Hydroinformatik II: Gerinnehydraulik

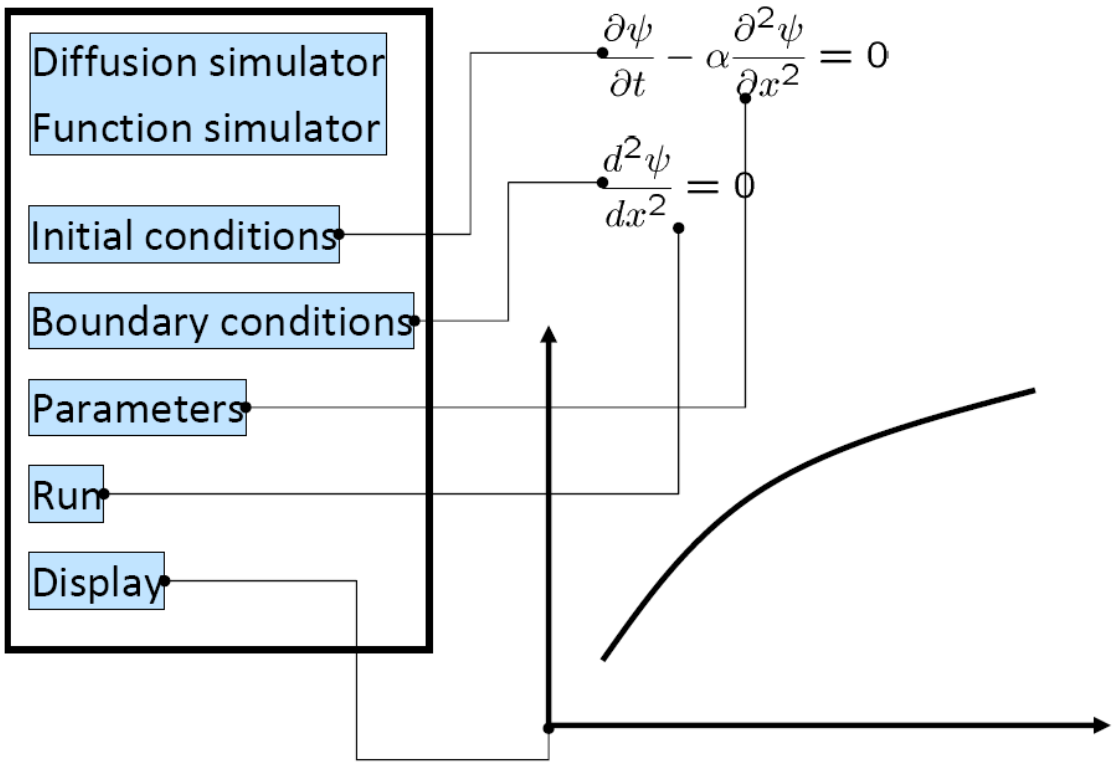
[1]Helmholtz Centre for Environmental Research – UFZ, Leipzig
[2]Technische Universität Dresden – TUD, Dresden

Dresden, 10. Juli 2015

Vorlesungsplan Hydroinformatik II SoSe 2015

#	Datum	Thema
01	17.04.2015	Einführung, Grundlagen: Kontinuumsmechanik
02	24.04.2015	Grundlagen: Kontinuumsmechanik/Hydromechanik
-	01.05.2015	Maifeiertag
03	08.05.2015	HW: Einführung in Qt (Installation)
04	15.05.2015	Grundlagen: Partielle Differentialgleichungen / TEX
05	22.05.2015	Grundlagen: Numerische Methoden
-	29.05.2015	Pfingsten
06	05.06.2016	Numerik: (exp) Finite Differenzen Methode
07	12.06.2015	Numerik: (imp) Finite Differenzen Methode
08	19.06.2015	Gerinnehydraulik: Theorie - Grundlagen
09	26.06.2015	Gerinnehydraulik: Programmierung, Übung 1
10	03.07.2015	Gerinnehydraulik: Programmierung, Übung 2
11	10.07.2015	Gerinnehydraulik: Programmierung, Übung 3
12	17.07.2015	Kurs-Zusammenfassung und Ausgabe Beleg

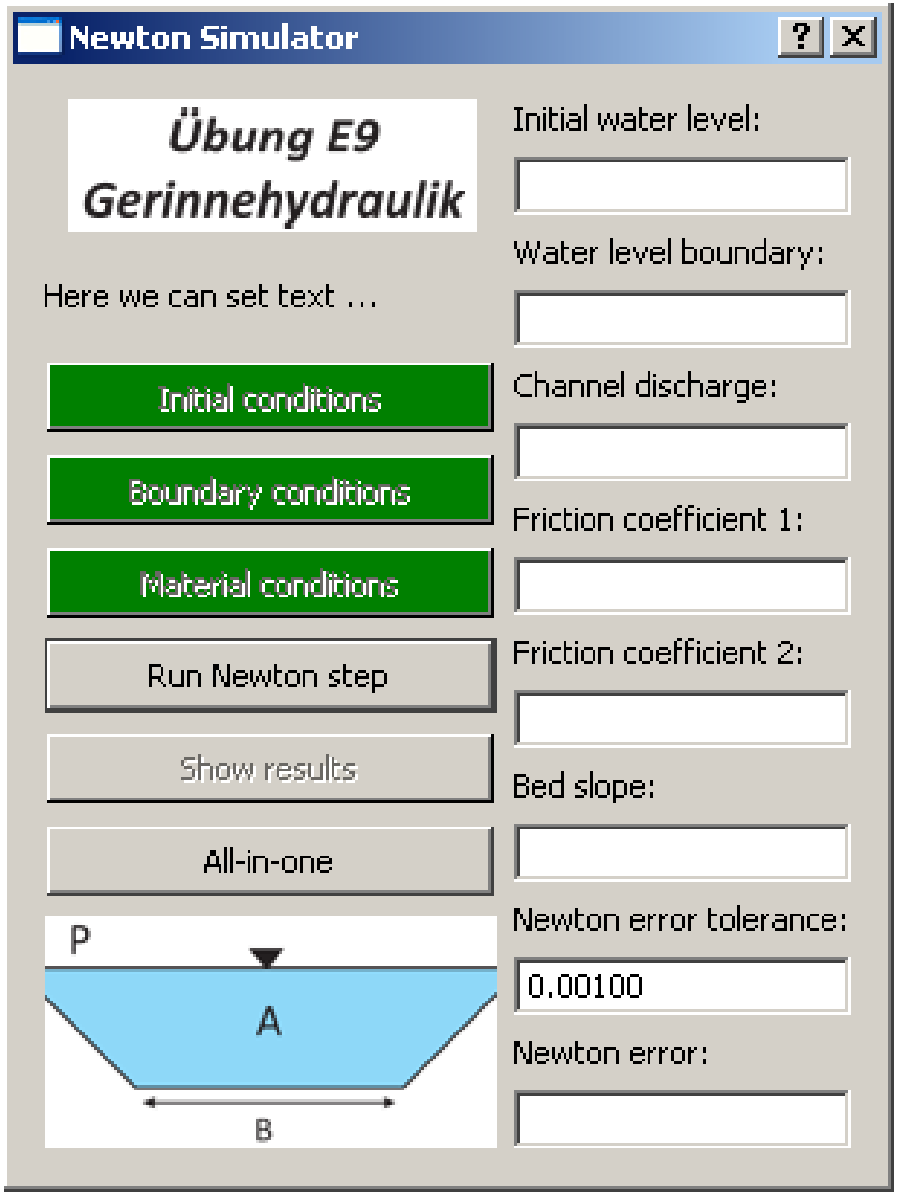
0 - Zur Erinnerung: ... Konzept und Fazit

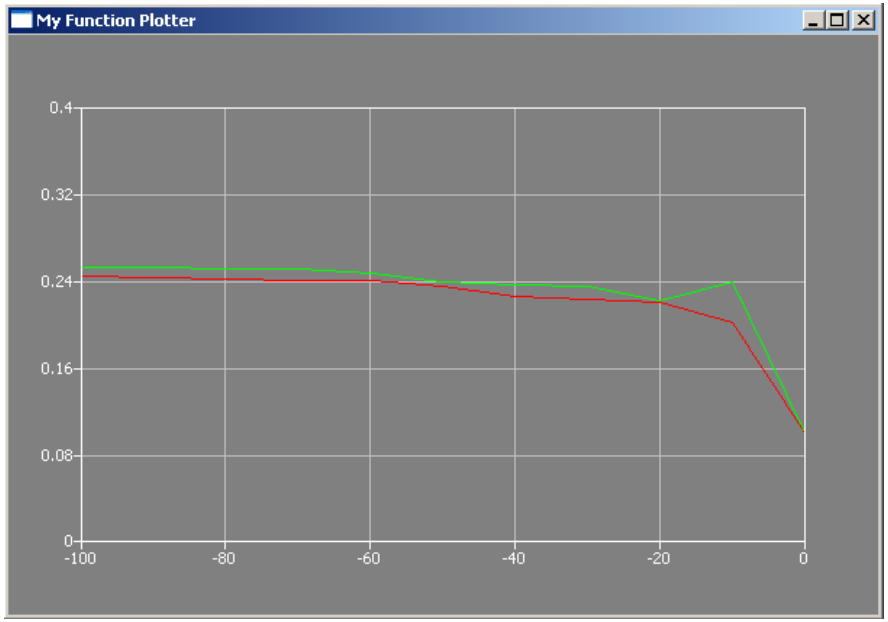


Eigenes MatLab ...

- Funktions-Simulator FDM Simulator (explizit und implizit)
- Newton Simulator
- ... alles noch 1D, schau'n wir mal (Systemanalyse)

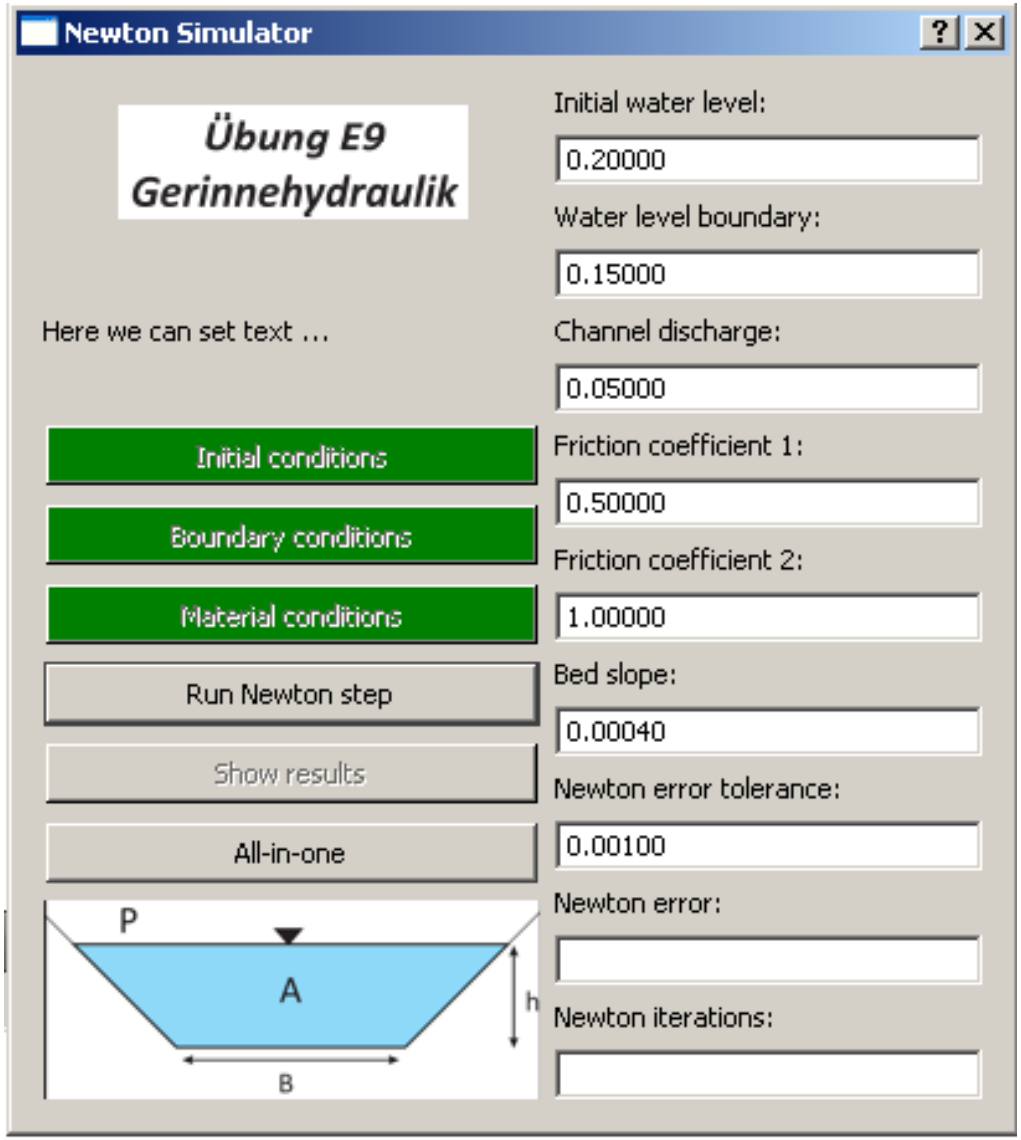
0 - Newton-Schritt Simulator





- **Reminder**:
- Newton-Schritt Simulator
- `QBasics: QLineEdit`
- Plotter (local / global variables)

1 - Parameter anzeigen #1



- Tafelbild:
- Datentransfer
- Beispiel: Anfangsbedingungen

1 - Parameter anzeigen #2

```
void Dialog::on_pushButtonIC_clicked()
{
  // Anfangsbedingungen setzen
  for(int i=0;i<n;i++)
  {
      u_old[i] = ICValue;
  }
  // Daten im Dialog sichtbar machen
  sDummy.setNum(ICValue,'f',5);
  lineEditIC->setText(sDummy);
  // Schnick-Schnack
  pushButtonIC->setStyleSheet("background-color: green");
}
```

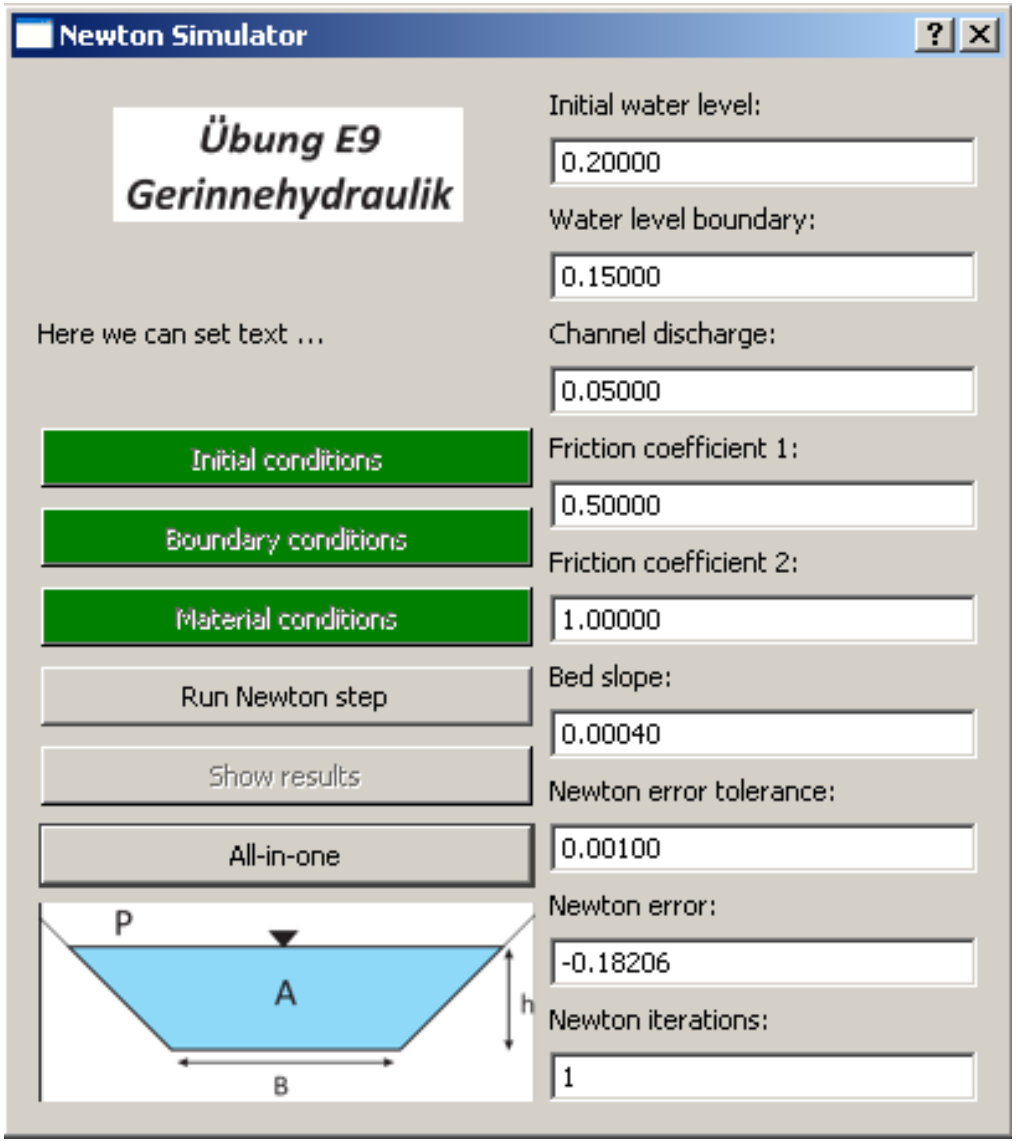
↪ Important: ICValue

1 - Parameter anzeigen #3

```
void Dialog::on_pushButtonBC_clicked()
{...
  sDummy.setNum(BCValue,'f',5);
  lineEditBCR->setText(sDummy);
}

void Dialog::on_pushButtonMAT_clicked()
{
  sDummy.setNum(friction_law_exponent,'f',5);
  lineEditFrictionLawExponent->setText(sDummy);
  sDummy.setNum(m,'f',5);
  lineEditFrictionCoefficient->setText(sDummy);
  sDummy.setNum(bed_slope,'f',5);
  lineEditBedSlope->setText(sDummy);
  sDummy.setNum(discharge,'f',5);
  lineEditDischarge->setText(sDummy);
  sDummy.setNum(error_tolerance,'f',5);
  lineEditNewtonTolerance->setText(sDummy);
}
```

2 - Iterationszähler #1



- Wo einbauen?
- `while()` Schleife

2 - Iterationszähler #2

- Implementierung: Welche Funktion(en)?

```
int main(int argc, char *argv[])

public:
    Dialog(QWidget *parent = 0);
    ~Dialog();

private slots:
    void on_pushButtonIC_clicked();
    void on_pushButtonBC_clicked();
    void on_pushButtonMAT_clicked();
    void on_pushButtonRUN_clicked();
    void on_pushButtonALL_clicked();
    double RUN_NewtonStep();
```

2 - Iterationszähler #3

- Implementierung: `while()`

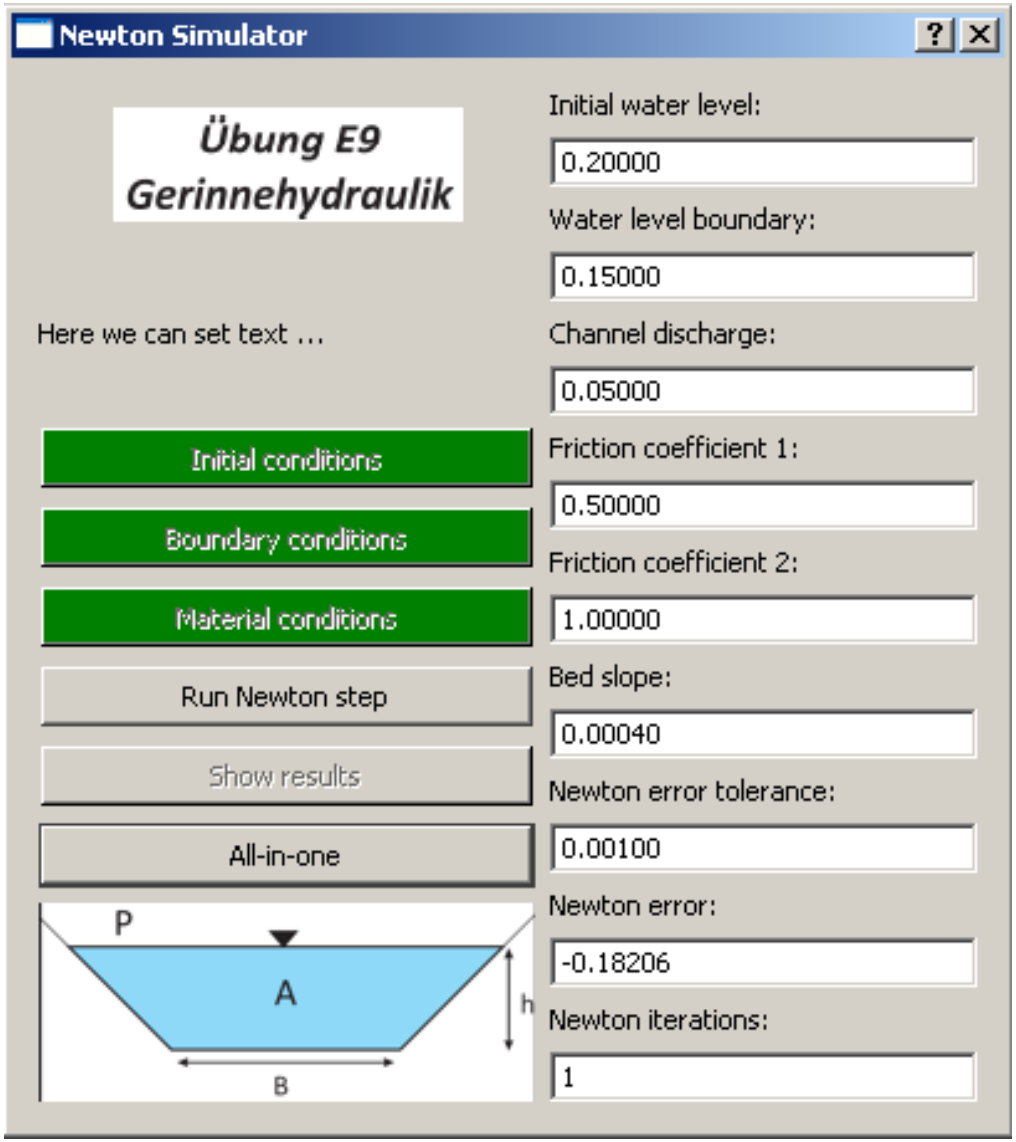
```
void Dialog::on_pushButtonALL_clicked()
{...
  float error = 1.1*error_tolerance;
  while(error>error_tolerance)
  {
    error = RUN_NewtonStep();
  }
}
```

2 - Iterationszähler #4

- Implementierung: Komplett

```
void Dialog::on_pushButtonALL_clicked()
{...
  float error = 1.1*error_tolerance;
  while(error>error_tolerance)
  {
    error = RUN_NewtonStep();
    //Plotter
    QVector<QPointF> points0;
    for(int i=0;i<n-1;i++)
    {
      points0.append(QPointF(x[i],u_new[i]));
    }
    points0.append(QPointF(x[10],u_new[10]));
    plotter->setCurveData(k++, points0);
    plotter->show();
    //Datenübergabe Dialog
    sDummy.setNum(error,'f',5);
    lineEditNewtonError->setText(sDummy);
    sDummy.setNum(k,5);
    lineEditIterations->setText(sDummy);
  }
}
```

3 - Fehlerberechnung #1



- Warum nur eine Iteration?

3 - Fehlerberechnung #2

```
while(error>error_tolerance)
{...}
```

3 - Fehlerberechnung #2

```
while(error>error_tolerance)
{...}
```

$$\varepsilon = \sqrt{\sum_i (h^{k+1} - h^k)^2}$$

```
error = sqrt(error*error);
```

3 - Fehlerberechnung #2

```
while(error>error_tolerance)
{...}
```

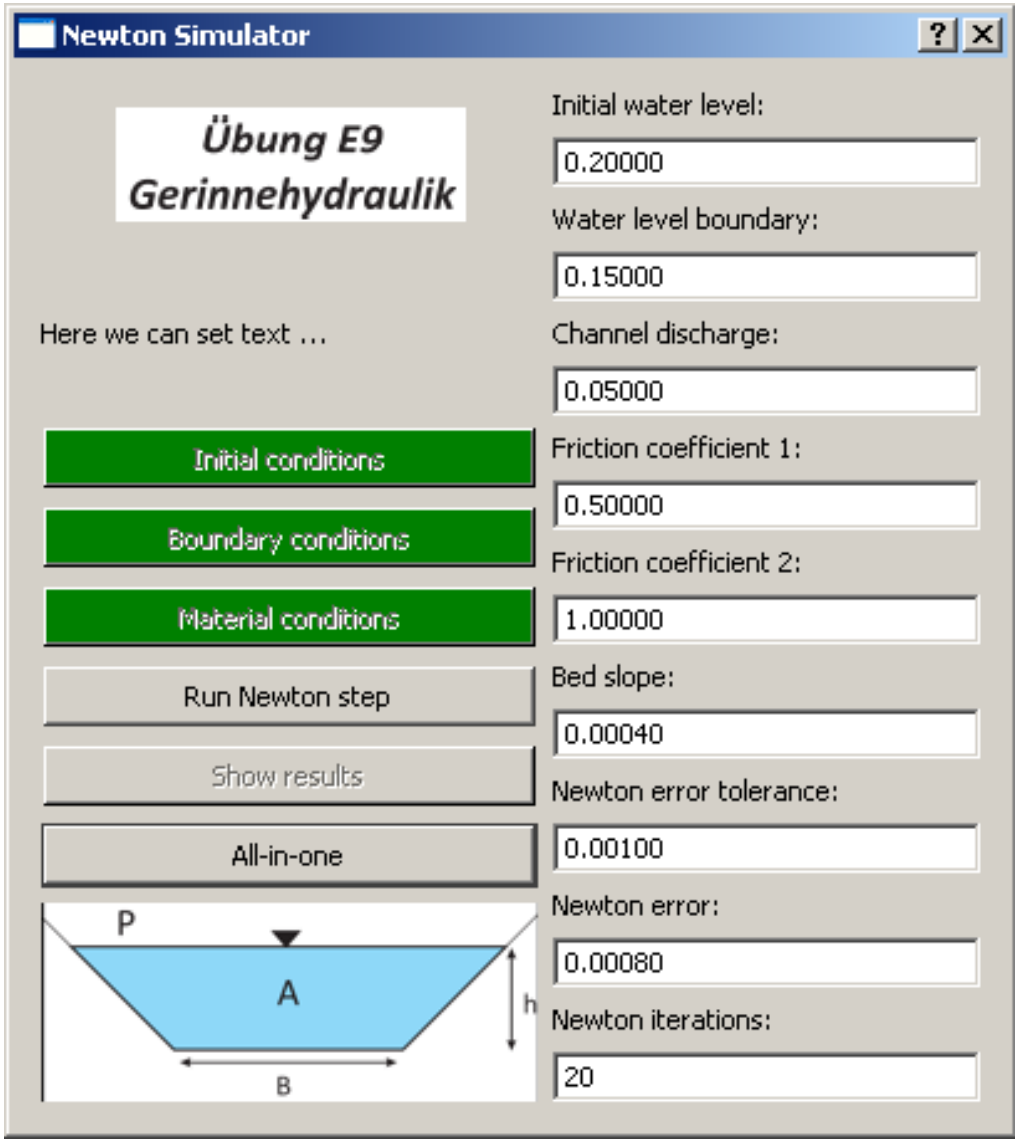
$$\varepsilon = \sqrt{\sum_i (h^{k+1} - h^k)^2}$$

```
error = sqrt(error*error);
```

$$\varepsilon = |h^{k+1} - h^k)|$$

```
error = abs(error);
```

3 - Fehlerberechnung #3

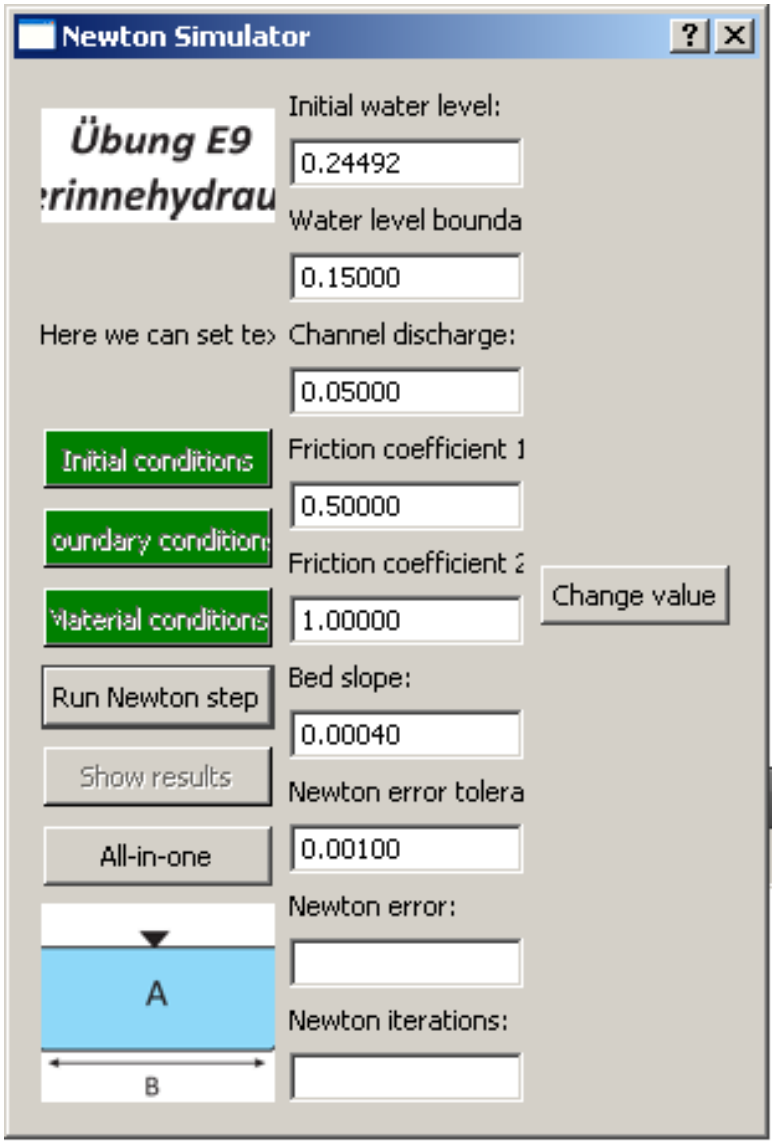


- Jetzt klappts.

4 - Parameter ändern #1

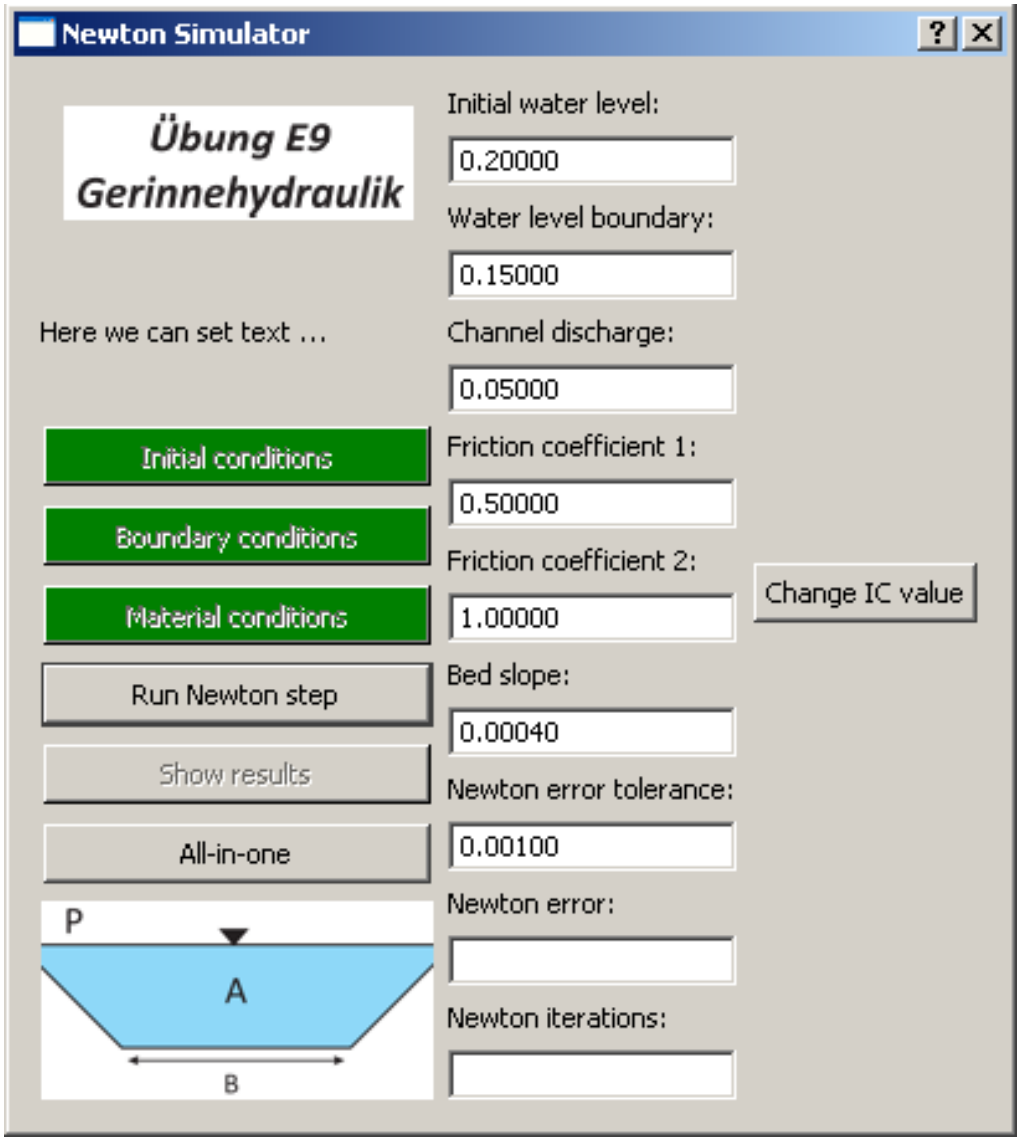
- Erstmal überlegen.
- $\hookrightarrow$ Tafelbild

4 - Parameter ändern #2



- Layout: Blöd gelaufen.

4 - Parameter ändern #2



- Wo ändern?
- `w.setFixedWidth(400)`
- Na also.

QBasics: QLineEdit #0

```
#include <QLineEdit>

//declaration
QLineEdit* lineEditIC;
//definition
lineEditIC = new QLineEdit();
//use
Get- und Set-Funktionen
```

- Warum auf ein mal Unterscheidung zwischen Deklaration und Definition?

QBasics: QLineEdit #1

```
QString ICString = lineEditIC->text()
double ICValue = lineEditIC->text().toDouble()
```

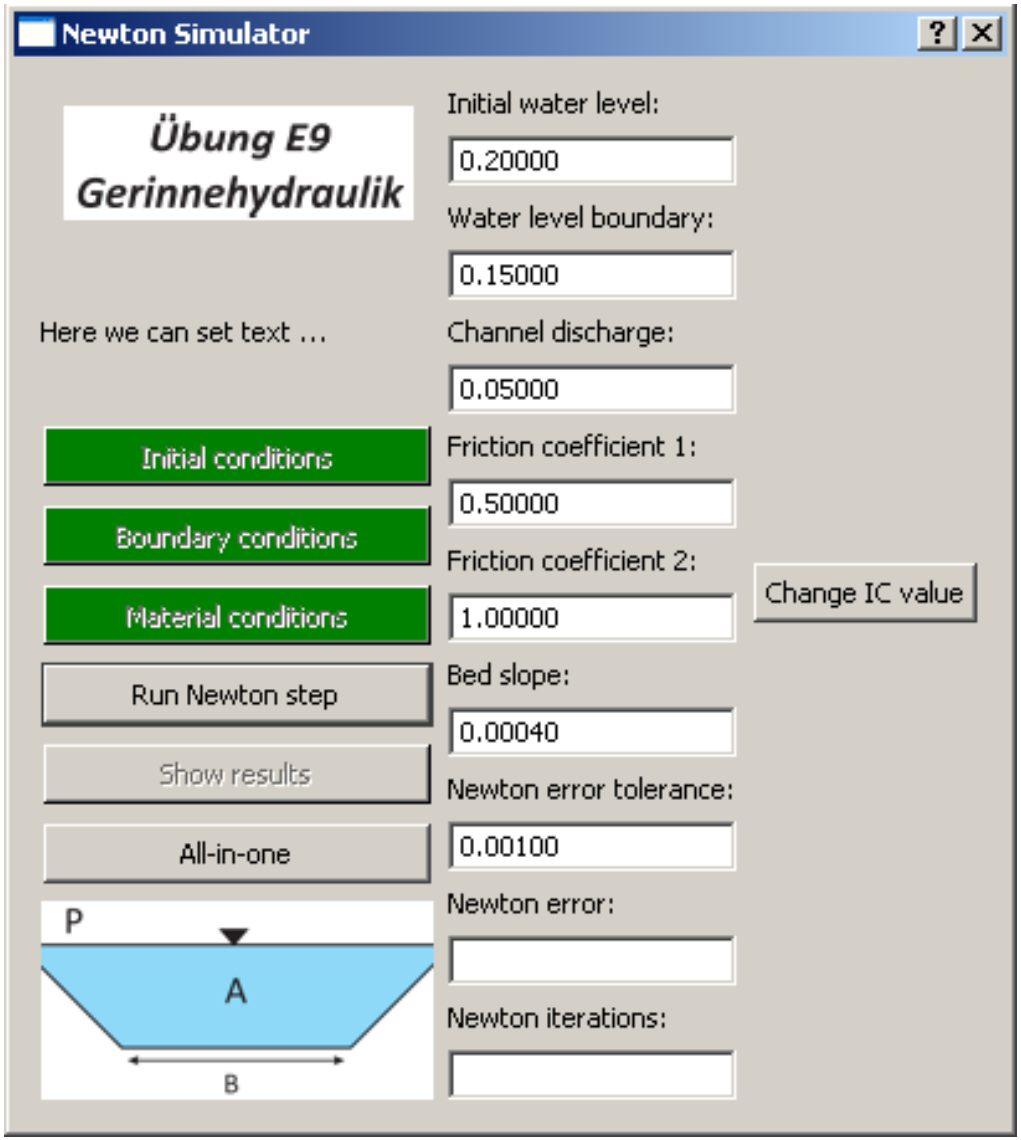
QBasics: QLineEdit #2 Sum

```
#include <QLineEdit>

//declaration
QLineEdit* lineEditIC;
//definition
lineEditIC = new QLineEdit();
//use
 //set
QString ICString;
ICString.setNum(ICValue,'f',5);
lineEditIC->setText(sIC);
 //get
QString ICString = lineEditIC->text()
double ICValue = lineEditIC->text().toDouble()

K
```

5 - Anfangsbedingungen ändern #1 Wie



- Change IC Value

5 - Anfangsbedingungen ändern #2 Programmierung

```
Deklaration:
class Dialog : public QDialog
{...
private slots:
    void on_pushButtonICChange_clicked();
...}

Definition:
void Dialog::on_pushButtonICChange_clicked()
{
  QString sICValue = lineEditIC->text();
  ICValue = lineEditIC->text().toDouble();
  for(int i=0;i<n-1;i++)
  {
      u_old[i] = ICValue;
  }
}
```

5 - Anfangsbedingungen ändern #2 Programmierung

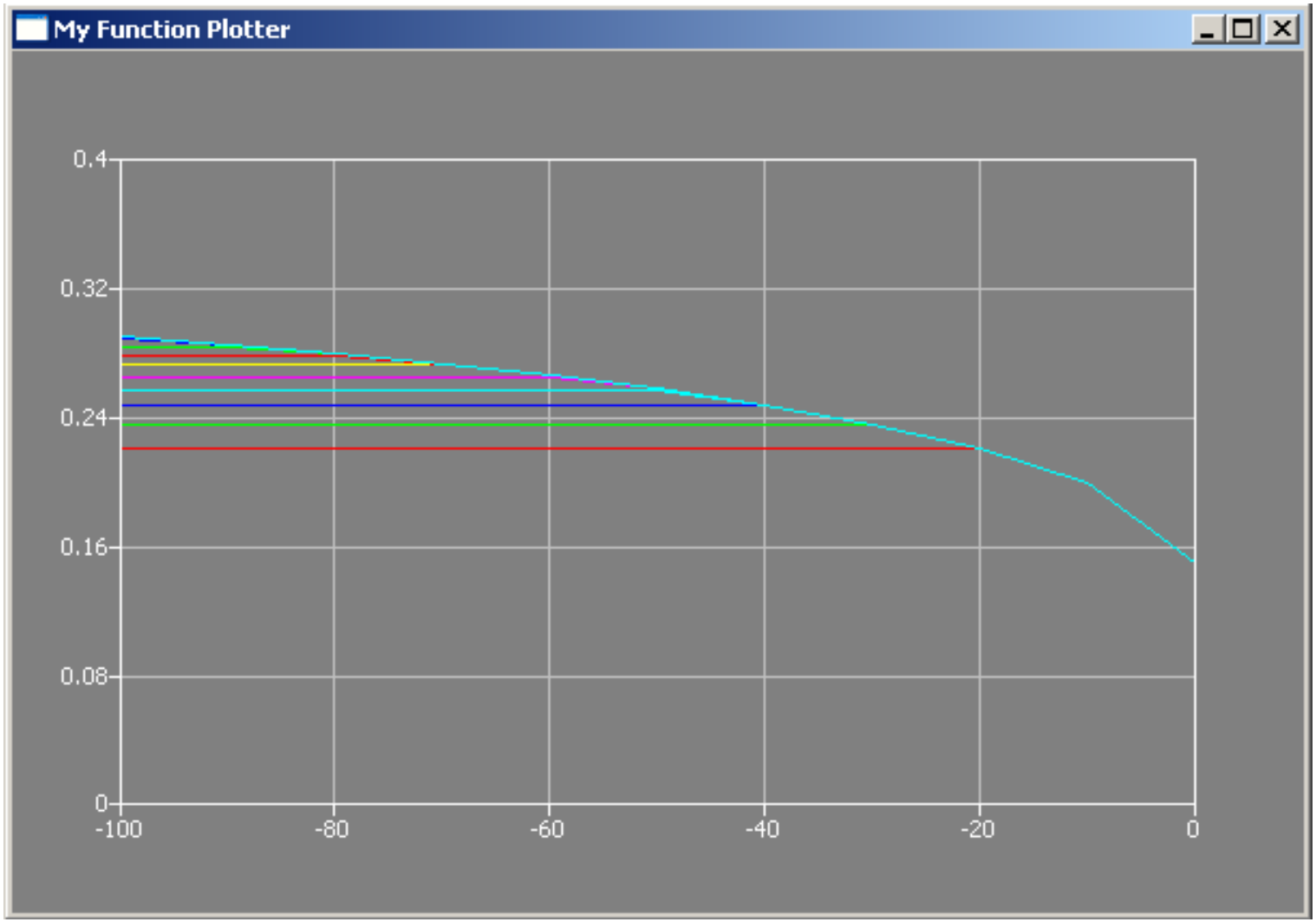
Verknüpfung einer Funktion mit einem Ereignis:

```
connect(pushButtonICChange,
        SIGNAL(clicked()),
        this,
        SLOT(on_pushButtonICChange_clicked()));
```

Typische K-Fragen:

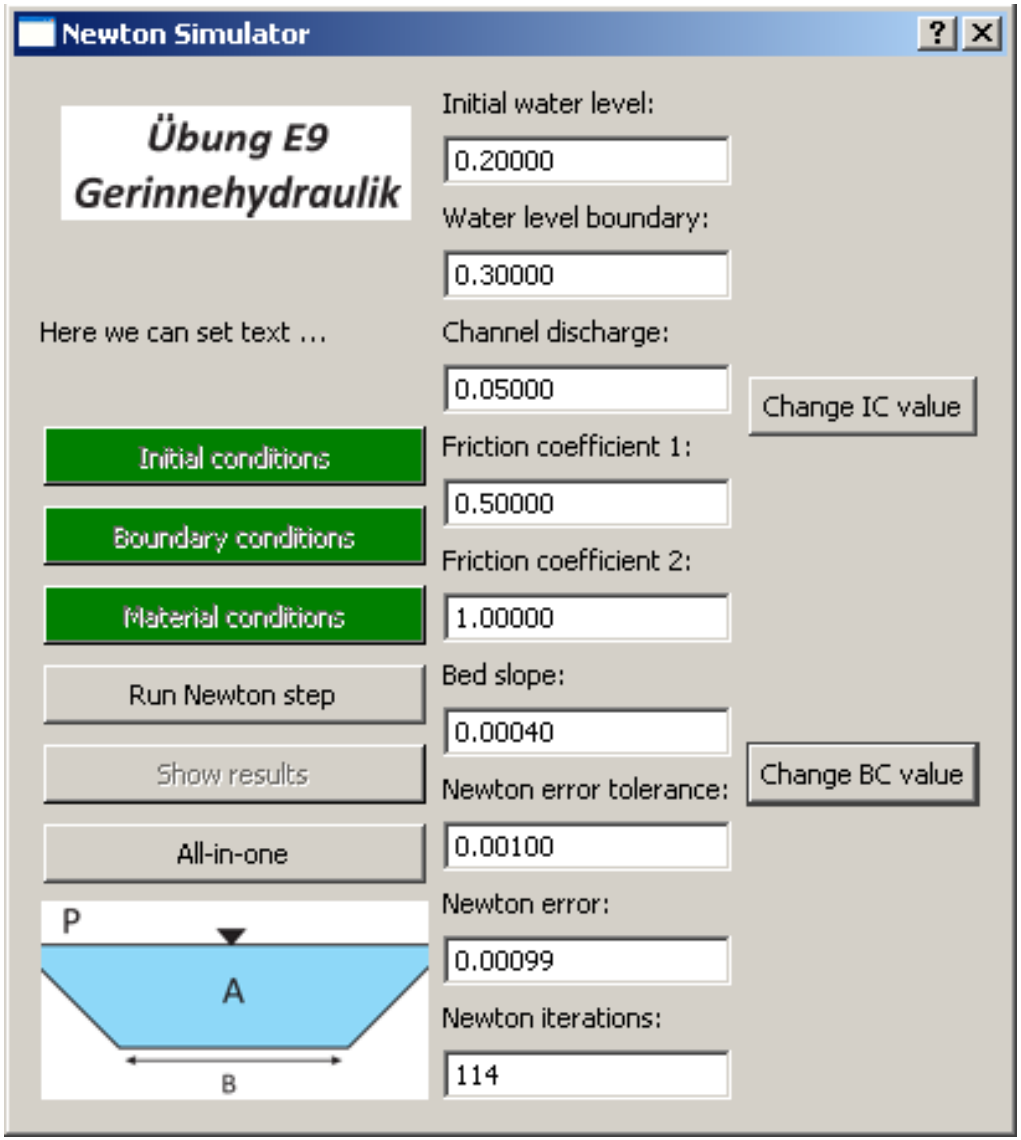
- Wo werden Member-Funktionen einer Klasse deklariert?
- Wo werden Member-Funktionen einer Klasse definiert?
- Schreiben sie die Verknüpfung einer Funktion mit einem Ereignis, wenn eine Schaltfläche gedrückt wurde?

5 - Anfangsbedingungen ändern #3 Analyse



- ↪ Run simulation.

6 - Randbedingungen #1



- the same procedure as ...
- Wenn Zeit: Implemetierung zeigen (step-by-step).

6 - Randbedingungen #2

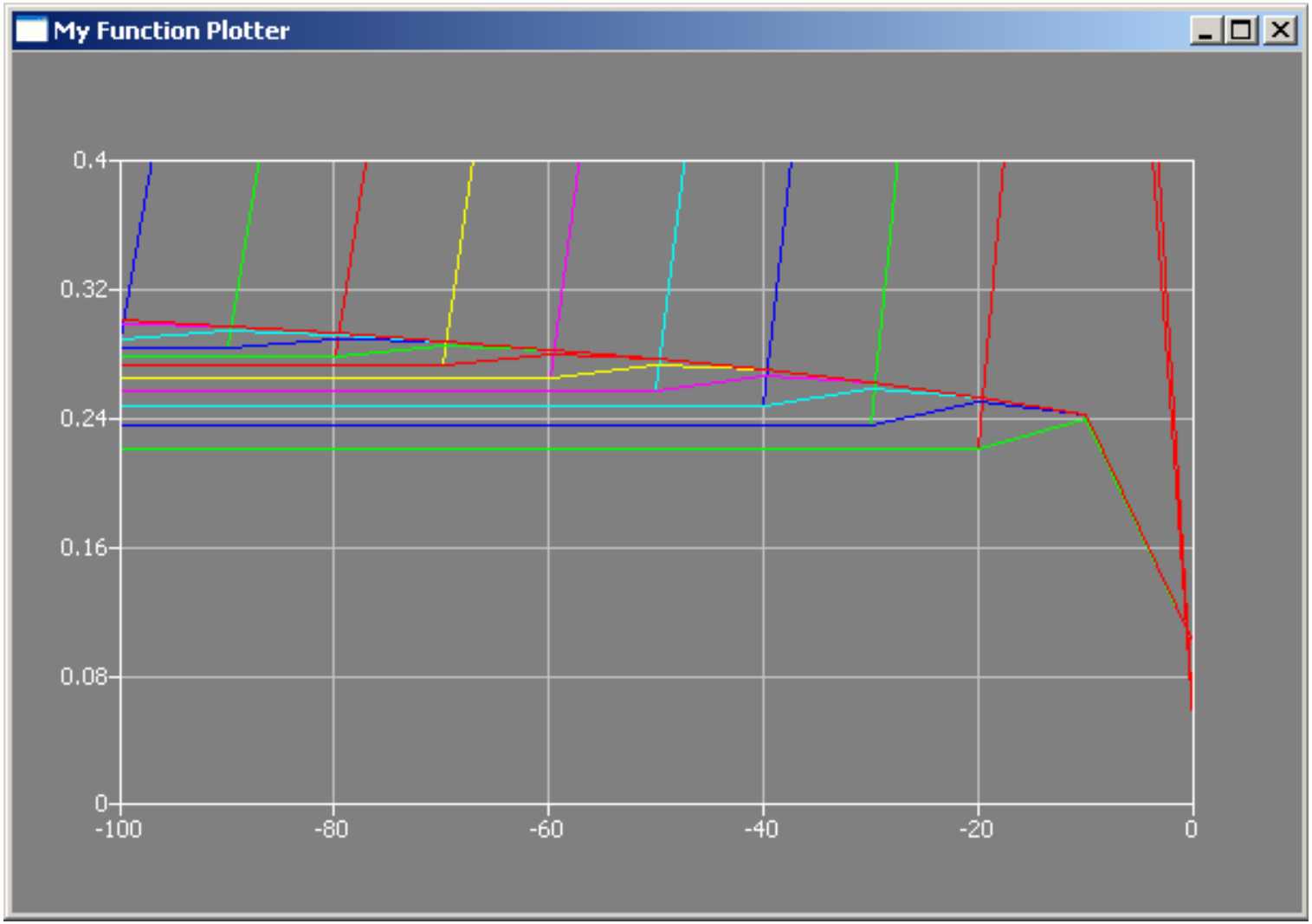
- Advanced OOP

```
void Dialog::on_pushButtonBCChange_clicked()
{
  QString sBCValue = lineEditBCR->text();
  BCValue = lineEditBCR->text().toDouble();
  on_pushButtonBC_clicked();
}
```

QBasics: Funktionen anlegen

1. Funktion deklarieren
2. Funktion definieren (leer) [Komplierbarkeit]
3. Funktion schreiben
4. Dialogelement anlegen [z.B. Schaltfläche]
   - Element deklarieren
   - Element definieren
   - Element ins Layout einfügen
5. Verknüpfung zwischen Funktion und Dialogelement herstellen
6. Funktionalität testen
   - Bei Fehlern: Debuggen
   - 2. (verbesserte) Implementierung
   - Namenskonventionen ...

6 - Randbedingungen #2



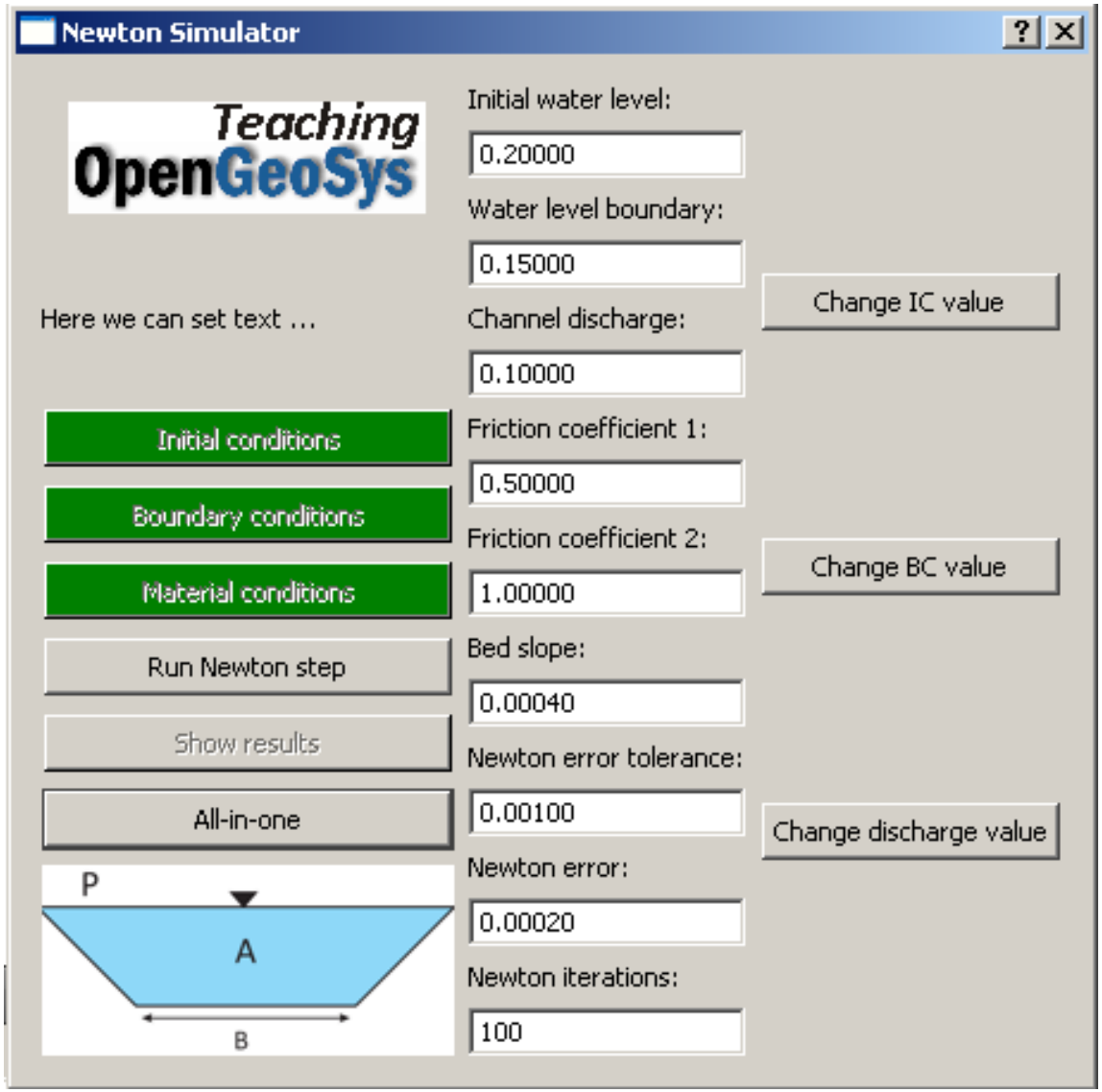
7 - Discharge #1

- Nochmal lokale und globale Variablen ...
- Nochmal, die wichtige Rolle des Konstruktors ...

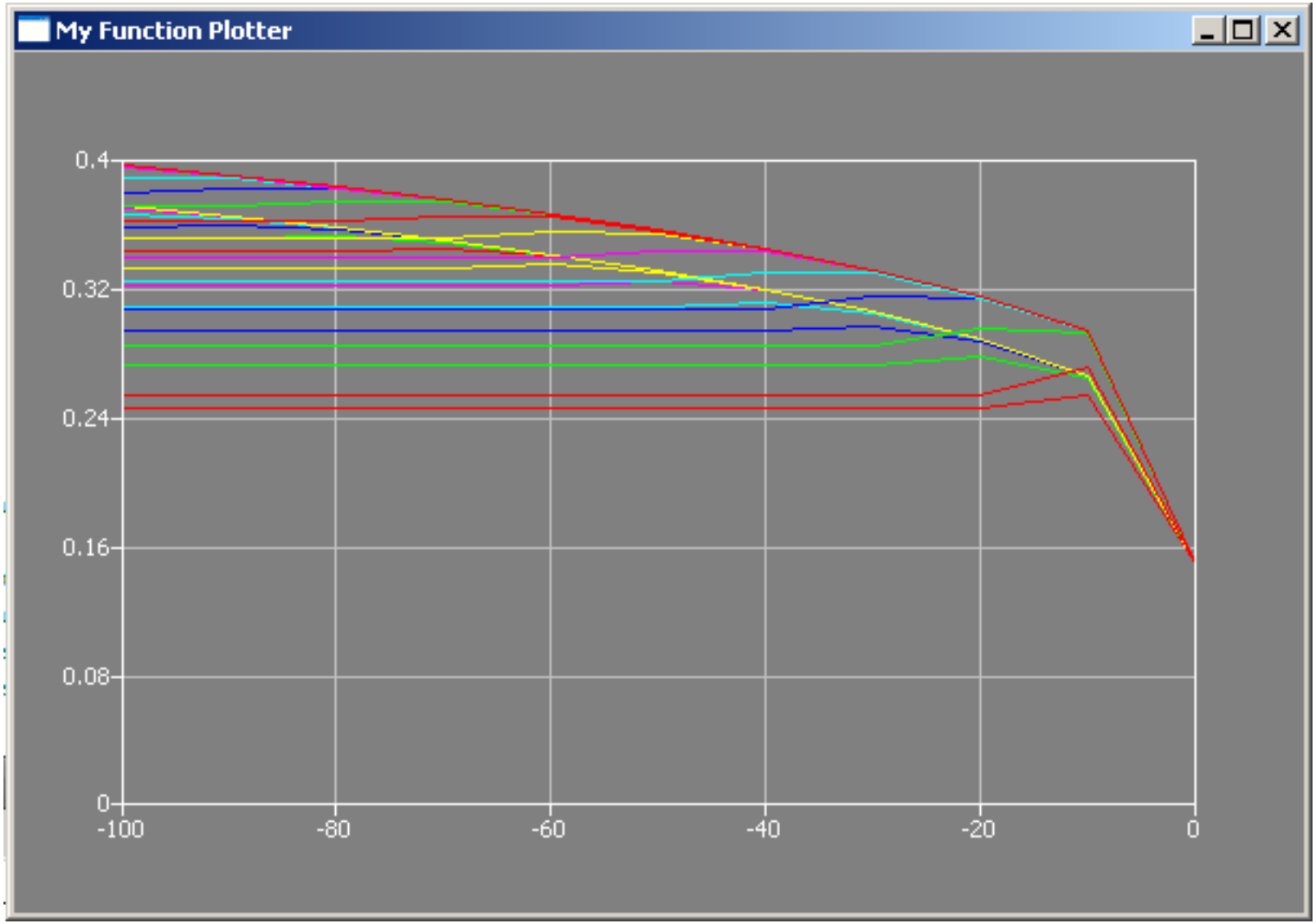
```
void Dialog::on_pushButtonMAT_clicked()
{...
  //ab in den Konstruktor! discharge = 0.05; // Volumenflie
...}

Dialog::Dialog(QWidget *parent) : QDialog(parent)
{...
  //hier bin ich glücklich ...
  discharge = 0.05; // Volumenfließrate [m3/s]
...}
```

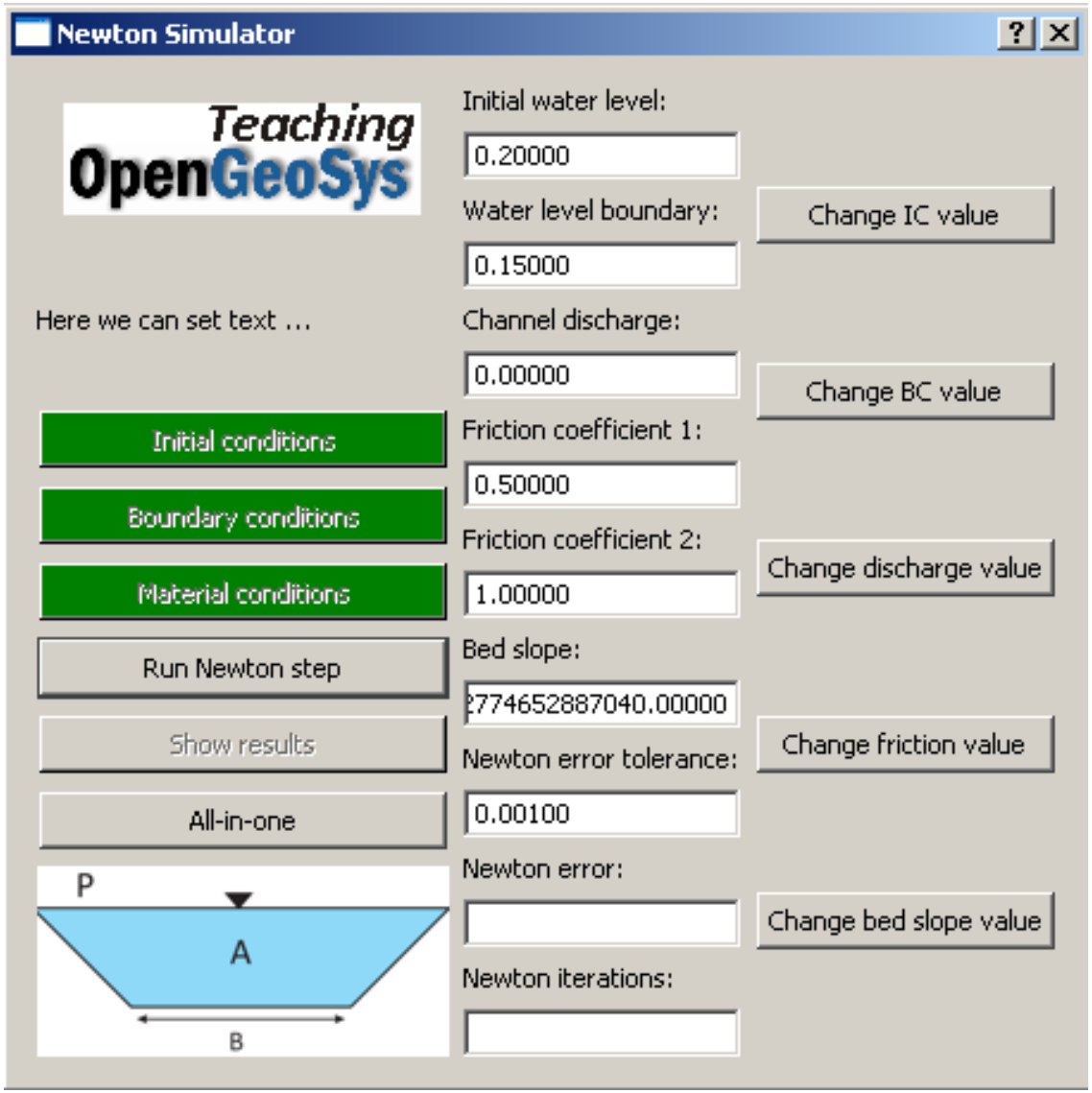
7 - Discharge #2



7 - Discharge #3



8 - Fast fertig #1



8 - Fast fertig #2

Bug list:

-
-
-
-
-
-
-
-

BSc Examples

- Untersuchen sie den Einfluß der Streckenverluste auf die Auslegung eines Tragezgerinnes.
- ...