• Gradient iterative methods

- basic idea: for symmetric and positive definite matrices the solution of $Ax=b$ is equivalent of finding a minimum of the quadratic form

$$\boxed{\phi(x) = \frac{1}{2} x^T A x - x^T b} \qquad x \in \mathbb{R}^n$$

because

$$\nabla\phi(x) = \frac{1}{2}\left[Ax + (x^T A)^T\right] - b = Ax - b = 0$$

(for $A = A^T$)

- we search for x iteratively by choosing an initial guess x_0 and then by choosing a suitable direction p_k at step k with a new approximation taken as a minimum of $\phi(x_k + \alpha p_k)$ in the direction of p_k with respect to α

by rewriting

$$\phi(x_k+\alpha p_k) = \frac{1}{2}(x_k+\alpha p_k)^T A (x_k+\alpha p_k) - (x_k+\alpha p_k)^T b$$

$$= \underbrace{\frac{1}{2} x_k^T A x_k - x_k^T b}_{\text{constant term}} + \frac{\alpha}{2}\underbrace{(p_k^T A x_k + x_k^T A p_k)}_{2 p_k^T A x_k \text{ if } A \text{ is symmetric}} - \alpha p_k^T b + \frac{\alpha^2}{2} p_k^T A p_k =$$

$$= \frac{1}{2} x_k^T A x_k - x_k^T b + \alpha\left[p_k^T \underbrace{(Ax_k - b)}_{-r_k}\right] + \frac{\alpha^2}{2} p_k^T A p_k$$

and by taking derivative we set

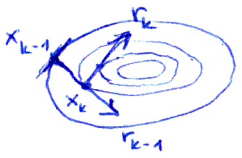
$$0 = \frac{\partial\phi}{\partial\alpha} = -p_k^T r_k + \alpha p_k^T A p_k \Rightarrow \boxed{\alpha_k = \frac{p_k^T r_k}{p_k^T A p_k}}$$

where a residual vector in the k-th iteration is

$$\boxed{r_k = b - Ax_k}$$

- choice of the direction:

method of the steepest descent



we search for minimum in the direction of gradient $\nabla\phi$

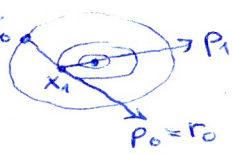
or $\nabla\phi(x_k) = Ax_k - b = -r_k$ and thus $p_k = r_k$

and $\alpha_k = (r_k^T r_k)/(r_k^T A r_k)$

conjugate gradient method

we search for minimum in the A-conjugated direction

$p_k^T A p_i = 0$ for all $i = 0, \dots, k-1$



in 2D we get in two steps to solution (see later)

a) Method of steepest descent

simple implementation:

pick x_0

for $k = 1, 2, \dots$

$r_{k-1} = b - A x_{k-1}$

if $\| r_{k-1} \| < \varepsilon$ stop

$\alpha_{k-1} = \frac{r_{k-1}^T r_{k-1}}{r_{k-1}^T A r_{k-1}}$ } two multiplications by A !

$x_k = x_{k-1} + \alpha_{k-1} r_{k-1}$

standard algorithm:

pick x_0

$r_0 = b - A x_0$

for $k = 1, 2, \dots$

$w_{k-1} = A r_{k-1}$

$\alpha_{k-1} = \frac{r_{k-1}^T r_{k-1}}{r_{k-1}^T w_{k-1}}$

$x_k = x_{k-1} + \alpha_{k-1} r_{k-1}$

$r_k = r_{k-1} - \alpha_{k-1} w_{k-1}$

if $\| r_k \| < \varepsilon$ stop

- it is better to get rid of $A x_{k-1}$

using $r_k = b - A x_k = b - A(x_{k-1} + \alpha_{k-1} r_{k-1})$

$= r_{k-1} - \alpha_{k-1} A r_{k-1}$

- for badly conditioned matrices A we get slow convergence

if condition number

$$\mathrm{cond}(A) = \frac{\max_j |\lambda_j|}{\min_j |\lambda_j|} \gg 1$$

then we have very disproportionate ellipses

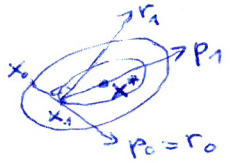
we always go in the direction perpendicular to the previous one:

$$r_{k-1}^T r_k = r_{k-1}^T (r_{k-1} - \alpha_{k-1} A r_{k-1}) = 0$$ (thanks to the choice of α_{k-1})

which is not optimal at all.

b) Conjugate gradient method (CG)

- motivation for choice of the direction in 2D



we know that $r_0^T r_1 = 0 = p_0^T r_1$ where $r_1 = -\nabla \phi(x_1)$

and thus

$$0 = p_0^T r_1 = p_0^T (b - A x_1) = p_0^T (A x^* - A x_1) =$$

$$= p_0^T A \underbrace{(x^* - x_1)}_{\alpha p_1} = \alpha\, p_0^T A p_1 \quad \text{for certain } \alpha$$

and if A is symmetric

we can write the condition $p_1^T A p_0 = 0$ (p_1 must be A-conjugated to p_0)

- choice is obvious in 2D, but in higher dimensions the choice of p_1 is not unique (we don't know x^* :-))

- but if we would be able to choose p_k A-conjugated to all previous p_j we at most n steps (for $n \times n$ matrix) would end in x^* :)

- let us write down first the algorithm of CG method without proving that p_k is A-conjugated to the previous directions

CG algorithm without preconditioning (which will be shown later to be important for convergence)

pick x_0 (e.g. $x_0 = 0$)

$r_0 = b - A x_0$

$p_0 = r_0$, $\gamma_0 = r_0^T r_0$

for $k = 1, 2, \dots$

this part is basically the same as in the method of steepest descent

$w_{k-1} = A p_{k-1}$

$\alpha_{k-1} = \gamma_{k-1} / (p_{k-1}^T w_{k-1})$

$x_k = x_{k-1} + \alpha_{k-1} p_{k-1}$

$r_k = r_{k-1} - \alpha_{k-1} w_{k-1}$

if $\|r_k\| < \varepsilon$ stop

choice of a new optimal direction

$\gamma_k = r_k^T r_k$

$\beta_{k-1} = \gamma_k / \gamma_{k-1}$

$p_k = r_k + \beta_{k-1} p_{k-1}$

notes:

1) in exact arithmetics it is not an iterative method as such because in n steps will end, finds the solution

but convergence is not guaranteed in floating-point arithmetics due to round-off errors

and it often converges in less than n steps (especially with preconditioning)

2) again only one A.v evaluation and moreover just two scalar products because instead of $p_{k-1}^T r_{k-1}$ it is used $r_{k-1}^T r_{k-1}$ because thanks to $p_{k-2}^T r_{k-1} = 0$ we have $p_{k-1}^T r_{k-1} = \{$ thanks to the choice of $\alpha \}$ $= (r_{k-1} + \beta_{k-2} p_{k-2})^T r_{k-1} = r_{k-1}^T r_{k-1}$

- we can show by induction (see the next page) that r_k and p_k satisfy

1) $p_k^T A p_j = 0$ for all $j = 0, \dots, k-1$

2) $r_k^T r_j = 0$ for all $j = 0, \dots, k-1$

moreover subspaces

$\mathcal{L}(p_0, \dots p_{k-1})$, $\mathcal{L}(r_0, A r_0, \dots, A^{k-1} r_0)$

and $\mathcal{L}(A e_0, \dots A^k e_0)$ with $e_0 = x^* - x_0$

are the same and this subspace is called k-th Krylov subspace (A, r_0)

(note that for $x_0 = 0$, we have $r_0 = b$ and thus it is (A, b))

this can be seen from:

$p_k = r_k + \beta_{k-1} p_{k-1} = r_k + \beta_{k-1} (r_{k-1} + \beta_{k-2}(r_{k-2} + \dots))$ and $p_0 = r_0$

we have $\mathcal{L}(p_0, \dots p_k) = \mathcal{L}(r_0, \dots, r_k) = \mathcal{L}(r_0, A r_0, \dots A^{k-1} r_0) =$

and from $r_k = r_{k-1} - \alpha_{k-1} A(r_{k-1} + \beta_{k-2}(r_{k-2} \dots))$ $= \mathcal{L}(A e_0, A^2 e_0, \dots A^k e_0)$

and because $A e_0 = A(x^* - x_0) = b - A x_0 = r_0$

- let us show that p_k are A-conjugated to p_j, $j = 0, \dots, k-1$
and that moreover $r_k^T r_j = 0$ for all $j \leq k-1$

(*) we use $p_k^T r_{k+1} = p_k^T (r_k - \alpha_k A p_k) = 0 = r_{k+1}^T p_k$ (thanks to $\alpha_k = \frac{p_k^T r_k}{p_k^T A p_k}$)

(**) and $p_k^T r_k = (r_k + \beta_{k-1} \underbrace{p_{k-1}}_{0})^T r_k = r_k^T r_k = \|r_k\|_2^2 \Rightarrow \alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$

1) $r_1^T r_0 = r_1^T p_0 = 0$ (thanks to (*))

$r_1 = r_0 - \alpha_0 A p_0$

2) $p_1^T A p_0 = (r_1 + \beta_0 p_0)^T A p_0 = (r_1 + \beta_0 p_0)^T \left(\frac{r_1 - r_0}{-\alpha_0}\right) = -\frac{1}{\alpha_0}\left(r_1^T r_1 - \beta_0 r_0^T r_0\right) = 0$

thanks to the choice of β_0

(actually this is why β_0 is chosen as $r_1^T r_1 / r_0^T r_0$!)

3) induction: let us assume $r_{k-1}^T r_j = 0$ for $j = 0, \dots, k-2$

$p_{k-1}^T A p_j = 0$ for $j = 0, \dots, k-2$

and $p_{k-1}^T A r_j = p_{k-1}^T A (p_j - \beta_{j-1} p_{j-1}) = 0$ for $j = 0, \dots, k-2$

then

a) $r_k^T r_j = (r_{k-1}^T - \alpha_{k-1} (A p_{k-1})^T) r_j = 0$ for $j = 0, \dots, k-2$

from induction assumptions

b) $r_k^T r_{k-1} = r_{k-1}^T r_{k-1} - \alpha_{k-1} p_{k-1}^T A r_{k-1} = r_{k-1}^T r_{k-1} - \frac{r_{k-1}^T r_{k-1}}{p_{k-1}^T A p_{k-1}} p_{k-1}^T A r_{k-1}$

but $p_{k-1}^T A p_{k-1} = p_{k-1}^T A r_{k-1} + \beta_{k-2} \underbrace{p_{k-1}^T A p_{k-2}}_{0 \text{ (induction assumptions)}} = p_{k-1}^T A r_{k-1}$

and thus $r_k^T r_{k-1} = 0$ $\quad r_j - \alpha_j A p_j = r_{j+1}$

c) $p_k^T A p_j = (r_k + \beta_{k-1} \underbrace{p_{k-1}}_{0 \text{ for } j = 0, \dots, k-2})^T A p_j = r_k^T \left(\frac{(r_j - r_{j+1})}{\alpha_j}\right) = 0$ thanks to a) and b)

for all $j = 0, \dots, k-2$

d) $p_k^T A p_{k-1} = (r_k + \beta_{k-1} p_{k-1})^T \frac{(r_{k-1} - r_k)}{\alpha_{k-1}} = \left\{ \text{using } r_k^T r_{k-1} = 0 \text{ b)} \right\}$

$= \frac{1}{\alpha_{k-1}} \left[\beta_{k-1} \underbrace{p_{k-1}^T r_{k-1}}_{r_{k-1}^T r_{k-1} \text{ (see (**))}} - r_k^T r_k - \beta_{k-1} \underbrace{p_{k-1}^T r_k}_{0 \text{ (see (*))}} \right] = 0$

$\beta_{k-1} = \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}}$

again, the last equality shows why β_{k-1} is chosen as it is in algorithm

Convergence of the conjugate gradient method

- because we deal with symmetric, positive definite matrices, we can define a norm $\|v\|_A = \sqrt{v^T A v}$ as $v^T A v = 0$ only for $v = 0$ which is used to estimate the error of the CG method

- if we denote $e_k = x_k - x^*$ the error of the approximate solution x_k after k steps, x^* being the exact solution of $Ax^* = b$, we get

$$\|e_k\|_A^2 = (x_k - x^*)^T A (x_k - x^*) =$$
$$= x_k^T A x_k - 2 x_k^T \underbrace{A x^*}_{b} + x^{*T} A x^* =$$
$$= 2\phi(x_k) + (x^*)^T A x^*, \quad \text{where } \phi(x) = \tfrac{1}{2} x^T A x - x^T b$$

thus, if we minimalize $\phi(x)$, we minimalize also the error expressed using the A-norm $\|e\|_A$

- we search for a solution in the space $x_0 + K_k$ where

$$K_k = \mathcal{L}(p_0, p_1, \dots, p_{k-1}) = \underbrace{\mathcal{L}(r_0, Ar_0, \dots, A^{k-1} r_0)}_{\text{Krylov subspace } K_k(r_0, A)} = \mathcal{L}(Ae_0, \dots, A^k e_0)$$

or

$$x_k = x_0 + \alpha_0 p_0 + \dots + \alpha_{k-1} p_{k-1}$$

and by subtracting x^* we have

$$e_k = e_0 + \alpha_0 p_0 + \dots + \alpha_{k-1} p_{k-1} = e_0 + c_1 A e_0 + \dots + c_k A^k e_0$$

for certain coefficients $c_1, \dots, c_k$

- in other words, we can express the error after k steps using a matrix polynomial

$$e_k = P_k(A)\, e_0 \quad \text{where } P_k(z) = 1 + \sum_{i=1}^{k} c_i z^i$$

is a polynomial from $\mathcal{P}_k = \{P(z) \text{ of order } k, P(0) = 1\}$

- the CG method constructs directly this polynomial P_k, which minimalizes $\|e_k\|_A = \|P_k(A)\, e_0\|_A$ on $\mathcal{P}_k$

- to estimate this error, it is sufficient to find a polynomial in $\mathcal{P}_k$ for which we can estimate $P(A)$ because for any $\tilde{P}_k \in \mathcal{P}_k$ it is true that $\|e_k\| \le \|\tilde{P}_k(A)\, e_0\|_A$

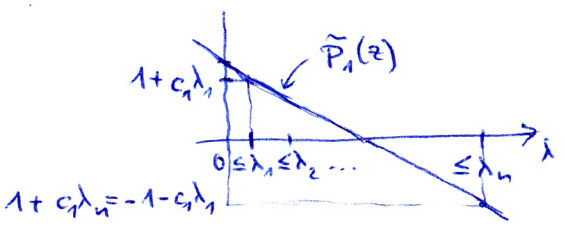
- as the matrix A is SPD (sym., positive def.), it is diagonalizable, i.e. $A = V \Lambda V^T$ with $\Lambda = \mathrm{diag}(\lambda_j)$ and λ_j, $j = 1, \dots, n$ are eigenvalues of A

and we set write $P(A) = V P(\Lambda) V^T$ for an arbitrary polynomial

giving
$$\|P(A)e_0\|_A^2 = e_0^T P(A)^T A P(A) e_0 =$$
$$= e_0^T V P(\Lambda)^T V^T A V P(\Lambda) V^T e_0 =$$
$$= e_0^T V \,\mathrm{diag}\left(\lambda_j P(\lambda_j)^2\right) V^T e_0 \le$$
$$\le \max_j P(\lambda_j)^2 \cdot e_0^T V \Lambda V^T e_0 = \max_j P(\lambda_j)^2 \|e_0\|_A^2$$

- to get an estimate we choose a polynomial $P(z)$ for which we easily estimate its values at eigenvalues λ_j (positive, real numbers)

for $k=1$:

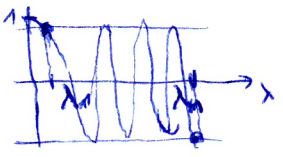


if we take $\tilde{P}_1(z) = 1 + c_1 z$ in such a way that it has maxima (in absolute value) at λ_1 and λ_n, i.e. $\tilde{P}_1(z) = 1 - \frac{2z}{\lambda_1 + \lambda_n}$

we get
$$\max_j |\tilde{P}_1(\lambda_j)| = 1 - \frac{2\lambda_1}{\lambda_1+\lambda_n} = \frac{\frac{\lambda_n}{\lambda_1} - 1}{\frac{\lambda_n}{\lambda_1} + 1} = \frac{\varkappa(A)-1}{\varkappa(A)+1} = 1 - \frac{2}{\varkappa(A)+1}$$

where $\varkappa(A)$ is the condition number for A

for general k we can use the Chebyshev polynomials of order k



$$\tilde{P}_k(z) = \frac{T_k\left(\frac{\lambda_n+\lambda_1-2z}{\lambda_n-\lambda_1}\right)}{T_k\left(\frac{\lambda_n+\lambda_1}{\lambda_n-\lambda_1}\right)}$$

← shifted and scaled to $\langle \lambda_1, \lambda_n \rangle$

← normalized to have $\tilde{P}_k(z=0) = 1$

it can be then shown (see e.g. Trefethen) that

$$\|e_k\|_A \le \max_j P(\lambda_j) \|e_0\|_A = \tilde{P}_k(\lambda_1) \|e_0\|_A =$$
$$= 2\left[\left(\frac{\sqrt{\varkappa}+1}{\sqrt{\varkappa}-1}\right)^k + \left(\frac{\sqrt{\varkappa}+1}{\sqrt{\varkappa}-1}\right)^{-k}\right]^{-1} \|e_0\|_A \le 2\left(\frac{\sqrt{\varkappa}-1}{\sqrt{\varkappa}+1}\right)^k \|e_0\|_A = 2\left(1 - \frac{2}{\sqrt{\varkappa}+1}\right)^k \|e_0\|_A$$

and thus to get a certain accuracy we need $\mathcal{O}(\sqrt{\varkappa})$ iterations
(Note that it is just an estimate, the method can converge faster!)

- we see that $\varkappa(A)$ can influence the speed of convergence of the CG method

therefore, it is advantageous to decrease the condition number by multiplying the system $Ax = b$ by an inverse of a certain matrix M (supposing that M^{-1} can be easily evaluated)

$$M^{-1}Ax = M^{-1}b$$

with the same solution but hopefully $\varkappa(M^{-1}A) \ll \varkappa(A)$

- but even if both A and M are SPD matrices, $M^{-1}A$ is not necessarily SPD

moreover $M^{-1}A$ can be much denser than A and M

therefore we apply A and M^{-1} separately

- later we will discuss possible choices of M, either general or suitable for a certain type of problems

- here we consider the case when $M^{-1}A$ is not SPD, then we can solve instead

$$\left(C^{-1}\right)^T A C^{-1}\right)(Cx) = (C^{-1})^T b \quad \Rightarrow \quad \tilde{A}\tilde{x} = \tilde{b}$$

for a certain non-singular C. Now $\tilde{A}$ is symmetric for an arbitrary C and also positive definite:

$$\tilde{A}^T = \left((C^{-1})^T A C^{-1}\right)^T = (C^{-1})^T A^T \left(C^{-1}\right)^T\right)^T = \tilde{A}$$

and $$v^T \tilde{A} v = v^T (C^{-1})^T A C^{-1} v = (C^{-1}v)^T A (C^{-1}v) > 0 \text{ for all } v \neq 0$$

- eigenvalues of $\tilde{A}$ are the same as of

$$C^{-1}\tilde{A}C = C^{-1}(C^{-1})^T A = (C^T C)^{-1} A$$

if we would have a reasonable M we could take $C^T C$ as its Cholesky decomposition

but it can be shown that such a decomposition is not necesary to do, instead, we can use the CG algorithm modified just a little bit using M as a preconditioner.

- algorithm of the CG method with preconditioning

$r_0 = b - Ax_0$

solve $Mz_0 = r_0$ to get z_0

$p_0 = z_0$

for $k = 1, 2, \ldots$

$w_{k-1} = Ap_{k-1}$

$\alpha_{k-1} = (z_{k-1}^T r_{k-1}) / (p_{k-1}^T w_{k-1})$

$x_k = x_{k-1} + \alpha_{k-1} p_{k-1}$

$r_k = r_{k-1} - \alpha_{k-1} w_{k-1}$

if $\|r_k\| < \varepsilon$ stop

otherwise

solve $Mz_k = r_k$ to get z_k

$\beta_{k-1} = (z_k^T r_k) / (z_{k-1}^T r_{k-1})$

$p_k = z_k + \beta_{k-1} p_{k-1}$

Modifications of conjugate gradient method

- CG method works for symmetric, positive definite (SPD) matrices
- generalization to hermitian, pos. def. matrices is straightforward
  - just replace all v^T by $v^\dagger$
- in principle, we could use CG to any system $Ax=b$
  if we multiply it by A^T (or $A^\dagger$)
  in other words, we solve normal equations
  $$A^TAx = A^Tb \quad (\text{or } A^\dagger Ax = A^\dagger b \text{ for complex matrices})$$
  because A^TA and $A^\dagger A$ are SPD matrices
  - the problem is that $\varkappa(A^TA) = \varkappa^2(A)$, thus for a matrix which is not well-conditioned we get even worse matrix $\Rightarrow$ it can be reasonably used only for well-conditioned matrices
  - this method is usually denoted CGNR (normal, residual) and A^TA is not constructed, instead A and A^T are applied subsequently to x
  - if there is a good preconditioner, it can be quite fast and stable method
- for complex, symmetric (not hermitian) matrices $A^T=A$
  one can use a variant called conjugate orthogonal CG = COCG
  in this method we replace all $x^\dagger y$ by symmetric product $x\cdot y$
  - but convergence is not guaranteed, but if converges it is much faster than other, more general methods
- for general matrices, the method of choice can also depend on available storage, if this is not a problem than one can use GMRES method (see later)
  if memory is limited and A is very sparse one can try methods as BiCG stab or GMRES(k) (version of GMRES which restarts after k steps)

solves $Ad = r_k = b - Ax_k$ $\downarrow \Rightarrow x \approx x_k + d$

  - discussion of BiCG and more stable and faster BiCG stab is beyond this course (see e.g. wikipedia and ref. there)

Arnoldi algorithm and GMRES method

- Arnoldi algorithm is used to ortogonalize a basis in Krylov subspace and the generalized minimal residual method (GMRES) is then used to find an optimal solution of $Ax=b$ on such subspace
  - as in the CG method, we search for a solution (approximative) of $Ax=b$ in the affine subspace (in the k-th iteration)
    $$x_0 + K_k = x_0 + \mathcal{L}(r_0, Ar_0, \dots, A^{k-1} r_0) \quad , r_0 = b - Ax_0$$
    using a method of least squares
  - Arnoldi algorithm does not construct K_k by multiplying r_0 by A, because $A^j r_0$ are for increasing j more and more linearly dependent (they converge usually to the eigenvector corresponding to the largest eigenvalue of A)
  - instead, in K_k we construct an orthonormalized basis $q_1, q_2, \dots, q_k$ in such away, that a new vector Aq_k (which should be almost $\perp$ to $A^{k-1} r_0$) is orthogonalized to all others
  - this process corresponds to subsequent transformation of A to the Hessenberg form:
    $$AQ_k = Q_{k+1} \tilde{H}_k$$
    ↙ matrix $(k+1) \times k$!
    or
    $$A \begin{pmatrix} | & & | \\ q_1 & \dots & q_k \\ | & & | \end{pmatrix} = \begin{pmatrix} | & & | \\ q_1 & \dots & q_{k+1} \\ | & & | \end{pmatrix} \begin{pmatrix} h_{11} & \dots & h_{1k} \\ h_{21} & h_{22} & h_{2k} \\ & \ddots & \vdots \\ & & h_{kk} \\ & & h_{k+1,k} \end{pmatrix}$$
    or recursively (relation for q_{k+1})
    $$v = Aq_k = h_{1k} q_1 + \dots + h_{kk} q_k + h_{k+1,k} q_{k+1}$$
    where $h_{ik} = q_i^T v$
    but for better stability we actually use modified Gram-Schmidt method

- algorithm (Arnoldi) , $r_0 = b - Ax_0$

$q_1 = r_0 / \|r_0\|_2$

for $k = 1, 2, 3 \ldots$

$v = Aq_k$

for $j = 1, \ldots, k$

$h_{jk} = q_j^t v$ } modified Gram-Schmidt

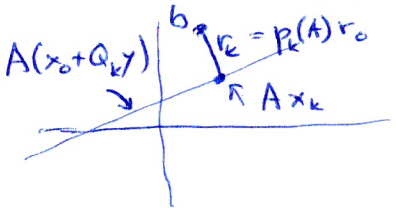
$v = v - h_{jk} q_j$ }

$h_{k+1,k} = \|v\|_2$ ← if $v = 0 \Rightarrow h_{k+1,k} = 0$ and we would have to choose q_{k+1} orthogonal to $\mathcal{L}(q_1, \ldots, q_k)$

$q_{k+1} = v / h_{k+1,k}$

• approximate solution in $x_0 + K_k$ - GMRES

- minimal residual method



we search for an approximate solution

$x_k = x_0 + Q_k y_k$ from all $x_0 + Q_k y$

↑ basis in K_k

to minimize $r_k = b - Ax_k$

- in other words, the norm

$$\|r_k\|_2 = \|Ax_k - b\|_2 = \|Ax_0 + AQ_k y_k - b\|_2 = \|AQ_k y_k - r_0\|_2$$

must be minimal.

- from Arnoldi we have $AQ_k = Q_{k+1} \tilde{H}_k$ and we get

$$\|r_k\|_2 = \|Q_{k+1} \tilde{H}_k y_k - r_0\|_2 = \|\tilde{H}_k y_k - Q_{k+1}^t r_0\| = \|\tilde{H}_k y_k - \|r_0\|_2 e_1\|_2$$

($Q_{k+1} \tilde{H}_k y_k - r_0 \in K_{k+1}$; the norm is not changed by multiplying vector from K_{k+1} by Q_{k+1}^t; $q_1 \sim r_0$)

- thus y_k is the solution of the overdetermined system

$\tilde{H}_k y = \|r_0\|_2 e_1$ ← we solve this in each iteration of Arnoldi algorithm and get $x_k = x_0 + Q_k y_k$

($\tilde{H}_k$: $(k+1) \times k$ matrix)

- this can be solved via the least-square method using QR factorization on $\tilde{H}_k$ matrix, with advantage we use here Givens rotations (see below) because $\tilde{H}_k$ is „almost" upper triangular (we have to make zero only one element under the diagonal)

Lanczos algorithm and MINRES for symmetric matrices

- if the matrix A is symmetric, $A^T = A$, then

$$h_{ij} = q_j^T A^T q_i = h_{ji}$$

and thus $h_{ij} = 0$ not only for $i > j+1$

but also for $j > i+1$ $\Rightarrow$ $\tilde{H}_k$ are <u>triagonal matrices</u>

now we get three-term recurrent relations

$$A q_k = h_{k-1,k}\, q_{k-1} + h_{kk}\, q_k + h_{k+1,k}\, q_{k+1}$$

in this case, Arnoldi algorithm simplifies to much more efficient <u>Lanczos algorithm</u>:

$r_0 = b - A x_0$
$\beta_0 = 0,\ q_0 = 0$
$q_1 = r_0 / \|r_0\|_2$
for $k = 1, 2, 3 \ldots$
  $v = A q_k$
  $\alpha_k = q_k^t v$
  $v = v - \beta_{k-1} q_{k-1} - \alpha_k q_k$
  $\beta_k = \|v\|_2$
  $q_{k+1} = v / \beta_k$

combined with minimal residual method described for GMRES we get so-called <u>MINRES method</u> with triagonal

$$\tilde{H}_k = \begin{pmatrix} \alpha_1 & \beta_1 & & 0 \\ \beta_1 & \alpha_2 & \beta_2 & \\ & \beta_2 & \ddots & \beta_{k-1} \\ 0 & & \beta_{k-1} & \alpha_k \\ & & & \beta_k \end{pmatrix}$$

again $(k+1) \times k$

- if A is SPD, it is better to use directly CG
but for indefinite symmetric matrices which are not well-conditioned this method can be a good option

<u>Givens rotations</u> - idea similar to Householder reflections

but now we rotate a vector $\begin{pmatrix} x \\ y \end{pmatrix}$ to get $\begin{pmatrix} \sqrt{x^2+y^2} \\ 0 \end{pmatrix}$ using

$$\cos\theta = \frac{x}{\sqrt{x^2+y^2}}, \quad \sin\theta = \frac{-y}{\sqrt{x^2+y^2}}$$

to nulify an element a_{ij} of a matrix A (i, j) we multiply it by

$$R(i,j,\theta) = \begin{pmatrix} 1 & & & & & 0 \\ & 1 & & & & \\ & & \cos\theta & & -\sin\theta & \\ & & & 1 & & \\ & & \sin\theta & & \cos\theta & \\ 0 & & & & & 1 \end{pmatrix} \begin{matrix} \\ \\ i \\ \\ j \\ \\ \end{matrix}$$

in each step we modify rows i and j of the current matrix setting finally R in QR factorization

it is slower then Householder method but great for Hessenberg matrices

Notes on preconditioning

- it is used in general for acceleration of convergence
  when instead of $Ax = b$
  we solve $\tilde{M}^{-1}Ax = \tilde{M}^{-1}b$ with suitable, simply invertible M

- for GMRES method, we can use it straightforwardly
  1) as q_1 we use the normalized vector $r_0 = \tilde{M}^{-1}(b - Ax_0)$
     (again by solving $Mr_0 = b - Ax_0$, $\tilde{M}^{-1}$ is not constructed)
  2) instead of $v = Aq_k$ we use $v = \tilde{M}^{-1}Aq_k$

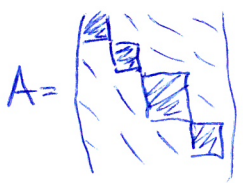
- for methods dealing with symmetric matrices,
  one has to be more careful as in the CG method

• general preconditioners (independent of the problem)

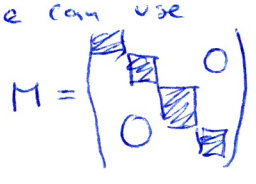
1) Jacobi preconditioner - $M = \mathrm{diag}(A)$
   - if $\tilde{M}^{-1}$ exists and we change $\varkappa$ sufficiently - $\varkappa(\tilde{M}^{-1}A) \ll \varkappa(A)$
     then it can work quite well (see Treffethen's example in Mathematica notebooks)

2) incomplete Cholesky or LU decomposition
   (IC) (ILU)
   - even for sparse matrices A, their Cholesky or LU decompositions can be much denser (more filled)
   - an idea of incomplete decompositions is simple
     - as M use $\tilde{R}^T\tilde{R}$ or $\tilde{L}\cdot\tilde{U}$
       where $\sim$ means that we keep non-zero elements only at positions, where the original A has non-zero elements
     - it turns out that for some matrices this works well
   - there is also version $ILU(k)$ when we store non-zero elements of A^k which can be less sparse than A

3) block Jacobi preconditioner
   - if A has clear dominant block-diagonal structure



we can use



where we invert each block separately (usually using a direct method - LU)

- problem-dependent preconditioners
  - in physics, we usually work with systems $Ax=b$ which originate from discretization of some PDE
    - as a suitable preconditioner, we can often use either coarser discretization, or some simplification of the problem (turning off some interactions etc.)
  - coarse grid
    - idea as for multigrid

      M^{-1} = Interpolation $\circ$ $\tilde{A}^{-1}_{coarser}$ $\circ$ Restriction

      here we solve $A_{coarser}\tilde{x}=\tilde{b}$
    - local or short-range approximation
      - if distant parts of a system under study interact weakly, then, if we organize parts properly, strong interactions can be localized around the diagonal of A and by turning off weak long-distance interactions, we can get a system which can be much more simpler to solve

        $\Rightarrow$ it can serve as a preconditioner
    - discretization with lower order approximations
      - if we need higher accuracy, we can use higher-order approximations for derivatives in PDE, but that leads to denser matrices
      - lower-order approximations lead to simpler matrices and can be again used as preconditioners

- for more techniques, see for example Trefethen, pp. 316-318 or an online book Templates for .... available at netlib.org etc.