Supplement of Geosci. Model Dev., 7, 2313–2332, 2014
http://www.geosci-model-dev.net/7/2313/2014/
doi:10.5194/gmd-7-2313-2014-supplement

Geoscientific Model Development
Open Access

Supplement of

The Wageningen Lowland Runoff Simulator (WALRUS): a lumped rainfall–runoff model for catchments with shallow groundwater

C. C. Brauer et al.

Correspondence to: C. C. Brauer (claudia.brauer@wur.nl)

1 Introduction

In this supplementary document we provide the main code for the Wageningen Lowland Runoff Simulator (WALRUS). The code is written in R and will be made available as an R package (with additional pre- and postprocessing scripts) on the R CRAN website. Please contact the first author for more information. WALRUS is licensed under the GPL v3 licence.

2 Script 1: loop over time steps

```
WALRUS_loop = function(pars)
  {
  # compute number of o_steps
  L            = length(output_date)
  # make empty vectors for output states and fluxes
  o            = data.frame(matrix(nrow=L, ncol=11, dimnames=list(NULL,
                 c("ETact","Q","fGS","fQS","dV","dVeq","dG","hQ","hS","W","dt_ok"))))

  # look up soil type parameters
  pars$b          = soil_char[["b"]]       [soil_char[["st"]]==pars$st]
  pars$psi_ae     = soil_char[["psi_ae"]]  [soil_char[["st"]]==pars$st]
  pars$theta_s    = soil_char[["theta_s"]][soil_char[["st"]]==pars$st]
  pars$aG         = 1-pars$aS

  # INITIAL CONDITIONS
  # Q[1] is necessary for stepsize-check (if dQ too large)
  o$Q     [1]  = func_Qobs(output_date[2]) / (output_date[2]-output_date[1]) *3600
  # hS from first Q measurement and Qh-relation
  o$hS    [1]  = uniroot(f=function(x){return(
                 func_Q_hS(x,pars,hSmin=func_hSmin(output_date[1]))-o$Q[1])},
                 lower=0, upper=pars$cD)$root
  # dG and hQ
  if(is.null(pars$dG0)==FALSE)                          # if dG0 provided
  {
    o$dG  [1]  = pars$dG0
    if((pars$cD-o$dG[1])<o$hS[1])                       # if groundwater below surface water level
    {
      o$hQ  [1] = o$Q[1]*pars$cQ                        # all Q from quickflow
    }else{                                              # if groundwater above surface water level
      o$hQ  [1] = max(0,(o$Q[1]-(pars$cD-o$dG[1]-o$hS[1])*(pars$cD-o$dG[1])/pars$cG) *pars$cQ)
    }

  }else{                                                # if dG0 not provided
    if(is.null(pars$Gfrac)==TRUE){pars$Gfrac=1}         # if Gfrac also not provided, make Gfrac 1
    # if fGS not possible with current hS and cG, make Gfrac smaller
    while(((pars$cD-o$hS[1])*pars$cD/pars$cG) < (pars$Gfrac*o$Q[1])) {pars$Gfrac = pars$Gfrac/2}
    # compute dG leading to the right fGS
    o$dG  [1]  = uniroot(f=function(x){return((pars$cD-x-o$hS[1])*(pars$cD-x)/pars$cG -
                         o$Q[1]*pars$Gfrac)},
                         lower=1, upper=(pars$cD-o$hS[1]))$root
    o$hQ  [1]  = o$Q[1] *(1-pars$Gfrac) *pars$cQ
  }
  # dependent variables
  o$dVeq  [1]  = func_dVeq_dG(o$dG[1], pars)
  o$dV    [1]  = o$dVeq[1]
```

```
  o$W      [1]  = func_W_dV(o$dV[1], pars)

  #
  o_step       = o[1,]
  i            = o[1,]

  # RUN FOR-LOOP OVER ALL TIME STEPS
  for (t in 2:L)
  {
    start_step     = output_date[t-1]                  # start at begin of output step
    end_step       = output_date[t]                    # first try whole output step
    sums_step      = rep(0,4)                          # to sum fluxes of substeps
    # as long as you're not at the end of the original time_step yet
    while(start_step < (output_date[t] - p_num$min_timestep))
    {
      o_step[1,]   = WALRUS_step(p=p, i=i, t1=start_step, t2=end_step)
      # if time step too large (and not very small)
      if((o_step$dt_ok == FALSE) & ((end_step-start_step) > p_num$min_timestep))
      {
        end_step   = (start_step + end_step)/2         # decrease step and run model
      }else{                                           # if one step completed (dt small enough)
        start_step = end_step                          # start of next step
        end_step   = output_date[t]                    # try to the end of the step
        sums_step  = sums_step + o_step[1:4]           # remember sums of fluxes
        i          = o_step                            # initial conditions for next step
      }
    }
    # final output of the step
    o[t,   ] = o_step                                  # keep states of last step
    o[t,1:4] = sums_step                               # replace fluxes with sums of steps
  }

  # remove dt_ok column
  o = o[,1:10]

  return(o)
  } # end function

# compile to decrease runtime
WALRUS_loop  = cmpfun(WALRUS_loop)
```

3 Script 2: one time step

```
WALRUS_step = function(pars, i, t1, t2)
{
  ### FORCING
  # convert input to current stepsize [mm/timestep]
  P_t     = func_P    (t2)  - func_P    (t1)
  ETpot_t = func_ETpot(t2)  - func_ETpot(t1)
  fXG_t   = func_fXG  (t2)  - func_fXG  (t1)
  fXS_t   = func_fXS  (t2)  - func_fXS  (t1)
  hSmin_t = (func_hSmin(t2) + func_hSmin(t1))/2
```

```
### STEPSIZE
dt      = (t2 - t1)/3600            # compute dt (in hours because parameters are in hours)
dt_ok   = TRUE                      # stepsize small enough as default

### FLUXES (based on states from the start of this timestep [mm/timestep])
PQ      = P_t * i$W                                             *pars$aG
PV      = P_t * (1-i$W)                                         *pars$aG
PS      = P_t                                                   *pars$aS
ETV     = ETpot_t * func_beta_dV(i$dV)                          *pars$aG
ETS     = ETpot_t                                               *pars$aS
if(i$hS < p_num$min_h*1000){ETS = 0}            # no ET from empty channel
ETact   = ETV + ETS
fQS     = i$hQ                                                  /pars$cQ *dt
fGS     = (pars$cD - i$dG - i$hS) * max((pars$cD - i$dG),i$hS) /pars$cG *dt
Q       = func_Q_hS(i$hS, pars=pars, hSmin=hSmin_t)                      *dt

### STATES (at the end of this time step / start of next time step) [mm])
# note that fluxes are already for the whole time step (multiplied with dt)
dV      = i$dV  - (fXG_t + PV - ETV - fGS          )            /pars$aG
hQ      = i$hQ  + (PQ - fQS                        )            /pars$aG
hS      = i$hS  + (fXS_t + PS - ETS + fGS + fQS - Q)            /pars$aS
dG      = i$dG  + (i$dV - i$dVeq) /pars$cV *dt



### SPECIAL CASE: LARGE-SCALE PONDING AND FLOODING
if((dV < 0) | (hS > pars$cD))
{
  if((dV < 0) & (hS <= pars$cD))                # if ponding and no flooding
  {
    hS  = hS + (-dV) *pars$aG /pars$aS          # all ponds to surface water
    dV = 0                                      # soil moisture deficit to surface
  }
  if((dV >= 0) & (hS > pars$cD))                # if no ponding and flooding
  {
    dV  = dV - (hS-pars$cD) *pars$aS /pars$aG   # all floods into soil
    hS  = pars$cD                               # channel bankfull
  }
  if((dV <= 0) & (hS >= pars$cD))               # if ponding and flooding
  {
  dV  = dV*pars$aG - (hS-pars$cD)*pars$aS       # compute total excess water
  hS  = pars$cD - dV
  }
  if(dV < 0){dG = dV}                           # if ponding, groundwater to pond level
}

### TEST IF STEP SIZE IS SMALL ENOUGH
if(hS < -p_num$min_h)                           # if hS below channel bottom
{
  dt_ok = FALSE
  hS = p_num$min_h*100
}else if(hQ < -p_num$min_h)                     # if hQ below bottom Q-res.
{
  dt_ok = FALSE
```

```
    hQ = p_num$min_h
  }else if(P_t > p_num$max_P_step)                  # if too much rainfall added
  {
    dt_ok = FALSE
  }else if(abs(i$Q-Q) > p_num$max_dQ_step)          # if change in Q too big
  {
    dt_ok = FALSE
  }else if(abs(i$hS-hS) > p_num$max_h_change)       # if change in hS too big
  {
    dt_ok = FALSE
  }else if(abs(i$dG-dG) > p_num$max_h_change)       # if change in dG too big
  {
    dt_ok = FALSE
  }

  ### OUTPUT
  # compute dependent variables (at end of time step)
  W      = func_W_dV(dV,pars)
  dVeq   = func_dVeq_dG(dG,pars)

  # bind output together in a vector
  return(c(ETact, Q, fGS, fQS, dV, dVeq, dG, hQ, hS, W, dt_ok))

} # end function

# compile to decrease runtime
WALRUS_step  = cmpfun(WALRUS_step)
```