('最佳的位置是：'); disp(best_x)
73  disp('此时最优值是：'); disp(max_y)
74
75  pause(0.5)
76  h.XData = [];   h.YData = [];  % 将原来的散点删除
77  scatter(best_x,max_y,'*r');  % 在最大值处重新标上散点
78  title(['模拟退火找到的最大值为', num2str(max_y)])  % 加上图的标题
79
80  %% 画出每次迭代后找到的最大y的图形
81  figure
82  plot(1:maxgen,MAXY,'b-');
83  xlabel('迭代次数');
84  ylabel('y的值');
```

模拟退火算法稍作改进，可以改出一个量子模拟退火算法（这绝对不是诈骗性质的算法，量子模拟退火是解决 QUBO 问题的很好的方法），高度适配量子计算机硬件。

第七章　基于 Baltamatica 的回归拟合

本章我们学习的重点内容是 Baltamatica 的回归拟合问题。从高中阶段我们就接触过线性回归，现在再重提线性回归是为了将线性回归问题扩展到一般函数的回归拟合问题，并试着编写最小二乘法实现自己的任意函数拟合器。

7.1　线性回归与正则化

一元线性回归方程的系数公式叫做最小二乘公式，那各位有想过它为什么被称为最小二乘公式吗？这个公式又为什么正确？我们今天就来一探究竟：

我们拿一元线性回归为例。线性回归的目的无非是想找一个最能拟合上数据集的一个直线，这条直线是定义在一个平面直角坐标系中的。那么，怎么定义“最能拟合上”这么一个程度呢？在每一个横坐标下，我这条直线预测的值也就有一个点，我们希望直线上的预测点和实际的数据点差异尽可能小。为了描述这种差异，我们可以用距离来描述。对于方程 $y = W^T X + b$，以均方误差作为损失函数，也就是实际值和预测值的偏差方差：

$$J(w,b) = \frac{1}{n}\sum_{i=1}^{n}(y_i - wx_i - b)^2 \tag{7.1}$$

如图7.2所示：

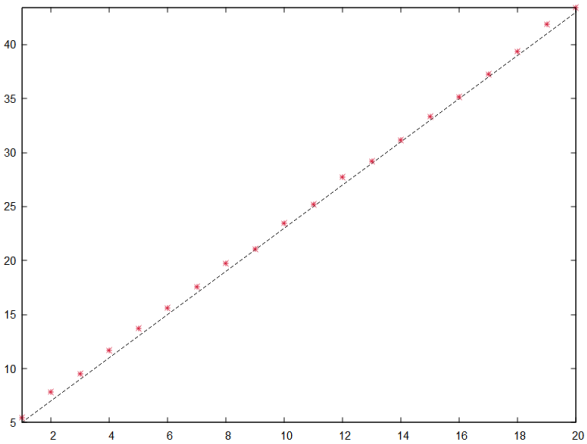


图 7.1: 线性回归示例图

用以拟合的一系列数据点都是已知的，所以这个函数是 n 个有关 w 和 b 的二次式求和而成，终归是二元二次函数。现在希望误差最小，所以对这个函数求极值也就是：

$$\begin{cases} \frac{\partial J}{\partial w} = 0 \\ \frac{\partial J}{\partial b} = 0 \end{cases} \tag{7.2}$$

对 b 的偏导为 0 很容易解得一个重要的性质：回归方程通过样本中心点。然后将其带入对 w 的偏导，我们才有了下面的公式：

$$\begin{cases} \hat{w} = \frac{\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2} \\ \hat{b} = \bar{y} - \hat{w}\bar{x} \end{cases} \tag{7.3}$$

可如果现在回归方程是多元的，形如：

$$y = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b \tag{7.4}$$

或者写成矩阵的形式：

$$y = W^T X + b \tag{7.5}$$

这样的问题我们用同样的方法，还是写出均方误差，然后对每个待估计参数分别求偏导就可以得到它们的结果了。同样的方法也适用于指数拟合、对数拟合、三角拟合等，这就是最小二乘法的来历。虽然理论上最小二乘都可以通过求偏导的方式求极值，但这个导数的符号解有可能很不好解。更多情况下我们会用梯度下降等方法去求误差函数的数值极值解。

不过好在，这是一个线性方程，可以用线性运算描述它。所以，对于这一矩阵方程，我们其实是有系数公式的：

$$(W|b) = (X^T X)^{-1} X^T Y \tag{7.6}$$

那么我们有没有可能把本来不应该具备线性关系的数据硬用线性套呢？如何知道整体的拟合情况和每个变量是否应该被添加进来呢？这个时候我们可以用假设检验的方法。在一个线性回归中，常数项与误差之间经过变换会服从一个 F 分布，可以用 F 检验；而每个变量的系数则可以经过变换形成 T 分布结构。若数据有 n 条，m 个变量（自变量 + 因变量），那么 F 分布的自由度为 F(m-1,n-m)，T 分布的自由度为 T(n-m)。可以参考如下实现代码：

```
[n,m]=size(data);
Y=data(:,end);
X=data(:,1:(end-1));
X=[ones(n,1),X];
beta=inv(X'*X)*(X'*Y);

p=zeros(m,1);
tss=(Y-mean(Y))'*(Y-mean(Y));
```

```
rss=Y'*Y-beta'*X'*Y;
R=(tss-rss)/tss;
F=((tss-rss)/(m-1))/(rss/(n-m));
%p(1)=1-fcdf(F,m-1,n-m);
%F累计分布函数，这个地方是我查的表
if F>4
        p(1)=0.01;
elseif F>2.735
        p(1)=0.05;
else
        p(1)=0.9;
end

C=diag(inv(X'*X));
sigma=sqrt(rss/(n-m));
t=zeros(m-1,1);
for i=1:(m-1)
    t(i)=beta(i+1)/(sqrt(C(i+1))*sigma);
    %p(i+1)=2*(1-tcdf(abs(t(i)),n-m));
    %查了T累计分布函数表
    if abs(t(i))>2.5
        p(i+1)=0.01;
    elseif abs(t(i))>2
            p(i+1)=0.05;
    else
            p(i+1)=0.9;
    end
end
```

练习应用

- 前面的过程中我们已经讨论了多元线性回归方程。那如果说我现在想做一个多项式方程，比如一元多项式方程 $y = x^2 - 4x + 3$，或者多元多项式方程 $y = x_1 + x_2 - 0.5x_2x_1 + 3$，类似于这两种形式的话，我应该如何修改函数？尝试给出一个思路（提示：可以从数据预处理的角度下手）

笔记

7.2 多项式拟合

我们从一元线性回归扩展开去，考虑用多项式进行回归拟合：

$$y_t = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n \tag{7.7}$$

在 Baltamatica 中已经内置了 polyfit 函数可以进行多项式拟合。polyfit 函数是通过调用函数的方式，将自变量和对应的因变量进行多项式进行拟合的方式。其调用格式形如：

```
x=1:10;
y=0.03*x.^3-x+10*rand(1,10);
polyfit(x,y,2)
```

这一调用模式下返回的是一个数组 [0.5870,-4.3028,10.4829], 表示了一个多项式 $0.5870x^2 - 4.3028x + 10.4829$。同理，如果我们希望用 n 阶多项式，只需要把 polyfit 的第三项改为 n 就可以了。返回的数组维度为 n+1，数组首项表示了最高项的系数。

那得到这样一个多项式又如何还原回来呢？Baltamatica 提供了一个比较好的方法就是 polyval，只需要把这个数组作为系数输入，并输入对应的自变量即可反馈对应的因变量。例如我们可以写下如下代码：

```
a=polyfit(x,y,2)
y1=polyval(a,x)
```

那么我们如何评价几次回归的效果最好呢？这涉及到回归指标评价问题。常见的回归拟合效果评价指标有不少，例如均方误差、均方根误差、MAE 等，这里介绍几种常见的效果评价方案：

- 第一种方案是用均方误差描述。在线性回归中我们介绍过最小二乘法，那么这个地方的均方误差就很自然地写出了它的计算公式：

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{7.8}$$

- 第二种方案是为了防止本身 y 值比较小，而 MSE 又做了一次平方，所以对 MSE 开方得到了 RMSE；

- 第三种方案是使用偏差平均值而非平方后开根号：

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{7.9}$$

- 第四种方案是最常用的一个，又叫拟合优度法。它的基本原理不局限于皮尔逊相关系数，而是比较残差平方和与总平方和的问题：

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{7.10}$$

大家请自行思考，如何书写这几个指标对应的函数。

进一步地，我更想为大家声明一个概念，就是欠拟合与过拟合。我们的拟合不是希望函数完完本本的通过每一个数据点，那对我们后面的工作没有意义。我们应当容许有一定的偏差，或者说在训练的时候要允许它有一定的容错空间，这样它预测未来的趋势才能更加准确。比如说，同样对于一组数据，我们可以看看下面三个拟合效果：

```
clc;clear;
x=1:20;
y=0.03*x.^3-x+10*rand(1,20);
a=polyfit(x(1:15),y(1:15),1);
b=polyfit(x(1:15),y(1:15),2);
c=polyfit(x(1:15),y(1:15),7);
subplot(1,3,1);y1=polyval(a,x);
plot(x,y,'r-');hold on;plot(x,y1,'k--');
subplot(1,3,2);y2=polyval(b,x);
plot(x,y,'r-');hold on;plot(x,y2,'k--');
subplot(1,3,3);y3=polyval(c,x);
plot(x,y,'r-');hold on;plot(x,y3,'k--');
```

最左侧的拟合是使用线性函数进行的，可以看到，函数在前面一些步长上拟合效果就不行，后面的趋势只会越差越大。中间的函数用二次式，拟合效果是最好的。后面第三个函数可以看到

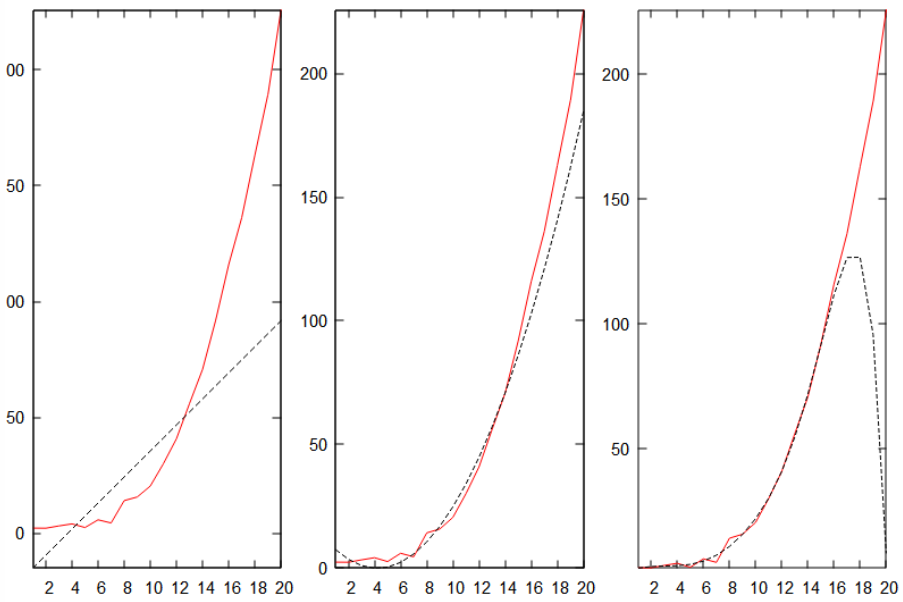


图 7.2: 何为过拟合

前面的步长上面几乎是能做到重合，都没什么偏差，但由于用了过高的多项式导致后面推理阶段偏差拉的很大。

模型并非越复杂越好，拟合也不是误差要做到绝对的小，容错率极度的低，现在的低错误很可能造成推理阶段的高偏差。取乎其中，这是建模人的一条核心法则。

练习应用

- 这一节的练习题就只有一道，假设 a 和 b 分别是两个 polyfit 得到的 m 阶多项式和 n 阶多项式，它们都以数组的形式存储。现在我们希望将多项式 a 和多项式 b 相乘得到新的多项式 c，那么这个 c 用数组应该如何表示？请编写对应的函数 polymul(a,b) 并阐述逻辑。

7.3 最小二乘法与任意函数拟合

从线性回归中得到的最小二乘公式是一个非常有效的通用方法，它可以适用于大多数函数的拟合过程。本质上，最小二乘是一个最优化问题，构造的目标函数为：

$$J(args) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i, args))^2 \tag{7.11}$$

接着就是对 J 求极值，极值点即为最优参数。那么对于更加一般的函数，我们通常采用数值极值求解例如梯度下降，更直接地可能使用 fminsearch 等内置函数。例如，对于类似于这种形式的逻辑回归：

$$y = \frac{1}{1 + e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n}} \tag{7.12}$$

可以采用如下代码进行最小二乘函数的极值求解：

```
x=[1,2,3;1 2 3;4 5 6];
y=[1 1 1];
args=[1,1,1];
n=size(x,2)
sigmoid = @(x,args) 1./(1+exp(-(args*x')));
J = @(args) 1/n * sum((y-sigmoid(x,args)).^2)
a=fminsearch(J,args)
```

更一般地，可以把它写成函数：

```
fuction a = lsqcurvefit(x,y,f,args)
  n=size(x,2);
  J=@(args) 1/n * sum((y-f(x,args)).^2);
  a=fminsearch(J,args);
end
```

练习应用

- 设计数据与代码，使用 lsqcurvefit 拟合曲线 $\frac{1}{a+bx^2}$，其中 ab 为参数
- 设计数据与代码，使用 lsqcurvefit 拟合曲线 $axsin(wx)+c$，其中 awc 为参数
- 对前面两问的拟合结果进行画图，并且求解对应的 R2 分数

笔记

7.4 拉格朗日插值与样条插值

本节我们学习插值方法，插值和拟合有着很大的区别：插值只能在自变量内部进行填充而无法对自变量范围以外的数据进行预测。插值方法实际上除了是用于处理缺失值一个很好的方法，也是做数据填充的很好的方法。比如，我获取了按日的数据，需要按小时对数据进行填充，这个时候就可以用插值去做填充任务。这里介绍几种比较常见的插值方法。

线性插值对两个点中的解析式是按照线性方程来建模。比如我们得到的原始数据列 y 和数据的下标 x，这里数据下标 x 可能并不是固定频率的连续取值而是和 y 一样存在缺失的。给定了数据点 (xk,yk) 和 (xk+1,yk+1)，需要对两个点之间构造直线进行填充。很显然，根据直线的点斜式方程，这个直线解析式为：

$$L_1(x) = y_k + \frac{y_{k+1} - y_k}{x_{k+1} - x_k}(x - x_k) \tag{7.13}$$

三次样条插值是一种非常自然的插值方法。它将两个数据点之间的填充模式设置为三次多项式。它假设在数据点 (xk,yk) 和 (xk+1,yk+1) 之间的三次式叫做 Ik，那么这一组三次式需要满足条件：

$$\begin{cases} a_i x_i^3 + b_i x_i^2 + c_i x_i + d_i = a_{i+1} x_{i+1}^3 + b_{i+1} x_{i+1}^2 + c_{i+1} x_{i+1} + d_{i+1} \\ 3a_i x_i^2 + 2b_i x_i + c_i = 3a_{i+1} x_{i+1}^2 + 2b_{i+1} x_{i+1} + c_{i+1} \\ 6a_i x_i + 2b_i = 6a_{i+1} x_{i+1} + 2b_{i+1} \end{cases} \tag{7.14}$$

通过解方程的形式可以解出每一只三次式。而简而言之，也就是说某个数据点前后两条三次函数不仅在当前的函数值相等，一次导数值和二次导数值也要保持相等。

对于一组数据 y 和下标 x，定义 n 个拉格朗日插值基函数：

$$l_k(x) = \prod_{\substack{=0 \\ \neq k}}^{n} \frac{x - x_i}{x_k - x_i} \tag{7.15}$$

这本质上是一个分式，当 x=xk 时 lk(x)=1，这一操作实现了离散数据的连续化。按照对应下标的函数值加权求和可以得到整体的拉格朗日插值函数：

$$L(x) = \sum_{k=0}^{n} y_k l_k(x) \tag{7.16}$$

虽然 Baltamatica 中没有为我们提供拉格朗日插值的代码，但我们可以自行实现：

```
function yh= lagrange_interpolate(x,y,xh)
  n = length(x);
  m = length(xh);
  x = x(:);
  y = y(:);
  xh = xh(:);
  yh = zeros(m,1);
```

笔记

```
8    c1 = ones(1,n-1);
9    c2 = ones(m,1);
10   for i=1:n,
11     xp = x([1:i-1 i+1:n]);
12     yh = yh + y(i) * prod((xh*c1-c2*xp')./(c2*(x(i)*c1-xp')),2);
13   end
14 end
```

练习应用

- 仿照拉格朗日插值的原理，写出线性插值的函数
- 仿照拉格朗日插值的原理，写出三次样条插值的函数
- 设计一个 demo，探索三种插值函数的效果

笔记

第八章　基于 Baltamatica 的数据挖掘

本章我们学习的重点内容是 Baltamatica 的数据挖掘。数据挖掘的算法有很多，我肯定不可能全讲完（毕竟我也什么都没有能复现这么多实属不易），真正做数据挖掘的任务 Python 远远盖过了 MATLAB 和 Baltamatica。不过这里为了方便同学们使用，也方便同学们理解，我还是放上了十大挖掘算法的底层原理和复现代码，可以进行一定程度的封装。

8.1　什么是机器学习

机器学习本质上就是根据数据训练一个数学模型，并对未知数据能够成功进行合理预测。换而言之，机器学习的要素包括训练材料与算法（数据，限制条件）、训练目的（分类？回归？或者其他?）、训练效果的衡量标准（目标函数）。但业界更习惯于采用这三个要素：模型、学习准则、优化算法。通常学习一个好的模型，分为以下三步：

1. 选择一个合适的模型，这通常需要依据实际问题而定，针对不同的问题和任务需要选取恰当的模型，模型就是一组函数的集合。

2. 判断一个函数的好坏，这需要确定一个衡量标准，也就是我们通常说的损失函数，损失函数的确定也需要依据具体问题而定，如回归问题一般采用欧式距离，分类问题一般采用交叉熵代价函数。

3. 找出“最好”的函数，如何从众多函数中最快的找出“最好”的那一个，这一步是最大的难点，做到又快又准往往不是一件容易的事情。常用的方法有梯度下降算法，最小二乘法等。

机器学习可以分成图8.1的分类

注意：如果数据条数不上 100 条万万不可用机器学习！用了就算准确率 100% 也是有问题的！这种情况下尤其是神经网络更不能用！

数学建模中常见的就是图8.1中的有监督学习和无监督学习。可能截至目前读者还是对所谓机器学习一头雾水，那么幻想这样一个场面：你的面前出现了一张 excel，excel 的最后一列是一个离散的属性，告诉你每一条数据的标记。比如鸢尾花数据集，又名水仙花数据集，它就收集了 150 朵花的萼片长度、萼片宽度等四个形状指标，最后又来一列告诉你这条数据是哪种鸢尾花。这种数据可以用来预测一朵新的鸢尾花是三种鸢尾花中的哪一种，但是分类模型需要基于后续的标签来指导，这叫分类问题。

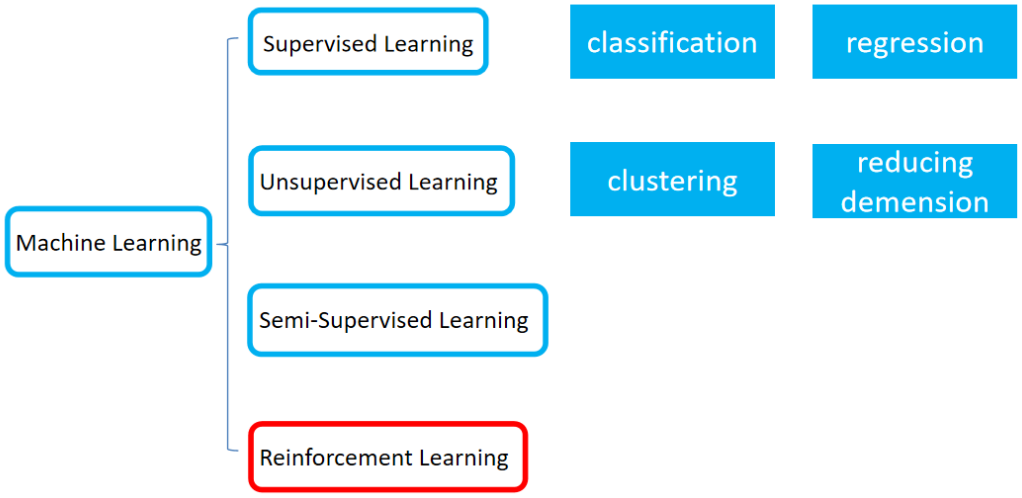


图 8.1: 机器学习的分类

另一类比较有意思的就是聚类问题，比如从野外采回来一堆野豌豆，这些豌豆有大有小，有圆有瘪，有绿的还有黄的。我们可以根据它们的长相将它们分堆，相似的豌豆会被分到一堆里面去，这叫聚类。我不需要知道这个数据里面这些豌豆都叫什么名字，不需要知道每一株豌豆是亚洲产还是美洲产还是非洲产，我只有自变量，也只需要自变量就可以完成，这叫聚类。当然，鸢尾花的区分也可以用聚类实现。没有标签指导模型训练过程，也就是只有自变量的数据就是无监督学习。

回归问题则是指，数据同样有标签，但这个标签是连续实数。比如，同样有一堆自变量，包括房子的位置，离商场的距离，离学校的距离，离地铁站的距离，坐北朝南，透光等等很多因素，最后的标签是房价多少一平。分类是预测花的种类，它无非三种取值；但房价多少一平，这就有点类似于线性回归的味道了。我们也称之为回归，并且机器学习中的回归比线性回归、多项式回归等更复杂。有标签数据的学习，并且是需要利用标签来引导模型学习的方法，叫有监督学习算法。

降维本质上就是通过变量分解与组合的模式，把数据的列数减少。其实常见的主成分分析、因子分析前面已经介绍过，线性判别后面也会提到。至于半监督学习和强化学习数学建模里面介绍的很少，就不展开了。

以分类算法为例展开，一个有标签的数据集需要进行切分，分成训练集和测试集两个部分（如果想严格一点可以分成训练、验证和测试三个部分），训练集用于参数训练而测试集用于评价模型效果。一般是三七开或二八开。如果不理解这个概念可以举个例子哈：反正大家上大学了，老师的题库也就那么多，高数老师无非是从一本吉米多维奇里面抽题目给你们写。但你在不写高数作业或者写了也没改没讲过的情况下直接去裸考，即使你听完了课考的也不是很高除非你是第二个韦东奕，这是无监督学习。有监督学习你肯定得写作业以后改，首先你得写作业，并且写作业肯定要比考试题多。如果写了十道题就能弄懂吉米多维奇的一千题，举一反百的这种学习模式

我们称为 few shot。训练集就是从一千题的吉米多维奇当中抽了七百个题当做作业给你写，写错了答案就相当于标签让你去订正，从而训练你脑壳里面的模型参数。剩下的考试就从你没做过的三百题里面出，这叫测试。

交叉验证就是反复刷题，以最常用的 10 折交叉验证为例，它随机把吉米多维奇的一千题分成 10 等份，一份就是 100 题。这一套复习十轮，每一轮抽一份当测试题，剩下九百题用作作业。十轮复习，这吉米多维奇还学不好那就是个稀奇了，不然就二十折、三十折……这样得到的模型效果会更具有一般性，训练的模型也更稳定。

对于二分类问题，机器学习中有诸多指标评价分类器的分类效果，这里我们主要比较 AUC、F1 分数和准确率。若用混淆矩阵表示，准确率即分类结果与实际相同的样本在总样本中占比，是对样本分类整体精度的描述，其能够直接反映总样本中被正确分类的样本的占比；精准率和召回率是对样本分类中细节精度的描述，而 F1 分数则是对分类问题中细节的权衡考量。F1 分数为查准率和查全率的调和平均数，在一定程度上可以反映模型预测的鲁棒性；将预测为正类的样本中正例率和反例率绘制到图像中构成 ROC 曲线，ROC 曲线与横轴围成的面积即为 AUC 值。AUC 值和 F1 分数、准确率一样，越接近 1 说明效果越佳。

8.2 KNN

KNN 是一种典型的懒惰学习算法。KNN 是最基本的分类算法之一, 是一种典型的监督学习方法，由于不需要进行模型训练在输入测试样本时才开始进行运算，故又被称为“懒惰学习”的典型算法。KNN 的思想概括起来就是八个字：“物以类聚，人以群分”。KNN 用距离衡量样本之间的相似度，读者可以理解为两个样本的自变量属性之间相似度越高，它们的标签也就越可能一样。测试集中每一条数据与训练数据集中所有样本最小的 k 个距离对应的样本标签进行投票即可得到测试样本的分类。

大致的分类流程如下所示：

1. 计算测试集样本与其他训练集样本之间的距离。

2. 统计距离最近的 k 个邻居。

3. 对于 k 个最近的邻居，它们属于哪个分类最多，待分类样本就属于哪一类。

从上面的过程可以看出，KNN 算法的好坏决定性因素有三个：距离的计算方式，K 值的选取和数据集。在计算距离时可以采用多种不同的距离衡量方式，例如欧几里得距离、曼哈顿距离、车比雪夫距离等或者其他，但用的最多的还是欧几里得距离。数据集我们没有办法影响，那 k 值如何选择好呢?

如果 k 值比较小，就相当于测试样本只会受与它最近的邻居影响。这样产生的一个问题就是，如果邻居点是个噪声点，那么未分类物体的分类也会产生误差。如果 k 值比较大，相当于距离过远的点也会对测试样本的分类产生影响，虽然这种情况的好处是鲁棒性强，但是不足也很明显，会产生欠拟合情况，也就是没有把测试样本真正分类出来。

所以 k 值应该是个实践出来的结果，并不是我们事先而定的。通常会选择 k 值为一个 5-15 之间的奇数，但具体选多少除了测试与交叉验证，还可以根据数据量与训练集标签的特征预估。如图8.2所示为 KNN 的大致原理示意图：

为了加快最近邻居的搜索可以利用 KD 树进行数据结构的优化。KD 树是对数据点在 k 维空间中划分的一种数据结构。在 KD 的构造中，每个节点都是 k 维数值点的二叉树。既然是二叉树，就可以采用二叉树的增删改查操作，这样就大大提升了搜索效率。

KNN 最后有一个投票的操作，但 Baltamatica 中没有内置求众数的操作，这样我们就写一个吧：

```
function a = mode(x)
    v=unique(x);
    tar=[];
    for i = v
        count=0;
        for i = x
```

笔记

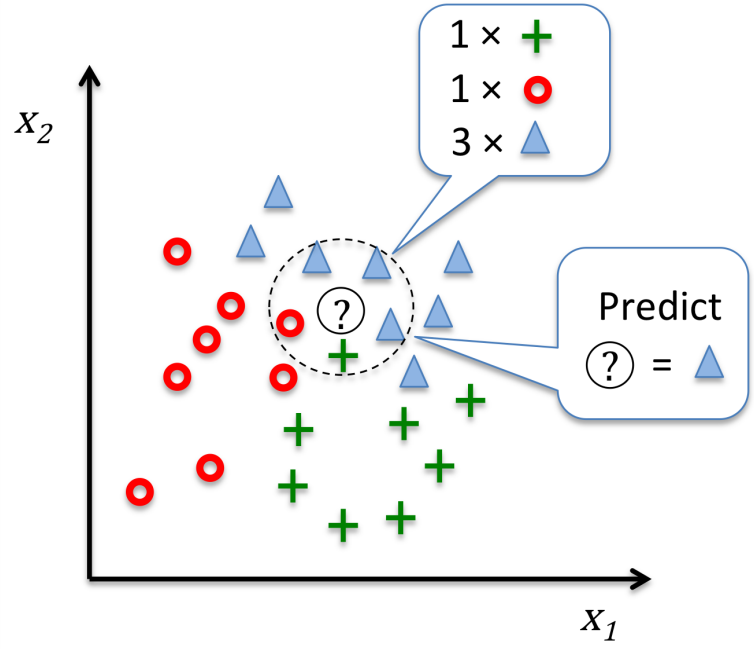


图 8.2: KNN 的执行原理

```
            if i==j
                count=count+1;
            end
        end
        tar=[tar count];
    end
    a=v(find(tar==max(tar)));
end
```

现在我们总算是能够敞开了去写 KNN 的代码了:

```
function out=knn(test_set,train_set,K)
    [n ,~]=size(test_set);
    [m,~]=size(train_set);
    for i=1:n
        for j=1:m
            %求距离
            distance(j)=sqrt(sum((test_set(i,:)-train_set(j,1:4)).^2));
        end
        [~,index]=sort(distance,'ascend');
        label=train_set(index,5);%按照距离大小排序
```

```
11          out(i)=mode(label(1:K));%取前K个标签里出现次数最多的那个标签
12      end
13      out=out';
14  end
```

笔记

8.3 决策树

决策树是一种以树状结构完成分类的任务。什么是决策树？决策树是一种利用树状数据结构来进行分类或者回归的算法。在决策树的生成中，我们通过信息论里面几个数值的计算判断分类例如熵，信息增益，增益率，基尼指数等，然后通过这些指标去生成一整个决策的树形图。如图8.3为决策树生成的基本原理：

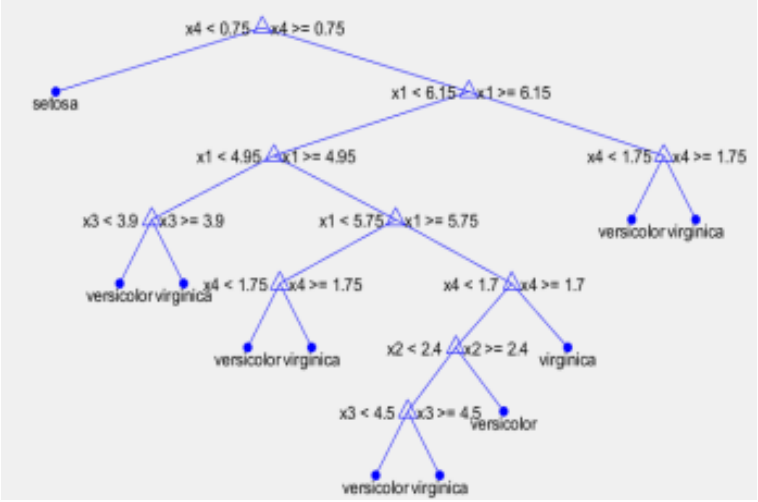


图 8.3: 决策树的基本原理

在决策树的生成中，我们通过信息论里面几个数值的计算判断分类：熵，信息增益，增益率，基尼指数。

首先是信息熵，如果对于样本集合 S，一共可以划分为 k 个类，每个类概率是 pk，那么信息熵定义为：

$$E(S) = -\sum_{i=1}^{k} p_k \log p_k \tag{8.1}$$

如果对数据集 S 按照某一个属性 A 对 S 进行划分，将它划分成 v 个子集，定义属性 A 的信息熵为：

$$E(S, A) = \sum_{i=1}^{v} \frac{|A_i|}{|S|} E(A_i) \tag{8.2}$$

数据集本身的 k 个类是按照最后的返回标签排的，所以数据集本身的信息熵与属性的信息熵并不是一个东西。信息增益就是这个差值，看看按照这个属性划分我的信息到底增长了多少。

$$Gain(A) = E(S) - E(S, A) \tag{8.3}$$

增益率是增益与信息熵的比值。

$$GainRatio(A) = \frac{Gain(A)}{E(S, A)} \tag{8.4}$$

假设数据集有 n 个类，第 k 类的概率是 pk，定义基尼指数：

$$GINI(S) = 1 - \sum_{i=1}^{n} p_k^2 \tag{8.5}$$

决策树，从数据结构的角度而言，使用了树的结构。不一定会是一棵二叉树，可能是普通的。但是并不妨碍我们从数据结构的角度理解它。决策树是一种递归生成的树。毕竟在数据结构里面树的算法很大程度上是递归算法。常见的决策树有三类：

- 基于信息增益的 ID3 决策树，是最简单的决策树算法，只能做离散属性的分类。
- 基于信息增益率的 C4.5 决策树，这时决策树能够处理连续属性，通过阈值划分的方式解决，但还是只能做分类任务。
- 基于基尼指数的 CART 决策树，这时决策树开始能够处理回归。

但是决策树，四平八稳横平竖直，最大的问题就是可能过拟合。为了降低过拟合，你总不能加正则化进来吧？所以，我们不是从凸优化的角度而是从数据结构的角度降低过拟合，并且提出了剪枝的概念。决策树剪枝分两种，预剪枝和后剪枝。预剪枝就是先估计哪些地方的子树应该先砍，而后剪枝则是自底而上先生成完毕再去判断。预剪枝的停止准则包括树的深度、叶子结点的数目、准确度等缺点就是万一剪多了就反而欠拟合了；而后剪枝最大的缺点就是，计算这么多信息熵会不会耗费的时间太多了一点？

不同决策树的生成过程也不尽相同。例如，ID3 决策树是以信息增益作为评价准则的。要求数据只能有离散属性，且只能做分类。它的生成过程如：

- 创建根节点，确定什么是属性。
- 若全部样本都是一类那就别分了统统叶子节点。否则，根据每个属性计算信息增益，根据最大的划分。
- 根据划分属性把属性值不同的样本划到对应边上。
- 根据不同属性递归生成树。

C4.5 决策树是以增益率作为评价准则的。数据此时可以有连续属性了，但还是只能分类。本质上 C4.5 是分治策略，另外最近还有 C5.0。它的构建过程如下：

- 将连续数值离散化，创建树。
- 确定连续属性的阈值，计算信息增益率确定划分属性。
- 根据划分属性把属性值不同的样本划到对应边上。
- 根据不同属性递归生成树。

笔记

CART 决策树基于基尼指数，使用最多最广，它容忍连续属性，既能分类又能回归。自顶而下递归生成的还是一棵二叉树。

- 将连续数值离散化，创建树。
- 确定连续属性的阈值，计算基尼指数增长，确定划分属性按最小者开始划分，注意要二分。
- 根据划分属性把属性值不同的样本划到对应边上。
- 根据不同属性递归计算基尼指数增长并划分。

如果 CART 做的是一个回归问题，则回归中的基尼指数为：

$$GainGINI = \sqrt{y_{k1} - \mu_1} + \sqrt{y_{k2} - \mu_2} \tag{8.6}$$

8.4 PageRank 算法

这一节介绍的是一个在图数据中得到广泛应用的算法——PageRank 算法。PageRank 算法是 Google 创始人拉里·佩奇和谢尔盖·布林于 1997 年构建早期的搜索系统原型时提出的链接分析算法，自从 Google 在商业上获得空前的成功后，该算法也成为其他搜索引擎和学术界十分关注的计算模型。目前很多重要的链接分析算法都是在 PageRank 算法基础上衍生出来的。

PageRank 是 Google 用于用来标识网页的等级/重要性的一种方法，衡量一个网站的好坏的唯一标准。在揉合了诸如 Title 标识和 Keywords 标识等所有其它因素之后，Google 通过 PageRank 来调整结果，使那些更具“等级/重要性”的网页在搜索结果中另网站排名获得提升，从而提高搜索结果的相关性和质量。

级别从 0 到 10 级，10 级为满分。PR 值越高说明该网页越受欢迎（越重要）。例如：一个 PR 值为 1 的网站表明这个网站不太具有流行度，而 PR 值为 7 到 10 则表明这个网站非常受欢迎（或者说极其重要）。一般 PR 值达到 4，就算是一个不错的网站了。Google 把自己的网站的 PR 值定到 10，这说明这个网站是非常受欢迎的，也可以说这个网站非常重要。

在 PageRank 提出之前，已经有研究者提出利用网页的入链数量来进行链接分析计算，这种入链方法假设一个网页的入链越多，则该网页越重要。早期的很多搜索引擎也采纳了入链数量作为链接分析方法，对于搜索引擎效果提升也有较明显的效果。PageRank 除了考虑到入链数量的影响，还参考了网页质量因素，两者相结合获得了更好的网页重要性评价标准。对于某个互联网网页 A 来说，该网页 PageRank 的计算基于以下两个基本假设：

数量假设：在 Web 图模型中，如果一个页面节点接收到的其他网页指向的入链数量越多，那么这个页面越重要。

质量假设：指向页面 A 的入链质量不同，质量高的页面会通过链接向其他页面传递更多的权重。所以越是质量高的页面指向页面 A，则页面 A 越重要。

利用以上两个假设，PageRank 算法刚开始赋予每个网页相同的重要性得分，通过迭代递归计算来更新每个页面节点的 PageRank 得分，直到得分稳定为止。PageRank 计算得出的结果是网页的重要性评价，这和用户输入的查询是没有任何关系的，即算法是主题无关的。假设有一个搜索引擎，其相似度计算函数不考虑内容相似因素，完全采用 PageRank 来进行排序，那么这个搜索引擎的表现是什么样子的呢？这个搜索引擎对于任意不同的查询请求，返回的结果都是相同的，即返回 PageRank 值最高的页面。

PageRank 的计算充分利用了两个假设：数量假设和质量假设。步骤如下：

- 在初始阶段：网页通过链接关系构建起 Web 图，每个页面设置相同的 PageRank 值，通过若干轮的计算，会得到每个页面所获得的最终 PageRank 值。随着每一轮的计算进行，网页当前的 PageRank 值会不断得到更新。

- 在一轮中更新页面 PageRank 得分的计算方法：在一轮更新页面 PageRank 得分的计算中，每个页面将其当前的 PageRank 值平均分配到本页面包含的出链上，这样每个链接即获得了相

笔记

应的权值。而每个页面将所有指向本页面的入链所传入的权值求和，即可得到新的 PageRank 得分。当每个页面都获得了更新后的 PageRank 值，就完成了一轮 PageRank 计算。

对于一个页面 A，如果有 n 个页面链接到它，我们把网页之间的拓扑图（有向图）列出来，每个节点的出度分别为 $deg(K_i)$，那么我们可以列出 pagerank 公式：

$$PR(A) = \sum_{i=1}^{k} \frac{PR(K_i)}{deg(K_i)} \tag{8.7}$$

但是我们得考虑一个问题，就是有的人看到这个网页以后他可能就不会往下翻了。所以我们加上一个概率（阻尼系数）

$$PR(A) = q\sum_{i=1}^{k} \frac{PR(K_i)}{deg(K_i)} + \frac{1-q}{N} \tag{8.8}$$

现在我们开始推导它的基本方程：记 $R = \begin{bmatrix} PR(A_1) \\ PR(A_2) \\ \vdots \\ PR(A_N) \end{bmatrix}$，那我们可以把问题写成矩阵形式：

$$R = \begin{bmatrix} (1-q)/N \\ (1-q)/N \\ \vdots \\ (1-q)/N \end{bmatrix} + q \begin{bmatrix} l(A_1,A_1) & l(A_1,A_2) & \cdots & l(A_1,A_N) \\ l(A_2,A_1) & l(A_2,A_2) & \cdots & l(A_2,A_N) \\ \vdots & \vdots & \cdots & \vdots \\ l(A_N,A_1) & l(A_N,A_2) & \cdots & l(A_N,A_N) \end{bmatrix} R \tag{8.9}$$

其中，如果网页 i 有指向 j 的链接，那 $\sum_{i=1}^{N} l(A_i,A_j) = 1$，否则 $l(A_i,A_j) = 0$

我们先根据网页的连接情况列出概率矩阵，再计算

$$A = qP + ee^T \times \frac{1-q}{N}, e = [1,1,...,1] \tag{8.10}$$

然后循环计算 $A^n X$，其中 X 为一个 N 维向量全部是 1。当 X 收敛的时候，最后每个分量的值就是最后的 PageRank 数值。用幂法计算 PageRank 值总是收敛的，即计算的次数是有限的。Larry Page 和 Sergey Brin 两人从理论上证明了不论初始值如何选取，这种算法都保证了网页排名的估计值能收敛到他们的真实值。

由于互联网上网页的数量是巨大的，上面提到的二维矩阵从理论上讲有网页数目平方之多个元素。如果我们假定有十亿个网页，那么这个矩阵就有一百亿亿个元素。这样大的矩阵相乘，计算量是非常大的。Larry Page 和 Sergey Brin 两人利用稀疏矩阵计算的技巧，大大的简化了计算量。

pagerank 的 Baltamatica 实现如下：

```
function c=pagerank(A,n)    %传入邻接图A和网络规模n
%1.计算每个点的出度
outDegree=zeros(n,1);
```

笔记

```
for i=1:n
    current_outDegree=0;
    for j=1:n
        if A(i,j)>0
            current_outDegree=current_outDegree+1;
        end
    end
    outDegree(i,1)=current_outDegree;
end

%2.基本的pagerank算法，得到的google矩阵A1如下
A1=A;
for i=1:n
    for j=1:n
        if outDegree(i,1)==0                    %如果一个点出度为0，为了防止这个
            A1(i,j)=1/n;
        else
        A1(i,j)=A1(i,j)/outDegree(i,1);
        end
    end
end
A1
%3.初始化PR向量，使得浏览每个页面的概率相同
PR=zeros(n,1);
for i=1:n
    PR(i,1)=1/n;
end
PR
%4.把谷歌矩阵A1转置，右边乘上PR矩阵，就是一次迭代
%迭代收敛的意思是，这次迭代和下次迭代结果相同，就停止
num=0;
%runpagerank(A1,PR)
while 1
    if round(PR,4)==round(A1'*PR,4) %这里如果不用估值（前4），一直没法终止算法
        c=A1'*PR;
        break;
```

```
40      else
41          PR=A1'*PR;
42          PR
43          num=num+1
44      end
45  end
```

8.5 EM 算法

本节我们看到一个无监督方法，也就是期望最大化算法。期望最大化算法是一种无监督学习，在概率模型中寻找参数最大的最大似然估计或者最大后验估计。其中概率模型依赖于无法观测的隐藏变量。

EM 算法的核心是什么呢？比如说我们知道了 AB 两个参数，开始二者都是不知道的。但是我知道了 A 就可以推导 B，得到 B 也可以推导 A。那我可以随机给 A 一个初始值，以此得到 B 的估计值，再由 B 的实际情况出发反推 A 直到二者收敛。有没有感觉和神经网络的正向传播和反向传播有点像呢？这个互相反馈的思想才是最重要的。

EM 搜寻使得 $E[\ln P(Y|h')]$ 最大的 h' 来寻找极大似然假设。期望值是在 Y 遵循的概率分布上计算，此分布由未知参数 θ 确定。

补充:

1. $P(Y|h')$ 是给定假设下所有数据的似然性。合理性在于找一个合适的 h' 使得函数最大化
2. 使概率似然 $\ln P(Y|h')$ 是最大化
3. 引入 $E[\ln P(Y|h')]$ 是因为 Y 本身也是一组变量

整体上这个算法有待求目标

$$Q(h'|h) = E[\ln P(Y|h')|h, X] \tag{8.11}$$

求解分成两步，E 步期望步和 M 步最大化步：

- $Q(h'|h) := E[\ln P(Y|h')|h, X]$
- $h := \arg\max_{h'} Q(h'|h)$

GMM，高斯混合模型，也可以简写为 MOG。高斯模型就是用高斯概率密度函数（正态分布曲线）精确地量化事物，将一个事物分解为若干的基于高斯概率密度函数（正态分布曲线）形成的模型。GMMs 已经在数值逼近、语音识别、图像分类、图像去噪、图像重构、故障诊断、视频分析、邮件过滤、密度估计、目标识别与跟踪等领域取得了良好的效果。

高斯混合模型 (GMM) 是一种机器学习算法。它们用于根据概率分布将数据分类为不同的类别。高斯混合模型可用于许多不同的领域，包括金融、营销等等。这里要对高斯混合模型进行介绍以及真实世界的示例、它们的作用以及何时应该使用 GMM。高斯混合模型 (GMM) 是一个概率概念，用于对真实世界的数据集进行建模。GMM 是高斯分布的泛化，可用于表示可聚类为多个高斯分布的任何数据集。高斯混合模型是一种概率模型，它假设所有数据点都是从具有未知参数的高斯分布的混合中生成的。

高斯混合模型可用于聚类，这是将一组数据点分组为聚类的任务。GMM 可用于在数据集中可能没有明确定义的集群中查找集群。此外，GMM 可用于估计新数据点属于每个集群的概率。高斯混合模型对异常值也相对稳健，这意味着即使有一些数据点不能完全适合任何集群，它们仍然可以产生准确的结果。这使得 GMM 成为一种灵活而强大的数据聚类工具。它可以被理解为一个概率模型，其中为每个组假设高斯分布，并且它们具有定义其参数的均值和协方差。GMM 由两部分组成——均值向量 (μ) 和协方差矩阵 (Σ)。高斯分布被定义为呈钟形曲线的连续概率分布。高斯分布的另一个名称是正态分布。它可以被理解为一个概率模型，其中为每个组假设高斯分布，并且它们具有定义其参数的均值和协方差。GMM 由两部分组成——均值向量 (μ) 和协方差矩阵 (Σ)。本质上这就是一个聚类任务，而这种概率生成式模型用来参数估算的一个好办法就是 EM 算法，下面使用 Baltamatica 写一个 EM 估计 GMM 的过程：

```
%EM algorithm
clc;
clear;

sigma = 1.5;
miu1 = 3;
miu2 = 7;
N = 1000;
x = zeros(1,N);
for i = 1:N
    if rand>0.5
        x(1,i) = randn*sigma + miu1;
        y(1,i) = randn*sigma + miu1;
    else
        %sigma = 0.5;
        x(1,i) = randn*sigma + miu2;
        y(1,i) = randn*sigma + miu2;
    end
end

```

```
21 plot(x,y,'o');
22
23 k = 2;
24 %miu = rand(1,k)*40;
25 miu(1) = 4;
26 miu(2) = 6;
27 cov(1) = 2;
28 cov(2) = 2;
29 %cov = rand(1,k)*6;
30 a(1) = 1.5;
31 a(2) = 1.5;
32 % expectations = zeros(N,k);
33 num = [0,0];
34 n = 1;
35 for step = 1:10000
36     n = 1;
37     m = 1;
38     x1 = [];
39     y1 = [];
40     x2 = [];
41     y2 = [];
42     num = [1  1];
43     for i = 1:N
44         p1 = exp(-(x(i)-miu(1))*(x(i)-miu(1))/(2*cov(1)*cov(1)))/sqrt((2*pi))*
45         p2 = exp(-(x(i)-miu(2))*(x(i)-miu(2))/(2*cov(2)*cov(2)))/sqrt((2*pi))*
46
47         p(i) = a(1)*p1+a(2)*p2;
48         if p1>p2
49             x1(n) = x(i);
50             y1(n) = y(i);
51             n = n+1;
52             num(1) = num(1) + 1;
53         else
54             x2(m) = x(i);
55             y2(m) = y(i);
56             m = m+1;
```

```
57              num(2) = num(2) + 1;
58          end
59      end
60
61          oldmiu = miu;
62          oldcov = cov;
63          miu(1) = sum(x1)/num(1);
64          miu(2) = sum(x2)/num(2);
65          cov(1) = sqrt(sum((x1-miu(1))*(x1-miu(1))')/num(1));
66          cov(2) = sqrt(sum((x2-miu(2))*(x2-miu(2))')/num(2));
67          a(1) = num(1)/N;
68          a(2) = num(2)/N;
69
70      plot(x1,y1,'ro',x2,y2,'go');
71      epsilon = 0.0001;
72      if sum(abs(oldmiu-miu))<epsilon
73          break;
74      end
75      miu
76  end
77  plot(x1,y1,'ro',x2,y2,'go');
```

最终得到的结果参数为均值 miu 分别达到了 2.6588 和 6.9295，聚类图像如图8.4所示：

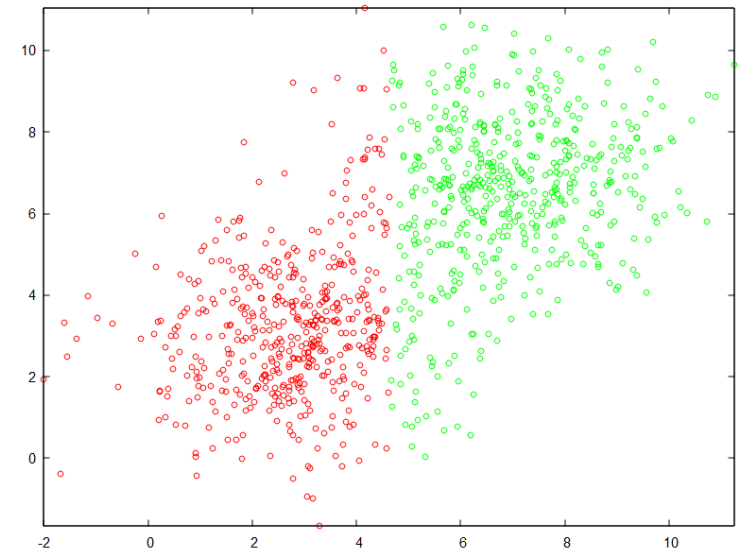


图 8.4: 使用 EM 算法进行 GMM 聚类

8.6 apriori 算法

本节我们将学习如何使用 apriori 分析关联关系。关联规则挖掘发现大量数据中项集之间有趣的关联或者相互联系。关联规则挖掘的一个典型例子就是购物篮分析，该过程通过发现顾客放入其购物篮中不同商品之间的联系，分析出顾客的购买习惯，通过了解哪些商品频繁地被顾客同时买入，能够帮助零售商制定合理的营销策略。

在几十年前的美国有一个怪现象，一个超市的员工发现，超市里面啤酒和尿布在同一个购物篮里面的频率特别高。这个现象引起了统计学家的注意，他们开始思考：是否还具有这类我们没想到的关联物品？由此引发了关联关系挖掘的研究。其实，想要解释这件事并不难，因为在几十年前的美国，家里面打酱油啊买尿布啊这些事都是爸爸在做。爸爸在出门给孩子买尿布的时候捎几瓶啤酒晚上回去宵夜，很合理吧？

但是挖掘关联关系只能靠统计一起出现的商品频率吗？这个频率应该多少次算高呢？这个量化够吗？

我们先介绍一些关联关系挖掘中的基本定义与概念吧。假设一家超市里面有所有的商品 I=i1,i2,i3……id,而在清理单据的时候这段时间一共有 N 个顾客来消费,他们的票据数据为 T=t1,t2,t3….tN。定义：

- 如果一个项集包含 K 个项，则称它为 K-项集。空集是指不包含任何项的项集。例如，{啤酒，尿布，牛奶} 是一个 3-项集。
- 项集的一个重要性质是它的支持度计数，即包含特定项集的事务个数。说白了就是这些订单里面包含想研究的关联商品的单数。
- 关联规则是形如 X→Y 的蕴含表达式，其中 X 和 Y 是不相交的项集。关联规则的强度可以用它的支持度和置信度来度量。支持度确定规则可以用于给定数据集的频繁程度，而置信度确定 Y 在包含 X 的事务中出现的频繁程度。

支持度被定义为：

$$S(X \to Y) = \frac{\sigma(X \cup Y)}{N} \tag{8.12}$$

而置信度为：

$$C(X \to Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} \tag{8.13}$$

关联规则发现就是给定历史订单票据，找出支持度大于等于最小支持度并且置信度大于等于最小置信度的所有规则。它是一个两步的过程：

1. 频繁项集的挖掘：其目标是发现满足最小支持度阈值的所有项集（至少和预定义的最小支持计数一样），这些项集称作频繁项集。
2. 规则的挖掘：其目标是从上一步发现的频繁项集中提取所有高置信度的规则，这些规则称作强规则。

图8.5为 Apriori 算法的操作流程:

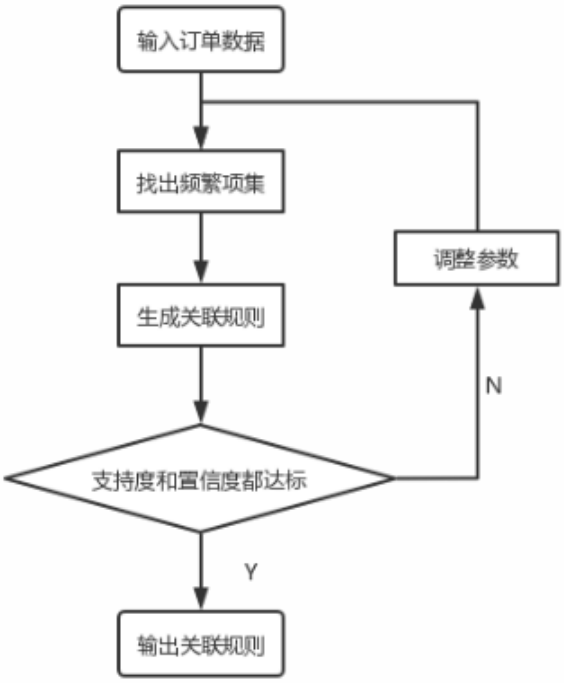


图 8.5: apriori 算法的操作流程

一旦由数据库 D 中的事务找出频繁项集，由它们产生强关联规则是直接了当的（强关联规则满足最小支持度和最小置信度）。对于置信度，可以用下式，其中条件概率用项集支持度计数表示。

$$C(X \to Y) = \frac{S(X \cup Y)}{S(X)} \tag{8.14}$$

根据该式，关联规则可以产生如下:

- 对于每个频繁项集 I，产生 I 的所有非空子集。
- 对于 I 的每个非空子集 s，如果规则置信度达到, 则输出规则。

由于规则由频繁项集产生，每个规则都自动满足最小支持度。频繁项集连同它们的支持度预先存放在 hash 表中，使得它们可以快速被访问。

Apriori 算法因为要反复扫描数据库，每一次遍历都很耗费时间，所以为了提高速度，中国科学家改进了 Apriori 算法于是诞生了基于哈希的 PCY 算法，也有基于树状结构的 FP-Growth 算法也可以对其进行速度改进。目前 Baltamatica 已经对 apriori 有了插件补充。我们可以通过 load plugin 导入 apriori 包，然后使用对应函数。使用起来非常简单，只需要按照如下格式进行输入即可:

```
1 load_plugin("apriori")
2 input = "aprioriEx1_input.txt";
```

笔记

```
output = ”output.txt”;
maxsup = 20;  %表示支持度阈值为 20%
maxconf = 40;  %表示置信度阈值为 40%;
apriori(maxsup, maxconf, input, output)
```

在输入文件中输入 Input 的格式为:

```
1 2
1 3 5
2 4 5
1 2 3
2 3
2 3 5
1 2 3 5
1 2 3
```

8.7 感知机与简单神经网络

这一节是很难学习的一节，我们会从底层矩阵运算开始一步步给大家推出一个简单神经网络。

多层感知机（MLP, Multilayer Perceptron）也叫人工神经网络（ANN, Artificial Neural Network），除了输入输出层，它中间可以有多个隐层。这东西是个非常厉害的模型，有神经网络，才有了今天的深度学习，后面我们会了解到。图8.6所示是一个多层感知机的结构图：

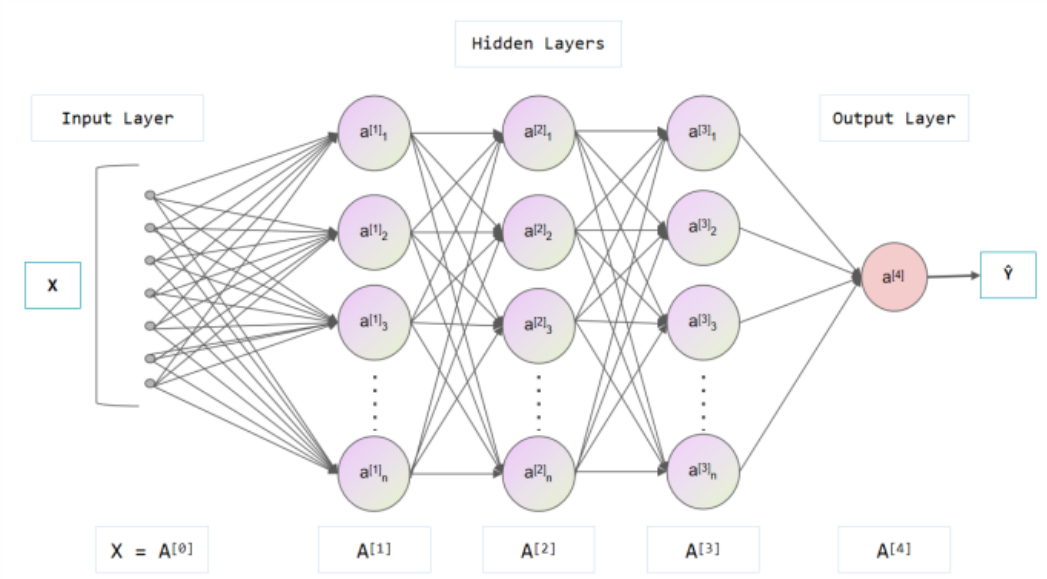


图 8.6: 一个多层感知机的结构

如图8.6所示，对于每两层之间神经元与神经元之间都有连接，这个连接用一个权重表示。那么，如果上一层有 m 个神经元，下一层 n 个神经元，那么权值就有 mn 个，可以排成一个矩阵。加上自己的偏置项，一个数据信息在这两层之间的传播实际上就是一个线性方程 yn(X)=WnX+b。没完，在神经网络中，还有个很重要的成分叫做激活函数。激活函数再将通过线性变换的数据映射成一个新的变量才能作为响应，再来反馈给下一层。常用的激活函数有 sigmoid，tanh，softmax 等。

现在咱们知道了，多层感知机实际上是有三种层，输入层，中间层和输出层。两层之间的过程是线性映射，再激活函数，然后输出去。每两层前后的输入输出递推形式如下，本质上就是一个复合函数：

$$X_n = f_n(W_n X_{n-1} + b_n) \tag{8.15}$$

基本的模型搭建完成后，训练的时候所做的就是完成模型参数的学习。由于存在多层的网络结构，因此即使是数值解梯度下降都非常困难，一个方法就是用误差反向传播算法。误差反向传播是一种非常行之有效的更新算法。每次我们的数据从输入层逐层传入到输出层，发现和实际数据有偏差。但是具体的偏差是多少呢？我们对每一层逐层求导，每一层的误差都与上一层的输出

笔记

有关。这样就建立了复合函数偏导数的关系，形成一种链式反应。

后向传播算法是基于梯度下降的一种改进。还记得我们在线性回归里面说到的凸优化方法，对损失函数求导。这里也是一样，思想还是打算对权值求导。但是神经网络相比于线性回归和逻辑回归模型复杂了不少，再想像那样直接求导就很难了。但是从作用原理出发，对于某一层而言，权值 wij 先影响了对应的输入值，再影响输出值，从而影响损失函数 E。那么，我们可以进行一个链式求导的过程：

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \widehat{y_j}} \frac{\partial \widehat{y_j}}{\partial \beta_j} \frac{\partial \beta_j}{\partial w_{ij}} \tag{8.16}$$

值得注意的是最后一项，这最后一项本质上就是上一层网络的输出啊孩子们，这样本层损失与上层输出就建立关系了。另外我们将损失函数对输入层求微分：

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y_n} \frac{\partial y_n}{\partial y_{n-1}} \cdots \frac{\partial y_1}{\partial x} \tag{8.17}$$

可以看到嗷，这个东西本质上就还是会和上一层的输出扯上关系。误差项就通过这样一个方程，实现了后向传播。通过不断地迭代和训练，最终权重系数等参数都会稳定下来，达到逼近的目的。

训练过程是一个迭代的过程，不断迭代不断更新权重。所以它是吃时间吃算力的。如果真的做深度学习研究那得配置比较好的显卡。深度学习这一概念是机器学习的一个部分。它属于机器学习但是表现力却高于普通的机器学习。深度学习的基石是神经网络，也就是我们之前介绍过的多层感知机。传统的机器学习方法确实具有很高的效率，但是它们似乎都非常偏重“算法”这一充满智慧与技巧的东西，而对数据本身没有太多深入。当处理一些高维海量数据集的时候它们的表现并不是太理想尤其是在图像处理这一方面。而深度学习这一概念的提出恰恰弥补了这一空缺。

多层感知机神经元模型的三个要素，就是权重矩阵，偏置项阈值和激活函数。这其实也是一种递推式的线性方程，是关于层级结构的递推。而从仿生学的角度来看，多层感知机实际上是模拟生物大脑内神经元的连接与活动方式而提出的一种数学模型。所以，我们更多的会把多层感知机称作神经网络。

那为什么深度学习叫做深度学习呢？这个深度指的是什么？在神经网络中，如果我们增加层级结构，将更多的隐藏层添加进神经网络，这个神经网络的“深度”就会增加，效果也会变好一些。这就是为什么它叫深度学习。不过一个很大的问题就是一旦网络层级过多参数也会爆炸式增长，比如 openAI 最近推出的 GPT-3 架构包含了上万亿个参数。

使用 baltamatica 分类鸢尾花数据集是一个经典的机器学习任务。鸢尾花数据集中包含了鸢尾花的四种不同特征，比如萼片宽度、萼片长度、花瓣宽度、花瓣长度，用这四个特征可以分类三种不同的鸢尾花。数据集随处可见，可以在 UCI 下载到，用 baltamatica 模拟一个 BP 神经网络的代码如下：

```
1 load("iris_training.mat")
2 X = [iristraining(1:30, 1:4); iristraining(41:70, 1:4); iristraining(81:110, 1:4)]
```

```
3  D = [iristraining(1:30,5);iristraining(41:70,5);iristraining(81:110,5)];
4  X_test = [iristraining(31:40,1:4);iristraining(71:80,1:4);iristraining(111:12
5  D_test = [iristraining(31:40,5);iristraining(71:80,5);iristraining(111:120,5)
6  %%Bp neural network
7  %三层bp神经网络，四维特征，三分类问题
8  %最后三个节点的输出分别代表三个类别，当输入类别为i是，当且仅当第i个神经元输出
9  %%bp神经网络共分为三层，输入层，中间层，输出层，其中输入层4节点，中间层4节点，
10 w1 = 2*rand(5,4)-1;                    %连接权重
11 w2 = 2*rand(5,4)-1;
12 w3 = 2*rand(5,3)-1;
13 s = 0.01;                              %误差
14 a = 0.05;                              %学习率
15 d_correct = 0.9;
16 d_wrong = 0.1;
17 err = 1;
18 gen = 0;
19 %BP训练
20 while err>s && gen<1000
21     gen = gen + 1;
22     err = 0;
23     for i = 1:90
24
25         %前馈
26         for m = 1:4
27             x(m) = logsig(w1(1,m)*X(i,1) + w1(2,m)*X(i,2) + w1(3,m)*X(i,3) + w
   %输入层
28         end
29         for m = 1:4
30             y(m) = logsig(w2(1,m)*x(1) + w2(2,m)*x(2) + w2(3,m)*x(3) + w2(4,m)
   %中间层
31         end
32         for m = 1:3
33             z(m) = logsig(w3(1,m)*y(1) + w3(2,m)*y(2) + w3(3,m)*y(3) + w3(4,m)
   %输出层
34         end
35         for m =1:3
```

笔记

```
            if (D(i)+1)==m
                err = err + (z(m)-d_correct)^2;
            else
                err = err + (z(m)-d_wrong)^2;
            end
        end
        %反馈
        for m = 1:3
            delta_w3(m) = z(m)*(1-z(m))*(z(m)-(d_wrong+(d_correct-d_wrong)*((D
        end
        for m = 1:4
            delta_w2(m) = y(m)*(1-y(m))*(delta_w3(1)*w3(m,1) + delta_w3(2)*w3(
        end
        for m = 1:4
            delta_w1(m) = x(m)*(1-x(m))*(delta_w2(1)*w2(m,1) + delta_w2(2)*w2(
        end

        %更新网络权重
        w1 = w1 - a*[X(i,:) -1]'*delta_w1;
        w2 = w2 - a*[x -1]'*delta_w2;
        w3 = w3 - a*[y -1]'*delta_w3;
    end
    err = err/270;
end
%测试效果
correct = 0;
for i = 1:30
    corr = 1;
    %前馈
    for m = 1:4
        x(m) = logsig(w1(1,m)*X_test(i,1) + w1(2,m)*X_test(i,2) + w1(3,m)*X_t
%输入层
    end
    for m = 1:4
        y(m) = logsig(w2(1,m)*x(1) + w2(2,m)*x(2) + w2(3,m)*x(3) + w2(4,m)*x(4
%中间层
```

```
    end
    for m = 1:3
        z(m) = logsig(w3(1,m)*y(1) + w3(2,m)*y(2) + w3(3,m)*y(3) + w3(4,m)*y(4
%输出层
    end
    for m = 1:3
        if m==(D_test(i)+1) && z(m)<0.5
            corr = 0;
        end
        if m~=(D_test(i)+1) && z(m)>0.5
            corr = 0;
        end
    end
    if corr==1
        correct = correct + 1;
    end
end
correct = correct/30;
fprintf('correction is %d%%',correct*100)
```

最终结果还是比较客观的，能达到 96.3% 的准确率。

8.8 KMeans 聚类算法

本节我们看到 KMeans 聚类算法。聚类理论上是一种无监督学习问题，样本是没有标记的，也就算不出来准确率。这一节的算法虽然形式上和 KNN、决策树这些内容有一定的相似和互通之处，但核心上这是两类完全不同的问题！

K-means 算法，也称为 K 平均或者 K 均值，一般作为掌握聚类算法的第一个算法。这里的 K 为常数，需事先设定，通俗地说该算法是将没有标注的 M 个样本通过迭代的方式聚集成 K 个簇。在对样本进行聚集的过程往往是以样本之间的距离作为指标来划分。

它和 KNN 有很多共同之处，同样需要计算距离，同样会进行近邻寻找，但它的过程没有标签指导，所以具体流程也就有所不同。通俗一点来说，聚类如果希望我们把数据分成 K 堆，那么首先会随机抽 K 个幸运儿作为样本的中心点，然后对于数据中每个样本，找到它最近的那个中心点作为“大哥”，然后加入他的帮派；每个小弟都找到自己大哥以后帮派内会“比武”选出新的“大哥”，这个“比武”就是比帮派内谁与这个帮派的中心点最近；然后所有的小弟集中起来，再去给他们重新选择自己想加入的堂口的机会，于是诞生了一批二五仔换到其他的帮派里面去……当这个数据集里面不再存在二五仔的时候聚类就结束了，每个堂口的老大也确定了（举这个例子是因为比较喜欢看古惑仔谢谢）。

如果用科学语言叙述，算法的执行步骤如下：

1. 选取 K 个点做为初始聚集的簇心（也可选择非样本点）。
2. 分别计算每个样本点到 K 个簇核心的距离，找到离该点最近的簇核心，将它归属到对应的簇。
3. 所有点都归属到簇之后，M 个点就分为了 K 个簇。之后重新计算每个簇的中心，也就是每个堂口的老大。
4. 反复迭代 2 - 3 步骤，直到达到某个中止条件。常用的中止条件有迭代次数、最小平方误差 MSE、簇中心点变化率等。

对于 KMean 算法来说有三个比较重要的因素要考虑, 分别如下所述；

1. k 值的选择：k 值对最终结果的影响至关重要，而它却必须要预先给定。比较合适的方法有肘部图法、轮廓系数法等。
2. 异常点的干扰：K-means 算法在迭代的过程中使用所有点的均值作为新的中心，如果簇中存在异常点，将导致均值偏差比较严重。
3. 初值敏感：K-means 算法是初值敏感的，选择不同的初始值可能导致不同的簇划分规则。为了避免这种敏感性导致的最终结果异常性，可以采用初始化多套初始节点构造不同的分类规则, 然后选择最优的构造规则。针对这点后面因此衍生了：二分 K-Means 算法、K-Means++ 算法、Canopy 算法等。

衡量聚类好坏的标准可以用轮廓系数来描述。轮廓系数的定义为:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{8.18}$$

轮廓系数在 [-1,1] 之间，越大越合理。

判断最优的 k 值会采取肘部图策略。肘部法则的计算原理是损失函数，损失函数是每个变量点到其类别中心的位置距离平方和。在选择类别数量上，肘部法则会把不同值的成本函数值画出来。肘部就是指这个图的拐点，下降从快到慢的点。这个点可能需要目测一下，有一点偏差，但通常八九不离十。

8.9 支持向量机

本节我们将学习支持向量机的底层逻辑。

支持向量机 SVM 是从线性可分情况下的最优分类面提出的。所谓最优分类，就是要求分类线不但能够将两类无错误的分开，而且两类之间的分类间隔最大。推广到高维空间，最优分类线就成为最优分类面。如图 8.12 所示：

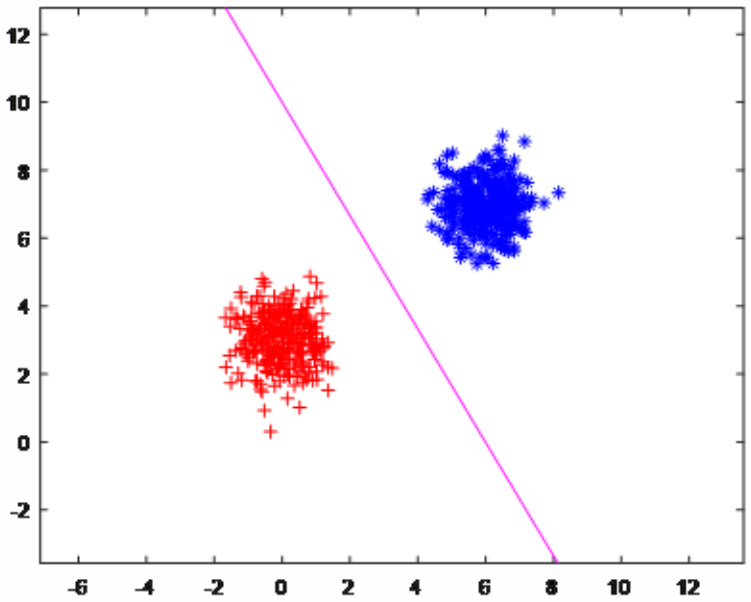


图 8.7: SVM 中的分割超平面

图 8.12 中所示的是两类样本之间如何用一个最优分割超平面去切分两类样本的范围。这条直线到最近的一个红色点和最近的一个蓝色点距离之和最大。那么对于二分类问题（我们暂且拿二分类为例），先找两个平行的分割超平面，能够接近红色和蓝色的边界。

$$H1 : w^T x + b = +1$$
$$H2 : w^T x + b = -1$$

超平面不一定得是平面或者直线，它代表的是变量之间的线性组合。两个超平面之间的间距。就这样，我们导出来一个凸优化问题：

$$\min_w \frac{||w||^2}{2}$$
$$\text{s.}t. y_i(w^T x_i + b) \geq 1$$

超平面或者说支持向量本质上就只是一个线性方程。在现实任务中往往很难确定合适的核函数使训练集在特征空间中线性可分，退一步说，即使瞧好找到了这样的核函数使得样本在特征

空间中线性可分，也很难判断是不是由于过拟合造成。也就是说，我们可能并不能刻意要求所有的变量都线性可分，我们只能说让绝大部分数据正常，而允许一小部分极端分子被误分类。这就是为什么我们要引入软间隔的概念。软间隔与正则化十分相似，加入一些松弛变量我们的模型就变成了：

$$\min_{w} \frac{||w||^2}{2} + C\sum_{i=1}^{n}\xi_i$$
$$s.t. y_i(w^T x_i + b) \geq 1 - \xi_i,$$

支持向量机曾一度掀起了机器学习的浪潮，其原因就在于核方法的引入。核方法使得数据能够由低维映射向高维，让形如异或问题这样的线性不可分问题得到解决。

非线性问题往往不好求解，所以希望能用解线性分类问题的方法求解，因此可以采用非线性变换，将非线性问题变换成线性问题。对于这样的问题，可以将训练样本从原始空间映射到一个更高维的空间，使得样本在这个空间中线性可分，如果原始空间维数是有限的，即属性是有限的，那么一定存在一个高维特征空间是样本可分。

我们用核方法映射过后的数值代替 X，就可以得到新的超平面形式为:$y = w^T K(X) + b$ 并且很容易发现这就是一个典型的凸优化问题，所以我们考虑使用拉格朗日方法。先不考虑核化，引入一系列拉格朗日乘子以后问题可以变形成：

$$L(w, b, \alpha) = \frac{1}{2}||w||^2 + \sum_{i=1}^{2}\alpha_i(1 - y_i(w^T x_i + b))$$

对目标函数求偏导就等于：

$$\begin{cases} \frac{\partial L}{\partial b} = \sum_{i=1}^{n}\alpha_i y_i \\ \frac{\partial L}{\partial w} = w - \sum_{i=1}^{n}\alpha_i y_i x_i \end{cases}$$

解方程然后代入，将 w 和 b 消掉就只剩下了拉格朗日乘子：

$$\max_{\alpha} \sum_{i=1}^{n} a_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i x_j$$
$$s.t. \sum_{i=1}^{n}\alpha_i y_i = 0, \alpha_i \geq 0, i = 1, 2, ..., n$$

整个过程的 KKT 条件就是：

$$\begin{cases} \alpha_i \geq 0 \\ y_i f(x_i) - 1 \geq 0 \\ \alpha_i(y_i f(x_i) - 1) = 0 \end{cases}$$

当训练完成后，大部分样本都不需要保留，最终模型只与支持向量有关。如果是经过核化的，我们就把目标函数变成：

$$\max_{\alpha} \sum_{i=1}^{n} a_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$

这个问题变量太多，传统的优化算法很难求解。在运筹学当中有一种方法叫序贯优化算法（SMO）。SMO 是用来求解拉格朗日乘子的，它的思想概括为不断地变换主元，只留下两个主元给我自由活动的最小空间（因为只留一个的话那一个可以通过代换的方式定下来所以要确定主元就至少要选两个可变其他的为常量）我们可以简记为，反带入：

$$W = a_1 + a_2 - \frac{1}{2} K_{11} {a_1}^2 - \frac{1}{2} K_{22} {a_2}^2 - y_1 y_2 a_1 a_2 K_{12} - y_1 a_1 v_1 - y_2 a_2 v_2 + C_0$$

然后这一凸优化问题可以转化求导就可以解出来了。求出来两个以后我们再换组合，将这些乘子逐一击破。注意：SMO 这个过程比较繁琐，也是训练为啥这么费时间的原因。

对于一般的回归问题，给定训练样本，我们希望学习到一个 f(x) 使得其与 y 尽可能的接近，w，b 是待确定的参数。在这个模型中，只有当 f(x) 与 y 完全相同时，损失才为零，而支持向量回归假设我们能容忍的 f(x) 与 y 之间最多有 ε 的偏差，当且仅当 f(x) 与 y 的差别绝对值大于 ε 时，才计算损失，此时相当于以 f(x) 为中心，构建一个宽度为 2ε 的间隔带，若训练样本落入此间隔带，则认为是被预测正确的。支持向量回归的模型形式为：

$$\frac{||w||^2}{2} + C \sum_{i=1}^{n} l_\epsilon (f(x_i) - y_i)$$

其中：

$$l_\epsilon(x) = \begin{cases} 0 (x \leq \epsilon) \\ (|x| - \epsilon)(x > \epsilon) \end{cases}$$

具体的求解过程得先引入松弛变量然后再来引入拉格朗日乘子再对偶。具体我不展开了，直接上最后的模型形式吧：

$$f(x) = \sum_{i=1}^{n} (\widehat{\alpha_i} - \alpha_i) x_i^T x + b$$

$$b = y_i + \epsilon - \sum_{i=1}^{n} (\widehat{\alpha_i} - \alpha_i) x_i^T x$$

支持向量机的理论是很复杂的一套优化理论，近年来还有对半监督的 SVM、few-shot 与 SVM 结合的研究等等，有兴趣的话是可以深入了解的。至于说 SVM 的实现，我只在网上查到一份代码对这一块讲的比较清楚，把这份代码放出来给大家看看哈：

```
%%
% * svm 简单算法设计
%
%% 加载数据
% * 最终data格式：m*n，m样本数，n维度
% * label:m*1  标签必须为-1与1这两类
clc
clear
% close all
%data = load('data_test1.mat');
data = rand(100,3);
data(:,3)=[zeros(50,1);ones(50,1)];
train_data = data(1:end-1,:)';
label = data(end,:)';
[num_data,d] = size(train_data);
data = train_data;
%% 定义向量机参数
alphas = zeros(num_data,1);
% 系数
b = 0;
% 松弛变量影响因子
C = 0.6;
iter = 0;
max_iter = 80;
% 核函数的参数
sigma = 4;
%%
while iter < max_iter
    alpha_change = 0;
    for i = 1:num_data
        %输出目标值
        pre_Li = (alphas.*label)'*Kernel(data,data(i,:),sigma) + b;
        %样本i误差
        Ei = pre_Li - label(i);
        % 满足KKT条件
        if (label(i)*Ei<-0.001 && alphas(i)<C)||(label(i)*Ei>0.001 && alphas(
```

```
37          % 选择一个和 i 不相同的待改变的alpha(2)--alpha(j)
38          j = randi(num_data,1);
39          if j == i
40              temp = 1;
41              while temp
42                  j = randi(num_data,1);
43                  if j ~= i
44                      temp = 0;
45                  end
46              end
47          end
48          % 样本j的输出值
49          pre_Lj = (alphas.*label)'*Kernel(data,data(j,:),sigma) + b;
50          %样本j误差
51          Ej = pre_Lj - label(j);
52          %更新上下限
53          if label(i) ~= label(j) %类标签相同
54              L = max(0,alphas(j) - alphas(i));
55              H = min(C,C + alphas(j) - alphas(i));
56          else
57              L = max(0,alphas(j) + alphas(i) -C);
58              H = min(C,alphas(j) + alphas(i));
59          end
60          if L==H  %上下限一样结束本次循环
61              continue;end
62          %计算eta
63          eta = 2*Kernel(data(i,:),data(j,:),sigma)- ...
64              Kernel(data(i,:),data(i,:),sigma)...
65              - Kernel(data(j,:),data(j,:),sigma);
66          %保存旧值
67          alphasI_old = alphas(i);
68          alphasJ_old = alphas(j);
69          %更新alpha(2)，也就是alpha(j)
70          alphas(j) = alphas(j) - label(j)*(Ei-Ej)/eta;
71          %限制范围
72          if alphas(j) > H
```

```
73                  alphas(j) = H;
74              elseif alphas(j) < L
75                  alphas(j) = L;
76              end
77              %如果alpha(j)没怎么改变，结束本次循环
78              if abs(alphas(j) - alphasJ_old)<1e-4
79                  continue;end
80              %更新alpha(1)，也就是alpha(i)
81              alphas(i) = alphas(i) + label(i)*label(j)*(alphasJ_old-alphas(j));
82              %更新系数b
83              b1 = b - Ei - label(i)*(alphas(i)-alphasI_old)*...
84                  Kernel(data(i,:),data(i,:),sigma) - label(j)*...
85                  (alphas(j)-alphasJ_old)*Kernel(data(i,:),data(j,:),sigma);
86              b2 = b - Ej - label(i)*(alphas(i)-alphasI_old)*...
87                  Kernel(data(i,:),data(j,:),sigma)- label(j)*...
88                  (alphas(j)-alphasJ_old)*Kernel(data(j,:),data(j,:),sigma);
89              %b的几种选择机制
90              if alphas(i)>0 && alphas(i)<C
91                  b = b1;
92              elseif alphas(j)>0 && alphas(j)<C
93                  b = b2;
94              else
95                  b = (b1+b2)/2;
96              end
97              %确定更新了，记录一次
98              alpha_change = alpha_change + 1;
99          end
100     end
101     % 没有实行alpha交换，迭代加1
102     if alpha_change == 0
103         iter = iter + 1;
104     %实行了交换，迭代清0
105     else
106         iter = 0;
107     end
108     disp(['iter ================== ',num2str(iter)]);
```

```
109 end
110 %% 计算权值W
111 % W = (alphas.*label)'*data;
112 %记录支持向量位置
113 index_sup = find(alphas ~= 0);
114 %计算预测结果
115 predict = (alphas.*label)'*Kernel(data,data,sigma) + b;
116 predict = sign(predict);
117 %% 显示结果
118 figure;
119 index1 = find(predict==-1);
120 data1 = (data(index1,:))';
121 plot(data1(1,:),data1(2,:),'+r');
122 hold on
123 index2 = find(predict==1);
124 data2 = (data(index2,:))';
125 plot(data2(1,:),data2(2,:),'*');
126 hold on
127 dataw = (data(index_sup,:))';
128 plot(dataw(1,:),dataw(2,:),'og','LineWidth',2);
129 title(['核函数参数sigma = ',num2str(sigma)]);
130
131 function Ek = calEk(data,alphas,label,b,k)
132 pre_Li = (alphas.*label)'*(data*data(k,:)') + b;
133 Ek = pre_Li - label(k);
134
135 function [J,Ej] = select(i,data,num_data,alphas,label,b,C,Ei,choose)
136 maxDeltaE = 0;maxJ = -1;
137 if choose == 1 %全遍历---随机选择alphas
138     j = randi(num_data ,1);
139     if j == i
140         temp = 1;
141         while temp
142             j = randi(num_data,1);
143             if j ~= i
144                 temp = 0;
```

```
145             end
146         end
147     end
148     J = j;
149     Ej = calEk(data,alphas,label,b,J);
150 else %部分遍历--启发式的选择alphas
151     index = find(alphas>0 & alphas < C);
152     for k = 1:length(index)
153         if i == index(k)
154             continue;
155         end
156         temp_e = calEk(data,alphas,label,b,k);
157         deltaE = abs(Ei - temp_e); %选择与Ei误差最大的alphas
158         if deltaE > maxDeltaE
159             maxJ = k;
160             maxDeltaE = deltaE;
161             Ej = temp_e;
162         end
163     end
164     J = maxJ;
165 end
166
167 function result = Kernel(data1,data2,sigma)
168 % data里面每一行数据是一个样本(的行向量)
169 [m1,~] = size(data1);
170 [m2,~] = size(data2);
171 result = zeros(m1,m2);
172 for i = 1:m1
173     for j = 1:m2
174         result(i,j) = exp(-norm(data1(i,:)-data2(j,:))/(2*sigma^2));
175     end
176 end
```

后记

本书是我根据我的书稿《MATLAB 数学建模导论》和我早年的草案《数据科学基础: From 0 to 1》整理而成，由于 Baltamatica 这一软件也是起步阶段，还存在一些问题，欢迎读者批评斧正！

另外，特别鸣谢北京大学卢朓老师和李若老师的大力支持和帮助！