A Short Course in Parallel Scientific Computing

Technical University of Budapest

Graham Horton, University of Erlangen-Nürnberg

Talk 5 : Eigenvalue Computations

Contents

Introduction

Consider a matrix A

The *eigenvalue problem*: Find (one or all) λ and u such that

$$Au = \lambda u$$

λ is an *eigenvalue*, u is an *eigenvector*

A common problem in Science and Engineering

A is typically very large

This leads to long computation times

We will consider the special case:

$$\begin{aligned} A &\in \mathcal{R}^{n\times n} \\ A &= A^T \;\; (A\ symmetric) \end{aligned}$$

Motivation - Example

A historic example :

Kon¡anz, 23. Februar, 1549 :

Uff disen tag ... morgens frueh, i¡ ϒr see an und abgeloffen, wol einer elen `Λ, ϒr ge¡alt, so i¡ er ... sΛier bis zu ϒr Spitals Egk Ωruff gegangen; so er abgeloffen, ... so er so klain geworϒn, so i¡ er Γld mit einem rusΛen ... wiϒr angeloffen. Und solΛs i¡ etwa in ainer ¡und vier oϒr funf mal gesΛeΩn.

Eigenmode of the Bodensee (Lake Constance)

Possibly initiated by an earthquake

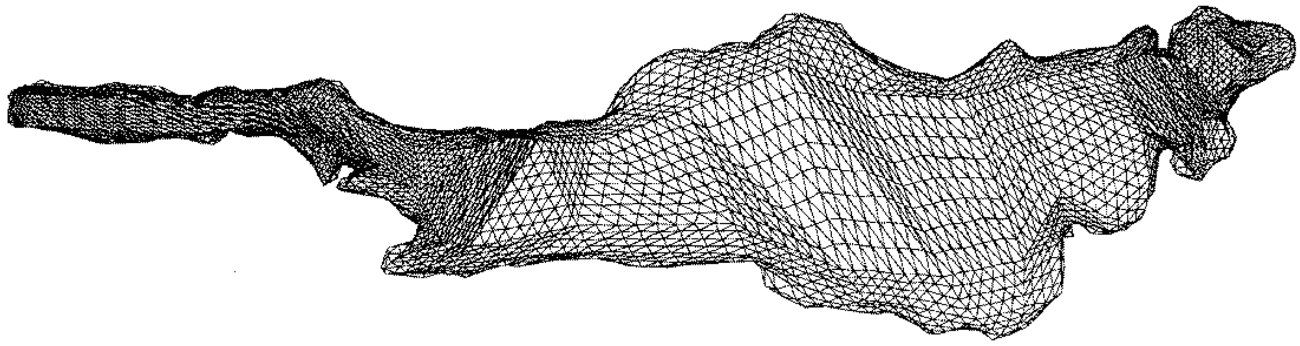
10th eigenmode has

- a maximum in Konstanz and
- a period of 12 minutes!

Motivation - Example

Compute the eigenmode by *discretisation*

Discretise the domain into triangles:



Build discrete version of the water motion equations

Compute eigenvalues/eigenvectors of the discretisation matrix

Eigenvalues define the period of the eigenmode

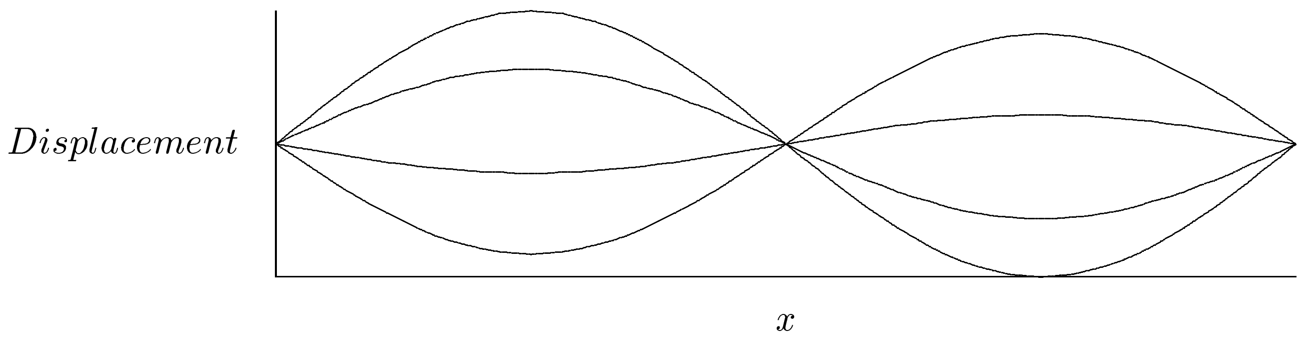
Eigenvectors describe the surface heights of the eigenmodes

Motivation - Example

Natural vibration of a fixed elastic rope

Example: Guitar or violin string

Problem: compute displacement of the string $u(x,t)$



The problem is described by the differential equation

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}$$

in the domain $0 < x < 1$

additionally :

$$u(0) = u(1) = 0$$

Example

Simplify the equation with the

$\rightarrow$ *Separation of variables*

$$u(t, x) = v(t)w(x)$$

giving

$$\begin{aligned} \frac{\partial^2 v}{\partial t^2} &= cv \\ \frac{\partial^2 w}{\partial x^2} &= cw \end{aligned}$$

$+$ boundary conditions

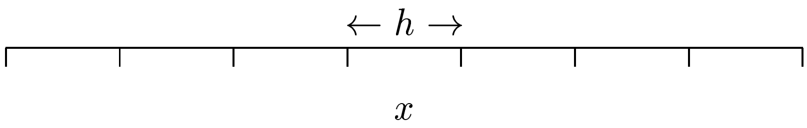
Such equations cannot normally be solved analytically in general

$\rightarrow$ *Approximate solution via discretisation*

(In this case there is an analytic solution)

Discretisation

We discretise the domain $(0, 1)$:



We replace the differential equation by the approximation

$$\frac{w_{i-1} - 2w_i + w_{i+1}}{h^2} = cw_i$$

Writing this linear system as $Aw = \lambda w$ with

$$A = \frac{1}{h^2} \begin{bmatrix} -2 & 1 & & & & & \\ 1 & -2 & 1 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & & & 1 & -2 & 1 \\ & & & & & 1 & -2 \end{bmatrix}$$

Results in an eigenvalue problem

Similarly for Bodensee problem :

Discretise shallow water equations $\Rightarrow$ linear system

Compute eigenvalues of the resulting matrix

The Jacobi Eigenvalue Method

The classical method for the computation of eigenvalues is the

Jacobi method (Jacobi, 1846)

It consists of transforming the matrix A into a diagonal matrix B

B has the same eigenvalues as A

(These are the diagonal elements $b_{i,i}$)

The method is *iterative*

The diagonalisation is performed by *orthogonal transformations*:

$$A_{(i+1)} = R_i^T A_{(i)} R_i \qquad B = A_{(\infty)}$$

since the matrices R are chosen to be orthogonal

Algorithmically :

```
procedure Jacobi
   while ?? do
      compute new R
      compute A = R^T A R
```

The Jacobi Eigenvalue Method

Three definitions:

- The matrix $R \in \mathcal{R}^{n \times n}$ is *orthogonal*, if

$$RR^T = I$$

  ($I =$ Identity matrix)

- The *Frobenius norm* $\|A\|_F$ of the matrix A is

$$\|A\|_F = \left(\sum_{i=1}^{i=n} \sum_{j=1}^{j=n} a_{ij}^2 \right)^{\frac{1}{2}}$$

- The set of eigenvalues of the matix A is the *spectrum* of A:

$$\lambda(A)$$

Then for an orthogonal matrix R:

$$\|A\|_F = \|R^T A R\|_F$$

$$\lambda(A) = \lambda(R^T A R)$$

The Jacobi Eigenvalue Method

The orthogonal matrices of the Jacobi method are:

$$R = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & c & & s & & \\ & & & \ddots & & & \\ & & -s & & c & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix}$$

with

$$c = \cos(\theta) \qquad s = \sin(\theta)$$

The four non-identity matrix elements are in the positions

(p,p), (p,q), (q,p) and (q,q) with $p < q$

Then for an appropriate choice of θ :

$$b_{p,q} = b_{q,p} = 0 \qquad with \;\; B = R^T A R$$

For θ holds:

$$\begin{aligned} \tan 2\theta \; &= \; \frac{2a_{pq}}{a_{pp} - a_{qq}} \\ 0 \; &< \; \theta \leq \frac{\pi}{4} \end{aligned}$$

The Jacobi Eigenvalue Method

We can make two coefficients of A equal to zero

Doing this modifies the p-th and q-th row and column of A

We therefore successively choose different p and q

But: elements of A that were made to zero thereby become $\neq 0$

However, overall all off-diagonal elements of A are reduced

Define :

$$off(A) = \left(\sum_{i \neq j} a_{ij}^2 \right)^{\frac{1}{2}}$$

(off = Off-diagonal)

$off(A)$ measures how much A differs from a diagonal matrix

We would like to reduce $off(A)$ in each iteration by as much as possible

Then diagonal form will be reached as quickly as possible

N.B. The eigenvectors are the columns of the matrix

$$S = R_1 R_2 \ldots$$

The Jacobi Eigenvalue Method

Consider the sub-matrix equation:

$$\begin{bmatrix} b_{pp} & b_{pq} \\ b_{qp} & b_{qq} \end{bmatrix} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$

We have

$$b_{pp}^2 + b_{qq}^2 + 2b_{pq}^2 = a_{pp}^2 + a_{qq}^2 + 2a_{pq}^2$$

Because of

$$\|B\|_F^2 = \|A\|_F^2$$

follows

$$off(B) = off(A) - 2a_{pq}^2 + 2b_{pq}^2$$

$off(B)$ is to be minimised

b_{pq} becomes $= 0$, so

choose p and q so that a_{pq} is maximal:

$$|a_{pq}| \geq |a_{ij}| \quad 1 \leq i, j \leq n$$

```
procedure Jacobi
   while off(A) > ε do
      find p, q so that |a_pq| ≥ |a_ij|
      compute R
      compute A = R^T A R
```

Example Computation

```
Eigenvalue computation with the Jacobi method

10.00   3.00   4.00   5.00
 3.00  20.00   5.00   6.00     Iter.  0 Err. = 4.47e+00
 4.00   5.00  30.00   7.00               Max el. (3,4)
 5.00   6.00   7.00  40.00


10.00   3.00   1.27   6.28
 3.00  20.00   1.70   7.62     Iter.  1 Err. = 3.72e+00
 1.27   1.70  26.40   0.00               Max el. (2,4)
 6.28   7.62   0.00  43.60


10.00   1.10   1.27   6.87
 1.10  17.75   1.63   0.00     Iter.  2 Err. = 2.57e+00
 1.27   1.63  26.40   0.48               Max el. (1,4)
 6.87   0.00   0.48  45.85


 8.73   1.08   1.16  -0.00
 1.08  17.75   1.63   0.20     Iter.  3 Err. = 8.45e-01
 1.16   1.63  26.40   0.70               Max el. (2,3)
-0.00   0.20   0.70  47.12
```

Beispielrechnung

```
 8.73   0.86   1.34  -0.00
 0.86  17.45  -0.00   0.07      Iter.  4 Err. = 6.18e-01
 1.34  -0.00  26.70   0.73                Max el. (1,3)
-0.00   0.07   0.73  47.12

 8.63   0.86   0.00  -0.05
 0.86  17.45   0.06   0.07      Iter.  5 Err. = 3.99e-01
 0.00   0.06  26.79   0.73                Max el. (1,2)
-0.05   0.07   0.73  47.12

              .                 Iter.  6 Err. = 2.60e-01
              .                 Iter.  7 Err. = 3.86e-02
              .                 Iter.  8 Err. = 3.03e-02
              .                 Iter.  9 Err. = 2.14e-02
              .                 Iter. 10 Err. = 1.38e-03
              .                 Iter. 11 Err. = 7.73e-05
              .                 Iter. 12 Err. = 5.12e-05
              .                 Iter. 13 Err. = 4.13e-07

 8.55   0.00  -0.00   0.00
 0.00  17.54  -0.00  -0.00
-0.00  -0.00  26.77  -0.00
 0.00  -0.00  -0.00  47.15

The eigenvalues are :   8.55  17.54  26.77  47.15
```

Parallelisation

The key to the parallelisation is the following observation:

Only the p-th and q-th row and column of A are changed

All other matrix entries are unchanged

We can therefore make several entries a_{pq} equal to zero simultaneously

Requirement:

$$\begin{aligned} p_i &\neq p_j \quad \forall i \neq j \\ q_i &\neq q_j \quad \forall i \neq j \\ p_i &\neq q_j \quad \forall i \neq j \end{aligned}$$

(All rows and columns are different)

Example :

$$R = \begin{bmatrix} 1 & & & & & & & & \\ & c & & s & & & & & \\ & & \ddots & & & & & & \\ & -s & & c & & & & & \\ & & & & \ddots & & & & \\ & & & & & 1 & & & \\ & & & & & & c & s & \\ & & & & & & -s & c & \\ & & & & & & & & 1 \end{bmatrix}$$

Parallelisation

Consider first a multiprocessor with global memory

Assign each processor a different index pair (p, q)

The program for each processor is very simple:

```
procedure Jacobi
   while off(A) > ε do
      compute p, q
      compute segment of R
      compute segment of A = AR
      synchronise
      compute segment of A = R^T A
```

Segments of R are the $\cos$ and $\sin$ values

Segments of A are two rows or columns

Synchronisation necessary between $A = AR$ and $A = R^T A$

(Because of danger of overwriting)

Parallelisation

Comments:

- Determine p and q so that
  - In one iteration no overlapping occurs
  - After several iterations all elements have been visited
- Computation of $off(A)$ contains a global sum

Example for p, q when $A \in \mathcal{R}^{8\times 8}$:

$Iter$				
1	$(1,2)$	$(3,4)$	$(5,6)$	$(7,8)$
2	$(1,3)$	$(2,4)$	$(5,7)$	$(6,8)$
3	$(1,4)$	$(2,3)$	$(5,8)$	$(6,7)$
4	$(1,5)$	$(2,6)$	$(3,7)$	$(4,8)$
5	$(1,6)$	$(2,5)$	$(3,8)$	$(4,7)$
6	$(1,7)$	$(2,8)$	$(3,5)$	$(4,6)$
7	$(1,8)$	$(2,7)$	$(3,6)$	$(4,5)$

Parallelisiation

Consider multiprocessor with distributed memory

The comments from the previous slide are still valid

If we distribute the indices p, q as before ...

Each processor must process two rows and columns

However p and q change from one iteration to the next

Therefore we have to exchange rows and columns before each iteration (!)

This is much too expensive

Solution: partition matrix A onto processors

Each processor works on its part of the corresponding rows and columns p, q

Problem: further condition on p and q: load balance!

Parallelisation

Analysis of the parallel method

Each processor receives a portion of the matrix A, so

The message-passing algorithm is memory optimal

Speedup is $\mathcal{O}(p)$ for matrix multiplications

The method is optimal, except

Computation of $off(A)$ contains global sum

But: There has been a change in the ordering of elements

$\Rightarrow$ Higher number of iterations is possible

This means we cannot make any statement about the overall speedup

In practice even sequentially a fixed ordering is used

(Because of the cost of searching for the largest a_{pq})

The matrices are large, hence also the maximal degree of parallelism