Comprehension an implementation of the Lanczos algorithm for data analysis.

Author: Daniiar Volf (runsolar@mail.ru).

date: October 2019.

ver: 1.0

This article may be useful as educational material for data science engineers, data science programmers and other computer scientists.

Remark

On Oct 21-th 2013 the Center for Nuclear Study of University of Tokyo in Japan announced that a new code for nuclear shell-model calculations, “KSHELL”, is developed [1, 2]. It aims at carrying out both massively parallel computation and single-node computation in the same manner. The authors solved the Schrodinger equation in the M-scheme shell-model model space, utilizing Thick-Restart Lanczos method [3, 4]. During the Lanczos iteration, the whole Hamiltonian matrix elements are generated “onthe-fly” in every matrix-vector multiplication. The vectors of the Lanczos method are distributed and stored on memory of each parallel node. Also the authors reported that the newly developed code has high parallel efficiency on FX10 supercomputer and a PC with multi-cores [5].

Formulation of the problem

1. What a structure of the data we have?

$$A = (a_0\ a_1\ a_2\ ...\ a_{K-1}), (a_1\ a_2\ a_3\ ...\ a_K), (a_2\ a_3\ a_4\ ...\ a_{K+1}), ..., (a_{L-1}\ a_L\ a_{L+1}\ ...\ a_{N-1})$$

where ***A*** is the trajectory matrix of observations, which is a Hankel matrix.

2. What should be do?

As fast as possible obtain a pair of (***D***, ***V***),

where: ***D*** is a diagonal matrix composed of eigenvalues λ_i of the initial matrix ***A***;

V is a matrix composed of eigenvectors of the initial matrix ***A***.

3. Why it is problem?

There are no simple algorithms for directly calculating eigenvalues for matrices in the general case, but for many special classes of matrices, eigenvalues

can be calculated directly. Iterative algorithms solve the problem of calculating eigenvalues by constructing sequences that converge to eigenvalues (Table 1).

Table 1 - Most popular iterative methods

Method	Convergence	One step cost
Power method	linear	$O(n^2)$
Inverse Power Method	linear	
Rayleigh quotient iteration	cubic	
LOBPCG		
Bisection method	linear	
Laguerre iteration	cubic	
QR algorithm	cubic	$O(n^2)$
Jacobi Method	quadratic	$O(n^3)$
Divide and rule		$O(n^2)$
Homotopy method		$O(n^2)$
MRRR algorithm		$O(n^2)$

How can this problem be solved ?

Most often, sequences of eigenvalues are expressed through sequences of similar matrices that converge to a triangular or diagonal shape, which then allows our to simply obtain the eigenvalues. Sequences of eigenvectors are expressed through the corresponding similarity matrix. For example it may be Householder Transforms [6], Givens Rotations [7] or Arnoldi iteration [8]. As a result of such transformations, matrices of a special form are obtained, for example, the Hessenberg matrix [9].

In the chapter 13 of [10], Parlett describes the Lanczos algorithm for solving the (usually partial) eigenvalue problem for symmetric matrices. Theoretically (i.e. with the use of exact arithmetic), this method can bring a full symmetric matrix to a tridiagonal form when the number of Lanczos steps *m* is equal to *n* [10, 11]. The

problem can be solved by projecting the matrix ***A*** onto a certain *m*-dimensional Krylov subspace [10, 12] like

$$T_m = Q_m^{-1} A Q_m, m = 1, 2, ..., n$$

where: ***T*** is a tridiagonal matrix generated by the Lanczos algorithm;

Q is a matrix consisting of Lanczos vectors.

It shall be noted that for $m = n$, the properties of Hankel matrix ***A*** in the Krylov subspace make the matrix ***T*** symmetric, so the matrix ***T*** can be considered a symmetric tridiagonal matrix (see section 12.6 of [10]).

By solving the spectral problem for the tridiagonal matrix ***T***, we can obtain the eigenvalues of the initial matrix ***A***. By multiplying the matrix of Lanczos vectors (the vectors forming the basis of the Krylov subspace) ***Q*** by the eigenvectors of the tridiagonal matrix ***T***, we obtain the eigenvectors of ***A***. For $m < n$, instead of the exact spectrum of the initial matrix we obtain approximations of some eigenpairs. In other words, the Lanczos algorithm constructs a Rayleigh-Ritz approximation [10, 13] in the Krylov subspace, and, unlike the existing methods (for example, power method for approximating eigenvalues or method of Danilevsky), does not require the polynomials to be explicitly defined; they are defined instead by ternary recursive relationships (i.e. with no “induction on eigenvalue index”).

Considering the above remarks regarding the Lanczos algorithm, a function implementing this algorithm is shown by Listing 1 below (due to the fact that the emphasis of this material is designed for those who involved in the implementation of computational algorithms we will firstly study the implementation). This function is declared (it is chosen as a C language notation) as

```
void matrixTandMatrixQisResultOfLanczosTridiagonalization
        MatrixA(double *A, int RS, int CS, double *T, double *Q),
```

where: input parameter A is a pointer to the data array in memory containing the Hankel matrix;

input parameters RS and CS define the matrix dimensions;

output parameter T is a pointer to the data array in memory containing the symmetric tridiagonal matrix ***T***;

output parameter Q is a pointer to the data array in memory containing the matrix of Lanczos vectors ***Q***.

Listing 1 – a C language Implementation of the Lanczos algorithm

```
1.void matrixTandMatrixQisResult
                OfLanczosTridiagonalization
                MatrixA(double *A, int RS,
                      int CS, double *T,
                      double *Q) {
2.    double *z = (double *)malloc(RS*sizeof(double));
3.    int m=0;
4.    memset(z, 0, RS*sizeof(double));
5.    memset(T, 0, RS*CS*sizeof(double));
6.    memset(Q, 0, RS*CS*sizeof(double));
7.    Q[0]=1.0;
8.    for(int i=0; i<CS; i++) {
9.         vectorZisResultOfMultiplyingMatrixA
                        onVectorQ(A, RS, CS, Q+i, z);
10.        argumentTthisResultOfMultiplying
                VectorZonVectorQ(T+CS*i+i, Q+i, z, RS);
11.        if(i != CS-1) {
12.             for(int n=0; n<RS; n++) {
13.                  Q[n*CS+i+1]=(i==0)?z[n]-
                     T[CS*i+i]*Q[n*CS+i]:z[n]-
                     T[CS*i+i]*Q[n*CS+i]-
                     T[(i-1)*CS+i]*Q[n*CS+i-1];
14.             }
15.             double R=0;
16.             for(int n=0; n<m; n++) {
17.                  R=0;
18.                  for(int k=0; k<RS; k++) {
19.                       R=R+Q[k*CS+n]*Q[k*CS+i+1];
```

```
20.                 }
21.                 for(int k=0; k<RS; k++) {
22.                     Q[k*CS+i+1]=Q[k*CS+i+1]-
                        Q[k*CS+n]*R;
23.                 }
24.             }
25.             argumentTisResultOfNormalization
                        VectorQ(Q+i+1, CS, T+i*CS+i+1);
26.             for(int n=0; n<RS; n++)
27.                 Q[n*CS+i+1] = Q[n*CS+i+1]/T[i*CS+i+1];
28.             T[(i+1)*CS+i]=T[i*CS+i+1];
29.         }
30.         m++;
31.   }
32.   free(z);
33.}
```

For a better comprehension an implementation of the Lanczos algorithm let's explore the key expressions in Listing 1 :

Line 2 defines a pointer to a data array containing a vector of residuals [10];

Lines 6 and 7 generate the initial vector $q^{<0>}$ in the first column of the matrix ***Q***, (not orthogonal to the eigenvectors of the matrix ***A***);

Line 8 is the main recursion loop to generate the basis ***Q*** of the Krylov subspace and compute a projection of the matrix ***A*** onto that basis;

Line 9 generates the *i*-th element of the Krylov subspace

$$K_m\left(q^{<0>}, A = span\left\{q^{<0>}, Aq^{<0>}, A^2q^{<0>}, ..., A^{m-1}q^{<0>}\right\}\right),$$

where K_{m-1} is a linearly independent combination of $A^m q^{<0>}$. As the corresponding element of the orthonormal basis of the subspace is computed at every *i*-th step with an addition of a vector $Aq^{<i-1>}$ that is orthogonal to K_{j-1}, we implicitly generate the *i*-th element of the subspace in the loop on line 8 of the Listing 1 (i.e. $span\,(q^{<0>}, q^{<1>}, q^{<2>}, Aq^{<2>}, ..., Aq^{<i-1>})$). It shall also be noted that for

$m \to \infty : A^m q^{<0>}$ is an eigenvector corresponding to the eigenvalue with the maximum spectral radius;

Line 10 computes the diagonal element of the matrix ***T***;

Lines 9 and 10 contain two functions to calculate the diagonal element

$$\alpha_i = (q^{<i>})^T A q^{<i>},$$

where α_i is the *i*-th diagonal element of the matrix ***T*** (same as T[CS*i+i] in the Listing 1);

Line 12 is the loop computing the vector of residuals and storing the result in the (*i*+1)-th column of the matrix ***Q*** (the Lanczos method stores the Lanczos vectors in core, which allows fast access). In accordance with the section 12.6.1 of [10], in the main loop 8 (Listing 1) with *i*=[0,1,...], line 13 (Listing 1) calculates the following trinomial for *n*=[0,1,...]:

$$q_n^{<i+1>} = A q_n^{<i>} - \alpha_i q_n^{<i>} - \beta_{i-1} q_n^{<i-1>}, q^{<i>} \in Q,$$

where: $q^{<i>}$ is a vector in *i*-th column of the matrix ***Q*** ($q_n^{<i>}$ being the same as Q[n*CS+i] in Listing 1);

β_{i-1} is the (*i*-1)-th supradiagonal element of the matrix ***T*** (same as T[(i-1)*CS+i] in Listing 1), where

$$\beta_i = (q^{<i+1>})^T A q^{<i>}.$$

Line 16 is a loop to maintain orthogonality (reorthogonalize) the Lanczos vectors ***Q*** stored in the array Q (see section 13.7 in [10]). The algorithm may be accelerated by using selective orthogonalization, as discussed in the section 13.8 of [1];

Line 25 calls a function computing the supradiagonal element β_i of the matrix ***T*** as a magnitude of the vector of residuals (β_i being the same as T[i*CS+i+1] in Listing 1);

Line 27 computes (updates) the Lanczos vector stored in the (*i*+1)-th column of the matrix ***Q*** (Q[n*CS+i+1]).

To summarize, the programmatic implementation of the Lanczos algorithm in the Listing 1 performs orthogonalization via ternary recursion to determine the basis of the Krylov subspace ***Q*** and a symmetric tridiagonal projection of the Hankel matrix ***A*** into that subspace (***A***→***T***).

What about a parallelization?

Returning to the remark at the beginning, let's try to guess what can be parallelized in the Lanczos algorithm? Let's consider two steps of iterations of the program that is reviewed above (Listing 1). For example, consider a small symmetric matrix - 4x4. And let's carefully observe what will happen in the r/w memory.

Lanczos iteration 1:

1.1) **z=A** $\cdot\ Q^{<0>}$; a temporary vector **z** = (1, 0, 0, 0)

1.2) $T_{0,0}$ = a_0 = $(Q^{<0>})^T \cdot$ **z** ; calculating a_0

1.3) *i* = 0, 1, 2, 3 { $Q_i^{<1>} = z_i - T_{0,0} Q_i^{<0>}$ }; forming the vector of residual

1.4) $T_{1,0}$ = $T_{0,1}$ = b_0 = $\left\| Q^{<1>} \right\|$; calculating b_0

1.5) *i* = 0, 1, 2, 3 { $Q_i^{<1>} = Q_i^{<0>} / T_{0,1}$ }; forming the vector $Q^{<1>}$

The results after first Lanczos iteration you can see in the table 2.

Table 2 - Results of Lanczos iteration 1

In r/w memory

A, A=A^T

	0	**1**	**2**	**3**
0	$A_{0,0}$	$A_{0,1}$	$A_{0,2}$	$A_{0,3}$
1	$A_{1,0}$	$A_{1,1}$	$A_{1,2}$	$A_{1,3}$
2	$A_{2,0}$	$A_{2,1}$	$A_{2,2}$	$A_{2,3}$
3	$A_{3,0}$	$A_{3,1}$	$A_{3,2}$	$A_{3,3}$

Q

	0	**1**	**2**	**3**
0	1	$Q_{0,1}$		
1	0	$Q_{1,1}$		
2	0	$Q_{2,1}$		
3	0	$Q_{3,1}$		

T

	0	**1**	**2**	**3**
0	a_0	b_0		
1	b_0			
2				
3				

Lanczos iteration 2:

2.1) **z=A** $\cdot\ Q^{<1>}$;

2.2) $T_{1,1}$ = a_1 = $(Q^{<1>})^T \cdot$ **z** ;

2.3) *i* = 0, 1, 2, 3 { $Q_i^{<2>} = z_i - T_{1,1} Q_i^{<1>} - T_{0,1} Q_i^{<0>}$ };

2.4) $T_{1,2}$ = $T_{2,1}$ = b_1 = $\left\| Q^{<2>} \right\|$;

2.5) *i* = 0, 1, 2, 3 { $Q_i^{<2>} = Q_i^{<2>} / T_{1,2}$ }; forming the vector $Q^{<2>}$

The results after second Lanczos iteration you can see in the table 3.

Table 3 - Results of Lanczos iteration 2

In r/w memory

A, $A=A^T$

	0	**1**	**2**	**3**
0	$A_{0,0}$	$A_{0,1}$	$A_{0,2}$	$A_{0,3}$
1	$A_{1,0}$	$A_{1,1}$	$A_{1,2}$	$A_{1,3}$
2	$A_{2,0}$	$A_{2,1}$	$A_{2,2}$	$A_{2,3}$
3	$A_{3,0}$	$A_{3,1}$	$A_{3,2}$	$A_{3,3}$

Q

	0	**1**	**2**	**3**
0	1	$Q_{0,1}$	$Q_{0,2}$	
1	0	$Q_{1,1}$	$Q_{1,2}$	
2	0	$Q_{2,1}$	$Q_{2,2}$	
3	0	$Q_{3,1}$	$Q_{3,2}$	

T

	0	**1**	**2**	**3**
0	a_0	b_0		
1	b_0	a_1	b_1	
2		b_1		
3				

It can continue by induction, but the full result is obvious. In the last iteration it will have

$$T_{3,3} = a_3 = (Q^{<3>})^T \cdot \mathbf{z}\ ,\ \mathbf{z}=\mathbf{A}\cdot Q^{<3>}\ .$$

As you could have noticed in each iteration the steps 3 and 5 can be parallelized because you also could see that each *i*-th operation is independent.

Conclusion

The use of Lanczos algorithm for large matrices presents a significant advantage, as confirmed by Kaniel-Saad estimates for the spectral edge in a partial problem provided by Parlett in the section 12.4 of [10]. In accordance with [10], we shall also strive to maintain orthogonality of the Lanczos vectors in the programmatic implementation in order to minimize the hardware rounding errors, as they tend to lose orthogonality while converging. Thus, using the Lanczos algorithm provides for a noticeably faster determination of the singular values mining. Cause afterwards, for example, you can find the Ritz matrix pair, it may be a pair (***Y***, ***D***), using the QR algorithm based on Givens rotations [7, 14],

where: ***Y*** = ***QS*** is a matrix consisting of Ritz vectors;

S is a matrix consisting of eigenvectors of ***T***;

D is a diagonal matrix consisting of eigenvalues of ***T*** in the ascending order.

Also using the Givens rotation-based QR algorithm for a symmetric matrix ***T*** allows the data to be accumulated in the matrix ***S*** without using matrix multiplication operations explicitly.

References:

1. Shimizu N. Nuclear shell-model code for massive parallel computation, “KSHELL” // arXiv:1310.5431., 21 Oct., 2013.

2. Shimizu N., Utsuno Y. Microscopic description of nuclear level density in shell-model calculations // Proceedings of the 2017 Symposium on Nuclear Data, November 16-17, 2017 — pp. 33–38.

3. Wu K., Simon H. Thick-restart Lanczos method for symmetric eigenvalue problems // Tech. Rep. 41412, Lawrence Berkeley National Laboratory, 1998.

4. Wu K., Simon H. Thick-restart Lanczos method for large symmetric eigenvalue problems // Lawrence Berkeley National Laboratory/NERSC, Berkeley, CA 94720. — [accessed electronically].
URL: https://sdm.lbl.gov/~kewu/ps/trlan-siam.pdf (retrieved on: 17.10.2019).

5. Introduction of FX10 Supercomputer System // Supercomputing Division, Information Technology Center The University of Tokyo. — [accessed electronically]. URL: https://www.cc.u-tokyo.ac.jp/supercomputer/fx10/fx10_intro-e.html (retrieved on: 17.10.2019)

6. Alston S. Householder, Unitary Triangularization of a Nonsymmetric Matrix // Journal ACM, 5 (4), 1958 — pp. 339-342. DOI:10.1145/320941.32094

7. Bindel, D.; Demmel, J.; Kahan, W.; Marques, O. (2000), On Computing Givens rotations reliably and efficiently. LAPACK Working Note 148, University of Tennessee, UT-CS-00-449, January 31, 2001.

8. Arnoldi W. E. The principle of minimized iterations in the solution of the matrix eigenvalue problem // Quarterly of Applied Mathematics, volume 9, 1951 — pp. 17–29.

9. Learn more about Hessenberg Matrix // ScienceDirect. — [accessed electronically].

URL: https://www.sciencedirect.com/topics/engineering/hessenberg-matrix (retrieved on: 17.10.2019)

10. Parlett Beresford N. The Symmetric Eigenvalue Problem // Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1998. — 398 p.

11. Borcea L., Druskin V., Knizhnerman L. On the sensitivity of Lanczos recursions to the spectrum // Linear Algebra, v. 396, Appl., 2005. — pp. 103–125.

12. Knizhnerman L., Simoncini V. A new investigation of the extended Krylov subspace method for matrix function evaluations // Numerical Linear Algebra with Applic., v. 17, No. 4, 2010. — pp. 615–638.

13. Bhat R.B. Natural frequencies of rectangular plates using characteristic orthogonal polynomials in rayleigh-ritz method //Journal of Sound and Vibration Volume 102, Issue 4, 22 October 1985. — pp. 493–499.

14. Jolliffe I. Principal component analysis // Springer Series in Statistics, Springer, 2002. — 488 p.