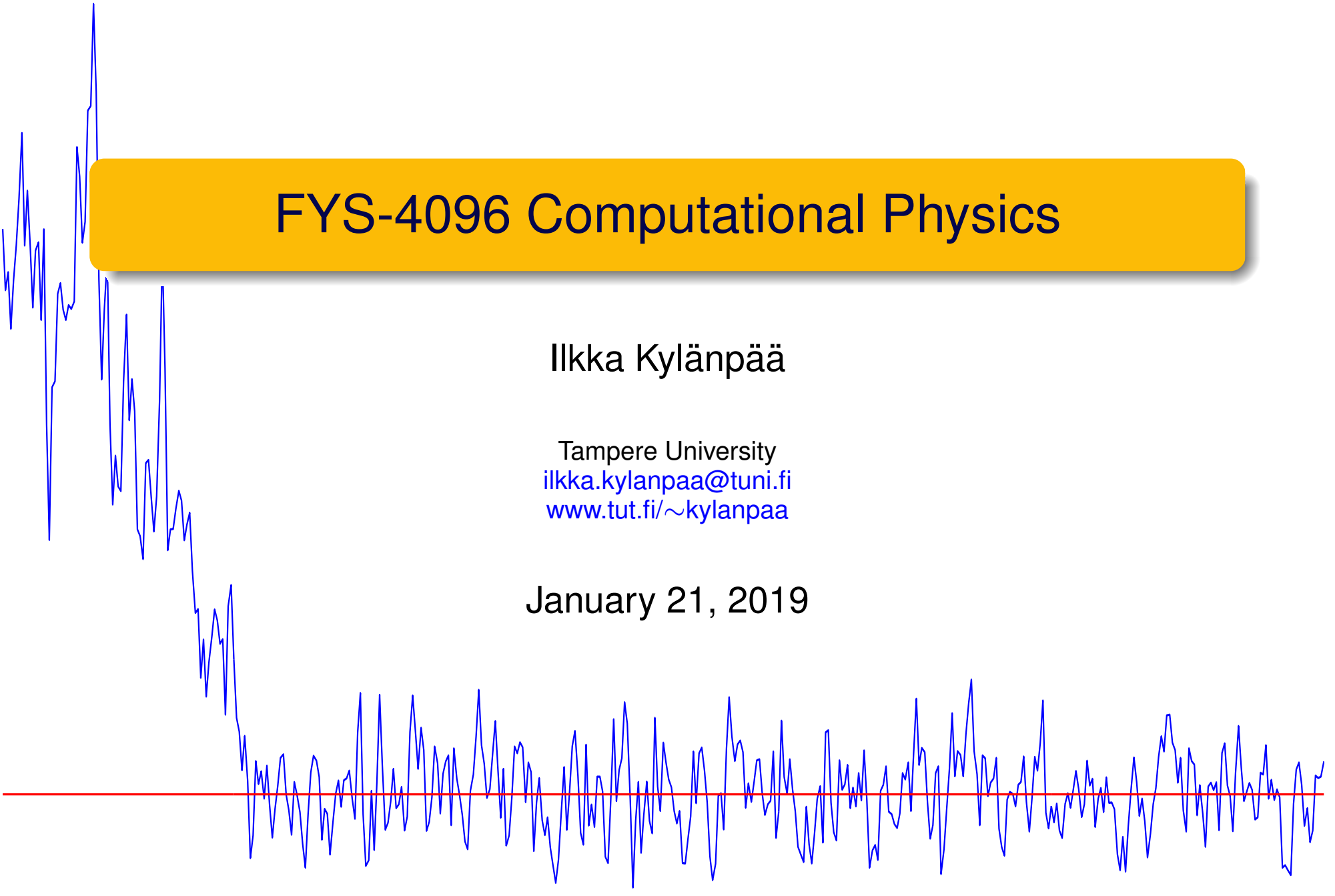
FYS-4096 Computational Physics

Ilkka Kylänpää

Tampere University
ilkka.kylanpaa@tuni.fi
www.tut.fi/∼kylanpaa

January 21, 2019



Course details

Active participation for 12 weeks that include

- ~ 2 hours of lectures on Mondays (12-14)
- $3 - 4$ hours in computer laboratory on Tuesdays (14-18)
- ~ 2 hours of exercises on Fridays (12-14)

Course requirements and practicalities:

- Good participation and mandatory weekly exercises (70% of the grade)
- Two project works (30% of the grade)
- However, both project works required to pass the course.
- Weekly exercises available latest at the start of Tuesday's laboratory session.
- Weekly exercises are to be presented and discussed on Friday, that is, prepare a few slides demonstrating your solutions.

Timetable (in calendar weeks)

Third period

Week	Description
2	Course details, Linux, Git, Python, derivatives and integration
3	Interpolation, root search, steepest descent
4	Gaussian elimination, LU decomposition, conjugate gradient, eigenvalue equations and Krylov subspace
5	Periodic boundary conditions, reciprocal lattice and Wigner-Seitz cell, crystal structure and electron densities
6-7	Solving differential equations: finite difference and finite element method
8-9	Project work I and exam week (No teaching or exercises)

Fourth period

10-15	Methods and applications
16-17	Project work II and exam week (No teaching or exercises)

Solving systems of (linear) equations

These lectures (basically) deal with numerical manipulation of matrices. Why?

- Matrix operations are present, e.g., in various scientific and data analysis related problems
- Commonly after discretization our problems reduce to matrix equations of some sort, e.g.,
  - with ordinary and partial differential equations
  - function fitting
  - etc.
- Quite often these problems reduce to finding the roots of a function, i.e., $f(\mathbf{x}) = 0$.
- It is good to be familiar with the machinery you are using.

Gaussian elimination

Let's first consider the case where $f_i(\{x_j\})$ is a linear function. In that case our problem reduces to solving a linear system of N equations and N unknowns:

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1N}x_N - b_1 & = & 0 \\ a_{21}x_1 & + & a_{22}x_2 & + & \dots & + & a_{2N}x_N - b_2 & = & 0 \\ \vdots & & & & \vdots & & & & \vdots \\ a_{N1}x_1 & + & a_{N2}x_2 & + & \dots & + & a_{NN}x_N - b_N & = & 0, \end{array} \tag{1}$$

which in matrix form is given as

$$\mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0}, \tag{2}$$

where

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots \\ a_{21} & a_{22} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \end{bmatrix}. \tag{3}$$

Gaussian elimination

The basic idea in Gaussian elimination:

- perform elementary row operations to modify the matrix until it (hopefully) becomes upper triangular matrix
  - that is, the lower left-hand corner of the matrix is full of zeros (or as much zeros as possible)

The elementary row operations are

- swapping of two rows
- multiplying a row by a nonzero number
- adding a multiple of one row to another row.

Gaussian elimination

For example, let's consider the set of equations:

$$\begin{array}{rcrcrcl} x_1 & + & x_2 & + & x_3 & = & 6 \\ -x_1 & + & 2x_2 & & & = & 3 \\ 2x_1 & & & + & x_3 & = & 5. \end{array} \tag{4}$$

We would like to eliminate the x_1 from second and third equations. How can we achieve this?

Gaussian elimination

(1) Let's add the first row to the second row, and

(2) let's subtract twice the first row from the third row.

This leads to

$$\begin{array}{rcrcrcr} x_1 & + & x_2 & + & x_3 & = & 6 \\ & + & 3x_2 & + & x_3 & = & 9 \\ & & -2x_2 & - & x_3 & = & -7. \end{array} \tag{5}$$

Then our task is to eliminate x_2 from the last equation.

We can do this by adding second row multiplied by $2/3$ to the third row. After this our set of equations looks like

Gaussian elimination

After this our set of equations looks like

$$\begin{aligned} x_1 &+ x_2 &+ x_3 &= 6 \\ &+ 3x_2 &+ x_3 &= 9 \\ & & -\tfrac{1}{3}x_3 &= -1. \end{aligned} \tag{6}$$

The procedure performed so far is called **forward elimination**, that is, we eliminate x_1 from equations 2 through N, continue to eliminate x_1 and x_2 from equations 3 through N, and so on, until the last equation only contains the variable x_N.

After this process the actual solution is straightforward to obtain by going "the ladder" in the opposite direction. This procedure is called as **back-substitution**, which in case of the above example goes as: we readily get $x_3 = 3$. Then substituting this in the second equation gives us $3x_2 + 3 = 9$, and thus, $x_2 = 2$. Finally, substituting $x_2 = 2$ and $x_3 = 3$ into the first equation, we have $x_1 + 2 + 3 = 6$ yielding $x_1 = 1$.

Gaussian elimination

In a nutshell Gaussian elimination is the process of combined forward elimination and back-substitution in solving systems of linear equations. However, due to numerical reasons the procedure is not always that straightforward, which is why so called **pivoting** is important, in practice. For this let's consider the set of equations

$$\begin{array}{rcrcrcl} \epsilon x_1 & + & x_2 & + & x_3 & = & 5 \\ x_1 & + & x_2 & & & = & 3 \\ x_1 & + & & & x_3 & = & 4, \end{array} \tag{7}$$

where in the limit $\epsilon \to 0$ the solution is $x_1 = 1$, $x_2 = 2$, and $x_3 = 3$.

In this case the forward elimination would start by multiplying the first equation by $1/\epsilon$ and subtracting it from the second and third equations. This would give us

$$\begin{array}{rcrcrcl} \epsilon x_1 & + & x_2 & + & x_3 & = & 5 \\ & & (1-1/\epsilon)x_2 & & -x_3/\epsilon & = & 3-5/\epsilon \\ & & -1/\epsilon x_2 & & (1-1/\epsilon)x_3 & = & 4-5/\epsilon, \end{array} \tag{8}$$

which obviously will lead to problems for small ϵ. What kind of problems?

Gaussian elimination

That is, within numerical precision we could have $1 - 1/\epsilon \approx -1/\epsilon$ etc.

These problems can largely be dealt with pivoting in which we ensure that the diagonal entry is as large as possible in the set of equations. In the above example this means that we interchange the first and second row, and thus, our "initial" equation is

$$\begin{array}{rcrcrcl} x_1 & + & x_2 & & & = & 3 \\ \epsilon x_1 & + & x_2 & + & x_3 & = & 5 \\ x_1 & & & + & x_3 & = & 4. \end{array} \tag{9}$$

Gaussian elimination

Now the first step in forward elimination leads to

$$\begin{array}{rcrcrcl} x_1 & + & x_2 & & & = & 3 \\ & + & (1-\epsilon)x_2 & + & x_3 & = & 5-3\epsilon \\ & & -x_2 & + & x_3 & = & 1. \end{array} \tag{10}$$

For really small ϵ the round-off leads to $1-\epsilon \approx 1$ and $5-3\epsilon \approx 5$, which again is a well behaving equation, and will give us the correct answer as $\epsilon \to 0$. Algorithms that rearrange the equations when they notice small diagonal elements are said to pivot, and this is very essential in numerical algorithms.

Gaussian elimination

Forward elimination with pivoting:

1. Find largest $|a_{ik}|$ from $i \in [k, N]$.
2. If $k \neq j$, then swap row k with row i and update $P = P + 1$.
3. Subtract $a_{jk}/a_{kk} \times \{\text{row k}\}$ from all rows for $j \in [k+1, N]$.
4. Do steps from 1. to 3. until $k + 1 = N$

Notice that the rows include the elements of vector $\mathbf{b}$ also. After forward elimination we can proceed with back-substitution as

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j=i+1}^{N} a_{ij} x_j \right) \tag{11}$$

for $i = N-1, N-2, \ldots, 1$, starting with $x_N = b_N/a_{NN}$, where all the a_{mn} and b_n are regarded as the updated values from the forward elimination (including pivoting).

Gaussian elimination

Calculation of the determinant is also often needed.

- When the original matrix is modified with the forward elimination process, the product of the diagonal elements gives the value of the determinant.
- If (and when) pivoting has been applied the value needs to be further multiplied by $(-1)^{|P|}$, where $|P|$ refers to the number of performed pivoting operations. That is, for odd number $|P|$ the product would need to multiplied by minus one.

LU decomposition

LU decomposition or lower-upper decomposition can be regarded as matrix form of Gaussian elimination. The matrix of interest $\mathbf{A}$ is expressed as $\mathbf{A} = \mathbf{LU}$, where $\mathbf{L}$ is a lower triangular matrix and $\mathbf{U}$ is an upper triangular matrix. For example, in 3×3 case we have

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}. \tag{12}$$

LU decomposition

The so called Doolittle factorization is obtained by fixing the diagonal element of $\mathbf{L}$ to 1, that is, $l_{ii} = 1$ for all i. A general recursion is expressed as

$$l_{ij} = \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right) \tag{13}$$

$$u_{ij} = \frac{1}{l_{ii}} \left(a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \right), \tag{14}$$

where $l_{i1} = a_{i1}/u_{11}$ and $u_{1j} = a_{1j}/l_{11}$. Since pivoting was noticed important (or necessary in general), the expression for LU with partial pivoting, i.e., row interchanges, takes the form

$$\mathbf{PA} = \mathbf{LU}, \tag{15}$$

where $\mathbf{P}$ is a permutation matrix. Evaluation of the determinant is given as a product of the diagonal elements, i.e., $\det(\mathbf{A}) = (-1)^{|P|} \prod_i l_{ii} u_{ii}$, where $|P|$ refers to the number of pivot operations.

LU decomposition

For solving linear equations LU decomposition takes on the form

$$\mathbf{Ax} = \mathbf{b} \quad \Longleftrightarrow \quad \mathbf{PAx} = \mathbf{Pb} \quad \Longleftrightarrow \quad \mathbf{LUx} = \mathbf{Pb}. \tag{16}$$

This is solved in two steps via equations:

$$\mathbf{Ly} = \mathbf{Pb} = \widetilde{\mathbf{b}}, \tag{17}$$

$$\mathbf{Ux} = \mathbf{y}. \tag{18}$$

If we compare the elements on both sides of these equations, we obtain

$$y_i = \frac{1}{l_{ii}} \left(\widetilde{b}_i - \sum_{k=1}^{i-1} l_{ik} y_k \right), \tag{19}$$

where $y_1 = \widetilde{b}_1 / l_{11}$ and $i = 2, 3, \ldots, N$. After this we can obtain the solution for $\mathbf{x}$ as

$$x_i = \frac{1}{u_{ii}} \left(y_i - \sum_{k=i+1}^{N} u_{ik} x_k \right), \tag{20}$$

where $i = N-1, N-2, \ldots, 1$ and $x_N = y_N / u_{NN}$.

Conjugate Gradient

Conjugate gradient (CG) can be regarded as a more sophisticated version of steepest descent/gradient descent. In CG we modify the search direction by also utilizing the knowledge of previous search directions. This is based on forming a set of mutually conjugate vectors. Suppose that $P = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N\}$ is a set of N mutually conjugate vectors, that is, $\mathbf{p}_i^T \mathbf{A} \mathbf{p}_j = 0$ for all $i \neq j$. In this case the N vectors in the set are linearly independent, and thus, the set P forms a complete basis. Therefore, the solution to $\mathbf{A}\mathbf{x} = \mathbf{b}$ is given as

$$\mathbf{x} = \sum_{i=1}^{N} a_i \mathbf{p}_i. \tag{21}$$

Substituting this into our linear equation yields

$$\mathbf{A}\mathbf{x} = \sum_{i=1}^{N} a_i \mathbf{A}\mathbf{p}_i \quad \Longleftrightarrow \quad \mathbf{b} = \sum_{i=1}^{N} a_i \mathbf{A}\mathbf{p}_i. \tag{22}$$

Conjugate Gradient

From there we can obtain a coefficient a_k by multiplying the previous from the left by $\mathbf{p}_k^T$, since $\mathbf{p}_k^T A \mathbf{p}_i = 0$ for all $i \neq k$. This leads to

$$a_k = \frac{\mathbf{p}_k^T \mathbf{b}}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}, \tag{23}$$

which implies that for solving the equation we would need to look for the conjugate directions.

It turns out that a new conjugate vector $\mathbf{p}_{k+1}$ can be computed by using the previous vector $\mathbf{p}_k$; still ensuring that the new vector is conjugate to all the previous ones. Each new direction is chosen as a linear combination of the negative residual $-\mathbf{r}_{k+1} = \mathbf{b} - \mathbf{A}\mathbf{x}_{k+1}$ and the previous search vector $\mathbf{p}_k$.

For a quadratic function the negative residual is equal to the negative gradient direction, i.e., the steepest descent direction.

Conjugate Gradient

The equation for determining the new step becomes

$$\mathbf{p}_{k+1} = -\mathbf{r}_{k+1} + \beta_k \mathbf{p}_k, \tag{24}$$

where by multiplying with $\mathbf{p}_k^T \mathbf{A}$ and rearranging we get

$$\beta_k = \frac{\mathbf{p}_k^T \mathbf{A} \mathbf{r}_{k+1}}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k} = \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}. \tag{25}$$

Therefore, a basic conjugate gradient algorithm is

1. Compute $\mathbf{r}_0 = \mathbf{A}\mathbf{x}_0 - \mathbf{b}$ and $\mathbf{p}_0 = -\mathbf{r}_0$.
2. Compute until convergence

$$\begin{aligned} a_k &= \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k} \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + a_k \mathbf{p}_k \\ \mathbf{r}_{k+1} &= \mathbf{r}_k + a_k \mathbf{A} \mathbf{p}_k \\ \beta_k &= \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k} \\ \mathbf{p}_{k+1} &= -\mathbf{r}_{k+1} + \beta_k \mathbf{p}_k. \end{aligned} \tag{26}$$

Conjugate Gradient

In general so called preconditioning is needed for fast convergence. This is accomplished by adding a symmetric positive definite matrix $\mathbf{M}$ in the process. This matrix needs to be fixed and the algorithm will be only slightly modified, i.e.,

1. Compute $\mathbf{r}_0 = \mathbf{A}\mathbf{x}_0 - \mathbf{b}$, $\mathbf{z}_0 = \mathbf{M}^{-1}\mathbf{r}_0$ and $\mathbf{p}_0 = -\mathbf{z}_0$.
2. Compute until convergence

$$
\begin{aligned}
a_k &= \frac{\mathbf{r}_k^T \mathbf{z}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k} \\
\mathbf{x}_{k+1} &= \mathbf{x}_k + a_k \mathbf{p}_k \\
\mathbf{r}_{k+1} &= \mathbf{r}_k + a_k \mathbf{A} \mathbf{p}_k \\
\mathbf{z}_{k+1} &= \mathbf{M}^{-1} \mathbf{r}_{k+1} \\
\beta_k &= \frac{\mathbf{z}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{z}_k^T \mathbf{r}_k} \\
\mathbf{p}_{k+1} &= -\mathbf{z}_{k+1} + \beta_k \mathbf{p}_k .
\end{aligned}
\tag{27}
$$

Eigenvalues, eigenvectors, and Krylov subspace

Many physical (as well as other) problems lead to eigenvalue equations, i.e.,

$$\mathbf{A}\mathbf{v}_k = \lambda_k \mathbf{v}_k, \tag{28}$$

where $\mathbf{v}_k$ is a (normalized) eigenvector corresponding to (non-degenerate) eigenvalue λ_k. Let's assume that the eigenvectors form a complete space, and thus, any vector can be expressed as

$$\mathbf{x} = \sum_{i=1}^{N} c_i \mathbf{v}_i. \tag{29}$$

If $c_i \neq 0$, then the vector $\mathbf{x}$ is not orthogonal to any of the eigenvectors.

Eigenvalues, eigenvectors, and Krylov subspace

Multiplying the vector $\mathbf{x}$ n times by matrix $\mathbf{A}$ gives us

$$\mathbf{A}^n\mathbf{x} = \mathbf{A}^n \left[\sum_{i=1}^{N} c_i \mathbf{v}_i\right] = \sum_{i=1}^{N} c_i \lambda_i^n \mathbf{v}_i. \tag{30}$$

When the eigenvalues are ordered by decreasing absolute magnitude we get for large n limit

$$\mathbf{A}^n\mathbf{x} \approx c_1 \lambda_1^n \mathbf{v}_1. \tag{31}$$

Therefore, for large n the normalized iteration vector

$$\mathbf{b}_n = \frac{\mathbf{A}^n\mathbf{x}}{|\mathbf{A}^n\mathbf{x}|} \approx \mathbf{v}_1, \tag{32}$$

where the approximate symbol becomes exact in the limit $n \longrightarrow \infty$. This is a so-called **power method** that is a straightforward way for finding one eigenvalue–eigenvector pair corresponding to the largest eigenvalue ($|\lambda_1|$).

Eigenvalues, eigenvectors, and Krylov subspace

The method can also be extended to obtain the next largest eigenvalue, e.g., by forming a new matrix

$$\widetilde{\mathbf{A}} = \mathbf{A} - \lambda_1 \frac{\mathbf{v}_1 \mathbf{v}_1^T}{\mathbf{v}_1^T \mathbf{v}_1}. \tag{33}$$

The eigenvalues of $\widetilde{\mathbf{A}}$ are the same as with $\mathbf{A}$ apart from λ_1, which is now "set" to zero. If the eigenvectors are orthogonal to each other, then also the eigenvectors are the same for the two matrices, since in that case $\mathbf{v}_1^T \mathbf{v}_k = 0$ for all $k \neq 1$. That is,

$$\widetilde{\mathbf{A}}\mathbf{v}_1 = \mathbf{A}\mathbf{v}_1 - \lambda_1 \frac{\mathbf{v}_1 \mathbf{v}_1^T \mathbf{v}_1}{\mathbf{v}_1^T \mathbf{v}_1} = \lambda_1 \mathbf{v}_1 - \lambda_1 \mathbf{v}_1 = 0, \tag{34}$$

$$\widetilde{\mathbf{A}}\mathbf{v}_2 = \mathbf{A}\mathbf{v}_2 - \lambda_1 \frac{\mathbf{v}_1 \mathbf{v}_1^T \mathbf{v}_2}{\mathbf{v}_1^T \mathbf{v}_1} = \lambda_2 \left(\mathbf{v}_2 - \frac{\lambda_1}{\lambda_2} \frac{\mathbf{v}_1^T \mathbf{v}_2}{\mathbf{v}_1^T \mathbf{v}_1} \mathbf{v}_1 \right), \tag{35}$$

which demonstrates that the eigenvectors are preserved if and only if $\mathbf{v}_1^T \mathbf{v}_2 = 0$. Notice that the vector in parenthesis after λ_2 is an eigenvector for $\widetilde{\mathbf{A}}$ with eigenvalue λ_2, since $\widetilde{\mathbf{A}}\mathbf{v}_1 = \mathbf{0}$.

Eigenvalues, eigenvectors, and Krylov subspace

Some algorithms take advantage of the whole subspace generated by the vectors $\{\mathbf{b}_i\}$ for $i = 0, 1, \ldots, n$.

Since these vectors are not generally orthogonal, an orthogonalization procedure such as Gram-Schmidt can be used in obtaining an orthogonal (or orthonormal basis).

An orthonormal subspace generalized in this way is known as the **Krylov subspace**, which is commonly computed by Arnoldi iteration (general matrices) or Lanczos iteration (Hermitian matrices). Also conjugate gradient method can be used in generating the Krylov subspace.

Since for large n the vector $\mathbf{b}_n$ is a good approximation for the eigenvector corresponding to the largest eigenvalue, it is expected that the vectors in the Krylov subspace result in good approximations of the eigenvectors for the n largest eigenvalues.

Eigenvalues, eigenvectors, and Krylov subspace

The **Gram-Schmidt orthogonalization** process would be described as follows:

$$\mathbf{u}_0 = \mathbf{b}_0, \tag{36}$$

$$\mathbf{u}_1 = \mathbf{b}_1 - \frac{\mathbf{u}_0^T \mathbf{b}_1}{\mathbf{u}_0^T \mathbf{u}_0} \mathbf{u}_0 \tag{37}$$

$$\mathbf{u}_2 = \mathbf{b}_2 - \frac{\mathbf{u}_1^T \mathbf{b}_2}{\mathbf{u}_1^T \mathbf{u}_1} \mathbf{u}_1 - \frac{\mathbf{u}_0^T \mathbf{b}_2}{\mathbf{u}_0^T \mathbf{u}_0} \mathbf{u}_0 \tag{38}$$

$$\vdots$$

$$\mathbf{u}_n = \mathbf{b}_n - \sum_{i=0}^{n-1} \frac{\mathbf{u}_i^T \mathbf{b}_n}{\mathbf{u}_i^T \mathbf{u}_i} \mathbf{u}_i . \tag{39}$$

Once these $\mathbf{u}_i$ are also normalized, then the set is a proper Krylov subspace.