

---

# Fast Krylov Methods for Clustering

---

Chao Yan\*

Department of Computer Science  
University of British Columbia  
Vancouver, BC, V6T 1Z4  
*chaoyan@cs.ubc.ca*

## Abstract

At the heart of unsupervised clustering and semi-supervised clustering is the calculation of matrix eigenvalues(eigenvectors) or matrix inversion. In generally, its complexity is  $O(N^3)$ . By using *Krylov Subspace Methods* and *Fast Methods*, we improve the performance to  $O(N\log N)$ . We also make a thorough evaluation of errors introduced by the fast algorithm.

## 1 Introduction

Data Clustering and graph segmentation are important operations for machine learning and computer vision. It includes both unsupervised and semi-supervised clustering. For unsupervised clustering, spectral method[4][5][6][3] has been the focus of considerable research. However, all these methods suffers from the slow computation of large matrix. Solving systems of linear equations  $Ax = b$  and computing eigenvalues and eigenvectors of large matrices  $Ax = \lambda x$  are two fundamental computation for clustering problem. They also have great practical uses in other machine learning problems. For example, for semi-supervised clustering, to label unlabeled points can be transformed into a problem of solving linear systems. And for the spectral clustering problem, it turns out to be a problem of computing the largest  $k$  eigenvectors of the weighted matrix  $W$ . Hence how to solve these two problems correctly and fast becomes very important.

### 1.1 Iterative methods and Krylov space

Iterative methods are dominant in computing large matrices because direct methods are either impossible or too slow hence infeasible in practice. First, there is no direct method for eigenvalue problems when dimension of the matrix is greater than 5. Any eigenvalue solvers must be iterative. On the other hand, direct methods for solving linear systems like Gaussian elimination require  $O(N^3)$  operations, which is too time-consuming. Iterative methods are approximated methods, which only require  $O(N^2)$  operations. They can compute solutions much faster with errors which can be tolerant. In practice, this is often good enough.

Krylov methods are one important types of iterative methods. These methods

---

\*student number: 20378048

	$Ax = \lambda x$	$Ax = b$
$A = A^T$	Lanczos	CG
$A \neq A^T$	Arnoldi	GMRES

Table 1: Krylov subspace methods

often attempt to generate better approximations from Krylov subspace. Given a matrix  $A$  and a vector  $b$ , the associated Krylov sequence is the set of vectors:  $b, Ab, A^2b, A^3b, \dots$ . The corresponding Krylov subspaces are the spaces spanned by successively larger groups of these vectors in the Krylov sequence. Krylov methods can be summarized in Table1:

Although these four methods work in different situations, they share some similar structures. They all tend to construct the orthogonal basis of Krylov subspace by multiplying matrix  $A$  with a vector. They all have one step which computes the multiplication of a matrix and a vector. See Figure1 and Figure2. The computation of  $y = Ax$  is the most time-consuming part of each iteration. It costs  $O(N^2)$  time. Suppose the number of iterations is  $k$ , the running time for Krylov methods totally is  $O(N^2k)$ .

<pre> <b>b</b> = arbitrary, <b>q</b><sub>1</sub> = <b>b</b>/<b>  b  </b> <b>for</b> <math>n = 1, 2, 3, \dots</math> <b>do</b>   <b>v</b> = <b>Aq</b><sub><math>n</math></sub>   <b>for</b> <math>j = 1</math> <b>to</b> <math>n</math> <b>do</b>     <math>h_{jn} = \mathbf{q}_j^T \mathbf{v}</math>     <b>v</b> = <b>v</b> - <math>h_{jn} \mathbf{q}_j</math>   <b>end for</b>   <math>h_{n+1,n} = \ \mathbf{v}\ </math>   <b>q</b><sub><math>n+1</math></sub> = <b>v</b>/<math>h_{n+1,n}</math> <b>end for</b> </pre>	<pre> <math>\beta_0 = 0, \mathbf{q}_0 = 0, \mathbf{b} = \text{arbitrary}, \mathbf{q}_1 = \mathbf{b}/\ \mathbf{b}\ </math> <b>for</b> <math>n = 1, 2, 3, \dots</math> <b>do</b>   <b>v</b> = <b>Aq</b><sub><math>n</math></sub>   <math>\alpha_n = \mathbf{q}_n^T \mathbf{v}</math>   <b>v</b> = <b>v</b> - <math>\beta_{n-1} \mathbf{q}_{n-1} - \alpha_n \mathbf{q}_n</math>   <math>\beta_n = \ \mathbf{v}\ </math>   <b>q</b><sub><math>n+1</math></sub> = <b>v</b>/<math>\beta_n</math>   Compute eigenvalues and eigenvectors of <b>H</b><sub><math>n</math></sub>. <b>end for</b> </pre>
--	---

Figure 1: Arnoldi Algorithm

Figure 2: Lanczos Algorithm

## 1.2 Fast Gauss method

The fast Gauss transform[2] introduced by Greengard and Strain is an important variant of more general fast multipole methods. It's for the Sum-Product problem within  $O(N \log N)$  operations, while the direct method requires  $O(N^2)$  operations.

The multiplication of some particular form matrix (Standard Gaussian Kernel Matrix) and a positive vector can be translated to the Sum-Product problem. For the standard Gaussian kernel matrix  $A$ ,  $A_{ij} = \exp(-\frac{\|x_i - y_j\|^2}{\sigma^2})$ ; therefore, the  $i^{th}$  element of  $A * w$  is the sum-product of the  $i^{th}$  row of  $A$  and the weight vector  $w$ .

The average running time for the fast method is  $O(N \log N)$ , in the worst case it's  $O(N^2)$  and  $O(N)$  for the best case. During the computation, the matrix  $A$  never needs to be constructed explicitly, hence space required can be reduced from  $O(N^2)$  to  $O(N)$ .

## 2 Fast Krylov methods

Krylov subspace methods provide an iterative algorithm running at  $O(N^2)$  due to direct multiplication of a matrix and a vector. Fast Gauss method allows this multiplication to be reduced to  $O(N \log N)$  for standard Gaussian kernel matrices. Hence it is natural to combine these two methods to solve linear systems and compute eigen problems for this particular type of matrices. As a result, we could reduce running time from  $O(N^3)$  to  $O(N \log N)$ . Because this type of matrices occur a lot in machine learning problems, it will be useful if we can combine these two algorithms and still solve original problems correctly. We already have a proof on their running time. However, because the Fast Gauss method will introduce a small error at each iteration, how fast the error will accumulate and how much the error will affect solutions is unknown. Hence it is important to do some experiments to see how error in the Fast Gauss method affects solutions obtained by Krylov methods.

We first implement four naive Krylov methods. Then we implement fast versions of each by replacing the matrix-vector multiplication ( $y = Ax$ ) with a function call to the fast Gauss method ( $y = \text{boxtree\_sum}(S, S, x, \text{sigma}, \text{eps})$ ). However, for the fast Gauss method, all entries in vector  $x$  must be non-negative, which is not true for Krylov iterative steps. To cope with this, instead of passing vector  $x$  directly, we add some constant  $q$  to all entries of vector  $x$  to ensure all entries are positive, then use the identity  $Ax = A(x + q) - Aq$ . Both  $A(x + q)$  and  $Aq$  can be computed by the fast method. Usually, we can set  $-q$  as the minimum element of  $x^1$ . However, by doing this, we might introduce more error into each step of iterations, which needs to be further investigated.

In order to see how error affects solutions, we compare Krylov methods with their corresponding fast versions, e.g., Arnoldi vs Fast Arnoldi. We compare the solutions generated by these algorithms with the true solution. Results are shown in the figures below. In each figure, the red dashed line represents the naive Krylov method and the blue solid line represents the corresponding fast version method.

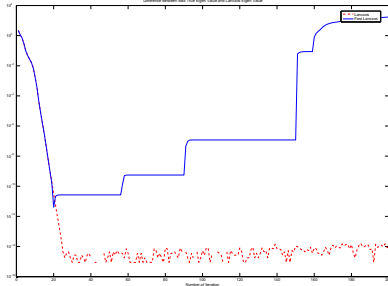


Figure 3: Eigenvalue of Lanczos

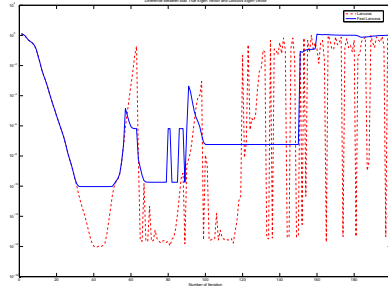


Figure 4: First Eigenvector of Lanczos

From Figure 3 and Figure 4, we see that for the fast Lanczos and Lanczos algorithms, they are almost the same in the first 20 iterations and the differences between partial solutions to the true solution drop below  $10^{-10}$ . After that, the Fast Lanczos stays for a while and then goes up, while the naive Lanczos's eigenvalue stays but its eigenvector becomes oscillating. This shows that Lanczos should be stopped around 20 iterations since we already obtain fairly good solutions. More iterations will cause solutions to become unstable.

<sup>1</sup>If we know some lower bound of elements of  $x$ , we can set  $-q$  as this lower bound. Then we don't need to compute  $Aq$  in each iteration.

Since naive Lanczos is not stable, we can say this is mainly due to Lanczos, not due to error of fast Gauss method.

For Arnoldi algorithms (See Figure 5 and Figure 6), both methods are almost the same in the first 20 iterations and differences between partial solutions and true solution drop below  $10^{-10}$ . After that, Fast Arnoldi stays around  $10^{-11}$  and naive Arnoldi stays around  $10^{-14}$ . This shows Arnoldi is very stable. Fast Arnoldi cannot drop further because the specified error  $\epsilon$  of fast Gauss method is  $10^{-10}$ . We believe if we allow  $\epsilon$  to be smaller, then Fast Arnoldi should be able to get a better solution. In practice, error rate of  $10^{-11}$  is already good enough.

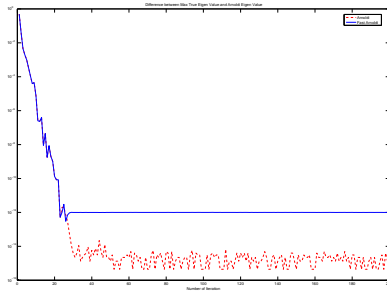


Figure 5: Eigenvalue of Arnoldi

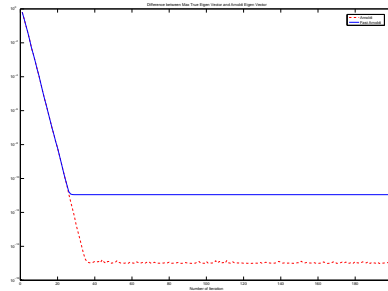


Figure 6: First Eigenvector of Arnoldi

In CG and GMRES, we almost cannot tell the difference between two methods. From Figure 7 and Figure 8, we see both curves are overlapped most of the time. After 200 iterations, CG stops at  $10^{-6}$  and GMRES stops at  $10^{-8}$ . This shows fast versions of both methods have very good performance.

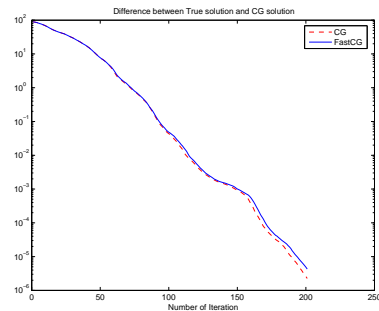


Figure 7: Solution of CG

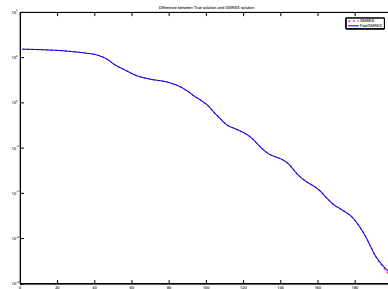


Figure 8: Solution of GMRES

As discussed above, the absolute error introduced by fast method is small. The dominant error comes from iteration approximation.

However we find that performance is very sensitive to kernel bandwidth  $h$ . In solving linear systems, solutions tend to have better performance in small  $h$ . For eigen problems, too big or too small  $h$  will both cause slow converges and big errors.

This is because  $h$  controls the condition of matrix  $A$ . We know condition number measures how far the matrix is from singular. Big condition number normally indicates matrix is close singular, which causes solution to be unstable and unreliable.

In figure9, when A is symmetric, the condition of A will exponentially increase with h. When h = 0.1, it is around  $10^3$ ; when h = 1, it jumps to  $10^{20}$ . When A is not symmetric, we observe similar results: when h = 0.1, it is already around  $10^6$ . This will explain why linear system solutions tend to have better performance on small h. On the other hand, let's consider when A is symmetric. Increasing h will cause A close to singular, but decreasing h will cause A close to identity matrix I, which has multiply max eigenvalues. Hence we will lose eigen gap. This explains why eigen problems will converge slow in very small h.

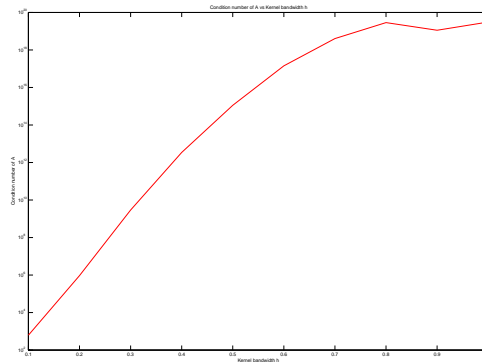


Figure 9: Condition Number of A vs Kernel bandwidth h

### 3 Applications

We can apply the *Fast Krylov Methods* to the matrix eigenvalue/eigenvector problem and matrix inversion problem. This makes the problems be solved in  $O(N \log(N))$  time. The idea can be used for both unsupervised clustering and semi-supervised clustering with some modifications.

#### 3.1 Unsupervised Clustering

Spectral clustering is an importance method for unsupervised clustering. In [1] Andrew et al proposed a spectral graph partitioning algorithm which use the first k eigenvectors of weight matrix  $L$ . We use Fast Krylov Methods to find them.

From the algorithm1 and algorithm2, we can see, for each iteration, the most time-consuming step is to calculate  $y = Lx$ . However,  $L = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$  is not a standard Gaussian kernel matrix, we can't use fast method directly.

- Rewrite  $y = Lx$  as  $y' = Ax'$  where  $y = D^{-\frac{1}{2}}y'$  and  $x' = D^{-\frac{1}{2}}x$ . First, we calculate  $x'$  using  $O(N)$  time because  $D$  is a diagonal matrix<sup>2</sup>; then we use fast method for  $y'$  using  $O(N \log(N))$  time, at last we get  $y$ .
- However, here  $A$  is not a standard gaussian kernel matrix. The diagonal value is 0 rather than 1. We define  $A = A' - E$ , then  $A$  is standard gaussian kernel matrix. We get  $y = Ax' = (A' - E)x' = A'x' - x'$ , where fast method can be used for  $A'x'$ .

<sup>2</sup>In fact, we use a vector to represents  $D$

- We don't need to construct  $D$  from  $A$ , it can be calculated from  $S$  using fast method.  $D$  is the sum of each row of  $A$ , that is,  $D = A * 1 = (A' - E) * 1 = A' * 1 - 1$ .
- As mentioned above, when  $x$  contains negative elements, we should add constant to make it positive:  $A'x = A'(x + q) - A'q$

Totally, we replace the step  $y = Lx$  of *Arnoldi* and *lanczos* algorithm with the codes:

```

ONE =boxtree_sum(S,S,ones,sigma,eps);
d= (ONE - 1)^(-1/2); //D=A'*1 - 1
x' = d*x; //x'
// y' = A'(x'+q) - A'q -x'
y' = boxtree_sum(S,S,x'+q,sigma,eps)-q*ONE-x';
y = d*y' //y

```

### 3.2 Semi-supervised Clustering

For the semi-supervised clustering with gaussian kernels, the solution is in terms of a linear system equations:  $y_u = (D_{uu} - W_{uu})^{-1}W_{ul}y_l$ . We can use *CG* or *GRMES* algorithm to find the inversion.

Similar with unsupervised clustering, we apply fast method to the most time-consuming step  $x = Ab$ . Unfortunately,  $A = D_{uu} - W_{uu}$  is not a standard gaussian kernel matrix, we use several tricks to solve it by fast methods.

- First,  $Ab = (D_{uu} - W_{uu})b = D_{uu}b - W_{uu}b$ .  $D_{uu}$  is a diagonal matrix,  $D_{uu}b$  uses  $O(N)$  time, while  $W$  is a standard gaussian kernel matrix, we use fast method for  $W_{uu}b$ .
- But  $D_{uu}$  can't be constructed from  $S_u$ . We should first calculate  $D$  from  $S$ , then  $D_{uu} = D(1 : u)$ . So for the fast *CG* or fast *GRMES* algorithm, the input is  $S$  rather than  $S_u$ .
- Similarly, we should make  $b$  be positive.  $x = Ab = (D_{uu} - W_{uu})b = D_{uu}b - W_{uu}(b + q) + W_{uu}q$ .

Therefore, we replace  $x = Ab$  of algorithm *CG* and *GRMES* with codes as:

```

D =boxtree_sum(S,S,ones,sigma,eps); //D = W*1
D_uu = D(1:u);
ONE = boxtree_sum(S_u,S_u,ones,sigma,eps) //W_uu*1
//x = D_uu*b - W_uu(b + q) + W_uuq
x = D_uu*b - boxtree_sum(S_u,S_u,b+q,sigma,eps)-q*ONE;

```

## 4 Experience

### 4.1 Unsupervised Clustering

We now turn to empirical evaluation of algorithms for unsupervised clustering problem. We use randomly generated cycles as example. Our fast algorithm is correct, see Figure10. We compare the performance of *matlab eig function*, *arnoldi*, *lanczos*, *fast arnoldi*, and *fast lanczos*. We also made a thorough evaluation of the effect of  $\sigma$  for each algorithm.

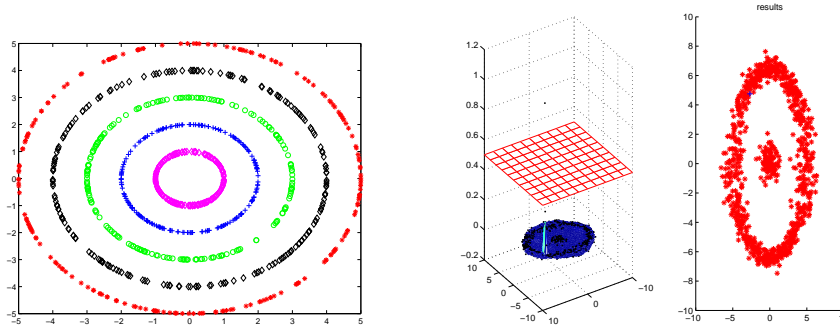


Figure 10: Result of Fast Method Clustering Figure 11: Result of Semi-supervised Clustering

From figure12, we see that the naive *eig* method uses about  $O(N^3)$  time and *arnoldi* or *lanczos* algorithm costs a little more than  $O(N^2)$  time. While the running time for *fast arnoldi* or *fast lanczos* method is between  $O(N\log(N))$  and  $O(N^2)$ . The running time can be approximate by  $0.5x^3 - 0.5x^2 + 9.5x - 13$ ,  $3x^2 - 10x + 12$ ,  $0.3x^2 + 6x - 7$  separately<sup>3</sup>. The fast Krylov algorithm uses more than  $O(N\log(N))$ , it may be caused by 1)  $O(N(\log(N)))$  is the average performance of fast method, however, here the points are special and it's difficult to be separate them; 2) it costs time to call *c* routine from matlab.

In addition, *fast lanczos(lanczos)* is little faster than *fast arnoldi(arnoldi)*<sup>4</sup>.

The performance of unsupervised clustering is improved from  $O(n^3)$  to a little more than  $O(N\log(N))$  by fast Krylov methods, especially when the data is large. We can run 2000 points example by *fast lanczos* methods in 80 seconds, however, in the same time, the naive method can only solve problem with about 1000 points.

We also notice *fast lanczos(lanczos)* is more sensitive of  $\sigma$  than *fast arnoldi(arnoldi)*. We find when  $\sigma$  is in the range of (0.1, 0.2), *fast arnoldi(arnoldi)* works. However, *fast lanczos(lanczos)* only works when  $\sigma$  is close to 0.15<sup>5</sup>. Therefore, we prefer *fast arnoldi*, it's robust with little performance harming. In some cases, it's even better than the naive method.

## 4.2 Semi-supervised Clustering

However, the fast Krylov method doesn't work for semi-supervised clustering. Usually it puts all points to one cluster, see Figure11.

Although *CG* or *GRMES* algorithm introduces some error, they work well for semi-supervised clustering by our test. Here, fast method introduces large error. We think the critical reason is that the matrix  $D_{uu} - W_{uu}$  is highly ill-condition (the condition number is about  $e^8$  while for the unsupervised learning, the condition number is only about  $e^2$ ). Therefore, the relative error is quite large although the absolute error is small. And error accumulates during the iterations.

<sup>3</sup> $x = \frac{\text{number of points}}{200}$

<sup>4</sup>Less than 1%

<sup>5</sup>Of course, with more points at each cycle, it's easier to separate them, so the range of  $\sigma$  is bigger. Here. we use the smallest range with 100 points of each cycle.

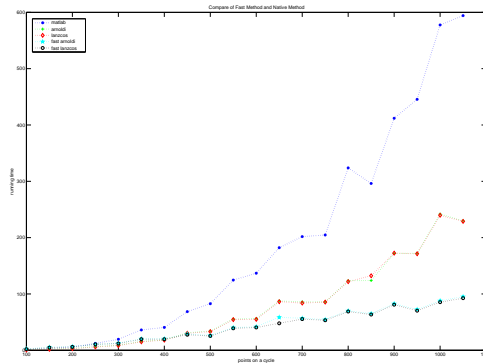


Figure 12: Runing Time for Each Method

## 5 Conclusion

Clustering is an importance problem for machine learning. We have applied *Krylov Subspace Method* and *Fast Method* to both unsupervised clustering and semi-supervised clustering. It improve the performance from  $O(N^3)$  to  $O(N\log(N))$ .

However, the fast methods are more sensitive with the parameter. And when the weight matrix is highly singular, it can't be used for semi-supervised learning.

Generally, Krylov Subspace methods can be applied to matrix eigen problem and solving of linear system, improving the performance to  $O(N^2)$ .Fast method is often used for problem like  $y = Ax$ , where A is standard gaussian kernel matrix or related matrix.

## Acknowledgments

Thanks Sam first, Sam and I have discussed and implemented the project together. Sam cared more about the analysis of *Fast Krylov Methods*, while I paid more attention to apply the idea to real application. This work is finished under the instruction of Nando de Freitas. Also we are grateful to Dustin Lang who provides us the source code of fast method.

## References

- [1] Andrew, Y.Ng. & Michael, I.J. & Yair W. (2002) On Spectral Clustering: Analysis and an algorithm. *NIPS 2002*.
- [2] Dustin L. (2004) Fast Methods for Inference in Graphical Models and Beat Tracking the Graphical Model Way. *Master Thesis of UBC*.
- [3] Charless, F. & Serge, B. & Fan C. & Jitendra M. (2004) Spectral Grouping Using the Nystrom Method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. pp. 214-225.
- [4] Marina, M. & Susan, S. & Liang X. (2004) Regularized spectral learning. *UW-Stat Dept TR # 465*.
- [5] Timothee, C. & Nicolas, G. & Jianbo S. (2005) Learning Spectral Graph Segmentation. *AISTATS 2005*
- [6] Francis, R.B. & Michael, I.J. (2003) Learning Spectral Clustering. *NIPS 2003*