

Supplement of Hellmers, 2020: Code and parameters to compute backwater effects

Dr.-Ing. Sandra Hellmers

Institute of River and Coastal Engineering, Hamburg University of Technology, Hamburg, 21073, Germany



DOI: 10.15480/882.3522, Link: <http://hdl.handle.net/11420/9508>

Content

1	Introduction	1
2	Source code of subroutines to compute backwater effects	2
2.1	FORTTRAN code: subroutine to compute backwater effects.....	2
2.2	FORTTRAN code: computation of the flood routing parameters	7
3	Input parameters to compute the flood routing and backwater effects in streams and adjusted lowland areas	11
4	Explanation of output parameters	14
5	References	15

1 Introduction

This document summarises the worked out source code and the input as well as output parameters for the hydrological rainfall-runoff model Kalypso-NA (version 4.0) to compute backwater effects in surface waters of tidal lowland catchments including control functions. This document is a supplement to the dissertation (Hellmers, 2020). The theoretical approaches, methodology, implementation and evaluation results are described in (Hellmers and Fröhle, 2021) and (Hellmers, 2020).

2 Source code of subroutines to compute backwater effects

2.1 FORTRAN code: subroutine to compute backwater effects

```
subroutine check_reverseFlow (ngaruh)
  !input
  integer, intent(in)      :: ngeruh
  integer                  :: m_stepCount
  real (dp)                :: delta_timestep

  !locals
  integer                  :: i time
  integer                  :: iStrand, jStrand, iStrand_upstream, iStrand_downstream
  type (Strand), pointer   :: m_strands
  integer                  :: id_upstreamNode
  real(dp)                 :: q_upstream_node
  integer                  :: reverseSystemNumber
  integer                  :: max_strand_reverseSystemNumber
  integer                  :: m_strand_reverseSystemNumber
  real(dp)                 :: H_actStrand !actual water level of that active strand
  real(dp)                 :: H_actStrand_upstream, H_actStrand_downstream !actual water level
of the upstream strand of that active strand
  integer                  :: strand_name, strand_name_act, strand_name_upstream, strand_name_downstream
!info
  logical                  :: check_reRun_reverseSystemLoop, reRun_reverseSystemLoop
  logical                  :: has_backwater_routing_results, write_backwater_routing_results
  integer                  :: counter_reverseSystemLoop
  real(dp)                 :: min_diffWaterLevel, H_bedlevel_diff
  real(dp)                 :: H_delta
  integer                  :: last_reRun_activeStrandOrdinal
  integer                  :: max_reverseSystemLoops
  integer                  :: node_upstream_ascii_id
  logical                  :: upstream_strand_found
  logical                  :: m_has_backwater_routing, has_backwater_routing, has_backwaterVolume
  logical                  :: has_ctrl_min_waterlevelsystem_setPerITime !initialisation per reverseSystem
  real(dp)                 :: target_min_waterlevel_system_setPerITime !initialisation per reverseSystem
  logical, allocatable     :: has_interactiveSystem(:) !array of the backwatersystems from 1 to max
giving the current interactive systems which require another flood routing computation
  integer                  :: max_interactiveSystemLoop, interactiveSystemLoop
  logical                  :: flag_reCalc_internal_sys_loop, activate_reCalc_internal_sys_loop
  character (len = 15)     :: m_strand_crossSection_form
  logical                  :: m_upstream_strand_flag
  max_strand_reverseSystemNumber = 0
  ! console info
  write (consoleUnit, '(//1X,a)') 'Backwater subroutine started'
  write(getUnit(f_resultOutput), '(//1X,a)') 'Backwater subroutine started'

  delta_timestep = dt ! defined in module simulation settings

  call getCurrentStepCount(m_stepCount)

  !initialisation of reverse system loops
  !find the maximal strandSystemNumber
  strand_loop_ini: do iStrand = 1, maxStrand
    call get_m_strand_reverseSystemNumber (iStrand, m_strand_reverseSystemNumber)
    !each HRB with internal downstream controlled system may be the start of a new system?!
    if (max_strand_reverseSystemNumber < m_strand_reverseSystemNumber) then
      max_strand_reverseSystemNumber = m_strand_reverseSystemNumber
    end if
  end do strand_loop_ini
  (... next page)
```

```

! for the first reverse system loop the waterlevel and volume is defined in the initial
strand loop
! loop 1 = define the actual water level in the current timestep for all the active strands
in the reverse loop
! initialization of the backwater routing status per timestep and system
m_has_backwater_routing = .false.
has_backwater_routing = .false.
strand_loop_1: do iStrand = maxStrand, 1, - 1
    call get_m_strand_reverseSystemNumber (iStrand, m_strand_reverseSystemNumber)
    call get_strand_name (iStrand, strand_name) ! info

    if (reverseSystemNumber == m_strand_reverseSystemNumber) then
        call get_strand_upstream_node(iStrand, node_upstream_ascii_id)
        id_upstreamNode = getOrdinal (node_upstream_ascii_id, naNode)
        !call set_hasbackwaterRouting_node(id_upstreamNode, reverseSystemNumber)
        q_upstream_node = qg(id_upstreamNode, i_time)

        call set_initial_stages_per_Strand_per_iTime (i_time, delta_timestep, iStrand,
            q_upstream_node, m_has_backwater_routing, has_backwater_routing,
            has_ctrl_min_waterlevelsystem_setPerITime,
            target_min_waterlevel_system_setPerITime)
        !if any of the strands in the current reverse system and current timestep has a
        strand routing upstream (backwater) = 1 = .true.,
        !the backwater routing is active

        if (m_has_backwater_routing == .true.) then
            has_backwater_routing = .true.
        end if
    end if
end do strand_loop_1
!*****
! BACKWATER LOOP if active:
!*****
if (has_backwater_routing == .true.) then ! has_backwater_routing = .true. = has afflux in the
control segment
    strand_loop_reverse: do iStrand = maxStrand, 1, -1

        call get_m_strand_reverseSystemNumber (iStrand, m_strand_reverseSystemNumber)
        upstream_strand_found = .false.
        call get_m_strand_crossSection_form (iStrand, m_strand_crossSection_form) /=
        'circular'
        !for each reverse strand system the reverse flow is computed individually
        !the reverse strand system is from downstream (1) to the upstream (tributaries 2 to x):
        !each new reverse strand system has a downstream control " feature"
        if (reverseSystemNumber == m_strand_reverseSystemNumber .and.
            m_strand_crossSection_form /= 'circular') then
            !circular (closed) profiles are not considered to have free storage for retaining
            backwater
            !for circular profiles instantenious routing of backwater is considered.
            call get_strand_name (iStrand, strand_name_act) ! info
            call get_act_H_Strand (iStrand, H_actStrand)
            call check_backwaterVolume(iStrand, has_backwaterVolume)

            if (has_backwaterVolume == .true.) then

                !search next active upstream strand of that reverse strand system:
                inner_strand_loop_2: do jStrand = iStrand-1, 1, - 1
                    m_strand_reverseSystemNumber = 0
                    call get_m_strand_reverseSystemNumber (jStrand, m_strand_reverseSystemNumber)
                    call get_m_strand_crossSection_form (jStrand, m_strand_crossSection_form)
                    if (reverseSystemNumber == m_strand_reverseSystemNumber .and.
                        m_strand_crossSection_form /= 'circular') then
                        upstream_strand_found = .true.
                        iStrand_upstream = jStrand
                        H_delta = 0.0_dp
                        call get_min_backwater_h_m (iStrand, min_diffWaterLevel) !minimum water
                        level to activate the correction and reverseloop calculation given in m NN
                        E.g. 0,01 = 1cm
                        call get_strand_name (iStrand_upstream, strand_name_upstream) ! info
                        call get_act_H_Strand (iStrand_upstream, H_actStrand_upstream)
                        call calc_H_delta(H_actStrand, H_actStrand_upstream, H_delta)

```

(... next page)

```

if (H_actStrand > H_actStrand_upstream .and. H_delta > min_diffWaterLevel) then
!*****
!Approach 1: computing the water-level difference on the basis of given valuen in
reference to mean sea level (mNN = m.s.l.)
!*****
check_reRun_reverseSystemLoop = .true. ! if any strand requires a correction in this
loop, the reverseSystemLoop is started again.
call calc_routed_waterlevel_volume_upstream (iStrand, iStrand_upstream, i_time,
delta_timestep, counter_reverseSystemLoop) !, w_act)
! if any of the strands waterlevels or strand volumes was corrected, the system
loop is running again to check another correction in the cascade.
last_reRun_activeStrandOrdinal = iStrand
exit inner_strand_loop_2
end if
end if
! if the most upstream strand = 1 is reached at this position, the strand has no upstream strand,
! the backwater does not flow further upstream in strands, but "may" flow in a user defined
backwater area. (or is stored in the strand, if not)
end do inner_strand_loop_2 !loop to find upstream strand
!Here the upstream strand with possible flood plane is computed only if the current strand has
backwater volume. ***
!the upstream flood prone area is computed already in the strands loop as upstream strand
if ( upstream_strand_found == .false.) then
!m_upstream_strand_flag = .true.
!call set_upstream_strand_flag (iStrand, m_upstream_strand_flag)
call calc_upstream_backwater_area ( iStrand)
last_reRun_activeStrandOrdinal = iStrand
end if
end if ! query about having any backwater volume in the current strand
end if ! if the actual reverse system fits
end do strand_loop_reverse ! loop over all strands within one reverseLoop Number

else if (has_backwater_routing == .false.) then
!*****
! CHECK FOR BACKWATER VOLUME ROUTING DOWNSTREAM
!*****
! After the first upstream directed backwater loop another loop is required from upstream to downstream
according to find
!strands where the correction of waterlevels need to be averaged between the corrections or the backwater
is drained downstream again.
strand_loop_check_corrWaterLevel: do iStrand = 1, maxStrand
call get_m_strand_reverseSystemNumber (iStrand, m_strand_reverseSystemNumber)
call get_m_strand_crossSection_form (iStrand, m_strand_crossSection_form)
upstream_strand_found = .false.
if (reverseSystemNumber == m_strand_reverseSystemNumber .and. m_strand_crossSection_form /=
'circular') then
call get_act_H_Strand (iStrand, H_actStrand)
call get_strand_name (iStrand, strand_name_act) ! info
call get_min_backwater_h_m (iStrand, min_diffWaterLevel) !minimum water level to activate the
correction and reverseloop calculation given in m NN 0,01 = 1cm
has_backwaterVolume = .false.
call check_backwaterVolume(iStrand, has_backwaterVolume)
! !for each reverse strand system the reverse flow is computed individually
if (has_backwaterVolume ==.true.) then ! backwater volume is routed downstream if downstream
strands waterlevel is lower than in actual strand.
inner_strand_loop_4: do jStrand = iStrand+1, maxStrand
!
m_strand_reverseSystemNumber = 0
call get_m_strand_crossSection_form (jStrand, m_strand_crossSection_form)
call get_m_strand_reverseSystemNumber (jStrand, m_strand_reverseSystemNumber)
if (reverseSystemNumber == m_strand_reverseSystemNumber .and.
m_strand_crossSection_form /= 'circular') then
upstream_strand_found = .true.
!downstream_strand_found = .true. info not required currently
iStrand_downstream = jStrand
H_delta = 0.0_dp
call get_act_H_Strand (iStrand_downstream, H_actStrand_downstream)
call get_strand_name (iStrand_downstream, strand_name_downstream) ! info
call calc_H_delta(H_actStrand, H_actStrand_downstream, H_delta)
call get_bedlevel_difference(iStrand, iStrand_downstream, H_bedlevel_diff)

(... next page)

```

```

        if ((H_actStrand-min_diffWaterLevel) > H_actStrand_downstream .and. H_delta >
            (min_diffWaterLevel)) then ! the upstream waterlevel is assumed to
            remain a waterlevel corresponding to bedlevel difference (instantaneous
            state)
            !*****
            !Approach: computing the water-level difference on the basis of given
            values in reference to mean sea level (mNN = m.s.l.)
            !*****
            check_reRun_reverseSystemLoop = .true. ! if any strand requires a
            correction in this loop, the reverseSystemLoop is started again.
            call calc_routed_waterlevel_volume_downstream (iStrand,
                iStrand_downstream) !, w_act)
            ! if any of the strands waterlevels or strand volumes was corrected, the
            system loop is running again to check another correction in the cascade.
            last_reRun_activeStrandOrdinal = iStrand
        end if
        exit inner_strand_loop_4
    end if
end do inner_strand_loop_4
!Here the upstream strand with possible flood plane is computed only if the current strand
has backwater volume. ***
if (upstream_strand_found == .false.) then
    call calc_upstream_backwater_area (iStrand)
    last_reRun_activeStrandOrdinal = iStrand
end if
end if
end if
end do strand_loop_check_corrWaterLevel
if ! backwater routing is finished and water is routed downstream.

!after strand loop, check for another reRun:
if (check_reRun_reverseSystemLoop == .true. .and. counter_reverseSystemLoop < max_reverseSystemLoops)
    then
        reRun_reverseSystemLoop = .true. !the final flag for reRun the loop is remained to be set
        to.true.
        check_reRun_reverseSystemLoop = .false. !the temporary flag within the check-loops is set back to
        .false. for the next run

    else ! the check flag remained false over all strands within the reverse system loop. The loop is
        finished or the max number of loops is reached.
        !if the reverse system loop reached a number of max_strand_reverseSystemNumber, the modeller is
        informed / warned.
        if (counter_reverseSystemLoop >= max_reverseSystemLoops) then
            call warn_max_strand_reverseSystemNumberReached(last_reRun_activeStrandOrdinal,
                max_reverseSystemLoops, i_time)
        end if
        reRun_reverseSystemLoop = .false. !the final flag for reRun the loop is set to .false. (the
        While loop is finished)
    end if
end do ! check_reRun_reverseSystemLoop = end

strand_check_corrDischarge: do iStrand = 1, maxStrand
    call get_m_strand_reverseSystemNumber (iStrand, m_strand_reverseSystemNumber)
    if (reverseSystemNumber == m_strand_reverseSystemNumber) then
        has_backwater_routing_results = .false.
        call check_corr_discharge (iStrand, i_time, delta_timestep, has_backwater_routing_results)
        !if any of the strands has_backwater_routing_results, the flag for output results is
        activated
        if (has_backwater_routing_results == .true.) then
            write_backwater_routing_results = .true.
        end if
        call set_strandResults_for_iTime(i_time, iStrand)
    end if
end do strand_check_corrDischarge
end do time_loop

```

(... next page)

```

call has_reCalc_internal_sys_loop(reverseSystemNumber, flag_reCalc_internal_sys_loop)
if (flag_reCalc_internal_sys_loop == .true.) then
    activate_reCalc_internal_sys_loop = .true.
else
    activate_reCalc_internal_sys_loop = .false.
    !restart the same
end if
end do ! do loop of the internal activate_reCalc_internal_sys_loop till it is .false.
end do ReverseSystem_loop !loop over all reverseSystems, after that the next reverseSystemNumber + 1 is
                           computed
if (any(has_interactiveSystem)) then
    interactiveSystemLoop = interactiveSystemLoop + 1 !go to the next loop
    ! the loop stops when the counter reaches (max_interactiveSystemLoop)
else
    interactiveSystemLoop = max_interactiveSystemLoop ! stop function
end if

end do ! while loop for max_interactiveSystemLoop

!after the backwater system loops, the final water level and volumes in the strands are computed.
!The final step is the computation of the actual drainage (including backwater volume) and open water
    evaporation losses from the open water surfaces.
!the waterlevels and volumes are adjusted as final step.

call check_drainage_backwaterVolume_Overlays()
call calc_openWater_evaporation() !strands and flood prone areas

! giving here an information for the user, that the method of backwater routine is activated.
if (write_backwater_routing_results == .true.) then
    call writeLogString(f_gmlErrorLog, 5, 'Die Methodik zur Berechnung rueckgestauter Abfluesse ist
        aktiviert. Ergebnisse unter C Plugin_KalypsoNA_4.0', 'The method of backwater routing is
        activated. The results are written in C Plugin_KalypsoNA_4.0', '', '', '', '', '',
        'mod_strands' )
    !wrting the output files:
    !using here the strand_ini and strand_routed results:
    call output_backwater_routing_results_strands ()
    write(getUnit(f_resultOutput), '/a')' written Strand results timeseries: waterlevel m NHN, Volume
    hm3, Discharge m3/s (see: C:\Plugin_KalypsoNA_4.0\results\... '
    call output_backwater_routing_results_nodes ()
    write(getUnit(f_resultOutput), '/a')' written Node results timeseries: backwater effected
    Discharge m3/s (see: C:\Plugin_KalypsoNA_4.0\results\... '
    call output_backwater_routing_results_backwaterAreas ()
    write(getUnit(f_resultOutput), '/a')' written Backwater area results timeseries: waterlevel m NHN
    and Volume hm3 (see: C:\Plugin_KalypsoNA_4.0\results\... '
end if

end subroutine !check_reverseFlow

```

2.2 FORTRAN code: computation of the flood routing parameters

```

subroutine calc_kami_strand (iStrand_ordinal)
!input
integer, intent (in)          :: iStrand_ordinal

!locals
type (Strand), pointer        :: m_act_strand
type (strand_KM_calc), pointer :: m_strand_KM_calc
type (strand_km_input), pointer :: m_strand_km_input
type (strand_km_results), pointer :: m_strand_km_results
integer                       :: max_steps, step_i, size_wvq
real(dp), allocatable         :: volume_V (:), discharge_Q (:), discharge_Q_m (:),
                                discharge_Q_delta (:), length_Lc_i (:)! results
real(dp)                      :: length, strand_gradient ! input data
real(dp)                      :: slope_result_VQ
real(dp)                      :: length_Lc_circularProfile,
                                Q_full_circularProfile

!preparation of KaMi-Routing computation
m_act_strand => strands(iStrand_ordinal)
m_strand_KM_calc => m_act_strand%m_strand_KM_calc
m_strand_km_input => m_strand_KM_calc%m_strand_km_input
m_strand_km_results => m_strand_KM_calc%m_strand_km_results

max_steps = m_strand_km_input%max_steps !

!*****

strand_gradient = m_act_strand%strand_gradient
length = m_act_strand%strand_length

if (m_strand_km_input%crossSection_form == 'circular') then
!for pipe flow only the empty and full stage are considered
size_wvq = 2
else
size_wvq = max_steps+1
end if

allocate (volume_V (size_wvq))
allocate (discharge_Q (size_wvq))
allocate (discharge_Q_m (size_wvq))
allocate (discharge_Q_delta (size_wvq))
allocate (length_Lc_i (size_wvq))
allocate (m_strand_km_results%discharge_Q_m3s(size_wvq))
allocate (m_strand_km_results%waterlevel_h_mNN(size_wvq))
allocate (m_strand_km_results%crossSection_A_m2(size_wvq))
! the computation of the Kalinin-Miljukov Parameters is defined in a subroutine which can be used
by other routing elements
call calc_kami_parameters (m_act_strand%gml_name, size_wvq,
    m_strand_km_input%bankfull_height_m, m_strand_km_input%bed_width_m ,
    m_strand_km_input%bank_gradient, &
    m_strand_km_input%roughness_kst_ms, length, strand_gradient, &
    m_strand_km_input%hydr_diameter_m, m_strand_km_input%roughness_ks_m,
    m_strand_km_input%crossSection_form, m_strand_km_input%calc_method, &
    crossSection_A_m2)
                                (... next page)

```

```

!results:
volume_V, discharge_Q, discharge_Q_m, &
discharge_Q_delta, length_Lc_i, m_strand_km_results%waterlevel_h_mNN,
m_strand_km_results%discharge_Q_m3s, m_strand_km_results%crossSection_A_m2)

m_strand_km_results%km_l_length = sum(length_Lc_i)/(size_wvq-1)
m_strand_km_results%km_n_storages = nint(length / m_strand_km_results%km_l_length)

!implementation of a slope function for angular profiles.
(Y_values; X_values)
if (m_strand_km_input%crossSection_form == 'circular') then
    !computation according to (Euler, 1983).
    m_strand_km_results%km_k_retConstant_h = 0.64_dp *
    m_strand_km_results%km_l_length *
    (m_strand_km_input%hydr_diameter_m**2)/m_strand_km_results%discharge_Q_m3s(2) /3600.0_dp
    ![h]
else
    call calc_slope_function (size_wvq, volume_V, discharge_Q, slope_result_VQ) !input
    is here the discharge and volume to compute the slope function

    m_strand_km_results%km_k_retConstant_h = (slope_result_VQ /
    m_strand_km_results%km_n_storages)/60/60 ! transfer from [s] to [h]
end if
!Setting the results of the Kalinin-Miljukov method to the strands data for the
backwater routing:
call allocate_strands_wqa (size_wvq,iStrand_ordinal)

do step_i = 1, size_wvq
    m_act_strand%strand_waterLevel (step_i) = m_act_strand%strand_bedLevel_mNN +
    (m_strand_km_results%waterlevel_h_mNN(step_i)+m_act_strand%adding_W_wq_relations)
    m_act_strand%strand_discharge (step_i) =
    m_strand_km_results%discharge_Q_m3s(step_i)* m_act_strand%factor_Q_wq_relations
    m_act_strand%strand_crossSection (step_i) = (m_strand_km_results%crossSection_A_m2
    (step_i)+m_act_strand%adding_A_wq_relations)
    m_act_strand%strand_storageVolume (step_i) =
    (m_strand_km_results%crossSection_A_m2 (step_i)+m_act_strand%adding_A_wq_relations)
    * length
end do
m_act_strand%max_wqa_size = size_wvq

end subroutine calc_kami_strand

```



```

subroutine calc_kami_parameters (m_gml_name, size_wvq, bankfull_height_m, width,
    bank_gradient, kst, length, strand_gradient, &
    m_hydr_diameter_m, m_roughness_ks_m, m_crossSection_form, m_calc_method, &
!results:
volume_V, discharge_Q, discharge_Q_m, discharge_Q_delta, length_Lc_i,
waterLevel_mNN, discharge_Q_m3s, crossSection_A_m2)
!input
character (len = maxLineLength), intent(in) :: m_gml_name
integer,intent (in) :: size_wvq
real(dp),intent (in) :: bankfull_height_m, width,
    bank_gradient, kst, length, strand_gradient
real(dp), intent(in) :: m_hydr_diameter_m, m_roughness_ks_m
character (len = 15), intent(in) :: m_crossSection_form, m_calc_method

real(dp),intent (out) :: volume_V (size_wvq), discharge_Q
    (size_wvq), discharge_Q_m (size_wvq), discharge_Q_delta (size_wvq),
    length_Lc_i (size_wvq), waterLevel_mNN (size_wvq),
    discharge_Q_m3s(size_wvq), crossSection_A_m2(size_wvq)
! results which are allocated before

!locals:
real(dp) :: h_delta
integer :: step_i
real(dp) :: h_act_i
real(dp) :: velocity_act_i, velocity_next_i,
    A_next_i, A_act_i
real(dp) :: P_act_i, P_next_i, R_hyd_next_i,
    R_hyd_act_i

!darcy parameter:
real(dp) :: lambda_act, lambda_next,
    f_shape_coefficient_act, f_shape_coefficient_next
real(dp) :: length_Lc_circularProfile,
    Q_full_circularProfile

!result file
integer :: kamiStreamResultsID
character(len=81) :: kamiStreamResultsFile

h_delta = bankfull_height_m / (size_wvq-1) ! Computation of the step size for wvq-function
h_act_i = - h_delta
if (m_crossSection_form == 'circular') then
!the equation is according to (Euler, 1983)
!the characteristical length is not changed for full or empty pipe
length_Lc_circularProfile = 0.4 * m_hydr_diameter_m /(strand_gradient)
!calculation of the maximal capacity of the drainage pipe according to darcy
    with the pipe roughness
    A_act_i = pi / 4 * m_hydr_diameter_m**2
    P_act_i = pi * m_hydr_diameter_m
    R_hyd_act_i = A_act_i / P_act_i
    lambda_act=(1.0_dp/(-2.0_dp*LOG10(m_roughness_ks_m/ (14.84_dp *
        R_hyd_act_i))))**2.0_dp
!flow velocity calculation with the Darcy Weissbach approach:
    velocity_act_i = SQRT((8.0_dp * gravity * R_hyd_act_i
        *strand_gradient)/lambda_act) ![m/s]
    Q_full_circularProfile = velocity_act_i * A_act_i
! setting the parameters: wvq = 1 => empty pipe; wvq = 2 => full pipe
    length_Lc_i(1) = 0.0_dp
    length_Lc_i(2) = length_Lc_circularProfile
    waterLevel_mNN(1) = 0.0_dp
    waterLevel_mNN(2) = m_hydr_diameter_m
    discharge_Q_m3s(1) = 0.0_dp
    discharge_Q_m3s(2) = Q_full_circularProfile
    crossSection_A_m2(1) = 0.0_dp
    crossSection_A_m2(2) = A_act_i

```

(... next page)

```

else !for angular profiles the w-v-q-steps are computed
do step_i = 1, size_wvq
    h_act_i = (step_i - 1) * h_delta ! the computation starts with a
    waterlevel of 0 at step = 1
    waterLevel_mNN (step_i) = h_act_i
    ! computation of wetted perimeter and cross-section according to
    m_crossSection_form
    A_act_i = (width + bank_gradient * h_act_i) * h_act_i
    P_act_i = width + 2.0_dp * h_act_i * sqrt(1.0_dp + bank_gradient**2)
    A_next_i = (width + bank_gradient * (h_act_i + h_delta)) * (h_act_i +
    h_delta)
    P_next_i = width + 2.0_dp * (h_act_i + h_delta) * sqrt(1.0_dp +
    bank_gradient**2)
    R_hyd_act_i = A_act_i / P_act_i
    R_hyd_next_i = A_next_i / P_next_i
    crossSection_A_m2(step_i) = A_act_i
    ! computation of velocity according to method: darcy or manning
if (m_calc_method == 'darcy') then ! Darcy Weissbach method
    call calc_shape_coefficient(h_act_i, width, R_hyd_act_i, m_crossSection_form,
    f_shape_coefficient_act) !According to (BWK, 2009) page 24ff
    ! lambda calculation according to Colebrook White (turbulent flow =
    without the Reynolds term) for the first wvq-step, the lambda and the velocity is zero:
    if (step_i == 1) then
        lambda_act = 0.0_dp
        velocity_act_i = 0.0_dp
    else
        lambda_act = (1.0_dp / (-2.0_dp * LOG10(m_roughness_ks_m /
        (f_shape_coefficient_act * 14.84_dp * R_hyd_act_i))))**2.0_dp
        !flow velocity calculation with the Darcy Weissbach approach:
        velocity_act_i = SQRT((8.0_dp * gravity * R_hyd_act_i
        * strand_gradient) / lambda_act) ![m/s]
    end if
    call calc_shape_coefficient((h_act_i + h_delta), width, R_hyd_act_i,
    m_crossSection_form, f_shape_coefficient_next)
    lambda_next = (1.0_dp / (-2.0_dp * LOG10(m_roughness_ks_m /
    (f_shape_coefficient_next * 14.84_dp * R_hyd_next_i))))**2.0_dp
    velocity_next_i = SQRT((8.0_dp * gravity * R_hyd_next_i
    * strand_gradient) / lambda_next) ![m/s]
else
    !calculation method = "manning-strickler":
    velocity_act_i = kst * ((R_hyd_act_i)**(2.0_dp/3.0_dp)) *
    (strand_gradient**0.5_dp)
    velocity_next_i = kst * ((R_hyd_next_i)**(2.0_dp/3.0_dp)) *
    (strand_gradient**0.5_dp)
end if
    ! computation of discharge and volume
    discharge_Q (step_i) = velocity_act_i * A_act_i
    discharge_Q_m3s(step_i) = discharge_Q (step_i)
    volume_V (step_i) = length * A_act_i
    ! computation of averaged discharges for W-V-Q-relations (for angular
    profiles!)

```

(... next page)

```

if (step_i < size_wvq) then
  discharge_Q (step_i + 1) = velocity_next_i * A_next_i
  discharge_Q_m (step_i) = (discharge_Q (step_i) + discharge_Q(step_i +
    1))/2.0_dp
else
  !last discharge at maximum water-level
  discharge_Q_m (step_i) = 0.0_dp ! bankfull discharge is reached in the step
    before.
end if

if (step_i == 1) then
  discharge_Q_delta (step_i) = discharge_Q_m (step_i) ! no averaging of the
    first discharge at waterlevel = 0
else if (step_i == size_wvq) then
  discharge_Q_delta (step_i) = 0.0_dp ! bankfull discharge is reached in the step before.
    no additional averaged discharge is taken in the computation
else
  discharge_Q_delta (step_i) = discharge_Q_m (step_i) - discharge_Q_m (step_i - 1)
end if

if (step_i == size_wvq) then
  length_Lc_i (step_i) = 0.0_dp ! bankfull discharge is reached in the step before.
else
  length_Lc_i (step_i) = h_delta * discharge_Q_m(step_i) /(strand_gradient *
    discharge_Q_delta(step_i))
end if

end do !loop over wvq-steps
end if ! query about circular or angular profile
end subroutine calc_kami_parameters

```

3 Input parameters to compute the flood routing and backwater effects in streams and adjusted lowland areas

ASCII file	Description of input parameters
strands.ctrl	Configuration of criteria for control structures of linear data structures (streams).
strands.kami	Input parameters for the flood routing computation in stream segments on meso scale.
strands.topo	Input parameters to describe stream profiles.
nodes.xyz	Geographical location defined by the coordinates x (longitude), y (latitude) and z (height) for junction nodes.
"node-name".w	Time series of water levels (e.g. of gauging stations) for defined junction nodes.
"strand-name".wvq	(Optional) input parameters of the WVQ-relations for example from existing hydrodynamic model results when using the KM5-method.

strands.ctrl:

Input parameters and explanation:

m_ctrl_flag	Kind of driver for the control function
m_ctrl_element	Name of element to which the threshold value is dedicated
m_ctrl_value	Threshold value to start the control function
m_ctrl_duration	Minimum duration of the control function after start
m_ctrl_value_end	Threshold value to end the control function
m_ctrl_duration_end	Additional duration of the control function to end
m_ctrl_wvqFunction	File of control function parameters
m_flag_external_timeseries	Query if the time series is computed during the run-time (=false) or given as external (pre-set) time series (=true).
m_advance_ctrl_duration	Forecast function: Start of control function before threshold value is reached
m_advance_ctrl_target_value	Forecast function: control function activation till a target volume in the control element is reached
m_mother_backwaterSystem	(optional) Order number of the interactive backwater system
m_min_activation_duration	Minimum duration of the control function

Example of a control structure (distinct name: 6031ctrl) with five control functions:

```
6031ctrl 5
waterlevel 3160 -9999.0 120 -9999.0 120 6031_0.wvq false 0 0 0 0
waterlevel 3160 -0.95 30 -0.98 30 6031_01.wvq false 0 0 0 0
waterlevel 3160 -0.85 30 -0.87 30 6031_1.wvq false 0 0 0 0
waterlevel 3160 -0.80 30 -0.83 30 6031_2.wvq false 0 0 0 0
waterlevel 3160 -0.75 30 -0.77 30 6031_3.wvq false 0 0 0 0
ENDE
```

strands.kami:

Input parameters and explanation:

this_strand_name_gml	Distinct name of river stream section
m_bed_width_m	(optional angular profile) Profile bed width (m)
m_bank_gradient	(optional angular profile) Bank gradient (m),
m_roughness_kst_ms	(optional angular profile) Manning-Strickler roughness kst (m ^{1/3} /s),
m_bankfull_height_m	(optional angular profile) Bankfull height (m),
m_hydr_diameter_m	(optional circular profile) Hydraulic diameter (m)
m_roughness_ks_m	(optional circular profile) Equivalent sand roughness ks (m),
m_max_steps	Number of supporting points to compute the WVQ-functions
m_crossSection_form	Cross section form (circular or rectangular),
m_calc_method	Choice of the computational approach (Manning-Strickler or Darcy Weisbach).

Example of three strands with rectangular profile; computation method: manning-strickler:

```
3160km1 7.0 1.0 30 3.3 0 0 25 rectangular manning
3161km1 13.0 1.5 30 4.3 0 0 25 rectangular manning
3170km1 25.0 1.0 30 2.3 0 0 25 rectangular manning
```

strands.topo:

Input parameters and explanation:

this_strand_name_gml	Distinct name of river stream section
m_strand_length	Stream length (m) (optional if geographical data not given or an exact flow path length is defined)
m_strand_gradient	Gradient (-) (optional if geographical data not given)
m_strand_bedLevel_mNN	Lowest bed level (m a.s.l.),
m_min_waterLevel_mNN	Minimum water level (m a.s.l.)
m_strand_wq_flag	Flag for: given w-q data = 1, w-q curve = 2, computed kalinin curve = 3 or ctrl storage strand = 4
m_factor_Q_WQ_relations, m_adding_W_WQ_relations, m_adding_A_WQ_relations:	(optional adjustment factors) discharge (default = 1.0), water level variance (default = 0.0 m), cross-section adjustment (default = 0.0 m ²).
m_strand_reverseSystemNumber	Order number of the backwater system
m_min_backwater_h_m	Minimum tolerable backwater level difference (m)
m_strand_calc_kalinin_flag	Flag for KM computation method = 1; KM parameters pre-set = 0; Flag = 4 if it is a ctrl-element.
m_nrOfCatchments	(optional) number of adjusted spatial data structures
strand_overflow_overlay_id(iCatchm)	(optional) linked spatial structure
m_overflow_height_mNHN(iCatchm)	(optional) overflow height in m a.s.l.

Example of five strands and one with an adjusted lowland area:

```
3151km1 1397.0 0.001 -1.60 -0.87 3 0.0 0.73 0 6 0.00001 1 1
24 -0.4
3152km1 493.0 0.001 -1.92 -0.88 3 0.0 1.04 0 6 0.00001 1 0
3160km1 1026.0 0.005 -2.30 -0.89 3 0.0 1.41 0 6 0.00001 1 0
3161km1 4685.0 0.005 -2.35 -0.91 3 0.0 1.44 0 6 0.00001 1 0
3170km1 247.0 0.002 -2.35 -0.92 3 0 1.43 0 6 0.00001 1 1
```

nodes.xyz

Input parameters and explanation:

m_node_id	Distinct name of junction node in the hydrological network
m_descr_node	(optional) description
m_x_coordinate	X-coordinate
m_y_coordinate	Y-coordinate
m_z_coordinate	Z-coordinate
m_factor_flowpath	(optional) Prolongation of computed flow path length to the linked element in the hydrological network

Example:

```
3151 Moorfleet_3 572087.7859 5929285.224 -0.99 1.0
3161 Moorfleet_2 572651.6114 5928117.698 -0.99 1.0
3140 Moorfleet 4 572414.6276 5929117.823 -0.99 1.0
```

"node-name".w

Input parameters and explanation:

"node-name".	Distinct name of junction node in the hydrological network
.w	Time series of "w" = water level; alternatives: p = precipitation and q = discharge
201101010000	Start date: yyyyymmddmmss
Grap	Format

Example:

```
201101010000
grap
01.01.2011 00:00:00 2.000
01.01.2011 00:05:00 2.030
01.01.2011 00:10:00 2.050
01.01.2011 00:15:00 2.080
01.01.2011 00:20:00 2.100
01.01.2011 00:25:00 2.120
01.01.2011 00:30:00 2.140
...
...
```

"strand-name".wvq

Input parameters and explanation:

"strand-name". Distinct name of stream section with control function in the hydrological network
.wvq pre-set water level-volume-discharge function

Example:

```
SPEICHER 1155 6031_ov 6031 6031
Fakt_SeeV 1.00
text;text
6031ctrl 0.000135 0.009000 0.000000 201
        -1.05 0.000045 0.30 0.000 0.000
        -1.00 0.000090 0.30 0.000 0.000
...
...
```

Explanation:

The input parameters, to model the flood routing with the KM-method, comprise the profile data of stream segments on different scales. For the geometrical profiles using the KM1-method, the parameters are given in the file "strands.kami". In this file the parameter values of district and meso scale streams are written. The file comprises the following data: profile bed width (m), bank gradient (m), Manning-Strickler roughness k_{st} ($m^{1/3}/s$), bankfull height (m), hydraulic diameter (m), equivalent sand roughness k_s (m), number of interpolation steps (-), cross section form (circular or rectangular), minimal longitudinal gradient (m/m) and a choice of the computational approach (Manning-Strickler or Darcy Weisbach). According the chosen approach (Manning-Strickler or Darcy-Weisbach) and the cross section (circular or rectangular) the mandatory input parameters are specified. The topographical data of stream segments and the input parameters for the backwater effect computation are given in the file "strands.topo". The stream length (m) and gradient (-) are computed with the geographical data of the upstream and downstream junction nodes or can be provided as user defined input values. A flow path prolongation factor is given optionally per data structure in the hydrological network. The stream segment input parameters comprise: the lowest bed level (m a.s.l.), the minimal water level (m a.s.l.) and the number of supporting points to compute the WVQ-functions.

For calibration purposes an adjustment of the WVQ-functions per stream segment profile is provided (optionally) in the input files in the form of a percentage factor for the discharge (default = 1.0), a water level variance value (default = 0.0 m) and a cross-section adjustment using a variance value (default = 0.0 m^2). The stream segments are part of different backwater systems which are defined with an ordinal number (1 to n) from downstream to upstream. The tolerable backwater level difference is defined in (m) per stream segment, whereas larger values are suggested for streams with wider profiles than for narrow ones. Another input parameter in the file (here: "strands.topo") gives the number of linked spatial data structures (for example, riparian or retention areas). The indexes of the linked spatial data structures are listed in the input file with an overflow height in m a.s.l. When the water level of a stream segment reaches this overflow height, water is flowing into the free storage volume of the linked spatial structure.

4 Explanation of output parameters

The results of linear data structures are given per downstream junction node. One cross section is defined per stream segment in the main flow direction from upstream to downstream point of view. The output parameters of the calculation code Kalypso-NA are written in ASCII files. These files are post-processed with the module Kalypso Hydrology to analyse the results (namely hydrographs of discharges and storage volumes) of the meso scale data structures. Additional output parameters of the implemented methods, are written in ASCII files of the defined extension folder (see Fig. 2). The output parameters are

processed for the purpose of model evaluation and application. The output parameters of flood routing computations are the formal parameters and the WVQ-relations before and after the calculation of backwater effects with the KM1- and KM5-method. A comparison of the inflow and outflow hydrograph volumes per stream segment is provided for evaluation purpose. Additionally, the time series with a temporal resolution of Δt for the following output parameters per stream segment and control structure are given:

- Storage volume before and after backwater effect computation (m^3).
- Water level before and after backwater effect computation (m a.s.l.).
- Control system settings per time step (-).
- Discharge per junction node (m^3/s).
- Evaporation rates from open water surfaces ($\text{mm}/\Delta t$).
- Exceedance flow of reservoir stream segments (m^3/s).

For evaluation purposes in the form of mass-conservation and for checking the calculated flood routing parameters, the following output is given per stream segment:

- Total inflow hydrograph volume (m^3).
- Total outflow hydrograph volume (m^3).
- Change in water storage per simulation run (m^3).
- Computed (formal) parameters of the KM1-method:
 - Retention coefficient K_{km} (s).
 - Characteristic length L_c (m).
 - Number of characteristic lengths n (-).
- Computed (formal) parameters of the hydraulic capacity per stream segment (in the form of WVQ-relations):
 - Water level (m).
 - Volume (m^3).
 - Discharge (m^3/s).
 - Wetted cross section (m^2).
 - Hydraulic radius (m).
 - Flow velocity (m/s).

5 References

- Euler, G.: Ein hydrologisches Näherungsverfahren für die Berechnung des Wellenablaufs in Kreisrohren, Wasser und Boden, 1983, 1983.
- Hellmers, S.: Integrating local scale drainage measures in meso scale hydrological modelling of backwater affected catchments, Dissertation, Institute of River and Coastal Engineering, Technical University Hamburg, Hamburg, Germany, 2020. <https://doi.org/10.15480/882.2627>
- Hellmers, S. and Fröhle, P.: Computation of backwater effects in surface waters of tidal lowland catchments including control structures – An efficient and re-usable method implemented in the hydrological open source model Kalypso-NA (4.0), 2021. <https://doi.org/10.5194/gmd-2021-140>, in review