CH2

对于许多工程问题，我们得不到解析解analytical solutions

解析解，是指通过严格的公式所求得的解。即包含分式、三角函数、指数、对数甚至无限级数等基本函数的解的形式。给出解的具体函数形式，从解的表达式中就可以算出任何对应值。用来求得解析解的方法称为解析法，解析法是常见的微积分技巧，如分离变量法等。解析解为一封闭形式的函数，因此对任一独立变量，皆可将其代入解析函数求得正确的相依变量。因此，解析解也称为闭式解。[1]

通过计算方法我们能得到approximate results近似解，我们无法精确计算这个 计算方法的错误

在现实中，很少有输入数据是精确地，因为他们都是通过测量得到的，会有误差

数学方法本身也会带来误差，比如unavoidable round-off，不可避免的舍入

近似解可以接受的范围取决于：我们可以容忍*tolerable的误差程度*

Accuracy 精确度：计算值或测量值 与正确值相似程度

Precision (reproducibility) 可重复性：这一次测量或计算与上一次测量或计算的相似程度

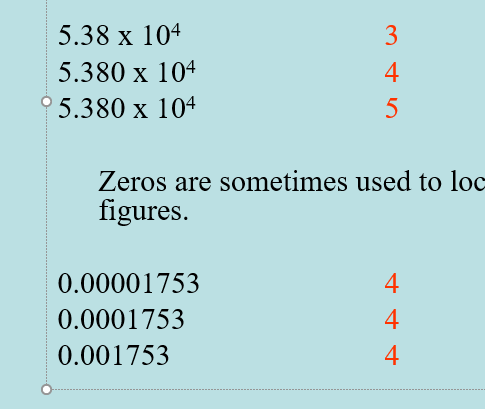
Inaccuracy（bias） 偏差：与实际值的系统偏差

Inprecision(uncertainty) 不确定性： 散射 的大小magnitude



Accuracy越大，越接近中心，precision越大，越靠近。

Significant figures有效数字



Error的定义

真实值=近似值+Error



真实分数相对误差



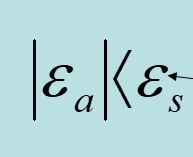
真实百分比相对误差

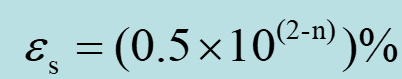
近似百分比相对误差

*Iterative approach 迭代接近法：*

**

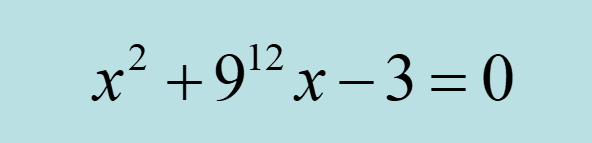
stopping criterion终止条件

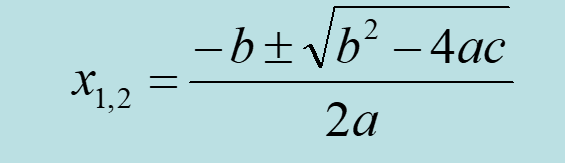
多次计算直到

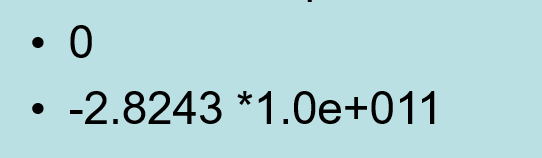
当达到了终止条件，，那么我们就可以确定答案是对的

**Round off error:**

比如用浮点法记录无限不循环小数pi时，会舍去最后几位，（去一个近似值），导致与实际不符







但实际上0代入，我们发现是错的，因为b^2相较于4AC来说实在太大了

概括地说：计算机可以存储的数位是有限的

由这个现象导致的error叫做round-off error

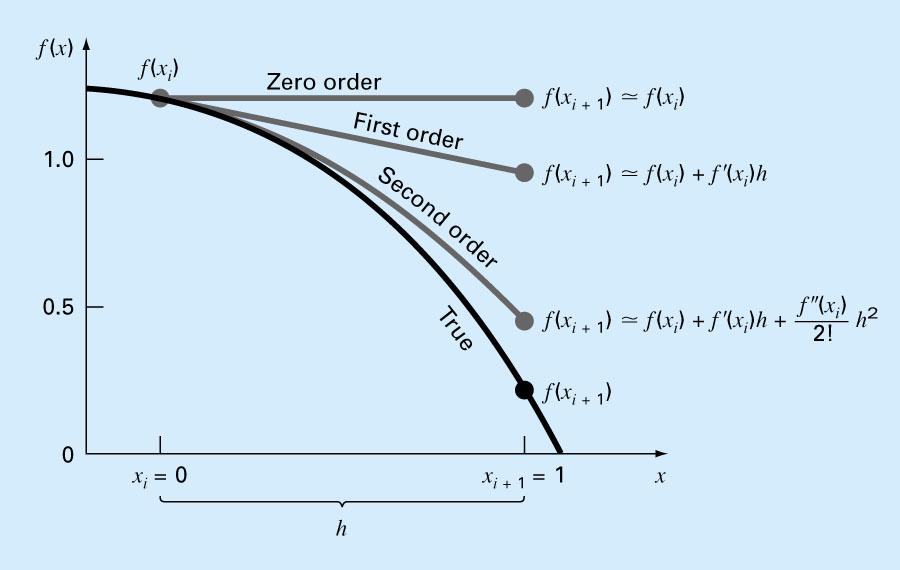
最据代表性的情况就是对最后几位数的丢失

Truncation error:我们用近似方法解决一个精确问题所产生的error

这个error经常存在于数学方法中

对于非初等函数比如说三角函数这种，都可以用taylor series泰勒级数来表示，当他们的值，导数，积分被计算完的时候

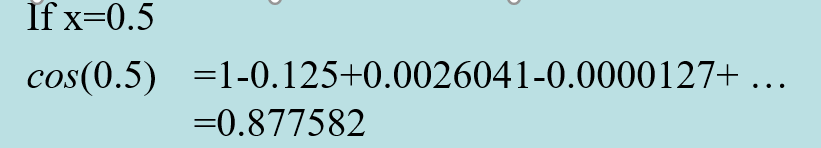
任意光滑曲线都可以表示成多项式，泰勒级数提供了一种方法，可以在某一时刻根据函数值和它的导数来预测一个函数的值。



H是横轴两个x之间的差值，f’(x)是负数

随着级数越来越多（多项式加的越来越细），我们所估计的f(x+1)会越来越接近真实值

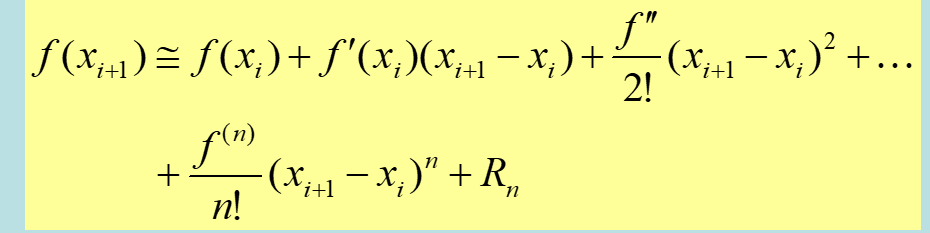
 这是cosx的具体表示

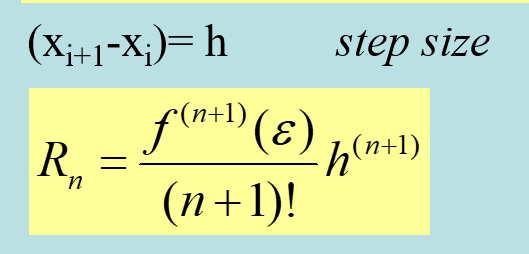


任意光滑曲线都可以表示成多项式

f(xi+1) ≈ f(xi) *zero order* approximation, only true if xi+1 and xi are very close to each other. 零级类似，只有当xi+1与xi非常相似的时候

f(xi+1) ≈ f(xi) + f′(xi) (xi+1-xi) *first order* approximation, in form of a straight line 一级近似，是一条直线





N级近似

我们无法确切的知道

如果我们们知道f(x)，就没必要用泰勒级数

泰勒级数展开的越高，truncation error越小

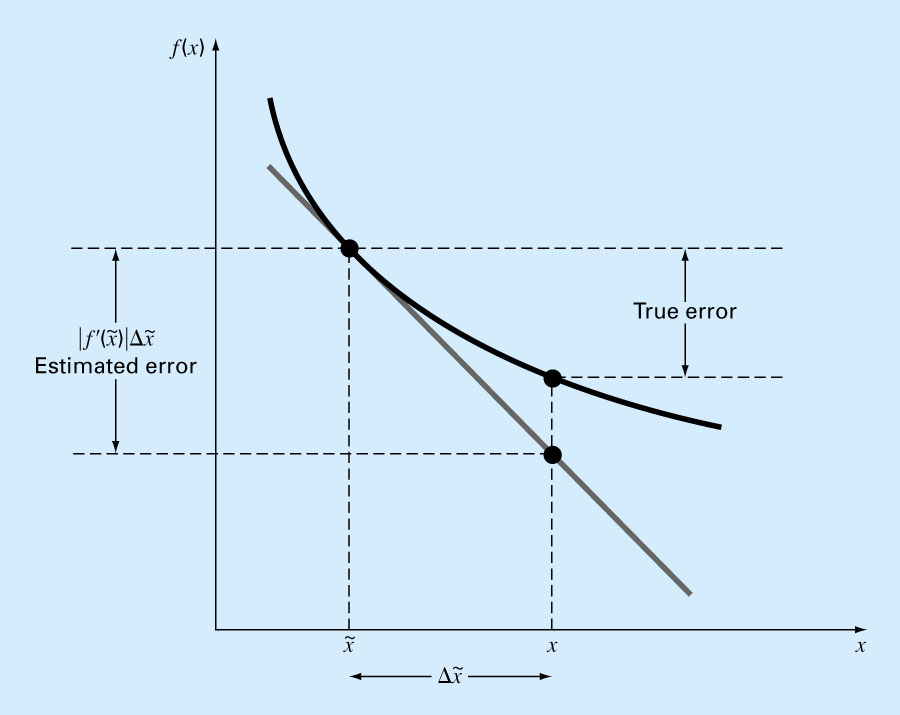
Error propagation

因为计算机存储的是有限浮点数，可能会导致round off error，当使用浮点数时，error 将会propagate传播

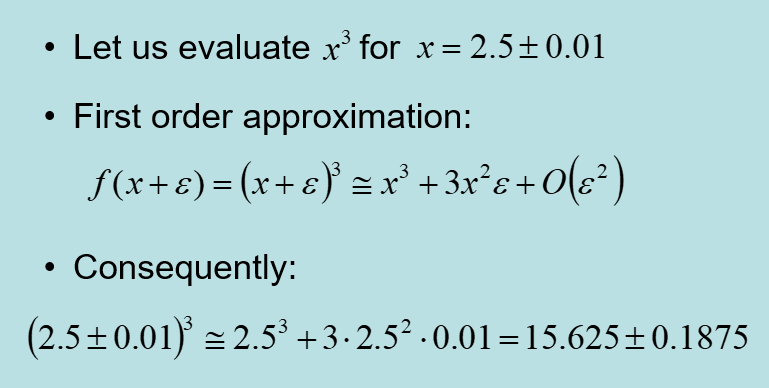
我们通过用近似法来估计这个错误

我们用泰勒级数来估计这个错误，只用一阶近似

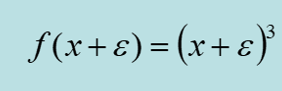
Error=这个函数的导数乘以两个X的差值



估计error=delta x\*导数



First order approxiamation步骤解释：

先得出F(X)

代入式子，=X^3(f(x))+3x^2(导数)\*E(h)

有多个变量的function

 =原函数+X偏导数\*DELTA X+Y偏导数 \*DELTA Y

Illustration

Numerical algorithms 数值算法有可能是1.unstable 2. stable but converging to a false value 3. Stable and converging to the correct value

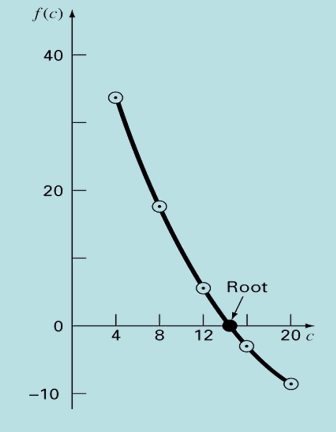
LESSON 3

Graphical methods

通过对将要解决的等式描点来估计出答案

缺陷：仅限于一维问题，不是很精确

优势：简单粗暴



Bracketing method交叉法

逐步减少根所在的a和b的间隔

优势：良好的控制error， 简单

劣势，慢，主要局限于一维

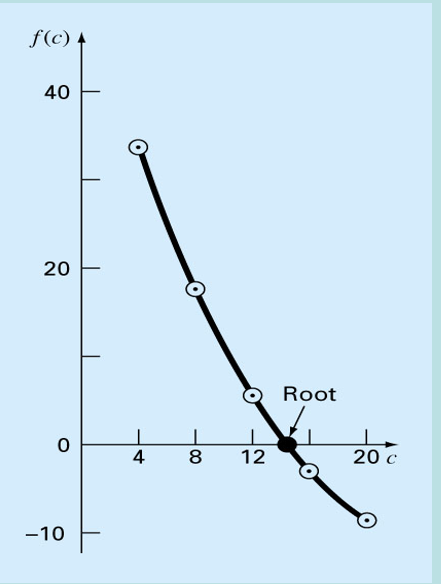
Open methods 开型法

猜测这个根并逐渐通过迭代重复增加这个猜测的精度

优势：快，不局限于1D，非常简单

劣势：不是很牢靠

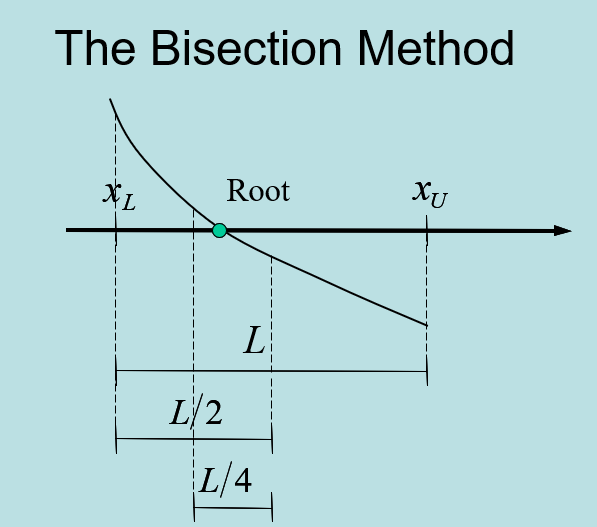
Bracketing methods: Bisection method对分法

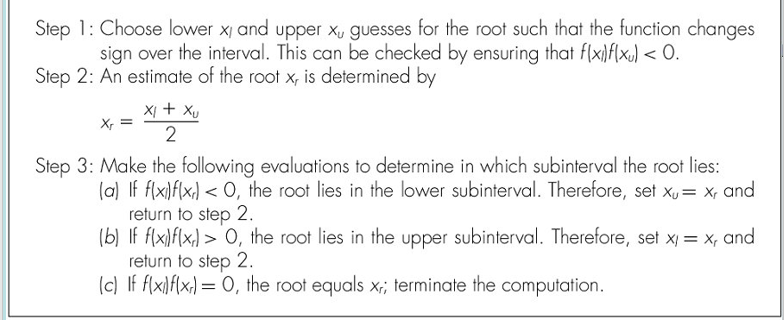


需要用两个初始猜测，这些猜测需要在根的两边把他夹住

如过这个函数是连续函数，那么对于实根F(X)=0，初始猜测

而bisection method就是在乘积仍然小于0的情况下，不停对半分





STEP1.选一个小于0的X1一个大于0的XU，

STEP2.求中间值xr

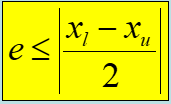
STEP3.XIXR>0，XI=XR因为他们正负符号相同

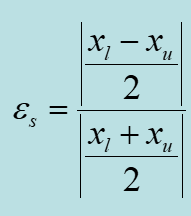
小于0XR=XU

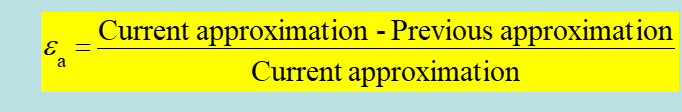
=0 的话XR=XI

什么时候停止：当你对你的error可以完全掌控的时候

绝对absolute error

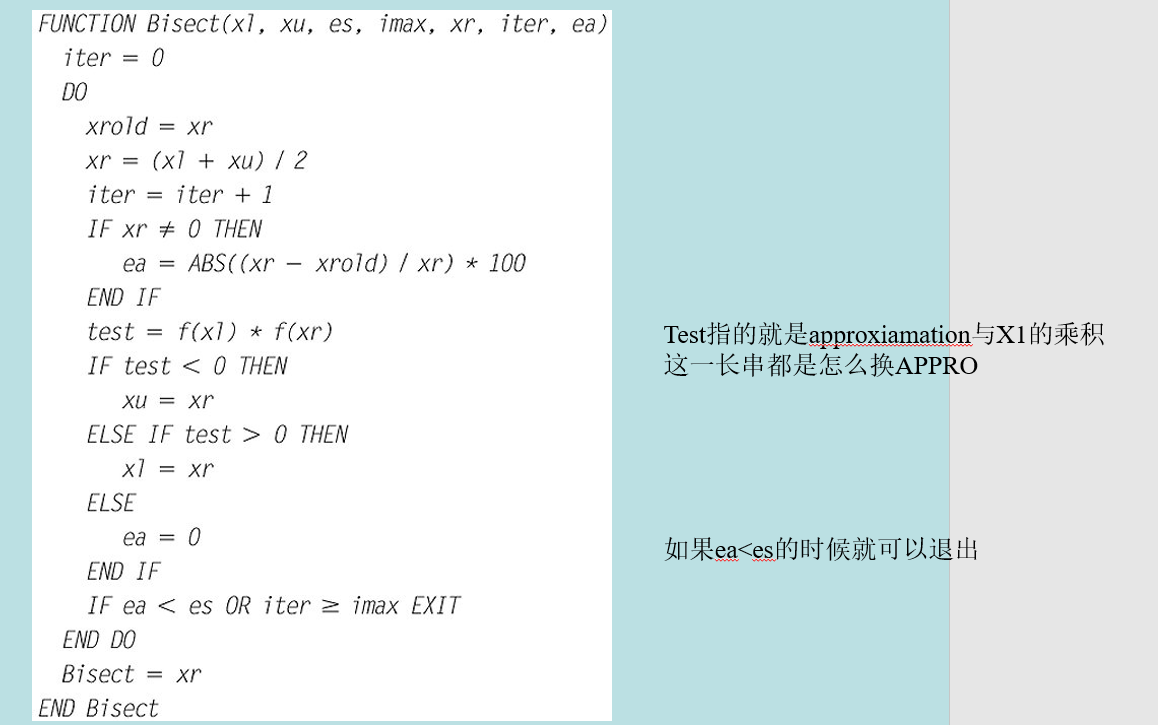






Approxiamation就等于X1+X2/2

EA就等于现在的X2+X2/2-之前的X1+X2/2

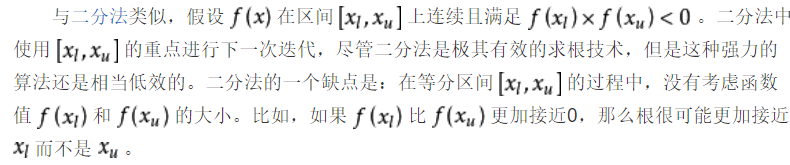


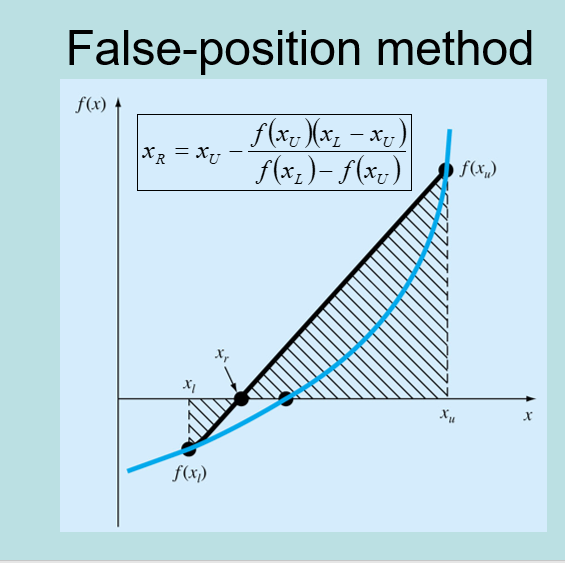
Bisection可控稳定，但是有些慢

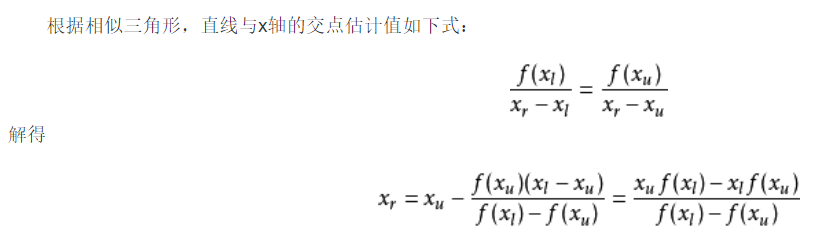
False-position method试位方法

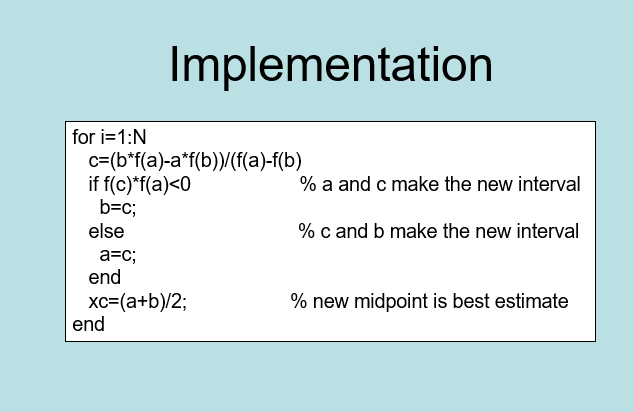
基本IDEA

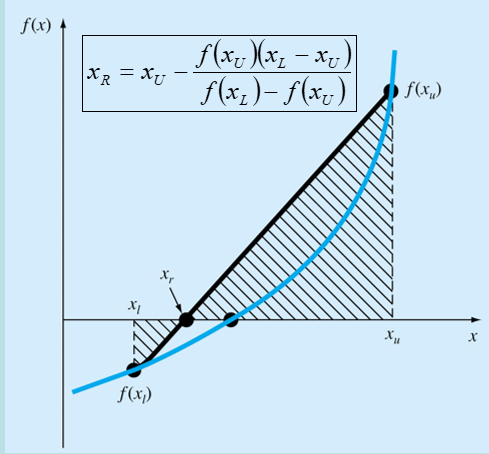
Bisection只是单纯的减半来缩小左右括号范围，有没有可能更快的缩小左右括号范围？



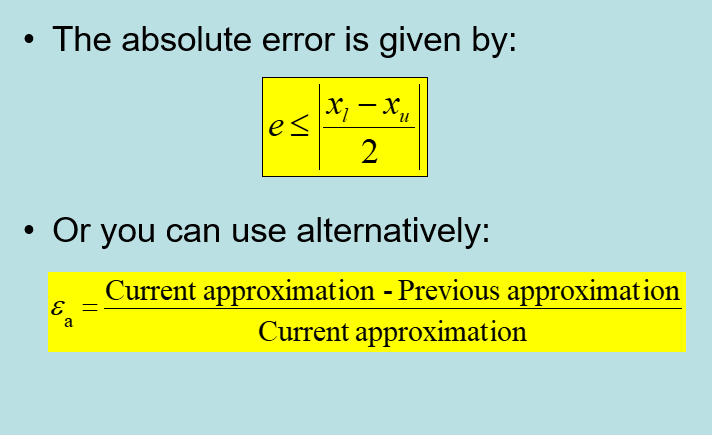








C是我们所求的XR，吧F(C)带回蓝色曲线，如果与F(A)乘积小于0，那么他就是新B，要不然就是新A。

停止

当ERROR小于差值除以2

Pitfall缺陷

错位法不一定就比二元法更快



前进的越来越慢，导致不动，而二元法很快就能缩小一半

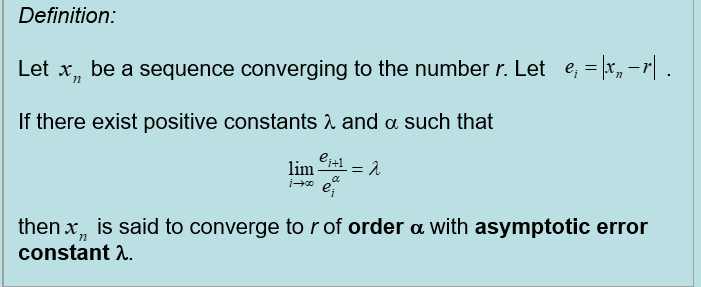
简单来说，稳定有效，绝大部分情况下比二元法快，但也有例外

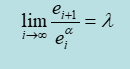
LESSON4

Convergence of iterative methods迭代方法的收敛

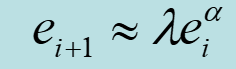
上节课我们用了bisection method，可以看见error是越来越小，但是我们怎么观测这个算法的快慢呢？

Order of convergence收敛级数



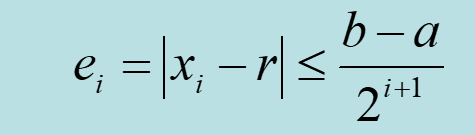
让Xn成为一个无限接近数字r的数列，让,如果存在一个正数入与α让

那么我们就说XN是以级数α ORDER α 向r无限收敛 并且with asymptotic error渐进误差常数入

意思是

这个式子告诉了ERROR与之前一个ERROR相比减速的快慢，

比如bisection method



每次的error都是这样，换句话说，error每次缩小了近似一半

所以这个error是线性减小以log0.5的坡度

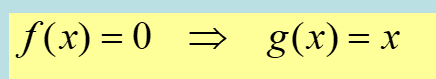
Fixed point

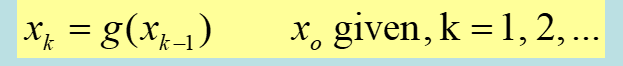
Open methods

Open methods只需要一个starting value x或两个starting value但这两个starting value并不强制bracket 根，即不用强制一个大于0一个小于0

简单fixed-point iteration定点迭代

重新排列function让左边的X式子变为



左右倒换，我们可以得到

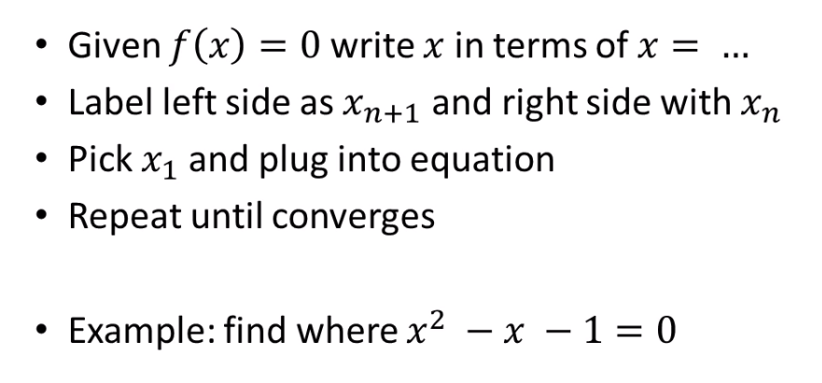
Bracketing method是收敛的，

Fixed-point有时候是偏离的diverge,取决于初始点（initial guess）与这个function本身

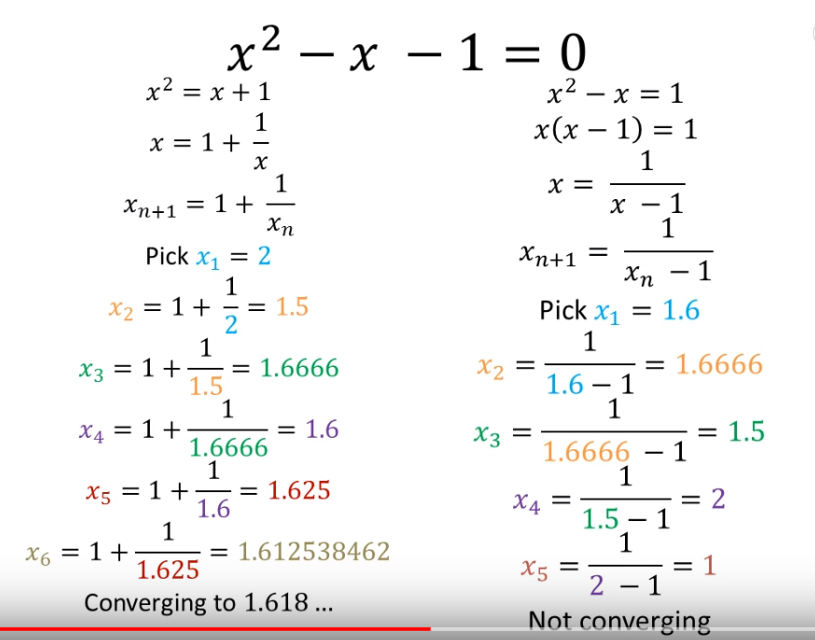
比如对一下f(x)我们可以提取出多个g(x)

记住f(x)=0,g(x)=x

网上找的方法



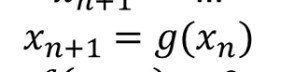
repeat until convergence重复直到收敛



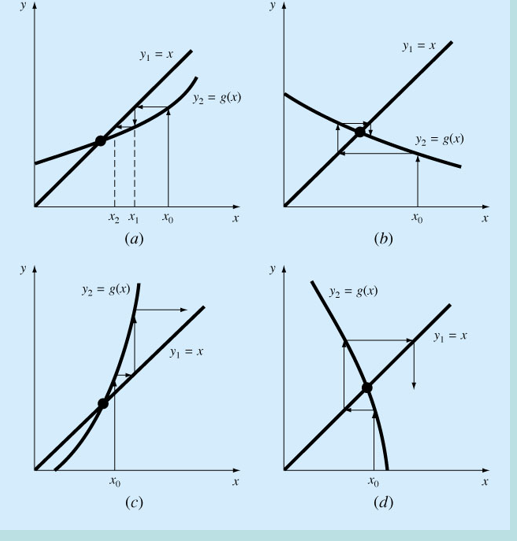
分成一个X在左边其余X在右边后，左边的X是XN+1，右边的X是XN

先任选一个x1=多少，然后重复代入，最后无限收敛于一个值

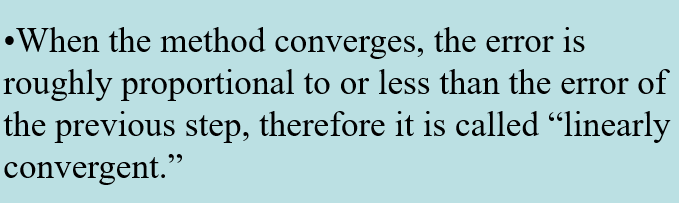
为什么左式convergge了而右式没有

我们需要让g(xn)的导数绝对值小于1，这样才能确保他越来越小



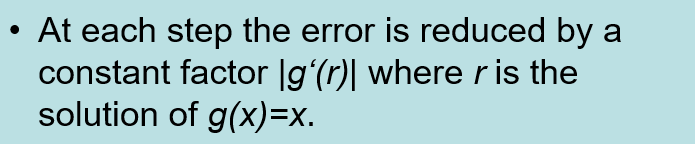
回到PPT

这幅图的意思就是选一个X0然后在我们的Y=G(X)上求出Y,再利用Y=X，把Y作为下一个X1

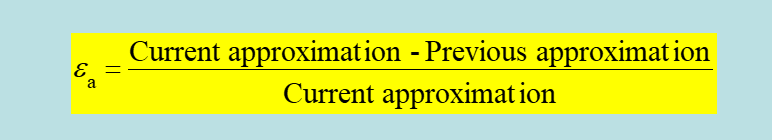


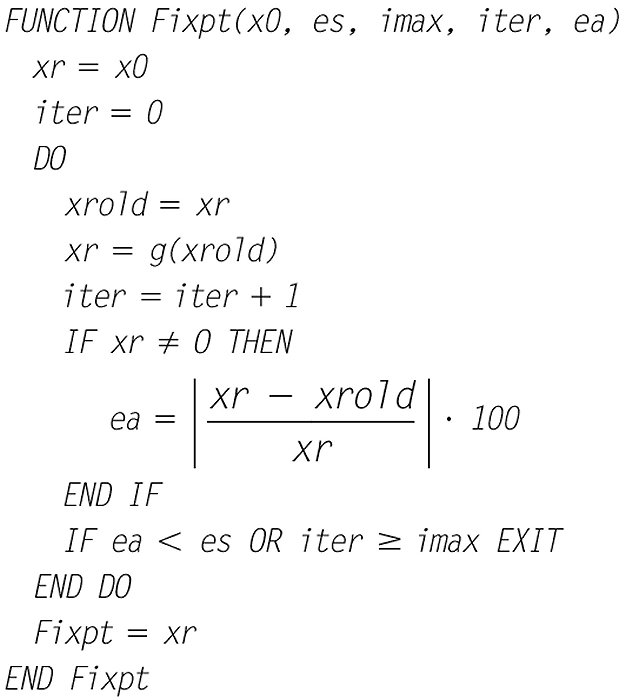
当用这种方法时，每一次的error都与上一次的error成比例或小于，所以叫线性收敛

每一次循环error都以变小，gr是当前X的解

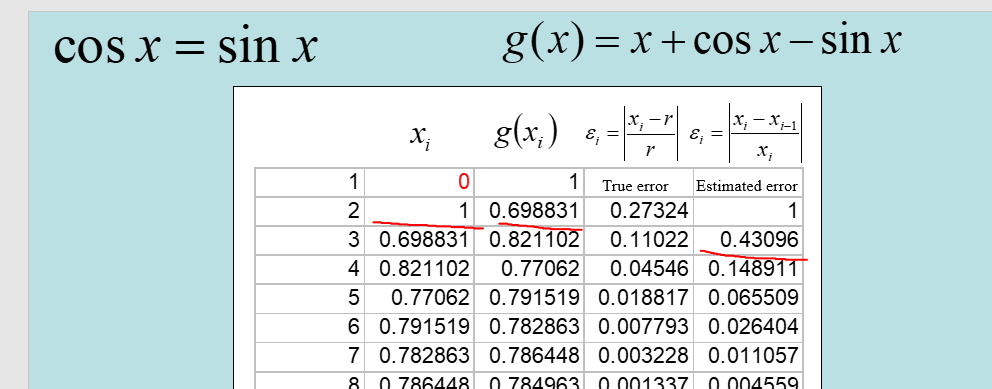


什么时候停止





iter 是iteration，总之就是设置一个X0，检查当前解XR-上个解XR OLD/XR 乘100是不是小于我们设定好的ES



0.43096=0.6981-1/0.698831

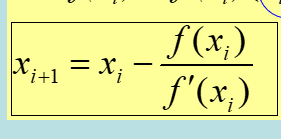
总结：fixed point法解决了X=G（X）这类问题

并不总是收敛，需要G(X)的导数小于1

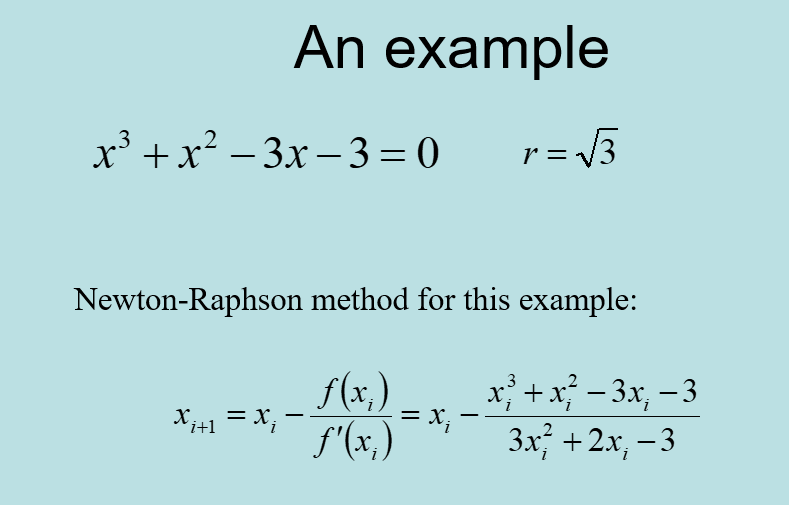
是线性收敛的（每一次ERROR以一个常系数）成比例缩小

该方法的停止取决于对error的监测estimation

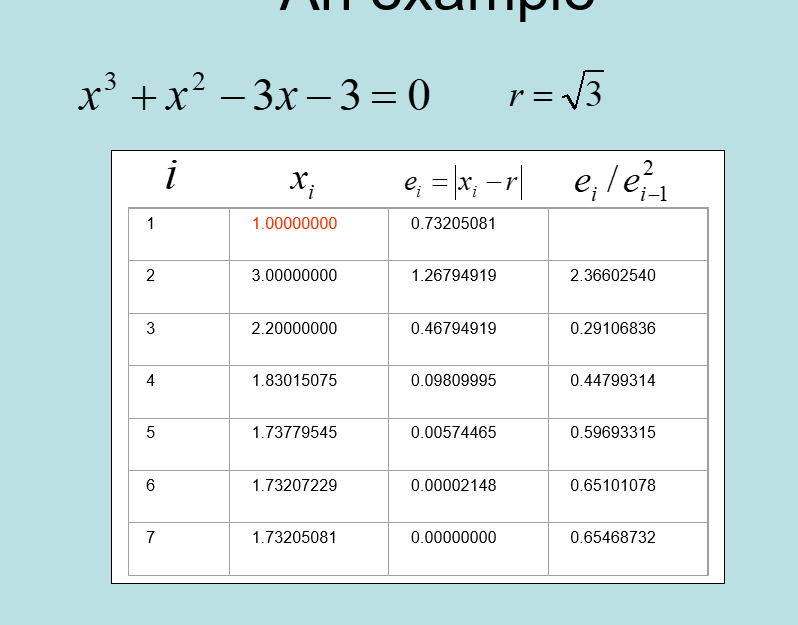
Newton-Raphson 牛顿拉普森



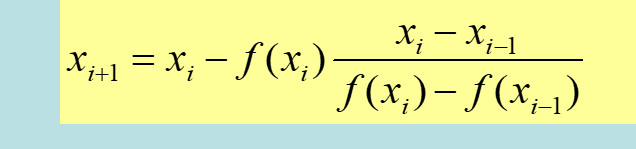
局限性，如果这个f(x)的导数可以被轻松得到，牛顿法EZPZ，如果求不出他的导数，就不好使



是quadratic收敛（很快）但需要知道导数

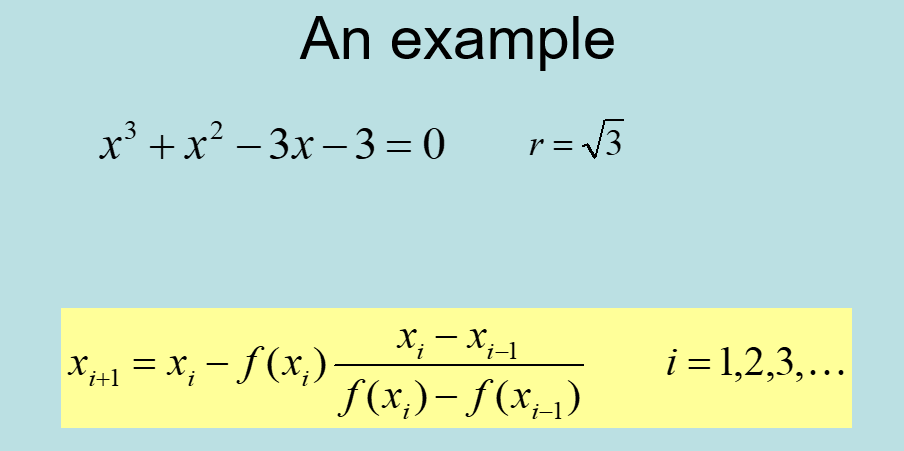


Secant METHOD 割线法



牛顿法的一种变种，求不出导数时使用

需要预先估计两个值，XI XI-1，才能求出XI+1,但并不要求两个值符号不同，所以不在bracketing method分类里

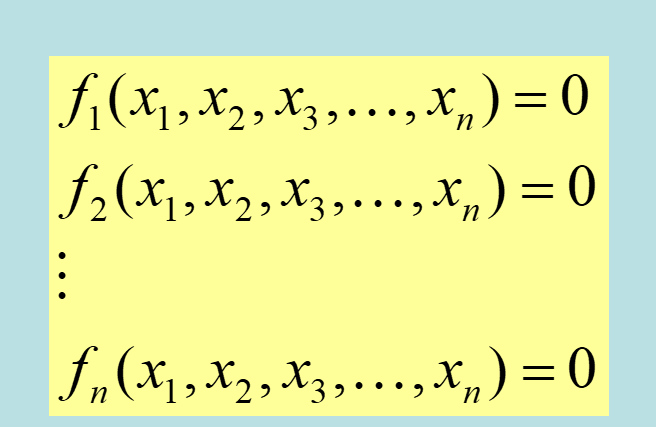




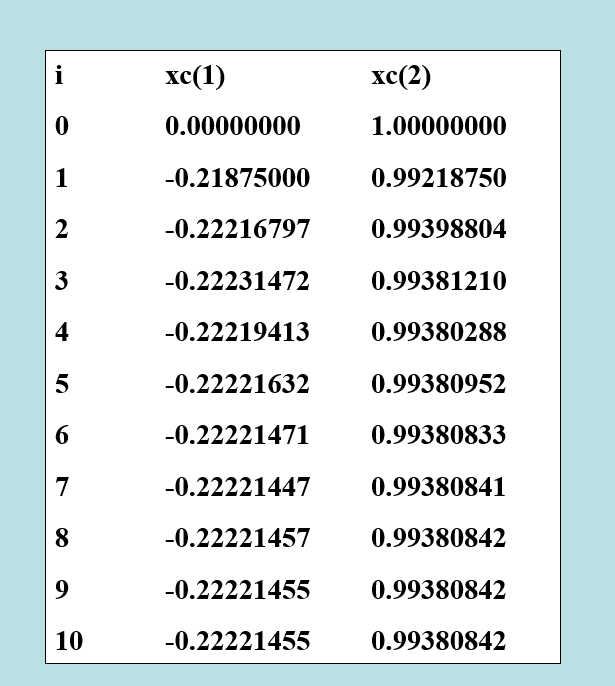
superlinear 收敛，比quadratic慢，比Linear快

Lesson5

**System of equations 方程组**







-0.75 1

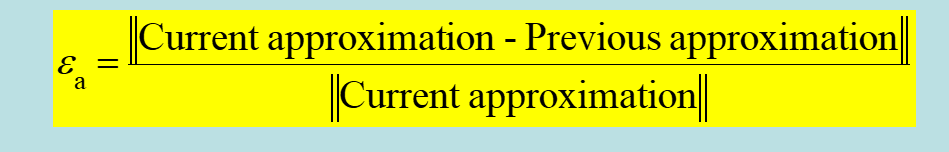
-0.53125 0.9296875

-0.07373046875 0.9622497559

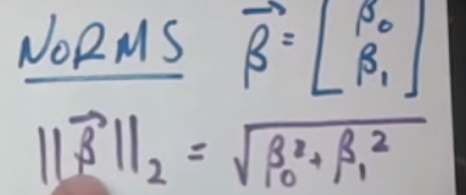
-0.2284067876 0.9986079368

over

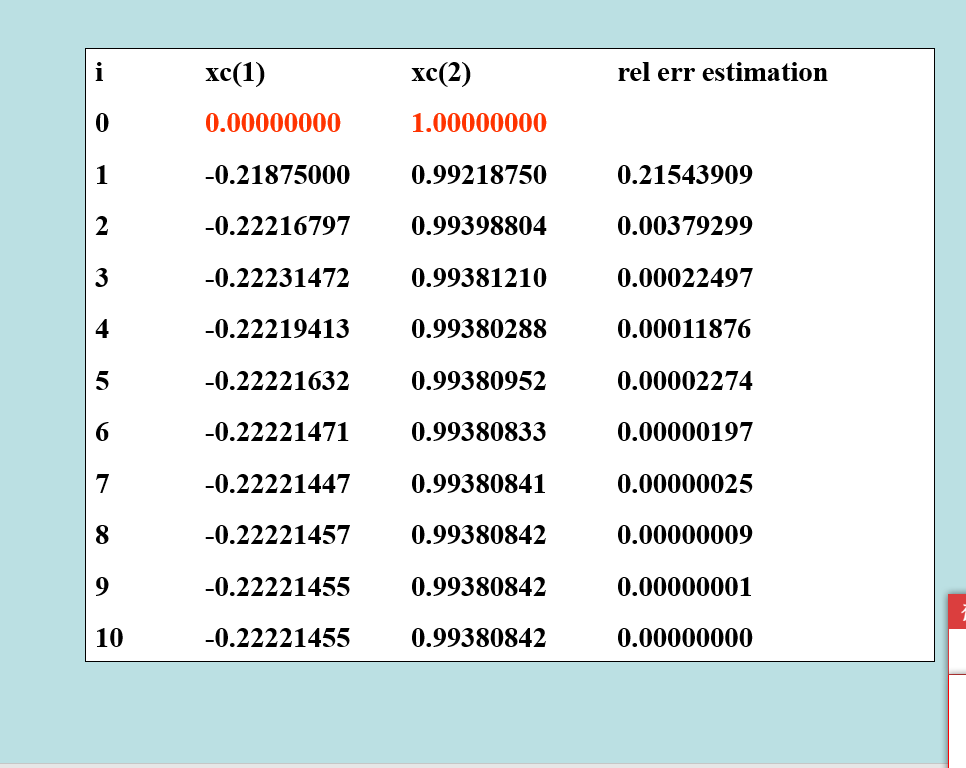
它上面第二步在瞎几把求



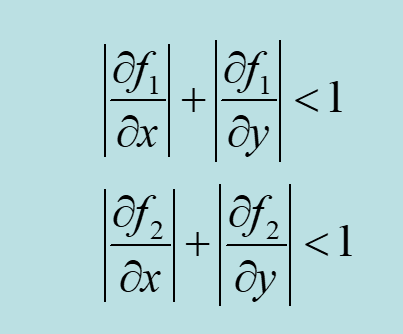
两个竖杠意思是NORM



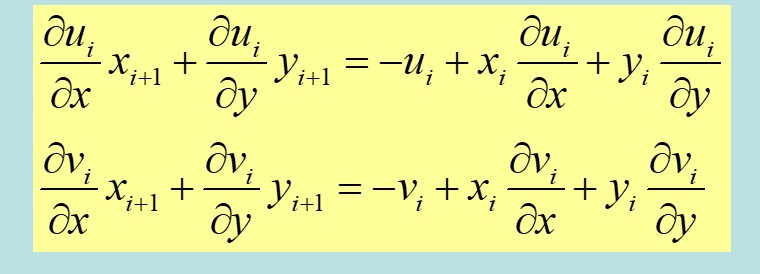
等于两个坐标平方和开根号



0.21543909=（（-0.21875000-0）^2+(0.99218750-1)^2）^0.5/((0.2187500)^2+(0.99218750)^2)^0.5

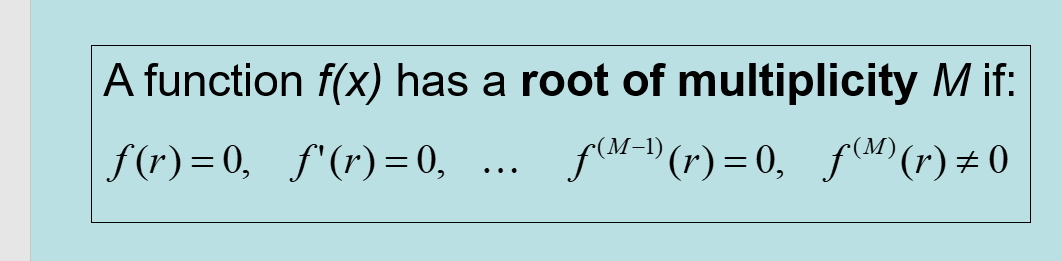
收敛所需条件

Newton-Raphson



这节课我们只要知道有这个方法就行了，具体的以后会说

Promblems and Dangers



一个function有root of multiplicity M取决于他的倒数是不是等于0，一阶导数等于0，说明有两个重复解2阶三个…

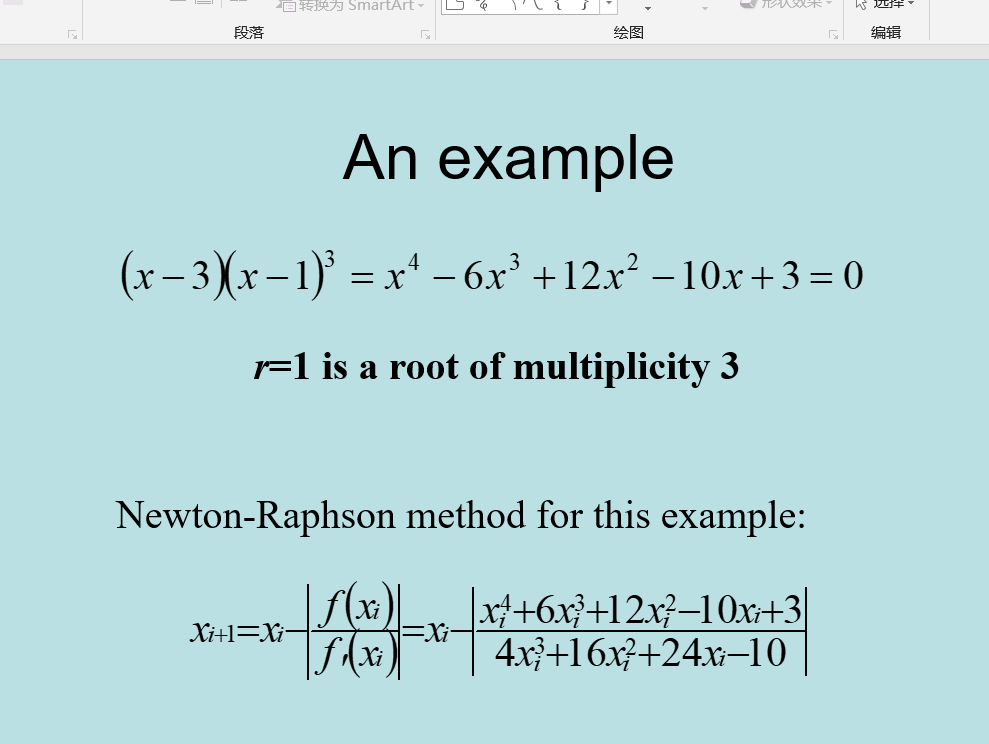
ROOT OF MULTIPLICITY 指数值相同的重复解

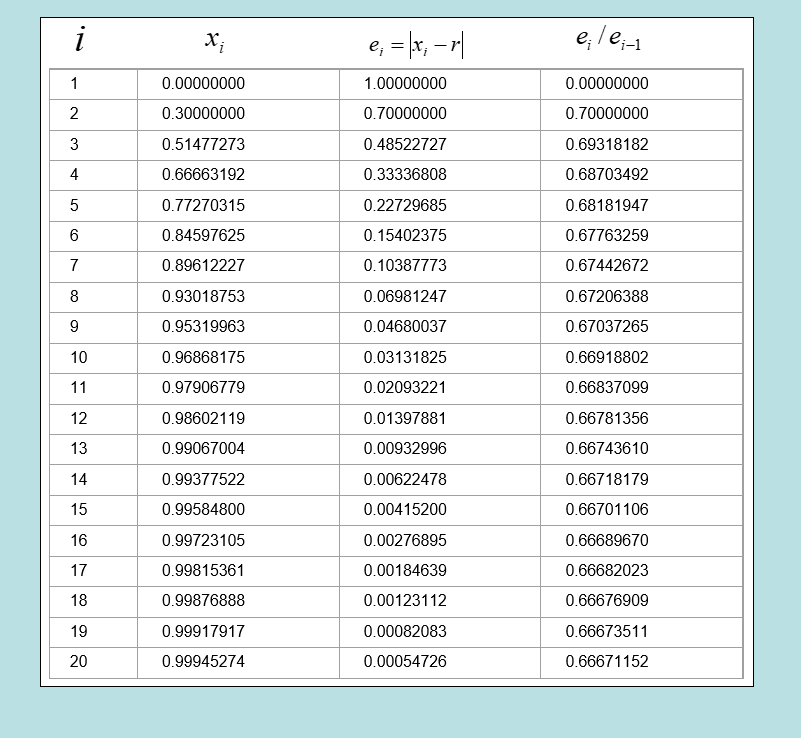
比如X^2-2X+1=0

有两个解X=1 X=1， 他的导数时2X-2=0

X=1时就有两个重复解

当multiple root的时候，newton-raphson不再是longer quadratically convergent而是线性收敛





可以看见error比值趋于固定

the bisection algorithm 也无法很好的解决重根问题

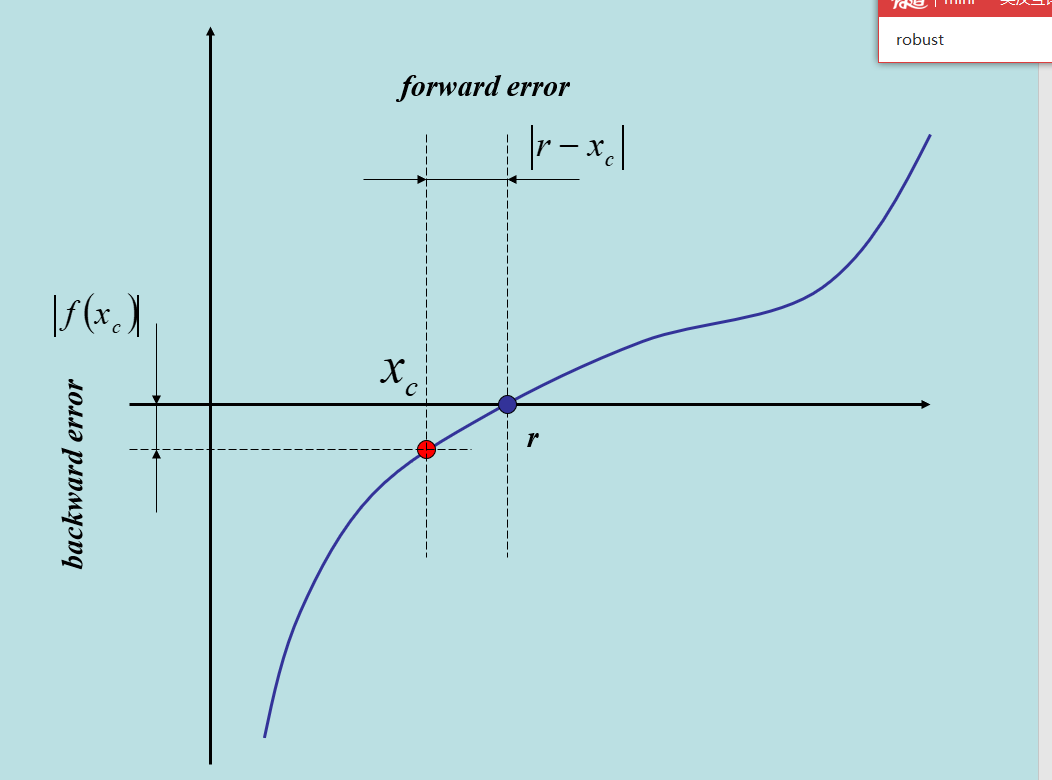
这类问题叫做ill-conditioned problems，没有正确算法能解决他们

lesson6

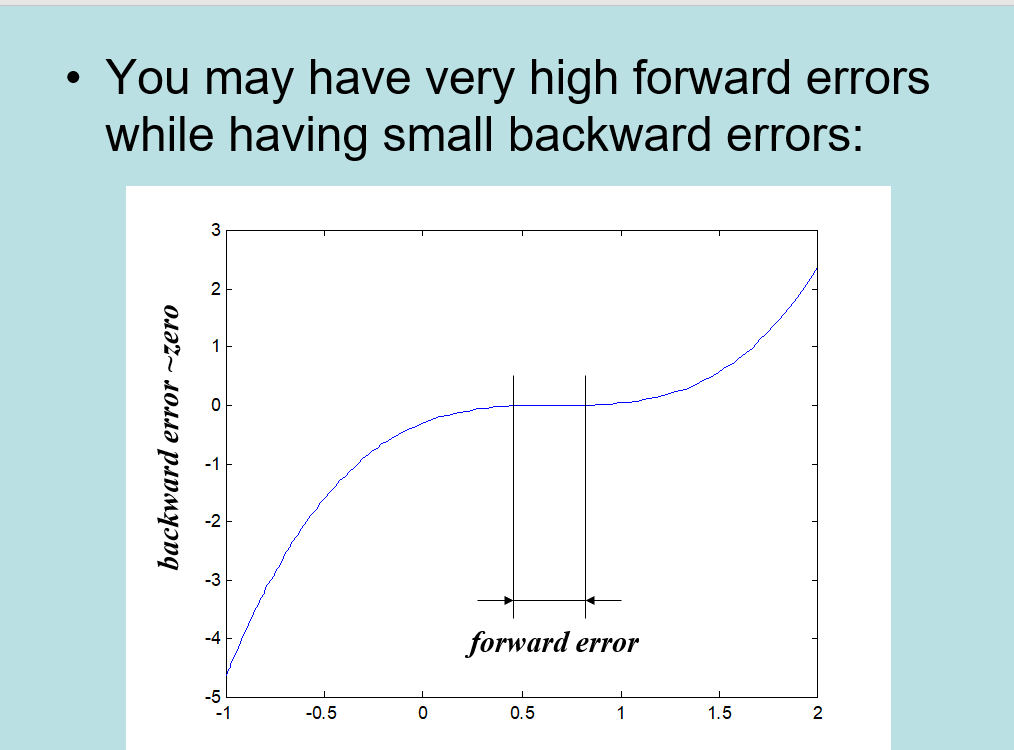
假设f(r)=0,XC是关于r的近似解，那么BACKWORD ERROR IS



FORWORD ERROR IS 

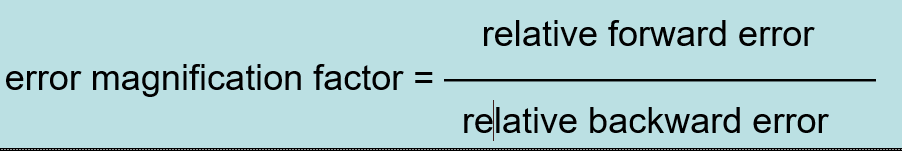


你可以有特别大的forward error同时又很小的backward error



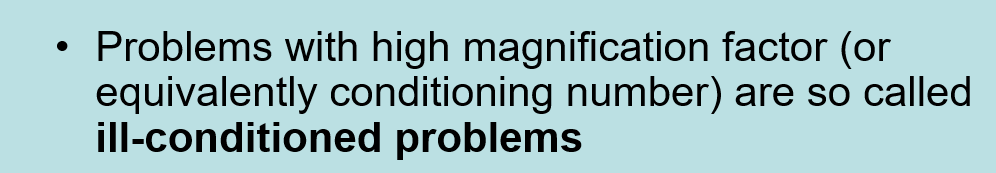
**error magnification factor:**

error大小系数



一个有着比较大的error magnification factor 得问题叫做





error magnification factor取决于所选function本身，与用了什么数学算法无关

这个系数过大意味着round-off error

我们每一次计算只能做到一定精度，因此我们实际上解决的并不是f(x)=0,而是他的近似值

比如

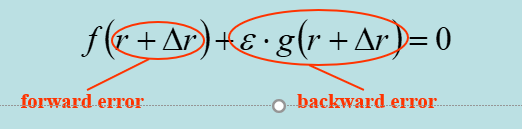
我们实际上求出来的是他的泰勒展开式

更准确的说当我们想要解决f(x)=0时，

我们把round-off error记做

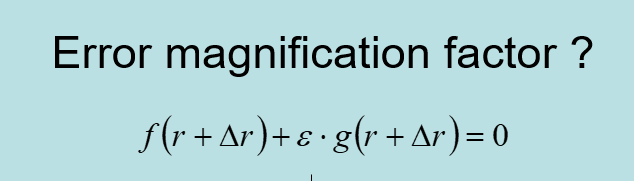
根将只定位于一定精度：

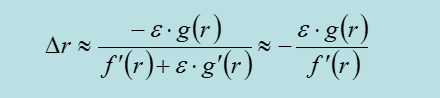
我们实际上解决的问题是



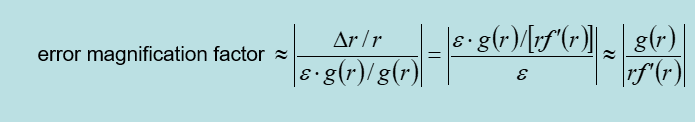
解释：r是正解，r+△r是我们求的近似解，e.g(x)是对应forward error产生的backward error

而我们的x是r+△r



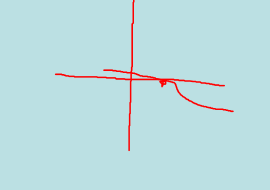


这个关系告诉我们，当我们有了一个希望达到的给定精度（forward error），根所在区间的错误的大小（backward error）



每一次函数水平靠近要定位的根时，我们就会遇到麻烦

这个问题不仅限于重根，

水平切过去，因为很小的round-off error都会被放大

LESSON6跳过，有点水（不水，会考）

LESSON7

Linear Algebraic Equations (part3) 线性代数方程

叫做关于变量X Y的线性方程

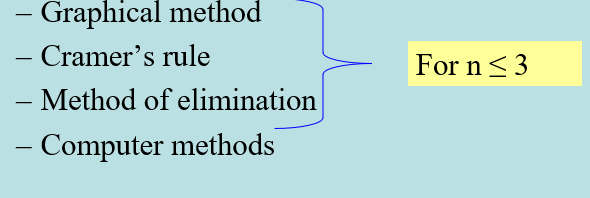
叫做关于变量X1，X2，X3，…XN的线性方程

这样养一个方程的解是实数

对于N<=3的线性方程组，我们只需要用elimination这种消除法就完事儿了（就是二元一次方程组通过通分除掉某个X2这种）

Linear algebra线性代数提供了工具来解决这样的线性方程组

对于n<=3，我们有以下解法



图像法

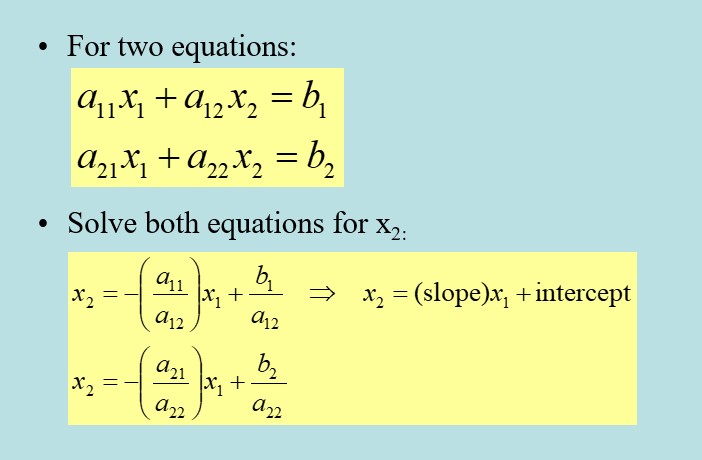
克莱姆法则

消除法

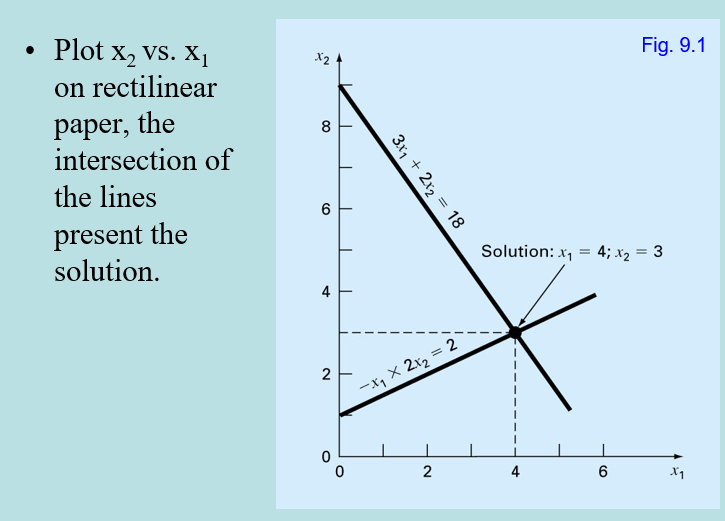
计算机方法

图像法

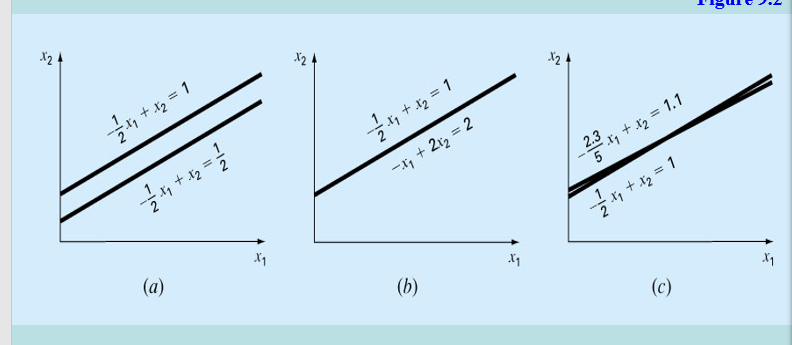
对于这个方程组



我们可以得到X2这两个解，

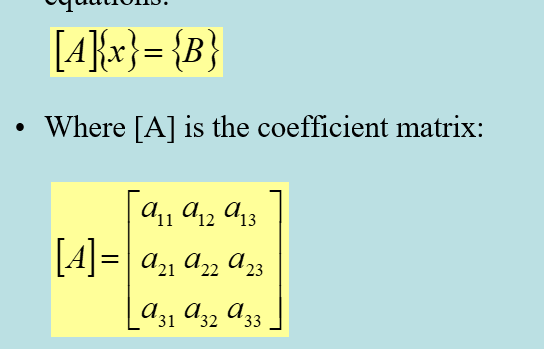


描在智商，求出交点



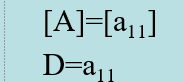
Cramer’S Rule

用行列式determinant来表示三个方程组,A是系数矩阵



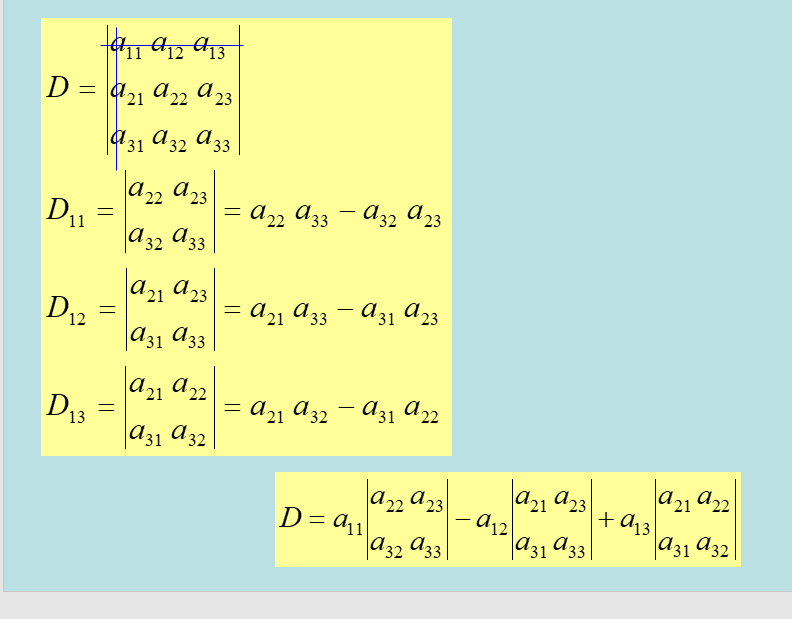
如果每个矩阵都是正方形矩阵，那么有一个和每个矩阵[A]对应的数D，determinant行列式

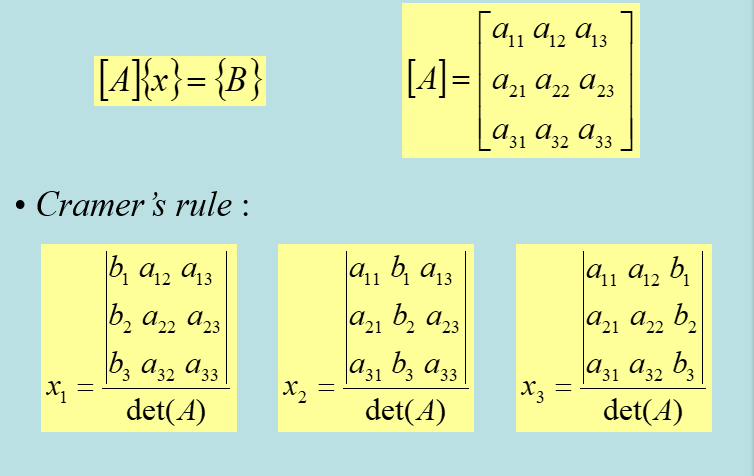
如果[A]是一阶矩阵



如果是三阶矩阵

这个D的求法

cramer’S rule



好处是非常系统非常无脑，坏处是你他妈算不完

消除法

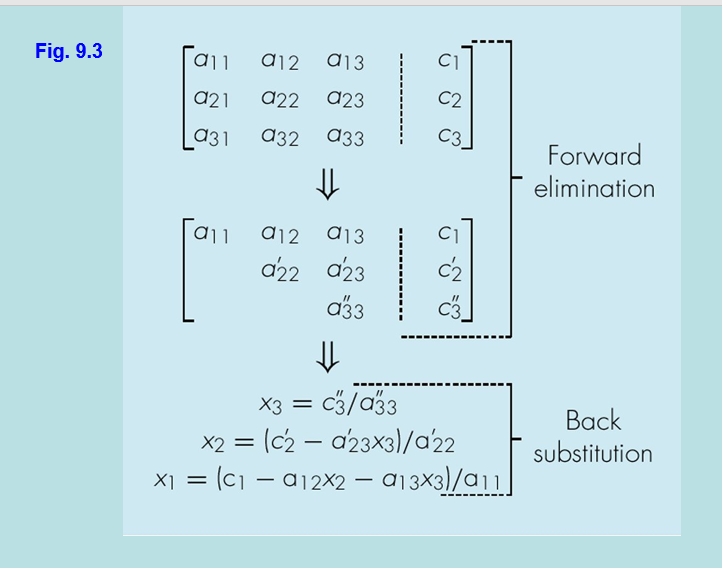
最基本的策略通过其他方程消除某一变量

消除法可以对三个or以上的方程组管用

但是手算很难

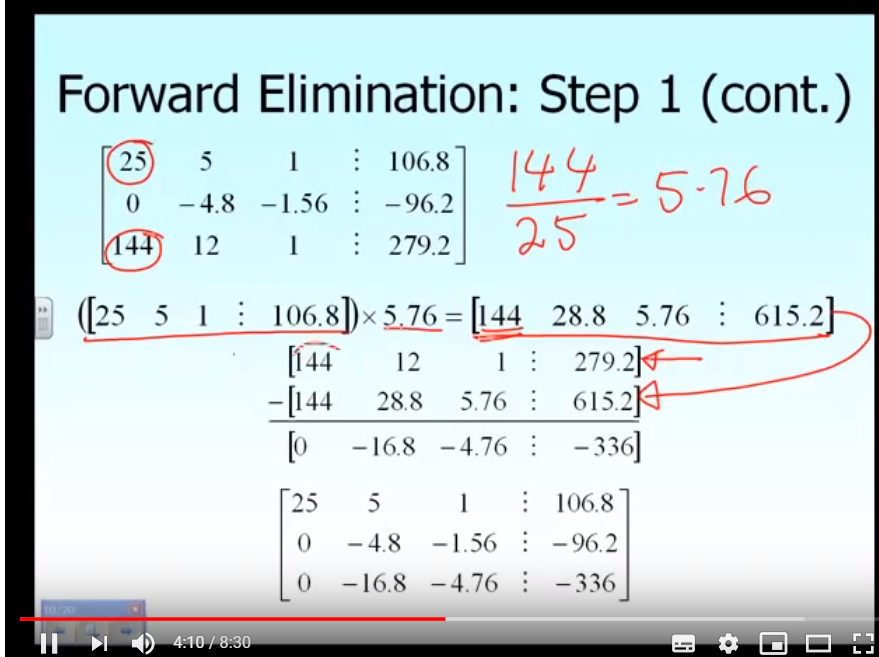
Naïve Gauss elimination

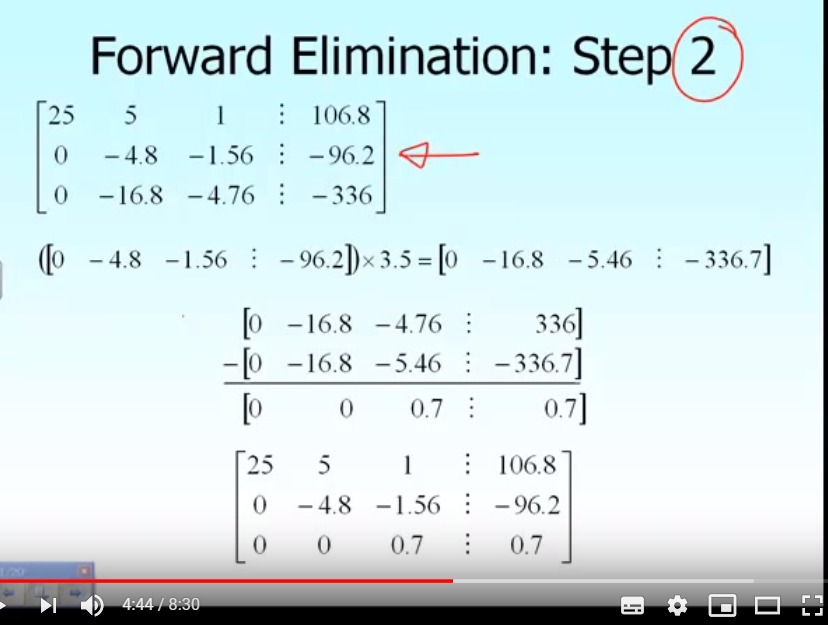
天真的高斯消去法

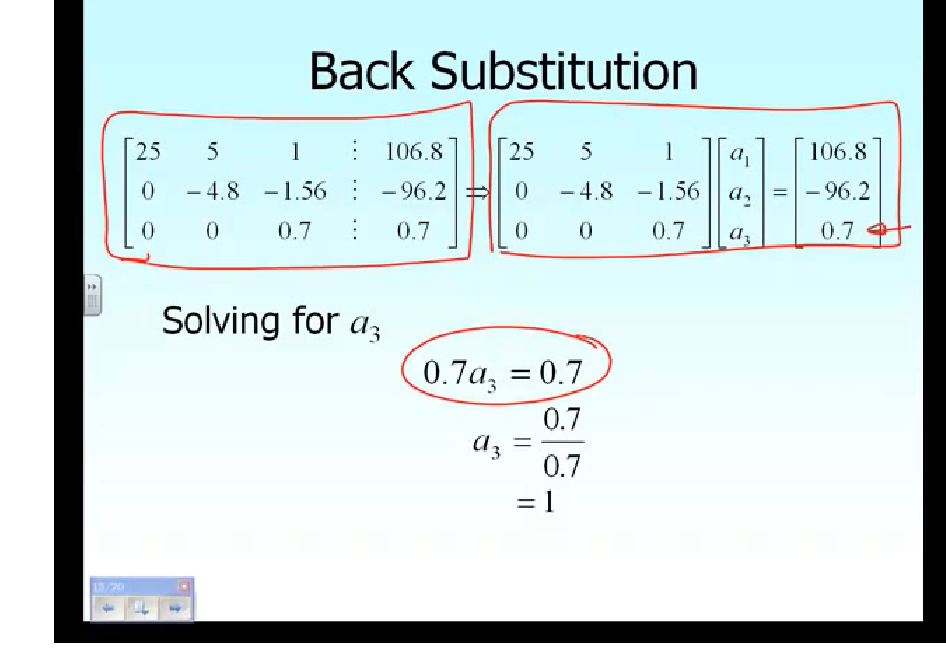


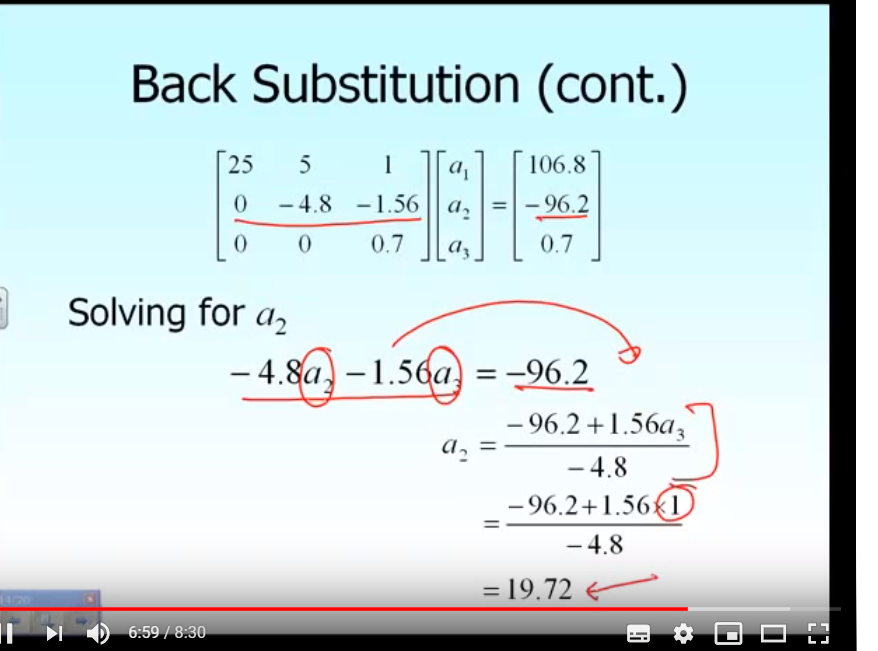
C是方程右边的数字

网上看的



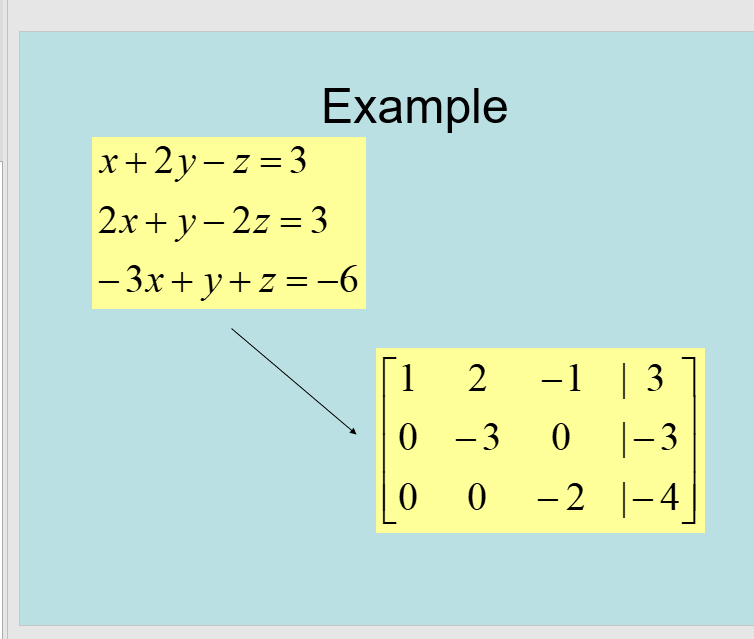






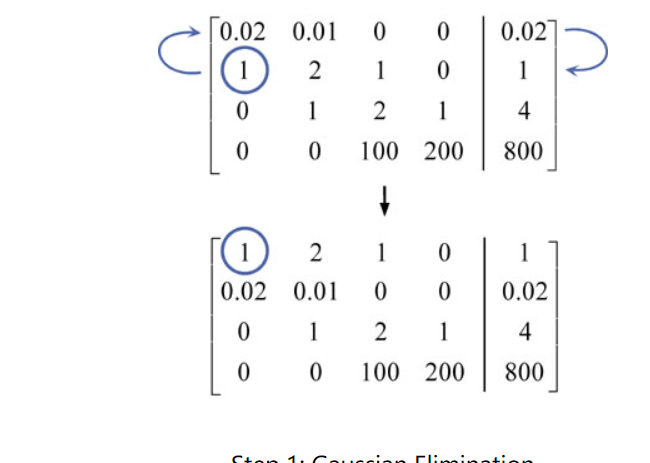
总而言之用第一个式子通过系数变化把第一列下面两个变成0，然后第二排通过系数变化把下面一行第二列变成0，然后对应系数求出a3,求出a2，求出1

化0的过程叫elimination，带回的过程叫back-substitution



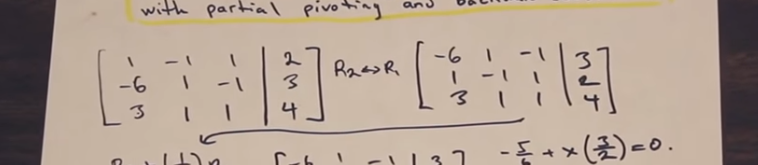
partial pivot 部分pivot中枢

我们找到第一列最大的数，放到第一行，然后再右下角的数中找到第二列最大的数，，放到第二行，无限重复





三阶就换一次就够用了·

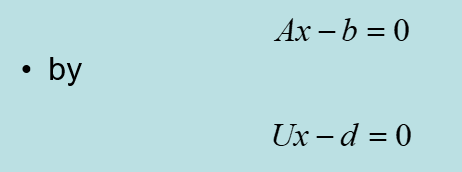


LESSON08

LU decomposition

LU 分解

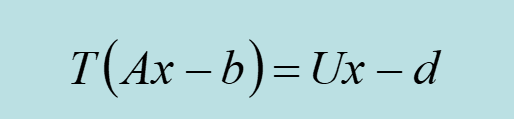
将galse算法替换成



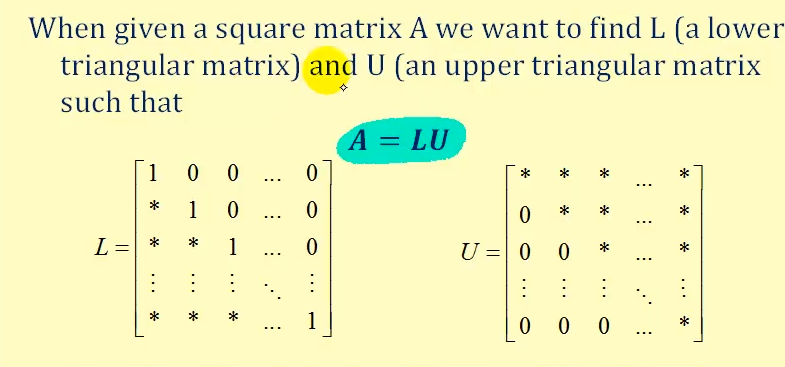
U是一个upper diagonal matrix 上对角矩阵



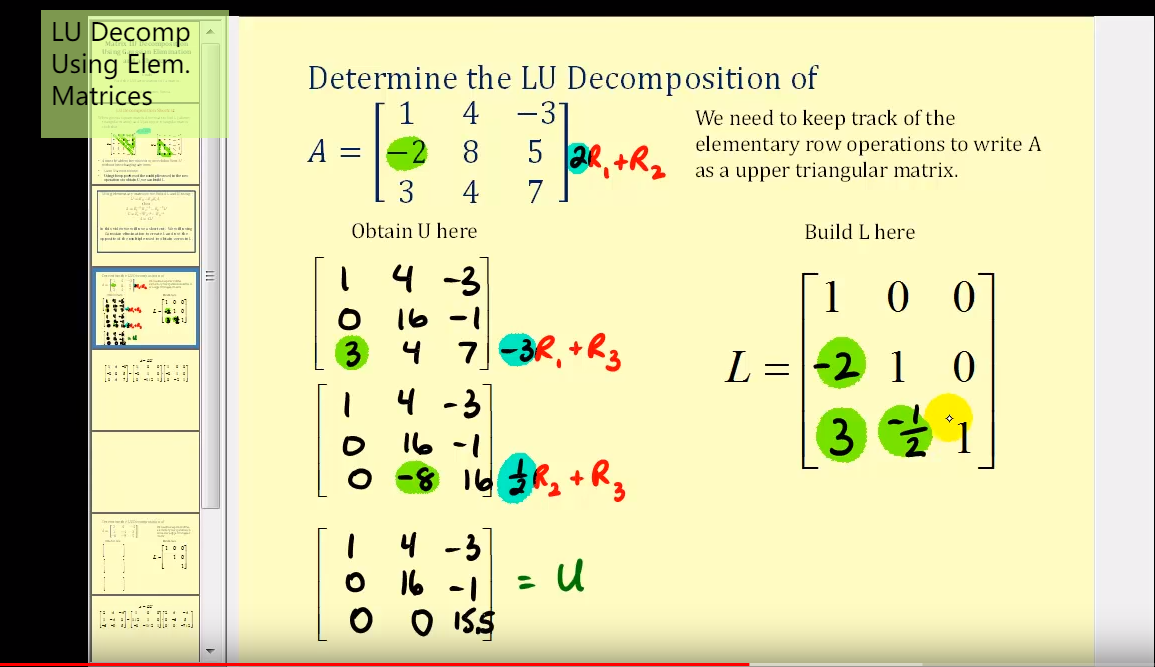
我们要通过什么矩阵T达到这个转换呢·？



YOUTUBE上看的，我们要把原矩阵A转换成LU

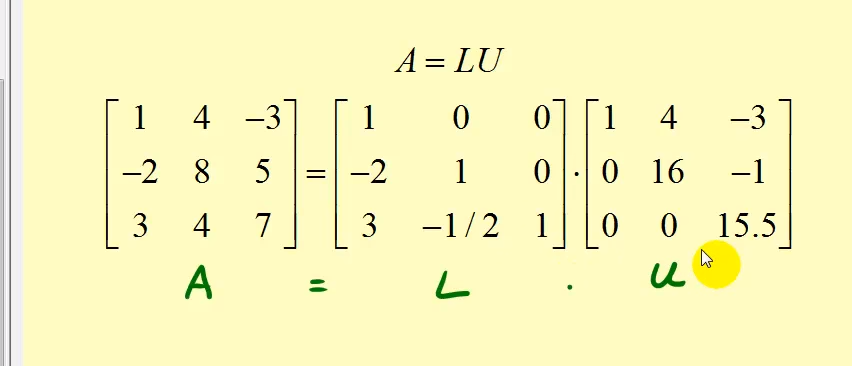


L上下矩阵，U是上矩阵



U是常规解法，再消除第二行时，我们用XR1+R2把第一个数消去，然后把-X带入L第二行第一列

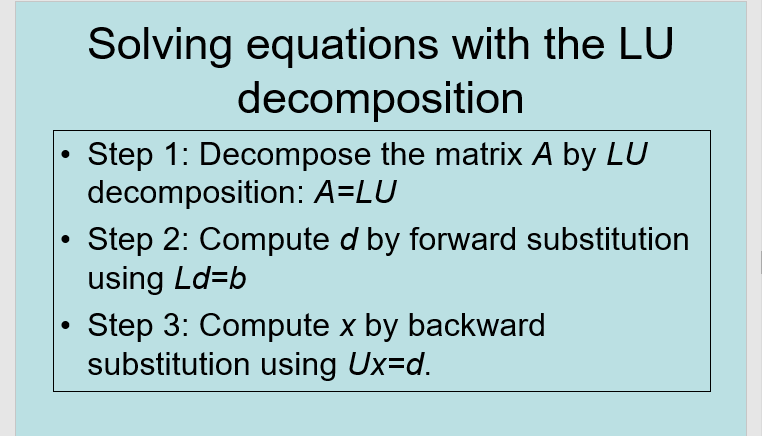
XR1+R3消除第三行，-X带入第三行第一列，XR2+R3消除第三行第二列，-X带入第三行第二列



然后通过L和U找到D和X







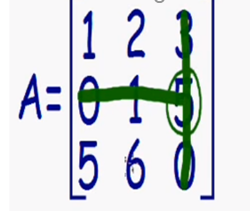
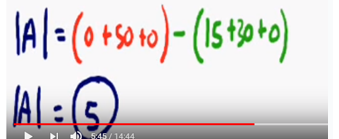
Matrix inverse

反矩阵

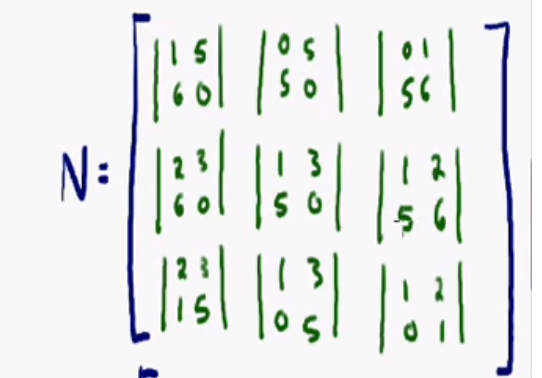
网上方法

第一步

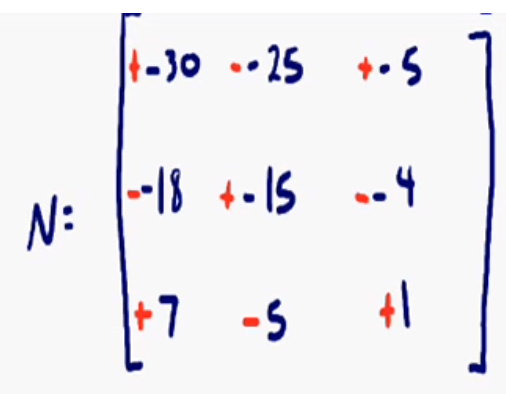
求这个矩阵大小

. 

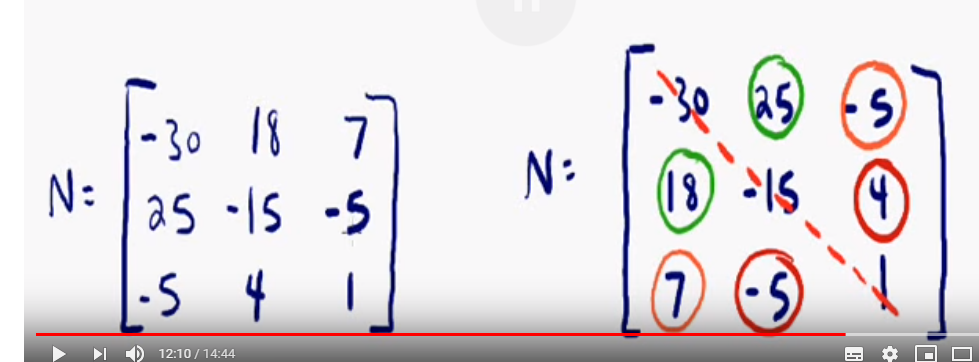
然后求对应格子



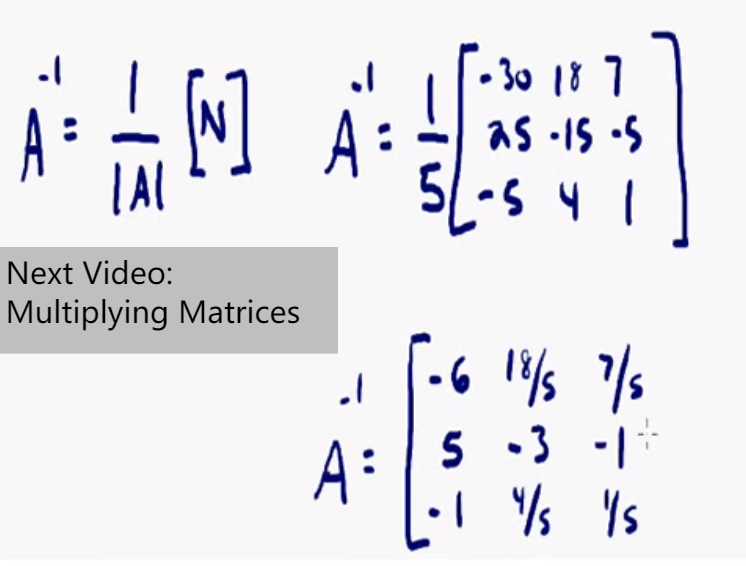
加上对应正负号



沿着对角线反转



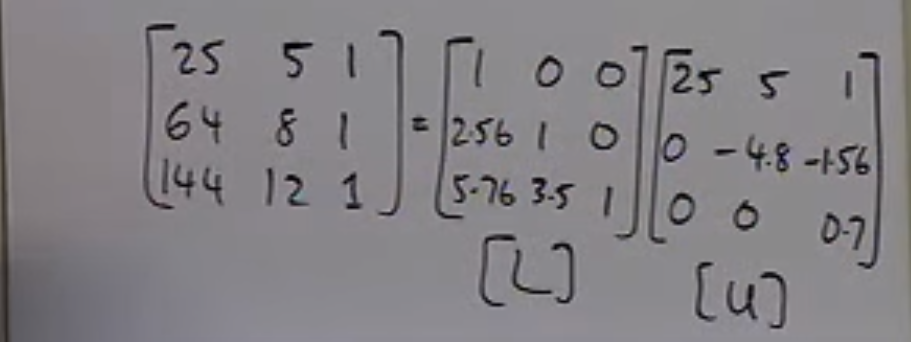
乘矩阵原大小分数之1



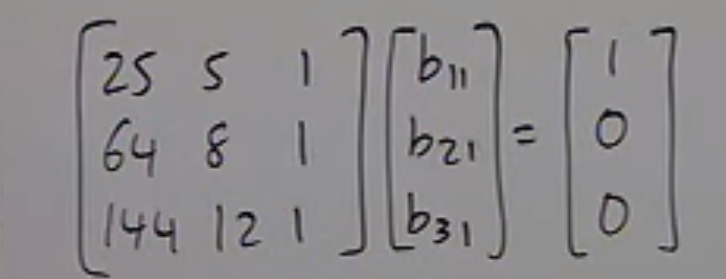
用LU方法

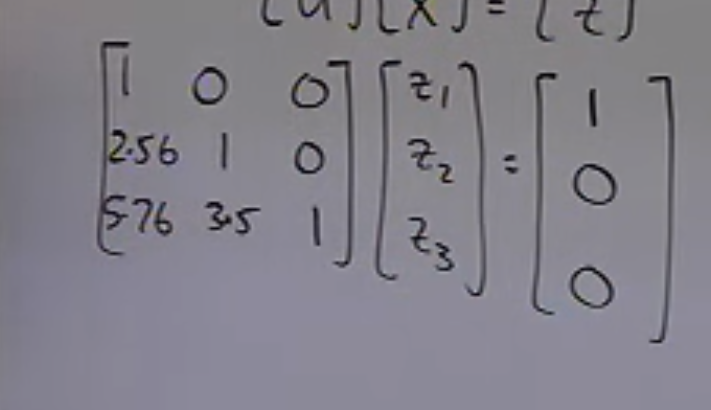
第一步

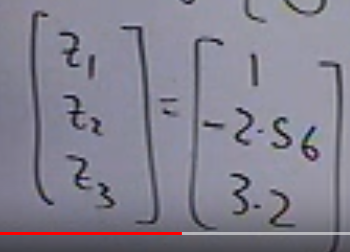
对A找到LU



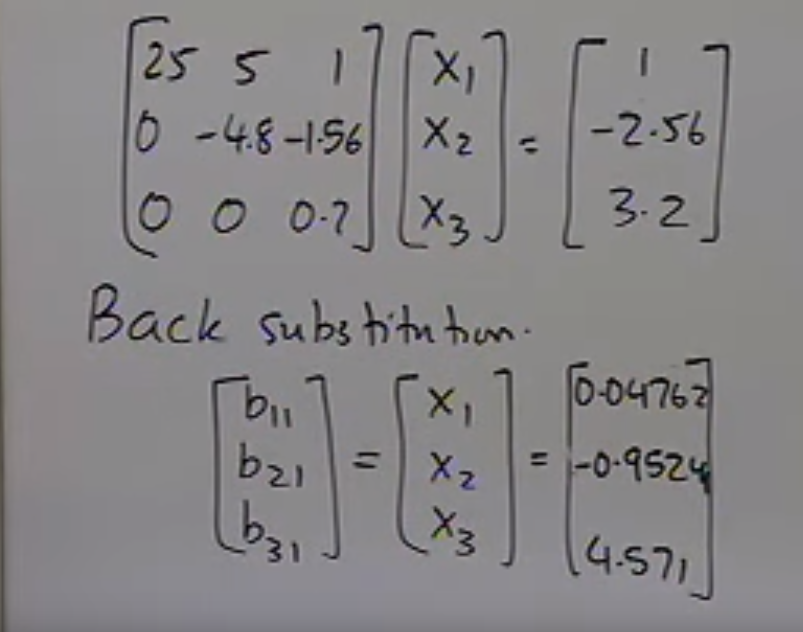
第一列我们追求 先用L找到Z，



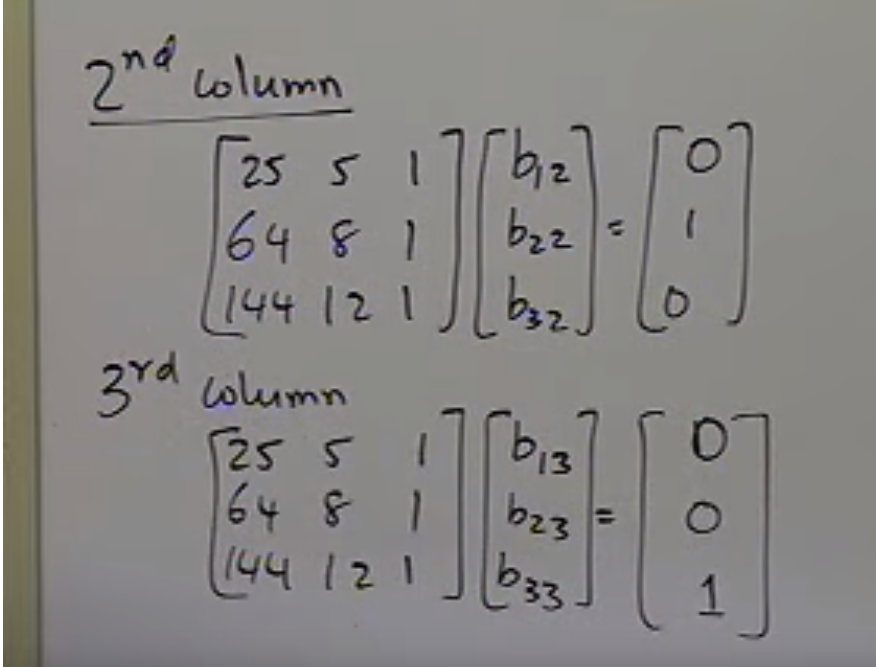




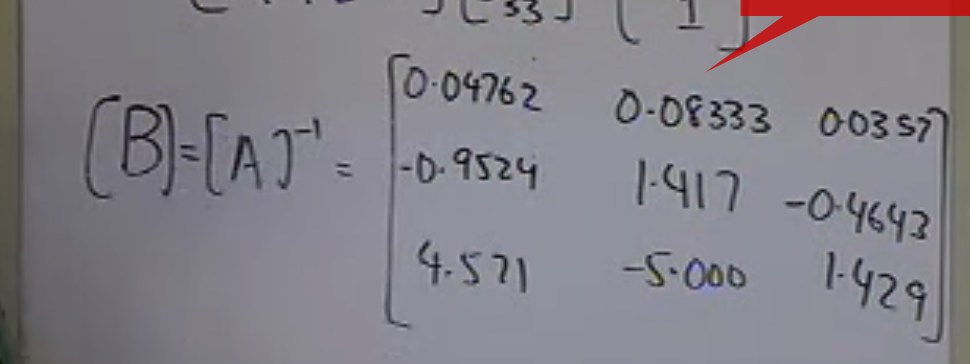
再用U找到X



23同理



带入



PA=LU