

# Iterative Methods for Linear Systems

---

# Generalities

---

- Find a solution  $x$  of  $A*x=b$ 
  - $A$  is square  $n \times n$  matrix (given)
  - $b$  is an  $n \times 1$  vector (given)
- Iterative methods generate a sequence of estimates  $x_0, x_1, x_2, \dots$  to  $x$
- Typically use the matrix  $A$  in simple operations:
  - multiply  $A$  times a vector:  $w = A*d$
  - multiply  $A^T$  times a vector:  $w_h = A^T*d_h$
  - solve a linear system with the lower or upper triangular part of  $A$

# Generalities

---

- **Stopping tests** become critical: how do you know when to quit iterating?
- Iterative methods still are a **black art**

# Generalities

---

- *Preconditioning* the linear system
  - Replace  $Ax=b$  with  $M^{-1}Ax=M^{-1}b$
  - If  $M$  is a good approximation to  $A$ , makes system look more like one with identity matrix  $I \approx M^{-1}A$  as coefficient matrix
  - Can accelerate convergence rate dramatically
  - Best preconditioners: application dependent
    - For PDEs, use a simpler PDE by discarding difficult terms, generate its coefficient matrix, use it as  $M$
    - Discard some physics in the problem that make  $A$  difficult to work with
    - Use known analytic properties of PDE or  $A$  to get  $M$
  - We may look at **generic** preconditioners
  - Iterative solvers **do not work in practice** without a preconditioner

# Preconditioning

---

- Mathematically replacing  $A$  with  $M^{-1} * A$  but in practice *never* form or store  $M^{-1}$
- When need matrix-vector product  $w = M^{-1} * A * d$  with  $M^{-1}A$ , carry out in two steps:
  - 1) Set  $t = A * d$
  - 2) Solve system  $Mw = t$  for  $w$
- Leads to three requirements for  $M$ :
  - 1) Should approximate  $A$ , so that  $M^{-1}A \sim I$
  - 2) Should be easy to solve systems  $Mw = t$
  - 3) Should not take too much additional storage

# Preconditioning

---

- General matrix preconditioners
  - **None** at all; equivalent to  $M = I$
  - **Diagonal**:  $M = \text{diag}(A)$
  - **ILU(0)**:  $M = LU$  factors of  $A$  with all fill-in entries discarded during factorization
  - **ILU(s)**:  $M = LU$  factors of  $A$  with first  $s$  levels of fill-in retained, other fill entries discarded
  - **MILU(0)**:  $M = LU$  factors of  $A$  with each fill-in entry added to the diagonal element in its row rather than discarding it
  - **Block Jacobi**: partition  $A$  into blocks, set  $M =$  LU factors of diagonal blocks of  $A$

# Stopping Tests

---

# Stopping Test: Maximum Iterations

---

- Anything can and will go wrong with iterative solvers, so plan for it
- First test: **maximum number of iterations**
  - Theory not helpful at all; estimates based on analytic bounds are rarely useful
  - Typically want *maxits* to depend on number  $n$  of unknowns, but not to be  $O(n)$
  - Common choice: *maxits* = *sqrt(n)*
  - “Really good iterative method will take only 10-20 steps” - Yousef Saad
  - There are no really good iterative methods for general linear systems

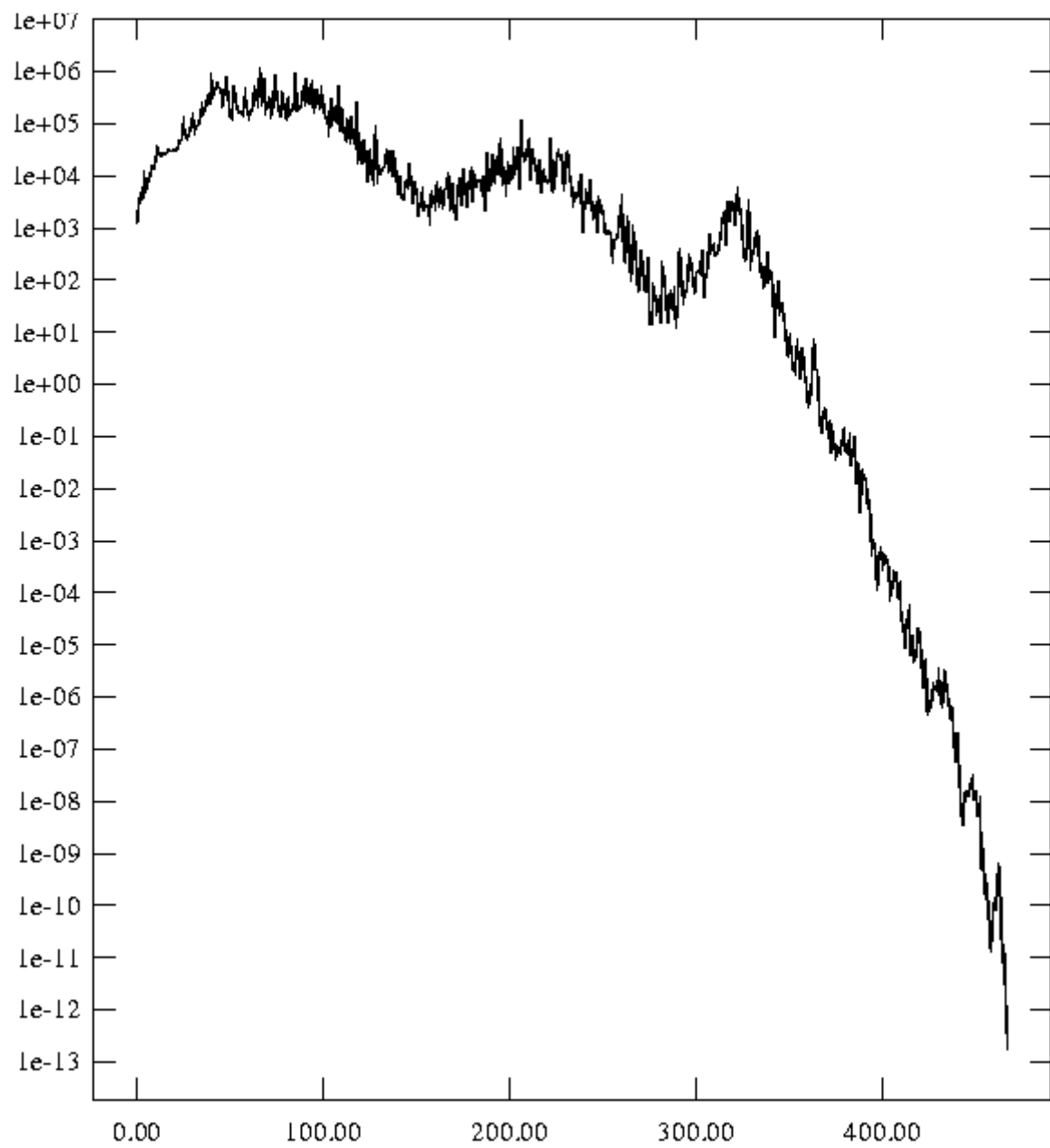
# Residual

---

- Want to know **error vector**:  $x_k - x^*$  where  $x^*$  is the true solution,  $x_k =$  current iterate
- If  $x^*$  is known, no point in iterating
- Instead, can know the **residual vector**

$$r_k = A^*(x_k - x^*) = Ax_k - b$$

- Because residual is  $A^*$ *error*, it can be much larger or smaller than error vector
- Typical plot: log of residual norm versus iteration number ...



# Stalling

---

- Track changes from iteration to iteration in  $x$  or residual vector  $r$
- If changes are small, quit
- If residual norm  $\|r\|$  increases, quit
- For *linear stationary methods*, small changes in  $x$  imply near stalling of method
- For Krylov subspace methods, size of change in  $x$  or  $r$  is not good basis for stopping
  - Last slide shows this

# Residual-based Tests

---

- One possible test:  $\|r_k\| \leq \text{tol}$ 
  - Problem: easy to solve any linear system by scaling. Compute  $r_0 = b - A \cdot x_0$ , and then scale system by  $s = \text{tol}/r_0$
  - The resulting scaled system has initial residual that satisfies stopping test:  $sAx = sb$  has residual
  - $sb - sAx_0 = s(b - Ax_0) = \text{tol}/r_0 \cdot (r_0) \leq \text{tol}$
  - Another problem:  $\text{tol}$  is often  $10^{-6}$  to  $10^{-8}$ , and if the linear system's data has scales around  $10^{12}$ , the test will never be satisfied

# Residual-based Tests

---

- Most commonly used test: *relative residual*  $\|r_k\| \leq \text{tol} * \|b\|$ 
  - Avoids scaling problems
  - Problem: can be too stringent when  $\|A\|$  or  $\|x^*\|$  are large compared to  $\|b\|$
  - Consider  $A = \begin{bmatrix} 1.0 & 1.0 + d \\ 1.0 & 1.0 \end{bmatrix}$   
and  $b = [-d \ 0.0]^T$  (here  $x^* = [1.0 \ -1.0]^T$ )
  - Multiplying  $\text{tol}$  by  $\|b\|$  makes it too small if  $d$  is small.

# Residual-based Tests

---

- Another relative residual test:

$$\|r_k\| \leq \text{tol} * \|r_0\|$$

- Depends on luck of guess for  $x_0$ .
- Would actually penalize computation (and make it iterate much further) if you were lucky enough to have a good starting estimate for  $x_0$ .

# Error-Norm Based Test

---

- Suggested by Oetli and Prager (1963)
- Based on backward error estimate

$$\|Ax-b\|_2 \leq \|A\|_1 * \|x\|_{\text{inf}} + \|b\|_2$$

- Test  $\|r_k\|_2 \leq \text{tol} * (\|A\|_1 * \|x_k\|_{\text{inf}} + \|b\|_2)$
- O&P showed that if residual norm satisfies this inequality, then you have exact solution to a “nearby” perturbed linear system

$$(A+dA)x = b+db$$

- Distance of original system to perturbed system is bounded by tol
- Requires computing  $\|x_k\|_{\text{inf}}$  on each iteration

# Stopping Tests

---

- If you choose just one stopping test, use Oetli-Prager one
- If collaborating with someone who is not, convince them to do so
- It identifies cases where linear system is **ill-conditioned** or even **singular**
- If  $\text{tol} * (\|A\|_1 * \|x_k\|_{\text{inf}} + \|b\|_2)$  is around machine precision, quit iterating - further work won't help
- If can't afford to compute  $\|x_k\|_{\text{inf}}$  each iteration, at least test Oetli-Prager condition when iterations end

# Preconditioned residual

---

- Recall most practical linear solvers are *preconditioned* so really have matrix-vector products that look like

$$w = M^{-1} * A * d$$

- So in practice have access to *preconditioned residual*

$$s_k = M^{-1} r_k = M^{-1} * (b - A * x_k)$$

- Same ideas hold for it but don't have access to  $\|M^{-1}A\|_1$  for Oetli-Prager test. Just use  $\|A\|_1$  instead
- Additionally: if  $\|s_0\| \gg \|r_0\|$  don't bother iterating. Compute a better preconditioner instead

# Stopping Test Summary

---

- Best to have at least three tests in an iterative solver:
  - 1) Maximum number of iterations reached
  - 2) Oetli-Prager condition met
  - 3) Initial preconditioned residual not vastly larger than unpreconditioned residual

# Linear Stationary Methods

---

- Numerical analysis classes cover Jacobi, Gauss-Seidel, SOR methods
- Derived by solving  $k^{\text{th}}$  equation for the  $k^{\text{th}}$  unknown, plugging that value back into  $x$ 
  - Jacobi: hold off updating  $x$  until a full sweep of  $k = 1:n$  has been made
  - Gauss-Seidel: update  $x$  as each  $x_k$  is found
  - SOR: variant of Gauss-Seidel
- Matrix terms:  $A = L+D+U$ 
  - $L$  = strictly lower triangular part of  $A$
  - $D$  = diagonal of  $A$
  - $U$  = strictly upper triangular part of  $A$
- $x_{k+1} = (D+L)^{-1}(b-Ux_k)$

# Linear Stationary Methods

---

# Linear Stationary Methods

---

- Convergence: slow or none at all
- Based on sweeping through rows of linear system, solving for  $k^{\text{th}}$  unknown on step  $k$
- Carry out full sweep, then update  $x$ :  
*Jacobi* algorithm
- Update entries of  $x$  as soon as they are found: *Gauss-Seidel*
- Look at  $d_k = x_{k+1} - x_k$  and consider it as update added to  $x_k$  (search direction).
  - If moving in that direction was good, moving a little further might be better:  $x_k = x_k + wd_k$ , with  $w > 1.0$
  - Gives *SOR*, successive overrelaxation method

# Linear Stationary Methods

---

- Whole industry grown up around linear stationary methods
  - Follow forward sweep through rows of  $A$  with a backward sweep: **SSOR**, symmetric SOR
  - Keep two previous search directions  $d_k$  and  $d_{k-1}$ , then update  $x_k = w_1 d_k + w_2 d_{k-1}$ : **Generalized SOR**
- In general, not competitive with Krylov subspace methods

# Krylov Methods

---

- *Krylov subspace methods* are invariably faster for problems in modern scientific and engineering computing
- Ur-version of Krylov methods is the **conjugate gradient algorithm**
  - Only applicable to **symmetric, positive definite** linear systems
  - Positive definite:  $v^T A v > 0$  for all nonzero  $v$ ;  $v^T A v$  is the curvature in direction  $v$  of the function  $f(x) = Ax$
  - SPD occurs when underlying problem satisfies a **minimum principle** of some kind (min energy, minimum stresses, etc)
  - Minimizing quadratic  $f(x) = c + b^T x - x^T A x$ : take derivative w/r to  $x$ , set to zero, get  $Ax = b$

# CG Algorithm (after initializing)

---

```
while (rho > tol and k <= maxits )
```

$$w = A*d$$

$$\text{temp} = w^T d$$

$$\alpha = \text{rho}/\text{temp}$$

$$x = x + \alpha*d$$

$$r = r - \alpha*w$$

$$\text{temp} = r^T r; \beta = \text{temp}/\text{rho}; \text{rho} = \text{temp}$$

$$d = r + \beta*d$$

$$k = k + 1$$

```
end while
```

# CG Algorithm

---

while (rho > tol and k <= maxits )

$$w = A*d$$

Matrix-vector product

$$\text{temp} = w^T d$$

$$\alpha = \text{rho} / \text{temp}$$

$$x = x + \alpha * d$$

$$r = r - \alpha * w$$

$$\text{temp} = r^T r; \beta = \text{temp} / \text{rho}; \text{rho} = \text{temp}$$

$$d = r + \beta * d$$

$$k = k + 1$$

end while

# CG Algorithm

---

while (rho > tol and k <= maxits )

$$w = A*d$$

dotproducts, vector updates

$$temp = w^T d$$

$$alpha = rho/temp$$

$$x = x + alpha*d$$

$$r = r - alpha*w$$

$$temp = r^T r; \beta = temp/rho; \rho = temp$$

$$d = r + \beta*d$$

$$k = k + 1$$

end while

# CG Algorithm

---

while (rho > tol and k <= maxits )

$$w = A*d$$

Scalar operations

$$\text{temp} = w^T d$$

$$\text{alpha} = \text{rho}/\text{temp}$$

$$x = x + \text{alpha}*d$$

$$r = r - \text{alpha}*w$$

$$\text{temp} = r^T r; \text{beta} = \text{temp}/\text{rho}; \text{rho} = \text{temp}$$

$$d = r + \text{beta}*d$$

$$k = k + 1$$

end while

# Eigenvalues

---

- Why does CG work? What is role of the matrix-vector products?
- Recall eigenvalues/eigenvectors
  - $Av = s*v$ ,  $v = \text{nonzero}$  vector,  $s = \text{scalar}$
  - If  $A$  is symmetric and pos def, then it has a “complete” set of eigenvectors  $[v_1, v_2, \dots, v_n]$
  - **Complete** means the set spans all of  $\mathbb{R}^n$
  - Can also choose  $v_i$ 's so that  $V = [v_1 \ v_2 \ \dots \ v_n]$  is an orthogonal matrix

# Eigenvalues

---

- If  $A$  is symmetric and pos def, all the eigenvalues  $s_i$  are positive and real
  - Can write eigenvalue/eigenvector equations as  $A = V^T D V$ , where  $V$  has the eigenvectors as its columns and  $D$  is diagonal matrix with eigenvalues on main diagonal
  - Called the **Schur decomposition**
  - If have Schur decomposition, it's easy to solve a linear system  $Ax=b$  but nobody does this; eigenvalues much more expensive and difficult to compute than LU factorization

# Eigenvalues

---

- *Power method* for finding largest eigenvalue/eigenvector of a spd matrix:
  - Keep multiplying an initial vector  $q$  by  $A$ , getting sequence of vectors  $Aq, A^2q, \dots$
  - Since  $A$  has complete set of eigenvectors,  $q = \text{sum}(w_i * v_i)$ ,  $w_i = \text{some scalars}$
  - $A * q = \text{sum}(w_i * s_i * v_i)$  since  $Av_i = s_i * v_i$
  - $A^2 * q = \text{sum}(w_i * s_i^2 * v_i)$
  - $A^3 * q = \text{sum}(w_i * s_i^3 * v_i)$

# Eigenvalues

---

- *Power method* continued:
  - Let  $s_i =$  largest eigenvalue
  - Divide through equation by  $s_i^k$
  - Means i-th component has coefficient of  $w_i * v_i$ , all others have  $w_k * s_k^k / s_i^k$
  - So all terms go to zero except i-th one
  - Without scaling by  $s_i^k$ , means the i-th component growing faster than all others

# Eigenvalues and Krylov Solvers

---

- So repeated multiplication by  $A$  implicitly **finds largest eigenvalue and eigenvector**
- Largest eigenvalue is max effect the linear transformation  $A$  can apply to a vector
- Largest eigenvector is direction that max effect occurs
- By repeatedly multiplying by  $A$ , Krylov solvers implicitly **find and remove largest eigenvector's contribution to the residual vector  $r_k$**

# Eigenvalues and Krylov Solvers

---

- Rest of a Krylov method computations assure that
  - residuals stay orthogonal to each other for numerical stability
  - once largest eigenvector has been removed, *second* largest is targeted
  - *group* of eigenvectors targeted simultaneously
- For non-symmetric matrices,  $A^T$  can have different eigenvectors than  $A$  does
  - Rarely see multiplication by  $A^T$  in an iterative method

# CG Algorithm for SPD Matrices

---

- **Conjugate gradient** algorithm derived around 1950
  - *conjugate* means some vectors are kept orthogonal in a certain sense:  $v_k^T A v_i = 0$  for  $i \neq k$
  - Define A-norm as  $\|v\|_A = \sqrt{v^T A v}$
  - *gradient* comes from treating problem as one of minimizing  $f(x) = \frac{1}{2} \|b - Ax\|_A$ ; the gradient of  $f$  is residual vector  $r$

# CG Algorithm

---

```
while (rho > tol and k <= maxits )  
    w = A*d  
    temp = wTd  
    alpha = rho/temp  
    x = x + alpha*d  
    r = r - alpha*w  
    temp = rTr; beta = temp/rho; rho = temp  
    d = r + beta*d  
    k = k + 1  
end while
```

# CG Algorithm for SPD Matrices

---

- Theoretical CG properties:
  - $r_k$  is orthogonal to all previous  $d_i$ 's
  - $r_k$  is orthogonal to all previous  $r_i$ 's
  - $d_k$  are “conjugate w/r to  $A$ ”:  $d_i^T A d_i = 0, i \neq j$
  - $\text{span}[d_1, d_2, \dots, d_k] = \text{span}[r_0, A r_0, A^2 r_0, \dots, A^{k-1} r_0]$
  - $r_k = p(A) * r_0$ , where  $p(A)$  is the optimal polynomial of degree  $k$  that minimizes  $e_k^T A e_k = (x^* - x_k)^T A (x^* - x_k)$
  - Short recursion: CG is a **three-term recursion** so only takes 3 extra vectors of storage

# CG Algorithm for SPD Matrices

---

- Theoretical convergence properties:
  - If  $A$  has only  $s$  distinct eigenvalues, then CG converges in  $s+1$  or fewer iterations
  - If  $c = \text{cond number of } A$ , then number of iterations is bounded by  $1 + 1/2\sqrt{c} \cdot \log(1/\text{tol})$ ,  $\text{tol} = \text{tolerance}$
  - “Kaniel-Paige theory”: if  $r_k = \sum(g_k^i * v^i)$  is the expansion of residual  $k$  in terms of the eigenvectors  $v^i$ , then  $g_k^1$  and  $g_k^n$  will go to zero faster than the other eigencomponents of the residual. [“CG kills off extremal eigenvalues first”]

# Krylov Methods for Unsymmetric Matrices

---

# (Bi)-Conjugate Gradients Stabilized

---

- **Initialize**

$$r = b - A*x$$

$$p = r$$

$$v = q = 0 \text{ (vectors, not scalars)}$$

$$\omega_h = \beta = \alpha = 1$$

# (Bi)-Conjugate Gradients Stabilized

---

- **while (stopping tests not satisfied)**

$$\beta_{h_n} = p^T r; \omega = (\beta_{h_n} * \omega_{h_n}) / (\beta_{h_n} * \alpha); \beta = \beta_{h_n}$$

$$q = r + \omega * (q - \alpha * v)$$

$$v = A * q$$

$$\tau = p^T v$$

$$\omega_{h_n} = \beta_{h_n} / \tau$$

$$s = r - \omega_{h_n} * v$$

$$t = A * x$$

$$\sigma = t^T s; \tau = t^T t$$

$$\alpha = \sigma / \tau$$

$$x = x + \omega_{h_n} * q + \alpha * s$$

$$r = s - \alpha * t$$

# (Bi)-Conjugate Gradients Stabilized

---

- while (stopping tests not satisfied)

$$\text{beta}_h = \mathbf{p}^T \mathbf{r}; \text{omega} = (\text{beta}_h * \text{omega}_h) / (\text{beta} * \text{alpha}); \text{beta} = \text{beta}_h$$

$$\mathbf{q} = \mathbf{r} + \text{omega} * (\mathbf{q} - \text{alpha} * \mathbf{v})$$

$$\mathbf{v} = \mathbf{A} * \mathbf{q}$$

$$\text{tau} = \mathbf{p}^T \mathbf{v}$$

$$\text{omega}_h = \text{beta}_h / \text{tau}$$

Dotproducts

$$\mathbf{s} = \mathbf{r} - \text{omega}_h * \mathbf{v}$$

$$\mathbf{t} = \mathbf{A} * \mathbf{x}$$

$$\text{sigma} = \mathbf{t}^T \mathbf{s}; \text{tau} = \mathbf{t}^T \mathbf{t}$$

$$\text{alpha} = \text{sigma} / \text{tau}$$

$$\mathbf{x} = \mathbf{x} + \text{omega}_h * \mathbf{q} + \text{alpha} * \mathbf{s}$$

$$\mathbf{r} = \mathbf{s} - \text{alpha} * \mathbf{t}$$

# (Bi)-Conjugate Gradients Stabilized

---

- while (stopping tests not satisfied)

$$\beta_{h_n} = p^T r; \omega = (\beta_{h_n} * \omega_{h_n}) / (\beta_{h_n} * \alpha); \beta = \beta_{h_n}$$

$$q = r + \omega * (q - \alpha * v)$$

$$v = A * q$$

$$\tau = p^T v$$

$$\omega_{h_n} = \beta_{h_n} / \tau$$

Vector updates

$$s = r - \omega_{h_n} * v$$

$$t = A * x$$

$$\sigma = t^T s; \tau = t^T t$$

$$\alpha = \sigma / \tau$$

$$x = x + \omega_{h_n} * q + \alpha * s$$

$$r = s - \alpha * t$$

# (Bi)-Conjugate Gradients Stabilized

---

- while (stopping tests not satisfied)

$$\beta_{h_n} = p^T r; \omega = (\beta_{h_n} * \omega_{h_n}) / (\beta_{h_n} * \alpha); \beta = \beta_{h_n}$$

$$q = r + \omega * (q - \alpha * v)$$

$$v = A * q$$

$$\tau = p^T v$$

$$\omega_{h_n} = \beta_{h_n} / \tau$$

Matrix-vector products

$$s = r - \omega_{h_n} * v$$

$$t = A * x$$

$$\sigma = t^T s; \tau = t^T t$$

$$\alpha = \sigma / \tau$$

$$x = x + \omega_{h_n} * q + \alpha * s$$

$$r = s - \alpha * t$$

# (Bi)-Conjugate Gradients Stabilized

---

- while (stopping tests not satisfied)

$$\beta_{h_n} = p^T r; \text{omega} = (\beta_{h_n} * \text{omega}_{h_n}) / (\beta_{h_n} * \alpha); \beta = \beta_{h_n}$$

$$q = r + \text{omega} * (q - \alpha * v)$$

$$v = A * q$$

$$\tau = p^T v$$

$$\text{omega}_{h_n} = \beta_{h_n} / \tau$$

Scalar updates

$$s = r - \text{omega}_{h_n} * v$$

$$t = A * x$$

$$\sigma = t^T s; \tau = t^T t$$

$$\alpha = \sigma / \tau$$

$$x = x + \text{omega}_{h_n} * q + \alpha * s$$

$$r = s - \alpha * t$$

# (Bi)-Conjugate Gradients Stabilized

---

- What could go wrong in BiCGStab?
  - Divide by near-zero value in scalar updates
  - Why didn't we worry about it in CG algorithm?

$$w = A*d$$

$$\text{temp} = w^T d$$

$$\text{alpha} = \text{rho}/\text{temp}$$

$$x = x + \text{alpha}*d$$

$$r = r - \text{alpha}*w$$

$$\text{temp} = r^T r; \text{beta} = \text{temp}/\text{rho}; \text{rho} = \text{temp} \dots$$

# (Bi)-Conjugate Gradients Stabilized

---

- What could go wrong in BiCGStab?
  - Divide by near-zero value in scalar updates
  - CG had

$$w = A*d$$

$$\text{temp} = w^T d \quad (\text{so } \text{temp} = w^T A w > 0)$$

$$\text{alpha} = \text{rho}/\text{temp}$$

$$x = x + \text{alpha}*d$$

$$r = r - \text{alpha}*w$$

$$\text{temp} = r^T r; \text{beta} = \text{temp}/\text{rho}; \text{rho} = \text{temp} \dots$$

# (Bi)-Conjugate Gradients Stabilized

---

- What could go wrong in BiCGStab?
  - Divide by near-zero value in scalar updates
  - CG had

$$w = A*d$$

$$\text{temp} = w^T d$$

$$\text{alpha} = \text{rho}/\text{temp}$$

$$x = x + \text{alpha}*d$$

*We want rho = 0!*

$$r = r - \text{alpha}*w$$

$$\text{temp} = r^T r; \text{beta} = \text{temp}/\text{rho}; \text{rho} = \text{temp} \dots$$