# Hydroinformatik II: Gerinnehydraulik

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

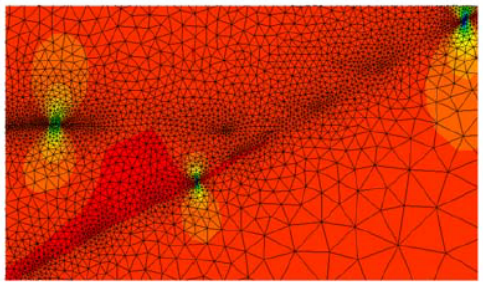
Dresden, 03. Juli 2015

# 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 Abschluss |

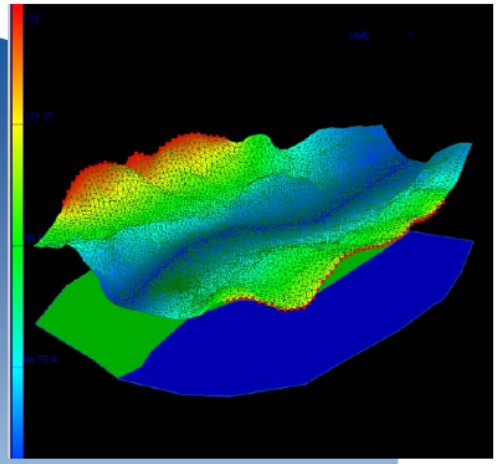
# Konzept

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

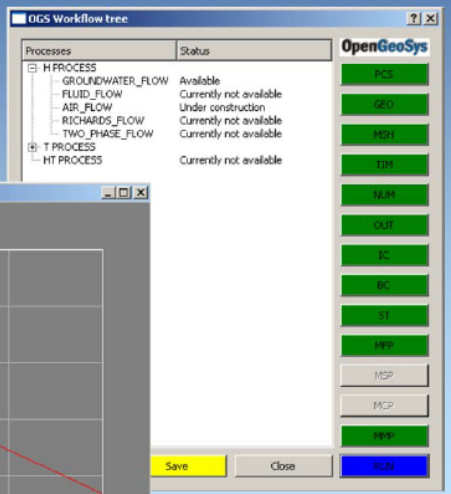


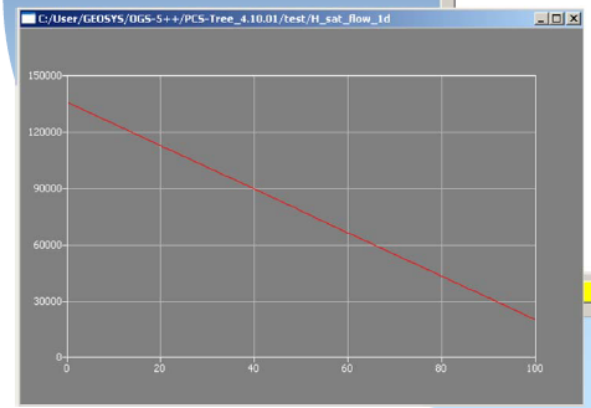
Basics
Mechanik

Anwendung



Numerische
Methoden




Programmierung
Visual C++

Prozessverständnis

# Hydroinformatik - Anwendungen

1. Abfallwirtschaft: Diffusionsprozesse
2. Hydrology: Gerinnehydraulik ($\rightarrow$ this)
3. Grundwasserwirtschaft: Grundwasserhydraulik ($\rightarrow$ "Systemanalyse")

# 3 ways of programming ...

- Q&D
- OOP
- **GUI** ("nicely" $\longmapsto$ efficiently)

# Vorlesungsfahrplan

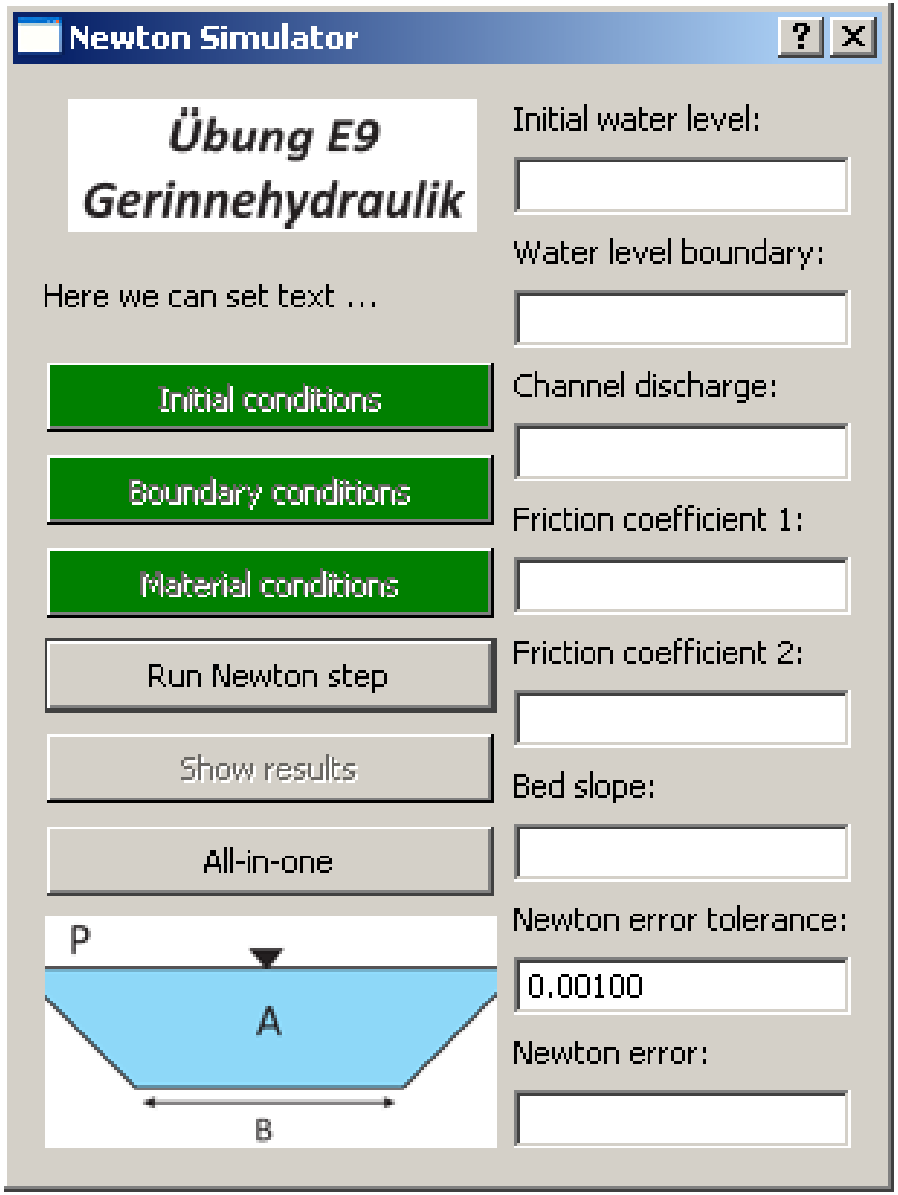
Qt Basics (Handwerkszeug):

- `QLabel`
- `QLineEdit`
- `Get` und `Set` Functions
- `Plotter` issues

Übungen:

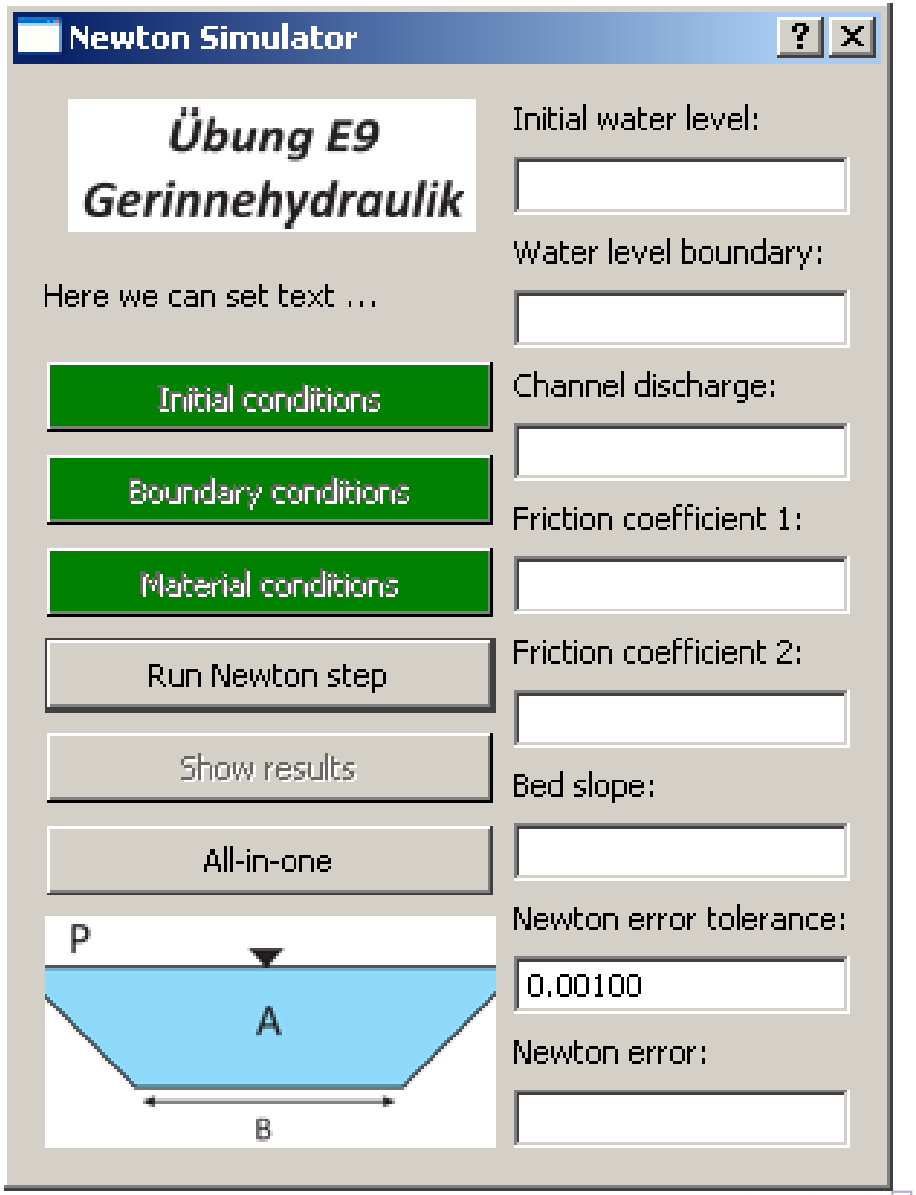
- `E9_ChannelFlow_QAD` (system fix ;-)
- `E10_ChannelFlow_Newton` (OOP)
- `E11_ChannelFlow_Parameters` (how to change values)
- `E12_ChannelFlow_Analysis` (analysing the system)

# Newton Simulator




- Newton iteration (step-by-step)
- Parameter anzeigen und ändern (`QLineEdit`)

# GUI Layout A #1



# GUI Layout A #2

```
//layout
QVBoxLayout *leftLayout = new QVBoxLayout;
leftLayout->addWidget(label_ogs);
leftLayout->addWidget(pushButtonIC);
...
QVBoxLayout *rightLayout = new QVBoxLayout;
rightLayout->addWidget(labelIC);
rightLayout->addWidget(lineEditIC);
...
QHBoxLayout *mainLayout = new QHBoxLayout;
mainLayout->addLayout(leftLayout);
mainLayout->addLayout(rightLayout);
setLayout(mainLayout);
...
rightLayout->addStretch();
```

↪ Tafelbild

# GUI Layout A #3

```
#include <QVBoxLayout>

Dialog::Dialog(QWidget *parent) : QDialog(parent)
{...
  //layout
  QVBoxLayout *leftLayout = new QVBoxLayout;
  leftLayout->addWidget(label_ogs);
  ...
  QVBoxLayout *rightLayout = new QVBoxLayout;
  rightLayout->addWidget(labelIC);
  ...
  QHBoxLayout *mainLayout = new QHBoxLayout;
  mainLayout->addLayout(leftLayout);
  mainLayout->addLayout(rightLayout);
  setLayout(mainLayout);
...}
```

# QBasics: QLabel Text Label

```
#include <QLineEdit>

//text labels
  // definition
QLabel* labelIC = new QLabel(tr("Initial water level:"));
  // use
rightLayout->addWidget(labelIC);
```

# QBasics: QLabel Grafische Label

```
//graphic labels
  // definition
QLabel *label_ogs = new QLabel();
label_ogs->setAlignment(Qt::AlignCenter);
label_ogs->setPixmap(QPixmap("../E9.png"));
  // use
leftLayout->addWidget(label_ogs);
```

# QBasics: QLabel

```
Dialog::Dialog(QWidget *parent) : QDialog(parent)
{...
//text labels
  // definition
  // use
//graphic labels
  // definition
  // use
...}
```

- In welcher Funktion werden Text und Grafik Label definiert und gesetzt?
- Welche Bedeutung hat diese Funktion für unser Programm?
- Können Text-Labels lokal definiert werden? ($\hookrightarrow$ Tafelbild)

# QBasics: QLineEdit

```
#include <QLineEdit>

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

- Warum auf ein mal Unterscheidung zwischen Deklaration und Definition?

↪ Tafelbild

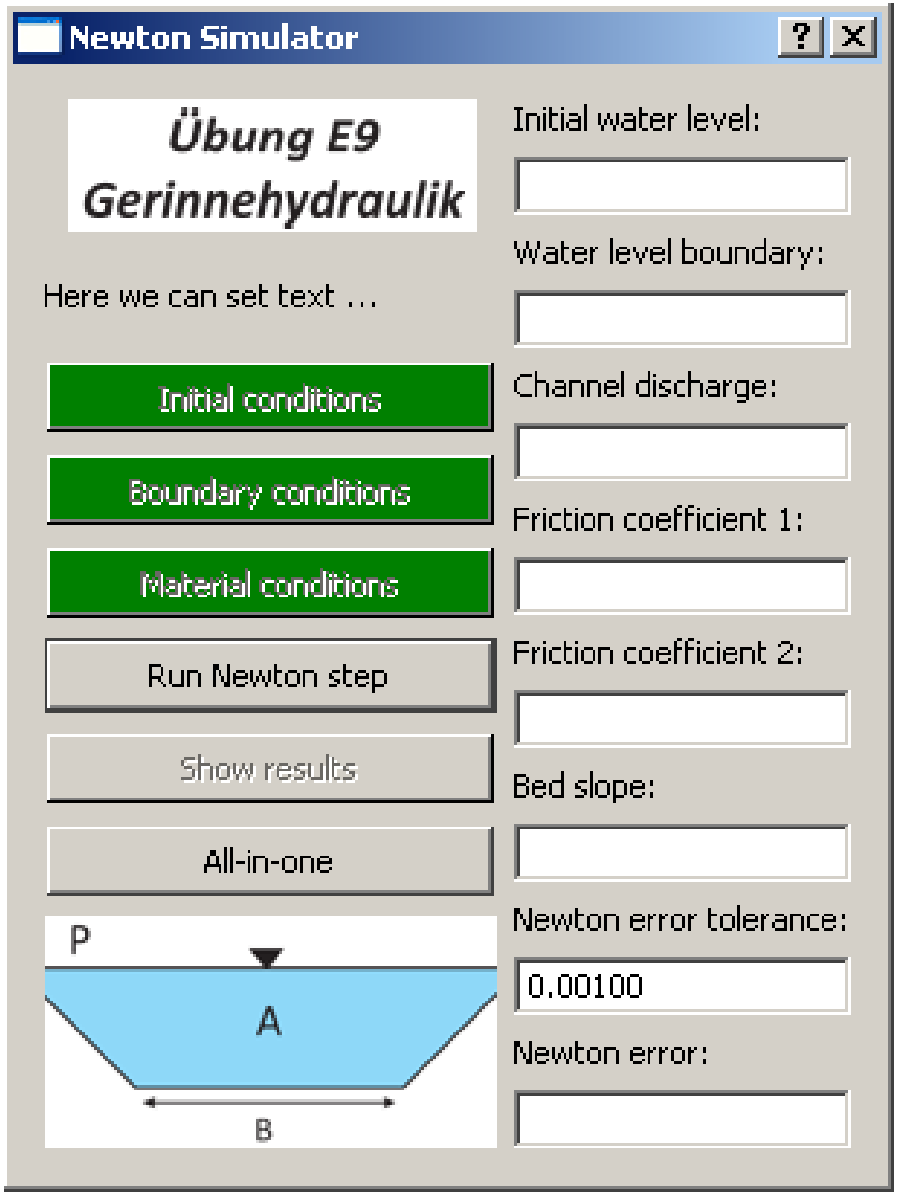
# QBasics: QLineEdit

```
H:
class Dialog : public QDialog
{...
QLineEdit* lineEditIC;
...}

CPP:
Dialog::Dialog(QWidget *parent) : QDialog(parent)
{...
lineEditIC = new QLineEdit();
...}

void Dialog::on_pushButtonIC_clicked()
{...
u_old[0] = lineEditIC->text().toDouble();
...}
```

# QBasics: QLineEdit Get- und Set-Funktion




- Set-Funktion (Daten anzeigen)
- Get-Funktion (Daten manipulieren)

# QBasics: QLineEdit Daten anzeigen

```
void Dialog::on_pushButtonIC_clicked()
{...
// set ->
u_old[0] = 0.25;
QString sIC;
sIC.setNum(u_old[0],'f',5);
lineEditIC->setText(sIC);
...}
```

↪ Tafelbild: Daten konvertieren

# QBasics: QString strings konvertieren

```
double dValue = 1.0;
QString sDummy;

sDummy.setNum(dValue,'f',5); //Konvertierung fX.5

lineEditIC->setText(sDummy);
```

# QBasics: QLineEdit Daten empfangen

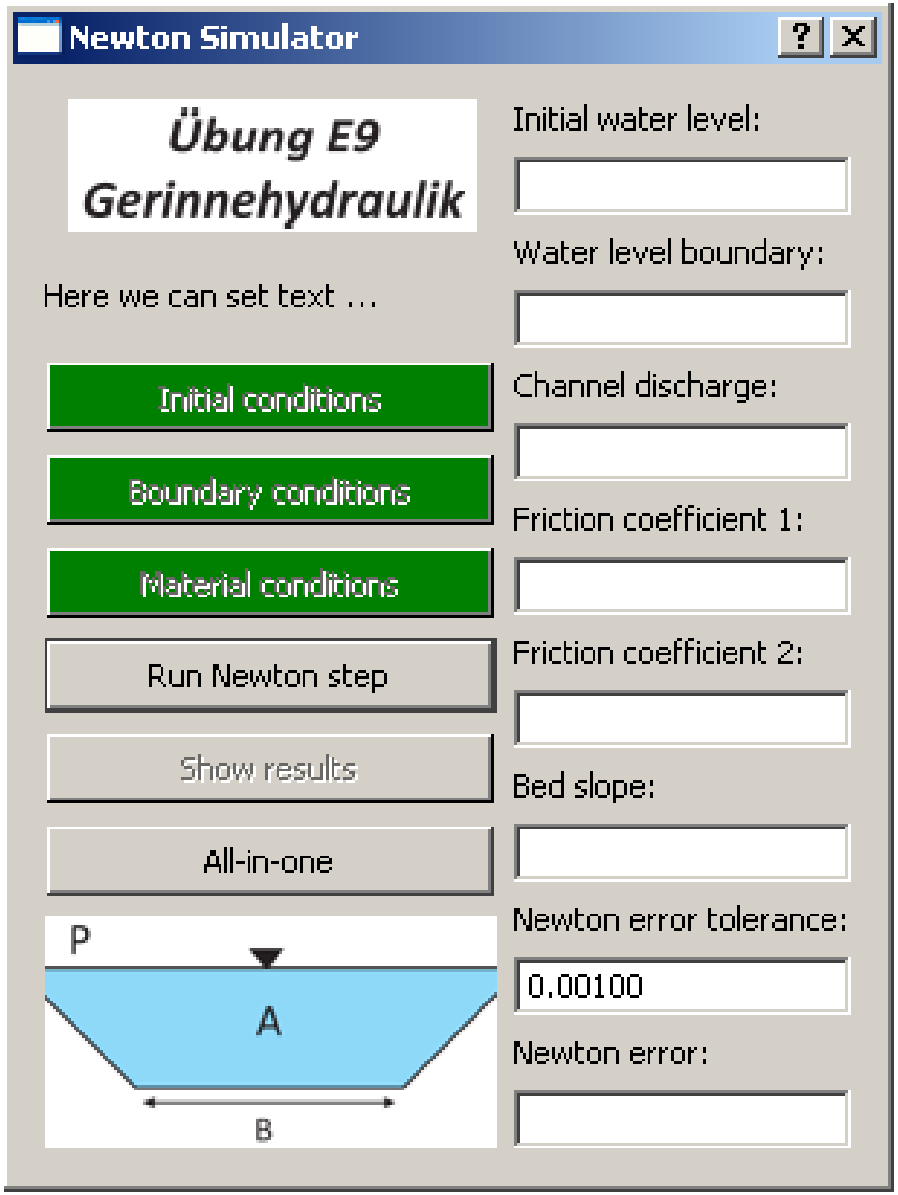
```
void Dialog::on_pushButtonIC_clicked()
{...
// get <-
double dDummy = lineEditIC->text().toDouble();
u_old[0] = dDummy;
...}
```

↪ Tafelbild: Daten konvertieren, Richtungen

# Set- und Get-Funktionen verknüpfen

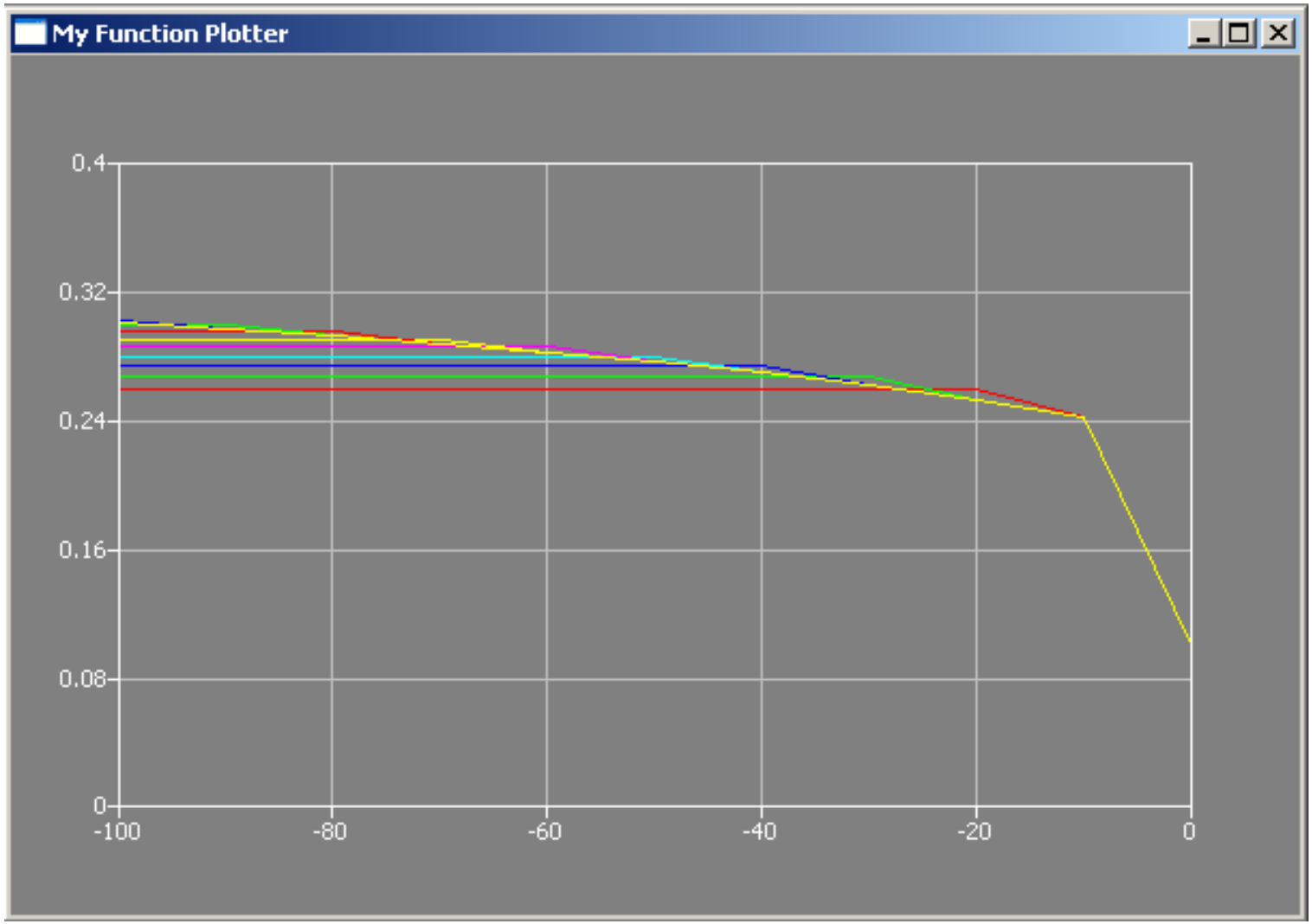
```
QString sIC = lineEditIC->text();
// get ->
if(sIC.length()==0)
{
  sIC.setNum(u_old[0],'f',5);
  lineEditIC->setText(sIC);
}
// set <-
else
{
  double dIC = lineEditIC->text().toDouble();
  for(int i=0;i<n-1;i++)
  {
    u_old[i] = dIC;
  }
}
```

# Newton Simulator




- **Newton iteration (step-by-step)**
- Parameter anzeigen und ändern (`QLineEdit`)

# Newton Iteration



# Modellgleichung

Wir können nun folgendes Funktional konstruieren

$$f(h) = \left(h + \frac{v^2}{2g}\right)|_D - \left(h + \frac{v^2}{2g}\right)|_U + \frac{\Delta x}{2}(S_{f,U} + S_{f,D}) \qquad (1)$$

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

# Berechnung #1

```
void Dialog::on_pushButtonRUN_clicked()
{...
  // Newton iteration
  for(k=0;k<kn;k++)
  {
    double N,N1,N2,N3,D,D1,D2,D21,D22;
    // Grid loop
    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;
    }
  }
...}
```

$$h_i^{k+1} = h_i^k - \frac{N^k}{D^k}$$

# Berechnung #2 test backwards

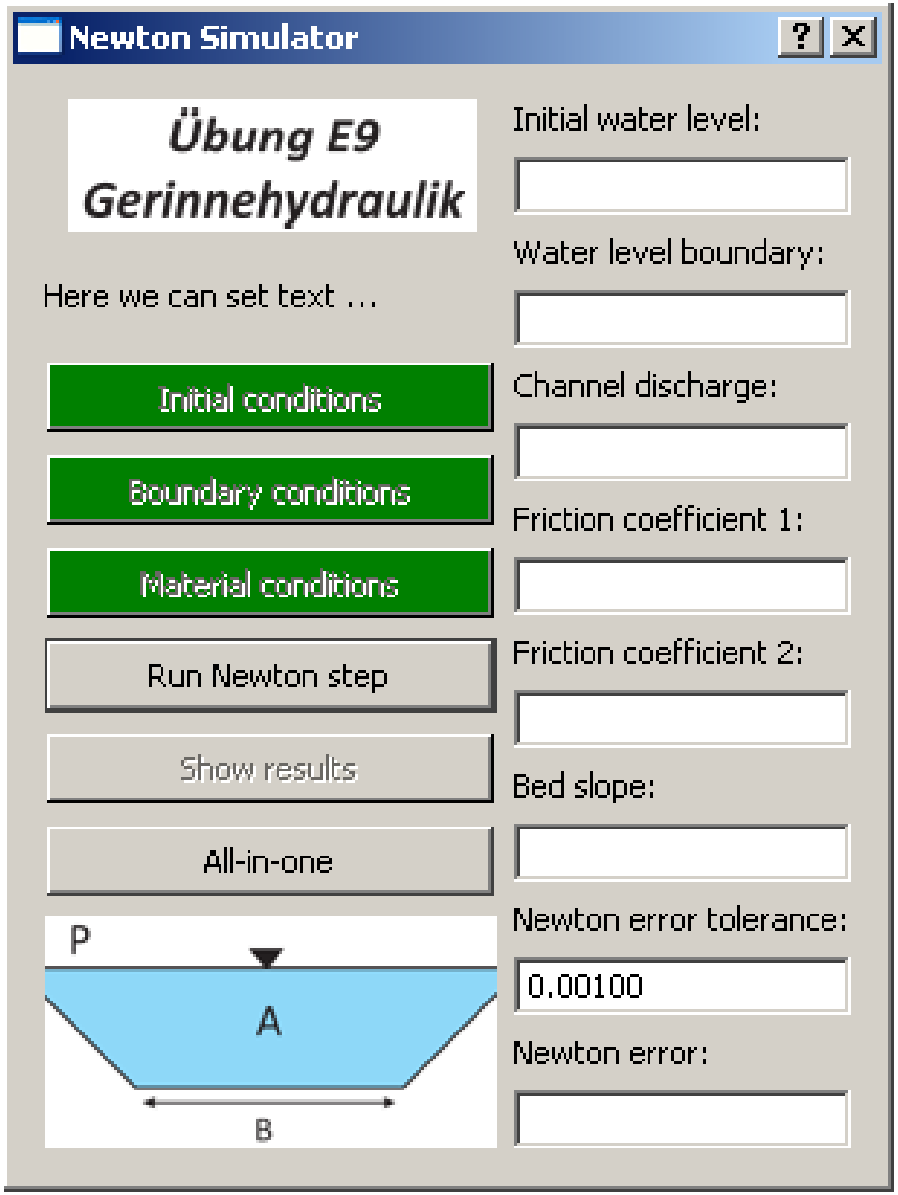
... Sie müssen bei der Gleichung (4.51) wieder landen.

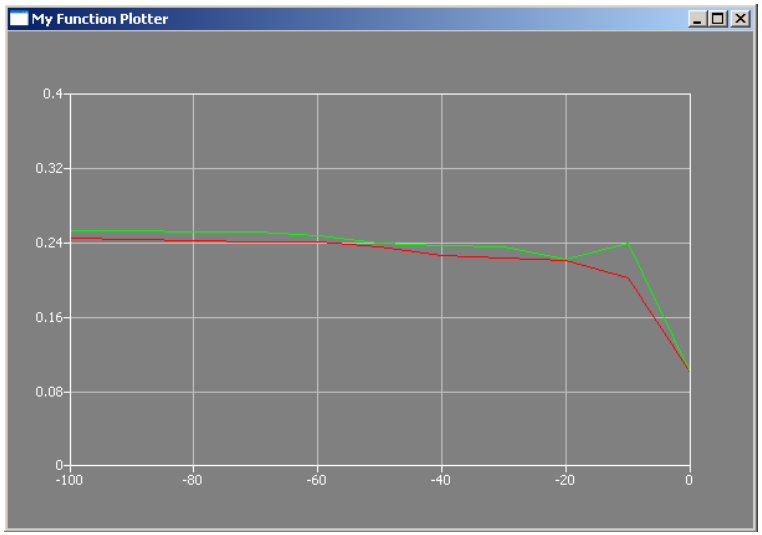
# Newton-Verfahren: Diskretisierung

$$h_i^{k+1} = h_i^k - \frac{N(h^k)}{D(h^k)}, k = 0, ... \tag{2}$$

$$h_i^0 : \textit{Initial condition} \tag{3}$$

# Newton-Schritt Simulator




- Newton-Schritt
- Plotter

# Newton-Schritt

```
void Dialog::on_pushButtonRUN_clicked()
{
  //plotter settings
  ...
  int k=0;

  //run Newton step
  RUN_NewtonStep();

  //plott results
  ...
  plotter->setCurveData(k++, points0);
  plotter->show();
}
```

# Newton-Schritt: Implementierung

```
double Dialog::RUN_NewtonStep()
{
  //local variables
  double N,N1,N2,N3,D,D1,D2,D21,D22;
  double error = 0;
  {
    //start values
    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_
    }
    //Newton step
    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;
    }
```

# Plotter implementation #1

H

```
class Dialog : public QDialog
{...
private:
  Plotter *plotter;
};
```

CPP

```
Dialog::Dialog(QWidget *parent) : QDialog(parent)
{...
  plotter = new Plotter;
}
```

# Plotter implementation #2

H

```
class Dialog : public QDialog
{...
private:
  Plotter *plotter;
  int k;
};
```

CPP

```
Dialog::Dialog(QWidget *parent) : QDialog(parent)
{...
  plotter = new Plotter;
  k=0;
}
```

```
plotter->setCurveData(k++, points0);
```

# Newton error

```
void Dialog::RUN_NewtonStep()
{...
    //calc Newton error
    double error = 0;
    for(int i=0;i<n-1;i++)
    {
      error += u_old[i] -u_new[i];
    }
...}
```

$$\epsilon = \sum_{i=0}^{i=10} (h_i^k - h_i^{k+1})$$

# Newton error: Rückgabewert

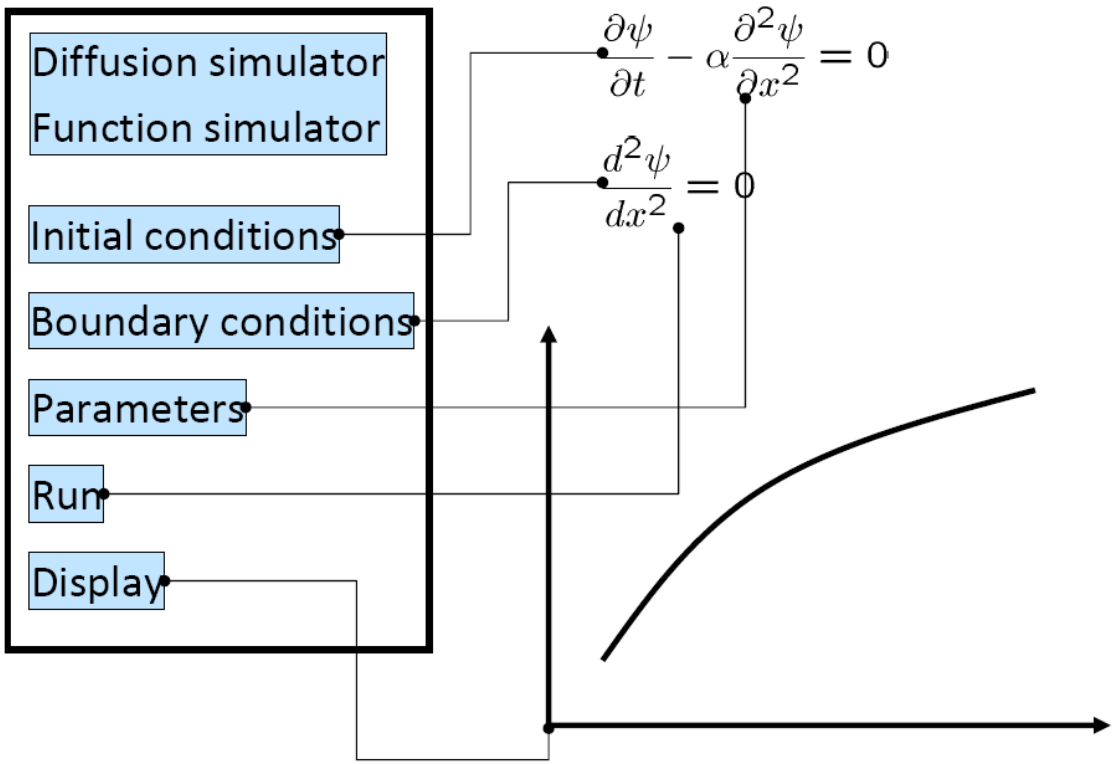
```
Dialog::Dialog(QWidget *parent) : QDialog(parent)
{...
  double RUN_NewtonStep(); //Rückgabewert!
...}

double Dialog::RUN_NewtonStep()
{...
    //calc Newton error
    double error = 0;
    for(int i=0;i<n-1;i++)
    {
      error += u_old[i] -u_new[i];
    }
  return error;
}
```

# Newton-Schritt

```
void Dialog::on_pushButtonRUN_clicked()
{
  double 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();
  // error
  QString sError.setNum(error,'f',5);
  lineEditNewtonError->setText(sError);
}
```

# Zur Erinnerung: ... Konzept und Fazit




Eigenes MatLab ...

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