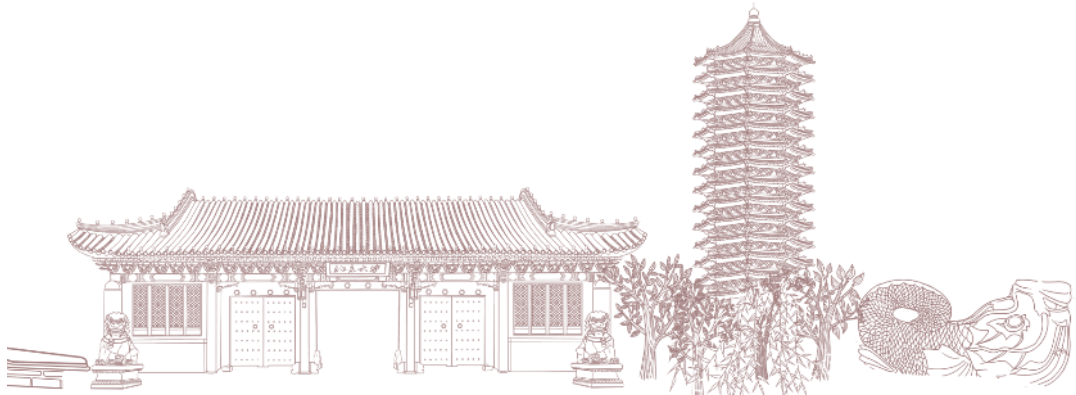
北京大学
PEKING UNIVERSITY

线性系统求解
Solution of Linear Systems

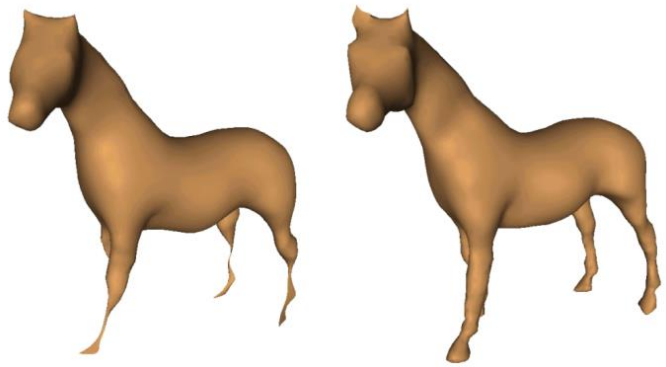
阮良旺

2024.6.17

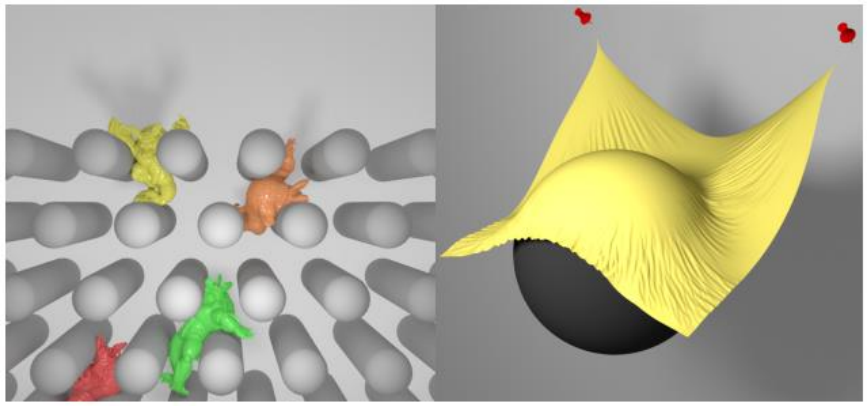


$$Ax = b$$

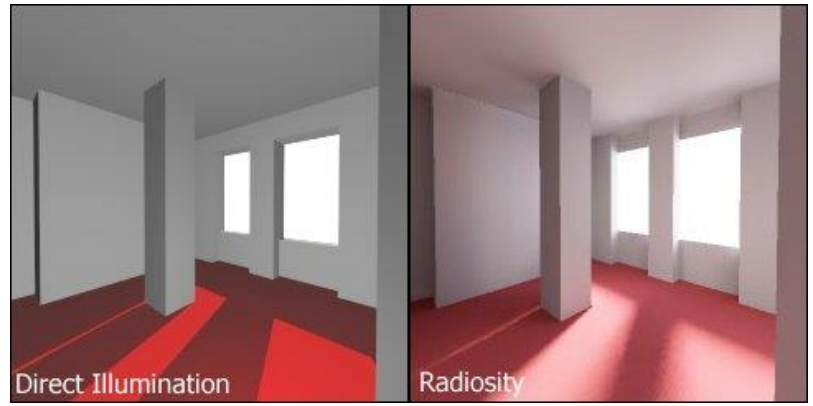
矩阵方程出现在图形学的各个地方



几何处理



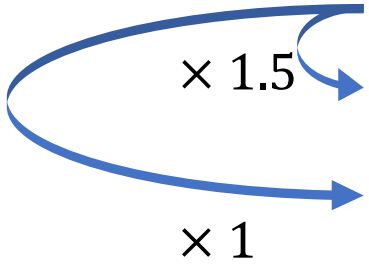
物理模拟



全局渲染

直接求解
Direct Solver

高斯消元 (Gaussian elimination)



$$\begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 8 \\ -11 \\ -3 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 1 & -1 \\ 0 & 0.5 & 0.5 \\ 0 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 8 \\ 1 \\ 5 \end{bmatrix}$$

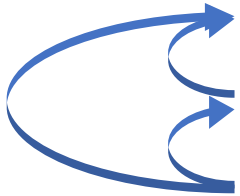
高斯消元 (Gaussian elimination)

$$\times(-4) \curvearrowright \begin{bmatrix} 2 & 1 & -1 \\ 0 & 0.5 & 0.5 \\ 0 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 8 \\ 1 \\ 5 \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} 2 & 1 & -1 \\ 0 & 0.5 & 0.5 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 8 \\ 1 \\ 1 \end{bmatrix}$$

高斯消元 (Gaussian elimination)



$$\begin{bmatrix} 2 & 1 & -1 \\ 0 & 0.5 & 0.5 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 8 \\ 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ -1 \end{bmatrix}$$

上三角矩阵可以通过反向的高斯消元进行求解

高斯消元 (Gaussian elimination)

$$\times 1.5 \curvearrowright \begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & 1 & -1 \\ 0 & 0.5 & 0.5 \\ -2 & 1 & 2 \end{bmatrix}$$

行变换等价于左乘矩阵

$$\begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & & \\ -1.5 & 1 & \\ & & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & -1 \\ 0 & 0.5 & 0.5 \\ -2 & 1 & 2 \end{bmatrix}$$

高斯消元 (Gaussian elimination)

高斯消元法构建了矩阵的LU分解

$$A = LU, A^{-1} = U^{-1}L^{-1}$$

$$\begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & & \\ -1.5 & 1 & \\ -1 & 4 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & -1 \\ & 0.5 & 0.5 \\ & & -1 \end{bmatrix}$$

L矩阵（下三角）　　U矩阵（上三角）

对角线为1

LU分解算法

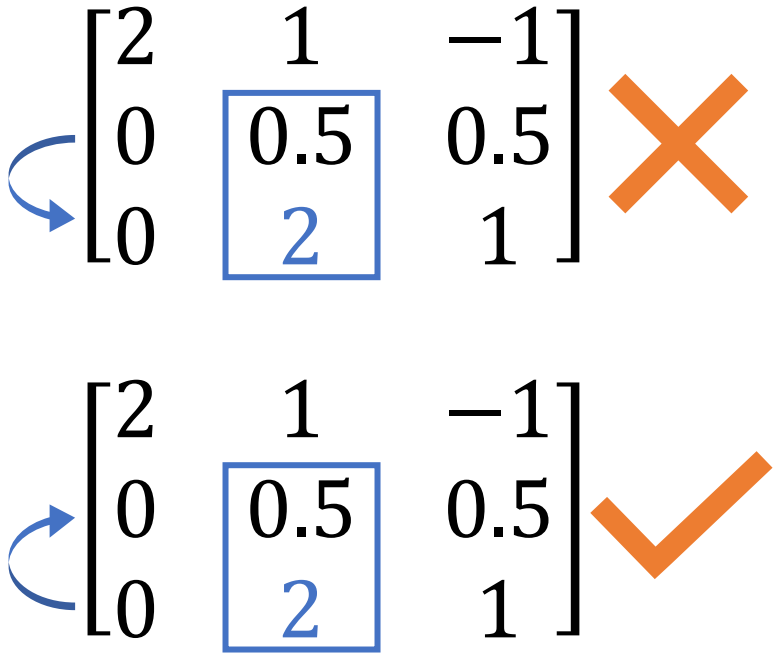
Algorithm 1 The LU decomposition of a matrix A. Upon exit, the entries of A have been overwritten with the entries of L (below the main diagonal) and the entries of U (main diagonal and above). The diagonal entries of L are all equal to 1.

for $k = 1, \ldots, n-1$ **do**
  for $i = k+1, \ldots, n$ **do**
    $a_{i,k} = \frac{a_{i,k}}{a_{k,k}}$
    for $j = k+1, \ldots, n$ **do**
      $a_{i,j} = a_{i,j} - a_{i,k}a_{k,j}$
    end for
  end for
end for

时间复杂度 $O(n^3)$
可以设计对应的并行算法

主元选择 (Pivoting)

- 当矩阵在高斯消元的某一步中对角线元素接近0，会造成数值不稳定
- 可以在高斯消元的过程中，每次选择当前列中绝对值最大的对角元为主元，去消去其他行
- 这样只要矩阵满秩，我们就不会出现除小量的情况
- 带这种选主元方法的 Gauss 消去法就称为 列主元 Gauss 消去法 或 部分选主元 Gauss 消去法 (Gaussian Elimination with Partial Pivoting, GEPP)

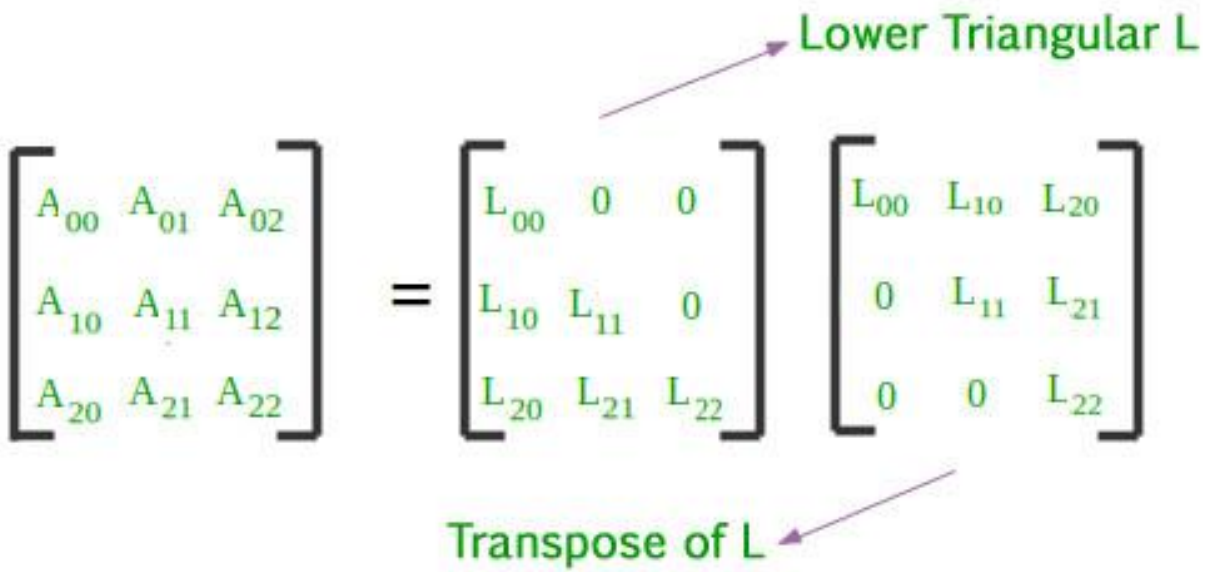


科列斯基分解 (Cholesky Decomposition)

对于对称半正定的矩阵，LU分解简化为科列斯基分解，不需要进行Pivoting操作：

$$A = LL^T$$

→ 也可以称为LLT分解



$$\begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{00} & 0 & 0 \\ L_{10} & L_{11} & 0 \\ L_{20} & L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} L_{00} & L_{10} & L_{20} \\ 0 & L_{11} & L_{21} \\ 0 & 0 & L_{22} \end{bmatrix}$$

科列斯基分解

分解过程：

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T = \begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} L_{11} & L_{21} & L_{31} \\ 0 & L_{22} & L_{32} \\ 0 & 0 & L_{33} \end{pmatrix}$$

$$= \begin{pmatrix} L_{11}^2 & & (\text{symmetric}) \\ L_{21}L_{11} & L_{21}^2 + L_{22}^2 & \\ L_{31}L_{11} & L_{31}L_{21} + L_{32}L_{22} & L_{31}^2 + L_{32}^2 + L_{33}^2 \end{pmatrix}$$

$$\mathbf{L} = \begin{pmatrix} \sqrt{A_{11}} & 0 & 0 \\ A_{21}/L_{11} & \sqrt{A_{22} - L_{21}^2} & 0 \\ A_{31}/L_{11} & \left(A_{32} - L_{31}L_{21}\right)/L_{22} & \sqrt{A_{33} - L_{31}^2 - L_{32}^2} \end{pmatrix}$$

科列斯基分解算法

Algorithm 2 The Cholesky decomposition of a symmetric positive definite matrix A. Upon exit, the entries of A on its diagonal and below it have been overwritten with the entries of the lower triangular Cholesky factor F.

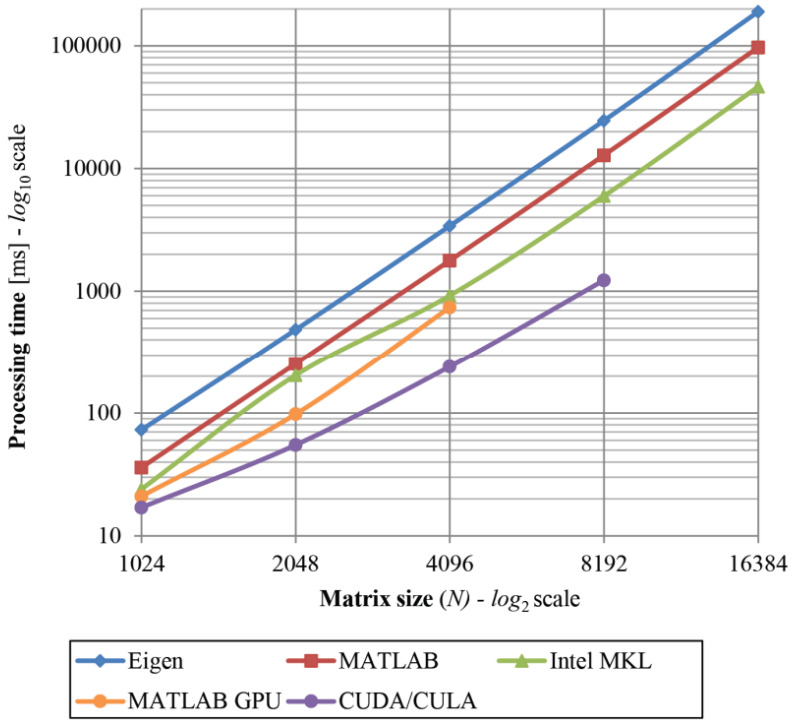
for $k = 1, \ldots, n$ **do**
  $a_{k,k} = \sqrt{a_{k,k}}$
  for $i = k+1, \ldots, n$ **do**
    $a_{i,k} = \frac{a_{i,k}}{a_{k,k}}$
  end for
  for $j = k+1, \ldots, n$ **do**
    for $i = j, \ldots, n$ **do**
      $a_{i,j} = a_{i,j} - a_{i,k}a_{k,j}$
    end for
  end for
end for

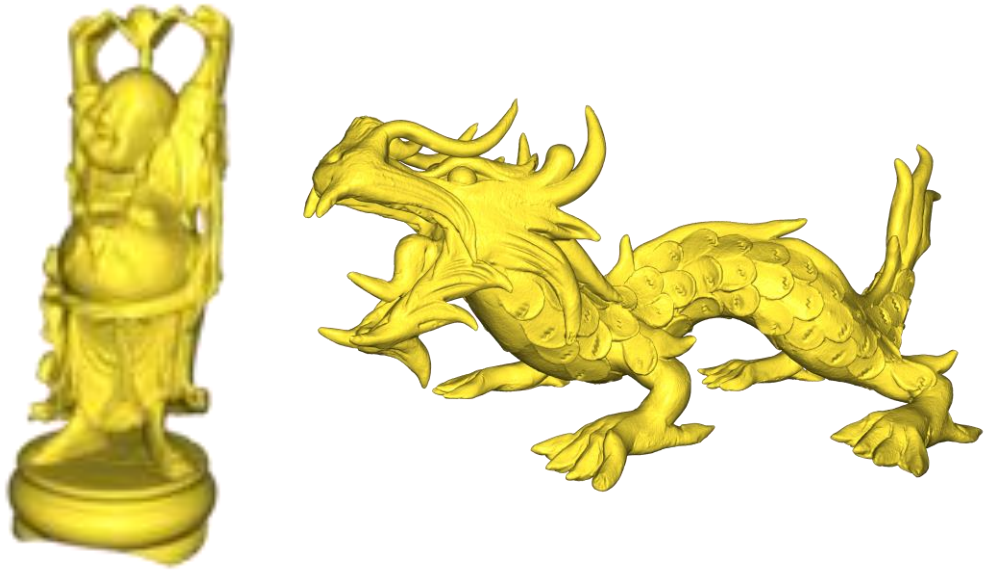
时间复杂度 $O(n^3)$

可以设计对应的并行算法

直接求解法

- LU分解可以求解任意矩阵方程，LLT分解用于求解对称半正定矩阵
- 由于$O(n^3)$的复杂度，一般适用于求解小矩阵的方程
- 但是在图形学中，我们遇到的矩阵往往有非常大的规模…





>100k个顶点

迭代求解
Iterative Solver

迭代方法

从一个初始猜解（Initial Guess）x^0 出发，反复执行某个步骤

$$x^{k+1} \leftarrow \Psi(x^k)$$

在迭代次数达到上限，或者残差达到阈值时终止

$$\|r^k\| = \|b - \mathrm{A}x^k\| < \varepsilon$$

- 可以随时拿到一个差不多的解，而不需要像直接求解中那样等到全部分解完，并且很多时候一个差不多的解就够用了
- 可以充分利用初始猜解大大减少需要的迭代次数
- 针对特定问题优化的迭代算法速度明显优于直接求解
- 迭代算法可以推广到矩阵A并没有显式给出的情况：只提供一个接口输入x输出$\mathrm{A}x$
- …

加速效果

使用优化的迭代求解我们可以在几十毫秒内实时求解60万维的矩阵方程，而直接求解可能需要几十分钟

A Scalable Galerkin Multigrid Method for Real-time Simulation of Deformable Objects

ZANGYUEYANG XIAN*, Shanghai Jiao Tong University and Microsoft Research Asia
XIN TONG, Microsoft Research Asia
TIANTIAN LIU, Microsoft Research Asia

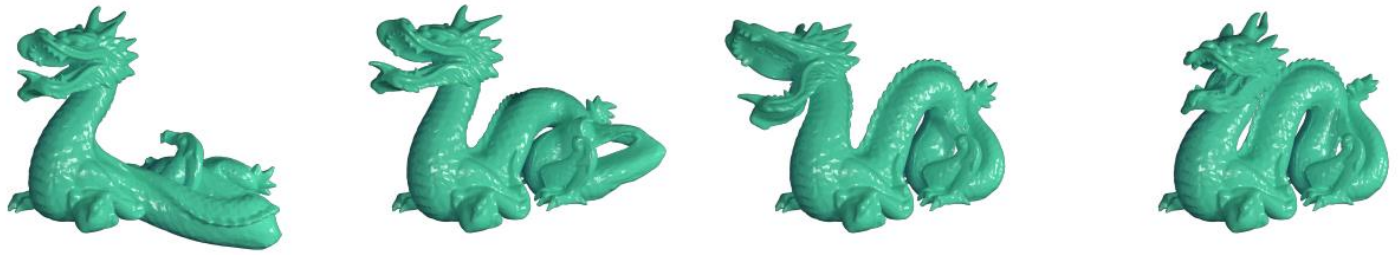


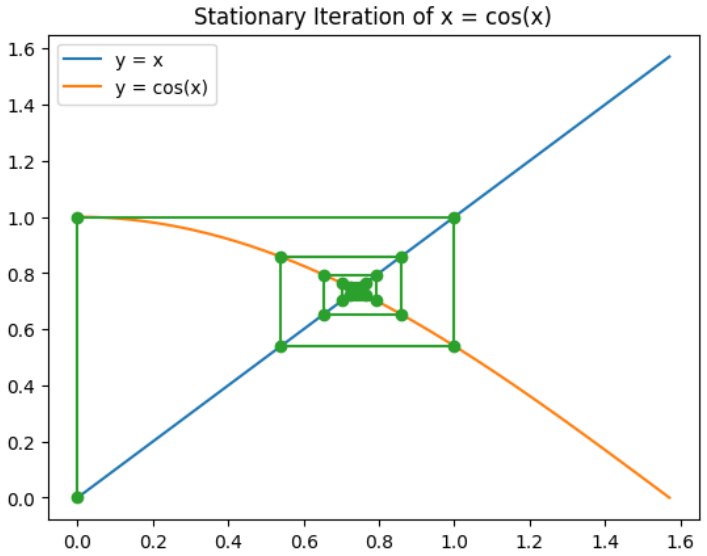
Fig. 1. Our multigrid method simulates a deformable dragon model with 200094 vertices and 676675 elements in its full space at 39.4 frames per second.

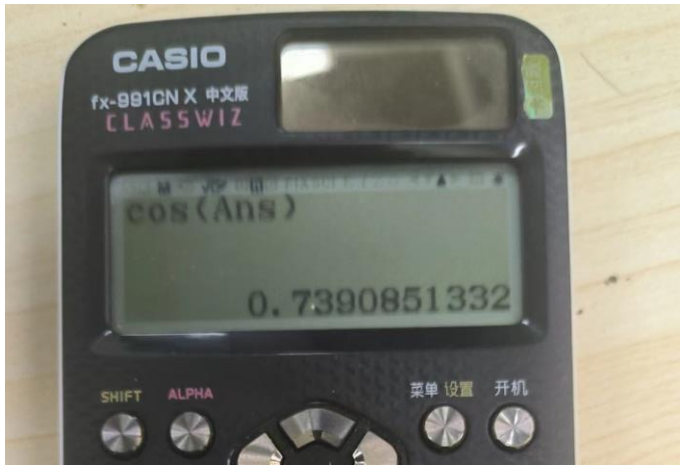
第一类：不动点迭代 (Stationary Iteration)

一个不动点迭代的简单例子：

求解 $x - \cos x = 0$

- $x_0 = 0$
- $x_1 = \cos x_0 = 1$
- $x_2 = \cos x_1 = 0.5403$
- $x_3 = \cos x_2 = 0.8575$
- $x_4 = \cos x_3 = 0.6542$
- $x_5 = \cos x_4 = 0.7934$
- $\cdots$
- $x_n = 0.7391$





第一类：不动点迭代 (Stationary Iteration)

将A改写为$A = M - N$，于是有

$$Ax = Mx - Nx = b$$
$$x = M^{-1}(Nx + b)$$

上面这个形式可以变为下面的不动点迭代

$$x^{k+1} \leftarrow M^{-1}(Nx^k + b)$$

可以反推回去验证，如果迭代收敛，x^k趋向于固定值x^*，那么一定有$Ax^* = b$

- M^{-1}如何计算，还是得直接求解吗?
- 如何保证迭代收敛?
- 收敛效率与什么有关?

雅可比迭代 (Jacobi Iteration)

- 我们取M为A的对角部分D, 则剩下的N为A的非对角部分的相反数

$$x^{k+1} \leftarrow \mathrm{D}^{-1}(\mathrm{N}x^{k} + b)$$

- 示例：

$$\mathrm{A} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}, \mathrm{D} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \mathrm{N} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

- 对角矩阵D的逆矩阵是非常好求的，因此每次迭代的时候我们只需要进行矩阵向量乘法和向量加法就可以

雅可比迭代

算法 Jacobi 迭代算法

1: Given an initial guess $x^{(0)}$

2: **while** not converge **do**　% 停机准则

3: **for** $i = 1$ to n **do**

4: $$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right)$$

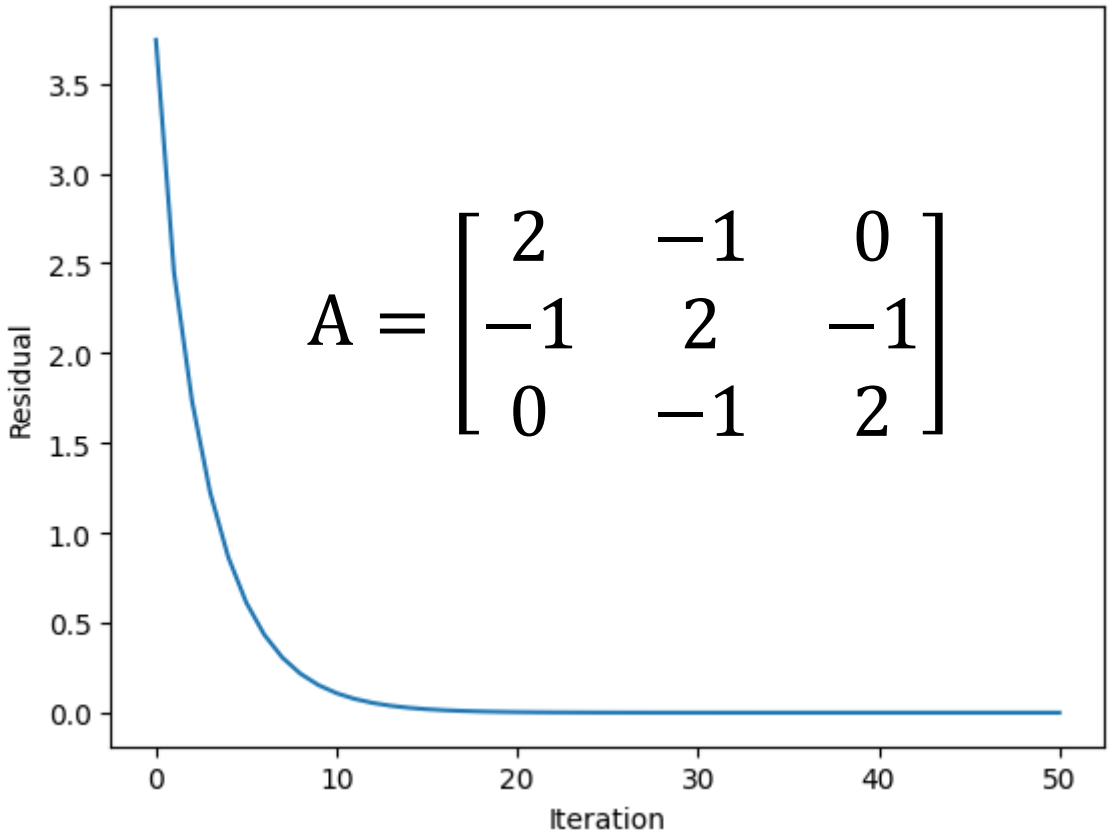
5: **end for**

6: **end while**

雅可比迭代：收敛曲线

$$x^{k+1} \leftarrow \mathrm{D}^{-1}(\mathrm{N}x^{k} + b)$$

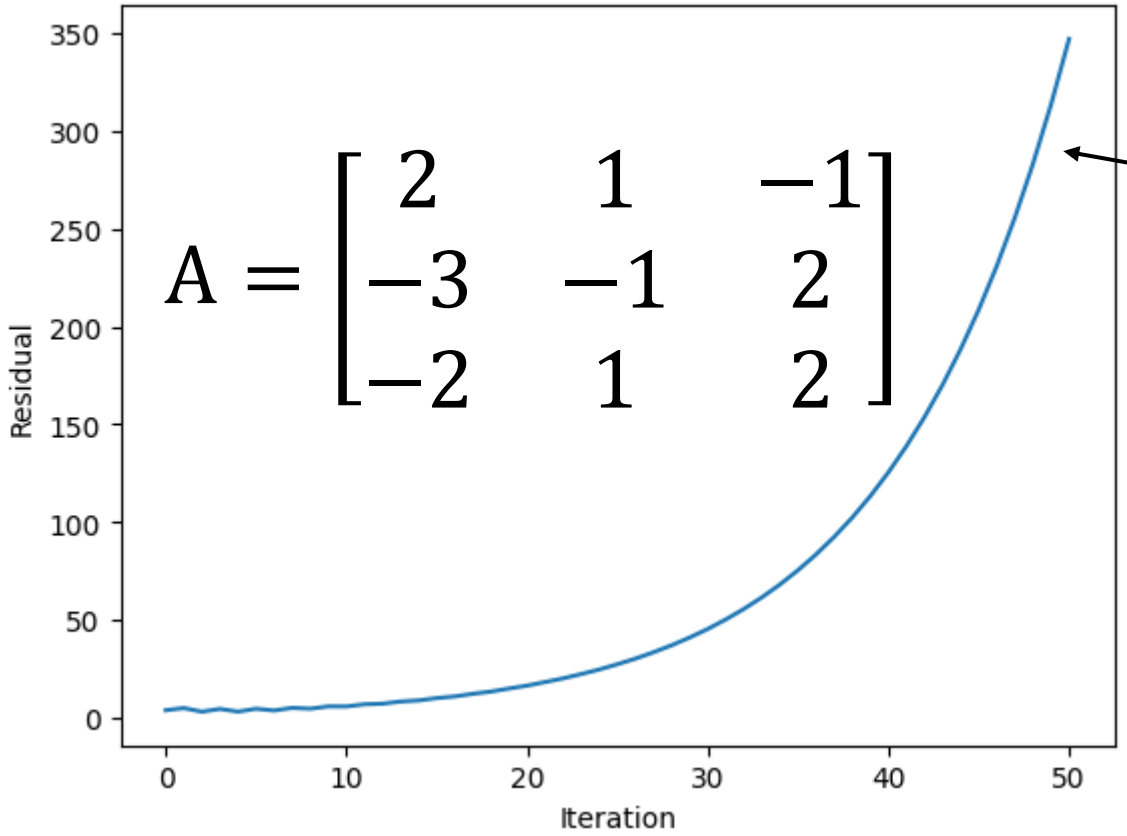
$$\|b - \mathrm{A}x^{k}\|$$



$$\mathrm{A} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

雅可比迭代：失败情况

$$A = \begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix}$$



数值发散!

不动点迭代收敛准则

$$x^{k+1} \leftarrow \mathrm{M}^{-1}\left(\mathrm{N}x^{k} + b\right)$$

- 上面的形式可以改写为：

$$x^{k+1} \leftarrow x^{k} + \mathrm{M}^{-1}\left(b - \mathrm{A}x^{k}\right)$$

- 定义误差（error）e^k为x^k与真实解$x^* = A^{-1}b$之间的差，其与残差（residual）r^k的关系为：

$$r^{k} = b - \mathrm{A}x^{k} = \mathrm{A}\left(x^{*} - x^{k}\right) = \mathrm{A}e^{k}$$

- 我们可以推出下面的误差更新关系：

$$x^{k+1} \leftarrow x^{k} + \mathrm{M}^{-1}\mathrm{A}e^{k}$$

$$e^{k+1} \leftarrow (\mathrm{I} - \mathrm{M}^{-1}\mathrm{A})e^{k} = \mathrm{M}^{-1}\mathrm{N}e^{k}$$

不动点迭代收敛准则

$$e^{k+1} \leftarrow (\mathrm{I} - \mathrm{M}^{-1}\mathrm{A})e^{k} = \mathrm{M}^{-1}\mathrm{N}e^{k}$$

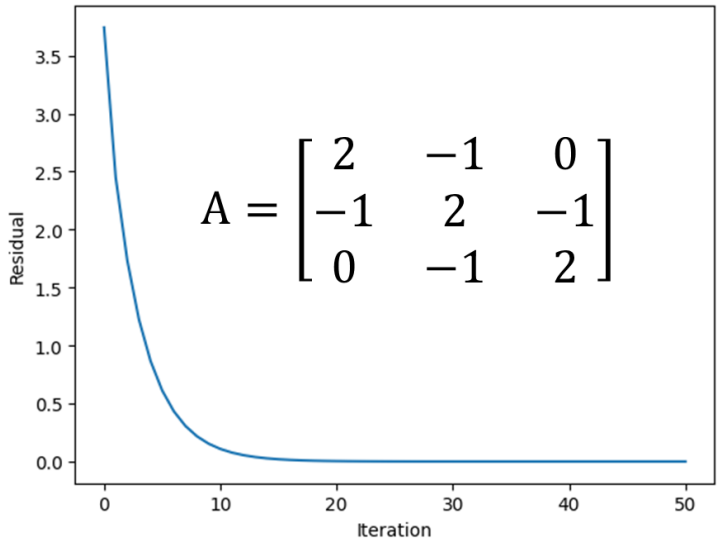
- 在迭代的每一步，我们在误差前面乘上了一个矩阵T
$$\mathrm{T} = \mathrm{I} - \mathrm{M}^{-1}\mathrm{A} = \mathrm{M}^{-1}\mathrm{N}$$
- 我们可以对T进行特征值分解$\mathrm{T} = \mathrm{Q}\Lambda\mathrm{Q}^{-1}$，可以得到只有当T的所有特征值（包括复数特征值）的模长小于1时（也即谱半径小于1），反复迭代可以保证e^{k}最终可以收敛到0
- 而一旦T的特征值中有模长大于1的值，就可能导致不收敛
- 并且对于T特征值越小的分量，误差收敛的越快
- 如果$\mathrm{M} = \mathrm{A}$，$\mathrm{T} = 0$代表一步就能收敛的最优情况，因此我们应该让M尽可能近似A，同时M^{-1}可以方便求解

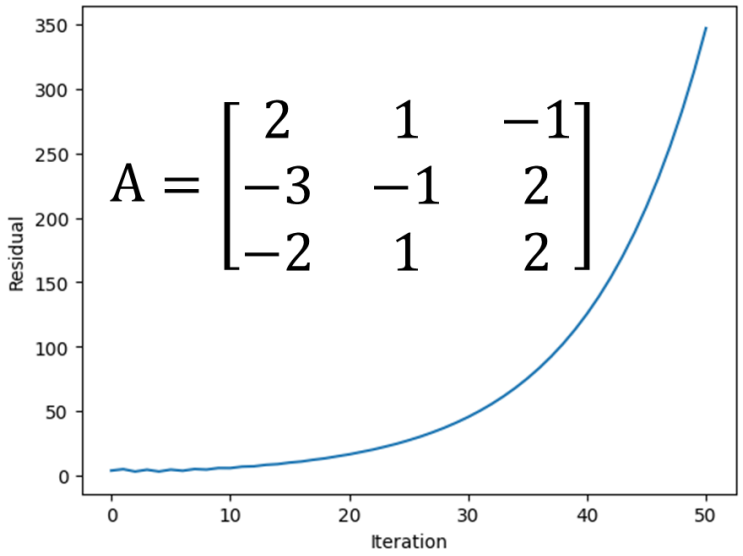
雅可比迭代收敛充分条件

- 判断T的谱半径是否小于1并不直接，针对雅可比迭代还有一个简单的充分但不必要条件：如果矩阵是对角占优的：

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|$$

那么雅可比迭代一定收敛

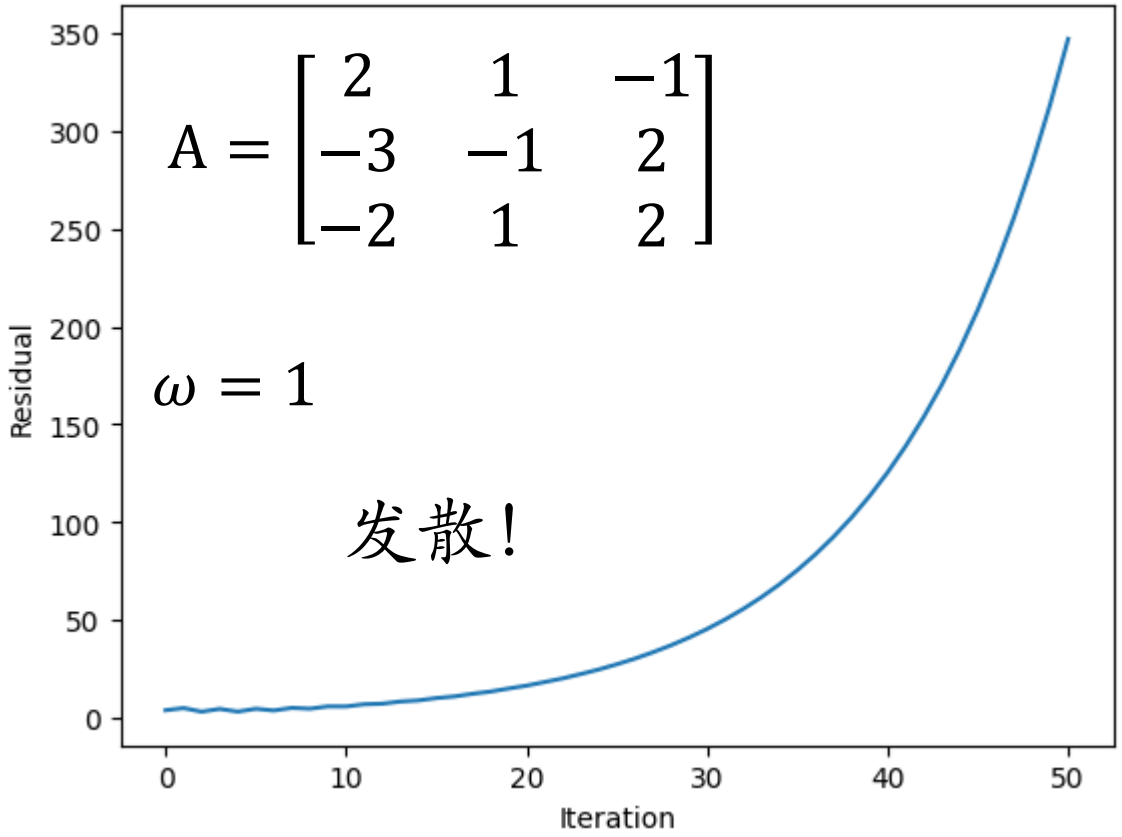


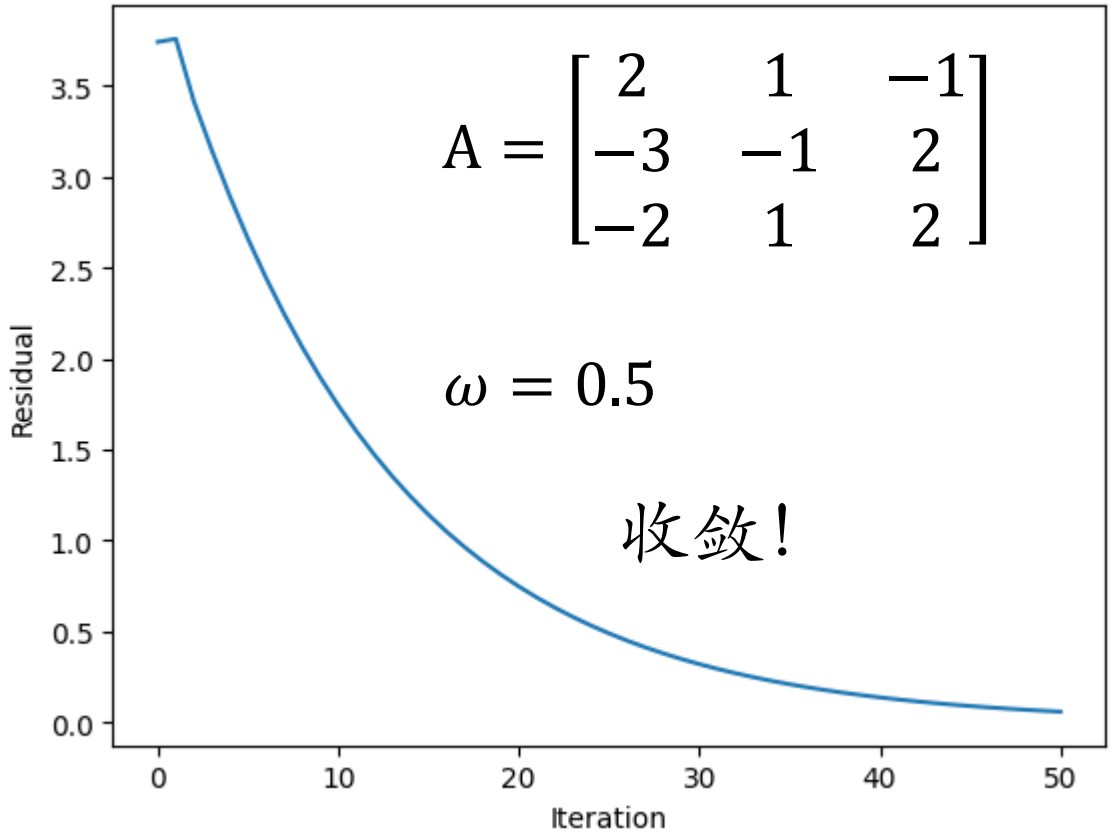


松弛 (Relaxation)

$$x^{k+1} \leftarrow x^k + \omega \mathrm{M}^{-1}\left(b - \mathrm{A}x^k\right)$$

- 每次更新的步长都乘以一个ω的常数因子





雅可比迭代速度

- 雅可比迭代每一步的代价非常低，非常利于并行
- 但是总体来说雅可比迭代需要很多很多步才能收敛，比如在我们展示的3×3矩阵情况中，雅可比迭代需要10多步才能收敛，如果矩阵到达几万维时，需要的迭代次数也是万级以上的
- 考虑迭代矩阵 $\mathrm{T} = \mathrm{I} - \mathrm{D}^{-1}\mathrm{A}$，如果A是对角矩阵，那么$\mathrm{T} = 0$，意味着我们只需要一步就可以收敛
- 一般来说，如果A矩阵越是对角占优的，那么雅可比迭代收敛越快

高斯-赛德尔迭代 (Gauss-Seidel Iteration)

- 与雅可比迭代不同，高斯-赛德尔迭代(GS迭代)使用A的对角和下三角部分作为M

$$x^{k+1} \leftarrow x^k + \mathrm{M}^{-1}\left(b - \mathrm{A}x^k\right), \mathrm{M} = \mathrm{D} + \mathrm{L}$$

- 注意到M^{-1}乘以向量可以利用M是下三角矩阵的特性，用高斯消元法从第一行开始求解
- 由于GS迭代中M与A更为接近，因此其迭代矩阵 $\mathrm{T} = \mathrm{I} - \mathrm{M}^{-1}\mathrm{A}$ 比雅可比迭代的性质要更好，收敛步数更少，但是相应每一步的代价更高
- GS收敛的充分但不必要条件是A矩阵是对称半正定的

高斯-赛德尔迭代 (Gauss-Seidel Iteration)

算法 Gauss-Seidel 迭代算法

1: Given an initial guess $x^{(0)}$
2: **while** not converge **do**
3: **for** $i = 1$ to n **do**
4: $$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right)$$
5: **end for**
6: **end while**

算法 Jacobi 迭代算法

1: Given an initial guess $x^{(0)}$
2: **while** not converge **do** % 停机准则
3: **for** $i = 1$ to n **do**
4: $$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right)$$
5: **end for**
6: **end while**

对比算法可以发现，GS迭代和Jacobi迭代最大的区别在于GS迭代在更新时考虑了当前步之前更新的结果，而Jacobi迭代只使用了上一步的结果

不动点迭代总结

- Jacobi迭代，GS迭代的最大优势在于实现简单，但是收敛速度并不快
- Jacobi迭代和GS迭代都有可能不收敛，要分析矩阵的具体情况，并选择合适的松弛变量ω
- 通过分析不动点迭代的收敛特性，我们可以不断用小规模的矩阵近似A矩阵，构造多层的矩阵方程进行求解，也就是多重网格求解器（Multigrid Solver），其在很多情况下都是速度最快的大规模矩阵求解器

第二类：子空间迭代 (Subspace Iteration)

- $Ax = b$的求解问题可以看成是$\min\|Ax - b\|$的优化问题
- 优化的整个空间为$\mathbb{R}^n$，假设其一组基为$(e_1, \cdots, e_n)$
- 子空间迭代的基本想法是在第m步的求解中，我们只在$(e_1, \cdots, e_m)$张成的m维子空间上考虑这个优化问题，得到优化的解为x^m
- 如果基构造的方式满足某些特性，那么我们可以通过少量计算改进x_{m-1}的方式得到x_m
- 这样最多需要n步我们就可以得到最终的解

Krylov 子空间

- 对于矩阵方程$\mathrm{A}x = b$，Krylov子空间定义为
$$K_r(\mathrm{A}, b) = span\{b, \mathrm{A}b, \cdots, \mathrm{A}^{r-1}b\}$$
- K_r是r维的子空间，并且满足嵌套关系
$$K_1 \subset K_2 \subset K_3 \subset \cdots$$
- 如果矩阵满秩，那么K_n就是全空间$\mathbb{R}^n$，之后的$K_{n+1}, K_{n+2}, \cdots$不会增加维度
- 使用Krylov子空间进行迭代，第m步就是在K_m中寻找最优的x_m

Aleksey Nikolaevich Krylov

共轭梯度下降 (Conjugate Gradient)

- 共轭梯度下降(CG)是Krylov子空间迭代在对称半正定矩阵中的形式，其优化的是

$$\min f(x) = \frac{1}{2} x^T \mathrm{A} x - b^T x$$

- 对应梯度为

$$\nabla f = Ax - b = -r$$

- 要想x_m在K_m中最小化$f(x)$，那么就是要求

$$\nabla f = -r_m \perp K_m$$

共轭梯度下降 (Conjugate Gradient)

共轭梯度下降的核心：

- 如果我们能构造Krylov子空间的一系列共轭基p_m满足

$$K_m = span\{p_1, \cdots, p_m\}, \qquad p_i^T A p_j = 0 \ \forall i \neq j$$

- 那么已知x_{m-1}求x_m的过程就是在p_m的方向上求最小化的过程

$$x_m = x_{m-1} + \alpha_m p_m, \qquad \alpha_m = \arg\min f(x_{m-1} + \alpha_m p_m)$$

- 求α_m问题是一维的二次函数优化问题，可以有显式解

为什么？

共轭梯度下降 (Conjugate Gradient)

- 我们归纳地证明$r_m \perp K_m$
- 开始时，$K_1 = span\{p_1\}$，直接求解一维优化问题，保证$r_1 \perp K_1$
- 如果$r_{m-1} \perp K_{m-1}$，考虑r_m与K_m的关系

$$r_m = b - \mathrm{A}(x_{m-1} + \alpha_m p_m) = r_{m-1} - \alpha_m \mathrm{A} p_m$$

- 由于x_m在p_m的方向上最小化了$f(x)$，因此$r_m \perp p_m$
- 由于$r_{m-1} \perp K_{m-1}$以及$\mathrm{A}p_m \perp K_{m-1}$（共轭），因此$r_m \perp K_{m-1}$
- 可得

$$r_m \perp K_{m-1} \cup \{p_m\} = K_m$$

- 换言之，我们证明了只要能够构造出共轭基$\{p_1, \cdots p_n\}$，CG算法就可以进行下去

共轭梯度下降 (Conjugate Gradient)

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
if $\mathbf{r}_0$ is sufficiently small, then return $\mathbf{x}_0$ as the result
$\mathbf{p}_0 := \mathbf{r}_0$
$k := 0$
repeat
$\alpha_k := \frac{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k}$
$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$
$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$
if $\mathbf{r}_{k+1}$ is sufficiently small, then exit loop
$\beta_k := \frac{\mathbf{r}_{k+1}^\mathsf{T}\mathbf{r}_{k+1}}{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}$
$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$
$k := k + 1$
end repeat
return $\mathbf{x}_{k+1}$ as the result

求解在p_k方向的一维优化问题

构造p_{k+1}

共轭梯度下降的速度

- 共轭梯度下降法保证能在n步能收敛到最终解，但是一般不需要跑太多迭代误差就能收敛到阈值，对于常见的几万维的矩阵一般只需要1000步就差不多
- 并且有下面的收敛速度保证

$$\frac{\|x_k - x^*\|}{\|x_0 - x^*\|} \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^k$$

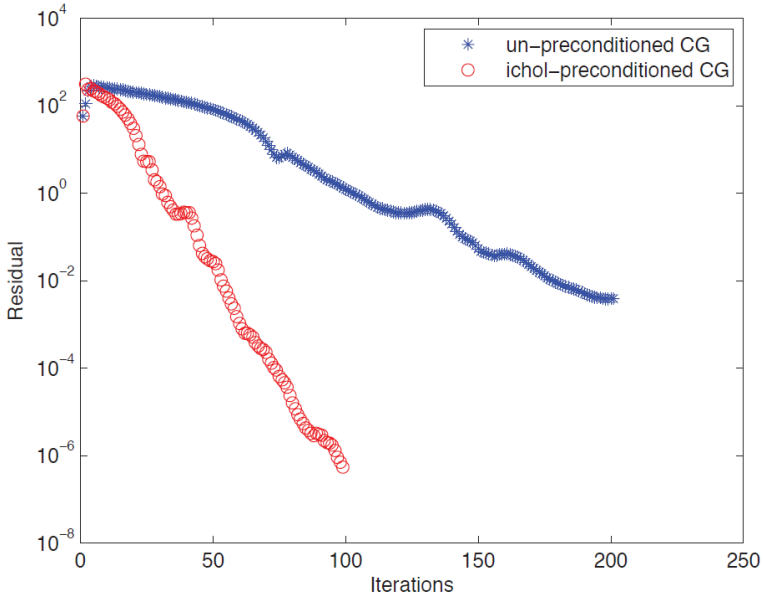
- 其中κ是矩阵的条件数（condition number），定义为A的最大特征值与最小特征值的比值，保证$\kappa \geq 1$
- 可以发现当条件数越接近1时，收敛速度越快，条件数越大，收敛速度越慢

预处理 (Preconditioning)

- 提高共轭梯度下降的收敛速度，核心在于改善矩阵的条件数
- 如果我们能找到A^{-1}的近似矩阵$\mathrm{M} = \mathrm{P}^{\mathrm{T}}\mathrm{P}$，转而求解

$$\mathrm{PAP}^{\mathrm{T}}x = \mathrm{P}b$$

- 那么$\mathrm{PAP}^{\mathrm{T}}$的条件数一定更小，因为在极端情况$\mathrm{M} = \mathrm{A}^{-1}$时，$\mathrm{PAP}^{\mathrm{T}} = \mathrm{I}$的条件数是1
- 常见的选择M的方式包括对角矩阵 (Diagonal Preconditioned CG, DPCG), 不完全的Cholesky分解 (Incomplete Cholesky Preconditioned CG, ICPCG)等



其他Krylov子空间求解器

- 注意CG只试用于对称正定的矩阵，对于其他情况
- MINRES (Minimal Residual): 求解一般对称矩阵
- GMRES (Generalized Minimal Residual): 求解一般矩阵
- BiCG (Biconjugate Gradients): 求解一般的矩阵
- …

总结

- 对于小矩阵(<1000维)，通常直接求解是最快的，比如LU分解，LLT分解（矩阵对称正定）
- 当矩阵规模比较大时，迭代求解器往往更有优势
- 不动点迭代（Jacobi，GS）优点是实现简单，但是收敛速度并不快，一般需要调节松弛系数平衡稳定性和收敛速度
- 不动点迭代可以改进为多重网格方法，可以获得极大的速度提升
- Krylov子空间迭代收敛速度一般很快，不同的矩阵需要对应使用不同的方法，比如CG，MINRES，GMRES，BiCG…
- 对矩阵进行预处理能够大大提高收敛速度，比如DPCG，ICPCG…

参考内容

- https://math.ecnu.edu.cn/~jypan/Teaching/NA/
- https://www.mathworks.com/help/matlab/ref/
- An Introduction to the Conjugate Gradient Method Without the Agonizing Pain
- A Sampler of Useful Computational Tools for Applied Geometry, Computer Graphics, and Image Processing

北京大学
PEKING UNIVERSITY

谢谢