

## Krylov subspace methods: motivation

A family of methods particularly suitable for large, sparse matrices. They borrow some ideas from optimization, and combine them with (partial) factorizations.

**Idea 1** We can use optimization algorithms to solve linear systems. For instance, for a symmetric  $A \in \mathbb{R}^{m \times m}$ , the minimum of

$$f(x) = \frac{1}{2}x^T Ax - x^T b$$

will be in a point where  $0 = \nabla f(x) = Ax - b$ .

**Idea 2** Most variants of gradient descent produce iterates that lie all in the same subspace.

Start from  $x_0 = 0$  for simplicity. Gradient descent will give

$x_1 = x_0 + \alpha \nabla f(x_0)$   
 $x_1$  = a multiple of  $\nabla f(x_0) = -b$ ,

$x_2$  = a linear combination of  $x_1 = \alpha b$  and  $\nabla f(x_1) = Ax_1 - b$ ,

$x_3$  = a linear combination of  $x_1$ ,  $x_2$ , and  $\nabla f(x_2) = Ax_2 - b \dots$

$$X_0 = 0 \quad \underbrace{AX_0 - b = -b}$$

$$X_1 = X_0 + \alpha \underbrace{\nabla f(X_0)} = \text{multiple of } b$$

$$\underbrace{AX_1 - b} = \text{linear comb. of } b \text{ and } Ab$$

$$X_2 = X_1 + \alpha \underbrace{\nabla f(X_1)} = \text{lin. comb. of } b \text{ and } Ab$$

$$A \cdot (\delta_1 b + \delta_2 Ab) = \delta_1 Ab + \delta_2 A^2 b$$

$$X_3 = X_2 + \alpha \underbrace{\nabla f(X_2)} = \text{lin. comb. of } b, Ab, A^2 b$$

conjugate gradient: gives another lin. comb. of  $\boxed{b, Ab, A^2 b}$

## Krylov spaces

$$\left[ \begin{array}{l} x_1 \in \text{span}(b), \\ x_2 \in \text{span}(b, Ab), \\ x_3 \in \text{span}(b, Ab, A^2b), \\ x_4 \in \text{span}(b, Ab, A^2b, A^3b) \dots \end{array} \right.$$

Each new iterate is a linear combination of vectors in the previous space and  $A$  times them.

If  $v = \alpha_0 b + \alpha_1 Ab + \dots + \alpha_k A^k b$ , then  
 $Av = \alpha_0 Ab + \alpha_1 A^2 b + \dots + \alpha_k A^{k+1} b$

### Definition

$$K_n(A, b) = \text{span}(b, Ab, A^2b, \dots, A^{n-1}b).$$

$K_n(A, b)$  = everything you can get if you start from  $b$  and make linear combination & at most  $n-1$  products with the matrix  $A$

E.g.

all these belong to  $K_3(A, b)$

$$c = \alpha b$$

$$d = Ac = \alpha Ab$$

$$e = d + \beta b + \gamma c = (\alpha Ab) + \beta b + \gamma \alpha b = \alpha A b + \beta b + \gamma \alpha b$$

$$f = Ae + \delta d + \epsilon c = A(\alpha Ab + (\beta + \gamma \alpha)b) + \delta \alpha Ab + \epsilon \alpha b = \alpha \underline{A^2} b + (\beta + \gamma \alpha + \delta \alpha) \underline{A} b + \epsilon \alpha \underline{b}$$

# Using Krylov subspaces

$$A \in \mathbb{R}^{m \times m}$$

Idea First generate the whole  $K_n$ , then decide what vector to choose inside it. For instance: construct

$$V = \begin{bmatrix} \underline{b} & \underline{Ab} & \underline{A^2b} & \dots & \underline{A^{n-1}b} \end{bmatrix},$$

then look for the 'best' solution to  $A\underline{x} = b$  in  $\text{Im } V$ , e.g.,  $x = Vy$

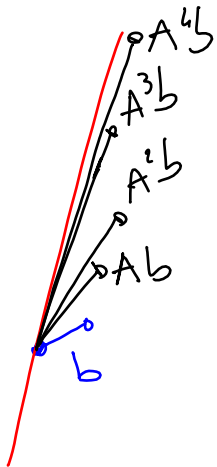
$$\min \|A(Vy) - b\|. \quad AV \in \mathbb{R}^{m \times n} \text{ (tall thin)}$$

The good: biggest cost: computing  $n$  products ( $v \mapsto Av$ )

We only need a function `compute_product(v) = A*v` (black box algorithm).

Fast whenever `compute_product` is fast: sparse  $A$ , but not only.

The bad Working with that  $V$  is problematic: its columns 'tend' to be aligned (cfr. power method). We need a better basis for  $K_n$ .



$$\begin{pmatrix} & x & & & \\ & & x & & \\ x & & & x & \\ & & & & x \\ & & x & & \end{pmatrix} b = c$$

$$\begin{array}{l} i_1 j_1 \quad A_{i_1 j_1} \quad \cancel{\sigma} \\ i_2 j_2 \quad A_{i_2 j_2} \quad \cancel{\sigma} \\ \vdots \\ i_k j_k \quad A_{i_k j_k} \end{array}$$

$$C_i = \sum_{j=1}^n A_{ij} b_j = A_{i1} b_1 + \cancel{A_{i2} b_2} + \dots + A_{in} b_n$$

I can drop off zero entries

$$= \sum_{A_{ij} \neq 0} A_{ij} b_j$$

$c_i = 0$   
for  $(i, j, A_{ij})$  in the set of nonzero elements

$$c_i = c_i + A_{ij} b_j$$

end

$$\text{cost: } O(\underbrace{\text{nnz}(A)})$$

↓  
number of nonzero entries of  $A$



## Properties of Krylov spaces

- ▶ It's a subspace: if  $v, w \in K_n(A, b)$ , then  $\alpha v + \beta w \in K_n(A, b)$ .
- ▶  $\dim K_n(A, b) \leq n$ .
- ▶  $v \in K_n(A, b)$  iff  $v = p(A)b$  for a polynomial  $p$  of degree  $< n$ .
- ▶ If  $v \in K_n(A, b)$ , then  $Av \in K_{n+1}(A, b)$ .
- ▶ If  $v = p(A)b$  for  $p$  of degree exactly  $n - 1$ , then  $Av = q(A)b$  for  $q$  of degree exactly  $n$ .

Assume for now that  $\dim K_n(A, b) = n$ , i.e.,  $b, Ab, \dots, A^{n-1}b$  are linearly independent.

## Arnoldi algorithm: the plan

Incremental algorithm to construct a matrix with **orthonormal columns** that spans  $K_n(A, b)$ : that is,  $Q_1$  in  $\text{qr}(V, 0)$ .

Given vectors such that

$$K_j(A, b) = \text{span}(q_1, q_2, \dots, q_j),$$

construct  $q_{j+1}$  such that

$$K_{j+1}(A, b) = \text{span}(q_1, q_2, \dots, q_j, q_{j+1}).$$

Additional invariant: the last vector  $q_j$  satisfies  $q_j = p(A)b$  with a  $p$  of degree **exactly**  $j - 1$ .

## Arnoldi algorithm: the iteration

**Step 1** Generate a vector in  $K_{j+1}$  (that was not already in  $K_j$ ):

$$w = Aq_j.$$

**Step 2** Determine  $q_{j+1}$ : we have

$$w = \beta_1 q_1 + \beta_2 q_2 + \cdots + \beta_j q_j + \beta_{j+1} q_{j+1}.$$

Since the basis is orthonormal, for all  $i$  we have

$$q_i^T w = \beta_1 (q_i^T q_1) + \beta_2 (q_i^T q_2) + \cdots + \beta_{j+1} (q_i^T q_{j+1}) = \beta_i.$$

Hence we can compute  $\beta_1 = q_1^T w, \beta_2 = q_2^T w, \dots, \beta_j = q_j^T w$ .

**Step 3** Subtracting,

$$z := \beta_{j+1} q_{j+1} = w - \beta_1 q_1 - \beta_2 q_2 - \cdots - \beta_j q_j.$$

$q_{j+1}$  must have norm 1: we set  $q_{j+1} = \frac{1}{\|z\|} z$  and  $\beta_{j+1} = \|z\|$ .

## Arnoldi algorithm: the code

**Input**  $b \in \mathbb{R}^m$ ,  $A \in \mathbb{R}^{m \times m}$  (possibly as 'anonymous function'  $v \mapsto Av$ ),  $n$ .

**Output** Orthonormal basis  $Q = [q_1, q_2, \dots, q_{n+1}]$  of  $K_{n+1}(A, b)$ .

```
function Q = arnoldi(A, b, n)
Q = zeros(length(b), n); %will be filled in
Q(:, 1) = b / norm(b);
for j = 1 : n
    w = A * Q(:, j);
    for i = 1:j
        % not what we showed earlier here, but stabler
        betai = Q(:, i)' * w;
        w = w - betai * Q(:, i);
    end
    nrm = norm(w);
    Q(:, j+1) = w / nrm;
end
```

## Arnoldi algorithm: the factorization

For  $j = 1, 2, \dots, n$ , we have written

$$Aq_j = \beta_{1,j}q_1 + \beta_{2,j}q_2 + \dots + \beta_{j,j}q_j + \beta_{j+1,j}q_{j+1} = Q \begin{bmatrix} \beta_{1,j} \\ \beta_{2,j} \\ \vdots \\ \beta_{j+1,j} \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Write them down one next to the other:

$$A \begin{bmatrix} q_1 & q_2 & \dots & q_n \end{bmatrix} = \begin{bmatrix} q_1 & q_2 & \dots & q_n & q_{n+1} \end{bmatrix} \begin{bmatrix} * & * & * & \dots & * \\ * & * & * & \dots & * \\ 0 & * & * & \dots & * \\ 0 & 0 & * & \dots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & * \end{bmatrix}$$

$$AQ_n = Q_{n+1} \underline{H}_n \text{ for some matrix } \underline{H}_n \in \mathbb{R}^{(n+1) \times n}$$

## The factorization: different forms

$$AQ_n = Q_{n+1}\underline{H}_n \text{ for some matrix } \underline{H}_n \in \mathbb{R}^{(n+1) \times n}$$

Here  $Q_n \in \mathbb{R}^{m \times n}$ ,  $Q_{n+1} \in \mathbb{R}^{m \times (n+1)}$ .

It's easy to modify the code to store the entries of  $\underline{H}_n$ .

**Variant** if you want the same (rectangular) matrix  $Q_n$  in both terms, you can write

$$AQ_n = Q_n H_n + q_{n+1} \beta_{n+1,n} e_n^T$$

with  $e_n^T = [0 \ 0 \ \dots \ 0 \ 1]$  and  $H_n \in \mathbb{R}^{n \times n}$  are the first  $n$  rows of  $\underline{H}_n$ .

(divide  $\underline{H}_n$  into blocks...)

**Remark**  $A \neq Q_{n+1} \underline{H}_n Q_n^T$  (rectangular matrices can't be inverted!)

## Exercises

1. Check that the orthogonalization loop (the one labelled 'not what we showed earlier, but stabler') computes the same quantities ( $\beta_j$  and  $z$ ) as the procedure that we have described earlier. (Note that the inner products are computed with a different vector  $w$  at each step!)  
Ignore computational errors.
2. Try Arnoldi on a large matrix ( $m \approx 1000$ ). Does the computed  $Q$  satisfy  $Q^T Q = I$  exactly? What is the residual  $\|Q^T Q - I\|$ ?
3. Modify the code for Arnoldi so that it computes the matrix  $\underline{H}_n$  as well. Compute the residual of  $AQ_n = Q_{n+1}\underline{H}_n$ .
4. Modify the code for Arnoldi so that the orthogonalization is done with the earlier formula, instead (hint: you can obtain this result with just a few matrix-vector products). Are the computed residuals better or worse?
5. Show that  $Q_n^T A Q_n = \underline{H}_n$ .