Todays lecture

- Natural orbitals, SVD, and all that.
- Matrix diagonalization: The classical Jacobi is not so bad.
- Larger matrices: Krylov methods, Lanczos, Davidson
- Large scale linear and mildly non-linear equation systems: The Pre-conditioned Conjugate Gradient method.

A useful set of eigenvectors: Natural orbitals.

Suppose that the exact wave function Ψ were known, and contains n electrons. We want to use at most N (where $N \geq n$) one-electron spin-orbitals $\{|\psi_p\rangle\}$, with $p \in \{1, 2, 3 \ldots, N\}$. One criterion on a good orbital set is that they allow a Full-CI wave function with large overlap with Ψ. There exists such an ordered set of such orbitals, that selecting the first N solves the N-orbital maximum-overlap problem, for any N.

This set of orbitals are the eigenfunctions of the 1-particle spin density of the wave function, ordered by decreasing eigenvalue. They are called natural (spin-)orbitals, and the eigenvalues are called natural occupation numbers. These are between 0 and 1, and add up to n – or slightly less, if only the first few natural orbitals are used.

This application to Quantum Machanical systems was done by P. O. Löwdin, who showed a number of interesting properties.

In general, the use of eigenvectors to find an optimal approximation to e.g. matrices is old, and much used in numerical algebra, with great practical and economic value.

'Condensed' data: The Singular Value Decomposition.

Similarly to the natural orbitals: Suppose that a large (or even infinite) general $n \times m$ matrix $\mathbf{A}$ is to be approximated as well as possible (in a least-square sense) by factorizing into smaller matrices:

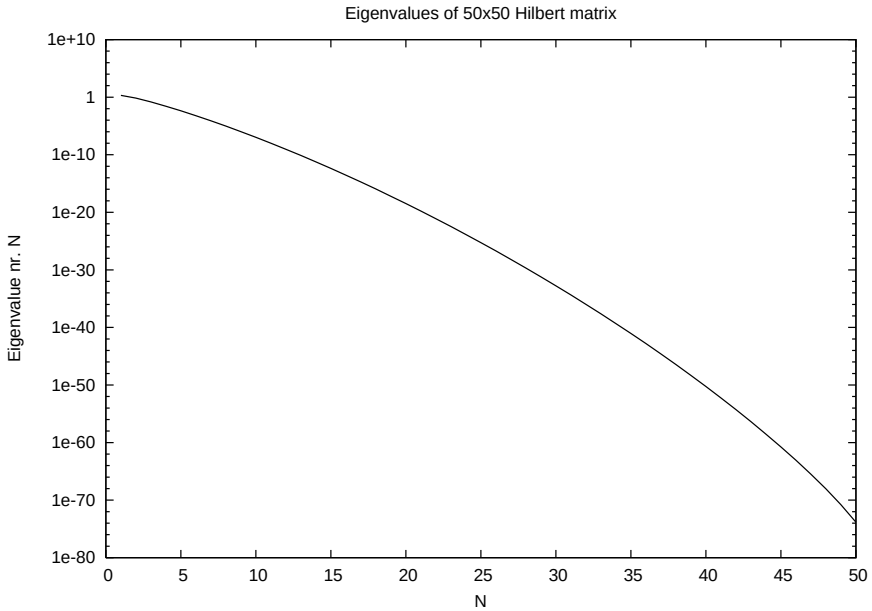
$$\mathbf{A} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \quad \text{or} \quad A_{ij} = \sum_{k=1}^{N} \sigma_k U_{ik} V_{jk} + R_{ij}$$

where N is assumed to be much smaller than n and m, and where the weights σ_k and the arrays $\mathbf{U}$ and $\mathbf{V}$ are determined such as to minimize $||\mathbf{R}||^2$. Then this is a well-known minimization problem, yielding the weights as so-called 'Singular Values' and the columns of $\mathbf{U}$ and $\mathbf{V}$ as corresponding 'Singular Vectors' in a Singular Value Decomposition, SVD.

Just as for natural orbitals, the optimal choice for any N is to choose the first N singular vectors (ordered by decreasing singular value).

Example of an SVD.

The 50×50 Hilbert matrix $H_{kl} := 1/(k + l - 1)$ has 2500 elements, ranging in value from $1/99$ up to 1. Only a handful of eigenvalues are appreciably different from 0:



Since it is a square and symmetric matrix, its SVD is actually an ordinary spectral decomposition. Truncated to 9 vectors, the matrix is represented with a precision a precision of about 10^{-8}; 16 vectors give precision 10^{-16}.

Typical savings using matrix decomposition (SVD, Cholesky. . .) of two-electron integral data sets: 90% – 99.9%.

How an SVD is done.

Decomposing a **small** $n \times m$ matrix $\mathbf{A}$ is usually done by full diagonalization. Assume $n \geq m$, so rank($\mathbf{A}$) is at most m. Then compute the square positive (semi-)definite matrix $\mathbf{A}^\dagger\mathbf{A}$ and proceed by diagonalizing it:

$$\mathbf{A}^\dagger\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\dagger, \text{where} \quad \mathbf{V}\mathbf{V}^\dagger = \mathbf{V}^\dagger\mathbf{V} = \mathbf{1},$$

Λ is diagonal, and has m real non-negative diagonal elements.

Then let $\mathbf{\Sigma} = \mathbf{\Lambda}^{1/2}$, and compute

$$\mathbf{U} = \mathbf{A}\mathbf{V}\mathbf{\Sigma}^{-1}$$

.

(If $m \geq n$, use instead $\mathbf{A}\mathbf{A}^\dagger$, and take the Hermitian conjugate of the whole procedure; then interchange $\mathbf{A}$ and $\mathbf{A}$).

Note that $\mathbf{\Sigma}$ is a diagonal matrix. We assume all elements are positive; if not, some vectors can be thrown away.

If neither n nor m is small, the SVD can be computed numerically using specialized linear algebra library calls.

Eigenvalues, characteristic equations

For very small matrices, eigenvalues are found from the characteristic equation: Iff $(\mathbf{A} - z\mathbf{1})$ is singular, then the determinant must be zero. That determinant is simply an $n-$th degree polynomial in z, having n roots (counting multiplicity), which can be found by analytic formula if n is less than 5, and in any case easily determined with arbitrary precision.

Example:

$$\det\left(\begin{pmatrix} 263 & 180 \\ 180 & -94 \end{pmatrix} - z\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\right) = \det\begin{pmatrix} 263-z & 180 \\ 180 & -94-z \end{pmatrix}$$

$$= (263-z)(-94-z) - 180\cdot 180 = 24722 - 169z + z^2 - 32400$$

$$= z^2 - 169z - 7678$$

The roots of this characteristic equation are

$$\frac{169}{2} \pm \sqrt{\left(\frac{169}{2}\right)^2 + 7678} = \{338, -169\}$$

Jacobi's method

For hermitian $n \times n$ matrices with n up to a few hundred, the Jacobi method is a good choice. It is utterly safe, handles massive degeneracy perfectly and gives accurately ortonormal eigenvectors. Start by the diagonalization of a two by two matrix by rotation: The matrix

$$\mathbf{R}(\theta) \stackrel{\text{def}}{=} \begin{pmatrix} c & s \\ -s & c \end{pmatrix},$$

where $c = \cos\theta$ and $s = \sin\theta$, is called a rotation matrix.

This matrix is unitary (in this case: real orthonormal), so its inverse is its transpose. A similarity transformation that diagonalizes the matrix is thus

$$\mathbf{R}^T\mathbf{A}\mathbf{R} = \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} \\ A_{12} & A_{22} \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix} = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

The values $c = \cos\theta$ and $s = \sin\theta$ are easily obtained without need of the angle θ:

Two by two diagonalization by rotation

A numerically stable solution for the rotation matrix is obtained as

$$\begin{aligned} u &= (A_{22} - A_{11})/(2A_{12}) \\ t &= \mathrm{sign}(u)/(|u| + \sqrt{1+u^2}) \\ c &= 1/\sqrt{1+u^2} \\ s &= ct \end{aligned}$$

Example: $A_{11} = 263, A_{22} = -94, A_{12} = 180$ gives $u = \frac{119}{120}$, $t = \frac{5}{12}$, $c = \frac{12}{13}$ and $s = \frac{5}{13}$ (please confirm!).

and

$$\begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} 263 & 180 \\ 180 & -94 \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix} = \begin{pmatrix} 338 & 0 \\ 0 & -169 \end{pmatrix}$$

(please confirm!).

Iterative diagonalization by 2×2 rotations

In the Jacobi method, one systematically goes through the $n(n-1)/2$ pairs of indices $n \geq i > j \geq 1$, and makes the elements A_{ij} and A_{ji} equal to zero by a 2×2 rotation. This affects also all elements A_{ik}, etc., but it can be shown that the quantity

$$\tau^2(\mathbf{A}) \stackrel{\text{def}}{=} \sum_{i>j} A_{ij}^2$$

will be lowered with exactly A_{ij}^2 when these elements are zeroed.

This can be used to show that the procedure must eventually converge towards a diagonal matrix.

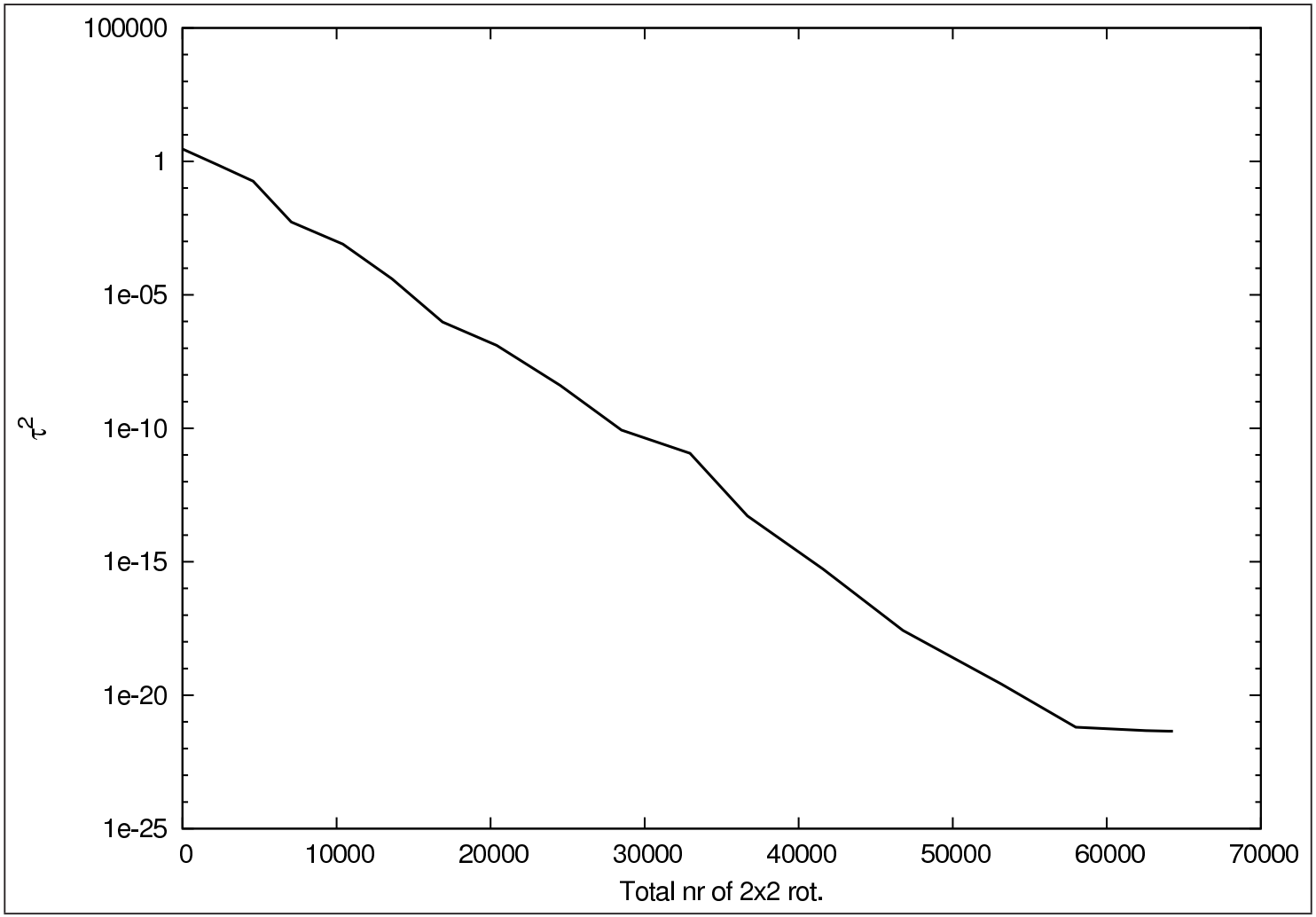
Actually the asymptotic convergence is quadratic.

Jacobi iteration statistics, Hilbert matrix

500×500 Hilbert matrix, 18 eigenvalues are larger than 10^{-10}.

NSWEEP	NR	NROT	VNSUM	SBDMAX
0			2.93273830790	0.500000000000
1	2250	2250	0.758507193959	0.261011633900
2	2339	4589	0.182568611379	0.365825130020
3	2478	7067	0.536986782233E-02	0.291822216536E-01
4	3341	10408	0.798664249152E-03	0.207448153681E-01
5	3188	13596	0.396225883696E-04	0.327062209017E-02
6	3280	16876	0.954122256056E-06	0.317102128824E-03
7	3524	20400	0.127960316210E-06	0.291097431498E-03
8	4126	24526	0.402018035910E-08	0.442552481362E-04
9	3968	28494	0.853806236323E-10	0.566486729997E-05
10	4449	32943	0.116339621069E-10	0.311665103047E-05
11	3724	36667	0.524136421267E-13	0.143072616057E-06
12	4920	41587	0.531301978611E-15	0.115003537767E-07
13	5176	46763	0.265352132524E-17	0.424514779212E-09
14	6396	53159	0.261636387705E-19	0.354669310870E-10
15	4825	57984	0.633179746661E-21	0.486714224086E-11

Jacobi iteration statistics



Jacobi, conclusions

The advantages with the Jacobi methods are:

Easy to program.

No problems with massive degeneracy.

Accurately orthonormal eigenvectors.

Almost-diagonal matrices converge fast.

The disadvantages are:

Always giving full diagonalization.(Cannot save time when only a few eigenvectors are needed).

Slower than modern Householder, QR, MRRR...

MUCH slower for large problems ($>$ a few hundred variables)

Subspace iteration (Krylov) methods

Need to find solutions to very large systems of linear equations,

$$\mathbf{A}\mathbf{x} = \mathbf{y}$$

with thousands or millions of variables. $\mathbf{A}$ cannot be stored.

The problem can be solved by projection using some small set of basis vectors, $\{\mathbf{b}_k\}_{k=1}^{n}$, with $n \ll N$, which we collect in the $N \times n$ matrix $\mathbf{B}$.

The solution is then approximated as

$$\mathbf{x} \approx \sum_{k=1}^{n} z_k \mathbf{b}_k$$

with a 'backwards error', the residual vector

$$\mathbf{r} = \mathbf{y} - \mathbf{A}\mathbf{B}\mathbf{z}$$

The small solution vector $\mathbf{z}$ which minimizes $\mathbf{r}$ in norm, is given by

$$\mathbf{z} = \left(\mathbf{B}^{\dagger}\mathbf{A}\mathbf{B}\right)^{-1}\mathbf{B}^{\dagger}\mathbf{y}$$

if the vectors are orthonormal.

Subspace iteration (Krylov) methods

As can be seen, this will provide an approximate solution only if we have a suitable basis, and if the products $\mathbf{A}\mathbf{b}_k$ can be computed.

The idea in subspace iteration methods is that as long as the residual is non-zero, it can be used to produce a new basis vector.

The subspace spanned by the basis is then growing, until the residual is so small that the problem is solved.

For positive definite, hermitian (or symmetric) matrices $\mathbf{A}$, a good idea is to orthonormalize the residual vector against the old basis. There are several other methods as well, and often a preconditioner $\mathbf{P}$ is applied to the residual. Anyhow, the basis vectors produced are linear combinations in a Krylov sequence, $\{\mathbf{A}^n\}$ or $\{\mathbf{P}\mathbf{A}^n\}$, and the basis vectors are thus obtained as polynomials acting on the starting vector,

$$p_n(\mathbf{A})\mathbf{b}_1 \quad \text{or} \quad p_n(\mathbf{P}\mathbf{A})\mathbf{b}_1$$

where the polynomials are obtained recursively.

The Lanczos method

For positive definite, Hermitian matrices, the basis vectors are automatically being orthonormalized by just orthonormalizing against the previous two basis vectors, in exact analogy to recursion formulae for classical orthogonal polynomials.

Moreover, the small matrix representation of the problem will be tridiagonal.

The procedure is called Lanczos method, and can be described by the algorithm

$$\begin{aligned} \boldsymbol{b}_i &= \boldsymbol{x}_i/|x|_i \\ \boldsymbol{\sigma}_i &= \boldsymbol{A}\boldsymbol{b}_i \\ d_i &= \boldsymbol{\sigma}_i^T \boldsymbol{b}_i \\ \boldsymbol{x}_{i+1} &= \boldsymbol{\sigma}_i - d_i \boldsymbol{b}_i - t_{i-1}\boldsymbol{b}_{i-1} \\ t_i &= |x|_{i+1} \\ & \text{(Terminate if } t_i = 0\text{, else repeat)} \end{aligned}$$

Diagonalizing the tridiagonal matrix is very fast and efficient, resulting in eigenvalues and eigenvectors. $\boldsymbol{A}$ is large, and the process is terminated after a suitable number of steps, so the eigenvalues and eigenvectors are usually approximate.

Handcomputed example of Lanczos algorithm.

The 9×9 matrix to diagonalize in Exc. 75.8 pg 726 has elements $A_{ij} = 0.1 + 0.9\delta_{ij}$. The result $\boldsymbol{\sigma}$ of multiplying it with an arbitrary vector $\mathbf{x}$ is then

$$\sigma_i = 0.1 \sum_{j=1}^{9} x_j + 0.9 \sum_{j=1}^{9} \delta_{ij}\, x_j = 0.1 \sum_{j=1}^{9} x_j + 0.9 x_i$$

which is our 'sigma routine' for this example.

The algorithm in Eq. 4.39 requires a starting vector; Choose arbitrarily the vector $\mathbf{x}_1 = (1, 0, 0, 0, 0, 0, 0, 0, 0)^{\mathrm{T}}$.

Since then $\sum_{j=1}^{9} x_j = 1$, we obtain e.g. $(\boldsymbol{\sigma}_1)_1 = 0.1 + 0.9 = 1$, while $(\boldsymbol{\sigma}_1)_i = 0.1$ for i=2...9.

The algorithm can be followed on the next page.

Handcomputed example of Lanczos algorithm (2).

The algorithm proceeds as follows:

$$
\begin{aligned}
x_1 &= 1.0 \\
\mathbf{b}_1 &= (1,0,0,0,0,0,0,0,0)^{\mathrm{T}} \\
\boldsymbol{\sigma}_1 &= (1,0.1,0.1,\ldots,0.1)^{\mathrm{T}} \\
d_1 &= \boldsymbol{\sigma}_1^T \mathbf{b}_1 = 1 \\
\mathbf{x}_2 &= \boldsymbol{\sigma}_1 - d_1 \mathbf{b}_1 = (0,0.1,0.1,\ldots,0.1)^{\mathrm{T}} \\
t_1 &= x_2 = 0.282842712 \\
\mathbf{b}_2 &= (0,0.35355\ldots,\ldots,0.35355339)^{\mathrm{T}} \\
\boldsymbol{\sigma}_2 &= (0.28284271,0.60104076,\ldots,0.60104076)^{\mathrm{T}} \\
d_2 &= 1.7 \\
\mathbf{x}_3 &= \boldsymbol{\sigma}_2 - d_2 \mathbf{b}_2 - t_1 \mathbf{b}_1 = (0,0,0,0,0,0,0,0,0)^{\mathrm{T}}
\end{aligned}
$$

If the goal was to perform a tridiagonalization, we have failed, since the algorithm breaks off here.

Handcomputed example of Lanczos algorithm (3).

The tridiagonal result, so far, is

$$\begin{pmatrix} d_1 & t_1 \\ t_1 & d_2 \end{pmatrix} = \begin{pmatrix} 1 & 0.282842712 \\ 0.282842712 & 1.7 \end{pmatrix}$$

Eigenvalues are 0.9 and 1.8. The eigenvector with eigenvalue 1.8 is $(0.333333333, 0.9428090416)^{\mathrm{T}}$. The corresponding eigenvector of the original matrix is

$$0.3333333\,\mathbf{b}_1 + 0.942809\,\mathbf{b}_2 = (0.333333333, 0.333333333, \ldots, 0.333333333)^{\mathrm{T}}$$

The result is exact. The reason for early termination: The matrix has only two distinct eigenvalues, namely 0.9 (8-fold degenerate) and 1.8. Lanczos converges quickly if the matrix has clustered eigenvalues.

An outline of 'Davidson-type' methods

Initialize: N_{start} vectors $\{\mathbf{q}\}$ are given or selected somehow. Let $N_{\text{old}} \leftarrow 0$; $N_{\text{new}} \leftarrow N_{\text{start}}$;

Iterate: 1. **Orthonormalize:** Orthonormalize, e.g. by the Gram-Schmidt procedure, the N_{new} $\mathbf{q}$-vectors against the earlier N_{old} $\mathbf{b}$ vectors, and among themselves. In the process, discard N_{dep} linearly dependent vectors. Retain $N_{\text{new}} - N_{\text{dep}}$ vectors, and set $N_{\text{new}} \leftarrow N_{\text{new}} - N_{\text{dep}}$. If $N_{\text{new}} = 0$, the method has failed.

2. **Sigma vector generation:** N_{new} σ-vectors are computed as $\mathbf{s}^i = \mathbf{A}\mathbf{b}^i$. This is normally the time-consuming step.

3. **Extend $\tilde{\mathbf{A}}$:** $\tilde{A}_{ij} \leftarrow (\mathbf{s}^i\mathbf{b}^j)$ for $i = N_{\text{old}} + 1, \ldots, N_{\text{old}} + N_{\text{new}}$; j=1,...,i.
$N_{\text{old}} \leftarrow N_{\text{old}} + N_{\text{new}}$.

4. **Diagonalize $\tilde{\mathbf{A}}$:** Obtain N_{old} eigenvectors and eigenvalues.
$\mathbf{A}\mathbf{v}^k = \lambda^k\mathbf{v}^k$.

5. **Root selection:** Select N_{sel} of these. Criteria can be: smallest eigenvalue, maximum overlap with earlier solutions, or other.

6. **Residual vectors:** Compute N_{sel} residual vectors $\{\mathbf{r}^k\}$ as linear combinations of the $\mathbf{s}$ and $\mathbf{b}$ vectors:
$\mathbf{r}^k = \sum_i v_i^k\mathbf{s}^i - \lambda^k\mathbf{b}^i$

7. **Converged?** Discard any residual vectors that are smaller than a given threshold. The remaining N_{new} vectors are unconverged. If $N_{\text{new}} = 0$, we are finished: Break iteration loop.

8. **Preconditioning:** From N_{new} residual $\mathbf{r}$ vectors, form N_{new} proposed update vectors $\{\mathbf{q}\}$.

End of iteration loop: The eigenvectors can be obtained as $\mathbf{r}^k = \sum_i v_i^k\mathbf{b}^i$ for the selected roots.

A small example

Assume, as a demonstration example, that we wish to find the lowest eigenvalue and corresponding eigenvector of the 5×5 matrix

$$\mathbf{A} = \begin{pmatrix} 1.0 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 2.0 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 3.0 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 3.0 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 3.0 \end{pmatrix}$$

We will use the standard Davidson method, i.e. preconditioning by dividing the residual elements with a diagonal approximation to $\mathbf{A} - \lambda$, and using just a single vector at a time. We assume the calculation is converged when we obtain a residual vector smaller than 0.05.

Start vector: $\mathbf{b}^1 = (1, 0, 0, 0, 0)^{\mathsf{T}}$.

The first iteration

1 No need to orthonormalize now, of course: There is only one vector so far.

2 The sigma vector generation is generally the time consuming step. It is usually not done by a matrix-times-vector operation, but rather a large number of gather, multiple-daxpy, and scatter operations involving two-electron integrals. In this example, we illustrate this by simplifying the calculation: $\mathbf{s} = \mathbf{A}\mathbf{b}$ can be executed as

$$t \leftarrow 0.1 \sum_{1}^{5} b_i$$

$$s_1 = 0.9b_1 + t, \qquad s_2 = 1.9b_2 + t, \quad \text{etc.}$$

which gives $\mathbf{s}^1 = (1.0, 0.1, 0.1, 0.1, 0.1)^{\mathsf{T}}$.

3 When the algorithm starts, the $\tilde{\mathbf{A}}$ matrix is empty. Thus, in the first iteration, extending it actually means to compute the 1×1 matrix $(\tilde{A}_{11}) = (\mathbf{s}^1\mathbf{b}^1) = (1)$.

4 This matrix is of course already diagonal.

The first iteration

5 Root selection: In the first iteration, there is no choice. The only root is $\lambda = 1$ and $\mathbf{v} = (1)$.

6 The residual vector is

$$\mathbf{r}^1 = \mathbf{s}^1 - \lambda \mathbf{b}^1 = (0, 0.1, 0.1, 0.1, 0.1)^{\mathsf{T}}$$

7 Preconditioning is by elementwise division $/(A_{ii} - \lambda)$.

(In general, the denominator can be small. If that happens, set the result to zero – Note that the denominator can be small sometimes. In that case, set the result 0.

$$\mathbf{q}^1 = (0, 0.1, 0.05, 0.05, 0.05)^{\mathsf{T}}$$

The second iteration

(Entered with $\mathbf{q}^1 = (0, 0.1, 0.05, 0.05, 0.05)^{\mathsf{T}}$).

1 $\mathbf{q}^1$ is already orthogonal to $\mathbf{b}^1$. Normalize:

$$\mathbf{b}^2 = (0, 2, 1, 1, 1)^{\mathsf{T}}/\sqrt{7}$$

2 Sigma vector generation, as described before:

$$\mathbf{s}^2 = (0.5, 4.3, 3.4, 3.4, 3.4)^{\mathsf{T}}/\sqrt{7}$$

3 Extend the $\tilde{\mathbf{A}}$ matrix:

$$\begin{aligned} A_{21} &= (\mathbf{s}^2\mathbf{b}^1) = 0.5/\sqrt{7} \approx 0.188982 \\ A_{22} &= (\mathbf{s}^2\mathbf{b}^2) = 18.8/7 \approx 2.685714 \\ N_{\text{old}} &\leftarrow 2 \end{aligned}$$

The second iteration

4 Diagonalizing $\tilde{\mathbf{A}}$ gives

$$\tilde{\mathbf{A}} = \begin{pmatrix} 0.99392 & 0.11006 \\ -0.11006 & 0.99392 \end{pmatrix} \begin{pmatrix} 0.97907 & 0 \\ 0 & 2.70664 \end{pmatrix} \begin{pmatrix} 0.99392 & -0.11006 \\ 0.11006 & 0.99392 \end{pmatrix}$$

5 Select the lowest root.

6 The residual vector is thus

$$\begin{aligned} \mathbf{r}^2 &= 0.99392(\mathbf{s}^1 - 0.97907\,\mathbf{b}^1) - 0.11006(\mathbf{s}^2 - 0.97907\,\mathbf{b}^2) \\ &\approx (0.041, -0.005, 0.000, 0.000, 0.000)^{\mathsf{T}} \end{aligned}$$

7 $|\mathbf{r}^2| < 0.05$ so the calculation is converged.

The eigenvalue is ca. 0.979, and the eigenvector is

$$\begin{aligned} \mathbf{c} &= 0.99392\,\mathbf{b}^1 - 0.11006\,\mathbf{b}^2 \\ &\approx (0.994, -0.083, -0.042, -0.042, 0.042)^{\mathsf{T}} \end{aligned}$$

Improvements to Davidson's method

Remember: the Davidson method uses the update

$$\mathbf{q} \leftarrow (\mathbf{A}_0 - \lambda)^{-1}\,\mathbf{r}$$

1. Use several roots simultaneously
This is a fairly obvious extension, called Davidson-Liu method.

2. Use better preconditioner, e.g. by full diagonalization of a submatrix of A.
Again, fairly obvious. Has been done by many people.

3. Get rid of the normalization.
(This allows replacing oldest vectors by the newest ones. Disk space will not grow forever.)

4. Replace the update with a *linear combination* of $(\mathbf{A}_0 - \lambda)^{-1}\,\mathbf{r}$ and $(\mathbf{A}_0 - \lambda)^{-1}\,\mathbf{v}$.
(This gives much improved update vectors when $\mathbf{A}_0$ is a good approximation to $\mathbf{A}$ (J. Olsen))

Better preconditioning.

Davidson's recipy, but with the diagonal approximation replaced by a better one, would be $\mathbf{q} \leftarrow (\mathbf{A}_0 - \lambda)^{-1}\,\mathbf{r}$

Olsen's observation: Using better and better $\mathbf{A}_0$ fails to improve convergence rate significantly.

Olsen knew that another recipy, namely the 'inverse iteration method', would give 3-rd order convergence (!): $\mathbf{q} \leftarrow (\mathbf{A} - \lambda)^{-1}\,\mathbf{v}$

He proposed the update

$$\mathbf{q} \;\leftarrow\; c_1\,(\mathbf{A}_0 - \lambda)^{-1}\,\mathbf{r} + c_2\,(\mathbf{A}_0 - \lambda)^{-1}\,\mathbf{v}$$

with c_1, c_2 given by the condition $\mathbf{q} \cdot \mathbf{v} = 0$. This is also the form suggested by perturbation theory.

Much improved convergence rates were obtained.

Skipping orthonormalization

Do **not** form basis vectors $\mathbf{b}$.
Use directly the pairs $\mathbf{v}^{(k)}$ and $\mathbf{s}^{(k)} = \mathbf{A}\mathbf{v}^{(k)}$!

Need compute not only $\tilde{A}_{kl} =< \mathbf{v}^{(k)}, \mathbf{s}^{(l)} >$
but also $\tilde{S}_{kl} =< \mathbf{v}^{(k)}, \mathbf{v}^{(l)} >$!

Do not solve $\tilde{\mathbf{A}}\mathbf{U} = \mathbf{U}\mathbf{D}$.
Instead, $\tilde{\mathbf{A}}\mathbf{U} = \tilde{\mathbf{S}}\mathbf{U}\mathbf{D}$; $\mathbf{U}$ is no longer unitary.

When solving, $\tilde{\mathbf{S}}$ must be orthonormalized.
Use Gram-Schmidt backwards – this is very important!

Now, we no longer need to just add to a growing store of vectors.
Instead, we can discard old ones and reuse disk space.

The Preconditioned Conjugate Gradient method, example calculation

Assume we wish to solve a large system of linear equations $\mathbf{Hd} = -\mathbf{g}$, such as a linear response calculation. We have available two subroutine calls. Both take a large vector $\mathbf{x}$ as input, and then compute a new vector $\mathbf{y}$. The first, the *sigma* routine, computes the equation matrix times the input vector, i.e. $\mathbf{y} = \mathbf{Hx}$. The other, a *preconditioner*, multiplies an approximate inverse $\mathbf{A} \approx \mathbf{H}^{-1}$ with the vector. Both $\mathbf{H}$ and $\mathbf{A}$ are usually assumed to be positive definite and symmetric. We will actually use the matrix with undefinite matrices, to demonstrate that it still works. It usually does: Positive definiteness is useful in order to *guarantee* nice convergence properties, but it is not *necessary*.

The PCG algorithm

The algorithm is taken directly from the ESQC book, ch 4.

Select an arbitrary positive scale factor ρ_1 and set

$$r_1 = -g, \quad p_1 = \rho_1 A r_1, \quad s_1 = H p_1, \quad \gamma_1 = r_1^{\mathrm{T}} A r_1$$

Then iterate:

$$\delta_k = p_k^{\mathrm{T}} s_k, \quad \eta_k = \rho_k \gamma_k, \quad \alpha_k = \frac{\eta_k}{\delta_k},$$

$$d_{k+1} = d_k + \alpha_k p_k, \quad r_{k+1} = r_k - \alpha_k s_k,$$

$$\gamma_{k+1} = r_{k+1}^{\mathrm{T}} A r_{k+1}, \quad \beta_k = \frac{\gamma_{k+1}}{\eta_k}$$

$$p_{k+1} = \rho_{k+1}(A r_{k+1} + \beta_k p_k), \quad s_{k+1} = H p_{k+1}$$

where ρ_{k+1} is some positive scale factor.

The PCG demonstration example

In this example, we use a rather small matrix (24×24). It has eigenvalues in the range $[-0.9, 22.1]$. The approximate inverse is a diagonal approximation. The product $\mathbf{HA}$ should ideally be $= \mathbf{1}$. The problem would then be solved immediately. In reality, the quality of the approximation depends on the eigenvalues of $\mathbf{HA}$, and on the start vector. Inspection of the algorithm shows that the residual vector $\mathbf{r}$ will then be a linear combination of terms $(\mathbf{HA})^k\mathbf{g}$. Assume that we know a complete spectral resolution of $\mathbf{HA}$, and a corresponding partitioning of $\mathbf{H}$:

$$\mathbf{HA} = \sum_i \lambda_i \mathbf{P}_i, \quad \mathbf{P}_i\mathbf{P}_j = \delta_{ij}\mathbf{P}_i, \quad \mathbf{g} = \sigma\omega_i\mathbf{v}_i$$

where $\mathbf{g}_i$ are eigenvectors and $\mathbf{P}_i$ are corresponding projectors.

The PCG demonstration example (2)

The residual vector is then a k-th degree polynomial in $\mathbf{HA}$ multiplied by $\mathbf{H}$:

$$\mathbf{r}_k = \sum_{n=0}^{k} c_n^{(k)} (\mathbf{HA})^n \mathbf{g} = \sum_i p(\lambda_i) \omega_i \mathbf{H}_i$$

In this example, the eigenvalues of $\mathbf{HA}$ were spread in the range $[-1, 2]$. The start approximation is $\mathbf{H} = (0, 0, \ldots, 0)^{\mathrm{T}}$, and the right-hand side was $\mathbf{H} = (1, 1, \ldots, 1)^{\mathrm{T}}$.

The PCG results

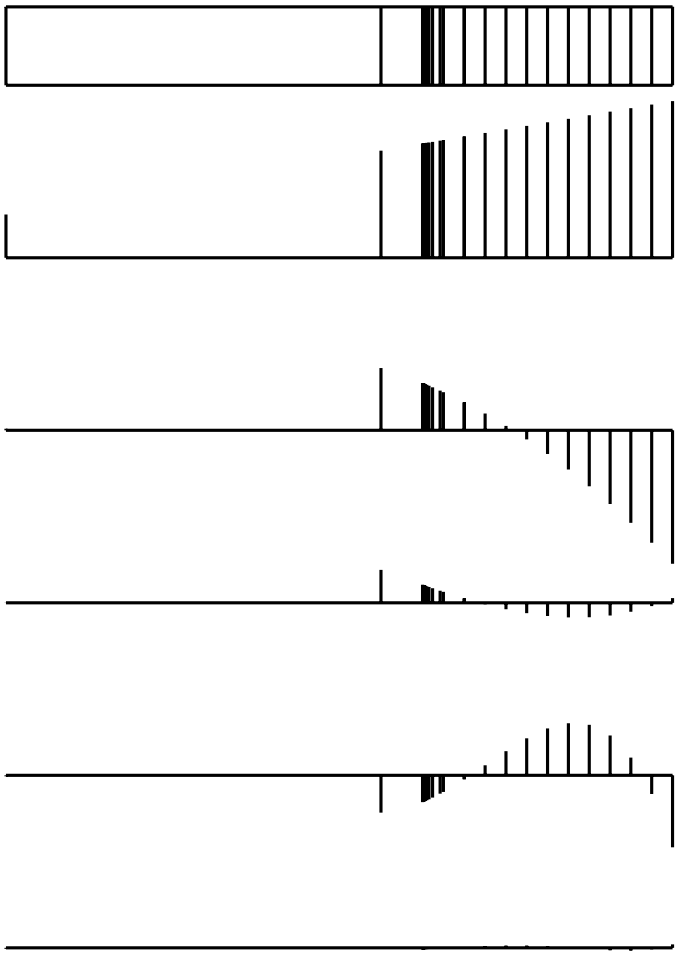
The size of the residual norm after each iteration is tabulated below.

0–th iteration	4.9 (This is the norm of $\mathbf{g}$).
1	7.81
2	3.50
3	0.87
4	1.98
5	0.10
6	0.019
7	0.0037
8	0.00060
9	0.00011
10	0.000023
17	0.0000000000049

The PCG results (2)

A good picture of how the method works is given in the following plots, which show the expansion of $\mathbf{r}_k$ in terms of eigenvectors of $\mathbf{HA}$, i.e. the weights in $\mathbf{r}_k$

It is immediately apparent how the method constructs higher and higher degree polynomials to multiply the weights.



Other modern techniques

Not treated today:

- Many, many useful special matrices: Cauchy, Toeplitz, Circulant, Vandermonde...
- Many special formats for data compression: Sparse matrices, CUR factorization, Interpolatory decomposition, ...
- Randomized algorithms

A few favorite books