Pseudocode for FSF model

Numerical Method Assignment

Shaoxu Zheng (1064735), Water Science & Engineering (HI), IHE-Delft, Institute for Water Education, Delft, the Netherlands.

Contents

1. main.py

The argument represents one of the case that you run the code for:

Table 1. The test dataset's argument

1--static_test_1	2--static_test_2
3--static_test_3	4--steady_test
5--transient_test_1_a	6--transient_test_1_b
7--transient_test_2_a	8--transient_test_2_b
9--transient_test_3_a	10--transient_test_3_b
11--seiche_test	

This part is the main part of the model. It is calculated by introducing fsf_engine, and then the data is displayed through two modules – fsf_report and fsf_gif. The last file is used to test the boundary data and store it in the report folder. It can be used in conjunction with the main part or can be used alone for test testing.

Table 2. Key variables in main.py

Variable	Meaning
file_num	Referenced test number
filename	Specific test filename
flag_method	Referential method (matrix inversion and double sweep)
h0,q0	Test's upstream boundary dataset
h_end,q_end	Test's downstream boundary dataset

```
# ================================================================================
# Input the dataset
# ================================================================================
# Choose the test dataset
[file_num,filename] ← fsf_input.input_file()
# Read the dataset
read_data ← fsf_input.read(filename)


# ================================================================================
# Calculation
# ================================================================================
# Choose the calculating method (matrix inversion or double sweep)
flag_method ← fsf_input.flag_method()
read_data.append(flag_method)
# Calculation module
[Q,H,X_series,T_series] ← fsf_engine.calculation(read_data)


# ================================================================================
# Plot the graph
# ================================================================================
# plot the graph for the chosen grid point
fsf_report.graph(Q,H,X_series,T_series)
# plot the dynamic graph with the time changing
fsf_gif.gif(Q,H,X_series,T_series)


# ================================================================================
# Return a vector for all time steps in both boundaries
# ================================================================================
# Choose whether output the boundary dataset
```

```
flag_boundary ← fsf_report.flag()
if flag_boundary == 1 then
    # Print the current test's boundary dataset: h0,q0,h_end,q_end
    # Then save in the report file
    h0 ← test_module.test_h0([file_num,flag_method])
    q0 ← test_module.test_q0([file_num,flag_method])
    h_end ← test_module.test_h_end([file_num,flag_method])
    q_end ← test_module.test_q_end([file_num,flag_method])
end if
```

2. fsf_input.py

The module aims to read the specific data in the dataset and put it into the variable of the application. At the same time, the code judges the validity of the data and the existence of empty data.

Table 3. Key variables in fsf_input.py

Variable	Meaning
upstream	The data in Upstream boundary table
downstream	The data in Downstream boundary table
initial	The data in Initial condition table
geometry	The data in cancel table
Parameters	The data in numerical parameters

```
function read(filename)
    # Intercept the table part (note: all the element is string)
    function table(string)
        value = []
        position ← data.find(string)
        if position == 0 then
            get the total table part and store it
        end if
        return value
    end function

    # Obtain each table value
    function obtain(dataset,string)
        if dataset == []:
            dataset ← table(string)
        return dataset
    end function

    # Get the variable name (Each table name)
    function Variable_name(name, locals)
        # print(locals)
        for key in locals:
            if locals[key] == name:
                return key
    end function

    strings= ['\tTable.1','\tTable.2','\tTable.3','\tTable.4','\tTable.5']
    # open the txtfile
    with open(filename,'r') as fname:
        data_all = fname.readlines()
        # flag = 0 -- h
        # flag = -1 -- Q
        flag_up ← -1
        flag_down ← -1
        while data_all
            data ← data_all.pop(0)
            determine the flag_up and flag_down ← 0 or -1
            # Obtain the table part
            upstream,downstream,initial,geometry,parameters ← obtain(..,strings)
        end while

    # Check the validity of each element of data and pop up error
    check_valid()

    # Assign variable names to data
    # Upstream boundary
    H_up, Q_up

```

```
    # Downstream boundary
    H_down, Q_down

    # Initial boundary
    H_ini,Q_ini

    # The data of the cancel
    # Channel length,Contant channel width,Bed slope,Chezy resistance coefficient
    L,b,S0,C

    # The numerical parameters
    # Computational time step,Total time,Cell size,Time weighting parameter
    # Space weighting parameter,Coefficient,Max No. of iteration,Convergence criteria, Gravity
    t,T,x,theta,psi,beta,Iter,Err,g

    result ← [flag_up,flag_down,H_up,Q_up,H_down,Q_down,H_ini,Q_ini,L,b,S0,C,t,T,x,theta,psi,beta,Iter,Err,g]
    return result
end function
```

3. fsf_engine.py

This part contains two algorithms, and the algorithms can be applied separately. Both methods are applied with the idea of iteration to calculate.

Table 4. Key variables in matrix inversion

Variable	Meaning
Coef	The matrix coefficient
En	The equation right term
QH	Discharge and water level
Error	The error compared with previous one
Iteration	The total iteration times

Table 5. Key variables in double sweep

Variable	Meaning
Q	Discharge
H	Water level
Error	The error compared with previous one
Iteration	The total iteration times

```
# ============================================================================
# Matrix inversion
# ============================================================================
# Algorithm for each row
function calculation_mi(n,qh)
    for m 1 to M-1 by 1 do
        # Assign four points
        q_j,h_j,q_j1,h_j1 ← QH
        h1_j,h1_j1 ← qh
        # Calculate the coefficients
        calculate Coef, En
    end for
    # Matrix calculation
    QH_test ← linalg.solve(Coef[n,:,:], En[:,n])
    return QH_test
end function

# Root Mean Square Error
function RMSE_mi(qh_1,qh,Iteration)
    Error += (qh_1[i] - qh[i])**2
    Error ← np.sqrt(Error/(2*M))
    Iteration += 1
    return [Error,Iteration]
end function

# Main code for matrix inversion
function matrix_inversion()
    # Initial the value
    Coef, En, QH

    # Coefficient in both boundaries
    Coef[:,0,1] or Coef[:,0,0] ← 1
    Coef[:,2*M-1,2*M-1] or Coef[:,2*M-1,2*M-2] ← 1

    # Initial condition
    En[0,:],En[2*M-1,:] ← np.interp( )
```

```
    QH[:,0] ← np.interp( )

    # Iterative solution of nonlinear equations
    for n 1 to N-1 by 1 do
        qh ← QH[:,n-1]
        Error ← 1
        Iteration ← 0
        while Error > Err and Iteration <= Iter do
            qh_1 ← calculation_mi(n,qh)
            [Error,Iteration] ← RMSE_mi(qh_1,qh,Iteration)
            qh ← qh_1
        end while
        QH[:,n] ← qh
    end for

    # Transpose the matrix
    Q,H ← QH
    return [Q,H,X_series,T_series]
end function



# ================================================================================
# Double sweep algorithm
# ================================================================================
# Algorithm for each row
function calculation_ds(n,q,h)
    # Initial the value
    q_1,h_1,E1,A2,B2,C2,D2,E2,R,i,J,F,G

    # Forward sweep
    # Upstream boundary condition
    if flag_up == 0 then # Water level
        F[0] ← 10**6
        G[0] ← -F[0] * H_up
    else # discharge
        F[0] ← 0
        G[0] ← Q_up
    end if

    for m in range(M-1):
        # Assign four points
        q_j,q_j1 ← Q
        h_j,h_j1 ← H
        h1_j,h1_j1 ← h
        # Calculate the coefficient
        Calculate E1,A2,B2,C2,D2,E2,R,I,J,F,G
    end for

    # Downstream boundary condition
    if flag_down == 0 then # Water level
        h_1[M-1] ← H_down
        q_1[M-1] ← F[M-1] * h_1[M-1] + G[M-1]
    else # Discharge
        q_1[M-1] ← Q_down
        h_1[M-1] ← (q_1[M-1]-G[M-1])/F[M-1]
    end if
    # Backward sweep
    for m M-1 to 1 by -1 do
        h_1[m-1] ← R[m-1] * q_1[m] + I[m-1] * h_1[m] + J[m-1]
        q_1[m-1] ← F[m-1] * h_1[m-1] + G[m-1]
    end for
    result = [q_1,h_1]
            return result
       end function

```

```
103.
104.       # Root Mean Square Error
105.       function RMSE_ds(q_1,h_1,q,h,Iteration)
106.           Error += (q_1[i] - q[i])**2 + (h_1[i] - h[i])**2
107.           Error ← np.sqrt(Error/(2*M))
108.           Iteration += 1
109.           return [Error,Iteration]
110.       end function
111.
112.       # Main code for double sweep algorithm
113.       function double_sweep()
114.           # Initial value
115.               Q[0,:],H[0,m]
116.           # Iterate and compare with the Err
117.           for n 0 to N-2 do
118.               q,h ← Q,H
119.               Iteration ← 0
120.               Error ← 1
121.               while Error > Err and Iteration <= Iter do
122.                   [q_1,h_1] ← calculation_ds(n,q,h)
123.                   [Error,Iteration] = RMSE_ds(q_1,h_1,q,h,Iteration)
124.                   q,h ← q_1,h_1
125.               end while
126.               Q[n+1,:],H[n+1,:] = q,h
127.           return [Q,H,X_series,T_series]
128.       end function
129.
130.
131.       # ================================================================================
132.       # Main code imported in the main.py
133.       # ================================================================================
134.       function calculation(read_data)
135.           # Initial value
136.           [flag_up,flag_down,H_up,Q_up,H_down,Q_down,H_ini,Q_ini,L,b,S0,C,t,T,x,th
    eta,psi,beta,Iter,Err,g,flag_method] ← read_data
137.           M,N,X_series,T_series,A1,B1,C1,D1,Q,H
138.
139.           # Choose method
140.           if flag_method == 1 then
141.               result ← matrix_inversion()
142.           else
143.               result ← double_sweep()
144.           end if
145.           return result
146.       end function
```

4. fsf_report.py

This module is mainly responsible for export the data graph. Users can choose export as water level or discharge. They can also choose time or distance as the abscissa. After that, users will customize output filename.

Table 6. Key variables in fsf_report.py

Variable	Meaning
flag_qh	Choose between discharge and water level
flag_xt	Choose between grid point and time point

```
# ================================================================================
# Input fuction, let the user choose
# ================================================================================
function output_file(file_type)
    Fill in filename
    return filename
end function

function flag()
    Flag means choosing answer
    return flag
end function


def number(string)
    Let the user fill in the number
    return num
end function


# ================================================================================
# Main code imported in the main.py
# ================================================================================
function graph(Q,H,X_series,T_series)
    # Choose between discharge and water level
    flag_qh ← flag()

    # Choose between grid point and time point
    flag_xt ← flag()

    num ← int(number('\nPlease type the distance(m):\t')/x_step)

    if flag_qh, flage_xt then
        # Accodring to different flag_qh and flag_xt build different graph
        Title,Xlabel
        Y,X ← Q[:,num],T_series
    end if

    Plot the graph
end function
```

5. fsf_gif.py

This module implements the function of outputting dynamic pictures. User can freely choose water level and discharge and name it later.

Table 7. Key variables in fsf_gif.py

Variable	Meaning
flag_judge	Choose between discharge and water level

```
# ================================================================================
# Input fuction, let the user choose
# ================================================================================

function output_file(file_type)
    Fill in filename
    return filename
end function

function flag()
    Flag means choosing answer
    return flag
end function

function process(N,label,title,Y,X_series,string)
    # Produce dynamic graph
    ims = []
    for n 0 to N-1 by 1 do
        im ← plt.plot(X_series,Y[10*n])
        ims.append(im)
    end if
    ani ← animation.ArtistAnimation(fig, ims, interval=200, repeat_delay=1000)
    filename ← output_file(string,'.gif')
    ani.save(filename,writer=animation.PillowWriter(fps=10))
    plt.close()
end function


# ================================================================================
# Main code imported in the main.py
# ================================================================================
function gif(Q,H,X_series,T_series)
    flag_judge ← flag()
    if flag_judge == 1 then
        N ← int(len(T_series)/10)
        # Discharge graph
        process(N,'Discharge (m^3/s)','Discharge changing with distance',Q,X_series
,'Discharge dynamic graph.Please fill in the filename:\t')
        # Water level graph
        process(N,'Water level (m)','Water level changing with distance',H,X_series
,'Water level dynamic graph.Please fill in the filename:\t')
    end if
end function
```

6. tests_module.py

This part is used as a test block to check whether the data calculation is correct. There are two methods to call it. One is to run the main program directly. The other is to run test_module and then operate in the control panel. The data will eventually be saved in the report folder.

E.g. test_h0(*[flag_num,flag_method])

```
In [3]: test_h0()

Please type the test number you want. (1~12):   5

Now you should choose which method you apply, whether matrix
inversion or double sweep.

For matrix inversion, type 1; for double sweep, type 2.Please type
the number: 1
Out[3]:
array([13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8,
13.8,
       13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8,
13.8,
       13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8,
13.8,
       13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8,
```

Figure 1. test_h0()

```
In [4]: test_h0([5,1])
Out[4]:
array([13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8,
13.8,
       13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8,
13.8,
       13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8,
13.8,
       13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8,
13.8,
       13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8,
13.8,
       13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8,
13.8,
       13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8,
13.8,
       13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8, 13.8,
```

Figure 2. test_h0([5,1])

```
'''''
The user has two ways to call this program
In the main.py, call it and obtain the current txtfile data
Directly run this module and get the data
file_num -- argument represents one of the case that you run the code
flag_method -- Choosing between matrix inversion and doubel sweep

E.g.
test_h0([5,1]) -- test the fifth dataset and apply matrix inversion
The result will save in the report folder
'''

# ============================================================================
# Read the data in the required txtfile
# ============================================================================
function data(*information)
    if information == () then
        [file_num,filename] ← input_module.input_file()
        flag_method ← input_module.flag_method()
    else
        [file_num,flag_method] ← information[0]
        filename ← '../inputs/'+ str(file_num)+'.txt'
```

```
    end if
    read_data ← input_module.read(filename)
    read_data.append(flag_method)
    data ← engine_module.calculation(read_data)
    data.append(file_num)
    return data
end function


# ================================================================================
# Main code imported in the main.py
# ================================================================================
# Water level in upstream boundary
function test_h0(*information)
    result ← data(*information)
    test_h0 ← result[1][0]
    txtname ← '..\\report\\test_h0_'+ str(result[-1]) +'.txt'
    np.savetxt(txtname,test_h0, fmt='%.02f')
    return test_h0
end function

# Discharge in upstream boundary
fuction test_q0(*information)

# Water level in downstream boundary
function test_h_end(*information)

# Water level in downstream boundary
function test_q_end(*information)
```