Krylov and Preconditioned GMRES

1. Spaces Spanned by Matrix-Vector Products
   - Krylov subspace methods
   - the power method
2. The Generalized Minimum Residual Method
   - an iterative least squares solver
   - a Julia function
3. preconditioned GMRES
   - Jacobi and Gauss-Seidel preconditioners
   - an experiment with Julia

MCS 471 Lecture 20
Numerical Analysis
Jan Verschelde, 7 October 2022

Krylov and Preconditioned GMRES

1. Spaces Spanned by Matrix-Vector Products
   - Krylov subspace methods
   - the power method
2. The Generalized Minimum Residual Method
   - an iterative least squares solver
   - a Julia function
3. preconditioned GMRES
   - Jacobi and Gauss-Seidel preconditioners
   - an experiment with Julia

spaces spanned by matrix-vector products

Input: a right hand side vector $\mathbf{b} \in \mathbb{R}^n$;
some algorithm to evaluate $A\mathbf{x}$.
Output: an approximation for $A^{-1}\mathbf{b}$.

Example ($n = 4$), consider $\mathbf{y}_1 = \mathbf{b}$, $\mathbf{y}_2 = A\mathbf{y}_1$, $\mathbf{y}_3 = A\mathbf{y}_2$, $\mathbf{y}_4 = Ay_3$, stored in the columns of $K = [\mathbf{y}_1\ \mathbf{y}_2\ \mathbf{y}_3\ \mathbf{y}_4]$.

$$
\begin{aligned}
AK &= [A\mathbf{y}_1\ A\mathbf{y}_2\ A\mathbf{y}_3\ A\mathbf{y}_4] = [\mathbf{y}_2\ \mathbf{y}_3\ \mathbf{y}_4\ A^4\mathbf{y}_1] \\
&= K \begin{bmatrix} 0 & 0 & 0 & -c_1 \\ 1 & 0 & 0 & -c_2 \\ 0 & 1 & 0 & -c_3 \\ 0 & 0 & 1 & -c_4 \end{bmatrix}, \quad \mathbf{c} = -K^{-1}A^4\mathbf{y}_1.
\end{aligned}
$$

$\Rightarrow AK = KC$, where C is a companion matrix.

Krylov subspaces

Set

$$K = [\mathbf{b} \;\; A\mathbf{b} \;\; A^2\mathbf{b} \;\; \cdots \;\; A^k\mathbf{b}].$$

Compute

$$K = QR.$$

Then the columns of Q span the *Krylov subspace*.

Look for an approximate solution of $A\mathbf{x} = \mathbf{b}$
in the span of the columns of Q.

We can interpret this as a modified Gram-Schmidt method,
stopped after k projections of $\mathbf{b}$.

Another connection is the *power method* to compute
the eigenvector with the largest eigenvalue,
as $A^k\mathbf{b}$ converges in the direction of the dominant eigenvector.

Krylov and Preconditioned GMRES

1. Spaces Spanned by Matrix-Vector Products
   - Krylov subspace methods
   - the power method
2. The Generalized Minimum Residual Method
   - an iterative least squares solver
   - a Julia function
3. preconditioned GMRES
   - Jacobi and Gauss-Seidel preconditioners
   - an experiment with Julia

the eigenvector with the largest eigenvalue

Consider an n-by-n matrix A with eigenvectors $\mathbf{v}_i$ and corresponding eigenvalues λ_i: $A\mathbf{v}_i = \lambda_i \mathbf{v}_i$, for $i = 1, 2, \ldots, n$.
Assume λ_1 dominates: $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$.

We can write $\mathbf{x}$ in the basis of eigenvectors:

$$
\begin{aligned}
\mathbf{x} &= c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n \\
A\mathbf{x} &= c_1 A\mathbf{v}_1 + c_2 A\mathbf{v}_2 + \cdots + c_n A\mathbf{v}_n \\
&= c_1\lambda_1\mathbf{v}_1 + c_2\lambda_2\mathbf{v}_2 + \cdots + c_n\lambda_n\mathbf{v}_n \\
A^k\mathbf{x} &= c_1\lambda_1^k\mathbf{v}_1 + c_2\lambda_2^k\mathbf{v}_2 + \cdots + c_n\lambda_n^k\mathbf{v}_n \\
&= |\lambda_1^k| \left(c_1 \frac{\lambda_1^k}{|\lambda_1^k|}\mathbf{v}_1 + c_2 \frac{\lambda_2^k}{|\lambda_1^k|}\mathbf{v}_2 + \cdots + c_n \frac{\lambda_n^k}{|\lambda_1^k|}\mathbf{v}_n \right)
\end{aligned}
$$

Because $|\lambda_1| > |\lambda_i|$: $\frac{\lambda_i^k}{|\lambda_i|} \rightarrow 0$ as $k \rightarrow \infty$.

convergence to the dominant eigenvector

Consider an n-by-n matrix A with eigenvectors $\mathbf{v}_i$ and corresponding eigenvalues λ_i: $A\mathbf{v}_i = \lambda_i \mathbf{v}_i$, for $i = 1, 2, \ldots, n$.

Assume λ_1 dominates: $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$.

Set $\mathbf{y} = A^k \mathbf{x}$, $\dfrac{\mathbf{y}}{\|\mathbf{y}\|} \rightarrow \mathbf{v}_1$, as $k \rightarrow \infty$.

Once we know $\mathbf{v}_1$, then we can compute $|\lambda_1|$:

$$
\begin{aligned}
A\mathbf{v}_1 = \lambda_1 \mathbf{v}_1 \quad &\Rightarrow \quad \|A\mathbf{v}_1\| = |\lambda_1| \|\mathbf{v}_1\| \\
&\Rightarrow \quad |\lambda_1| = \frac{\|A\mathbf{v}_1\|}{\|\mathbf{v}_1\|}
\end{aligned}
$$

Using $\| \cdot \|_2$, and for $\|\mathbf{v}_1\|_2 = 1$, $|\lambda_1| = \|A\|_2$, the *spectral radius of A*.

defining the power method in a Julia function

```
using Printf
Base.show(io::IO, f::Float64) = @printf(io, "%.3e", f)

using LinearAlgebra

"""
    power(A::Array{Float64,2},x0::Array{Float64,1},
          tol::Float64=1.0e-4,maxit::Int=10,
          verbose::Bool=true)

Runs the power method on A, starting at the vector x0.

Does no more than maxit steps.

Stops when the error is less than the tolerance.

By default, the verbose flag is true.
"""
```

the function `power`

```
function power(A::Array{Float64,2},x0::Array{Float64,1},
               tol::Float64=1.0e-4,maxit::Int=10,verbose::Bool=true)
    x = deepcopy(x0)
    (L, previousL, err) = (0.0, 0.0, 0.0)
    for i=1:maxit
        if verbose
            show(stdout, "text/plain", transpose(x)); println("")
        end
        x = A*x
        previousL = L
        L = norm(x)
        x = x/L
        err = abs(previousL - L)
        if verbose
            strL = @sprintf("%.16e", L)
            strE = @sprintf("%.2e", err)
            println("|lambda1| = $strL, error = $strE")
        end
        if err < tol
            return (x, L, err, false)
        end
    end
    return (x, L, err, true)
end
```

running on a random 4-by-4 matrix

```
$ julia eigvalpower.jl
1×4 Transpose{Float64,Array{Float64,1}}:
 9.963e-01  6.504e-01  1.469e-01  7.738e-01
|lambda1| = 2.6391467617899127e+00, error = 2.64e+00
1×4 Transpose{Float64,Array{Float64,1}}:
 2.804e-01  6.176e-01  6.502e-01  3.423e-01
|lambda1| = 1.8410993853309483e+00, error = 7.98e-01
1×4 Transpose{Float64,Array{Float64,1}}:
 4.474e-01  6.353e-01  4.679e-01  4.210e-01
|lambda1| = 1.9400644869532246e+00, error = 9.90e-02
1×4 Transpose{Float64,Array{Float64,1}}:
 3.849e-01  6.274e-01  5.413e-01  4.065e-01
|lambda1| = 1.9203144420218707e+00, error = 1.98e-02
1×4 Transpose{Float64,Array{Float64,1}}:
 4.077e-01  6.336e-01  5.132e-01  4.111e-01
|lambda1| = 1.9283996912144405e+00, error = 8.09e-03
1×4 Transpose{Float64,Array{Float64,1}}:
 3.988e-01  6.310e-01  5.240e-01  4.102e-01
|lambda1| = 1.9258253803775072e+00, error = 2.57e-03
```

output continued ...

```
1×4 Transpose{Float64,Array{Float64,1}}:
 4.021e-01  6.321e-01  5.198e-01  4.105e-01
|lambda1| = 1.9267900238791311e+00, error = 9.65e-04
1×4 Transpose{Float64,Array{Float64,1}}:
 4.008e-01  6.317e-01  5.215e-01  4.104e-01
|lambda1| = 1.9264351147228276e+00, error = 3.55e-04
1×4 Transpose{Float64,Array{Float64,1}}:
 4.013e-01  6.319e-01  5.208e-01  4.104e-01
|lambda1| = 1.9265707251169688e+00, error = 1.36e-04
1×4 Transpose{Float64,Array{Float64,1}}:
 4.011e-01  6.318e-01  5.211e-01  4.104e-01
|lambda1| = 1.9265185316155506e+00, error = 5.22e-05
The dominant eigenvector :
1×4 Transpose{Float64,Array{Float64,1}}:
 4.012e-01  6.318e-01  5.210e-01  4.104e-01
The spectral radius : 1.9265185316155506e+00,
with error 5.22e-05.  Reached the tolerance.
$
```

how fast is the convergence?

The speed at which the power method converges
is determined by the ratio $|\lambda_2/\lambda_1|$,
where λ_2 is the second largest eigenvalue, w.r.t $|\cdot|$.

Let us verify this with a matrix with prescribed eigenvalues.

$$D = \begin{bmatrix} 123456 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad Q^T Q = I, \quad A = Q D Q^T.$$

Observe $AQ = QD$.

- The eigenvalues of A are the diagonal elements of D.
- The eigenvectors of A are the columns of Q.

an interactive session in Julia

```
julia> using LinearAlgebra

julia> D = [123456 0 0; 0 1 0; 0 0 1]
3×3 Array{Int64,2}:
 123456  0  0
      0  1  0
      0  0  1

julia> Q, R = qr(rand(3,3));

julia> A = Q*D*transpose(Q);

julia> eigvals(A)
3-element Array{Float64,1}:
      0.9999999999999316
      1.0000000000153024
 123456.00000000001

julia>
```

an exercise

Exercise 1:

Use the statements on the previous slide to make a random matrix with 8 rows and 8 columns and with the largest eigenvalue equal to 10^8, and all other eigenvalues equal to one.

Run the function `power` on this matrix, starting at a random vector, three times: first with the default tolerance `1.0e-4` and then with `1.0e-8` and `1.0e-12` as tolerances.

How many steps does it take for `power` to converge in each case?

Explain the convergence with the ratio of the eigenvalues.

Krylov and Preconditioned GMRES

1. Spaces Spanned by Matrix-Vector Products
   - Krylov subspace methods
   - the power method
2. The Generalized Minimum Residual Method
   - an iterative least squares solver
   - a Julia function
3. preconditioned GMRES
   - Jacobi and Gauss-Seidel preconditioners
   - an experiment with Julia

an iterative least squares solver

Starting with an initial guess $\mathbf{x}_0$ for the solution of $A\mathbf{x} = \mathbf{b}$,

look for improvements to $\mathbf{x}_0$ in the Krylov space $[\mathbf{r} \;\; A\mathbf{r} \;\; A^2\mathbf{r} \;\; \cdots \;\; A^k\mathbf{r}]$,

where $\mathbf{r}$ is the residual.

At step k:

- enlarge the Krylov space by adding $A^k\mathbf{r}$,
- reorthogonalize the basis,
- use least squares to find the best improvement to add to $\mathbf{x}_0$.

Generalized Minimum Residual Method (GMRES)

Input: A, a matrix;
$\mathbf{b}$, a right hand side vector,
$\mathbf{x}_0$, an initial guess,
N, maximum number of iterations.

$\mathbf{r} := \mathbf{b} - A\mathbf{x}_0$; $\mathbf{q}_1 := \mathbf{r}/\|\mathbf{r}\|_2$
for $k = 1, 2, \ldots, N$ do
  $\mathbf{y} := A\mathbf{q}_k$
  for $j = 1, 2, \ldots, k$ do
    $h_{j,k} := \mathbf{q}_j^T \mathbf{y}$; $\mathbf{y} := \mathbf{y} - h_{j,k}\mathbf{q}_j$
  $h_{k+1,k} := \|\mathbf{y}\|_2$
  if $h_{k+1,k} \neq 0$ then $\mathbf{q}_{k+1} := \mathbf{y}/h_{k+1,k}$
  minimize $\|H\mathbf{c}_k - [\|\mathbf{r}\|_2\ 0\ 0 \cdots 0]^T\|_2$ for $\mathbf{c}_k$
  $Q_k := [\mathbf{q}_1\ \mathbf{q}_2\ \cdots\ \mathbf{q}_k]$
  $\mathbf{x}_k := Q_k\mathbf{c}_k + \mathbf{x}_0$

the matrices H and Q_k

Starting with $\mathbf{r} = \mathbf{b} = A\mathbf{x}_0$, construct $[\mathbf{r}\ A\mathbf{r}\ A^2\mathbf{r} \cdots A^k\mathbf{r}]$, and apply the modified Gram-Schmidt orthogonalization.

This results in $AQ_k = Q_{k+1}H$, written in full as

$$A[\mathbf{q}_1\ \mathbf{q}_2 \cdots \mathbf{q}_k] = [\mathbf{q}_1\ \mathbf{q}_2 \cdots \mathbf{q}_k\ \mathbf{q}_{k+1}] \begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,k} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,k} \\ & h_{3,2} & \cdots & h_{3,k} \\ & & \ddots & \vdots \\ & & & h_{k+1,k} \end{bmatrix}.$$

Q_k is an n-by-k matrix, where $k \ll n$.

The columns of Q_{k+1} are an orthogonal basis for the Krylov space spanned by $[\mathbf{r}\ A\mathbf{r}\ A^2\mathbf{r} \cdots A^k\mathbf{r}]$.

augmenting $\mathbf{x}_0$

Vectors in the space spanned by Q_k are written as $Q_k\mathbf{c}$, where $\mathbf{c} = (c_1, c_2, \ldots, c_k)$ are the coordinates of the vectors:

$$Q_k\mathbf{c} = [\mathbf{q}_1 \ \mathbf{q}_2 \cdots \mathbf{q}_k]\mathbf{c} = c_1\mathbf{q}_1 + c_2\mathbf{q}_2 + \cdots + c_k\mathbf{q}_k.$$

We want to find $\Delta\mathbf{x}$ to minimize the residual:

$$\mathbf{b} - A(\mathbf{x}_0 + \Delta\mathbf{x}) = \mathbf{r} - A\Delta\mathbf{x}, \quad A\mathbf{x} = \mathbf{b}.$$

This means finding $\mathbf{c}$ that minimizes

$$\begin{aligned} \|A\Delta\mathbf{x} - \mathbf{r}\|_2 &= \|AQ_k\mathbf{c} - \mathbf{r}\|_2 \\ &= \|Q_{k+1}H\mathbf{c} - \mathbf{r}\|_2 \\ &= \|H\mathbf{c} - Q_{k+1}^T\mathbf{r}\|_2. \end{aligned}$$

In the last step, we applied $\|Q\|_2 = 1$ for any orthogonal Q and the norm of a product is the product of the norms.

computing $\Delta \mathbf{x}$

As $\mathbf{q}_1 = \mathbf{r}/\|\mathbf{r}\|_2$, we have $Q_{k+1}^T \mathbf{r} = [\|\mathbf{r}\|_2 \; 0 \; 0 \cdots 0]^T$.

Minimizing $\|H\mathbf{c} - Q_{k+1}^T \mathbf{r}\|_2$ translates into

$$\begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,k} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,k} \\ & h_{3,2} & \cdots & h_{3,k} \\ & & \ddots & \vdots \\ & & & h_{k+1,k} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_k \end{bmatrix} = \begin{bmatrix} \|\mathbf{r}\|_2 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

We solve for $\mathbf{c}$ in the least squares sense, and then update:

$$\mathbf{x}_k = \mathbf{x}_0 + \Delta \mathbf{x} = \mathbf{x}_0 + Q_k \mathbf{c}.$$

a least squares minimization step

The step

$$\text{minimize } \|H\mathbf{c}_k - [\|\mathbf{r}\|_2\ 0\ 0 \cdots 0]^T\|_2 \text{ for } \mathbf{c}_k$$

involves $k+1$ equations in k unknowns.

The minimize step is a least squares problem.

If $h_{k+1,k} = 0$, then step k is the final step and
the minimization will arrive at the exact solution of $A\mathbf{x} = \mathbf{b}$.

By this minimization, $\|\mathbf{b} - A\mathbf{x}\|_2$ decreases monotonically in each step.

Krylov and Preconditioned GMRES

1. Spaces Spanned by Matrix-Vector Products
   - Krylov subspace methods
   - the power method
2. The Generalized Minimum Residual Method
   - an iterative least squares solver
   - a Julia function
3. preconditioned GMRES
   - Jacobi and Gauss-Seidel preconditioners
   - an experiment with Julia

the program gmres.jl

```
"""
    gmres(A::Array{Float64,2}, b::Array{Float64,1},
          x::Array{Float64,1}, N::Int=8)

runs N steps of the Generalized Minimum Residual
Method, to solve the system with matrix A with
right hand side vector b, starting at x.

The number N of iterations should be not be larger
than the number of columns of A.
"""
function gmres(A::Array{Float64,2},
               b::Array{Float64,1},
               x0::Array{Float64,1}, N::Int,
               verbose::Bool=true)
```

the initialization of the loop

```
function gmres(A::Array{Float64,2}, b::Array{Float64,1},
               x0::Array{Float64,1}, N::Int,
               verbose::Bool=true)
    nrows = size(A,1)
    ncols = size(A,2)
    x = zeros(ncols)
    Q = zeros(nrows, ncols+1)
    H = zeros(nrows+1, ncols+1)
    r = b - A*x0[1:ncols]
    q = r/norm(r)
    Q[:,1] = q
    stop = false
    for k=1:N
        y = A*Q[1:ncols,k]
        for j=1:k
            H[j,k] = transpose(Q[:,j])*y
            y = y - H[j,k]*Q[:,j]
        end
```

the loop continues ...

```
        H[k+1,k] = norm(y)
        if H[k+1,k] == 0
            stop = true
        else
            Q[:,k+1] = y/H[k+1,k]
        end
        rhs = zeros(k+1)
        rhs[1] = norm(r)
        c = H[1:k+1,1:k]\rhs  # least squares solving
        dx = Q[:,1:k]*c
        x = x0 + dx
        if stop
            return x
        end
    end
    return x
end
```

run a numerical example

Exercise 2:

Consider the system $A\mathbf{x} = \mathbf{b}$ with $A = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}$, and $\mathbf{b} = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}$.

1. Do two steps with the GMRES method ($N = 2$), with Julia.
2. Compare your result with the output of $\mathbf{x} = A \backslash \mathbf{b}$.

   Compute the residual $\|(\mathbf{b} - A\mathbf{x})^T A\|_2$.

Krylov and Preconditioned GMRES

1. Spaces Spanned by Matrix-Vector Products
   - Krylov subspace methods
   - the power method
2. The Generalized Minimum Residual Method
   - an iterative least squares solver
   - a Julia function
3. preconditioned GMRES
   - Jacobi and Gauss-Seidel preconditioners
   - an experiment with Julia

preconditioned GMRES

The preconditioned GMRES starts with $\mathbf{r} := M^{-1}(\mathbf{b} - A\mathbf{x}_0)$.

The goal of multiplying with M^{-1} is to reduce the condition number.

- The ideal choice for M is A, but computing A^{-1} is too hard.
- The easiest choice for M is I, but then $\kappa(M^{-1}A) = \kappa(A)$.

Two common choices for M:

- The Jacobi preconditioner: $M = D$, D is the diagonal of A.
- The Gauss-Seidel preconditioner: $M = L + D$,
  L is lower diagonal part of A.

Krylov and Preconditioned GMRES

1. Spaces Spanned by Matrix-Vector Products
   - Krylov subspace methods
   - the power method
2. The Generalized Minimum Residual Method
   - an iterative least squares solver
   - a Julia function
3. preconditioned GMRES
   - Jacobi and Gauss-Seidel preconditioners
   - an experiment with Julia

an experiment with Julia

Consider a random 3-by-3 matrix, which is well conditioned.

```
julia> A = rand(3,3)
3x3 Array{Float64,2}:
 0.340899  0.614219   0.961586
 0.83794   0.0868184  0.0767624
 0.94175   0.0577572  0.563789

julia> using LinearAlgebra

julia> cond(A)
8.237642629908706

julia> A[3,:] = 1.0e-8*A[3,:];
```

The last statement multiplies every element on the third row with 10^{-8}.

an ill conditioned matrix

```
julia> A
3x3 Array{Float64,2}:
 0.340899    0.614219     0.961586
 0.83794     0.0868184    0.0767624
 9.4175e-9   5.77572e-10  5.63789e-9

julia> cond(A)
4.437203960744239 7e8
```

The condition number is 10^8.

Can we improve the numerical conditioning of A?

making a Jacobi preconditioner

```
julia> d = diag(A)
3-element Array{Float64,1}:
 0.340899
 0.0868184
 5.63789e-9

julia> M = d.*Matrix(I, 3, 3)
3x3 Array{Float64,2}:
 0.340899  0.0         0.0
 0.0       0.0868184   0.0
 0.0       0.0         5.63789e-9
```

The `.*` is the componentwise multiplication.

applying the Jacobi preconditioner

```
julia> B = inv(M)*A
3x3 Array{Float64,2}:
 1.0       1.80176   2.82073
 9.65164   1.0       0.884172
 1.6704    0.102445  1.0

julia> cond(B)
20.431912707183 42
```

We see that $M^{-1}A$ is well conditioned.

an exercise

Exercise 3:

Consider the making of an ill-conditioned matrix by multiplying the first column with 10^{-8}.

```
julia> A = rand(3,3)
3x3 Array{Float64,2}:
 0.906183  0.193067  0.904161
 0.768742  0.202677  0.398809
 0.558759  0.840976  0.77066

julia> A[:,1] = 1.0E-8*A[:,1];

julia> cond(A)
3.8254496346866 44e8
```

Does a Jacobi preconditioner improve the conditioning?

setup for Gauss-Seidel preconditioner

```
julia> A = rand(3,3)
3x3 Array{Float64,2}:
 0.525208  0.666893  0.577695
 0.67043   0.536658  0.132378
 0.216361  0.969999  0.619464

julia> A[3,:] = 1.0e-8*A[3,:];

julia> cond(A)
3.4838132191896 83e8
```

make a Gauss-Seidel preconditioner

```
julia> M = LowerTriangular(A)
3x3 LowerTriangular{Float64,Array{Float64,2}}:
 0.525208     .            .
 0.67043     0.536658      .
 2.16361e-9  9.69999e-9   6.19464e-9

julia> B = inv(M)*A
3x3 Array{Float64,2}:
 1.0           1.26977    1.09994
 0.0          -0.586282  -1.12744
 1.11022e-16   2.04041    2.38125

julia> cond(B)
14.79930403149552
```

an exercise

Exercise 4:

Consider the making of an ill-conditioned matrix by multiplying the first column with 10^{-8}.

```
julia> A = rand(3,3)
3x3 Array{Float64,2}:
 0.906183  0.193067  0.904161
 0.768742  0.202677  0.398809
 0.558759  0.840976  0.77066

julia> A[:,1] = 1.0E-8*A[:,1];

julia> cond(A)
3.8254496346866 44e8
```

Does a Gauss-Seidel preconditioner improve the conditioning?