Version 3.01 - 13. Oktober 2011

Hydroinformatik II

"Prozess-Simulation und Systemanalyse"

Prof. Dr.-Ing. Olaf Kolditz

TU Dresden / UFZ Leipzig
Angewandte Umweltsystemanalyse
Umweltinformatik

WS 2011/2012

Vorlesungskonzept

Es geht weiter ... Nachdem wir uns im erstem Semester Hydroinformatik mit dem objekt-orientierten Programmierung mit C++ "angefreundet" haben, soll es in diesem Semester auch um die Anwendung in den Disziplinen Hydrologie, Wasser- und Abfallwirtschaft gehen. Natürlich werden wir uns mit sehr einfachen Beispielen beschäftigen (und z.B. keine kompletten Kläranlagen rechnen, dafür sind die Kollegen von der Siedlungswaaserwirtschaft zuständig). Dennoch wollen wir ihnen am Ende des Semesters auch einen Einblick in unsere laufende Forschung geben, z.B. den Einsatz von numerischen Modellen bei der Abfalllagerung [11], in der Hydrologie [5] und in der Grundwassermodellierung [15].

Der Vorlesungsfahrplan sieht wir folgt aus:

- Grundlagen - Mechanik (2-3V)
- Grundlagen - Numerik (2-3V)
- Prozessverständnis - Benchmarks (3V)
- Anwendungen (2V)
- Visual C++ mit Qt (4V)

Abbildung 1: Lehrbuch zur "Numerischen Hydromechanik" [8]

Grundlagen

Für die ersten beiden Teile: Mechanik und Numerik, verwende ich weitgehend Material aus meinem Springer-Lehrbuch "Numerical Methods in Environmental Fluid Mechanics" [8] (für Bauingenieure, siehe Abb. 1). Als deutsprachige Literatur empfehle ich ihnen natürlich die Dresdner Klassiker [7] und [2] sowie [3] (Hydromechanik, generell) [12] (Gerinnehydraulik), [6] (Grundwasserhydraulik). Die TU Dresden hat eine lange Tradition in der Wasserforschung.

Prozessverständnis

Anhand von einfachen Beispielen (Benchmarks) werden wir uns mit der Simulation von Diffusionprozessen, Grundwasser- und Gerinnehydraulik beschäftigen (siehe Anlage A). Diese Testbeispiele stammen aus der OpenGeoSys-(OGS)-Benchmarksammlung, die für die OGS-Entwicklung benutzen (komplett unter www.opengeosys.net)

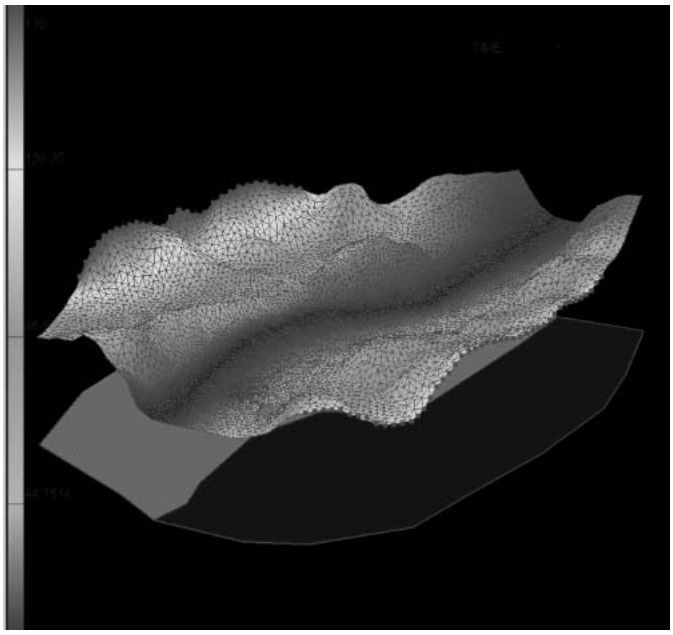

Abbildung 2: Übung zur Vorlesung "Prozesssimulation": Tübinger Grundwassermodell
(siehe http://www.uni-tuebingen.de//uni/epi/teaching/Case_studies_index.htm)

Anwendungen

Bei der Anwendung wollen wir ein (einfaches) Dresdner-Grundwassermodell entwickeln. Dabei haben wir mit echten Daten zu tun (z.B. digitalen Geländemodellen).

Visual C++ mit Qt

Natürlich geht's auch mit der C++ Programmierung (Qt) weiter, wir wollen soviel wie möglich selber implementieren.

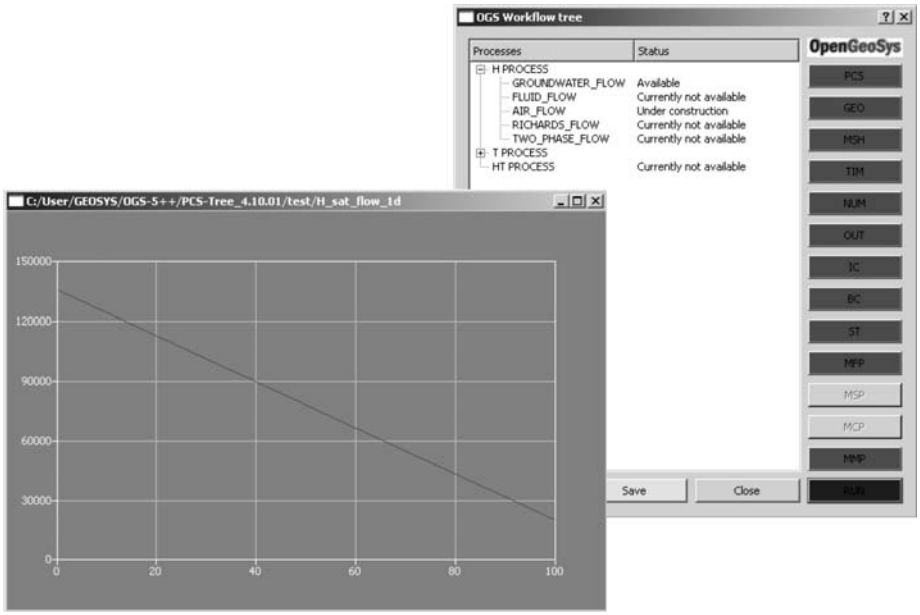

Abbildung 3: OGS (light) Studentenversion (daran werden wir gemeinsam arbeiten)

Organisatorisches

Die Prüfung 'Hydroinformatik I' (erstes Semester) ist eine Klausur. Die Prüfung 'Hydroinformatik II' (zweites Semester) besteht aus zwei Teilen, einer Klausur und einer Programmier-Hausarbeit. Bei den Klausuren geht es um die Beantwortung von Verständnisfragen (siehe Testfragen zu jeder Vorlesung).

Sprache: normalerweise in Deutsch. Da die Syntax von Computersprachen (wie C++) immer in Englisch ist, würde ich auch mal eine Vorlesung in Englisch anbieten.

Konsultationen: immer nach der Vorlesung (Freitag ab 15 Uhr).

Kontakt: jederzeit by e-mail (olaf.kolditz@ufz.de), in dringenden Fällen auch per Handy (0151 52739034).

Vorlesungsunterlagen: sind in bewährter Weise auf meinem UFZ Server zu finden (`http://www.ufz.de/index.php?de=11877`).

Part I

Mechanik

Kapitel 1

Balance Equations of Fluid Mechanics

1.1 General Conservation Law

1.1.1 Basic Equations of Fluid Dynamics

The basic idea of continuum mechanics is that the evolution of a physical system is completely determined by conservation laws, i.e. basic properties such as mass, momentum, and energy are conserved during the considered process at all times. Any physical system can be completely determined by this conservation properties. In contrast, other quantities such as pressure or entropy do not obey conservation laws. The only additional information concerns the consistence of the material (e.g. fluids, solids, porous medium) in form of constitutive laws.

The concept of conservation means that the variation of a conservation quantity within a given control volume is due to the net effect of internal *sources* and of the amount of the quantity which is crossing the boundary surface of the considered volume - *fluxes*. Sources and fluxes are, in general, dependent of space-time coordinates as well as on mechanical and thermodynamic factors. Fluxes result from two contributions: first due to advective transport by fluid motion and second due to diffusion/dispersion processes. Diffusion is always present even when the fluid is at rest. Diffusion is the tendency towards equilibrium or homogeneity of a physical system.

Kinematics of Continua

In continuum mechanics we distinguish between two methods concerning the derivation of balance equations. In the **Lagrangian formulation** we follow the quantity along a pathline, i.e. following particles (Fig. 1.1). In the **Eulerian**

formulation of motion we consider variations of the quantity with respect to a fixed control volume at fixed places (Fig. 1.2).

A **pathline** is a curve along which a fixed particle of a continuum moves during a sequence of time. Pathline is Lagrangian concept of motion. A **streamline** is a curve along which a sequence of particles moves at a given time. By definition, the tangent to a streamline coincides with the velocity vector at that point. Streamline is Eulerian concept of motion. Note, for unsteady flow the streamline may vary from one instant to the next, whereas for steady flow streamlines remain unchanged with time. For steady motion both pathlines and streamlines coincide. Any particle will remain on a given streamline as time proceeds. Additional terms associated with kinematics of continua are the following:

- particle
- material coordinates (Lagrangian coordinates) (Fig. 1.1)

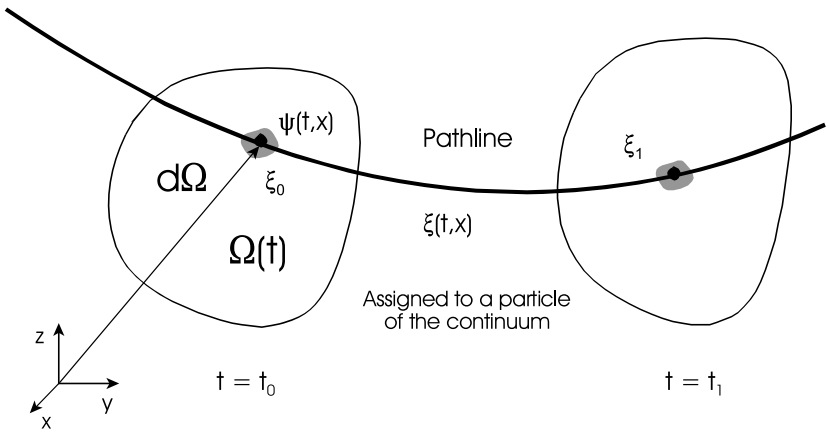

Abbildung 1.1: Lagrangian description of motion

- spatial coordinates (Eulerian coordinates) (Fig. 1.2)
- displacement $\mathbf{w}$ and volumetric dilatation (relative growth of volume with respect to the original one)

$$\varepsilon = \nabla \cdot \mathbf{w} = \varepsilon_{11} + \varepsilon_{22} + \varepsilon_{33}$$

- material derivative (see section 1.1.5)

$$\frac{d}{dt} = \frac{\partial}{\partial t} + \mathbf{v} \cdot \nabla \tag{1.1}$$

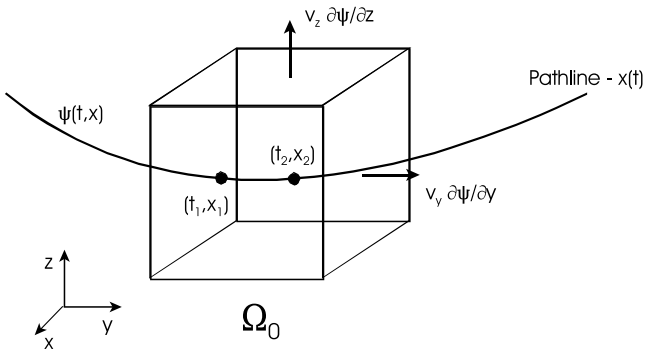

Abbildung 1.2: Eulerian description of motion

1.1.2 Conservation Quantities

The amount of a quantity in a defined volume Ω is given by

$$\Psi = \int_{\Omega} \psi d\Omega(t) \tag{1.2}$$

where Ψ is an extensive conservation quantity (i.e. mass, momentum, energy) and ψ is the corresponding intensive conservation quantity such as mass density ρ, momentum density $\rho\mathbf{v}$ or energy density e (see Table 1.1).

Tabelle 1.1: Conservation quantities

Extensive quantity	Symbol	Intensive quantity	Symbol
Mass	M	Mass density	ρ
Linear momentum	$\mathbf{m}$	Linear momentum density	$\rho\mathbf{v}$
Energy	E	Energy density	$e = \rho i + \frac{1}{2}\rho v^2$

1.1.3 Lagrangian Description of Motion

From the Lagrangian view point, the general conservation law postulates that the quantity remains conserved during the physical process within Ω.

$$\frac{d}{dt}\int_{\Omega} \psi d\Omega = \int_{\Omega} q^{\psi} d\Omega \tag{1.3}$$

where d/dt is the material derivative (eqn 1.1). From the definition of the derivative we have (Fig. 1.3)

$$\begin{aligned}
&\frac{d}{dt}\int_{\Omega}\psi d\Omega = \\
&= \lim_{\Delta t\to 0}\frac{1}{\Delta t}\left\{\int_{\Omega(t+\Delta t)}\psi(t+\Delta t)d\Omega - \int_{\Omega(t)}\psi(t)d\Omega\right\} \\
&= \lim_{\Delta t\to 0}\frac{1}{\Delta t}\left\{\int_{\Omega_2}[\psi(t+\Delta t)-\psi(t)]d\Omega + \int_{\Omega_3}\psi(t+\Delta t)d\Omega - \int_{\Omega_1}\psi(t)d\Omega\right\} \\
&= \int_{\Omega_2}\frac{\partial\psi}{\partial t}d\Omega + \lim_{\Delta t\to 0}\frac{1}{\Delta t}\left\{\int_{\Omega_3}\psi(t+\Delta t)d\Omega - \int_{\Omega_1}\psi(t)d\Omega\right\} \qquad (1.4)
\end{aligned}$$

According to Fig. 1.3, the area elements can be defined as (note the time direction)

$$\begin{aligned}
d\Omega_1 = (\mathbf{v}\cdot d\mathbf{S})(-\Delta t) \quad &, \quad d\Omega_3 = (\mathbf{v}\cdot d\mathbf{S})\Delta t \\
t+\Delta t \to t \quad &, \quad t \to t+\Delta t \qquad (1.5)
\end{aligned}$$

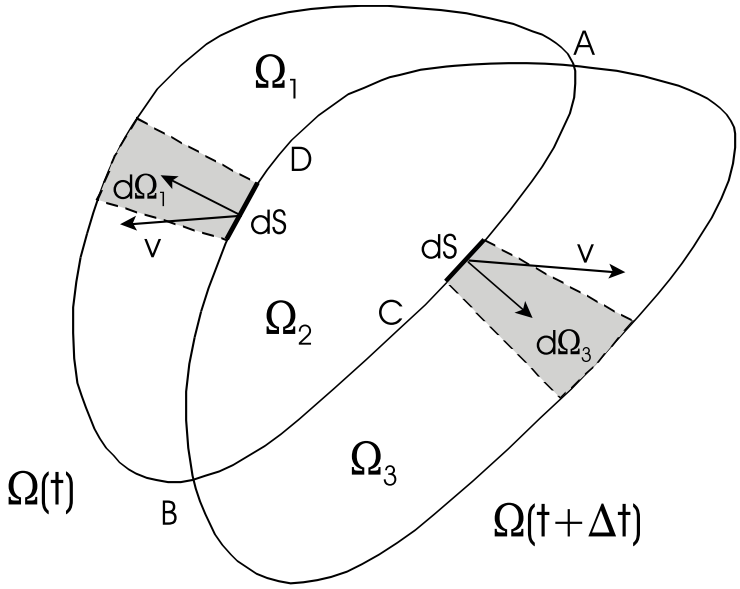

Abbildung 1.3: Lagrangian description of motion - proof

Therefore, we have

$$\begin{aligned}
&\lim_{\Delta t\to 0}\frac{1}{\Delta t}\left\{\int\limits_{\Omega_3}\psi(t+\Delta t)d\Omega-\int\limits_{\Omega_1}\psi(t)d\Omega\right\}=\\
=\;&\lim_{\Delta t\to 0}\frac{1}{\Delta t}\left\{\oint\limits_{BCA}\psi(t+\Delta t)(\mathbf{v}\cdot d\mathbf{S})\Delta t+\oint\limits_{ADB}\psi(t)(\mathbf{v}\cdot d\mathbf{S})\Delta t\right\}\\
=\;&\oint\limits_{\partial\Omega}\psi(t)\mathbf{v}\cdot d\mathbf{S} \qquad (1.6)
\end{aligned}$$

where $\partial\Omega$ is the domain boundary. Finally, we have following expression for the general conservation law

$$\frac{d}{dt}\int\limits_{\Omega}\psi d\Omega=\int\limits_{\Omega}\frac{\partial\psi}{\partial t}d\Omega+\oint\limits_{\partial\Omega}\psi(t)\mathbf{v}\cdot d\mathbf{S}=\int\limits_{\Omega}q^{\psi}d\Omega \qquad (1.7)$$

1.1.4 Eulerian Description of Motion

On the other hand, in the Eulerian sense, the variation of the quantity results only from internal sources and from fluxes through the surface of a fixed control volume $\partial\Omega$. Variation of the considered quantity per unit time is

$$\frac{\partial}{\partial t}\int\limits_{\Omega}\psi d\Omega \qquad (1.8)$$

Incoming flux through the fixed control volume surface is given by

$$-\oint\limits_{\partial\Omega}\mathbf{\Phi}^{\psi}\cdot d\mathbf{S} \qquad (1.9)$$

where $\mathbf{\Phi}^{\psi}$ is the total flux vector of quantity ψ. Internal sources of the considered quantity can be written as

$$\int\limits_{\Omega}q^{\psi}d\Omega \qquad (1.10)$$

where q^{ψ} is the internal sources density of ψ. Therefore, conservation of quantity ψ in the Eulerian description can be written in following form

$$\underbrace{\frac{\partial}{\partial t}\int\limits_{\Omega}\psi d\Omega}_{1}=\underbrace{-\oint\limits_{\partial\Omega}\mathbf{\Phi}^{\psi}\cdot d\mathbf{S}}_{2}+\underbrace{\int\limits_{\Omega}q^{\psi}d\Omega}_{3} \qquad (1.11)$$

with:

1. Rate of change of total amount of quantity ψ in the control volume,
2. Net rate of increase / decrease of ψ due to fluxes,
3. Rate of increase / decrease of ψ due to sources.

Note equations (1.7) and (1.11) are equivalent for $\mathbf{\Phi} = \mathbf{v}\psi$

The **Gauss-Ostrogradskian integral theorem** (Gaussian divergence theorem) can be used to transform surface into volume integrals for the flux term.

$$\oint_{\partial\Omega} \mathbf{\Phi}^{\psi} \cdot d\mathbf{S} = \int_{\Omega} \nabla \cdot \mathbf{\Phi}^{\psi} d\Omega \tag{1.12}$$

We can rewrite the general conservation law now in following form

$$\int_{\Omega} \frac{\partial \psi}{\partial t} d\Omega = -\int_{\Omega} \nabla \cdot \mathbf{\Phi}^{\psi} d\Omega + \int_{\Omega} q^{\psi} d\Omega \tag{1.13}$$

where additionally we used the fact that the control volume is fixed for the moment $\partial\Omega/\partial t = 0$

We consider a parallelepiped control volume in a Cartesian coordinate system (Fig. 1.4). The limes of net efflux through the control volume surface $\partial\Omega$ as $\Omega \to 0$ will be

$$\begin{aligned}
&\lim_{\Omega\to 0} \frac{1}{\Omega} \oint_{\partial\Omega} \mathbf{\Phi} \cdot d\mathbf{S} = \\
&= \lim_{\Delta x\Delta y\Delta z\to 0} \frac{1}{\Delta x\Delta y\Delta z} \Big\{ (\Phi_x \mid_{x+\frac{\Delta x}{2},y,z} \; - \; \Phi_x \mid_{x-\frac{\Delta x}{2},y,z}) \Delta y\Delta z + \\
&\qquad\qquad (\Phi_y \mid_{x,y+\frac{\Delta y}{2},z} \; - \; \Phi_y \mid_{x,y-\frac{\Delta y}{2},z}) \Delta x\Delta z + \\
&\qquad\qquad (\Phi_z \mid_{x,y,z+\frac{\Delta z}{2}} \; - \; \Phi_z \mid_{x,y,z-\frac{\Delta z}{2}}) \Delta x\Delta y \Big\} \\
&= \frac{\partial \mathbf{\Phi}}{\partial x} + \frac{\partial \mathbf{\Phi}}{\partial y} + \frac{\partial \mathbf{\Phi}}{\partial z} = \nabla \cdot \mathbf{\Phi}
\end{aligned} \tag{1.14}$$

The above expression provides a physical interpretation of the divergence at a point as the excess of efflux over influx of quantity ψ through a closed surface surrounding the point of interest, per unit volume.

$$\nabla \cdot \mathbf{\Phi} = \lim_{\Omega\to 0} \frac{1}{\Omega} \oint_{\partial\Omega} \mathbf{\Phi} \cdot d\mathbf{S} \tag{1.15}$$

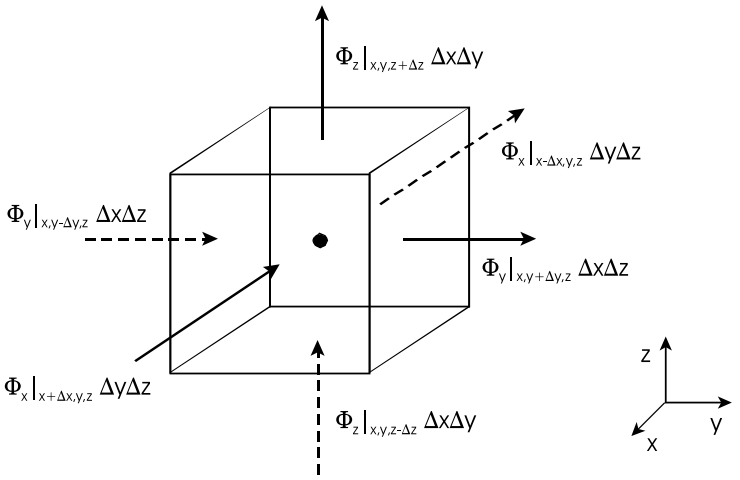

Abbildung 1.4: Eulerian description of motion - fluxes

1.1.5 Reynolds Transport Theorem

Of course, conservation does not depend on the specific method for derivation. Both the Lagrangian (1.3) and the Eulerian point of view (1.13) must result in identical expressions of conservation laws.

$$\frac{d}{dt}\int\limits_{\Omega}\psi d\Omega = \int\limits_{\Omega}\left(\frac{\partial\psi}{\partial t} + \nabla\cdot\mathbf{\Phi}^{\psi}\right)d\Omega = \int\limits_{\Omega} q^{\psi}d\Omega \tag{1.16}$$

The above equation is known as the **Reynolds transport theorem** which can be understood as the relationship between the Lagrangian and the Eulerian descriptions. To derive the corresponding differential equation of the above integral equation, we make use of the fact that the integral expression must be valid for all (in particular infinitesimal) volumes Ω.

1.1.6 Fluxes

The total flux $\mathbf{\Phi}^{\psi}$ of a quantity ψ is defined as

$$\mathbf{\Phi}^{\psi} = \mathbf{v}^{E}\psi \tag{1.17}$$

where $\mathbf{v}^{E}$ is a mean particle velocity. Physically $\mathbf{\Phi}^{\psi}$ represents the quantity of ψ passing through a unit area of the continuum, normal to $\mathbf{v}^{E}$, per unit time with respect to a fixed coordinate system.

For the case of a multi-component continuum let $\mathbf{v}$ denote the mass-weighted velocity describing a more ordered motion of the particles of a fluid element. The total flux can be written as

$$\mathbf{\Phi}^{\psi} = \mathbf{v}^{E}\psi = \underbrace{\mathbf{v}\psi}_{\mathbf{\Phi}_A^{\psi}} + \underbrace{(\mathbf{v}^{E} - \mathbf{v})\psi}_{\mathbf{\Phi}_D^{\psi}} \tag{1.18}$$

and, therefore, decomposed into two parts: an advective flux $\mathbf{\Phi}_A^{\psi}$ and a diffusive flux $\mathbf{\Phi}_D^{\psi}$ relative to the mass-weighted velocity:

- advective flux of quantity ψ

$$\mathbf{\Phi}_A^{\psi} = \mathbf{v}\psi \tag{1.19}$$

- diffusive flux of quantity ψ (Fick's law)

$$\mathbf{\Phi}_D^{\psi} = -\alpha\nabla\psi \tag{1.20}$$

This means, diffusive flux is positive in the direction of negative gradient.

If the conservation quantity is a vector (e.g. linear momentum) then the flux becomes a tensor and the source term a vector (e.g. body forces):

- advective flux of vector quantity $\boldsymbol{\psi}$

$$\mathbf{\Phi}_A^{\psi} = \mathbf{v} : \boldsymbol{\psi} = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix} \begin{bmatrix} \psi_x \\ \psi_y \\ \psi_z \end{bmatrix} = \begin{vmatrix} v_x\psi_x & v_x\psi_y & v_x\psi_z \\ v_y\psi_x & v_y\psi_y & v_y\psi_z \\ v_z\psi_x & v_z\psi_y & v_z\psi_z \end{vmatrix} \tag{1.21}$$

- diffusive flux of vector quantity $\boldsymbol{\psi}$

$$\mathbf{\Phi}_D^{\psi} = -\rho\nabla : \boldsymbol{\psi} = -\alpha \begin{vmatrix} \frac{\partial\psi_x}{\partial x} & \frac{\partial\psi_y}{\partial y} & \frac{\partial\psi_z}{\partial z} \\ \frac{\partial\psi_x}{\partial x} & \frac{\partial\psi_y}{\partial y} & \frac{\partial\psi_z}{\partial z} \\ \frac{\partial\psi_x}{\partial x} & \frac{\partial\psi_y}{\partial y} & \frac{\partial\psi_z}{\partial z} \end{vmatrix} \tag{1.22}$$

Using Fick's law the total flux can be written as

$$\mathbf{v} : \psi = \mathbf{v}\psi - \alpha\nabla\psi \tag{1.23}$$

When substituting this equation into the general balance equation, we yield the so-called transport equation

$$\underbrace{\int_{\Omega} \frac{\partial\psi}{\partial t}}_{1} = \underbrace{-\int_{\Omega} \nabla\cdot(\mathbf{v}\psi)}_{2} + \underbrace{\int_{\Omega} \nabla\cdot(\alpha\nabla\psi)}_{3} + \underbrace{\int_{\Omega} q^{\psi}}_{4} \tag{1.24}$$

with:

1. Rate of increase of ψ within a fluid element
2. Net rate of ψ due to flux out of the fluid element
3. Rate of increase / decrease of ψ due to diffusion
4. Rate of increase / decrease of ψ due to sources

1.1.7 General Balance Equation

Now a differential form of the general conservation law can be derived, because the conservation law in integral form must be valid for an arbitrary small volume. We obtain

$$\frac{\partial \psi}{\partial t} = -\nabla \cdot \mathbf{\Phi}^{\psi} + q^{\psi} = -\nabla \cdot (\mathbf{v}^{E} \psi) + q^{\psi} \tag{1.25}$$

which is denoted as the general balance equation for quantity ψ in differential form. Using partial differentiation we can introduce the convective form of general balance equation.

$$\begin{aligned} \frac{\partial \psi}{\partial t} &= -\mathbf{v}^{E} \nabla \psi - \psi \nabla \cdot \mathbf{v}^{E} + q^{\psi} \\ \frac{d\psi}{dt} &= \frac{\partial \psi}{\partial t} + \mathbf{v}^{E} \nabla \psi = -\psi \nabla \cdot \mathbf{v}^{E} + q^{\psi} \end{aligned} \tag{1.26}$$

While eqn (1.25) is the Eulerian form of the general balance equation the above equation is the corresponding Lagrangian form. The term $d\psi/dt$ [see eqn (1.1) for definition of material derivative] represents the rate of change of ψ as a particle moves along a pathline, while the right-hand-side represents the sources that cause this change.

Integral forms are more general formulations of conservation laws, since the remain valid also in the presence of discontinuous variation of the quantities (e.g. such as inviscid shock waves or contact discontinuities). Only if the thermodynamical properties remain continuous, integral and differential forms are fully equivalent.

1.2 Mass Conservation

The conservation law of mass states the empirical fact that in the absence of sources mass cannot disappear from a system nor be created.

$$\frac{dM}{dt} = \frac{d}{dt}\int\limits_{\Omega} \rho d\Omega = 0 \tag{1.27}$$

where M is mass and ρ is the mass density (specific mass). This means, no diffusive fluxes exist for mass transport in a homogeneous, single phase medium. Mass can only be transported through advection. Applying Reynolds transport theorem we have for $\psi = \rho$

$$\frac{d}{dt}\int\limits_{\Omega} \rho d\Omega = \int\limits_{\Omega} \left[\frac{\partial \rho}{\partial t} + \nabla \cdot (\mathbf{v}\rho)\right] d\Omega = 0 \tag{1.28}$$

The differential equation of mass conservation in divergence form becomes

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\mathbf{v}\rho) = 0 \tag{1.29}$$

Partial differentiation of the above equation gives

$$\frac{\partial \rho}{\partial t} + \mathbf{v} \cdot \nabla \rho + \rho \nabla \cdot \mathbf{v} = 0 \tag{1.30}$$

Using the material (or convective) derivative (1.1) the mass conservation equation can be rewritten as

$$\frac{d\rho}{dt} = -\rho \nabla \cdot \mathbf{v} \tag{1.31}$$

Note, above convective form of mass conservation equation becomes zero only for incompressible flows, i.e.

$$\frac{d\rho}{dt} = 0 \tag{1.32}$$

requires divergence-free flow.

$$\nabla \cdot \mathbf{v} = 0 \tag{1.33}$$

From eqn. (1.30) results that the above expression is the continuity equation for a homogeneous fluid ($\rho = \text{const}$).

1.3 Momentum Conservation

1.3.1 General Momentum Equation

As momentum is a vector quantity we need the general conservation law in vector form to state the balance equation for momentum conservation. Starting from the Eulerian general balance equation (1.11) for momentum density $\psi = \rho\mathbf{v}$ we have

$$\frac{\partial}{\partial t}\int\limits_{\Omega} \rho\mathbf{v}d\Omega + \oint\limits_{\partial\Omega} \mathbf{\Phi}^m \cdot d\mathbf{S} = \int\limits_{\Omega} \rho\mathbf{f}d\Omega \tag{1.34}$$

Flux term: The advective momentum flux is defined as

$$\mathbf{\Phi}^m = (\rho\mathbf{v}) \otimes \mathbf{v} = (\rho\mathbf{v})\mathbf{v} \tag{1.35}$$

Note that momentum flux is a tensor $\Phi^m_{ij} = \rho v_i v_j$.

Source term: From Newton's second law it is known, that variation of momentum result from forces acting on the physical system on consideration. We distinguish between external forces $\mathbf{F}^e$ and internal forces $\mathbf{F}^i$.

$$\mathbf{F} = \int\limits_{\Omega} \rho\mathbf{f}d\Omega = \int\limits_{\Omega} \rho(\mathbf{f}^e + \mathbf{f}^i)d\Omega = \underbrace{\int\limits_{\Omega} \rho\mathbf{f}^e d\Omega}_{\text{External forces}} + \underbrace{\oint\limits_{\partial\Omega} \boldsymbol{\sigma} : d\mathbf{S}}_{\text{Internal forces}} = \mathbf{F}^e + \mathbf{F}^i \tag{1.36}$$

The latter are dependent on the nature of the fluid. Internal forces (stresses) express the resistance of the fluid due to internal deformation. Internal forces within the volume Ω are those acting on points of the boundary surface. Since they have no opposite counterparts within the considered volume internal forces act as surface forces.

Substituting now flux and source terms of momentum we obtain

$$\frac{\partial}{\partial t}\int\limits_{\Omega} \rho\mathbf{v}d\Omega + \oint\limits_{\partial\Omega} \rho\mathbf{v}(\mathbf{v} \cdot d\mathbf{S}) = \int\limits_{\Omega} \rho\mathbf{f}^e d\Omega + \oint\limits_{\partial\Omega} \boldsymbol{\sigma} d\mathbf{S} \tag{1.37}$$

Applying the Gauss-Ostrogradskian theorem to the surface integrals

$$\begin{aligned} \oint\limits_{\partial\Omega} \rho\mathbf{v}(\mathbf{v} \cdot d\mathbf{S}) &= \int\limits_{\Omega} \nabla \cdot (\rho\mathbf{v}\mathbf{v})d\Omega \\ \oint\limits_{\partial\Omega} \boldsymbol{\sigma} d\mathbf{S} &= \int\limits_{\Omega} \nabla \cdot \boldsymbol{\sigma} d\Omega \end{aligned} \tag{1.38}$$

the momentum balance equation becomes

$$\frac{\partial}{\partial t}\int\limits_{\Omega}\rho\mathbf{v}d\Omega+\int\limits_{\Omega}\nabla\cdot(\rho\mathbf{v}\mathbf{v})d\Omega=\int\limits_{\Omega}\rho\mathbf{f}^{e}d\Omega+\int\limits_{\Omega}\nabla\cdot\boldsymbol{\sigma}d\Omega \tag{1.39}$$

The differential form of the momentum conservation law is then

$$\frac{\partial}{\partial t}(\rho\mathbf{v})+\nabla\cdot(\rho\mathbf{v}\mathbf{v})=\rho\mathbf{f}^{e}+\nabla\cdot\boldsymbol{\sigma} \tag{1.40}$$

The above equation is now extended by partial integration

$$\begin{aligned}\rho\frac{\partial\mathbf{v}}{\partial t}+\mathbf{v}\frac{\partial\rho}{\partial t}+(\rho\mathbf{v})\cdot\nabla\mathbf{v}+\mathbf{v}\nabla\cdot(\rho\mathbf{v}) &= \rho[\frac{\partial\mathbf{v}}{\partial t}+\mathbf{v}\cdot\nabla\mathbf{v}]+\mathbf{v}[\frac{\partial\rho}{\partial t}+\nabla\cdot(\rho\mathbf{v})]\\ &= \rho\mathbf{f}^{e}+\nabla\cdot\boldsymbol{\sigma}\end{aligned} \tag{1.41}$$

Using the mass conservation equation (1.29) and dividing by ρ we obtain

$$\frac{\partial\mathbf{v}}{\partial t}+(\mathbf{v}\cdot\nabla)\mathbf{v}=\mathbf{f}^{e}+\frac{1}{\rho}\nabla\cdot\boldsymbol{\sigma} \tag{1.42}$$

In index notation the above vector equation is written as

$$\begin{aligned}\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+w\frac{\partial u}{\partial z} &= \frac{1}{\rho}(\frac{\partial\sigma_{xx}}{\partial x}+\frac{\partial\sigma_{xy}}{\partial y}+\frac{\partial\sigma_{xz}}{\partial z})\\ \frac{\partial v}{\partial t}+v\frac{\partial u}{\partial x}+v\frac{\partial v}{\partial y}+w\frac{\partial v}{\partial z} &= \frac{1}{\rho}(\frac{\partial\sigma_{yx}}{\partial x}+\frac{\partial\sigma_{yy}}{\partial y}+\frac{\partial\sigma_{yz}}{\partial z})\\ \frac{\partial w}{\partial t}+u\frac{\partial w}{\partial x}+v\frac{\partial w}{\partial y}+w\frac{\partial w}{\partial z} &= g+\frac{1}{\rho}(\frac{\partial\sigma_{zx}}{\partial x}+\frac{\partial\sigma_{zy}}{\partial y}+\frac{\partial\sigma_{zz}}{\partial z})\end{aligned} \tag{1.43}$$

with $u=v_x, v=v_y, w=v_z$ and $\mathbf{f}^{e}=\mathbf{g}$.

1.3.2 Stress Tensor - σ

Surface forces on a fluid element result from pressure and viscous forces. The stress tensor can be, therefore, defined as

$$\sigma=-p\mathbf{I}+\tau \tag{1.44}$$

Pressure is the normal stress on the fluid element denoted by p. Fig. 1.5 shows the nine viscous stress components acting on a fluid element in motion. Suffix notation is applied to indicate the direction of the stress components. Suffices i and j in τ_{ij} mean, that stress acts in j-direction on a surface normal to the i-direction.

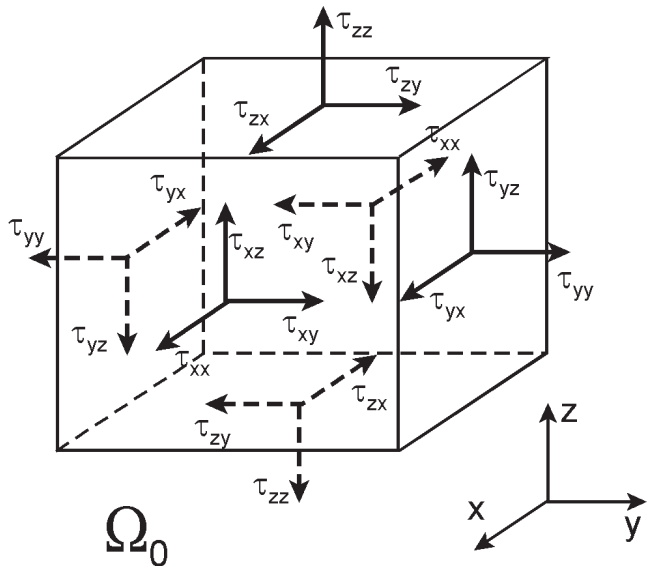

Abbildung 1.5: Components of the viscous stress tensor τ_{ij}

1.3.3 Euler Equations

For an ideal fluid without internal shear stresses (inviscid fluid),

$$\sigma = -p\mathbf{I} \tag{1.45}$$

the momentum conservation law (1.42) reduces to the **Euler equations**

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = \mathbf{f}^e - \frac{1}{\rho}\nabla p \tag{1.46}$$

1.3.4 Navier-Stokes Equations

For a Newtonian viscid fluid with stress tensor

$$\sigma = -p\mathbf{I} + \tau \tag{1.47}$$

the momentum conservation law (1.42) is

$$\boxed{\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = \mathbf{f}^e - \frac{1}{\rho}\nabla p + \nu\Delta\mathbf{v}} \tag{1.48}$$

which are called the **Navier-Stokes equations**. In general fluid mass continuity equation (1.29) has to be added to obtain a closed set of equations for determination of all unknown functions $\rho, \mathbf{v}$.

Dimensional analysis of the Navier-Stokes equation yields the non-dimensionalized form

$$\frac{v^* t^*}{L^*}\frac{\partial \mathbf{v}_D}{\partial t_D} + (\mathbf{v}_D \cdot \nabla)\mathbf{v}_D = -\nabla p_D + \frac{1}{Re}\Delta \mathbf{v}_D \tag{1.49}$$

with $Re = v^* L^*/\nu^*$ the Reynolds number. Asterix * indicates characteristic values of the quantity and D denotes dimensionless quantity, i.e. $\mathbf{v}_D = \mathbf{v}/v^*$.

1.3.5 Stokes Equations

For strongly viscous dominated flows (i.e. small Reynolds numbers) convective acceleration can be neglected with respect to the viscous term. As the result we obtain the Stokes equation.

$$\frac{\partial \mathbf{v}}{\partial t} = \mathbf{f}^e - \frac{1}{\rho}\nabla p + \nu \Delta \mathbf{v} \tag{1.50}$$

In non-dimensionalized form we have

$$\frac{{v^*}^2 t^*}{\nu^*}\frac{\partial \mathbf{v}_D}{\partial t_D} = \mathbf{f}^e_D - Re \nabla p_D + \Delta \mathbf{v}_D \tag{1.51}$$

1.3.6 Darcy Equations

For slow motion (e.g. in porous media) frequently we can neglect inertial forces and turbulence effects. Then we obtain the Darcy equations.

$$0 = \mathbf{f}^e - \frac{1}{\rho}\nabla p + \nu \Delta \mathbf{v} \tag{1.52}$$

The friction term in porous media is discussed in chapter 4.4.

1.4 Problems

General Conservation Law

1 What are the two fundamental concepts for description of motion in mechanics ? Explain the conceptual difference between them ? [sec. 1.1.1]

2 What are the fundamental conservation quantities in mechanics ? [sec. 1.1.2]

3 Derive the general conservation law (1.7). [sec. 1.1.3]

4 Explain mathematical and physical meaning of the Gauss-Ostrogradskian integral theorem (1.7). Give a physical interpretation of the divergence of fluxes $\nabla \cdot \mathbf{\Phi}$. Hint: Use eqn.(1.15). [sec. 1.1.4]

5 What is the meaning of Reynolds transport theorem concerning the Lagrangian and Eulerian concepts for motion. [sec. 1.1.5]

6 What are the two fundamental flux types ? Explain the physical reason for diffusion processes using the term mass-weighted velocity of quantity. [sec. 1.1.6]

7 Derive the general conservation law in differential form from its integral expression (1.24). [sec. 1.1.7]

Mass Conservation

8 Derive the integral balance equation of mass from the general conservation law (1.11). [sec. 1.2]

9 Derive the Lagrangian form of the mass conservation equation (1.29). Hint: Use chain rule and material derivative $d/dt = \partial/\partial t + \mathbf{v} \cdot \nabla$. What assumption is required for mass conservation $d\rho/dt = 0$. [sec. 1.2]

Momentum Conservation

10 Derive the integral balance equation of linear momentum from the general conservation law (1.11). [sec. 1.3.1]

11 Explain the physical background of the constitutive equation for the stress tensor of fluids (1.44). [sec. 1.3.2]

12 What are the four basic movements of fluid elements ? [sec. 1.3.3]

13 Explain the physical difference between the Euler (1.46) and Navier-Stokes (1.48) equations. [sec. 1.3.4-6]

14 What assumptions has to be applied to the Navier-Stokes equations (1.48) to obtain the Stokes equation (1.50) ? [sec. 1.3.7]

15 What assumptions has to be applied to the Navier-Stokes equations (1.48) to obtain the Darcy equation (1.52) ? [sec. 1.3.8]

16 Give some examples of flows which are described by Euler equations (1.46), Navier-Stokes equations (1.48), Stokes equation (1.50), and Darcy equation (1.52). [sec. 1.3.5-8]

1.5 Problem Classification

The governing equations for fluid flow and related transport processes are partial differential equations (PDE) containing first and second order derivatives in spatial coordinates and first order derivatives in time. Whereas time derivatives appear linearly, spatial derivatives often have a non-linear form. Frequently, systems of partial differential equations occur rather than a single equation.

Tabelle 1.2: Mathematical classification of PDE's

PDE type	Discriminant	Eigenvalues	Canonical form	Example
Elliptic 1.5.3	$B^2 - 4AC < 0$ complex characteristics	$\forall\lambda > 0$ equal signs	$\frac{\partial^2\psi}{\partial\xi^2} + \frac{\partial^2\psi}{\partial\eta^2} = 0$	Laplace equation
Parabolic 1.5.4	$B^2 - 4AC = 0$	$\exists\lambda = 0$	$\frac{\partial^2\psi}{\partial\eta^2} = G$	Diffusion, Burgers equations
Hyperbolic	$B^2 - 4AC > 0$ real characteristics	$\exists\lambda < 0$ different signs	$\frac{\partial^2\psi}{\partial\xi^2} - \frac{\partial^2\psi}{\partial\eta^2} = 0$	Wave equation

1.5.1 Mathematical Classification

The mathematical classification of PDEs should be tempered by a knowledge of the physical processes which are governed by the specific type of the PDE. Several procedures are available for classifying partial differential equations, such as algebraic methods (discriminant and eigenvalue evaluation), characteristics as well as Fourier analysis (Tab. 1.2).

A common formulation of a PDE in $\mathcal{R}^3$ is

$$L(\psi) = F(t, x_i, \psi, \frac{\partial\psi}{\partial x_i}, ..., \frac{\partial^n\psi}{\partial x_i^n}) = 0 \qquad , \qquad i = 3 \tag{1.53}$$

where L is s differential operator. Second-order PDE with two independent variables are given by

$$A\frac{\partial^2\psi}{\partial x^2} + B\frac{\partial^2\psi}{\partial x\partial y} + C\frac{\partial^2\psi}{\partial y^2} + D\frac{\partial\psi}{\partial x} + E\frac{\partial\psi}{\partial y} + F\psi + G = 0 \tag{1.54}$$

It is apparent that the classification depends only on the highest-order derivatives in each independent variable. Second-order PDEs with more independent variables can be classified by examination of the eigenvalues of the matrix a_{ij}.

$$\sum_i \sum_j a_{ij} \frac{\partial \psi^2}{\partial x_i \partial x_j} + G = 0 \tag{1.55}$$

1.5.2 Physical Classification

Now we examine different types of PDEs from a physical point of view and indicate the flow categories for which they occur (Tab. 1.3).

Elliptic PDEs are associated with steady-state problems (equilibrium problems). For example, potential flow is described by an elliptic equation. Parabolic PDEs are usually related to propagation problems with dissipation, e.g. from viscous or heat diffusion effects. Examples are many of the reduced forms of the Navier-Stokes equation, transport equation and Burgers equation. Hyperbolic PDEs are usually connected with propagation problems without dissipation (see section **??**), eg. if a wave-like motion remains unattenuated.

Tabelle 1.3: Physical classification of PDEs

Physical problem	Math. problem	Examples
Equilibrium problems	Elliptic equations	Irrotational incompressible flow Inviscid incompressible flow Steady state heat conduction
Propagation problems (infinite propagation speed)	Parabolic equations	Unsteady viscous flow Transient heat transfer
Propagation problems (finite propagation speed)	Hyperbolic equations	Wave propagation (vibration) Inviscid supersonic flow

Example: Subsonic/supersonic flow

Change of PDE type in different areas of the computational domain may be illustrated by considering 2-D subsonic/supersonic flow. The governing equation for steady, compressible, potential flow can be written as

$$(1 - M)\frac{\partial \psi^2}{\partial x^2} + \frac{\partial \psi^2}{\partial y^2} = 0 \tag{1.56}$$

where ψ is the potential function and M is the Mach number. Therefore, the equation may change its type in different parts of the flow domain depending on the local Mach number. Also for non-linear equations the PDE type can change locally.

Example: Forms of the Navier-Stokes equation

As shown in section 1.3.4 momentum conservation for incompressible, viscid fluid is given by

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = -\frac{1}{\rho}\nabla p + \nu \Delta \mathbf{v} \tag{1.57}$$

where external forces are neglected.

If using ($p^* = \rho^*(v^*)^2$) the non-dimensional form of the Navier-Stokes equation is then

$$\frac{v^* t^*}{L^*}\frac{\partial \mathbf{v}_D}{\partial t_D} + (\mathbf{v}_D \cdot \nabla)\mathbf{v}_D = -\nabla p_D + \frac{1}{Re}\Delta \mathbf{v}_D \tag{1.58}$$

In general, the Navier-Stokes equation has a parabolic-hyperbolic structure for unsteady flow problems and it has an elliptic-hyperbolic structure for steady flow problems. We see the equation type depends strongly on the Reynolds number.

(1) Strongly viscous dominated flow: $Re \ll 1$ - **Stokes equation**

In this case the advection term can be neglected with respect to the viscous one.

$$\frac{(v^*)^2 t^*}{\nu^*}\frac{\partial \mathbf{v}_D}{\partial t_D} = -Re \nabla p_D + \Delta \mathbf{v}_D \tag{1.59}$$

The above Stokes equation is elliptic in the steady-state case and parabolic in the transient case. Dissipation is isotropic, it works in all directions.

(2) Inviscid flow - **Euler equation**

In the case of large Reynolds numbers, viscous terms are small.

$$\frac{v^* t^*}{L^*}\frac{\partial \mathbf{v}_D}{\partial t_D} + (\mathbf{v}_D \cdot \nabla)\mathbf{v}_D = -\nabla p_D \tag{1.60}$$

The above Euler equation is hyperbolic in space and time. It describes a propagation problem with negligible dissipation effects. There is an essential direction of propagation.

1.5.3 Elliptic Equations

Elliptic PDEs are related to equilibrium and steady-state problems, e.g. steady state temperature distribution in a rod of solid material, equilibrium stress of a solid under a given load or irrotational flow of an incompressible fluid as well as potential flow.

Example: Steady state heat conduction (1-D)

A very simple example of an equilibrium problem is steady state heat conduction ($\psi = T$) in an insulated rod whose ends are kept at different constant temperatures. This problem is governed by the following equation

$$\frac{d^2\psi}{dx^2} = 0 \tag{1.61}$$

The solution will be a linear temperature distribution (Fig. 1.6).

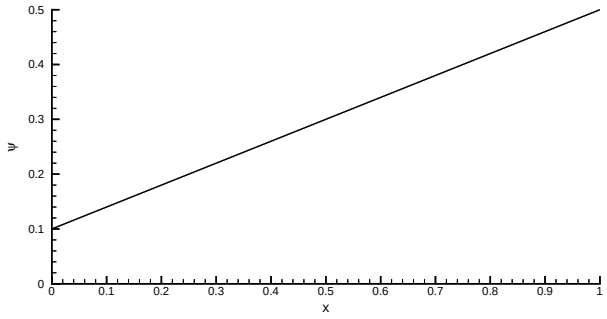

Abbildung 1.6: Solution of an elliptic equation

Example: Laplace equation (2-D potential flow)

The prototype of an elliptic equation is the Laplace equation.

$$\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} = 0 \tag{1.62}$$

By substitution it can be easily verified that the exact solution of the Laplace equation is

$$\psi = sin(\pi x)exp(-\pi y) \tag{1.63}$$

A unique solution of the Laplace equation and in general to all elliptic problems can be obtained by specifying values of the dependent variable ψ on all boundaries of the solution domain.

An important feature of elliptic problems is, that local disturbances in the interior changes the values of the dependent variable everywhere else in the solution domain, i.e. the signal propagates in all directions. Consequently, solutions of elliptic problems are always smooth even for discontinuous boundary conditions.

1.5.4 Parabolic Equations

Parabolic PDEs are usually related to (time-dependent) propagation problems with dissipation processes (e.g. from viscous shear or heat diffusion effects). A prototype of a parabolic equation is the heat conduction equation.

Example: Heat conduction in a rod (1-D problem)

$$\frac{\partial \psi}{\partial t} = \alpha \frac{\partial^2 \psi}{\partial x^2} \tag{1.64}$$

It can be easily shown, that the general solution of the linear diffusion equation can be expressed as

$$\psi(t, x) = sin(\pi x) exp(-\pi t^2) \tag{1.65}$$

If the rod cools down after an initially uniform temperature (at $t = 0$), where the ends of the rod are kept at equal constant temperature, the transient temperature distribution is given in Fig. 1.7. It can be seen, that the solution march forward in time but diffuse in space. At the beginning, temperature distributions have parabolic forms. The final, steady state consists of a uniform distribution ($\psi = 0$).

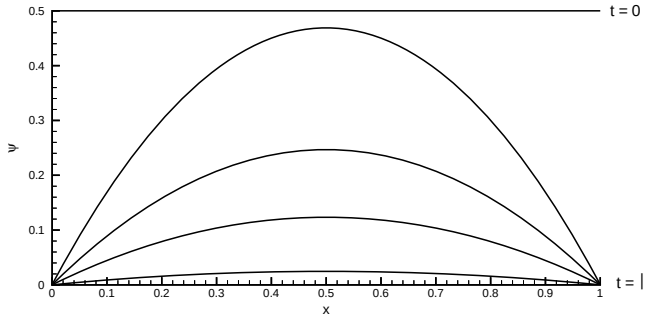

Abbildung 1.7: Solution of a parabolic equation

For the above parabolic problem appropriate initial as well as boundary conditions are needed. This type of problem is, therefore, termed an initial-boundary-value problem. The parabolic equation converts into an elliptic one for steady state conditions.

Note, the linear diffusion equation has a single characteristic direction dt/dx normal to the diffusion direction.

1.5.5 Equation Types

The following table gives typical examples of balance equations for the denoted quantities and their PDE types.

Physical meaning	Equation structure	Examples
Continuity	$\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} = 0$	Laplace equation
Mass/energy	$\frac{\partial\psi}{\partial t} + u\frac{\partial\psi}{\partial x} - \alpha\frac{\partial^2\psi}{\partial x^2} = 0$	Fokker-Planck equation
Momentum	$\frac{\partial\psi}{\partial t} + \psi\frac{\partial\psi}{\partial x} - \frac{\partial}{\partial x}[\alpha(\psi)\frac{\partial\psi}{\partial x}] = 0$	Navier-Stokes equation

1.5.6 Boundary Conditions

The solution of an initial-boundary-value-problem (e.g. partial differential equations for flow and transport problems) requires the specification of initial and boundary conditions. In this section we describe the most common boundary conditions and their physical meaning. Methods for implementation of boundary conditions in discrete equations are discussed in the part of numerical methods. Examples of common flow boundary conditions are e.g. inlet, outlet, wall, prescribed pressure values.

Constant pressure boundary condition: The constant pressure boundary condition is used in situations where exact details of the flow distribution are unknown but the boundary values of pressure are known.

No-slip boundary condition at a wall: This boundary condition is most common encountered in confined flow problems. The no-slip condition is the appropriate one for the velocity components at solid walls. The normal velocity component can simply be set to zero at this boundary.

The following table gives an overview on common boundary condition types and its mathematical representation.

To describe conditions at boundaries we can use flux expressions of conservation quantities.

Tabelle 1.4: Boundary conditions types

Type of BC	Mathematical Meaning	Physical Meaning
Dirichlet type	ψ	prescribed value potential surface
Neumann type	$\nabla\psi$	prescribed flux stream surface
Cauchy type	$\psi + A\nabla\psi$	resistance between potential and stream surface

Tabelle 1.5: Fluxes through surface boundaries

Quantity	Flux term
Mass	$\rho\mathbf{v}$
Momentum	$\rho\mathbf{v}\mathbf{v} - \sigma$
Energy	$\rho e\mathbf{v} - \lambda\nabla\mathbf{T}$

1.6 Problems

PDE Classification

1 What are the basic types of PDEs ?

2 What mathematical methods exist for classification of PDEs ?

3 Prove that the following system of PDEs (Riemann equations) is elliptic.

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \quad , \quad \frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} = 0 \tag{1.66}$$

5 Give examples for PDEs and corresponding physical problems.

Elliptic PDEs

6 Prove that eqn (1.63) is a special solution of the Laplace equation (1.62).

7 Prove that the expressions $u = x/(x^2 + y^2)$, $v = y/(x^2 + y^2)$ are solutions of eqn (1.66).

Boundary Conditions

11 What are boundary conditions needed for with respect to solving PDEs ?

12 What are the basic types of boundary conditions ? Explain their physical meaning.

Part II
Numerik

Kapitel 2

Numerical Methods

There are many alternative methods to solve initial-boundary-value problems arising from flow and transport processes in subsurface systems. In general these methods can be classified into analytical and numerical ones. Analytical solutions can be obtained for a number of problems involving linear or quasi-linear equations and calculation domains of simple geometry. For non-linear equations or problems with complex geometry or boundary conditions, exact solutions usually do not exist, and approximate solutions must be obtained. For such problems the use of numerical methods is advantageous. In this chapter we use the Finite Difference Method to approximate time derivatives. The Finite Element Method as well as the Finite Volume Method are employed for spatial discretization of the region. The Galerkin weighted residual approach is used to provide a weak formulation of the PDEs. This methodology is more general in application than variational methods. The Galerkin approach works also for problems which cannot be casted in variational form.

Fig. 2.1 shows an overview on approximation methods to solve partial differential equations together with the associated boundary and initial conditions. There are many alternative methods for solving boundary and initial value problems. In general, these method can be classified as discrete (numerical) and analytical ones.

2.1 Solution Procedure

For a specified mechanical problem the governing equations as well as initial and boundary conditions will be known. Numerical methods are used to obtain an approximate solution of the governing equations with the corresponding initial and boundary conditions. The procedure of obtaining the approximate solution consists of two steps that are shown schematically in Fig. 2.2. The first

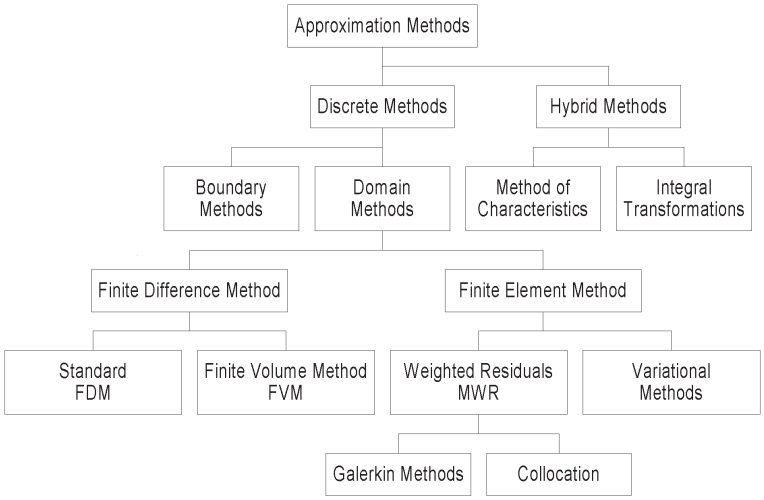

Abbildung 2.1: Overview of approximation methods and related sections for discussion

step converts the continuous partial differential equations and auxiliary conditions (IC and BC) into a discrete system of algebraic equations. This first step is called discretization. The process of discretization is easily identified if the finite difference method is used but it is slightly less obvious with more complicated methods as the finite element method (FEM), the finite volume method (FVM), and combined Lagrangian-Eulerian methods (method of characteristics, operator split methods).

The replacement of partial differential equations (PDE) by algebraic expressions introduces a defined truncation error. Of course it is of great interest to chose algebraic expressions in a way that only small errors occur to obtain accuracy. Equally important as the error in representing the differentiated terms in the governing equation is the error in the solution. Those errors can be examined as shown in section (2.2).

The second step of the solution procedure, shown in Fig. 2.2, requires the solution of the resulting algebraic equations. This process can also introduce an error but this is usually small compared with those involved in the above mentioned discretization step, unless the solver scheme is unstable. Appropriate methods to solve systems of algebraic equations are discussed in Habbar (1995). The approximate solution of the PDE is exact solution of the corresponding system of algebraic equations (SAE).

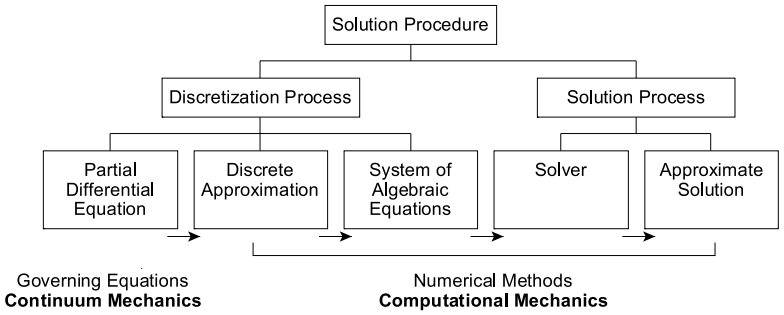

Abbildung 2.2: Steps of the overall solution procedure

2.2 Theory of Discrete Approximation

2.2.1 Terminology

In the first part of this script we developed the governing equations for fluid flow, heat and mass transfer from basic conservation principles. We have seen that hydromechanical field problems (as well as mechanical equilibrium problems) have to be described by partial differential equations (PDEs). The process of translating the PDEs to systems of algebraic equations is called - discretization (Fig. 2.3). This discretization process is necessary to convert PDEs into an equivalent system of algebraic equations that can be solved using computational methods.

$$L(u) = \tilde{L}(\tilde{u}) = 0 \tag{2.1}$$

In the following, we have to deal with discrete equations $\tilde{L}$ and with discrete solutions $\tilde{u}$.

An important question concerning the overall solution procedure for discrete methods is what guarantee can be given that the approximate solution will be close to the exact one of the PDE. From truncation error analysis, it is expected that more accurate solutions could be obtained on refined grids. The approximate solution should converge to the exact one as time step sizes and grid spacing shrink to zero. However, convergence is very difficult to obtain directly, so that usually two steps are required to achieve convergence:

Consistency	+	Stability	=	Convergence

This formula is known as the **Lax equivalence axiom**. That means, the system of algebraic equations resulting from the discretization process should be consistent with the corresponding PDE. Consistency guarantees that the PDE

is represented by the algebraic equations. Additionally, the solution process, i.e. solving the system of algebraic equations, must be stable.

Fig. 2.3 presents a graphic to illustrate the relationship between the above introduced basic terms of discrete approximation theory: convergence, stability, truncation, and consistency. These fundamental terms of discrete mathematics are explained further and illustrated by examples in the following.

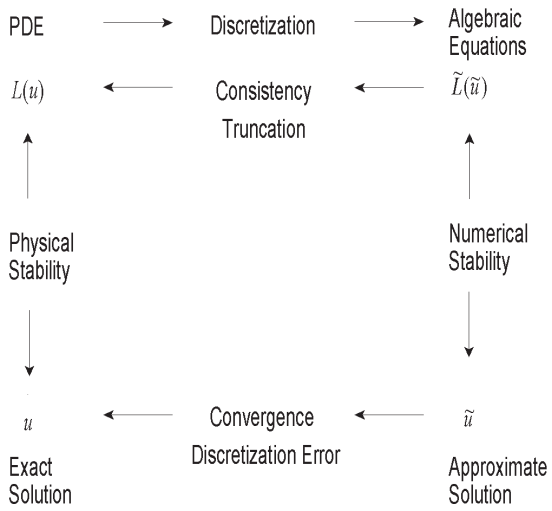

Abbildung 2.3: Discrete approximation of a PDE and its solution (after Fletcher 1990)

2.2.2 Errors and Accuracy

The following discussion of convergence, consistency, and stability is concerned with the behavior of the approximate solution if discretization sizes $(\Delta t, \Delta x)$ tends to zero. In practice, approximate solutions have to be obtained on finite grids which must be calculable on available computers. Therefore, errors and achievable accuracy are of great interest.

If we want to represent continuous systems with the help of discrete methods, of course, we introduce a number of errors. Following types of errors may occur: solution error, discretization error, truncation error, and round-off errors. Round-off errors may result from solving equation systems. Truncation errors

are omitted from finite difference approximations. This means, the representation of differentiated terms by algebraic expressions connecting nodal values on a finite grid introduces a certain error. It is desirable to choose the algebraic terms in a way that only errors as small as possible are invoked. The accuracy of replaced differentiated terms by algebraic ones can be evaluated by considering the so-called truncation error. Truncation error analysis can be conducted by Taylor series expansion (TSE). However, the evaluation of this terms in the TSE relies on the exact solution being known. The truncation error is likely to be a progressively more accurate indicator of the solution error as the discretization grid is refined.

There exist two techniques to evaluate accuracy of numerical schemes. At first, the algorithm can be applied to a related but simplified problem, which possesses an analytical solution (e.g. Burgers´ equation which models convective and diffusive momentum transport). The second method is to obtain solutions on progressively refined grids and to proof convergence. In general, accuracy can be improved by use of higher-order schemes or grid refinement.

2.2.3 Convergence

Definition: A solution of the algebraic equations which approximate a given PDE is said to be convergent if the approximate solution approaches the exact solution of the PDE for each value of the independent variable as the grid spacing tends to zero. Thus we require

$$\lim_{\Delta t, \Delta x \to 0} \mid u_j^n - u(t_n, x_j) \mid = 0 \tag{2.2}$$

Or in other words, the approximate solution converges to the exact one as the grid sizes becomes infinitely small. The difference between exact and approximate solution is the solution error, denoted by

$$\varepsilon_j^n = \mid u_j^n - u(t_n, x_j) \mid \tag{2.3}$$

The magnitude of the solution error typically depends on grid spacing and approximations to the derivatives in the PDE.

Theoretical proof of convergence is generally difficult, except very simple cases. As an example, proof of convergence for the approximate solution to the diffusion equation is given by Noye (1984, pp. 117-119) who used a FTCS scheme (forward differences in time - central difference in space, see section 3.2.2). For indication of convergence, comparison of approximate solutions on progressively refined grids is used in practice. For PDEs which possesses an analytical solution, like the 1-D advection-diffusion problem, it is possible to test convergence by comparison of numerical solutions on progressively refine grids with the exact solution of the PDE.

2.2.4 Consistency

Definition: The system of algebraic equations (SAE) generated by the discretization process is said to be consistent with the original partial differential equation (PDE) if, in the limit that the grid spacing tends to zero, the SAE is equivalent to the PDE at each grid point. Thus we require

$$\lim_{\Delta t, \Delta x \to 0} \mid \tilde{L}(u_j^n) - L(u[t_n, x_j]) \mid = 0 \tag{2.4}$$

Or in other words, the SAE converges to the PDE as the grid size becomes zero. Obviously, consistency is necessary for convergence of the approximate solution. However, consistency is not sufficient to guarantee convergence. Although the SAE might be consistent, it does not follow that the approximate solution converges to the exact one, e.g. for unstable schemes. As an example, solutions of the FTCS algorithm diverge rapidly if the scheme is weighted backwards ($\theta > 0.5$). This example emphases that, as indicated by the Lax-Equivalence-Axiom, both consistency and stability are necessary for convergence. Consistency analysis can be conducted by substitution of the exact solution into the algebraic equations resulting from the discretization process. The exact solution is represented as a TSE. Finally, we obtain an equation which consists the original PDE plus a reminder. For consistency the reminder should vanish as the grid size tends to zero. In section 3 we present several examples for consistency analysis.

2.2.5 Stability

Frequently, the matrix method and the von Neumann method are used for stability analysis. In both cases possible growth of the error between approximate and exact solution will be examined. It is generally accepted that the matrix method is less reliable as the von Neumann method. Using the von Neumann method, error at one time level is expanded as a finite Fourier series. For this purpose, initial conditions are represented by a Fourier series. Each mode of the series will grow or decay depending on the discretization. If a particular mode grows without bounds, then an unstable solution exists for this discretization.

2.3 Solution Process

We recall, that the overall solution procedure for PDEs consists of the two major steps: discretization and solution processes (Fig. 2.2). In this section we give a brief introduction to the solution process for equation systems, resulting from discretization methods such as finite difference (FDM), finite element (FEM) and finite volume methods (FVM) (see chapters 6-8). More details on the solution of equation systems can be found e.g. in Meyer (1990), Hackbusch (1991), Habbar (1995), Schwetlick & Kretschmar (1991), Knabner & Angermann (2000), Wriggers (2001).

Several problems in environmental fluid dynamics lead to non-linear PDEs such as non-linear flow (chapter 12), density-dependent flow (chapter 14), multi-phase flow (chapters 15,16) [8]. The resulting algebraic equation system can be written in a general way, indicating the dependency of system matrix $\mathbf{A}$ and right-hand-side vector $\mathbf{b}$ on the solution vector $\mathbf{x}$. Consequently, it is necessary to employ iterative methods to obtain a solution.

$$\mathbf{A}(\mathbf{x})\,\mathbf{x} - \mathbf{b}(\mathbf{x}) = \mathbf{0} \tag{2.5}$$

In the following we consider methods for solving linear equation systems (section 2.3.1) and non-linear equation systems (section 2.3.2). Aspects of the implementation of solvers in an object-oriented way are discussed in chapter 10.

2.3.1 Linear Solver

The linear version of equation (2.5) is given by

$$\mathbf{A}\,\mathbf{x} - \mathbf{b} = \mathbf{0} \tag{2.6}$$

In general there are two types of methods: direct and iterative algorithms. Direct methods may be advantageous for some non-linear problems. A solution will be produced even for systems with ill-conditioned matrices. On the other hand, direct schemes are very memory consuming. The required memory is in the order of $0(nb^2)$, with n the number of unknowns and b the bandwidth of the system matrix. Therefore, it is always useful to apply algorithms for bandwidth reduction (Cuthill & McKee (1969), Gibbs et al. (1976)). Iterative methods have certain advantages in particular for large systems with sparse matrices. They can be very efficient in combination with non-linear solver.

The following list reveals an overview on existing methods for solving linear algebraic equation systems.

- Direct methods
  - Gaussian elimination
  - Block elimination (to reduce memory requirements for large problems)
  - Cholesky decomposition
  - Frontal solver
- Iterative methods
  - Linear steady methods (Jacobian, Gauss-Seidel, Richardson and block iteration methods)
  - Gradient methods (CG) (also denoted as Krylov subspace methods)

2.3.1.1 Direct Methods

Application of direct methods to determine the solution of equation (2.6)

$$\mathbf{x} = \mathbf{A}^{-1}\,\mathbf{b} \tag{2.7}$$

requires an efficient techniques to invert the system matrix.

As a first example we consider the Gaussian elimination technique. If matrix $\mathbf{A}$ is not singular (i.e. $\det \mathbf{A} \neq 0$), can be composed in following way.

$$\mathbf{P}\,\mathbf{A} = \mathbf{L}\,\mathbf{U} \tag{2.8}$$

with a permutation matrix $\mathbf{P}$ and the lower $\mathbf{L}$ as well as the upper matrices $\mathbf{U}$ in triangle forms.

$$\mathbf{L} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ l_{n1} & \cdots & 1 \end{bmatrix} \quad , \quad \mathbf{U} = \begin{bmatrix} u_{11} & \cdots & u_{1n} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & u_{nn} \end{bmatrix} \tag{2.9}$$

If $\mathbf{P} = \mathbf{I}$ the matrix $\mathbf{A}$ has a so-called LU-decomposition: $\mathbf{A} = \mathbf{L}\,\mathbf{U}$. The task reduces then to calculate the lower and upper matrices and invert them. Once $\mathbf{L}$ and $\mathbf{U}$ are determined, the inversion of $\mathbf{A}$ is trivial due to the triangle structure of $\mathbf{L}$ and $\mathbf{U}$.

Assuming that beside non-singularity and existing LU-decomposition, $\mathbf{A}$ is symmetrical additionally, we have $\mathbf{U} = \mathbf{D}\,\mathbf{L}^T$ with $\mathbf{D} = diag(d_i)$. Now we can conduct the following transformations.

$$\mathbf{A} = \mathbf{L}\,\mathbf{U} = \mathbf{L}\,\mathbf{D}\,\mathbf{L}^T = \underbrace{\mathbf{L}\sqrt{\mathbf{D}}}_{\tilde{\mathbf{L}}}\underbrace{\sqrt{\mathbf{D}}\,\mathbf{L}^T}_{\tilde{\mathbf{L}}^T} \tag{2.10}$$

The splitting of $\mathbf{D}$ requires that $\mathbf{A}$ is positive definite thus that $\forall d_i > 0$. The expression

$$\mathbf{A} = \tilde{\mathbf{L}}\,\tilde{\mathbf{L}}^T \tag{2.11}$$

is denoted as Cholesky decomposition. Therefore, the lower triangle matrices of both the Cholesky and the Gaussian method are connected via

$$\tilde{\mathbf{L}} \; = \mathbf{L}^T\sqrt{\mathbf{D}} \tag{2.12}$$

2.3.1.2 Iterative Methods

High resolution FEM leads to large equation systems with sparse system matrices. For this type of problems iterative equation solver are much more efficient than direct solvers. Concerning the application of iterative solver we have to distinguish between symmetrical and non-symmetrical system matrices with

Symmetric Matrices	Non-symmetric Matrices
CG	BiCG
Lanczos	CGStab
Gauss-Seidel, Jacobian, Richards	GMRES
SOR and block-iteration	CGNR

different solution methods. The efficiency of iterative algorithms, i.e. the reduction of iteration numbers, can be improved by the use of pre-conditioning techniques).

The last two rows of solver for symmetric problems belong to the linear steady iteration methods. The algorithms for solving non-symmetrical systems are also denoted as Krylov subspace methods.

2.3.2 Non-Linear Solver

In this section we present a description of selected iterative methods that are commonly applied to solve non-linear problems.

- Picard method (fixpoint iteration) (section 2.3.2.1)
- Newton methods (section 2.3.2.2)
- Cord slope method
- Dynamic relaxation method

All methods call for an initial guess of the solution to start but each algorithm uses a different scheme to produce a new (and hopefully closer) estimate to the exact solution. The general idea is to construct a sequence of linear sub-problems which can be solved with ordinary linear solver (see section 2.3.1).

2.3.2.1 Picard Method

The general algorithm of the Picard method can be described as follows. We consider a non-linear equation written in the form

$$\mathbf{A}(\mathbf{x})\,\mathbf{x} - \mathbf{b}(\mathbf{x}) = 0 \tag{2.13}$$

We start the iteration by assuming an initial guess $\mathbf{x}_0$ and we use this to evaluate the system matrix $\mathbf{A}(\mathbf{x}_0)$ as well as the right-hand-side vector $\mathbf{b}(\mathbf{x}_0)$. Thus this equation becomes linear and it can be solved for the next set of $\mathbf{x}$ values.

$$\begin{aligned} \mathbf{A}(\mathbf{x}_{k-1})\,\mathbf{x}_k - \mathbf{b}(\mathbf{x}_{k-1}) &= 0 \\ \mathbf{x}_k &= \mathbf{A}^{-1}(\mathbf{x}_{k-1})\,\mathbf{b}(\mathbf{x}_{k-1}) \end{aligned} \tag{2.14}$$

Repeating this procedure we obtain a sequence of successive solutions for $\mathbf{x}_k$. During each iteration loop the system matrix and the right-hand-side vector must be updated with the previous solution. The iteration is performed until satisfactory convergence is achieved. A typical criterion is e.g.

$$\varepsilon \geq \frac{\parallel \mathbf{x}_k - \mathbf{x}_{k-1} \parallel}{\parallel \mathbf{x}_k \parallel} \tag{2.15}$$

where ε is a user-defined tolerance criterion. For the simple case of a non-linear equation $\mathbf{x} = \mathbf{b}(\mathbf{x})$ (i.e. $\mathbf{A} = \mathbf{I}$), the iteration procedure is graphically illustrated in Fig. 2.4. To achieve convergence of the scheme it has to be guaranteed that the iteration error

$$e_k = \parallel \mathbf{x}_k - \mathbf{x} \parallel < C \parallel \mathbf{x}_{k-1} - \mathbf{x} \parallel^p = e_{k-1} \tag{2.16}$$

or, alternatively, the distance between successive solutions will reduce

$$\parallel \mathbf{x}_{k+1} - \mathbf{x}_k \parallel < \parallel \mathbf{x}_k - \mathbf{x}_{k-1} \parallel^p \tag{2.17}$$

where p denotes the convergence order of the iteration scheme. It can be shown that the iteration error of the Picard method decreases linearly with the error at the previous iteration step. Therefore, the Picard method is a first-order convergence scheme.

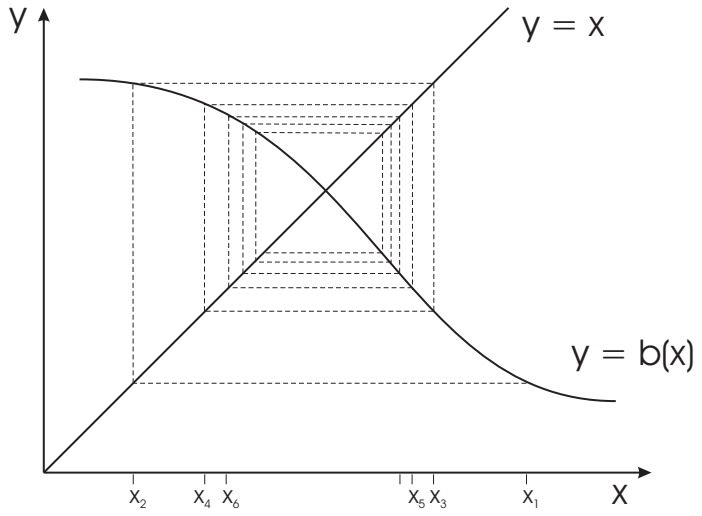

Abbildung 2.4: Graphical illustration of the Picard iteration method

2.3.2.2 Newton Method

In order to improve the convergence order of non-linear iteration methods, i.e. derive higher-order schemes, the Newton-Raphson method can be employed. To

describe this approach, we consider once again the non-linear equation (2.5).

$$\mathbf{R}(\mathbf{x}) = \mathbf{A}(\mathbf{x})\,\mathbf{x} - \mathbf{b}(\mathbf{x}) = 0 \tag{2.18}$$

Assuming that the residuum $\mathbf{R}(\mathbf{x})$ is a continuous function, we can develop a Taylor series expansion about any known approximate solution $\mathbf{x}_k$.

$$\mathbf{R}_{k+1} = \mathbf{R}_k + \left[\frac{\partial \mathbf{R}}{\partial \mathbf{x}}\right]_k \Delta\mathbf{x}_{k+1} + 0(\Delta\mathbf{x}_{k+1}^2) \tag{2.19}$$

Second- and higher-order terms are truncated in the following. The term $\partial\mathbf{R}/\partial\mathbf{x}$ represents tangential slopes of $\mathbf{R}$ with respect to the solution vector and it is denoted as the Jacobian matrix $\mathbf{J}$. As a first approximation we can assume $\mathbf{R}_{k+1} = 0$. Then the solution increment can be immediately calculated from the remaining terms in equation (4.47).

$$\Delta\mathbf{x}_{k+1} = -\mathbf{J}_k^{-1}\,\mathbf{R}_k \tag{2.20}$$

where we have to cope with the inverse of the Jacobian. The iterative approximation of the solution vector can be computed now from the increment.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_{k+1} \tag{2.21}$$

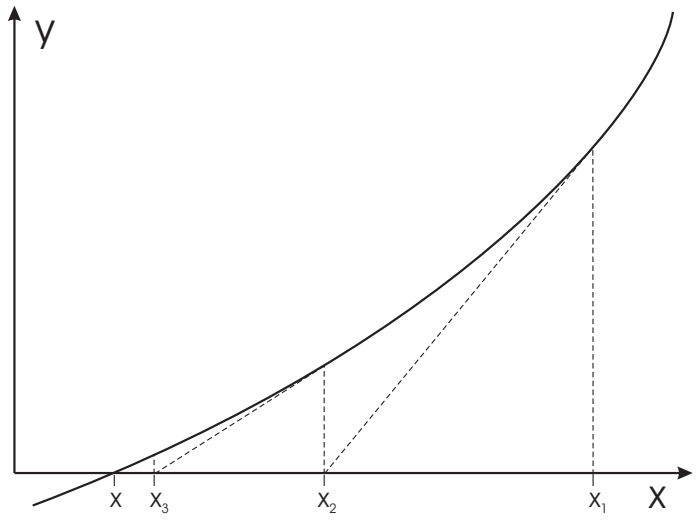

Abbildung 2.5: Graphical illustration of the Newton-Raphson iteration method

Once an initial guess is provided, successive solutions of $\mathbf{x}_{k+1}$ can be determined using equations (4.48) and (4.49) (Fig. 4.18). The Jacobian has to re-evaluated and inversed at every iteration step, which is a very time-consuming procedure

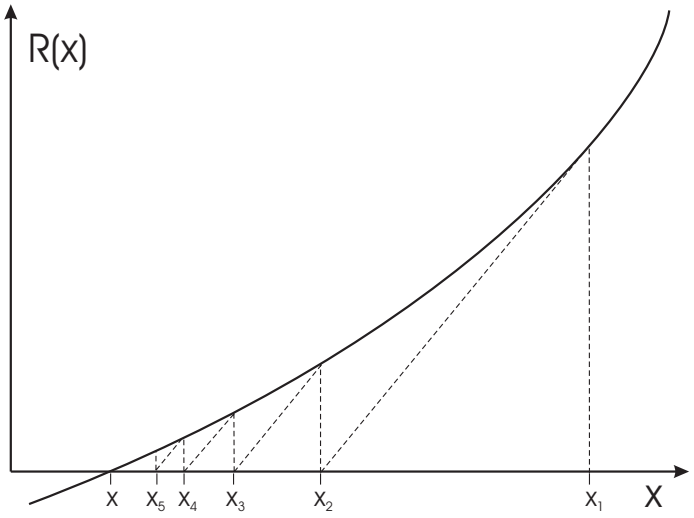

Abbildung 2.6: Graphical illustration of the modified Newton-Raphson iteration method

in fact. At the expense of slower convergence, the initial Jacobian $\mathbf{J}_0$ may be kept and used in the subsequent iterations. Alternatively, the Jacobian can be updated in certain iteration intervals. This procedure is denoted as modified or 'initial slope' Newton method (Fig. 4.15).

The convergence velocity of the Newton-Raphson method is second-order. It is characterized by the expression.

$$\| \mathbf{x}_{k+1} - \mathbf{x} \| \leq C \| \mathbf{x}_k - \mathbf{x} \|^2 \tag{2.22}$$

2.4 Problems

Solution Procedure

1 Give examples of approximation methods for solving differential equations.

2 What are the two basic steps of the solution procedure for discrete approximation methods ?

Theory of Discrete Approximation

3 Explain the term convergence of an approximate solution for a PDE. Give a mathematical definition for that.

4 Explain the term consistency of an approximation scheme for a PDE. Give a mathematical definition for that.

5 Explain the term stability of an approximate solution for a PDE. What general methods for stability analysis do you know ?

6 Explain the relationships between the terms convergence, consistency, and stability using Fig. (2.3).

7 What does the Lax equivalence theorem postulate ?

8 What are the three analysis steps for discrete approximation schemes ?

Solution Process

9 Using the Newton-Raphson method solve the following set of non-linear equations:

$$f_1(x_1, x_2) = x_1^2 + x_2^2 - 5 = 0$$
$$f_2(x_1, x_2) = x_1 + x_2 - 1 = 0$$

Kapitel 3

Finite Difference Method

The basic steps in order to set up a finite difference scheme are:

- definition of a space discretization by which the mesh points are distributed along families of non-intersecting lines,
- development of the unknown functions by means of Taylor series expansion (TSE) around grid points
- replacement of derivative terms in the partial differential equations (PDE) with equivalent finite difference expressions.

3.1 Approximation of Derivatives

In this section we present basic techniques to discretize derivatives. Algebraic formulas are constructed first by Taylor series expansion, to evaluate the truncation error and second by a more general technique.

3.1.1 Taylor Series Expansion (TSE)

A well-known technique to represent the value of a function $u(t,x)$ at a node (j,n) in terms of values of the function at nearby points is the Taylor series expansion

in time

$$u_j^{n+1} = \sum_{m=0}^{\infty} \frac{\Delta t^m}{m!} \left[\frac{\partial^m u}{\partial t^m}\right]_j^n \tag{3.1}$$

in space

$$u_{j+1}^n = \sum_{m=0}^{\infty} \frac{\Delta x^m}{m!} \left[\frac{\partial^m u}{\partial x^m}\right]_j^n \tag{3.2}$$

This series may be truncated after any number of terms. The resulting **truncation error** is then defined by the next term being left in the series expansion. The above expressions can be rewritten in following forms.

$$u_j^{n+1} = u_j^n + \Delta t \left[\frac{\partial u}{\partial t}\right]_j^n + \frac{\Delta t^2}{2} \left[\frac{\partial^2 u}{\partial t^2}\right]_j^n + 0(\Delta t^3) \tag{3.3}$$

$$u_{j+1}^n = u_j^n + \Delta x \left[\frac{\partial u}{\partial x}\right]_j^n + \frac{\Delta x^2}{2} \left[\frac{\partial^2 u}{\partial x^2}\right]_j^n + 0(\Delta x^3) \tag{3.4}$$

As can be seen, the truncation error depends mainly on the step size Δt or Δx, respectively. The error introduced in this approximation rapidly reduces as the step size decreases.

3.1.2 First-Order Derivatives

By rearranging the above equations, a finite difference expression for first-order derivatives can be directly obtained.

$$\left[\frac{\partial u}{\partial t}\right]_j^n = \frac{u_j^{n+1} - u_j^n}{\Delta t} - \frac{\Delta t}{2} \left[\frac{\partial^2 u}{\partial t^2}\right]_j^n + 0(\Delta t^2) \tag{3.5}$$

$$\left[\frac{\partial u}{\partial x}\right]_j^n = \frac{u_{j+1}^n - u_j^n}{\Delta x} - \frac{\Delta x}{2} \left[\frac{\partial^2 u}{\partial x^2}\right]_j^n + 0(\Delta x^2) \tag{3.6}$$

Clearly, the following standard finite difference expressions for first-order derivatives are accurate at first order:

Forward difference approximation

$$\left[\frac{\partial u}{\partial x}\right]_j^n = \frac{u_{j+1}^n - u_j^n}{\Delta x} + 0(\Delta x) \tag{3.7}$$

Backward difference approximation

$$\left[\frac{\partial u}{\partial x}\right]_j^n = \frac{u_j^n - u_{j-1}^n}{\Delta x} + 0(\Delta x) \tag{3.8}$$

A second-order scheme is obtained from central differences. To this purpose, we subtract the following TSE from each other.

$$\begin{aligned} u_{j+1}^n &= u_j^n + \Delta x \left[\frac{\partial u}{\partial x}\right]_j^n + \frac{\Delta x^2}{2}\left[\frac{\partial^2 u}{\partial x^2}\right]_j^n + 0(\Delta x^3) \\ u_{j-1}^n &= u_j^n - \Delta x \left[\frac{\partial u}{\partial x}\right]_j^n + \frac{\Delta x^2}{2}\left[\frac{\partial^2 u}{\partial x^2}\right]_j^n - 0(\Delta x^3) \end{aligned} \tag{3.9}$$

Central difference approximation

$$\left[\frac{\partial u}{\partial x}\right]_j^n = \frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} + 0(\Delta x^2) \tag{3.10}$$

A geometric interpretation (i.e. slopes) of forward and backward difference formulas is given in following graphic. Note, forward and backward differences only use information from one side of the corresponding grid point, whereas central differences use information from both sides.

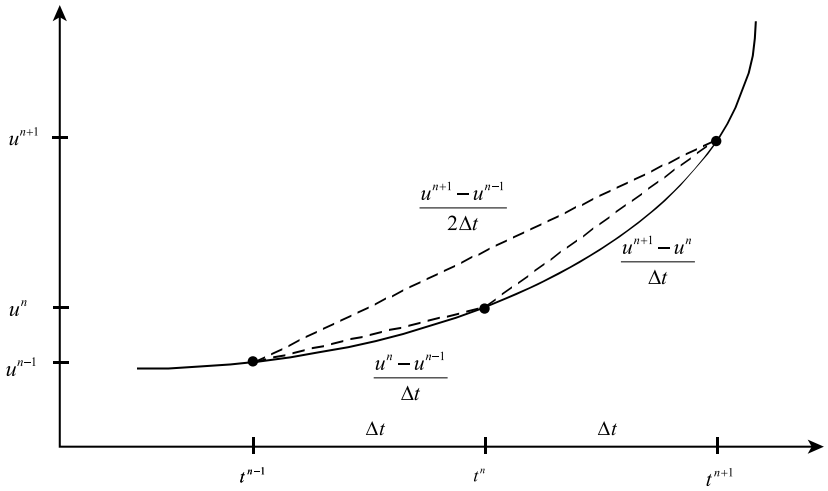

Abbildung 3.1: Geometric interpretation of finite difference approximations

3.1.3 Second-Order Derivatives

Finite difference schemes for higher-order derivatives can be obtained by repeated application of first-order approximations.

$$\left[\frac{\partial^2 u}{\partial x^2}\right]_j^n \approx \frac{1}{\Delta x}\left(\left[\frac{\partial u}{\partial x}\right]_{j+1}^n - \left[\frac{\partial u}{\partial x}\right]_j^n\right)$$

$$\approx \quad \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2} \tag{3.11}$$

The FD scheme for a second-order derivative can be obtained by adding the first-order schemes given in eqn (3.9). From the TSE can be seen that the symmetrical, central differences scheme is of second-order accuracy.

$$\left[\frac{\partial^2 u}{\partial x^2}\right]_j^n = \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2} + \frac{\Delta x^2}{12}\left[\frac{\partial^4 u}{\partial x^4}\right]_j^n + ... \tag{3.12}$$

Note, that first-order schemes for n-th derivatives will introduce higher-order $n+1$-th derivative terms, which result in large numerical diffusion/dispersion terms. Therefore, we may infer the general guidance, higher-order schemes for derivatives should be preferred.

3.2 Diffusion Equation

Typical diffusion problems are mass diffusion, heat conduction, and viscous fluid flow. The diffusion equation is a parabolic PDE.

$$\frac{\partial u}{\partial t} - \alpha \frac{\partial^2 u}{\partial x^2} = 0 \tag{3.13}$$

The unknown field variable u may be interpreted as velocity, vorticity, temperature or concentration depending on whether the diffusion of momentum, vorticity, heat or mass is being considered. Boundary as well as initial conditions must be specified to obtain an unique solution of such a PDE.

In this chapter we introduce several (explicit and implicit) finite difference schemes for the numerical solution of the diffusion equation (Fig. 3.2) Attention will be paid to stability and accuracy of the several algorithms. Moreover, the implementation of initial as well as boundary conditions will also be considered.

Analysis of approximation schemes consists of three steps:

- Develop the **algebraic scheme**,
- Check **consistency** of the algebraic approximate equation,
- Investigate **stability** behavior of the scheme.

3.2.1 Explicit and Implicit Schemes

A discrete scheme gives the solution at a new time level in terms of the known solution at earlier time levels. An algebraic scheme is denoted as an explicit one if the unknown at new time level u_j^{n+1} depends only on values from old time

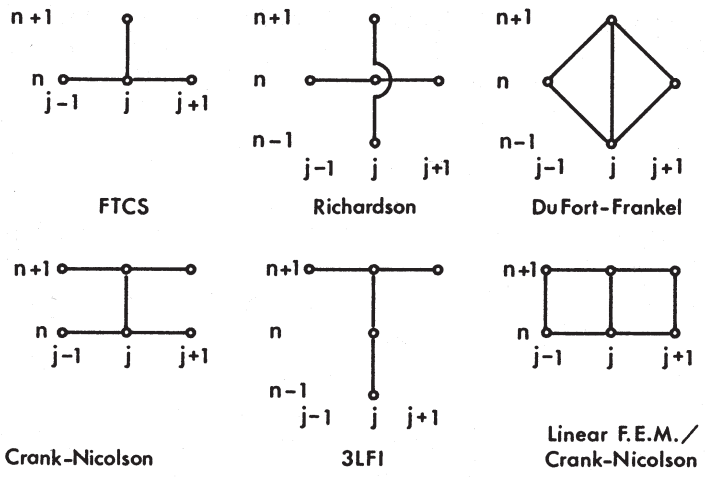

Abbildung 3.2: Overview on finite difference schemes

level u_j^n. The FTCS, Richardson, and DuFort-Frankel algorithms are examples of explicit schemes. For implicit schemes spatial derivatives are evaluated also at the new time level. As a result equations are coupled for each node at new time level and, therefore, a system of algebraic equations has to be solved.

Typically implicit schemes are unconditionally stable. However the use of implicit schemes leads to a system of coupled algebraic equations at new time level. As a result, perturbations introduced at one node affects the solution to all other nodes at the next time level. Those perturbations are propagated immediately through entire domain. Physically this behavior corresponds to a diffusion process. Therefore, this type of error propagation is denoted as numerical diffusion or dispersion.

3.2.2 Explicit FTCS Scheme

Algebraic Scheme

One of the simplest finite difference representations of the diffusion equation is the forward time, centered space scheme (FTCS). To obtain a discretized form of the PDE we replace the time derivatives by a two-point forward difference formula

$$\left[\frac{\partial u}{\partial t}\right]_j^n \approx \frac{u_j^{n+1} - u_j^n}{\Delta t} \tag{3.14}$$

and the spatial derivative by a three-point centered difference formula.

$$\left[\frac{\partial^2 u}{\partial x^2}\right]_j^n \approx \frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2} \tag{3.15}$$

The resulting algebraic equation is denoted as FTCS scheme.

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} - \alpha \frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2} = 0 \tag{3.16}$$

The discretization process implies, that the original problem (i.e. PDE for the exact, continuous solution) $u(t, x)$ has been replaced with the problem of finding discrete values u_j^n at node j at time n. Rearranging the above formula we obtain the expression for the unknown value u_j^{n+1} at new time level in terms of known values at old time level.

$$u_j^{n+1} = u_j^n + \frac{\alpha \Delta t}{\Delta x^2}(u_{j-1}^n - 2u_j^n + u_{j+1}^n) \tag{3.17}$$

where

$$\boxed{Ne = \frac{\alpha \Delta t}{\Delta x^2}} \tag{3.18}$$

is the **Neumann number**. For the above developed explicit scheme only a single unknown appears on the left hand side of the algebraic equation.

Consistency Analysis

Substitution of the exact solutions (i.e. its Taylor series expansion) for the differential terms (eqns 3.5, 3.12) into the algebraic scheme yields

$$\left[\frac{\partial u}{\partial t} - \alpha \frac{\partial^2 u}{\partial x^2}\right]_j^n + E_j^n = 0 \tag{3.19}$$

with truncation error

$$E_j^n = \left[\frac{\Delta t}{2}\frac{\partial^2 u}{\partial t^2} - \alpha \frac{\Delta x^2}{12}\frac{\partial^4 u}{\partial x^4}\right]_j^n + 0(\Delta t^2, \Delta x^4) = 0 \tag{3.20}$$

The leading term of the truncation error indicates that the FTCS scheme is first-order accurate in time and second-order accurate in space. The FTCS scheme is consistent with the PDE. As the step size tends to zero both equations (SAE and PDE) will coincide.

Stability Analysis

Stability is the tendency for any perturbations in the approximate solution to decay. As mentioned in the previous section, several methods exist for stability analysis such as the matrix method and the von Neumann method.

(1) Matrix Method

Using the matrix method, the system of algebraic equations is expressed in matrix form first. For stability consideration then the eigenvalues are examined. The algebraic scheme (3.17) can be written as

$$\mathbf{u}^{n+1} = \mathbf{A}\mathbf{u}^n \qquad , \qquad n = 0, 1, 2, ... \tag{3.21}$$

where $\mathbf{A}$ is a $(np-2)$ (np = number of grid points) square matrix transforming the result vector u from old to new time level

$$\mathbf{A} = \begin{bmatrix} 1-2\frac{\alpha\Delta t}{\Delta x^2} & \frac{\alpha\Delta t}{\Delta x^2} & & & & \\ \frac{\alpha\Delta t}{\Delta x^2} & \ddots & \ddots & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \frac{\alpha\Delta t}{\Delta x^2} & \\ & & & \frac{\alpha\Delta t}{\Delta x^2} & 1-2\frac{\alpha\Delta t}{\Delta x^2} & \end{bmatrix} \tag{3.22}$$

$$\mathbf{u}^n = \begin{bmatrix} u_2^n \\ u_3^n \\ ... \\ u_{np-2}^n \\ u_{np-1}^n \end{bmatrix} \tag{3.23}$$

Note, it is assumed that the boundary values u_1^n , u_{np}^n are known. It can be shown that the result vector is bounded for propagating time if the eigenvalues λ_j of matrix $\mathbf{A}$ are defined and have absolute values less or equal to one, i.e.

$$|\lambda_j| \leq 1 \qquad , \qquad \forall j \tag{3.24}$$

The eigenvalues of the above tridiagonal matrix are

$$\lambda_j = 1 - 4\frac{\alpha\Delta t}{\Delta x^2}\sin^2\left[\frac{j\pi}{2(np-1)}\right] \qquad , \qquad j = 1, ..., np-2 \tag{3.25}$$

Then stability condition (3.24) can be written as

$$-1 \leq 1 - 4\frac{\alpha\Delta t}{\Delta x^2}\sin^2\left[\frac{j\pi}{2(np-1)}\right] \leq +1 \tag{3.26}$$

Whereas the right-hand side of the inequality is always satisfied, the left-hand side yields the stability condition for the FTCS scheme.

$$\boxed{Ne = \frac{\alpha \Delta t}{\Delta x^2} \leq \frac{1}{2}} \tag{3.27}$$

The matrix method is based on Fourier series representation of the result vector. Therefore, this method yields growth or decay of individual modes in a Fourier series representation of the initial conditions.

Incorporation of Neumann boundary conditions result in a slightly different scheme. Suppose at point $(j = 1)$

$$\frac{\partial u(t, x_1)}{\partial x} = g(t) \tag{3.28}$$

the algebraic equation (4.4) for this point can be written as

$$\begin{aligned} \frac{u_1^{n+1} - u_1^n}{\Delta t} - \alpha \left(\frac{u_2^n - u_1^n}{\Delta x^2} - \frac{1}{\Delta x} \left[\frac{\partial u}{\partial x} \right]_1^n \right) &= \\ = \frac{u_1^{n+1} - u_1^n}{\Delta t} - \alpha \left(\frac{u_2^n - u_1^n}{\Delta x^2} - \frac{1}{\Delta x} g(t) \right) &= 0 \end{aligned} \tag{3.29}$$

Rearranging the above equation, we obtain the following explicit scheme for the unknown value at new time level.

$$u_1^{n+1} = u_1^n + \frac{\alpha \Delta t}{\Delta x^2} [u_2^n - u_1^n - \Delta x \, g(t)] \tag{3.30}$$

The matrix form for the system of algebraic equations is now

$$\mathbf{u}^{n+1} = \mathbf{A}\mathbf{u}^n + \mathbf{b} \qquad , \qquad n = 0, 1, 2, ... \tag{3.31}$$

with a modified system matrix and an additional vector resulting from Neumann type boundary conditions.

$$\mathbf{A} = \begin{bmatrix} 1 - \frac{\alpha \Delta t}{\Delta x^2} & \frac{\alpha \Delta t}{\Delta x^2} & & & \\ \frac{\alpha \Delta t}{\Delta x^2} & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \frac{\alpha \Delta t}{\Delta x^2} \\ & & & \frac{\alpha \Delta t}{\Delta x^2} & 1 - \frac{\alpha \Delta t}{\Delta x^2} \end{bmatrix} \tag{3.32}$$

$$\mathbf{b} = \begin{bmatrix} -\frac{\alpha \Delta t}{\Delta x} g(t^n) \\ 0 \\ ... \\ 0 \\ 0 \end{bmatrix} \tag{3.33}$$

It can be seen, that one eigenvalue will be slightly modified. The maximum eigenvalues can be determined by use of the power method. In general, the implementation of Neumann boundary conditions slightly reduces numerical stability (Fletcher 1990).

(2) Von Neumann Method

The von Neumann method is most commonly used, but it is restricted to linear initial value problems with constant coefficients. For more sophisticated problems including variable coefficients, non-linearities, and complicated boundary conditions, this method is useful to determine necessary conditions. But it will be fail to define sufficient conditions for stability. The von Neumann method is based on Fourier analysis. Stability or instability of the algebraic scheme is investigated by considering whether defined Fourier components of the error decay or amplify in progressing to the next time level.

3.2.3 Fully Implicit Scheme

Algebraic Scheme

For the fully implicit scheme, the spatial derivative (diffusion term) is evaluated completely at the unknown time level.

$$\left[\frac{\partial^2 u}{\partial x^2}\right]_j^{n+1} \approx \frac{u_{j-1}^{n+1} - 2u_j^{n+1} + u_{j+1}^{n+1}}{\Delta x^2} \tag{3.34}$$

The algebraic scheme for the fully implicit discretization of the diffusion equation is then

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} - \alpha \frac{u_{j-1}^{n+1} - 2u_j^{n+1} + u_{j+1}^{n+1}}{\Delta x^2} = 0 \tag{3.35}$$

Rearranging the above formula we obtain an expression for the unknown values at new time level in terms of known values at old time level.

$$\frac{\alpha \Delta t}{\Delta x^2}\left(-u_{j-1}^{n+1} + 2u_j^{n+1} - u_{j+1}^{n+1}\right) + u_j^{n+1} = u_j^n \tag{3.36}$$

Consistency Analysis

To analyze consistency we start from equation (3.35) which will be applied to the exact solution. First, outside terms are expanded as Taylor series about the central node j at new time level and substituted into equation (3.35).

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} - \alpha \left\{ \left[\frac{\partial^2 u}{\partial x^2}\right]_j^{n+1} + \frac{\Delta x^2}{12}\left[\frac{\partial^4 u}{\partial x^4}\right]_j^{n+1} + \frac{\Delta x^4}{360}\left[\frac{\partial^6 u}{\partial x^6}\right]_j^{n+1} + ... \right\} = 0 \tag{3.37}$$

Second, central terms at new time level are expanded about the (j,n)-th node and then substituted.

$$\begin{aligned}
&\left[\frac{\partial u}{\partial t}\right]_j^n + \frac{\Delta t}{2}\left[\frac{\partial^2 u}{\partial t^2}\right]_j^n + \frac{\Delta t^2}{6}\left[\frac{\partial^3 u}{\partial t^3}\right]_j^n + ... \\
-\alpha&\left[\frac{\partial^2 u}{\partial x^2}\right]_j^n + \Delta t\left[\frac{\partial^3 u}{\partial x^2 \partial t}\right]_j^n + \frac{\Delta t^2}{2}\left[\frac{\partial^4 u}{\partial x^2 \partial t^2}\right]_j^n + ... \\
&+\frac{\Delta x^2}{12}\left(\left[\frac{\partial^4 u}{\partial x^4}\right]_j^n + \Delta t\left[\frac{\partial^5 u}{\partial x^4 \partial t}\right]_j^n + ...\right) \\
&+\frac{\Delta x^4}{360}\left(\left[\frac{\partial^6 u}{\partial x^6}\right]_j^n + ...\right) = 0
\end{aligned} \tag{3.38}$$

We use the original diffusion equation and related time derivatives to substitute all spatial derivatives.

$$\left[\frac{\partial u}{\partial t} - \alpha\frac{\partial^2 u}{\partial x^2}\right]_j^n = 0 \quad , \quad \left[\frac{\partial^2 u}{\partial t^2} - \alpha^2\frac{\partial^4 u}{\partial x^4}\right]_j^n = 0 \quad , \quad \left[\frac{\partial^3 u}{\partial t^3} - \alpha^3\frac{\partial^6 u}{\partial x^6}\right]_j^n = 0 \tag{3.39}$$

By this step it would be possible to represent the truncation error in terms of derivatives and spatial grid size only. Now we can simply the above equation to

$$\left[\frac{\partial u}{\partial t} - \alpha\frac{\partial^2 u}{\partial x^2}\right]_j^n + E_j^n = 0 \tag{3.40}$$

and, finally, we determine the truncation error of the fully implicit scheme.

$$\begin{aligned}
E_j^n = -\frac{\Delta t}{2}(1 + \frac{1}{6}\frac{\Delta x^2}{\alpha \Delta t})\left[\frac{\partial^2 u}{\partial t^2}\right]_j^n &- \frac{\Delta t^2}{3}(1 + \frac{1}{4}\frac{\Delta x^2}{\alpha \Delta t} + \frac{1}{120}\frac{\Delta x^4}{\alpha^2 \Delta t^2})\left[\frac{\partial^3 u}{\partial t^3}\right]_j^n + \\
&+ \quad 0(\Delta t^3, \Delta x^4)
\end{aligned} \tag{3.41}$$

The truncation error vanishes as time step size tends to zero, i.e. algebraic and partial differential equations coincide in the limit. Consequently, the algebraic equation is consistent with the PDE. In contrast to the explicit FTCS scheme (3.20), there is no specific choice of time and spatial grid sizes to reduce the truncation error further. Consistency order of the fully implicit scheme is therefore: $0(\Delta t, \Delta x^2)$.

Stability Analysis

We use the matrix method for stability analysis. In analogy to the procedure for the explicit FTCS scheme (section 3.2.2) we write the system of algebraic

equations (3.36) in matrix form. If neglecting boundary conditions, the system matrix for the fully implicit scheme is

$$\mathbf{A} = \begin{bmatrix} 1+2\frac{\alpha\Delta t}{\Delta x^2} & -\frac{\alpha\Delta t}{\Delta x^2} & & & \\ -\frac{\alpha\Delta t}{\Delta x^2} & \dots & \dots & & \\ & \dots & \dots & \dots & \\ & & \dots & \dots & -\frac{\alpha\Delta t}{\Delta x^2} \\ & & & -\frac{\alpha\Delta t}{\Delta x^2} & 1+2\frac{\alpha\Delta t}{\Delta x^2} \end{bmatrix}^{-1} \tag{3.42}$$

and the corresponding eigenvalues are

$$\lambda_j = \left[1 + 4\frac{\alpha\Delta t}{\Delta x^2}\sin^2\left(\frac{j\pi}{2(np-1)}\right)\right]^{-1} \quad , \quad j = 1, ..., np-2 \tag{3.43}$$

For any choice of time and spatial step sizes the value of eigenvalues will be less or equal to one.

$$|\lambda_j| \leq 1 \quad , \quad \forall \Delta t, \Delta x \tag{3.44}$$

Consequently, the fully implicit scheme is unconditionally stable. This is clearly an advantage in comparison to the conditionally stable explicit FTCS scheme. However, for implicit schemes we have to solve the complete equation system, because unknown values at new time level in several nodes are connected. It is apparent that the system of equations to be solved has tridiagonal structure.

3.2.4 Crank-Nicolson Scheme (CNS)

Algebraic Scheme

A specific implicit algorithm with improved consistency order is the Crank-Nicolson scheme (CNS). Spatial derivatives are evaluated at old n-th as well as at new $(n+1)$-th time level, i.e. at intermediate $(n+1/2)$-th level. The CNS, which is also called a semi-implicit algorithm, is given by

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} - \frac{\alpha}{2}\frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2} - \frac{\alpha}{2}\frac{u_{j-1}^{n+1} - 2u_j^{n+1} + u_{j+1}^{n+1}}{\Delta x^2} = 0 \tag{3.45}$$

Rearranging the above formula we obtain an expression for the unknown values at new time level in terms of known values at old time level. Again the implicit three-point scheme produces a tridiagonal SAE.

$$\frac{\alpha\Delta t}{\Delta x^2}(-\frac{1}{2}u_{j-1}^{n+1} + u_j^{n+1} - \frac{1}{2}u_{j+1}^{n+1}) + u_j^{n+1} = u_j^n \tag{3.46}$$

Consistency - Applying the procedure based on TSE about node j at $n+1/2$-th level we found that the scheme is consistent with a truncation error of $0(\Delta t^2, \Delta x^2)$.

Stability - Stability analysis yields that the CNS is unconditionally stable. However, this scheme is on the limit of unconditionally stability. The CNS can produce oscillatory solutions for unfavourable time step and grid sizes.

Mitchell & Griffits (1980, pp 47-53) investigated the application of the CNS to the diffusion equation in detail. Additionally, they considered various boundary conditions. In connection with Dirichlet boundary conditions the CNS is unconditionally stable, whereas Neumann boundary conditions introduce eigenvalues equal to unity, which produces oscillatory solutions. For Cauchy boundary conditions additional restrictions are necessary to ensure stability.

3.2.5 Generalized Scheme

The above discussed discretization schemes can be generalized by writing

$$\frac{u_j^{n+1}-u_j^n}{\Delta t}-\alpha\left(1-\theta\,\frac{u_{j-1}^n-2u_j^n+u_{j+1}^n}{\Delta x^2}+\theta\,\frac{u_{j-1}^{n+1}-2u_j^{n+1}+u_{j+1}^{n+1}}{\Delta x^2}\right)=0 \tag{3.47}$$

where $0 \leq \theta \leq 1$ is a weighting (collocation) factor. The following table summarizes some properties of several specified schemes. Generalized three-level schemes and higher-order schemes are discussed e.g. by Fletcher (1990).

Collocation factor	Scheme	Consistency order	Stability condition
$\theta = 0$	explicit	$0(\Delta t, \Delta x^2)$	$\Delta t \leq \dfrac{\Delta x^2}{2\alpha(1-2\theta)}$
$\theta = 0$	improved FTCS	$0(\Delta t^2, \Delta x^4)$	
$0 < \theta < 0.5$	implicit	$0(\Delta t, \Delta x^2)$	
$\theta = 0.5$	CNS	$0(\Delta t^2, \Delta x^2)$	unconditionally stable
$0.5 < \theta < 1$	implicit	$0(\Delta t, \Delta x^2)$	
$\theta = 1$	fully implicit	$0(\Delta t, \Delta x^2)$	

3.2.6 Initial and Boundary Conditions

Initial Conditions

In general, the implementation of initial conditions does not cause any difficulty. Problems rise if initial and boundary conditions are not compatible, i.e. if the value at a boundary node specified by an initial condition is different from the value specified by the boundary condition. In this case a suitable strategy is to average the values for the first time step, and use the boundary condition for

subsequent time. Sometimes (e.g. for multi-level schemes), integration in time should be used to improve accuracy.

Dirichlet Boundary Conditions

Up to now we developed algebraic schemes for PDEs by approximation of the corresponding derivatives. We discussed the numerical properties (consistency, stability) of this approximation schemes resulting from the discretization process. Beside the approximation of derivatives, additional errors can be introduced from the implementation of boundary and initial conditions. Implementation of Dirichlet boundary conditions is trivial, because the value of unknown is explicitly given. The treatment of Neumann boundary conditions is more expensive.

Neumann Boundary Conditions

Use of centered differences for approximation of Neumann boundary conditions would require information from outside the domain. A simple way to represent the boundary condition is the use of one-sided differences.

$$\left[\frac{\partial u}{\partial x}\right]_1^{n+1} = g^{n+1} = \frac{u_2^{n+1} - u_1^{n+1}}{\Delta x} \tag{3.48}$$

The value of unknown at boundary can be calculated now by

$$u_2^{n+1} = u_1^{n+1} + g^{n+1}\Delta x \tag{3.49}$$

The problem here is that this boundary condition representation has a first-order truncation error. Therefore using this formula, the accuracy of higher-order schemes (e.g. FTCS scheme) will be disturbed. In particular for diffusion problems (parabolic PDE) lower-order accuracy at the boundary will affect the accuracy of the solution within the whole domain for all later time. Therefore, we should design a scheme for nodes with Neumann boundary conditions that has the same order of accuracy as the scheme for interior nodes.

To achieve that, we introduce a fictious node outside the domain at old time level. Now a second-order accurate formula in space can be applied to fit the boundary condition (see Chapter 3.1).

$$\left[\frac{\partial u}{\partial x}\right]_1^{n} = g^{n} = \frac{u_2^{n} - u_0^{n}}{2\Delta x} \tag{3.50}$$

Due to the temporary expansion of the computational domain we can apply an appropriate scheme for internal nodes (e.g. explicit FTCS) to obtain the value of the 'boundary' node at new time level.

$$u_1^{n+1} = u_1^n + \frac{\alpha\Delta t}{\Delta x^2}(u_0^n - 2u_1^n + u_2^n) \tag{3.51}$$

Now the above boundary condition representation can be used to eliminate the fictious value u_0^n again. Finally we obtain

$$u_1^{n+1} = -2\frac{\alpha\Delta t}{\Delta x}g^n + (1 - \frac{2\alpha\Delta t}{\Delta x^2})u_1^n - \frac{\alpha\Delta t}{\Delta x^2}u_2^n \tag{3.52}$$

and the truncation error will be of second order in space everywhere in the computational domain.

If an implicit scheme is being used for discretization of the PDE, the combination of boundary condition representation and the scheme for interior nodes yields.

$$(1 + \frac{2\alpha\Delta t}{\Delta x^2})u_1^{n+1} - \frac{2\alpha\Delta t}{\Delta x^2}u_2^{n+1} = u_1^n - \frac{2\alpha\Delta t}{\Delta x}g^n \tag{3.53}$$

The accuracy of Neumann boundary condition implementation is discussed in detail by Fletcher (1990, pp. 238-241).

3.3 Problems

Taylor Series Expansion

1 Develop the function $u(t, x)$ in x at nearby node x_i using the Taylor series expansion.

2 Use the Taylor series expansion for first order derivatives in time and space of function $u(i, x)$.

Diffusion Equation

3 Write the diffusion equation in one dimension.

4 Examine the consistency of the explicit FTCS scheme (3.17) for the diffusion equation (3.13). Determine the truncation error (3.20) by substituting the Taylor series expansion into the PDE.

5 Determine the stability bounds (Neumann criterion) of the explicit FTCS scheme (3.17) for the diffusion equation (3.13) using the matrix method.

6 Determine the stability bounds of the explicit FTCS scheme (3.17) for the diffusion equation (3.13) using the von Neumann method.

7 Determine the stability bounds of the semi-implicit Crank-Nicolson scheme (3.46) for the diffusion equation (3.13) using the von Neumann method.

8 Extend the explicit FTCS scheme (3.17) for the diffusion equation (3.13) in two-dimensions.

Part III

Prozessverständnis - Anwendungen

Kapitel 4

Prozessverständnis

Nachdem wir uns im ersten Teil der Vorlesung mit den mechanischen und numerischen Grundlagen herumgeschlagen haben, kommen wir nun zur Anwendung der Methodik. Wir beschäftigen uns mit drei Beispielanwendungen, die einen konkreten Bezug zu den Fachrichtungen Abfall- und Wasserwirtschaft sowie Hydrologie haben. Dabei geht es um Diffusions- und Strömungsprozesse in Fließgewässern und im Grundwasser.

4.1 Diffusionsgleichung - Explizite FDM

Wir implementieren nun ein finite Differenzen Verfahren (FDM) für die Diffusionsgleichung in unseren Funktions-Plotter. Zur Erinnerung, das FDM ist in Abschnitt 3.2 beschrieben. Die eindimensionale, quellenfreie Diffusionsgleichung ist eine parabolische partialle Differentialgleichung (PDE) und sieht wie folgt aus.

$$\frac{\partial u}{\partial t} - \alpha \frac{\partial^2 u}{\partial x^2} = 0 \tag{4.1}$$

Wir benutzen das explizite FTCS Schema (Abschn. 3.2.2) für die Diskretisierung der PDE. FTCS - steht für Forward Time / Centered Space. Die wichtigsten Teile, die wir im Folgenden benötigen, hier noch einmal zusammengefasst und übersetzt.

Eines der einfachsten finite Differenzen (FD) Ansätze für die Diffusionsgleichung ist Forward Time / Centered Space (FTCS) Schemata. Für die Diskretsierung der partiellen Differenzialgleichung (PDE) ersetzen wir die Zeitableitung durch folgende Vorwärtsdifferenz

$$\left[\frac{\partial u}{\partial t}\right]_j^n \approx \frac{u_j^{n+1} - u_j^n}{\Delta t} \tag{4.2}$$

und die räumliche Ableitung (2. Ordnung) durch das 3-Punkt zentrale Differenzenschema.

$$\left[\frac{\partial^2 u}{\partial x^2}\right]_j^n \approx \frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2} \tag{4.3}$$

Nach Einsetzen in die PDE (4.1) erhalten wir folgende algebraische Gleichung - das FTCS Schema.

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} - \alpha \frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2} = 0 \tag{4.4}$$

Die Diskretisierung bedeuted, das Original-Problem (d.h. die PDE für die exakte Lösung $u(t, x)$) wurde durch ein anderes Problem ersetzt, in dem es darum geht, diskrete (also einzelne) Werte u_j^n im Knoten j zum Zeitpunkt n zu finden. Nach dem Umstellen der Gleichung (4.4) bezüglich der unbekannten Größe u_j^{n+1} zum neuen Zeitpunkt $n + 1$ erhalten wir folgenden Ausdruck.

$$u_j^{n+1} = u_j^n + \frac{\alpha \Delta t}{\Delta x^2}(u_{j-1}^n - 2u_j^n + u_{j+1}^n) \tag{4.5}$$

wobei

$$\boxed{Ne = \frac{\alpha \Delta t}{\Delta x^2}} \tag{4.6}$$

die sogenannte Neumann-Zahl ist. In der oben genannten FD-Gleichung (4.6) stehen auf der linken Seite die unbekannten und auf der rechten Seite die bekannten Größen (d.h. zum alten Zeitschritt). Wenn wir alle unbekannten Größen (neue Zeit) in dieser Weise komplett durch bekannte Größen (alte Zeit) darstellen können, sprechen wir von einem expliziten Verfahren.

Schließlich können wir das FTCS Schema wie folgt schreiben.

$$u_j^{n+1} = u_j^n + Ne(u_{j-1}^n - 2u_j^n + u_{j+1}^n) \tag{4.7}$$

Machen wir uns nun an die Programmierung. Das heisst, erstmal überlegen, was wir alles brauchen.

1. Datenstrukturen: u_j^{n+1}, wir benötigen Vektorn für die Knotenwerte
2. Parameter: Ne, wir haben eigentlich nur einen Parameter. Dahinter verbergen sich aber $\alpha, \Delta t, \Delta x$

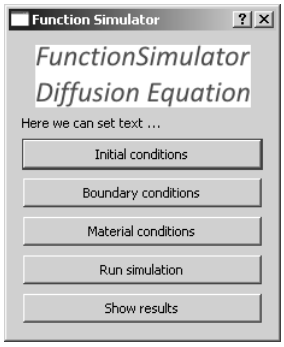

Abbildung 4.1: Funktionssimulator

4.1.1 Berechnungsfunktion

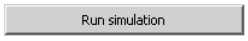

Abbildung 4.2: Berechnungsfunktion

Eigentlich ganz einfach, z.B. erster Versuch für die Daten(strukturen).

```
double u_new, u_old, alpha, dt, dx, u_l_old, u_r_old;
u_new = u_old + alpha * dt / dx*dx * (u_l_old - 2*u_old + u_r_old);
```

Wie müssen aber bedenken, dass u_j Knotenwerte sind, wir brauchen also Vektoren. Zweiter Versuch.

```
vector<double>u_new, u_old, u_old_l, u_old_r;
double alpha, dt, dx;
u_new = u_old + alpha * dt / dx*dx * (u_old_l - 2*u_old + u_old_r);
```

Wenn wir jetzt das Programm schon mal laufen lassen, kracht es erstmal, warum?

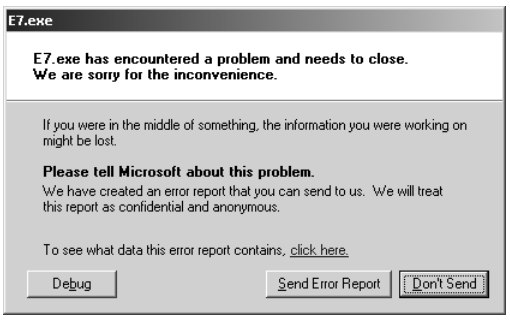

Abbildung 4.3: Programmabsturz

Wir haben ja noch keinen Speicher für die Vektoren reserviert. Hierfür gibt es verschiedene Möglichkeiten (siehe Skript Hydroinformatik I, Abschn. 8), z.B.

```
vector<double>u_new(10), u_old(10), u_old_l(10), u_old_r(10);
vector<double>u_new; u_new.resize(10);
```

Nun möchten wir am liebsten die Ergebnisse gleich plotten und ändern die Plotter-Funktion wie folgt.

```
void Dialog::on_pushButtonSHO_clicked()
{...
  for (int i = 0; i < numPoints; ++i)
  {
    x = dx*i;
    points0.append(QPointF(x,u_new[i]));
  }
...}
```

... stossen aber gleich auf das nächste Problem: u_new ist in der Plotterfunktion gar nicht bekannt. Wenn wir Datentypen in verschiedenen Funktionen verwenden wollen, müssen wir diese als Klassen-Variable im Header-File vereinbaren (siehe Skript Hydroinformatik I, Abschn. 4), nicht den `<vector>`Inklude vergessen.

```
class Dialog : public QDialog
{...
  private:
  double dx;
  vector<double>u_new;
...}
```

Gleich die Speicherverwaltung für die Klassenvariable richtig machen, richtig, diese gehört in den Konstruktor.

```
Dialog::Dialog(QWidget *parent) : QDialog(parent)
{...
  u_new.resize(10);
...}
```

... nächster Versuch, Programm läuft jetzt, aber produziert keine Ergebnisse, alle Werte sind 0. Wir müssen uns ja noch um die Anfangs- und Randbedingungen für die Gleichung kümmern. Wir sehen, dass wir schnell durcheinander kommen können bei den zu erledigenden Schritten. Das kann uns der Dialog helfen, in dem wir die Schaltflächen zunächst deaktivieren und nur das einschalten, was als Nächstes gebraucht wird (Abb. 4.4)

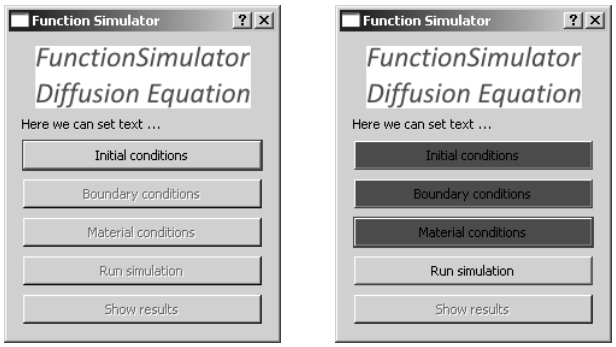

Abbildung 4.4: Deaktivieren und Gestalten von Schaltflächen

Die nachfolgenden Funktionen sind sehr nützlich für die Arbeit mit Schaltflächen, Info, Ein- und Ausschalten, Einfärben.

```
pushButtonBC->setToolTip("This is to specify boundary conditions for the PDE"); // Info
pushButtonBC->setEnabled(false); // Deaktivieren
pushButtonBC->setStyleSheet("background-color: green"); // Einfärben
```

Gehen wir also mal durch alle Funktionen.

4.1.2 Anfangsbedingungen

Hier werden alle Werte u^n das alten Zeitschritts auf Null gesetzt.

```
void Dialog::on_pushButtonIC_clicked()
{...
  for(int i=0;i<n;i++)
  {
    u_old[i] = 0.;
  }
...}
```

4.1.3 Randbedingungen

Hier werden die Randbedingungen links ($i = 0$) und rechts ($i = 10$) auf Eins gesetzt.

```
void Dialog::on_pushButtonBC_clicked()
{
  double u_bc_l = 1.;
  double u_bc_r = 1.;
  u_new[0] = u_old[0] = u_bc_l;
  u_new[10] = u_old[10] = u_bc_r;
}
```

4.1.4 Materialeigenschaften

Zur Lösung der finite Differenzengleichung (4.7) müssen wir nur einen Parameter berechnen, $Ne = \alpha dt/dx^2$, die Neumann-Zahl

```
void Dialog::on_pushButtonMAT_clicked()
{
  double alpha = 1.;         // Diffusionskoeffizient
  dt = 1.;                   // Zeitschritt
  dx = 0.1;                  // räumliche Auflösung
  dt = 0.5 * dx*dx / alpha;  // Zeitschrittsteuerung
  Ne = alpha * dt / (dx*dx); // Neumann-Zahl
}
```

Und schon kann es losgehen. Das Ergebnis des ersten Zeitschritts ist in Abb. 4.5 zu sehen.

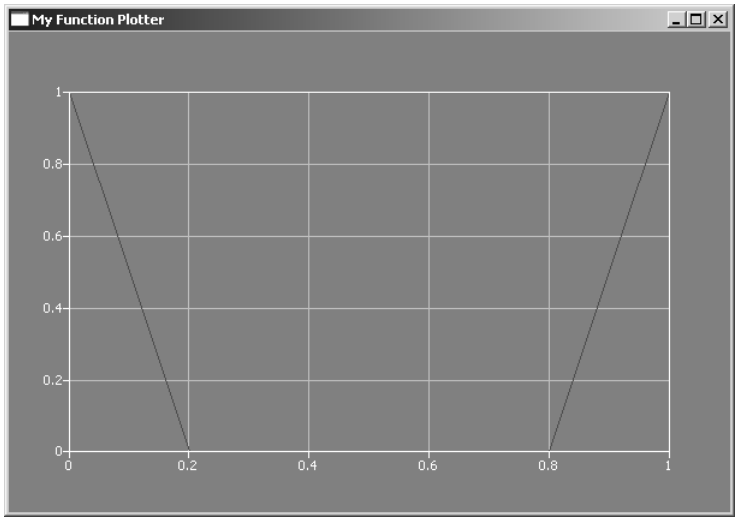

Abbildung 4.5: Berechnung, erster Zeitschritt

Den C++ Quelltext (inklusive Visualisierung) finden wir wie gewohnt im Übungsteil der WebSite unter E4.1a.

4.1.5 Zeitschleife

Jetzt würden wir gerne sehen, wie sich das Diffusionsprofil mit der Zeit ändert. Dafür müssen wir eine Zeitschleife einbauen. Wir legen die Zeitschleife, die von $t = 0$ bis nt läuft, um die Schleife entlang des Stabes, die von $i = 0$ bis $n - 1$ läuft, herum. In jeden Zeitschritt werden die Werte zum Plotten im QVector<QPointF> points0 abgelegt. Das Entscheidende passiert ganz unten, das Umspeichern der Werte $u^n[i] = u^{n+1}[i]$. Damit werden gerade berechneten Werte $u^{n+1}[i]$ die Anfangsbedingungen für den nächsten Zeitschritt.

```
void Dialog::RunTimeLoop()
{..
  for(int t=0;t<nt;t++) // Zeitschleife
  {
    QVector<QPointF> points0;
    points0.append(QPointF(0,u_new[0]));
    for(int i=1;i<n-1;i++) // Schleife entlang des Stabes
    {
      u_new[i] = u_old[i] + Ne * (u_old[i-1] - 2*u_old[i] + u_old[i+1]);
      x = dx*i;
      points0.append(QPointF(x,u_new[i]));
    }
    plotter->setCurveData(t, points0);
```

```
    plotter->show();
    // Daten speichern
    for(int i=1;i<n-1;i++)
    {
      u_old[i] = u_new[i];
    }
  }
...}
```

Das Ergebnis ist in Abb. 4.6 zu sehen.

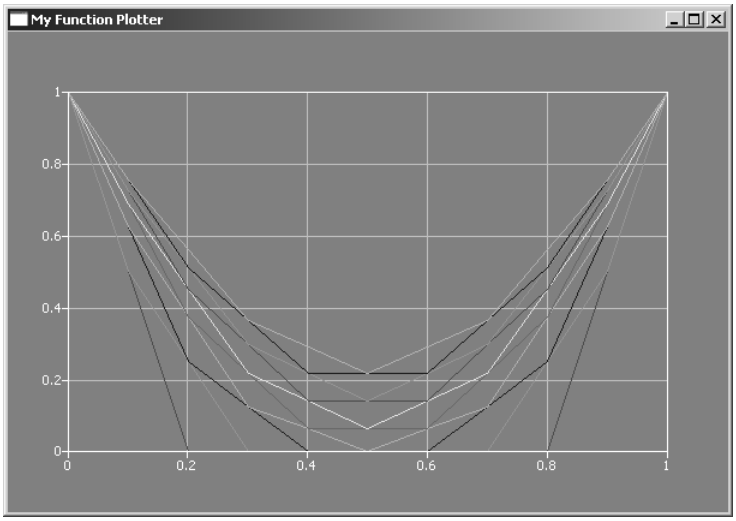

Abbildung 4.6: Zeitliche Entwicklung des Diffusionsprofils

Den C++ Quelltext (inklusive Visualisierung) finden wir im Übungsteil der WebSite unter E4.1b.

4.2 Diffusionsgleichung - Implizite FDM

Wie wir im Numerik-Teil der Vorlesung gelernt haben, gibt es neben den expliziten Verfahren auch sogenannte implizite. Implizite Berechnungsverfahren sind eigentlich der Standardfall. Explizite Verfahren lassen meistens nur für sehr einfache Problemstellungen entwickeln. Bei der Anwendung von impliziten Verfahren müssen wir uns mit der Lösung von Gleichungssystemen beschäftigen.

Die englische Beschreibung des impliziten FDM Verfahrens zur Lösung der Diffusionsgleichung finden sie im Abschn. 3.2.3. Hier noch einmal das Wichtigste in Kürze.

Das implizite, zentrale Differenzenschema für den Diffusionsterm (räumliche Ableitung zweiter Ordnung) zum neuen Zeitpunkt n^1 term) lautet folgendermaßen.

$$\left[\frac{\partial^2 u}{\partial x^2}\right]_j^{n+1} \approx \frac{u_{j-1}^{n+1} - 2u_j^{n+1} + u_{j+1}^{n+1}}{\Delta x^2} \tag{4.8}$$

Unter weiterer Verwendung von Vorwärtsdifferenzen für die Zeitableitung ergibt sich folgendes algebraisches Schema für das implizite FD-Verfahren der eindimensionalen, quellenfreien Diffusionsgleichung.

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} - \alpha \frac{u_{j-1}^{n+1} - 2u_j^{n+1} + u_{j+1}^{n+1}}{\Delta x^2} = 0 \tag{4.9}$$

Nach Umstellen der Gleichung bezüglich der unbekannten Terme (neue Zeit) auf der linken Seiten und der bekannten Terme (alte Zeit) auf der rechten Seite erhält man folgenden algebraischen Ausdruck.

$$\frac{\alpha \Delta t}{\Delta x^2} \left(-u_{j-1}^{n+1} + 2u_j^{n+1} - u_{j+1}^{n+1}\right) + u_j^{n+1} = u_j^n \tag{4.10}$$

Wir sehen eine Verküpfung von unbekannten Größen auf der linken Seite, daher - wie bereits angekündigt, müssen wir uns nun mit dem Lösen von Gleichungssystemen beschäftigen.

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{4.11}$$

Dafür benötigen wir entsprechende Datenstrukturen für die Systemmatrix $\mathbf{A}$, den Lösungsvektor $\mathbf{x}$ und den Rechte-Seite-Vektor $\mathbf{b}$. Dies erledigen wir natürlich im Konstruktor unseres Dialogs, entsprechende Speicherfreigabe im Destruktor.

```
Dialog::Dialog(QWidget *parent) : QDialog(parent)
{
  matrix = new double[n*n];
  vecb = new double[n];
  vecx = new double[n];
}

Dialog::~Dialog()
{
  delete [] matrix;
  delete [] vecb;
  delete [] vecx;
}
```

Eigentlich bleibt alles beim Altem, ausser zwei neue Funktionen zum Aufstellen und zum Lösen des Gleichungssystems, die es aber in sich haben.

```
AssembleEquationSystem();
Gauss(matrix,vecb,vecx,n);
```

Aus der impliziten FDM Gleichung (4.10) ergibt sich folgende Struktur des zu lösenden Gleichungssystems

$$\underbrace{\begin{bmatrix} 1+2\frac{\alpha\Delta t}{\Delta x^2} & -\frac{\alpha\Delta t}{\Delta x^2} & & & \\ -\frac{\alpha\Delta t}{\Delta x^2} & \dots & \dots & & \\ & \dots & \dots & \dots & \\ & & \dots & \dots & -\frac{\alpha\Delta t}{\Delta x^2} \\ & & & -\frac{\alpha\Delta t}{\Delta x^2} & 1+2\frac{\alpha\Delta t}{\Delta x^2} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ \dots \\ u_{n-1} \\ u_n \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} b_0 \\ b_1 \\ \dots \\ b_{n-1} \\ b_n \end{bmatrix}}_{\mathbf{b}} \tag{4.12}$$

Trauen wir uns mal ran an die Assemblierfunktion, sie besteht aus zwei Teilen:

- Einträge in die Matrix gemäß Gleichung (4.10)
- Behandeln der Randbedingungen

In beiden Fällen haben wir zwei Schleifen über die Zeilen und über die Spalten. Der Trick besteht darin, durch die Abfrage `if(i==j)` zunächst die Hauptdiagonale herauszufischen. Die Nebendiagonalen finden wir durch die Abfrage `if(abs((i-j))==1)`, alles was eine Position links oder rechts der Hauptdiagonale ist.

```
void Dialog::AssembleEquationSystem()
{
  int i,j;
  // Matrix entries
  for(i=0;i<n;i++)
  {
    vecb[i] = u_old[i]; // RHS Vektor
    for(j=0;j<n;j++)
    {
      matrix[i*n+j] = 0.0;
      if(i==j) // Hauptdiagonale
        matrix[i*n+j] = 1. + 2.*Ne;
      else if(abs((i-j))==1) // Nebendiagonalen
        matrix[i*n+j] = - Ne;
    }
  }
...}
```

Das Behandeln von Randbedingungen heisst eigentlich die ensprechenden Zeilen und Spalten aus dem Gleichungssystem zu nehmen. Eine Randbedingung ist ein

vorgegebener Wert, hier muss eigentlich nichts gelöst werden, da fest vorgegeben. Wir lösen das Problem, indem wir Matrix manipulieren, wir setzen den Wert auf der Hauptiagonale gleich Eins, setzen alle anderen Werte in der entsprechenden Zeile gleich Null, dann entspricht der Wert der rechten Seite dem Wert der Randbedingung.

$$[1\,0\,0\,...\,0] \begin{bmatrix} u_0 \\ u_1 \\ ... \\ u_n \end{bmatrix} = \begin{bmatrix} u_0 \\ 0 \\ ... \\ 0 \end{bmatrix} \tag{4.13}$$

```
void Dialog::AssembleEquationSystem()
{...
  // Treat boundary conditions
  for(i=0;i<n;i++)
    for(j=0;j<n;j++)
    {
      if(i==0||i==n-1)
        matrix[i*n+j] = 0.0;
    }
  for(i=0;i<n;i++)
  {
    if(i!=0&&i!=n-1)
      continue;
    for(j=0;j<n;j++)
    {
       if(i==j)
         matrix[i*n+j] = 1.0;
       else
         matrix[i*n+j] = 0.0;
    }
  }
}
```

Der Löser `Gauss()` funktioniert nach dem Gaußschen Eliminierungsalgorithmus. Anhand der Parameterliste sehen sie, dass es sich noch um eine C-Funktion handeln ... Hauptsache sie funktioniert ... nein im Ernst, wir sollten schon wissen, was wir tun. Wir schauen uns ein ganz einfaches Gleichungssystem an, um das Prinzip zu erkären

$$a_{11}u_1 + a_{12}u_2 = b_1 \tag{4.14}$$

$$a_{21}u_1 + a_{22}u_2 = b_2 \tag{4.15}$$

Wenn wir nun die zweite Gleichung mit a_{11}/a_{21} multiplizieren

$$a_{21}\frac{a_{11}}{a_{21}}u_1 + a_{22}\frac{a_{11}}{a_{21}}u_2 = \frac{a_{11}}{a_{21}}b_2 \tag{4.16}$$

und dann von der ersten Gleichung abziehen, erhalten wir

$$\left(\frac{a_{22}a_{11}}{a_{21}} - a_{12}\right) u_2 = \frac{a_{11}}{a_{21}} b_2 - b_1 \tag{4.17}$$

also schließlich

$$u_2 = \frac{\frac{a_{11}}{a_{21}} b_2 - b_1}{\frac{a_{22}a_{11}}{a_{21}} - a_{12}} \tag{4.18}$$

Damit können wir u_2 berechnen, dann die die erste Gleichung (4.14) einsetzen und somit u_1 ermitteln. Dieses Verfahren lässt sich auf beliebig große lineare Gleichungssysteme anwenden, sie sehen aber, dass der Rechenaufwand erheblich ist. Wir benötigen genau 5 FLOPs (floating point operations).

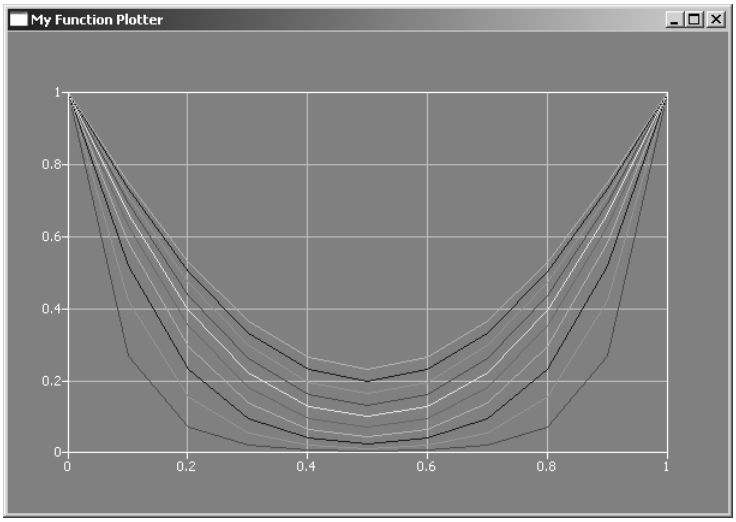

Abbildung 4.7: Zeitliche Entwicklung des Diffusionsprofils - implizites Verfahren

Den C++ Quelltext (inklusive Visualisierung) finden wir im Übungsteil der WebSite unter E4.2.

Was fällt ihnen auf, wenn sie die Abbildungen 4.6 und 4.7, die Lösungen der expliziten und impliziten Gleichungen vergleichen?

4.3 Gerinnehydraulik

Für die Beschreibung von Strömungsprozessen in Flüssen (d.h. Gerinne, Abb. 4.8) gibt es verschiedene Möglichkeiten basierend auf den sogenannten Bernoulli oder Saint-Venant Gleichungen sowie von Energiebetrachtungen. Wir verwenden letztere für die Implementierung in C++.

Abbildung 4.8: (Quelle: http://www.ifh.uni-karlsruhe.de/lehre/gerinnehydraulik)

4.3.1 Bernoulli-Gleichung

Die Bernoulli-Gleichung beschreibt die Energieerhaltung bei der Strömung von Flüssigkeiten. Physikalisch gesehen bedeutet Energie, die Fähigkeit Arbeit zu verrichten. In unserem einfachen Strömungsproblem geht es um Lageenergie, Bewegungsenergie (kinetische) und sogenannte 'Energieverluste'. Die Bernoulli-Gleichung lautet.

$$z_{i+1} + \frac{p_{i+1}}{\rho g} + \frac{v_{i+1}^2}{2g} = z_i + \frac{p_i}{\rho g} + \frac{v_i^2}{2g} + h_v \tag{4.19}$$

Dabei sind z_i und z_{i+1} die Sohlhöhen an den Punkten x_i und x_{i+1} des Gerinnes (Abb. 4.9), p_i, p_{i+1} und v_i, v_{i+1} die entsprechenden Drücke und Geschwindigkeiten in den Gerinnepositionen. h_v sind streckenabhängige Verluste aufgrund der Reibung des fliessenden Wasser auf der Gerinnesohle. Nach dem Gesetz von Darcy-Weisbach lassen sich die streckenabhängigen Verluste wie folgt beschreiben.

$$h_v = \lambda \frac{x_{i+1} - x_i}{d_{hy}} \frac{v^2}{2g} \tag{4.20}$$

dabei sind λ der Reibungsbeiwert und d_{hy} der hydraulische Durchmesser.

$$d_{hy} = 4r_{hy} = 4\frac{A}{U} \tag{4.21}$$

Da auf der Oberfläche des Gewässers atmosphärischer Druck herrscht (freie Oberfläche) gilt laut Hydrostatik

$$p = \rho g h \tag{4.22}$$

Damit lässt sich die Bernoulli-Gleichung folgendermaßen umschreiben.

$$z_{i+1} + h_{i+1} + \frac{v_{i+1}^2}{2g} = z_i + h_i + \frac{v_i^2}{2g} + h_v \tag{4.23}$$

Der Term aus der Bernoulli-Gleichung (4.23)

$$E = h + \frac{v^2}{2g} \tag{4.24}$$

wird auch spezifische Energiehöhe genannt.

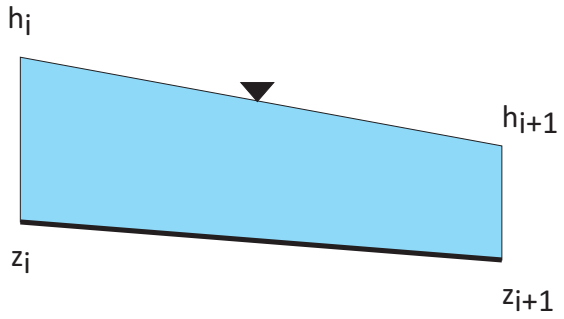

Abbildung 4.9: Längsprofil eines hydraulischen Gerinnes

Für ein Rechteckgerinne (Abb. 4.10) gilt für den Durchfluss

$$Q = A\,v = bh\,v \tag{4.25}$$

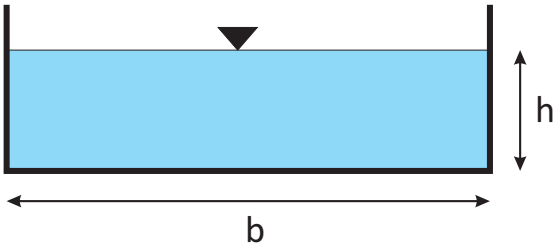

Abbildung 4.10: Rechteckgerinne

Die Bernoulli-Gleichung für ein Rechteckgerinne ist somit.

$$z_{i+1} + h_{i+1} + \frac{Q_{i+1}^2}{2g\,b^2 h_{i+1}^2} = z_i + h_i + \frac{Q_i^2}{2g\,b^2 h_i^2} + h_v \tag{4.26}$$

Die Gleichung (4.26) ist nichtlinear bezüglich des Wasserspiegels h selbst bei stationären Bedingungen, d.h. bei konstantem Abfluss $Q = \text{const}$.

Bei sogenanntem Normalabfluß lässt sich das Sohlgefälle aus dem Darcy-Weisbach Gesetz (4.21) bestimmen.

$$I_S = \frac{h_v}{\Delta x} = \lambda \frac{v^2}{2 g d_{hy}} \tag{4.27}$$

Aufgabe: Berechnen sie das Sohlgefälle I_S für ein Rechteckgerinne mit einer Sohlbreite von $b = 1.5$ m, einer Wassertiefe von $h = 1.2$ m, einer Strömungsgeschwindigkeit von $v = 0.7$ m/s. Der Reibungsbeiwert λ beträgt 0.016.

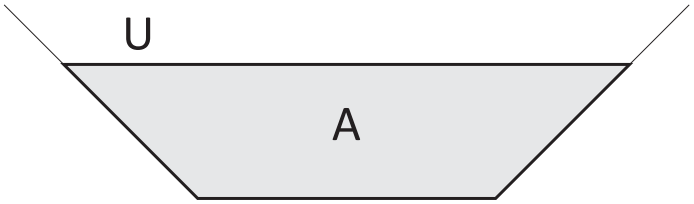

Abbildung 4.11: Trapezgerinne

4.3.2 Saint-Venant-Gleichungen

Eine weitere (und allgemeinere) Möglichkeit zur Beschreibung von Fließprozessen in Oberflächengewässern sind die Navier-Stokes-Gleichnungen (NSE) (siehe Abschn. 1.3.4). Oft können vertikale Prozesse vernachlässigt werden, z.B. wenn die Gewässertiefe relativ klein zur horizontalen Gewässerausdehnung ist. Dann können die NSE vertikal integriert werden, diese heissen dann wie oben schon erwähnt - Saint-Venant-Gleichungen.

Das Eulersche Prinzip (siehe Abschn. 1.1.4) für einen Fluß lässt sich z.B. wie folgt formulieren. Dabei handelt es sich um Vereinfachungen der Navier-Stokes Gleichungen (1.50) indem die 3D Gleichungen über die Querschnittsfläche $A(x, h)$ des Gerinnes integriert wird. Das heisst uns interessiert eigentlich nur die Fließgeschwindigkeit $v = Q/A$ in Flußrichtung.

$$\begin{aligned} \frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} &= 0 \\ \frac{\partial}{\partial t}\frac{Q}{A} + \frac{\partial}{\partial x}\left(\frac{Q^2}{2A^2} + g h(x, A)\right) &= g(S_o - S_f) \end{aligned} \tag{4.28}$$

dabei sind t die Zeit, x die räumliche Auflösung entlang des Flußverlaufs, Q die Volumenstromrate, g die Erdbeschleunigung, h der Flußwasserspiegel und

S_* Fließraten. Wir schreiben nun die Gleichungen (4.28) in Geschwindigkeiten $v = Q/A$ ($h - v$ Formulierung) um.

$$\begin{aligned} \frac{\partial A(x,h)}{\partial t} + \frac{\partial}{\partial x}\left(A(x,h)v\right) &= 0 \\ \frac{\partial v}{\partial t} + \frac{\partial}{\partial x}\left(\frac{v^2}{2} + gh(x,A)\right) &= g(S_o - S_f) \end{aligned} \tag{4.29}$$

Unter stationären Bedingungen lassen sich die Gleichungen (4.29) weiter vereinfachen.

$$\begin{aligned} \frac{d}{dx}\left(A(x,h)v\right) &= 0 \\ \frac{d}{dx}\left(\frac{v^2}{2} + gh(x,A)\right) &= g(S_o - S_f) \end{aligned} \tag{4.30}$$

oder unter Verwendung der Primärvariablen A und Q

$$\frac{d}{dx}\left(\frac{Q(h)^2}{2A(h)^2} + gh(x,A)\right) = g(S_o - S_f) \tag{4.31}$$

Die Saint-Venant Gleichungen, wie man sieht, sind nichtlinear. Für die Lösung nichtlinearer Probleme benötigen wie spezielle Methoden. Unsere Gleichungslöser können immer nur lineare oder linearisierte Gleichungen lösen.

4.3.3 Energiebetrachtung

Die dritte Möglichkeit zur Berechnung von Wasserspiegelprofilen ist die Energiebetrachtung nach [10]. Diese Arbeit enhält auch den Quelltext für ein Berechnungsprogramm in FORTRAN. Die Methode von [10] ist anwendbar auf beliebige Trapezgerinne. Anhand der Abb. 4.12 können die hydraulischen Parameter von Gerinnen erklärt werden (Tab. 4.1).

Symbol	Parameter	Englisch	Einheit
h	Wasserspiegel	water surface elevation	m
i	Sohlenspiegel	channel datum	m
y	Wassertiefe	water depth	m
Z_L, Z_R	linke / rechte Böschung	left / right channel slope	m
Q	Durchfluss	discharge	$\mathrm{m^3 s^{-1}}$
A	Querschnittsfläche	channel cross-section	$\mathrm{m^2}$

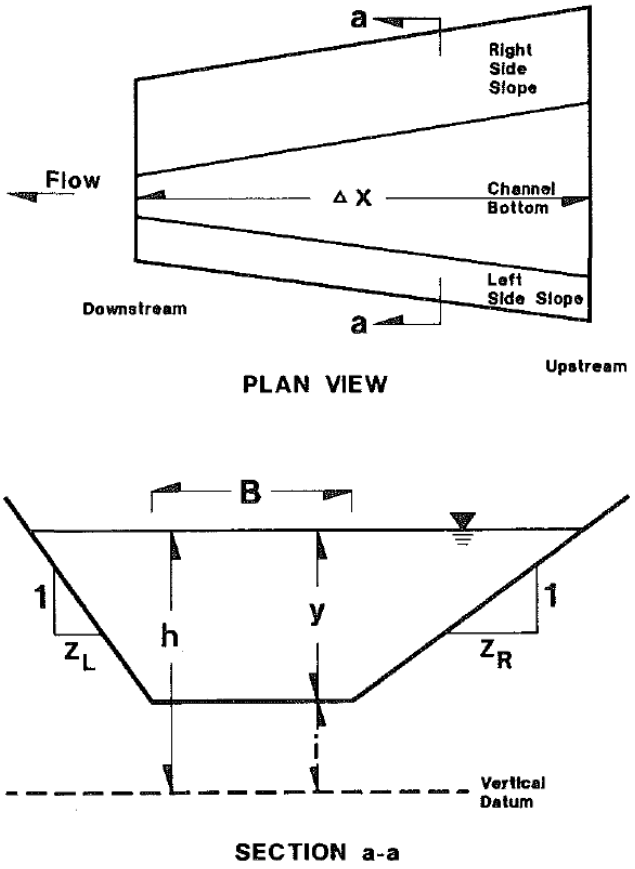

Abbildung 4.12: Trapezgerinne [10]

Symbol	Parameter	Englisch	Einheit
B	Breite	channel width	m^2
P	benetzter Umfang	wetted perimeter	m
R	hydraulische Radius	hydraulic radius	m
U	stromaufwärts	upstream	-
D	stromabwärts	downstream	-
Δx	Abstand zwischen U und D	distance between U and D	m
h_v	kinetische Energie	velocity head	m
h_t	Reibungsverluste	friction head	m
S_o	Sohlgefälle	bed slope	-
S_f	Streckenverlust	friction slope	-

Tabelle 4.1: Gerinne-Parameter

Zur Berechnung des Wasserprofils bei gegebenen Durchfluss muss der Wasserspiegel an einem Ende des Kanals bekannt sein.

- Die Wassertiefe ist.

$$y = h - i \tag{4.32}$$

- Die spezifische Energiehöhe ist.

$$E = i + y + \frac{v^2}{2g} = h + \frac{v^2}{2g} \tag{4.33}$$

- Die trapezoide Querschnittsfläche des Gerinnes kann folgendermaßen berechnet werden (Abb. 4.12).

$$A = y \left[B + y \frac{Z_L + Z_R}{2} \right] \tag{4.34}$$

- Der benetzte Umfang des Gerinnes ist.

$$P = B + y \left[\sqrt{1 + Z_L^2} + \sqrt{1 + Z_R^2} \right] \tag{4.35}$$

- Der hydraulische Radius ergibt sich dann als.

$$R = \frac{A}{P} \tag{4.36}$$

- Letztlich benötigen wir noch den Zusammenhang von Fließgeschwindigkeit und Durchfluss.

$$V = \frac{Q}{A} = \frac{Q}{y \left[B + y \frac{Z_L + Z_R}{2} \right]} \tag{4.37}$$

Die Streckenverluste lassen sich nach Manning wie folgt beschreiben

$$S_f = \left(\frac{Q}{AR^{2/3}} \right)^2 \tag{4.38}$$

$$S_f = Q^2 A^{-2} R^{-4/3} \tag{4.39}$$

mit

$$R = \frac{y(B + yC_4)}{B + yC_5} \tag{4.40}$$

$$C_4 = \frac{Z_L + Z_R}{2} \tag{4.41}$$

$$C_5 = \sqrt{1 + Z_L^2} + \sqrt{1 + Z_R^2} \tag{4.42}$$

$$S_f = Q^2(y(B + C_4 y))^{-2} \left(\frac{y(B + yC_4)}{B + yC_5} \right)^{-4/3} \tag{4.43}$$

$$S_f = Q^2(By + C_4 y)^{-2}(By + C_4 y^2)^{-4/3}(B + C_5 y)^{4/3} \tag{4.44}$$

$$S_f = Q^2(By + C_4 y^2)^{-10/3}(B + C_5 y)^{4/3} \tag{4.45}$$

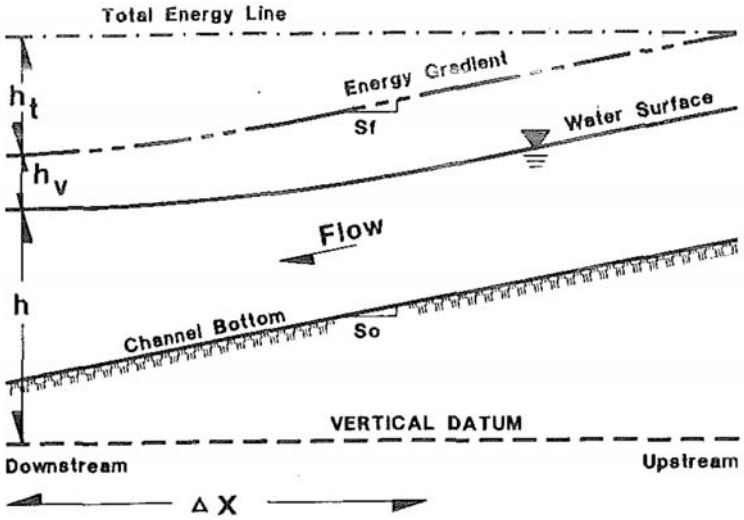

Abbildung 4.13: Fließprofil im Gerinne [10]

Auch die Gleichung (4.50) ist nicht-linear, so dass wir zu deren Lösung ein entsprechendes Verfahren, wie das Newton-Verfahren benötigen.

4.3.4 Newton-Verfahren zur Lösung nichtlinearer Gleichungen

Das Newton-Verfahren zur Lösung nichtlinearer Gleichungen haben wir schon in Abschn. 2.3.2.2 kennengelernt. Hier noch mal die Übersetzung des Wichtigsten ins Deutsche.

Das Newton-Verfahren wird oft angewendet, da die Genauigkeit und die Konvergenzgeschwindigkeit besser als andere Methoden (z.B. Fixpunkt-Iteration) sind. Die Newton-Methode ist ein sog. Verfahren höherer Ordnung. Dafür müssen wir natürlich einen Preis zahlen, die mathematische Vorbereitung zur Lösung ist aufwendiger (wie wir später sehen werden). Wir starten mit der Gleichung (2.5) aus dem Theorie-Kapitel.

$$\mathbf{R}(\mathbf{x}) = \mathbf{A}(\mathbf{x})\,\mathbf{x} - \mathbf{b}(\mathbf{x}) = 0 \tag{4.46}$$

Wenn das Residuum $\mathbf{R}(\mathbf{x})$ eine stetige Funktion ist, können wir folgende Taylor-Reihenentwicklung bezüglich der gesuchten Feldfunktion $\mathbf{x}_k$ schreiben.

$$\mathbf{R}_{k+1} = \mathbf{R}_k + \left[\frac{\partial \mathbf{R}}{\partial \mathbf{x}}\right]_k \Delta\mathbf{x}_{k+1} + 0(\Delta\mathbf{x}_{k+1}^2) \tag{4.47}$$

Die Terme zweiter und höherer Ordnung vernachlässigen wir (lineare Näherung). Der Term $\partial\mathbf{R}/\partial\mathbf{x}$ entspricht dann der Steigung der Tangente der Funktion $\mathbf{R}$ und wird Jacobian $\mathbf{J}$ genannt. Die Idee des Newton-Verfahrens ist sukzessiv Lösungen zu finden, welche das Funktional immer besser erfüllen. Das Ziel ist also $\mathbf{R}_{k+1} = 0$. Dann können wir schreiben (4.47).

$$\Delta\mathbf{x}_{k+1} = -\mathbf{J}_k^{-1}\,\mathbf{R}_k \tag{4.48}$$

Dabei müssen wir den Jacobian invertieren, also die Inverse der dazugehörigen Matrix bestimmen. Wir werden sehen, dass dies einen gehörigen Rechenaufwand erfordert. Die Lösungvorschrift ist nun

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{J}_k^{-1}\,\mathbf{R}_k \tag{4.49}$$

Mit einer Startlösung können wir nun sukzessive neue Lösungen $\mathbf{x}_{k+1}$ berechnen, wenn wir die Gleichungen (4.48) und (4.49) verwenden (Fig. 4.18). Dabei muss der Jacobian eigentlich ständig neu (bei jedem Iterationsschritt) berechnet und inverviert werden, was einen gehörigen Aufwand bedeutet. Daher gibt es ein sogenanntes Newton-Raphson-Verfahren, das den Jacobian nur einmal am Anfang berechnet $\mathbf{J}_0$ und mit diesem die Lösungsvorschrift abarbeitet. Dies geht natürlich auf Kosten der Genauigkeit, was letztlich aber nur bedeutet, dass mehr Iterationsschritte notwendig sind. Alternativ kann der Jacobian auch ab und zu aufgefrischt werden, wenn die Genauigkeit zu stark nachlässt. Man muss also einen Kompromiss zwischen Genauigkeit und Aufwand finden. In der Abb. 4.15 ist das Newton-Raphson-Verfahren grafisch dargestellt. Im Unterschied zum echten Newton-Verfahren haben die Jacobi-Tangenten hier eine konstante Steigung. Wie bereits erwähnt ist die Konvergenzgeschwindigkeit des Newton-Verfahrens zweiter Ordnung.

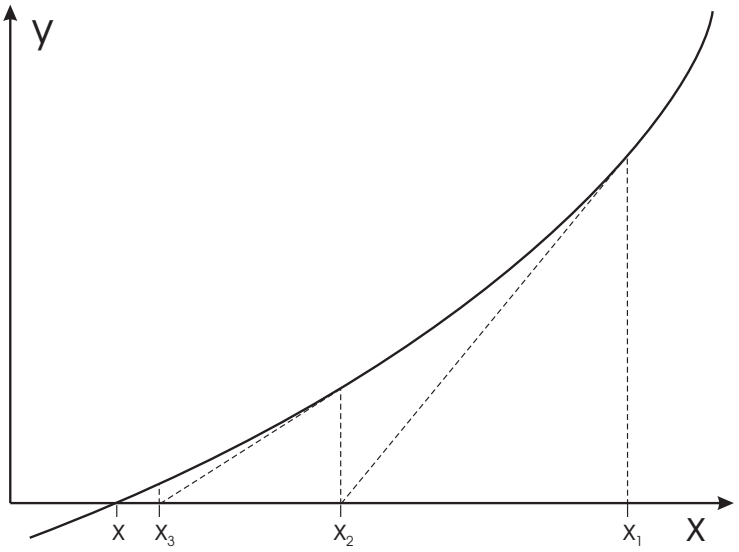

Abbildung 4.14: Graphical illustration of the Newton-Raphson iteration method

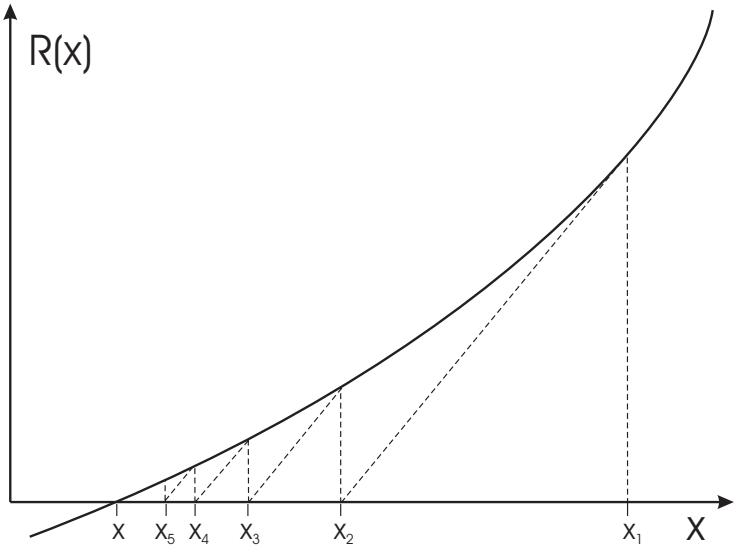

Abbildung 4.15: Graphical illustration of the modified Newton-Raphson iteration method

4.3.5 Numerisches Verfahren

Wir können nun folgendes Funktional konstruieren (Abb. 4.13)

$$f(h) = \left(h + \frac{v^2}{2g}\right)|_D - \left(h + \frac{v^2}{2g}\right)|_U + \frac{\Delta x}{2}(S_{f,U} + S_{f,D}) \tag{4.50}$$

das physikalisch gesehen, der Bernoulli-Gleichung (Energieerhaltung) entspricht.

Das Newtonverfahren zur Bestimmung der Wasserspiegelhöhe für Gerinne sieht nun folgendermaßen aus.

$$h_{k+1} = h_k + \frac{f(h_k)}{f'(h_k)} \tag{4.51}$$

Für die Konstruktion des Newton-Verfahrens zur Lösung der Gleichung (4.50) benötigen wir die Ableitung des Funktionals $f(h)$. Die Ableitung der spezifischen Energie nach der Wasserspiegelhöhe ist für den Fall des Normalabflusses (Q=const).

$$\frac{d}{dh}\left(h + \frac{v^2}{2g}\right) = \frac{d}{dh}\left(h + \frac{Q^2}{2gA^2}\right) = 1 - \frac{Q^2}{2gA^3}\frac{dA}{dh} \tag{4.52}$$

$$\frac{dA}{dh} = \frac{d}{dh}\left(y(B + C_4 y)\right) = B + C_4 y + yC_4 = B + 2C_4 y \tag{4.53}$$

Bleibt noch die Differenzierung der Streckenverluste

$$\frac{dS_f}{dh} = S'_f = \frac{d}{dh}\left(\frac{Q}{AR^{2/3}}\right)^2 \tag{4.54}$$

$$\begin{aligned} S'_f &= \left[Q^2(By + C_4y^2)^{10/3}\frac{4}{3}(B + C_5 y)^{1/3}C_5\right] \\ &+ \left[(B + yC_5)^{4/3}\frac{-10Q^2}{3}(By + C_4y^2)^{13/3}(B + 2C_4y)\right] \end{aligned} \tag{4.55}$$

$$\begin{aligned} S'_f &= \frac{4}{3}Q^2C_5(By + C_4y^2)^{-10/3}(B + C_5y)^{1/3} \\ &- \frac{10}{3}Q^2(B + 2C_4y)(B + C_5y)^{4/3}(By + C_4y^2)^{-13/3} \end{aligned} \tag{4.56}$$

Nun haben wir den ganzen Kram beisammen und können uns an die Programmierung machen.

4.3.6 Programmtechnische Umsetzung

Vorüberlegungen: Eigentlich bleibt alles wie gehabt, wir brauchen Anfangs- und Randbedingungen sowie Materialparameter zur Lösung der PDE. Das Newton-Verfahren erfordert eine Iterationsschleife, die können wir ähnlich wie die Zeitschleife zur Lösung der zeitabhängigen Diffusionsgleichung behandeln: Iterationsschleife analog zur Zeitschleife.

4.3.6.1 1 - "Quick and Dirty"

Am Anfang muss immer ein Konzept, eine Idee, für das Programmierprojekt vorliegen. Aber bilden sie sich nicht ein, dass am Reissbrett alles komplett entworfen werden kann. Gerade beim Lernen einer Programmiersprache muss man auch ausloten, was eigentlich geht ... Der erste Versuch des Programms wird in Fachkreisen oft als "Quick and Dirty" bezeichnet.

Sie erinnern sich, im ersten Semester habe ich ihnen Objekt-Orientierung gepredigt. Jetzt bin ich inkonsequent und sage - wir schauen erstmal, ob wir unser Programm irgendwie zum Laufen kriegen und kümmern uns dann um eine saubere objekt-orientierte Programmierung (OOP vs Funktionalität).

Wir klatschen erstmal alles in eine Funktion ... (Funktionalität).

```
void Dialog::on_pushButtonRUN_clicked()
{
  // Anfangsbedingungen
  double x[n];
  for(int i=0;i<n;i++)
    x[i] = -100. + i*10.;
  double bottom_elevation[n];
  for(int i=0;i<n;i++)
    bottom_elevation[i] = 0.04 - i*0.004;
  u_old[0]=0.244918436659073; //-100
  u_old[1]=0.243;             //-90
  u_old[2]=0.242293545352681; //-80
  u_old[3]=0.241;             //-70
  u_old[4]=0.240216955447788; //-60
  u_old[5]=0.235;             //-50
  u_old[6]=0.225684124843115; //-40
  u_old[7]=0.223;             //-30
  u_old[8]=0.220898136369048; //-20
  u_old[9]=0.201434531839821; //-10
  u_old[10]=0.1;              //0
  // Randbedingungen
  u_old[10] = 0.1; // Wasserstand flußabwärts [m]
  u_new[10] = 0.1; // Wasserstand flußabwärts [m]
  // Parameter
  double discharge = 0.05; // Volumenfließrate [m3/s]
  double gravity = 9.81; // [m/s2]
  double friction_law_exponent = 0.5; // Chezy, Manning-Strickler [-]
  double error_tolerance = 1e-3; // [m]
  double bed_slope = 0.0004; // [m/m]
  double bottom_width = 1.; // [m]
  double m = 1.; //
  double friction_coefficient = 10.; //
  double wetted_cross_section[n];
  double water_level_elevation[n];
  double flow_velocity[n];
```

```
  double Froude_number[n];
  double wetted_perimeter[n];
  double hydraulic_radius[n];
  double friction_slope[n];
  // Startwerte
  for(int i=0;i<n;i++)
  {
    wetted_perimeter[i] = bottom_width + 2.*sqrt(1.+m*m)*u_old[i];
    wetted_cross_section[i] = (bottom_width + m*u_old[i])*u_old[i];
    hydraulic_radius[i] = wetted_cross_section[i] / wetted_perimeter[i];
    water_level_elevation[i] = bottom_elevation[i] + u_old[i];
    flow_velocity[i] = discharge/wetted_cross_section[i];
    Froude_number[i] = flow_velocity[i]/(sqrt(gravity*wetted_cross_section[i]\
                       /sqrt(bottom_width*bottom_width+4.*m*wetted_cross_section[i])));
    friction_slope[i] = pow(flow_velocity[i]/ \
              (friction_coefficient*pow(hydraulic_radius[i],friction_law_exponent)),2);
  }
  // Test output
  // Newton-Schritt
  double N,N1,N2,N3,D,D1,D2,D21,D22;
  for(int i=0;i<n-1;i++)
  {
    N1 = pow(discharge,2)/pow(wetted_cross_section[i+1],2) + gravity*u_old[i+1];
    N2 = pow(discharge,2)/pow(wetted_cross_section[i],2) + gravity*u_old[i];
    N3 = gravity*(bed_slope - (friction_slope[i+1]+friction_slope[i])/2.)*(x[i+1]-x[i]);
    N = N1 - N2 - N3;
    D1 = pow(discharge,2)/pow(wetted_cross_section[i],3) * (bottom_width+2.*m*u_old[i])
         - gravity;
    D21 = friction_law_exponent*2.*(sqrt(1+m*m))/wetted_perimeter[i];
    D22 = (1.+friction_law_exponent)/wetted_cross_section[i] * (bottom_width+2.*m*u_old[i]);
    D2 = gravity*friction_slope[i]*(D21-D22)*(x[i+1]-x[i]);
    D = D1 + D2;
    u_new[i] = u_old[i] - N/D;
  }
}
```

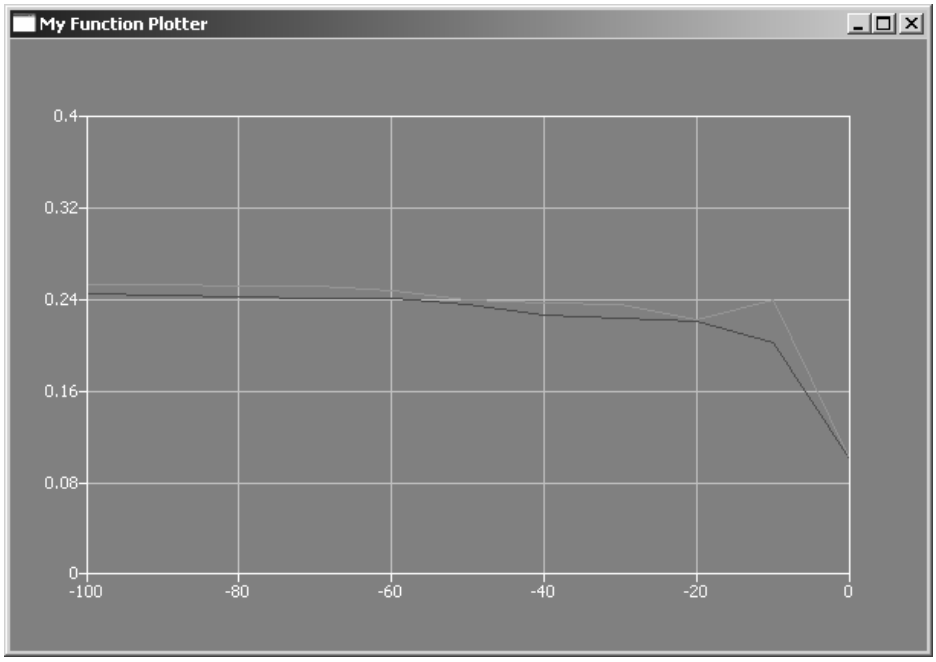

Abbildung 4.16: QAD ... der erste Newton-Schritt

Das Newton-Verfahren scheint zu funktionieren ...

4.3.6.2 2 - OOP

Auch, wenn's schon mal funktioniert, jetzt nicht schwach werden ... jetzt kommt die zweite Programmversion - OOP. Dazu müssen wir eigentlich nur die Funktionen in die entsprechenden Funktion aufteilen und uns Gedanken über lokale und Klassen-Variablen machen.

```
void Dialog::on_pushButtonIC_clicked() {...}
void Dialog::on_pushButtonBC_clicked() {...}
void Dialog::on_pushButtonMAT_clicked() {...}
void Dialog::on_pushButtonRUN_clicked() {...}
```

Der ultimate Test, ob unsere Objekt-Orientierung geklappt hat, zeigt der Härtetest: Funktioniert unsere "All-in-one" Methode.

```
void Dialog::on_pushButtonALL_clicked()
{
  on_pushButtonIC_clicked();
  on_pushButtonBC_clicked();
  on_pushButtonMAT_clicked();
  on_pushButtonRUN_clicked();
}
```

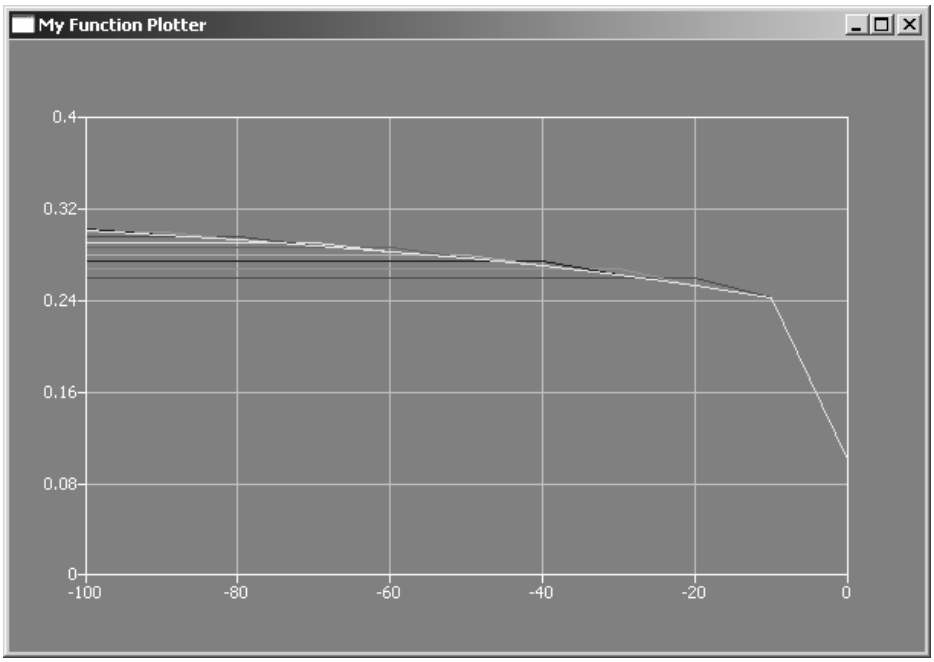

Abbildung 4.17: OOP ... Hurra, das Newton-Verfahren konvergiert

4.3.6.3 3 - Nicely

Wenn wir jetzt noch Kraft und Zeit haben, können wir anfangen, unser Programm schick zu machen, z.B. Bildchens und eine dialog-gestützte Benutzerführung ...

Für das GUI benutzen wir zwei neue Qt Elemente: Strings und editierbare Textfelder.

```
QString sIC;
QLineEdit* lineEditIC;
```

Die Zuweisung von Werten für Textfelder geschieht folgendermaßen.

```
lineEditIC = new QLineEdit();
sIC = "0.25";
lineEditIC->setText(sIC);
```

Ereignis: Mit der folgenden connect-Funktion (im Konstruktor) wird sichergestellt, dass der in das Textfeld eingetragene Wert zunächst gespeichert wird. In der Zuweisungsfunktion für die Anfangsbedingungen wird dann der Wert zunächst in eine Gleitkommazahl gewandelt und anschließend zur Aktualisierung der Anfangsbedingungen benutzt.

```
connect(lineEditIC,SIGNAL(returnPressed()),this,SLOT(setText(sIC)));
...
double IC = lineEditIC->text().toDouble();
for(int i=0;i<n-1;i++)
{
  u_old[i] = IC;
}
```

Layouts lassen sich beliebig ineinander verschachteln. Die Grundstruktur unseres Dialogs (`mainLayout`) sind zwei Spalten (`QVBoxLayout: leftLayout und rightLayout`) die nebeneinander angelegt sind (`QHBoxLayout`).

```
QHBoxLayout *mainLayout = new QHBoxLayout;
mainLayout->addLayout(leftLayout);
mainLayout->addLayout(rightLayout);
setLayout(mainLayout);
```

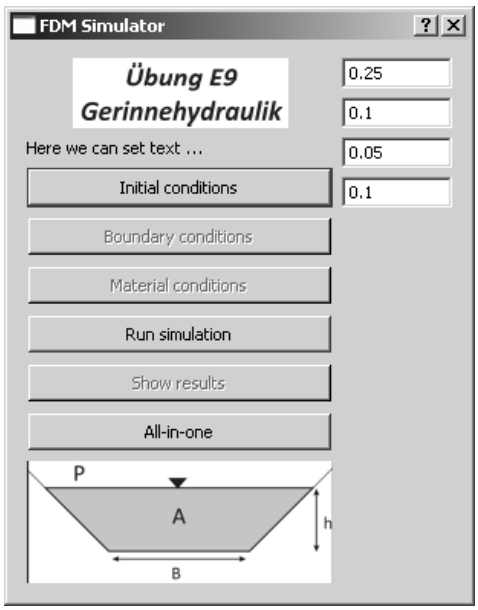

Abbildung 4.18: GUI ... matters

4.4 Grundwasserhydraulik

Wir haben uns bisher mit linearen und nicht-linearen Diffusionsgleichungen beschäftigt. Bisher war alles ein-dimensional. Mit numerischen Methoden gibt es keine Beschränkungen für die Geometrie. Im letzten Teil unserer Vorlesung lösen wir nun ein zwei-dimensionales Problem für eine horizontale Grundwasserströmung.

4.4.1 Grundwassergleichung

Das theoretische Grundgerüst liefert das Euler-Konzept (siehe Abschn. 1.1). Wir starten mit der Massenbilanzgleichung (siehe Abschn. 1.2). Für ein poröses Medium reduziert sich der für Fluide betretbare Raum um die Porosität n, die sich ändern kann.

$$\frac{\partial n\rho}{\partial t} + \nabla \cdot (n\rho \mathbf{v}) = Q_\rho \tag{4.57}$$

Für ein inkompressibles Fluid gilt dann (PF)

$$\rho \frac{\partial n}{\partial t} + \rho \nabla \cdot (n\mathbf{v}) = Q_\rho \tag{4.58}$$

oder noch besser

$$\frac{\partial n}{\partial t} + \nabla \cdot (n\mathbf{v}) = \frac{Q_\rho}{\rho_0} \tag{4.59}$$

In der Grundwasserhydraulik gilt

$$\frac{\partial n}{\partial t} = S \frac{\partial h}{\partial t} \tag{4.60}$$

$$n\mathbf{v} = \mathbf{q} = -\mathbf{K}\nabla h \quad \text{(Darcy Gesetz)} \tag{4.61}$$

Dabei sind: S der Speicherkoeffizient, h die Piezometer- oder hydraulische Höhe, $\mathbf{q}$ die Darcy- oder Filtergeschwindigkeit und $\mathbf{K}$ der hydraulische Leitfähigkeitstensor.

Die Grundwassergleichung lässt sich nun wie folgt schreiben.

$$S \frac{\partial h}{\partial t} + \nabla \cdot (n\mathbf{v}) = Q \tag{4.62}$$

$$S \frac{\partial h}{\partial t} - \nabla \cdot (\mathbf{K}\nabla h) = Q \tag{4.63}$$

$$S \frac{\partial h}{\partial t} - \frac{\partial}{\partial x}\left(K_x \frac{\partial h}{\partial x}\right) - \frac{\partial}{\partial y}\left(K_y \frac{\partial h}{\partial y}\right) - \frac{\partial}{\partial z}\left(K_z \frac{\partial h}{\partial z}\right) = Q \tag{4.64}$$

Wir begnügen uns mit einem 2-D horizontalen Modell.

$$S\frac{\partial h}{\partial t} - \frac{\partial}{\partial x}\left(K_x\frac{\partial h}{\partial x}\right) - \frac{\partial}{\partial y}\left(K_y\frac{\partial h}{\partial y}\right) = Q \tag{4.65}$$

4.4.2 Testbeispiel

Wir halten unser Anwendungsbeispiel natürlich so einfach wie möglich, dennoch enthält es viele wichtige "Features" von Grundwassermodellen (alles Weitere lernen sie natürlich bei Prof. Liedl ...). Das Beispiel stammt noch aus der Tübinger Zeit (Master Course in Applied Geosciences - AEG, daher auch alles in Englisch) und wurde durch Sebastian Bauer erarbeitet (jetzt Professor für GeoHydroModellierung in Kiel).

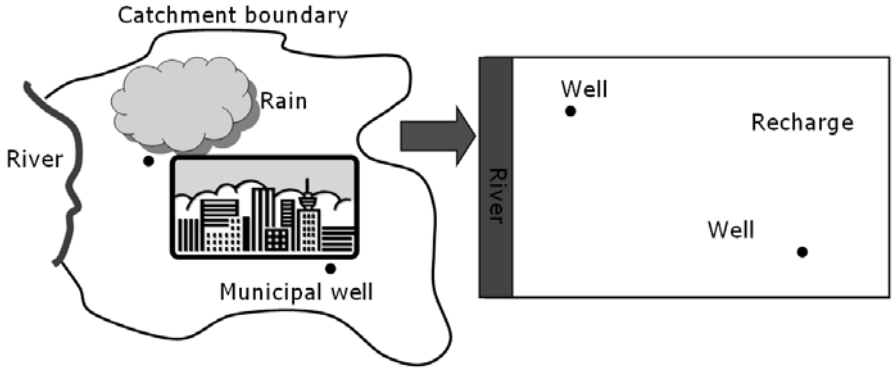

Abbildung 4.19: GW Testbeispiel

Die Abb. 4.19 zeigt uns ein Systemmodell für ein sogenanntes Grundwasser-Catchment mit einem Fluss (Vorfluter) und zwei Brunnen. Weitere wichtige geohydrologische Begriffe sind die Grundwasserneubildung (recharge) durch Niederschläge und Wasserscheiden. Das besondere Merkmal eines Catchments (Einzugsgebiet) ist seine Abgeschlossenheit durch Wasserscheiden, d.h. die Catchment-Randbedingung ist eine "no-flow" boundary condition (da geht nichts durch). In unserem Fall heisst das, die komplette Grundwasserneubildung geht ab in den Vorfluter.

Die Abb. 4.20 ist die mathematische Übersetzung des geohydrologischen Konzeptmodells in Abb. 4.19.

- Fluss: ist eine Randbedingung erster Art (siehe Abschn.), der Wasserstand h wird vorgegeben. Catchment: wie gesagt - no flow, also Fließgeschwindigkeit ist Null, was nach Darcy heisst, der Piezometer-Gradient ∇h ist Null.

- Grundwasserneubildung: ist ein flächenhafter Quellterm für das Grundwassersystem.
- Brunnen: sind lokale Senkenterme.

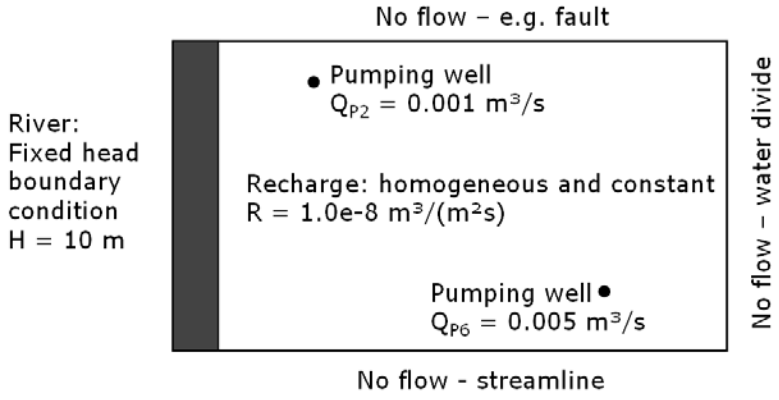

Abbildung 4.20: GW Testbeispiel - Randbedingungen

4.4.3 Finite-Differenzen-Verfahren

Nach der mathematischen Übersetzung (Abb. 4.20) nun die numerische Translation. Abb. 4.21 zeigt uns das finite Differenzen Gitter. Das Netz ist blockzentriert, d.h. wir nehmen das geometrische Zentrum der Zelle als Bezugspunkt.

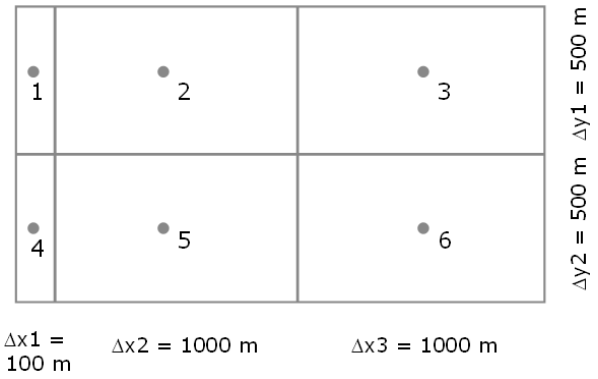

Abbildung 4.21: GW Testbeispiel - Finite Differenzen Schemata

Unser Grundwasserleiter (Aquifer) ist heterogen, d.h. es sind verschiedene geologische Schichten beteiligt (Abb. 4.22). T ist die sogenannte Transmissivität,

der Quotient von hydraulischer Durchlässigkeit und Speicherkoeffizient.

$$T = \frac{K}{S} \tag{4.66}$$

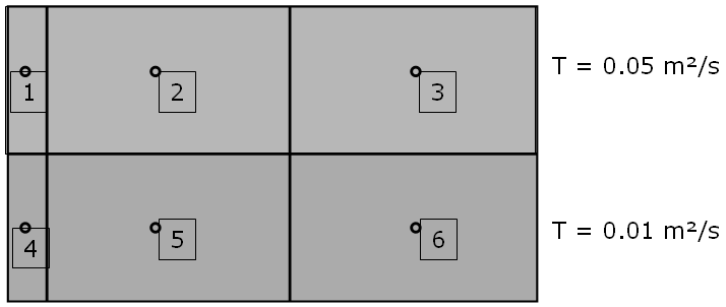

Abbildung 4.22: GW Testbeispiel - Heterogener Aquifer

Schauen wir uns mal die Wasserbilanz für einen FD-Knoten genauer an. Q_{ij} ist der Zufluss zum Knoten j aus der Zelle i, also Q_{52} ist der Zufluss zum Knoten 2 aus der Zelle 5. Für den Knoten 2 gilt

$$Q_{12} + Q_{32} + Q_{52} + Q_R + Q_{P2} = 0 \tag{4.67}$$

Dabei sind Q_R der Zufluss durch die Grundwasserneubildung und Q_{P2} der Abfluss durch den Brunnen im Knoten 2.

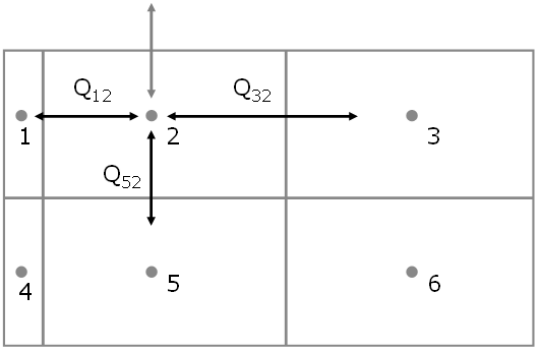

Abbildung 4.23: GW Testbeispiel - Wasserbilanz

Wir benutzen folgendes Differenzenschema.

$$\frac{\partial h}{\partial x} = \frac{h_i - h_j}{x_i - x_j} \tag{4.68}$$

$$\frac{\partial h}{\partial y} = \frac{h_i - h_j}{y_i - y_j} \tag{4.69}$$

Da unser FD-Gitter nicht equidistant ist und unser Aquifer heterogen schreiben wir besser.

$$\frac{\partial h}{\partial x}|_{12} = \frac{h_1 - h_2}{\Delta x_1/2 + \Delta x_5/2} \tag{4.70}$$

Damit können wir für die Flussterme schreiben.

$$Q_{12} = \Delta y_1 \frac{\Delta x_1 + \Delta x_2}{\Delta x_1/T_1 + \Delta x_2/T_2} \times \frac{h_1 - h_2}{\Delta x_1/2 + \Delta x_2/2} \tag{4.71}$$

$$Q_{52} = \Delta x_2 \frac{\Delta y_5 + \Delta y_2}{\Delta y_5/T_5 + \Delta y_2/T_2} \times \frac{h_5 - h_2}{\Delta y_5/2 + \Delta y_2/2} \tag{4.72}$$

Die Zahlen eingesetzt ergibt sich für

$$Q_{12} = 0.454545 - 0.0454545 h_2 \tag{4.73}$$

Die restlichen Terme für die Wasserbilanz in der Zelle 2 sind.

$$Q_{52} = 0.03333 h_5 - 0.03333 h_2 \tag{4.74}$$

$$Q_{32} = 0.02500 h_3 - 0.02500 h_2 \tag{4.75}$$

$$Q_R = R \Delta x_2 \Delta y_1 = 0.005 \tag{4.76}$$

$$Q_{P2} = -0.001 \tag{4.77}$$

Damit ist die Wasserbilanz für die Zelle 2.

$$2 : 0.458545 - 0.103788 h_2 + 0.025 h_3 + 0.03333 h_5 = 0 \tag{4.78}$$

Für die anderen Zellen ergibt sich.

$$\begin{aligned} 3 &: 0.0050 + 0.0250 h_2 - 0.0583 h_3 + 0.0333 h_6 = 0 \\ 5 &: 0.0959 + 0.0333 h_2 - 0.0474 h_3 + 0.0050 h_6 = 0 \\ 6 &: 0.0000 + 0.0333 h_3 + 0.0050 h_5 - 0.0383 h_6 = 0 \end{aligned} \tag{4.79}$$

Das sieht doch wieder ganz verdächtig nach

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{4.80}$$

aus, wir müssen ein Gleichungssystem lösen, eine leichte Übung für uns ... (Gauss Löser).

Ergebnis:

$$\begin{aligned} h_1 &= 10.00 \\ h_2 &= 10.24 \\ h_3 &= 10.41 \\ h_4 &= 10.00 \\ h_5 &= 10.31 \\ h_6 &= 10.39 \end{aligned} \tag{4.81}$$

(4.82)

4.4.4 Programmtechnische Umsetzung

Letzter Akt: die Programmierung des Grundwassermodells (siehe Übung E10 auf der WebSeite).

Die Abb. 4.24 zeigt die Dialog-Gestaltung für unseren einfachen Grundwassersimulator.

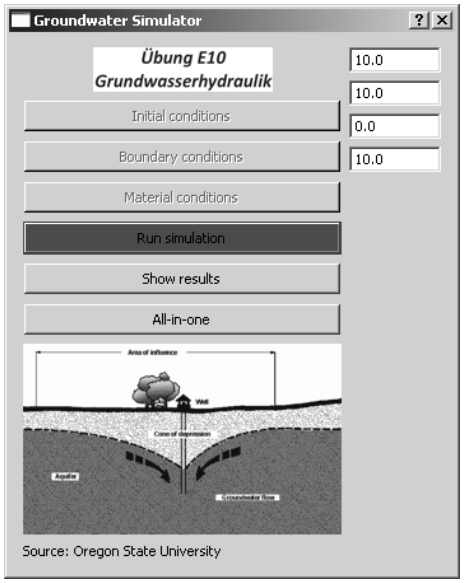

Abbildung 4.24: GW Testbeispiel - Dialog

Für den Test, dass das Gleichungssystem korrekt aufgestellt wird, schreiben wir uns eine kleine Funktion.

```
void Dialog::TestOutput(double*a,double*b)
{

  for(int i=0;i<6;i++)
  {
    for(int j=0;j<6;j++)
    {
      out_file << a[6*i+j] << "\t";
    }
    out_file << " : " << b[i] << endl;
  }
}
```

die folgendes Textfile produziert.

```
1       0       0       0       0       0      : 10
0       -0.104  0.025   0       0.0333  0      : -0.459
0       0.025   -0.0583 0       0       0.0333 : -0.005
0       0       0       1       0       0      : 10
0       0.0333  0       0       -0.0474 0.005  : -0.0959
0       0       0.0333  0       0.005   -0.383 : 0
```

Wir vergleichen mit der Theory (4.78) und (4.80) und sehen, dass alles geklappt hat.

```
void Dialog::TestOutput(double*x)
{
  for(int i=0;i<6;i++)
  {
    out_file << "h" << QString::number(i).toStdString() << ":" << x[i]
             << endl;
  }
}
```

Sie sehen, wir können die gleiche Schreibfunktion `TestOutput` für verschiedene Aufgaben nehmen. Dies nennt sich Polymorphismus in C++. Der Kompiler erkennt die verschiedenen Funktionen anhand der unterschiedlichen Parameterliste. Auch die Ergebnisse passen.

```
h1 = 10.00
h2 = 10.24
h3 = 10.41
h4 = 10.00
h5 = 10.31
h6 = 10.39
```

Für die Lösung des Gleichungssystems haben wir wieder unseren Gauss-Löser eingesetzt (siehe Abschn. 4.2). Wenn noch Zeit ist, schauen wir uns alternativ noch anderes Lösungsverfahren an: Gauss-Seidel.

Das war's (leider schon), unsere Vorlesung "Hydroinformatik" mit zahlreichen Übungen (die sind viel zeitaufwendiger in der Vorbereitung) ... Ich hoffe, sie haben Einiges gelernt (ich auf jeden Fall...). Es war mir eine große Freude, sie ein Jahr in ihrem Studium begleiten zu dürfen. Alles Gute für sie (und ich bin zuversichtlich, dass sich ihre Hydroinformatik-Kenntnisse für ihren beruflichen Weg als vorteilhaft erweisen werden ...)

Last and least...

Um unsere Programme noch schicker zu machen, können wir sogennante SplashScreens `QSplashScreen` vor das Programm spannen. Also ein nettes Bildchen, das Aufmerksamkeit erwecken soll (Abb. 4.25).

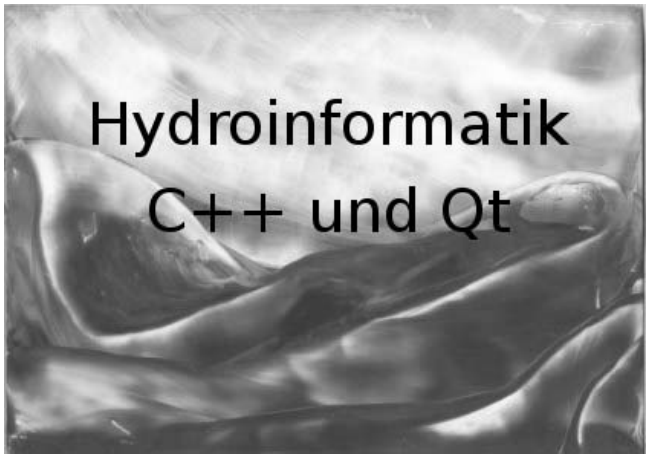

Abbildung 4.25: SplashScreen für unsere Vorlesung

Nun aber wirklich die letzte Übung, die mein Sohn Bastian zusammengebastelt hat.

Übung E11: SplashScreen

```
int main(int argc, char *argv[])
{
  QApplication a(argc, argv);
  //.....................................................
  QPixmap pixmap ("../bk-1.png");
  QSplashScreen splash(pixmap);
  splash.show();
  splash.showMessage(QObject::tr("Übung E10 wird geladen..."), Qt::black);
  QTime dieTime = QTime::currentTime().addSecs(3); while(QTime::currentTime()<dieTime)\
                  QCoreApplication::processEvents(QEventLoop::AllEvents, 100);
  // Warte-Funktion (http://lists.trolltech.com/qt-interest/2007-01/thread00133-0.html)
  //.....................................................
  Dialog w;
  w.setWindowTitle("Groundwater Simulator");
  w.setFixedWidth(350);
  w.show();
  return a.exec();
}
```

Part IV

Visual C++ mit Qt

Kapitel 5

Qt

Die Geschichte: Bei der ersten HI-I Vorlesung gabe es ziemlich peinliche eine Panne. Die (mit viel Mühe vorbereiteten Übungen) liefen zwar mit meiner MSVC++ (professional) Version aber (nicht ohne Weiteres) mit freien MSVC++ Express Version, wo die MFC standartmäßig nicht mehr dabei ist. Dies zeigt uns aber, auch beim Programmieren niemals nur auf ein Pferd setzen. Wichtig ist, unser eigener Code muss C++ Standard sein, den können wir dann problemlos in verschiedene GUI Frameworks, wie MSVC++, Qt etc. einbauen. Warum jetzt noch Qt ? Wir werden in den Übungen sehen, dass Qt sehr dicht an C++ Standards dran ist und Qt ist a cross-platform GUI. Qt läuft auf Windows, Linux, Mac ... In der Anlage 5.1 finden sie eine Anleitung zur Installation von Qt.

Am Ende des letzten Semesters haben wir uns bereits ein bisschen mit Qt beschäftigt [9]. Nun wollen wir das Visual C++ für unsere numerischen Anwendungen benutzen.

5.1 Qt Projekt

Es ist ihnen hoffentlich gelungen, Qt für ihr Betriebssystem erfolgreich zu installieren. Eine kurze Installationsanleitung war im Skript [9] (Abschn. 12.8) zu finden. Hier noch mal die Web-Seite für den Download
`http://qt.nokia.com/products`. Der große Vorteil von Qt ist die *Plattformunabhängigkeit* sowie die freie Verfügbarkeit unter der GPL (Gnu Public License).

Wir starten Qt mit einem Doppelklick auf das Desktop-Symbol (Abb. 5.1).

Wenn alles gut geht gegangen ist sollten sie im Qt User Interface (UI) landen (Abb. 5.2).

Abbildung 5.1: Qt Start

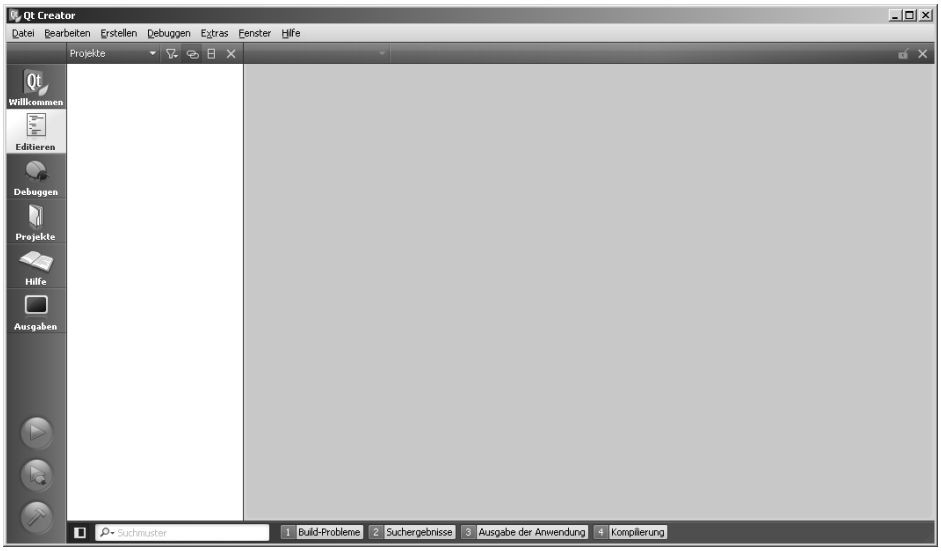

Abbildung 5.2: Qt Creator

Zunächst müssen wir ein neues (leeres) Qt Projekt anlegen (Abb. 5.3-5.6).

Abbildung 5.3: Neues Qt Projekt anlegen - Schritt 1

Schritte:

- pro Datei

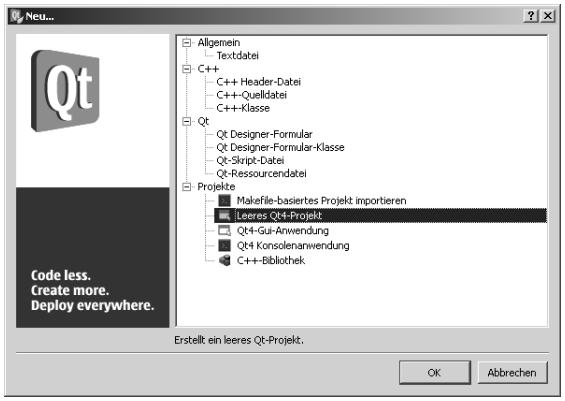

Abbildung 5.4: Neues Qt Projekt anlegen - Schritt 2

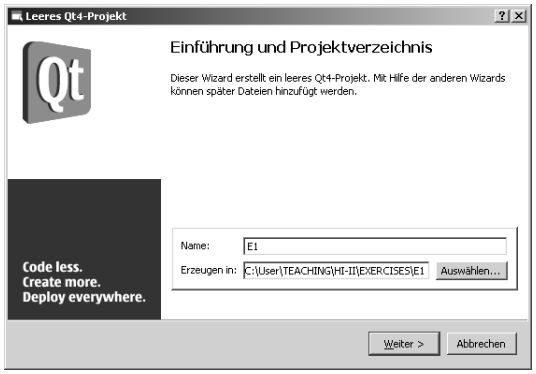

Abbildung 5.5: Neues Qt Projekt anlegen - Schritt 3

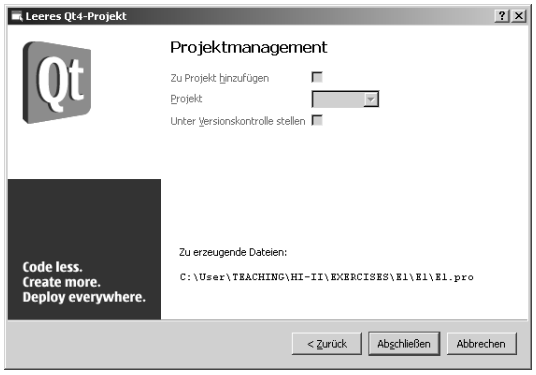

Abbildung 5.6: Neues Qt Projekt anlegen - Schritt 4

Nachdem ein leeres Qt Projekt angelegt ist, fügen wir zunächst eine existierende Quell-Datei mit der `main` Funktion hinzu (Abb. 5.7).

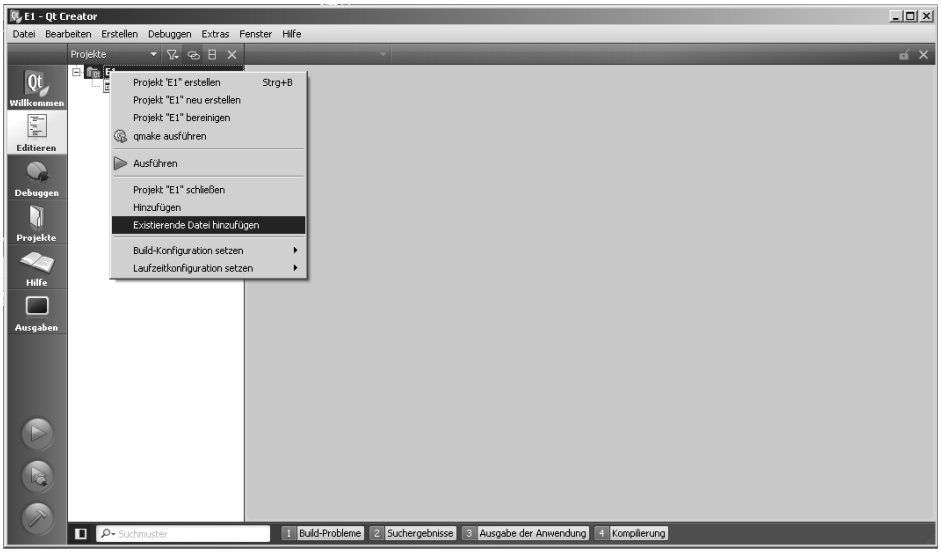

Abbildung 5.7: Hinzufügen einer Quell-Datei

Es können weitere Quell-Dateien hinzugefügt werden, wie z.B. der Qt Plotter (Abb. 5.8).

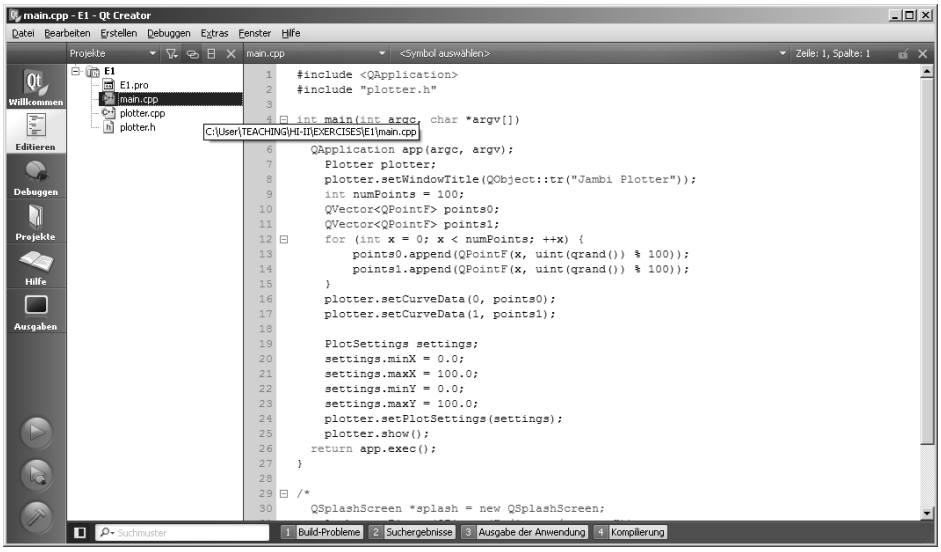

Abbildung 5.8: Qt Editor

Unser Projekt ist schon kompilierfähig. Mit einem Druck auf das "grüne Knöpfchen" (Abb. 5.9) wird kompliert und das Programm ausgeführt.

Abbildung 5.9: Kompilation und Programmstart

Im Ergebnis haben wir unseren ersten Qt Plot (Abb. 5.10), unheimlich - nicht wahr.

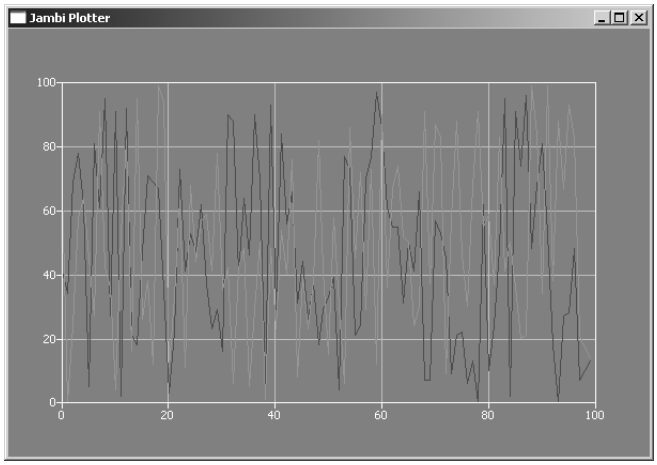

Abbildung 5.10: Qt Plotter

Jetzt schauen wir uns die Sache mal etwas genauer an.

Übung: E5.1

```
#include <QApplication>
#include "plotter.h"

int main(int argc, char *argv[])
{
  QApplication app(argc, argv);
  // Data
  int numPoints = 100;
  QVector<QPointF> points0;
  QVector<QPointF> points1;
  for (int x = 0; x < numPoints; ++x) {
    points0.append(QPointF(x, uint(qrand()) % 100));
    points1.append(QPointF(x, uint(qrand()) % 100));
```

```
  }
  // Plotter
  Plotter plotter;
  plotter.setWindowTitle(QObject::tr("Jambi Plotter"));
  plotter.setCurveData(0, points0);
  plotter.setCurveData(1, points1);
  PlotSettings settings;
  settings.minX = 0.0;
  settings.maxX = 100.0;
  settings.minY = 0.0;
  settings.maxY = 100.0;
  plotter.setPlotSettings(settings);
  plotter.show();
  //
  return app.exec();
}
```

5.2 Function Viewer

Wir haben gesehen, wie unkompliziert ist, einfache Plots mit Qt zu erzeugen ("Matlab" mit Qt). Dabei haben wir den Qt x-y Plotter in unsere Anwendung eingebunden. Nun wollen wir aber nicht irgendwelche Bildchens produzieren, sondern unsere Anwendungsbeispiele bearbeiten.

5.2.1 Steady diffusion

Die (quellenfreie) stationäre Diffusionsgleichung hat im eindimensionalen Fall

$$\frac{d^2y}{dx^2} = 0 \tag{5.1}$$

ein ziemlich triviale Lösung, eine lineare Funktion.

$$y = ax + b \tag{5.2}$$

Wir wollen nun diese Funktion in unseren "function viewer" implementieren.

Übung: E5.2.1

```
double a=1.,b=0.,y;
// y = ax + b
for (int x = 0; x < numPoints; ++x)
{
  y = a*x + b;
  points0.append(QPointF(x,y));
}
```

Das war einfach. Nun müssen wir uns Gedanken über die physikalische Bedeutung der Koeffizienten a und b machen. Dies ist ihre 3. HW.

HW3: Berechnen sie die Koeffizienten a und b für Wärmediffusion, wenn die linke Temperatur $y = 283$ K und die rechte Temperatur $y = 300$ K sind.

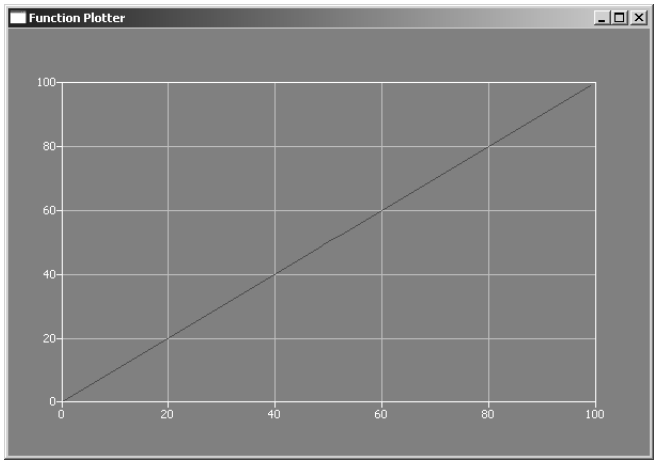

Abbildung 5.11: Stationäre 1-D Diffusion

5.2.2 "Richtige" Diffusion

Bei einer "richtigen" Diffusion müssen wir eine PDE lösen.

$$\frac{\partial y}{\partial t} - \alpha \frac{\partial^2 y}{\partial x^2} = 0 \tag{5.3}$$

Eine Basislösung für den eindimensionalen Fall ist

$$y = sin(\pi x) e^{-\alpha t^2} \tag{5.4}$$

Übung: E5.2.2

```
#include <math.h>
#define PI 3.14159265358979323846
double x,y,alpha=1.,t=0.5,pi=3.14;
// y = sin(pi*x) * exp(-alpha*t^2)
for (int i = 0; i < numPoints+1; ++i)
{
  x = double(i)/double(numPoints);
  y = sin(PI*x) * exp(-alpha*t*t);
  points0.append(QPointF(i,y));
}
```

.

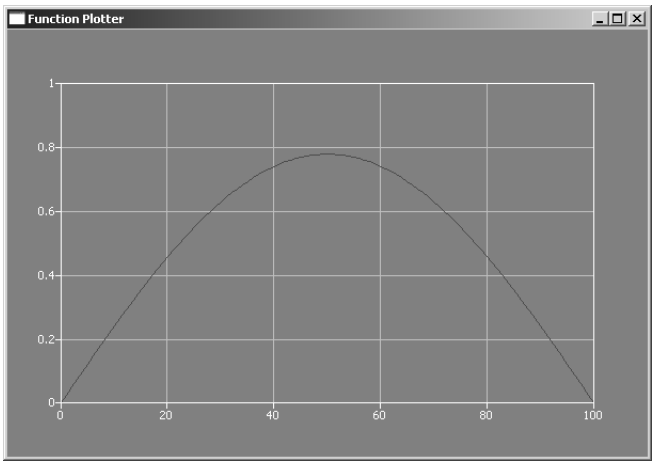

Abbildung 5.12: Diffusion für einen Zeitpunkt

Für jede Zeit t müssen wir nun das Programm neu kompilieren. Viel schicker wäre es doch, wenn wir eine Animation über einen Zeitbereich hätten und Parameter eingeben könnten - also ein benutzerdefiniertes, dialogbasiertes Programm. Zunächst behelfen wir uns in der nächsten Übung mit einigen ausgewählten Zeitschritten. Hiefür müssen wir die entsprechende Anzahl von Vektoren einrichten.

Übung: E5.2.2b

```
QVector<QPointF> points0;
QVector<QPointF> points1;
QVector<QPointF> points2;
for (int i = 0; i < numPoints+1; ++i)
{
  x = double(i)/double(numPoints);
  t = 0.1;
  y = sin(PI*x) * exp(-alpha*t*t);
  points0.append(QPointF(i,y));
  t = 0.5;
  y = sin(PI*x) * exp(-alpha*t*t);
  points1.append(QPointF(i,y));
  t = 0.9;
  y = sin(PI*x) * exp(-alpha*t*t);
  points2.append(QPointF(i,y));
}
```

Um mehrere Kurven gleichzeitig plotten zu können, brauchen wir diese nur wie folgt setzen.

```
plotter.setCurveData(0, points0);
plotter.setCurveData(1, points1);
plotter.setCurveData(2, points2);
```

Das Ergebnis für die heutige Veranstaltung (unser kleines "Matlab") kann sich schon sehen lassen.

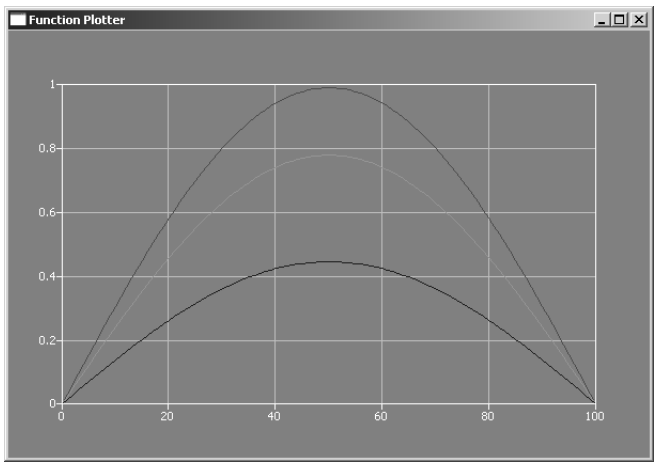

Abbildung 5.13: Diffusion für verschiedene Zeitpunkte

HW4: Implementieren sie die zweite Basislösung für die quellenfreie eindimensionale Diffusionsgleichung.

$$y = sin(\sqrt{\pi\alpha}x)e^{-\pi t} \tag{5.5}$$

5.3 Qt Dialog

Wir knüpfen direkt an das Kapitel 11.4 des Erst-Semester-Skriptes an. Dort haben wir uns bereits mit Qt-Dialogen (Klasse `QDialog`) für unsere Studenten-Datenbank beschäftigt. Nun wollen wir die Dialog-Anwendung etwas komplexer gestalten, in dem wir den Qt-Plotter mit einbinden. Das Konzept für den Funktions-Plotter ist in der Abbildung 5.14 dargestellt.

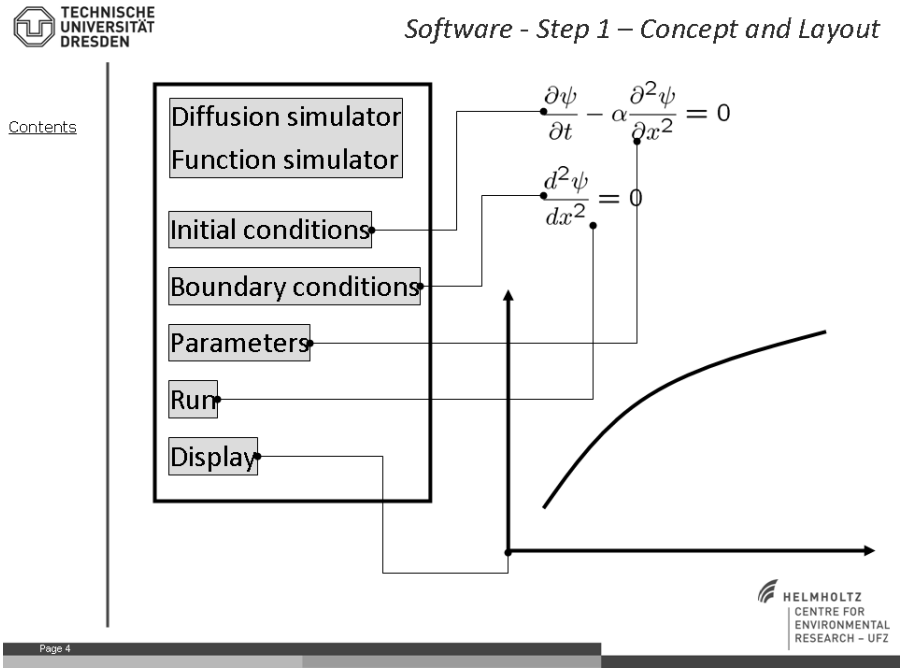

Abbildung 5.14: Konzept für den Function-Plotter

Die Dialogführung des Funktions-Plotters besteht aus vier Teilen:

1. Auswahl der zu lösenden Gleichung,
2. Festlegen von Bedingungen (Anfangs- und Randbedingungen sowie Materialparameter),
3. Berechnung der Lösung,
4. Grafische Darstellung der Ergebnisse.

5.3.1 Die `main` Funktion

Das Programm besteht aus drei Dateien (Übung E5.3):

- main.cpp - die `main` Funktion
- dialog.h/cpp - alles was zum Dialog gehört
- plotter.h/cpp - der Qt Plotter (den benutzen wir einfach, ohne genau zu wissen, wie er funktioniert (wir müssen nicht alles wisse))

Die Aufgabe der main Funktion ist es die Applikation `QApplication a` sowie den Dialog `Dialog w` anzulegen und zu aufzurufen `w.show()`. MIt den Dialogfunktionen `setWindowTitle` und `setFixedWidth` können den Dialog-Titel sowie die Breite (in Pixeln) einstellen.

```
#include <QtGui/QApplication>
#include "dialog.h"
int main(int argc, char *argv[])
{
  QApplication a(argc, argv);
  Dialog w;
  w.setWindowTitle("Function Simulator");
  w.setFixedWidth(200);
  w.show();
  return a.exec();
}
```

Ein ausführbares Qt Programm können wir entweder mit dem Qt Creator erzeugen oder aber die entsprechende Befehle als Kommando-Zeilen einzugeben. Hierzu benötigen wir 4+1 Schritte (siehe auch Skript Hydroinformatik I):

- Gehen sie in ihre Start - Programme Menü (Windows) und starten sie die den Qt command prompt (Fig. 5.15).

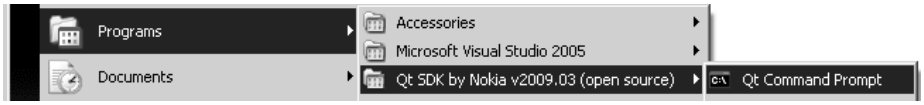

Abbildung 5.15: Qt Kommando Zeile

- Gehen sie dann in das Verzeichnis, wo ihr Quell-Code liegt (Fig. 5.16).
- Mit `qmake -project` erzeugen sie ein Qt Projekt.
- Mit `qmake` legen sie ein makefile an.
- Mit `mingw32-make` (kein Leerzeichen) kompilieren sie das Projekt und generieren ein ausführbares Programm.
- Mit `E10_1_HelloQt.exe` starten sie das Programm.

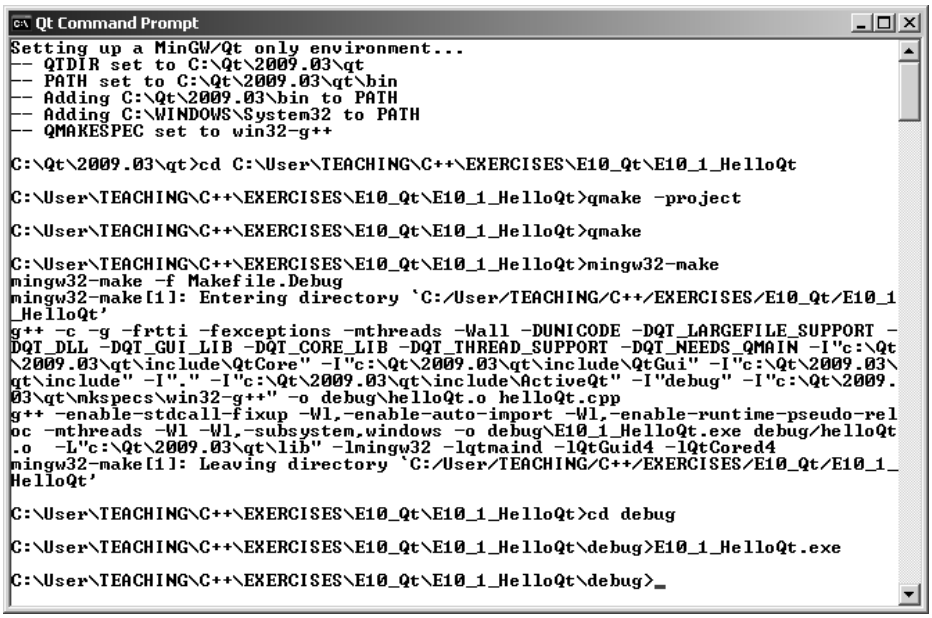

Abbildung 5.16: Kompilation mit Qt Kommando Zeilen

5.3.2 Dialog-Gestaltung

Entsprechend unserem Layout (Abb. 5.14) benötigen wir lediglich eine vertikale Box `QVBoxLayout` in die alle Elemente eingefügt werden.

```
QVBoxLayout *mainLayout = new QVBoxLayout;
mainLayout->addWidget(label_ogs);
mainLayout->addWidget(labelHeader);
mainLayout->addWidget(pushButtonIC);
mainLayout->addWidget(pushButtonBC);
mainLayout->addWidget(pushButtonMAT);
mainLayout->addWidget(pushButtonRUN);
mainLayout->addWidget(pushButtonSHO);
setLayout(mainLayout);
```

Damit das Layout nicht ganz so langweilig wird, bauen wir auch eine kleine Grafik ein (natürlich das OGS Logo). Dies kann mit einem `QLabel` erfolgen.

```
QLabel *label_ogs = new QLabel();
label_ogs->setAlignment(Qt::AlignCenter);
label_ogs->setPixmap(QPixmap("../ogs_15.png"));
```

Das Ergebnis ist in Abb. 5.17 zu sehen.

Abbildung 5.17: Layout des Dialogs

HW: Bauen Sie anstelle des OGS-Logos eine andere Grafik ein.

5.3.3 Signal-Slots

Um die Verbindung zwischen einem Ereignis (Signal) und einer Reaktion (Slot) herzustellen, wird die Function `connect` benutzt.

```
connect(pushButtonIC,SIGNAL(clicked()),this,SLOT(on_pushButtonIC_clicked()));
```

Die Nachrichtenbox meldet uns, das die Verbindung zwischen dem Knopfdruck und dem Aufrufen der vernüpften Funktion geklappt hat (Abb. 5.18).

Abbildung 5.18: Test der Signal-Slot-Verbindung mit einer MessageBox

5.3.4 Plotter-Einbindung

Bleibt noch der Aufruf der Plotters im Dialog, wenn das Knöpfchen 'Show' gedrückt wurde. Hierfür muss die entsprechende Signal-Slot Verbindung hergestellt werden.

```
connect(pushButtonSHO,SIGNAL(clicked()),this,SLOT(on_pushButtonSHO_clicked()));
```

Die Benutzung des Plotters in der Slot-Funktion ist genau wie in der Übung E5.1 mit einem wichtigen Unterschied: Die Plotter-Instanz wird als Zeiger, `Plotter *plotter` implementiert. Warum ?

```
void Dialog::on_pushButtonSHO_clicked()
{
  Plotter *plotter = new Plotter;
  plotter->setWindowTitle(QObject::tr("My Function Plotter"));
  int numPoints = 100;
  QVector<QPointF> points0;
  QVector<QPointF> points1;
  for (int x = 0; x < numPoints; ++x) {
    points0.append(QPointF(x, uint(qrand()) % 100));
    points1.append(QPointF(x, uint(qrand()) % 100));
  }
  plotter->setCurveData(0, points0);
  plotter->setCurveData(1, points1);
  PlotSettings settings;
  settings.minX = 0.0;
  settings.maxX = 100.0;
  settings.minY = 0.0;
  settings.maxY = 100.0;
  plotter->setPlotSettings(settings);
  plotter->show();
}
```

Kapitel 6

Fragenkatalog

6.1 Mechanik

1. Erklären sie kurz die Euler'sche Betrachtungsweise der Mechanik.
2. Erklären sie kurz die Lagrange'sche Betrachtungsweise der Mechanik.
3. Was ist ein Kontrollvolumen? Ist die Größe eines Kontrollvolumens definiert?
4. Wie ist eine Bahnlinie definiert (mathematische Beschreibung)?
5. Bitte ordnen sie folgende Begriffe zu: Euler-Konzept, Lagrange-Konzept, Bahnlinie, Stromlinie, Kontrollvolumen.
6. Was ist eine partielle Ableitung? Was bedeutet z.B. $\partial T/\partial x$ wenn $T(t, x)$ eine Funktion von Raum x und Zeit t ist.
7. Wie ist die mathematische Beziehung zwischen totaler (oder materieller) und partieller Ableitung definiert?
8. Wie ist die mathematisch Divergenz definiert? Schreiben sie die Divergenz des Geschwindigkeitsfeldes in Vektor- und Koordinatenschreibweise.
9. Was bedeutet die Divergenz physikalisch?
10. Was bedeutet der Ausdruck $\oint_A \mathbf{\Phi} \cdot d\mathbf{S}$? Dabei ist A die Oberfläche des Kontrollvolumens, $\mathbf{\Phi}$ der Flussvektor, $\mathbf{S}$ der Normalenvektor auf die Oberfläche des Kontrollvolumens.
11. Was besagt das Reynolds'sche Transport-Theorem?
12. Aus welchen Komponenten besteht ein mechanischer Fluss?

13. Wie ist der advektive Fluss mechanisch definiert?
14. Wie ist der diffusive Fluss mechanisch definiert?
15. Erklären sie die physikalische Bedeutung der einzelnen Terme in der allgemeinen Bilanzgleichung (1.25).
16. Welche Erhaltungsgrößen gibt es in der Mechanik?
17. Was sind die physikalischen Bedingungen für eine divergenzfreie Strömung $\nabla \cdot \mathbf{v} = 0$?
18. Erklären sie die physikalische Bedeutung der einzelnen Terme in der Navier-Stokes-Gleichung (1.25).
19. Welche Kräfte wirken in der Darcy-Gleichung (1.76)?

6.2 Numerik

1. Was ist ein diskretes Näherungsverfahren für eine partielle Differentialgleichung (PDE)?
2. Was ist das Ergebnis eines diskretes Näherungsverfahren für eine partielle Differentialgleichung (PDE)?
3. Was ist der Unterschied zwischen Finite-Differenzen-Methode (FDM) und Finite-Elemente-Methode (FEM)?
4. Wie ist eine konsistente Näherungslösung einer partiellen Differentialgleichung (PDE) mathematisch definiert?
5. Wie ist eine konvergente Näherungslösung einer partiellen Differentialgleichung (PDE) mathematisch definiert?
6. Welche Verfahren kennen sie zur Lösung linearer Gleichungssysteme?
7. Welche Verfahren kennen sie zur Lösung nichtlinearer Gleichungssysteme?
8. Erklären sie kurz das Prinzip des Newton-Verfahrens.
9. Worin besteht der Unterschied zwischen Newton und Newton-Raphson Verfahren?
10. Wozu benötigen wir Fehlertoleranzen bei der Lösung iterativer Verfahren (z.B. Newton-Verfahren)? Wie groß sind Fehlertoleranzen anzusetzen?
11. Schreiben sie die ersten vier (4) Terme der Taylor-Reihenentwicklung der Funktion $f(u)$ an der Stelle u_0. Welche Genauigkeit hat diese Taylor-Reihe für $f(u)$ bezüglich $\Delta u = u - u_0$

12. Auf welcher theoretischen Grundlage können Differenzengleichungen entwickelt werden?

13. Es gibt Vorwärts- und Rückwärtsdifferenzen-Verfahren. Schreiben sie beide Differenzen-Gleichungen für

$$\left[\frac{\partial u}{\partial y}\right]_i^n \tag{6.1}$$

14. Was bedeutet FTCS?

15. Was ist der Vorteil zentraler Differenzen gegenüber Vorwärts- und Rückwärtsdifferenzen-Verfahren?

16. Es gibt explizite und implizite Verfahren für die Lösung von PDEs. Worin besteht der Unterschied?

17. Ordnen sie zu: explizites, iteratives, implizites Verfahren und stabil, instabil, oszillatorisch, konvergent.

18. Was ist Neumann-Zahl? Schreiben sie mathematische Definition.

19. Welche Rolle spielt die Neumann-Zahl bei der numerischen Stabilität der diskreten Lösung der Diffusionsgleichung?

6.3 Prozessverständnis

1. Was hat die Diffusionsgleichung mit der Abfallwirtschaft zu tun?

2. Schreiben sie die Diffusionsgleichung (4.1) zweidimensional um.

3. Ordnen sie Folgendes zu: lineare / nichtlineare Gleichungen und diffusiver Stofftransport, Gerinnehydraulik, Grundwasserhydraulik.

4. Welche Verfahrens-Schritte benötigen wir zur numerischen Lösung einer PDE (siehe z.B. Übung E7)?

5. Schreiben sie das explizite FTCS-Schemata für die eindimensionale lineare Diffusionsgleichung für den Gitterpunkt i zum Zeitpunkt j.

6. Schreiben sie das implizite FTCS-Schemata für die eindimensionale lineare Diffusionsgleichung für den Gitterpunkt i zum Zeitpunkt j.

7. Welche Datenstrukturen benötigen wir für explizite und implizite Verfahren bezüglich des Lösungsvektors $\mathbf{u}$?

8. Wovon hängt die Neumann-Zahl ab? Ortsdiskretisierung, Zeitdiskretisierung oder beides.

9. Spielt der physiko-chemische Diffusionkoeffizient eine Rolle für die Stabilität eines impliziten Differenzen-Verfahrens?

10. Wie können wir Speicher für Vektoren bereitstellen und wieder freigeben. Schreiben sie die Instruktionen für den Knoten-Vektor u_i.

11. Was passiert, wenn Speicher z.B. für Vektoren in Anspruch genommen wird, der vorher nicht reserviert wurde?

12. Wie können Randbedingungen erster Art in ein Gleichungssystem eingebaut werden?

13. Wie können Randbedingungen zweiter Art in ein Gleichungssystem eingebaut werden?

14. Wie funktioniert das Gauss'sche Eliminierungsverfahren zur Lösung eines linearen Gleichungssystems?

15. Wie funktioniert das Gauss-Seidel'sche Verfahren zur Lösung eines linearen Gleichungssystems?

16. Warum erhalten wir leicht unterschiedliche Ergebnisse bei expliziten und impliziten Verfahren z.B. zur Lösung der Diffusionsgleichung?

17. Aus welchen Grundprinzipien können wir die Gerinne-Hydraulik-Gleichung (Wasser-Massenbilanz) herleiten?

18. Welches physikalische Grundprinzip repräsentiert die Bernoulli-Gleichung?

19. Wie ist der hydraulische Radius definiert, für eine Rohr- und eine Gerinne-Strömung?

20. Wie ist der benetzte Umfang definiert, für eine Rohr- und eine Gerinne-Strömung?

21. Was sind Streckenverluste bei einer Gerinne-Strömung?

22. Welche empirischen Gleichungen gibt es für die Beschreibung von Streckenverlusten bei Gerinne-Strömungen?

23. Welche physikalischen Prozesse dominieren die Streckenverluste bei Gerinne-Strömungen?

24. Was ist das Sohlgefälle bei Gerinne-Strömungen? Welche physikalischen Prozesse dominieren das Sohl-Gefälle?

25. $\rho g h$ ist gleich?

26. Was sagt ihnen die Zahl 86400?

27. Welche physikalische Bedeutung hat das Funktional für das Newton-Verfahren zur Lösung der Gerinne-Hydraulik-Gleichung?

28. Wie ist die Porosität eines porösen Mediums definiert?
29. Welche physikalischen Annahmen müssen wir treffem um von Gleichung (4.57) zu (4.58) kommen
30. Was besagt das Darcy-Gesetz für die Grundwasser-Hydraulik, welche physikalischen Prozesse sind hierfür relevant?
31. Was ist der Unterschied zwischen Filter- und Abstandsgeschwindigkeit?
32. Schreiben sie die Grundwasser-Gleichung in zwei Dimensionen, d.h. für einen Vertikalschnitt in $x-z$ Koordinaten. Benutzen sie beides die Vektor- und Index-Schreibweise.
33. Was sind die wesentlichen hydraulischen Komponenten eines Grundwassersystems?
34. Übersetzen sie die Randbedingungstypen Grundwasserstand, Einzugsgebietsgrenze, Wasserscheide in mathematische Audrücke.
35. Schreiben sie die physikalischen Einheiten für die Parameter: hydraulische Durchlässigkeit, Speicherkoeffizient, Transmissivität (SI Einheiten).
36. Welche Beiträge benötigen wir für die Erstellung der Wasserbilanz für den Knoten 6 (siehe Abb. 4.22)?
37. Source Code erklären

6.4 C++ und Qt

1. Ordnen sie zu: C++, Qt, Objekt-Orientierung, GUI, Dialog, Windows.
2. Welche Software ist frei verfügbar: Qt, Visual-Studio-Express, cygwin, gnu?
3. Potenzen: Schreiben sie die Anweisung $y = x^4$ in C++. Benötigen wir zusätzliche Inkludes?
4. Funktionen: Schreiben sie die Anweisung $y = \sin(\pi x^2)$ in C++. Benötigen wir zusätzliche Inkludes?
5. Funktionen: Schreiben sie die Anweisung $y = \exp^{-\lambda t}$ in C++. Benötigen wir zusätzliche Inkludes?
6. Wie können wir in Qt QString nach double konvertieren?
7. Wie können wir in Qt integer nach double konvertieren?
8. Erlären sie kurz das Signal-Slot Konzept von Qt?
   `connect(pushButton,SIGNAL(clicked()),this,SLOT(on_pushButton)));`

9. Wofür steht die Klasse QPushButton in Qt?
10. Wofür steht die Klasse QEdit in Qt?
11. Wofür steht die Klasse QLabel in Qt?
12. Mit welcher Klasse können wir in Qt Grafiken in Dialoge einbauen?
13. Schreiben sie schematisch den Quelltext für folgendes Dialog-Layout.
14. Ordnen sie Folgendes zu:
    `*plotter,&plotter,plotter.SetWindowsTitle(),plotter->SetWindowsTitle().`
15. Was bewirkt die Anweisung `tr("Text")`bei Zeichenketten?
16. Was bewirkt die Instruktion `qrand()`?Ist diese Anweisung eine Standard C++ Funktion?

Literaturverzeichnis

[1] Stroustrup B. *The programming languages C++*. Addison-Wesley, Reading, 1991.

[2] K.-F. Busch, L. Luckner, and K. Tiemer. *Geohydraulik*. Borntraeger, Berlin-Stuttgart, 1993.

[3] D. Gross, W. Hauger, and P. Wriggers. *Formeln und Aufgaben zur Technischen Mechanik*. Springer-Lehrbuch, 2009.

[4] Blanchette J and Summerfield M. *C++ GUI Programming with Qt 4*. Prentice Hall, Boston, 2006.

[5] Delfs J-O, Kalbus E, Park C-H, and Kolditz O. Ein physikalisch basiertes Modellkonzept zur Transportmodellierung in gekoppelten Hydrosystemen. *Grundwasser*, 14(3):219–230, 2009.

[6] W. Kinzelbach. *Numerische Methoden zur Modellierung des Transports von Schadstoffen im Grundwasser*. Oldenburg Verlag, München, 1992.

[7] H. Martin and R. Pohl. *Technische Hydromechanik*. Verlag Bauwesen, 2000.

[8] Kolditz O. *Computational methods in environmental fluid mechanics*. Springer, Berlin-Heidelberg, 2002.

[9] Kolditz O and Kolditz B. *Hydroinformatik I: C++ und Visual C++*. UFZ / TUD, Leipzig / Dresden, 2009.

[10] J.N. Paine. Open-channel flow algorithmen in newton form. *Journal of Irrigation and Drainage Engineering*, 118(2):306, 1992.

[11] H. Shao, S.V. Dmytrieva, O. Kolditz, D.A. Kulik, W. Pfingsten, and G. Kosakowski. Modeling reactive transport in non-ideal aqueous-solid solution system. *Applied Geochemistry*, 24:1287–1300, 2009.

[12] J. Stribny. *Ohne Panik - Strömungsmechanik*. Oldenburg Verlag, München, 2002.

[13] Breymann U. *C++ Einführung und professionelle Programmierung*. Hanser, München-Wien, 2001.

[14] Kirch-Prinz U and Prinz P. *C++ Lernen und professionell anwenden*. mitp, Heidelberg, 2007.

[15] Wu Y, Mathias Toll, Wenqing Wang, Thomas Kalbacher, Martin Sauter, and Olaf Kolditz. Development of a groundwater flow model in the wadi kafrein area. *Environmental Earth Science*, submitted, bei mir erhältlich.

Inhaltsverzeichnis