神经网络和深度学习笔记

张晋
北京航空航天大学
数学与系统科学学院

最后更新于：May 26, 2017

目录 | 0

1 神经网络简介

“神经网络是由具有适应性的简单单元组成的广泛并行互连的网络，它的组织能够模拟生物神经系统对真实世界物体所作出的交互反应”—Kohonen

1.1 生物神经网络 (Biological Neural Networks)

大脑可视作为 1000 多亿神经元组成的神经网络。

神经元的信息传递和处理是一种电化学活动. 树突由于电化学作用接受外界的刺激；通过胞体内的活动体现为轴突电位，当轴突电位达到一定的值则形成神经脉冲或动作电位；再通过轴突末梢传递给其它的神经元. 从控制论的观点来看；这一过程可以看作一个多输入单输出非线性系统的动态过程。

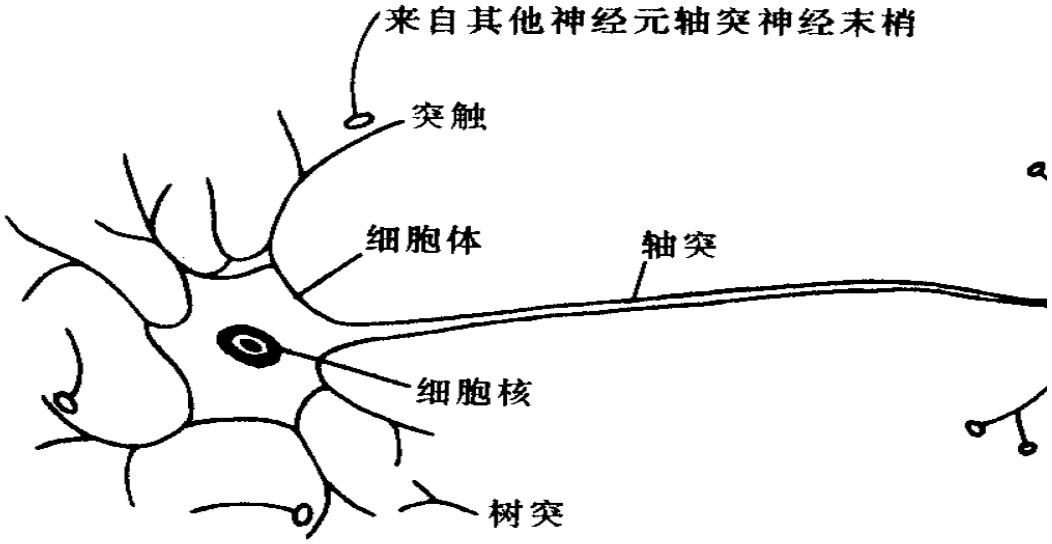


图 1.1: 神经元信息的传递

大脑处理信息的特点

- **分布存储与冗余性**: 记忆在大量元中，每个元存在许多信息的部分内容，信息在神经网络中的记忆反映在神经元间的突触连接强度上
- **并行处理**: 经网络既是处理器又是存储器 (并行处理不同于并行机)
- **信息处理与存储合一**: 每个元兼有二者功能
- **可塑性与自组织性**: 可塑性是学习记忆的基础
- **鲁棒性**: 高连接度导致一定的误差和噪声不会使网络性能恶化。是智能演化的重要因素

1.2 人工神经网络 (Artificial Neural Networks)

神经网络是一个并行和分布式的信息处理网络结构，它一般由许多个神经元组成，每个神经元只有一个输出，它可以连接到很多其他的神经元，每个神经元输入有多个连接通道，每个连接通道对应于一个连接权系数。

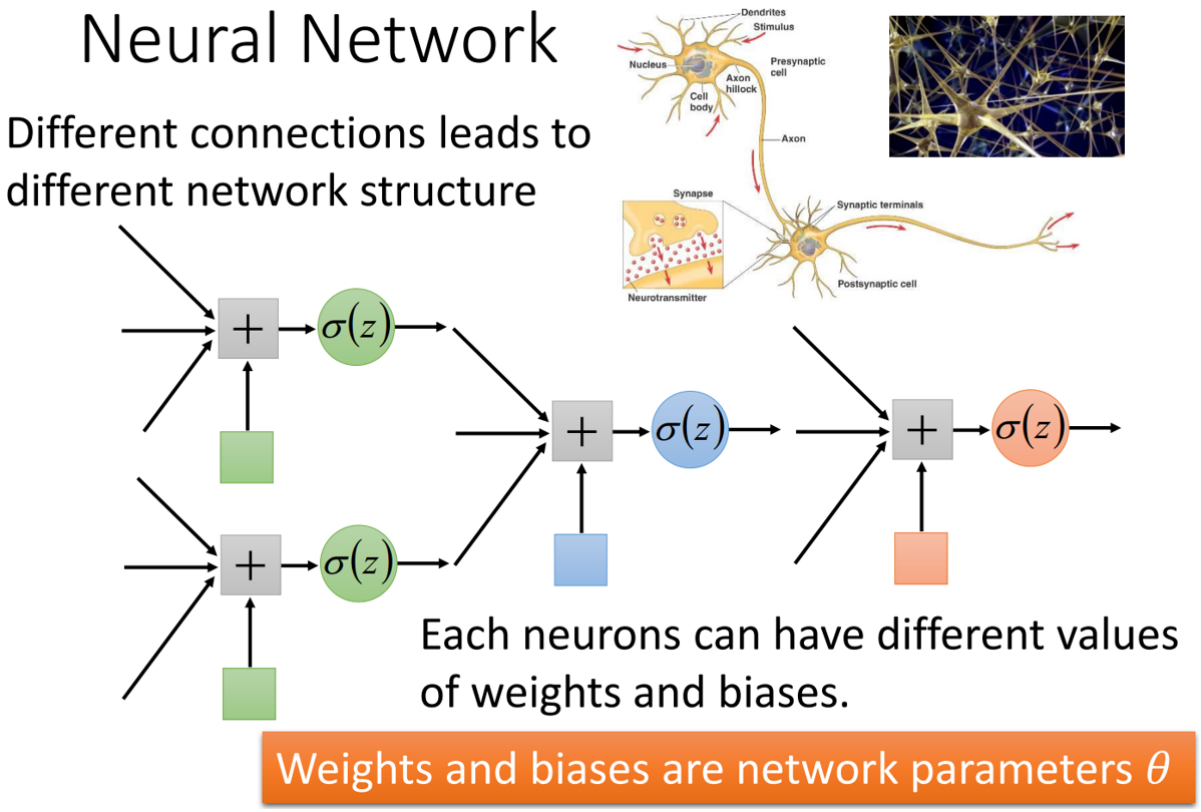


图 1.2: Artificial Neural Networks

1.2.1 人工神经元模型

神经网络中最基本的成分是神经元 (neuron) 模型，即上述定义中的“简单单元”。在生物神经网络中，每个神经元与其他神经元相连，当它“兴奋”时，就会向相连的神经元发送化学物质，从而改变这些神经元内的电位；如果某神经元的电位超过了一个”阈值”(threshold)，那么它就会被激活，即“兴奋”起来，向其他神经元发送化学物质。

1943 年，[Mccu11ochandPitts,1943] 将上述情形抽象为图所示的简单模型，这就是一直沿用至今的“M-P 神经元模型”在这个模型中，神经元接收到来自 n 个其他神经元传递过来的输入信号，这些输入信号通过带权重的连接 (connection) 进行传递，神经元接收到的总输入值将与神经元的阈值进行比较然后通过“激活函数”(activation function)处理已产生神经元的输出。

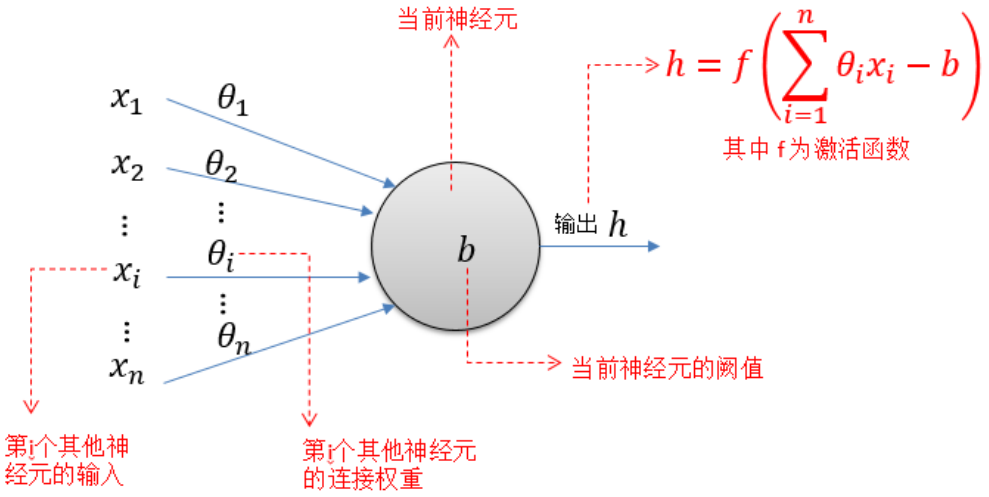
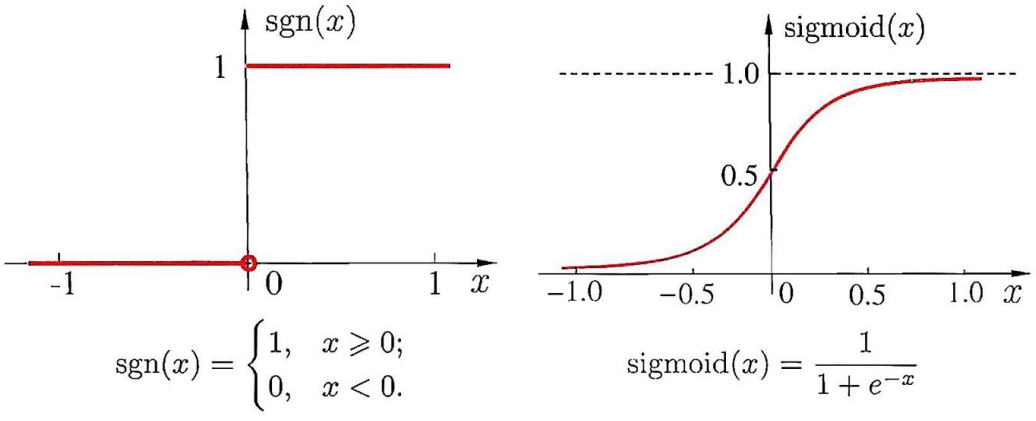


图 1.3: Artificial Neuron Model

激活函数

理想中的激活函数 $f(x)$ 是图 (a) 所示的阶跃函数，它将输入值映射为输出值“0”或“1”,“1”对应神经元兴奋，“0”对应神经元抑制。但是，阶跃函数具有不连续，不光滑 (不连续可导) 等不太好的性质[1]，因此实际中常用 Logistic 回归中应用到的sigmoid 函数作为激活函数。典型的 sigmoid 函数如图 (b) 所示，它把可能在较大范围内变化的输入值挤压到 (0,1) 输出值范围内，因此有时又称之为“挤压函数”(squashing function).



(a) 阶跃函数　　(b) Sigmoid 函数

图 1.4: 激活函数

[1]事实上在使用阶跃函数时，神经网络中的权值或偏置的微小改变可能会造成输出结果天翻地覆的变化，这使得网络的行为变得复杂且难以控制。

1.2.2 人工神经网络的构成

■ 感知机 (Perceptron)

感知机 (Perceptron) 由两层神经元组成，如图所示，输入层接收外界输入信号后传递给输出层，输出层是 M-P 神经元，亦称” 阈值逻辑单元”(threshold logic unit)

感知器处理单元对 n 个**二进制**输入 $x_1, x_2, \ldots$，进行加权和操作并产生一个二进制输出:

$$\boxed{y = f\Big(\sum_i w_i x_i - \theta\Big)}$$

感知机能容易地实现逻辑**与、或、非**运算：

- 先定义 $f(x)$ 为阶跃函数
- “与” $(x_1 \wedge x_2)$: 令 $w_1 = w_2 = 1, \theta = 2$; 则仅在 $x_1 = x_2 = 1$ 时,$y = 1$;
- “或” $(x_1 \vee x_2)$: 令 $w_1 = w_2 = 1, \theta = 0.5$; 则仅在 $x_1 = x_2 = 1$ 时,$y = 1$;
- “非” $(\neg x_1)$: 令 $w_1 = -0.6, w_2 = 0, \theta = -0.5$; 那么当 $x_1 = 1$ 时,$y = 0$;

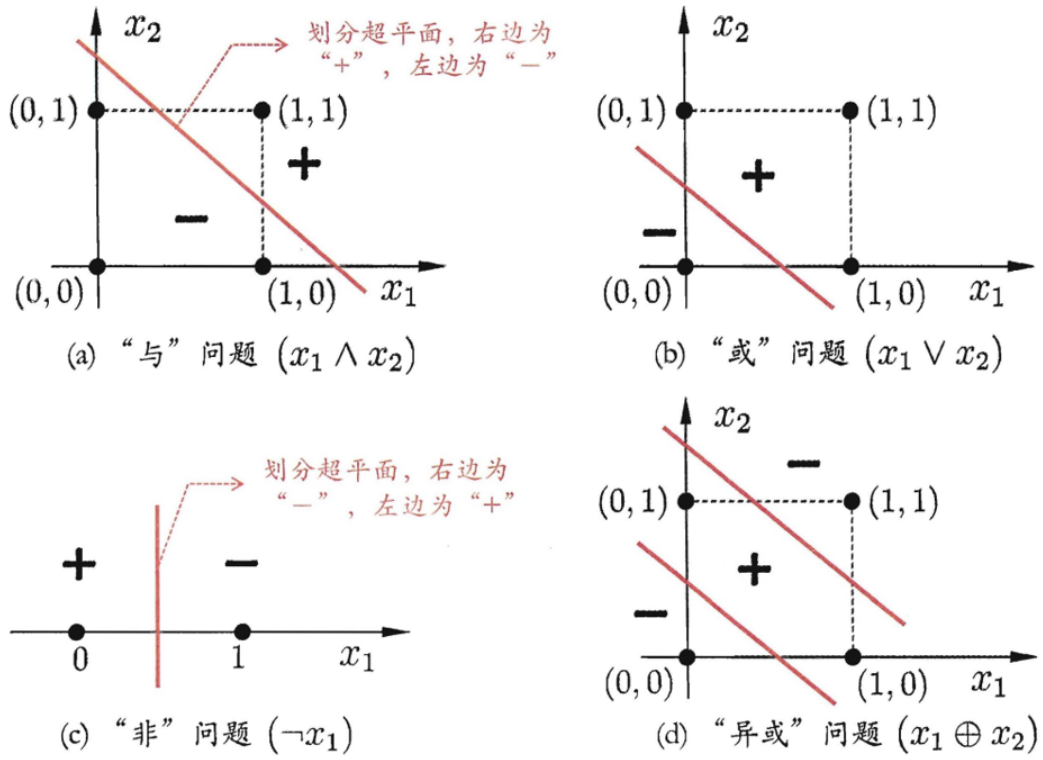


图 1.5: 与、或、非运算的实现

需注意的是，感知机只有输出层神经元进行激活函数处理，即只拥有一层功能神经元 (functional neuron), 其学习能力非常有限，只能处理线性可分 (linearly separable)的问题。

从图中可以看出，与、或、非问题的分类样本是可以用一条直线分开的，即具有“线性可分”性质。

■ 多层感知机 (MLP)

要解决非线性可分问题, 就需考虑多层功能神经元。

而“异或”问题的分类样本需要两条线才能将其分开，故需要两层感知机才能实现，如图 1.6

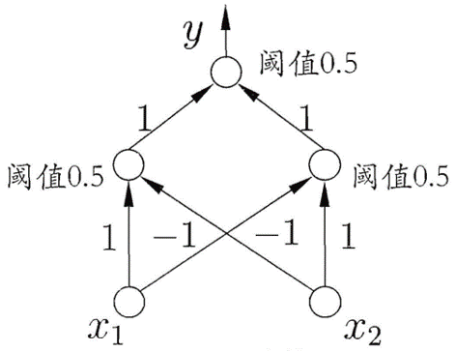
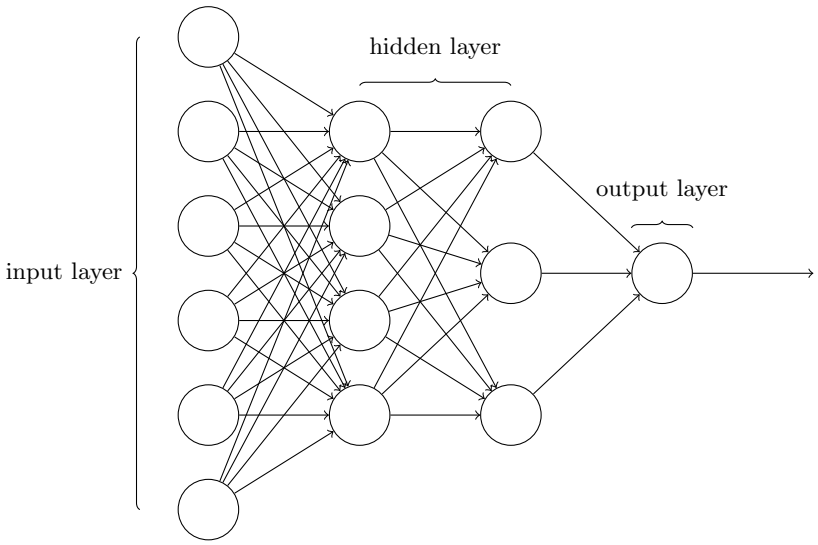


图 1.6: 能解决异或问题的两层感知机



- 这个网络中最左边的称为**输入层**，其中的神经元称为**输入神经元**。
- 最右边的，即**输出层**包含有**输出神经元**。
- 中间层，这层中的神经元既不是输入也不是输出，被称为**隐藏层**。
- 隐含层和输出层神经元都是拥有激活函数的功能神经元.
- 只需包含隐层，即可称为多层网络. 神经网络的学习过程，就是根据训练数据来调整神经元之间的“连接权”(connection weight)以及每个功能神经元的阈值;
- 换言之，神经网络“学”到的东西，蕴涵在连接权与阈值中。

1.2.3 人工神经网络的学习

1. 试用数值方法求出 $f(x,y)$ 在区域 $D=\{(x,y)|0\le x\le 0.8, 0.5\le y\le 1.5\}$ 上的一个近似表达式：

$$\boxed{p(x,y)=\sum_{r,s=0}^{k} c_{rs}x^r y^s}$$

要求 $p(x,y)$ 最小的 k 值达到以下的精度：

$$\sigma=\sum_{i=0}^{10}\sum_{j=0}^{20}\left(f(x_i,y_j)-p(x_i,y_j)\right)^2\le 10^{-7}$$

其中，$x_i=0.08i, y_j=0.5+0.05j$。

2. 计算 $f(x_i^*,y_j^*), p(x_i^*,y_j^*) \quad (i=1,2,\cdots,8; j=1,2,\cdots,5)$ 的值，以观察 $p(x,y)$ 逼近 $f(x,y)$ 的效果，其中,$x_i^*=0.1\boldsymbol{i}, y_j^*=0.5+0.2\boldsymbol{j}$。

说明：

1. 用迭代方法求解非线性方程组时，要求近似解向量 $\boldsymbol{x}^{(k)}$ 满足以下精度

$$\frac{\|\boldsymbol{x}^{(k)}-\boldsymbol{x}^{(k-1)}\|_\infty}{\|\boldsymbol{x}^{(k)}\|_\infty}\le 10^{-12}$$

2. 作二元插值时，要使用分片二次代数插值。

3. 要由程序自动确定最小的 k 值。

4. 打印以下内容：

   (a) 全部源程序;

   (b) 数表:$(x_i,y_j,f(x_i,y_j)) \quad (i=0,1,2,\cdots,10; j=0,1,2,\cdots,20)$;

   (c) 选择过程的 k,σ 值;

   (d) 达到精度要求时的 k 和 σ 值以及 $p(x,y)$ 中的系数 $c_{rs}(r=0,1,\cdots,k; s=0,1,\cdots,k)$;

   (e) 数表:$(x_i^*,y_j^*,f(x_i^*,y_j^*)), p(x_i^*,y_j^*) \quad (i=1,2,\cdots,8; j=1,2,\cdots,5)$。

5. 采用 f 型输出 x_i,y_j,x_i^*,y_j^* 的准确值，其余实型数采用 e 型输出并且至少显示 12 位有效数字。

2 算法设计方案

2.1 方案

1. 将 $x_i = 0.08i, y_j = 0.5 + 0.05j \quad (i = 0, 1, \cdots, 10; j = 0, 1, \cdots, 20)$ 代入方程组中，用 Newton 迭代法解出 t_{ij} 和 u_{ij};
2. 对数表 $z(t, u)$ 进行分片二次代数插值, 求得 $z_{ij} = f(t_{ij}, u_{ij})$
3. 根据 z_{ij} 的值进行曲面拟合，得 $p(x, y)$
4. 观察比较

2.2 Newton 迭代法

$$\begin{cases} 0.5\cos t + u + v + w - x = 2.67 \\ t + 0.5\sin u + v + w - y = 1.07 \\ 0.5t + u + \cos v + w - x = 3.74 \\ t + 0.5u + v + \sin w - y = 0.79 \end{cases}$$

对于该非线性方程组方程组来说，x, y 为已知量，需解出 t, u, v, w。
设 $\boldsymbol{x} = (t, u, v, w)^T$, 并设定精度水平 $\varepsilon = 10^{-12}$ 和最大迭代次数 M
先在 $\boldsymbol{x}^*$ 附件选取 $\boldsymbol{x}^{(0)} = (t^{(0)}, u^{(0)}, v^{(0)}, w^{(0)})^T$
具体算法如下：

Algorithm 1 Newton's method

Set $\boldsymbol{x}^{(0)} \in D$ and $k = 0$
while k<M **do**
  Compute $\boldsymbol{F}(\boldsymbol{x}^{(k)})$ and $\boldsymbol{F}'(\boldsymbol{x}^{(k)})$
  Compute $\Delta\boldsymbol{x}^{(k)} = -[\boldsymbol{F}'(\boldsymbol{x}^{(k)})]^{-1}\boldsymbol{F}(\boldsymbol{x}^{(k)})$
  if $\|\Delta\boldsymbol{x}^{(k)}\|/\|\boldsymbol{x}^{(k+1)}\| \leq \varepsilon$ **then**
    $\boldsymbol{x}^* = \boldsymbol{x}^{(k)}$
    Break
  end if
  $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \Delta\boldsymbol{x}^{(k)}$
  k=k+1
end while

2.3 分片双二次插值

为了加快 QR 迭代的收敛速度, 我们可以采用位移策略和反迭代思想.

Algorithm 2 QR Iteration with shift

Set $A_1 = A$ and $k = 1$
while not convergence **do**
  Choose a shift σ_k
  $A_k - \sigma_k I = Q_k R_k$
  Compute $A_{k+1} = R_k Q_k + \sigma_k I$
  k=k+1
end while

我们有

$$\begin{aligned} A_{k+1} = R_k Q_k + \sigma_k I &= (Q_k^T Q_k) R_k Q_k + \sigma_k I \\ &= Q_k^T (A_k - \sigma_k I) Q_k + \sigma_k I \\ &= Q_k^T A_k Q_k \end{aligned}$$

所以, 带位移的 QR 算法中所得到的所有矩阵 A_k 都与 A 正交相似.

3 问题求解

1. 初始化定义 A 矩阵
2. 采用 $Householder$ 变化将 A 矩阵拟上三角化得到矩阵 A^{n-1}
3. 采用 QR 分解将矩阵 A^{n-1} 分解为矩阵 Q 与 R，并打印 RQ
4. 用带双步位移的 QR 方法求解所有特征值
5. 用 Gauss 消去法求解 $(A-\lambda I)X=0$, 得到对应的特征向量
6. 输出

源程序 4

4.1 C 语言版大作业

```c
#include<stdio.h>
#include<math.h>
#include<string.h>
#define N 20
const double eps=1e-12;
double a[N][N],B[N][N],C[N][N];
int n=10;

typedef struct{
//定义复数结构体
    double Re;
    double Im;
}ComplexNumber;

void zeroMat(double a[N][N]){
    for(int i= 1; i<=10; i++) {
        for(int j = 1; j<=10; j++) {
            if (fabs(a[i][j])<eps)
                a[i][j] = 0;
        }
    }
}

void def(){
    //初始化A数组
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++){
            if(i==j)
                a[i][j]=1.52*cos(i+1.2*j);
            else if(i!=j)
                a[i][j]=sin(0.5*i+0.2*j);
```

```
32          }
33  }
34
35  void eye(double Q[N][N]){
36      //定义单位阵
37      for(int i=1;i<=n;i++)
38          for(int j=1;j<=n;j++)
39              Q[i][j]=(i==j);
40  }
41
42  int Is_zero(double Q[N][N],int r,int k){
43      //K>0表示在QR中调用，K<0表示在 DQR中调用
44      int kk=n;//更改数组大小
45      if(k<0){kk=-k;k=1;}
46      for(int i=r+k;i<=kk;i++)
47          if(fabs(Q[i][r])>eps)
48              return 0;
49      return 1;
50  }
51
52
53  int sgn(double x){
54      if(x>0)
55          return 1;
56      return -1;
57  }
58
59  void Matirix_M(double A[N][N],double x[N],double y[N],double h){
60      for(int i=1;i<=n;i++)y[i]=0;
61      for(int i=1;i<=n;i++)
62          for(int j=1;j<=n;j++)
63              y[i]+=A[i][j]*x[j]/h;
64  }
65
66  double Vector_M(double x[N],double y[N]){
67      double sum=0;
68      for(int i=1;i<=n;i++)
69          sum+=x[i]*y[i];
70      return sum;
71  }
72
73  void MatirixM_M(double X[N][N],double Y[N][N]){
74      for(int i=1;i<=n;i++)
75          for(int j=1;j<=n;j++)
76              for(int k=1;k<=n;k++)
77                  B[i][j]+=X[i][k]*Y[k][j];
78  }
79
80  void copy(double X[N][N],double Y[N][N]){
```

```
81      for(int i=1;i<=n;i++)
82          for(int j=1;j<=n;j++)
83              X[i][j]=Y[i][j];
84  }
85
86  void output(double A[N][N]){
87      for(int i=1;i<=n;i++){
88          for(int j=1;j<=n;j++)
89              printf("%.12e ",A[i][j]);
90          printf("\n");
91      }
92  }
93
94  void QR(double A[N][N],double Q[N][N]){
95      double u[N],w[N],p[N],d,c,h;
96      eye(Q);
97      for(int r=1;r<n;r++){
98          d=0;
99          if(Is_zero(A,r,1))continue;
100         for(int i=r;i<=n;i++)
101             d+=A[i][r]*A[i][r];
102         d=sqrt(d);
103         c=-sgn(A[r][r])*d;
104         h=c*c-c*A[r][r];
105         for(int i=1;i<=n;i++){
106             if(i<r)
107                 u[i]=0;
108             else if(i==r)
109                 u[i]=A[r][r]-c;
110             else if(i>r)
111                 u[i]=A[i][r];
112         }
113         memset(w,0,sizeof(w));
114         Matirix_M(Q,u,w,1);
115         for(int i=1;i<=n;i++)
116             for(int j=1;j<=n;j++)
117                 Q[i][j]-=(w[i]*u[j]/h);
118         memset(p,0,sizeof(p));
119         for(int i=1;i<=n;i++)
120             for(int j=1;j<=n;j++)
121                 p[i]+=(A[j][i]*u[j]/h);
122         for(int i=1;i<=n;i++)
123             for(int j=1;j<=n;j++)
124                 A[i][j]-=(u[i]*p[j]);
125     }
126 }
127
128 void Householder_Triangularization(double A[N][N]){
129     double u[N],p[N],q[N],d,c,h;
```

```
130     for(int r=1;r<n-1;r++){
131         d=0;
132         if(Is_zero(A,r,2))continue;
133         for(int i=r+1;i<=n;i++)
134             d+=A[i][r]*A[i][r];
135         d=sqrt(d);
136         c=-sgn(A[r+1][r])*d;
137         h=c*c-c*A[r+1][r];
138         for(int i=1;i<=n;i++){
139             if(i<=r)
140                 u[i]=0;
141             else if(i==(r+1))
142                 u[i]=A[i][r]-c;
143             else if(i>r)
144                 u[i]=A[i][r];
145         }
146         memset(p,0,sizeof(p));
147         for(int i=1;i<=n;i++)
148             for(int j=1;j<=n;j++)
149                 p[i]+=(A[j][i]*u[j]/h);
150         Matirix_M(A,u,q,h);
151         c=Vector_M(p,u)/h;
152         for(int i=1;i<=n;i++)
153             q[i]-=u[i]*c;
154         for(int i=1;i<=n;i++)
155             for(int j=1;j<=n;j++)
156                 A[i][j]-=(q[i]*u[j]+u[i]*p[j]);
157     }
158 }
159 
160 
161 void select(int k,double b[N][N]){
162     double max,c;
163     int m=k;
164     max=b[k][k];
165     for(int i=k+1;i<=n;i++)
166         if(fabs(b[i][k])>fabs(max)){
167             max=b[i][k];
168             m=i;
169         }
170     if(m!=k)
171         for(int i=k;i<=n+1;i++){
172         c=b[m][i];
173         b[m][i]=b[k][i];
174         b[k][i]=c;
175         }
176 }
177 
178 //Gauss消元法
```

```
179 void gauss(double lambda)
180 {
181     double X[N];
182     double m,sum,t;
183     def();
184     memset(X,0,sizeof(X));
185     for(int i=1;i<=n;i++){
186         a[i][i]-=lambda;
187         a[i][n+1]=0;
188     }
189     for(int k=1;k<n;k++){
190         select(k,a);
191         for(int i=k+1;i<=n;i++){
192             m=a[i][k]/a[k][k];
193         for(int j=k;j<=n+1;j++)
194             a[i][j]-=m*a[k][j];
195         }
196     }
197     X[n]=1;
198     for(int i=n-1;i>=1;i--){
199         sum=0;
200         for(int j=i+1;j<=n;j++)
201             sum+=a[i][j]*X[j];
202         if(fabs(a[i][i])>eps)
203             X[i]=(a[i][n+1]-sum)/a[i][i];
204         else
205             X[i]=0;
206     }
207     t=0;
208     for(int i=1;i<=n;i++)
209         t+=X[i]*X[i];
210     t=sqrt(t);
211     printf("Eigenvector=(");
212     for(int i=1;i<=n;i++)
213         printf("%lf ", X[i]/t);
214     printf(")\n");
215 }
216
217 void DQR(double A[N][N]){
218     double Q[N][N],M[N][N],s,t,re,im,a,b,c,d;
219     double u[N],v[N],p[N],q[N],h,det;
220     ComplexNumber L[N];
221     int m=n,LL=1000,r=1;
222     for(int k=1;k<=100;k++){
223         if (m == 1) {
224             L[r].Re = A[m][m];
225             L[r].Im = 0; break;
226         }
227         else if(m<=0)break;
```

```
228         if(fabs(A[m][m-1])<=eps){
229             L[r].Re = A[m][m];
230             L[r].Im = 0;
231             m--;r++;continue;
232         }
233         a=A[m-1][m-1];
234         b=A[m-1][m];
235         c=A[m][m-1];
236         d=A[m][m];
237         re=(a+d)/2;
238         det=(a-d)*(a-d)+4*b*c;
239 
240         if((m==2)||(fabs(A[m-1][m-2])<=eps)){
241             if(det>0){
242                 L[r].Re = re+sqrt(det)/2;
243                 L[r].Im = 0;
244                 L[r+1].Re = re-sqrt(det)/2;
245                 L[r+1].Im = 0;
246             }
247             else if(det<0){
248                 L[r].Re = re;
249                 L[r].Im=sqrt(fabs(det))/2;
250                 L[r+1].Re = re;
251                 L[r+1].Im = -L[r].Im;
252             }
253             m-=2;
254             r+=2;
255             continue;
256         }
257         if(k==LL)break;
258         s=a+d;
259         t=a*d-c*b;
260         memset(M,0,sizeof(M));
261         for(int i=1;i<=m;i++)
262             for(int j=1;j<=m;j++){
263                 for(int l=1;l<=m;l++)
264                     M[i][j]+=A[i][l]*A[l][j];
265                 M[i][j] -= s*A[i][j];
266                 M[i][j] += t*(i==j);
267             }
268 
269         for(int r=1;r<m;r++){
270             d=0;
271             if(Is_zero(M,r,-m))continue;
272             for(int i=r;i<=m;i++)
273                 d+=M[i][r]*M[i][r];
274             d=sqrt(d);
275             c=-sgn(M[r][r])*d;
276             h=c*c-c*M[r][r];
```

```
277             for(int i=1;i<=m;i++){
278                 if(i<r)
279                     u[i]=0;
280                 else if(i==r)
281                     u[i]=M[r][r]-c;
282                 else if(i>r)
283                     u[i]=M[i][r];
284             }
285             memset(p,0,sizeof(p));
286             memset(q,0,sizeof(q));
287             memset(v,0,sizeof(v));
288             for(int i=1;i<=m;i++)
289                 for(int j=1;j<=m;j++)
290                     v[i]+=(M[j][i]*u[j]/h);
291             for(int i=1;i<=m;i++)
292                 for(int j=1;j<=m;j++){
293                     M[i][j]-=(u[i]*v[j]);
294                     p[i]+=(A[j][i]*u[j]/h);
295                     q[i]+=(A[i][j]*u[j]/h);
296                 }
297 
298             c=0;
299             for(int i=1;i<=m;i++)
300                 c+=p[i]*u[i]/h;
301             for(int i=1;i<=m;i++)
302                 q[i]-=u[i]*c;
303             for(int i=1;i<=m;i++)
304                 for(int j=1;j<=m;j++)
305                     A[i][j]-=(q[i]*u[j]+u[i]*p[j]);
306             zeroMat(A);
307         }
308 
309     }
310     zeroMat(A);
311     for(int r = 1; r<=10; r++){
312         printf("\n");
313         if (L[r].Im == 0) {
314             printf("lambda[%d] = %.12e \n", r , L[r].Re);
315             gauss(L[r].Re);
316         }
317         else {
318             printf("lambda[%d] = %.12e + i*%.12e\n", r , L[r].Re, L[
                    r].Im);
319         }
320     }
321 }
322 
323 int main(){
324     double Q[N][N];
```

```
325     freopen("Works.in","r",stdin);
326     freopen("Works.out","w",stdout);
327     def();
328     Householder_Triangularization(a);
329     printf("A_n-1:\n");
330     output(a);
331     QR(a,Q);
332     printf("\nR:\n");
333     output(a);
334     printf("\nQ:\n");
335     output(Q);
336     memset(B,0,sizeof(B));
337     MatirixM_M(a,Q);
338     printf("\nR*Q:\n");
339     output(B);
340     def();
341     DQR(a);
342     return 0;
343 }
```

5 计算结果

后面为输出结果:

特征值与向量如下:

$\lambda_1 = 9.432879572769e-001$

Eigenvector=(0.079620, 0.045421, -0.018272, -0.047961, -0.349567, 0.207215, -0.152312, 0.820634, -0.355466, 0.028866)

$\lambda_2 = 6.489488202110e-001$

Eigenvector=(0.108435, 0.071344, 0.382502, -0.047100, -0.717804, 0.181519, -0.226006, 0.388381, 0.289696, 0.024333)

$\lambda_3 = -9.891143464725e-001$+ i*1.084758631513e-001

$\lambda_4 = -9.891143464725e-001$- i*1.084758631513e-001

$\lambda_5 = 4.954990923624e-002$

Eigenvector=(-0.213768, -0.206774, 0.386829, -0.031112, -0.380939, -0.125174, 0.644716, -0.308201, -0.295977, 0.043723)

$\lambda_6 = -1.493147080915e+000$

Eigenvector=(-0.561341, 0.778192, 0.014364, -0.277602, 0.003568, -0.002548, -0.022061, -0.011758, -0.013173, 0.035016)

$\lambda_7 = 1.590313458807e+000$

Eigenvector=(0.062377, -0.011231, -0.252846, -0.130988, -0.381985, 0.815575, -0.123377, -0.067721, 0.271945, 0.100282)

$\lambda_8 = -2.336865932238e+000$+ i*8.934379210213e-001

$\lambda_9 = -2.336865932238e+000$- i*8.934379210213e-001

$\lambda_{10} = 3.389613438816e+000$

Eigenvector=(-0.104872, -0.217677, -0.474694, -0.259384, -0.304665, -0.259452, 0.086866, 0.405258, 0.509628, 0.239515)

讨论 6

1. 在对 $n \times n$ 实矩阵 $\boldsymbol{A}$ 作 QR 分解或双步位移分解时不需要形成具体的矩阵 $\boldsymbol{H}_i$，这样可以避免矩阵与矩阵相乘，大大减少了计算量。

2. 在 QR 分解中，R 矩阵可以直接储存在 A 的上三角部分，而在 M_k 的 QR 分解中，不需要再生成 $\boldsymbol{B}, \boldsymbol{C}$ 矩阵，直接在 $\boldsymbol{M}, \boldsymbol{A}$ 矩阵中迭代即可，以节约储存空间。

3.QR 方法适用于计算一般实矩阵的全部特征值，但对于大型实矩阵，则收敛速度不够用了，这时我们为了加速收敛，可以引入位移量, 通常, 位移越离特征值越近, 收敛速度就越快，如果位移 σ 与某个特征值非常接近, 则 $\boldsymbol{A}_{n,n}^{(k)} - \sigma$ 就非常接近于 0. 这说明 $\boldsymbol{A}_{n,n}^{(k)}$ 通常会首先收敛到 $\boldsymbol{A}$ 的一个特征值. 所以令 $\sigma = \boldsymbol{A}_{n,n}^{(k)}$ 是一个不错的选择. 但是, 如果这个特征值是复数, 这种位移选取方法就可能失效.

假设 $\sigma \in \mathbb{C}$ 是 $\boldsymbol{A}$ 的某个复特征值 λ 的一个很好的近似, 则其共轭 $\overline{\sigma}$ 也应该是 $\overline{\lambda}$ 的一个很好的近似. 因此我们可以考虑**双位移策略**, 即先以 σ 为位移迭代一次, 然后再以 $\overline{\sigma}$ 为位移迭代一次, 如此不断交替进行迭代. 这样就有

$$\begin{aligned} A_1 - \sigma I &= Q_1 R_1, \\ A_2 &= R_1 Q_1 + \sigma I, \\ A_2 - \overline{\sigma} I &= Q_2 R_2, \\ A_3 &= R_2 Q_2 + \overline{\sigma} I. \end{aligned}$$

容易验证

$$A_3 = Q_2^T A_2 Q_2 = Q_2^* Q_1^* A_1 Q_1 Q_2 = Q^* A_1 Q,$$

其中 $Q = Q_1 Q_2$.

我们注意到 σ 可能是复的, 所以 Q_1 和 Q_2 都可能是复矩阵. 但我们却可以选取适当的 Q_1 和 Q_2, 使的 $Q = Q_1 Q_2$ 是实正交矩阵从而 $A_3 = Q^T A_1 Q$ 也是实矩阵. 因此我们无需计算 A_2, 而是直接由 A_1 计算出 A_3.

4. 最后，在用 Gauss 消去法求解 $(A - \lambda I)X = 0$, 求得到对应的特征向量时, 需要将 $X[n]$ 预设为 1，因为由于线性方程组 $AX = b$ 中的 b 向量全为 0，所以解的自由度为 1，需要自己先将 $X[n]$ 的值定下来，才能迭代出剩下的解，不然求出的的解全为 0.