**第 14 次课 学时 2**

|  |  |
| --- | --- |
| **授课章节内容** | 第八章 有噪信道编码 |
|  | 8.4 检错和纠错能力 |
| 8.5 标准阵的用途 |
| **教学目标** | 教学目标5 |
| **支撑毕业要求** | 毕业要求2-1 |

**教学要求：**

1. 知识目标

* 掌握线性分组码的纠检错能力；学会标准阵的用途。

1. 能力目标

* 具备判断一个线性码性能优劣和利用标准阵设计符合需求的线性码的能力。

1. 素质目标

* 激发学生对电子信息相关专业课程的学习热情，通过对标准阵的学习，提升运用理论指导实践的工程素质。

**教学重点与难点**：

* 利用标准阵设计线性码

**教学过程设计：**

在前一次课基础上，进一步分析线性码的纠检错能力，运用标准阵设计线性码

| **讲授与指导内容** | | **讲课、互动内容设计** |
| --- | --- | --- |
| 8.5标准阵的用途  8.5.1 估码能力  可以将标准阵列想象成一个有组织的管理工具，一个被填满的小屋，它装着所有的*n*元组空间中的个项——没有遗漏，也没有重复。直觉上看，这个工具的好处局限于小的分组码，因为码字长度大于*n*=20时，空间中将有数以百万计的*n*元组。不过，即使对于较大的码本，标准阵列也可以将一些重要性能问题，例如在纠错和检错间权衡的可能性以及纠错能力的界限等，进行可视化处理。一种称为汉明界限（Hamming bound）的界限表述如下：  （8.39*a*）  或陪集数目： （8.39b）  其中，表示*n*位中有*j*比特出错的可能形式的数目。注意，式(8.39)中方括号内各项之和得出了标准阵中所需的、纠正所有*t*位错误线性组合的最小行数。不等式给出了监督比特的数目（或个陪集的数目）的下届是码的*t*位纠错能力的函数。类似地，不等式可以描述成*t*位纠错能力上界是个监督比特（或个陪集）的函数。对于任意一个能提供*t*位纠错能力的线性分组码，满足汉明界限是一个必要条件。  为了说明标准阵列是怎样提供这个界限的视图的，不妨选择（127,106）BCH码作为一个例子。阵列包含了空间中的所有个*n*元组。阵列最上面的一行包含了个码字；因此，这就是阵列的列数。最左边的一列包含了个陪集首，因此，这就是阵列的行数。尽管*n*元组和码字的数目巨大，但我们并不关心每一个单独的条目，主要关心的是陪集的数目；共有2097152个陪集，因此，这种码最多只能纠正2097152个错误图样。另外，它显示了陪集的数目是怎样决定码的*t*位纠错能力的上界的。  由于每个码字都包含127比特，因此有127种方式形成单个错误。可以计算出构成两个错误有多少种情况，即。下面继续分析出3个错误的情况，因为到目前为止，总共2097151个可纠正的错误图样中，只使用了很少一部分。一共有种方式可以组成3个错误。表8.4中列出了这些计算结果，说明全0错误图样要求第一个陪集的出现，其后是单错、双错和3个错误。表中同样说明了每种错误类型所需要的陪集数和到该种错误类型为止所需要的累积的陪集数。从该表中可以看出。（127,106）码可以纠正所有的单错、双错、三错图样，而这些只需要所有可用的2097152个陪集中的341504种。没有使用的1755648行表明了更多位纠错的可能性。实际上，可以试图把所有的4错图样映射到阵列中，但是由表8.4可以看出，这是不可能的，因为正如表中最后一行显示的那样，阵列中剩下的陪集数比纠正4错图样所需的累积陪集数小的多。因此，对于这个（127,106）的例子，该码的汉明界限保证了可以纠正小于等于3个错误的所有图样。  表 8.4 （127,106）码的纠错界限   |  |  |  | | --- | --- | --- | | 错误的比特数 | 需要的陪集数 | 累积需要的陪集数 | | 0 | 1 | 1 | | 1 | 127 | 128 | | 2 | 8001 | 8129 | | 3 | 333375 | 341504 | | 4 | 10334625 | 10676129 |   8.5.2 (*n,k*)码的一个例子  标准阵列提供了在纠错和检错之间权衡的可能性。考虑另一个码的例子，并考虑影响取值的因素。   1. 为了使在纠错和检错之间的权衡有意义，编码的纠错能力至少应为*t*=2。寻找最小距离如下：。 2. 对于有意义的编码系统，数据比特的数目至少为*k*=2，这样，有个码字，现在编码可以设计成（*n*，2）码。 3. 寻找最小的*n*值以允许纠正所有的单错和双错。在这个例子中，每一个阵列中的个*n*元组都将被列入表中。显然希望*n*的值最小，因为*n*值每增加1，标准阵列中*n*元组的数目都将增加1倍。当然，还希望表的大小易于管理。对于实际应用中的编码，希望*n*最小是鉴于其他的原因——频带利用率和简洁性。如果根据汉明界限选择*n*，那么将会选择*n*=7。但这样的（7,2）码的尺寸并不能满足*t*=2的纠错能力和的要求。因此，有必要引入*t*位纠错能力（或）的另一个上界。这个上界，称为普洛特金界线（Plotkin bound）：   （8.40）  一般而言，一个线性码必须满足与纠错能力或最小距离有关的所有上界。对于高码率编码，如果满足汉明界限，则同样能满足普洛特金界线；这种情况可见于前面（127,106）码的例子。而对于低码率编码，就是另一种情况了。因为此例要求低码率编码，因此，根据普洛特金界线检验其纠错能力就显得很重要。因为，由式（8.40）可以看出，*n*必须等于8，因此，码的最小尺寸是（8,2）才能满足要求。会有人使用诸如（8,2）码这样的低码率纠2比特错误的编码吗？没有。相比于那些码率更高的编码，这种编码将带宽扩展很多。在这里使用这种编码只是为了教学的目的，因为其标准阵列的大小易于管理。  8.5.3 （8,2）码的设计  也许有人会问这样的问题：怎样从有28个8元组的码字空间中选择码字？有多种解决方法，但是选择时是有限制的。下面是解决这个问题时需要考虑的一些要素：   1. 码字的个数为； 2. 全0矢量必须是其中的一个码字； 3. 必须满足封闭性。这个特性要求空间中两个码字的和必须是空间中另一个有效的码字； 4. 每个码字的长度都是8比特； 5. 因为，每个码字的重量（除去全0码）至少为5（由封闭性原理可得），矢量的重量定义为矢量中的非0元素的数量； 6. 假定是系统码，因而这种码的最右两比特对应于信息比特。   下面是对消息的一种码字分配候选方案，它满足前面所有的条件：   |  |  | | --- | --- | | 消息 | 码字 | | 00 | 00000000 | | 01 | 11110001 | | 10 | 00111110 | | 11 | 11001111 |   8.5.4检错和纠错的权衡  对于前一小节的（8,2）码系统，生成矩阵可以写为    从计算伴随式开始进行译码，这可以认为是了解错误的“症状”。对于码，一个比特的伴随式S是*n*比特的接收矢量***r***和一个的监督矩阵***H***转置的乘积。这里，***H***的行正交于***G***的行，即。对于这个（8,2）码的例子，***S***是一个6比特矢量，而***H***是一个的矩阵，即    对于每个错误图样，可以通过式（6.37）计算伴随式，即    其中，是个伴随式之一，是标准阵列中64个陪集首（错误图样）之一。图8.9以表格形式列出了标准阵列和（8,2）码的所有64个伴随式。伴随式的集合由式（8.26）计算而得。标准阵列的每一行（陪集）的元素都有相同的伴随式。对受到干扰的码字的纠正是通过计算它的伴随式找出其对应的错误图样。随后错误图样和受到干扰的码字模2加，生成正确的输出。纠错能力和检错能力是可以权衡的，只要距离关系  成立；其中，代表了需要纠正的错误比特数，代表了需要检测的错误比特数，。对于（8,2）码的例子，可以采用的权衡选择如下：   |  |  | | --- | --- | | 检错 | 纠错 | | 2 | 2 | | 3 | 1 | | 4 | 0 |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 伴随式 |  |  | 标准阵列 | |  | | 000000 | 1 | 00000000 | 11110001 | 00111110 | 11001111 | | 111100 | 2 | 00000001 | 11110000 | 00111111 | 11001110 | | 001111 | 3 | 00000010 | 11110011 | 00111100 | 11001101 | | 000001 | 4 | 00000100 | 11110101 | 00111010 | 11001011 | | 000010 | 5 | 00001000 | 11111001 | 00110110 | 11000111 | | 000100 | 6 | 00010000 | 11100001 | 00101110 | 11011111 | | 001000 | 7 | 00100000 | 11010001 | 00011110 | 11101111 | | 010000 | 8 | 01000000 | 10110001 | 0111110 | 10001111 | | 100000 | 9 | 10000000 | 01110001 | 10111110 | 01001111 | | 110011 | 10 | 00000011 | 11110010 | 00111101 | 11001100 | | 111101 | 11 | 00000101 | 11110100 | 00111011 | 11001010 | | 111110 | 12 | 00001001 | 11111000 | 00110111 | 11000110 | | 111000 | 13 | 00010001 | 11100000 | 00101111 | 11011110 | | 110100 | 14 | 00100001 | 11010000 | 00011111 | 11101110 | | 101100 | 15 | 01000001 | 10110000 | 01111111 | 10001110 | | 011100 | 16 | 10000001 | 01110000 | 10111111 | 01001110 | | 001110 | 17 | 00000110 | 11110111 | 00111000 | 11001001 | | 001101 | 18 | 00001010 | 11111011 | 00110100 | 11000101 | | 001011 | 19 | 00010010 | 11100011 | 00101100 | 11011101 | | 000111 | 20 | 00100010 | 11010011 | 00011100 | 11101101 | | 011111 | 21 | 01000010 | 10110011 | 01111100 | 10001101 | | 101111 | 22 | 10000010 | 01110011 | 10111100 | 01001101 | | 000011 | 23 | 00001100 | 11111101 | 00110010 | 11000011 | | 000101 | 24 | 00010100 | 11100101 | 00101010 | 11011011 | | 001001 | 25 | 00100100 | 11010101 | 00011010 | 11101011 | | 010001 | 26 | 01000100 | 10110101 | 01111010 | 10001011 | | 100001 | 27 | 10000100 | 01110101 | 10111010 | 01001011 | | 000110 | 28 | 00011000 | 11101111 | 00100110 | 11010111 | | 001010 | 29 | 00101000 | 11011001 | 00010110 | 11100111 | | 010010 | 30 | 01001000 | 10111001 | 01110110 | 10000111 | | 100010 | 31 | 10001000 | 01111001 | 10110110 | 01000111 | | 001100 | 32 | 00110000 | 11000001 | 00001110 | 11111111 | | 010100 | 33 | 01010000 | 10100001 | 01101110 | 10011111 | | 100100 | 34 | 10010000 | 01100001 | 10101110 | 01011111 | | 011000 | 35 | 01100000 | 10010001 | 01011110 | 10101111 | | 101000 | 36 | 10100000 | 01010001 | 10011110 | 01101111 | | 110000 | 37 | 11000000 | 00110001 | 11111110 | 00001111 | | 110010 | 38 | 00000111 | 11100010 | 00111001 | 11101000 | | 110111 | 39 | 00010011 | 11100010 | 00101101 | 11011100 | | 111011 | 40 | 00100011 | 11010010 | 00011101 | 11101100 | | 100011 | 41 | 01000011 | 10110010 | 01111101 | 10001100 | | 010011 | 42 | 10000011 | 01110010 | 10111101 | 01001100 | | 111111 | 43 | 00001101 | 11111100 | 00110011 | 11000010 | | 111001 | 44 | 00010101 | 11100100 | 00101011 | 11011010 | | 110101 | 45 | 00100101 | 11010100 | 00011011 | 11101010 | | 101101 | 46 | 01000101 | 10110100 | 01111011 | 10001010 | | 011101 | 47 | 10000101 | 01110100 | 10111011 | 01001010 | | 011110 | 48 | 01000110 | 10110111 | 01111000 | 10001001 | | 101110 | 49 | 10000110 | 01110111 | 10111000 | 01001001 | | 100101 | 50 | 10010100 | 01100101 | 10101010 | 01011011 | | 011001 | 51 | 01100100 | 10010101 | 01011010 | 10101011 | | 110001 | 52 | 11000100 | 00110101 | 11111010 | 00001011 | | 011010 | 53 | 01101000 | 10011001 | 01010110 | 10100111 | | 010110 | 54 | 01011000 | 10101001 | 01100110 | 10010111 | | 100110 | 55 | 10011000 | 01101001 | 10100110 | 01010111 | | 101010 | 56 | 10101000 | 01011001 | 10010110 | 01100111 | | 101001 | 57 | 10100100 | 01010101 | 10011010 | 01101011 | | 100111 | 58 | 10100010 | 01010011 | 10011100 | 01101101 | | 010111 | 59 | 01100010 | 10010011 | 01011100 | 10101101 | | 010101 | 60 | 01010100 | 10100101 | 01101010 | 10011011 | | 011011 | 61 | 01010010 | 10100011 | 01101100 | 10011101 | | 110110 | 62 | 00101001 | 11011000 | 00010111 | 11100110 | | 111010 | 63 | 00011001 | 11101000 | 00100111 | 11010110 | | 101011 | 64 | 10010010 | 01100011 | 10101100 | 01011101 |   图8.9 标准阵列和（8,2）码的所有64个伴随式  该表显示了（8,2）码可以只用来实现纠错，这意味着它首先检测到多达个错误，然后纠正它们。如果减少纠错能力使码只能纠一个错误，那么，检错能力上升到可以检测到个错误。最后，如果完全放弃纠错能力，那么译码器可以设计成能检测到所有个错误。在只检错的情况下，电路十分简单。计算伴随式，当只要有一个伴随式非零时，即检测出有错。  对于纠正单个错误，译码器可以通过门电路实现，类似于图8.6的电路，接收矢量进入两个位置。在图的上半部分，接收矢量进入异或门，产生伴随式。对于所有给定的接收矢量，伴随式可以通过式（8.24）获得    根据（8,2）码的的值，接收矢量和异或门之间的连线电路类似于图8.6中的那一个，连接后可以得到    组成伴随式的每一个数字按下列方式与输入端接收的码字矢量相关联：    要实现类似于图8.6中的（8,2）码的译码器电路，需要将8个接收数字与上面描述的6个模2加法器产生的伴随式数字联系起来。相应地，需要对此图做额外的修改。  如果译码器只要求纠正单个错误，即且，那么，这等于在图8.9的陪集9下面划一条线，只有当与单个错误相对应的8个伴随式出现时才进行纠错。然后译码电路（与图8.6类似）将伴随式变换为它对应的错误图样。随后，错误图样与可能受到干扰的的接收矢量模2相加，产生正确的结果。另外需要一些附加的门在产生了非零伴随式，但是并没有设计其对应的纠正时作检测使用。对于纠正单错的情况，这种事件发生于任何一个序号在10到64之间的伴随式。其结果用来指示检测到了错误。  如果译码器用于纠正所有的单错和双错，这意味着的错误被检测并被纠正，这等于在图8.9标准阵列的陪集37下划一条线。尽管这个（8,2）码有能力检测到一些三错的组合，这时对应于38到64的陪集首，但我们通常把译码器设计成界限距离译码器，这意味着它能纠正所有少于或等于*t*个错误组合，却不能纠正多于*t*个错误的组合。没有使用的纠错能力被用来增强一些检测能力。与前面的一样，译码器可以类似于图8.6，用门电路实现。  8.5.5标准阵列的进一步说明  在图8.9中，（8,2）码满足汉明界限。也就是说，从标准阵列中可知，（8,2）码可以纠正所有单、双错的组合。考虑下面的问题：如果在一个总是带来突发的3比特错码的信道中进行传输，这时纠正单、双错就没有意义。能否建立一个仅对应于3个错误的陪集首呢？很明显，8位的序列有种方式发生3个错误。如果只想纠正3个错误的56种组合，在标准阵列中就有足够的空间（足够的陪集数），因为阵列有64行。这样行吗？不行。因为对于任何码，纠错能力的决定性参数是。对于（8,2）码，决定了它只可能纠正2比特错误。  怎样通过标准阵列理解这种方式是不行的呢？为了使一组比特的错误图样能够进行比特的纠错，所有重量为的矢量组都必须是陪集首，即它们必须只占据最左边的一列。在图8.9中可以看到，所有重量为1和2的矢量都仅出现在标准阵列最左边的一列。即使把所有重量为3的矢量塞入2到57行中，也可以发现这些矢量将在阵列的其他地方重复出现（违反了标准阵列的基本属性）。在图8.9中，56个重量为3的矢量中的每一个都画了阴影区。观察阵列中第38行，41~43行，46~49行和52行中代表3比特错误图样陪集首，再看一看最后一列中相同行的内容，同样是阴影，代表了另外一个重量为3的矢量，就会发现上述各行中存在模糊性问题，以及不能用（8,2）码纠正所有的3比特错误图样的原因。假定译码器收到重量为3的矢量11001111而受到了错误图样00000111的干扰；另一种则可能是发送码字00000000而受到了错误图样11001000的干扰。  8.4 检错和纠错能力  8.4.1二进制矢量的重量和距离  必须明确，并不是所有的错误图样都能够正确译码。一个码的纠错能力必须首先通过定义它的结构来考察。一个码字*U*的汉明重量（Hamming weight）定义为*U*中非0元素的数目。对于一个二进制矢量，这相当于矢量中1的个数。举例来说，如果*U*=100101101，那么。两个码字*U*和*V*的汉明距离（Hamming distance）标记为，定义为它们之间不同元素的数目，例如    根据模2加的属性，将两个二进制矢量的和记为另一个矢量，该矢量的数字1存在于两个矢量元素不同的那些位置上，例如    这样，观察两个码字间的汉明距离等价于这两个码字和的汉明重量，即。同样，一个码字的汉明重量等于它与全零矢量间的汉明距离。  8.4.2 线性码的最小距离  考虑空间中所有码字对之间的距离的集合。集合中最小的元素就是码的最小距离，标记为。为什么我们对最小距离感兴趣，而不是最大距离？因为最小距离如同链条中最弱的一环，能够衡量码的最小能力，从而刻画出编码的能力。  如前所述，两个码字的和产生了子空间中的另一个码字元素。线性码的这一特性可以简单地描述为：如果*U*和*V*都是码字，那么也是码字。因为两个码字之间的距离等于第三个码字的重量：即，这样，线性码的最小距离就可以不通过检查码字对的距离也可以求出。实际上只需要检查在子空间中每个码字的重量（全零码字除外）；最小的重量对应于最小的距离。  8.4.3检错和纠错  译码器的任务是，在接收到矢量***r***后，估计出传输的码字。最佳的译码器策略可以用最大似然（maximum likelihood）算法表示如下：如果  （8.30）  则判决为。因为在二进制对称信道（BSC）中，*r*是的可能性，与*r*到之间的距离成反比，可以写为：如果  （8.31）  则判决为。换句话说，译码器计算*r*与每一个可能的传输码字的距离并选出最可能的满足下式的：  （8.32）  其中，是码字集合的大小。如果最小距离不是唯一的，那么任意选择最小距离码字中的一个。  在图8.7中，两个码字*U*和*V*的距离表现为一条由汉明距离作刻度的数字线。每一个黑点代表一个受到干扰的码字。图8.7a中接收矢量，它与*U*的距离为1而与*V*的距离为4。采用最大似然准则的纠错译码器，在收到时将选择*U*。如果是传输矢量*U*被1比特干扰的结果，那么纠错是正确的。但是如果是传输矢量***V***受到4比特干扰的结果，那么译码就是错误的。类似地，*U*在传输中出2个错的结果是，它与*U*的距离为2而与*V*的距离为3，如图8.7b所示，这里，译码器也将在收到时选择*U*。如果*U*在传输时发生3个错误，则将导致接收到，与Ｕ的距离为３而与*V*的距离为２，如图8.7c中所示，这样译码器将在收到时选择*V*，从而在译码时出错。从图8.7可以很清楚地看出，如果任务是检错而不用纠错，一个受干扰的码字（用黑点表示，代表1比特、2比特、3比特、4比特错误）可以被检测到。不过，出5比特错时，在传*U*的时候会接收到*V*；这样的差错是不可检测的。    图8.7 纠错和检错能力  从图8.7可以看出，码的检错和纠错能力是由码字间的最小距离决定的。图中的判决线在译码和解调过程中起相同的作用，即定义判决区间。在图8.7的例子中，如果*r*落入区域1就选择*U*，落入区域2就选择*V*，这种判决标准表明，对于这样的码(=5)可以纠正2比特错误。一般说来，码的纠错能力*t*是由每个码字确保的最大纠错个数来定义的，即  （8.33）  其中，代表不超过*x*的最大整数。通常，一个可以纠正所有个或少于个错误的序列的编码，也可以纠正一些特定的个错误的序列，这可以从图8.5中看出。在这个例子中=3，这样由式(8.33)看到，所有*t*=1比特的错误图样都是可以纠正的。同样，一个*t*+1即2比特的错误图样也是可纠正的。一般说米，一个*t*比特纠错()线性码有能力纠正总共2*n*-*k*个错误图样。如果一个比特纠错分组码被严格地用来在一个转移错误概率为*p*的二进制对称信道( BSC )中纠错，消息错误的概率*P*M即译码器做出错误译码，*N*位分组出错的概率，可以得出上界:  (8.34)  只有当译码器能够纠正所有小于等*t*的错误组合，而不能纠正多于*t*个错误组合的时候，这个界限才取等号。这种译码器称为界限距离译码器( bounded distance decoder )。译码的误比特率*P*B取决于特定的码本和译码器，它可以通过下式近似表示:  (8.35)  一个分组码需要在纠错之前先检错,或者它可能只用于检错。从图8.7中可以看出，所有标了黑点的接收矢量(受到干扰的码字)都可以判为有错。这样，检错的能用的形式定叉就是  (8.36)  最小码间距离为的分组码，可以保证所有具有-1或更少错误的图样能够检测到。这样的码也能检测出大部分有或更多错误的图样。实际上，一个(*n*，*k*)码有能力检测2*n*-2*k*个长度为*n*的错误图样。其原因如下:对于2*n*个可能的*n*元组空间，共有2*n*-1种可能的错误图样。即便是一个有效的码字也可能是潜在的错误图样。这样2*k*-1个错误图样就与2*k*-1个非零码字相同，如果这2*k*-1个错误图样中任何一个出现，它将传输的码字转变为另一个码字。这样，将被接收而它的伴随式是0，译码器接收作为传输的码字，于是做了错误的译码。因此，有2*k*-1种不能检测的错误图样。如果错误图样不同于2*k*个码字中的任一个，那么，伴随式检测对于接收矢量会产生一个非0伴随式，从而检出错误。因此，一共有恰好2*n*-2*k*种可检测的错误图样。对于大的*n*，有2*k*<2*n*，只有很小一部分的错误图样不能检测。  1. 码字重量分布  令为(*n*，*k*)线性码中重量为*j*的码字的数目，那么A0，A1，...，A*n*称为码的重量分布（weight distribution）。如果在二进制对称信道上编码只用于错误检测，则译码器不能检测到错误的概率可以通过码的重量分布求出：  （8.37）  其中，*p*是二进制对称信道的传输概率。如果码的最小距离是，则从到的值都是0。  【**例8.6**】 检错码中出现不能检测错误的概率  考虑前面给出的（6,3）码，只用于检错。计算信道是二进制对称信道且转移概率为时，一个错误不能被检测到的概率。  解：该码的重量分布为，，，，，。因此，根据式（8.37），可得    对于，一个错误不能检测出的概率是。  2.同时纠错和检错  可以将由式（8.33）给出的最大可保证的纠错能力（*t*）降低，从而同时能够检测一些错误。如果最小码距满足下述条件，则这种码可以同时纠正个错，并检测个错（其中）：  （8.38）  当有*t*个或更少的错误出现时，码有能力检测并纠正之。当多于*t*但少于个错误出现[其中，e当由式（8.36）定义]时，码有能力检测出错码的存在，但只能纠正它们的一个子集。举例来说，一个的码可以采用以下的任一种方式同时检错和纠错：   |  |  | | --- | --- | | 检错 | 纠错 | | 3 | 3 | | 4 | 2 | | 5 | 1 | | 6 | 0 |   注意，纠错意味着先检错。对于上面的例子，当有3个错时，它们都能被检测出来并被纠正。当有5个错时，它们都能被检测，但只有其中的子集（1个）可被纠正。  8.4.4 6元组空间的视图  图8.8是前面的例子中8个码字的视图。码字组由式（8.16）中三个独立的6元组经线性组合而成。这些码字构成了一个三维子空间。此图表示了一个由8个码字完全占据的（大黑圆圈）的子空间；子空间的坐标特意被画出以强调其并非正交。尽管并没有精确的方法来画出或构建这样的模型，但是图8.8仍试图说明一个包含64个6元组的整个空间。球形的层或者壳围绕着每个码字。每一个不相交的内层与它对应的码字间的汉明距离为1，每一个外层与它对应的码明距离为2。此例中没有使用更大的距离。对于每一个码字，画出的两层被受到干扰的码字占据。每个内层都有6个这样的点，（总共有48个点），代表了对应于各个码字的6个可能的1比特干扰错误矢量。这些1比特干扰的码字仅与一个码字呈现最佳关联，从这个意义上说，它们可以互相区分，因此可以被纠正。正如从图8.5中看到的标准阵列，只有一个2比特错误图样可以纠正。总共有种不同的2比特错误图样可以加于每个码字上，但其中只有1个可以被纠正（在我们的例子中是010001错误图样）。其他的14种2比特错误图样产生的矢量，难以被唯一地鉴别为单一的码字；这些不可纠正的错误图样产生的矢量，与两个或更多码字受到错误干扰的矢量相同。在图中，所有可纠正的（56）1比特或2比特错误干扰的码字都被画成小黑圈，受到干扰而不能纠正的码字被画成白圈。    图8.8 6元组空间中的8个码字示例  图8.8所示的一类码称为完备码（perfect code），图中描述了这种码的性质。如果一个*t*比特纠错码的标准阵列包括了所有*t*位或更少位错误的错误图样，且不含有其他的陪集首（没有剩余的纠错能力），它就称为完备码。根据图8.8，一个*t*位纠错的完备码，采用最大似然译码时，能够纠正所有与原码字的汉明距离为*t*或小于*t*的干扰码字，而不能纠正与原码字距离大于*t*的码字。  图8.8对于理解寻找良好性能编码的目标也是十分有用的。我们希望空间中存在尽可能多的码字（有效利用增加的冗余度），但同时也希望码字间的距离尽可能大。显然，这些目标是互相矛盾的。 | | 陪集的数目是怎样决定码*t*位纠错能力的上界？  这是典型设计案例，通过案例能理解影响码设计的因素及设计过程  讨论码纠检错能力与最小距离的关系 |
| **小结** | 利用标准阵设计线性分组码；最小距离与纠检错能力的关系 | |
| **复习要点** | 能够设计线性分组码 | |
| **思考题** | 最小距离与纠检错能力是怎样的关系？ | |
| **作业题** | 8.9 | |

作者签名：

****