

Modellierung Hydrosysteme: Finite-Differenzen-Methode (FDM)

Prof. Dr.-Ing. habil. Olaf Kolditz

¹Helmholtz Centre for Environmental Research – UFZ, Leipzig

²Technische Universität Dresden – TUD, Dresden

Dresden, 16. Juni 2017

Semester-Fahrplan

April	Mai	Juni	Juli				
	5.5.2017	6.2.2017	7.3.2014				Kolditz
				SCH/A 251/H	07.04.2017	1	Einführung / Hydrologische Modellierung
				CHE/089/E	21.04.2017	3	Hydrologische Modellierung II
				CHE/089/E	28.04.2017	5	DR DECOVALEX
				CHE/089/E	05.05.2017	7	Wdh. Hydromechanik / Übung Einzugsgebiet
					12.05.2017	9	DR China
					19.05.2017	11	Exkursion VisLab
4.7.2017	5.12.2017	6.9.2017	7.14.2017				
	Qt Übung	Pfingsten		SCH/A 251/H	26.05.2017	13	Finite-Differenzen-Verfahren 2D
	HW			CHE/089/E	02.06.2017	15	Objekt-Orientierte FDM / Randbedingungen
	HW			CHE/089/E	16.06.2017	17	FDM Anwendung Selke-Einzugsgebiet
	China			CHE/089/E	23.06.2017	19	Implizite FDM
					30.06.2017	21	Finite-Elemente-Methode I
					07.07.2017	23	Finite-Elemente-Methode II
4.14.2017	5.19.2017	6.16.2017					
Ostern	Qt Übung			SCH/A 251/H	14.07.2017	25	Zusammenfassung, Klausurvorbereitung
	VISLAB			CHE/089/E			
	VISLAB			CHE/089/E			
	China			CHE/089/E			
	Brücke ntag						
4.21.2017	5.26.2017	6.23.2017					
				SCH/A 251/H			
				CHE/089/E	21.04.2017	4	V2 Ursprung & Verbreitungswege Ü2 Fließwege
					28.04.2017	6	Ü2b Fließwege (als HA)
				CHE/089/E	05.05.2017	8	V3 Visualisierung Ü3 Visualisierung
				CHE/089/E	12.05.2017	10	V4 Detektion I
				CHE/089/E	19.05.2017	12	Exkursion VisLab? oder Ü3b Visualisierung (als HA)
				CHE/089/E	26.05.2017	14	V5 Detektion II Ü4 NDVI
		Wasserseminar			02.06.2017	16	V6 Exkurs Salzwasserintrusion
4.28.2017		6.30.2017	Klausur		16.06.2017	18	V7 GW-Modellierung Ü5 GW analytisch
Qt Intro			Termin	SCH/A 251/H	23.06.2017	20	V8 Benchmarking Ü6 OGS Einführung
Ü1				CHE/089/E	30.06.2017	22	V9 Meshing Ü6b Darcy num. (HA Meshing)
Ü2				CHE/089/E	07.07.2017	24	V10 Konservativer Stofftransport Ü7 Kons. ST
DECOVALEX				CHE/089/E	14.07.2017	26	Zusammenfassung, Klausurvorbereitung

Fahrplan für heute ...

- ▶ Anwendung Selke Catchment, irgendetwas stimmt noch nicht ...
- ▶ FDM - OOP: erstmal aufräumen ... (BHYWI-22-E4)
- ▶ Übung: Geometrisches Modellieren
- ▶ Übung: Visualisierung (VTK Filter, BHYWI-22-E5)

Selke Einzugsgebiet

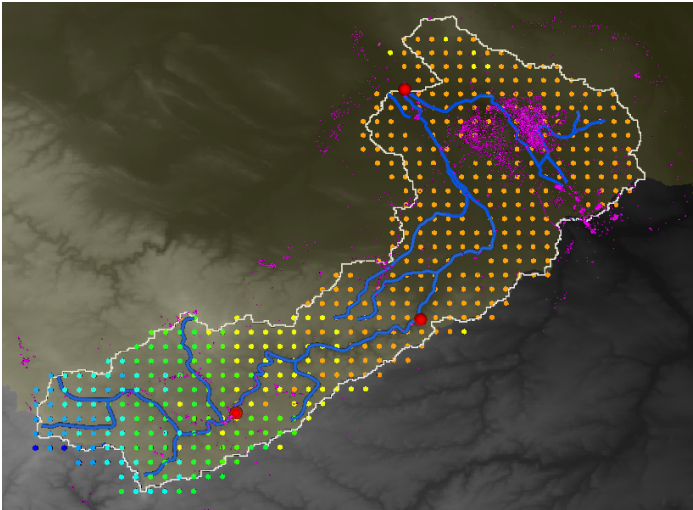


Abbildung: Untersuchungsgebiet - Selke

Selke Einzugsgebiet

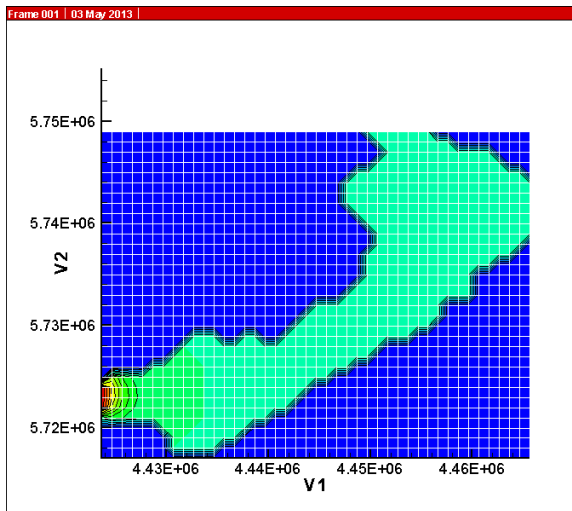


Abbildung: Hier stimmt was nicht

FDM OOP

Wie sie bereits wissen, geht nachdem das FDM Programm funktioniert, die Arbeit erst richtig los. Aus dem QAD Programm wird nun ein OOP, d.h. wir legen eine C++ Klasse an und packen die einzelnen Rechenschritte in Methoden der Klasse. Das Ergebnis sehen wir in der neuen main Function: das ganze FDM passt nun auf eine halbe Seite Quelltext (wobei ein großer Teil noch für die Zeitmessung draufgeht).

Übung BHYWI-22-E4 - FDM OOP

- ▶ Geometrie (siehe letzte Vorlesung)
- ▶ Anfangsbedingungen
- ▶ Randbedingungen
- ▶ Zeitschleife
- ▶ Ergebnisse speichern und schreiben ...

FDM OOP: die main() Funktion

```
#include <iostream>
#include "fdm.h"
int main(int argc, char *argv[])
{
    //-----
    FDM* fdm = new FDM();
    fdm->SetActiveNodes();
    fdm->SetInactiveNodes();
    fdm->SetInitialConditions();
    fdm->SetBoundaryConditions();
    //-----
    int tn = 100;
    for(int t=0;t<tn;t++)
    {
        fdm->RunTimeStep();
        fdm->SaveTimeStep();
        fdm->OutputResults(t);
    }
    //-----
    return 0;
}
```


FDM OOP: die Klasse

```
class FDM
{
    //data structures
    std::vector<float>u_new;
    std::vector<float>u;
    std::vector<float>u_bc;
    float u0;
    float dx,dy,dt;
    float S0,Kf,Q;
    float x,y,x0,y0;
    int ix;
    int jy;
    std::ofstream out_file;
    std::vector<int>bc_nodes;
    std::vector<int>nodes_inactive;
    std::list<int>nodes_active;
    std::ifstream active_nodes_file;
    //methods
    ...
};
```

FDM OOP: die Klasse

```
class FDM
{
    //data structures
    ...
    //methods
    FDM();
    void SetActiveNodes();
    void SetInactiveNodes();
    void SetInitialConditions();
    void SetBoundaryConditions();
    void RunTimeStep();
    void SaveTimeStep();
    void OutputResults(int);
    void OutputMesh();
    bool IsBCNode(int, std::vector<int>);
    bool IsNodeInactive(int, std::vector<int>);
    bool NodeInList(int, std::list<int>);
};
```

FDM OOP: Konstruktor

```
FDM::FDM()
{
    ix = 43;
    jy = 33;
    dx = 1000.;
    dy = 1000.;
    dt = 0.25e6; // sec
    S0 = 1e-5;
    Kf = 1e-5; // m/s
    Q = 0.; //1e-5;
    u0 = 300.;
    x0 = 4423656.0991422;
    y0 = 5716944.1927754;
    //memory allocation
    u.resize(ix*jy);
    u_new.resize(ix*jy);
    //output
    out_file.open("../out.txt");
    dx2 = dx*dx;
    dy2 = dy*dy;
}
```

FDM OOP: Geometrie

```
bool FDM::IsNodeInactive(int n, std::vector<int>nodes_inactive)
{
    bool is_node_inactive = false;
    for(int k=0;k<(size_t)nodes_inactive.size();k++)
    {
        if(n==nodes_inactive[k])
        {
            is_node_inactive = true;
            return is_node_inactive;
        }
    }
    return is_node_inactive;
}
```

FDM OOP: Geometrie

```
bool FDM::IsBCNode(int n, std::vector<int>bc_nodes)
{
    bool is_node_bc = false;
    size_t k; // size_t is the unsigned integer type, may not good for big size p
    for(k=0;k<(size_t)bc_nodes.size();k++)
    {
        if(n==bc_nodes[k])
        {
            is_node_bc = true;
            return is_node_bc;
        }
    }
    return is_node_bc;
}
```

FDM OOP: Anfangsbedingungen

```
void FDM::SetInitialConditions()
{
    for(j=0;j<jy;j++)
    {
        nn = j*ix;
        for( i=0;i<ix;i++)
        {
            n = nn+i;
            u[n] = u0;
            u_new[n] = u0;
        }
    }
}
```

FDM OOP: Randbedingungen

```
void FDM::SetBoundaryConditions()
{
    //top and bottom
    int l;
    for( i=0;i<ix;i++)
    {
        bc_nodes.push_back(i); u[i] = u0;  u_new[i] = u0;
        l = ix*(jy-1)+i;
        if(l>1402&&1<1408)
        {
            bc_nodes.push_back(l); u[l] = u0;  u_new[l] = u0;
        }
        else
        {
            bc_nodes.push_back(l); u[l] = u0;  u_new[l] = u0;
        }
    }
    //left and right side
    ...
}
```

FDM OOP: Randbedingungen

```
void FDM::SetBoundaryConditions()
{
    //top and bottom
    ...
    //left and right side
    for(j=1;j<jy-1;j++)
    {
        l = ix*j;
        if(j>4&& j<9)
        {
            bc_nodes.push_back(1); u[l] = 800.; u_new[l] = 800.;
        }
        else
        {
            bc_nodes.push_back(1); u[l] = u0; u_new[l] = u0;
        }
        l = ix*j+ix-1;
        bc_nodes.push_back(1); u[l] = u0; u_new[l] = u0;
    }
}
```


FDM OOP: Zeitschritt rechnen

```
void FDM::RunTimeStep()
{
    for( j=0;j<jy;j++)
    {
        nn = j*ix;
        for( i=0;i<ix;i++)
        {
            n = nn+i;
            if(IsBCNode(n,bc_nodes))
                continue;
            if(IsNodeInactive(n,nodes_inactive))
                continue;
            u_new[n] = u[n] \
                + Kf/S0*dt/dx2 * (u[n+1]-2*u[n]+u[n-1]) \
                + Kf/S0*dt/dy2 * (u[(j+1)*ix+i]-2*u[n]+u[(j-1)*ix+i]) \
                + Q/S0;
        }
    }
}
```

FDM OOP: Zeitschritt sichern

```
void FDM::SaveTimeStep()
{
    //save time step
    for(int j=0;j<jy;j++)
        for(int i=0;i<ix;i++)
            {
                u[j*ix+i] = u_new[j*ix+i];
            }
}
```

FDM OOP: Ergebnisse schreiben - TECPLOT

```
void FDM::OutputResults(int t)
{
    if((t%10)==0)
    {
        out_file << "ZONE T=\"BIG ZONE\", I=" << ix << ", J=" << jy << ", DATAPACKIN
        for(int j=0;j<jy;j++) //y
        {
            y = y0 + j*dy;
            nn = j*ix;
            for(int i=0;i<ix;i++) //x
            {
                n = nn+i;
                x = x0 + i*dx;
                if(IsNodeInactive(n,nodes_inactive))
                    out_file << x << "\t" << y << "\t" << 0.0 << std::endl;
                else
                    out_file << x << "\t" << y << "\t" << u_new[n] << std::endl;
            }
        }
    }
}
```

VTK

```

# vtk DataFile Version 2.0           ](1)
Really cool data                     ](2)
ASCII | BINARY                       ](3)
DATASET type                        ](4)
...
POINT_DATA n                        ](5)
...
CELL_DATA n
...

```

Part 1: Header

Part 2: Title (256 characters maximum, terminated with newline \n character)

Part 3: Data type, either ASCII or BINARY

Part 4: Geometry/topology. *Type* is one of:

```

STRUCTURED_POINTS
STRUCTURED_GRID
UNSTRUCTURED_GRID
POLYDATA
RECTILINEAR_GRID
FIELD

```

Part 5: Dataset attributes. The number of data items *n* of each type must match the number of points or cells in the dataset. (If *type* is

FDM OOP: Ergebnisse schreiben - VTK

► File-Struktur

```
# vtk DataFile Version 3.0
Unstructured Grid from OpenGeoSys
ASCII
DATASET UNSTRUCTURED_GRID
POINTS 325 double
0.000000000000e+000 0.000000000000e+000 0.000000000000e+000
...
1.916666666667e+001 1.000000000000e+001 0.000000000000e+000
CELLS 288 1440
4 0 4 28 27
...
4 300 301 2 324
CELL_TYPES 288
9
...
9
POINT_DATA 325
SCALARS HEAD double 1
LOOKUP_TABLE default
1.266090102200e-001
```

FDM OOP: Ergebnisse schreiben - VTK - BHYWI-22-E5

```
void FDM::OutputResultsVTK(int t)
{
    ... in Übung GW2
}
```