Eigenvalue problems

Morten Hjorth-Jensen[1,2]

[1]Department of Physics, University of Oslo, Norway
[2]Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State University, USA

May 16-20 2016

Overview of eigenvalue discussion

Eigenvalue problems.

- Discussion of Jacobi's algorithm
- Presentation of quantum dot problem for two electrons.
- Discussion of Householder's and Francis' algorithms
- Power methods
- Lanczos' method

The physical problem

- Two fermions in a trap in two or three dimensions
- Or two bosons in a trap in two or three dimensions
- How to solve a one-body problem in an external potential
- Ground state and excited states

© 1999-2016, Morten Hjorth-Jensen. Released under CC Attribution-NonCommercial 4.0 license

Eigenvalue problems, basic definitions

Let us consider the matrix $\mathbf{A}$ of dimension n. The eigenvalues of $\mathbf{A}$ are defined through the matrix equation

$$\mathbf{A}\mathbf{x}^{(\nu)} = \lambda^{(\nu)}\mathbf{x}^{(\nu)},$$

where $\lambda^{(\nu)}$ are the eigenvalues and $\mathbf{x}^{(\nu)}$ the corresponding eigenvectors. Unless otherwise stated, when we use the wording eigenvector we mean the right eigenvector. The left eigenvalue problem is defined as

$$\mathbf{x}_L^{(\nu)}\mathbf{A} = \lambda^{(\nu)}\mathbf{x}_L^{(\nu)}$$

The above right eigenvector problem is equivalent to a set of n equations with n unknowns x_i.

Eigenvalue problems, basic definitions

The eigenvalue problem can be rewritten as

$$\left(\mathbf{A} - \lambda^{(\nu)}\mathbf{I}\right)\mathbf{x}^{(\nu)} = 0,$$

with $\mathbf{I}$ being the unity matrix. This equation provides a solution to the problem if and only if the determinant is zero, namely

$$\left|\mathbf{A} - \lambda^{(\nu)}\mathbf{I}\right| = 0,$$

which in turn means that the determinant is a polynomial of degree n in λ and in general we will have n distinct zeros.

Eigenvalue problems, basic definitions

The eigenvalues of a matrix $\mathbf{A} \in \mathbb{C}^{n\times n}$ are thus the n roots of its characteristic polynomial

$$P(\lambda) = det(\lambda\mathbf{I} - \mathbf{A}),$$

or

$$P(\lambda) = \prod_{i=1}^{n}(\lambda_i - \lambda).$$

The set of these roots is called the spectrum and is denoted as $\lambda(\mathbf{A})$. If $\lambda(\mathbf{A}) = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ then we have

$$det(\mathbf{A}) = \lambda_1\lambda_2\ldots\lambda_n,$$

and if we define the trace of $\mathbf{A}$ as

$$Tr(\mathbf{A}) = \sum_{i=1}^{n} a_{ii}$$

then

$$Tr(\mathbf{A}) = \lambda_1 + \lambda_2 + \cdots + \lambda_n.$$

Abel-Ruffini Impossibility Theorem

The *Abel-Ruffini* theorem (also known as Abel's impossibility theorem) states that there is no general solution in radicals to polynomial equations of degree five or higher.

The content of this theorem is frequently misunderstood. It does not assert that higher-degree polynomial equations are unsolvable. In fact, if the polynomial has real or complex coefficients, and we allow complex solutions, then every polynomial equation has solutions; this is the fundamental theorem of algebra. Although these solutions cannot always be computed exactly with radicals, they can be computed to any desired degree of accuracy using numerical methods such as the Newton-Raphson method or Laguerre method, and in this way they are no different from solutions to polynomial equations of the second, third, or fourth degrees.

The theorem only concerns the form that such a solution must take. The content of the theorem is that the solution of a higher-degree equation cannot in all cases be expressed in terms of the polynomial coefficients with a finite number of operations of addition, subtraction, multiplication, division and root extraction. Some polynomials of arbitrary degree, of which the simplest nontrivial example is the monomial equation $ax^n = b$, are always solvable with a radical.

Abel-Ruffini Impossibility Theorem

The *Abel-Ruffini* theorem says that there are some fifth-degree equations whose solution cannot be so expressed. The equation $x^5 - x + 1 = 0$ is an example. Some other fifth degree equations can be solved by radicals, for example $x^5 - x^4 - x + 1 = 0$. The precise criterion that distinguishes between those equations that can be solved by radicals and those that cannot was given by Galois and is now part of Galois theory: a polynomial equation can be solved by radicals if and only if its Galois group is a solvable group.

Today, in the modern algebraic context, we say that second, third and fourth degree polynomial equations can always be solved by radicals because the symmetric groups S_2, S_3 and S_4 are solvable groups, whereas S_n is not solvable for $n \geq 5$.

Eigenvalue problems, basic definitions

In the present discussion we assume that our matrix is real and symmetric, that is $\mathbf{A} \in \mathbb{R}^{n\times n}$. The matrix $\mathbf{A}$ has n eigenvalues $\lambda_1 \dots \lambda_n$ (distinct or not). Let $\mathbf{D}$ be the diagonal matrix with the eigenvalues on the diagonal

$$\mathbf{D} = \begin{pmatrix} \lambda_1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & \lambda_{n-1} & \\ 0 & \dots & \dots & \dots & \dots & 0 & \lambda_n \end{pmatrix}.$$

If $\mathbf{A}$ is real and symmetric then there exists a real orthogonal matrix $\mathbf{S}$ such that

$$\mathbf{S}^T\mathbf{A}\mathbf{S} = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n),$$

and for $j = 1 : n$ we have $\mathbf{A}\mathbf{S}(:, j) = \lambda_j \mathbf{S}(:, j)$.

Eigenvalue problems, basic definitions

To obtain the eigenvalues of $\mathbf{A} \in \mathbb{R}^{n\times n}$, the strategy is to perform a series of similarity transformations on the original matrix $\mathbf{A}$, in order to reduce it either into a diagonal form as above or into a tridiagonal form.

We say that a matrix $\mathbf{B}$ is a similarity transform of $\mathbf{A}$ if

$$\mathbf{B} = \mathbf{S}^T\mathbf{A}\mathbf{S}, \qquad \text{where} \qquad \mathbf{S}^T\mathbf{S} = \mathbf{S}^{-1}\mathbf{S} = \mathbf{I}.$$

The importance of a similarity transformation lies in the fact that the resulting matrix has the same eigenvalues, but the eigenvectors are in general different.

Eigenvalue problems, basic definitions

To prove this we start with the eigenvalue problem and a similarity transformed matrix $\mathbf{B}$.

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \qquad \text{and} \qquad \mathbf{B} = \mathbf{S}^T\mathbf{A}\mathbf{S}.$$

We multiply the first equation on the left by $\mathbf{S}^T$ and insert $\mathbf{S}^T\mathbf{S} = \mathbf{I}$ between $\mathbf{A}$ and $\mathbf{x}$. Then we get

$$(\mathbf{S}^T\mathbf{A}\mathbf{S})(\mathbf{S}^T\mathbf{x}) = \lambda\mathbf{S}^T\mathbf{x}, \tag{1}$$

which is the same as

$$\mathbf{B}\left(\mathbf{S}^T\mathbf{x}\right) = \lambda\left(\mathbf{S}^T\mathbf{x}\right).$$

The variable λ is an eigenvalue of $\mathbf{B}$ as well, but with eigenvector $\mathbf{S}^T\mathbf{x}$.

Eigenvalue problems, basic definitions

The basic philosophy is to

- Either apply subsequent similarity transformations (direct method) so that

$$\mathbf{S}_N^T \ldots \mathbf{S}_1^T\mathbf{A}\mathbf{S}_1 \ldots \mathbf{S}_N = \mathbf{D}, \tag{2}$$

- Or apply subsequent similarity transformations so that $\mathbf{A}$ becomes tridiagonal (Householder) or upper/lower triangular (the QR method to be discussed later).
- Thereafter, techniques for obtaining eigenvalues from tridiagonal matrices can be used.
- Or use so-called power methods
- Or use iterative methods (Krylov, Lanczos, Arnoldi). These methods are popular for huge matrix problems.

Discussion of Jacobi's method for eigenvalues

The general overview. One speaks normally of two main approaches to solving the eigenvalue problem.

- The first is the formal method, involving determinants and the characteristic polynomial. This proves how many eigenvalues there are, and is the way most of you learned about how to solve the eigenvalue problem, but for matrices of dimensions greater than 2 or 3, it is rather impractical.
- The other general approach is to use similarity or unitary tranformations to reduce a matrix to diagonal form. This is normally done in two steps: first reduce to for example a *tridiagonal* form, and then to diagonal form. The main algorithms we will discuss in detail, Jacobi's and Householder's (so-called direct method) and Lanczos algorithms (an iterative method), follow this methodology.

Discussion of Jacobi's method for eigenvalues

Direct or non-iterative methods require for matrices of dimensionality $n \times n$ typically $O(n^3)$ operations. These methods are normally called standard methods and are used for dimensionalities $n \sim 10^5$ or smaller. A brief historical overview

Year	n	
1950	$n = 20$	(Wilkinson)
1965	$n = 200$	(Forsythe et al.)
1980	$n = 2000$	Linpack
1995	$n = 20000$	Lapack
2012	$n \sim 10^5$	Lapack

shows that in the course of 60 years the dimension that direct diagonalization methods can handle has increased by almost a factor of 10^4. However, it pales beside the progress achieved by computer hardware, from flops to petaflops, a factor of almost 10^{15}. We see clearly played out in history the $O(n^3)$ bottleneck of direct matrix algorithms.

Sloppily speaking, when $n \sim 10^4$ is cubed we have $O(10^{12})$ operations, which is smaller than the 10^{15} increase in flops.

Discussion of Jacobi's method for eigenvalues

If the matrix to diagonalize is large and sparse, direct methods simply become impractical, also because many of the direct methods tend to destroy sparsity. As a result large dense matrices may arise during the diagonalization procedure. The idea behind iterative methods is to project the $n-$dimensional problem in smaller spaces, so-called Krylov subspaces. Given a matrix $\mathbf{A}$ and a vector $\mathbf{v}$, the associated Krylov sequences of vectors (and thereby subspaces) $\mathbf{v}$, $\mathbf{Av}$, $\mathbf{A}^2\mathbf{v}$, $\mathbf{A}^3\mathbf{v}, \ldots$, represent successively larger Krylov subspaces.

Matrix	$\mathbf{Ax} = \mathbf{b}$	$\mathbf{Ax} = \lambda\mathbf{x}$
$\mathbf{A} = \mathbf{A}^*$	Conjugate gradient	Lanczos
$\mathbf{A} \neq \mathbf{A}^*$	GMRES etc	Arnoldi

Discussion of Jacobi's method for eigenvalues

The Numerical Recipes codes have been rewritten in Fortran 90/95 and C/C++ by us. The original source codes are taken from the widely used software package LAPACK, which follows two other popular packages developed in the 1970s, namely EISPACK and LINPACK.

- LINPACK: package for linear equations and least square problems.
- LAPACK:package for solving symmetric, unsymmetric and generalized eigenvalue problems. From LAPACK's website `http://www.netlib.org` it is possible to download for free all source codes from this library. Both C/C++ and Fortran versions are available.
- BLAS (I, II and III): (Basic Linear Algebra Subprograms) are routines that provide standard building blocks for performing basic vector and matrix operations. Blas I is vector operations, II vector-matrix operations and III matrix-matrix operations.

Discussion of Jacobi's method for eigenvalues

Consider an example of an $(n \times n)$ orthogonal transformation matrix

$$\mathbf{S} = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & 0 & \dots \\ 0 & 0 & \dots & \cos\theta & 0 & \dots & 0 & \sin\theta \\ 0 & 0 & \dots & 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & 1 & \dots \\ 0 & 0 & \dots & -\sin\theta & 0 & \dots & 0 & \cos\theta \end{pmatrix}$$

with property $\mathbf{S^T} = \mathbf{S^{-1}}$. It performs a plane rotation around an angle θ in the Euclidean $n-$dimensional space.

Discussion of Jacobi's method for eigenvalues

It means that its matrix elements that differ from zero are given by

$$s_{kk} = s_{ll} = \cos\theta, s_{kl} = -s_{lk} = -\sin\theta, s_{ii} = 1 \quad i \neq k \quad i \neq l,$$

A similarity transformation

$$\mathbf{B} = \mathbf{S}^T\mathbf{AS},$$

results in

$$\begin{aligned}
b_{ik} =& a_{ik}\cos\theta - a_{il}\sin\theta, i \neq k, i \neq l \\
b_{il} =& a_{il}\cos\theta + a_{ik}\sin\theta, i \neq k, i \neq l \\
b_{kk} =& a_{kk}\cos^2\theta - 2a_{kl}\cos\theta\sin\theta + a_{ll}\sin^2\theta \\
b_{ll} =& a_{ll}\cos^2\theta + 2a_{kl}\cos\theta sin\theta + a_{kk}\sin^2\theta \\
b_{kl} =& (a_{kk} - a_{ll})\cos\theta\sin\theta + a_{kl}(\cos^2\theta - \sin^2\theta)
\end{aligned}$$

The angle θ is arbitrary. The recipe is to choose θ so that all non-diagonal matrix elements b_{kl} become zero.

Discussion of Jacobi's method for eigenvalues

The main idea is thus to reduce systematically the norm of the off-diagonal matrix elements of a matrix $\mathbf{A}$

$$\mathrm{off}(\mathbf{A}) = \sqrt{\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} a_{ij}^2}.$$

To demonstrate the algorithm, we consider the simple 2×2 similarity transformation of the full matrix. The matrix is symmetric, we single out $1 \leq k < l \leq n$ and use the abbreviations $c = \cos\theta$ and $s = \sin\theta$ to obtain

$$\begin{pmatrix} b_{kk} & 0 \\ 0 & b_{ll} \end{pmatrix} = \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} a_{kk} & a_{kl} \\ a_{lk} & a_{ll} \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix}.$$

Discussion of Jacobi's method for eigenvalues

We require that the non-diagonal matrix elements $b_{kl} = b_{lk} = 0$, implying that

$$a_{kl}(c^2 - s^2) + (a_{kk} - a_{ll})cs = b_{kl} = 0.$$

If $a_{kl} = 0$ one sees immediately that $\cos\theta = 1$ and $\sin\theta = 0$.

Discussion of Jacobi's method for eigenvalues

The Frobenius norm of an orthogonal transformation is always preserved. The Frobenius norm is defined as

$$\mathrm{norm}(\mathbf{A})_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}|^2}.$$

This means that for our 2×2 case we have

$$2a_{kl}^2 + a_{kk}^2 + a_{ll}^2 = b_{kk}^2 + b_{ll}^2,$$

which leads to

$$\mathrm{off}(\mathbf{B})^2 = \mathrm{norm}(\mathbf{B})_F^2 - \sum_{i=1}^{n} b_{ii}^2 = \mathrm{off}(\mathbf{A})^2 - 2a_{kl}^2,$$

since

$$\mathrm{norm}(\mathbf{B})_F^2 - \sum_{i=1}^{n} b_{ii}^2 = \mathrm{norm}(\mathbf{A})_F^2 - \sum_{i=1}^{n} a_{ii}^2 + (a_{kk}^2 + a_{ll}^2 - b_{kk}^2 - b_{ll}^2).$$

This results means that the matrix $\mathbf{A}$ moves closer to diagonal form for each transformation.

Discussion of Jacobi's method for eigenvalues

Defining the quantities $\tan\theta = t = s/c$ and

$$\cot 2\theta = \tau = \frac{a_{ll} - a_{kk}}{2a_{kl}},$$

we obtain the quadratic equation (using $\cot 2\theta = 1/2(\cot\theta - \tan\theta)$

$$t^2 + 2\tau t - 1 = 0,$$

resulting in

$$t = -\tau \pm \sqrt{1+\tau^2},$$

and c and s are easily obtained via

$$c = \frac{1}{\sqrt{1+t^2}},$$

and $s = tc$. Convince yourself that we have $|\theta| \leq \pi/4$. This has the effect of minimizing the difference between the matrices $\mathbf{B}$ and $\mathbf{A}$ since

$$\mathrm{norm}(\mathbf{B} - \mathbf{A})_F^2 = 4(1-c) \sum_{i=1, i\neq k,l}^{n} (a_{ik}^2 + a_{il}^2) + \frac{2a_{kl}^2}{c^2}.$$

Discussion of Jacobi's method for eigenvalues

- Choose a tolerance ϵ, making it a small number, typically 10^{-8} or smaller.
- Setup a *while* test where one compares the norm of the newly computed off-diagonal matrix elements

$$\mathrm{off}(\mathbf{A}) = \sqrt{\sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} a_{ij}^2} > \epsilon.$$

- Now choose the matrix elements a_{kl} so that we have those with largest value, that is $|a_{kl}| = \max_{i \neq j} |a_{ij}|$.
- Compute thereafter $\tau = (a_{ll} - a_{kk})/2a_{kl}$, $\tan\theta$, $\cos\theta$ and $\sin\theta$.
- Compute thereafter the similarity transformation for this set of values (k, l), obtaining the new matrix $\mathbf{B} = \mathbf{S}(k, l, \theta)^T \mathbf{A}\mathbf{S}(k, l, \theta)$.
- Compute the new norm of the off-diagonal matrix elements and continue till you have satisfied $\mathrm{off}(\mathbf{B}) \leq \epsilon$

Discussion of Jacobi's method for eigenvalues

The convergence rate of the Jacobi method is however poor, one needs typically $3n^2 - 5n^2$ rotations and each rotation requires $4n$ operations, resulting in a total of $12n^3 - 20n^3$ operations in order to zero out non-diagonal matrix elements.

Discussion of Jacobi's method for eigenvalues

We specialize to a symmetric 3×3 matrix $\mathbf{A}$. We start the process as follows (assuming that $a_{23} = a_{32}$ is the largest non-diagonal) with $c = \cos\theta$ and $s = \sin\theta$

$$\mathbf{B} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{pmatrix}.$$

We will choose the angle θ in order to have $a_{23} = a_{32} = 0$. We get (symmetric matrix)

$$\mathbf{B} = \begin{pmatrix} a_{11} & a_{12}c - a_{13}s & a_{12}s + a_{13}c \\ a_{12}c - a_{13}s & a_{22}c^2 + a_{33}s^2 - 2a_{23}sc & (a_{22} - a_{33})sc + a_{23}(c^2 - s^2) \\ a_{12}s + a_{13}c & (a_{22} - a_{33})sc + a_{23}(c^2 - s^2) & a_{22}s^2 + a_{33}c^2 + 2a_{23}sc \end{pmatrix}.$$

Note that a_{11} is unchanged! As it should.

Discussion of Jacobi's method for eigenvalues

We have

$$\mathbf{B} = \begin{pmatrix} a_{11} & a_{12}c - a_{13}s & a_{12}s + a_{13}c \\ a_{12}c - a_{13}s & a_{22}c^2 + a_{33}s^2 - 2a_{23}sc & (a_{22} - a_{33})sc + a_{23}(c^2 - s^2) \\ a_{12}s + a_{13}c & (a_{22} - a_{33})sc + a_{23}(c^2 - s^2) & a_{22}s^2 + a_{33}c^2 + 2a_{23}sc \end{pmatrix}.$$

or

$$
\begin{aligned}
b_{11} =& a_{11} \\
b_{12} =& a_{12}\cos\theta - a_{13}\sin\theta, 1 \neq 2, 1 \neq 3 \\
b_{13} =& a_{13}\cos\theta + a_{12}\sin\theta, 1 \neq 2, 1 \neq 3 \\
b_{22} =& a_{22}\cos^2\theta - 2a_{23}\cos\theta\sin\theta + a_{33}\sin^2\theta \\
b_{33} =& a_{33}\cos^2\theta + 2a_{23}\cos\theta\sin\theta + a_{22}\sin^2\theta \\
b_{23} =& (a_{22} - a_{33})\cos\theta\sin\theta + a_{23}(\cos^2\theta - \sin^2\theta)
\end{aligned}
$$

We will fix the angle θ so that $b_{23} = 0$.

Discussion of Jacobi's method for eigenvalues

We get then a new matrix

$$
\mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{12} & b_{22} & 0 \\ b_{13} & 0 & a_{33} \end{pmatrix}.
$$

We repeat then assuming that b_{12} is the largest non-diagonal matrix element and get a new matrix

$$
\mathbf{C} = \begin{pmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{12} & b_{22} & 0 \\ b_{13} & 0 & b_{33} \end{pmatrix} \begin{pmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{pmatrix}.
$$

We continue this process till all non-diagonal matrix elements are zero (ideally). You will notice that performing the above operations that the matrix element b_{23} which was previous zero becomes different from zero. This is one of the problems which slows down the jacobi procedure.

Discussion of Jacobi's method for eigenvalues

The more general expression for the new matrix elements are

$$
\begin{aligned}
b_{ii} =& a_{ii}, i \neq k, i \neq l \\
b_{ik} =& a_{ik}\cos\theta - a_{il}\sin\theta, i \neq k, i \neq l \\
b_{il} =& a_{il}\cos\theta + a_{ik}\sin\theta, i \neq k, i \neq l \\
b_{kk} =& a_{kk}\cos^2\theta - 2a_{kl}\cos\theta\sin\theta + a_{ll}\sin^2\theta \\
b_{ll} =& a_{ll}\cos^2\theta + 2a_{kl}\cos\theta\sin\theta + a_{kk}\sin^2\theta \\
b_{kl} =& (a_{kk} - a_{ll})\cos\theta\sin\theta + a_{kl}(\cos^2\theta - \sin^2\theta)
\end{aligned}
$$

This is what we will need to code.

Discussion of Jacobi's method for eigenvalues

Code example.

```
//  we have defined a matrix A and a matrix R for the eigenvector, both of dim n x n
//  The final matrix R has the eigenvectors in its row elements, it is set to one
//  for the diagonal elements in the beginning, zero else.
....
double tolerance = 1.0E-10;
int iterations = 0;
while ( maxnondiag > tolerance && iterations <= maxiter)
{
   int p, q;
   maxnondiag  = offdiag(A, p, q, n);
   Jacobi_rotate(A, R, p, q, n);
   iterations++;
}
...
```

Discussion of Jacobi's method for eigenvalues

Finding the max nondiagonal element

```
//  the offdiag function, using Armadillo
double offdiag(mat A, int p, int q, int n);
{
   double max;
   for (int i = 0; i < n; ++i)
   {
       for ( int j = i+1; j < n; ++j)
       {
           double aij = fabs(A(i,j));
           if ( aij > max)
           {
              max = aij;  p = i; q = j;
           }
       }
   }
   return max;
}
// more statements
```

Discussion of Jacobi's method for eigenvalues

Finding the new matrix elements

```
void Jacobi_rotate ( mat A, mat R, int k, int l, int n )
{
  double s, c;
  if ( A(k,l) != 0.0 ) {
    double t, tau;
    tau = (A(l,l) - A(k,k))/(2*A(k,l));

    if ( tau >= 0 ) {
      t = 1.0/(tau + sqrt(1.0 + tau*tau));
    } else {
      t = -1.0/(-tau +sqrt(1.0 + tau*tau));
    }
```

```
      c = 1/sqrt(1+t*t);
      s = c*t;
    } else {
      c = 1.0;
      s = 0.0;
    }
    double a_kk, a_ll, a_ik, a_il, r_ik, r_il;
    a_kk = A(k,k);
    a_ll = A(l,l);
    A(k,k) = c*c*a_kk - 2.0*c*s*A(k,l) + s*s*a_ll;
    A(l,l) = s*s*a_kk + 2.0*c*s*A(k,l) + c*c*a_ll;
    A(k,l) = 0.0;  // hard-coding non-diagonal elements by hand
    A(l,k) = 0.0;  // same here
    for ( int i = 0; i < n; i++ ) {
      if ( i != k && i != l ) {
        a_ik = A(i,k);
        a_il = A(i,l);
        A(i,k) = c*a_ik - s*a_il;
        A(k,i) = A(i,k);
        A(i,l) = c*a_il + s*a_ik;
        A(l,i) = A(i,l);
      }
//  And finally the new eigenvectors
      r_ik = R(i,k);
      r_il = R(i,l);

      R(i,k) = c*r_ik - s*r_il;
      R(i,l) = c*r_il + s*r_ik;
    }
    return;
  } // end of function jacobi_rotate
```

Discussion of numerical project for two electrons

We can write our original differential equation in terms of a discretized equation with approximations to the derivatives as

$$-\frac{u_{i+1}-2u_i+u_{i-i}}{h^2} = f(x_i,u(x_i)),$$

with $i = 1, 2, \ldots, n$. We need to add to this system the two boundary conditions $u(a) = u_0$ and $u(b) = u_{n+1}$. If we define a matrix

$$\mathbf{A} = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & \ldots & \ldots & \ldots & \ldots & \ldots \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix}$$

and the corresponding vectors $\mathbf{u} = (u_1, u_2, \ldots, u_n)^T$ and $\mathbf{f}(\mathbf{u}) = f(x_1, x_2, \ldots, x_n, u_1, u_2, \ldots, u_n)^T$ we can rewrite the differential equation including the boundary conditions as a system of linear equations with a large number of unknowns

$$\mathbf{Au} = \mathbf{f}(\mathbf{u}).$$

Discussion of numerical project

We are first interested in the solution of the radial part of Schroedinger's equation for one electron. This equation reads

$$-\frac{\hbar^2}{2m}\left(\frac{1}{r^2}\frac{d}{dr}r^2\frac{d}{dr}-\frac{l(l+1)}{r^2}\right)R(r)+V(r)R(r)=ER(r).$$

In our case $V(r)$ is the harmonic oscillator potential $(1/2)kr^2$ with $k=m\omega^2$ and E is the energy of the harmonic oscillator in three dimensions. The oscillator frequency is ω and the energies are

$$E_{nl}=\hbar\omega\left(2n+l+\frac{3}{2}\right),$$

with $n=0,1,2,\dots$ and $l=0,1,2,\dots$.

Discussion of numerical project

Since we have made a transformation to spherical coordinates it means that $r\in[0,\infty)$. The quantum number l is the orbital momentum of the electron. Then we substitute $R(r)=(1/r)u(r)$ and obtain

$$-\frac{\hbar^2}{2m}\frac{d^2}{dr^2}u(r)+\left(V(r)+\frac{l(l+1)}{r^2}\frac{\hbar^2}{2m}\right)u(r)=Eu(r).$$

The boundary conditions are $u(0)=0$ and $u(\infty)=0$.

Discussion of numerical project

We introduce a dimensionless variable $\rho=(1/\alpha)r$ where α is a constant with dimension length and get

$$-\frac{\hbar^2}{2m\alpha^2}\frac{d^2}{d\rho^2}u(\rho)+\left(V(\rho)+\frac{l(l+1)}{\rho^2}\frac{\hbar^2}{2m\alpha^2}\right)u(\rho)=Eu(\rho).$$

In numerical project we choose $l=0$. Inserting $V(\rho)=(1/2)k\alpha^2\rho^2$ we end up with

$$-\frac{\hbar^2}{2m\alpha^2}\frac{d^2}{d\rho^2}u(\rho)+\frac{k}{2}\alpha^2\rho^2u(\rho)=Eu(\rho).$$

We multiply thereafter with $2m\alpha^2/\hbar^2$ on both sides and obtain

$$-\frac{d^2}{d\rho^2}u(\rho)+\frac{mk}{\hbar^2}\alpha^4\rho^2u(\rho)=\frac{2m\alpha^2}{\hbar^2}Eu(\rho).$$

Discussion of numerical project

We have thus

$$-\frac{d^2}{d\rho^2}u(\rho)+\frac{mk}{\hbar^2}\alpha^4\rho^2u(\rho)=\frac{2m\alpha^2}{\hbar^2}Eu(\rho).$$

The constant α can now be fixed so that

$$\frac{mk}{\hbar^2}\alpha^4=1,$$

or

$$\alpha=\left(\frac{\hbar^2}{mk}\right)^{1/4}.$$

Defining

$$\lambda=\frac{2m\alpha^2}{\hbar^2}E,$$

we can rewrite Schroedinger's equation as

$$-\frac{d^2}{d\rho^2}u(\rho)+\rho^2u(\rho)=\lambda u(\rho).$$

This is the first equation to solve numerically. In three dimensions the eigenvalues for $l=0$ are $\lambda_0=3,\lambda_1=7,\lambda_2=11,\dots$.

Discussion of numerical project

We use the by now standard expression for the second derivative of a function u

$$u''=\frac{u(\rho+h)-2u(\rho)+u(\rho-h)}{h^2}+O(h^2), \tag{3}$$

where h is our step. Next we define minimum and maximum values for the variable ρ, $\rho_{\rm min}=0$ and $\rho_{\rm max}$, respectively. You need to check your results for the energies against different values $\rho_{\rm max}$, since we cannot set $\rho_{\rm max}=\infty$.

Discussion of numerical project

With a given number of steps, $n_{\rm step}$, we then define the step h as

$$h=\frac{\rho_{\rm max}-\rho_{\rm min}}{n_{\rm step}}.$$

Define an arbitrary value of ρ as

$$\rho_i=\rho_{\rm min}+ih \qquad i=0,1,2,\dots,n_{\rm step}$$

we can rewrite the Schrödinger equation for ρ_i as

$$-\frac{u(\rho_i+h)-2u(\rho_i)+u(\rho_i-h)}{h^2}+\rho_i^2u(\rho_i)=\lambda u(\rho_i),$$

or in a more compact way

$$-\frac{u_{i+1}-2u_i+u_{i-1}}{h^2}+\rho_i^2u_i=-\frac{u_{i+1}-2u_i+u_{i-1}}{h^2}+V_iu_i=\lambda u_i,$$

where $V_i=\rho_i^2$ is the harmonic oscillator potential.

Discussion of numerical project

Define first the diagonal matrix element

$$d_i=\frac{2}{h^2}+V_i,$$

and the non-diagonal matrix element

$$e_i=-\frac{1}{h^2}.$$

In this case the non-diagonal matrix elements are given by a mere constant. *All non-diagonal matrix elements are equal.*

With these definitions the Schroedinger equation takes the following form

$$d_iu_i+e_{i-1}u_{i-1}+e_{i+1}u_{i+1}=\lambda u_i,$$

where u_i is unknown. We can write the latter equation as a matrix eigenvalue problem

$$\begin{pmatrix} d_1 & e_1 & 0 & 0 & \dots & 0 & 0 \\ e_1 & d_2 & e_2 & 0 & \dots & 0 & 0 \\ 0 & e_2 & d_3 & e_3 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & d_{n_{\rm step}-2} & e_{n_{\rm step}-1} \\ 0 & \dots & \dots & \dots & \dots & e_{n_{\rm step}-1} & d_{n_{\rm step}-1} \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ \dots \\ \dots \\ \dots \\ u_{n_{\rm step}-1} \end{pmatrix}=\lambda\begin{pmatrix} u_1 \\ u_2 \\ \dots \\ \dots \\ \dots \\ u_{n_{\rm step}-1} \end{pmatrix} \tag{4}$$

or if we wish to be more detailed, we can write the tridiagonal matrix as

$$\begin{pmatrix} \frac{2}{h^2}+V_1 & -\frac{1}{h^2} & 0 & 0 & \dots & 0 & 0 \\ -\frac{1}{h^2} & \frac{2}{h^2}+V_2 & -\frac{1}{h^2} & 0 & \dots & 0 & 0 \\ 0 & -\frac{1}{h^2} & \frac{2}{h^2}+V_3 & -\frac{1}{h^2} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & \frac{2}{h^2}+V_{n_{\rm step}-2} & -\frac{1}{h^2} \\ 0 & \dots & \dots & \dots & \dots & -\frac{1}{h^2} & \frac{2}{h^2}+V_{n_{\rm step}-1} \end{pmatrix} \tag{5}$$

Recall that the solutions are known via the boundary conditions at $i=n_{\rm step}$ and at the other end point, that is for ρ_0. The solution is zero in both cases.

Discussion of numerical project

We are going to study two electrons in a harmonic oscillator well which also interact via a repulsive Coulomb interaction. Let us start with the single-electron equation written as

$$-\frac{\hbar^2}{2m}\frac{d^2}{dr^2}u(r)+\frac{1}{2}kr^2u(r)=E^{(1)}u(r),$$

where $E^{(1)}$ stands for the energy with one electron only. For two electrons with no repulsive Coulomb interaction, we have the following Schroedinger equation

$$\left(-\frac{\hbar^2}{2m}\frac{d^2}{dr_1^2}-\frac{\hbar^2}{2m}\frac{d^2}{dr_2^2}+\frac{1}{2}kr_1^2+\frac{1}{2}kr_2^2\right)u(r_1,r_2)=E^{(2)}u(r_1,r_2).$$

Discussion of numerical project

Note that we deal with a two-electron wave function $u(r_1,r_2)$ and two-electron energy $E^{(2)}$.

With no interaction this can be written out as the product of two single-electron wave functions, that is we have a solution on closed form.

We introduce the relative coordinate $\mathbf{r}=\mathbf{r}_1-\mathbf{r}_2$ and the center-of-mass coordinate $\mathbf{R}=1/2(\mathbf{r}_1+\mathbf{r}_2)$. With these new coordinates, the radial Schroedinger equation reads

$$\left(-\frac{\hbar^2}{m}\frac{d^2}{dr^2}-\frac{\hbar^2}{4m}\frac{d^2}{dR^2}+\frac{1}{4}kr^2+kR^2\right)u(r,R)=E^{(2)}u(r,R).$$

Discussion of numerical project

The equations for r and R can be separated via the ansatz for the wave function $u(r,R)=\psi(r)\phi(R)$ and the energy is given by the sum of the relative energy E_r and the center-of-mass energy E_R, that is

$$E^{(2)}=E_r+E_R.$$

We add then the repulsive Coulomb interaction between two electrons, namely a term

$$V(r_1,r_2)=\frac{\beta e^2}{|\mathbf{r}_1-\mathbf{r}_2|}=\frac{\beta e^2}{r},$$

with $\beta e^2=1.44$ eVnm.

Discussion of numerical project

Adding this term, the r-dependent Schroedinger equation becomes

$$\left(-\frac{\hbar^2}{m}\frac{d^2}{dr^2}+\frac{1}{4}kr^2+\frac{\beta e^2}{r}\right)\psi(r)=E_r\psi(r).$$

This equation is similar to the one we had previously in parts (a) and (b) and we introduce again a dimensionless variable $\rho = r/\alpha$. Repeating the same steps, we arrive at

$$-\frac{d^2}{d\rho^2}\psi(\rho) + \frac{mk}{4\hbar^2}\alpha^4\rho^2\psi(\rho) + \frac{m\alpha\beta e^2}{\rho\hbar^2}\psi(\rho) = \frac{m\alpha^2}{\hbar^2}E_r\psi(\rho).$$

Discussion of numerical project

We want to manipulate this equation further to make it as similar to that in (a) as possible. We define a 'frequency'

$$\omega_r^2 = \frac{1}{4}\frac{mk}{\hbar^2}\alpha^4,$$

and fix the constant α by requiring

$$\frac{m\alpha\beta e^2}{\hbar^2} = 1$$

or

$$\alpha = \frac{\hbar^2}{m\beta e^2}.$$

Discussion of numerical project

Defining

$$\lambda = \frac{m\alpha^2}{\hbar^2}E,$$

we can rewrite Schroedinger's equation as

$$-\frac{d^2}{d\rho^2}\psi(\rho) + \omega_r^2\rho^2\psi(\rho) + \frac{1}{\rho}\psi(\rho) = \lambda\psi(\rho).$$

Discussion of numerical project

We treat ω_r as a parameter which reflects the strength of the oscillator potential.

Here we will study the cases $\omega_r = 0.01$, $\omega_r = 0.5$, $\omega_r = 1$, and $\omega_r = 5$ for the ground state only, that is the lowest-lying state.

Discussion of numerical project

With no repulsive Coulomb interaction you should get a result which corresponds to the relative energy of a non-interacting system. Make sure your results are stable as functions of $\rho_{\max}$ and the number of steps.

We are only interested in the ground state with $l = 0$. We omit the center-of-mass energy.

For specific oscillator frequencies, the above equation has analytic answers, see the article by M. Taut, Phys. Rev. A 48, 3561 - 3566 (1993). The article can be retrieved from the following web address http://prola.aps.org/abstract/PRA/v48/i5/p3561_1.

Discussion of numerical project, simple program for one particle in a harmonic oscillator trap

The following program uses the eigenvalue solver provided by Armadillo and returns the eigenvalues for the lowest states. You can run this code interactively if you use ipython notebook. To install armadillo, please go back to the introduction slides.

```
/*
  Solves the one-particle Schrodinger equation
  for a potential specified in function
  potential(). This example is for the harmonic oscillator in 3d
*/
#include <cmath>
#include <iostream>
#include <fstream>
#include <iomanip>
#include <armadillo>

using namespace  std;
using namespace  arma;

double potential(double);
void output(double, double, int, vec& );


// Begin of main program

int main(int argc, char* argv[])
{
  int       i, j, Dim, lOrbital;
  double    RMin, RMax, Step, DiagConst, NondiagConst, OrbitalFactor;
  // With spherical coordinates RMin = 0 always
  RMin = 0.0;

  RMax = 8.0;  lOrbital = 0;  Dim =2000;
  mat Hamiltonian = zeros<mat>(Dim,Dim);
  // Integration step length
  Step    = RMax/ Dim;
  DiagConst = 2.0 / (Step*Step);
  NondiagConst =  -1.0 / (Step*Step);
  OrbitalFactor = lOrbital * (lOrbital + 1.0);

  // local memory for r and the potential w[r]
  vec r(Dim); vec w(Dim);
  for(i = 0; i < Dim; i++) {
    r(i) = RMin + (i+1) * Step;
    w(i) = potential(r(i)) + OrbitalFactor/(r(i) * r(i));
  }


  // Setting up tridiagonal matrix and brute diagonalization using Armadillo
  Hamiltonian(0,0) = DiagConst + w(0);
  Hamiltonian(0,1) = NondiagConst;
  for(i = 1; i < Dim-1; i++) {
    Hamiltonian(i,i-1)    = NondiagConst;
    Hamiltonian(i,i)    = DiagConst + w(i);
    Hamiltonian(i,i+1)    = NondiagConst;
  }
```

```
  Hamiltonian(Dim-1,Dim-2) = NondiagConst;
  Hamiltonian(Dim-1,Dim-1) = DiagConst + w(Dim-1);
  // diagonalize and obtain eigenvalues
  vec Eigval(Dim);
  eig_sym(Eigval, Hamiltonian);
  output(RMin , RMax, Dim, Eigval);

  return 0;
}  //  end of main function


/*
  The function potential()
  calculates and return the value of the
  potential for a given argument x.
  The potential here is for the hydrogen atom
*/

double potential(double x)
{
  return x*x;

} // End: function potential()


void output(double RMin , double RMax, int Dim, vec& d)
{
  int i;
  cout << "RESULTS:" << endl;
  cout << setiosflags(ios::showpoint | ios::uppercase);
  cout <<"Rmin = " << setw(15) << setprecision(8) << RMin << endl;
  cout <<"Rmax = " << setw(15) << setprecision(8) << RMax << endl;
  cout <<"Number of steps = " << setw(15) << Dim << endl;
  cout << "Five lowest eigenvalues:" << endl;
  for(i = 0; i < 5; i++) {
    cout << setw(15) << setprecision(8) << d[i] << endl;
  }
}  // end of function output
```

The corresponding Python program

The code sets up the Hamiltonian matrix by defining the the minimun and maximum values of r with a maximum value of integration points. These are set in the initialization function. It plots the eigenfunctions of the three lowest eigenstates.

```
#Program which solves the one-particle Schrodinger equation
#for a potential specified in function
#potential(). This example is for the harmonic oscillator in 3d

from  matplotlib import pyplot as plt
import numpy as np
#Function for initialization of parameters
def initialize():
    RMin = 0.0
    RMax = 10.0
    lOrbital = 0
    Dim = 400
    return RMin, RMax, lOrbital, Dim
```

```
# Here we set up the harmonic oscillator potential
def potential(r):
    return r*r

#Get the boundary, orbital momentum and number of integration points
RMin, RMax, lOrbital, Dim = initialize()

#Initialize constants
Step    = RMax/(Dim+1)
DiagConst = 2.0 / (Step*Step)
NondiagConst =  -1.0 / (Step*Step)
OrbitalFactor = lOrbital * (lOrbital + 1.0)

#Calculate array of potential values
v = np.zeros(Dim)
r = np.linspace(RMin,RMax,Dim)
for i in xrange(Dim):
    r[i] = RMin + (i+1) * Step;
    v[i] = potential(r[i]) + OrbitalFactor/(r[i]*r[i]);

#Setting up a tridiagonal matrix and finding eigenvectors and eigenvalues
Hamiltonian = np.zeros((Dim,Dim))
Hamiltonian[0,0] = DiagConst + v[0];
Hamiltonian[0,1] = NondiagConst;
for i in xrange(1,Dim-1):
    Hamiltonian[i,i-1]  = NondiagConst;
    Hamiltonian[i,i]    = DiagConst + v[i];
    Hamiltonian[i,i+1]  = NondiagConst;
Hamiltonian[Dim-1,Dim-2] = NondiagConst;
Hamiltonian[Dim-1,Dim-1] = DiagConst + v[Dim-1];
# diagonalize and obtain eigenvalues, not necessarily sorted
EigValues, EigVectors = np.linalg.eig(Hamiltonian)
# sort eigenvectors and eigenvalues
permute = EigValues.argsort()
EigValues = EigValues[permute]
EigVectors = EigVectors[:,permute]
# now plot the results for the three lowest lying eigenstates
for i in xrange(3):
    print EigValues[i]
FirstEigvector = EigVectors[:,0]
SecondEigvector = EigVectors[:,1]
ThirdEigvector = EigVectors[:,2]
plt.plot(r, FirstEigvector**2 ,'b-',r, SecondEigvector**2 ,'g-',r, ThirdEigvector**2 ,'r-')
plt.axis([0,4.6,0.0, 0.025])
plt.xlabel(r'$r$')
plt.ylabel(r'Radial probability $r^2|R(r)|^2$')
plt.title(r'Radial probability distributions for three lowest-lying states')
plt.savefig('eigenvector.pdf')
plt.show()
```

The corresponding Python program for the interactive case

```
#Program which solves the one-particle Schrodinger equation
#for a potential specified in function
#potential(). This example is for the harmonic oscillator in 3d with a repulsive Coulomb interact

from  matplotlib import pyplot as plt
import numpy as np
#Function for initialization of parameters
def initialize():
```

```
    RMin = 0.0
    RMax = 10.0
    lOrbital = 0
    Dim = 600
    omega = 1.0
    return RMin, RMax, lOrbital, Dim, omega
# Here we set up the harmonic oscillator potential
def potential(r):
    return r*r
# Here we set up the harmonic oscillator potential with interaction
def intpotential(r,omega):
    return omega*omega*r*r+1.0/r

#Get the boundary, orbital momentum and number of integration points
RMin, RMax, lOrbital, Dim, omega = initialize()

#Initialize constants
Step    = RMax/(Dim+1)
DiagConst = 2.0 / (Step*Step)
NondiagConst =  -1.0 / (Step*Step)
OrbitalFactor = lOrbital * (lOrbital + 1.0)

#Calculate array of potential values
v = np.zeros(Dim)
vint = np.zeros(Dim)
r = np.linspace(RMin,RMax,Dim)
for i in xrange(Dim):
    r[i] = RMin + (i+1) * Step;
    v[i] = potential(r[i]) + OrbitalFactor/(r[i]*r[i]);
    vint[i] = intpotential(r[i],omega) + OrbitalFactor/(r[i]*r[i]);

#Setting up a tridiagonal matrix and finding eigenvectors and eigenvalues
Hamiltonian = np.zeros((Dim,Dim))
IntHamiltonian = np.zeros((Dim,Dim))
Hamiltonian[0,0] = DiagConst + v[0];
Hamiltonian[0,1] = NondiagConst;
IntHamiltonian[0,0] = DiagConst + vint[0];
IntHamiltonian[0,1] = NondiagConst;
for i in xrange(1,Dim-1):
    Hamiltonian[i,i-1]  = NondiagConst;
    Hamiltonian[i,i]    = DiagConst + v[i];
    Hamiltonian[i,i+1]  = NondiagConst;
    IntHamiltonian[i,i-1]  = NondiagConst;
    IntHamiltonian[i,i]    = DiagConst + vint[i];
    IntHamiltonian[i,i+1]  = NondiagConst;

Hamiltonian[Dim-1,Dim-2] = NondiagConst;
Hamiltonian[Dim-1,Dim-1] = DiagConst + v[Dim-1];
IntHamiltonian[Dim-1,Dim-2] = NondiagConst;
IntHamiltonian[Dim-1,Dim-1] = DiagConst + vint[Dim-1];
# diagonalize and obtain eigenvalues, not necessarily sorted
EigValues, EigVectors = np.linalg.eig(Hamiltonian)
EigValuesInt, EigVectorsInt = np.linalg.eig(IntHamiltonian)
# sort eigenvectors and eigenvalues
permute = EigValues.argsort()
EigValues = EigValues[permute]
EigVectors = EigVectors[:,permute]
permute = EigValuesInt.argsort()
EigValuesInt = EigValuesInt[permute]
EigVectorsInt = EigVectorsInt[:,permute]
# now plot the results for the three lowest lying eigenstates
```

```
for i in xrange(1):
    print EigValues[i], EigValuesInt[i],
FirstEigvector = EigVectors[:,0]
FirstEigvectorInt = EigVectorsInt[:,0]
plt.plot(r, FirstEigvector**2 ,'b-',r, FirstEigvectorInt**2 ,'g-')
plt.axis([0,3.0,0.0, 0.015])
plt.xlabel(r'$r$')
plt.ylabel(r'Radial probability $r^2|R(r)|^2$')
plt.title(r'Radial probability distributions for three lowest-lying states')
plt.savefig('eigenvectorint.pdf')
plt.show()
```

Discussion of Householder's method for eigenvalues

The drawbacks with Jacobi's method are rather obvious, with perhaps the most negative feature being the fact that we cannot tell * a priori* how many transformations are needed. Can we do better? The answer to this is yes and is given by a clever algorithm outlined by Householder. It was ranked among the top ten algorithms in the previous century. We will discuss this algorithm in more detail below.

The first step consists in finding an orthogonal matrix $\mathbf{S}$ which is the product of $(n-2)$ orthogonal matrices

$$\mathbf{S} = \mathbf{S}_1\mathbf{S}_2\dots\mathbf{S}_{n-2},$$

each of which successively transforms one row and one column of $\mathbf{A}$ into the required tridiagonal form. Only $n-2$ transformations are required, since the last two elements are already in tridiagonal form.

Discussion of Householder's method for eigenvalues

In order to determine each $\mathbf{S}_i$ let us see what happens after the first multiplication, namely,

$$\mathbf{S}_1^T\mathbf{A}\mathbf{S}_1 = \begin{pmatrix} a_{11} & e_1 & 0 & 0 & \dots & 0 & 0 \\ e_1 & a'_{22} & a'_{23} & \dots & \dots & \dots & a'_{2n} \\ 0 & a'_{32} & a'_{33} & \dots & \dots & \dots & a'_{3n} \\ 0 & \dots & \dots & \dots & \dots & \dots & \\ 0 & a'_{n2} & a'_{n3} & \dots & \dots & \dots & a'_{nn} \end{pmatrix}$$

where the primed quantities represent a matrix $\mathbf{A}'$ of dimension $n-1$ which will subsequentely be transformed by $\mathbf{S}_2$.

Discussion of Householder's method for eigenvalues

The factor e_1 is a possibly non-vanishing element. The next transformation produced by $\mathbf{S}_2$ has the same effect as $\mathbf{S}s$ but now on the submatirx $\mathbf{A}^{'}$ only

$$(\mathbf{S}_1\mathbf{S}_2)^T\,\mathbf{A}\mathbf{S}_1\mathbf{S}_2 = \begin{pmatrix} a_{11} & e_1 & 0 & 0 & \dots & 0 & 0 \\ e_1 & a'_{22} & e_2 & 0 & \dots & \dots & 0 \\ 0 & e_2 & a''_{33} & \dots & \dots & \dots & a''_{3n} \\ 0 & \dots & \dots & \dots & \dots & \dots & \\ 0 & 0 & a''_{n3} & \dots & \dots & \dots & a''_{nn} \end{pmatrix}$$

Note that the effective size of the matrix on which we apply the transformation reduces for every new step. In the previous Jacobi method each similarity transformation is in principle performed on the full size of the original matrix.

Discussion of Householder's method for eigenvalues

After a series of such transformations, we end with a set of diagonal matrix elements

$$a_{11}, a'_{22}, a''_{33} \dots a^{n-1}_{nn},$$

and off-diagonal matrix elements

$$e_1, e_2, e_3, \dots, e_{n-1}.$$

Discussion of Householder's method for eigenvalues

The resulting matrix reads

$$\mathbf{S}^T\mathbf{A}\mathbf{S} = \begin{pmatrix} a_{11} & e_1 & 0 & 0 & \dots & 0 & 0 \\ e_1 & a'_{22} & e_2 & 0 & \dots & 0 & 0 \\ 0 & e_2 & a''_{33} & e_3 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & a^{(n-1)}_{n-2} & e_{n-1} \\ 0 & \dots & \dots & \dots & \dots & e_{n-1} & a^{(n-1)}_{nn} \end{pmatrix}.$$

Discussion of Householder's method for eigenvalues

It remains to find a recipe for determining the transformation $\mathbf{S}_n$. We illustrate the method for $\mathbf{S}_1$ which we assume takes the form

$$\mathbf{S_1} = \begin{pmatrix} 1 & \mathbf{0^T} \\ \mathbf{0} & \mathbf{P} \end{pmatrix},$$

with $\mathbf{0^T}$ being a zero row vector, $\mathbf{0^T} = \{0, 0, \cdots\}$ of dimension $(n-1)$. The matrix $\mathbf{P}$ is symmetric with dimension $((n-1)\times(n-1))$ satisfying $\mathbf{P}^2 = \mathbf{I}$ and $\mathbf{P}^T = \mathbf{P}$. A possible choice which fullfils the latter two requirements is

$$\mathbf{P} = \mathbf{I} - 2\mathbf{u}\mathbf{u}^T,$$

where $\mathbf{I}$ is the $(n-1)$ unity matrix and $\mathbf{u}$ is an $n-1$ column vector with norm $\mathbf{u}^T\mathbf{u}$ (inner product).

Discussion of Householder's method for eigenvalues

Note that $\mathbf{uu}^T$ is an outer product giving a matrix of dimension $((n-1)\times(n-1))$. Each matrix element of $\mathbf{P}$ then reads

$$P_{ij} = \delta_{ij} - 2u_i u_j,$$

where i and j range from 1 to $n-1$. Applying the transformation $\mathbf{S}_1$ results in

$$\mathbf{S}_1^T\mathbf{A}\mathbf{S}_1 = \begin{pmatrix} a_{11} & (\mathbf{Pv})^T \\ \mathbf{Pv} & \mathbf{A}' \end{pmatrix},$$

where $\mathbf{v^T} = \{a_{21}, a_{31}, \cdots, a_{n1}\}$ and $\mathbf{P}$s must satisfy $(\mathbf{Pv})^T = \{k, 0, 0, \cdots\}$. Then

$$\mathbf{Pv} = \mathbf{v} - 2\mathbf{u}(\mathbf{u}^T\mathbf{v}) = k\mathbf{e}, \tag{6}$$

with $\mathbf{e^T} = \{1, 0, 0, \ldots 0\}$.

Discussion of Householder's method for eigenvalues

Solving the latter equation gives us $\mathbf{u}$ and thus the needed transformation $\mathbf{P}$. We do first however need to compute the scalar k by taking the scalar product of the last equation with its transpose and using the fact that $\mathbf{P}^2 = \mathbf{I}$. We get then

$$(\mathbf{Pv})^T\mathbf{Pv} = k^2 = \mathbf{v}^T\mathbf{v} = |v|^2 = \sum_{i=2}^{n} a_{i1}^2,$$

which determines the constant $k = \pm v$.

Discussion of Householder's method for eigenvalues

Now we can rewrite Eq. (6) as

$$\mathbf{v} - k\mathbf{e} = 2\mathbf{u}(\mathbf{u}^T\mathbf{v}),$$

and taking the scalar product of this equation with itself and obtain

$$2(\mathbf{u}^T\mathbf{v})^2 = (v^2 \pm a_{21}v), \tag{7}$$

which finally determines

$$\mathbf{u} = \frac{\mathbf{v} - k\mathbf{e}}{2(\mathbf{u}^T\mathbf{v})}.$$

In solving Eq. (7) great care has to be exercised so as to choose those values which make the right-hand largest in order to avoid loss of numerical precision. The above steps are then repeated for every transformations till we have a tridiagonal matrix suitable for obtaining the eigenvalues.

Discussion of Householder's method for eigenvalues

Our Householder transformation has given us a tridiagonal matrix. We discuss here how one can use Householder's iterative procedure to obtain the eigenvalues. Let us specialize to a 4×4 matrix. The tridiagonal matrix takes the form

$$\mathbf{A} = \begin{pmatrix} d_1 & e_1 & 0 & 0 \\ e_1 & d_2 & e_2 & 0 \\ 0 & e_2 & d_3 & e_3 \\ 0 & 0 & e_3 & d_4 \end{pmatrix}.$$

As a first observation, if any of the elements e_i are zero the matrix can be separated into smaller pieces before diagonalization. Specifically, if $e_1 = 0$ then d_1 is an eigenvalue.

Discussion of Householder's method for eigenvalues

Thus, let us introduce a transformation $\mathbf{S_1}$ which operates like

$$\mathbf{S_1} = \begin{pmatrix} \cos\theta & 0 & 0 & \sin\theta \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \cos\theta & 0 & 0 & \cos\theta \end{pmatrix}$$

Then the similarity transformation

$$\mathbf{S_1^T A S_1} = \mathbf{A'} = \begin{pmatrix} d_1' & e_1' & 0 & 0 \\ e_1' & d_2 & e_2 & 0 \\ 0 & e_2 & d_3 & e'3 \\ 0 & 0 & e_3' & d_4' \end{pmatrix}$$

produces a matrix where the primed elements in $\mathbf{A'}$ have been changed by the transformation whereas the unprimed elements are unchanged. If we now choose θ to give the element $a_{21}^{'} = e^{'} = 0$ then we have the first eigenvalue $= a_{11}^{'} = d_1^{'}$.

Discussion of Householder's method for eigenvalues

This procedure can be continued on the remaining three-dimensional submatrix for the next eigenvalue. Thus after few transformations we have the wanted diagonal form.

What we see here is just a special case of the more general procedure developed by Francis in two articles in 1961 and 1962.

The algorithm is based on the so-called QR method (or just QR-algorithm). It follows from a theorem by Schur which states that any square matrix can be written out in terms of an orthogonal matrix $\mathbf{Q}$ and an upper triangular matrix $\mathbf{U}$. Historically R was used instead of U since the wording right triangular matrix was first used. The method is based on an iterative procedure similar to Jacobi's method, by a succession of planar rotations. For a tridiagonal matrix it is simple to carry out in principle, but complicated in detail! We will discuss this in more detail during week 38.

Eigenvalues with the *QR* and Lanczos methods

Our Householder transformation has given us a tridiagonal matrix. We discuss here how one can use Jacobi's iterative procedure to obtain the eigenvalues, although it may not be the best approach. Let us specialize to a 4×4 matrix. The tridiagonal matrix takes the form

$$\mathbf{A} = \begin{pmatrix} d_1 & e_1 & 0 & 0 \\ e_1 & d_2 & e_2 & 0 \\ 0 & e_2 & d_3 & e_3 \\ 0 & 0 & e_3 & d_4 \end{pmatrix}.$$

As a first observation, if any of the elements e_i are zero the matrix can be separated into smaller pieces before diagonalization. Specifically, if $e_1 = 0$ then d_1 is an eigenvalue.

Eigenvalues with the *QR* and Lanczos methods

Thus, let us introduce a transformation $\mathbf{S_1}$ which operates like

$$\mathbf{S_1} = \begin{pmatrix} \cos\theta & 0 & 0 & \sin\theta \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \cos\theta & 0 & 0 & \cos\theta \end{pmatrix}$$

Then the similarity transformation

$$\mathbf{S_1^T A S_1} = \mathbf{A'} = \begin{pmatrix} d_1' & e_1' & 0 & 0 \\ e_1' & d_2 & e_2 & 0 \\ 0 & e_2 & d_3 & e'3 \\ 0 & 0 & e_3' & d_4' \end{pmatrix}$$

produces a matrix where the primed elements in $\mathbf{A'}$ have been changed by the transformation whereas the unprimed elements are unchanged.

Eigenvalues with the *QR* and Lanczos methods

If we now choose θ to give the element $a_{21}^{'} = e^{'} = 0$ then we have the first eigenvalue $= a_{11}^{'} = d_1^{'}$.

This procedure can be continued on the remaining three-dimensional submatrix for the next eigenvalue. Thus after few transformations we have the wanted diagonal form.

What we see here is just a special case of the more general procedure developed by Francis in two articles in 1961 and 1962. Using Jacobi's method is not very efficient ether.

The algorithm is based on the so-called **QR** method (or just **QR**-algorithm). It follows from a theorem by Schur which states that any square matrix can be written out in terms of an orthogonal matrix $\hat{Q}$ and an upper triangular matrix $\hat{U}$. Historically R was used instead of U since the wording right triangular matrix was first used.

Eigenvalues with the QR algorithm and Lanczos' method

The method is based on an iterative procedure similar to Jacobi's method, by a succession of planar rotations. For a tridiagonal matrix it is simple to carry out in principle, but complicated in detail!

Schur's theorem

$$\hat{A} = \hat{Q}\hat{U},$$

is used to rewrite any square matrix into a unitary matrix times an upper triangular matrix. We say that a square matrix is similar to a triangular matrix.

Householder's algorithm which we have derived is just a special case of the general Householder algorithm. For a symmetric square matrix we obtain a tridiagonal matrix.

There is a corollary to Schur's theorem which states that every Hermitian matrix is unitarily similar to a diagonal matrix.

Eigenvalues with the QR algorithm and Lanczos' method

It follows that we can define a new matrix

$$\hat{A}\hat{Q} = \hat{Q}\hat{U}\hat{Q},$$

and multiply from the left with $\hat{Q}^{-1}$ we get

$$\hat{Q}^{-1}\hat{A}\hat{Q} = \hat{B} = \hat{U}\hat{Q},$$

where the matrix $\hat{B}$ is a similarity transformation of $\hat{A}$ and has the same eigenvalues as $\hat{B}$.

Eigenvalues with the QR algorithm and Lanczos' method

Suppose $\hat{A}$ is the triangular matrix we obtained after the Householder transformation,

$$\hat{A} = \hat{Q}\hat{U},$$

and multiply from the left with $\hat{Q}^{-1}$ resulting in

$$\hat{Q}^{-1}\hat{A} = \hat{U}.$$

Suppose that $\hat{Q}$ consists of a series of planar Jacobi like rotations acting on sub blocks of $\hat{A}$ so that all elements below the diagonal are zeroed out

$$\hat{Q} = \hat{R}_{12}\hat{R}_{23}\dots\hat{R}_{n-1,n}.$$

Eigenvalues with the QR algorithm and Lanczos' method

A transformation of the type $\hat{R}_{12}$ looks like

$$\hat{R}_{12} = \begin{pmatrix} c & s & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ -s & c & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & & & \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix}$$

Eigenvalues with the QR algorithm and Lanczos' method

The matrix $\hat{U}$ takes then the form

$$\hat{U} = \begin{pmatrix} x & x & x & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & x & x & x & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & x & x & x & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & & & \\ 0 & 0 & 0 & 0 & 0 & \dots & x & x & x \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & x & x \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & x \end{pmatrix}$$

which has a second superdiagonal.

Eigenvalues with the QR algorithm and Lanczos' method

We have now found $\hat{Q}$ and $\hat{U}$ and this allows us to find the matrix $\hat{B}$ which is, due to Schur's theorem, unitarily similar to a triangular matrix (upper in our case) since we have that

$$\hat{Q}^{-1}\hat{A}\hat{Q} = \hat{B},$$

from Schur's theorem the matrix $\hat{B}$ is triangular and the eigenvalues the same as those of $\hat{A}$ and are given by the diagonal matrix elements of $\hat{B}$. Why?

Our matrix $\hat{B} = \hat{U}\hat{Q}$.

Eigenvalues with the QR algorithm and Lanczos' method

The matrix $\hat{A}$ is transformed into a tridiagonal form and the last step is to transform it into a diagonal matrix giving the eigenvalues on the diagonal.

The eigenvalues of a matrix can be obtained using the characteristic polynomial

$$P(\lambda) = det(\lambda\mathbf{I} - \mathbf{A}) = \prod_{i=1}^{n} (\lambda_i - \lambda),$$

which rewritten in matrix form reads

$$P(\lambda) = \begin{pmatrix} d_1-\lambda & e_1 & 0 & 0 & \dots & 0 & 0 \\ e_1 & d_2-\lambda & e_2 & 0 & \dots & 0 & 0 \\ 0 & e_2 & d_3-\lambda & e_3 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & d_{N_{\rm step}-2}-\lambda & e_{N_{\rm step}-1} \\ 0 & \dots & \dots & \dots & \dots & e_{N_{\rm step}-1} & d_{N_{\rm step}-1}-\lambda \end{pmatrix}$$

Eigenvalues with the QR algorithm and Lanczos' method

We can solve this equation in an iterative manner. We let $P_k(\lambda)$ be the value of k subdeterminant of the above matrix of dimension $n \times n$. The polynomial $P_k(\lambda)$ is clearly a polynomial of degree k. Starting with $P_1(\lambda)$ we have $P_1(\lambda) = d_1 - \lambda$. The next polynomial reads $P_2(\lambda) = (d_2 - \lambda)P_1(\lambda) - e_1^2$. By expanding the determinant for $P_k(\lambda)$ in terms of the minors of the nth column we arrive at the recursion relation

$$P_k(\lambda) = (d_k - \lambda)P_{k-1}(\lambda) - e_{k-1}^2 P_{k-2}(\lambda).$$

Together with the starting values $P_1(\lambda)$ and $P_2(\lambda)$ and good root searching methods we arrive at an efficient computational scheme for finding the roots of $P_n(\lambda)$. However, for large matrices this algorithm is rather inefficient and time-consuming.

Eigenvalues and Lanczos' method

Basic features with a real symmetric matrix (and normally huge $n > 10^6$ and sparse) $\hat{A}$ of dimension $n \times n$:

- Lanczos' algorithm generates a sequence of real tridiagonal matrices T_k of dimension $k \times k$ with $k \leq n$, with the property that the extremal eigenvalues of T_k are progressively better estimates of $\hat{A}$' extremal eigenvalues.* The method converges to the extremal eigenvalues.

- The similarity transformation is

$$\hat{T} = \hat{Q}^T \hat{A} \hat{Q},$$

with the first vector $\hat{Q}\hat{e}_1 = \hat{q}_1$.

We are going to solve iteratively

$$\hat{T} = \hat{Q}^T \hat{A} \hat{Q},$$

with the first vector $\hat{Q}\hat{e}_1 = \hat{q}_1$. We can write out the matrix $\hat{Q}$ in terms of its column vectors

$$\hat{Q} = [\hat{q}_1 \hat{q}_2 \dots \hat{q}_n].$$

Eigenvalues and Lanczos' method, tridiagonal matrix

The matrix

$$\hat{T} = \hat{Q}^T \hat{A} \hat{Q},$$

can be written as

$$\hat{T} = \begin{pmatrix} \alpha_1 & \beta_1 & 0 & \dots & \dots & 0 \\ \beta_1 & \alpha_2 & \beta_2 & 0 & \dots & 0 \\ 0 & \beta_2 & \alpha_3 & \beta_3 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & 0 \\ \dots & & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ 0 & \dots & \dots & 0 & \beta_{n-1} & \alpha_n \end{pmatrix}$$

Eigenvalues and Lanczos' method, tridiagonal and orthogonal matrices

Using the fact that

$$\hat{Q}\hat{Q}^T = \hat{I},$$

we can rewrite

$$\hat{T} = \hat{Q}^T \hat{A} \hat{Q},$$

as

$$\hat{Q}\hat{T} = \hat{A}\hat{Q}.$$

Eigenvalues and Lanczos' method

If we equate columns

$$\hat{T} = \begin{pmatrix} \alpha_1 & \beta_1 & 0 & \dots & \dots & 0 \\ \beta_1 & \alpha_2 & \beta_2 & 0 & \dots & 0 \\ 0 & \beta_2 & \alpha_3 & \beta_3 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & 0 \\ \dots & & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ 0 & \dots & \dots & 0 & \beta_{n-1} & \alpha_n \end{pmatrix}$$

we obtain

$$\hat{A}\hat{q}_k = \beta_{k-1}\hat{q}_{k-1} + \alpha_k \hat{q}_k + \beta_k \hat{q}_{k+1}.$$

Eigenvalues and Lanczos' method, defining the Lanczos' vectors

We have thus

$$\hat{A}\hat{q}_k = \beta_{k-1}\hat{q}_{k-1} + \alpha_k \hat{q}_k + \beta_k \hat{q}_{k+1},$$

with $\beta_0 \hat{q}_0 = 0$ for $k = 1 : n-1$. Remember that the vectors $\hat{q}_k$ are orthornormal and this implies

$$\alpha_k = \hat{q}_k^T \hat{A} \hat{q}_k,$$

and these vectors are called Lanczos vectors.

Eigenvalues and Lanczos' method, basic steps

We have thus

$$\hat{A}\hat{q}_k = \beta_{k-1}\hat{q}_{k-1} + \alpha_k\hat{q}_k + \beta_k\hat{q}_{k+1},$$

with $\beta_0\hat{q}_0 = 0$ for $k = 1 : n-1$ and

$$\alpha_k = \hat{q}_k^T\hat{A}\hat{q}_k.$$

If

$$\hat{r}_k = (\hat{A} - \alpha_k\hat{I})\hat{q}_k - \beta_{k-1}\hat{q}_{k-1},$$

is non-zero, then

$$\hat{q}_{k+1} = \hat{r}_k/\beta_k,$$

with $\beta_k = \pm||\hat{r}_k||_2$.