The Lanczos Algorithm

Shane Harding

February 19, 2015

Outline of talk

Introduction

The Lanczos Algorithm is a very efficient method of finding, a few, ‘extreme’ eigenvalues of symmetric matrices, A.
By extreme, we mean the eigenvalues with the largest and smallest ablsoute values. The algorithm will only find a small number, m, of the total number, n, of eigenvalues.

What is an Eigenvalue

If we consider the matrix equation: $Ax = b$. Then x is an eigenvector if, and only if, applying A to x yields a scalar multiple of x. The factor x is rescaled by is called the eigenvalue and is denoted λ.

$$Ax = \lambda x$$

Brief History

The Lanczos algorithm was developed by Cornelius Lanczos. He developed the method while working at the National Bureau of Standards in Washington DC. It was then forgotten for a number of years due to the lack of computers meaning that it couldn't be properly utilised. When it was then "rediscovered" it was modified multiple times to fix some instability issues.

Brief History

Lanczos later joined the Institute of Numerical Analysis (INA). While working there he entered correspondence with Erwin Schrödinger, who was the director of the Dublin Institute of Advanced Studies (DIAS) at the time. Schroödinger gave him one year visiting professorship at DIAS. In 1954 Eamon de Valera invited Lanczos to be a senior professor in the School of Theoretical Physics.

The Lanczos Algorithm is used to solve large scale eigenvalue problems. So give a large $n \times n$ matrix A we have:

Eigenvalues and eigenvectors

$$Av_i = \lambda_i v_i$$

Where the vectors v_i are the eigenvectors and the scalars λ_i are the eigenvalues.

Power Iteration

- The Lanczos algorithm is an adaptation of the power method.
- The power methods (or power iteration or Von Mises iteration) is an eigenvalue algorithm.
- This algorithm will only produce one eigenvalue and may converge slowly.
- It finds the eigenvalue with the greatest absolute value.

Power Iteration

The power method can be summarised as follows: If x_0 is some random vector and $x_{n+1} = Ax_n$ then if we consider the limit of n being large we find that $\frac{x_n}{||x_n||}$ approaches the normed eigenvector with the greatest eigen value.

- Then if $A = Udiag(\sigma_i)U'$ is the eigendeecomposition of A then $A^n = Udiag(\sigma_i^n)U'$.
- As n gets big this will be dominated by the largest eigenvalue

The Lanczos Algorithm

- We wish to find the eigenvalues of the matrix A.
- Where $A \in \mathbb{R}^{N \times N}$ and $v_1 \in \mathbb{R}^N$ be a random vector, with norm one.
- We will calculate the tridiagonal, symmetric matrix $T_{mm} = V_m^* A V_m$. With the diagonal elements denoted $\alpha_i = t_{ii}$ and the off diagonal elements given by $\beta_i = t_{i-1,i}$. And note that $t_{i-1,i} = t_{i,i-1}$ due to symmetry.

The Lanczos Algorithm

```
1. v_1 = random vec; v_0 = 0; beta_1 = 0;
2. for i = 1,...,m -1do
     w_i = A v_i
     alpha_i = (r_i,v_i)
     w_i = w_i - alpha_i v_i - beta_i v_{i=1}
     beta_{i+1} = abs(w_i)
     v_{i+1} = w_i/beta_{i+1}
   endfor
3. w_m = A v_m
   alpha_m = (w_m,v_m)
   return
```

Where (x, y) represents the dot product of x and y. m is chosen by the user. It is the number of eigenvalues that want to be obtained.

The Lanczos Algorithm

The previous routine yields the matrix:

$$T_{mm} = \begin{pmatrix} \alpha_1 & \beta_2 & & 0 \\ \beta_2 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_m \\ 0 & & \beta_{m,} & \alpha_m \end{pmatrix}$$

The vectors v_i are called the Lanczos vectors. They are then used to construct the transformation matrix $V_m = (v_1, v_2, \ldots, v_m)$. This is very useful for computing the eigenvectors.

The Lanczos Algorithm

- The vectors v_i in our algorithm should all be orthogonal to each other. Rounding errors however will grow over time so every few iterations a reorthogonalisation process should be run.
- Once T_{mm} is obtained, its eigenvalues $\lambda_i^{(m)}$ and their matching eigenvectors $u_i^{(m)}$ can be calculated.
- There are a few commonly used methods to do this. Two examples are: multiple relatively robust representations (MRRR); and QR algorithm.
- It can be proved that the eigenvalues of T_{mm} are approximate eigenvalues of the original matrix A.

Parallelising The Lanczos Algorithm

- There are a few different ways of parallelising the Lanczos Algorithm.
- They involve various ways to partition the matrix and distribute the calculations.
- Essentially it involves parallelising the matrix-vector product.

Per element matrix distribution

- Each processor is given a subset of, nonzero, elements of the matrix.
- The distribution is balanced such that each processor gets the same number of elements.
- The actual algorithm is preformed by the master process, while the matrix-vector operation is distributed.
- The slave processors complete the following:
  - Master decides which parts of the vector each slave needs.
  - Each slave computes its part of the matrix-vector product and sends its data back
  - The master recieves the data and does its thing for the final result.

Per column matrix distribution

- Divide the columns of the matrix evenly between the processors. Ideally it should be preprocessed so that each processor has same amount of nonzero elements.
- All the computations for the algorithm are done by all the processors. Each processor has $1/NP$ of the vector.
- The matrix-vector product is preformed in the stages:
    - Each processor computes its part of the matrix-vector product
    - These parts are then combined.
- Assuming that we perfectly balance the load, we should get a scaling factor of PN^{-1}.

Per submatrix matrix distribution

- This requires a mesh architecture on the cluster.
- Split the matrix into P submatrices, of roughly the same size, and distribute them between the processors.
  - Each proc computes the matrix vector product with its submatrix and the relevant part of the vector
  - These results are then combined
- This method has a scale factor of $PN^{-1/2}$.

Applications

There are many many applications of the Lanczos method. It is a popular method in Condensed Matter Physics where the Hamiltonians of electron systems need to be solved. It is also in solving non-linear inverse problems, which is very useful in modeling oil and gas reservoirs.
Eigenvectors are important in large scale ranking methods. Google's PageRank algorithm is good example of this.

Google PageRank Algorithm

When you make a Google search, Google uses an algorithm, called PageRank, to rank the results of your search. This is what Google says the PageRank algorithm is:

PageRank

PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites

Google PageRank Algorithm

- PageRank gives a probability distribution that represents the likelihood that if you randomly click on links you will arrive on a given page.
- Each page, u is given a value, $PR(u)$, between zero and one; and the sum over all the PR's must be one.
- The initial value of $PR(u)$ is simply one divided by the number of pages for each page u. So if there are 12 pages, each $PR(u_i)$ is $\frac{1}{12}$.

Google PageRank Algorithm

- Let B_p be the set of all the pages that link to the page p.
- Let $L(q)$ be the number of unique outgoing links from a page q.
- Then the general PageRank value can be written as:

$$PR(p) = \sum_{q \in B_p} \frac{PR(q)}{L(q)}$$

- This is then modified to include a 'damping' factor:

$$PR(p_1) = \frac{1-d}{N} + d\left(\frac{PR(p_2)}{L(p_2)} + \frac{PR(p_3)}{L(p_3)} + \dots\right)$$

Where's Lanczos?

Can write a column vector R as:

$$R = \begin{pmatrix} PR(p_1) \\ \vdots \\ PR(p_N) \end{pmatrix}$$

Then we can write the equation from the last slide as:

$$R = \begin{pmatrix} \frac{1-d}{N} \\ \vdots \\ \frac{1-d}{N} \end{pmatrix} + d \begin{pmatrix} l(p_1, p_1) & l(p_1, p_2) & \dots & l(p_1, p_N) \\ l(p_2, p_1) & \ddots & & \vdots \\ \vdots & & l(p_i, p_j) & \\ l(p_N, p_1) & \dots & & l(p_N, p_N) \end{pmatrix} R$$

Questions?

Thank you!