Iterative Methods and Preconditioning for Sparse Linear Systems

Luca Bergamaschi
Department of Civil Environmental and Architectural Engineering
e-mail:luca.bergamaschi@unipd.it, webpage: www.dmsa.unipd.it/~berga

Ángeles Martinez
Department of Mathematics "Tullio Levi-Civita"
e-mail:angeles.martinez@unipd.it, webpage: www.math.unipd.it/~acalomar

Università degli Studi di Padova

Outline

- **Lecture # 1**. Sparse matrices. Why iterative methods. Linear algebra preliminaries. Basics on iterative methods. The Conjugate Gradient (CG) method. Convergence properties and implementation.
- **Lecture # 2**. Optimality properties of the CG method. Krylov subspaces. Nonsymmetric linear systems. Krylov-based iterative methods. The Generalized Minimal RESidual method (GMRES).
- **Lecture # 3**. Acceleration of iterative methods: preconditioning.The Incomplete Cholesky (IC) and the Incomplete LU (ILU) preconditioners. Parallel-oriented preconditioners. (Block) Jacobi, sparse approximate inverse preconditioners.
- **Lecture # 4**. Low rank updates of preconditioner for sequences of linear systems arising from the Newton's method.
- **Lecture # 5**. The tuned preconditioners. Applications to eigenvalues problems and sequences of shifted linear systems arising from PDEs.
- **Lecture # 6**. Block preconditioners for saddle-point problems.
- **Lecture # 7**. The Constraint Preconditioner for linear systems arising in Interior Point methods for Constrained Optimization. Acceleration by low-rank preconditioners.

Update of preconditioners by low-rank matrices

Update of preconditioners by low-rank matrices

Newton's method for nonlinear systems

We need to efficiently solve nonlinear systems of equations of the type

$$\mathbf{F}(\mathbf{x}) = 0$$

where $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n$, differentiable in an open set $\Omega \subset \mathbb{R}^n$. by the Newton method

$$\begin{cases} J(\mathbf{x}_k)\mathbf{s}_k & = & -\mathbf{F}(\mathbf{x}_k) \\ \mathbf{x}_{k+1} & = & \mathbf{x}_k + \mathbf{s}_k \end{cases}$$

and $J(x)$ is the Jacobian matrix.

The linear systems are large, sparse, and possibly nonsymmetric.

Problem

- We look for a sequence of preconditioner $\{B_k\}$ such that $||I - B_k J(x_k)||$ is sufficiently small
- To construct B_k, we use information from the nonlinear iteration

Quasi-Newton approach

- The idea is to start with a preconditioner B_0^{-1} for $J_0 \equiv J(\mathbf{x}_0)$.
- Correct the previous preconditioner by a rank one update

$$B_{k+1} = B_k + \mathbf{u}\mathbf{v}^T$$

- B_{k+1} must satisfy the **secant condition**, namely

$$B_{k+1}\mathbf{s}_k = \mathbf{y}_k$$

- where $\mathbf{y}_k = \mathbf{F}_{k+1} - \mathbf{F}_k$.
- **Remark**. There are infinitely many matrices B_{k+1} satisfying the secant condition (n constraint n^2 degrees of freedom).

Determining vectors u and v

- From the secant condition and the definition of B_{k+1}, we obtain that

$$\mathbf{u} = \frac{\mathbf{y}_k - B_k \mathbf{s}_k}{\mathbf{v}^T \mathbf{s}_k}.$$

- To get a unique B_{k+1} we impose that B_{k+1} is the closest matrix to B_k in the Frobenius norm

$$B_{k+1} = \underset{B:\ Bs_k = y_k}{\operatorname{argmin}} \|B - B_k\|_F$$

(Recall $\|A\|_F = \sqrt{\sum_{i=1}^n \sum_{j=1}^n a_{ij}^2}$).

- obtaining

$$\mathbf{v} = \frac{\mathbf{s}_k}{\|\mathbf{s}_k\|}.$$

Preconditioners

- Given B_k, B_{k+1} is defined as

$$B_{k+1} = B_k + \frac{(\mathbf{y}_k - B_k \mathbf{s}_k)\mathbf{s}_k^T}{\mathbf{s}_k^T \mathbf{s}_k}.$$

- It is advisable to have the preconditioner in its inverse form. Defined $P_k = B_k^{-1}$ then
- Applying the Shermann-Morrison inverse formula

$$P_{k+1} = P_k - \frac{(P_k \mathbf{y}_k - \mathbf{s}_k)\mathbf{s}_k^T P_k}{\mathbf{s}_k^T P_k \mathbf{y}_k}.$$

- The $k-$th linear system: $J(\mathbf{x}_k)\mathbf{s}_k = -\mathbf{F}(x_k)$ is solved by an iterative method preconditioned with P_k.

Theoretical analysis of the preconditioners

We will make the *standard assumptions* on $\mathbf{F}$.

1. Equation $F(x) = 0$ has a solution $\mathbf{x}^*$.
2. $F' : \Omega \to \mathbb{R}^n \times \mathbb{R}^n$ is Lipschitz continuous with constant γ.
3. $F'(\mathbf{x}^*)$ is nonsingular.

Notation

- $J_k \mathbf{s} = -\mathbf{F}_k \quad$ where $\quad \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}$.
- Error vectors $\mathbf{e}_k = \mathbf{x}^* - \mathbf{x}_k$
- Error matrices $E_k = B_k - J(\mathbf{x}^k)$,

Bounded deterioration property

Lemma

Let the standard assumptions hold. Then

$$\|E_+\| \leq \|E_c\| + \gamma \frac{(\|\mathbf{e}_c\| + \|\mathbf{e}_+\|)}{2}$$

This property assures that the distance of B_k to the Jacobian in the exact solution does not grow.

Moreover the sequence of B_k is well defined as

Theorem

Let the standard assumptions hold. Define $\alpha = \|J(\mathbf{x}^*)^{-1}\|$. Fixed $0 < \delta_1 < \frac{1}{\alpha}$, then there exist δ and δ_B such that if $\|\mathbf{e}_0\| < \delta$ and $\|E_0\| < \delta_B$ then

$$\|B_k^{-1}\| < \frac{\alpha}{1 - \delta_1 \alpha}, \quad \forall k > 0$$

Properties: $\|I - P_k J(\mathbf{x}_k)\|$ bounded.

Finally, the distance between the preconditioned matrix $P_k J(\mathbf{x}_k)$ and the identity matrix can be made as small as desired by starting sufficiently close to the solution, and choosing a good initial preconditioner.

Theorem

Let the standard assumptions hold. Define $\alpha = \|J(\mathbf{x}^*)^{-1}\|$. *Fixed* $0 < \delta_1 < \dfrac{1}{\alpha}$, *then there are* δ, δ_B *such that if* $\|\mathbf{e}_0\| < \delta$, $\|E_0\| < \delta_B$ *then*

$$\|I - P_k J_k\| < \frac{\delta_1 \alpha}{1 - \delta_1 \alpha}, \qquad \forall k > 0$$

Bergamaschi Bru Martinez Putti
Quasi Newton preconditioners for the Inexact Newton method.
Electronic Transaction on Numerical Analysis, 2006

Notes on implementation

How to apply the preconditioner to a vector.

- At a certain nonlinear iteration level, k, and given $\mathbf{z}_k^{(l)}$, we want to compute

$$\mathbf{c} = P_k \mathbf{z}_k^{(l)}$$

- Recall the final preconditioner

$$P_{k+1} = P_k - \frac{(P_k \mathbf{y}_k - \mathbf{s}_k)\, \mathbf{s}_k^T P_k}{\mathbf{s}_k^T P_k \mathbf{y}_k}.$$

- Setting $\mathbf{v}_k = \dfrac{\mathbf{s}_k}{\|\mathbf{s}_k\|}$, $\quad \mathbf{u}_k = \dfrac{\mathbf{y}_k - B_k \mathbf{s}_k}{\|\mathbf{s}_k\|}$ and $\mathbf{w}_k = \dfrac{P_k \mathbf{u}_k}{1 + \mathbf{v}_k P_k \mathbf{u}_k}$,
- $P_k = \left(I - \mathbf{w}_{k-1} \mathbf{v}_{k-1}^T\right) P_{k-1}$

$$= \left(I - \mathbf{w}_{k-1} \mathbf{v}_{k-1}^T\right) \left(I - \mathbf{w}_{k-2} \mathbf{v}_{k-2}^T\right) \cdots \left(I - \mathbf{w}_0 \mathbf{v}_0^T\right) P_0$$

Notes on implementation

- $k = 0$, we compute, at each iteration

$$\mathbf{c} = P_0 \mathbf{z}_k^{(l)}$$

- $k > 0$, before starting the iteration we have to compute

$$\mathbf{u}'_{k-1} = P_{k-1}\mathbf{u}_{k-1}, \qquad \text{and} \qquad \mathbf{w}_{k-1} = \frac{\mathbf{u}'_{k-1}}{1 + \mathbf{v}_{k-1}^T \mathbf{u}'_{k-1}}$$

and, at every linear iteration,

$$\begin{aligned} \mathbf{c} &= P_k \mathbf{z}_k^{(l)} = \left(I - \mathbf{w}_{k-1}\mathbf{v}_{k-1}^T\right) P_{k-1}\mathbf{z}_k^{(l)} \\ &= \left(I - \mathbf{w}_{k-1}\mathbf{v}_{k-1}^T\right)\cdots\left(I - \mathbf{w}_0\mathbf{v}_0^T\right) P_0 \mathbf{z}_k^{(l)} \end{aligned}$$

Notes on implementation

In summary

- Before the $k-$th linear system solution. Assume we know $\mathbf{u}_i, \mathbf{v}_i, \mathbf{w}_i, \ i = 1, \dots, k-2$.

  Compute $\mathbf{u}_{k-1}, \mathbf{v}_{k-1}$. Then compute $P_{k-1}\mathbf{u}_{k-1}$ at the price of an application of P_0, $k-2$ dot products and $k-2$ daxpy operations, by

  $$P_{k-1}\mathbf{u}_{k-1} = \left(I - \mathbf{w}_{k-2}\mathbf{v}_{k-2}^T\right) \cdots \left(I - \mathbf{w}_0\mathbf{v}_0^T\right) P_0\mathbf{u}_{k-1}$$

  implemented as

  $$\begin{array}{ll} \mathbf{z} = P_0\mathbf{u}_{k-1}; & \alpha = \mathbf{v}_0^T\mathbf{z}, \ \mathbf{z} = \mathbf{z} - \alpha\mathbf{w}_0 \\ & \alpha = \mathbf{v}_1^T\mathbf{z}, \ \mathbf{z} = \mathbf{z} - \alpha\mathbf{w}_1 \\ & \dots \\ & \alpha = \mathbf{v}_{k-2}^T\mathbf{z}, \ \mathbf{z} = \mathbf{z} - \alpha\mathbf{w}_{k-2} \end{array}$$

- At each linear iteration

  $$\mathbf{c} = \left(I - \mathbf{w}_{k-1}\mathbf{v}_{k-1}^T\right) \cdots \left(I - \mathbf{w}_0\mathbf{v}_0^T\right) P_0\mathbf{z}_k^{(l)}$$

  at the price of an application of P_0, $k-1$ dot products and $k-1$ daxpys.

Newton-Broyden (NB) Algorithm

- Stopping criterion (Inexact Newton methods)

$$\|J(\mathbf{x}_k)\mathbf{s}_k + \mathbf{F}(\mathbf{x}_k)\| \le \eta_k \|\mathbf{F}(\mathbf{x}_k)\|.$$

with $\lim_{k \to \infty} \eta_k = 0$.

Linear systems are solved with increasing accuracy as the Newton iteration proceed.

```
NEWTON-BROYDEN (NB) ALGORITHM
Input: $\mathbf{x}_0, \mathbf{F}, nlmax$, toll
  ■ WHILE $\|\mathbf{F}(\mathbf{x}_k)\| >$ toll AND $k < nlmax$ DO
      1 Compute $P_0$ a preconditioner for $J_0$; $k = 0$
      2 IF $k > 0$ THEN update $P_k$ from $P_{k-1}$.
      3 Solve $J(\mathbf{x}_k)\mathbf{s}_k = -\mathbf{F}(\mathbf{x}_k)$ by an iterative method with preconditioner $P_k$ and
        tolerance $\eta_k$.
      4 $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$
      5 $k = k + 1$
  ■ END WHILE
```

Drawback: increasing costs of memory for w_k and v_k.

Restarted Newton-Broyden (RNB) Algorithm

Like in GMRES only $k_{\max}$ vectors $\mathbf{w}_k$ and $\mathbf{v}_k$ are stored.

After $k_{\max}$ Newton iterations a new "initial" preconditioner P_0 is computed and the sequence of preconditioners is restarted.

```
RESTARTED NEWTON-BROYDEN (RNB) ALGORITHM
Input: $\mathbf{x}_0, \mathbf{F}, k_{\max}, nlmax$, toll
  ■ Compute $P_0$, a preconditioner for $J_0$; $k = 0$
  ■ WHILE $\|\mathbf{F}(\mathbf{x}_k)\| > \text{toll}$ AND $k < nlmax$ DO
      1 IF $k > 0$ THEN update $P_k$ from $P_{k-1}$.
      2 Solve $J(\mathbf{x}_k)\mathbf{s}_k = -\mathbf{F}(\mathbf{x}_k)$ by an iterative method with preconditioner $P_k$.
      3 $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$
      4 $k = k + 1$
      5 IF $k$ MOD $k_{\max} = 0$ THEN
            ■ RESTART: $\mathbf{x}_0 = \mathbf{x}_k$; $k = 0$; compute $P_0$ a preconditioner for $J_0$
  ■ END WHILE
```

Numerical problem

- Bratu problem:

$$-A\mathbf{u} = \lambda D(u), \qquad D = \mathrm{diag}(\exp(u_1), \ldots, \exp(u_n))$$

  where A is a matrix arising form a 2d or 3d discretization of the diffusion equation on a unitary domain, and λ is a real parameter.

- Matrices arising from discretization with 3D Finite Differences (FD), Mixed Finite Elements (MFE) of the diffusion equation.

- $\mathbf{x}_0 = (1, \ldots, 1)^T$.

- Fortran code

2d MFE discretization, $\lambda = 1$

Matrix A with 28600 rows and 142204 nonzeros.
Mixed Finite Element discretization of the diffusion equation.

Table: Results on MFE matrix with $B_0 = \text{ILU}(0)$.

preconditioner	$k_{\max}$	nlit	iter	cpu	
				tot	precond
ILU(0)(J_0)	–	7	851	11.59	0.01
ILU(0)(J_k)	–	7	754	8.65	0.04
RNB-ILU(0)	1	7	442	6.51	0.08
RNB-ILU(0)	2	7	470	6.56	0.06
RNB-ILU(0)	3	7	501	6.93	0.06
RNB-ILU(0)	5	7	529	7.09	0.06
NB-ILU(0)		6	515	9.67	0.06

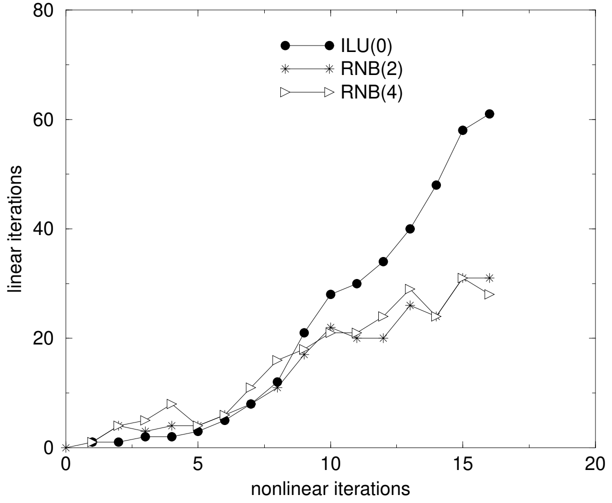
2d MFE discretization, $\lambda = 1$

Matrix A with 28600 rows and 142204 nonzeros.
Mixed Finite Element discretization of the diffusion equation.

Table: Results on MFE matrix with $B_0 =$ AINV(0.1).

preconditioner	$k_{\max}$	nlit	iter	cpu	
				tot	precond
AINV(0.1) (J_0)	–	7	882	18.35	0.17
AINV(0.1) (J_k)	–	7	908	19.70	1.27
RNB-AINV(0.1)	1	8	574	14.62	1.57
RNB-AINV(0.1)	2	7	517	13.07	0.81
RNB-AINV(0.1)	4	7	502	12.61	0.44
NB-AINV(0.1)		7	655	17.15	0.26

3d FE discretization



Quasi-Newton preconditioners for sequences of linear systems

Recall Newton's method for

$$F(\mathbf{x}) = 0, \qquad F : \mathbb{R}^n \to \mathbb{R}^n$$

$$\begin{aligned} F'(\mathbf{x}_k)\mathbf{s}_k &= -F(\mathbf{x}_k) \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \mathbf{s}_k \end{aligned}$$

Quasi-Newton methods construct a sequence of approximations of the Jacobians $B_k \approx F'(\mathbf{x}_k)$.

Each B_k is defined by a low-rank update of the previous matrix in the sequence B_{k-1}.

Most common Quasi-Newton formulæ

Broyden's method: $$B_{k+1} = B_k + \frac{(\mathbf{y}_k - B_k\mathbf{s}_k)\mathbf{s}_k^T}{\mathbf{s}_k^T\mathbf{s}_k}$$

SR1 (Symmetric Rank-1): $$B_{k+1} = B_k + \frac{(\mathbf{y}_k - B_k\mathbf{s}_k)(\mathbf{y}_k - B_k\mathbf{s}_k)^T}{(\mathbf{y}_k - B_k\mathbf{s}_k)^T\mathbf{s}_k}$$

BFGS update: $$B_{k+1} = B_k + \frac{\mathbf{y}_k\mathbf{y}_k^T}{\mathbf{y}_k^T\mathbf{s}_k} - \frac{B_k\mathbf{s}_k\mathbf{s}_k^T B_k}{\mathbf{s}_k^T B_k\mathbf{s}_k}$$

Tuning property

All these updates satisfy the *secant condition* as they all satisfy

$$B_{k+1}\mathbf{s}_k = \mathbf{y}_k.$$

Consider now the following substitutions:

$$\mathbf{s}_k \longrightarrow \mathbf{w}, \qquad \mathbf{y}_k \longrightarrow A\mathbf{w}, \qquad B_k \longrightarrow M_0, \qquad B_{k+1} \longrightarrow M,$$

which transform the secant condition into

$$M\mathbf{w} = A\mathbf{w},$$

We call this *TUNING PROPERTY*.

The preconditioner M acts as the coefficient matrix A in at least one direction.
or, equivalently

$$PA\mathbf{w} = \mathbf{w},$$

The preconditioned matrix PA has at least one additional eigenvalue at 1.

Direct, inverse and block formulations

Broyden (nonsymmetric) update.

direct	$M = M_0 + \frac{(A - M_0)\mathbf{w}\mathbf{w}^T}{\mathbf{w}^T\mathbf{w}}$	$M\mathbf{w} = A\mathbf{w}$
inverse	$P = P_0 - \frac{(P_0 A\mathbf{w} - \mathbf{w})\mathbf{w}^T P_0}{\mathbf{w}^T P_0 A\mathbf{w}}$	$PA\mathbf{w} = \mathbf{w}$
block	$P = P_0 - (P_0 AW - W)\left(W^T P_0 AW\right)^{-1} W^T P_0$	$PAW = W$

SR1 (symmetric but not PD) update.

direct	$M = M_0 + \frac{\mathbf{u}\mathbf{u}^T}{\mathbf{w}^T\mathbf{u}}$,	$\mathbf{u} = (A - M_0)\mathbf{w}$
inverse	$P = P_0 - \frac{\mathbf{z}\mathbf{z}^T}{\mathbf{z}^T A\mathbf{w}}$,	$\mathbf{z} = P_0 A\mathbf{w} - \mathbf{w}$
block	$P = P_0 - Z\left(Z^T AW\right)^{-1} Z^T$,	$Z = P_0 AW - W$

BFGS (SPD) update.

direct	$M = M_0 + \frac{A\mathbf{w}\mathbf{w}^T A}{\mathbf{w}^T A\mathbf{w}} - \frac{M_0\mathbf{w}\mathbf{w}^T M_0}{\mathbf{w}^T M_0\mathbf{w}}$
inverse	$P = \frac{\mathbf{w}\mathbf{w}^T}{\mathbf{w}^T A\mathbf{w}} + \left(I - \frac{\mathbf{w}\mathbf{w}^T A}{\mathbf{w}^T A\mathbf{w}}\right) P_0 \left(I - \frac{A\mathbf{w}\mathbf{w}^T}{\mathbf{w}^T A\mathbf{w}}\right)$
block	$P = W\Pi^{-1}W^T + HP_0H^T \qquad \Pi = W^T AW \qquad H = I - W\Pi^{-1}W^T A$

The BFGS sequence is SPD

The Broyden tuning strategy must be used for nonsymmetric problems, the BFGS formula is well suited to accelerate the PCG method due to the following result:

Theorem

The preconditioner P yielded by the BFGS update formula is SPD provided P_0 is so.

Proof.

For every nonzero $\mathbf{x} \in \mathbb{R}^n$ we set $\mathbf{z} = H^T\mathbf{x}$ and $\mathbf{u} = W^T\mathbf{x}$. Then we have

$$\mathbf{x}^T P\mathbf{x} = (W^T\mathbf{x})^T \Pi^{-1}(W^T\mathbf{x}) + \mathbf{x}^T HP_0H^T\mathbf{x} = \mathbf{u}^T\Pi^{-1}\mathbf{u} + \mathbf{z}^T P_0\mathbf{z} \geq 0.$$

the last inequality holding since both Π^{-1} and P_0 are SPD matrices.

The inequality is strict since if $\mathbf{u} = 0$ then $W^T\mathbf{x} = 0$ and hence

$$\mathbf{z} = (I - AW\Pi^{-1}W^T)\mathbf{x} = \mathbf{x} \neq 0.$$

□

When is the SR1 update SPD too?

Theorem

Let A be an SPD matrix and P_0 and SPD preconditioner. If the columns of W are eigenvectors corresponding to the p smallest eigenvalues of P_0A, $\mu_j, j=1,\dots,p$ and $\mu_j < 1, \quad j=1,\dots,p$ then $P = P_0 - Z\left(Z^TAW\right)^{-1}Z^T$ is SPD.

Proof.

It is sufficient to prove that $-Z^TAW$ is SPD (Recall $Z = W - P_0AW$).

Matrix W satisfies $\quad P_0AW = W\Theta, \quad$ with $\quad \Theta = \text{diag}(\mu_1, \dots, \mu_p)$.

$$P_0AW = W\Theta \quad \Longrightarrow \quad \underbrace{P_0^{1/2}AP_0^{1/2}}_{\hat{A}}\underbrace{P_0^{-1/2}W}_{U} = \underbrace{P_0^{-1/2}W}_{U}\Theta \quad \Longrightarrow \quad \hat{A}U = U\Theta.$$

Now $\hat{A}$ is SPD so it has orthonormal eigenvectors (the columns of U). Hence

$$U^TU = I \quad \Longrightarrow \quad W^TP_0^{-1}W = I \quad \Longrightarrow \quad W^TAW = \Theta.$$

It follows that

$$-Z^TAW = (W - P_0AW)^TAW = (I-\Theta)W^TAW = (I-\Theta)\Theta$$

is SPD.

□

Other approaches

Deflation.

Saad et al (2000) proposed a *deflated version of the CG* in which the follwing deflated preconditioner is defined

$$H = I - W(W^T AW)^{-1} W^T A$$

The action of this precondtioner is to put m eigenvalues to zero!

However all the CG residuals are forced to lie in the rank of H (no breakdown can occurr).

Saad, Y. and Yeung, M. and Erhel, J. and Guyomarch, F.,
A deflated version of the conjugate gradient algorithm,
SIAM J. Sci. Comput., 2000

Spectral Preconditioners

Given a full-rank rectangular (tall) matrix W and an initial preconditioner P_0, the preconditioner P is defined as

$$P = P_0 + W(W^T AW)^{-1} W^T$$

The action of this preconditioner, if W contain the approximate eigenvectors corresponding to the smallest eigenvalues, is to add 1 to these eigenvalues

Assume $P_0 AW = W\Lambda$ then

$$(PA)W = (P_0 A)W + W(W^T AW)^{-1} W^T AW = W\Lambda + W = W(\Lambda + I).$$

B. Carpentieri and I. S. Duff and L. Giraud,
A class of spectral two-level preconditioners,
SIAM J. Sci. Comput., 2003

Choice of the vectors $\{\mathbf{w}_j\}$.

In all cases: optimal choice for columns of W: the eigenvectors of the **preconditioned matrix** P_0A corresponding to the smallest eigenvalues.

Shall we compute these eigenvector accurately?

To test this we used perturbed eigenvectors i.e. satisfying $\|P_0A\mathbf{w}_j - \mu_j\mathbf{w}_j\| \approx \delta$.

Coefficient matrix

```
A = delsq(numgrid('L',500));
```

which returns a sparse matrix of order $n = 186003$.

The linear system $A\mathbf{x} = \mathbf{b}$, with $\mathbf{b}$ a random uniformly distributed vector, has been solved by the PCG method with various low-rank update techniques with $P_0 = IC(0)$.

	no update	tuned	deflated	spectral
exact	466	254	254	254
$\delta = 0.01$	466	261	259	290
$\delta = 0.05$	466	378	260	286

Notable improvement of number of iterations even with badly approximated eigenvectors (the tuned version being more sensitive to accuracy).

Can we use different vectors?

What if one uses **eigenvectors of** A as columns of W?

Again we use either exact eigenvectors or vectors satisfying $\|A\mathbf{w}_j - \lambda_j \mathbf{w}_j\| \approx \delta$.

	no update	tuned	deflated	spectral
exact	466	254	254	254
$\delta = 10^{-3}$	466	296	296	297
$\delta = 0.01$	466	362	361	369

With exact eigenvectors there is still an important reduction of the number of iterations. Why?

An experimental answer

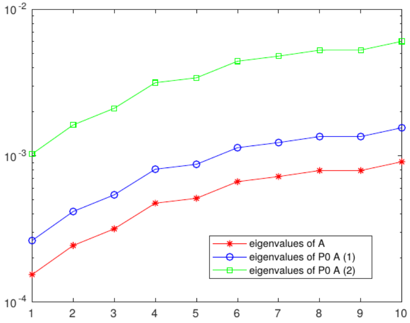
Usually IC/ILU preconditioners leave almost unchanged the eigenvectors corresponding to the smallest eigenvalues (though increasing the latter ones).

Compare the eigenpairs $(\lambda_j, \mathbf{v}_j)$ of A, with those of the preconditioned matrix with two choices of P_0:

- $IC(0)$ $(\mu_j^{P1}, \mathbf{v}_j^{P1})$
- IC$(1e-2)$ $(\mu_j^{P2}, \mathbf{v}_j^{P2})$.

Eigenvalues and angle between eigenvectors of A and IC-preconditioned A.

j	$\angle\left(\mathbf{v}_j, \mathbf{v}_j^{P1}\right)$	$\angle\left(\mathbf{v}_j, \mathbf{v}_j^{P2}\right)$
1	1.7445e-05	2.6952e-04
2	3.6118e-05	4.3401e-04
3	9.6668e-05	1.5363e-04
4	1.9743e-04	4.5222e-03
5	1.7449e-04	4.9131e-03



Recovering the leftmost eigenpairs

Lanczos' method

As known Lanczos' method aims at computing the extremal eigenvalues of an SPD matrix A by constructing an orthogonal basis of the Kryov subspace generated by A and a given initial vector $\mathbf{q}_1$.

Matrix Q defined as

$$Q_m = \begin{bmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \cdots \mathbf{q}_m \end{bmatrix}$$

satisfies

$$Q_m^T A Q_m = T_m$$

where T_m is a tridiagonal matrix of size m. (why does this remind me GMRES? in that case we had $V_m^T A V_m = H_m$.)

Finally some of the extremal eigenvalues are approximated by the extremal eigenvalues of the "small" matrix T_m.

PCG-Lanczos connection

A simple way to recover some of the extremal eigenpairs of the preconditioned matrix is to exploit the so called *Lanczos connection* (Golub and van Loan, Matrix Computations).

During the PCG method, preconditioned with P_0, it is possible to save the first m (scaled) preconditioned residuals as columns of a matrix V_m:

$$V_m = \left[\frac{P_0 \mathbf{r}_0}{\sqrt{\mathbf{r}_0^T P_0 \mathbf{r}_0}}, \frac{P_0 \mathbf{r}_1}{\sqrt{\mathbf{r}_1^T P_0 \mathbf{r}_1}}, \ldots, \frac{P_0 \mathbf{r}_{m-1}}{\sqrt{\mathbf{r}_{m-1}^T P_0 \mathbf{r}_{m-1}}} \right] = \left[\frac{\mathbf{z}_0}{\sqrt{\rho_0}}, \frac{\mathbf{z}_1}{\sqrt{\rho_1}}, \ldots, \frac{\mathbf{z}_{m-1}}{\sqrt{\rho_{m-1}.}} \right].$$

Note all these vectors an scalars are computed during PCG. No additional cost.

Matrix V_m satisfies $V_m^T P_0^{-1} V_m = I_m$.

Implicit Lanczos method within PCG

The Lanczos tridiagonal matrix can be formed using the PCG coefficients α_k, β_k:

$$T_m = \begin{bmatrix} \frac{1}{\alpha_0} & -\frac{\sqrt{\beta_1}}{\alpha_0} & & & \\ -\frac{\sqrt{\beta_1}}{\alpha_0} & \frac{1}{\alpha_1}+\frac{\beta_1}{\alpha_0} & -\frac{\sqrt{\beta_2}}{\alpha_1} & & \\ & & \ddots & & \\ & & & & -\frac{\sqrt{\beta_{m-1}}}{\alpha_{m-2}} \\ & & & -\frac{\sqrt{\beta_{m-1}}}{\alpha_{m-2}} & \frac{1}{\alpha_{m-1}}+\frac{\beta_{m-1}}{\alpha_{m-2}} \end{bmatrix}$$

Matrices V_m and T_m obey to the classical Lanczos relation i.e.:

$$V_m^T A V_m = T_m.$$

Practically during PCG:

1. Collect $m = 50, 70, 100$ preconditioned residuals, and T_m.
2. Eigensolve T_m obtaining $T_m = Q \Lambda_m Q^T$.
3. Select the p smallest eigenvalues and eigenvectors $Q_p = Q(1:p)$.
4. Project the small matrix Q_m to obtain approximation of the eigenvectors of A: $W_p = V_m Q_p$

Remarks

- This procedure can be implemented to a very little computational cost but it has a number of disadvantages:
- First, it requires the storage of m preconditioned residuals,
- Second, as the convergence for the Lanczos process to the smallest eigenvalues is relatively slow, it sometimes happens that PCG convergence takes place before eigenvector convergence.
- Third, some of the leftmost eigenpairs can be missing by the non-restarted Lanczos procedure.

Remedy to drawbacks 2 and 3

If a sequence of linear systems has to be solved then the eigeninformation for matrix $P_0 A$ can be refined during the first linear systems and then used for the next ones.

Stathopoulos, A. and Orginos, K.
Computing and deflating eigenvalues while solving multiple right-hand side linear systems with an application to quantum chromodynamics
SIAM Journal on Scientific Computing

Exercise

Implementation of the SR1 update within the Matlab PCG.

$$P = P_0 - Z\left(Z^T AW\right)^{-1} Z^T, \qquad Z = P_0 AW - W$$

```
function z = sr1(x,L,Z,H)
y = L'\(L\x);
u = Z'*x;
u = H*u;
z = y - Z*u;
```

Approximation of the leftmost eigenpairs of $P_0 A$ by solving the generalized eigenproblem $A\mathbf{x} = \lambda(LL^T)\mathbf{x}$ by function eigs.

```
[W, Lambda] = eigs(A, L*L',p,'sm','tolerance',1e-3);
```

Preprocessing

```
Z = L'\(L\(A*W))-W;
H = inv(W'*(A*Z));
```

Invoking the PCG with a function handle as the preconditioner

```
[x,f,rel,it,resvecTUN] = pcg(A, b, TOL,MAXIT,@(x) sr1(x,L,Z,H));
```