Gekoppelte Simulation zwischen hygrothermischem Transportmodell DELPHIN und Python-basierter Anlagen- und Regelungstechnik unter Verwendung der Wave-Form-Relaxationsmethode

Technischer Bericht

Forschungsprojekt: Erdeisspeicher und oberflächennahe Geothermie
Förderkennzeichen: 03ET1382A

Andreas Nicolai*
Rüdiger Heyn

13. Februar 2019

Zusammenfassung

Der Anwendungsfall ist der Einsatz oberflächennaher Geomethermie. Das Simulationsprogramm DELPHIN erlaubt eine detaillierte Abbildung von physikalischen Vorgängen in Materialien, Bauteilen, einschließlich Böden und der klimatischen Bewitterung, und wird für die Modellierung des oberen Erdreiches und Kollektorrohs verwendet. DELPHIN unterstützt eine flexible Modellierung und erlaubt unter anderem die Verwendung von beliebig definierten Zeitreihen (Variablen als Funktion der Zeit) als Eingangsgrößen. Rückwirkungen der Simulationsergebnisse auf Eingangsdaten sind jedoch nur für typische, häufig verwendete Modelle implementiert. Soll eine komplexere Interaktion mit einem externen Modell stattfinden, beispielsweise eine Anlagenregelung, welche je nach Simulationsbedingungen die größe einer Wärmequellen anpasst, kann eine Simulationskopplungstechnik genutzt werden. Der Artikel beschreibt einen technischen Lösungsansatz (die Wave-Form-Relaxationsmethode), die Arbeitsschritte zum Aufsetzen einer solchen Bibliothek (unter Verwendung bereitgestellter Python-Skripte) und relevante Parameter.

*Institut für Bauklimatik, Technische Universität Dresden, andreas.nicolai@tu-dresden.de

Inhaltsverzeichnis

1 Einleitung

Im vorliegenden Tutorial soll anhand eines einfachen Beispiels die Kopplung von DELPHIN 6 an andere Simulationsprogramme mithilfe der Wave-Form-Relaxations-Methode beschrieben werden. Grundidee ist die abwechselnde Ausführung separater Simulationsmodelle, welche jeweils über Ein- und Ausgabedateien kommunizieren. Während der Simulations erfolgt *kein* Datenaustausch. Dafür werden sowohl DELPHIN als auch die Python-basierte Simulationen mehrfach für den gesamten Zeitbereich ausgeführt und die jeweiligen Ausgabedateien ausgetauscht, bis sich Konvergenz einstellt. Konkret werden Ergebnisse der DELPHIN Simulation als Eingangsdaten für das Python-Modell verwendet und Ergebnisse der Python-Berechnung werden DELPHIN zur Verfügung gestellt. Ergebnisse sind hierbei stets *Zeitreihen*. Ein Python-Script kontrolliert das wiederholte Anstoßen der Simulation und den Datenaustausch.

In diesem Tutorial wird ein Erdboden mittels Erdkollektor abgekühlt. Das bewitterte Erdreich wird im DELPHIN 6 Modell abgebildet. Die Anlagenseite wird in einer in Python geschrieben Funktion umgesetzt. Nachfolgend werden die einzelnen Schritte erläutert, um eine gekoppelte Simulation auszuführen.

Zum Verständnis des Tutorials wird ausreichende Kenntnis des Simulationsprogramms DELPHIN 6 vorausgesetzt. Desweiteren sollten rudimentäre Kenntnisse der Scriptsprache Python vorhanden sein.

2 Vorbereiten des DELPHIN Simulationsmodells

In diesem Beispiel wird ein Erdreichmodell in DELPHIN erstellt (Abbildung 1). Das Erdreich wird durch einen Erdkollektor abgekühlt. Der Erdkollektor ist als Schicht idealisiert, welche vom Kühlmedium von links nach rechts durchflossen wird.

Abbildung 1: Prinziskizze des Simulationsmodell

Die Einlasstemperatur des Rohres soll eine zeitabhängig geregelte Größe sein. Daher wird eine Klimabedingung `Einlasstemperatur` definiert, welche auf eine Klimadatei im tsv-Format verweist. Die Klimabedingung wird durch „Datenpunkte“ definiert und verweist auf eine Klimadatei, entweder im Format ccd oder als Tabulator-getrennte Datei, z.B. mit folgendem Format:

```
1 Time [d]   Temperature [C]
2 0          5
3 364.9      5
```

Alternativ kann auch eine Klimadatei im CCD Format verwendet werden (erfordert jedoch Anpassungen im Python-Script unten). Bei diesem Format ist in der ersten Zeile jeweils die physikalische Größe gefolgt von der Einheit in eckigen Klammern anzugeben. Die einzelnen Spalten sind tabulatorgetrennt anzugeben. Der eigentliche Inhalt der Datei wird nur als Anfangszeitverlauf verwendet und daher nicht von Bedeutung für das Endergebnis. Der Pfad der Klimadatei, z.B. `C:\projects\eisspeicher\T_in.ccd` ist für die spätere Kopplung wichtig.

Die Simulation kann nun zu Testzwecken durchgeführt werden. Für die Funktion der angekoppelten Wärmepumpe ist die Rücklauftemperatur wichtig. Diese wird als Ausgabegröße in DELPHIN am Rohrende als skalare Ausgabevariable definiert und in die Datei `C:\projects\eisspeicher\t_out.d6o` geschrieben. Dabei ist zu beachten, dass das Pythonscript folgende Erwartungen an die Variablen in der Ausgabedatei hat:

- Ausgabezeiteinheit ist Stunde [h]
- Ausgabewerteeinheit ist Grad Celsius [C]

Nachdem die Simulation so erstellt und getested wurde, sind folgende Pfade für die Python-Scriptdatei relevant:

- Pfad zur Klimadatei (Zeitreihe als Eingabe in DELPHIN Simulationsmodell):
  `C:\projects\eisspeicher\T_in.ccd`

- Pfad zur Ausgabedatei (Zeitreihe als Eingabe in Python Simulationsmodell):
  `C:\projects\eisspeicher\t_out.d6o`
- Pfad zur Projektdatei:
  `C:\projects\eisspeicher\eisspeicher.d6p`

3 Anpassen des Co-Simulations-Python-Scripts

In diesem Tutorial wird eine Python-Scriptvorlage python_cosim_template.py angepasst und zunächst in das Projektverzeichnis unter dem Namen `C:\projects\eisspeicher\eisspeicher.py` kopiert.

3.1 Definition der Programmkonstanten

Im Python-Script müssen zunächst die Pfade und Ausführungskonstanten angepasst werden. Dies geschieht am Anfang der Datei, entsprechend der Quelltextkommentare.

```
1  # Define path to DELPHIN Solver executable, preset assumes a standard Windows installation
2  DELPHIN_SOLVER = r'C:\Program Files\IBK\Delphin 6.0\DelphinSolver.exe'
3
4  # Define DELPHIN project file (xxx.d6p)
5  DELPHIN_PROJECT_FILE = 'eisspeicher.d6p'
6
7  # Define output directory for this simulation run
8  DELPHIN_OUTPUT_DIR = 'eisspeicher'
9
10 # Define the output file that we want to change and use for the input of the next iteration
11 # Specify here the path to the DELPHIN output file without extension (so that _xxx.d6o is appended)
12 DELPHIN_OUTPUT_FILE = DELPHIN_OUTPUT_DIR+'/results/t_out'
13
14 # Define if convergence check shall be used
15 # If CHECK_FOR_CONVERGENCE is True, we check the last value in the output file and compare it to the
       previous run,
16 # and if the difference is smaller than CONVERGENCE_THRESHOLD, the iteration is stopped.
17 CHECK_FOR_CONVERGENCE = True
18 # Define convergence threshold in DegC, to check convergence and finish iteration if under threshold
19 CONVERGENCE_THRESHOLD = 0.01
20 # Define max number of iterations
21 MAX_ITERATIONS = 50
```

Listing 1: Python-Script-Konstanten

Folgende Konstanten sind zu setzen:

- `DELPHIN_SOLVER` - Pfad zum DELPHIN Solver
- `DELPHIN_PROJECT_FILE` - Pfad zur Projektdatei, entweder als relativer Pfad zum Arbeitsverzeichnis oder als absoluter Pfad (in diesem Beispiel `C:\projects\eisspeicher\eisspeicher.d6p`)
- `DELPHIN_OUTPUT_DIR` - Pfad zu den Simulationsergebnissen, sinnvoll anzupassen wenn Variationsstudien gerechnet werden sollen (z.B. bei Änderungen der Berechnungsfunktionalität im Pythonscript)
- `DELPHIN_OUTPUT_FILE` - Pfad zur Ausgabedatei, sollte relative zu `DELPHIN_OUTPUT_DIR` definiert werden.

Weiterhin sind Konstanten für das Iterationsverfahren zu setzen, wobei die Standardwerte sinnvolle Voreinstellungen sind. Wichtig ist, dass die Konstante CONVERGENCE_THRESHOLD als physikalische Größe in der Einheit der von DELPHIN generierten Ergebnisgröße ist, hier also die Rohrrücklauftemperatur in Grad Celsius.

3.2 Anpassung der Python-Regelungs/Simulationsfunktion

Als nächstes muss die Funktionalität für die Berechnungsfunktion im Python-Skript angepasst werden. In der Vorlage-Datei ist das die Funktion `run_HEAT_EXCHANGER_model(t, T)`. Diese Funktion erhält die Rohrauslasstemperatur, gelesen aus der Ergebnisdatei `t_out.d6o` und berechnet daraus eine Zeitreihe der neuen Vorlauftemperaturen, entsprechend der integrierten Regelungsvorschrift. Die Zeitreihe wird dabei als Datentabelle betrachtet, wobei für jeden Zeitpunkt `t` eine Temperatur `T` berechnet wird (die Python Funktionsargumente sind Arrays).

Beispielhaft ist die Funktion gezeigt:

```
def run_HEAT_EXCHANGER_model(t, T):
    """
    Simple heat pump model

    Arguments:

    * t - Expects argument t to be a time series in [h]
    * T - time series of temperatures in [C] for the corresponding time points in vector t
    """

    # defines the maximum reduction of temperature between outlet and inlet
    DELTA_T = -2

    # process all time points in the input time series
    for i in range(len(T)):
        # Apply control algorithm:
        # - heat pump only operates between 06:00 and 18:00
        # - above 0 degree C a constant delta is applied
        #    T[i] += DELTA_T
        # - below -5 degree C heat pump is turned off
        # - linear decrease of cooling load between 0 and -5
        #    T[i] += delta(T[i])
        hour_of_day = round((t[i] * 24.0) % 24, 0)
        if 6 <= hour_of_day <= 18:
            t_in = T[i]
            if t_in < -5:
                delta = 0
            if t_in > 0:
                delta = DELTA_T
            delta = (- 0.4 * t_in + DELTA_T)
            T[i] = T[i] + delta # modify temperature for this time point
    return (t,T)
```

Listing 2: Wärmepumpenfunktion

Wichtig ist in diesem Script, dass die vom DELPHIN generierten Zeitreihen mit der Zeiteinheit Stunde [h] und in der Werteeinheit [C] ausgegeben werden. Ansonsten müssen entsprechende Einheitenkonvertierungen in das Script eingebaut werden.

3.3 Kopieren der Anfangsbedingung

Die Simulation beginnt mit der Ausführung der DELPHIN Simulation, welche die Klimadatei `T_in.csv` benötigt. Nach jeder Iteration des Co-Simulationsalgorithmus wird diese Datei neu geschrieben. Beim Neustart der Simulation wird daher diese Datei aus einer Kopie `T_in_0.csv` wiederhergestellt. Daher sollte vor Start der Simulation die Datei `T_in.csv` nach `T_in_0.csv` kopiert werden. Da die Anfangsbedingung jedoch keinen Einfluss auf das konvergierte Ergebnis hat (bzw. haben sollte), kann man eine neue Variationsrechnung auch mit einer alten `T_in.csv` Datei beginnen.

4 Ausführen der Simulation

Die Simulation wird durch Starten des angepassten Python-Scripts eisspeicher.py ausgeführt. Das führt dabei in der Schleife folgende Operationen aus:

1. Erstellen der Datei `T_in.csv` (außer beim ersten Iterationsschritt)
2. Starten der DELPHIN Simulation
3. Kopieren der generierten Ausgabedatei T_out.d6o nach T_out_xx.d6o, wobei xx eine fortlaufende Nummer darstellt. Im Ausgabeverzeichnis liegen abschließend die Zwischenergebnisse der Rücklauftemperaturen für jeden Iterationsschritt.
4. Einlesen der Ergebnisdatei in Vektoren t und T
5. Ausführen der Wärmepumpen/Regelungsfunktion und Generierung der neuen Eingangszeitreihe für DELPHIN
6. Durchführen des Konvergenztests (wenn eingeschaltet, siehe Konstantendefinition oben) und Beenden der Simulation im Fall von Konvergenz