Kratka ponovitev

Rešiti želimo sistem $Ax = b$.

Pri *Richardsonovi iteraciji* izberemo x_0 in generiramo zaporedje vektorjev

$$x_k = b + (I - A)x_{k-1} = x_{k-1} + r_{k-1},$$

kjer je $r_{k-1} = b - Ax_{k-1}$.

Velja $b - Ax_k = b - Ax_{k-1} - Ar_{k-1}$, torej $r_k = (I - A)r_{k-1}$. Od tod sledi

$$r_k = (I - A)^k r_0.$$

Konvergenco za poljuben x_0 imamo natanko tedaj, ko je $\rho(I - A) < 1$.

Opazimo tudi

$$r_k = P_k(A)r_0,$$

kjer je P_k polinom stopnje k z lastnostjo $P_k(0) = 1$.

V primeru nesingularne matrike A dobimo še $x - x_k = P_k(A)(x - x_0)$.

Za konvergenco je pomembno, kako dobro polinom P_k zaduši lastne vrednosti matrike A.

Prva posodobitev Richardsonove iteracije

Vpeljemo skalarje α_i in iteriramo

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

Dobimo

$$r_k = (I - \alpha_{k-1} A) r_k$$

in

$$P_k(A) = \prod_{j=0}^{k-1} (I - \alpha_j A).$$

Tako lahko pridemo do polinomov, ki bolje zadušijo lastne vrednosti. Za konverenco ni več nujno potrebno, da je $\rho(I - A) < 1$.

Če vemo, kje ležijo lastne vrednosti, si lahko pomagamo s polinomi Čebiševa in računamo nove približke preko tričlenskih rekurzivnih formul. To je *iteracija Čebiševa*.

Druga posodobitev - podprostor Krilova

Shranimo vse približke $x_0, x_1, \ldots, x_k$ in iz linearnih kombinacij pridemo do še boljših približkov. Za x_k iz Richardsonove iteracije velja $x_k \in x_0 + \mathrm{Lin}(r_0, Ar_0, \ldots, A^{k-1}r_0)$.

Definicija 1. *Za matriko A in vektor v definiramo podprostor Krilova $\mathcal{K}_k(A, v)$ kot*

$$\mathcal{K}_k(A, v) = \mathrm{Lin}(v, Av, \ldots, A^{k-1}v).$$

Približek iščemo v afinem podprostoru $x_0 + \mathcal{K}_k(A, r_0)$, ki ga generira $r_0 = b - Ax_0$.

Običajno na začetku vzamemo $x_0 = 0$, torej $r_0 = b$, in iščemo $x_k \in \mathcal{K}_k(A, b)$.

$x_k = x_0 + Q_{k-1}(A)r_0$ za nek polinom Q_{k-1} stopnje $k - 1$. Dobimo

$$r_k = b - Ax_k = (I - AQ_{k-1}(A))r_0 = P_k(A)r_0,$$

kjer je P_k polinom stopnje k z lastnostjo $P_k(0) = 1$.

S podprostori Krilova lahko pridemo do boljših približkov kot z Richardsonovo iteracijo, kjer je $r_k = (I - A)^k r_0$.

Kako dobimo približke iz podprostora Krilova

Rešujemo sistem $Ax = b$. Poznamo štiri pristope, kako metode podprostorov Krilova pridejo do približka x_k iz $x_0 + \mathcal{K}_k(A, r_0)$.

1. *Ritz-Galerkin:* Izberemo x_k, pri katerem je $b - Ax_k \perp \mathcal{K}_k(A, r_0)$.
   Metode: FOM, Lanczos, CG, GENCG.

2. *Minimalni ostanek:* Izberemo x_k, ki minimizira $\|b - Ax_k\|_2$.
   Metode: GMRES, MINRES, ORTHODIR.

3. *Petrov-Galerkin:* Izberemo x_k, pri katerem je $b - Ax_k \perp \mathcal{W}_k$.
   Metode: Bi-CG, QMR.

4. *Minimalna napaka:* Izberemo $x_k \in A^T\mathcal{K}_k(A^T, r_0)$, ki minimizira $\|x - x_k\|_2$.
   Metode: SYMMLQ, GMERR.

Baza za podprostor Krilova

Vektorji $r_0, Ar_0, \ldots, A^{k-1}r_0$ so nestabilna baza za $\mathcal{K}_k(A, r_0)$.

Naj bo $u_j = A^{j-1}r_0$. Če označimo $U_k = [u_1\ u_2\ \cdots\ u_k]$, velja

$$AU_k = U_k B_k + u_{k+1} e_k^T,$$

kjer je B_k matrika $k \times k$, ki ima na glavni podiagonali 1, drugje pa 0.

Če poznamo QR razcep $U_k = Q_k R_k$, kjer je $Q_k^T Q_k = I$ in R_k zgornja trikotna, sledi

$$AQ_k R_k = Q_k R_k B_k + u_{k+1} e_k^T,$$

od tod pa

$$AQ_k = Q_k R_k B_k R_k^{-1} + u_{k+1} e_k^T R_k^{-1} = Q_k \widetilde{H}_k + \frac{1}{r_{kk}} u_{k+1} e_k^T,$$

kjer je matrika $\widetilde{H}_k = R_k B_k R_k^{-1}$ zgornja Hessenbergova.

Dobimo $Q_k^T A Q_k = H_k$, kjer je H_k zgornja Hessenbergova in

$$AQ_k = Q_k H_k + h_{k+1,k} q_{k+1} e_k^T.$$

Kako izračunamo Q_k in H_k?

1.4 Arnoldijev algoritem (1951)

Po izreku o implicitnem Q so stolpci ortogonalne matrike Q, ki reducira matriko A v zgornjo Hessenbergovo matriko Q^TAQ, do predznaka določeni s prvim stolpcem.

Če dobimo matriko Q_k s prvim stolpcem $q_1 = \frac{1}{\|r_0\|} r_0$, da je $Q_k^T Q_k = I$ in $Q_k^T A Q_k$ zgornja Hessenbergova, potem po izreku o implicitnem Q stolpci Q_k tvorijo ortonormirano bazo za $\mathcal{K}_k(A, r_0)$.

Denimo, da stolpci $V_k = [v_1 \ \cdots \ v_k]$ sestavljajo ortonormirano bazo za $\mathcal{K}_k(A, r_0)$. Potem je $\mathcal{K}_{k+1}(A, r_0) = \mathrm{Lin}(v_1, \ldots, v_k, Av_k)$ in naslednji bazni vektor dobimo tako, da Av_k ortgonaliziramo glede na $v_1, \ldots, v_k$.

Arnoldijev algoritem za ortogonalizacijo uporablja modificirano Gram-Schmidtovo metodo.

$v_1 = r_0/\|r_0\|$
$j = 1, \ldots, k$
  $z = Av_j$
  $i = 1, \ldots, j$
    $h_{ij} = v_i^T z$
    $z = z - h_{ij} v_i$
  $h_{j+1,j} = \|z\|$
  če je $h_{j+1,j} = 0$, potem prekini računanje
  $v_{j+1} = z/h_{j+1,j}$

Arnoldijev algoritem

$v_1 = r_0/\|r_0\|$
$j = 1, \ldots, k$
  $z = Av_j$
  $i = 1, \ldots, j$
    $h_{ij} = v_i^T z$
    $z = z - h_{ij} v_i$
  $h_{j+1,j} = \|z\|$
  če je $h_{j+1,j} = 0$, potem prekini računanje
  $v_{j+1} = z/h_{j+1,j}$

Algoritem se konča pri izbranem k ali pa, ko je $h_{j+1,j} = 0$.

Trditev 1. *Arnoldijev algoritem se lahko izvaja do $j = k$, kjer je $k = \dim \mathcal{K}_n(A, r_0)$. Za $j = 1, \ldots, k$ velja*

$$AV_j = V_j H_j + h_{j+1,j} v_{j+1} e_j^T,$$

kjer je H_j zgornja Hessenbergova matrika $j \times j$, stolpci $V_j = [v_1 \ \cdots \ v_j]$ pa so ortonormirana baza za $\mathcal{K}_j(A, r_0)$.

Pišemo lahko tudi $AV_k = V_{k+1} H_{k+1,k}$.

Časovna zahtevnost je k MV(A) + $2k^2 n$ operacij. Prostorska zahtevnost je $(k+1)n$.

Izguba ortogonalnosti

Če želimo ohraniti numerično stabilnost in ortogonalnost baze, moramo po potrebi Gram-Schmidtov postopek ponoviti. To je t.i. RGS (repeated Gram-Schmidt) (1976).

$v_1 = r_0/\|r_0\|$
$j = 1, 2, \ldots, k$
  $z = Av_j, \;\; \xi = \|z\|$
  $i = 1, \ldots, j$
    $h_{ij} = v_i^T z$
    $z = z - h_{ij} v_i$
  če je $\|z\| < 0.7\xi$, potem
    $i = 1, \ldots, j$
      $\widetilde{h}_{ij} = v_i^T z$
      $h_{ij} = h_{ij} + \widetilde{h}_{ij}$
      $z = z - \widetilde{h}_{ij} v_i$
  $h_{j+1,j} = \|z\|$
  če je $h_{j+1,j} = 0$, potem prekini računanje
  $v_{j+1} = z/h_{j+1,j}$

Časovna zahtevnost je k MV(A) in $4k^2 n$ operacij. Prostorska zahtevnost je $(k+1)n$. Obstaja tudi varianta (Walker 1988), ki namesto MGS uporablja Householderjeva zrcaljenja.

Lanczosev algoritem (1950)

Če je A simetrična, se simetričnost pri Arnoldijevem algoritmu prenese na zgornjo Hessenbergovo matriko H, ki postane simetrična in tridiagonalna.

$v_1 = r_0/\|r_0\|$, $\beta_0 = 0$, $v_0 = 0$
$j = 1, 2, \ldots, k$
  $z = Av_j$
  $\alpha_j = v_j^T z$
  $z = z - \alpha_j v_j - \beta_{j-1} v_{j-1}$
  $\beta_j = \|z\|$
  če je $\beta_j = 0$, potem prekini računanje
  $v_{j+1} = z/\beta_j$

Dobimo $AV_k = V_k T_k + \beta_k v_{k+1} e_k^T = V_{k+1} T_{k+1,k}$, kjer je

$$T_{k+1,k} = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{k-1} \\ & & \beta_{k-1} & \alpha_k \\ & & & \beta_k \end{bmatrix}.$$

Časovna zahtevnost in prostorske zahteve so manjše kot pri Arnoldiju, a imamo večje težave z izgubo ortogonalnosti.

2. Metode, ki temeljijo na Ritz-Galerkinovem pristopu

Iščemo $x_k \in x_0 + \mathcal{K}_k(A, r_0)$, da je $r_k \perp \mathcal{K}_k(A, r_0)$. To je ekvivalentno

$$V_k^T(b - Ax_k) = 0.$$

Ker je $r_0 = \|r_0\|v_1$, sledi $V_k^T(b - Ax_0) = \|r_0\|e_1$.

Če je $x_k = x_0 + V_k y$, kjer je $y \in \mathbb{R}^k$, potem moramo rešiti sistem

$$V_k^T A V_k y = \|r_0\|e_1.$$

Matriko $V_k^T A V_k$ smo izračunali že med konstrukcijo ortonormirane baze podprostora Krilova. Ker je $V_k^T A V_k = H_k$, moramo rešiti sistem

$$H_k y = \|r_0\|e_1,$$

potem pa vzamemo $x_k = x_0 + V_k y$. To je metoda *FOM*.

Če je A simetrična, je H_k tridiagonalna in dobimo *Lanczosevo metodo*.

Če je A simetrična in pozitivno definitna, potem dobimo *metodo konjugiranih gradientov*.

2.1 FOM (full orthogonalization method)

Grob algoritem za metodo FOM je

$$
\begin{aligned}
&r_0 = b - Ax_0 \\
&\text{Arnoldi}(A, r_0, k) \Longrightarrow AV_k = V_k H_k + h_{k+1,k} v_{k+1} e_k^T \\
&y_k = H_k^{-1} \|r_0\| e_1 \\
&x_k = x_0 + V_k y_k
\end{aligned}
$$

Ustrezen k lahko določimo brez sprotnega računanja x_k. Velja namreč:

Izrek 1. *Za ostanek r_k, ki ga dobimo pri uporabi FOM, velja*

$$r_k = b - Ax_k = -h_{k+1,k} e_k^T y_k v_{k+1}$$

in zato

$$\|r_k\| = h_{k+1,k} |e_k^T y_k|.$$

Če se Arnoldijev algoritem konča predčasno, je $h_{k+1,k} = 0$. To je srečni dogodek, saj je potem $r_k = 0$ in x_k je točna rešitev.

Če je k prevelik, postane metoda neučinkovita, saj potrebuje veliko spomina za V_k, podraži pa se tudi ortogonalizacija novih vektorjev.

FOM s ponovnim zagonom

Ena od možnih rešitev je, da naredimo samo m korakov metode FOM, potem pa končni približek vzamemo za nov začetni približek.

Če naredimo naenkrat m korakov, to imenujemo FOM(m).

$$r_0 = b - Ax_0$$
$$\text{Arnoldi}(A, r_0, m) \Longrightarrow AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T$$
$$y_m = H_m^{-1} \|r_0\| e_1$$
$$x_m = x_0 + V_m y_m$$

če ni konvergence, vzemi $x_0 = x_m$ in se vrni na začetek

Možno je tudi, da med algoritmom spreminjamo vrednost parametra m.

2.2 IOM (incomplete orthogonalization method)

Alternativa FOM s ponovnim zagonom je, da pri računanju baze za $\mathcal{K}_k(A, r_0)$ novi vektor ortogonaliziramo le na nekaj zadnjih vektorjev. Tako sicer ne dobimo ortonormirane baze, a še vedno lahko pridemo do približka.

Nepopolni Arnoldi, kjer upoštevamo le zadnjih m vektorjev, je

$v_1 = r_0/\|r_0\|$
$j = \max(1, k-m+1), \ldots, k$
  $z = Av_j$
  $i = 1, \ldots, j$
    $h_{ij} = v_i^T z$
    $z = z - h_{ij} v_i$
  $h_{j+1,j} = \|z\|$
  če je $h_{j+1,j} = 0$, potem prekini računanje
  $v_{j+1} = z/h_{j+1,j}$

Za ortogonalizacijo potrebujemo v spominu le zadnjih m vektorjev. Matrika H_k, ki jo dobimo z nepopolnim Arnoldijem, ima pasovno obliko in je sestavljena iz $m+1$ diagonal.

Za izračun x_k po formuli $y_k = H_k^{-1}\|r_0\|e_1$, $x_k = x_0 + V_k y_k$, še vedno potrebujemo vse stolpce matrike V_k, a se temu da izogniti z razvojem rekurzivnih formul za x_k.

DIOM (direct incomplete orthogonalization method)

Denimo, da za pasovno Hessenbergovo matriko H_k iz IOM računamo osnovni LU razcep $H_k = L_k U_k$. Potem je L_k bidiagonalna, U_k pa pasovna zgornja trikotna z m diagonalami.

Za rešitev x_k potem velja

$$x_k = x_0 + V_k H_k^{-1} \|r_0\| e_1 = x_0 + V_k U_k^{-1} L_k^{-1} \|r_0\| e_1.$$

Če definiramo $P_k = V_k U_k^{-1}$ in $z_k = L_k^{-1} \|r_0\| e_1$, dobimo $x_k = x_0 + P_k z_k$.

Stolpce P_k in vektorje z_k lahko posodabljamo rekurzivno.

Iz $P_k U_k = V_k$ sledi $\sum_{i=k-m+1}^{k} u_{ik} p_i = v_k$, od tod pa dobimo

$$p_k = \frac{1}{u_{kk}} \left(v_k - \sum_{i=k-m+1}^{k-1} u_{ik} p_i \right) .$$

Vektor z_k je oblike $z_k = \begin{bmatrix} z_{k-1} \\ \zeta_k \end{bmatrix}$, kjer je $\zeta_k = -l_{k,k-1} \zeta_{k-1}$.

Ker je $x_{k-1} = x_0 + P_{k-1} z_{k-1}$ dobimo

$$x_k = x_{k-1} + \zeta_k p_k.$$

DIOM

$r_0 = b - Ax_0,\ \ v_1 = r_0/\|r_0\|$
$j = \max(1, k-m+1), \ldots, k$
$\quad z = Av_j$
$\quad i = 1, \ldots, j$
$\quad\quad h_{ij} = v_i^T z$
$\quad\quad z = z - h_{ij} v_i$
$\quad h_{j+1,j} = \|z\|$
posodobi LU razcep matrike H_j, če je $u_{jj} = 0$, končaj
$\zeta_j = -l_{j,j-1}\zeta_{j-1}$ *(na začetku* $\zeta_1 = \|r_0\|$*)*
$p_j = (1/u_{jj})\left(v_j - \sum_{i=j-m+1}^{j-1} u_{ij} p_i\right)$
$x_j = x_{j-1} + \zeta_j p_j.$
če je $h_{j+1,j} = 0$, potem prekini računanje
$v_{j+1} = z/h_{j+1,j}$

Za ostanek r_k, ki ga dobimo pri uporabi DIOM, še vedno velja

$$\|r_k\| = h_{k+1,k}|e_k^T y_k| = h_{k+1,k}\frac{|\zeta_k|}{|u_{kk}|}.$$

Če LU razcep brez pivotiranja ne obstaja, se algoritem kritično zaustavi pri $u_{jj} = 0$. Temu se izognemo z varianto, ki uporablja LU razcep z delnim pivotiranjem.

2.3 Lanczos

Če je A simetrična, za generiranje ON baze za $\mathcal{K}_k(A, r_0)$ uporabimo Lanczosevo metodo.

$v_1 = r_0/\|r_0\|$, $\beta_0 = 0$, $v_0 = 0$
$j = 1, 2, \ldots, k$
$z = Av_j$
$\alpha_j = v_j^T z$
$z = z - \alpha_j v_j - \beta_{j-1} v_{j-1}$
$\beta_j = \|z\|$
če je $\beta_j = 0$, prekini računanje
$v_{j+1} = z/\beta_j$

Dobimo

$$AV_k = V_k T_k + \beta_k v_{k+1} e_k^T,$$

kjer je

$$T_k = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{k-1} \\ & & \beta_{k-1} & \alpha_k \end{bmatrix}.$$

Za reševanje sistema $Ax = b$ uporabimo *Lanczosevo metodo za linearni sistem*.

$r_0 = b - Ax_0$
Lanczos$(A, r_0, k) \Longrightarrow AV_k = V_k T_k + \beta_k v_{k+1} e_k^T$
$y_k = T_k^{-1} \|r_0\| e_1$
$x_k = x_0 + V_k y_k$

Za ostanek velja $\|r_k\| = \beta_k |e_k^T y_k|$.

D-Lanczos

Podobno kot pri DIOM lahko preko LU razcepa matrike T_k pridemo do rekurzivnih formul za vektorje x_k in se tako izognemo temu, da bi morali na koncu za izračun $x_k = x_0 + V_k y_k$ poznati vse stolpce matrike V_k.

Ker je T_k tridiagonalna, ima LU razcep obliko $T_k = L_k U_k$, kjer je

$$T_k = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{k-1} \\ & & \beta_{k-1} & \alpha_k \end{bmatrix} = \begin{bmatrix} 1 & & & \\ \lambda_2 & 1 & & \\ & \ddots & \ddots & \\ & & \lambda_k & 1 \end{bmatrix} \begin{bmatrix} \eta_1 & \beta_1 & & \\ & \eta_2 & \ddots & \\ & & \ddots & \beta_{k-1} \\ & & & \eta_k \end{bmatrix}.$$

Za rešitev x_k velja $x_k = x_0 + V_k T_k^{-1} \|r_0\| e_1 = x_0 + V_k U_k^{-1} L_k^{-1} \|r_0\| e_1$.

Če definiramo $P_k = V_k U_k^{-1}$ in $z_k = L_k^{-1} \|r_0\| e_1$, dobimo $x_k = x_0 + P_k z_k$.

Iz $P_k U_k = V_k$ sledi $\beta_{k-1} p_{k-1} + \eta_k p_k = v_k$ oziroma $p_k = \frac{1}{\eta_k} \left(v_k - \beta_{k-1} p_{k-1} \right)$.

Vektor z_k je oblike $z_k = \begin{bmatrix} z_{k-1} \\ \zeta_k \end{bmatrix}$, kjer je $\zeta_k = -\lambda_k \zeta_{k-1}$.

Ker je $x_{k-1} = x_0 + P_{k-1} z_{k-1}$ dobimo $x_k = x_{k-1} + \zeta_k p_k$.

D-Lanczos

$v_1 = r_0/\|r_0\|$, $\beta_0 = 0$, $v_0 = 0$
$j = 1, 2, \ldots, k$
$z = Av_j$
$\alpha_j = v_j^T z$
$z = z - \alpha_j v_j - \beta_{j-1} v_{j-1}$
$\beta_j = \|z\|$
$\lambda_j = \beta_{j-1}/\eta_{j-1}$, (na začetku $\lambda_1 = 0$)
$\eta_j = \alpha_j - \beta_{j-1}\lambda_j$, če je $\eta_j = 0$, končaj
$\zeta_j = -\lambda_j \zeta_{j-1}$ *(na začetku $\zeta_1 = \|r_0\|$)*
$p_j = (1/\eta_j)\,(v_j - \beta_{j-1} p_{j-1})$
$x_j = x_{j-1} + \zeta_j p_j$.
če je $\beta_j = 0$, prekini računanje
$v_{j+1} = z/\beta_j$

Za ostanek r_k, ki ga dobimo pri uporabi D-Lanczosa, velja

$$\|r_k\| = \beta_k |e_k^T y_k| = \beta_k \frac{|\zeta_k|}{|\eta_k|}.$$

Če LU razcep brez pivotiranja ne obstaja, se algoritem kritično zaustavi pri $\eta_j = 0$. Temu se izognemo z varianto, ki uporablja LU razcep z delnim pivotiranjem ali pa QR razcep.