6. Iterative Methods: Roots and Optima

Citius, Altius, Fortius!

6.1. Large Sparse Systems of Linear Equations I – Relaxation Methods

Introduction

- Systems of linear equations, which have to be solved numerically, often stem from the discretization of ordinary (for boundary value problems see chapter 5) or partial differential equations. The **direct** solving methods studied in chapter 4 usually are not applicable:
  - First, n most times is so big (usually n directly correlates with the number of grid points, which leads to very big n, especially for unsteady partial differential equations (three spatial and one time variable)), such that a computational cost of the complexity $O(n^3)$ is not acceptable.
  - Second, such matrices are usually **sparsely populated** (i.e. $O(1)$ or $O(\log n)$ non-zero values per row, e. g.) and have a certain **structure** (tridiagonal, band structure, block structure etc.), which, of course, has a positive effect on memory and computing time; methods of elimination often destroy this structure and, therefore, undo those structural advantages.

- For large and sparse matrices or systems of linear equations, therefore, we prefer **iterative** methods:
  - Those start with an *initial approximation* $x^{(0)}$ and generate a sequence of approximate values $x^{(i)}$, $i = 1, 2, \ldots$, which, in case of convergence, converges to the exact solution x.
  - One iteration step typically costs $O(n)$ arithmetic operations in case of a sparse matrix (less is hardly possible if you want to update every vector component of $x^{(i)}$ in every step). Therefore, the construction of iterative algorithms depends on how many iteration steps are required to obtain a certain accuracy.

Overview of Relaxation Methods

- The probably oldest iterative methods for solving systems of linear equations $Ax = b$ with $A \in \mathbb{R}^{n,n}$ and $x, b \in \mathbb{R}^n$ are the so-called **relaxation** or **smoothing methods**:
  - the **Richardson** method,
  - the **Jacobi** method,
  - the **Gauss-Seidel** method as well as
  - SOR, the **successive over-relaxation**.
- Basic principles:
  - For all methods mentioned, the starting point is the **residual** $r^{(i)}$ already introduced,

$$r^{(i)} := b - Ax^{(i)} = Ax - Ax^{(i)} = A\left(x - x^{(i)}\right) = -Ae^{(i)},$$

    where $x^{(i)}$ is the current approximation for the exact solution x after i iteration steps and $e^{(i)}$ denotes the respective error.
  - As $e^{(i)}$ is not available (the error can not be determined if the exact solution x is unknown), due to the relation above, it turns out to be reasonable to take the vector $r^{(i)}$ as *direction* in which we want to look for an update of $x^{(i)}$.

– The Richardson method directly takes the residual as adjustment for $x^{(i)}$. The Jacobi and the Gauss-Seidel method try harder. Their idea to adjust the k-th component of $x^{(i)}$ is to eliminate $r_k^{(i)}$. The SOR method and its counterpart, the **damped** relaxation, additionally take into account that such an update often either overshoots the mark or is not sufficient.

Important Methods of Relaxation

- **Richardson iteration**:

```
for i = 0,1,...
    for k = 1,...,n:
```

$$x_k^{(i+1)} := x_k^{(i)} + r_k^{(i)}$$

  Here, the residual $r^{(i)}$ is simply used componentwise as update to adjust the active approximation $x^{(i)}$.

- **Jacobi iteration**:

```
for i = 0,1,...
    for k = 1,...,n:
    for k = 1,...,n:
```

$$y_k := \frac{1}{a_{kk}} \cdot r_k^{(i)}$$

$$x_k^{(i+1)} := x_k^{(i)} + y_k$$

  - In every substep k of a step i, an update y_k is computed and stored.
  - Applied immediately, this would lead to the (momentary) disappearance of the k-th component of the residual $r^{(i)}$ (easy to verify by inserting).
  - With this current approximation for x, equation k would therefore be solved exactly – an improvement that would be lost, of course, in the following substep for the equation $k+1$.
  - However, these updates of a component are not carried out immediately, but only at the end of an iteration step (second k-loop).

Important Methods of Relaxation (2)

- **Gauss-Seidel iteration**:

```
for i = 0,1,...
    for k = 1,...,n:
```

$$r_k^{(i)} := b_k - \sum_{j=1}^{k-1} a_{kj} x_j^{(i+1)} - \sum_{j=k}^{n} a_{kj} x_j^{(i)}$$

$$y_k := \frac{1}{a_{kk}} \cdot r_k^{(i)}, \quad x_k^{(i+1)} := x_k^{(i)} + y_k$$

  - In principle, the update calculated here is almost the same as in the Jacobi method. However, the update is not performed at the end of the iteration step, but always immediately.
  - Therefore, the new modified values for the components 1 to $k-1$ are already available for the update of component k.

- Sometimes, a **damping** (multiplying the update with a factor $0 < \alpha < 1$) or an **over-relaxation** (factor $1 < \alpha < 2$) leads to better convergence behavior for each of the three methods outlined:

$$x_k^{(i+1)} := x_k^{(i)} + \alpha y_k \, .$$

  - In the case of Gauss-Seidel method, the version with $\alpha > 1$ is mainly used. It is denoted as **SOR method** (**successive over-relaxation**).
  - In the case of Jacobi method, in contrary, damping mostly is in use.

Discussion: Additive Decomposition of the System Matrix

- In order to quickly analyze the convergence of the methods above, we need an algebraic formulation (instead of the algorithmic one).
- Every approach shown is based on the simple idea of writing the matrix A as a sum $A = M + (A - M)$, where $Mx = b$ is quite simple to solve and the difference $A - M$ should not be too big with regard to some matrix norm.
- With the help of such an adequate M, we will be able to write the Richardson, Jacobi, Gauss-Seidel, and SOR methods as

$$Mx^{(i+1)} + (A - M)x^{(i)} = b$$

or, solved for $x^{(i+1)}$, as:

$$x^{(i+1)} := M^{-1}b - M^{-1}(A-M)x^{(i)} = M^{-1}b - (M^{-1}A - I)x^{(i)} = x^{(i)} + M^{-1}r^{(i)} .$$

- Furthermore, we decompose A additively in its diagonal part D_A, its strict lower triangular part L_A as well as its strict upper triangular part U_A:

$$A =: L_A + D_A + U_A .$$

With this, we can show the following relations:

- Richardson: $M := I$,
- Jacobi: $M := D_A$,
- Gauss-Seidel: $M := D_A + L_A$,
- SOR: $M := \frac{1}{\alpha} D_A + L_A$.

Discussion: Additive Decomposition of the System Matrix (2)

- Considering the algorithmic formulation of the Richardson as well as that of the Jacobi method, the first two equations are obvious:
  - At Richardson, the residual is directly used as update. Therefore, the identity I results as the prefactor M.
  - At Jacobi, the residual is divided by the diagonal element. Therefore, the inverse of the diagonal part D_A results as the prefactor M.
- As the Gauss-Seidel iteration is a special case of the SOR method ($\alpha = 1$), it's sufficient to prove the formula above for M in the general SOR case. From the algorithm, it follows immediately

$$
\begin{aligned}
& x_k^{(i+1)} := x_k^{(i)} + \alpha \left(b_k - \sum_{j=1}^{k-1} a_{kj} x_j^{(i+1)} - \sum_{j=k}^{n} a_{kj} x_j^{(i)} \right) / a_{kk} \\
\Leftrightarrow \quad & x^{(i+1)} := x^{(i)} + \alpha D_A^{-1} \left(b - L_A x^{(i+1)} - (D_A + U_A) x^{(i)} \right) \\
\Leftrightarrow \quad & \frac{1}{\alpha} D_A x^{(i+1)} = \frac{1}{\alpha} D_A x^{(i)} + b - L_A x^{(i+1)} - (D_A + U_A) x^{(i)} \\
\Leftrightarrow \quad & \left(\frac{1}{\alpha} D_A + L_A \right) x^{(i+1)} + \left((1 - \frac{1}{\alpha}) D_A + U_A \right) x^{(i)} = b \\
\Leftrightarrow \quad & M x^{(i+1)} + (A - M) x^{(i)} = b \,,
\end{aligned}
$$

  which proves the statement for the SOR method.

Discussion: General Behavior of Convergence

- As far as convergence is concerned, two immediate consequences follow from the approach

$$Mx^{(i+1)} + (A - M)x^{(i)} = b :$$

  - If the sequence $\left(x^{(i)}\right)$ is convergent, then the exact solution x of our system $Ax = b$ is the limit.
  - For the analysis, assume that the iteration matrix $-M^{-1}(A - M)$ (i.e. the matrix that is applied to $e^{(i)}$ to get $e^{(i+1)}$; see below) is symmetric. Then the spectral radius ρ (i.e. the eigenvalue with the biggest absolute value) is the decisive parameter for convergence:

$$\left(\forall x^{(0)} \in \mathbb{R}^n : \lim_{i\to\infty} x^{(i)} = x = A^{-1}b\right) \quad \Leftrightarrow \quad \rho < 1\,.$$

    To see that, subtract $Mx + (A - M)x = b$ from the equation at the very top:

$$Me^{(i+1)} + (A - M)e^{(i)} = 0 \quad \Leftrightarrow \quad e^{(i+1)} = -M^{-1}(A - M)e^{(i)}\,.$$

    If all absolute values of the eigenvalues are smaller than 1 and, therefore, $\rho < 1$ holds, all error components are reduced in every iteration step. In case of $\rho > 1$, at least one error component will build up.
  - The goal of the construction of iteration matrix must of course be a spectral radius of the iterative matrix that is as small as possible (as close to zero as possible).

Discussion: Convergence Statements

- There are a number of results for the convergence of the different methods of which we should mention a few important ones:
  - For the convergence of the SOR method, $0 < \alpha < 2$ is necessary.
  - If A is positive definite, the SOR method (for $0 < \alpha < 2$) as well as the Gauss-Seidel iteration are convergent.
  - If A and $2D_A - A$ are both positive definite, the Jacobi method converges.
  - If A is strictly diagonally dominant (i.e. $a_{ii} > \sum_{j \neq i} \mid a_{ij} \mid$ for all i), then the Jacobi and the Gauss-Seidel method are convergent.
  - In certain cases, the ideal parameter α can be determined (ρ minimal, such that the error reduction per iteration step is maximal).
- The Gauss-Seidel iteration is not generally better than the Jacobi method (as you might assume because of the update performed immediately). There are examples in which the former converges and the latter diverges or the other way round. However, in many cases, the Gauss-Seidel method does with only half of the iteration steps of the Jacobi method.

The Spectral Radius of Typical Iterative Matrices

- Obviously, ρ is not only decisive for the question whether the iteration converges at all, but also for its quality, i.e. its speed of convergence: The smaller ρ is, the faster all components of the error $e^{(i)}$ are reduced in every iteration step.
- In practice, the results about convergence above unfortunately have rather theoretical value because ρ is often so close to 1 that – in spite of convergence – the number of iteration steps necessary to reach a sufficient accuracy is way too big.
- An important sample scenario is the discretization of partial differential equations:
  - It is characteristic that ρ depends on the dimension n of the problem and, therefore, on the resolution h of the underlying grid, i.e. for example

$$\rho = O(1 - h_l^2) = O\left(1 - \frac{1}{4^l}\right)$$

    with the mesh width $h_l = 2^{-l}$.
  - This is a huge disadvantage: The finer and, therefore, the more precise our grid is, the more miserable the behavior of convergence of our iterative methods gets. Therefore, better iterative solvers are a must (e.g. multigrid methods, cf. last section)!

6.2. Large Sparse Systems of Linear Equations II – Minimization Methods

Formulation as a Problem of Minimization

- One of the best known solution methods, the method of **conjugate gradients (cg)**, is based on a different principle than relaxation or smoothing. To see that, we will tackle the problem of solving systems of linear equations using a detour.
- In the following, let $A \in \mathbb{R}^{n,n}$ be a symmetric and positive definite matrix, i.e. $A = A^T$ and $x^T Ax > 0 \; \forall x \neq 0$. In this case, solving the linear system $Ax = b$ is equivalent to minimizing the quadratic function

$$f(x) := \frac{1}{2} x^T Ax - b^T x + c$$

for an arbitrary scalar constant $c \in \mathbb{R}$.

- As A is positive definite, the hyperplane given by $z := f(x)$ defines a paraboloid in $\mathbb{R}^{n+1}$ with n-dimensional ellipsoids as isosurfaces $f(x) = const.$, and f has a global minimum in x. The equivalence of the problems is obvious:

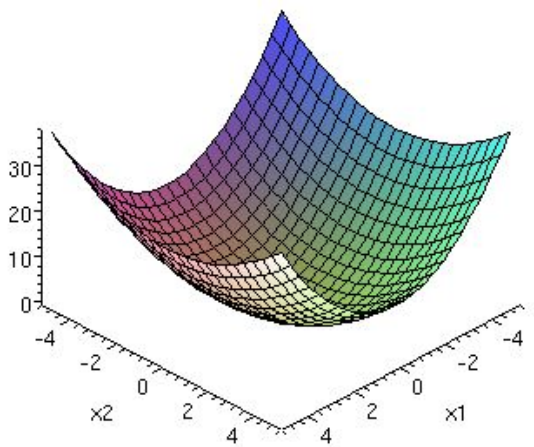
$$f'(x) = \frac{1}{2} A^T x + \frac{1}{2} Ax - b = Ax - b = -r(x) = 0 \quad \Leftrightarrow \quad Ax = b\,.$$

Clearly: Every disturbance $e \neq 0$ of the solution x of $Ax = b$ increases the value of f. Thus, the point x is in deed the minimum of f:

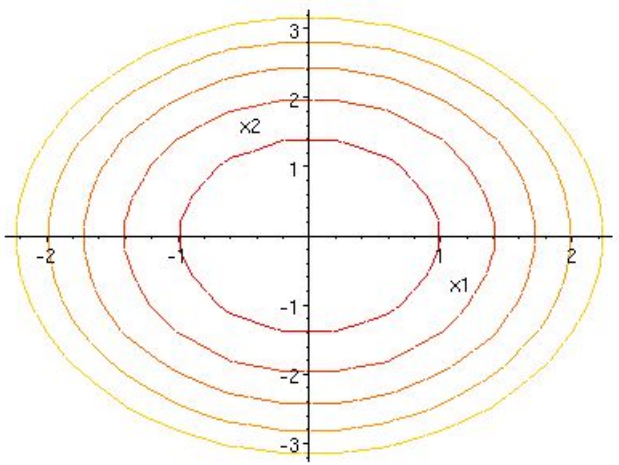
$$\begin{aligned} f(x+e) &= \frac{1}{2}x^T Ax + e^T Ax + \frac{1}{2}e^T Ae - b^T x - b^T e + c \\ &= f(x) + e^T(Ax - b) + \frac{1}{2}e^T Ae \\ &= f(x) + \frac{1}{2}e^T Ae \; > \; f(x)\,. \end{aligned}$$

- Example: $n = 2, A = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}, \; b = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \; c = 0$

$$f(x) = x_1^2 + \frac{1}{2}x_2^2$$



graph of $f(x)$



isolines $f(x) = c$

The Method of Steepest Descent

- What do we gain by reformulating? We can also apply *optimization methods* and, therefore, increase the set of possible solvers now.
- Let's examine the techniques to find minima. An obvious possibility is provided by the **method of steepest descent**. For $i = 0, 1, \ldots$, repeat

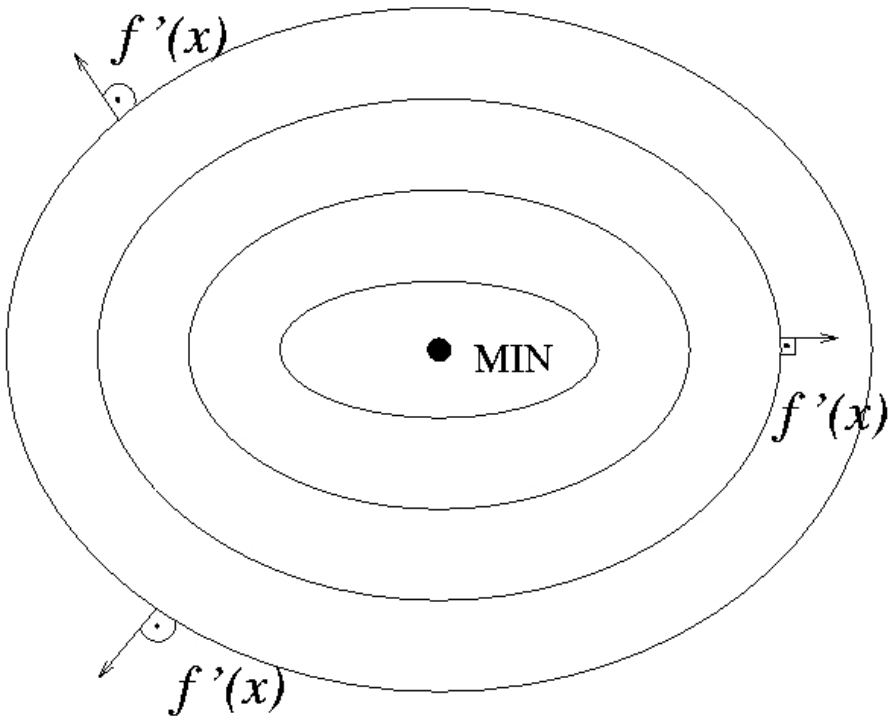
$$\begin{aligned} r^{(i)} &:= b - Ax^{(i)}, \\ \alpha_i &:= \frac{{r^{(i)}}^T r^{(i)}}{{r^{(i)}}^T A r^{(i)}}, \\ x^{(i+1)} &:= x^{(i)} + \alpha_i r^{(i)}, \end{aligned}$$

or start with $r^{(0)} := b - Ax^{(0)}$ and repeat for $i = 0, 1, \ldots$

$$\begin{aligned} \alpha_i &:= \frac{{r^{(i)}}^T r^{(i)}}{{r^{(i)}}^T A r^{(i)}}, \\ x^{(i+1)} &:= x^{(i)} + \alpha_i r^{(i)}, \\ r^{(i+1)} &:= r^{(i)} - \alpha_i A r^{(i)}, \end{aligned}$$

which saves one of the two matrix vector products (the only really expensive step of the algorithm).

- The method of steepest descent tries to find an update in the direction of the negative gradient $-f'(x^{(i)}) = r^{(i)}$, which indeed indicates the steepest descent (thus the name). Of course, it would be better to search in the direction of the error $e^{(i)} := x^{(i)} - x$, but the latter is unfortunately unknown.
- However, the direction itself is not sufficient, an adequate step width is also needed. To find one, we try to find the minimum of $f(x^{(i)} + \alpha_i r^{(i)})$ as a function of α_i (set the partial derivative with respect to α_i to zero), which, after a short calculation, provides the value above for α_i.



Discussion of the Steepest Descent

- By the way, when choosing unit vectors in the coordinate directions instead of the residuals $r^{(i)}$ as search directions and determining the optimal step width concerning those search directions again, the Gauss-Seidel iteration results. Despite of all differences, there are still similarities between the approach of relaxation and that of minimization!
- The convergence behavior of the method of steepest descent is poor. One of the few trivial special cases is the identity matrix: Here, the isosurfaces are spheres, the gradient always points towards the center (the minimum), and we reach our goal with only one step! Generally, we eventually get arbitrarily close to the minimum but that can also last arbitrarily long (as we can always destroy something of the already achieved).
- To eliminate this drawback, we stick to our minimization approach but keep looking for better search directions. If all search directions were orthogonal, and if the error was orthogonal to all previous search directions after i steps, then the results already achieved would never be lost again, and after at most n steps we would be at the minimum – just as with a direct solver. For this reason, the cg method and similar methods are also called **semi-iterative methods**.

Conjugate Directions

- We will now look for alternative search directions $d^{(i)}$, $i = 0, 1, \ldots$, instead of the residuals or negative gradients $r^{(i)}$:

$$x^{(i+1)} := x^{(i)} + \alpha_i d^{(i)} .$$

  - For a short moment, let's think about the ideal case of above: Let the active error $e^{(i)}$ be orthogonal to the subspace spanned by $d^{(0)}, \ldots, d^{(i-1)}$. Then, the new update $\alpha_i d^{(i)}$ has to be chosen appropriately in a way such that $d^{(i)}$ is orthogonal to $d^{(0)}, \ldots, d^{(i-1)}$ as well and that the new error $e^{(i+1)}$ is additionally orthogonal to $d^{(i)}$, i.e.

$$\begin{aligned} 0 &= {d^{(i)}}^T e^{(i+1)} = {d^{(i)}}^T \left(e^{(i)} + \alpha_i d^{(i)} \right) \\ \Rightarrow \quad \alpha_i &:= -\frac{{d^{(i)}}^T e^{(i)}}{{d^{(i)}}^T d^{(i)}} . \end{aligned}$$

  - Even if someone gave us the new direction $d^{(i)}$ as a gift, it would be of no use, as we do not know $e^{(i)}$. Therefore, we once again consider the residuals instead of errors and construct **A-orthogonal** or **conjugate** search directions instead of orthogonal ones.

– Two vectors $u \neq 0$ and $v \neq 0$ are called **A-orthogonal** or **conjugate**, if $u^T A v = 0$. The according requirement for $e^{(i+1)}$ is now

$$\begin{aligned} 0 &= {d^{(i)}}^T A e^{(i+1)} = {d^{(i)}}^T A \left(e^{(i)} + \alpha_i d^{(i)} \right) \\ \Rightarrow \quad \alpha_i &:= -\frac{{d^{(i)}}^T A e^{(i)}}{{d^{(i)}}^T A d^{(i)}} = \frac{{d^{(i)}}^T r^{(i)}}{{d^{(i)}}^T A d^{(i)}} . \end{aligned}$$

– Effectively, A is “inserted” into the usual definition of orthogonality – in case of the identity ($A = I$), there is no change.

Conjugate Directions (2)

- Interestingly, the mentioned constraint is equivalent to trying to find the minimum in the direction $d^{(i)}$, the way we did it for the steepest descent:

$$0 = \frac{\partial f}{\partial \alpha_i}(x^{(i+1)}) = f'(x^{(i+1)})\frac{\partial x^{(i+1)}}{\partial \alpha_i} = -r^{(i+1)^T} d^{(i)} = d^{(i)^T} A e^{(i+1)} .$$

- In contrast to what has been said before, here, all terms are known. Therefore, we get the following preliminary algorithm – 'preliminary' because we still don't know how to get the conjugate search directions $d^{(i)}$:
  - We begin with $d^{(0)} := r^{(0)} = b - Ax^{(0)}$ and iterate for $i = 0, 1, \ldots$:

$$\begin{aligned} \alpha_i &:= \frac{d^{(i)^T} r^{(i)}}{d^{(i)^T} A d^{(i)}} , \\ x^{(i+1)} &:= x^{(i)} + \alpha_i d^{(i)} , \\ r^{(i+1)} &:= r^{(i)} - \alpha_i A d^{(i)} . \end{aligned}$$

  - A comparison of this method with steepest descent shows the close relation between those two methods.

How to Construct Conjugate Directions

- Therefore, the only problem remaining is how to construct the conjugate directions. This can be done, for example, with a **Gram-Schmidt A-process**, which works totally analogously to the common Gram-Schmidt process (we start with a basis $u^{(0)}, \ldots, u^{(n-1)}$ and subtract all fractions that are not A-orthogonal to $u^{(0)}, \ldots, u^{(i-1)}$ from $u^{(i)}$).
- In case of a basis of unit vectors, we can prove that the resulting algorithm is equivalent to the Gaussian elimination – thus, we have closed the circle. However, that was not what we intended, there must be a cheaper way suitable for an iteration.
- If the unit vectors can't help us, we'll once again have to choose the residuals $r^{(i)}$ as the basis to be conjugated. An analysis of this procedure reveals surprising facts:
  - The residuals are orthogonal: ${r^{(i)}}^T r^{(j)} = 0$ for $i \neq j$.
  - $r^{(i+1)}$ is *orthogonal* to $d^{(0)}, \ldots, d^{(i)}$ – therefore, the error $e^{(i+1)}$ is *conjugate* to $d^{(0)}, \ldots, d^{(i-1)}$ (as required).
  - $r^{(i+1)}$ is *conjugate* to $d^{(0)}, \ldots, d^{(i-1)}$.
- Nearly everything we need arose automatically, only a conjugation of $r^{(i+1)}$ concerning $d^{(i)}$ is still necessary and has to be done explicitly.
- Therefore, we have all the ingredients for the CG algorithm.

The Conjugate Gradient Method (CG)

- We start with $d^{(0)} := r^{(0)} = b - Ax^{(0)}$ and iterate for $i = 0, 1, \ldots$:

$$\begin{aligned}
\alpha_i &:= \frac{{r^{(i)}}^T r^{(i)}}{{d^{(i)}}^T A d^{(i)}}, \\
x^{(i+1)} &:= x^{(i)} + \alpha_i d^{(i)}, \\
r^{(i+1)} &:= r^{(i)} - \alpha_i A d^{(i)}, \\
\beta_{i+1} &:= \frac{{r^{(i+1)}}^T r^{(i+1)}}{{r^{(i)}}^T r^{(i)}}, \\
d^{(i+1)} &:= r^{(i+1)} + \beta_{i+1} d^{(i)}.
\end{aligned}$$

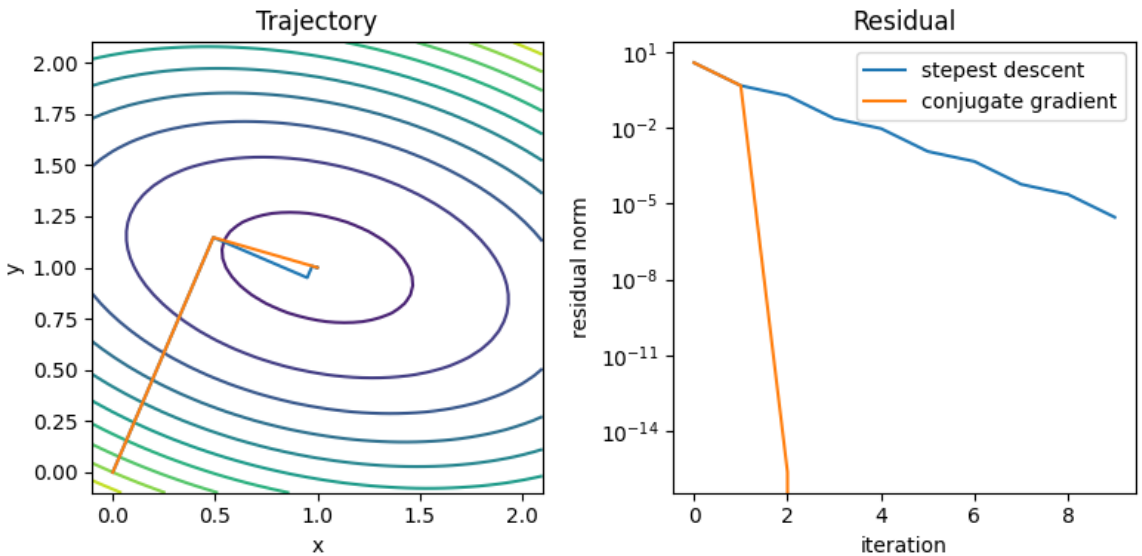
- The CG algorithm can also be formulated in a very compact way, is elegant, and, in addition to that, even very efficient: The construction of the conjugate directions costs only the calculation of a constant β_{i+1} per step as well as an SAXPY-operation (multiplication of a vector with a scalar plus a vector addition) for $d^{(i+1)}$.
- Two remarks:
  - First, since ${d^{(i)}}^T r^{(i)} = {r^{(i)}}^T r^{(i)}$, α_i can be computed without the d-scalar product.
  - Second, the conjugation of $r^{(i+1)}$ concerning $d^{(i)}$ appears in the parameter β_{i+1}.

Discussion of the CG Method

- In principle, the CG algorithm is a direct solver. However, numerical problems (rounding errors and cancellation, which can especially perturb the A-orthogonality of the search directions) as well as the mere size of n in realistic applications are responsible for it being used almost only as an iterative method today (almost nobody could await n iterative steps).
- For the CG method – as for the relaxation methods – the convergence speed for discretized differential equations, unfortunately, depends on the mesh width: the finer the solution, the slower the convergence.
- For the CG method, the magic word **preconditioning** puts things right:
  - Solve $M^{-1}Ax = M^{-1}b$ instead of $Ax = b$, where the **preconditioner** M arranges for a crucial improvement of the behavior of convergence (since the new system matrix now is $M^{-1}A$).
  - Of course, the choice $M^{-1} := A^{-1}$ would be ideal, but then, computing the preconditioner would be more expensive than solving the initial problem.
  - The trick is to construct an effective preconditioner in a cheap way. For this, the principle of multigrid, that will be explained in the last section, plays a crucial role.

Example

- We want to solve the linear system of equations $\begin{pmatrix} 1 & 0.5 \\ 0.5 & 3 \end{pmatrix} x = \begin{pmatrix} 1.5 \\ 3.5 \end{pmatrix}$.
- The exact solution is $x^* = \begin{pmatrix} 1 & 1 \end{pmatrix}^T$.
- We use the method of steepest descent and the method of conjugate gradients with the initial guess $x_0 = \begin{pmatrix} 0 & 0 \end{pmatrix}^T$.
- We see that the residual for the method of steepest descent decays slowly, whereas the conjugate gradient method finds the exact solution within two steps.



6.3. Non-Linear Equations

Introductory Remarks

- Now, we know how to solve systems of linear equations, directly or iteratively. By the way, the solution of $Ax = b$ can also be interpreted as root-finding for the linear function $F : \mathbb{R}^n \to \mathbb{R}^n,\ F(x) := b - Ax$.
- Unfortunately, the world is not linear but non-linear. Therefore, a numerical programmer also has to deal with solving **non-linear equations**. Usually, it is impossible to do this directly, but it has to be done iteratively. As we do not want to use the calculus of multiple variables at this point, we will confine ourselves to the simple case of $n = 1$ in the following (i.e. *one* non-linear equation).
- Let's consider a continuously differentiable (non-linear) function $f : \mathbb{R} \to \mathbb{R}$, which has a root $\bar{x} \in]a, b[$ (think of finding roots or extrema).
- Assume an iterative method (yet to be determined):
  - It provides a sequence of approximate values, $(x^{(i)})$, $i = 0, 1, \ldots$, that (hopefully) converge to the root, or rather one root $\bar{x}$ of $f(x)$.

– As a measure of the **convergence speed**, we examine the reduction of the error in every step. In the case of convergence,

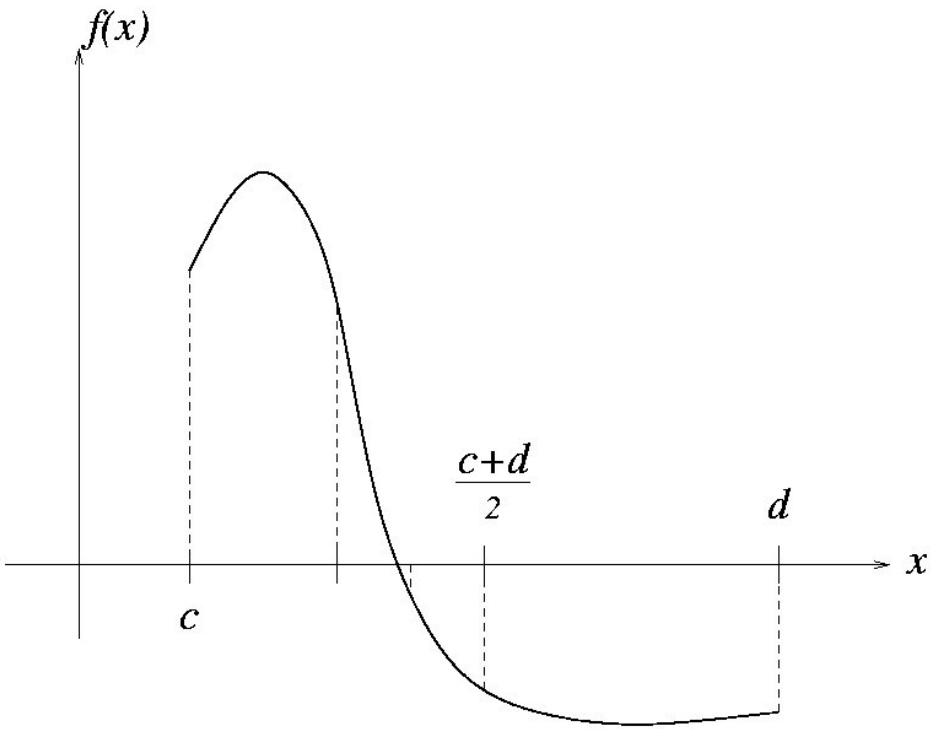
$$| x^{(i+1)} - \bar{x} | \leq c \cdot | x^{(i)} - \bar{x} |^{\alpha} ,$$

according to the maximum possible value of the parameter α, we speak of **linear** ($\alpha = 1$ and, in addition, $0 < c < 1$) or **quadratic** ($\alpha = 2$) convergence and so on.

– There are **conditionally** or **locally** convergent methods, whose convergence is only secured when the start value $x^{(0)}$ is already sufficiently good, and **unconditionally** or **globally** convergent methods, for which the iteration leads to a root of f independent of the choice of starting point.

Three Simple Methods

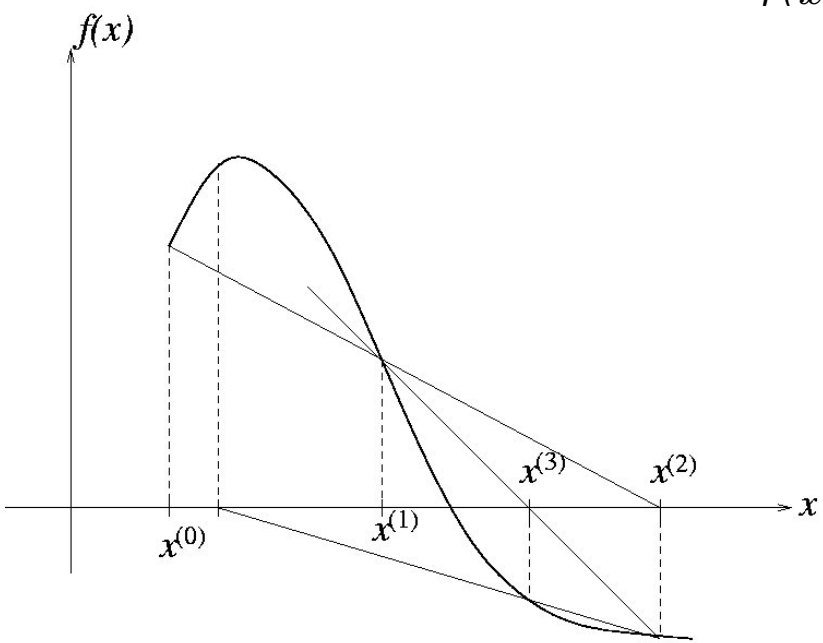
- **Bisection method**:
  - Start with $[c, d] \subseteq [a, b]$ where $f(c) \cdot f(d) \leq 0$ – the continuity of f guarantees the existence of (at least) one root in $[c, d]$. Bisection of the interval $[c, d]$ allows – depending on the sign of $f((c + d)/2)$ – to limit the search to one of the subintervals $[c, (c + d)/2]$ or $[(c + d)/2, d]$, and so on.
  - Stop when finding a root or when falling below a tolerance ε for the active interval length $d - c$.



- **Regula falsi** (**false position method**):
  - A variant of the bisection method where not the midpoint of the interval is chosen as one endpoint of the new and smaller interval but the zero-crossing of the line between the points $(c, f(c))$ and $(d, f(d))$ (possibly converges slower than the simple bisection method!).

- **Secant method**:
  - Start with *two* initial approximations $x^{(0)}$ and $x^{(1)}$; in the following, we will determine $x^{(i+1)}$ from $x^{(i-1)}$ and $x^{(i)}$ by trying to find the root of the straight line through $(x^{(i-1)}, f(x^{(i-1)}))$ and $(x^{(i)}, f(x^{(i)}))$ (the *secant* $s(x)$):

$$
\begin{aligned}
s(x) &:= f(x^{(i)}) + \frac{f(x^{(i)}) - f(x^{(i-1)})}{x^{(i)} - x^{(i-1)}} \cdot (x - x^{(i)}) \,, \\
0 = s(x^{(i+1)}) &= f(x^{(i)}) + \frac{f(x^{(i)}) - f(x^{(i-1)})}{x^{(i)} - x^{(i-1)}} \cdot (x^{(i+1)} - x^{(i)}) \,, \\
x^{(i+1)} &:= x^{(i)} - (x^{(i)} - x^{(i-1)}) \cdot \frac{f(x^{(i)})}{f(x^{(i)}) - f(x^{(i-1)})} \,.
\end{aligned}
$$



Newton's Method, Remarks about the Speed of Convergence

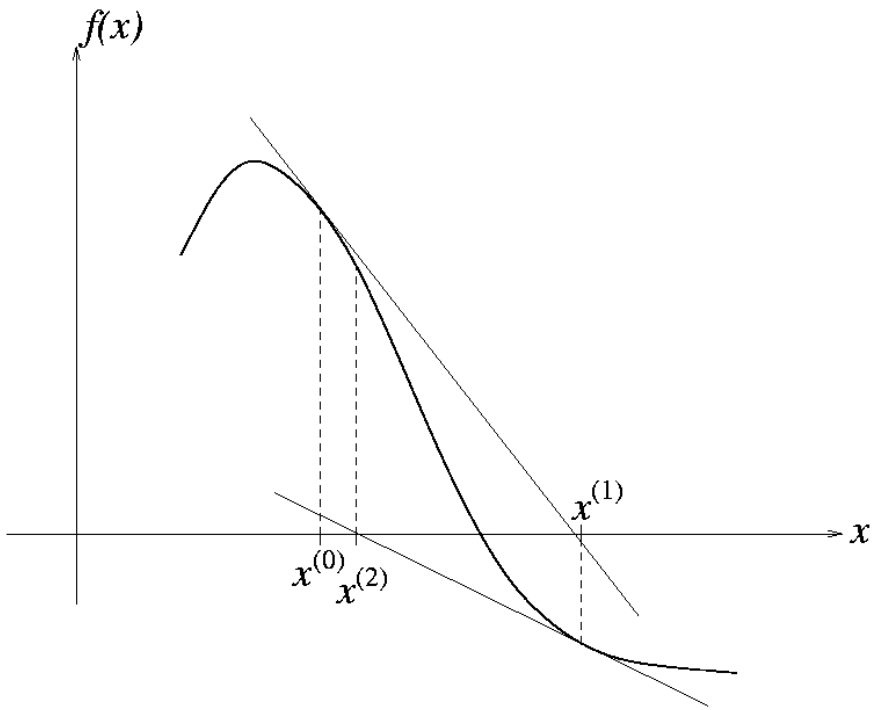
- **Newton's method**:
  - Here, we begin with *one* initial approximation $x^{(0)}$ and, in the following, determine $x^{(i+1)}$ from $x^{(i)}$ by trying to find the root of the *tangent* $t(x)$ to $f(x)$ at the point $x^{(i)}$ (**linearization**: replace f with its tangent or rather its Taylor polynomial of first degree, respectively):

$$\begin{aligned} t(x) &:= f(x^{(i)}) + f'(x^{(i)}) \cdot (x - x^{(i)}) \,, \\ 0 = t(x^{(i+1)}) &= f(x^{(i)}) + f'(x^{(i)}) \cdot (x^{(i+1)} - x^{(i)}) \,, \\ x^{(i+1)} &:= x^{(i)} - \frac{f(x^{(i)})}{f'(x^{(i)})} \,. \end{aligned}$$

  - Newton's method corresponds to the secant method in the limit case $x^{(i-1)} = x^{(i)}$.
- **Order of convergence** of the methods introduced:
  - globally linear for the bisection method and regula falsi,
  - locally quadratic for Newton,
  - locally 1.618 for the secant method.

  - However, as one Newton step requires an evaluation of both f and f', it has to be compared with *two* steps of the other methods. Furthermore, the calculation of the derivative is often problematic.
  - Hence, the secant method is looking very good!

- Figure for Newton's method:



- **Remark**: For finding the roots of polynomials of higher degree (often an extremely ill conditioned problem) there are special methods.

Systems of Non-linear Equations

- In most applications in practice, we have $n \gg 1$ (since unknowns are for example function values at grid points!), such that we have to deal with very big systems of non-linear equations.
- In the multidimensional case, the place of the simple derivative $f'(x)$ is taken by the so called **Jacobian** $F'(x)$, the matrix of the partial derivatives of all vector components of F with respect to all variables.
- The Newton iteration method therefore becomes

$$x^{(i+1)} := x^{(i)} - F'(x^{(i)})^{-1} F(x^{(i)}) ,$$

  where, of course, the matrix $F'(x^{(i)})$ is not inverted, but the corresponding system of linear equations with the right-hand side $-F(x^{(i)})$ is solved (directly):

  calculate $F'(x)$;
  factorize $F'(x) =: LU$;
  solve $LUs = -F(x)$;
  update $x := x + s$;
  evaluate $F(x)$;

- The repeated calculation of the Jacobian is very expensive. Often, it is only possible to compute it approximately, as most times numerical differentiation is necessary. Also, solving a system of linear equations directly in every Newton step is expensive. Therefore, Newton's method was only the starting point for a multitude of algorithmic developments.

Simplifications

- **Newton chord** or **Shamanskii method**:
  Here, the Jacobian is not calculated and inverted in every Newton step, but $F'(x^{(0)})$ is always used (chord), or an $F'(x^{(i)})$ is re-used for several Newton steps (Shamanskii).
- **inexact Newton method**:
  Here, the system of linear equations is not solved directly (i.e. exactly, for example by means of the LU factorization) in every Newton step, but iteratively; this is denoted as an **inner** iteration within the (**outer**) Newton iteration.
- **quasi-Newton method**:
  Here, a sequence $B^{(i)}$ of approximations of $F'(\bar{x})$ is created, using cheap **updates** instead of an expensive recalculation. We utilize the simplicity of inverting a **rank-1 update** $(B + uv^T)$ with two arbitrary vectors $u, v \in \mathbb{R}^n$ (invertible if and only if $1 + v^T B^{-1} u \neq 0$):

$$\left(B + uv^T\right)^{-1} = \left(I - \frac{(B^{-1}u)v^T}{1 + v^T B^{-1} u}\right) B^{-1} .$$

  Broyden provided an adequate choice for u and v (s as above in the algorithm):

$$B^{(i+1)} := B^{(i)} + \frac{F(x^{(i+1)})s^T}{s^T s} .$$

6.4. The Principle of Multigrid

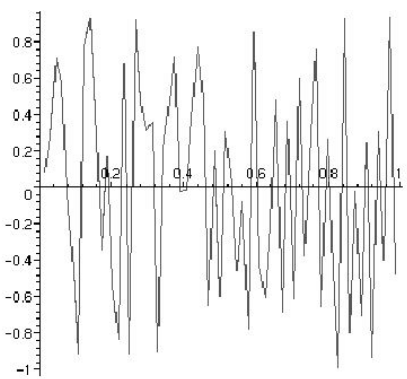
Basics

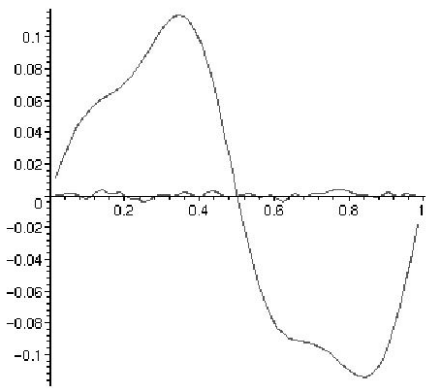
- In the previous sections, we have seen that the speed of convergence of both the relaxation methods and the CG method decreases with the resolution for systems of linear equations resulting from the discretization of partial differential equations (probably the most important case): The finer we discretize, the more iterative steps are necessary to reduce the error by a given factor.
- Here, the **principle of multigrid**, one of the most fruitful developments of younger numerical mathematics, can help. In literature, different names appear such as **multigrid**, **multilevel**, or **multiresolution** – the concrete methods differ, but the underlying principle is always the same.
- We want to use this principle to accelerate relaxation methods.
- The idea is simple but requires at least superficial knowledge of Fourier analysis. Imagine the error $e^{(i)}$ of a relaxation method decomposed in terms of Fourier transformation into components of different frequencies. Put roughly, this is a series expansion, where the elements of the series are formed not by monomials x^k, but by trigonometric terms $\sin(kx)$ and $\cos(kx)$; here, k indicates the frequency.

- Typically, our relaxation methods quickly reduce the error components that are high-frequency and still representable concerning the underlying grid, whereas they reduce the low-frequency parts only extremely slowly – they **smooth** the error (therefore the name).

Error Reduction at Relaxation

- We will examine this phenomenon on the basis of a very simple example of a one-dimensional Laplace equation (i.e. nothing else but $u''(x) = 0$), which ought to be solved with a finite difference approach (one of the most widespread methods of discretization for PDE) on an equidistant grid with mesh width $h = 2^{-6}$:
  - On the left-hand side, the randomly chosen initial error $e^{(0)}$ is plotted.
  - On the right-hand side, we can see the error of a damped Jacobi iteration after 100 iterative steps (a lot for such a lousy example) with a damping parameter $\alpha = 0.5$.
  - Obviously, the error is smoothed: The highly oscillating parts have almost vanished, the low-frequency parts are still there. The multigrid method that will be introduced in the following deals with all frequencies in only two (!) steps, as we can see in the picture on the right with the hardly visible curve oscillating around zero.





The Principle of Coarse Grid Correction

- **Consequence**:
  - Obviously a (too) fine grid has problems with eliminating the long wave error components. However, it does handle the high-frequency ones (of course only as long as they are presentable on the grid – memento Shannon!).
  - As terms such as ‘long wave’ are relative, it suggests itself to switch to a coarser grid to reduce the low-frequency parts. Furthermore, this is an economic decision, as coarse grids are linked with a decreased computational effort.
  - Recursively pursuing this **two-grid approach** (the components that are still too long-waved for the coarse grid are dealt with on an even coarser grid and so forth) leads us to **multigrid methods**.
- The initially apparent procedure to discretize a given PDE separately on grids of different fine resolution and, afterwards, to solve the systems of equations by means of relaxation, leads us nowhere: In every “solution”, only one part of the work was done. How can we bring the information together?
- **Correction schemes** that regard the computations of the coarser grid as corrections for the solution of the fine grid are more favourable.
- For reasons of simplicity, we will only observe the case of two regular grids.

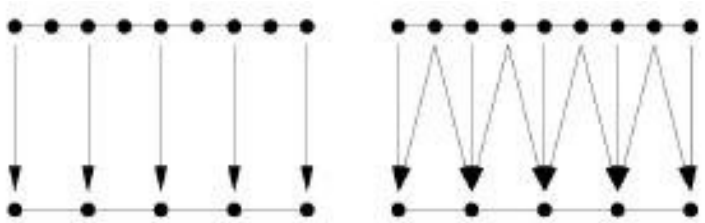
The Algorithm of Coarse Grid Correction

- Let be given
  - a fine grid Ω_f and a coarse grid Ω_c with the mesh widths h_f and h_c, respectively (coarsening usually means the transition to double the mesh width, i.e. $h_c = 2h_f$),
  - an approximate solution x_f on Ω_f,
  - matrices A_f, A_c and right-hand sides b_f, b_c, which describe the discretization of the PDE on the respective grids.
- With this we can formulate the following algorithm for **coarse grid correction**:

  smooth the active approximation x_f;
  compute the residual $r_f := b_f - A_f x_f$;
  restrict r_f on the coarse grid Ω_c;
  find a solution e_c for $A_c e_c = -r_c$;
  prolongate e_c on the fine grid Ω_f;
  subtract the resulting correction of x_f (maybe smooth again);

The Components of Coarse Grid Correction

- About the single steps:
  - **presmoothing**: smoothes the error (i.e. reduces the high-frequency error components); typically, a few steps of damped Jacobi or Gauss-Seidel;
  - **restriction**: simple adoption of the values at the grid points (**injection**) or an adequate averaging process over the values in the neighboring fine grid points (**full weighting**);

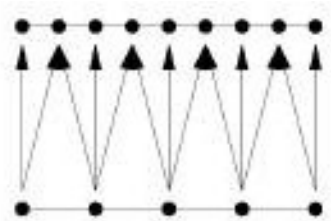


  - **coarse grid equation**: solved on the fine grid, the correction equation would bring us to the goal in a single step:

$$A_f \left(x_f - e_f \right) \;=\; Ax \;=\; b_f \,;$$

    on the coarse grid (i.e. $A_c e_c = -r_c$), only a correction results; the coarse grid equation can be solved directly (if Ω_c is already coarse enough) by means of relaxation or recursively by means of another coarse grid correction (then, we have the first real multigrid method);

– **prolongation**: the computed coarse grid correction e_c has to be interpolated back to Ω_f (direct adoption of the value in the coarse grid points and averaging in the fine grid points, e.g.);



– **post-smoothing**: sometimes, it is also advisable to carry out some relaxation steps *after* the excursion to the coarse grid.

The Multigrid Method

- If the coarse grid equation is solved recursively by means of another coarse grid correction, we get from the two-grid to the multigrid method with a hierarchy of grids Ω_l, $k = L, \ldots, 1$, here as **V-cycle**:

  smooth the active approximation x_l;

  compute the residual $r_l := b_l - A_l x_l$;

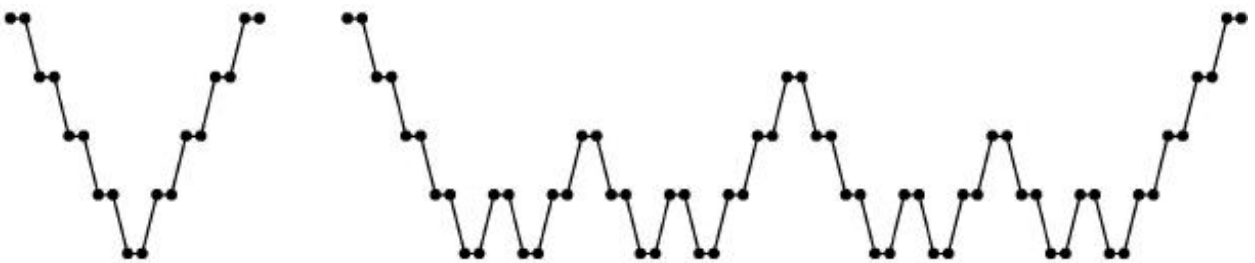
  restrict r_l to the next coarser grid Ω_{l-1};

  find a solution e_{l-1} for $A_{l-1} e_{l-1} = -r_{l-1}$ by means of the coarse grid correction;

  prolongate e_{l-1} to the next finer grid Ω_l;

  subtract the resulting correction from x_l (maybe smooth again);

- On the coarsest grid Ω_1, it is often possible to solve directly (few variables). Depending on the nesting of the recursion, different algorithmic alternatives result.



Analysis of the V-Cycle

- What do we obtain by applying multigrid algorithms instead of simple relaxations?
  - First, the total effort is dominated by the finest grid. When C denotes the number of arithmetic operations of a smoothing step in Ω_L, then $C/2$, $C/4$ or $C/8$ operations accumulate in Ω_{L-1} in the 1 D, 2 D, or 3 D case respectively. The same holds for restriction and prolongation. The total cost of a V-cycle, therefore, is of the same order of magnitude as the one of a single relaxation step on the finest occurring grid (geometric series!), i.e. kC with small k. The same holds for the memory effort. In this regard, one multigrid step does not cost more than a simple relaxation step.
  - Concerning the speed of convergence, in many cases, the spectral radius ρ of the multigrid iteration matrix does not depend on the number n of unknowns and, therefore, not on the resolution of the discretization. Reduction rates of 0.2 or smaller per iterative step are no rarity. The number of necessary V-cycles, therefore, always stays within the one-digit range.
- Those two attributes let multigrid methods be the ideal iterative methods. This is why the principle of multigrid today is used very commonly and in multiple variations and enhancements, respectively, of the basic method introduced here.

6.5. Applications of Iterative Methods in Computer Science

- **Partial differential equations (PDE)**:
  When discretizing them with finite differences or finite elements, large sparse systems of equations emerge, which, if necessary, have to be linearized and then to be solved iteratively. PDE occur for example in numerical simulation or in image processing.
- **Computer graphics**:
  Fractal (self-similar) curves and surfaces are produced via iterations. They are used for modeling natural objects such as plants, clouds, or mountain ranges in computer graphics.
- **Neural networks**:
  Neural networks serve to model complex systems and are based on the functionality of the human brain. In the simplest case, the network consists of n nodes N_i that each receive an input signal x_i, amplify it with a weight w_i, and forward it to a collective output node, which adds up all incoming signals to $y := \sum_{i=1}^{n} w_i x_i$. A neural network is therefore mainly defined by its weights w_i, which have to be chosen in a way that enables the network to find and denote the solution of a problem. This task is carried out by an iterative learning algorithm, which optimizes the network on the basis of test samples with given solutions y such that the network will hopefully deliver the desired answer.
- ...