BHYWI-08: Semester-Fahrplan 2019

Vorlesungen

Datum	V	Thema
05.04.2019	01	IT: GitHub / Qt Installation
12.04.2019	02	Grundlagen: Kontinuumsmechanik
19.04.2019	--	Ostern
26.04.2019	03	Grundlagen: Hydromechanik
03.05.2019	04	Grundlagen: Partielle Differentialgleichungen
10.05.2019	05	Grundlagen: Numerik, Qt Übung: Funktionsrechner
17.05.2019	06	Numerik: Finite Differenzen Methode I (explizit)
24.05.2019	07	Numerik: Finite Differenzen Methode II (implizit)
31.05.2019	08	Gerinnehydraulik: Theorie – Grundlagen
07.06.2019	09	Gerinnehydraulik: Programmierung, Übung
14.06.2019		Pfingsten
21.06.2019	10	Grundwassermodellierung: Catchment Übung
28.06.2019	11	Grundwassermodellierung: Datenbasierte Methoden I
05.07.2019	12	Grundwassermodellierung: Datenbasierte Methoden II
12.07.2019	13	Beleg

TECHNISCHE UNIVERSITÄT DRESDEN

Hydroinformatik II
"Prozesssimulation und Systemanalyse"
BHYWI-08-05 @ 2019
Übung Funktionsrechner E2-for-python

Olaf Kolditz

*Helmholtz Centre for Environmental Research – UFZ

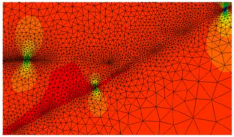
[1]Technische Universität Dresden – TUDD

[2]Centre for Advanced Water Research – CAWR

17.05.2019 - Dresden

Konzept

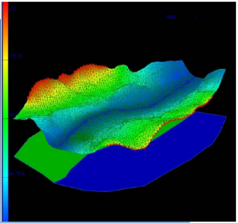
$$\frac{d\psi}{dt} = \frac{\partial\psi}{\partial t} + \mathbf{v}^E \nabla\psi$$



Basics

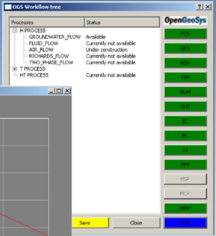
Mechanik

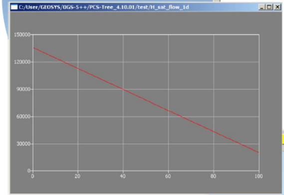
Anwendung



Numerische

Methoden





Programmierung

Visual C++

Prozessverständnis

Fahrplan für heute

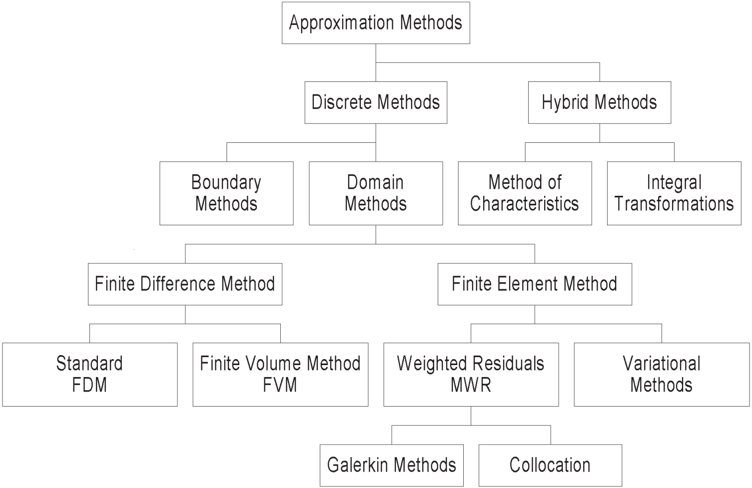
- Nichtlineare Löser (Abschluss letzte Vorlesung)

- Übung: BHYWI-08-02-E: Funktionsrechner
- Ergebnisse
- Qt console
- Python
- BHYWI-08-02-E-for-python

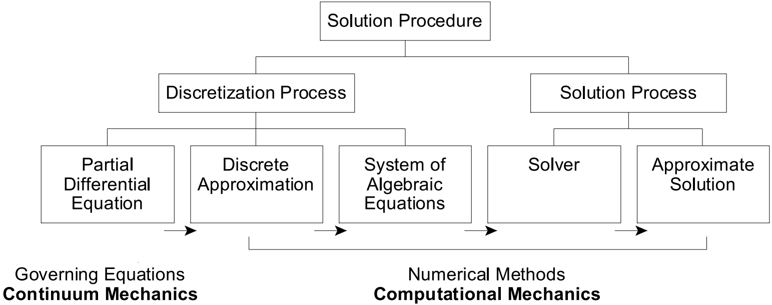
- Finite-Differenzen-Methode (FDM) (Neue Vorlesung)

TECHNISCHE UNIVERSITÄT DRESDEN

Näherungsverfahren



Lösungsverfahren



Lösen nichtlinearer Gleichungen

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

- Picard method (fixpoint iteration)
- Newton methods
- 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

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

$$\mathbf{A}(\mathbf{x})\,\mathbf{x} - \mathbf{b}(\mathbf{x}) = 0 \tag{1}$$

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.

$$\begin{aligned} \mathbf{A}(\mathbf{x}_{k-1})\,\mathbf{x}_k - \mathbf{b}(\mathbf{x}_{k-1}) &= 0 \\ \mathbf{x}_k &= \mathbf{A}^{-1}(\mathbf{x}_{k-1})\,\mathbf{b}(\mathbf{x}_{k-1}) \end{aligned} \tag{2}$$

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.

$$\varepsilon \geq \frac{\| \mathbf{x}_k - \mathbf{x}_{k-1} \|}{\| \mathbf{x}_k \|} \tag{3}$$

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. 1. To achieve convergence of the scheme it has to be guaranteed that the iteration error

$$e_k = \| \mathbf{x}_k - \mathbf{x} \| < C \| \mathbf{x}_{k-1} - \mathbf{x} \|^p = e_{k-1} \tag{4}$$

or, alternatively, the distance between successive solutions will reduce

$$\| \mathbf{x}_{k+1} - \mathbf{x}_k \| < \| \mathbf{x}_k - \mathbf{x}_{k-1} \|^p \tag{5}$$

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.

Lösen nichtlinearer Gleichungen

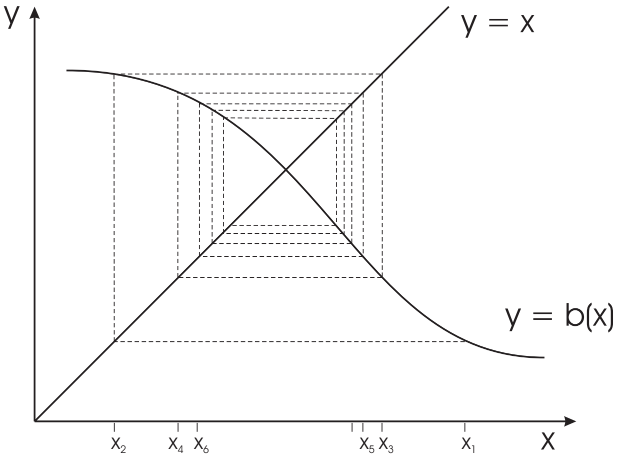


Figure: Graphical illustration of the Picard iteration method

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

$$\mathbf{R}(\mathbf{x}) = \mathbf{A}(\mathbf{x})\,\mathbf{x} - \mathbf{b}(\mathbf{x}) = 0 \tag{6}$$

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

$$\mathbf{R}_{k+1} = \mathbf{R}_k + \left[\frac{\partial \mathbf{R}}{\partial \mathbf{x}}\right]_k \Delta \mathbf{x}_{k+1} + 0(\Delta \mathbf{x}_{k+1}^2) \tag{7}$$

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 (7).

$$\Delta\mathbf{x}_{k+1} = -\mathbf{J}_k^{-1}\,\mathbf{R}_k \tag{8}$$

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.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_{k+1} \tag{9}$$

Lösen nichtlinearer Gleichungen - Newton-Verfahren

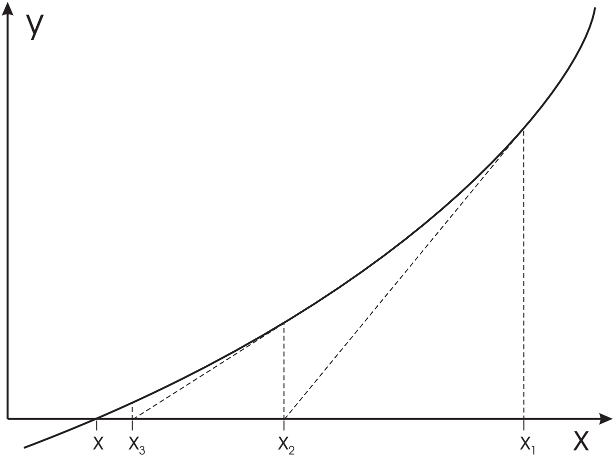


Figure: Graphical illustration of the Newton-Raphson iteration method

TECHNISCHE UNIVERSITÄT DRESDEN

Once an initial guess is provided, successive solutions of $\mathbf{x}_{k+1}$ can be determined using equations (8) and (9) (Fig. 2). The Jacobian has to re-evaluated and inversed at every iteration step, which is a very time-consuming procedure 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. 3).
The convergence velocity of the Newton-Raphson method is second-order. It is characterized by the expression.

$$\| \mathbf{x}_{k+1} - \mathbf{x} \| \leq C \, \| \mathbf{x}_k - \mathbf{x} \|^2 \tag{10}$$

Lösen nichtlinearer Gleichungen - Newton-Verfahren

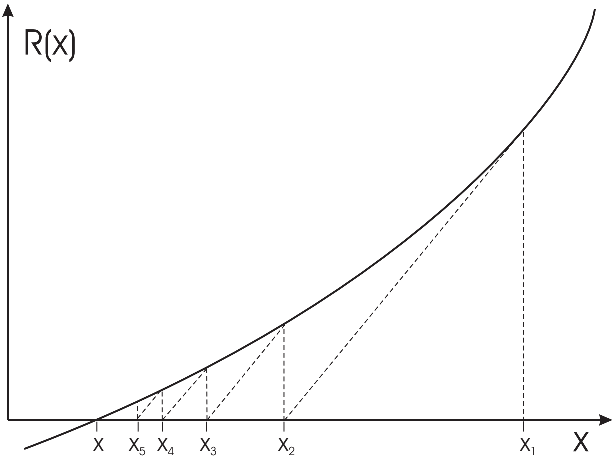
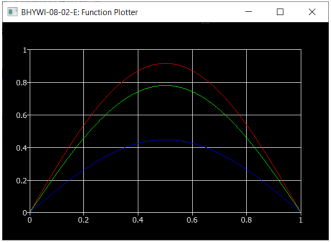


Figure: Graphical illustration of the modified Newton-Raphson iteration method

Übung: BHYWI-08-02-E: Funktionsrechner

Übung: BHYWI-08-02-E: Funktionsrechner



Parabolic equation

$$\frac{\partial \psi}{\partial t} = \alpha \frac{\partial^2 \psi}{\partial x^2} \tag{11}$$

$$\psi(t, x) = sin(\pi x) exp(-\alpha \pi^2 t) \tag{12}$$

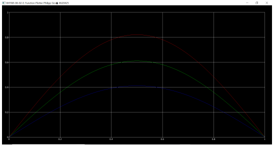
$$\psi(t, x) = sin(\sqrt{\pi \alpha} x) exp(-\pi t) \tag{13}$$

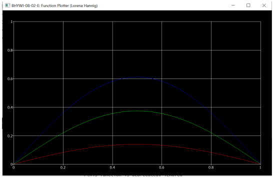
$$\psi(t, x) = sin(\pi / \sqrt{\alpha} x) exp(-\pi^2 t) \tag{14}$$

TECHNISCHE UNIVERSITÄT DRESDEN

Hausaufgabe: BHYWI-08-02-E: Funktionsrechner

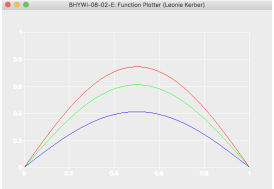
Ergebnisse (8): Betreff: HYDROINFORMATIK

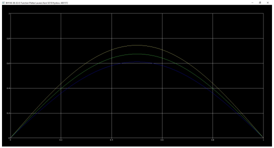


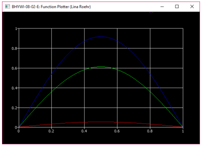


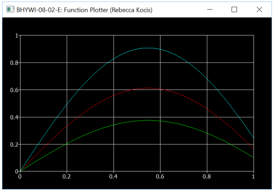
t= 0,2 für rote Kurve
t=0,1 für grüne Kurve
t=0,05 für blaue Kurve

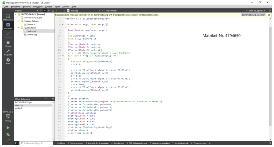
Lorena Hannig, Hydroinformatik II

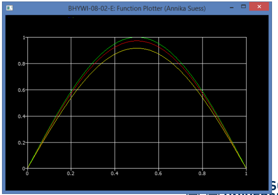












Entwicklungsumgebungen

- Compiler: g++ (console), CLI - command line interface), cyqwin, mingw, ...
- IDE: Qt, ...(GUI - graphical user interface), "all-in-one", ...
- Scriptsprachen: Python, ... (flexibility)

TECHNISCHE UNIVERSITÄT DRESDEN

Hausaufgabe: BHYWI-08-02-E: Funktionsrechner

ohne Qt ... (Quelltext auf der Lehre-Webseite)

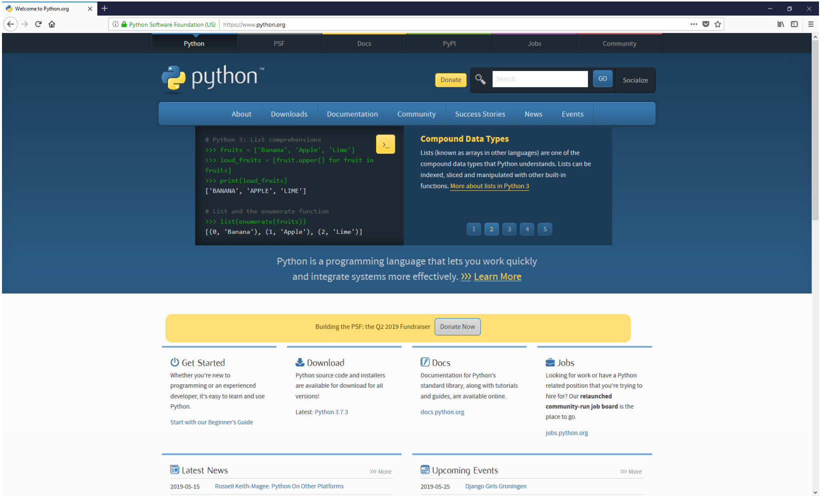
```
#include <cmath>
#include <fstream>
#define PI 3.14159265358979323846

int main(int argc, char *argv[])
{
  //1-Definitionen
  int numPoints = 1000;
  double x,y,alpha=1.,t=0.01;
  std::ofstream out_file;
  out_file.open("out.csv");
  //2-Berechnung
  //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);
    //y=sin(sqrt(PI*alpha)*x) * exp(-PI*t);
    //y=sin(PI/sqrt(alpha)*x) * exp(-PI*PI*t);
    y=sin(PI*x) * exp(-alpha*PI*PI*t);
    out_file << x << "," << y << std::endl;
  }
  //3-Ausgabe
}
//HW1 Lösung: y=sin(sqrt(pi*alpha)*x) * exp(-pi*t) plotten
//HW2 Lösung: y=sin(pi/sqrt(alpha)*x) * exp(-pi*pi*t) plotten
//HW3 Lösung: y=sin(pi*x) * exp(-alpha*pi*pi*t) plotten
//HW4 Lösungen vergleichen
//HW5 Verschiedene Zeiten (aus Eingabedatei) lesen und rechnen
```

TECHNISCHE UNIVERSITÄT DRESDEN

Qt von der Konsole

```
compiler console
go to directory
qmake -project
qmake
mingw32-make
```

Python



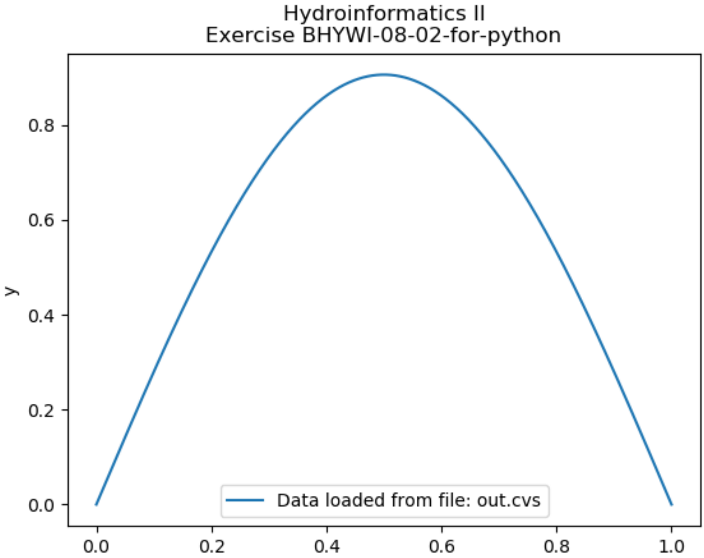
TECHNISCHE UNIVERSITÄT DRESDEN

Python

```
import matplotlib.pyplot as plt
import csv
x = []
y = []
with open('out.csv','r') as csvfile:
    plots = csv.reader(csvfile, delimiter=',')
    for row in plots:
        x.append(float(row[0]))
        y.append(float(row[1]))
plt.plot(x,y, label='Data loaded from file: out.csv')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Hydroinformatics II\nExercise BHYWI-08-02-for-py
plt.legend()
plt.savefig("test1.png")
plt.show()
```

Hausaufgabe: BHYWI-08-02-E: Funktionsrechner

mit Python ...



ECHNISCHE NIVERSITÄT RESDEN

Batch

```
echo Compilation
g++ main.cpp
echo Execution
a.exe
echo Ploting
data_from_file.py
echo End
```

BHYWI-08: Semester-Fahrplan

Übungen

Datum	E	Übungen
05.04.2019	00	Git und QT (Lars Bilke)
03.05.2019	01	Qt: Hallo World
10.05.2019 17.05.2019	02	Qt: Funktionsrechner
	03	Qt: Explizite Finite-Differenzen-Methode
	04	Qt: Implizite Finite-Differenzen-Methode
	05	Qt: Gerinnehydraulik I (QAD)
	06	Qt: Gerinnehydraulik II (OOP)
	08	Qt: Gerinnehydraulik IV (interaktiv)
		...

`https://github.com/envinf/Hydroinformatik-II`

TECHNISCHE UNIVERSITÄT DRESDEN