Applied Mathematics 205

Unit V: Eigenvalue Problems

Lecturer: Dr. David Knezevic

Unit V: Eigenvalue Problems

Chapter V.4: Krylov Subspace Methods

Krylov Subspace Methods

In this chapter we give an overview of the role of Krylov[1] subspace methods in Scientific Computing

Given a matrix A and vector b, a Krylov sequence is the set of vectors

$$\{b, Ab, A^2b, A^3b, \ldots\}$$

The corresponding Krylov subspaces are the spaces spanned by successive groups of these vectors

$$\mathcal{K}_m(A, b) \equiv \text{span}\{b, Ab, A^2b, \ldots, A^{m-1}b\}$$

[1] Aleksey Krylov, 1863–1945, wrote a paper on this idea in 1931

Krylov Subspace Methods

Krylov subspaces are the basis for iterative methods for eigenvalue problems (and also for solving linear systems)

An important advantage: Krylov methods do not deal directly with A, but rather with matrix-vector products involving A

This is particularly helpful when A is large and sparse, since then "matvecs" are relatively cheap

Also, Krylov sequence is closely related to power iteration, hence not surprising it is useful for solving eigenproblems

Arnoldi Iteration

Arnoldi Iteration

We know (from Abel and Galois) that we can't transform a matrix to triangular form via finite sequence of similarity transformations

However, it is possible to "similarity transform" any matrix to Hessenberg form

- A is called upper-Hessenberg if $a_{ij} = 0$ for all $i > j + 1$
- A is called lower-Hessenberg if $a_{ij} = 0$ for all $j > i + 1$

The Arnoldi iteration is a Krylov subspace iterative method that reduces A to upper-Hessenberg form

As we'll see, we can then use this simpler form to approximate some eigenvalues of A

Arnoldi Iteration

For $A \in \mathbb{C}^{n \times n}$, we want to compute $A = QHQ^*$, where H is upper Hessenberg and Q is unitary (i.e. $QQ^* = \mathrm{I}$)

However, we suppose that n is huge! Hence we do not try to compute the "full factorization"

Instead, let us consider just the first $m \ll n$ columns of the factorization $AQ = QH$

Therefore, on the left-hand side, we only need the matrix $Q_m \in \mathbb{C}^{n \times m}$:

$$Q_m = \left[\begin{array}{c|c|c|c} & & & \\ q_1 & q_2 & \cdots & q_m \\ & & & \end{array}\right]$$

Arnoldi Iteration

On the right-hand side, we only need the first m columns of H

More specifically, due to upper-Hessenberg structure, we only need $\widetilde{H}_m$, which is the $(m+1) \times m$ upper-left section of H:

$$\widetilde{H}_m = \begin{bmatrix} h_{11} & & \cdots & h_{1m} \\ h_{21} & h_{22} & & \\ & \ddots & \ddots & \vdots \\ & & h_{m,m-1} & h_{mm} \\ & & & h_{m+1,m} \end{bmatrix}$$

$\widetilde{H}_m$ only interacts with the first $m+1$ columns of Q, hence we have

$$AQ_m = Q_{m+1}\widetilde{H}_m$$

Arnoldi Iteration

$$\begin{bmatrix} & A & \end{bmatrix} \begin{bmatrix} q_1 & \cdots & q_m \end{bmatrix} = \begin{bmatrix} q_1 & \cdots & q_{m+1} \end{bmatrix} \begin{bmatrix} h_{11} & \cdots & h_{1m} \\ h_{21} & \cdots & h_{2m} \\ & \ddots & \vdots \\ & & h_{m+1,m} \end{bmatrix}$$

The m^{th} column can be written as

$$Aq_m = h_{1m}q_1 + \cdots + h_{mm}q_m + h_{m+1,m}q_{m+1}$$

Or, equivalently

$$q_{m+1} = (Aq_m - h_{1m}q_1 - \cdots - h_{mm}q_m)/h_{m+1,m}$$

Arnoldi iteration is just the Gram-Schmidt method that constructs the h_{ij} and the (orthonormal) vectors q_j, $j = 1, 2, \ldots$

Arnoldi Iteration

1: choose b arbitrarily, then $q_1 = b/\|b\|_2$
2: **for** $m = 1, 2, 3, \ldots$ **do**
3: $\quad v = Aq_m$
4: $\quad$ **for** $j = 1, 2, \ldots, m$ **do**
5: $\quad\quad h_{jm} = q_j^* v$
6: $\quad\quad v = v - h_{jm} q_j$
7: $\quad$ **end for**
8: $\quad h_{m+1,m} = \|v\|_2$
9: $\quad q_{m+1} = v/h_{m+1,m}$
10: **end for**

This is akin to the modified Gram-Schmidt method because the updated vector v is used in line 5 (vs. the "raw vector" Aq_m)

Also, we only need to evaluate Aq_m and perform some vector operations in each iteration

Arnoldi Iteration

The Arnoldi iteration is useful because the q_j form orthonormal bases of the successive Krylov spaces

$$\mathcal{K}_m(A, b) = \text{span}\{b, Ab, \ldots, A^{m-1}b\} = \text{span}\{q_1, q_2, \ldots, q_m\}$$

We expect $\mathcal{K}_m(A, b)$ to provide good information about the dominant eigenvalues/eigenvectors of A

Note that this looks similar to the QR algorithm, but QR algorithm was based on QR factorization of

$$\left[\begin{array}{c|c|c|c} & & & \\ A^k e_1 & A^k e_2 & \ldots & A^k e_n \\ & & & \end{array}\right]$$

Arnoldi Iteration

Question: How do we find eigenvalues from the Arnoldi iteration?

Let $H_m = Q_m^* A Q_m$ be the $m \times m$ matrix obtained by removing the last row from $\widetilde{H}_m$

Answer: At each step m, we compute the eigenvalues of the Hessenberg matrix H_m (via, say, the QR algorithm)[2]

This provides estimates for m eigenvalues/eigenvectors ($m \ll n$) called Ritz values, Ritz vectors, respectively

Just as with the power method, the Ritz values will typically converge to extreme eigenvalues of the spectrum

[2] This is how `eigs` in Matlab works

Arnoldi Iteration

We now examine why eigenvalues of H_m approximate extreme eigenvalues of A

Let[3] $\mathbb{P}^m_{\text{monic}}$ denote the monic polynomials of degree m

Theorem: The characteristic polynomial of H_m is the unique solution of the approximation problem: find $p \in \mathbb{P}^m_{\text{monic}}$ such that

$$\|p(A)b\|_2 = \text{minimum}$$

Proof: See Trefethen & Bau

[3] Recall that a monic polynomial has coefficient of highest order term of 1

Arnoldi Iteration

This theorem implies that Ritz values (i.e. eigenvalues of H_m) are the roots of the optimal polynomial

$$p^* = \arg \min_{p \in \mathbb{P}^m_{\text{monic}}} \|p(A)b\|_2$$

Now, let's consider what p^* should "look like" in order to minimize $\|p(A)b\|_2$

We can illustrate the important ideas with a simple case, suppose:

- A has only m ($\ll n$) distinct eigenvalues
- $b = \sum_{j=1}^{m} \alpha_j v_j$, where v_j is an eigenvector corresponding to λ_j

Arnoldi Iteration

Then, for $p \in \mathbb{P}^m_{\text{monic}}$, we have

$$p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + x^m$$

for some coefficients $c_0, c_1, \ldots, c_{m-1}$

Applying this polynomial to a matrix A gives

$$\begin{aligned}
p(A)b &= \left(c_0 \mathrm{I} + c_1 A + c_2 A^2 + \cdots + A^m\right) b \\
&= \sum_{j=1}^{m} \alpha_j \left(c_0 \mathrm{I} + c_1 A + c_2 A^2 + \cdots + A^m\right) v_j \\
&= \sum_{j=1}^{m} \alpha_j \left(c_0 + c_1 \lambda_j + c_2 \lambda_j^2 + \cdots + \lambda_j^m\right) v_j \\
&= \sum_{j=1}^{m} \alpha_j p(\lambda_j) v_j
\end{aligned}$$

Arnoldi Iteration

Then the polynomial $p^* \in \mathbb{P}^m_{\text{monic}}$ with roots at $\lambda_1, \lambda_2, \ldots, \lambda_m$ minimizes $\|p(A)b\|_2$, since $\|p^*(A)b\|_2 = 0$

Hence, in this simple case the Arnoldi method finds p^* after m iterations

The Ritz values after m iterations are then exactly the m distinct eigenvalues of A

Arnoldi Iteration

Suppose now that there are more than m distinct eigenvalues (as is generally the case in practice)

It is intuitive that in order to minimize $\|p(A)b\|_2$, p^* should have roots close to the dominant eigenvalues of A

Also, we expect Ritz values to converge more rapidly for extreme eigenvalues that are well-separated from the rest of the spectrum

(We'll see a concrete example of this for a symmetric matrix A shortly)

Lanczos Iteration

Lanczos Iteration

Lanczos iteration is the Arnoldi iteration in the special case that A is hermitian

However, we obtain some significant computational savings in this special case

Let us suppose for simplicity that A is symmetric with real entries, and hence has real eigenvalues

Then $H_m = Q_m^T A Q_m$ is also symmetric $\implies$ Ritz values (i.e. eigenvalue estimates) are also real

Lanczos Iteration

Also, we can show that H_m is tridiagonal: Consider the ij entry of H_m, $h_{ij} = q_i^T A q_j$

Recall first that $\{q_1, q_2, \ldots, q_j\}$ is an orthonormal basis for $\mathcal{K}_j(A, b)$

Then we have $Aq_j \in \mathcal{K}_{j+1}(A, b) = \operatorname{span}\{q_1, q_2, \ldots, q_{j+1}\}$, and hence $h_{ij} = q_i^T (Aq_j) = 0$ for $i > j + 1$ since

$$q_i \perp \operatorname{span}\{q_1, q_2, \ldots, q_{j+1}\}, \text{ for } i > j + 1$$

Also, since H_m is symmetric, we have $h_{ij} = h_{ji} = q_j^T (Aq_i)$, which implies $h_{ij} = 0$ for $j > i + 1$, by the same reasoning as above

Lanczos Iteration

Since H_m is now tridiagonal, we shall write it as

$$T_m = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_{m-1} \\ & & & \beta_{m-1} & \alpha_m \end{bmatrix}$$

The consequence of tridiagonality: Lanczos iteration is much cheaper than Arnoldi iteration!

Lanczos Iteration

The inner loop in Lanczos iteration only runs from $m-1$ to m, instead of 1 to m as in Arnoldi

This is due to the three-term recurrence at step m:

$$Aq_m = \beta_{m-1} q_{m-1} + \alpha_m q_m + \beta_m q_{m+1}$$

(This follows from our discussion of the Arnoldi case, with $\widetilde{T}_m$ replacing $\widetilde{H}_m$)

As before, we rearrange this to give

$$q_{m+1} = (Aq_m - \beta_{m-1} q_{m-1} - \alpha_m q_m)/\beta_m$$

Lanczos Iteration

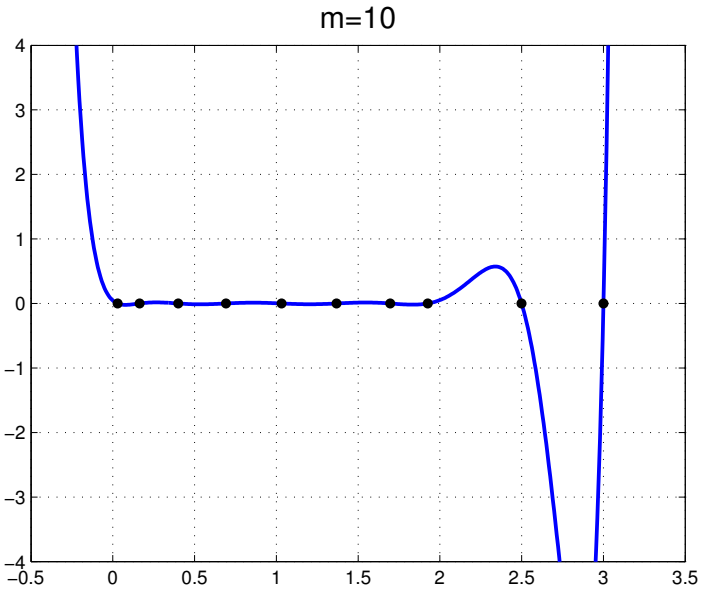
Which leads to the Lanczos iteration

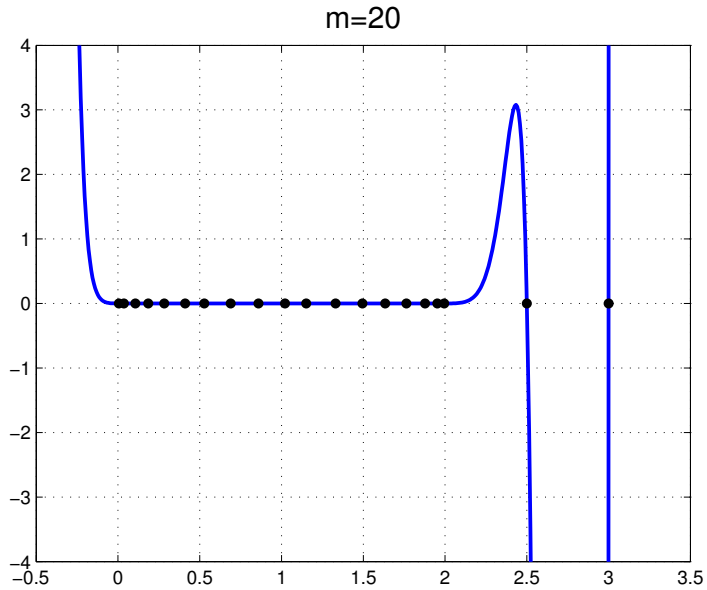
1: $\beta_0 = 0,\ q_0 = 0$
2: choose b arbitrarily, then $q_1 = b/\|b\|_2$
3: **for** $m = 1, 2, 3, \ldots$ **do**
4: $\quad v = Aq_m$
5: $\quad \alpha_m = q_m^T v$
6: $\quad v = v - \beta_{m-1} q_{m-1} - \alpha_m q_m$
7: $\quad \beta_m = \|v\|_2$
8: $\quad q_{m+1} = v/\beta_m$
9: **end for**

Lanczos Iteration

Matlab demo: Lanczos iteration for a diagonal matrix

Lanczos Iteration





We can see that Lanczos minimizes $\|p(A)b\|_2$:

- p is uniformly small in the region of clustered eigenvalues
- roots of p match isolated eigenvalues very closely

Note that in general p will be very steep near isolated eigenvalues, hence convergence for isolated eigenvalues is rapid!

Lanczos Iteration

Matlab demo: Lanczos iteration for smallest eigenpairs of Laplacian on the unit square

(We apply Lanczos iteration to A^{-1} since then the smallest eigenvalues become the largest and (more importantly) isolated)

Conjugate Gradient Method

Conjugate Gradient Method

We now turn to another Krylov subspace method: the conjugate gradient method,[4] or CG

(This is a detour in this Unit since CG is not an eigenvalue algorithm)

CG is an iterative method for solving $Ax = b$ in the case that A is symmetric positive definite (SPD)

CG is the original (and perhaps most famous) Krylov subspace method, and is a mainstay of Scientific Computing

[4] Due to Hestenes and Stiefel in 1952

Conjugate Gradient Method

Recall that we discussed CG in Unit IV — the approach in Unit IV is also often called nonlinear conjugate gradients

Nonlinear conjugate gradients applies the approach of Hestenes and Stiefel to nonlinear unconstrained optimization

Conjugate Gradient Method

Iterative solvers (e.g. CG) and direct solvers (e.g. Gaussian elimination) for solving $Ax = b$ are fundamentally different:

- Direct solvers: In exact arithmetic, gives exact answer after finitely many steps
- Iterative solvers: In principle require infinitely many iterations, but should give accurate approximation after few iterations

Conjugate Gradient Method

Direct methods are very successful, but iterative methods are typically more efficient for very large, sparse systems

Krylov subspace methods only require matvecs and vector-vector (e.g. dot product) operations

Hence Krylov methods require $O(n)$ operations per iteration for sparse A, no issues with "fill-in" etc

Also, iterative methods are generally better suited to parallelization, hence an important topic in supercomputing

Conjugate Gradient Method

The CG algorithm is given by

1: $x_0 = 0,\ r_0 = b,\ p_0 = r_0$
2: **for** $k = 1, 2, 3, \ldots$ **do**
3: $\quad \alpha_k = (r_{k-1}^T r_{k-1})/(p_{k-1}^T A p_{k-1})$
4: $\quad x_k = x_{k-1} + \alpha_k p_{k-1}$
5: $\quad r_k = r_{k-1} - \alpha_k A p_{k-1}$
6: $\quad \beta_k = (r_k^T r_k)/(r_{k-1}^T r_{k-1})$
7: $\quad p_k = r_k + \beta_k p_{k-1}$
8: **end for**

Conjugate Gradient Method

We shall now discuss CG in more detail — it's certainly not obvious upfront why this is a useful algorithm!

Let $x_* = A^{-1}b$ denote the exact solution, and let $e_k \equiv x_* - x_k$ denote the error at step k

Also, let $\|\cdot\|_A$ denote the norm

$$\|x\|_A \equiv \sqrt{x^T A x}$$

Conjugate Gradient Method

Theorem: The CG iterate x_k is the unique member of $\mathcal{K}_k(A, b)$ which minimizes $\|e_k\|_A$. Also, $x_k = x_*$ for some $k \leq n$.

Proof: This result relies on a set of identities which can be derived (by induction) from the CG algorithm:

(i) $\mathcal{K}_k(A, b) = \text{span}\{x_1, x_2, \ldots, x_k\} = \text{span}\{p_0, p_1 \ldots, p_{k-1}\}$
$= \text{span}\{r_0, r_1, \ldots, r_{k-1}\}$

(ii) $r_k^T r_j = 0$ for $j < k$

(iii) $p_k^T A p_j = 0$ for $j < k$

Conjugate Gradient Method

From the first identity above, it follows that $x_k \in \mathcal{K}_k(A, b)$

We will now show that x_k is the unique minimizer in $\mathcal{K}_k(A, b)$

Let $\tilde{x} \in \mathcal{K}_k(A, b)$ be another "candidate minimizer" and let $\Delta x \equiv x_k - \tilde{x}$, then

$$\begin{aligned} \|x_* - \tilde{x}\|_A^2 &= \|(x_* - x_k) + (x_k - \tilde{x})\|_A^2 \\ &= \|e_k + \Delta x\|_A^2 \\ &= (e_k + \Delta x)^T A(e_k + \Delta x) \\ &= e_k^T A e_k + 2e_k^T A\Delta x + \Delta x^T A\Delta x \end{aligned}$$

Conjugate Gradient Method

Next, let $r(x_k) = b - Ax_k$ denote the residual at step k, so that

$$r(x_k) = b - Ax_k = b - A(x_{k-1} + \alpha_k p_{k-1}) = r(x_{k-1}) - \alpha_k A p_{k-1}$$

Since $r(x_0) = b = r_0$, by induction we see that for r_k computed in line 5 of CG,

$$r_k = r_{k-1} - \alpha_k A p_{k-1}$$

we have $r_k = r(x_k),\ k = 1, 2, \ldots$

Conjugate Gradient Method

Now, recall our expression for $\|x_* - \tilde{x}\|_A^2$:

$$\|x_* - \tilde{x}\|_A^2 = e_k^T A e_k + 2e_k^T A \Delta x + \Delta x^T A \Delta x$$

and note that

$$2e_k^T A \Delta x = 2\Delta x^T A(x_* - x_k) = 2\Delta x^T (b - Ax_k) = 2\Delta x^T r_k$$

Now,

- $\Delta x = x_k - \tilde{x} \in \mathcal{K}_k(A, b)$
- from (i), we have that $\mathcal{K}_k(A, b) = \text{span}\{r_0, r_1, \ldots, r_{k-1}\}$
- from (ii), we have that $r_k \perp \text{span}\{r_0, r_1, \ldots, r_{k-1}\}$

Therefore, we have $2e_k^T A \Delta x = 2\Delta x^T r_k = 0$

Conjugate Gradient Method

This implies that,

$$\|x_* - \tilde{x}\|_A^2 = e_k^T A e_k + \Delta x^T A \Delta x \geq \|e_k\|_A^2,$$

with equality only when $\Delta x = 0$, hence $x_k \in \mathcal{K}_k(A, b)$ is the unique minimizer!

This also tells us that if $x_* \in \mathcal{K}_k(A, b)$, then $x_k = x_*$

Therefore[5] CG will converge to x_* in at most n iterations since $\mathcal{K}_k(A, b)$ is a subspace of $\mathbb{R}^n$ of dimension k $\square$

[5] Assuming exact arithmetic!

Conjugate Gradient Method

Note that the theoretical guarantee that CG will converge in n steps is of no practical use

In floating point arithmetic we will not get exact convergence to x_*

More importantly, we assume n is huge, so we want to terminate CG well before n iterations anyway

Nevertheless, the guarantee of convergence in at most n steps is of historical interest

Hestenes and Stiefel originally viewed CG as a direct method that will converge after a finite number of steps

Conjugate Gradient Method

Steps of CG are chosen to give the orthogonality properties (ii), (iii), which lead to the remarkable CG optimality property:

> CG minimizes the error over the Krylov subspace $\mathcal{K}_k(A, b)$ at step k

Question: Where did the steps in the CG algorithm come from?

Answer: It turns out that CG can be derived by developing an optimization algorithm for $\phi : \mathbb{R}^n \to \mathbb{R}$ given by

$$\phi(x) \equiv \frac{1}{2} x^T A x - x^T b$$

e.g. lines 3 and 4 in CG perform line search, line 7 gives a search direction p_k

Conjugate Gradient Method

[Aside: Note that $-\nabla\phi(x) = b - Ax = r(x)$

The name "Conjugate Gradient" then comes from the property

$$\text{(ii)} \quad \nabla\phi(x_k)^T \nabla\phi(x_j) = r_k^T r_j = 0 \text{ for } j < k$$

That is, the gradient directions are orthogonal, or "conjugate"]

Question: Why is the quadratic objective function ϕ relevant to solving $Ax = b$?

Conjugate Gradient Method

Answer: Minimizing ϕ is equivalent to minimizing $\|e_k\|_A^2$, since

$$\begin{aligned} \|e_k\|_A^2 &= (x_* - x_k)^T A(x_* - x_k) \\ &= x_k^T A x_k - 2x_k^T A x_* + x_*^T A x_* \\ &= x_k^T A x_k - 2x_k^T b + x_*^T b \\ &= 2\phi(x_k) + \text{const.} \end{aligned}$$

Hence, our argument from above shows that, at iteration k, CG solves the optimization problem

$$\min_{x \in \mathcal{K}_k(A,b)} \phi(x)$$

Conjugate Gradient Method

An important topic (that we will not cover in detail) is convergence analysis of CG: How fast does $\|e_k\|_A$ converge?

A famous result for CG is that if A has 2-norm condition number κ, then

$$\frac{\|e_k\|_A}{\|e_0\|_A} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k$$

Hence smaller condition number implies faster convergence!

Conjugate Gradient Method

Suppose we want to terminate CG when

$$\frac{\|e_k\|_A}{\|e_0\|_A} \leq \epsilon$$

for some $\epsilon > 0$, how many CG iterations will this require?

We have the identities

$$\begin{aligned} 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k &= 2\left(\frac{\sqrt{\kappa}+(1-1)-1}{\sqrt{\kappa}+1}\right)^k \\ &= 2\left(1-\frac{2}{\sqrt{\kappa}+1}\right)^k \\ &= 2\left(1-\frac{2/\sqrt{\kappa}}{1+1/\sqrt{\kappa}}\right)^k \end{aligned}$$

Conjugate Gradient Method

And for large κ it follows that

$$\frac{\|e_k\|_A}{\|e_0\|_A} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k \approx 2\left(1-\frac{2}{\sqrt{\kappa}}\right)^k$$

Hence we terminate CG when

$$\left(1-\frac{2}{\sqrt{\kappa}}\right)^k \approx \frac{\epsilon}{2}$$

Conjugate Gradient Method

Taking logs gives

$$k \approx \log(\epsilon/2)/\log\left(1 - \frac{2}{\sqrt{\kappa}}\right) \approx \frac{1}{2}|\log(\epsilon/2)|\sqrt{\kappa}$$

where the last expression follows from the Taylor expansion:

$$\log\left(1 - \frac{2}{\sqrt{\kappa}}\right) \approx \log(1) - \frac{2}{\sqrt{\kappa}} = -\frac{2}{\sqrt{\kappa}}$$

This analysis shows that the number of CG iterations for a given tolerance ϵ grows approximately as $\sqrt{\kappa}$

Conjugate Gradient Method

Matlab demo: Convergence of CG for the discrete Laplacian for different choices of h

Conjugate Gradient Method

For the discrete Laplacian, we have $\kappa = O(h^{-2})$,[6] hence number of CG iterations should grow as $O(\sqrt{\kappa}) = O(h^{-1})$

For $\epsilon = 10^{-4}$, we obtain the following convergence results

h	κ	CG iterations
4×10^{-2}	3.67×10^{2}	32
2×10^{-2}	1.47×10^{3}	65
1×10^{-2}	5.89×10^{3}	133
5×10^{-3}	2.36×10^{4}	272

[6] This is a standard result in finite element analysis

Conjugate Gradient Method

These results indicate that CG gets more expensive for Poisson equation as h is reduced for two reasons:

- The matrix and vectors get larger, hence each CG iteration is more expensive
- We require more iterations since the condition number gets worse

Conjugate Gradient Method: Preconditioning

The final crucial idea that we will mention is preconditiong

The idea is that we premultiply $Ax = b$ by the preconditioning matrix M to obtain the system $MAx = Mb$

The CG convergence rate will then depend on the properties of MA rather than A

e.g. one intuitive idea is to choose $M \approx A^{-1}$ so that $MA \approx \mathrm{I}$ has a smaller condition number than A

Conjugate Gradient Method: Preconditioning

Good preconditioners must be cheap to compute, and should significantly accelerate convergence of an iterative method

Preconditioners can have a dramatic effect on convergence!

For example, preconditioning can ensure that number of CG iterations required for Poisson equation is independent of h

Preconditioning for Krylov subspace methods is a major topic in Scientific Computing: It is essential for large-scale problems!