Lanczos Vector Procedures

Travis Hoppe

Abstract:

This paper is intended as a graduate-level introduction to the theory and application of Lanczos procedures. A theoretical introduction will be developed for exact and finite precision matrices. The block symmetric method for real symmetric matrices will be introduced for their versatility in dealing with large, dense matrices that are common in many real-world engineering problems. A sample calculation is given to illustrate the Lanczos procedures in exact arithmetic. Sample code in C++ is also provided as a starting point for interested parties.

Introduction:

Cornelius Lanczos, born 1893, was a man of great mathematical talent often displaced by raging political climates. From changing his original name, Kornél Löwy, to avoid anti-German sentiment in Hungary, fleeing Hungary to avoid harsh anti-Jewish laws (to Germany, 1921), fleeing Germany to avoid the rise of anti-Semitism (to USA, 1931), fleeing the United States to avoid McCarthyism (1950), to finally arrive in Dublin Ireland[1]. It was while he was in the United States that Lanczos (*et al.)* developed the method that would eventually be known as the Fast Fourier Transform (FFT). The Lanczos vector procedures, when first introduced (1960), were met with harsh criticism due to the inevitable loss of orthogonality of the eigenvectors in finite precision. The orthogonality issue has been addressed in modern times, and the block representation has highly parallel properties for matrix calculations when the cost of I/O operations are computationally expensive.

Lanczos method in exact arithmetic

In general, we wish to solve

$$A\bar{x} = \lambda \bar{x}$$

and

$$\bar{y}^T A = \lambda \bar{y}^T$$

The presentation for exact arithmetic and finite precision will make no assumptions on A except that it is real. For block representation we will assume that A is symmetric as well. All methods can be extended to matrices that are unsymmetrical and complex. Readers may be unfamiliar with the second eigenvector equation. In general $\bar{x}$ and $\bar{y}$ are known as the right and left eigenvectors respectively. In most physical applications the eigenvector referred to is the right eigenvector, however the Lanczos procedure generates both[2]. The Lanczos method will generate two sets of biorthogonal vectors:

$$Y_n^T X_n = I \qquad Y_n^T A X_n = T_n$$

Where I is an *n*x*n* identity matrix, and T is a tridiagonal matrix.

We can manipulate by pre and post multiplications:

$$X_n Y_n^T A X_n = X_n T_n \qquad Y_n^T A X_n Y_n^T = T_n Y_n^T$$
$$A X_n = X_n T_n \qquad Y_n^T A = T_n Y_n^T$$

If we define the elements of T as:

$$T_n = \begin{bmatrix} \alpha_1 & \gamma_1 & & & & \\ \beta_1 & \alpha_2 & \gamma_2 & & & \\ & & \ldots & \ldots & \ldots & \\ & & & \beta_{n-2} & \alpha_{n-1} & \gamma_{n-1} \\ & & & & \beta_{n-1} & \alpha_n \end{bmatrix}$$

[1] B Gellai, Cornelius Lanczos: a biographical essay, in *Proceedings of the Cornelius*

[2] I have not come across code that speeds up when only the right eigenvector is needed.

Then by equating the columns of the previous equation we have:

$$Ax_k = \gamma_{k-1}x_{k-1} + \alpha_k x_k + \beta_k x_{k+1}$$
$$y_k^T A = \beta_{k-1}y_{k-1}^T + \alpha_k y_k^T + \gamma_k y_{k+1}^T$$

For all integer k < n, where the x and y are the kth columns of Y and X. Note that these are not the eigenvectors we are seeking (those were denoted with an overbar[3]). These x and y are known as the *Lanczos vectors*. Starting with some arbitrary initial conditions for the recurrence:

$$\gamma_0 x_0 = 0$$
$$\beta_0 y_0^T = 0$$

We can reorder to obtain, the 'famous[4]' three member recurrence:

$$\beta_k x_{k+1} = Ax_k - \gamma_{k-1}x_{k-1} - \alpha_k x_k$$
$$\gamma_k y_{k+1}^T = y_k^T A - \beta_{k-1}y_{k-1}^T - \alpha_k y_k^T$$

This expression is not unique, in fact it holds for any linear combination such that:

$$\beta_k \gamma_k = \bar{y}_{k+1}^T \bar{x}_{k+1}$$

A choice must be made then, to satisfy this condition. The particular choice:

$$\delta_k = \bar{y}_{k+1}^T \bar{x}_{k+1}$$
$$\beta_k = \sqrt{|\delta_k|}$$
$$\gamma_k = \beta_k sign(\delta_k),$$

not only satisfies the condition, but makes the tridiagonal matrix, T symmetric (with exception to the sign of γ). We can see that in exact arithmetic, the Lanczos vectors are orthonormal, i.e.

$$y_k^T x_k = 1$$
$$y_k^T x_j = 0 \qquad y_j^T x_k = 0$$

For all j < k. We can now solve for the final element in T, by using this condition and taking the three member recurrence relation to obtain:

$$\alpha_k = y_k^T A x_k$$

Our construction of T, which is a transformation of our original matrix A, leaves the eigenvalues invariant[5]. These eigenvalues, which are referred to as the Ritz values, are approximations to the eigenvalues of A (the designation seems superfluous, but in finite precision the two are *not* equal). The eigenvectors of A can be found from the Ritz vectors by:

$$\bar{x}_i = X_n u_i \qquad \bar{y}_i = Y_n v_i$$

Where X and Y are the matrices containing the Lanczos vectors, and u and v are the eigenvectors of T. These can be found rather quickly since T is almost a symmetric tridiagonal matrix (the exception being the minus sign). Various numerical packages have packages for tridiagonal matrices i.e. Numerical Recipes, *et al.* The most popular method seems to be a QR decomposition in finding the eigenvectors, whose convergence is guaranteed to be quick. The procedure breaks down with multiple eigenvalues but significant extensions have been made to handle these special cases. It is the loss of orthogonality in finite precision that presents a far more serious problem.

Lanczos method with finite precision; loss of orthogonality

In each step of the Lanczos procedure, there is an inherent round-off error in the step:

[3] The relation between the Lanczos vectors and the eigenvectors are $\bar{x}_{k+1} = \beta_k x_{k+1}$ and $\bar{x}_{k+1} = \beta_k x_{k+1}$

[4] Every book that introduces the recurrence calls it famous, so in deference to it's popularity I stick with convention.

[5] I was unable to find a proof of this, every treatment simply assumed it was so

$$\beta_k x_{k+1} = Ax_k - \gamma_{k-1} x_{k-1} - \alpha_k x_k + f_x$$
$$\gamma_k y_{k+1}^T = y_k^T A - \beta_{k-1} y_{k-1}^T - \alpha_k y_k^T + f_y$$

These errors accumulate and influence the δ_k term which will approach zero. When this happens the Lanczos procedure terminates and cannot be continued. The local orthogonality can be measured by the following metric:

$$\varepsilon_x = \frac{| x_{k+1}^T x_k |}{|| x_{k+1} || \, || x_k ||}$$

And likewise for the y vectors. The most drastic procedure, recommended by Lanczos, was to orthogonalize each new vector to the previous ones using the Gram-Schmidt process. This is a computationally expensive operation, and one that initially lead to the rejection of the Lanczos procedures. In more recent times, it has been realized that a full orthogonalization is not necessary, rather an adapted partial convergence algorithm is sufficient.

Symmetric Block Lanczos[6]:
The symmetric block method is where the real power of the Lanczos procedure shines. We will construct blocks of Lanczos vectors of size *b.* Once we have chosen a block size, it is only necessarily to access the matrix A for each block, rather than for each vector. If the speed of I/O operations is a critical bottleneck (which often is the case when *n* is large enough) the Lanczos method essentially results in a *b*-fold reduction in I/O access.

Since the Lanczos process is an iterative one, it converges on the eigenvector solution. Based off the geometry of the converge it naturally finds the extremal eigenvalues first. This may be a welcome event, but in event that the user wishes to find the eigenvalues in a spectral range, the procedure can be modified to converge along any spectral range known *a priori.* Often in industrial applications, the spectral range is well known so the method is well suited for such problems. Since each calculation is essentially independent, the implementation is begging for parallelization.

Now consider the block matrix:

$$Q^T AQ = T_j = \begin{bmatrix} M_1 & B_1^T & & \\ B_1 & M_2 & B_2^T & \cdots \\ & & \vdots & B_{r-1}^T \\ & & B_{r-1} & M_r \end{bmatrix}$$

With an orthogonal Q such that:

$$Q = [X_1, \ldots, X_r]$$

We can establish a similar three member recurrence as the previous problem by equating columns by noting that $AQ = QT$, thus

$$AX_j = X_{j-1} B_{j-1}^T + X_j M_j + X_{j+1} B_j$$

Again, we define an intermediate:

$$R_j = AX_j - X_j M_j - X_{j-1} B_{j-1}^T$$

Which allows us to calculate the elements of X and B, by taking the QR decomposition:

$$X_{j+1} B_j = R_j$$

And iteratively, calculating the next element of the matrix:

$$M_{j+1} = X_{j+1}^T A X_{j+1}$$

With the appropriate transformations, T can be reduced to a scalar (non-block) tridiagonal form. The solution is now of size J = j*b, where J << n.

The loss of orthogonality is even more complicated in this case. Any serious code will have to deal with each of these issues to the desired accuracy. Without going into detail, there are three types:

1] internal – within the current block
2] local – with respect the previous block
3] global – with respect to all previous blocks

[6] Golub, Matrix Computations.

Sample Calculation in exact arithmetic:
This presentation is similar to that of Komzsik[7] but fewer of the intermediate steps are worked out. Given the unsymmetrical 3x3 matrix:

$$A = \begin{pmatrix} 1/2 & 1/2 & -1/2 \\ 0 & 0 & -2 \\ 3/2 & -1/2 & 9/2 \end{pmatrix}$$

With the starting vectors randomly selected (which do not have to be orthogonal, as in this case):

$$x_1 = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \qquad y_1 = \begin{bmatrix} 1/2 \\ -1/2 \\ -1/2 \end{bmatrix}$$

We can find the first intermediate vectors:

$$z_1 = Ax_1 = \begin{bmatrix} 1 \\ 2 \\ -3 \end{bmatrix} \qquad w_1 = A^T y_1 = \begin{bmatrix} -1/2 \\ -1/2 \\ -3/2 \end{bmatrix}$$

Get first Lanczos coefficient and update our intermediates:

$$\alpha_1 = y_1^T z_1 = 1$$

$$z_1 = z_1 - \alpha_1 x_1 - \gamma_{k-1} x_{k-1} = \begin{bmatrix} 0 \\ 2 \\ -2 \end{bmatrix}$$

$$w_1 = w_1 - \alpha_1 y_1 - \beta_{k-1} y_{k-1}^T = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$$

The next Lanczos coefficients are:

$$\delta_1 = v_1^T z_1 = 4 \qquad \beta_1 = \sqrt{\delta_1} = 2 = \gamma_1$$

The next iteration of the Lanczos procedure requires the normalization:

$$x_2 = \frac{z_1}{\beta_1} = \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix} \qquad y_2 = \frac{w_1}{\gamma_1} = \begin{bmatrix} -1/2 \\ 1/2 \\ -1/2 \end{bmatrix}$$

The process is repeated for the next iteration, yielding the next block of coefficients and Lanczos vectors:

$$x_2 = \begin{bmatrix} -1 \\ -1 \\ 0 \end{bmatrix} \qquad y_2 = \begin{bmatrix} -1/2 \\ -1/2 \\ -1/2 \end{bmatrix}$$

When the process is completed we have:

$$T_3 = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 3 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

While T matrix seems underwhelming[8] in comparison to the work done, we have calculated along the way the Lanczos vectors. This enables us to find the eigenvalues of T, [1, $2 \pm \sqrt{6}$] which are the same as those in A. In finite precision, as evidenced by the code in the Appendix gives the same T exactly, so the loss of orthogonality is not an issue when *n* is small.

Scientific code available online:
While coding is always an enjoyable endeavor, the scientific community would never advance if every paper had to re-derive each equation from scratch. The internet is a vast repository of numerical code, Lanczos included. Listed below is a partial list of such codes:

Netlib, a collection of mathematical software, papers, and databases (FORTRAN):
http://www.netlib.org/lanczos/

UC Davis has Lanczos in several languages (C, C++, FORTRAN) for different flavors of matrices:
http://www.cs.ucdavis.edu/~bai/ET/lanczos_methods/lanczos_methods.html

An implementation (C) of the square-shift method (not discussed in this paper):
http://angel.elte.hu/lanczos/

[7] Komzsik, The Lanczos Method

[8] A 3x3 tridiagonal matrix isn't that different from an unsymmetrical one anyways!

Appendix:

Code to calculate the Lanczos vectors:

Highlights were used to illustrate the Lanczos procedure. The rest of the code consists of formatting and simple Matrix operations.

```
/* Easily adaptable code for calculating the
Lanczos vectors of a given matrix. Extensions
can be made to improve numerical stability by
any orthogonlization procedure due to the
inherent instabilities of finite precision. -
Travis Hoppe, 3.7.05 */

#include <iostream>
#include <iomanip>
#include <cmath>
#include "nrutil.h"
using namespace std;
int N=3;

// Matrix operations
void MdV( double ** A, double * b, double * out
) {
  int i,j;double value=0;
  for(j=0;j<N;j++){
    for(i=0,value=0;i<N;i++)
      value+=A[j][i]*b[i];
    out[j] = value;  }}

void VTdM( double * bT, double ** A, double *
out) {
  int i,j;double value=0;
  for(j=0;j<N;j++){
    for(i=0,value=0;i<N;i++)
      value+=A[i][j]*bT[i];
    out[j] = value;  }}

void VTdV( double *bT, double * c, double & out)
{
  out = 0;
  for(int i=0;i<N;i++)
    out += bT[i]*c[i]; }

void VmV( double *b, double & s, double *c) {
  for(int i=0;i<N;i++)
    b[i] -= s * c[i]; }

void VdS( double *b, double s) {
  for(int i=0;i<N;i++)
    b[i] *= s; }

// Various outputs
void printV( double * b ) {
  cout << " [";
  for(int i=0;i<N;i++)
    cout << setw(8) << b[i]; cout << "] "; }

void printV( double * b, int offset ) {
  cout << " [";
  for(int i=0+offset;i<N+offset;i++)
    cout << setw(8) << b[i]; cout << "] "; }

int main() {

  // allocate space for the variables
  int i,j,k;
  double ** A    = dmatrix(0,N,0,N);
  double ** T    = dmatrix(0,N,0,N);
  double *x[N+1],*y[N+1],*z[N+1],*w[N+1];
  double *alpha = dvector(0,N+1);
  double *beta  = dvector(0,N+1);
  double *gamma = dvector(0,N+1);
  double delta;

  for(i=0;i<N+1;i++) {
    x[i] = dvector(0,N);
    y[i] = dvector(0,N);
    z[i] = dvector(0,N);
    w[i] = dvector(0,N); }

  // set the sample matrix up
  A[0][0]=.5;  A[0][1]=.5;  A[0][2]=-.5;
  A[1][0]= 0;  A[1][1]= 0;  A[1][2]=- 2;
  A[2][0]=1.5; A[2][1]=-.5; A[2][2]=4.5;
  x[1][0]=1;   x[1][1]=0;   x[1][2]=-1;
  y[1][0]=.5;  y[1][1]=-.5; y[1][2]=-.5;

  // calculate the Lanczos vectors
  for(k=1;k<N;k++) {

    MdV (A,x[k],z[k]);
    VTdM(y[k],A,w[k]);
    VTdV(y[k],z[k],alpha[k]);

    VmV (z[k],alpha[k],x[k]);
    VmV (z[k],gamma[k-1],x[k-1]);

    VmV (w[k],alpha[k],y[k]);
    VmV (w[k],beta[k-1] ,y[k-1]);

    VTdV(w[k],z[k],delta);
    beta[k] = sqrt(abs(delta));

    gamma[k] = beta[k];
    if(gamma[k]<0) gamma[k] *= -1;

    x[k+1] = z[k];
    VdS(x[k+1], 1.0/beta[k]  );

    y[k+1] = w[k];
    VdS(y[k+1], 1.0/gamma[k] );

    cout << k << " -> ";
    cout << "x "; printV(x[k]);
    cout << "y "; printV(y[k]);
    cout << alpha[k] << ' '
         << beta[k] << ' ' << gamma[k];
    cout << endl;

    cout << "     ";
    cout << "v "; printV(z[k]);
    cout << "w "; printV(w[k]);
    cout << endl;

  }

  MdV (A,x[N],z[N]);
  VmV (z[N],alpha[N],x[N]);
  VTdV(y[N],z[N],alpha[N]);

  // display the output
  cout << k << " -> ";
  cout << "x "; printV(x[N]);
  cout << "y "; printV(y[N]);
  cout << alpha[N] << endl;

  cout << " -=-=-=-=-=-=-= " << endl;
  cout << " diagonal: ";  printV(alpha,1);  cout
<< " super-di: ";  printV(gamma,0); cout <<
endl;

  return 0;}
```