**信道可靠度处理方法**

1. **信道可靠度矩阵**

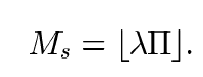
信道可靠度矩阵中，行表示有限域符号，行数为有限域大小。列表示接收符号，列数为帧长。矩阵中的每一个元素的意义如下：

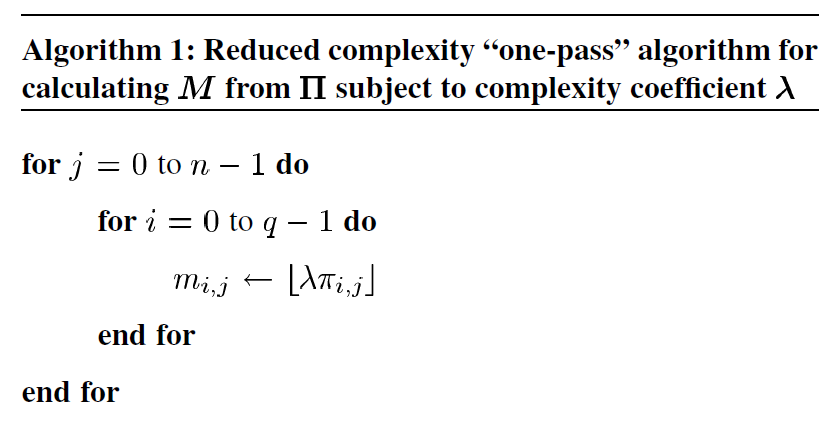


此即后验概率，意义即我的接收端观察到时，发送端发送的确实是的概率。注意，这里接收端观察到的，不一定指的是硬判决后得到的有限域符号，也有可能是解调软信息，或ADC采样，甚至是RF前端的信息。总之，接收通路上的所有信息都可以利用起来作为。

关于软信息的计算理论与实例，可以参考Mostafa Hosni Mohamed在Ulm大学的博士论文Algebraic decoding over finite and complex fields using reliability information第4.1节。

1. **重数分配问题**
2. 若是KV算法的greedy分配，可用直接对可靠度矩阵乘上1个常数来分配：



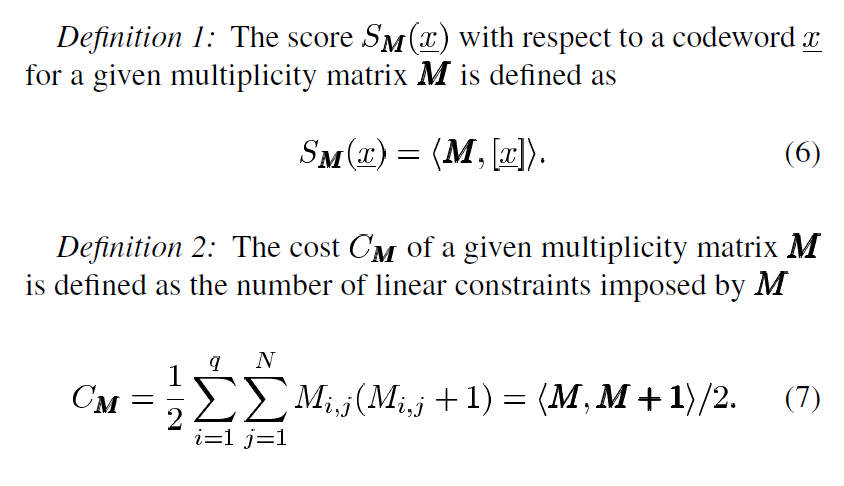


这种方法可以解决计算对重数分配时的计算复杂度问题，出信道可靠度的时候可以顺带把重数也给算出来。

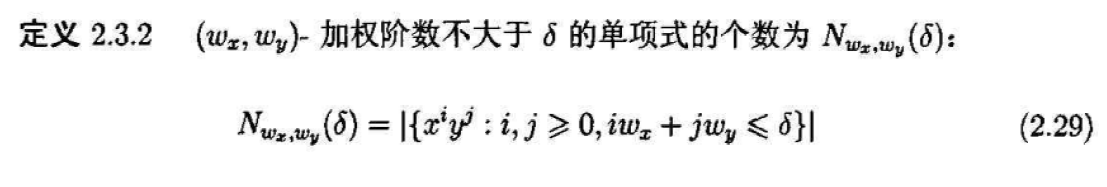
重数矩阵的存储方式也可以做文章，如只存有重数的位置。

比特级别擦除或错误(翻转)信息的利用可以参考Jiang, Jing, and Krishna R. Narayanan. "Algebraic soft-decision decoding of Reed–Solomon codes using bit-level soft information." IEEE transactions on information theory 54.9 (2008): 3907-3928. 此文提出了利用比特级别信息的重数分配策略，并可应用于同时具有比特擦除和比特错误的信道。

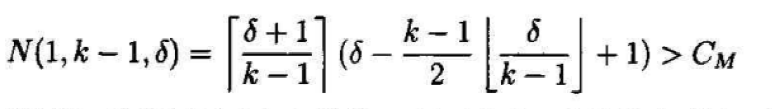
以下有重要概念：



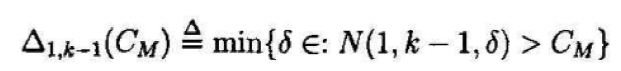
加权阶数的上限：



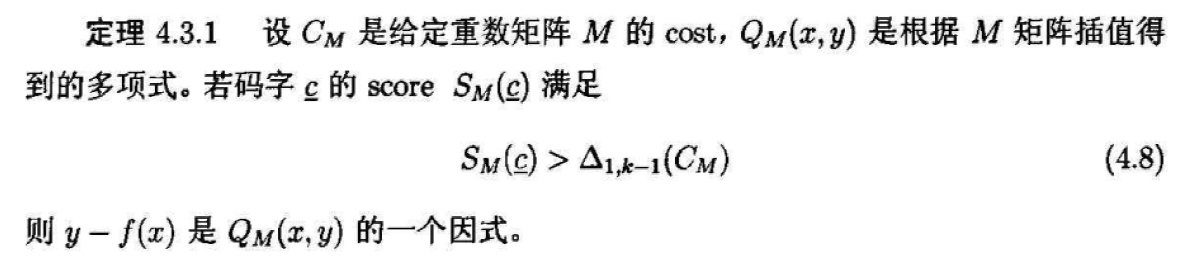
插值问题中满足C个重数约束条件时，需要使得：



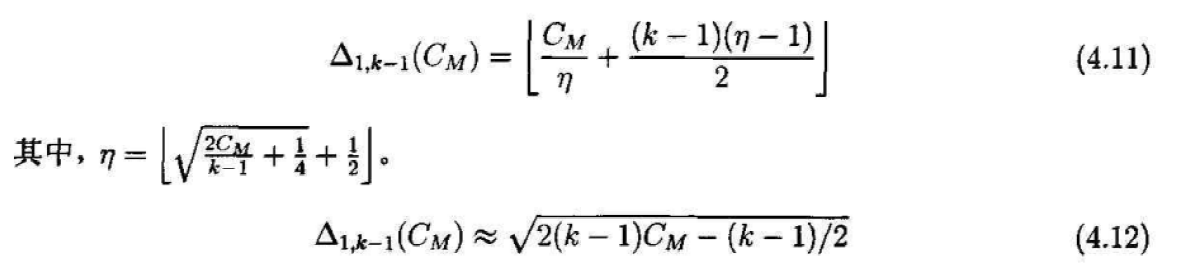
为了要求加权阶数尽可能小（插值复杂度就越小），则有：



则译码成功的条件为：



最小的加权阶数上限的界则为：

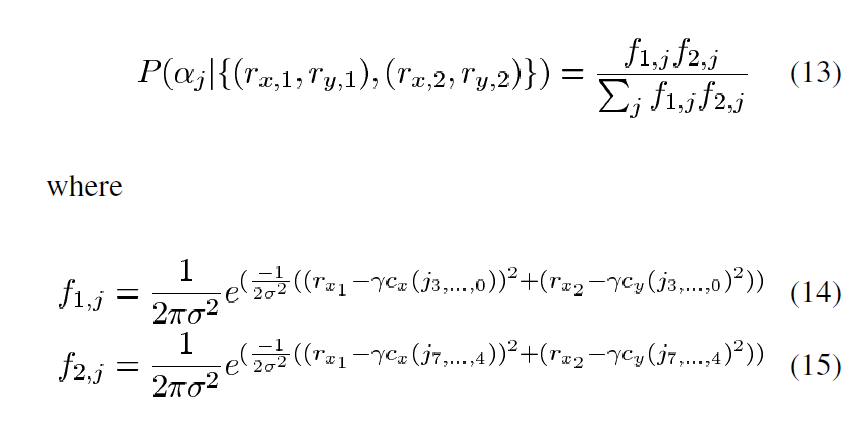


实际上，译码失败的概率则为，重数分配策略实际上就是在最小化此概率，优化的约束条件则是cost。

详细描述可以参考黄勤在东南时的硕士论文《Reed-Solomon码软译码技术研究》，里面有对若干文献的总结。

1. **量化问题**

信道软信息可以只存储部分解调信息即可，如当有限域大小与调制阶数匹配时，可以直接由解调后存储与解调信号欧式距离最小（或者多存一两个）调制符号的距离。如果不匹配时，计算稍微复杂些，且如果是衰落信道，还要考虑CSI、交织器、衰落快慢的影响。如果是完美CSI、理想交织器、慢衰落（不在一个符号内时变）的话，用GF(16)+256-QAM时，计算方法如下：



上述计算方法与阐述可以参考Gross, Warren J., et al. "Applications of algebraic soft-decision decoding of Reed-Solomon codes." IEEE transactions on communications 54.7 (2006): 1224-1234.

理想交织器体现在衰落的作用是独立、无记忆的，详细分析可参考Divsalar, Dariush, and Marvin K. Simon. "The design of trellis coded MPSK for fading channels: Performance criteria." IEEE transactions on Communications 36.9 (1988): 1004-1012.

注意，在有完美CSI信息时，软译码算法能获得的增益是很好的，详见KV原版论文的第7节末尾。

如果时强度调制的，且是LCC，可以直接看接受符号的绝对值来判断（系统实现时可能要考虑DC offset），参考Peng, Xingru, et al. "Reduced-complexity multiplicity assignment algorithm and architecture for low-complexity Chase decoder of Reed-Solomon codes." IEEE Communications Letters 19.11 (2015): 1865-1868.。

量化问题在一些论文中的处理方法：

Koetter, Ralf, and Alexander Vardy. "Algebraic soft-decision decoding of Reed-Solomon codes." IEEE Transactions on Information Theory 49.11 (2003): 2809-2825. 仿真时用8比特，未说明量化误差影响。

Gross, Warren J., et al. "Applications of algebraic soft-decision decoding of Reed-Solomon codes." IEEE transactions on communications 54.7 (2006): 1224-1234. 仿真用8比特，分析重数分配策略时说到量化比特用于说明one-pass分配方法是否等同于KV原版greedy方法。未说明量化误差影响。

Ratnakar, Niranjan, and Ralf Koetter. "Exponential error bounds for algebraic soft-decision decoding of Reed-Solomon codes." IEEE transactions on information theory 51.11 (2005): 3899-3917. 真时用8比特，未说明量化误差影响。

总结量化问题的解决方案：

1. 理论上可以分析信道软信息量化误差对译码错误概率的影响。大概思路是在特定的重数分配策略下，分析量化误差对重数分配结果的影响，由此结果是如何影响到重数分配score的。当然，是否有可能在考虑量化误差的情况下，改进重数分配策略，这可能是可以研究的问题。
2. 工程上，在特定的重数分配策略下，评估量化影响的最快速的方法是直接仿真。直接处理一下软信息矩阵即可，几乎没有代码工作量。
3. 量化问题在实际工程上，除了影响存储，实际上还会影响到下一节所述的排序问题。
4. **排序问题**

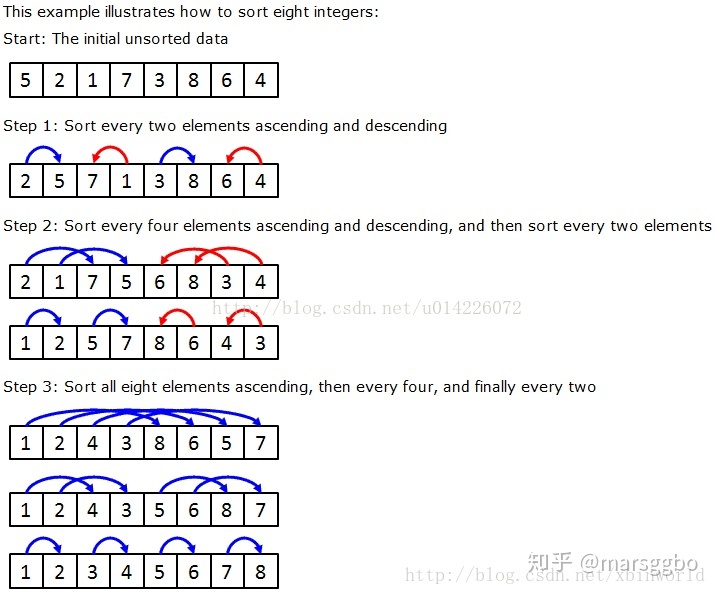
排序不需要全排序。

排序可以在通路上处理。

有两种看待这个问题的方法，一是将信道可靠度的获得、处理放到整个接收机系统中去考虑，与接收机的其他模块有耦合、联动，再将处理结果（重数分配及可靠度分组）送给译码器；二是一种糟糕的情况，上游模块确实存储了一个q×n的可靠度矩阵，这时，常用做法是对n列各自做选最大与次大可靠度的项，这时，可以结合量化做阈值mask，可全并行输出结果，也可输出最大值（容易做到）后，用一边遍历、一边计算的方法（可以直接做差替代除法）来获得次大值并同时判断其符号可靠度。

排序问题的解决方案总结如下：

1. 此处的排序并非全排序。排序的实际目的是为了分配重数或是划分可靠符号组与不可靠符号组。
2. 排序可以不在得到整个信道可靠度矩阵之后再进行。实际上，你有以下几种选择来处理此事：
3. 在解调的输出直接处理此事。无论你是要直接分配重数（如Gross策略那样乘以一个常数、或者如bit-level策略那样根据错了多少比特来做选择）、还是要看这个符号可靠与否，你在解调输出端其实就可以直接做判断了。这时，你可以利用解调输出端给你的解调软信息（最直接的是看星座点距离标准符号的欧式距离，与EVM的概率类似），如果你想利用更多额外信息，你还可以将更多上游模块的信息传过来做参考。
4. 你也可以在传输比特的过程中处理此事。传输无非有两个可能的目的：传给下游模块（译码器）处理、或者写进RAM或寄存器里面存起来。如果是串行地传，你大可在传输中途加电路做关卡，直接完成了遍历。如果是并行地传，这时候需要注意传输地顺序，如果是先传按行（先传n个符码字，共传q行），则可以在中途加关卡。如果是先按列传（先传q个符号，共传n列），则需要用加mask的方法来处理此事。
5. 顺序串行遍历时，提前退出的判断准则为最大值a > 0.5及次大值b > (1 - a) / 2，但实际上这种方法需要遍历两次（b的判断需要a，而a在顺序遍历时未必提前出现），你不能一次遍历就完成这件事情，会带来额外延时。
6. 做mask的方法，即分级比较阈值。如先全并行过0.5比较阈值，则最大值有可能就出来了，如果没有，就过0.4的，然后再过0.3的。如果仍然找不到，那么此码字可靠度应较低，可分配较低重数及划分至不可靠组。此方法搭配量化来做更好，你可以直接取量化后的比特做mask，而不用比较器，更经济实惠。此策略如果不做全并行、做部分并行也可以。
7. 常用的判断可靠度的方法是比较一个码字（列）内q个符号中，最大可靠度与次大可靠度的比值，这时，如果不想用除法，那么可以用对数减法。当然如果你做量化是做成了对数量化（有这种可能），那就操作起来太方便了，直接相减即可。如果不是对数量化，可以存个小型对数表，不建议使用直接除法或者用cordic算法求对数，时延和资源都不划算。
8. 在ML判决时，需要使用全排序（至少大部分排序），此时不得不使用双调排序，排序复杂度为O(nlog(n))（可做并行度n的计算），例子如下，简而言之，就是不断重复排序->合并成双调的过程：



From 12, 43, 56, 87 (aaaa)

From 14, 23, 85, 67 (aaaa)

From 18, 26, 54, 73 (aaaa)

From 21, 75, 68, 34 (aadd)

From 27, 51, 36, 84 (aadd)

From 52, 17, 38, 64 （adad）