

# Hydroinformatik II: Gerinnehydraulik

<sup>1</sup>Helmholtz Centre for Environmental Research – UFZ, Leipzig

<sup>2</sup>Technische Universität Dresden – TUD, Dresden

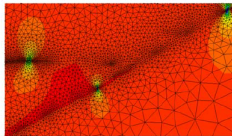
Dresden, 01. Juli 2016

# Vorlesungsplan Hydroinformatik II SoSe 2016

#	Datum	Thema
01	08.04.2016	Einführung
02	15.04.2016	Grundlagen: Kontinuumsmechanik
03	22.04.2016	Grundlagen: Hydromechanik
04	29.04.2016	Grundlagen: Partielle Differentialgleichungen / T <sub>E</sub> X
05	06.05.2016	HW: Qt Installation (2016)
06	13.05.2016	Qt Übung: Funktionsrechner; Grundlagen Numerik
07	20.05.2016	Pfingsten
08	27.05.2016	Numerik: (exp) Finite Differenzen Methode
09	03.06.2016	Vorbereitung für den Beleg
10	10.06.2016	Numerik: (imp) Finite Differenzen Methode
11	17.06.2016	Gerinnehydraulik: Theorie - Grundlagen
12	24.06.2016	HW: Gerinnehydraulik: Programmierung, Übung 1
13	01.07.2016	Gerinnehydraulik: Programmierung, Übung 2
14	08.07.2016	Einführung - Grundwassermodellierung
15	15.07.2016	Kurs-Zusammenfassung, Ausblick und Beleg

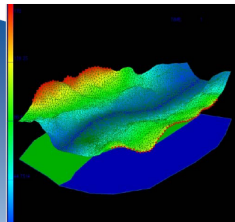
# Konzept

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

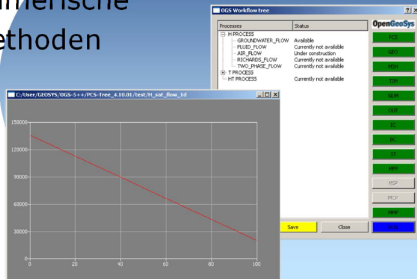


Basics  
Mechanik

Anwendung



Numerische  
Methoden



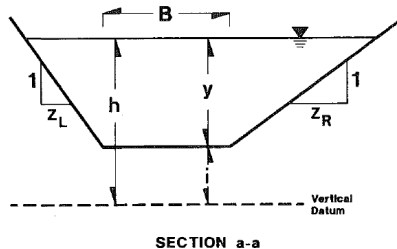
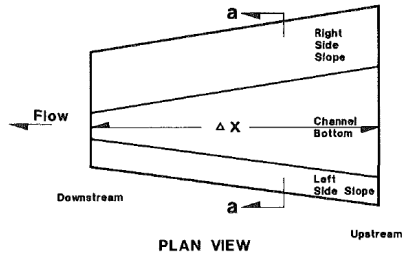
Programmierung  
Visual C++

Prozessverständnis

# Hydroinformatik - Anwendungen

1. Abfallwirtschaft: Diffusionsprozesse
2. Hydrology: Gerinnehydraulik (→ this)
3. Grundwasserwirtschaft: Grundwasserhydraulik (→ next)

# Energiebetrachtung #3: Bernoulli



# Numerisches Verfahren #1

- ▶ Funktional: Energieerhaltung

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

- ▶ Newton-Verfahren:  $\mathbf{x} = h$

$$h_{k+1} = h_k + \frac{f(h_k)}{f'(h_k)} = \frac{\mathbf{R}_k}{\mathbf{J}_k} \quad (2)$$

- ▶  $f'(h) = f'(y)$

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

$$\frac{dA}{dh} = \frac{d}{dh} (z(B + C_4y)) = B + C_4y + yC_4 = B + 2C_4y \quad (4)$$

## Numerisches Verfahren #2

Bleibt noch die Differenzierung der Streckenverluste

$$\frac{dS_f}{dh} = S'_f = \frac{d}{dh} \left( \frac{Q}{AR^{2/3}} \right)^2 \quad (5)$$

$$S'_f = \left[ Q^2 (By + C_4 y^2)^{10/3} \frac{4}{3} (B + C_5 y)^{1/3} C_5 \right] + \left[ (B + y C_5)^{4/3} \frac{-10 Q^2}{3} (By + C_4 y^2)^{13/3} (B + 2 C_4 y) \right] \quad (6)$$

$$S'_f = \frac{4}{3} Q^2 C_5 (By + C_4 y^2)^{-10/3} (B + C_5 y)^{1/3} - \frac{10}{3} Q^2 (B + 2 C_4 y) (B + C_5 y)^{4/3} (By + C_4 y^2)^{-13/3} \quad (7)$$

## 3 ways of programming ...

- ▶ Q&D: Übung 1 (BHYWI-08-09-E1)
- ▶ OOP: Übung 2 (BHYWI-08-09-E2)
- ▶ GUI ("nicely"): Übung 3 (Qt)



# Q&D #1

```
int main(int argc, char *argv[])
{
    // Geometrie
    // Anfangsbedingungen
    // Randbedingungen
    // Parameter
    // Berechnungsgrößen
    // Berechnung (1. Iteration des Newton-Verfahrens)
    // Ausgabe der Ergebnisse
    // File (Übung E10A)
    // x-y Plot (Übung E10B)
}
```

Exercises: E10\_ChannelFlow\_QAD(A,B)

## Q&amp;D #2

```
int main(int argc, char *argv[])
{...
    // Geometrie
    int n = 11;
    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;
...}
```

## Q&D #3

```
int main(int argc, char *argv[])
{...
  // Anfangsbedingungen
  double u_old[n];
  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
...}
```

Warum so genau?

## Q&D #4

```
int main(int argc, char *argv[])
{...
    // Randbedingungen
    u_old[10] = 0.1; // Wasserstand fluabwrts [m]
    double u_new[n];
    u_new[10] = 0.1; // Wasserstand fluabwrts [m]
...}
```

Tafelbild

## Q&amp;D #5

```
int main(int argc, char *argv[])
{...
    // Parameter
    double discharge = 0.05; // Volumenfließrate [m3/s]
    double gravity = 9.81; // [m/s2]
    double friction_law_exponent = 0.5; // Chezy, Manning-Str
    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.; //
...}
```

Tabelle

## Q&D #6 Berechnungsgrößen

```
int main(int argc, char *argv[])
{...
    // Berechnungsgrößen
    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];
...}
```

## Q&D #7 Berechnung Parameter

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

## Q&D #8 Berechnung

```
int main(int argc, char *argv[])
{...
  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_ol
    N2 = pow(discharge,2)/pow(wetted_cross_section[i],2) + gravity*u_ol
    N3 = gravity*(bed_slope - (friction_slope[i+1]+friction_slope[i])/2
    N = N1 - N2 - N3;
    D1 = pow(discharge,2)/pow(wetted_cross_section[i],3) * (bottom_widht
    D21 = friction_law_exponent*2.*(sqrt(1+m*m))/wetted_perimeter[i];
    D22 = (1.+friction_law_exponent)/wetted_cross_section[i] * (bottom
    D2 = gravity*friction_slope[i]*(D21-D22)*(x[i+1]-x[i]);
    D = D1 + D2;
    u_new[i] = u_old[i] - N/D;
  }
...}
```



## Q&D #9 Ausgabe File

```
int main(int argc, char *argv[])
{...
    ofstream out_file("out.txt");
    out_file.precision(4);
    out_file << "Water depth (old):\t";
    for(int i=0;i<n;i++)
    {
        out_file << "\t" << u_old[i] << " ";
    }
    out_file << endl;
...
    out_file << "Water depth (new):\t";
    for(int i=0;i<n;i++)
    {
        out_file << "\t" << u_new[i] << " ";
    }
    out_file << endl;
    out_file.close();
...}
```

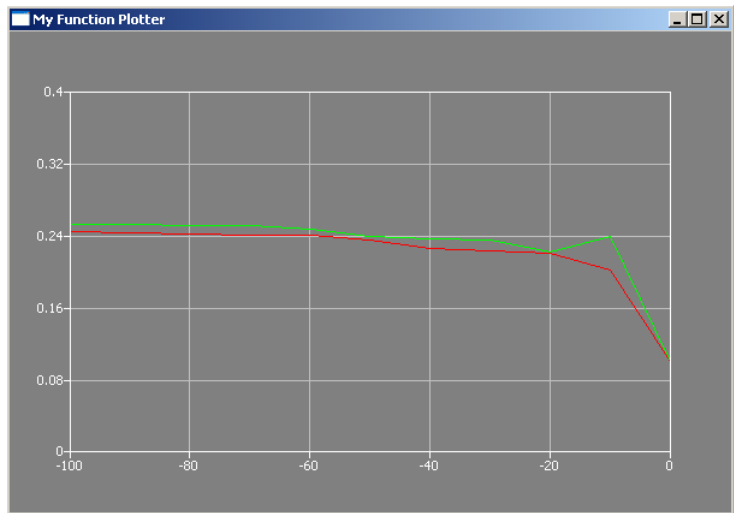
## Q&amp;D #9 Ausgabe File

Datei	Bearbeiten	Format	Ansicht	?									
0.25													
0.1													
Iteration: 0													
water depth (old):		0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.1
wetted perimeter:		1.71	1.71	1.71	1.71	1.71	1.71	1.71	1.71	1.71	1.71	1.71	1.28
wetted cross section:	0.313	0.313	0.313	0.313	0.313	0.313	0.313	0.313	0.313	0.313	0.313	0.11	
Hydraulic radius:		0.183	0.183	0.183	0.183	0.183	0.183	0.183	0.183	0.183	0.183	0.183	0.0857
water level elevation:	0.29	0.286	0.282	0.278	0.274	0.27	0.266	0.262	0.258	0.254	0.1		
Flow velocity:		0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.455
Froude number:		0.112	0.112	0.112	0.112	0.112	0.112	0.112	0.112	0.112	0.112	0.112	0.479
Friction slope:	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0241
water depth (new):		0.259	0.259	0.259	0.259	0.259	0.259	0.259	0.259	0.259	0.259	0.242	0.1
Iteration: 1													
water depth (old):		0.259	0.259	0.259	0.259	0.259	0.259	0.259	0.259	0.259	0.259	0.242	0.1
wetted perimeter:		1.73	1.73	1.73	1.73	1.73	1.73	1.73	1.73	1.73	1.73	1.69	1.28
wetted cross section:	0.326	0.326	0.326	0.326	0.326	0.326	0.326	0.326	0.326	0.301	0.11		
Hydraulic radius:		0.188	0.188	0.188	0.188	0.188	0.188	0.188	0.188	0.188	0.188	0.179	0.0857
water level elevation:	0.299	0.295	0.291	0.287	0.283	0.279	0.275	0.271	0.267	0.246	0.1		
Flow velocity:	0.153	0.153	0.153	0.153	0.153	0.153	0.153	0.153	0.153	0.166	0.455		
Froude number:	0.105	0.105	0.105	0.105	0.105	0.105	0.105	0.105	0.105	0.118	0.479		
Friction slope:	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125
water depth (new):		0.267	0.267	0.267	0.267	0.267	0.267	0.267	0.267	0.253	0.242	0.1	
Iteration: 2													
water depth (old):		0.267	0.267	0.267	0.267	0.267	0.267	0.267	0.267	0.253	0.242	0.1	
wetted perimeter:		1.76	1.76	1.76	1.76	1.76	1.76	1.76	1.76	1.72	1.69	1.28	
wetted cross section:	0.339	0.339	0.339	0.339	0.339	0.339	0.339	0.339	0.339	0.317	0.301	0.11	
Hydraulic radius:		0.193	0.193	0.193	0.193	0.193	0.193	0.193	0.193	0.185	0.179	0.0857	
water level elevation:	0.307	0.303	0.299	0.295	0.291	0.287	0.283	0.279	0.261	0.246	0.1		
Flow velocity:	0.148	0.148	0.148	0.148	0.148	0.148	0.148	0.148	0.158	0.166	0.455		
Froude number:	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.11	0.118	0.479		
Friction slope:	0.00113	0.00113	0.00113	0.00113	0.00113	0.00113	0.00113	0.00113	0.00113	0.00113	0.00113	0.00113	0.00113
water depth (new):		0.274	0.274	0.274	0.274	0.274	0.274	0.274	0.262	0.253	0.242	0.1	
Iteration: 3													
water depth (old):		0.274	0.274	0.274	0.274	0.274	0.274	0.274	0.274	0.262	0.253	0.242	0.1
wetted perimeter:		1.78	1.78	1.78	1.78	1.78	1.78	1.78	1.74	1.72	1.69	1.28	
wetted cross section:	0.349	0.349	0.349	0.349	0.349	0.349	0.349	0.331	0.317	0.301	0.11		
Hydraulic radius:		0.197	0.197	0.197	0.197	0.197	0.197	0.197	0.19	0.185	0.179	0.0857	
water level elevation:	0.314	0.31	0.306	0.302	0.298	0.294	0.29	0.274	0.261	0.246	0.1		
Flow velocity:	0.143	0.143	0.143	0.143	0.143	0.143	0.143	0.151	0.158	0.166	0.455		
Froude number:	0.0962	0.0962	0.0962	0.0962	0.0962	0.0962	0.0962	0.104	0.11	0.118	0.479		
Friction slope:	0.00104	0.00104	0.00104	0.00104	0.00104	0.00104	0.00104	0.00104	0.00104	0.00104	0.00104	0.00104	0.00104
water depth (new):		0.28	0.28	0.28	0.28	0.28	0.28	0.27	0.262	0.253	0.242	0.1	

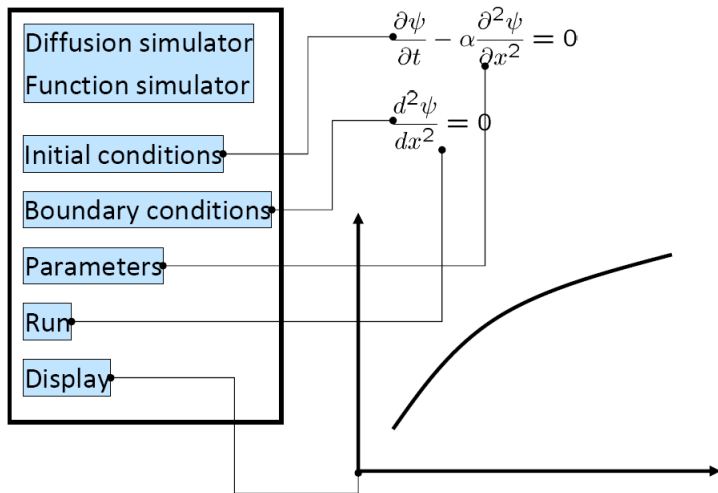
## Q&D #10 Ausgabe Grafik

```
int main(int argc, char *argv[])
{...
    Plotter plotter;
    plotter.setWindowTitle(QObject::tr("My Function Plotter"));
    QVector<QPointF> points0;
    QVector<QPointF> points1;
    double l;
    for (int i=0;i<n;++i)
    {
        l = -100. + 10.*i;
        points0.append(QPointF(l,u_old[i]));
        points1.append(QPointF(l,u_new[i]));
    }
    plotter.setCurveData(0, points0);
    plotter.setCurveData(1, points1);
    PlotSettings settings;
    settings.minX = -100.0;
    settings.maxX = 0.0;
    settings.minY = 0.0;
    settings.maxY = 0.4;
```

# Q&D #11 Ausgabe Grafik



# OOP #1



## OOP #2

```
int main(int argc, char *argv[])
{
    // Geometrie
    // Anfangsbedingungen
    // Randbedingungen
    // Parameter
    // Berechnungsgrößen
    // Berechnung (1. Iteration des Newton-Verfahrens)
    // Ausgabe der Ergebnisse
    // File (Übung E10A)
    // x-y Plot (Übung E10B)
}
```

# OOP #3

```
Dialog::~Dialog()  
void Dialog::on_pushButtonIC_clicked()  
void Dialog::on_pushButtonBC_clicked()  
void Dialog::on_pushButtonMAT_clicked()  
void Dialog::on_pushButtonRUN_clicked()  
void Dialog::on_pushButtonSHO_clicked()
```

## OOP #4

```
void Dialog::on_pushButtonIC_clicked()
{
    // 2 Anfangsbedingungen
    u_old[0]=0.244918436659073; //-100
    u_old[1]=0.243;           //-90
    ...
    u_old[9]=0.201434531839821; //-10
    u_old[10]=0.1;           //0
    pushButtonIC->setStyleSheet("background-color: green");
    pushButtonBC->setEnabled(true);
    out_file << lineEditIC->text().toStdString() << endl;
    double IC = lineEditIC->text().toDouble();
    for(int i=0;i<n-1;i++)
    {
        u_old[i] = IC;
    }
}
```



## OOP #4

```
void Dialog::on_pushButtonBC_clicked()
{
    // 3 Randbedingungen
    double BCR = 0.05;
    BCR = lineEditBCR->text().toDouble();
    out_file << BCR << endl;
    u_old[10] = BCR; // Wasserstand fluabwrts [m]
    u_new[10] = BCR; // Wasserstand fluabwrts [m]
   _pushButtonBC->setStyleSheet("background-color: green");
   _pushButtonMAT->setEnabled(true);
}
```

# OOP #5 MAT

```
void Dialog::on_pushButtonMAT_clicked()
{
    // Parameter
    discharge = 0.05; // Volumenfließrate [m3/s]
    gravity = 9.81; // [m/s2]
    friction_law_exponent = 0.5; // Chezy, Manning-Strickler
    error_tolerance = 1e-3; // [m]
    bed_slope = 0.0004; // [m/m]
    bottom_width = 1.; // [m]
    m = 1.; //
    friction_coefficient = 10.; //
    pushButtonMAT->setStyleSheet("background-color: green");
    pushButtonRUN->setEnabled(true);
}
```

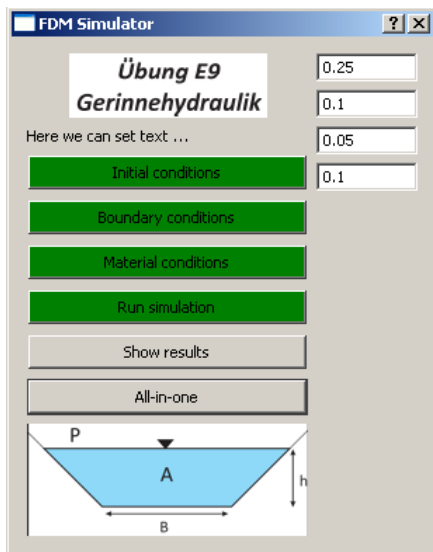
# OOP #6 RUN

```
// Newton iteration loop
for(int k=0;k<kn;k++)
{...
    RunNewtonStep();
...}
```

# OOP #7 AIO

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

## GUI #1



# GUI #2

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    //.....
    Dialog w;
    //splash.finish(&w); //test
    w.setWindowTitle("FDM Simulator");
    w.setFixedWidth(300);
    w.show();
    return a.exec();
}
```

# GUI #3

```
int main(int argc, char *argv[])
{
    //.....
    QPixmap pixmap ("../E9.jpg");
    QSplashScreen splash(pixmap);
    splash.show();
    splash.showMessage(QObject::tr("Übung E9 wird geladen..."), Qt::black);
    QTime dieTime = QTime::currentTime().addSecs(3); while( QTime::current
        QCoreApplication::processEvents(QEventLoop::AllEvents
        // Warte-Funktion (http://lists.trolltech.com/qt-inte
}
```

## GUI #4

