生物信息学算法与实践

讲义

曹同成

2005 年

目 录

第一章 前言

1.1 本门课的学习目的和学习内容

本门课的目的主要是通过学习一些生物信息学相关的算法，通过将这些算法用 C 语言程序实现，提高自己的编程能力和对算法的理解能力，还有将部分已有的算法在 Linux 系统上进行编译实现，提高自己对已有源程序算法的理解能力。

因此，本门课主要分三个部分：

第一部分，主要学习一些简单的数值算法，将这些算法利用 C 语言编制成程序，熟悉 C 语言的编程，提高自己的编程能力，在此基础上，形成一个自己的 C 语言的矩阵运算库。

第二章是牛顿迭代法求方程的根，在这一章里，通过对几种方程根的求解方法，掌握牛顿迭代法，并将牛顿迭代法形成程序，这一章程序比较简单，通过学习，熟悉 C 程序设计。

第三章主要是方程组的求解方法，也是对其中一种方法进行程序设计，提高自己的编程能力。

第四章是在第三章基础上进行矩阵求拟算法的编程，从这一章开始，逐步形成自己的矩阵运算库，包括矩阵转置、矩阵相乘、矩阵求拟、矩阵 SVD 分解等，为以后的算法编程打下基础。

以上主要是第一部分的内容，这部分的学习目的是熟悉和提高 C 语言编程能力，形成矩阵运算库，为以后的算法编程打下基础。

第二部分主要对一些常用的生物信息学算法进行编程，主要有下面几个程序构成。

第五章主要是最小二乘曲线拟合的学习，并且利用前面形成的矩阵算法库，编制最小二乘曲线拟合的 C 语言程序。

第六章主要是一些分类算法的学习，掌握模式识别和 PLS 方法，我们上面的矩阵库里的 SVD 分解在这里进行学习和编程。对模式识别和 PLS 分类要形成程序，对一些生物信息的数据进行处理。

第七章是优化算法的学习，在这一章中对遗传算法进行编程，掌握遗传算法的原理和应用，优化算法在生物信息学中有很大的应用，在这一章中我们首先对一套简单的数据进行处理，在下一章中会有一个很好的应用举例。

这一部分主要通过一些算法的学习，掌握这些算法的原理和在生物信息学上的应用，并且提高自己的编程能力，这部分的程序比较复杂，编起来需要掌握更多的计算机知识，但通过这部分的学习会对自己的编程有很大的提高。

第三部分主要是几个复杂的生物信息学算法，这部分自己是无法编写的，我们通过对别人编写的程序的安装和使用，提高自己利用已有程序的能力，并且要掌握这些算法。

第八章主要是计算机辅助药物设计的学习，我们通过将一个计算机辅助药物设计程序 AutoDock3.05 在 Linux 上的安装和计算，掌握分子对接算法的原理和应用，掌握在 Linux 下的编程及对已有源程序的编译和使用。

第九章是隐马尔可夫模型算法，通过对算法的介绍，学习这种算法的原理以及在基因识别上的应用，也是掌握一种已有的程序的安装和使用，计算一组基因上的数据。

1.2 主要生物信息学数据库介绍

数据库是生物信息学的主要内容，各种数据库几乎覆盖了生命科学的各个领域。核酸序列数据库有 GenBank, EMBL, DDB 等，蛋白质序列数据库有 SWISS-PROT, PIR, OWL, NRL3D, TrEMBL 等，蛋白质片段数据库有 PROSITE, BLOCKS, PRINTS 等，三维结构数据库有 PDB, NDB, BioMagResBank, CCSD 等，与蛋白质结构有关的数据库还有 SCOP, CATH, FSSP, 3D-ALI, DSSP 等，与基因组有关的数据库还有 ESTdb, OMIM, GDB, GSDB 等，文献数据库有 Medline, Uncover 等。另外一些公司还开发了商业数据库,如 MDL 等。生物信息学数据库覆盖面广，分布分散且格式不统一, 因此一些生物计算中心将多个数据库整合在一起提供综合服务，如 EBI 的 SRS(Sequence Retrieval System)包含了核酸序列库、蛋白质序列库，三维结构库等 30 多个数据库及 CLUSTALW、PROSITESEARCH 等强有力的搜索工具，用户可以进行多个数据库的多种查询。

基因和基因组数据库

1. Genbank 库包含了所有已知的核酸序列和蛋白质序列，以及与它们相关的文献著作和生物学注释。它是由美国国立生物技术信息中心(NCBI)建立和维护的。它的数据直接来源于测序工作者提交的序列；由测序中心提交的大量 EST 序列和其它测序数据；以及与其它数据机构协作交换数据而来。Genbank 每天都会与欧洲分子生物学实验室(EMBL)的数据库，和日本的 DNA 数据库(DDBJ)交换数据，使这三个数据库的数据同步。Genbank 的数据可以从

NCBI 的 FTP 服务器上免费下载完整的库，或下载积累的新数据。NCBI 还提供广泛的数据查询、序列相似性搜索以及其它分析服务，用户可以从 NCBI 的主页上找到这些服务。

Genbank 库里的数据按来源于约 55,000 个物种，其中 56%是人类的基因组序列(所有序列中的 34%是人类的 EST 序列)。每条 Genbank 数据记录包含了对序列的简要描述，它的科学命名，物种分类名称，参考文献，序列特征表，以及序列本身。序列特征表里包含对序列生物学特征注释如：编码区、转录单元、重复区域、突变位点或修饰位点等。所有数据记录被划分在若干个文件里，如细菌类、病毒类、灵长类、啮齿类，以及 EST 数据、基因组测序数据、大规模基因组序列数据等 16 类，其中 EST 数据等又被各自分成若干个文件。

(1)Genbank 数据检索

NCBI 的数据库检索查询系统是 Entrez。Entrez 是基于 Web 界面的综合生物信息数据库检索系统。利用 Entrez 系统，用户不仅可以方便地检索 Genbank 的核酸数据，还可以检索来自 Genbank 和其它库的蛋白质序列数据、基因组图谱数据、来自分子模型数据库(MMDB)的蛋白质三维结构数据、种群序列数据集、以及由 PubMed 获得 Medline 的文献数据。

Entrez 提供了方便实用的检索服务，所有操作都可以在网络浏览器上完成。用户可以利用 Entrez 界面上提供的限制条件(Limits)、索引(Index)、检索历史(History)和剪贴板(Clipboard)等功能来实现复杂的检索查询工作。对于检索获得的记录，用户可以选择需要显示的数据，保存查询结果，甚至以图形方式观看检索获得的序列。更详细的 Entrez 使用说明可以在该主页上获得。

(2)向 Genbank 提交序列数据

测序工作者可以把自己工作中获得的新序列提交给 NCBI，添加到 Genbank 数据库。这个任务可以由基于 Web 界面的 BankIt 或独立程序 Sequin 来完成。BankIt 是一系列表单，包括联络信息、发布要求、引用参考信息、序列来源信息、以及序列本身的信息等。用户提交序列后，会从电子邮件收到自动生成的数据条目，Genbank 的新序列编号，以及完成注释后的完整的数据记录。用户还可以在 BankIt 页面下修改已经发布序列的信息。BankIt 适合于独立测序工作者提交少量序列，而不适合大量序列的提交，也不适合提交很长的序列，EST 序列和 GSS 序列也不应用 BankIt 提交。BankIt 使用说明和对序列的要求可详见其主页面。

大量的序列提交可以由 Sequin 程序完成。Sequin 程序能方便的编辑和处理复杂注释，并包含一系列内建的检查函数来提高序列的质量保证。它还被设计用于提交来自系统进化、种群和突变研究的序列，可以加入比对的数据。Sequin 除了用于编辑和修改序列数据记录，还可以用于序列的分析，任何以 FASTA 或 ASN.1 格式序列为输入数据的序列分析程序都可

以整合到 Sequin 程序下。在不同操作系统下运行的 Sequin 程序都可以在 ftp://ncbi.nlm.nih.gov/sequin/下找到，Sequin 的使用说明可详见其网页。

NCBI 的网址是：http://www.ncbi.nlm.nih.gov。

Entrez 的网址是：http://www.ncbi.nlm.nih.gov/entrez/。

BankIt 的网址是：http://www.ncbi.nlm.nih.gov/BankIt。

Sequin 的相关网址是：http://www.ncbi.nlm.nih.gov/Sequin/。

2. EMBL 核酸序列数据库

EMBL 核酸序列数据库由欧洲生物信息学研究所(EBI)维护的核酸序列数据构成，由于与 Genbank 和 DDBJ 的数据合作交换，它也是一个全面的核酸序列数据库。该数据库由 Oracal 数据库系统管理维护，查询检索可以通过通过因特网上的序列提取系统(SRS)服务完成。向 EMBL 核酸序列数据库提交序列可以通过基于 Web 的 WEBIN 工具，也可以用 Sequin 软件来完成。

数据库网址是：http://www.ebi.ac.uk/embl/。

SRS 的网址是：http://srs.ebi.ac.uk/。

WEBIN 的网址是：http://www.ebi.ac.uk/embl/Submission/webin.html。

3. DDBJ 数据库

日本 DNA 数据仓库(DDBJ)也是一个全面的核酸序列数据库，与 Genbank 和 EMBL 核酸库合作交换数据。可以使用其主页上提供的 SRS 工具进行数据检索和序列分析。可以用 Sequin 软件向该数据库提交序列。

DDBJ 的网址是：http://www.ddbj.nig.ac.jp/。

4. GDB

基因组数据库(GDB)为人类基因组计划(HGP)保存和处理基因组图谱数据。GDB 的目标是构建关于人类基因组的百科全书，除了构建基因组图谱之外，还开发了描述序列水平的基因组内容的方法，包括序列变异和其它对功能和表型的描述。目前 GDB 中有：人类基因组区域(包括基因、克隆、amplimers PCR 标记、断点 breakpoints、细胞遗传标记 cytogenetic markers、易碎位点 fragile sites、EST 序列、综合区域 syndromic regions、contigs 和重复序列)；人类基因组图谱(包括细胞遗传图谱、连接图谱、放射性杂交图谱、content contig 图谱和综合图谱等)；人类基因组内的变异(包括突变和多态性，加上等位基因频率数据)。GDB 数据库以对象模型来保存数据，提供基于 Web 的数据对象检索服务，用户可以搜索各种类型的对象，并以图形方式观看基因组图谱。

GDB 的网址是：http://www.gdb.org。

GDB 的国内镜像是：http://gdb.pku.edu.cn/gdb/。

蛋白质数据库

1. PIR 和 PSD

PIR 国际蛋白质序列数据库(PSD)是由蛋白质信息资源(PIR)、慕尼黑蛋白质序列信息中心(MIPS)和日本国际蛋白质序列数据库(JIPID)共同维护的国际上最大的公共蛋白质序列数据库。这是一个全面的、经过注释的、非冗余的蛋白质序列数据库，其中包括来自几十个完整基因组的蛋白质序列。所有序列数据都经过整理，超过 99%的序列已按蛋白质家族分类，一半以上还按蛋白质超家族进行了分类。PSD 的注释中还包括对许多序列、结构、基因组和文献数据库的交叉索引，以及数据库内部条目之间的索引，这些内部索引帮助用户在包括复合物、酶－底物相互作用、活化和调控级联和具有共同特征的条目之间方便的检索。每季度都发行一次完整的数据库，每周可以得到更新部分。

PSD 数据库有几个辅助数据库，如基于超家族的非冗余库等。PIR 提供三类序列搜索服务：基于文本的交互式检索；标准的序列相似性搜索，包括 BLAST、FASTA 等；结合序列相似性、注释信息和蛋白质家族信息的高级搜索，包括按注释分类的相似性搜索、结构域搜索 GeneFIND 等。

PIR 和 PSD 的网址是：http://pir.georgetown.edu/。

数据库下载地址是：ftp://nbrfa.georgetown.edu/pir/。

2. SWISS-PROT

SWISS-PROT 是经过注释的蛋白质序列数据库，由欧洲生物信息学研究所(EBI)维护。数据库由蛋白质序列条目构成，每个条目包含蛋白质序列、引用文献信息、分类学信息、注释等，注释中包括蛋白质的功能、转录后修饰、特殊位点和区域、二级结构、四级结构、与其它序列的相似性、序列残缺与疾病的关系、序列变异体和冲突等信息。SWISS-PROT 中尽可能减少了冗余序列，并与其它 30 多个数据建立了交叉引用，其中包括核酸序列库、蛋白质序列库和蛋白质结构库等。

利用序列提取系统(SRS)可以方便地检索 SWISS-PROT 和其它 EBI 的数据库。SWISS-PROT 只接受直接测序获得的蛋白质序列，序列提交可以在其 Web 页面上完成。

SWISS-PROT 的网址是：http://www.ebi.ac.uk/swissprot/。

3. PROSITE

PROSITE 数据库收集了生物学有显著意义的蛋白质位点和序列模式，并能根据这些位点和模式快速和可靠地鉴别一个未知功能的蛋白质序列应该属于哪一个蛋白质家族。有的情况下，某个蛋白质与已知功能蛋白质的整体序列相似性很低，但由于功能的需要保留了与功能密切相关的序列模式，这样就可能通过 PROSITE 的搜索找到隐含的功能 motif，因此是序列分析的有效工具。PROSITE 中涉及的序列模式包括酶的催化位点、配体结合位点、与金属离子结合的残基、二硫键的半胱氨酸、与小分子或其它蛋白质结合的区域等；除了序列模式之外，PROSITE 还包括由多序列比对构建的 profile，能更敏感地发现序列与 profile 的相似性。PROSITE 的主页上提供各种相关检索服务。

PROSITE 的网址是：http://www.expasy.ch/prosite/。

4. PDB

蛋白质数据仓库(PDB)是国际上唯一的生物大分子结构数据档案库，由美国 Brookhaven 国家实验室建立。PDB 收集的数据来源于 X 光晶体衍射和核磁共振(NMR)的数据，经过整理和确认后存档而成。目前 PDB 数据库的维护由结构生物信息学研究合作组织(RCSB)负责。RCSB 的主服务器和世界各地的镜像服务器提供数据库的检索和下载服务，以及关于 PDB 数据文件格式和其它文档的说明，PDB 数据还可以从发行的光盘获得。使用 Rasmol 等软件可以在计算机上按 PDB 文件显示生物大分子的三维结构。

RCSB 的 PDB 数据库网址是：http://www.rcsb.org/pdb/。

5. SCOP

蛋白质结构分类(SCOP)数据库详细描述了已知的蛋白质结构之间的关系。分类基于若干层次：家族，描述相近的进化关系；超家族，描述远源的进化关系；折叠子(fold)，描述空间几何结构的关系；折叠类，所有折叠子被归于全 α、全 β、α/β、α＋β 和多结构域等几个大类。SCOP 还提供一个非冗余的 ASTRAIL 序列库，这个库通常被用来评估各种序列比对算法。此外，SCOP 还提供一个 PDB-ISL 中介序列库，通过与这个库中序列的两两比对，可以找到与未知结构序列远缘的已知结构序列。

SCOP 的网址是：http://scop.mrc-lmb.cam.ac.uk/scop/。

6. COG

蛋白质直系同源簇(COGs)数据库是对细菌、藻类和真核生物的 21 个完整基因组的编码蛋白，根据系统进化关系分类构建而成。COG 库对于预测单个蛋白质的功能和整个新基因组中蛋白质的功能都很有用。利用 COGNITOR 程序，可以把某个蛋白质与所有 COGs 中的

蛋白质进行比对，并把它归入适当的 COG 簇。COG 库提供了对 COG 分类数据的检索和查询，基于 Web 的 COGNITOR 服务，系统进化模式的查询服务等。

COG 库的网址是：http://www.ncbi.nlm.nih.gov/COG。

下载 COG 库和 COGNITOR 程序在：ftp://ncbi.nlm.nih.gov/pub/COG。

功能数据库

1. KEGG

京都基因和基因组百科全书(KEGG)是系统分析基因功能，联系基因组信息和功能信息的知识库。基因组信息存储在 GENES 数据库里，包括完整和部分测序的基因组序列；更高级的功能信息存储在 PATHWAY 数据库里，包括图解的细胞生化过程如代谢、膜转运、信号传递、细胞周期，还包括同系保守的子通路等信息；KEGG 的另一个数据库是 LIGAND，包含关于化学物质、酶分子、酶反应等信息。KEGG 提供了 Java 的图形工具来访问基因组图谱，比较基因组图谱和操作表达图谱，以及其它序列比较、图形比较和通路计算的工具，可以免费获取。

KEGG 的网址是：http://www.genome.ad.jp/kegg/。

2. DIP

相互作用的蛋白质数据库(DIP)收集了由实验验证的蛋白质－蛋白质相互作用。数据库包括蛋白质的信息、相互作用的信息和检测相互作用的实验技术三个部分。用户可以根据蛋白质、生物物种、蛋白质超家族、关键词、实验技术或引用文献来查询 DIP 数据库。

DIP 的网址是：http://dip.doe-mbi.ucla.edu/。

3. ASDB

可变剪接数据库(ASDB)包括蛋白质库和核酸库两部分。ASDB(蛋白质)部分来源于 SWISS-PROT 蛋白质序列库，通过选取有可变剪接注释的序列，搜索相关可变剪接的序列，经过序列比对、筛选和分类构建而成。ASDB(核酸)部分来自 Genbank 中提及和注释的可变剪接的完整基因构成。数据库提供了方便的搜索服务。

ASDB 的网址是：http://cbcg.nersc.gov/asdb。

4. TRRD

转录调控区数据库(TRRD)是在不断积累的真核生物基因调控区结构－功能特性信息基础上构建的。每一个 TRRD 的条目里包含特定基因各种结构－功能特性：转录因子结合位点、启动子、增强子、静默子、以及基因表达调控模式等。TRRD 包括五个相关的数据表：TRRDGENES(包含所有 TRRD 库基因的基本信息和调控单元信息)；TRRDSITES(包括调控

因子结合位点的具体信息)；TRRDFACTORS(包括 TRRD 中与各个位点结合的调控因子的具体信息)；TRRDEXP(包括对基因表达模式的具体描述)；TRRDBIB(包括所有注释涉及的参考文献)。TRRD 主页提供了对这几个数据表的检索服务。

TRRD 的网址是：http://wwwmgs.bionet.nsc.ru/mgs/dbases/trrd4/。

5. TRANSFAC

TRANSFAC 数据库是关于转录因子、它们在基因组上的结合位点和与 DNA 结合的 profiles 的数据库。由 Site、Gene、Factor、Class、Matrix、Cells、Method 和 Reference 等数据表构成。此外，还有几个与 TRANSFAC 密切相关的扩展库：PATHODB 库收集了可能导致病态的突变的转录因子和结合位点；S/MART DB 收集了与染色体结构变化相关的蛋白因子和位点的信息；TRANSPATH 库用于描述与转录因子调控相关的信号传递的网络；CYTOMER 库表现了人类转录因子在各个器官、细胞类型、生理系统和发育时期的表达状况。TRANSFAC 及其相关数据库可以免费下载，也可以通过 Web 进行检索和查询。

TRANSFAC 的网址是：http://www.gene-regulation.com/pub/databases.html。

其它数据库资源

1. DBCat

DBCat 是生物信息数据库的目录数据库，它收集了 500 多个生物信息学数据库的信息，并根据它们的应用领域进行了分类。包括 DNA、RNA、蛋白质、基因组、图谱、蛋白质结构、文献著作等基本类型。数据库可以免费下载或在网络上检索查询。

DBCat 的网址是：http://www.infobiogen.fr/services/dbcat/。

下载 DBCat 在：ftp://ftp.infobiogen.fr/pub/db/dbcat。

2. PubMed

PubMed 是 NCBI 维护的文献引用数据库，提供对 MEDLINE、Pre-MEDLINE 等文献数据库的引用查询和对大量网络科学类电子期刊的链接。利用 Entrez 系统可以对 PubMed 进行方便的查询检索。

PubMed 的网址是：http://www.ncbi.nlm.nih.gov/。

除了以上提及的数据之外，还有许许多多的专门生物信息数据库，涉及了目前生物学研究的各个层面和领域，由于篇幅所限无法一一详述。国内也有一些大数据库的镜像站点和自己开发的有特色的数据库，如欧洲分子生物学网络组织 EMBNet 中国节点北京大学分子生物信息镜像系统，上海博容基因公司与上海嘉瑞软件公司合作开发的国产汉化基因数据库及分析管理系统，同时国家级的生物信息学中心也在筹建之中。

第二章 牛顿迭代法

本章主要是讲非线性方程的数值解法。对于一元方程，只有几类简单情形，例如低次的代数方程和简单的超越方程，我们能有办法求出其准确解，至于一般情形，已不存在通常的求根公式，或者求准确解的方法。由于实际应用中，往往只需要能获得具有一定准确度的近似根就可以了，因此，本章主要介绍了在实际中计算机上求解方程的近似根的几个方法。其中，牛顿迭代法是应用范围最广的一种方法之一，在本章中主要讲解这种方法。同时，本章要求将牛顿迭代法编制成程序来解决一个实际问题。

2.1 闭区间套法（二分法）

2.1.1 根的隔离

我们所要介绍的求方程 $f(x)=0$ 的近似根的方法，都是基于已知方程 $f(x)=0$ 在一个区间 (a,b) 内恰好有一个根。为此，求方程 $f(x)=0$ 的根时，总是先要确定根的大概位置，也就是要确定出一个区间 (a,b)，使 (a,b) 内恰好只有 $f(x)=0$ 的一个根。这件工作，称为“根的隔离”。确定的有根区间 (a,b) 往往比较大，将有根区间逐步缩小的方法就是闭区间套法（即二分法）。

1. 隔离实根的依据

方程 $f(x)=0$ 的根的隔离，依据下列定理。

定理 2.1 设实函数 $f(x)$ 在区间 $[a,b]$ 单调连续，且有

$$f(a)f(b)<0$$

则方程 $f(x)=0$ 在区间 $[a,b]$ 内仅有一个实根。

2. 隔离实根的方法

在高等数学中，应用一元函数的导数，可以研究函数的若干性质，例如函数的凹凸性、函数的升降区间、函数的极值等等。当然，应用导数就可以确定方程 $f(x)=0$ 的左端函数 $f(x)$ 的零点所在区间，即方程 $f(x)=0$ 的根区间。当 $f(x)=0$ 不止一个根时，亦可以确

定出每个根所在的区间。

例 2.1 求方程

$$f(x)=x^3-5x^2+3x-5=0$$

的根所在区间。

解 因为

$$f'(x)=(x-3)(3x-1)$$
$$f''(x)=2(3x-5)$$

所以 $f'(3)=f'\left(\frac{1}{3}\right)=0, f''\left(\frac{5}{3}\right)=0$。于是，对于函数 $f(x)$ 我们有表 2.1。

表 2.1

区间	-∞	$(-\infty,\frac{1}{3})$	$\frac{1}{3}$	$(\frac{1}{3},\frac{5}{3})$	$\frac{5}{3}$	$(\frac{5}{3},3)$	3	(3,+∞)	+∞
$f'(x)$		+	0	-	-	-	0	+	
$f(x)$	-∞	↗	$\frac{148}{27}$	↘	$\frac{20}{27}$	↘	-4	↗	+∞
$f''(x)$		-	-	-	0	+	+	+	

从表 2.1 中即知，方程在区间

$$(-\infty,\frac{1}{3}] \qquad [\frac{1}{3},3] \qquad [3,\infty]$$

各仅有一个实根。考虑到 $f(-1)=-4$、$f(5)=20$，因此方程的有限区间分别是

$$[-1,\frac{1}{3}] \qquad [\frac{1}{3},3] \qquad [3,5]$$

确定的有限区间[a,b]比较大时，为了减少闭区间套法的计算量，我们可以用逐步搜索法将有根区间缩短。方法是，从端点 $x_0=a$ 出发，按某个预先选定的步长 h,一步步向右进行计算，即检查每一步的起点 x_0 和终点 x_0+h 的函数值是否同号。如果发生 $f(x_0)$与 $f(x_0+h)$异号，即不等式 $f(x_0)f(x_0+h)\le 0$ 成立，那么所求的根必在区间$[x_0, x_0+h]$中，这时有根区间就缩短$[x_0, x_0+h]$。缩短有根区间的基本计算步骤如下：

第一步 $x_0 \Leftarrow a$;

第二步若 $f(x_0)f(x_0+h)\le 0$ 则有根区间缩短为$[x_0, x_0+h]$，否则进行第三步；

第三步 $x_0 \Leftarrow x_0 + h$，转向第二步。

例 2.2 试缩短例 2.1 中方程的有根区间 $[-1,\frac{1}{3}]$ $[\frac{1}{3},3]$ $[3,5]$。

解：若取 $h=\frac{1}{3}$，那么按上述缩短有根区间的计算步骤计算，其结果可列表如表 2.2 所示。

表 2.2 例 2.2 缩短有限区间

x_0	-1	-2/3	-1/3	0	1/3				
$f(x_0)$	-	+							
x_0	1/3	2/3	1	4/3	5/3	2	7/3	8/3	3
$f(x_0)$	+	+	+	+	+	-			
x_0	3	10/3	11/3	4	13/3	14/3	5		
$f(x_0)$	-	-	-	+					

由表 2.2 即知，方程的有限区间分别为

$$[-1,-\frac{2}{3}] \qquad [\frac{5}{3},2] \qquad [\frac{11}{3},4]$$

2.1.2 闭区间套法

1. 闭区间套法的基本思想

经过缩短后的有根区间[a,b]内的任意一点均可作为方程的近似根，当然这样的近似根很“粗糙”，既是说精确度很低。为了提高近似根的精确度，进一步缩短有根区间，而又不具有上述缩短有根区间的计算步骤那样大的计算量，我们采用闭区间套法（即二分法）。下面介绍闭区间套法的基本思想。

假设方程 f(x)=0 在区间[a,b]上有唯一的根 α，将区间[a,b]分成两个小区间，通常采用二等分，分点为 $\frac{1}{2}(a+b)$, 计算 $f(\frac{a+b}{2})$ 的值，若碰巧 $f(\frac{a+b}{2})=0$，就得到方程的实根 $a=\frac{a+b}{2}$。否则判断不等式

$$f(a)f(\frac{a+b}{2})<0$$

是否成立，若不等式成立则记有根区间为

$$[a_1,b_1]=[a,\frac{a+b}{2}]$$

否则有根区间为

$$[a_1,b_1]=[\frac{a+b}{2},b]$$

再把区间 $[a_1,b_1]$ 二等分，分点 $\frac{1}{2}(a_1+b_1)$, 计算 $f(\frac{a_1+b_1}{2})$ 的值，若碰巧 $f(\frac{a_1+b_1}{2})=0$，就得到方程的实根 $\alpha=(\frac{a_1+b_1}{2})$。否则判断不等式

$$f(a_1)f(\frac{a_1+b_1}{2})<0$$

是否成立，若不等式成立，则记有根区间为$[a_2,b_2]=[a_1,\frac{a_1+b_1}{2}]$

否则有根区间为

$$[a_2,b_2]=[\frac{a_1+b_1}{2},b_1]$$

再把区间$[a_2,b_2]$二等分，分点为$\frac{1}{2}(a_2+b_2)$, 计算$f(\frac{a_2+b_2}{2})$的值，等等，重复上述过程，经过 n 步后我们就得到了一串有根区间，它们的关系是依次相套，即

$$[a,b][a_1,b_1]\supset[a_2,b_2]\supset\cdots\supset[a_n,b_n]$$

有根区间$[a_n,b_n]$的长度为$\frac{1}{2^n}(b-a)$。区间$[a_n,b_n]$内任意一点作为方程的近似根时，误差不超过原区间长度的$\frac{1}{2^n}$，如取$\alpha\approx\frac{1}{2}(a_n+b_n)$时，有

$$|\alpha-\frac{1}{2}(a_n+b_n)|\leq\frac{1}{2^{n+1}}(b-a) \qquad (2.1)$$

2. 闭区间套法的基本方法

由上述可知，闭区间套法是求方程 f(x)=0 的近似根的有效方法。用闭区间套法求方程 f(x)=0 的根$\alpha\in[a,b]$时，若预先给定误差不超过 ε>0,那么我们可以由不等式（2.1），得

$$\frac{1}{2^{n+1}}(b-a)\leq\varepsilon$$

解之有

$$n\geq\left[\frac{\ln(\frac{b-a}{\varepsilon})}{\ln 2}\right]$$

这样我们只需要将有根区间[a,b]二等分$\left[\frac{\ln(\frac{b-a}{\varepsilon})}{\ln 2}\right]$次，得到有根区间$[a_n,b_n]$，取其中点$\frac{a_n+b_n}{2}$作为方程的根 α 的近似值，$\alpha\approx\frac{1}{2}(a_n+b_n)$就是满足精度要求的近似根。闭区间套法求方程的近似根的优点是计算简单，收敛速度也不算太慢，与以 1/2 为公比的等比级数的收敛速度相同，但是它不适用于求二重根等等。用闭区间套法求单实根的基本计算步骤如下：

第一步　　输入有根区间的端点 a、b 及预先给定的精度 ε；

第二步　　$x \Leftarrow \frac{1}{2}(a+b)$

第三步　　若 $f(a)f(x) \le 0$ 则 $b \Leftarrow x$ 转向第四步，否则 $a \Leftarrow x$，转向第四步；

第四步　　若 $b-a<\varepsilon$，则输出方程满足精度要求的根 $x=\frac{1}{2}(a+b)$，结束，否则转向第二步。

算法 2.1　闭区间套法

1.算法入口。

2.读入数据 a, b, ε, δ。

3. $x \Leftarrow \frac{1}{2}(a+b)$。

4. $y_1 \Leftarrow f(b)$.

5. $y \Leftarrow f(x)$.

6.若 ABS(y)≤δ 或 ABS(b-x)≤ε 则取 root=x,并转向 9。

7.若 sign(y_1)sign(y)<0 则 $b \Leftarrow x$。

8.转向 3。

9.输出 root, y。

10.算法出口。

11.说明：①本算法求方程 f(x)=0 在区间[a,b]内的实根；

②ε 为根的允许误差，δ 为 f(x)=0 的允许误差；

③本算法得到的 $x_n=\frac{1}{2}(a_n+b_n)$, 一定收敛（当 n→∞）到 root。

例 2.3 求 2.1 中方程在区间 $\left[\frac{5}{3},2\right]$ 内的根的近似值，要求精确到 10^{-2}，用四位小数计算。

解: 根据精度要求预先估计所要二分的次数，按误差估计式（2.1），有

$$|a-x_n| \le \frac{1}{2^{n+1}}(b-a)$$

因 b-a=1/3,故只需要二分区间五次便可达到要求的精度 $|a-x_n| \le 0.0052$

计算结果如表 2.3 所示。

表 2.3 例 2.3 闭区间套法计算结果

n	a_n	b_n	x_n	Sing($f(x_n)$)
0	1.6667	2.0000	1.8334	-
1	1.6667	1.8334	1.7501	+
2	1.7501	1.8334	1.7918	+
3	1.7918	1.8334	1.8126	-
4	1.7918	1.8126	1.8022	+
5	1.8022	1.8126	1.8074	-

所以，原方程在区间$\left[\frac{5}{3},2\right]$内的根

$$\alpha\approx 1.8074$$

用闭区间套法可以将有根区间缩短到很小，近似根的精度可以得到提高。然而，如果要求近似根的精度比较高，应用这种方法的计算量就比较大，这时我们应当采用迭代法提高近似根的精度。

2.2 简单迭代法

2.2.1 简单跌代法的基本思想和计算步骤

简单跌代法是一种重要的逐次逼近方法。它的基本思想是：首先将方程 $f(x)=0$ 改写成某种等价形式，由等价形式构造相应的迭代格式，然后选取方程的某个初始近似根 x_0，代入迭代格式反复校正根的近似值，直到满足精度要求为止。简单迭代法的基本计算步骤如下：

第一步：构造方程 $f(x)=0$ 的等价形式 $x=\varphi(x)$；

第二步：给定 $x_0\in[a,b]$（[a,b]为 $f(x)=0$ 的有根区间），使 $f(x_0)\approx 0$

第三步：作计算 $x_{k+1}\Leftarrow\varphi(x_k),k=0,1,2,\ldots$；

第四步：当 $|x_{n+1}-x_n|<\varepsilon$ 时，取方程 $f(x)=0$ 的根 $\alpha\approx x_{n+1}$。

其中：$\varphi(x)$ 称为迭代函数，$x_{k+1}=\varphi(x_k)$ 称为迭代格式，x_0 称为迭代初值。

算法 2.2 简单迭代法

1. 算法入口。
2. 读入数据 x0（初值），ε，itmax（控制常数）
3. $itnum\Leftarrow 1$
4. $x_1\Leftarrow\varphi(x_0)$

5. 若 $fabs(x_1 - x_0) \le \varepsilon$，则取 root=x，并转向 10。

6. 若 itnum=itmax，则转向 11。

7. itnum=itnum+1。

8. $x_0 \Leftarrow x_1$

9. 转向 4。

10. 输出 root，并转向 12。

11. 打印“Vanish”标志。

12. 算法出口。

说明：

①. 本算法求方程 x=φ(x)在 x_0 附近的根；

②. x0,x1 分别为每次迭代的初值和终值；

③. ε 为根的允许误差，itmax 为最大迭代次数，“Vanish”为迭代失败标志。

例 2.3：求方程 $x^3 - x - 1 = 0$ 在 x=1.5 附近的一个根（用六位有效数字计算）

解：将原方程改写成等价形式：

$$x = \sqrt[3]{x+1}$$

从而有迭代格式

$$x_{k+1} = \sqrt[3]{x_k + 1} \qquad k = 0,1,2,\ldots$$

取迭代初值 x_0=1.5，用迭代格式反复校正方程的近似根，其各次迭代的数据结果如表 2.4。

从表 2.4 可以看出，如果只取六位有效数字，那么结果 x_7 与 x_8 已完全相同，这时可以认为 x_7 实际上已满足方程，从而得到所求方程的根为 $\alpha \approx 1.32472$

表 2.4 迭代结果

k	x_k
0	1.5
1	1.35721
2	1.33086
3	1.32588
4	1.32494
5	1.32476
6	1.32473
7	1.32472
8	1.32472

2.3 牛顿迭代法

牛顿迭代法主要就是将简单跌代法的迭代格式改变为以下格式：

$$x_{k+1} = x_k - \frac{f(x_k)}{f^{'}(x_k)} \qquad k = 0,1,2,\ldots$$

这就是著名的牛顿迭代格式。应用此格式来解方程 $f(x)=0$ 的这种方法就称为牛顿迭代法。

2.3.1 计算步骤

1. 基本牛顿迭代法计算步骤

用牛顿迭代法解方程 $f(x)=0$ 的基本计算步骤是：

第一步：给出初始近似根 x_0，及精度 ε。

第二步：计算

$$f_0 \Leftarrow f(x_0)$$
$$f_0^{'} \Leftarrow f^{'}(x_0)$$
$$x_1 \Leftarrow x_0 - f_0 \Big/ f_0^{'}$$

第三步：判断，若 $|x_1 - x_0| < \varepsilon$，则转向第四步，否则 $x_0 \Leftarrow x_1$，转向第二步。

第四步：输出满足精度的根 x_1，即 $\alpha \approx x_1$，结束。

2. 常用牛顿迭代法计算步骤

控制迭代终止，如果不单用 $|x_{k+1} - x_k| < \varepsilon$，我们有下列计算步骤：

第一步：给出初始近似根 x_0，及精度 ε_1，ε_2；并计算：

$$f_0 \Leftarrow f(x_0)$$
$$f_0^{'} \Leftarrow f^{'}(x_0)$$

第二步：作一次迭代计算：

$$x_1 \Leftarrow x_0 - f_0 \Big/ f_0^{'}$$

得到新的近似根 x_1，并计算：

$$f_1 \Leftarrow f(x_1)$$
$$f_1^{'} \Leftarrow f^{'}(x_1)$$

第三步：如果 x1 满足

$$|\delta| < \varepsilon_1 \text{ 或者} |f_1| < \varepsilon_2$$

则终止迭代，以 x_1 作为所求的近似根，即 $\alpha \approx x_1$；否则转第四步。这里

$$\delta = \begin{cases} |x_1 - x_0| & |x_1| < C \\ \dfrac{|x_1 - x_0|}{|x_1|} & |x_1| \geq C \end{cases}$$

其中 C 是取绝对误差或相对误差的控制常数，一般可取 C=1。

第四步：如果迭代次数达到预先指定的次数 N，或者 $f_1' = 0$，则方法失败；否则：

$$x_0 \Leftarrow x_1$$
$$f_0 \Leftarrow f_1$$
$$f_0' \Leftarrow f_1'$$

再转第二步继续迭代。

算法 2.3　牛顿迭代法

1. 算法入口。
2. 读入数据 x_0（初值），ε，itmax、C（控制常数）
3. $itnum \Leftarrow 1$ 并 $f_1 \Leftarrow f(x_0)$。
4. $denom \Leftarrow f'(x_0)$。
5. 若 *denom*=0 则转向 14。
6. $x_1 \Leftarrow x_0 - f_1 / denom$
7. $f_1 \Leftarrow f(x_1)$
8. 若 $fabs(x_1) \leq C$，则取 root=x，并转向 10。
9. 若 ABS(d)< ε_1,或 ABS(f_1)< ε_2,则取 root=x_1,并转向 15。
10. 若 itnum=itmax，则转向 16。
11. itnum ⇐ itnum+1。
12. $x_0 \Leftarrow x_1$
13. 转向 4。
14. 打印“singular”标志。
15. 输出 root，并转向 17。
16. 打印“Vanish”标志。

17. 算法出口。

18. 说明：①本算法求方程 f(x)=0 在 x_0 附近的实根；

② ε_1、ε_2 为误差界；

③x_0、x_1、itmax、“vanish”与算法 2.2 同，“singular”为奇异标志。

2.3.2 如何选取初值 x_0

要使应用牛顿迭代法格式在非局部的区间[a,b]上收敛，条件比较苛刻，判断起来常常并不容易，因而对此我们不作深入探讨。由于牛顿迭代法求单根具有局部收敛性，而且收敛速度很快，所以我们只需要注意选取初始值 x_0，就能得到满意的结果。一般选取的初值 x_0 应满足

$$\left|f'(x_{0)}\right|^2 < \left|\frac{f(x_0)f''(x_0)}{2}\right|$$

例 2.4 计算 $\sqrt{0.78265}$ 的近似值，$\varepsilon=10^{-6}$。

解：令 $x=\sqrt{c}$, 则问题化为求方程 $f(x)=x^2-c=0$ 的正根。建立方程的牛顿迭代法格式是：

$$x_{k+1} = x_k - \frac{x_k^2 - c}{2x_k} = \frac{1}{2}\left(x_k + \frac{c}{x_k}\right)$$

这里 c=0.78265,从简单的平方根表中查得 $x_0=0.88$,以此作为初值，进行迭代，计算结果如表 2.5。可以看到，只用三次迭代就得到了满足精度要求的结果，取 $\sqrt{0.78265} \approx 0.884675$。这是平方根精确化的有效且简单的方法。

表 2.5 例 2.4 迭代结果

k	x_k
0	0.88
1	0.884688
2	0.884675
3	0.884675

2.3.3 几何意义

牛顿迭代法有明显的几何意义，见图 2.1。我们知道，方程 f(x)=0 的根 α 在几何图形上表示曲线 y=f(x)与轴的交点的横坐标。设 x_k 是 α 的某个近似未知的横坐标，过曲线 y=f(x)上的对应点 $P_k(x_k)$, $f(x_k)$引切线，其方程为

$$y = f(x_k) + f'(x_k)(x - x_k)$$

切线与 x 轴的交点横坐标为 x_{k+1}，则点 x_{k+1} 必满足该切线方程，即是

$$y = f(x_k) + f'(x_k)(x_{k+1} - x_k)$$

这正是牛顿迭代法格式。因此，牛顿迭代法又称为切线法。

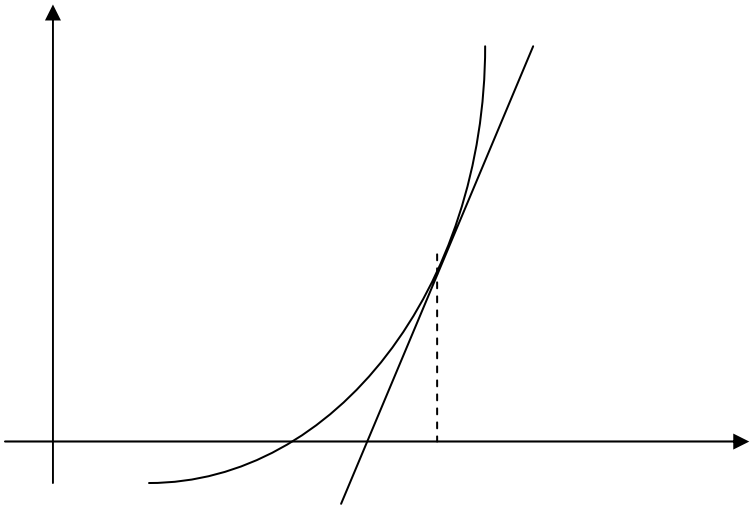


图 2.1

作业：求方程 x-ln3x=0 的两个根，利用简单跌代法和牛顿迭代法编程。

（根在 0.5 和 1.0 附近）

第三章 线性方程组的数值解法

在工程和科学计算中，许多问题的解决往往归结为解一个线性代数方程组。线性代数方程组的数值解决主要有两类；一类称为直接解法，直接解法是一种变换法，通过对原方程组的有限次变换，在没有原始数据误差和舍入误差的情况下，将能得到方程的精确解；另一类称为迭代法，把线性代数方程组的解看作是某个极限过程的极限，从所给初始向量出发，产生一个近似解向量序列，逐步逼近准确解，用有限个步骤计算出准确解的具有给定精确度的近似值。当然，不论用哪一种方法求解，首先须判明线性代数方程组有唯一解，即对于 n 阶线性代数方程组

$$Ax=b \qquad (3.1)$$

其系数矩阵 A 为非奇异，亦即 detA≠0,这时唯一解是

$$X^n=A^{-1}b \qquad (3.2)$$

本章主要介绍求解线性代数方程组的直接解法和迭代解法中的几个主要算法。

3.1 高斯消去法

3.1.1 基本思想

1. 当 A 为下三角矩阵

线性代数方程组中，若系数矩阵 A 为一下三角阵时，即式（3.1）有形式

$$\begin{bmatrix} a_{11} & & 0 \\ a_{21} & a_{22} & \\ \vdots & \vdots & \ddots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \bullet \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \qquad (3.3)$$

则我们可以从第一个方程中求出 x_1;代入第二个方程中求出 x_2; 将 x_1、x_2 代入第三个方程中求出 x_3；依次类推，直到将 x_1、x_2…x_{n-1} 代入第 n 个方程中求出 x_n,从而很容易第求得 x。这个求解过程可

2. 当 A 为上三角矩阵

线性代数方程组中，若系数矩阵 A 为一上三角阵时，即式（3.1）有形式

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ & a_{22} & & \vdots \\ & & \ddots & \vdots \\ & 0 & & a_{nn} \end{bmatrix} \bullet \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \qquad (3.4)$$

则我们可以从第 n 个方程中求出 x_n；代入第 $n-1$ 个方程中求出 x_{n-1}；将 x_n、x_{n-1} 代入倒数第三个方程中求出 x_{n-2}；依次类推，直到将 x_n、x_{n-1}、…、x_2 代入第一个方程中求出 x_1，从而即得方程得解向量。这个求解过程称为回代过程，可表示为数学算式

$$\begin{cases} x_n = b_n / a_{nn} \\ x_k = \left(b_k - \sum_{j=k+1}^{n} a_{kj} x_j \right) \Big/ a_{kk} \qquad k = n-1, n-2, \cdots, 1 \end{cases} \qquad (3.5)$$

3．当 A 为一般矩阵

一般地，若系数矩阵 A 不是一个三角矩阵，那么将 A 化成一个上三角阵，即将方程组式（3.1）化成方程组式（3.4）的形式（此过程称为消去过程），再按方程组式（3.4）的求解算式（3.5）计算，即可求出方程组式（3.1）的解向量 x。这种解线性代数方程组的方法称为高斯消去法。

3.1.2 基本方法

解线性代数方程组式（3.1）的高斯消去法，主要分为两个过程，消去过程和回代过程。

（1） 消去过程

高斯消去法的消去过程，是应用矩阵的初等变换将系数矩阵 A 化为上三角阵，与此同时将方程组的右端向量 b 增补作为 A 的第 n＋1 列，构成增广矩阵，同时参加变换，即将式（3.1）化为式（3.4）的形式。

消去过程共有 n－1 个步骤，每个步骤都是应用矩阵的初等行变换，将矩阵的某一列的主对角元以下的元素化为零。第一步将第一列主对角元以下 n－1 个元素化为零；第二步将第二列主对角元以下 n－2 个元素化为零；……；第 k 步将第 k 列主对角元以下 n－k 个元素化为零；……；第 n－1 步将第 n－1 列主对角元以下的 1 个元素化为零。这些变换同时施予第 n＋1 列。这个过程可表示为

$$A\sim\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} & a_{1,n+1}^{1} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} & a_{2,n+1}^{1} \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} & a_{n,n+1}^{1} \end{bmatrix}\sim\begin{bmatrix} a_{11}^{(k)} & \cdots & a_{1k}^{(k)}\cdots & a_{1n}^{(k)}\cdots & a_{1,n+1}^{(k)} \\ & \ddots & a_{kk}^{(k)} & \cdots & a_{kn}^{(k)}\cdots & a_{k,n+1}^{(k)} \\ & & 0 & & & \\ 0 & & \vdots & & \vdots & \vdots \\ & & 0 & \cdots & a_{nn}^{k} & a_{n,n+1}^{(k)} \end{bmatrix} \tag{3.6}$$

$$\sim\begin{bmatrix} a_{11}^{(n-1)} & \cdots & a_{1,n-1}^{(n-1)} & a_{1n}^{(n-1)} & a_{1,n+1}^{(n-1)} \\ & \ddots & \vdots & \vdots & \vdots \\ & & \vdots & \vdots & \vdots \\ 0 & & a_{n-1,n-1}^{(n-1)} & a_{n-1,n}^{(n-1)} & a_{n-1,n+1}^{(n-1)} \\ & & 0 & a_{nn}^{(n-1)} & a_{n,n+1}^{(n-1)} \end{bmatrix}$$

其中 $a_{ij}^{(k)}$ 表示经 k 次消元之后单元 a_{ij} 的值。

根据线性代数知识，我们不难写出消去过程的数学算式

$$\begin{cases} \text{对于} k=1,2,\cdots,n-1 \\ a_{ij}^{(k)}=a_{ij}^{(k-1)}-\left(\dfrac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}\right)a_{kj}^{(k-1)} \qquad i-k+1,\cdots,n \quad j=k,\cdots,n+1 \end{cases} \tag{3.7}$$

（2）回代过程

将线性代数方程组的增广矩阵，化为式(3.6)的最后一个矩阵的形式之后，再应用回代过程的数学算式(3.5)，在此写为

$$\begin{cases} x_n=a_{n,n+1}^{(n-1)}\Big/a_{nn}^{(n-1)} \\ x_k=(a_{k,n+1}^{(n-1)}-\sum_{j=k+1}^{n}a_{kj}^{(n-1)}x_j)\Big/a_{kk}^{(n-1)} \qquad k=n-1,\cdots,1 \end{cases} \tag{3.8}$$

由此可见，用高斯消去法解一个线性代数方程组，**计算步骤**十分简单：首先计算式(3.7)，对增广矩阵进行初等变换，使之化为上三角阵；然后计算式(3.8)，逐个求出 $x_n,x_{n-1},\cdots,x_1$，即得方程组的解。

算法 3.1　高斯消去法

1. 算法入口。
2. 读入 $A_{n\times(n+1)}$ 的数据。
3. 消去过程 4～12。
4. 对于 k＝1,2,…,n－1 执行到 12。
5. 若 a_{kk}=0 则转向 24。
6. 对于 i=k+1,k+2,…,n 执行到 11。

7. $a_{ik} \Leftarrow a_{ik}/a_{nk}$。

8. 对于 i=k+1,k+2,…,n－1 执行到 10。

9. $a_{ij} \Leftarrow a_{ij} - a_{ik}\, a_{kj}$

10. Continue。

11. Continue。

12. Continue。

13. 回代过程 14～22。

14. 若 $a_{nn}=0$ 则转向 24。

15. $x_n \Leftarrow a_{n,n+1}/a_{nn}$。

16. 对于 k=n－1,n－2,…,1 执行到 22。

17. $B \Leftarrow a_{k,n+1}$

18. 对于 j=k＋1,k＋2,…,n 执行到 20。

19. $B \Leftarrow B - a_{kj}x_j$。

20. Continue。

21. $x_k \Leftarrow B/a_{kk}$。

22. Continue。

23. 输出解 x，并转向 25。

24. 打印“Singular”标志。

25. 算法出口。

26. 说明：①本算法求解方程组 Ax=b 的解 x；

②$A_{n\times(n+1)}$是 A 的增广矩阵，第 n＋1 列存放 b，消去过程和回代过程都是在增广矩阵 A 中进行，解在 x 中；

③“Singular”为奇异标志。

3.1.3 高斯消去法的矩阵形式

由于高斯消去法中消去过程，是对增广矩阵[*A*,*b*]作 n－1 次初等行变换，每次初等行变换后产生出一列中所需零元素。由线性代数知识，作一次初等行变换，相当于左乘以一个初等矩阵，因此消去过程可用矩阵运算表示为

$$L_{n-1}\cdots L_2L_1[A,b]=\begin{bmatrix} a_{11}^{(n-1)} & a_{12}^{(n-1)} & \cdots & a_{1n}^{(n-1)} & b_1^{(n-1)} \\ & a_{22}^{(n-1)} & \cdots & a_{2n}^{(n-1)} & b_2^{(n-1)} \\ & & \ddots & \vdots & \vdots \\ & 0 & & a_{nn}^{(n-1)} & b_n^{(n-1)} \end{bmatrix} \tag{3.9}$$

其中，$L_j(j=1,2,\cdots,n-1)$为初等矩阵，对应于消去过程中第 j 步所作矩阵初等变换。若记式(3.9)等号右端的上三角阵为 R，第 n＋1 列为列矩阵 y，并记 $L=L_{n-1}L_{n-2}\cdots L_1$，则式(3.9)可简写为

$$L^{-1}[A,b]=[R,y] \tag{3.10}$$

即有

$$\begin{matrix} L^{-1}A=R \\ L^{-1}b=y \end{matrix}\Rightarrow\begin{matrix} A=LR \\ Ly=b \end{matrix}\Rightarrow LRx=Ly\Rightarrow Rx=y \tag{3.11}$$

由此说明，如果 A＝LR，则解线性代数方程组的过程，可以写成两步：

第一步　求列向量 y　Ly＝b

第二步　求解向量 x　Rx＝y

事实上，其中第一步包含在消去过程中，第二步就是回代过程。

式(3.10)中，初等矩阵 L_k 的具体形式为

$$L_k=\begin{bmatrix} 1 & & & & 0 \\ & \ddots & & & \\ & & 1 & & \\ & & -l_{k+1,k} & 1 & \\ 0 & & \vdots & & \ddots \\ & & -l_{nk} & 0 & 1 \end{bmatrix}\quad k=1,2,\cdots,n-1 \tag{3.12}$$

其中 $l_{ik}=a\,l_{ik}=a_{ik}^{(k-1)}/a_{kk}^{(k-1)}$,i=k+1,⋯,n。从而

$$L=(L^{-1})^{-1}=L_1^{-1}L_2^{-1}\ldots L_{n-1}^{-1}$$

$$L_k=\begin{bmatrix} 1 & & & & 0 \\ & \ddots & & & \\ & & 1 & & \\ & & l_{k+1,k} & 1 & \\ 0 & & \vdots & & \ddots \\ & & l_{nk} & 0 & 1 \end{bmatrix} \tag{3.13}$$

因此，高斯消去法的矩阵表达形式：消去过程是式(3.10)；回代过程是式(3.11)。

例3.1　用高斯消去法解方程组

$$\begin{cases} x_1 + 2x_2 - x_3 = 3 \\ x_1 - x_2 + 5x_3 = 0 \\ 4x_1 + x_2 - 2x_3 = 2 \end{cases} \tag{3.14}$$

解　消去过程

$$\begin{bmatrix} 1 & 2 & -1 & 3 \\ 1 & -1 & 5 & 0 \\ 4 & 1 & -2 & 2 \end{bmatrix} \sim \begin{bmatrix} 1 & 2 & -1 & 3 \\ 0 & -3 & 6 & -3 \\ 0 & -7 & 2 & -10 \end{bmatrix} \sim \begin{bmatrix} 1 & 2 & -1 & 3 \\ 0 & -3 & 6 & -3 \\ 0 & 0 & -12 & -3 \end{bmatrix}$$

回代过程：

$$x_3 = 3 \Big/ (-12) = \frac{1}{4}$$

$$x_2 = \left[(-3) - 6 \times \frac{1}{4} \right] \Big/ (-3) = \frac{3}{2}$$

$$x_1 = 3 - 2 \times \frac{3}{2} - (-1) \times \frac{1}{4} = \frac{1}{4}$$

所以，方程组式（3.14）的解为

$$x_1 = \frac{1}{4}, \quad x_2 = \frac{3}{2}, \quad x_3 = \frac{1}{4}$$

3.2 主元素消去法

根据数值运算若干原则之一“要避免绝对值很小的数作除数”，在高斯消去法消去过程中，为避免作除数的主对角线上的元素的绝对值过小，我们需采用选主元技术。

位于主对角线上的在消去过程中用作除数的元素称为主元素，简称主元。

选主元技术是在消去过程的 n－1 步中，每步总是选绝对值最大的元素，经过矩阵的行、列变换，变换至主对角线上，作为主元。由于选主元的范围不同，构成解线性代数方程组的列主元消去法和全主元消去法。

3.2.1 列主元消去法

所谓列主元消去法，是在高斯消去法的消去过程中，第 k 步(k=1,2,…,n—1)增加选主元操作，成为：首先在第 k 列中，从 $a_{kk}^{(k-1)}$，$a_{k+1,k}^{(k-1)}$，…，$a_{nk}^{(k-1)}$ 中选出绝对值最大者，经过行

交换（把绝对值最大者所在的行与第 k 行交换）把绝对值最大的元素交换到a_{kk}的单元中；然后作高斯消去法消去过程中的第 k 步，初等变换产生第 k 列的 n－k 个零元素。回代过程与高斯消去法的回代过程相同。

列主元消去法的基本计算步骤归纳为：

(1) 消去过程

对于 k＝1，2，…，n－1 做

第一步　选主元：$\omega \Leftarrow a_{kk}^{(k-1)}$；$t \Leftarrow k$;

若$\left|a_{ik}^{(k-1)}\right| > |\omega|$，则$\omega \Leftarrow a_{ik}^{(k-1)}$；$t \Leftarrow k$，i=k+1,k+2,…,n

第二步　判断：若$\omega = 0$说明方程组的系数矩阵是奇异阵，则终止计算，否则转第三步。

第三步　行交换：若 t≠k 则

$$u \Leftarrow a_{kj}^{(k-1)};\quad a_{kj}^{(k-1)} \Leftarrow a_{ij}^{(k-1)};\quad t\,a_{ij}^{(k-1)} \Leftarrow u$$

j=k，k＋1，…，n＋1

否则直接转第四步。

第四步　计算：$a_{ij}^{(k)} \Leftarrow a_{ij}^{(k-1)} + a_{kj}^{(k-1)}\left(-\frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}\right)$

i＝k＋1，k＋2，…，n　j＝k，k＋1，…，n＋1

(2) 回代过程

(同于式(3.9))

算法 3.2　列主元消去法

1．　算法入口。

2．　读入$A_{n\times(n+1)}$的数据。

3．　消去过程 4～15。

4．　对于 k＝1,2,…,n－1 执行到 15。

5．　$\omega \Leftarrow a_{kk}$，并$t \Leftarrow k$。

6．　对于 i=k+1,k+2,…,n 执行到 8。

7．　若 ABS(a_{kk})>ABS(ω)则$\omega \Leftarrow a_{ik}$，并$t \Leftarrow i$。

8. Continue。

9. 若 ω ＝0 则转向 20。

10 若 t＝k，则转向 14。

11 对于 j＝k,k+1,…,n+1 执行列 13。

12 $a_{kj} \Leftrightarrow ai_j$（两值交换）

13 Continue。

14 算法 3.1 步 6～11。

15 Continue。

16 回代过程 17～18。

17 若 $a_{nn} = 0$ 则转向 20。

18 算法 3.1 步 15～22。

19 输出解 x，并转向 21。

20 打印“Singular”标志。

21 算法出口。

22 说明：① 本算法求 Ax＝b 的解 x；

② 步 5～13 为选主元和行交换程序；

③ $A_{n\times(n+1)}$、“Singular”与算法 3.1 同。

3.2.2 全主元消去法

所谓全主元消去法，是在高斯消去法的消去过程中，第 k 步（k＝1，2，…，n－1）增加选主元操作，成为：首先在子矩阵

$$\begin{bmatrix} a_{kk}^{(k-1)} & \cdots & a_{kn}^{(k-1)} \\ \vdots & & \vdots \\ a_{nk}^{(k-1)} & \cdots & a_{nn}^{(k-1)} \end{bmatrix}$$

中，选出绝对值最大的元素，经过行、列交换（把绝对值最大的元素所在的行与第 k 行交换，再把绝对值最大的元素所在的列与第 k 列交换），把觉得值最大的元素交换到 a_{kk} 的位置；然后作高斯消去过程中第 k 步，初等变化产生第 k 列的 n－k 个零元素。回代过程与高斯消去法的回代过程相同。

列交换将改变未知数的排列顺序！

由于全主元消去法选主元的范围比列主元消去法大，除进行行行交换之外，一般还进行列交换，因此对应的未知数的排列顺序一般与原来不同，进行交换的两列所对应的未知数也作了对调。例如，若系数矩阵 A 的第 k 列与第 s 列进行了交换，那么 x_k 与 x_s 也相应地作了交换，x_k 的结果实质上是 x_s 的结果。因而需得在回代过程所得结果中再作相应得交换，恢复原来的顺序，从而得到方程组得实际解。

全主元消去法得基本计算步骤归纳为：

（1） 消去过程

对于 k＝1，2，…，n－1 做

第一步 选主元：$\omega \Leftarrow 0; t \Leftarrow 0; s \Leftarrow 0;$

若 $\left|a_{ij}^{(k-1)}\right| \rangle |\omega|$ 则 $\omega \Leftarrow a_{ij}^{(k-1)}; t \Leftarrow i; s \Leftarrow j$

$$i = k, k+1, \cdots, n$$
$$j = k, k+1, \cdots, n$$

第二步 判断：若 w＝0，说明方程做的系数矩阵是奇异阵，则终止计算，否则转第三步。

第三步 行交换：若 $t \neq k$ 则

$$u \Leftarrow a_{kj}^{(k-1)}; a_{kj}^{(k-1)} \Leftarrow a_{ij}^{(k-1)}; a_{ij}^{(k-1)} \Leftarrow u$$

$$j = k, k+1, \cdots, n+1$$

否则直接转第四步。

第四步 列交换：$e_k \Leftarrow k;$若$s \neq k$则$e_k = s$

$$v \Leftarrow a_{ik}^{(k-1)}; a_{ik}^{(k-1)} \Leftarrow a_{is}^{(k-1)}; a_{is}^{(k-1)} \Leftarrow v$$

否则直接转第五步。

第五步 计算：$a_{ij}^{(k)} \Leftarrow a_{ij}^{(k-1)} + a_{kj}^{(k-1)}\left(-\frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}\right)$

$$i = k+1, k+2, \cdots, n$$
$$j = k, k+1, \cdots, n+1$$

（2） 回代过程

（同于式（3.9））

（3） 理顺 x_j：对于k＝n－1，n－2,…,1做

$$若 e_k \neq k 则 y \Leftarrow x_{e_k}；\ x_{e_k} \Leftarrow x_k；\ x_k \Leftarrow y$$

算法 3.3 全主元消去法

1. 算法入口。
2. 读入 $A_{n*(n+1)}$ 的数据。
3. 消去过程 4～21。
4. 对于 k＝1，2，…，n-1 执行到 21。
5. ω⇐a_{kk}，并 t⇐k，且 s⇐k。
6. 对于 i＝k，k＋1，…，n 执行到 10。
7. 对于 r＝k，k＋1，…，n 执行到 9。
8. 若 ABS（ω）<ABS（a_{ir}）则 ω⇐a_{ir}，并 t⇐i，且 s⇐r。
9. Continue。
10. Continue。
11. 若 ω＝0 则转向 30。
12. 若 t＝k 则转向 14。
13. 算法 3.2 步 11～13。
14. e_k⇐k。
15. 若 s＝k 则转向 20。
16. e_k⇐s。
17. 对于 i＝1，2，…，n 执行到 19。
18. a_{ik}⇔a_{is}（两值交换）。
19. Continue。
20. 算法 3.2 步 14。
21. Continue。
22. 回代过程 23～24。
23. 若 a_{nn}＝0 则转向 30。
24. 算法 3.2 步 18。
25. 理顺 x_k（k＝1，2，…，n）的顺序 26～29。

26. 对于 k＝n-1，n-2，…，1 执行到 28。

27. 若 $e_k \neq k$ 则 $j \Leftarrow e_k$，并 $x_j \Leftrightarrow x_k$。

28. Continue。

29. 输出解 x，并转向 21。

30. 打印“Singular”标志。

31. 算法出口。

32. 说明：①本算法求方程组 Ax＝b 的解 x；

②步 5～19 为选主元和行、列交换程序；

③e_k 存放列交换信息；

④$A_{n*(n+1)}$，“Singular”与算法 3.1 同。

例 3.2 用全主元法消去法解方程组

$$\begin{cases} x_1 + 2x_2 + 3x_1 = 1 \\ 7x_1 + 8x_2 + 11x_3 = -3 \\ 5x_1 + x_2 - 3x_3 = -4 \end{cases} \qquad (3.15)$$

解 消去过程：（用 i←－－→j 表示列交换）

$$\begin{bmatrix} 1 & 2 & 3 & 1 \\ 7 & 8 & 11 & -3 \\ 5 & 1 & -3 & -4 \end{bmatrix} \overset{1\leftarrow\!-\!-\!\rightarrow 3}{\sim} \begin{bmatrix} 11 & 8 & 7 & -3 \\ 3 & 2 & 1 & 1 \\ -3 & 1 & 5 & -4 \end{bmatrix} \sim \begin{bmatrix} 11 & 8 & 7 & -3 \\ 0 & -\frac{2}{11} & -\frac{10}{11} & \frac{20}{11} \\ 0 & \frac{35}{11} & \frac{76}{11} & -\frac{53}{11} \end{bmatrix} \overset{2\leftarrow\!-\!-\!\rightarrow 3}{\sim}$$

$$\begin{bmatrix} 11 & 7 & 8 & -3 \\ 0 & \frac{76}{11} & \frac{35}{11} & -\frac{53}{11} \\ 0 & -\frac{10}{11} & -\frac{2}{11} & \frac{20}{11} \end{bmatrix} \sim \begin{bmatrix} 11 & 7 & 8 & 5 \\ 0 & \frac{76}{11} & \frac{35}{11} & -\frac{53}{11} \\ 0 & 0 & \frac{9}{38} & \frac{45}{38} \end{bmatrix}$$

回代过程：

$$x_3 = \frac{\frac{45}{38}}{\frac{9}{38}} = 5$$

$$x_2=\left(-\frac{53}{11}-\frac{35}{11}\times 5\right)\Big/\frac{76}{11}$$

$$=-3$$

$$x_1=\left[-3-8\times 5-7\times(-3)\right]\Big/11$$

$$=-2$$

所以 $$x=\begin{bmatrix}-2\\-3\\5\end{bmatrix}$$

理顺 x_i，求方程组的真实解：由回代过程所得解向量 x，并不是方程组的真实解，但只需将 x 按列交换的相反次序作行的交换，就能得真实解，于是（用 i←→j 表示行交换）

$$\begin{bmatrix}-2\\-3\\5\end{bmatrix}2\longleftrightarrow 3\sim\begin{bmatrix}-2\\5\\-3\end{bmatrix}1\longleftrightarrow 3\begin{bmatrix}-3\\5\\-2\end{bmatrix}$$

所以 $x_1=-3$，$x_2=5$，$x_3=-2$

作业：对例 3.2 数据进行编程求解，方法选用全主元消去法。

第四章 矩阵求拟

4.1 高斯-约当消去法解式（3.1）

解线形方程组的高斯消去法，是将方程组化为方程组式（3.5）的形式，再经回代过程求出方程组的解。其实回代过程也可以放在消去过程来完成，即消去过程中也把主对角线上方的元素化为零，使系数矩阵化为一个对角矩阵，即增广矩阵化为：

$$\begin{bmatrix} a_{11}^{(n)} & & & 0 & y_1 \\ & a_{22}^{(n)} & & & y_2 \\ & & \ddots & & \vdots \\ 0 & & & a_{nn}^{(n)} & y_n \end{bmatrix}$$

这时方程组的解是

$$x_k = y_k \big/ a_{kk}^{(n)} \qquad k = 1,2,\cdots,n$$

这种消去法称为高斯-约当消去法。高斯-约当消去法是无回代过程消去法。

高斯-约当消去法看似一种很可取的方法，但经计算，它的乘除次数为 $1/2*n^3$ 数量级，而高斯消去法等其他方法的乘除次数为 $1/3*n^3$。当 n 较大时，高斯-约当消去法比高斯消去法运算次数增加是惊人的。但是，用它来求逆矩阵确是行之有效的方法，比用伴随矩阵求逆矩阵要快得多，后者所作乘除次数约为 $n^2*n!$。

4.2 求逆矩阵的高斯-约当消去法

1. 基本思想

高斯-约当消去法是将矩阵 $\boldsymbol{A}$ 化为对角阵，进而化为单位阵 $\boldsymbol{E}$。根据线形代数知识，将 $\boldsymbol{A}$ 化为单位阵的一系列初等变换，同时依次施予单位阵 $\boldsymbol{E}$，则可将 $\boldsymbol{E}$ 化为 $\boldsymbol{A^{-1}}$。又知，对矩阵的一系列初等变换相当于对矩阵左乘一系列初等矩阵，因而有：

若 $\boldsymbol{A}$ 非奇异，$\boldsymbol{AA^{-1}=E}$, $\boldsymbol{M_1}$, $\boldsymbol{M_2}$, …, $\boldsymbol{M_n}$ 是初等矩阵，且 $\boldsymbol{M_n\cdots M_2M_1A=E}$, 则

$$\boldsymbol{M_n\cdots M_2M_1A\,A^{-1}= M_n\cdots M_2M_1E}$$

所以：

$$\boldsymbol{A^{-1}= M_n\cdots M_2M_1E}$$

2. 不选主元的高斯-约当消去法

若令初等矩阵为

$$M_k=\begin{bmatrix}1 & & 0 & m_{1k} & & \\ & \ddots & & \vdots & & 0 \\ & & 1 & & & \\ & & & m_{kk} & & \\ & & & \vdots & 1 & \\ & 0 & & & & \ddots \\ & & & m_{nk} & & 0 \quad 1\end{bmatrix} \qquad k=1,2,\cdots,n$$

其中

$$m_{kk}=1/a_{kk}^{(k)}$$
$$m_{ik}=-a_{ik}^{(k)}/a_{kk}^{(k)} \qquad i=1,\cdots,k-1,k+1,\cdots n$$

在这里 $a_{ik}^{(k)}(i=1,2,\cdots,n))$ 是 $\boldsymbol{A}$ 经左乘 $\boldsymbol{M}_1,\ldots\boldsymbol{M}_{k-1}$ 之后的 a_{ik} 单元内的值。

依次用 $\boldsymbol{M}_1, \boldsymbol{M}_2, \ldots, \boldsymbol{M}_n$ 左乘$[\boldsymbol{A},\boldsymbol{E}]$就有

$$\boldsymbol{M_n}\ldots\boldsymbol{M_2M_1}[\boldsymbol{A},\boldsymbol{E}]=[\boldsymbol{E},\boldsymbol{A^{-1}}]$$

考虑到增广矩阵$[\boldsymbol{A},\boldsymbol{E}]$，在每左乘一个初等矩阵 $\boldsymbol{M}_k$ 之后，$2n$ 列中总是有单位阵的 n 个列存在，因此可采用紧缩存储格式：左乘 $\boldsymbol{M}_1$ 之后，$\boldsymbol{A}$ 的第一列变为单位阵 $\boldsymbol{E}$ 的第 1 列，增广矩阵$[\boldsymbol{A},\boldsymbol{E}]$的第 n+1 列（即原单位阵的第 1 列）变为$(m_{11}m_{21}\ldots m_{n1})^T$，因此可将$(m_{11}m_{21}\ldots m_{n1})^T$ T 放在 $\boldsymbol{A}$ 的第一列；左乘 $\boldsymbol{M}_2$ 后，$\boldsymbol{A}$ 的第 2 列变成为 $\boldsymbol{E}$ 的第 2 列，增广矩阵的第 n+2 列（即原单位阵的第 2 列）变为$(m_{12}m_{22}\ldots m_{n2})^T$，因此可将放入 $\boldsymbol{A}$ 的第 2 列；等等。即在变换中，无须存储 $\boldsymbol{E}$ 的各列，变化可只在矩阵 $\boldsymbol{A}$ 的各列中进行。

于是，不选主元的高斯-约当消去法求逆矩阵，采用紧缩存储格式的基本计算步骤可归纳为：

对于 k=1,2,…,n 做

第一步　　计算 k 列元素：

$$\alpha \Leftarrow 1/a_{kk};\qquad a_{kk}\Leftarrow\alpha;\qquad a_{ik}\Leftarrow-\alpha\, a_{ik}\quad(i\neq k)$$

第二步　　计算除第 k 列、k 行以外的其余元素：

$$a_{ij}\Leftarrow a_{ij}+a_{ik}a_{kj}\quad(i,j\neq k)$$

第三步　　计算第 k 行其余元素：

$$a_{kj}\Leftarrow\alpha a_{kj}\quad(j\neq k)$$

3. 选主元的高斯-约当消去法

与高斯消去法一样，为了避免小主元，保证算法的稳定性，常常需采用选列主元技术。计算中，每进行一列的消元之前，增加选主元与行交换的步骤。与不选主元的高斯-约当消去法不同之处还在于，当 n 步消元结束之后，尚需按行交换的相反次序作列交换才能得到 $\boldsymbol{A}^{-1}$，这是因为：如果第 k 次消元（左乘 $\boldsymbol{M}_k$）时，主元在第 i 行，则第 i、k 两行交换，消去后列向量$(m_{1k}m_{2k}\ldots m_{nk})^T$ 应放到增广矩阵$[\boldsymbol{A},\boldsymbol{E}]$右半部的第 i 列，但现在的算法是将放到了增广矩阵$[\boldsymbol{A},\boldsymbol{E}]$右半部的第 k 列，要得到 $\boldsymbol{A}^{-1}$，当然需将最终将第 i、k 两列交换过来。至于按行交换的相反次序作列交换的反序问题，是不言而喻的。

选主元的高斯-约当消去法，也可采用紧缩存储方案，这时基本计算步骤可归纳为：

I　对于 k=1,2,…,n 做

第一步　选主元：

$$w \Leftarrow a_{kk};\quad I_0 \Leftarrow k$$

对于 J=k+1, k+2,…,n 若 $|a_{jk}| > |w|$

则 $w \Leftarrow a_{jk};\quad I_0 \Leftarrow j$

第二步　把第 k 行与第 I_0 行交换

若 $I_0 \neq k$ 则记下所交换行行号 $T_k \Leftarrow I_0$；

并对于 J=1,2,…,n 做

$$u \Leftarrow a_{kj};\quad a_{kk} \Leftarrow a_{I_0 j};\quad a_{I_0 j} \Leftarrow u$$

否则直接转第三步

第三步　计算第 k 列元素：

$$\alpha \Leftarrow 1/a_{kk};\qquad a_{kk} \Leftarrow \alpha;\qquad a_{ik} \Leftarrow -\alpha\, a_{ik}\quad (i \neq k)$$

第四步　计算除第 k 列、k 行以外的其余元素：

$$a_{ij} \Leftarrow a_{ij} + a_{ik}a_{kj}\quad (i, j \neq k)$$

第五步　计算第 k 行其余元素：

$$a_{kj} \Leftarrow \alpha a_{kj}\quad (j \neq k)$$

II　作列交换：对于 k=n-1,n-2,…,1 做

若 $T_k \neq k$ 则对于 i=1,2,…n 做

$$v \Leftarrow a_{iT_k};\quad a_{iT_k} \Leftarrow a_{ik};\quad a_{ik} \Leftarrow v;$$

III　写出 $\boldsymbol{A}^{-1}$。

算法：选主元的高斯-约当消去法求逆矩阵

1. 算法入口。
2. 读入 A 的数据。
3. 对于 k=1，2，…，n-1 执行到 30。
4. 选主元 5-9。
5. $w \Leftarrow a_{kk}$，并 $s \Leftarrow k$，且 $t[k] \Leftarrow k$。
6. 对于 i=k+1，…，n 执行到 9。
7. 若 $ABS(a_{ik}) \le ABS(w)$ 则转向 9
8. $w \Leftarrow a_{ik}$，并 $s \Leftarrow i$
9. Continue。
10. 若 w=0 则转向 40。
11. 第 k、s 交换 12-16
12. 若 s=k 则转向 17
13. $t[k] \Leftarrow s$
14. 对于 j=1，2，…，n 执行到 16
15. $a_{kj} \Leftrightarrow a_{sj}$（两值交换）
16. Continue
17. 第 k 步消元 18-29
18. $\alpha \Leftarrow 1/a_{kk}$　并 $a_{kk} \Leftarrow \alpha$
19. 对于 i=1,2,…,k-1,k+1,…,n 执行到 21
20. $a_{ik} = -\alpha a_{ik}$
21. Continue
22. 对于 i=1,2,…,k-1,k+1,…,n 执行到 26
23. 对于 j=1,2,…,k-1,k+1,…,n 执行到 25

24. $a_{ij} \Leftarrow a_{ij} + a_{ik}a_{kj}$

25. Continue

26. Continue

27. 对于 j=1,2,…,k-1,k+1,…,n 执行到 29

28. $a_{kj} = \alpha a_{kj}$

29. Continue

30. Continue

31. 作列交换 32-38

32. 对于 k=n-1,n-2,…,1 执行到 38

33. 若 $t[k]$=k 转向 38

34. $s \Leftarrow t[k]$

35. 对于 i＝1，2，…，n 执行到 37。

36. $a_{is} \Leftrightarrow a_{ik}$（两值交换）

37. Continue。

38. Continue。

39. 输出解 $\boldsymbol{A}^{-1}$，并转向 41。

40. 打印“Singular”标志。

41. 算法出口。

42. 说明：

①本算法采用紧缩存储方案，整个计算过程均在 A 中进行，结果在 A 中

②数组元素 t[k]记录第 k 次消元的主元行号

例　设 $A = \begin{bmatrix} 1 & -1 & 0 \\ 2 & 2 & 3 \\ -1 & 2 & 1 \end{bmatrix}$，求 $\boldsymbol{A}^{-1}$

其基本计算过程如下：

$$\begin{bmatrix}1&-1&0\\ [2]&2&3\\ 1&2&1\end{bmatrix}\xrightarrow{1\Leftrightarrow 2}\begin{bmatrix}[2]&2&3\\ 1&-1&0\\ 1&2&1\end{bmatrix}\to\begin{bmatrix}1/2&2&3\\ -1/2&-1&0\\ 1/2&2&1\end{bmatrix}\to\begin{bmatrix}1/2&2&3\\ -1/2&-2&-3/2\\ 1/2&3&5/2\end{bmatrix}\to$$

$$\begin{bmatrix}1/2&1&3/2\\ -1/2&-2&-3/2\\ 1/2&3&5/2\end{bmatrix}\xrightarrow{2\Leftrightarrow 3}\begin{bmatrix}1/2&1&3/2\\ 1/2&[3]&5/2\\ -1/2&-2&-3/2\end{bmatrix}\to\begin{bmatrix}1/2&-1/3&3/2\\ 1/2&1/3&5/2\\ -1/2&2/3&-3/2\end{bmatrix}\to$$

$$\begin{bmatrix}1/3&-1/3&2/3\\ -1/2&1/3&5/2\\ -1/6&2/3&1/6\end{bmatrix}\to\begin{bmatrix}1/3&-1/3&2/3\\ 1/6&1/3&5/6\\ -1/6&2/3&1/6\end{bmatrix}\to\begin{bmatrix}1/3&-1/3&-4\\ 1/6&1/3&-5\\ -1/6&2/3&6\end{bmatrix}\to$$

$$\begin{bmatrix}1&-3&-4\\ 1&-3&-5\\ -1/6&2/3&6\end{bmatrix}\to\begin{bmatrix}1&-3&-4\\ 1&-3&-5\\ -1&4&6\end{bmatrix}\xrightarrow{2\leftrightarrow 3}\begin{bmatrix}1&-4&-3\\ 1&-5&-3\\ -1&6&4\end{bmatrix}\xrightarrow{1\leftrightarrow 2}\begin{bmatrix}-4&1&-3\\ -5&1&-3\\ 6&-1&4\end{bmatrix}=A^{-1}$$

作业：编制一个矩阵大小未知时矩阵求拟的函数，并在主程序里调用。

第五章 最小二乘曲线拟合

5.1 曲线拟合

实践中经常要寻求两个（或多个）变量间存在的关系，人们希望用确定变量间的一个方程式来表达这种数学关系。

第一步是收集资料，显示变量的对应值。例如 x 和 y 分别表示一个成年男子的身高和重量，那么 n 个样本单元应显示身高 $x_1, x_2,\ldots,x_n$ 和对应的重量 $y_1, y_2,\ldots, y_n$。

第二步就是在直角坐标系中画出点$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$。这个点集图有时称为散点图。

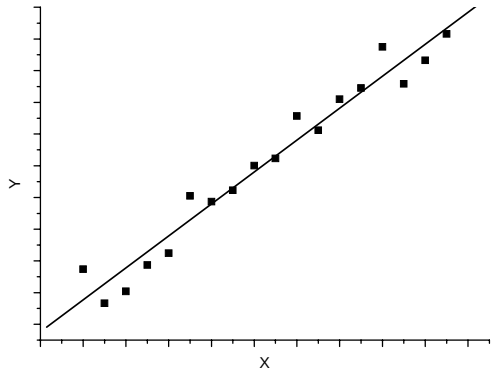


图 5.1

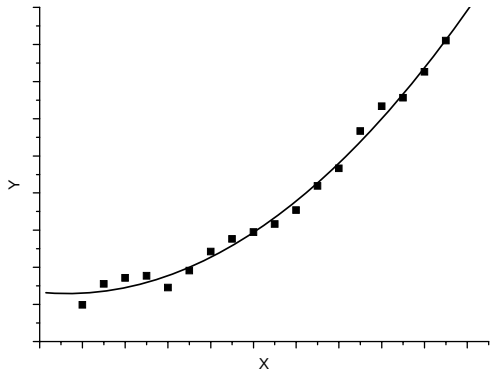


图 5.2

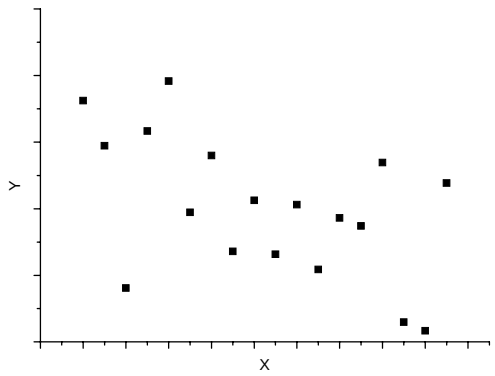


图 5.3

从散点图常可看出一条光滑的近似曲线。这样的曲线常称为近似曲线。例如在图 5.1 中，变量之间表现为较好地近似一条直线，我们说变量之间有线性关系。然而在图 5.2 中，变量之间虽然存在关系，但不是线性关系而是所谓的非线性关

系。在图 5.3 中，变量之间未表现出关系。

寻找拟合给定资料的近似曲线方程的问题，一般称为曲线拟合。在实践中散点图常能建议一个方程的形式。在图 5.1 中，我们可以使用直线

$$y = a + bx \tag{5.1}$$

而在图 5.2 中，我们一条抛物线或二次曲线

$$y = a + bx + cx^2 \tag{5.2}$$

有时候使用变换的变量作出散点图。例如，如果 log$y$ 对 x 导致一条直线，我们应试用 log$y = a + bx$ 作为近似曲线的方程。

5.2 回归

曲线拟合的主要目的之一是从一个变量（独立变量）估计另一个变量（相依变量）。估计的过程常涉及到回归。如果按照某个方程的意义从 x 估计到 y，我们称该方程为 y 关于 x 的回归方程，对应的曲线称为 y 关于 x 的回归曲线。

5.3 最小二乘

一般，有多条某一给定形式的曲线似乎都可以拟合资料集合。在构造直线、抛物线、或者其它近似曲线时，为了避免个人的评价，有必要一致同意“最佳拟合直线”、“最佳拟合抛物线”等等的定义。

为了导出一个可能的定义，如图 5.4，资料点是。对于一个给定值 x，比如 x_1，值 y_1 和由曲线 C 确定的对应值之间存在一个差。我们用 d_1 记这一个差，有时这个差称为偏差、误差等等，它可以是正的、负的或者零。类似地，对应 $x_2,\ldots x_n$ 有偏差 $d_2,\ldots,d_n$。

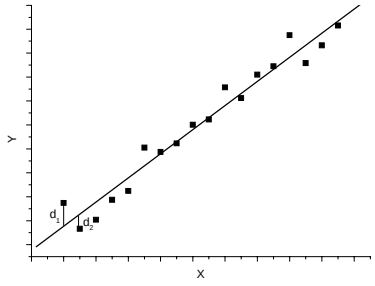


图 5.4

量$d_1^2+d_2^2+\cdots+d_n^2$提供了曲线 C 对资料集合的拟合优度的一个度量。如果该值比较小，则拟合是最好的，如果比较大，则拟合较差。因此作出下面的定义：

定义 若在近似 n 个资料点的集合时，对一给定的曲线族的全部曲线，其中一条曲线有性质：$d_1^2+d_2^2+\cdots+d_n^2$达到最小值，则称该曲线为给定曲线族中的最佳拟合曲线。

有这样一条性质的曲线称为最小二乘意义上对资料的拟合，该曲线称为最小二乘回归曲线，或简称最小二乘曲线。有这样一条性质的一条直线成为最小二乘直线，有这样一条性质的抛物线称为最小二乘抛物线，等等。

当 x 是独立变量而 y 是相依变量时，习惯上常采用上述定义。如果 x 是相依变量，则要修改定义，用水平偏差代替垂直偏差，这等于交换 x 和 y 轴。这两种定义方法会导致产生两条不同的最小二乘曲线。除非特别说明，通常将考虑 y 为相依变量而 x 是独立变量。

5.4 最小二乘曲线拟合

设有 m 组测量数据(x_i,y_i)(其中 i=1,2,…,m)，今用一个 n 次多项式来表示测量数据相应量 y_i 与自变量 x_i 之间的关系，即对测量数据进行拟合：

$$y_i=b_0+b_1x_i+b_2x_i^2+\cdots+b_nx_i^n+\varepsilon_i=\sum_{j=0}^{n}b_jx_i^j+\varepsilon_i$$

式中，b_j 为拟合系数，实际上是一个 m 个方程组成的方程组，要使其有解，必须 m≥n+1，写成矩阵形式就是：

$$\begin{bmatrix}y_1\\y_2\\\vdots\\y_n\end{bmatrix}=\begin{bmatrix}1&x_1&x_1^2&\cdots&x_1^n\\1&x_2&x_2^2&\cdots&x_2^n\\\vdots&&&&\\1&x_m&x_m^2&\cdots&x_m^n\end{bmatrix}\begin{bmatrix}b_0\\b_1\\\vdots\\b_n\end{bmatrix}+\begin{bmatrix}\varepsilon_1\\\varepsilon_2\\\vdots\\\varepsilon_m\end{bmatrix}$$

可写作：

$$y=Xb+\varepsilon$$

可得最小二乘解：

$$\hat{b}=(X^TX)^{-1}X^Ty$$

测量拟合值的计算为：

$$\hat{y} = X\hat{b} = X(X^T X)^{-1} X^T y$$

其具体的推导过程为：

1. $y = Xb$
2. $X^T y = X^T Xb$
3. $(X^T X)^{-1} X^T y = (X^T X)^{-1} X^T Xb$
4. $\hat{b} = (X^T X)^{-1} X^T y$

这样我们可得算法如下：

第一步：读入$(x_i),(y_i)$

第二步：计算 X 矩阵

第三步：计算 X^T

第四步：计算 $X^T X$，以及 $X^T X$ 的拟矩阵

第五步：计算$(X^T X)^{-1} X$

第六步：计算$\hat{b} = (X^T X)^{-1} X^T y$

作业：下载数据，编制最小二乘法程序，分别计算 m=3~7 时的 b 和测量值，并且分别计算$d_1^2 + d_2^2 + \cdots + d_n^2$以及原始 y 和计算得到的 y 的相关系数 R。

相关系数

两样本的相关系数定义为

$$r_{i,j} = \frac{\sum_{k=1}^{p}(x_{ik} - \bar{x}_i)(x_{jk} - \bar{x}_j)}{\sqrt{[\sum_{k=1}^{p}(x_{ik} - \bar{x}_i)^2][\sum_{k=1}^{p}(x_{jk} - \bar{x}_j)^2]}}$$

式中$\bar{x}_i$和$\bar{x}_j$分别为第i个和第j个样本的均值。$r_{i,j}$越接近于 1，相似性越大。

第六章 模式识别

6.1 模式识别

6.1.1 概述

模式识别方法最早是在 20 世纪 50 年代早期提出的，60-80 年代在各个学科得到广泛的应用。化学学科在此期间发表了数百篇文章。到 20 世纪 80 年代，模式识别方法发展成为一种非常成熟的多元分析方法。

模式识别的数学含义是指：假定样本集由 n 个观测对象即样本组成，每个样本用 p 个观测变量（在化学模式识别中观测变量常称为特征、指标或者参量等）进行描述，构成数据矩阵 $\mathbf{X}_{n\times p}$。这个数据阵中，包含有两方面的信息：一是样本性质的信息，一是变量或特征之间的信息。因而我们要处理的关系有 3 个：即变量与变量之间；样本与样本之间；样本与变量之间的关系。从几何角度看，样本集可以用在由观测变量或特征作坐标轴构成的 p 维变量空间或特征空间（也称模式空间）中的点集来表示，一个样本对应于一个点。模式识别理论认为，p 个特征包含了所讨论样本性质的全部信息，代表样本集的点集在特征或模式空间中的分布是“物以类聚”的，即不同性质的样本点在变量空间的不同区域（ 或称不同模式）中，相同性质的样本点聚集在一个区域中。所谓模式是指具有某种共同性质的一类现象的集合，比如一个聚集区域就可称为一个模式），模式识别的任务就是要找出这种分布（集合）的某些特征，进而根据这些特征预报另一些未知样本的性质，比如判别未知样本属于哪一个模式。

实际上，我们时时处处都在使用模式识别，如我们用眼睛可以区分道路、房屋、汽车，敲碗通过声音的高低来判别碗是否有裂缝，中医凭舌苔和脉搏进行诊断，化学家通过谱图来辨别化合物等。在低维空间如二维和三维空间，人类对模式的识别能力最强，但是在高维空间则必须借助数学的方法才能够对模式进行区分。模式识别属于多元分析方法，它是借助计算机来揭示隐含于事物内部规律的一种综合技术。

6.1.2 模式识别的主要步骤

模式识别的主要步骤包括数据预处理和模式识别方法分析两部分，其中预处理包括数据

预处理，特征提取、选择和压缩，相似系数和距离的计算等。模式识别方法主要包括有管理方法和无管理方法两种。

1、数据预处理

（1）数据预处理方法

模式识别中，所有分类和识别过程都基于样本数据矩阵 $\mathbf{X}_{n\times p}$。数据集 $\mathbf{X}_{n\times p}$ 直接影响计算结果，决定判别成败。数据集的质量是由于描述样本的各个特征或变量的类型差别引起的。如有的为谱图数据，有的为化合物的物理化学性质或结构方面的信息，数据类间不仅量纲不一样，其绝对值大小有时会有几个数量级的差别。通过数据的预处理可以改善数据的质量。如数据标准化对原始数据进行预处理。

数据标准化处理（data autoscaling）是将原样本数据矩阵中各元素减去该列元素的均值后再除去所在列元素的标准差的操作。变换公式为：

$$x_{ij}' = \frac{x_{ij} - \bar{x}_j}{S_j} \quad ,$$

式中

$$\bar{x}_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij} \quad , \quad S_j = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2} \quad ,$$

S_j，$\bar{x}_j$ 分别是数据矩阵 **X** 的第 j 列元素的标准偏差和平均值。经过标准化预处理的变量（一列元素）权重相同，均值为 0，方差或标准差都为 1。

数据预处理的方法还有中心化变换（data centralization）、对数变换(natural logarithm-transformation)和数据正规化变换（data normalization transformation）等。

（2）特征提取和选择

在模式识别中，特征提取是最关键的一步，用作特征的主要是化学量测数据，一般需根据化学量测实际和经验加以选择，如烟草近红外数据中量测的尼古丁、总氮、总糖、还原糖、氯、钾，烟气中的烟碱、水分、焦油和成品烟物理指标数据中的吸阻，单支重量，圆周等特征。

在模式识别中应用较广的特征选择方法有偏差权重法、Fisher 比率法、概率比率法、逐步判断法、学习机法和等方法。

2、模式识别方法

模式识别方法可分为有管理方法，无管理方法两类。其中，在有管理方法中需要有一训

练集。如对于两类的情况，在训练集中，有一些样本属于 A 类，另外一些样本属于 B 类，然后教给计算机，计算机经过训练之后则可识别未知样本. 有管理模式识别包括：线性判别，逐步判别分析，KNN 方法，SIMCA 方法，神经网络等；无管理模式识别方法则不需要已知类别关系的对象进行指导无管理方法包括：最小生成树，聚类分析，线性投影，非线性映射等，以下重点介绍一种无管理方法：主成分分析法。

6.2 主成分分析法

6.2.1 概述

主成分分析（principal component analysis，PCA）是一种最古老的多元统计分析技术。Pearcon 于 1901 年首次引入主成分分析的概念， Hotelling 在 20 世纪 30 年代对主成分分析进行了发展。如其他多元统计分析一样，在计算机出现之前，主成分分析的应用面很窄。当计算机出现后，主成分分析得以广泛地应用，特别是现在，在计算机的多元统计分析程序包中，几乎都含有主成分分析。

6.2.2 原理

主成分分析法是指在特征之间存在相关关系时，可以通过原始特征的线性组合，构成为数较少的不相关的新特征代替原始特征，而每个新特征都含有尽量多的原始特征的信息。新特征叫做原始特征的主成分。主成分分析在多元统计中主要是用作消除特征间相关性，是简化多元统计问题的一种有效的方法，也可以作为一种简单的模式识别方法。当取主成分为 2 时，即将原始多个特征线性组成得到两个新变量时，这相当于将原始高维空间中各模式点投影到二维平面上，然后利用二维平面上各模式点的分布进行分类和判别。

主成分分析法先对量测矩阵 $\boldsymbol{X}$ 直接进行分解，对 $\boldsymbol{X}$ 矩阵进行直接分解在数学上有几种算法。在化学计量学中采用的方法为非线性迭代偏最小二乘法（NIPALS）和奇异值分解方法（SVD），SVD 算法也可以用来求解大多数的线性最小二乘方问题。

6.2.3 SVD 分解

对一个样本的测量将产生一个向量 $x_i = (x_{i1}, x_{i2}, ..., x_{in})$，其中每一个元素代表一个

测量变量（在本文所处理的数据中，一个元素就代表谱图中的一个峰的峰面积）。对一系列的样本都进行测量，则将得到一个样本量测矩阵：

$$X=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1j} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2j} & \cdots & x_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{i1} & x_{i2} & \cdots & x_{ij} & \cdots & x_{in} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mi} & \cdots & x_{mn} \end{bmatrix} \tag{6.1}$$

在这个矩阵 $\boldsymbol{X}$ 中，每一行代表一个样本的测量矢量。对量测矩阵 $\boldsymbol{X}$ 进行 SVD 分解，即：$X=USV^t$，则得到三个矩阵 $\boldsymbol{U}$，$\boldsymbol{S}$，$\boldsymbol{V^t}$。其中，$\boldsymbol{S}$ 是对角矩阵，它收集了矩阵 X 的特征值。$\boldsymbol{U}$ 和 $\boldsymbol{V^t}$ 分别是标准列正交和标准行正交矩阵，收集了这些特征值所对应的列特征矢量和行特征矢量，在多元统计的主成分分析中，一般称为得分矩阵和载荷矩阵。如果体系的组分数是 k，则在矩阵 $\boldsymbol{S}$ 中，前 k 个特征值会显著大于其余的特征值，它们已包括了样本量测矩阵 $\boldsymbol{X}$ 的全部化学信息。一般只要取前 k 个特征值和特征矢量作为主成分，其余部分即可以看成误差并丢弃。

SVD 法是基于下面的线性代数定理：任意 $m\times n$ 的矩阵 $\boldsymbol{X}$，其行数 m 大于或等于列数 n，可以写成一个 $m\times n$ 的列正交距阵 $\boldsymbol{U}$，一个 $m\times n$ 的元素均为正数或零的对角矩阵 $\boldsymbol{S}$，以及一个 $n\times n$ 正交阵 $\boldsymbol{V}$ 的转置矩阵的乘积形式。不管矩阵是否是奇异的，这个分解几乎是唯一的。以下为对较简单的对称矩阵进行 SVD 分解的示例：

例：$A=\begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ $AA^T=A^TA=\begin{bmatrix} 6 & 10 & 6 \\ 10 & 17 & 10 \\ 6 & 10 & 6 \end{bmatrix}$ $\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix}=\begin{bmatrix} 28.86 \\ 0.14 \\ 0 \end{bmatrix}$

$u_1=v_1=\begin{bmatrix} 0.454 \\ 0.766 \\ 0.454 \end{bmatrix}$ $u_2=v_2=\begin{bmatrix} 0.542 \\ -0.643 \\ 0.542 \end{bmatrix}$ $u_3=v_3=\begin{bmatrix} -0.707 \\ 0 \\ -0.707 \end{bmatrix}$

$\Lambda=\begin{bmatrix} 5.37 & 0 & 0 \\ 0 & -0.372 & 0 \\ 0 & 0 & 0 \end{bmatrix}$忽略第2个奇异值

$$A\approx\begin{bmatrix} 5.37 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0.454 \\ 0.766 \\ 0.454 \end{bmatrix}\begin{bmatrix} 0.454 & 0.766 & 0.454 \end{bmatrix}=\begin{bmatrix} 1.11 & 1.87 & 1.11 \\ 1.87 & 3.15 & 1.87 \\ 1.11 & 1.87 & 1.11 \end{bmatrix}$$

6.2.4 主成分分析的数学与几何的意义

主成分分析的数学与几何的意义如图 6.1：

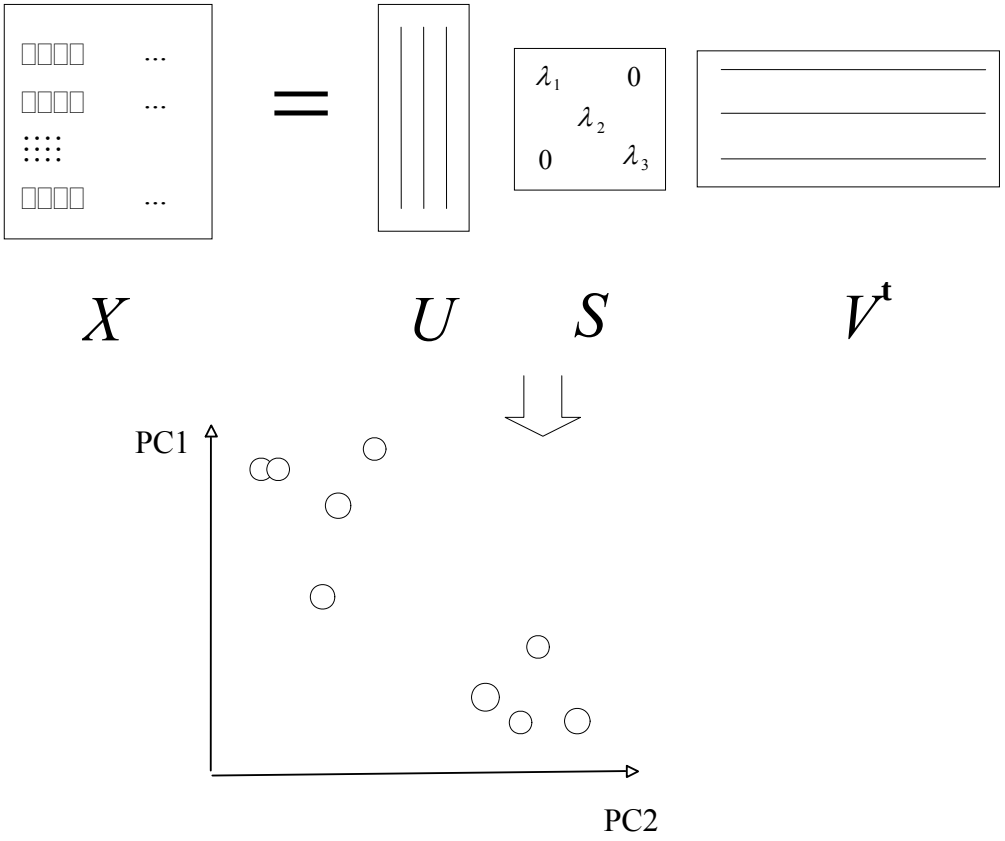


图 6.1 主成分分析的数学和几何意义

在上图中，左边是样本的量测矩阵 ***X***，右边依次为 SVD 分解得到的三个矩阵 ***U***，***S***，$\boldsymbol{V}^t$。如果取出得分矩阵 ***U*** 的第一列和第二列，分别作为 PC1 轴和 PC2 轴的坐标，在图上描出每一个样本所代表的点，则可以得到投影分析图。在投影图上，同一类样本的点会聚集在一起。样本点距离越远，则表示两个样本越不相似。根据这样的标准，就可以将一系列处在多维空间的样本测量数据在二维的平面图上展示出来，同时最大程度地保留了原有的样本信息，又提供了样本分类情况的直观表示。其数学解释如下：

设有 m 个样品，每个样品有两个观测变量 x_1，x_2 二维平面的散点图。n 个样本点，无论沿着 x_1 轴方向还是 x_2 轴方向，都有较大的离散性，其离散程度可以用 x_1 或 x_2 的方差表示。

当只考虑其中一个变量时，原始数据中的信息将会有较大的损失。若将坐标轴旋转一下：

$$f_1 = x_1\cos\theta + x_2\sin\theta \quad f_2 = x_2\cos\theta - x_1\sin\theta$$

即：

$$\begin{pmatrix} f_1 \\ f_2 \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = UX$$

且 ***U*** 是正交距阵，则 n 个样品在 f_1 轴的离散程度最大（方差最大），变量 f_1 代表了原始数据的绝大部分信息，即使不考虑 f_2，信息损失也不多。而且 f_1，f_2 不相关。只考虑 f_1 时，二维降为一维。

1、主成分的推导

定理 1. 若 ***A*** 是 n 阶实对称阵，则一定可以找到正交阵 ***U***，使 $U^{-1}AU = diag(\lambda_1, \lambda_2, ..., \lambda_n)$，其中 $\lambda_1, \lambda_2, ..., \lambda_n$ 是 ***A*** 的特征根。

定理 2. 若上述 ***A*** 的特征根所对应的单位特征向量为 $u_1, u_2, ..., u_n$，记 $U = (u_1, ..., u_n)$，则不同特征根对应的特征向量正交，即 $u_i^T u_j = 0$。

定理 3.设 Σ 的特征根为 $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n > 0$，则对应的标准正交基为 $u_1, u_2, ..., u_n$。

2、主成分的导出

设 $f = a_1x_1 + a_2x_2 + ... + a_nx_n = a^Tx$，找到系数使 $\mathrm{var}(f)$ (f 的方差) 最大，则 $\mathrm{var}(f) = \lambda$，$a = u$，所以，$f_1 = u_1^Tx$，$f_2 = u_2^Tx$，依此类推，且 $f_1, f_2, ...$ 不相关。称 f_1 为第一主成分，f_2 为第二主成分，……。而 $\lambda_i \Big/ \sum_{k=1}^{n} \lambda_k$ 为第 i 个主成份的贡献率 $\sum_{i=1}^{p} \lambda_i \Big/ \sum_{k=1}^{n} \lambda_k$ 为前 p 个主成分的累计贡献率。若 p 个主成分的累计贡献率超过 85%或更大，通常可认为前 p 个主成分基本包含了原来变量的信息。

3、样本主成分的导出

设有 m 个样品，n 个变量，得到的原始数据矩阵为公式 3.3.1 中所列矩阵，经过 SVD 分解之后，***U*** 矩阵中的每一列就代表一个主成分。

6.2.5 主成分分析的基本步骤

先对数据矩阵 ***X*** 进行奇异值分解(SVD)，这样得到三个矩阵 ***U***，***S***，***V***，即：$X = SVD(U, S, V)$ 。

取矩阵 ***U*** 的第一列和第二列，即：所含信息量最大的两个主成分，进行投影。第一列

对应 ***X*** 轴的坐标，第二列对应 ***Y*** 轴的坐标。每一行就是一个点，对应于数据矩阵 ***X*** 的同行所对应的样本。

在投影图上，正常情况下相同类型的样本点间的距离会比较接近。即：同类型样本点会在投影图上呈聚集的状态。

6.2.6 主成分分析的 Matlab 程序段

```
[U, S, V] = svd(Y);
lmd=diag(S);
n=size(lmd,1);
allsumlmd = 0;
for k = 1:n
    allsumlmd = allsumlmd + lmd(k)*lmd(k);
end
for k=1:n-1
    sumlmd=0;
    for j=(k+1):n
        sumlmd=sumlmd+lmd(j)*lmd(j);                    %{NOTE}
    end
    RSD(k,1)=sumlmd/(nw*(nt-k));
    RSD(k,1)=sqrt(RSD(k,1));                            %1st column of RSD is the real RSD value
    RSD(k,2)=1e6*RSD(k,1)/((nt-k)*(nt-k));              %2nd column of RSD is the IND value
    RSD(k,3)=lmd(k)^2*(nt-k)/sumlmd;                    %3rd column of RSD is the ratio of Fisher variance deviation
    RSD(k,4)=lmd(k)/lmd(k+1);                           %4th column of RSD is the ratio of consecutive eigenvalue
    RSD(k,5)=lmd(k)*lmd(k)/allsumlmd;
end
```

1. 将标准样本矩阵 Y 采用奇异值分解法（SVD）进行分解，得到得分矩阵 U、载荷矩阵 V 和对角矩阵 S，其中 S 收集了矩阵 Y 中的特征值。
2. 根据对角矩阵 S 中的特征值进行计算，分别得到剩余标准偏差（RE），因子指示函数（IND），Fish 方差比等表征矩阵 Y 中主成分信息的标准。

```
nc = 3;
U = U(:,1:nc);
S = S(1:nc,1:nc);
V = V(:,1:nc);
T = U * S;
P = V';
Y1 = T*P;
```

Err = Y-Y1;

3．根据各种 RSD 的计算方法综合确定的主成分数 nc，分别对 U、V 和 S 取前 nc 列，前 nc 列和前 nc 行、前 nc 列。

4．计算去除前 nc 个主成分数的样本信息矩阵 Y1 后的误差矩阵 Err。

6.2.7 主成分分析的应用示例

1、烟草化学成分近红外数据的主成分分析

（1）原始数据

随机选取了烟气化学成分数据、物理指标数据各 10 个共 20 个数据作为原始样本数据（表 6.1），以及烟叶化学成分数据各 10 个共 20 个数据作为原始样本数据（表 6.2）。其中 1—10 号样本为 H 牌卷烟，11—20 号为 Z 牌卷烟。

表 6.1 分类原始数据样本

尼古丁	总糖	还原糖	总氮	钾	氯
1.87	16.04	14.50	1.99	2.15	0.68
1.98	15.64	14.49	2.03	2.08	0.66
1.85	17.52	15.61	1.94	2.07	0.64
2.10	17.31	15.18	2.04	2.01	0.65
1.99	18.01	15.91	1.99	2.16	0.67
1.97	17.97	15.82	1.97	2.22	0.65
2.14	14.78	13.48	2.09	1.97	0.69
2.06	17.50	16.03	2.03	1.97	0.66
1.96	15.50	13.89	2.01	1.96	0.67
2.08	17.50	15.40	2.05	1.96	0.66
2.20	20.73	18.55	1.97	1.88	0.46
2.28	21.80	19.19	2.03	1.82	0.48
2.27	22.33	19.23	2.04	1.86	0.47
2.26	22.46	19.71	2.00	1.84	0.47
2.28	21.88	19.23	2.00	1.93	0.47
2.21	22.31	19.47	1.98	1.82	0.47
2.24	22.37	19.16	2.00	1.83	0.43
2.20	22.43	19.24	1.98	1.81	0.49
2.25	23.01	19.65	1.99	1.87	0.41
2.18	22.90	19.90	1.97	1.82	0.45

（2）数据预处理

采用数据标准化方法处理后的数据样本见表 6.2

表 6.2 预处理分类数据样本

尼古丁	总糖	还原糖	总氮	钾	氯
-1.7687	-1.1766	-1.1634	-0.4265	1.5554	1.1185
-0.9858	-1.3126	-1.1677	0.7109	1.0069	0.9297
-1.9111	-0.6732	-0.6819	-1.8482	0.9285	0.7410
-0.1317	-0.7447	-0.8684	0.9952	0.4584	0.8354
-0.9146	-0.5066	-0.5518	-0.4265	1.6337	1.0241
-1.0570	-0.5202	-0.5908	-0.9952	2.1039	0.8354
0.1530	-1.6051	-1.6058	2.4169	0.1450	1.2129
-0.4164	-0.6800	-0.4997	0.7109	0.1450	0.9297
-1.1281	-1.3602	-1.4280	0.1422	0.0666	1.0241
-0.2740	-0.6800	-0.7730	1.2795	0.0666	0.9297
0.5801	0.4185	0.5934	-0.9952	-0.5603	-0.9581
1.1495	0.7824	0.8710	0.7109	-1.0304	-0.7693
1.0783	0.9627	0.8884	0.9952	-0.7170	-0.8637
1.0071	1.0069	1.0966	-0.1422	-0.8737	-0.8637
1.1495	0.8096	0.8884	-0.1422	-0.1685	-0.8637
0.6513	0.9559	0.9925	-0.7109	-1.0304	-0.8637
0.8648	0.9763	0.8580	-0.1422	-0.9520	-1.2412
0.5801	0.9967	0.8927	-0.7109	-1.1088	-0.6749
0.9360	1.1939	1.0705	-0.4265	-0.6386	-1.4300
0.4377	1.1565	1.1790	-0.9952	-1.0304	-1.0525

（3）主成分数的确定

用主成分分析法处理后，由矩阵分解得到的所有主成分数据如下：

表 6.3 主成分数据

PC	λ	λ/sum(λ)	Se^2
1	4.4556	0.7426	0.7426
2	1.2587	0.2098	0.9524
3	0.2079	0.0346	0.9870
4	0.0464	0.0077	0.9948
5	0.0267	0.0045	0.9992
6	0.0048	0.0008	1.0000

其中，累计方差 Se^2 表示 d 个特征值与所有 p 个特征值加和的比值。

$$Se^2 = \frac{\sum_{i=1}^{d} \lambda_i}{\sum_{i=1}^{p} \lambda_i}$$

从表 6.3 中可知，使用两个主成分就可以解释 95.24%的数据方差，且第二个主成分的特征值大于 1，根据方差判据和特征值判据，可以确定烟草化学成分的数据有两个主成分。

（4）结果和讨论

根据主成分分析后得到的得分矩阵如图 6.2 所示：

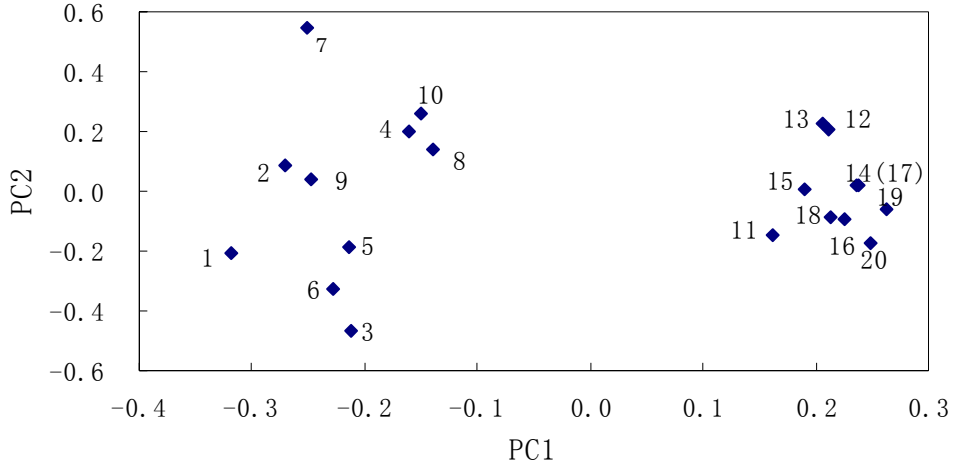


图 6.2　烟草化学成分数据的主成分得分图

由于已确定两个主成分，所以主成分得分图以二维图表示。由得分图可以看出，样本数据明显的聚为两类，且每个样本分类结果都正确，H 牌卷烟的 10 个烟草样本集中在图 1 的左侧，Z 牌卷烟的 10 个烟草样本集中在图 1 的右侧。H 牌卷烟的 10 个烟草样本明显比 Z 牌卷烟的 10 个烟草样本更离散，说明就这 20 个样本而言，Z 牌卷烟比 H 牌卷烟质量更稳定。

6.3 偏最小二乘回归

6.3.1 概述

偏最小二乘回归（Partial Least Square Regression, PLSR）是一种在工程技术与经济管理的分析预测、多元回归分析和建模中有着广泛应用的新的回归分析方法 。PLS 回归分析方法由 Wold 和 Alban 于 1966 年首先提出。最早起源于社会科学（1966 年，特别是经济学领域），但引起广泛重视则是在 1986 年以后，之后便在化学计量学，统计学等领域较多地使用起来。

PLSR 提供了一种多对多线性回归建模的方法．特别当变量的个数很多，且都存在多重相关性，而观测数据的数量又较少时．用偏最小二乘回归建立的模型具有传统的经典回归分析等方法所没有的优点。PLSR 方法中用的是替潜变量，其数学基础为主成分分析，因此也有人说，偏最小二乘回归 ≈ 多元线性回归分析＋典型相关分析＋主成分分析。

6.3.2 原理

偏最小二乘回归的目的是揭示变量空间里寻找某些线形组合，以能更好地解释反应变量的变异信息。该方法同时从解释变量与反应变量中提取两组主成分，它们分别是解释变量与反应变量的线性组合，满足以下两个条件：① 两组潜变量分别最大限度地承载解释变量或反应变量的变异信息；② 对应的解释潜变量与反应潜变量之间协方差最大化.

偏最小二乘回归与传统多元线性回归模型相比，偏最小二乘回归的特点是：（1）能够在自变量存在严重多重相关性的条件下进行回归建模；（2）允许在样本点个数少于变量个数的条件下进行回归建模；（3）偏最小二乘回归在最终模型中将包含原有的所有自变量；（4）偏最小二乘回归模型更易于辨识系统信息与噪声（甚至一些非随机性的噪声）；（5）在偏最小二乘回归模型中，每一个自变量的回归系数将更容易解释。

与其他建模方法相比，如神经网络等，PLS 具有简单稳定、计算量小等优点．模型参数估计时，无论是采用迭代法，还是 SVD 法，一般只需几步就可得到参数估计．预测未知样本时，计算量很小，精度也较高．然而，PLS 一般用于建立预测回归方程，对于未知参数分布特性的确定无能为力，它所给出解释变量与反应变量之间结构关系过于抽象，难以理解，只能作定性分析，无法确定它们之间准确的数量关系。

偏最小二乘与主成分分析很相似，其差别在于描述变量 Y 中因子的同时也用于描述变量 X。为了实现这一点，在数学上是以矩阵 Y 的列去参与矩阵 X 因子的计算，数学模型为：

$$X = TP' + E$$

以及

$$Y = UQ' + F$$

其中 T 和 U 的矩阵元——X 和 Y 的得分；

P 和 Q 的矩阵元——X 和 Y 的得分；

E 和 F——运用偏最小二乘模型法去拟合 X 和 Y 所引起的误差。

在理想的情况下，X 中误差的来源和 Y 中的误差来源完全相同，即影响 X 与 Y 的因素相同。但实际上， X 中的误差与 Y 中的误差并不相关， 因而 $t \neq u$ ，但当两个矩阵同时用于确定因子时，X 和 Y 具有如下关系

$$u = bt + e$$

式中 b 所表征的就是 u 和 t 的内在关系。

6.3.3 算法

1. 建模

首先对 X 和 Y 阵进行标准化处理，对于每一个主成分：

① 令 $u_{start} = y_j$

对于 X

② $w' = u'X / u'u$

③ $w'_{new} = w'_{old} / \|w'_{old}\|$

④ $t = Xw / w'w$

对于 Y

⑤ $q' = t'Y / t't$

⑥ $q'_{new} = q'_{old} / \|q'_{old}\|$

⑦ $u = Yq / q'q$

收敛测试：

⑧将步骤 4 中 t 与前一次迭代所得的 t 相比较，若二者相等（包括一定的舍入误差）

，到 9；否则到 2（若 Y 中仅有一个变量，则跳过步骤 5-8，并置 $q = 1$ ）

⑨ $p' = t'X / t't$

⑩ $p'_{new} = p'_{old} / \|p'_{old}\|$

⑪ $t'_{new} = t'_{old} / \|p_{old}\|$

⑫ $w'_{new} = w'_{old} / \|p'_{old}\|$ (p’,q’和 w’都用于预测；t 和 u 用于分类)

⑬ $b = u't / t't$

⑭ $E_h = E_{h-1} - t_h p'_h$

⑮ $F_h = F_{h-1} - b_h t_h q'_h$

之后，回到步骤 1，去进行下一个主成分的运算（注：当第一个主成分运算之后，X 在步骤 2、4 和 9 以及 Y 在步骤 5 和 7 分别由它们的残差 Eh 和 Fh 代替）。

2. 预报

数学模型建立起来之后目的用于未知样本预报。步骤为：

①如校正部分，将 X 及 Y 标准化（此时试样数为 n1 而不是 n）

②$h=0, Y=\hat{y}$（均值）

③$h=h+1$

$\hat{t}=Xw_h$

$y=y+b_h\hat{t}_h q'_h$

$x=x-\hat{t}p'_h$

④若 h 大于主成分数，到步骤 5；否则到步骤 3

⑤得到的 Y 为已经标准化的结果，因此按照标准化步骤地相反操作，将之恢复到原始坐标。

6.3.4 主成分数的判定

若 X 和 Y 间关系符合线形模型，则描述模型的主成分数应与模型的维数相等。主成分数是 PLS 模型的重要性质。由于存在量测误差，要精确确定未知体系中的主成分是有一定难度的。为解决这一问题，提出多种方法，一般分为两类：基于量测误差大小已知的方法或根据计算结果做出近似判断。

1. 量测误差大小已知的方法

这些方法的基点是假设量测误差已知，然后与算出的量测数据矩阵 X 的特征值，根据某一原则取某个主成分数所求出的误差进行比较，如算出的真实误差在已知量测误差范围内，即可据此确定主成分数。该法由 Malinowski 提出，见下式：

$$m(n-d)(RSD)^2=\sum_{i=1}^{m}\sum_{j=d+1}^{n}(\sigma_{ij})^2=\sum_{j=d+1}^{n}\lambda_j{}^0$$

经重新排列可得：

$$RE=RSD=\sum_{j=d+1}^{n}\lambda_j{}^0/[m(n-d)]$$

n 为样本数，m 为变量数

具体方法为，先把特征值最大所对应的特征向量看成主因子轴，其余的都作为次因子轴，按照上式计算 RSD，并将此计算值与已知的量测误差进行比较。依次类推，直至 RSD 与已

知量测误差近似相等，这时已计算过的特征向量的个数为体系中所含独立组分的个数。一般此种方法多用于仪器误差已知的波谱解析当中。

2. 量测误差大小未知的方法

主要根据一些经验函数估计主因子数。Malinowski 设计的因子指示函数法 IND 和 Fisher 方差比就是这种方法。

因子指示函数法（IND）：通过构造因子数的函数，随着主成分数的变化此函数值在某一点上达到最小，此时极小点对应的就是主成分数，定义如下：

$$RE = RSD/(n-k)^2$$

Fisher 方差比法：特征值代表量测矩阵的方差，所以它们的比值相当于 Fisher 方差比，因此 Fisher 检验的判别式为：

$$F(1,n-k) = \lambda_k (n-k) / \sqrt{\sum_{j=k+1}^{n} \lambda_j}$$

6.4 模式识别中的应用

6.4.1 概述

模式识别方法是 20 世纪 50 年代最早提出，60-80 年代在各个学科得到广泛应用，在 80 年代业已发展成为非常成熟的多元分析方法。多种算法被用于与模式识别领域，利用偏最小二乘进行聚类和分类的研究涵盖很多领域，如人脸识别、疾病诊断、波谱聚类等。

6.4.2 化整函数

在进行聚类分类研究当中，利用 PLS 直接的预报结果往往不是与类别标签值（如两类问题，一类目标值设为 0，另一类设为 1）直接对应，因为预报的值一般为非整型数值。通常用来化整地方法有两种，四舍五入法以及寻找最大距离法。前者原理非常简单，后者就是将预报值由小到大排序，然后计算相邻两个样本的预报值之差，差值最大的两个样本作为分界点，之前的样本为一类，之后的样本为另外一类。

6.4.3 应用实例

1. 概述

早期的癌症分类研究主要以肿瘤的形态学表征为基础，但这种分类方法存在着很大的局限性。因为具有相似的组织病理学表征的癌症，临床症状不同，对于治疗的反应也不同。只有少数情况下，具有相似形态学表征的癌症亚型可以依照不同的发病机理进行分类。1999年，麻省理工学院基因组研究中心的 Golub 等人在 Science 上发表了利用基因芯片技术将急性白血病分类并发现新的白血病亚类的文章，首次将借助基因表达图谱预测癌症类别的设想变为现实。

基因芯片就是利用点样机等机械装置，在玻璃等支持物表面整齐地点上高密度的、成千上万个“点”，每个“点”含有可与一种基因或表达序列标签（EST）杂交的一条或几条ＤＮＡ探针。由于芯片上有序地排列着 DNA 探针，因此也被称为微阵列(Microarray)。在芯片上滴加样品后，在合适的条件下，样品中含有的各种核酸片段就与相应的探针杂交。由于核酸片段上已标记有荧光素，激发后产生的荧光强度就与样品中所含有的相应核酸片段的量成正比，也就代表该基因的表达量。经激光共聚焦扫描仪等装置扫描后，所获得的信息经专用软件分析处理，即可获得成千上万种基因的表达情况。正因为这种特点，基因芯片技术已成为当前肿瘤研究的热点。

2. 数据描述

通过运用基因芯片，获得 72 组白血病病人样本的 7129 个基因的表达水平值，借助 PLS 建模预报，实现了从分子水平对两种不同的白血病：急性淋巴白血病（acute lymphoblastic leukemia，ALL）和急性骨髓白血病（acute myeloid leukemia，AML）。训练集含 38 个样本，包括 27 个 ALL 样本，11 个 AML 样本，测试集含 34 个样本，20 个 ALL 样本，14 个 AML 样本。该数据集可从如下链接下载到：

http://www.broad.mit.edu/cgi-bin/cancer/publications/pub_paper.cgi?mode=view&paper_id=43

处理流程

①主成分数确定：因为基因芯片数据的量测误差未知，因此采用因子指示法进行主成分数的确定。

②数据预处理：采用值域调整法进行数据预处理，即对于每一个点，减去所在变量的最小值，再除以该变量中最大值和最小值的差，具体见下式：

$$x_{ij} = (x_{ij} - x_{j\min})/(x_{j\max} - x_{j\min})$$

③PLS 建模：利用 38 个训练样本进行 PLS 建模，计算出回归系数 t，b，q，p。

④PLS 预报：首先利用已计算的回归系数，对 34 个测试集样本预报出实数型的 Y 值，再利用第二种化整函数，将其校正为整数型的类别值（如用 0 和 1 分别表示不同的两类）。

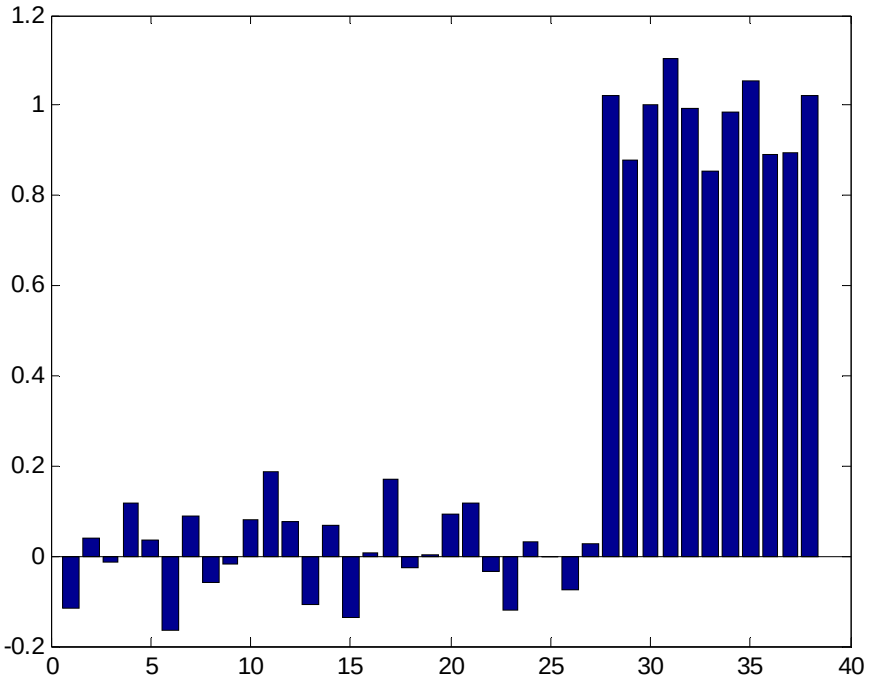
3. 结果讨论

首先对数据进行 IND 因子指示法分析，计算出前 10 个主成分的 IND 值，

	Pc1	Pc2	Pc3	Pc4	Pc5	Pc6	Pc7	Pc8	Pc9	Pc10
IND	8.2193	8.0812	7.9297	**7.7758**	7.8223	7.9289	8.1161	8.3958	8.7234	9.0967

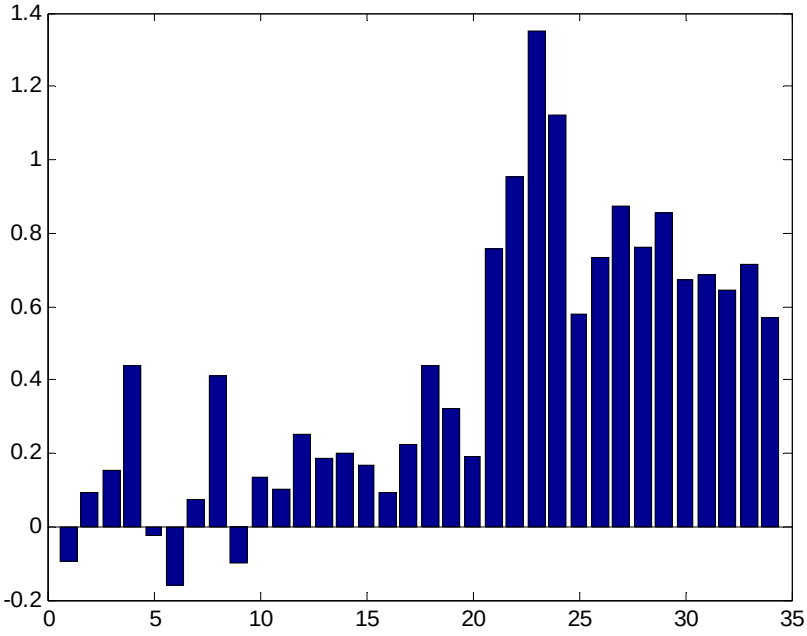
可以看到在主成分数为 4 的时候 IND 的值为最小，因此，进行 PLS 建模预报时主成分数选取 4。

由此对 38 个训练样本建模，自预报正确率达 100%，预报的实数值如下：



从图上可以清楚地看出，前 27 个 ALL 样本预报值在-0.2 和+0.2 之间，后 11 个 AML 的样本预报值都大于 0.8。可见 PLS 对训练集具有良好的分类效果。

在此基础上用建模时计算出的 t，b，q，p 对 34 个样本预报，正确率仍为 100%，见下图：



从图上可以清楚地看出，前 20 个 ALL 样本预报值在-0.2 和+0.45 之间，后 14 个 AML 的样本预报值都大于 0.5。因此由 PLS 建立的模型具有良好的分类预报能力。

再进行化整函数的处理，就得到预期的整数预报值，即对于训练样本，前 27 个样本预报值为 0，后 11 个样本预报值为 1；对于测试样本，前 20 个样本预报值为 0，后 14 个样本预报值为 1。

第七章 遗传算法

7.1 引言

大自然是人类获得灵感的源泉。几百年来，将生物界所提供的答案应用于实际问题求解已被证明是一个成功的方法，并且已形成仿生学一个专门的学科。遗传算法（Genetic Algorithm, GA）正是基于这种思想而发展起来的一种通用的问题求解方法。我们知道，自然界所提供的答案是经过漫长的自适应过程（成为进化的过程）而获得的结果。我们也可以利用这一过程本身来解决一些复杂的问题。遗传算法是一类借鉴生物界的进化规律（适者生存，优胜劣汰的进化机制）演化而来的随机化搜索方法。它是由美国的 J. Holland 教授 1975 年首先提出，随后，De Jong 和 T. E. Davis 等人对其进行了完善和发展。遗传算法主要特点是直接对结构对象进行操作，不需要求导和函数连续性的限定；具有内在的隐并行性和更好的全局寻优能力；采用概率化的寻优方法，能自动获取和指导优化的搜索空间，自适应地调整搜索方向，不需要确定的规则。遗传算法的这些性质，已被人们广泛地应用于组合优化、机器学习、信号处理、自适应控制和人工生命等领域。它是现代有关智能计算中的关键技术之一。当前，人们用进化计算（Evolutionary Computation）来包括遗传算法，进化规划，进化策略和遗传编程，并认为它是人工智能的未来。鉴于遗传算法的论文和书籍已经很多，有兴趣的读者可以参考其它书籍，这里我们简单介绍它的原理。

7.2 优化算法

从本质上讲，遗传算法是一种自适应的全局的概率搜索算法，是一种优化的方法。我们知道，传统的优化方法主要有三种：枚举法、启发式算法和搜索算法：

(1)枚举法　枚举出可行解集合内的所有可行解，以求出精确最优解。对于连续函数，该方法要求先对其进行离散化处理，这样就可能因离散处理而永远达不到最优解。此外，当枚举空间比较大时，该方法的求解效率比较低，有时甚至在目前最先进计算工具上都无法求解。

(2)启发式算法　寻求一种能产生可行解的启发式规则，比如梯度变化的方向等，以找到一个最优解或近似最优解。该方法的求解效率比较高，但对每一个需求解的问题必须找出

其特有的启发式规则，这个启发式规则要求对问题本身有比较深刻的了解，一般无通用性，不适合于其他问题。

(3)搜索算法　寻求一种搜索算法，该算法在可行解集合的一个子集内进行搜索操作，以找到问题的最优解或者近似最优解。该方法虽然保证不了一定能够得到问题的最优解，但若适当地利用一些启发知识，就可在近似解的质量和效率上达到一种较好的平衡。

随着问题种类的不同以及问题规模的扩大，人们在寻求一种能以有限的代价来解决搜索和优化的通用方法，遗传算法正是为我们提供的一个有效的途径，它不同于传统的搜索和优化方法。主要区别在于：

①自组织、自适应和自学习性(智能性)。应用遗传算法求解问题时，在编码方案、适应度函数及遗传算子确定后，算法将利用进化过程中获得的信息自行组织搜索。由于基于自然的选择策略为“适者生存，不适应者被淘汰”，因而适应度大的个体具有较高的生存概率。通常，适应度大的个体具有更适应环境的基因结构，再通过基因重组和基因突变等遗传操作，就可能产生更适应环境的后代。进化算法的这种自组织、自适应特征，使它同时具有能根据环境变化来自动发现环境的特性和规律的能力。自然选择消除了算法设计过程中的一个最大障碍，即需要事先描述问题的全部特点，并要说明针对问题的不同特点算法应采取的措施。因此，利用遗传算法的方法，我们可以解决那些复杂的非结构化问题。

②遗传算法的本质并行性。遗传算法按并行方式搜索一个种群数目的点，而不是单点。它的并行性表现在两个方面，一是遗传算法是内在并行的(inherent parallelism)，即遗传算法本身非常适合大规模并行。最简单的并行方式是让几百甚至数千台计算机各自进行独立种群的演化计算，运行过程中甚至不进行任何通信(独立的种群之间若有少量的通信一般会带来更好的结果)，等到运算结束时才通信比较，选取最佳个体。这种并行处理方式对并行系统结构没有什么限制和要求，可以说，遗传算法适合在目前所有的并行机或分布式系统上进行并行处理，而且对并行效率没有太大影响。二是遗传算法的内含并行性(implicit parallelism)。由于遗传算法采用种群的方式组织搜索，因而可同时搜索解空间内的多个区域，并相互交流信息。使用这种搜索方式，虽然每次只执行与种群规模 n 成比例的计算，但实质上已进行了大约 $O(n^3)$次有效搜索，这就使遗传算法能以较少的计算获得较大的收益。

③遗传算法不需要求导或其他辅助知识，而只需要影响搜索方向的目标函数和相应的适应度函数。

④遗传算法强调概率转换规则，而不是确定的转换规则。

⑤遗传算法可以更加直接地应用。

⑥遗传算法对给定问题，可以产生许多的潜在解，最终选择可以由使用者确定(在某些特殊情况下，如多目标优化问题不止一个解存在，有一组最优解。这种遗传算法对于确认可替代解集而言是特别合适的)。

7.3 简单遗传算法（SGA）

7.3.1 简介

把计算机科学与进化论结合起来的尝试开始于 20 世纪 50 年代末，但由于缺乏一种通用的编码方案，使得人们只能依赖变异而不是交配来产生新的基因结构，故而收效甚微。到 20 世纪 60 年代中期， Holland 等人的主要贡献是提出了位编码技术，这种编码即适合于变异又适合交配(即杂交)操作，并且他强调将交配作为主要的遗传操作。我们习惯上把 Holland 1975 年提出的遗传算法（GA）称为传统的 GA。它的主要内容可包括：基因编码、产生祖先(群体)、基因评价、基因选择和遗传操作。

编码（CODING）

编码起着遗传信息与优化控制变量之间的纽带作用，编码的优劣有时直接关系到能否收敛到最优解。GA 在进行搜索之前先将解空间的解数据表示成遗传空间的基因型串结构数据，这些串结构数据的不同组合便构成了不同的解。基因型串结构是指由 0 和 1 组成的字串，从理论上讲，这种字串能表达任意的数字和文字解。在实际应用中，还有用实数直接编码的方式，它容易理解，也容易编程，详见后文。

群体（POPULATION）

随机产生 N 个初始串结构数据，每个串结构数据称为一个个体，N 个个体构成了一个群体。这是 GA 与其它算法最大的不同。GA 以这 N 个串结构数据作为初始点开始迭代。母体数 N 是遗传算法的一个重要参数，称为群规模，它的大小由用户指定，N 不能太小，太小搜索范围小，不容易找到全局最优，但也不宜过大，过大计算量增大，效率急剧降低。一般从十开始，以十为单位递增。

评价（EVALUATION）

评价的标准因所研究的问题不同而异，在遗传算法中评价值被称为适应函数值（FITNESS)。评价是 GA 中最重要的环节，是个体适应环境的度量，是竞争对手生死存亡的标准，也是控制进化过程的关键。从另一方面来讲，如果没有正确的评价函数，GA 就无

法运行。

选择（SELECTION）

选择的目的是为了从当前群体中选出优良的个体，使它们有机会作为父代为下一代繁殖子孙。遗传算法通过选择过程体现这一思想，进行选择的原则是适应性强的个体为下一代贡献一个或多个后代的概率大。选择实现了达尔文的适者生存原则。选择可以有许多方式，比如俄国轮盘赌等，也可以简单地选择所有个体进入下一轮进化。

遗传操作（GENETIC OPERATOR）

GA 能够进化，变化数据结构的动力是遗传操作。通常，遗传操作包括杂交、变异和复制。这种模仿生物系统的遗传操作一方面保存和继承原先个体中优秀的基因，另一方面改变某些不好的基因，使得群体不断更新，逐渐趋近优化解。

杂交（CROSSOVER）

杂交是遗传操作的主要操作，它担负着优秀基因遗传的功能。进行杂交操作时，从群体中随机选择两个个体作为父本或称双亲，对它们的片断进行从新组合，产生两个后代（offsprings）。组合的方式有三种：

（1） 一点杂交：随机选取一个截断点，将双亲码串自此断开后交换其尾部。例：

	双亲	后代
A	1001|0110	10010001
B	1101|0001	11010110

（2） 多点杂交：随机选取多个截断点，交换有关部分。例：

	双亲	后代
A	10|01|01|10	10010101
B	11|01|00|01	11010010

在实际应用中通常采用两点杂交，即交换双亲中的一个基因片断。

（3） 模板杂交：随机选取一个模板，按照模板内容交换有关部分。

例：

	双亲	后代
A	10010110	11010100
B	11010001	10010011

模板：01101010（其中 0 为选 A 基因，1 为选 B 基因）

变异（MUTATION）

变异操作是模拟生物在自然环境中由于各种偶然因素引起的基因突变过程，表现为码串字符的翻转，如在二进制码串中，0 变为 1 或 1 变为 0。变异也是 GA 中重要的遗传操作，

它可以产生群体中从没有出现过的基因和数据结构。变异操作可使适应值差的个体或群体性能趋于一致时的个体发生变化，同时防止适应值好的个体变异，从而使每一代保持新鲜个体，避免进化停滞，过早收敛。变异通常在群体中随机选择一个父本，它也有三种方式：（1）一点变异；（2）多点变异；（3）模板变异。

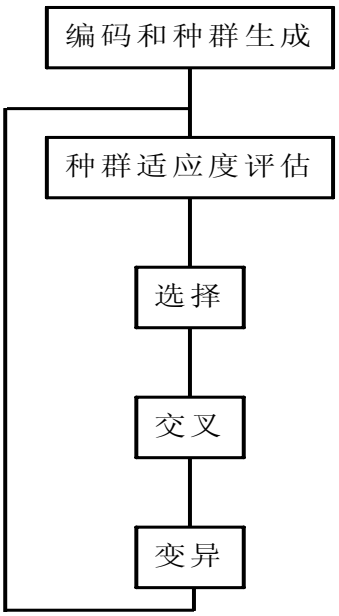
比如：

父本	后代
10010110	10010100
	↑ 由 1 变为 0

要正确有效地运行 GA 还需要一些控制参数，比如群体大小、交叉概率（Pc）、变异概率（Pm）和终止条件等等，这些参数的选取一般可按照两个原则：一是不太严格，既在一定范围内对计算结果和运营时间没有太大影响；另一个是参数在处理不同问题时没有统一的最佳值，需要调试。比如交叉概率和变异概率的取值，一般 Pc 比较大（0.8），Pm 比较小（0.2），当然也可以相反，应从实际问题出发灵活掌握。

最后我们讨论一下终止条件。常用的终止条件有根据适应值不再变动一段时间后终止或进行指定遗传代数后终止等，或者两种条件一起应用。由于使用遗传算法时往往是先不知道正确的解，终止条件给的宽松一点为好，当然这意味着可能多花费了计算资源。

GA 基本流程如图 1.1 所示。



7.3.2 遗传算法的优缺点

遗传算法是一种随机算法，它不需要关于体系的先验知识，对于求解的问题本身可以一

无所知，只需要一个适应值来评价染色体。它的优越性表现在利用简单的编码技术和繁殖机制来表现复杂的现象，从而解决非常困难的问题。特别是它不受搜寻空间的限制，不受各种假设的约束。遗传算法通过在解空间内不同区域多个点的搜寻，在搜寻过程中不易陷入局部最优，并且由于它的并行性，它在最优化并行处理方面得到了广泛的应用。

但一切事物都具有两面性，尽管遗传算法有许多方面值得褒扬，但也有其不足，遗传算法的缺陷之一就是“爬山”能力弱，当搜索到局部最优时，有时很难跳向解空间的其他区域，从而陷入到局部最优之中，使整个搜索停滞不前。造成这种停滞最主要的原因之一就是遗传算法的变异概率太小、群体太小以及群体失去多样性等原因，以至于不能驱使搜索转移。经常找到的是组合问题的次优解。而且，传统的遗传算法在计算时间上也比较慢，在大规模库搜索优化时就需要改进。

为了提高遗传算法的搜寻能力，很多研究者提出了新的方法，使遗传算法在近年来得到了很大的发展。如 Syswerda 提出了一致杂交因子，它是随机交换基因序列位置来保证群体的多样性。Eshelman 提出了阻止类似个体之间的杂交来防止过早的收敛。Whitley 提出了编码方法通过对远离一些先前的局部节的距离进行编码来找到更精确的解。

值得注意的是在运用 GA 时，初学者往往会犯两类错误：一类是将 GA 用于简单的优化问题，比如在用于选择变量时，变量的个数不多，一共只有 8-10 个，用穷举的方法很快就能得到结果，这时用 GA，明显是不划算的。再比如对于普通线性体系的求解，前人已经研究了许多成熟的解析算法，在大部分的情况下，这些算法又快又准，完全没有不必要用 GA，GA 也无法得到更好的结果。另一类错误是将 GA 的功能理想化，认为 GA 能解决所有大规模优化问题，这也是不现实的。比如优化空间扩大到实数域时，同时能找到 50 个以上的实数解是相当困难的，只要问题复杂一些，往往多次运行也无法找到全局最优解。

根据我们自己的经验，GA 比较适宜的应用领域是：非线性问题，需要迭代即初值问题，以及我们对问题的解题过程了解很少但却知道如何评价的问题。在生物和化学计算中，典型的非线性问题有平衡常数问题，它常常涉及非线性方程和非线性方程组，传统算法是用迭代法，迭代法需要好的初值，初值不对导致收敛到其它解或者发散，特别是在多变量情况下，问题更加严重。这时采用 GA 就能很好地解决问题。另外一个应用 GA 的领域是大规模优化，比如分子设计中的分子对接等。GA 有时可以成为我们手边的有用工具，遇到一些一时无法想清楚的问题，可以用 GA 试算一下，往往能得到较好的结果。

7.4 数值遗传算法(Numeric Genetic Algorithm, NGA)

Holland 提出的遗传算法是采用二进制编码来表现个体的遗传基因型的，它使用的编码符合集由二进制 0 和 1 组成的，因此实际的遗传基因型是一个二进制符合串，其优点在于编码、解码操作简单，交叉、变异等遗传操作便于实现，而且便于利用模式定理进行理论分析等；其缺点在于，不便于反应所求问题的特点知识，对于一些连续函数的优化问题等，也由于遗传算法的随机特性而使得其局部搜索能力较差，对于一些多维、高精度要求的连续函数优化，二进制编码存在着连续函数离散化的映射误差，个体编码较短时，可能达不到精度要求；而个体编码较长时，虽然能提高精度，但却会使算法的搜索空间急剧扩大。显然，如果个体编码串特别长时，会造成遗传算法的性能降低。后来许多学者对遗传算法的编码方法进行了许多改进，例如，为提高遗传算法的局部搜索能力，提出了格雷码(Grey Code)编码。下面以一个实际的例子来说明数值遗传算法的设计与实现。我们给出了基本思路和大部分 C 语言的程序，希望对读者有所帮助。

1、问题的表示

对于一个实际的待优化问题，首先要将其表示为适合于遗传算法操作的形式。它包括以下几个步骤：

(1) 根据具体问题确定待确定的参数。在数值遗传算法中，我们将实数指标达成染色体，既将实数作为操作的直接对象，不再进行转换。对于不同的问题，这种表达要相应的变化。

(2) 确定评价函数。这是一个寻优问题是否能适合于遗传算法的关键，能有一个确定的评价函数，就可以利用遗传算法来进行寻优。值得指出的事，对于不同的问题，遗传算法只要改变表达和评价函数，其余的程序都不要变动就可以正确的运行。

(3) 对于控制参数确定它们的相应值，使遗传算法能够正常运行。

举一个实际问题为例：

这是一个非线性参数估计问题，色谱保留时间与淋洗条件的关系。y 是保留时间，x_1 是淋洗液中甲醇的百分浓度，x_2 是淋洗液的 pH 值。根据实验值，建立非线性模型如下：

$$y = ax_1^2 + bx_2^2 + cx_1x_2 + dx_1 + ex_2 + f$$

实验值如下表所示：

y	x_1	x_2
6.08	10.0	5.0
2.42	20.0	5.0
2.10	30.0	5.0
7.31	10.0	5.5
3.00	20.0	5.5
3.13	30.0	5.5
7.06	10.0	6.0
3.72	20.0	6.0
3.37	30.0	6.0

一共 9 组实验数据，待确定的参数为 a-f 的 6 个数值，为实数值，评价函数为$\Sigma(y_i-y_i')^2$，其中 y'为根据系数计算方程右式所得结果。

2、数据结构与数值遗传算法参数

数值遗传算法处理的对象主要是个体，因此设计了结构变量 Individual 来描述个体信息，其中包括染色体 chrom，个体评价值 evalu。这样的结构个体实际上是一个解，它的染色体就是要求解的变量，这种用实数直接编码的方法比较容易理解，也比较容易编程，当然它只能用于解实数解。以下程序出了定义数据结构外，还定义一些常数变量，数组变量和全局变量，他们都是以后程序需要使用的。

```
#define POPULATION  50     /*种群大小*/
#define CHROMSOME 6     /*染色体长度*/
#define PCross              10    /*交叉概率*/
#define PMutation         2     /*变异概率*/
#define EPSILON          1e-4 /*终止条件*/
#define COUNT             50000
#define SCALE             1.0   /*   调节后期突变的范围  */

typedef   struct
{
     double    chrom [CHROMSOME];
     double    evalu;
} Individual;                        /*个体*/

Individual      popul [POPULATION]; /*种群*/
double          scale;                  /*  调节后期突变的范围  */
int             best, worst;                /*最优个体位置，最差个体位置*/
double          y[9], x1[9], x2[9], var[9][CHROMSOME];     /*方程式的数值*/
time_t          t1, t2;                 /*时间定义*/

***** var
```

3、初始种群的产生

产生初始种群的方法通常有两种。一种是完全随机的方法产生的，它适合于对问题的解无任何先验知识的情况。另外一种是根据某些先验知识，在这些限制条件下再随机的选取样本，这样能使遗传算法更快的到达最优解。

在这里我们产生初始种群的变量是随机选取了-1 到+1 之间的数值。

```
void InitNGA ()
{
    int         i, j;
    time_t      t;

    srand((unsigned)time(&t));   /* 有关时间，可不用 */
    scale = 1.0;
    for (i=0; i<POPULATION; i++)
    {
        for (j=0; j<CHROMSOME; j++)
        {
            popul[i].chrom[j] = 2*(double)(rand())/RAND_MAX - 1.0;   /* 在-1 到 1 之间 */
        }
        Evalu (&popul[i]); /* 评价函数，定义见后 */
    }
    Sort ();                    /* 排队，定义见后 */
}
```

4、杂交操作(Crossover)

在数值遗传算法中，杂交操作是在种群中随机选取两个个体，将两个个体中随机选取的一个变量相加，取平均数，产生一个新的个体，然后评价这个新的个体是否是更优的适应值，如果是更优的个体就用新的个体取代当前种群中最差的个体。另一种方式是产生两个后代，既不是用平均数，而是分别给两个后代不同的权重，读者可以很容易编程。数值遗传算法的这种改进与传统遗传算法有相同和不同之处。相同的是这种杂交操作同样能保留父本中有用的信息，既后代的变化依据父本的树值；不同的是经过这种杂交操作后人们无法确切的指出哪一段信息是哪一个父本的。当群体中个体退化时，既所有的个体都差不多相同时，这种操作失效，这是一个缺点。

```
void Crossover ()
{
    int i, j, k;
    Individual offspring;

    do
```

```
    {
        i = random (POPULATION);
        j = random (POPULATION);
    }while (i == j);

    for (k=0; k<CHROMSOME; k++)
    {
        offspring.chrom[k]=popul[i].chrom[k];
    }
    k = random (CHROMSOME);
    offspring.chrom[k] += popul[j].chrom[k];
    offspring.chrom[k] /= 2;
    Evalu (&offspring);

    if (best != i) worst = i;        *****
    else worst = j;
    if (offspring.evalu < popul[worst].evalu)
    {
        for (k=0; k<CHROMSOME; k++)
        {
            popul[worst].chrom[k]=offspring.chrom[k];
        }
        popul[worst].evalu = offspring.evalu;
    }
    Sort();
}
```

5、突变操作(Mutation)

突变操作从群体中随机选取一个个体，将这个个体中的一个随机选取的变量加上一个突变随机数，这个突变随机数的大小是有 Scale 来控制的，产生一个新的个体，然后评价这个新的个体是否是更优的个体，如果是更优的个体就用新的个体取代上一代的个体。设置 scale 是控制突变随机数大小的，scale 为 1 时，突变随机数变化从 0 到数值的两倍，因此给数据变化足够大的范围，当程序长时间无法改变最优解时，Scale 变小，这样突变的范围减小，突变成功的可能性增大，这在需要较高精度实数解时，十分有用，同时，scale 也是控制终止的关键参数，当 scale 很小，比如 10^{-5} 时，程序又长时间没有更好的解出现，于是跳出循环，结束运行。

```
void Mutation ()
{
    int i, j, k;
    Individual offspring;
```

```
    do
    {
        i = random (POPULATION);
        j = random (POPULATION);
    }while (i == j);

    for (k=0; k<CHROMSOME; k++)
    {
        offspring.chrom[k]=popul[i].chrom[k];
    }
    k = random (CHROMSOME);
    offspring.chrom[k] *= 1+scale*(2*(double)(rand())/RAND_MAX-1);
    Evalu (&offspring);

    if (best != i) worst = i;    *****
    else worst = j;
    if (offspring.evalu < popul[worst].evalu)
    {
        for (k=0; k<CHROMSOME; k++)
        {
            popul[worst].chrom[k]=offspring.chrom[k];
        }
        popul[worst].evalu = offspring.evalu;
    }
    Sort();
}
```

6、评价(Evaluation)

数值遗传算法中，评价函数同样是非常重要的，改变评价函数可以适应不同的问题。在评价函数的程序中，我们用了指针变量，程序中也有一些指针运算，请读者注意，这是我们用的唯一一个有参数的函数，具体函数如下所示。

```
void Evalu (Individual *who)
{
        double temp, temp1=0;
          who->evalu = 0.0;
          for(int i=0;i<9;i++)
          {
                    temp=0;
                    for(int j=0;j<CHROMSOME;j++)
                    {
                              temp+=who->chrom[j]*var[i][j]; /*var 数组在主程序中说明*/
                    }
                    temp1+=(y[i]-temp)*(y[i]-temp);
```

```
        }
        who->evalu+=temp1;
}
```

7、排序(Sort)

排序不是数值遗传算法的一个操作，而是在初始化、CrossOver 或者 Mutation 后找出群体中的最优值和最差值。我们用的方法是只记录最好和最差数组下表，而不是真的按顺序排序。

```
void Sort ()
{
    int i;

    best = worst = 0;
    for (i=1; i<POPULATION; i++)
    {
        if (popul[i].evalu < popul[best].evalu)
            best = i;
        if (popul[i].evalu > popul[worst].evalu)
            worst = i;
    }
}
```

8、主程序

主程序显示了整个运行过程，还有一些数据文件、打印、控制等内容，请读者仔细阅读，帮助理解。

```
void main()
{
    FILE *fp, *out;
    int i, TotalCount = 0, count = 0;
    double    prev;
    double sumy, sumx1, sumx2;

    if ((fp = fopen("GA-data.txt", "r"))==NULL)                    /*打开数据文件*/
    {
        fprintf(stderr, "Cann't open data file.\n");
        getchar();
        exit(0);
    }

    if ((fp = fopen("result.txt", "w"))==NULL)                     /*保存数据文件*/
    {
        fprintf(stderr, "Cann't write result file.\n");
        getchar();
```

```
        exit(0);
}

t1 = time(NULL);                                                    /*计时*/
for (int i = 0; i<9; i++)                                           /*读取数据并初始化*/
{
        fscanf(fp, "%lf%lf%lf", &y[i], &x1[i], &x2[i]);         /*数据文件的格式见本节开始*/
        sumy += y[i];
        sumx1 += x1[i];
        sumx2 += x2[i];
}

sumy /= 9.0;
sumx1 /= 9.0;
sumx2 /= 9.0;                           /*  对 y 和 x1,x2 计算平均值  */

for(int i=0;i<9;i++)
{
        y[i] -= sumy;
        x1[i] -= sumx1;
        x2[i] -= sumx2;             /*这是数据中心化，数据中心化后，计算速度加快。*/
        var[i][0]=x1[i]*x1[i];      /*设立 var 数组是为了减少计算量  */
        var[i][1]=x2[i]*x2[i];
        var[i][2]=x1[i]*x2[i];
        var[i][3]=x1[i];
        var[i][4]=x2[i];
        var[i][5]=1.0;
}

InitNGA ();                                                         /*初始化 NGA*/
prev = popul[best].evalu;
while ((TotalCount<COUNT)&&(scale>EPSILON))     /*终止条件*/
{
        if (count >= 500)
        {
                scale *= 0.1;
                count = 0;
        }

        for (i=0; i<PCross; i++)
        {
                Crossover ();
        }
```

```
        for (i=0; i<PMutation; i++)
        {
            Mutation ();
        }

        if (popul[best].evalu == prev)
            count ++;
        else
        {
            prev = popul[best].evalu;
            count = 0;
        }
        fprintf (stderr, "%6d (%2d)\t%6.4f\t[%2d]\t%12.10g\t[%2d]\t%12.10g\n", TotalCount,
            count, scale,best, popul[best].evalu, worst, popul[worst].evalu);  }
    t2 = time(NULL);
    for (i =0; i<CHROMSOME; i++)                    /*记录结果*/
    {
        fprintf(out, "The best value is : %10.5f", popul[best].chrom[i]);
    }
    fprintf(out, "\n");
    fprintf(out, "The best value is : %10.5f\n", popul[best].evalu);
    fprintf(stderr, "This computer time is : %ld", t2-t1);
    getchar();
}
```

以上，我们给出了数值遗传算法的基本程序，读者应该能够运行简单的遗传算法，并且在此基础上进一步改进，以适应自己的需要。

7.5 遗传算法的应用

遗传算法提供了一种求解复杂系统优化问题的通用框架，它不依赖于问题的具体领域，对问题的种类有很强的鲁棒性，所以广泛应用于很多学科。下面是遗传算法的一些主要应用领域：

(1)函数优化　函数优化是遗传算法的经典应用领域，也是对遗传算法进行性能评价的常用算例。很多人构造出了各种各样的复杂形式的测试函数，有连续函数也有离散函数，有凸函数也有凹函数，有低维函数也有高维函数，有确定函数也有随机函数，有单峰函数也有多峰函数等，人们常用这些几何特性各异的函数来评价遗传算法的性能。而对于一些非线性、多模型、多目标的函数优化问题，用其他优化方法较难求解，遗传算法却可以方便地得到较好的结果。

(2)组合优化　随着问题规模的扩大，组合优化问题的搜索空间急剧扩大，有时在目前的计算机上用枚举法很难或者甚至不可能得到其精确最优解。对于这类复杂问题，人们已意识到应把精力放在寻求其满意解上，而遗传算法则是寻求这种满意解的最佳工具之一。实践证明，遗传算法对于组合优化中的 NP 完全问题非常有效。例如，遗传算法已经在求解旅行商问题、背包问题、装箱问题、图形划分问题等方面得到成功的应用。

(3)生产调度问题　生产调度问题在许多情况下所建立起来的数学模型难以精确求解，即使经过一些简化之后可以进行求解，也会因简化太多而使得求解结果与实际相差甚远。因此，目前在现实生产中也主要靠一些经验进行调度。遗传算法已成为解决复杂调度问题的有效工具，在单件生产车间调度、流水线生产车间调度、生产规划、任务分配等方面遗传算法都得到了有效的应用。

(4)自动控制　在自动控制领域中许多与优化相关的问题需要求解，遗传算法的应用日益增加，并显示了良好的效果。例如用遗传算法进行航空控制系统的优化、基于遗传算法的模糊控制器优化设计、基于遗传算法的参数辨识、利用遗传算法进行人工神经网络的结构优化设计和权值学习，都显示出了遗传算法在这些领域中应用的可能性。

(5)机器人智能控制　机器人是一类复杂的难以精确建模的人工系统，而遗传算法的起源就来自于对人工自适应系统的研究，所以机器人智能控制理所当然地成为遗传算法的一个重要应用领域。例如遗传算法已经在移动机器人路径规划、关节机器人运动轨迹规划、机器人逆运动学求解、细胞机器人的结构优化和行动协调等方面得到研究和应用。

(6)图像处理和模式识别　图像处理和模式识别是计算机视觉中的一个重要研究领域。在图像处理过程中，如扫描、特征提取、图像分割等不可避免地会产生一些误差，这些误差会影响到图像处理和识别的效果。如何使这些误差最小是使计算机视觉达到实用化的重要要求。遗传算法在图像处理中的优化计算方面是完全胜任的。目前已在图像恢复、图像边缘特征提取、几何形状识别等方面得到了应用。

(7)人工生命　人工生命是用计算机等人工媒体模拟或构造出具有自然生物系统特有行为的人造系统。自组织能力和自学习能力是人工生命的两大主要特征。人工生命与遗传算法有着密切的关系，基于遗传算法的进化模型是研究人工生命现象的重要理论基础。虽然人工生命的研究尚处于启蒙阶段，但遗传算法已在其进化模型、学习模型、行为模型等方面显示了初步的应用能力。可以预见，遗传算法在人工生命及复杂自适应系统的模拟与设计、复杂系统突现性理论研究中，将得到更为深入的发展。

(8)遗传程序设计　Koza 发展了遗传程序设计的概念，他使用了以 LISP 语言所表示的

编码方法，基于对一种树型结构所进行的遗传操作自动生成计算机程序。虽然遗传程序设计的理论尚未成熟，应用也有一些限制，但它已有一些成功的应用。

(9)机器学习　学习能力是高级自适应系统所应具备的能力之一。基于遗传算法的机器学习，特别是分类器系统，在许多领域得到了应用。例如，遗传算法被用于模糊控制规则的学习，利用遗传算法学习隶属度函数，从而更好地改进了模糊系统的性能。基于遗传算法的机器学习可用于调整人工神经网络的连接权，也可用于神经网络结构的优化设计。分类器系统在多机器入路径规划系统中得到了成功的应用。

7.6 遗传算法在生物和化学中的应用

遗传算法在生物和化学中得到广泛应用，近几年这种势头不但没有减小，反而有上升的趋势，几乎在化学的所有领域里都能发现 GA 在起作用，据不完全统计，从 2000 年以来，国内各个领域的 GA 论文数已超过千篇，而国外在化学和生物类杂志上发表的 GA 论文就有 400 多篇。这充分说明 GA 作为一种强有力的优化方法，有很强的生命力，学习、掌握和运用 GA 是十分重要的。

GA 在高维和有很多局部最优的最优搜索中特别有用。问题的高维使得完全搜索不可能，局部最优的存在使得直接优化方法（如陡升设计 steepest ascent）不可靠，因为这种方法很容易陷于局部最优。

这些特点使得 GA 特别适合用于分子模拟，因为能量超曲面非常复杂，有很多局部最优。因此，使用“标准”的方法，优化的结构非常的依赖于初值。

对于一个中等大小的蛋白质（100 个残基），如果没有限制，构象的数目可能越是 25100（＝每个残基 5 个可旋转键×每个可旋转键 5 个可能的旋转角度+100），如果残基数大到 1 万，这样一个巨大的搜索空间，显然超越任何超级计算机的能力，但 GA 可以找到接近最优解。这意味着，尽管不能证明 GA 已经真正找到了最优解，GA 得到的一些解实际上超越了以前任何一种方法得到的解。这对于没有分析解的许多问题是非常有用。

最近，Jin 等有效的用 GA 从一个五肽的全局最小能量结构分析得到其骨架构象特征。

另外还有 Fontain, Mestres 和 Scuseria, Hermann 和 Suhai, Meza, Pullan, Niesse 和 Magne，Wang, Hartke 以及 Kariuki 等人的文章中都报道用 GA 用于构象搜索。Brodmeier 和 Pretsch 给出了 GA 用于分子模拟的一般描述。

GA 应用的另一个领域是计算机辅助分子设计（CAMD），Venkatasubramanian 和

Sundaram 给出了一般描述，Venkatasubramanian 等人将其应用在多聚物设计，Deivillers 给出了分子设计中其他应用的综述。

在 CAMD 中，基于分子亚组的结构特征的性质预测被定义为向前问题，和基于一系列期望的宏观性质的分子结构的构建被定义为向后问题。前一个问题 De Weijer 等人用基于神经网络的方法解决了，后者用 GA 很有效。

Venkatasubramanian 等人用神经网络、GA 杂交体系用于设计相当复杂的聚合物分子。Devillers 和 Putavy 等人用来设计具有生物降解能力的有机分子。Burden 等用来寻找更有活性的二氢叶酸还原酶抑制剂。Sundaram 等用来设计燃料添加剂。Hopfinger 和 Patel 描述了用 GA 建立可靠的定量构效关系（QSARs）和进行分子多样性实验。Meusinger 和 Moros 用 GA 用来测定烃的定量结构－辛烷级别关系。

另一个复杂的问题是获取受体位点的原子级别的模型。Walters 和 Muhammad 用 GA 来改变和优化原子的类型，以最大化计算所得的药物－受体亲和力与测量所得药物活性之间的关系。

Jones 等人报道 GA 在化学结构处理和分子识别上的三个应用。在第一个应用中，GA 通过三维结构数据库子结构搜索，在搜索三维小分子的构象空间来寻找药效团模型上面特别有效。在第二个应用中，GA 用在柔性配体与部分柔性的蛋白质位点的分子对接中。在第三个应用中，GA 被用来自动叠加柔性的分子。

GA 用在蛋白质折叠这个问题中，Schulze-Kremer 给出了一个很好的教程。Ebeling 和 Nadler 以及 Krasnogor 等人在这个领域有有论文发表。

Tuffèry 等人比较了用不同的搜索算法在优化蛋白质侧链构象的结果（模拟退火，简单和改良 GA 以及一个启发式的组合方法）。

据 Van Kampen 和 Buydens 所说，在阐明一个七肽的扭转角空间结构时，重组并不总是有效，因为杂交不能与构建块结合来产生改进的试验解。因此在这样的问题中 GA 基本上基于选择和点突变，有着更复杂的突变－选择方案的模拟退火优于 GA，其收敛约比 GA 快三倍。

Hunger 和 Huttner 将 GA 和神经网络结合起来用在优化和分析用于三角架金属模板的力场参数中。当用训练的经网络计算的结果部分取代由力场方法计算的结构而得到的适应值函数的评价时，可以节省大量时间。

Reijmers 等人用 GA 来构建 G 蛋白结合受体序列的系统树。

Clark 汇编了一个全面的和更新的关于“CADD 中的进化算法”参考文献列表。

7.7 遗传算法的讨论和发展

自从 1975 年 J．H．Hollad 系统地提出遗传算法的完整结构和理论以来，众多学者一直致力于推动遗传算法的发展，对编码方式、控制参数的确定、选择方式和交叉机理等进行了深入的探究，引入了动态策略和自适应策略以改善遗传算法的性能，提出了各种变形的遗传算法(Variants of Canonical Genetic Algorithms, 简称 VCGA)。其基本途径概括起来有下面几个方面：

①改变遗传算法的组成成分或使用技术，如选用优化控制参数、适合问题特性的编码技术等；

②采用混合遗传算法；

③采用动态自适应技术，在进化过程中调整算法控制参数和编码粒度；

④采用非标准的遗传操作算子；

⑤采用并行遗传算法。

下面我们就简单的讨论几种遗传算法的变化：

1 选择部分(Selection)

基因选择是遗传算法中比较关键的一步，它是从母体中选取个体形成繁殖库的过程，有时直接关系到收敛速度问题。基因选择有很多种方法，比如确定方法、轮盘赌法、贝努利实验法等，我们经常用到的模拟退火方法也在基因选择中发挥很重要的作用。在基因选择中禁止近亲繁殖，因为性能比较接近的个体间相互交叉、变异不利于优良性能的传播，会导致局部收敛问题的发生。

2 分层遗传算法

N 个低层遗传算法中的每一个在经过一段时间后均可以获得位于个体串上的一些特定位置的优良模式。通过高层遗传算法的操作，GAi(i＝l，2，…，N)可以获得包含不同种类的优良模式的新个体，从而为它们提供了更加平等的竞争机会。这种改进的遗传算法与并行或分布遗传算法相比，在上一层的个体交换上，它不需要人为地控制应交换什么样的个体，也不需要人为地指定处理器将传送出的个体送往哪一个处理器，或者从哪个处理器接受个体。这样改进的遗传算法不但在每个处理器上运行着遗传算法，同时对各处理器不断生成的新种群进行着高一层的运算和控制。

3 混合遗传算法

我们知道，梯度法、爬山法、模拟退火法等一些优化算法具有很强的局部搜索能力，而

另一些含有问题相关的启发知识的启发式算法的运行效率也比较高。如果融合这些优化方法的思想，构成一种新的混合遗传算法(hybrid genetic algorithm)，是提高遗传算法运行效率和求解质量的一个有效手段。目前，混合遗传算法实现方法体现在两个方面，一是引入局部搜索过程，二是增加编码变换操作过程。在构成混合遗传算法时，De Jong 提出下面三个基本原则：

①尽量采用原有算法的编码；

②利用原有算法全局搜索的优点；

③改进遗传算子。

4 其他一些改进

其他一些发展的遗传算法有 Eshelman 于 1991 年提出的 CHC 算法，Goldberg 等于 1989 年提出的 messy GA 算法，还有自适应遗传算法(Adaptive GA, AGA)，基于小生物环境的遗传算法以及并行遗传算法等。

7.8 其他优化算法

7.8.1 禁忌搜索

禁忌搜索(Tabu Search)是 F. Glove 提出的一种智能启发式最优化方法。它在许多问题上取得了优于其它方法的结果，正在引起人们的越来越大的重视。由于 Tabu 搜索的灵活性，使它很容易和其它方法及特定问题的具体知识结合起来，组成面向具体对象的实用算法。

近年来，Tabu 搜索研究发展很快，许多研究者不断将 Tabu 搜索算法用于新的多个领域，并提出了许多新的理论和设想。例如 Anderson 等人将 Tabu 用于电话线路的排列；Józefowska, Glove 等人用于非连续和连续行程安排；Bouju 等人将其用于无线电频率排列问题；还有一些如色彩的排列，食品的管理，机器的布置，资源的管理问题等等。

Tabu 算法从一个初始解开始搜索，当它在某个解 x 的邻域 NB(x)中搜索出最好的解 y 时，即使解 y 的质量没有 x 好，也将它确定为下次迭代时的初始解，以此来避免陷入局部最优解。为防止以后搜索时再回到 x，可以构造一个 Tabu 集合(表)，记录解的履历，使新的候选解只能从 NB(x)与 Tabu 集合 T 的差集中来选。

Tabu 算法的一般框架为：

Step 0：选一初始解 x，将 x 作为暂定解(最优解)。令 k=1，Tabu 表 T＝Φ。

Step k：

(1) 若 NB(x) – T = Φ，转(2)。否则，在 NB(x)中搜索局部最优解 y，若 y 不在 T 中或是在 T 中但满足激活条件，令 x = y。若 y 比暂定解好，将 y 作暂定解。K=k+1，转 Step k。

(2) 若满足终止条件，输出暂定解，算法终止。否则，修改 Tabu 表，k=k+1，转 Step k。

算法中设置的 Tabu 表和其它表或数组可以作为搜索过程的短期和中长期记忆。表的内容可以是：

(1) 最近搜索到的若干个解。

(2) 最近搜索到的解变量 x_i 的变化方向。在后继的搜索中禁止向相反的方向移动。

(3) 解变量 x_i 取某值的频度，从而使算法能够在有希望的区域内加强搜索，同时又有意识地向迄今尚未搜索过的区域移动，增强搜索的广泛性。

激活条件一般利用解的优度。在实际应用中，可以把各种先验知识和搜索的履历、解的性质等以禁止集合或其它形式记录下来，从而控制以后的搜索过程。

7.8.2 粒子群算法

粒子群算法（Particle Swarm Optimization，简称 PSO）源于对鸟类捕食行为的研究，也是一种启发式进化计算技术，由 Eberhart 博士和 Kennedy 博士所提出。

粒子群算法产生的背景是人工生命的研究。人工生命是来研究具有某些生命基本特征的人工系统。人工生命包含两方面的内容：

1）研究如何利用计算技术研究生物现象；

2）研究如何利用生物技术研究计算问题。

启发式优化算法的研究属于第二方面的内容。现在已经有许多源于生物现象的算法，比如，人工神经网络是简化的人类神经系统模型；遗传算法是对基因进化过程的模拟。同时，生物界里存在着另一种生物社会系统，更确切地说，是在简单个体组成的群落与环境以及个体之间的互动行为，也可以称之为“群智能”（Swarm intelligence）。这些模拟系统利用局部信息来产生某些不可预测的群体行为。在对群智能的模拟中，一些研究者已经得到了满意的结果，例如 Floys 和 Boids 通过模拟鱼群和鸟群的运动规律，成功地解决了计算机视觉和计算机辅助设计等方面的问题。

在基于群智能的优化算法中，目前主要有两种比较成功的算法：蚁群算法（Ant Colony Algorithm，简称 ACA）和粒子群算法。前者是对蚂蚁群落食物采集过程的模拟，已经成功地应用到很多离散问题的优化方面。粒子群算法也是源于对简单社会系统的模拟，最初设想是模拟鸟群觅食的过程，但后来人们发现粒子群算法是一个很好的优化工具。

粒子群算法模拟鸟群的捕食行为。设想一群鸟在随机地搜索食物，在这个区域只有一块食物，所有的鸟都不知道食物在哪里，但是它们知道当前的位置离食物还有多远。那么找到食物的最优策略是什么呢？最简单有效的就是搜寻目前离食物最近的鸟的周围区域。

粒子群算法从这种模型中得到启示并用于解决优化问题。粒子群算法中，每个优化问题的解都是搜索空间中的一只鸟，我们称之为“粒子”。所有的粒子都有一个由被优化的函数所决定的适应值，每个粒子还有一个速度来决定它们飞翔的方向和距离，然后所有的粒子就追随当前的最优粒子在解空间中进行搜索。

粒子群算法初始化为一群粒子（随机解），然后通过迭代来寻找最优解。在每一次迭代中，粒子通过跟踪两个“极值”来更新自己的位置。第一个就是粒子本身所找到的当前最优解，这个解叫做个体极值 pBest。另一个极值是整个群体中目前所找到的最优解，这个极值是当前全局最优 gBest。在找到这两个最优值时，粒子根据如下公式(5-1)来更新自己的速度和新的位置：

$$V_{id} = w \times V_{id} + c_1 \times rand() \times (x_{id,pBest} - x_{id}) + c_2 \times rand() \times (x_{gBest} - x_{id})$$

其中，V 是粒子的速度，xid 为编号为 id 的粒子的当前位置，pBest 和 gBest 如前定义。

rand()为介于(0, 1)之间的随机数，c1 和 c2 为学习因子，w 为速度递减惯性因子。

另外，在变量空间的每一维上，粒子的最大速度被限制在一个参数 Vmax 之内。如果某一维更新后的速度超过设定的 Vmax，那么这一维的速度将被限定为 Vmax。

粒子群算法一提出就吸引了广泛的注意，各种关于粒子群算法应用研究的成果不断涌现，有力地推动了粒子群研究。粒子群算法的应用领域可以划分为：函数优化、神经网络训练、工业系统优化与控制以及其他遗传算法的应用领域等。许多实际问题都可以归结为函数优化问题，粒子群算法用于一般的函数优化问题并不令人意外。值得注意的是将粒子群算法用于各种复杂的优化问题也已经取得了一些进展，例如，粒子群算法用于求解 TSP 问题；用于在噪声和动态环境下的优化问题；用于多目标优化问题等，都取得了令人感兴趣的结果。粒子群算法也被应用于训练集单元神经网络进行模式分类，取得了很好的效果，显示出粒子

群算法是一种很有希望的训练神经网络的手段。粒子群算法还被应用于训练模糊前向神经网络进行模式分类，从而从网络的输出中抽取规则，也取得了满意的效果；以及用于实际系统的模拟和控制问题，也都得到了很好的效果。

7.8.3 人工蚁群算法

人工蚁群算法是一种新型的模拟进化算法。该算法是由意大利学者 M.Dorigo、V.Maniez-zo、A.Colorini 等人首先提出的，称之为蚁群系统（Ant Colony System），并应用该算法求解 TSP 问题、分配问题、job-shop 调度问题，取得了较好的结果。

人工蚁群算法是受到对真实的蚁群行为的研究的启发而提出的。仿生学家经过大量细致观察研究发现，蚂蚁个体之间是通过一种称之为外激素（pheromone）的物质进行信息传递的。蚂蚁在运动过程中，能够在它所经过的路径上留下该种物质，而且蚂蚁在运动过程中能够感知这种物质，并以此指导自己的运动方向，因此,由大量蚂蚁组成的蚁群的集体行为便表现出一种信息正反馈现象：某一路径上走过的蚂蚁越多，则后来者选择该路径的概率就越大。蚂蚁个体之间就是通过这种信息的交流达到搜索食物的目的。

蚁群算法具有如下优点：

1）较强的鲁棒性：经典的蚁群模型稍加修改，便可以应用于其它问题；

2）分布式计算：蚁群算法是一种基于种群的进化算法，具有本质并行性，易于并行实现；

3）易于与其它方法结合：该方法可以与多种启发式算法结合，以改善算法的性能；该算法的一个缺陷是计算时间较长，随着计算机的发展和计算速度的提高，可以弥补这一缺陷。

7.8.4 免疫算法

免疫算法来自于对免疫系统的模拟，免疫系统是抵抗细菌、病毒和其它致病因子入侵的基本防御系统。免疫系统通过一套复杂的机制来重组基因，以产生抗体对付入侵的抗原，达到消灭抗原的目的。为了有效地提供防御功能，免疫系统必须进行模式识别，把自身的分子和细胞与抗原区分开来。除了具有识别能力之外，免疫系统与其它低级生物防御系统的区别在于它能够学习，并且有记忆能力。正是因为拥有上述特点，免疫系统对同一抗原的防御反应，第二次比第一次来得更快、更强烈。

免疫算法与一般的确定性优化算法相比，有以下显著特点：

1）它同时搜索解空间中的一系列的点，而不只是一个点；

2）它处理的对象是表示待求解的参数的编码数字串，而不是参数本身；

3）它使用的是目标函数本身，而不是其导数或其它附加信息；

4）它的变化规则是随机的，而不是确定的。

免疫算法与其它非确定性算法（如遗传算法、进化策略等）之间有如下的区别：

1）它在记忆单元基础上运行，有利于快速收敛于全局最优解；

2）它有计算亲和性（Affinity）的程序，反映了真实的免疫系统的多样性（Diversity）；

3）它通过促进或抑制抗体的产生，体现了免疫反应的自我调节功能。

人工免疫系统中免疫算法已经用于机器学习、异常和故障诊断、机器人行为仿真、机器人控制、网络入侵检测、神经网络设计、谱分析、参数优化、工业设计和生产等领域，表现出较卓越的性能和效率。

7.8.5 其他算法

进化策略是从二十世纪六十年代由德国的Rechenberg等人发展起来的。进化策略与遗传算法相似：随机产生大量的基因，给出一种进化函数，通过进化运算——交叉和变异运算重组，将那些最合适的保留下来。进化策略在非线性优化、系统辨识等方面显示了其优于经典方法的效果，已被应用于机器学习，机器人控制等领域，以解决较困难的统计优化问题，引起了人工智能、计算科学等领域专家的重视。进化策略方法的具体实现在此不再赘述。

进化规划是1990年Fogel提出的。进化策略也与遗传算法相似：随机产生大量的基因，给出一种进化函数，通过进化运算——变异（进化规划中只有一种运算：变异），将那些最合适的保留下来。目前，进化规划已成为系统辨识、优化的有效工具，被广泛应用于机器学习、神经控制器构建等，以解决较困难的统计优化问题，取得了较好的效果。进化规划方法的具体实现在此不再赘述。

模拟退火算法最初由Kirkpatrick提出。这种算法主要是通过模拟物理力学系统在降低系统内能时状态变迁这一物理过程实现对复杂多变量目标函数的优化求解。模拟退火最初由Kirkpatrick 应用于集成电路布局布线设计，后来又被成功地应用于求解TSP，以及图像分割、图像处理和模式识别等工程应用问题。现在模拟退火已经发展成为非线性多变量函数优化极为有效的求解算法。模拟退火方法的具体实现在此不在赘述。

第八章 分子对接

8.1 计算机辅助药物设计概述

从 1964 年 Hansch 和 Fujita 经典定量构效关系方法起，计算机辅助药物设计(Computer-aided drug design, CADD)已经有三十多年的历史了。近年来，随着分子生物学和结构生物学的发展，许多具有药理作用的生物大分子的三维结构已经被测定，在此同时，计算机科学也得到了长足的进步，不但发展了用于大规模并行计算的超级计算机，而且图形工作站和个人计算机也有巨大的发展。在此基础上，科研工作者也发展了用于化学和生物学研究的各种功能先进的计算方法。因此，计算机辅助药物设计方法呈现了突飞猛进之势，已从原来的理论研究发展成为一门实用型的学科，许多经过计算机设计的或计算机参与设计的药物已经上市或进入了临床研究。计算机辅助药物设计已经和传统的药物设计一样，成为了药物设计中最重要的手段之一。因此，近年来应用各种理论计算方法和分子模拟技术，进行计算机辅助药物设计，已成为国际上十分活跃的研究领域。

计算机辅助药物设计的方法可分为一下三类：

1)、在药物小分子构效关系基础上的药物设计

大多数药物作用的受体生物大分子三维结构现在还不清楚，在这种情况下，一般以小分子的结构和活性为基础，对一系列化合物进行定量构效关系(quantitative structure activity relationship, QSAR)和三维定量构效关系(3D-QSAR)研究，得到预测能力较强的QSAR/3D-QSAR 模型，对设计化合物的活性进行预测。3D-QSAR 模型还可以给出结构改造的信息，这使新化合物的设计目标更为明确。其主要思路是：运用构象分析和分子模拟技术，得到化合物的活性构象，结合构效关系研究结果，建立药效基团和三维药效基团模型；然后，运用药物设计方法，如数据库搜索和全新药物设计方法，设计新的化合物。

2)、以受体的三维结构为基础的药物设计

近年来，随着分子生物学和结构生物学的发展，许多具有药理作用的生物大分子的三维结构已经被测定，或者现在还未测定，但其一级结构已经测定，可以用同源蛋白模建的方法建立其三维结构模型。在这种情况下，我们可以用基于结构的药物设计方法(Structure-based drug design, SBDD)设计新的先导化合物。一般是先根据受体的三维结构，用分子模拟或理论计算方法研究药物和生物大分子的作用方式，得到生物大分子特别是其活性部位或结合部

位的详细结构性质，如静电作用、疏水作用和氢键结合等区域的分布。然后，用全新药物设计或数据库搜索的方法，设计新的化合物。

3)、与组合化学相对应的计算机辅助药物设计

组合化学(Combinatorial chemistry)的建立和发展，也推动了计算机辅助药物设计方法的发展，随之也产生了计算机模拟组合化学方法。用分子模拟和计算机技术设计合成组合样品库的构造块(building block)、根据分子多样性(molecular diversity)评价样品库的质量，或者建立虚拟组合样品库。同时，高通量筛选所产生的大量信息也必须用计算机来处理。

8.2 基于结构的药物设计

基于结构的药物设计，又被称为合理药物设计方法(rational drug design)，是利用包含在受体生物大分子的三维结构中的信息和与相关的配体关系来进行药物设计的方法。

其中受体生物大分子的三维结构是通过 X 射线衍射以及核磁共振等技术测得，近十年来，许多制药公司发表的研究结果表明，蛋白质的三维结构信息在新药的发现过程中起重要作用。蛋白质的三维结构库在可以在互联网上查寻，如：http://www.rcsb.org/pdb 等等，也可以采用同源模建方法确立其结构。基于结构的药物设计可分为分子对接和全新药物设计两种，其基本途径如图 10-1 所示：

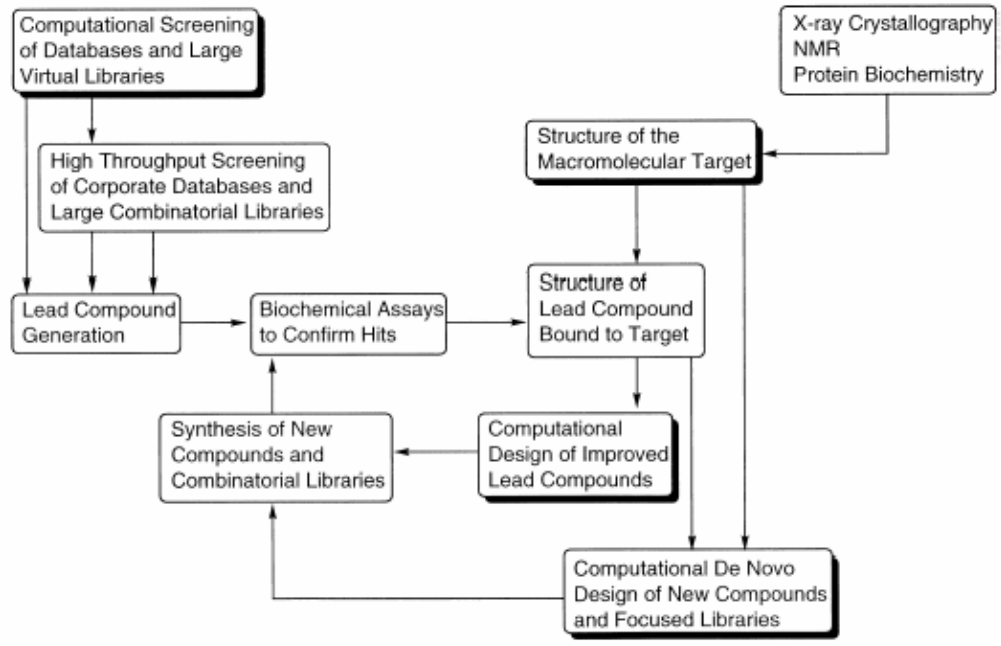


图 10-1 基于结构的药物设计流程图

1)、全新药物设计(*de novo* drug design)

全新药物设计是根据受体活性部位的形状性质要求，自动构建出形状性质互补的新分子。全新药物设计的计算方法有三种类型：碎片定位法(Fragment positioning methods)；分子生长法(Molecular growth methods)；碎片法结合数据库搜索(Fragment methods coupled to database searches)。

碎片定位法有两个著名的程序，Goodford 的 GRID 和 Miranker 和 Evensen 等人的 MCSS(Multiple Copy Simultaneous Search)；其他还有如 Gillet 等人的 HIPPO、Bohm 的 LUDI 等等。分子生长法也有很多计算方法，包括 DeWitte 等人的 SmoG(Small Molecule Growth)、Bohacek 和 McMartin 等人的 GrowMol、Rotstein 和 Murcko 等人的 GroupBuild、Moon 和 Howe 等人的 Grow 等等。碎片法结合数据库搜索的程序有 Eisen 等人的 HOOK 等等。

2)、分子对接(Molecular Docking)

分子对接法就是将小分子配体放置于受体的活性位点处，并寻找其合理的取向和构象，使得配体(ligand)与受体(receptor)的形状和相互作用的匹配最佳。在药物设计中，分子对接法主要是用来从小分子数据库里搜寻与受体生物大分子有较好亲和力的小分子，进行药理测试，从中发现新的先导化合物。这是本文研究的重点。

8.3 分子对接

天然蛋白质在生物进化的过程中学会了有选择地与细胞组分相互作用，形成确定的、紧密的结构，如给体－受体、酶－底物、酶－抑止剂、抗体－抗原的结合等，这种专一性的结合是蛋白质的特性，它决定了蛋白质的生物功能。因此，生物体系分子间的相互作用模式或结合模式一直是生物学家最为关心的课题之一，是一个具有重大理论和实际意义的基础性问题。它的研究对于了解生物分子间的识别机理、蛋白质结构和功能之间的关系和进行新药的设计都是非常重要的。

蛋白质和配体的对接模拟计算是二十世纪 70 年代末发展起来的方法，现在已经吸引了越来越多的生物学研究人员和化学计量学工作者的加入。近二十年来，随着计算机运算能力的大幅度提高、越来越多蛋白质序列和结构数据的获得。人们对生物分子的加深了解以及更加精确的原子相互作用模型的建立，分子对接的研究有了重大的发展。

分子对接法就是将小分子配体放置于受体的活性位点处，并寻找其合理的取向和构象，使得配体与受体的形状和相互作用的匹配最佳。在药物设计中，分子对接法主要是用来从小

分子数据库里搜寻与受体生物大分子有较好亲和力的小分子，进行药理测试，从中发现新的先导化合物。药物分子与受体结合时，必须遵循以下互补匹配规则：1）几何形状互补匹配；2）静电相互作用互补匹配（正电荷对应负电荷）；3）氢键相互作用互补匹配（氢键供体对应氢键受体）；4）疏水相互作用互补匹配（疏水区对应疏水区）。

所有分子对接计算都主要围绕着两个方面进行：一个是有效的全局优化过程的建立，另一个是用于判断实际分子构型的评价函数的确定。它们是任何旨在对蛋白质配体复合物进行结构预测的研究中最为重要同时也是问题最多的两个方面。前者要能够为具有几个到几十个变量的各项函数找到一个适当地全局最优值，而后者既要计算省时以便于构型搜索过程，又要足够精确以保证能够得到单一的天然构型。

8.3.1 分子对接程序

Dock 是第一个分子对接程序，是由加利福尼亚州立大学旧金山分校的 Kuntz 小组于 1982 年开发，最新版本为 5.0。Dock 1.0 考虑的是配体和受体之间的刚性形状对接；2.0 引入了“分而治之”算法，提高了计算速度；3.0 采用了分子力场势能函数作为评价函数；3.5 引入了打分函数优化以及化学性质匹配等；4.0 开始考虑配体的柔性。其他的对接方法有 McMartin 和 Bohacek 的 FLO98，Scripps 研究所 Olsen 小组的 Autodock，Welch 等人的 Hammerhead，Rarey 和 Kramer 等人的 FLEXX 等等。

近两年来，分子对接有着很大的进展，主要集中在分子的柔性方面，Zhu等人的F-DycoBlock主要是增加了蛋白质的柔性，Dock 4.0和5.0版本开始考虑分子的柔性，并且利用逐步增长的结构构建来处理分子的单轴旋转。Paul和Rognan的ConsDock作了100个蛋白质的测试集，60%的时候在第一次搜索得到了最好的解(Rank 0)。而TreeDock主要在半经验能量计算作了一些研究；Cai等人用球谐函数表面校正来进行分子对接等等。

8.3.2 分子对接的应用

分子对接不仅在理论上有了很大进展，它也在许多方面得到了应用。首先主要集中在新药发展上，利用分子对接设计新药，已经成为新药设计的一个主要途径，许多利用分子对接而进行库搜索研制的新药已经上市。目前，已在艾滋病、老年痴呆症、心血管等疾病的药物研制中得到了很大的应用。此外，分子对接在酶的研究，蛋白质网络研究中也具有重要的补充作用。

8.4 分子对接程序 AutoDock

分子对接法主要取决于两个方面：一是全局搜索算法，另一就是分子间结合自由能的评价函数。而 Scripps 研究所的分子对接程序 Autodock 采用了半经验的自由能公式作为评价函数，搜索算法采用了模拟退火（SA）和有局部搜索的遗传算法（GA-LS）两种。本文中利用 Autodock 3.0，将 NGA-TS 应用于分子对接中，评价函数采用 Autodock 中的半经验的自由能公式，而全局搜索算法采用 NGA-TS，形成了一个新的分子对接程序。并将计算结果与原始的 Autodock 3.0 计算的结果作了对比，取得了一定的进展。

8.4.1 Autodock 简介

AutoDock是一套应用新颖的和健壮的自动对接方法来预测柔性配体和已知结构的大分子目标（如酶、核酸等）之间相互作用的程序包，也可以用来预测蛋白质－蛋白质复合物的结构，目标是模拟小分子配体(如多巴胺)与大分子受体(如多巴胺能神经元中的DA受体)之间的分子自动对接过程。它最初是由David S. Goodsell用FORTRAN-77在Arthur J. Olson实验室写成，它现在由Scripps研究所的David S. Goodsell和Garrett M. Morris等利用Ansi C++设计，在SGI工作站上运行，现已发展到3.0版本，包含三个相互关联又相互独立的过程Autodock、Autogrid和Autotors，已经分发到全世界210多个学院站点上。在预报酶和抑制剂复合物，缩氨酸和抗体复合物，蛋白质和蛋白质的相互作用都取得了很大的成功。

在任何一种分子对接方案中必须协调一对矛盾，即健壮、精确的对接过程和要求计算结果保持在一个可以解释的水平之间的矛盾。理想的过程应该是搜索所有可能的自由度，最后找到目标蛋白质和配体之间相互作用能的全局最小值。但这样需要耗费大量的计算时间，为此必须简化对接过程。现在最常用的手段是采用人工辅助的对接方式，一边通过交互式对话控制可能的自由度，限制搜索空间等；一边使用自动方法如穷尽搜索，距离几何法等，研究人员只需要指定在蛋白质周围的三维搜索空间配体的可旋转键和起始搜索位置。

Autodock使用一种具有原子解析度的、基于网格的方法快速计算能量，使用Monte Carlo模拟退火算法、遗传算法进行结构搜索。Autodock采用手工辅助的方式，通过参数文件对对接过程进行控制。Autodock程序处理了一批受体－配体复合物后，就可以由分子动力学和它们的连接作用能，使用自由能模拟评价它们之间的相对稳定性。

8.4.2 评价函数的计算

1、Grid Maps

Autodock的快速能量计算是通过由预先计算配体分子中每种类型原子与生物大分子相互作用的能量网格(Grid Maps)。在过程Autogrid中，大分子周围一定体积空间被划分为三维网格，在每个三维网格点上放一个探针原子(Probe atom)，生物大分子和单个探针原子的相互作用能赋给这个网格点，如图8.1。对配体中的每种类型探针原子(典型的有C、O、N、H等)及带一个正电荷的探针电荷分别计算吸引势能。原子与蛋白质的作用能就可以通过它所在网格周围8个网格点的原子吸引势能，由三线性插值得到；电荷和蛋白质的作用能，则先由周围8个网格点的电荷势能通过三线性插值再简单地乘上电荷数。这样就可以计算出配体与生物大分子之间的作用能。

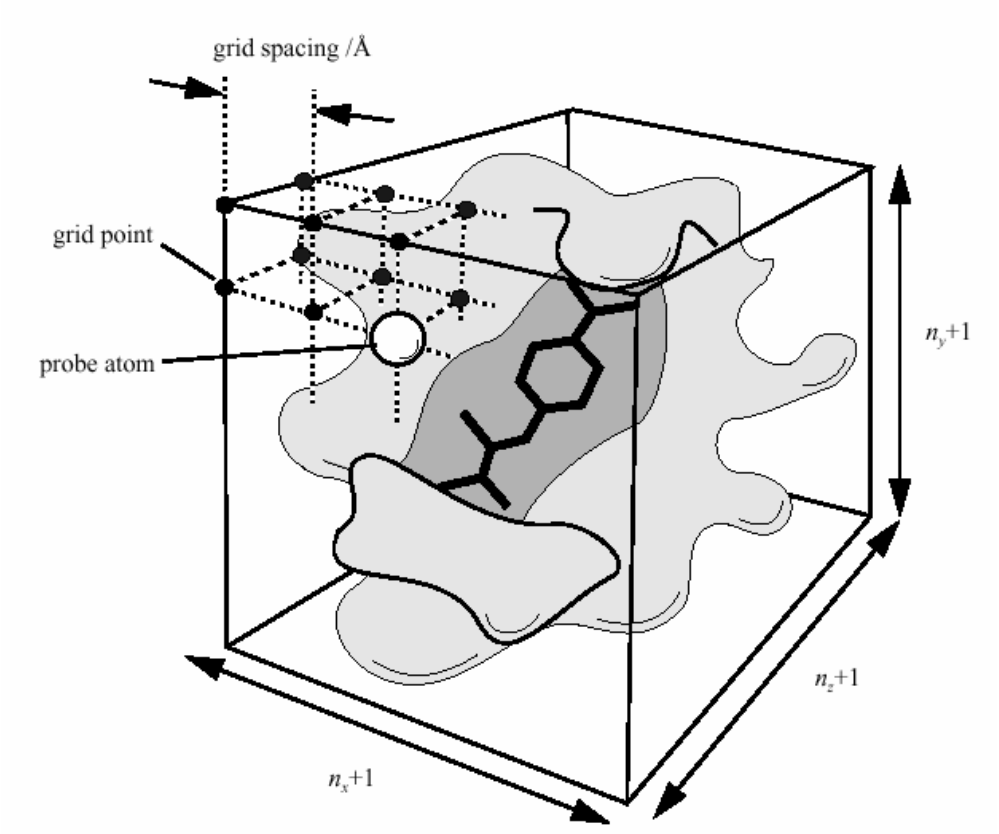


图 8.1 Grid Map

2、Van der Waals 势能

相互作用的势能函数 *V(r)*可以有以下公式来描述：

$$V(r) = \frac{Ae^{-br}}{r} - \frac{C_6}{r^6} \tag{8.1}$$

其中 A、b、C_6 是常数，函数图形如图 8.2 所示，r_{eqm} 是平衡核间距，ε 是平衡时的势阱深度。

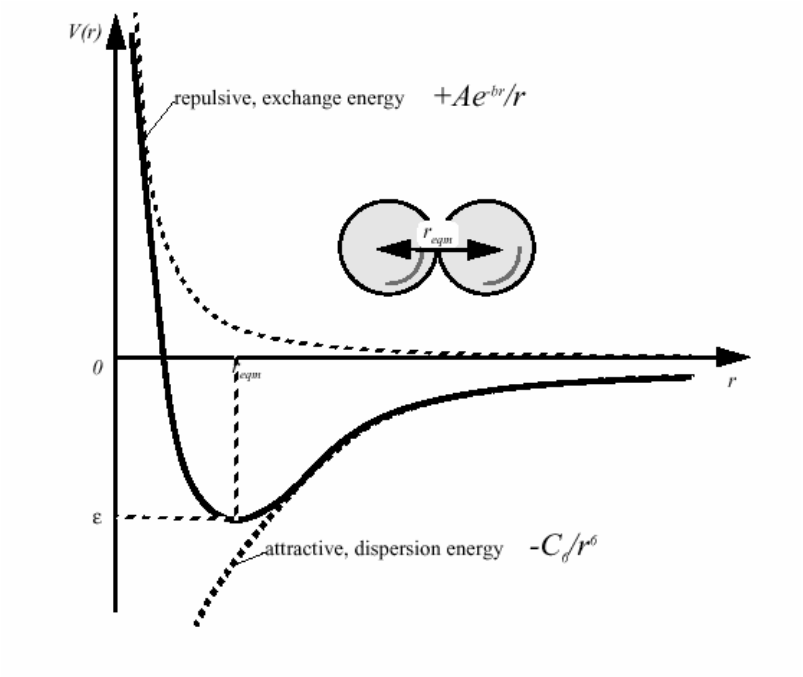


图 8.2

其中，交换能量可简化为：

$$\frac{A}{r}e^{-br} \approx \frac{C_{12}}{r^{12}} \tag{8.2}$$

所以原子间的能量可简化为：

$$V(r) \approx \frac{C_n}{r^n} - \frac{C_m}{r^m} = C_n r^{-n} - C_m r^{-m} \tag{8.3}$$

其中n和m是整数，C_n和C_m是常数，它们的值取决于平衡时势阱的深度及原子间的平衡核间距。一般来说12-6的Lennard-Jones参数(n=12, m=6)常用来对Van der Waals力建模，12-10的参数(n=12, m=10)常用来对氢键建模。

1)、自相关的Lennard-Jones 参数

可以通过如下方法计算出一套自相关的参数，任意给定一对原子，先分别求出它们的Van der Waals半径和阱深，令$r_{eqm,XX}$是两个相同原子间的平衡核间距，e_{XX}是它们平衡时的势能。那么两个不同类型原子X、Y间的平衡核间距和势能分别为：

$$\begin{aligned} r_{eqm,XY} &= \frac{1}{2}(r_{eqm,XX} + r_{eqm,YY}) \\ \varepsilon_{XY} &= \sqrt{\varepsilon_{XX}\varepsilon_{YY}} \end{aligned} \tag{8.4}$$

有时候Lennard-Jones势能可用参数σ表示为：

$$r_{eqm,XY} = 2^{\frac{1}{6}}\sigma \tag{8.5}$$

这样，Lennard-Jones势能就成为：

$$V_{12-6}(r) = 4\varepsilon_{XY}\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{8.6}$$

因此，系数C_{12}和C_{12}分别为：

$$\begin{aligned} C_{12} &= \varepsilon_{XY} r_{eqm,XY}^{12} \\ C_{6} &= 2\varepsilon_{XY} r_{eqm,XY}^{6} \end{aligned} \tag{8.7}$$

2)、一般关系式

我们可以推导出Lennard-Jones 系数、平衡核间距及阱深之间的一般关系。在平衡核间距r_{eqm}时势能最小，即$V(r_{eqm})=-\varepsilon$。势能最小时导数为0：

$$\frac{dV}{dr} = -\frac{nC_n}{r^{n+1}} + \frac{mC_m}{r^{m+1}} = 0 \tag{8.8}$$

因此，

$$\frac{nC_n}{r^{n+1}} = \frac{mC_m}{r^{m+1}} \tag{8.9}$$

即

$$C_m = \frac{nC_n r^{m+1}}{mr^{n+1}} = \frac{n}{m} C_n r^{(m-n)} \tag{8.10}$$

将C_m代人以前求出的$V(r)$中，得到：

$$-\varepsilon = \frac{C_n}{r_{eqm}^n} - \frac{nC_n r_{eqm}^{(m-n)}}{mr_{eqm}^m} \tag{8.11}$$

经过重排计算我们可以得到如下公式：

$$\boxed{C_n = \frac{m}{n-m}\varepsilon r_{eqm}^n} \tag{8.12}$$

再代回去得到：

$$\boxed{C_m = \frac{n}{n-m}\varepsilon r_{eqm}^m} \tag{8.13}$$

3)、**一些典型原子间的自相关Lennard-Jones参数**

一般原子间的Van der Waals力的计算采用12-6的Lennard-Jones参数，表11-1列出了一些

典型原子间的自相关Lennard-Jones参数，这些数据主要参考了Autodock网站，C_{12}和C_6数值是计算得来，有效数字位数比较高。其他一些原子如P、F、Cl、Br、I、Fe等的参数在网上可以查的到。

表8.1 一些自相关的Lennard-Jones 12-6参数

Atoms i-j	$R_{eqm,ij}$/ Å	E_{ij}/ kcal·mol^{-1}	C_{12}/ kcal·mol^{-1}Å^{12}	C_6/ kcal·mol^{-1}Å^{12}
C-C	4.00	0.150	2516582.400	1228.800000
C-N	3.75	0.155	1198066.249	861.634784
C-O	3.60	0.173	820711.722	754.059521
C-P	4.10	0.173	3908111.160	1645.484377
C-S	4.00	0.173	2905899.052	1418.896022
C-H	3.00	0.055	29108.222	79.857949
N-C	3.75	0.155	1198066.249	861.634784
N-N	3.50	0.160	540675.281	588.245000
N-O	3.35	0.179	357365.541	505.677729
N-P	3.85	0.179	1897159.056	1165.116525
N-S	3.75	0.179	1383407.742	994.930149
N-H	2.75	0.057	10581.989	48.932922
O-C	3.60	0.173	820711.722	754.059521
O-N	3.35	0.179	357365.541	505.677729
O-O	3.20	0.200	230584.301	429.496730
O-P	3.70	0.200	1316590.401	1026.290564
O-S	3.60	0.200	947676.268	870.712934
O-H	2.60	0.063	6035.457	39.075098
P-C	4.10	0.173	3908111.160	1645.484377
P-N	3.85	0.179	1897159.056	1165.116525
P-O	3.70	0.200	1316590.401	1026.290564
P-P	4.20	0.200	6025893.897	2195.612698
P-S	4.10	0.200	4512698.060	1900.041696
P-H	3.10	0.063	49816.168	112.261323
S-C	4.00	0.173	2905899.052	1418.896022
S-N	3.75	0.179	1383407.742	994.930149
S-O	3.60	0.200	947676.268	870.712934
S-P	4.10	0.200	4512698.060	1900.041696
S-S	4.00	0.200	3355443.200	1638.400000
S-H	3.00	0.063	33611.280	92.212017
H-C	3.00	0.055	29108.222	79.857949
H-N	2.75	0.057	10581.989	48.932922
H-O	2.60	0.063	6035.457	39.075098
H-P	3.10	0.063	49816.168	112.261323
H-S	3.00	0.063	33611.280	92.212017
H-H	2.00	0.020	81.920	2.560000

3、氢键作用力

氢键作用力的计算采用12-10的Lennard-Jones参数，一些典型氢键原子间的12-10 Lennard-Jones 参数见表8.2：

表8.2 一些典型氢键原子间的12-10 Lennard-Jones 参数

Atoms i-j	$R_{eqm,ij}$/Å	E_{ij}/kcal·mol^{-1}	C_{12}/ kcal·mol^{-1}Å^{12}	C_{10}/ kcal·mol^{-1}Å^{12}
N-H	1.90	5.00	55332.873	18393.199
O-H	1.90	5.00	55332.873	18393.199
S-H	2.50	1.00	298023.224	57220.459

4、静电作用力

静电作用力可由Autogrid程序或其他软件如MEAD、DELPHI等计算，Autogrid程序可以计算大分子和探针电荷e+1.60219×10^{-19}C之间的库仑作用力。计算静电作用力时不需考虑距离因素，在溶液中介电函数与距离有关，根据Mehler和Solmajer的工作，溶液中介电函数可用下面sigmoid型的函数表示：

$$\varepsilon(r) = A + \frac{B}{1 + ke^{-\lambda Br}} \tag{8.14}$$

其中$B = \varepsilon_0 - A$；ε_0是纯水的介电常数，在25℃时ε_0 = 78.4；参数A = -8.5525, λ = 0.003627, k = 7.7839。

Autogrid程序要求原子的电荷数必须以PDBQ格式保存在文件中。PDBQ格式是在标准PDB格式的基础上，另加一列用来记录原子的部分电荷数。原子的部分电荷数是用分子建模程序如：Insight II、SYBYL等计算出来的。

5、经验结合自由能

在Autodock 3.0中，采用经验结合自由能作为评价函数。经验结合自由能包括以下五项：

$$\Delta G = \Delta G_{vdw} + \Delta G_{hbond} + \Delta G_{elec} + \Delta G_{conform} + \Delta G_{tor} + \Delta G_{sol} \tag{8.15}$$

在上式中，前四项为范德华力（色散/排斥力）、氢键作用力、静电作用力和背离实际键长和键角的偏差，△G_{tor}为内部转动、全局转动和平动的限制，△G_{sol}为疏水作用能。

应用 Wesson 和 Eisenberg 的热力学循环方法，上面公式可以近似为以下包含五项的公式：

$$\begin{aligned}\Delta G = {} & \Delta G_{vdw}\sum_{i,j}\left(\frac{A_{ij}}{r^{12}} - \frac{B_{ij}}{r^{6}}\right) + \Delta G_{hbond}\sum_{i,j}E(t)\left(\frac{C_{ij}}{r^{12}} - \frac{D_{ij}}{r^{10}}\right) \\ & + \Delta G_{elec}\sum_{i,j}\frac{q_i q_j}{\varepsilon(r_{ij})r_{ij}} + \Delta G_{tor}N_{tor} + \Delta G_{sol}\sum_{i,j}\left(S_i V_j + S_j V_i\right)e^{(-r_{ij}^2/2\sigma^2)}\end{aligned} \tag{8.16}$$

在这里右式的五项△G为与实验拟合得出的各项系数，可以通过已知配体－蛋白质复合

物的结合常数计算得到。右边第一项是配体与受体之间的Van der Waals作用对结合自由能的贡献；第二项是氢键作用对结合自由能的贡献；第三项是静电作用对结合自由能的贡献；第四项是配体的内部旋转自由能被冻结引起的结合自由能变化；第五项是配体与受体结合时的去溶剂化效应对结合自由能的贡献。

为了实现配体的内部旋转和去溶剂化效应对自由能的贡献，在程序计算时，大分子在PDBQ格式的基础上，还要加上参数变为PDBQS格式的文件。

6、三线性插值

Autodock使用基于网格的快速能量计算方法，对于网格内空间任意一点它的能量，可由包围它的网格的8个顶点的能量通过三线性插值法得到。设点在x 方向上处在第u_0和u_1网格点之间，在y方向上处在第v_0和v_1网格点之间，在z方向上处在第w_0和w_1网格点之间，三线性插值法的具体计算步骤如下：

1). 计算点的网格位置记为(u, v, w)，则：

$$\begin{aligned} u_0 &= \mathrm{int}(u) & u_1 &= u_0 + 1 \\ v_0 &= \mathrm{int}(v) & v_1 &= v_0 + 1 \\ w_0 &= \mathrm{int}(w) & w_1 &= w_0 + 1 \end{aligned} \tag{8.17}$$

2). 计算各权重

$$\begin{aligned} p_{0u} &= u - u_0 & p_{1u} &= 1 - p_{0u} \\ p_{0v} &= v - v_0 & p_{1v} &= 1 - p_{0v} \\ p_{0w} &= w - w_0 & p_{1w} &= 1 - p_{0w} \end{aligned} \tag{8.18}$$

3). 计算点(u, v, w)处的能量

$$\begin{aligned} E(u,v,w) = {} & p_{1u}p_{1v}p_{1w}E(u_0,v_0,w_0) \\ & + p_{1u}p_{1v}p_{0w}E(u_0,v_0,w_1) \\ & + p_{1u}p_{0v}p_{1w}E(u_0,v_1,w_0) \\ & + p_{1u}p_{0v}p_{0w}E(u_0,v_1,w_1) \\ & + p_{0u}p_{1v}p_{1w}E(u_1,v_0,w_0) \\ & + p_{0u}p_{1v}p_{0w}E(u_1,v_0,w_1) \\ & + p_{0u}p_{0v}p_{1w}E(u_1,v_1,w_0) \\ & + p_{0u}p_{0v}p_{0w}E(u_1,v_1,w_1) \end{aligned} \tag{8.19}$$

8.4.3 程序的编译和运行

Autodock 3.05 版本的源程序可以从 Scripps 研究所得到，是用 GNU C++语言编写，在

SGI 工作站的 UNIX 下运行。其中 Autodock 有 85 个 C++源文件和 92 个头文件，Autogrid 有 12 个 C 语言源文件和 6 个头文件，还有象 Autotors、Addsol 等工具加起来大约有 1M 多的源程序。

每个分子的对接计算需要做以下步骤地准备：

1)、受体大分子的准备

利用网上 PDB 库里下载的蛋白质大分子，利用 SYBYL 和 Weblab Viewer 等软件，除去配体小分子和水分子，加入极性氢原子，计算出电荷分布，同时利用一些工具程序将 PDB 文件转化为 PDBQ 文件，然后利用程序 Addsol 加入去溶剂化效应常数，将 PDBQ 文件转化为 PDBQS 文件。

2)、配体小分子的准备

将药物分子或者蛋白质复合物中的小分子计算出电荷分布，产生一个 Mol2 文件格式的药物分子，然后利用 Autotors 程序产生一个 PDBQ 格式的文件，其中文件中定义了可旋转键。

3)、参数文件的准备

Autogrid 程序运行需要一个 GPF 的参数文件，包含了大分子的一些信息和原子间相互作用的一些参数，Autodock 程序运行需要一个 DPF 的参数文件，包含了 Grid 文件、配体分子以及搜索算法的一些信息。利用 mkgpf3 和 mkdpf3 这两个工具产生 GPF 文件和 DPF 文件。

4)、程序的运行和分析

准备好所有的文件后，运行 Autogrid 和 Autodock 文件，产生一个包含对接信息的输出文件，通过对输出文件的分析，就可以得出大分子和配体小分子对接的一些信息，如对接后的结合自由能、对接位置、预测误差等信息。

第九章 HMM 算法

隐马尔可夫模型（Hidden Markov Models），在现今广泛地被用在语音处理的各个领域中，是语音信号的一种统计模型。在语音识别中，可以使用 HMM 建立起相关基本序列家族的模型。而它的运用范围远远不止这些，在生命科学迅速发展的今天，许多生物信息学家已将其广泛的运用到基因和蛋白质序列的数据集搜寻和分类上，以及为人们熟知的结构分析和模式识别上。

9.1 HMM 的原理和应用

9.1.1 Markov 链

Markov 链是状态和时间参数都离散的 Markov 过程。在数学上它有如下定义：

随机序列 X_n，在任一时刻 n，它可以处在状态 $\theta_1, \ldots, \theta_N$（对于本文中的氨基酸序列，对应 20 个氨基酸，N 为 20），且它在 m+k 时刻所处的状态为 q_{m+k} 的概率，只与它在 m 时刻的状态 q_m 有关，而与 m 时刻之前它所处的状态无关。即有：

$$P(X_{m+k} = q_{m+k} / X_m = q_m, X_{m-1} = q_{m-1}, \ldots, X_1 = q_1)$$
$$= P(X_{m+k} = q_{m+k} / X_m = q_m)$$

其中，$q_1, q_2, \ldots, q_m, q_{m+k} \in (\theta_1, \theta_2, \ldots, \theta_N)$，P 为概率。

则称 X_n 为 Markov 链。

9.1.2 HMM

HMM 是从 Markov 链的基础上发展起来的，由于实际问题比 Markov 链模型所描述的更为复杂，观察到的事件并不是与状态一一对应，而是通过一组概率分布联系起来，这样的模型就称为 HMM。它是一个双重随机过程，其中一个过程描述的是 Markov 链，它描述状态的转移；另一个随机过程描述的是状态和观察值之间的统计对应关系。观察值与状态之间没有一一对应的关系，因此，只能通过观察值感知到状态的性质及其特性。

一阶的 HMM 可以看成由一个有限状态的集合 S，一个离散状态符号的集合 A，一个转移概率矩阵 $T=(t_{ji})$（表述状态的转移分布），每个状态的符号分布矩阵 $E=(e_{ix})$确定的一个时间

序列的随机模型。这样随着时间的展开，模型会产生不同的符号（T 决定状态的产生，E 决定具体是哪一个符号）。当系统处于状态 j 的时候，它下一个状态是 i 的概率就是 T_{ij}，在 j 状态产生状态 X 的概率是 E_{jx}。图 1 为一个简单 HMM 示意图，状态 1 有一个符号（对于 DNA，即四种碱基）出现的概率矢量(e_{1A}=0.25, e_{1C}=0.25, e_{1G}=0.25, e_{1T}=0.25)，状态 2 的各个符号出现的概率不完全相同（e_{1A}=0.1, e_{1C}=0.1, e_{1G}=0.1, e_{1T}=0.7），状态之间的转移概率如图所示。对于一个碱基序列 ATCCTTTTACTG，有开始状态 S 经过状态 1 和状态 2 之间的转移，到达中止状态。

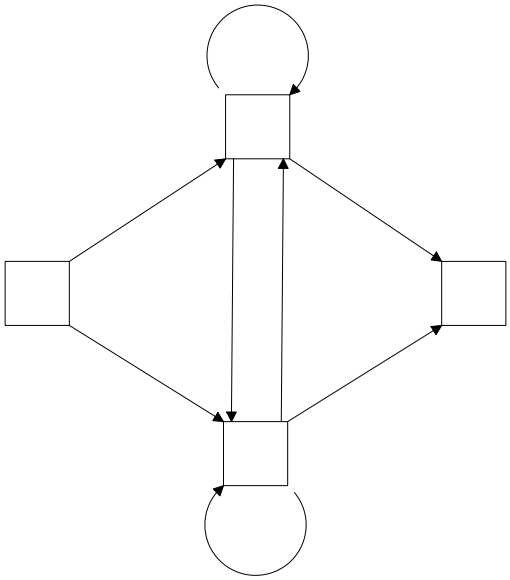


图 9.1　a simple HMM, with two states in addition to start and end states

对于我们观测到的序列，比如氨基酸序列，我们只是得到了最终的状态分布，而状态之间的转移，以及状态中各个氨基酸的分布都是隐藏的。如何得到模型的各个参数，就是 HMM 的训练问题，目前比较常用的训练算法是 Baum-Welch 算法。

如果给定了一个 HMM，对于未知的序列，我们可以计算由这个 HMM 产生该序列的概率，这个概率的大小可以反映序列是否隶属于该 HMM 对应的类。由于算法复杂度的问题，我们不能直接计算出序列产生的概率，目前采用前向算法或后向算法来简化该问题的计算。

0.5

不同长度的序列，状态数目是不同的。为此我们要在 HMM 的基础上引入插入和删除状态，模型如图 2 所示，图中的点虚线的形状对应模型的状态，所有箭头是单向的，箭头所指方向为可到达路径，而每一个状态都对应一个符号的分布，虚线箭头所指向的为各个主状态。

S

0.25

0.5

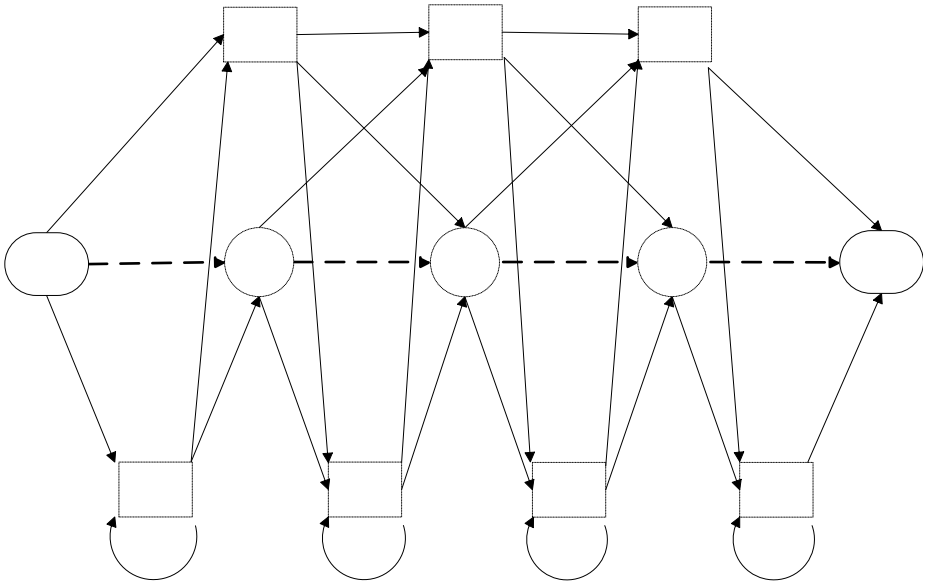


图 9.2　a HMM architecture used in this paper: S is the start state, E is the End State. And di, mi, ii denote delete, main, and insert states, respectively.

S

9.1.3 HMM 软件

HMM 在计算机上的实现已经是一个成熟的问题，可以采用了 Net-ID 公司开发的 HMMpro 2.0 软件包。HMMpro 是基于 HMM 的基本原理建立起来的，专门对生物序列进行分析的模拟器。它使用机器学习的技术，自动地建立其蛋白质和 DNA 序列的统计模型 ，它可以完成一系列的计算分子生物学的任务，包括多重比对、数据集搜寻、蛋白质分类、识别基因和识别新模式。HMMpro 可以在不同的软硬件体系下，从 PC 机到多处理机 UNIX 工作台，它都可以运行。

li

9.1.4 HMM 的应用

HMM 在国内的应用和最新进展还是主要停留在语音识别和语言处理方面。

东南大学的赵力等提出了一种新的语音识别模型 VQ-HMM。它综合了 VQ 和 HMM 的优点,在每个状态通过用矢量量化误差值取代传统 HMM 的输出概率值来建立模型,同时提出了基于分段模糊聚类算法的模型参数估计方法.文中给出了模型的实现方法,并通过实验证明了其在语音识别中具有很好的性能.，并且提出了一种新的语音识别方法,它综合了 VQ、HMM 和无教师说话人自适应算法的优点。

清华大学电子工程系的史媛媛等在隐含马尔科夫模型的状态层次上采用线性区分分析方法,将不同状态之间容易混淆的特征样本构成混淆模式类,针对混淆模式类进行线性区分分析.通过线性区分变换,在变换特征空间中仅保留那些能够有效区分该混淆类别的特征参数.这种基于状态的线性区分分析有效地提高了模型对混淆数码的区分能力.实验表明即使采用状态数很少的粗糙识别模型,也能很大幅度提高模型的识别性能;经过线性区分变换优化后的汉语数码识别模型,孤立汉语数码语音识别率可以达到 99.32%。

基于隐马尔可夫模型(HMM)的词性标注的应用研究，基于隐马尔可夫模型的语音单字识别研究等等，关于 HMM 算法本身也有很多研究：

上海交通大学电子工程系胡光锐提出了一种基于梯度的 HMM 参数重估方法，该算法基于最大似然准则,具有快速收敛和保证似然度单调增的优点；中国科学技术大学的刘鸣等为了提高话者确认系统的噪声鲁棒性,设计了一种基于子带隐 Markov 模型(HMM)和多层感知机(MLP)的话者确认系统。国防科学技术大学计算机学院陈火旺出了一种基于ＡＮＮ的ＨＭＭ的训练算法。由于ＡＮＮ和ＨＭＭ在ＭＭＳＥ训练准则上是一致的，可利用ＡＮＮ更新ＨＭＭ的观察符号输出概率分布，进而采用ＭＭＩ准则对全体ＨＭＭ的其他参数进行更新，从而在提高特定的ＨＭＭ识别能力的同时，保持不同ＨＭＭ之间的最大差异性。

在生物信息学上的应用还比较少，西安交通大学的吴晓明等采用 HMM 对 SCOP 数据库中的细胞色素 C 组进行了训练和预报，由它得到了能够代表该族特征的隐马尔可夫模型，并用该模型对一些蛋白质序列进行分析果表明，HMM 能够较好的表示同一族的蛋白质，并能够从许多蛋白质序列中识别出该族的蛋白质序列。

国外 HMMs 用于非生物信息学方面的应用也很多，我主要总结一下在生物信息学方面的进展和应用：

自从 HMMs 引入计算生物学界后（生物信息学的分支），HMM 作为一种序列比对、建模和评价工具日益得到业界的认可。Baldi 和 Brunak 定义了在计算生物学领域中，HMM 的应用很成功的三个类别：

1、 HMMs 可以用于 DNA 序列的多重比对。

蛋白质序列的比对方法，最近发展很快。用的最多的方法可能还是 BLAST。它使得我们可以在很大的蛋白质数据库重搜索同源序列，在整个蛋白质序列中搜索一个蛋白质序列时间很短。当然，也出现了一些更灵敏和准确的方法。我们的 HMM 就是一种，跟常规的方法相比，它可以发现序列久远（remote）的同源性。HMMs 也常用于数据库的搜索比较。

在此应用之前，动态规划法是序列比对的基本工具，最优化理论与算法，在蛋白质空间

结构预测和分子对接研究中有重要应用，然而不足之处就是在多重比对方面有局限性。HMMs 在多重比对方面有很好的应用。

SAM（Sequence Alignment and Modeling）是一个程序包，它可以对 DNA、RNA 或者蛋白质序列进行建模、对其可以相互区分。给定一组相关的序列，系统能够自动训练并且用一个线性的 HMMs 代表改家族。

2、训练好的 HMMs 可以揭示和发现生物学数据中的一些内在规律。一些规律已经被用于探索生物数据的特殊区域，比如编码区和非编码区，还可以用于预报哪些序列会对应某种特定的结构。

一种基于 HMM 方法来预报膜蛋白的β折叠区域的，取得了比较好的结果。它首先对 11β折叠的膜蛋白进行训练和优化 HMM 模型。结果，这种方法可以预测 172 个β折叠区域 97%正确位置。其中，有 10 个蛋白的β折叠区域大小预报完全正确。

阐明 MHC 分子与抗原氨基酸之间的相互作用对于我们理解免疫的进程又很重要的作用，一种采用 HMM，SSl 算法结合优化 HMM 结构的算法，预报人类 MHC II 分子 MLA-DRB1*001 取得了很好的效果。

3、大量数据集的分类问题。基于 HMMs 的方法已经被用于结构预测－按照碱基对应不同的结构进行分类的问题，另外 HMMs 也用于区别不同的蛋白质族，同时预报一个新的蛋白质族和子族。还有一个很成功的应用就是用于基因的发现。

Peter 等在基因调控网络中采用了一个基于蛋白质域的 HMMs 来判断一个特定的基因是否属于某一类。Ashish Bhan 等用马尔科夫模型用于整个染色体的表达时间序列的分析，利用酵母的基因芯片数据。Richard 等采用了基于 Pair HMMs 的 ab initio 预测基因结构的方法。sam 采用 profile HMM 来提高比对的准确度。Karplus 等利用 HMM 来预报蛋白质的结构。David Ussery 在分析大肠杆菌 K12 MG1655 染色体染色质相关的蛋白质的时候，采用 HMM 来预报 608IHF 在真个染色体中的分布。

9.2 应用 HMM 进行剪接位点识别

9.2.1 剪接位点识别的背景知识介绍

人类基因组序列的研究提出了对基因复杂本质探讨的课题。在此之前，科学家估计人类的基因组包含了数目极多的基因（用已表达序列聚类进行估计，人类大约有 150，000 个基

因），而果蝇则约有 14，000 个基因，简单的模式生物线虫约有 19000 个基因。从生物的复杂性来说，这很合理。但是人类基因数目的最终预报结果显示人类只有约 32，000 个基因。生物的复杂性与相应基因个数较少，这两者之间似乎产生了极大的矛盾。同时，这个现象也向人们暗示：人类表达序列（mRNA）的数目远远大于人类基因的数目。从而把矛头指向了真核生物基因复杂的本性。

直至 20 世纪 70 年代中期，分子生物学家才开始认识到真核生物的 DNA 序列由编码序列和非编码序列交错组合起来。诞生了 EXONS（expressed region）和 INTRONS(intervening sequences region)的概念。真核生物与原核生物在基因结构的复杂性方面有着本质的差异。对于原核生物来说，基因的结构相对比较简单。原核生物染色体通常只含有一个 DNA 分子，而且每个基因在 DNA 分子中只出现一个，除了为蛋白质编码的连续结构基因外，只有一小部分是调节序列和信号序列。具体来说，原核生物的基因结构仅包括启动子、起始密码子、编码区、终止密码子。对于真核生物来讲，其基因结构要复杂的多，一些基因在 DNA 中可以重复很多次。而且更为重要的是，与原核生物基因是编码 DNA 的一个完整片断不同，大多数为蛋白质编码的真核生物基因都含有“居间序列”，这些居间序列不编码蛋白质，被称作垃圾序列，但是下文我们将看到这些不编码蛋白质的序列有很重要的功能，特别是在 mRNA 前体的加工过程中，发挥着无可替代的重要作用。这些不编码的“垃圾序列”的含量和分布在不同的生物中有所不同。图 9.3 是真核生物 DNA 的结构示意图。

图 9.3 diagram of DNA Structure

真核生物基因结构特征

高等真核生物基因基本结构包括启动子、起始密码子、exon(外显子，有时也称作编码序列)，intron（内含子，有时也称为非编码序列）以及终止密码子等等。一个基因中所有的 exon 在一起被称作该基因的编码区。在图 4 中，intron 与 exon 相连的边界序列（位于 intron 序列中）被称为 3’端受体，而 exon 与 intron 相连的边界序列（位于 intron 序列中）被称作 5’端给体。在 DNA 的转录过程中，直接的产物是 pre－mRNA，它是 DNA 的直接录本，仅仅除去了启动子，包含一段 DNA 序列的全部 exon 和 intron，并且完成了加帽和加尾的修

饰。这些帽子和尾巴一直保存至 mRNA 中。

在 pre－mRNA 形成之后，开始了成熟 mRNA 的合成过程。在细胞核中除去 intron，并且在不同生物条件下选择不同的 exon 保留，按顺序连接，形成成熟 mRNA，转移出细胞核。值得一提的是 mRNA 仅仅含有 exon，并且有选择地拼接合适的 exon，而并非拼接所有的 exon。这实际上涉及到了可变剪接（Alternative Splicing）的概念。

所谓剪接即将 pre－mRNA 中的 intron 除去，将外显子顺序连接起来，产生了功能 mRNA 的过程。intron 和 exon 的边界称为剪接位点，本文的主要任务就是依据 HMM 算法及序列的局部信息，对真实剪接位点进行预报。剪接位点附近的序列是有保守性的，而且这种序列保守性在 intron 序列上表现的更加明显，这充分证明了基因中 intron（非编码序列）并非是垃圾序列。根据统计，在 DNA 中大多数的 intron 的 5’端，也就是受体位点附近，有这样的保守序列：－2AG|GTPuAGU+6，其中“|”以左表示的是属于 exon 的序列；而“|”以右表示的是属于 intron 的序列，“|”本身表示 5’端剪接位点；－2 表示剪接位点以左第二位碱基；＋6 表示剪接位点以右的第六位碱基；Pu 表示在该位上可能出现 A/G 嘌呤，而对于有下划线的位点，则说明这些位点十分保守。相应的，在 intron 的 3’端，也就是给体位点附近同样也有保守序列：－4NPyAG|PuN＋2，其中“|”以左表示的是属于 intron 的序列，而“|”以右表示的是属于 exon 的序列，“|”表是 3’端剪接位点；Py 表示 C/T 嘧啶；Pu 同样表示 A/G 嘌呤；N 表示未定的碱基。－4、－2 及有下划线的位点的含义与上述供体位点类似。我们可以很明显地发现大部分的 intron 都是以 GT 开头，以 AG 结尾的，这是一个十分重要的规律，但是并非以 GT 开头和 AG 结尾的序列都是真实 intron。以后我们在运用一些算法对这些位点进行识别的过程中，就是依据该规律选择虚假的剪接位点。从 DNA 到 蛋白质的过程，如图所示。

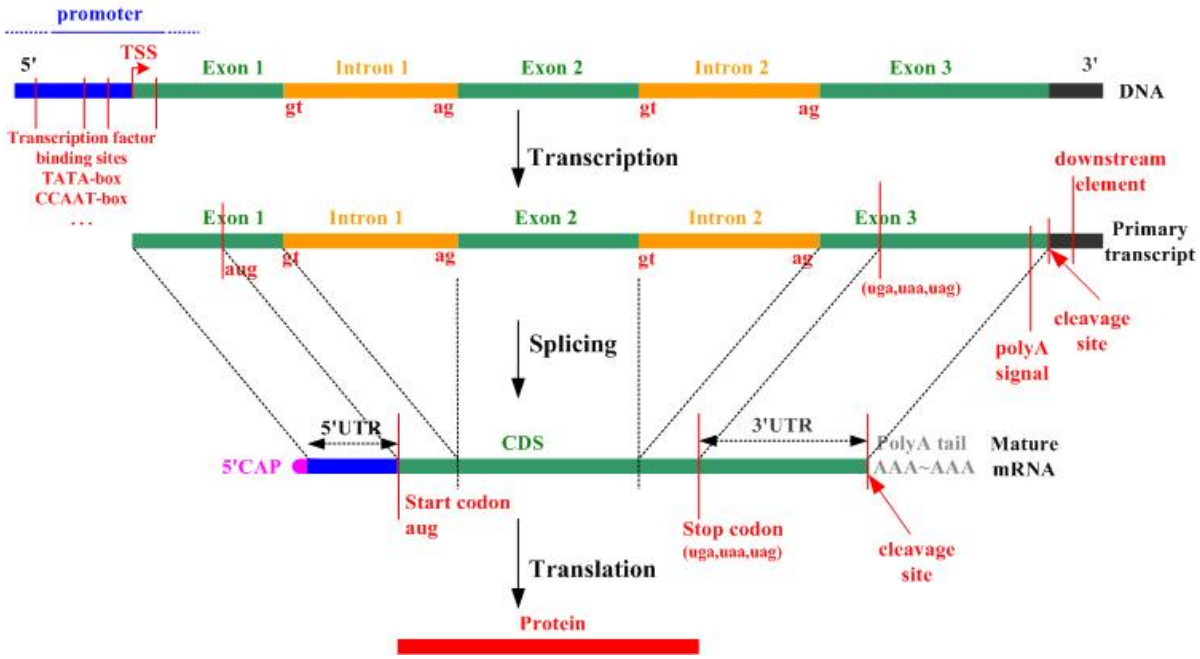


图 9.4 from DNA to Protein

9.2.2 基因识别的原理和基本方法

在人类基因组全图正式发表后，科学家十分关切一个更为复杂、更富有挑战意义、更有价值的任务——由这四个碱基构成的人类的 DNA 到底有什么有意义的信息？如何才能将这些复杂的信息全部注释出来？而这其中最为重要的莫过于标注出基因的位置，因为正是基因直接影响着丰富多彩的蛋白质世界。狭义的基因体注释的含义就是找出基因在 DNA 序列上的位置，并定义出 exon 和 intron 的界限。后半部分就是本文所要探索的问题。

我们知道愈是高等的生物，基因体就愈是复杂。因此没有一种万无一失的方法能够 100%的定义出基因。现存的方法，根据其本质我们可以将它们分为三类。

第一类的是以统计预测为基础的运算方法，它的主要特征是不需要实验资料作辅助，利用基因以及 exon-intron 结构在 DNA 序列上已知的一些特征（如起始密码子、终止密码子、基因组 DNA 中的外显子、内含子和剪接位点的保守性），在 DNA 序列上直接预测基因的位置。这类方法的缺点是：对过长或过短的外显子、内含子的预测准确性不高，且容易高估基因的数目，同时高估的程度又与所要预测的 DNA 的区域和物种息息相关。它的优点是当我们所要探讨的物种的实验资料非常缺乏的时候，可以采用这类方法，而且一般来说这类方法由于不需要大规模的库比对，因此速度比较快。这类方法根据其依据的理论不同又可粗分为 5 小类：1. 以隐马尔可夫链型为基础的算法,包括 GeneMark.hmm，GENSCAN，Genie, HMMgene，以及 Veil 等； 2. 以类神经网络为基础的算法，这类方法包括：Grail II 以及

GrailEXP_Perceval 等；3. 以决策树为基础的算法，这类方法主要包括：MZEF 以及 MZEF-SPC 等；4. 以结合多种预测方法而成的算法，这类方法主要有：FGENSH 等。其它还有一些方法，比较著名的就是 GeneID，GeneView 等算法。

第二类方法需要实验上的资料辅助，用 DNA 序列与各类实验得到的数据库，如 EST（表现的序列片断（Express Sequence Tags））、cDNA（互补 DNA）、蛋白质资料库等进行比对，得到可能的基因所在。虽然这种方法的正确性比较高，但是由于它进行的是序列比对，因此，耗费的计算时间和存储空间比较大，且易受到已知序列库品质的影响。该方法依据所使用比对方法的不同又可以分为区域性比对和基于模式的比对方法。

第三类方法是结合上述两类方法的优势，对未知基因进行预测。它有前提条件，即用来比对的资料库中已存在所要比对的蛋白质序列。这类方法的准确性比较高。其步骤是首先到蛋白质资料库中，利用 BLAST 比对找到适当的候选蛋白质序列，然后以统计预测为基础的第一类方法预测基因结构。缺点是用这类方法找到未知基因的可能性偏低，且操作起来不是很方便。

9.2.3 HMM 用于剪切点识别研究

1. 实验数据集来源和数据特点以及数据的预处理

由 Burset 和 Guigo 搜集的 DNA 序列集是一个比较经典的数据集合（http://www1.imim.es/databases/genomics96/index.html），被广泛用来对一些主要的基因识别算法进行比较。

这个集合中包含了 GenBank 脊椎动物版本的 DNA 序列，这些序列至少包含一个编码蛋白质的基因。在筛选的过程中首先除去了至少编码一个不完整蛋白质产物的序列、编码蛋白质的位置没有完全精确确定的序列、在互补链上编码蛋白质的序列、在别的数据库条目中定义了蛋白质编码基因的序列和包含伪基因的序列。然后得到了一个含有 1410 条脊椎动物 DNA 序列的集合，这些序列中每一个序列编码一个并且仅一个可剪接的功能蛋白产物。为了加强序列的完整性，又在这些序列中除去了蛋白质编码区不是由 ATG 这个起始密码子开头或者没有由结束密码子结束的序列、编码蛋白质的片断不是三的倍数的序列、给体位点不是由 GT 开始的序列或受体位点不是由 AG 结尾的序列或在编码蛋白质片断的结构框中含有终止密码子的序列，最后加上仅收录发布在 GenBank rel. 74,January 1993 之后的序列的条件，得到了 570 条 DNA 序列，这些序列含有 2，892，149 个碱基和 2649 个 exon。

综合来说，ALLSEQ 数据集有两个特点。第一，这些序列都比较短，且含有一个结构简单的完整基因，有很高的编码密度，没有序列错误；第二，这些序列在建立之初就考虑到使数据集有较小的交盖率，因此，只有在 1993 年 1 月之前收入进公共数据库的序列条目才被采用。所以，当我们用 ALLSEQ 集对程序和算法进行检验的时候，得到的准确度水平应该高于用真实数据集对程序和算法进行检验得到的准确度水平。因为真实序列可能只有比较低的编码密度、编码多重基因、基因的结构更为复杂，且序列出现错误的几率可能比较高一些。

由 ALLSEQ 得到的数据集一共由三个文本文件，一个是以 fasta 形式存贮的 570 条 DNA 的序列（DNASequences_fasta.txt），一个是以表格形式存贮的关于这 570 条序列编码区（exon）的起始和结束位点数据（CDS_tbl.txt），还有一个是以 fasta 形式存贮的由这 570 条 DNA 序列编码的蛋白质序列（AASequences_fasta.txt）。在本文中，所用到的数据就是前两个数据集。AASequences_fasta.txt 的内容如下所示

//Gene ID Codon Start and Stop Site

ACU08131 521 641 1066 1362 1860 2028 2637 2802 3558 3797 4131 4247

AGGGLINE 3066 3157 3281 3503 4393 4521

AGU04852 3951 3955 5434 5516 5810 5911 6665 6739 7637 8415

Fasta 格式的首行是对序列的描述，然后几行是数据序列。通常用“>”将作为描述行的首行与序列行相区别。通常一行数据长度不超过 80 字符。序列都是用标准的 IUB/IUPAC 的氨基酸和核苷酸来编码。本文所用的 DNA 序列中，除了 A，T，G，C 这四个编码，我们还可以看到一些特殊字符如：N（表示 A，T，G，C 中的任何一个），R（表示嘌呤），Y（表示嘧啶），S（表示含高 G、C 的位点）等等。

为了适合对数据的后续处理，本文编制一系列文本处理程序对原始数据的存贮方式进行了修改，同时将 570 条 DNA 的序列数据和位点数据导入关系数据库中，鉴于我们所要考虑剪接位点附近的序列数据，我们除去了每个基因序列的起始位点（以 ATG 为起始密码子）和每条 DNA 序列的终止位点（一些序列以 TAA 为终止密码子），最后得到 2079 个受体位点和 2079 个给体位点的序列数据，等待进一步按 DNA 的局部序列信息对整体集合进行分类，从而得到相应的训练集和测试集。

2. HMM 的识别键切点的总体思路

1) 序列片断的选择以及窗口的确定

跟其他的识别算法一样，首先是选择合适序列的片断。DNA 的剪切点模型如下图所示，

我们的目标是识别处给体位点和受体位点。在一条基因的 DNA 序列中，并不是所有的碱基都编码蛋白质。图中灰色的区域是最终编码的部分，DNA 在最后通过剪切后，把白色的非编码区域拿掉。这样我们得到一条基因的 DNA 序列后，必须确定了剪切点之后，才能够预测出最终的编码蛋白质。

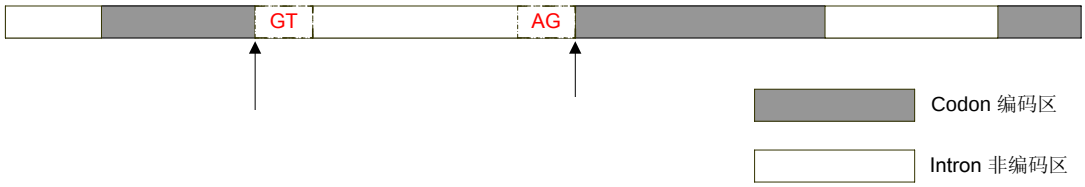


图 9.5　AT and AG rule

这里的剪切点其实包含两个位点，即给体位点和受体位点。这里所有真实的给体位点右侧都有两个相同的碱基 GT ，与之相对应的受体位点左侧都有两个相同的碱基 AG ，这个规则我们称之为 AGGT 规则。我们可以利用这个规则来对位置的基因序列预报它们的剪切点。在具体选择序列片断预报的时候，按照这个规则，我们选择合适的大小的窗口，在所有的 GT 或 AG 位点附件选择一些序列，这些序列可能是真实的剪切位点，当然大部分都是虚假位点。对一个未知的候选序列片断进行预报之前，有一个模型的训练问题。就是本文中的建模问题。

2) HMM 建模

按照前面的 AGGT 规则，加上选择合适的窗口之后，我们可以得到两类序列的数据，一类是真实位点的序列，另一类是虚假的序列。按照 HMM 的算法思想，对一类序列进行训练之后得到的模型，它可以对未知的序列作评价也就是打分，分值得大小反应了未知序列有该模型产生的概率。这样，从理论上将，用真实的序列训练好的 HMM 模型，分别对两类序列进行评价，两类序列的分值应当有较大的差异。当然差异大小反映了模型的预报能力。

3) 剪切位点预报

训练好的模型可以预测未知序列的类别。预报的时候有两种方式，训练单模型和训练双模型。所谓训练单模型，即在建模阶段利用真实剪切点的序列或者虚假位点的序列建模，后面预报未知的两类序列，类别的判断通过选择合适的得分阈值，得分小于和大于阈值的序列分别判断为两类。使用单模型训练的时候有两种选择，一种是采用真实的序列建模或虚假的序列建模。一般来说采用真实的序列建模，预测效果要高于采用虚假的序列建模。而训练双模型，是指在建模阶段，同时训练两个模型。对于未知的序列，通过两个模型的不同评价，其分值的趋向决定其类别信息。

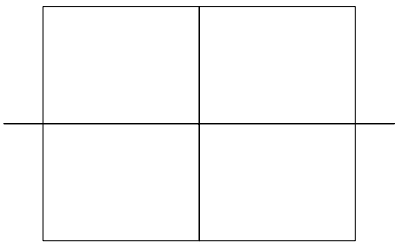
4) 结果的评价

国际上有一些通用的用来评价一个算法或是程序对基因结构预测的效果。本文中主要采用 Sn^{true} 和 Sn^{false}，TP，FP，TN，FN 对实验结果进行评价。TP 是 True Positive 的简写，表示预报为真的，事实上为真的位点；FP 是 False Positive 的简写，表示预报为真的，事实上为假的位点；TN 是 True Negative 的简写，表示预报为假的，事实上为假的位点；FN 是 False Negative 的简写，表示预报为假的，事实上为真的位点。从上述定义可知，TP 和 FN 表示的是所有真实剪接位点的序列集合，而 FP 和 TN 表示的是所有虚假位点的序列集合, 如图所示。

然后就可以比较方便的得到 Sn^{true} 和 Sn^{false} 的定义。Sn 是 Sensitivity（敏感度）的简写：

$$Sn^{true} = \frac{TP}{TP + FN}, \quad Sn^{false} = \frac{TN}{TN + FP}$$

由定义可知，Sn^{true} 表示为所有真实位点预报为真实位点的比率，即真实位点识别率；而 Sn^{false} 表示所有虚假位点中预报为虚假位点的比率，即错误位点识别率。



两类剪切点（ two class ）

+ -

附录 1 Linux 使用简介

Linux 常用命令

& &命令可用在其他任何命令的后面，它用来通知计算机在后台运行某一命令。通过把作业放在后台，用户可以继续使用当前的 shell 来处理其他命令；如果命令在前台运行的话，那么用户在此进程结束前不能继续使用当前的 shell。

adduser

adduser 命令由 root 或其他具有权限的管理员用来创建新用户，跟在 adduser 命令后面的是所要创建的帐号名，例如：adduser flying

alias

alias 命令用来设置命令的别名或替代名。一般说来别名往往是实际命令名的缩写。例如用户为 ls 设置一个别名 dir：

alias dir=ls

若仅输入 alias 本身时，系统将显示当前所有的别名。

bg

bg 命令用来迫使被挂起的进程在后台运行。例如，当你已经在前台启动了一个命令时（没有在此命令后使用&），你才想到这一命令将运行较长一段时间，但你这时还需使用 shell。在这种情况下，可通过 ctrl+z 挂起当前运行的进程。此时你既可以使它长期挂起，也可以通过输入 bg 把这一进程放到后台运行。这样 shell 就可以用来执行其他的命令了。

cat

cat 通常是用来在屏幕上滚动显示文件的内容。它的格式是：

cat〈filename〉

cd

cd 用来改变目录。这一命令非常有用，它有三种典型的使用方法。

cd 移到目录树的上一层

cd~移动到用户的主目录，与单独使用 cd 相同

cd directory name 改变到指定的目录

cp

cp 用来拷贝对象。例如要把 file1 拷贝到 file2，用如下命令：

cp file1 file2

dd

dd 命令用来转换文件格式。

fg

fg 命令用来激活某个被挂起的进程并使它在前台运行。当有一个进程正在运行时，由于某种原因需要挂起它，在执行完其他任务后，需要重新把这一进程调到前台运行，这时便可用 bg 命令使这一进程继续运行。

find

find 命令用来查找指定目录的文件。当找到后将按照用户的要求对文件进行处理。语法是：find 以它为起点进行搜索的目录想要查找的文件名或元字符对文件执行的操作

grep

grep 命令用来在指定的对象中搜索指定的文本。语法是：grep〈text〉〈file〉。它还可以和其他命令的结果联合使用，例如：

ps -ef|grep-v root

这一命令要求 grep 接受 ps 命令的输出，并除去所有包含单词 root 的进程（-v 的含义是显示与文本不匹配的内容）。在不使用-v 选项时，这一命令将显示进程清单中所有包含单词 root 的进程。

halt

halt 命令用来通知内核关闭系统，它是一个只能由超级用户执行的命令。

hostname

既可以用来显示系统当前的主机名或域名，也可用来设置系统的主机名。

login

当向系统注册时，将使用 login。login 命令也可用来随时从这一用户改变到另一用户。

logout

logout 命令用来使当前用户从系统中注销。如果这是你使用的注册到系统的唯一用户，那么将退出系统。

ls

ls 命令用来列出目录的内容，它的输出格式可通过选项来控制。没有任何选项的 ls 命令将按照字母顺序列出所有非隐藏文件，显示的列数以正好适合窗口的大小为准。最常用的一组

选项是-la。这里 a 表示要列出所有的文件；l 表示以长格式列出，它使输出结果成为一个详细的长列表。

more

more 是一个过滤程序，它可以每次一屏地翻阅文本文件的内容。这一命令只能对文件向下进行翻页。

mount

mount 命令用来把特殊文件（通常是设备名）指定的文件系统安装在作为一个参数被指定的目录上。只有超级用户能够安装文件。如果在运行 mount 命令时不使用任何参数，它将列出当前被安装的所有文件系统。

mv

mv 命令用来把某个对象从这一位置移动到另一位置。如果最后一个变量指定的是一个现有的目录，那么这一命令将把命令行中指定的所有文件移到这一目录中；如果给出的是两个文件，mv 将把第一个文件移到第二个文件中。只有当最后一个变量是一个目录时，mv 命令的变量才可以超过两个。

ps

ps 用来报告进程的状态，它将显示一张当前进程的快照。

rm

rm 用来删除指定的文件。利用-r 选项，rm 将递归地删除文件（warning：危险）。rm 可与 find 命令联合使用，查找某一文件并删除它。在缺省的情况下，rm 命令不能删除目录。

rmdir

rmdir 用来删除指定的空目录。语法是：

rm〈directory name〉

umount

umount 命令用来卸载文件。语法是：

umount〈filename〉

unalias

unalias 用来取消别名。如前面曾把 dir 设置为 ls 命令的别名，要取消它只须输入 unalias dir 即可。

unzip

unzip 用来列出、检测或从某个 zip 文档中抽取文件，它的缺省用法是从文档中抽取文件。其基本语法是：

unzip〈filename〉

who

who 命令用来显示当前注册到系统的每个用户的注册名、终端类型、注册时间和远程主机名。如果有两个非选项的参数传递给 who 命令，那么它将打印出正在运行此命令的用户的信息。如果想要查看某一对话所持续的时间，可利用-u 选项。

xset

xset 命令用来设置 X Window 环境中的一些选项，可利用它来设置响铃（xser b〈volume〉〈frequency〉〈duration in milliseconds〉）、鼠标速度（xset m〈acceleration〉〈threshold〉），以及其它参数。

zip

zip 命令用来列出、检测或向某个 zip 文档中添加文件，它的缺省用法就是向某个文档中添加文件。

VI-从入门到精通

正式开始

vi 的模式

vi 是在很久以前就写的程序。在那个时候，键盘上没有现在熟悉的那么多功能键。所以，vi 设计成通过输入字符和 ESC 来控制输入和修改文本。

可能这对于一些人来说是个遗憾，但是你会发现你不需要改变手在键盘上的位置就可以完成所有的功能。结果是你的输入将快速起来。

为了完成交互的全屏幕编辑工作，vi 有三种模式。插入模式(insert)用于输入文本。在插入模式下，你输入的任何字符都将显示在屏幕并存于文件。命令(command)模式用于大多数编辑功能。在命令模式下，所有的输入都将产生一定的响应而不是直接到文本中，例如移动光标、删除一块文本、拷贝文本等。第三个模式是执行另外的功能，例如查找、全局替换、处理多个文件等。这种模式是基于 ex 编辑器的。

启动 vi

当 vi 启动后，默认的模式是命令模式。按照下面的步骤试一试：输入程序名称启动 vi：

$vi

你将看到类似下面的东西：

~
~
~
~
~
~
Empty buffer

i--插入

现在我们输入"i"进入插入模式。字符"i"将不会回显。此后你输入的任何东西都将显示在缓存中。现在我们来输入一段文字。例子中的话来自英文版的孙子兵法。注意光标的位置在例子中是个下划线。

If wise, a commander is able to recognize changing circumstances and to
act expediently. If sincere, his men will have no doubt of the certainty of
rewards and punishments. If humane, he loves mankind, sympathizes with others,
and appreciates their industry and toil. If courageous, he gains victory by
seizing opportunity without hesitation. If strict, his troops are disciplined
because they are in awe of him and are afraid of punishment.
Shen Pao-hsu ... said: 'If a general is not courageous he will be unable
to conquer doubts or to create great plans.'
~
~
~

Esc--Cancel

当你输入完了，按下 Esc 键返回到命令模式。(如果你已经处于命令模式下，按 Esc 时会听到喇叭的声音。)Esc 可以撤消未完成的命令和终止插入模式。按 Esc 后，光标将停留在你最后输入的字符的下面。

很不幸，没有一个明显的标志表明你现在处于什么模式下。但是有简单的方法来告诉你现在你所处的模式。如果你按下键，相应的字符出现在屏幕上，那么你是在插入模式下，否则是命令模式下。如果你不能确定你现在所处的模式，那么按 Esc 两次以听到喇叭声来确信你在命令模式下。

移动光标和简单的编辑

是看一看基本的移动光标命令的时候了。训练训练你的手指，让他们以后自动的执行你所想的命令吧。

最重要的移动命令

让我们来看看有多少影响光标移动的命令。

hh--光标左移

首先，按 5 下 h 让光标左移动 5 个(如果你看到 h 跑到屏幕上了的话，一定是你忘记了按 Esc)。光标现在应该在"plans"中的"p"下面(看下面的例子)：

```
If wise, a commander is able to recognize changing circumstances and to
act expediently. If sincere, his men will have no doubt of the certainty of
rewards and punishments. If humane, he loves mankind, sympathizes with others,
and appreciates their industry and toil. If courageous, he gains victory by
seizing opportunity without hesitation. If strict, his troops are disciplined
because they are in awe of him and are afraid of punishment.
Shen Pao-hsu ... said: 'If a general is not courageous he will be unable
to conquer doubts or to create great plans.'
~
~
~
~
```

kk--光标上移

现在我们来按 5 次 k 让光标上移 5 行。也许你该认为应该有快捷方式了。好，现在就有个简单的方法：在你要采取的行动前加上数字。按下 5k 你可以和按 5 下 k 有相同的结果了。光标现在该在"he"的"e"下了。

```
If wise, a commander is able to recognize changing circumstances and to
act expediently. If sincere, his men will have no doubt of the certainty of
rewards and punishments. If humane, he loves mankind, sympathizes with others,
and appreciates their industry and toil. If courageous, he gains victory by
seizing opportunity without hesitation. If strict, his troops are disciplined
because they are in awe of him and are afraid of punishment.
Shen Pao-hsu ... said: 'If a general is not courageous he will be unable
to conquer doubts or to create great plans.'
~
~
~
~
~
~
~
~
```

在使用这些功能的时候有些限制的。例如使用 h 或 l 移动光标超过了一行文字的头或尾，光标将停留在头或尾部，喇叭鸣叫提醒你。

还有其他的类似 h 和 k 的吗？看一看表 1。最好的熟悉他们的的方法是多用他们。

表 1. 常见的方向键

命令移动

h 左一个字符

j 下一行

k 上一行

l 右一个字符

w,W 前一个单词(W 忽略标点)

b,B 后一个单词(B 忽略标点)

$到行尾

^到行首第一个非空字符

0 行首

G 到缓冲首

nG 到第 n 行

大小写的命令是有一些区别的。小写字母一般以标点区分"words."，而大写则忽略他们。

最重要的编辑程序

让我们看看最简单也是最常用的编辑过程：

修改没有人不犯错误。所以迟早你会碰到修改你输入的文本的时候。实际上花在字处理上的大多数时间是修改而不是输入新的东西。因此，你要知道如何方便地修改就很重要。

x--删除一个字符删除文字的最简单的方式是用 x。这个命令的结果是光标所处的字符的消失，后面的文字左移动。如果你删除的字符是一行最后的一个字符，那么光标将向左移动一个，这样光标就不会停留在不存在的字符的下面了。假如没有任何文字了，喇叭就叫。

d--删除对象这个命令的右边还要有一定的文字对象。文字对象就是一块文字。他右边接的就是在控制光标移动的那些字符。例如 w 表示向前一个单词，那么 dw 将删除下一个单词。5w 表示前进 5 个单词，那么 d5w 将删除他们。

dd--删除一行最常用的 d 系列命令之一。和前面一样，5dd 将删除 5 行

D--整个删除大写形式的 D 用来删除从光标到行尾。和 d$一样效果。

u--恢复要后悔吗？他不仅仅撤消删除，还撤消你所有的编辑工作。

.--重复重复编辑工作。

补充一些关于 set 命令的东西吧。

:set sw （使用 ai 时定义向后制表符的空格数）

:set ai （在插入模式，保持缩进，与 sw 一起使用）

:set aw 或 noaw（auto write）

:set nu 或 nonu（number/nonumber）

:set sm 或 nosm（配合小括号或大括号）

:set showmode 或 noshowmode

GNU make 指南

0) 介绍

本文将首先介绍为什么要将你的C源代码分离成几个合理的独立档案，什么时 候需要分，怎么才能分的好。然后将会告诉你 GNU Make 怎样使你的编译和连 接步骤自动化。对于其它 Make 工具的用户来说，虽然在用其它类似工具时要 做适当的调整，本文的内容仍然是非常有用的。如果对你自己的编程工具有怀 疑，可以实际的试一试，但请先阅读用户手册。

1) 多文件项目

1.1 为什么使用它们?

首先，多文件项目的好处在那里呢？

它们看起来把事情弄的复杂无比。又要 header 文件，又要 extern 声明，而且如果需要查找一个文件，你要在更多的文件里搜索。

但其实我们有很有力的理由支持我们把一个项目分解成小块。当你改 动一行代码，编译器需要全部重新编译来生成一个新的可执行文件。 但如果你的项目是分开在几个小文件里，当你改动其中一个文件的时 候，别的源文件的目标文件(object files)已经存在，所以没有什么 原因去重新编译它们。你所需要做的只是重现编译被改动过的那个文 件，然后重新连接所有的目标文件罢了。在大型的项目中，这意味着 从很长的（几分钟到几小时）重新编译缩短为十几，二十几秒的简单 调整。

只要通过基本的规划，将一个项目分解成多个小文件可使你更加容易 的找到一段代码。很简单，你根据代码的作用把你的代码分解到不同 的文件里。当你要看一段代码时，你可以准确的知道在那个文件中去 寻找它。

从很多目标文件生成一个程序包 (Library)比从一个单一的大目标文件 生成要好的多。当然实际上这是否真是一个优势则是由你所用的系统 来决定的。但是当使用 gcc/ld (一个 GNU C 编译 / 连接器) 把一个程 序包连接到一个程序时，在连接的过程中，它会尝试不去连接没有使 用到的部分。但它每次只能从程序包中把一个完整的目标文件排除在 外。因此如果你参考一个程序包中某一个目标档中任何一个符号的话， 那么这个目标文件整个都会被连接进来。要是一个程序包被非常充分 的分解了的话，那么经连接后，得到的可执行文件会比从一个大目标 文件组成的程序包连接得到的文件小得多。

又因为你的程序是很模块化的，文件之间的共享部分被减到最少，那 就有很多好处——可以很容易的追踪到臭虫，这些模块经常是可以用 在其它的项目里的，同时别人也可以更容易的理解你的一段代码是干 什么的。当然此外还有许多别的好处……

1.2 何时分解你的项目

很明显，把任何东西都分解是不合理的。象“世界，你们好”这样的 简单程序根本就不能分，因为实在也没什么可分的。把用于测试用的 小程序分解也是没什么意思的。但一般来说，当分解项目有助于布局、 发展和易读性的时候，我都会采取它。在大多数的情况下，这都是适 用的。（所谓“世界，你们好”，既 'hello world' ，只是一个介 绍一种编程语言时惯用的范例程序，它会在屏幕上显示一行 'hello world' 。是最简单的程序。）

如果你需要开发一个相当大的项目，在开始前，应该考虑一下你将 如何实现它，并且生成几个文件（用适当的名字）来放你的代码。 当然，在你的项目开发的过程中，你可以建立新的文件，但如果你 这么做的话，说明你可能改变了当初的想法，你应该想想是否需要 对整体结构也进行相应的调整。

对于中型的项目，你当然也可以采用上述技巧，但你也可以就那么开 始输入你的代码，当你的码多到难以管理的时候再把它们分解成不同 的档案。但以我的经验来说，开始时在脑子里形成一个大概的方案，并且尽量遵从它，或在开发过程中，随着程序的需要而修改，会使开发变得更加容易。

1.3 怎样分解项目

一些个人观点，供参考：

i) 不要用一个 header 文件指向多个源码文件（例外：程序包的 header 文件）。用一个 header 定义一个源码文件的方式 会更有效，也更容易查寻。否则改变一个源文件的结构（并且 它的 header 文件）就必须重新编译好几个文件。

ii) 如果可以的话，完全可以用超过一个的 header 文件来指向同 一个源码文件。有时

将不可公开调用的函数原型，类型定义 等等，从它们的C源码文件中分离出来是非常有用的。使用一 个 header 文件装公开符号，用另一个装私人符号意味着如果 你改变了这个源码文件的内部结构，你可以只是重新编译它而 不需要重新编译那些使用它的公开 header 文件的其它的源文件。

iii) 不要在多个 header 文件中重复定义信息。如果需要，在其中一个 header 文件里 #include 另一个，但 是不要重复输入相同的 header 信息两次。原因是如果你以后改变了这个信息，你只需要把它改变一次，不用搜索并改变另外一个重复的信息。

iv) 在每一个源码文件里，#include 那些声明了源码文件中的符 号的所有 header 文件。这样一来，你在源码文件和 header 文件对某些函数做出的矛盾声明可以比较容易的被编译器发现。

1.4 对于常见错误的注释

a) 定义符(Identifier)在源码文件中的矛盾：在C里，变量和函数的缺 省状态是公用的。因此，任何C源码档案都可以引用存在于其它源码档中的通用(global)函数和通用变量，既使这个档案没有那个变量或函数的声明或原型。因此你必须保证在不同的两个档案里不能用同一个符号名称，否则会有连接错误或者在编译时会有警告。

一种避免这种错误的方法是在公用的符号前加上跟其所在源文件有关的前缀。比如：所有在 gfx.c 里的函数都加上前缀“gfx_”。如果你很小心的分解你的程序，使用有意义的函数名称，并且不是过分使用通用变量，当然这根本就不是问题。

要防止一个符号在它被定义的源文件以外被看到，可在它的定义前加上关键字“static”。这对只在一个档案内部使用，其它档案都 都不会用到的简单函数是很有用的。

b) 多次定义的符号：header 档会被逐字的替换到你源文件里#include 的位置的。因此，如果 header 档被#include 到一个以上的源文件里，这个 header 档中所有的定义就会出现在每一个有关的源码文件里。这会使它们里的符号被定义一次以上，从而出现连接错误（见上）。

解决方法：不要在 header 档里定义变量。你只需要在 header 档里声明它们然后在适当的C源码文件（应该#include 那个 header 档的那个）里定义它们（一次）。对于初学者来说，定义和声明是很容易混淆的。声明的作用是告诉编译器其所声明的符号应该存在，并且要有所指定的类型。但是，它并不会使编译器分配贮存空间。而定义的做用是要求编译器分配贮存空间。当做一个声明而不是做 定义的时候，在声明前放一个关键字“extern”。

例如，我们有一个叫“counter”的变量，如果想让它成为公用的，我们在一个源码程序（只在一个里面）的开始定义它：“int counter;”，再在相关的 header 档里声明它：“extern int

counter;”。

函数原型里隐含着 extern 的意思，所以不需顾虑这个问题。

c) 重复定义，重复声明，矛盾类型：

请考虑如果在一个 C 源码文件中 #include 两个档 a.h 和 b.h，而 a.h 又#include 了 b.h 档（原因是 b.h 档定义了一些 a.h 需要的类型），会发生什么事呢？这时该 C 源码文件#include 了 b.h 两次。因此每一个在 b.h 中的#define 都发生了两次，每一个声明发生了两次，等等。理论上，因为它们是完全一样的拷贝，所以应该不会有什么问题，但在实际应用上，这是不符合 C 的语法的，可能在编译时出现错误，或至少是警告。

解决的方法是要确定每一个 header 档在任一个源码文件中只被包含了一次。我们一般是用预处理器来达到这个目的的。当我们进入每一个 header 档时，我们为这个 header 档 #define 一个巨集指令。只有在这个巨集指令没有被定义的前提下，我们才真正使用该 header 档的主体。在实际应用上，我们只要简单的把下面一段码放在每一个 header 档的开始部分：

#ifndef FILENAME_H

#define FILENAME_H

然后把下面一行码放在最后：

#endif

用 header 档的档名（大写的）代替上面的 FILENAME_H，用底线代替档名中的点。有些人喜欢在#endif 加上注释来提醒他们这个#endif 指的是什么。例如：

#endif /* #ifndef FILENAME_H */

我个人没有这个习惯，因为这其实是很明显的。当然这只是各人的 风格不同，无伤大雅。

你只需要在那些有编译错误的 header 档中加入这个技巧，但在所有的 header 档中都加入也没什么损失，到底这是个好习惯。

1.5 重新编译一个多文件项目

清楚的区别编译和连接是很重要的。编译器使用源码文件来产生某种 形式的目标文件 (object files)。在这个过程中，外部的符号参考并 没有被解释或替换。然后我们使用连接器来连接这些目标文件和一些 标准的程序包再加你指定的程序包，最后连接生成一个可执行程序。 在这个阶段，一个目标文件中对别的文件中的符号的参考被解释，并 报告不能被解释的参考，一般是以错误信息的形式报告出来。

基本的步骤就应该是，把你的源码文件一个一个的编译成目标文件的格式，最后把所有

的目标文件加上需要的程序包连接成一个可执行文件。具体怎么做是由你的编译器决定的。这里我只给出 gcc（GNU C 编译器）的有关命令，这些有可能对你的非 gcc 编译器也适用。

gcc 是一个多目标的工具。它在需要的时候呼叫其它的元件（预处理程序，编译器，组合程序，连接器）。具体的哪些元件被呼叫取决于输入文件的类型和你传递给它的开关。

一般来说，如果你只给它 C 源码文件，它将预处理，编译，组合所有的文件，然后把所得的目标文件连接成一个可执行文件（一般生成的 文件被命名为 a.out ）。你当然可以这么做，但这会破坏很多我们把一个项目分解成多个文件所得到的好处。

如果你给它一个-c 开关，gcc 只把给它的文件编译成目标文件，用源码文件的文件名命名但把其后缀由“.c”或“.cc”变成“.o”。 如果你给它的是一列目标文件，gcc 会把它们连接成可执行文件，缺省文件名是 a.out 。你可以改变缺省名，用开关-o 后跟你指定的文件名。

因此，当你改变了一个源码文件后，你需要重新编译它：'gcc -c filename.c'然后重新连接你的项目：'gcc -o exec_filename *.o'。如果你改变了一个 header 档，你需要重新编译所有#include 过这个档的源码文件，你可以用'gcc -c file1.c file2.c file3.c'然后象上边一样连接。

当然这么做是很繁琐的，幸亏我们有些工具使这个步骤变得简单。本文的第二部分就是介绍其中的一件工具：GNU Make 工具。

2) GNU Make 工具

2.1 基本 makefile 结构

GNU Make 的主要工作是读进一个文本文件，makefile。这个文件里主要是有关哪些文件（‘target’目的文件）是从哪些别的文件（‘dependencies’依靠文件）中产生的，用什么命令来进行这个产生过程。有了这些信息，make 会检查磁碟上的文件，如果目的文件的时间戳（该文件生成或被改动时的时间）比至少它的一 个依靠文件旧的话，make 就执行相应的命令，以便更新目的文件。（目的文件不一定是最后的可执行档，它可以是任何一个文件。）

makefile 一般被叫做“makefile”或“Makefile”。当然你可以在 make 的命令行指定别的文件名。如果你不特别指定，它会寻找“makefile”或“Makefile”，因此使用这两个名字是最简单的。

一个 makefile 主要含有一系列的规则，如下：

: ...

(tab)

(tab)

.

.

.

例如，考虑以下的 makefile：

```
=== makefile 开始 ===
myprog : foo.o bar.o
   gcc foo.o bar.o -o myprog
foo.o : foo.c foo.h bar.h
   gcc -c foo.c -o foo.o
bar.o : bar.c bar.h
   gcc -c bar.c -o bar.o
=== makefile 结束 ===
```

这是一个非常基本的 makefile——make 从最上面开始，把上面第一个目的，‘myprog’，做为它的主要目标（一个它需要保 证其总是最新的最终目标）。给出的规则说明只要文件‘myprog’ 比文件‘foo.o’或‘bar.o’中的任何一个旧，下一行的命令将会被执行。

但是，在检查文件 foo.o 和 bar.o 的时间戳之前，它会往下查找那些把 foo.o 或 bar.o 做为目标文件的规则。它找到的关于 foo.o 的规则，该文件的依靠文件是 foo.c, foo.h 和 bar.h。它从下面再找不到生成这些依靠文件的规则，它就开始检查磁碟上这些依靠文件的时间戳。如果这些文件中任何一个的时间戳比 foo.o 的新，命令'gcc -o foo.o foo.c'将会执行，从而更新文件 foo.o。

接下来对文件 bar.o 做类似的检查，依靠文件在这里是文件 bar.c 和 bar.h。

现在，make 回到‘myprog’的规则。如果刚才两个规则中的任何一个被执行，myprog 就需要重建（因为其中一个.o 档就会比‘myprog’新），因此连接命令将被执行。

希望到此，你可以看出使用 make 工具来建立程序的好处——前一章中所有繁琐的检查步骤都由 make 替你做了：检查时间戳。你的源码文件里一个简单改变都会造成那个文件被重新编译（因为.o 文件依靠.c 文件），进而可执行文件被重新连接（因为.o 文件被改变了）。其实真正的得益是在当你改变一个 header 档的时候——你不再需要记住那个源码文件依靠它，因为所有的资料都在 makefile 里。make 会很轻松的替你重新编译所有那些因依靠这个 header 文件而改变了的源码文件，如有需要，再进行重新连接。

当然，你要确定你在 makefile 中所写的规则是正确无误的，只列出那些在源码文件中被#include 的 header 档……

2.2 编写 make 规则(Rules)

最明显的（也是最简单的）编写规则的方法是一个一个的查看源码文件，把它们的目标文件做为目的，而 C 源码文件和被它#include 的 header 档做为依靠文件。但是你也要把其它被这些 header 档#include 的 header 档也列为依靠文件，还有那些被包括的文件所包括的文件……然后你会发现要对越来越多的文件进行管理，然后你的头发开始脱落，你的脾气开始变坏，你的脸色变成菜色，你走在路上开始跟电线杆子碰撞，终于你捣毁你的电脑显示器，停止编程。到低有没有些容易点儿的方法呢？

当然有！向编译器要！在编译每一个源码文件的时候，它实在应该知道应该包括什么样的 header 档。使用 gcc 的时候，用-M 开关，它会为每一个你给它的 C 文件输出一个规则，把目标文件做为目的，而这个 C 文件和所有应该被#include 的 header 文件将做为依靠文件。注意这个规则会加入所有 header 文件，包括被角括号(`<', `>')和双引号(`"")所包围的文件。其实我们可以相当肯定系统 header 档（比如 stdio.h, stdlib.h 等等）不会被我们更改，如果你用-MM 来代替-M 传递给 gcc，那些用角括号包围的 header 档将不会被包括。（这会节省一些编译时间）

由 gcc 输出的规则不会含有命令部分；你可以自己写入你的命令或者什么也不写，而让 make 使用它的隐含的规则（参考下面的 2.4 节）。

2.3 Makefile 变量

上面提到 makefiles 里主要包含一些规则。它们包含的其它的东 西是变量定义。

makefile 里的变量就像一个环境变量(environment variable)。事实上，环境变量在 make 过程中被解释成 make 的变量。这些变量是大小写敏感的，一般使用大写字母。它们可以从几乎任何地方被引用，也可以被用来做很多事情，比如：

i) 贮存一个文件名列表。在上面的例子里，生成可执行文件的规则包含一些目标文件名做为依靠。在这个规则的命令行里同样的那些文件被输送给 gcc 做为命令参数。如果在这里使用一个变数来贮存所有的目标文件名，加入新的目标 文件会变的简单而且较不易出错。

ii) 贮存可执行文件名。如果你的项目被用在一个非 gcc 的系统里，或者如果你想使用一个不同的编译器，你必须将所有使用编译器的地方改成用新的编译器名。但是如果使用一个变量来代替编译器名，那么你只需要改变一个地方，其它所有地方的命令名就都改变了。

iii) 贮存编译器旗标。假设你想给你所有的编译命令传递一组相同的选项（例如-Wall -O -g）；如果你把这组选项存入一个变量，那么你可以把这个变量放在所有呼叫编译器的地方。而当你要改变选项的时候，你只需在一个地方改 变这个变量的内容。

要设定一个变量，你只要在一行的开始写下这个变量的名字，后 面跟一个=号，后面

跟你要设定的这个变量的值。以后你要引用这个变量，写一个$符号，后面是围在括号里的变量名。比如在下面，我们把前面的 makefile 利用变量重写一遍：

```
=== makefile 开始===
OBJS = foo.o bar.o
CC = gcc
CFLAGS = -Wall -O –g

myprog : $(OBJS)
   $(CC) $(OBJS) -o myprog

foo.o : foo.c foo.h bar.h
   $(CC) $(CFLAGS) -c foo.c -o foo.o

bar.o : bar.c bar.h
   $(CC) $(CFLAGS) -c bar.c -o bar.o
=== makefile 结束===
```

还有一些设定好的内部变量，它们根据每一个规则内容定义。三个比较有用的变量是$@, $<和$^（这些变量不需要括号括住）。$@扩展成当前规则的目的文件名，$<扩展成依靠列表中的第一个依靠文件，而$^扩展成整个依靠的列表（除掉了里面所有重复的文件名）。利用这些变量，我们可以把上面的 makefile 写成：

```
=== makefile 开始===
OBJS = foo.o bar.o
CC = gcc
CFLAGS = -Wall -O –g

myprog : $(OBJS)
$(CC) $^ -o $@

foo.o : foo.c foo.h bar.h
  $(CC) $(CFLAGS) -c $<-o $@

bar.o : bar.c bar.h
  $(CC) $(CFLAGS) -c $<-o $@
=== makefile  结束  ===
```

你可以用变量做许多其它的事情，特别是当你把它们和函数混合使用的时候。如果需要更进一步的了解，请参考 GNU Make 手册。('man make', 'man makefile')

2.4 隐含规则 (Implicit Rules)

请注意，在上面的例子里，几个产生.o 文件的命令都是一样的。都是从.c 文件和相关文

件里产生.o 文件，这是一个标准的步骤。其实 make 已经知道怎么做——它有一些叫做隐含规则的内置的规则，这些规则告诉它当你没有给出某些命令的时候，应该怎么办。

如果你把生成 foo.o 和 bar.o 的命令从它们的规则中删除，make 将会查找它的隐含规则，然后会找到一个适当的命令。它的命令会 使用一些变量，因此你可以按照你的想法来设定它：它使用变量 CC 做为编译器（象我们在前面的例子），并且传递变量 CFLAGS（给 C 编译器，C++ 编译器用 CXXFLAGS），CPPFLAGS（C 预处理器旗标），TARGET_ARCH（现在不用考虑这个），然后它加入旗标'-c'，后面跟变量$<（第一个依靠名），然后是旗标'-o'跟变量$@（目的文件名）。一个 C 编译的具体命令将 会是：

$(CC) $(CFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c $<-o $@

当然你可以按照你自己的需要来定义这些变量。这就是为什么用 gcc 的-M 或-MM 开关输出的码可以直接用在一个 makefile 里。

2.5 假象目的(Phony Targets)

假设你的一个项目最后需要产生两个可执行文件。你的主要目标是产生两个可执行文件，但这两个文件是相互独立的——如果一个文件需要重建，并不影响另一个。你可以使用“假象目的”来达到这种效果。一个假象目的跟一个正常的目的几乎是一样的，只是这个目的文件是不存在的。因此，make 总是会假设它需要被生成，当把它的依赖文件更新后，就会执行它的规则里的命令行。

如果在我们的 makefile 开始处输入：

all : exec1 exec2

其中 exec1 和 exec2 是我们做为目的的两个可执行文件。make 把这个'all'做为它的主要目的，每次执行时都会尝试把'all'更新。但既然这行规则里没有哪个命令来作用在一个叫'all'的实际文件（事实上 all 并不会在磁碟上实际产生），所以这个规则并不真的改变'all'的状态。可既然这个文件并不存在，所以 make 会尝试更新 all 规则，因此就检查它的依靠 exec1, exec2 是否需要更新，如果需要，就把它们更新，从而达到我们的目的。

假象目的也可以用来描述一组非预设的动作。例如，你想把所有由 make 产生的文件删除，你可以在 makefile 里设立这样一个规则：

veryclean :

rm *.o

rm myprog

前提是没有其它的规则依靠这个'veryclean'目的，它将永远不会被执行。但是，如果你

明确的使用命令'make veryclean'，make 会把这个目的做为它的主要目标，执行那些 rm 命令。

如果你的磁碟上存在一个叫 veryclean 文件，会发生什么事？这时因为在这个规则里没有任何依靠文件，所以这个目的文件一定是最新的了（所有的依靠文件都已经是最新的了），所以既使用户明确命令 make 重新产生它，也不会有任何事情发生。解决方法是标明所有的假象目的（用.PHONY），这就告诉 make 不用检查它们是否存在于磁碟上，也不用查找任何隐含规则，直接假设指定的目的需要被更新。在 makefile 里加入下面这行包含上面规则的规则：

.PHONY : veryclean

就可以了。注意，这是一个特殊的 make 规则，make 知道.PHONY 是一个特殊目的，当然你可以在它的依靠里加入你想用的任何假象 目的，而 make 知道它们都是假象目的。

2.6 函数 (Functions)

makefile 里的函数跟它的变量很相似——使用的时候，你用一个$符号跟开括号，函数名，空格后跟一列由逗号分隔的参数，最后用关括号结束。例如，在 GNU Make 里有一个叫'wildcard'的函 数，它有一个参数，功能是展开成一列所有符合由其参数描述的文件名，文件间以空格间隔。你可以像下面所示使用这个命令：

SOURCES = $(wildcard *.c)

这行会产生一个所有以'.c'结尾的文件的列表，然后存入变量 SOURCES 里。当然你不需要一定要把结果存入一个变量。

另一个有用的函数是 patsubst（patten substitude，匹配替换的缩写）函数。它需要 3 个参数——第一个是一个需要匹配的式样，第二个表示用什么来替换它，第三个是一个需要被处理的由空格分隔的字列。例如，处理那个经过上面定义后的变量，

OBJS = $(patsubst %.c,%.o,$(SOURCES))

这行将处理所有在 SOURCES 字列中的字（一列文件名），如果它的结尾是'.c'，就用'.o'把'.c'取代。注意这里的%符号将匹配一个或多个字符，而它每次所匹配的字串叫做一个‘柄’(stem)。 在第二个参数里，%被解读成用第一参数所匹配的那个柄。

2.7 一个比较有效的 makefile

利用我们现在所学的，我们可以建立一个相当有效的 makefile。这个 makefile 可以完成大部分我们需要的依靠检查，不用做太大的改变就可直接用在大多数的项目里。

首先我们需要一个基本的 makefile 来建我们的程序。我们可以让它搜索当前目录，找到源码文件，并且假设它们都是属于我们的项目的，放进一个叫 SOURCES 的变量。这里如果

也包含所有的*.cc 文件，也许会更保险，因为源码文件可能是 C++码的。

```
SOURCES = $(wildcard *.c *.cc)
```

利用 patsubst，我们可以由源码文件名产生目标文件名，我们需 要编译出这些目标文件。如果我们的源码文件既有.c 文件，也有.cc 文件，我们需要使用相嵌的 patsubst 函数呼叫：

```
OBJS = $(patsubst %.c,%.o,$(patsubst %.cc,%.o,$(SOURCES)))
```

最里面一层 patsubst 的呼叫会对.cc 文件进行后缀替代，产生的结果被外层的 patsubst 呼叫处理，进行对.c 文件后缀的替代。

现在我们可以设立一个规则来建可执行文件：

```
myprog : $(OBJS)
  gcc -o myprog $(OBJS)
```

进一步的规则不一定需要，gcc 已经知道怎么去生成目标文件(object files)。下面我们可以设定产生依靠信息的规则：

```
depends : $(SOURCES)
  gcc -M $(SOURCES) > depends
```

在这里如果一个叫'depends'的文件不存在，或任何一个源码文件比一个已存在的 depends 文件新，那么一个 depends 文件会被生成。depends 文件将会含有由 gcc 产生的关于源码文件的规则（注意-M 开关）。现在我们要让 make 把这些规则当做 makefile 档 的一部分。这里使用的技巧很像 C 语言中的#include 系统——我们要求 make 把这个文件 include 到 makefile 里，如下：

```
include depends
```

GNU Make 看到这个，检查'depends'目的是否更新了，如果没有，它用我们给它的命令重新产生 depends 档。然后它会把这组（新）规则包含进来，继续处理最终目标'myprog'。当看到有关 myprog 的规则，它会检查所有的目标文件是否更新——利用 depends 文件里的规则，当然这些规则现在已经是更新过的了。

这个系统其实效率很低，因为每当一个源码文件被改动，所有的源码文件都要被预处理以产生一个新的'depends'文件。而且它也不是 100% 的安全，这是因为当一个 header 档被改动，依靠信息并不会被更新。但就基本工作来说，它也算相当有用的了。

2.8 一个更好的 makefile

这是一个我为我大多数项目设计的 makefile。它应该可以不需要修改的用在大部分项目里。我主要把它用在 djgpp 上，那是一个 DOS 版的 gcc 编译器。因此你可以看到执行的命

令名、'alleg'程序包、和 RM -F 变量都反映了这一点。

```
=== makefile 开始 ===
#######################################
#
# Generic makefile
#
# by George Foot
# email: george.foot@merton.ox.ac.uk
#
# Copyright (c) 1997 George Foot
# All rights reserved.
#
# No warranty, no liability;
# you use this at your own risk.
# 没保险，不负责
# 你要用这个，你自己担风险
#
# You are free to modify and
# distribute this without giving
# credit to the original author.
# 你可以随便更改和散发这个文件
# 而不需要给原作者什么荣誉。
# （你好意思？）
#
#######################################
### Customising
# 用户设定
#
# Adjust the following if necessary; EXECUTABLE is the target
# executable's filename, and LIBS is a list of libraries to link in
# (e.g. alleg, stdcx, iostr, etc). You can override these on make's
# command line of course, if you prefer to do it that way.
#
# 如果需要，调整下面的东西。EXECUTABLE 是目标的可执行文件名，LIBS
#是一个需要连接的程序包列表（例如 alleg, stdcx, iostr 等等）。当然你
# 可以在 make 的命令行覆盖它们，你愿意就没问题。
#

EXECUTABLE := mushroom.exe
LIBS := alleg

# Now alter any implicit rules' variables if you like, e.g.:
#
```

```
# 现在来改变任何你想改动的隐含规则中的变量，例如

CFLAGS := -g -Wall -O3 -m486
CXXFLAGS := $(CFLAGS)

# The next bit checks to see whether rm is in your djgpp bin
# directory; if not it uses del instead, but this can cause (harmless)
# `File not found' error messages. If you are not using DOS at all,
# set the variable to something which will unquestioningly remove
# files.
#
# 下面先检查你的 djgpp 命令目录下有没有 rm 命令，如果没有，我们使用
# del 命令来代替，但有可能给我们 'File not found' 这个错误信息，这没
# 什么大碍。如果你不是用 DOS ，把它设定成一个删文件而不废话的命令。
# （其实这一步在 UNIX 类的系统上是多余的，只是方便 DOS 用户。UNIX
# 用户可以删除这 5 行命令。）

ifneq ($(wildcard $(DJDIR)/bin/rm.exe),)
RM-F := rm –f
else
RM-F := del
endif

# You shouldn't need to change anything below this point.
#
# 从这里开始，你应该不需要改动任何东西。

SOURCE := $(wildcard *.c) $(wildcard *.cc)
OBJS := $(patsubst %.c,%.o,$(patsubst %.cc,%.o,$(SOURCE)))
DEPS := $(patsubst %.o,%.d,$(OBJS))
MISSING_DEPS := $(filter-out $(wildcard $(DEPS)),$(DEPS))
MISSING_DEPS_SOURCES := $(wildcard $(patsubst %.d,%.c,$(MISSING_DEPS)) \
$(patsubst %.d,%.cc,$(MISSING_DEPS)))
CPPFLAGS += -MD

.PHONY : everything deps objs clean veryclean rebuild

everything : $(EXECUTABLE)

deps : $(DEPS)

objs : $(OBJS)

clean :
```

```
  @$(RM-F) *.o
  @$(RM-F) *.d

veryclean: clean
  @$(RM-F) $(EXECUTABLE)

rebuild: veryclean everything

ifneq ($(MISSING_DEPS),)
$(MISSING_DEPS) :
  @$(RM-F) $(patsubst %.d,%.o,$@)
endif

-include $(DEPS)

$(EXECUTABLE) : $(OBJS)
  gcc -o $(EXECUTABLE) $(OBJS) $(addprefix -l,$(LIBS))
```

===makefile 结束 ===

有几个地方值得解释一下的。首先，我在定义大部分变量的时候使 用的是:=而不是=符号。它的作用是立即把定义中参考到的函数和变量都展开了。如果使用=的话，函数和变量参考会留在那儿，就是说改变一个变量的值会导致其它变量的值也被改变。例如：

```
A = foo

B = $(A)

# 现在 B 是 $(A) ，而 $(A) 是 'foo' 。

A = bar

# 现在 B 仍然是 $(A) ，但它的值已随着变成 'bar' 了。

B := $(A)

# 现在 B 的值是 'bar' 。

A = foo

# B 的值仍然是 'bar' 。
```

make 会忽略在 # 符号后面直到那一行结束的所有文字。

ifneg...else...endif 系统是 makefile 里让某一部分码有条件的失效 / 有效的工具。ifeq 使用两个参数，如果它们相同，它把直到 else（或者 endif，如果没有 else 的话）的一段码加进 makefile 里；如果不同，把 else 到 endif 间的一段码加入 makefile（如果有 else）。ifneq 的用法刚好相反。

'filter-out'函数使用两个用空格分开的列表，它把第二列表中所有的存在于第一列表中的项目删除。我用它来处理 DEPS 列表，把所有已经存在的项目都删除，而只保留缺少的那些。

我前面说过，CPPFLAGS 存有用于隐含规则中传给预处理器的一些旗标。而-MD 开关类似-M 开关，但是从源码文件.c 或.cc 中形成的文件名是使用后缀.d 的（这就解释了我形成 DEPS 变量的步骤）。DEPS 里提到的文件后来用'-include'加进了 makefile 里，它隐藏了所有因文件不存在而产生的错误信息。

如果任何依靠文件不存在，makefile 会把相应的.o 文件从磁碟上删除，从而使得 make 重建它。因为 CPPFLAGS 指定了-MD，它的.d 文件也被重新产生。

最后，'addprefix'函数把第二个参数列表的每一项前缀上第一个参数值。

这个 makefile 的那些目的是（这些目的可以传给 make 的命令行来直接选用）：

everything:（预设）更新主要的可执行程序，并且为每一个源码文件生成或更新一个'.d' 文件和一个 '.o' 文件。

deps: 只是为每一个源码程序产生或更新一个'.d'文件。

objs: 为每一个源码程序生成或更新'.d'文件和目标文件。

clean: 删除所有中介／依靠文件（*.d 和*.o）。

veryclean: 做`clean' 和删除可执行文件。

rebuild: 先做`veryclean'然后`everything'；既完全重建。

除了预设的 everything 以外，这里头只有 clean，veryclean，和 rebuild 对用户是有意义的。

我还没有发现当给出一个源码文件的目录，这个 makefile 会失败的情况，除非依靠文件被弄乱。如果这种弄乱的情况发生了，只要输入`make clean' ，所有的目标文件和依靠文件会被删除，问题就应该被解决了。当然，最好不要把它们弄乱。

3 总结

希望这篇文章足够详细的解释了多文件项目是怎么运作的，也说明了怎样安全而合理的使用它。到此，你应该可以轻松的利用 GNU Make 工具来管理小型的项目，如果你完全理解了后面几个部分的话，这些对于你来说应该没什么困难。

GNU Make 是一件强大的工具，虽然它主要是用来建立程序，它还有很多 别的用处。如果想要知道更多有关这个工具的知识，它的句法，函数，和许多别的特点，你应该参看它的参考文件(info pages, 别的 GNU 工具也一样，看它们的 info pages.)。