Beyond positive definite matrices

Recall: conjugate gradient solves linear systems $Q\mathbf{x} = \mathbf{y}$ with $Q \succ 0$.

Each step contains only a multiplication $Q\mathbf{d}_k$ and scalar products / linear combinations, so the algorithm is particularly efficient when Q is sparse.

Iterates $\mathbf{x}_j$ belong to successive Krylov spaces

$$K_j(Q, \mathbf{y}) = \text{span}(\mathbf{y}, Q\mathbf{y}, \ldots, Q^{j-1}\mathbf{y})$$

Convergence depends on polynomial approximation / clustering on the eigenvalues of Q.

Can we use the same ideas to solve general linear systems $A\mathbf{x} = \mathbf{y}$? Now $A \in \mathbb{R}^{m \times m}$ is square but no longer SPD.

Yes!

Using Krylov subspaces

Idea First generate the whole $K_n(A, \mathbf{y})$, then decide what vector to choose inside it. For instance: construct

$$V = \begin{bmatrix} \mathbf{y} & A\mathbf{y} & A^2\mathbf{y} & \dots & A^{n-1}\mathbf{y} \end{bmatrix},$$

then look for the 'best' solution to $A\mathbf{x} = \mathbf{y}$ in $\operatorname{Im} V$, e.g.,

$$\min_{\mathbf{z} \in \mathbb{R}^n} \|A(V\mathbf{z}) - \mathbf{b}\|.$$

The issue Working with that V is problematic: its columns tend to be aligned (see also: power method). We need a better basis for $K_n(A, \mathbf{y})$.

Taking Q from `[Q, R] = qr(V, 0)` won't do it: condition numbers tell us that the damage has already been done in forming V.

Arnoldi algorithm: the plan

Incremental algorithm to construct a matrix with orthonormal columns that spans $K_n(A, \mathbf{y})$: that is, Q_0 in `qr(V, 0)`.

Idea: if you remember the Gram–Schmidt algorithm from linear algebra, this is basically it.

Given vectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_j$ such that

$$K_j(A, b) = \text{span}(\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_j),$$

construct $\mathbf{q}_{j+1}$ (orthogonal to all of them) such that

$$K_{j+1}(A, b) = \text{span}(\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_j, \mathbf{q}_{j+1}).$$

Additional invariant: the last vector $\mathbf{q}_j$ satisfies $\mathbf{q}_j = p(A)\mathbf{y}$ 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):

$$\mathbf{w} = A\mathbf{q}_j$$

Since $\mathbf{q}_j$ has degree $j-1$, $A\mathbf{q}_j$ has degree j.

Step 2 Subtract $\mathbf{q}_1$ component:

$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{q}_1 h_1, \quad h_1 = \mathbf{q}_1^T \mathbf{w}.$$

This doesn't change the degree, and ensures that $\mathbf{q}_1^T \mathbf{w} = 0$.

Step 3 Repeat!

$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{q}_i h_i, \quad h_i = \mathbf{q}_i^T \mathbf{w}.$$

This ensures that $\mathbf{w}$ stays orthogonal to $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_{i-1}$, and becomes orthogonal to $\mathbf{q}_i$.

Step 4 After j steps, set $h_{j+1} = \|\mathbf{w}\|$, $\mathbf{q}_{j+1} = \mathbf{w}\frac{1}{h_{j+1}}$ to have a vector with $\|\mathbf{q}_{j+1}\| = 1$, $\mathbf{q}_i^T \mathbf{q}_j = 0$ for $i < j$.

Arnoldi algorithm: the code

Input $\mathbf{y} \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times m}$ (possibly as 'anonymous function' $\mathbf{v} \mapsto A\mathbf{v}$), number of steps n.
Output Orthonormal basis $Q = [\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_{n+1}]$ of $K_{n+1}(A, \mathbf{y})$.

```
function Q = arnoldi(A, y, n)
Q = zeros(length(b), n); %will be filled in
Q(:, 1) = y / norm(y);
for j = 1 : n
        w = A * Q(:, j);
        for i = 1:j
                betai = Q(:, i)' * w;
                w = w - Q(:, i) * betai;
        end
        nw = norm(w);
        Q(:, j+1) = w / nw;
end
```

Arnoldi algorithm: the factorization

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

$$A\mathbf{q}_j = \mathbf{q}_1 h_{1,j} + \mathbf{q}_2 h_{2,j} + \cdots + \mathbf{q}_j h_{j,j} + \mathbf{q}_{j+1} h_{j+1,j} = Q \begin{bmatrix} h_{1,j} \\ h_{2,j} \\ \vdots \\ h_{j+1,j} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Write these relations down one next to the other:

$$A \begin{bmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_n \end{bmatrix} = \begin{bmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_n & \mathbf{q}_{n+1} \end{bmatrix} \begin{bmatrix} * & * & * & \ldots & * \\ * & * & * & \ldots & * \\ 0 & * & * & \ldots & * \\ 0 & 0 & * & \ldots & * \\ & & \ddots & \ddots & \vdots \\ 0 & & \ddots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 & * \end{bmatrix}.$$

$AQ_n = Q_{n+1}\underline{H}_n$ for some matrix $\underline{H}_n \in \mathbb{R}^{(n+1)\times n}$ that contains the coefficients $h_{i,j} = (\underline{H}_n)_{ij}$.

The factorization: different forms

$AQ_n = Q_{n+1}\underline{H}_n$ 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 is 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

$$AQ_n = Q_n H_n + \mathbf{q}_{n+1} h_{n+1,n} \mathbf{e}_n^T$$

where $\mathbf{e}_n^T = \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \end{bmatrix}$ 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!

Breakdown

Arnoldi breaks down when $h_{j+1} = 0$, i.e., after orthogonalization $\mathbf{w} = \mathbf{0}$.
The vector $\mathbf{w}$ always has degree j, i.e., it is a linear combination $\mathbf{y}\alpha_0 + A\mathbf{y}\alpha_1 + \cdots + \alpha_j A^j\mathbf{y}$ with $\alpha_j \neq 0$. If this linear combination gives $\mathbf{0}$, then the vectors $\mathbf{y}, A\mathbf{y}, \ldots, A^j\mathbf{y}$ are not linearly independent.

Breakdown in Arnoldi happens when $\dim K_j(A, \mathbf{y}) < j$ (for the first time).

Book references

Trefethen-Bau, Lecture 33; Demmel, Section 6.6.1.

Word of warning Even if it is better than

```
>> [Q, R] = qr([v, A*v, A^2*v ...]),
```

Arnoldi is still not perfectly stable: if you check $\|Q_n^T Q_n - I\|$, it will slowly grow when the matrices are large (more about this in the exercises, if you are interested).

Thus, in practice Arnoldi-based algorithms often take a few more iterations to converge in practice than what theory predicts. When you need to work with large matrices ($m \gg 1000$), some stability trade-offs are needed.

Exercises

1. Check that the inner `for` loop in the Arnoldi algorithm is equivalent to

$$\begin{aligned} \mathbf{w} &= A\mathbf{q}_j, \\ h_i &= \mathbf{q}_i^T \mathbf{w}, \quad i = 1, 2, \ldots, j, \\ \mathbf{q}_{j+1} h_{j+1} &= \mathbf{w} - \mathbf{q}_1 h_1 - \mathbf{q}_2 h_2 - \cdots - \mathbf{q}_j h_j. \end{aligned}$$

   (The content of the variable $\mathbf{w}$ is different in the two variants!)

2. Implement both versions, and compare their stability.
   Theoretical note: The version in this slide is known as traditional Gram–Schmidt (GS), while the one we showed earlier is known as modified Gram–Schmidt (MGS). GS is more suitable to optimizations (parallelization, turning into block operations...), but also less stable. Intuitively, the reason is that there are other computations between when you compute h_i and when you subtract $\mathbf{q}_i h_i$, so numerical errors can creep in.

Exercises

1. Even with MGS, Arnoldi often suffers from loss of orthogonality. Try it on a large matrix ($m \approx 1000$). Does the computed Q_n satisfy $Q_n^T Q_n = I$ exactly? What is the residual $\|Q_n^T Q_n - I\|$?
2. Modify the code for Arnoldi so that the orthogonalization loop is run two times, one after the other (yes, just run the for loop twice). Has $\|Q_n^T Q_n - I\|$ improved? (This trick is called re-orthogonalization.)
3. Modify the code for Arnoldi so that it computes the matrix $\underline{H}_n$ as well. Compute the (relative!) residual of $AQ_n = Q_{n+1}\underline{H}_n$. Is it large, compared to that of $Q_n^T Q_n = I$?
4. Show that $Q_n^T A Q_n = H_n$.
5. Did we encounter already in this course other "black-box algorithms" that compute $v \mapsto Av$ for a certain matrix A? (Solution in ROT-13: (1) onpx-fhofgvghgvba sbe n gevnathyne flfgrz Gk=I, juvpu pbzchgrf vai(G)*I jvgubhg sbezvat G (2) fbyivat n yrnfg fdhnerf ceboyrzf, juvpu pbzchgrf cvai(N)*I jvgubhg sbezvat cvai(N).)