AMSC 600 /CMSC 760 Advanced Linear Numerical Analysis

Fall 2007

Preconditioning

Dianne P. O'Leary

Preconditioners

- How preconditioners are used (Sec 9.2, 9.3)
- Preconditioners from SIMs (Sec 10.2)
- Incomplete LU (ILU) (Sec 10.3, 10.4, 10.6)
- Approximate inverse preconditioners (Sec 10.5)
- Block preconditioners (Sec 10.7)
- Polynomial preconditioners (Sec 12.3)
- Multigrid preconditioners (Chapter 13 - discussed later)
- Variants for normal equations (Sec 10.8) (We won't discuss this)
- Handling inexactness in $\mathbf{A}$ or $\mathbf{M}$ (Sec 9.4)

How preconditioners are used (Sec 9.2, 9.3)

- We want to solve $\mathbf{Ax} = \mathbf{b}$.
- The cost of applying methods like cg, GMRES, and BiCGStab depends on how many iterations are needed.
- The number of iterations is determined in part by behavior of polynomials evaluated at the matrix $\mathbf{A}$.
- The behavior of the polynomials depends on the eigenvalues of $\mathbf{A}$.
- If we are not satisfied with the eigenvalues, we can apply the iterative method to a related problem instead of the original, leading to the preconditioned iterations.

Case 1: A symmetric

Recall that for CG, we can derive the preconditioned iteration by applying cg to the problem

$$\mathbf{M}^{-1/2}\mathbf{A}\mathbf{M}^{-1/2}\mathbf{y} = \mathbf{M}^{-1/2}\mathbf{b},$$

where $\mathbf{A}$ and $\mathbf{M}$ are symmetric and positive definite and $\mathbf{x} = \mathbf{M}^{-1/2}\mathbf{y}$.

- It is important to do a two-sided transformation so that the iteration matrix $\mathbf{M}^{-1/2}\mathbf{A}\mathbf{M}^{-1/2}$ is symmetric and positive definite if $\mathbf{A}$ is.
- The actual computational formulas require the solution to linear systems involving the matrix $\mathbf{M}$.
- The matrix $\mathbf{M}^{-1/2}$ is not used.

- Preconditioning makes convergence depend on the eigenvalues of $\mathbf{M}^{-1}\mathbf{A}$, so we choose $\mathbf{M}$ to make these eigenvalues better behaved than those of $\mathbf{A}$:
  - Perhaps they are more clustered.
  - Perhaps the condition number of the matrix is smaller.
- If $\mathbf{A}$ is symmetric but indefinite, the same trick allows us to derive a preconditioned symmetric Lanzcos tridiagonalization.

Case 2: A a general matrix

In this case, there is no symmetry to preserve, so we have three preconditioning options, obtained by applying your favorite algorithm (GMRES, BiCGStab, etc.) to one of the following problems:

- Left preconditioning: $\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}$.
- Right preconditioning: $\mathbf{A}\mathbf{M}^{-1}\mathbf{y} = \mathbf{b}$, with $\mathbf{x} = \mathbf{M}^{-1}\mathbf{y}$.
- Two-sided preconditioning: $\mathbf{M_1}^{-1}\mathbf{A}\mathbf{M_2}^{-1}\mathbf{y} = \mathbf{M_1}^{-1}\mathbf{b}$, with $\mathbf{x} = \mathbf{M_2}^{-1}\mathbf{y}$.

Again the goal is that the eigenvalues of the preconditioned matrix $\mathbf{M}^{-1}\mathbf{A}$ or $\mathbf{A}\mathbf{M}^{-1}$ or $\mathbf{M}_1^{-1}\mathbf{A}\mathbf{M}_2^{-1}$ are better behaved than those of $\mathbf{A}$.

Example: Suppose we know that $\mathbf{A} \approx \mathbf{L}\mathbf{U}$.

- Left preconditioning: $\mathbf{B}_1 = \mathbf{U}^{-1}\mathbf{L}^{-1}\mathbf{A}$.
- Right preconditioning: $\mathbf{B}_2 = \mathbf{A}\mathbf{U}^{-1}\mathbf{L}^{-1}$.
- Two-sided preconditioning: $\mathbf{B}_3 = \mathbf{L}^{-1}\mathbf{A}\mathbf{U}^{-1}$.

Note that the eigenvalues of the three matrices $\mathbf{B}_1$, $\mathbf{B}_2$, and $\mathbf{B}_3$ are the same.

So in some sense there is no difference, except that the residual norm for $\mathbf{B}_2$ will be the same $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|$, but this is not true for the other formulations, so the termination criterion for the other two iterations will be different.

Preconditioners from SIMs (Sec 10.2)

Recall that we split $\mathbf{A} = \mathbf{L}_S + \mathbf{D}_S + \mathbf{U}_S$, its strictly lower triangular, diagonal, and strictly upper triangular pieces.

- Jacobi: $\mathbf{M}_J = \mathbf{D}_S$.
- Gauss-Seidel: $\mathbf{M}_{GS} = \mathbf{D}_S + \mathbf{L}_S$.
- Symmetric Gauss-Seidel: $\mathbf{M}_{SGS} = (\mathbf{D}_S + \mathbf{L}_S)\mathbf{D}_S^{-1}(\mathbf{D}_S + \mathbf{U}_S)$. (This is what we would get from following a (forward) Gauss-Seidel step by a backward one, ordering the equations last to first.)
- Symmetric SOR: $\mathbf{M}_{SSOR} = (\mathbf{D}_S + \omega\mathbf{L}_S)\mathbf{D}_S^{-1}(\mathbf{D}_S + \omega\mathbf{U}_S)$

- The effectiveness of these preconditioners is independent of whether the stand-alone SIM is convergent or not!
- If $\mathbf{A}$ is symmetric, then so is $\mathbf{M}_J$, $\mathbf{M}_{SGS}$, and $\mathbf{M}_{SSOR}$, but $\mathbf{M}_{GS}$ and $\mathbf{M}_{SOR}$ are nonsymmetric.

Incomplete LU (ILU) (Sec 10.3, 10.4, 10.6)

The SGS preconditioner $\mathbf{M}_{SGS} = (\mathbf{D}_S + \mathbf{L}_S)\mathbf{D}_S^{-1}(\mathbf{D}_S + \mathbf{U}_S)$ has a very interesting form.

- It is the product of a lower triangular matrix, a diagonal, and an upper triangular matrix.
- The sparsity pattern of each factor matches that of $\mathbf{A}$.

If we factored $\mathbf{A} = \mathbf{LU}$ (LU) or $\mathbf{A} = \mathbf{LL}^T$ (Cholesky, for symmetric positive definite matrices), then we may lose sparsity.

Question: What are the optimal choices of entries in $\mathbf{L}$ and $\mathbf{U}$ if we constrain them to have a particular sparsity structure?

Another question: What does optimal mean?

- Ideally, we want it to be optimal in the sense of minimizing the time needed to solve the linear system using this preconditioner. This seems to be an intractable problem.

- Since the number of iterations is related to the location of the eigenvalues of the preconditioned matrix, we might seek to minimize the spread of the eigenvalues. This also seems intractable.
- We can get a bound on the spread by looking at $\|\mathbf{I} - \mathbf{M}^{-1}\mathbf{A}\|_F$. This is also too difficult. Hold this thought; we will use it again in approximate inverse preconditioners.
- Instead, we try to keep $\|\mathbf{A} - \mathbf{M}\|_F$ small, which also implies that the spread of the eigenvalues is small.

To see how the ILU factorization works, we review the LU factorization.

The LU decomposition (IKJ)

In the form written below, we completely update one row at a time, computing the multipliers (entries of $\mathbf{L}$) and the elements of $\mathbf{U}$. For convenience of notation, we store all of these entries in $\mathbf{A}$.

for $i = 2 : n$,
  for $k = 1 : i - 1$,
    Compute the multiplier $a_{ik} = a_{ik}/a_{kk}$.
    for $j = k + 1 : n$,
      $a_{ij} = a_{ij} - a_{ik}a_{kj}$
    end
  end
end

CAUTION

Caution: As written, in the previous algorithm, we are performing LU without pivoting for stability. This is dangerous and the algorithm will fail if

- it encounters a zero pivot a_{kk}.
- it encounters a pivot of small magnitude.

All of the LU algorithms we discuss inherit this breakdown property.

Therefore, unless $\mathbf{A}$ has some property that ensures stability (e.g., symmetric positive definite, M-matrix, diagonally dominant), all practical algorithms must allow row-interchanges or some other fix if they encounter a small pivot.

The ILU decomposition

For the ILU decomposition, we decide which elements of $\mathbf{L}$ and $\mathbf{U}$ should be kept and which should be discarded.

Let $\mathcal{S}$ be the set of indices of elements to be kept.

for $i = 2 : n$,
  for $k = 1 : i-1$, with $i, k \in \mathcal{S}$
    Compute the multiplier $a_{ik} = a_{ik}/a_{kk}$.
    for $j = k+1 : n$, with $i, j \in \mathcal{S}$
      $a_{ij} = a_{ij} - a_{ik}a_{kj}$
    end
  end
  Zero any entry in row i that is not to be kept.
end

Subtle point:

- The ILU factors $\mathbf{L}$ and $\mathbf{U}$ can be very different from the exact LU factors. In fact, virtually every element can be different!
- But the algorithm is self correcting, in the sense that

$$\mathbf{R} \equiv \mathbf{LU} - \mathbf{A}$$

  is nonzero only in the positions of elements not kept.

If we keep more, is $\mathbf{R}$ "smaller"?

Sadly, the answer is not always ...

... unless $\mathbf{A}$ is an M-matrix.

Recall (from the SIM handout) that this means that

- the main diagonal elements are positive.
- the off diagonal elements are nonpositive.
- the elements of $\mathbf{A}^{-1}$ are nonnegative.

If $\mathbf{A}$ is an M-matrix then $\rho((\mathbf{LU})^{-1}\mathbf{R})$ is monotonically decreasing as the set $\mathcal{S}$ grows.

This is a special case of a theorem on monotonicity of regular splittings; see Varga *Matrix Iterative Analysis* Theorem 3.15.

Determining $\mathcal{S}$

ILU(0)

ILU(0): The easiest choice is to take $\mathcal{S} = \{(i,j) : a_{ij} \neq 0\}$. Then $\mathbf{L}+\mathbf{U}$ has the same sparsity structure as $\mathbf{A}$ and can be stored used at the same cost.

ILU(p)

ILU(p): This was originally defined for 5-point operator matrices in a very natural way, but the extension to more general sparse matrices is a little harder to swallow.

Recall the update statement

$$a_{ij} \leftarrow a_{ij} - a_{ik}a_{kj}.$$

Suppose that each (nonzero) element of $\mathbf{A}$ is equal to ϵ raised to a power that depends on the row and column index. Then

$$|a_{ij}| \leftarrow |\epsilon^{\hat{p}(i,j)} - \epsilon^{\hat{p}(i,k)+\hat{p}(k,j)}| \approx \epsilon^{min(\hat{p}(i,j),\hat{p}(i,k)+\hat{p}(k,j))}.$$

We initially set

- $\hat{p} = 1$ for all main diagonal elements and all nonzero elements,
- $\hat{p} = \infty$ for all off-diagonal zero elements.

Then if we want to keep "large" elements of the factorization, we keep all elements with $\hat{p} \leq p+1$.

Try an example.

More correctly: Look at nonzero element (i,j) of $\mathbf{U}$ (or $\mathbf{L}$). We define its level this way:

- It may be nonzero because $a_{ij} \neq 0$. Such an element has level $\hat{p} = 1$.
- It may be nonzero because $a_{ij} = 0$ but a_{ik} and a_{kj} were both nonzero. Such an element has level 2.
- It may be nonzero because $a_{ij}, a_{ik} = 0$ but $a_{kj} \neq 0$ and u_{ik} fills in. Such an element has level one more than that for (i,k).
- In general, as an element (i,j) is updated, its level is updated to be the minimum of its old value and the sum of those for (i,k) and (k,j).

For some matrices, this algorithm is quite easy to implement. See, for example, Saad Section 10.3.4.

Disadvantages of ILU(p) for general matrices:

- Required storage cannot be predicted in advance.
- Updating the levels is expensive.
- We may throw away some big elements!
- Because $\mathbf{R}$ is nonzero only in positions where elements are dropped, the row sums of $\mathbf{LU}$ are probably different from those of $\mathbf{A}$, and this can be a difficulty in some applications (Markov chains, pde schemes that conserve energy, etc.)

MILU

This modified ILU algorithm fixes the row-sum difficulty by taking the sum of the dropped entries in row i of $\mathbf{U}$ and subtracting them from u_{ii}.

Recall that before this modification

$$\mathbf{R} \equiv \mathbf{LU} - \mathbf{A}.$$

Let $\mathbf{r}^T$, $\mathbf{a}^T$, and $\boldsymbol{\ell}^T$ be the ith rows of the matrices $\mathbf{R}$, $\mathbf{A}$, and $\mathbf{L}$. Then

$$\mathbf{r}^T = \boldsymbol{\ell}^T\mathbf{U} - \mathbf{a}^T,$$

so

$$\mathbf{r}^T\mathbf{e} = \boldsymbol{\ell}^T\mathbf{U}\mathbf{e} - \mathbf{a}^T\mathbf{e}.$$

Therefore, if we change u_{ii} to $u_{ii} - \mathbf{r}^T\mathbf{e}$, then $\mathbf{a}^T\mathbf{e}$ will equal $\boldsymbol{\ell}^T\mathbf{U}\mathbf{e}$ (since $\ell_{ii} = 1$).

MILU(p) is the same as ILU(p) except for this correction to the diagonal elements.

ILUT: Threshhold ILU

Rather than just specifying a specific drop level p, we might specify a drop tolerance τ to help us make the decision.

for $i = 2 : n$,
  for $k = 1 : i-1$,
    Compute the multiplier $a_{ik} = a_{ik}/a_{kk}$.
    for $j = k+1 : n$,
      $a_{ij} = a_{ij} - a_{ik}a_{kj}$

```
    end
  end
  Drop any elements that are very small or have high level.
end
```

- "Small" can mean absolute value small relative to the norm of the original row of the matrix.
- We can also specify a max number of elements allowed to be saved in any row, saving the largest magnitude elements.

Other variants

There are a multitude of other variants that I won't discuss, including

- ILUS
- ILUC
- Cholesky-infinity

All of the algorithms we discussed benefit from a good ordering of rows and columns determined by algorithms such as

- reverse Cuthill-McKee
- approx. minimum degree

that we discussed previously.

Such reorderings generally reduce the fill-in, so, for a fixed amount of storage space, they allow us to keep $\mathbf{R}$ smaller.

Convergence of iterative methods with ILU preconditioning

Recall

$$\mathbf{R} \equiv \mathbf{LU} - \mathbf{A}$$

and our preconditioned matrix is

$$\mathbf{L}^{-1}\mathbf{A}\mathbf{U}^{-1} = \mathbf{I} - \mathbf{L}^{-1}\mathbf{R}\mathbf{U}^{-1}.$$

So to have the eigenvalues of the preconditioned matrix behave better than those of $\mathbf{A}$, we want

- $\|\mathbf{R}\|$ small.
- $\mathbf{L}$ and $\mathbf{U}$ well conditioned – well, at least no worse than $\mathbf{A}$.

To guarantee these properties, we need more assumptions on $\mathbf{A}$, like symmetric positive definite or M-matrix.

Without such properties, alternative preconditioners such as the next one often perform better.

Approximate inverse preconditioners (Sec 10.5)

Back to fundamentals. We want to determine a matrix $\mathbf{M}$ that is close to $\mathbf{A}$. One way to measure this is through one of the functions

- Left preconditioning: $F(\mathbf{M}^{-1}) = \|\mathbf{I} - \mathbf{M}^{-1}\mathbf{A}\|_F^2$.
- Right preconditioning: $F(\mathbf{M}^{-1}) = \|\mathbf{I} - \mathbf{A}\mathbf{M}^{-1}\|_F^2$.
- Two-sided preconditioning: $F(\mathbf{M}_1^{-1}, \mathbf{M}_2^{-1}) = \|\mathbf{I} - \mathbf{M_1}^{-1}\mathbf{A}\mathbf{M_2}^{-1}\|_F^2$.

As an example of how approximate inverse preconditioners (AIP) are developed, consider the right preconditioner.

Right AIP

We want a matrix $\mathbf{W}$ to make

$$F(\mathbf{W}) = \|\mathbf{I} - \mathbf{A}\mathbf{W}\|_F^2$$

small.

Note that we don't want to make it zero – because the inverse of $\mathbf{A}$ is probably dense, and we can't afford to use it.

Our problem breaks into n independent problems, one for each column of $\mathbf{W}$:

$$\min_{\mathbf{w}_j} \|\mathbf{e}_j - \mathbf{A}\mathbf{w}_j\|_2^2$$

where $\mathbf{w}_j$ is constrained to be sparse.

We could, for example,

- Constrain $\mathbf{w}_j$ to have the same sparsity pattern as $\mathbf{A}$. Then each of the minimization problems involves only a small number of variables and they can be solved quickly.

- Use a few steps of GMRES to solve the problem. We need to store the basis vectors $\mathbf{V}_k$ as a sparse matrix rather than a dense one and stop before we get too much fill-in.

Saad Section 10.5.2 and 10.5.3 discusses other options.

One note: This is a situation in which we have many linear systems to (approximately) solve, all involving the same matrix $\mathbf{A}$, so we it is also natural to use the block version of GMRES to solve them, rather than treating the right hand sides $\mathbf{e}_j$ one at a time.

Block preconditioners (Sec 10.7)

If $\mathbf{A}$ has a natural block structure, then it is natural to look for a preconditioner with the same block structure.

Example 1: Saad p.337 works through an example for a block tridiagonal matrix.

Example 2: A saddle point problem. Let

$$\mathbf{A} = \begin{bmatrix} \mathbf{F} & \mathbf{B} \\ \mathbf{C} & \mathbf{0} \end{bmatrix},$$

and notice that

$$\mathbf{A} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{C}\mathbf{F}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{F} & \mathbf{0} \\ \mathbf{0} & -\mathbf{C}\mathbf{F}^{-1}\mathbf{B} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{F}^{-1}\mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}.$$

Therefore, if we can

- solve linear systems involving $\mathbf{F}$ or a matrix close to it and
- solve linear systems involving $\mathbf{C}\mathbf{F}^{-1}\mathbf{B}$ or a matrix close to it,

we have a good preconditioner.

Polynomial preconditioners (Sec 12.3)

If we choose $\mathbf{M}^{-1} = p_k(\mathbf{A})$ for some polynomial of degree k, then the cost of applying the preconditioner is just k times the cost of a matrix-vector product involving the matrix $\mathbf{A}$.

Question: Can we find a good polynomial?

Answer: Sometimes.

Example 1: Neumann Polynomial.

A useful identity:

$$\begin{aligned}(\mathbf{I}-\mathbf{N})(\mathbf{I}+\mathbf{N}+\ldots+\mathbf{N}^k) &= (\mathbf{I}+\mathbf{N}+\ldots+\mathbf{N}^k)-(\mathbf{N}+\mathbf{N}^2-\ldots-\mathbf{N}^{k+1}) \\ &= \mathbf{I}-\mathbf{N}^{k+1}.\end{aligned}$$

Suppose $\rho(\mathbf{N})$ is small. Then $\mathbf{N}^{k+1}$ is quite small, so

$$(\mathbf{I}-\mathbf{N})(\mathbf{I}+\mathbf{N}+\ldots+\mathbf{N}^k) \approx \mathbf{I},$$

so

$$(\mathbf{I}-\mathbf{N})^{-1} \approx (\mathbf{I}+\mathbf{N}+\ldots+\mathbf{N}^k).$$

How we can use this for preconditioning:

Let

$$\omega \mathbf{D}^{-1}\mathbf{A} = \mathbf{I}-\mathbf{N},$$

where the diagonal matrix $\mathbf{D}$ and the scalar ω are chosen to make $\mathbf{N}$ small.

(We can't always accomplish this, but suppose that we can for some particular $\mathbf{A}$.)

Then we have the preconditioner

$$\mathbf{M}^{-1} = (\mathbf{I}+\mathbf{N}+\ldots+\mathbf{N}^k).$$

Example 2: Chebyshev Polynomial.

Suppose we know that most of the eigenvalues of the symmetric positive definite matrix $\mathbf{A}$ lie in the interval $[a,b]$, where $a>0$. Then (our old friend) the Chebyshev polynomial scaled and shifted to this interval, is quite small at all of these eigenvalues.

- So a few iterations of pcg will make the error small for all of these eigencomponents.
- At the same time, the cg polynomial will "work on" components corresponding to eigenvalues outside this interval, so that their error is reduced, too.

The use of this idea dates back to the 1950s.

Handling inexactness in $\mathbf{A}$ or $\mathbf{M}$ (Sec 9.4)

- In some problems it is impossible to form a matrix-vector product exactly.
  Example: We can approximately form the product of a Jacobian matrix with a vector by using a finite difference approximation.
- In other problems it is convenient to use a preconditioning matrix that cannot be applied exactly.
  Example: Perhaps we form $\mathbf{z} = \mathbf{M}^{-1}\mathbf{r}$ by using a few steps of GMRES applied to the linear system $\mathbf{M}\mathbf{z} = \mathbf{r}$. (This is called an inner iteration.)

The resulting perturbations to $\mathbf{M}^{-1}\mathbf{A}$ can cause algorithms like pcg and GMRES to fail.

- For pcg, the orthogonality relations depend on a lot of (uncomputed) inner products being zero, and instead these values will start to grow.
- The residual in both algorithms is updated recursively, and error in it will also start to grow.

When the matrix-vector product is inexact

Recent work by Simoncini and Szyld talks about how accurate the inner iteration must be in order to guarantee convergence.

- Surprisingly, for Arnoldi methods, they prove that the matrix-vector products can be allowed to be less accurate as the iteration progresses without hurting the convergence!
- They produce computable bounds on how big we can allow the error to be by making the clever observation that the relation

  $$[(\mathbf{A}+\mathbf{E}_1)\mathbf{v}_1, (\mathbf{A}+\mathbf{E}_2)\mathbf{v}_2, \ldots, (\mathbf{A}+\mathbf{E}_k)\mathbf{v}_k] = \mathbf{V}_{k+1}\bar{\mathbf{H}}_k$$

  implies

  $$(\mathbf{A}+\mathbf{E})\mathbf{V}_k = \mathbf{V}_{k+1}\bar{\mathbf{H}}_k,$$

  where

  $$\mathbf{E} = \sum_{i=1}^{k} \mathbf{E}_i \mathbf{v}_i \mathbf{v}_i^T.$$

- Using this Arnoldi relation, they can measure how much error is introduced by minimizing the "wrong" function.

Reference: Valeria Simoncini and Daniel Szyld, Siam J. Sci. Computing 25 2003 454-477.

When the preconditioner is inexact: FGMRES

Flexible GMRES (FGMRES) is one way to make the iteration more robust in the presence of perturbations in applying $\mathbf{M}^{-1}$.

GMRES with right preconditioning:

$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$, $\beta = \|\mathbf{r}_0\|$, $\mathbf{v}_1 = \mathbf{r}/\beta$.
for $j = 1 : m$,
  $\mathbf{w} = \mathbf{A}\mathbf{M}^{-1}\mathbf{v}_j$ {Gram-Schmidt Process:}
  for $i = 1 : j$,
    $h_{ij} = \mathbf{w}^T\mathbf{v}_i$
    $\mathbf{w} = \mathbf{w} - h_{ij}\mathbf{v}_i$
  end
  $h_{j+1,j} = \|\mathbf{w}\|$
  $\mathbf{v}_{j+1} = \mathbf{w}/h_{j+1,j}$
end
Compute $\mathbf{y}_m$ to minimize $\|\beta\mathbf{e}_1 - \bar{\mathbf{H}}_m\mathbf{y}\|$.
Set $\mathbf{x}_m = \mathbf{x}_0 - \mathbf{M}^{-1}(\mathbf{V}_m\mathbf{y}_m)$.

The weakness: If $\mathbf{M}$ is applied only approximately, or if it actually changes from iteration to iteration, then the last statement applies a different operator than was used in the definition of each of the columns of $\mathbf{V}$.

This can cause severe numerical difficulties.

Instead we want a variant that remembers what the correct $\mathbf{M}$ is for each basis vector and preserves the relation

$$\mathbf{A}\mathbf{Z} = \mathbf{V}\mathbf{H}$$

where $\mathbf{Z} = [\mathbf{M}_1^{-1}\mathbf{v}_1, \mathbf{M}_2^{-1}\mathbf{v}_2, ..., \mathbf{M}_n^{-1}\mathbf{V}_n]$.

FGMRES:

$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$, $\beta = \|\mathbf{r}_0\|$, $\mathbf{v}_1 = \mathbf{r}/\beta$.
for $j = 1 : m$,
  $\mathbf{z}_j = \mathbf{M}_j^{-1}\mathbf{v}_j$ and $\mathbf{w} = \mathbf{A}\mathbf{z}_j$ {Gram-Schmidt Process:}
  for $i = 1 : j$,
    $h_{ij} = \mathbf{w}^T\mathbf{v}_i$
    $\mathbf{w} = \mathbf{w} - h_{ij}\mathbf{v}_i$
  end
  $h_{j+1,j} = \|\mathbf{w}\|$
  $\mathbf{v}_{j+1} = \mathbf{w}/h_{j+1,j}$
end
Compute $\mathbf{y}_m$ to minimize $\|\beta\mathbf{e}_1 - \bar{\mathbf{H}}_m\mathbf{y}\|$.
Set $\mathbf{x}_m = \mathbf{x}_0 - \mathbf{Z}_m\mathbf{y}_m$.

Properties of FGMRES

- We need to save m additional vectors, the columns of $\mathbf{Z}_m$.
- If $\mathbf{M}_j = \mathbf{M}$ for all j (i.e., the preconditioner is constant), then FGMRES produces the same iterates as GMRES.
- Otherwise, the iterates are different, and we don't even have a Krylov subspace.
  - FMGRES still minimizes the residual over the span of the columns of the matrix $\mathbf{Z}_m$.
  - We lose the "optimal polynomial" convergence results.
- In GMRES, if $h_{j+1,j} = 0$ then the algorithm breaks down, but this is good because $\mathbf{x}_j$ will solve $\mathbf{Ax} = \mathbf{b}$.
  In FGMRES, this is true so long as $\mathbf{H}_j$ is nonsingular, which happens when the columns of $\mathbf{Z}_j$ are linearly independent.

Final words

- Preconditioning is an art rather than a science.
- It involves exploiting structure in your matrix:
  - sparsity
  - positive definiteness
  - M-matrix
  - related operators
  - etc.
- The major open question: is there an "automatic" preconditioner that can be applied that always gives good results?
- Algebraic multigrid has potential, but so far results are disappointing.
- We'll talk about multigrid methods next.