1 5th of December 2018 — F. Poloni

Do you recall?

Arnoldi factorization: we wanted to build an orthonormal base $\{q_1, q_2, \dots, q_n\}$ of the Krylov space $K_n(A, b) = \{b, Ab, A^2b, \dots, A^{n-1}b\}$. We showed that the matrix A could be more of less factorized as $AQ_n = Q_{n+1}H_n = Q_nH_n + q_{n+1}h_{n+1,n}{e_{n+1}}^T$. Moreover, we concluded that we could approximate the eigenvalues of the matrix A through the eigenvalues of matrix H_n, while the eigenvectors are obtained as Q_nv, where v is an eigenvector of the matrix H_n.

In the first part of the lecture we run some experiments on MatLab and we observed that the eigenvalues are approximated better and better starting from the boundaries. Notice that the approximation is not always good, but it is good enough if we take into account the complexity of the algorithm.

We are interested in explaining the convergence of Arnoldi method.

1.1 Convergence of Arnoldi

The eigenvalues of H_n are eigenvalues of a "nearby matrix" obtained by taking $\widetilde{H}_m$ (result of the full process) and replacing $\widetilde{H}_{n+1,n}$ with zero.

$$\widetilde{H}_m = \left[\begin{array}{cccc|cccc} * & \cdots & * & * & * & \cdots & \cdots & * \\ * & \cdots & * & * & * & \cdots & \cdots & * \\ 0 & \ddots & * & * & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & * & * & * & \cdots & \cdots & * \\ \hline & & & \textcircled{0} & * & \cdots & * & * \\ & & & & * & \cdots & * & * \\ & & & & 0 & \ddots & * & * \\ & & & & 0 & 0 & * & * \end{array}\right] \rightarrow H_m = \left[\begin{array}{cccc|cccc} * & \cdots & * & * & * & \cdots & \cdots & * \\ * & \cdots & * & * & * & \cdots & \cdots & * \\ 0 & \ddots & * & * & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & * & * & * & \cdots & \cdots & * \\ \hline & & & \textcircled{*} & * & \cdots & * & * \\ & & & & * & \cdots & * & * \\ & & & & 0 & \ddots & * & * \\ & & & & 0 & 0 & * & * \end{array}\right]$$

We expect that this change does not lead to a significative change in the eigenvalues, in other words, the eigenvalues of $\widetilde{H}_m$ differ from the eigenvalues of H_m by $|h_{n+1,n}|$. Formally, $\left\|\widetilde{H}_m - H_m\right\| = h_{n+1,n}$.

1.1.1 Better explanation

The space $K_n(A, b) = \text{span}(b, Ab, \dots, A^{n-1}b)$ contains the right "features" to represent the eigenvectors of A with largest eigenvalues: if $A = V\Lambda V^{-1}$ is diagonalizable, then

$$\begin{aligned} A^k b &= (V\Lambda V^{-1}) \cdots (V\Lambda V^{-1})b \\ &= V\Lambda^k V^{-1} b \\ &= \begin{pmatrix} V^1 & V^2 & \cdots & V^m \end{pmatrix} \cdot \begin{pmatrix} {\lambda_1}^n & & & \\ & {\lambda_2}^n & & \\ & & \ddots & \\ & & & {\lambda_m}^n \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix} \\ &= V^1 \lambda_1^k c_1 + V^2 \lambda_2^k c_2 + \cdots + V^m \lambda_m^k c_m \end{aligned} \tag{1.1}$$

where $c = V^{-1}b$.

$A^k b$ is a linear combination of the eigenvectors V^i in which those with largest $|\lambda_i|$ are "more prominent", in other words as n increases the components involving the largest $|\lambda_i|$s grow faster.

This also tells us that $\text{span}(V^1, V^2, \dots, V^m)$ (that are the eigenvectors associated to largest eigenvalues in modulus) "represent well" $K_m(A, b) = \text{span}(b, Ab, \dots, A^{m-1}b)$.

We cannot provide a formal proof of the convergence, because there are some counter examples.

Example 1.1. *Let* $A \in M(m, \mathbb{R})$*, such that* $A = \begin{pmatrix} 0 & & & & 1 \\ 1 & 0 & & & \\ & 1 & \ddots & & \\ & & \ddots & & \\ & & & 1 & 0 \end{pmatrix}$

In this case the eigenvalues are 0 *until the last iteration. In the last step the eigenvalues become correct.*

Notice that in this case the absolute values of the eigenvalues are the same, hence we are not able to do the trick explained above.

Observation 1.1 (Matlab syntax)**.** *The command* `[V, D] = eigs(A)` *does not work on sparse matrices* A*. In this case we may run Arnoldi method and use the best values obtained by Arnoldi:* `[V, D] = eigs(A, n)`*, which computes approximations of the top-n (largest in modulus) eigenvalues.*

Notice that the Matlab "implementation" (the quotes are because the Arnoldi method de facto is not implmented in Matlab) of the Arnoldi method uses some tricks to converge fast.

It is possible to use the command `eigs` *and pass to it a* λ*-function which computes the matrix-vector product and this is useful when the matrix and the vector have a particular shape.*

We would like to understand how to compute some eigenvalues that are not the biggest.

Lemma 1.1. *Let $A \in M(m, \mathbb{R})$ and let $(\lambda_1, \mathbf{v}_1), \dots, (\lambda_k, \mathbf{v}_k)$ be the eigenvalues/vectors of A. The following holds:*

1. *$(\lambda_i + \alpha, \mathbf{v}_i)$ are eigenvalues/vectors of $A + \alpha I$;*
2. *$(\frac{1}{\lambda_i}, \mathbf{v}_i)$ are eigenvalues/vectors of A^{-1};*
3. *$(\lambda_i^k, \mathbf{v}_i)$ are eigenvalues/vectors of A^k.*

Proof. Let us omit the subscript i to ease notation:

1. $(A + \alpha I)\mathbf{v} = A\mathbf{v} + \alpha\mathbf{v} = \lambda\mathbf{v} + \alpha\mathbf{v} = (\lambda + \alpha)\mathbf{v}$;
2. $(\lambda^{-1}\mathbf{v})$ is an eigenpair of A^{-1}. We need to check that $A^{-1}\mathbf{v} = \lambda^{-1}\mathbf{v}$. If we multiply by λA both sides: $\lambda \cancel{A A^{-1}}\mathbf{v} = \lambda A \lambda^{-1}\mathbf{v} \Leftrightarrow \lambda\mathbf{v} = \cancel{\lambda\lambda^{-1}} A\mathbf{v}$, which is true by definition of eigenvalue/vector of A;
3. (by induction) $A^2\mathbf{v} = A(A\mathbf{v}) = A\lambda\mathbf{v} = \lambda A v = \lambda\lambda v = \lambda^2 v$.

□

This lemma gives us the chance to use Arnoldi algorithm to compute the eigenvalues of a slightly modified matrix $B = (A - \mu I)^{-1}$. If (λ, v) is an eigenpair of A, then $(\lambda - \mu^{-1}, v)$ is an eigenpair of B.

When is that $(\lambda - \mu^{-1})$ is large? When λ and μ are close.

If A has eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_m$, then the eigenvalues of B are $\nu_1 = \frac{1}{\lambda_1 - \mu}, \nu_2 = \frac{1}{\lambda_2 - \mu}, \dots, \nu_m = \frac{1}{\lambda_m - \mu}$.

The problem is that such matrix B is not sparse when A is sparse.

We perform a trick to overcome this issue: we pass the function `eig` a λ-function that computes the product $Bz = A - \mu I^{-1} z$ without computing B explicitly. The idea is to use factorizations: $A - \mu I = LU$, then $Bz = U^{-1}(L^{-1}z)$, that we can compute by back-substitution.

```
% computes 5 eigenvalues closest to mu=2
» fl = eigs(A, 5, mu);
» fl = eigs(A, 5, 'SM'); % smallest magnitude
» fl = eigs(A, 5, 'LM'); % largest magnitude
```

Notice that `eigs(A, 6, 1)` needs to factorize $A - 1I = LU$, which might induce a lot of fill-in.

As final observation, the equivalents to Matlab's `eigs` function are `scipy.linalgs.eigs` for Python and `arpack` for C/C++ and Fortran.