function [ fluxOutput, fluxInternal, storeInternal, waterBalance ] = ...

m\_nn\_example\_7p\_3s( fluxInput, storeInitial, theta, solver )

% Hydrologic conceptual model: [MARRMoT User Manual example model]

%

% Model reference

% MARRMoT User Manual, 2018.

% Steps

% --- Practical ---

% 0. Handle inputs

%

% --- Model setup ---

% 1. Set out ODE

% 2. Set out constitutive functions

% 3. Determine smoothing

% 4. Determine numerical scheme

% --- Model use ---

% 5. Solve

% --- Practical ---

% 6. Handle outputs

%% Setup

%%INPUTS

% Time step size

delta\_t = fluxInput.delta\_t;

% Data

P = fluxInput.precip./delta\_t; % [mm/delta\_t] -> [mm/d]

Ep = fluxInput.pet./delta\_t; % [mm/delta\_t] -> [mm/d]

T = fluxInput.temp;

t\_end = length(P);

% Parameters

% [name in documentation] = theta(order in which specified in parameter file)

crate = theta(1); % Maximum capillary rise rate [mm/d]

uzmax = theta(2); % Maximum upper zone storage [mm]

prate = theta(3); % Maximum percolation rate [mm/d]

klz = theta(4); % Lower zone runoff coefficient [d-1]

alpha = theta(5); % Fraction of lower zone runoff to groundwater [-]

kg = theta(6); % Groundwater runoff coefficient [d-1]

d = theta(7); % Routing delay [d]

%%INITIALISE MODEL STORES

S10 = storeInitial(1); % Initial upper zone storage

S20 = storeInitial(2); % Initial lower zone storage

S30 = storeInitial(3); % Initial groundwater storage

%%DEFINE STORE BOUNDARIES

store\_min = [0,0,0]; % lower bounds of stores

store\_upp = []; % optional higher bounds

%%INITIALISE STORAGE VECTORS

store\_S1 = zeros(1,t\_end);

store\_S2 = zeros(1,t\_end);

store\_S3 = zeros(1,t\_end);

flux\_qse = zeros(1,t\_end);

flux\_e = zeros(1,t\_end);

flux\_qp = zeros(1,t\_end);

flux\_qc = zeros(1,t\_end);

flux\_qlz = zeros(1,t\_end);

flux\_qf = zeros(1,t\_end);

flux\_qg = zeros(1,t\_end);

flux\_qs = zeros(1,t\_end);

flux\_qt = zeros(1,t\_end);

%%PREPARE UNIT HYDROGRAPHS

[~,uh\_full] = uh\_4\_full(1,d,delta\_t);

%%INITIALISE ROUTING VECTORS

tmp\_Qt\_old = zeros(1,length(uh\_full));

%% 1. ODEs

% Given in the documentation.

%% 2. Constitutive functions

% Given in the documentation

%% 3. Specify and smooth model functions

% Store numbering:

% S1. Upper zone

% S2. Lower zone

% S3. Groundwater

% Model smoothing

% With Matlab's fsolve, smoothing is only needed when the function is

% undefined (i.e. has thresholds). Angle discontinuities (such as from

% min(0,x)) can be dealt with by the solver. Thus, threshold

% discontinuities are smoothed with a logistic function (e.g. Kavetski and

% Kuczera, 2007) with default smoothing parameters (Clark et al, 2008).

%

% Kavetski and Kuczera, 2007. Model smoothing strategies to remove

% microscale discontinuities and spurious secondary optima in objective

% functions in hydrological calibration. Water Resources Research, 43,

% W03411, doi:10.1029/2006WR005195.

%

% Clark, Slater, Rupp, Woods, Vrugt, Gupta, Wagener and Hay, 2008.

% Framework for Understanding Structural Errors (FUSE): A modular framework

% to diagnose differences between hydrological models. Water Resources

% Research, 44, doi:10.1029/2007/WR006735.

% E(S1,uzmax,Ep(t),delta\_t): evaporation from upper zone (S1).

E = evap\_7;

% QSE(P(t),S1,uzmax): saturation excess from upper zone (S1).

% Has a threshold discontinuity and needs logistic smoothing

QSE = saturation\_1;

% QP(prate,S1,delta\_t): percolation from upper zone (S1) to lower zone (S2)

QP = percolation\_1;

% QC(crate,S1,uzmax,S2,delta\_t): capillary rise from lower (S2) to upper

% zone (S1)

QC = capillary\_1;

% QLZ(klz,S2): outflow from lower zone (S2)

QLZ = baseflow\_1;

% QF(1-alpha,QLZ(klz,S2)): fraction (1-alpha) of lower zone outflow (QLZ)

% that is fast flow

QF = split\_1;

% QG(alpha,QLZ(klz,S2)): fraction (alpha) of lower zone outflow (QLZ) that

% goes to groundwater (S3)

QG = split\_1;

% QS(kg,S3): outflow from groundwater (S3)

QS = baseflow\_1;

%% 4. Determine numerical scheme and solver settings

% Function name of the numerical scheme

scheme = solver.name;

% Define which storage values should be used to update fluxes

[~,store\_fun] = feval(scheme,storeInitial,delta\_t);

% Root-finding options

fsolve\_options = optimoptions('fsolve','Display','none',...

'JacobPattern', [1,1,0;

1,1,0;

0,1,1]);

lsqnonlin\_options = optimoptions('lsqnonlin',...

'Display','none',...

'JacobPattern', [1,1,0;

1,1,0;

0,1,1],...

'MaxFunEvals',1000);

% Prepare the options for the solver (saves time later)

[fsolve\_options,optionFeedback] = ...

prepareOptionsForSolver(fsolve\_options, 'fsolve');

%% 5. Solve the system for the full time series

for t = 1:t\_end

% Model setup -------------------------------------------------------------

% Determine the old storages

if t == 1; S1old = S10; else; S1old = store\_S1(t-1); end

if t == 1; S2old = S20; else; S2old = store\_S2(t-1); end

if t == 1; S3old = S30; else; S3old = store\_S3(t-1); end

% Create temporary store ODE's that need to be solved

tmpf\_S1 = ...

@(S1,S2,S3) ... % Change in S1 depends on ...

(P(t) + ... % Precipitation to S1 +

QC(crate,S1,uzmax,S2,delta\_t) - ... % Capillary rise to S1 -

E(S1,uzmax,Ep(t),delta\_t) - ... % Evaporation from S1 -

QSE(P(t),S1,uzmax) - ... % Surface runoff from S1 -

QP(prate,S1,delta\_t)); % Percolation from S1

tmpf\_S2 = ...

@(S1,S2,S3) ... % Change in S2 depends on ...

(QP(prate,S1,delta\_t) - ... % Percolation to S2 -

QC(crate,S1,uzmax,S2,delta\_t) - ... % Capillary rise from S2 -

QLZ(klz,S2)); % Lower zone outflow from S2

tmpf\_S3 = ...

@(S1,S2,S3) ... % Change in S2 depends on ...

(QG(alpha,QLZ(klz,S2)) - ... % Recharge to S3 -

QS(kg,S3)); % Slow flow from S3

% Call the numerical scheme function to create the ODE approximations.

% This returns a new anonymous function that we solve in the next step.

solve\_fun = feval(scheme,... % time-stepping function

[S1old,S2old,S3old],... % Store values at t-1

delta\_t,... % time step size

tmpf\_S1,tmpf\_S2,tmpf\_S3); % anonymous functions of ODEs

% Model solving -----------------------------------------------------------

% --- Use the specified numerical scheme to solve storages ---

[tmp\_sNew,tmp\_fval] = fsolve\_noMSG(@(eq\_sys) solve\_fun(...

eq\_sys(1),eq\_sys(2),eq\_sys(3)),...

[S1old,S2old,S3old],...

fsolve\_options,optionFeedback);

% --- Check if the solver has found an acceptable solution and re-run

% if not. The re-run uses the 'lsqnonlin' solver which is slower but

% more robust. It runs solver.resnorm\_iterations times, with different

% starting points for the solver on each iteration ---

tmp\_resnorm = sum(tmp\_fval.^2);

if tmp\_resnorm > solver.resnorm\_tolerance

[tmp\_sNew,~,~] = rerunSolver('lsqnonlin', ...

lsqnonlin\_options, ...

@(eq\_sys) solve\_fun(...

eq\_sys(1),eq\_sys(2),...

eq\_sys(3)), ...

solver.resnorm\_maxiter, ...

solver.resnorm\_tolerance, ...

tmp\_sNew, ...

[S1old,S2old,S3old], ...

store\_min, ...

store\_upp);

end

% Model states and fluxes -------------------------------------------------

% This line creates/updates a variable called 'tmp\_sFlux' which is used

% to update the model fluxes for the current time step. Which variables

% get assigned to 'tmp\_sFlux' is a feature of the chosen numerical time

% stepping scheme (see line 133-134).

eval(store\_fun);

% Calculate the fluxes

flux\_qse(t) = QSE(P(t),tmp\_sFlux(1),uzmax);

flux\_e(t) = E(tmp\_sFlux(1),uzmax,Ep(t),delta\_t);

flux\_qp(t) = QP(prate,tmp\_sFlux(1),delta\_t);

flux\_qc(t) = QC(crate,tmp\_sFlux(1),uzmax,tmp\_sFlux(2),delta\_t);

flux\_qlz(t) = QLZ(klz,tmp\_sFlux(2));

flux\_qf(t) = QF(1-alpha,flux\_qlz(t));

flux\_qg(t) = QG(alpha,flux\_qlz(t));

flux\_qs(t) = QS(kg,tmp\_sFlux(3));

% Update the stores

store\_S1(t) = S1old + (P(t) + flux\_qc(t) - flux\_e(t) - ...

flux\_qse(t) - flux\_qp(t)) \* delta\_t;

store\_S2(t) = S2old + (flux\_qp(t) - flux\_qc(t) - ...

flux\_qlz(t)) \* delta\_t;

store\_S3(t) = S3old + (flux\_qg(t) - flux\_qs(t)) \* delta\_t;

% Routing -----------------------------------------------------------------

% Total runoff Q = Qse + Qf + Qs. Apply a pre-determined (line 82)

% triangular Unit Hydrograph routing scheme to find lagged flow Qt.

tmp\_Qt\_cur = (flux\_qse(t) + flux\_qf(t) + flux\_qs(t)).\*uh\_full;

tmp\_Qt\_old = tmp\_Qt\_old + tmp\_Qt\_cur;

flux\_qt(t) = tmp\_Qt\_old(1);

tmp\_Qt\_old = circshift(tmp\_Qt\_old,-1);

tmp\_Qt\_old(end) = 0;

end

%% 6. Generate outputs

% --- Fluxes leaving the model ---

% 'Ea' and 'Q' are used outside the

% funcion and should NOT be renamed

fluxOutput.Ea = flux\_e \* delta\_t;

fluxOutput.Q = flux\_qt \* delta\_t;

% --- Fluxes internal to the model ---

fluxInternal.qse = flux\_qse \* delta\_t;

fluxInternal.qp = flux\_qp \* delta\_t;

fluxInternal.qc = flux\_qc \* delta\_t;

fluxInternal.qlz = flux\_qlz \* delta\_t;

fluxInternal.qf = flux\_qf \* delta\_t;

fluxInternal.qg = flux\_qg \* delta\_t;

fluxInternal.qs = flux\_qs \* delta\_t;

% --- Stores ---

storeInternal.S1 = store\_S1;

storeInternal.S2 = store\_S2;

storeInternal.S3 = store\_S3;

% Check water balance

if nargout == 4

waterBalance = ...

checkWaterBalance(...

P,... % Incoming precipitation

fluxOutput,... % Fluxes Q and Ea leaving the model

storeInternal,... % Time series of storages ...

storeInitial,... % And initial store values to calculate delta S

tmp\_Qt\_old); % Whether the model uses a routing scheme that

% still contains water. Use '0' for no routing

end