A1. The subroutine that was used in the simulations

```
  subroutine LARS_TEST_LAB()

    integer :: ind, count_rand_walks, j, i
    integer :: food_item_selected

    real :: step_rwalk, cost_step

    integer, dimension(proto_parents%population_size) ::
random_sample_individuals

    !> Lars' variables are prefixed with lars_
    !> OUTPUT: Declaring record which has the data values appended for each
individual
    character(len=2000) :: lars_file_record_append_data_gos_label
    character(len=2000) :: lars_file_record_append_data_gos_arousal
    character(len=2000) :: lars_file_record_append_data_gos_repeated

    !! OUTPUT: Declaring file names as character string variables
    character(len=:), allocatable :: lars_output_filename_data_gos_label
    character(len=:), allocatable :: lars_output_filename_data_gos_arousal
    character(len=:), allocatable ::
lars_output_filename_data_gos_repeated

    !> OUTPUT: Declaring file units as integer numbers. We need file units
for
    !! behind the scene work, even though they are not directly used here.
    !! All the CSV routines can refer to the file by its name.
    integer lars_output_fileunit_data_gos_label
    integer lars_output_fileunit_data_gos_arousal
    integer lars_output_fileunit_data_gos_repeated

    !> This variable keeps a short description component for the csv output
    !! file names:
    character(len=*), parameter :: lars_ADF_File_descript = "pattern_1"

    !> Make an array of random integers that we will use for sampling random
    !! fish from the whole population
    random_sample_individuals =
PERMUTE_RANDOM(proto_parents%population_size)


!+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    !> OUTPUT: Opening the output file for **gos label**.
    ! 1. we first set file name:
    lars_output_filename_data_gos_label = "0000_lars_gos_label_ADF_" //
&
                              lars_ADF_File_descript // csv
    ! 2. second, set internal file unit (we do not use the unit afterwards
but it is
    !!   used by fortran internally)
    lars_output_fileunit_data_gos_label = GET_FREE_FUNIT() ! get file unit
automatically
    ! 3. and physically open the output file for writing:
```

```
    call CSV_OPEN_WRITE ( lars_output_filename_data_gos_label,
&
                          lars_output_fileunit_data_gos_label )
    ! 4. producing a whole record with column labels using our function
    !    'do_row_header': VAR_001, VAR_002.... VAR_100
    lars_file_record_append_data_gos_label = do_row_header(100)
    ! 5. write this first record that contains column labels
    call CSV_RECORD_WRITE( record=lars_file_record_append_data_gos_label,
&

csv_file_name=lars_output_filename_data_gos_label )

    !> OUTPUT: Opening the output file for **gos arousal**.
    lars_output_filename_data_gos_arousal = "0000_lars_gos_arousal_ADF_"
//   &
                          lars_ADF_File_descript // csv
    lars_output_fileunit_data_gos_arousal = GET_FREE_FUNIT() ! get file unit
automatically
    call CSV_OPEN_WRITE ( lars_output_filename_data_gos_arousal,
&
                          lars_output_fileunit_data_gos_arousal )
    !> producing a whole record with column labels
    lars_file_record_append_data_gos_arousal = do_row_header(100)
    call CSV_RECORD_WRITE(
record=lars_file_record_append_data_gos_arousal,   &

csv_file_name=lars_output_filename_data_gos_arousal )

    !> OUTPUT: Opening the output file for **gos repeated counter**.
    lars_output_filename_data_gos_repeated = "0000_lars_gos_repeated_ADF_"
// &
                          lars_ADF_File_descript // csv
    lars_output_fileunit_data_gos_repeated = GET_FREE_FUNIT()  ! get file
unit automatically
    call CSV_OPEN_WRITE ( lars_output_filename_data_gos_repeated,
&
                          lars_output_fileunit_data_gos_repeated )
    !> producing a whole record with column labels
    lars_file_record_append_data_gos_repeated = do_row_header(100)
    call CSV_RECORD_WRITE(
record=lars_file_record_append_data_gos_repeated, &

csv_file_name=lars_output_filename_data_gos_repeated )

!++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

    ! First loop through a random sample of 10 fish out from the whole
population
    INDS: do j=1, 10

          ! Choose the current individual ID number to work with from the
random sample.
          ind = random_sample_individuals(j)

          ! Exclude dead fish.
          if (proto_parents%individual(ind)%is_dead()) then
```

```
            call LOG_MSG("WARNING: Found dead agent # " // TOSTR(ind) )
            exit INDS
          end if


!+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
          !> OUTPUT: Make the record an empty string when we start writing
data
          !! for each new individual
          lars_file_record_append_data_gos_label = ""
          lars_file_record_append_data_gos_arousal = ""
          lars_file_record_append_data_gos_repeated = ""

!+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

          ! Start random walks of the fish
          WALKS: do i=1, 100

            call LOG_DELIMITER(LOG_LEVEL_CHAPTER)
            call LOG_DBG("Agent walk no=" // TOSTR(i) // " , agent ID " //
&
                        TOSTR(proto_parents%individual(ind)%get_id()) //
&
                        " (# " // TOSTR(ind) // "), name:"
&
                        // proto_parents%individual(ind)%individ_label()
//".")

            ! do random walk
            step_rwalk = dist2step(170.0)
            call LOG_DBG("  Step size for random walk: " // TOSTR(step_rwalk)
// &
                          ", " // TOSTR(step_rwalk /
proto_parents%individual(ind)%get_length()) // &
                          " agent's body sizes." )

            call proto_parents%individual(ind)%rwalk( step_rwalk,0.5, &

habitat_safe)

            call LOG_DBG("  cycle ind:walk "// TOSTR(ind) // ":"// TOSTR(i)
// &

TOSTR(proto_parents%individual(ind)%location(.TRUE.)))
            call LOG_DBG("            way "//
&

TOSTR(proto_parents%individual(ind)%way()))

            cost_step =
proto_parents%individual(ind)%cost_swim_burst(step_rwalk)
            call LOG_DBG("  Cost of random walk step: " // TOSTR(cost_step)
// &
                          " is " // TOSTR(100.0_SRP * cost_step /
proto_parents%individual(ind)%body_mass ) // &
                          "% of agent's body mass." )
```

```
            !> Subtract the cost of swimming here:
proto_parents%individual(ind)%body_mass=proto_parents%individual(ind)%bod
y_mass - &
                                       cost_step

            !====================================================
            ! Inner perceptions: stomach, bodymass, energy, age
            call proto_parents%individual(ind)%perceptions_inner()

            !====================================================
            ! Environmental perceptions: light, depth
            call proto_parents%individual(ind)%perceptions_environ()
            call LOG_DBG("Environmental perceptions: light " //
&
TOSTR(proto_parents%individual(ind)%perceive_light%get_current()) // &
              ", depth " //
&
TOSTR(proto_parents%individual(ind)%perceive_depth%get_current()) )

            !====================================================
            ! Spatial perceptions food, conspecifics, predators
            call
proto_parents%individual(ind)%see_food(habitat_safe%food,1)

            call
proto_parents%individual(ind)%see_consp(proto_parents%individual,&
                                    proto_parents%individual%get_length(),
&
                                    proto_parents%individual%is_alive() )

            call
proto_parents%individual(ind)%see_pred(habitat_safe%predators, &
                    habitat_safe%predators%get_size())

            !====================================================
            call
proto_parents%individual(ind)%motivations_percept_components()
            call proto_parents%individual(ind)%motivations_primary_calc()
            call proto_parents%individual(ind)%modulation()
            call proto_parents%individual(ind)%motivations_to_memory()
            call proto_parents%individual(ind)%gos_find()


!++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!---------------------------------------------------------------------
            ! OUTPUT: We are to place some code for producing outputs of
motivational
            ! variables below here.
            call CSV_RECORD_APPEND(
lars_file_record_append_data_gos_label,   &
```

```
proto_parents%individual(ind)%gos_label() )

            call CSV_RECORD_APPEND(
lars_file_record_append_data_gos_arousal,   &

proto_parents%individual(ind)%arousal() )

            call CSV_RECORD_APPEND(
lars_file_record_append_data_gos_repeated,   &

proto_parents%individual(ind)%gos_repeated )

!++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


            !> Check if the fish has died of starvation
            if (proto_parents%individual(ind)%starved_death()) then
              call proto_parents%individual(ind)%dies()
              call LOG_DELIMITER(LOG_LEVEL_SECTION)
              call LOG_DBG ("INFO: Agent dies due to starvation, ID: " //
&

TOSTR(proto_parents%individual(ind)%get_id()))
              call LOG_DBG ("      Body length: " //
&
                    TOSTR(proto_parents%individual(ind)%body_length) //
&
                    ", body mass: " //
&
                    TOSTR(proto_parents%individual(ind)%body_mass) //
&
                    ", maximum mass: " //
&

TOSTR(proto_parents%individual(ind)%body_mass_maximum) // &
                    ", birth mass : " //
&
                    TOSTR(proto_parents%individual(ind)%body_mass_birth)
)
              call LOG_DBG("       Energy :" //
&
                    TOSTR(proto_parents%individual(ind)%energy_current)
//    &
                    ", energy maximum: " //
&
                    TOSTR(proto_parents%individual(ind)%energy_maximum)
)
              call LOG_DELIMITER(LOG_LEVEL_SECTION)
              exit WALKS
            end if

            call LOG_DBG( "GOS is      :" //
proto_parents%individual(ind)%gos_label() )
            call LOG_DBG( "GOS arousal :" //
TOSTR(proto_parents%individual(ind)%arousal()) )
```

```
            call LOG_DBG("**** can see food:  " //
TOSTR(proto_parents%individual(ind)%perceive_food%get_count()))

            !> Check if there is any food items in proximity (visibility
range)
            if ( proto_parents%individual(ind)%has_food() ) then
               call LOG_DBG("  distance    >" //
&

TOSTR(proto_parents%individual(ind)%perceive_food%foods_distances))
               call LOG_DBG("  dist. (d/l) >" //
&

TOSTR(proto_parents%individual(ind)%perceive_food%foods_distances &
                                /
proto_parents%individual(ind)%get_length()))

              !=====================================================
              call LOG_DBG("   +++ Current mass: " //
TOSTR(proto_parents%individual(ind)%mass()) //  &
                           ", length: " //
TOSTR(proto_parents%individual(ind)%length()) //       &
                           ", energy: " //
TOSTR(proto_parents%individual(ind)%get_energy())   )
              !> Select the optimal food item out from its perception:
              food_item_selected =
proto_parents%individual(ind)%food_item_select(rescale_max_motivation=6.0
_SRP)

              !> Try to eat the optimal food item:
              call
proto_parents%individual(ind)%food_item_eat(food_item_selected,
habitat_safe%food)

              call LOG_DBG("**** Tried to eat food item: " //
TOSTR(food_item_selected))
              call LOG_DBG("   +++ Updated mass: " //
TOSTR(proto_parents%individual(ind)%mass()) //  &
                           ", length: " //
TOSTR(proto_parents%individual(ind)%length()) //       &
                           ", energy: " //
TOSTR(proto_parents%individual(ind)%get_energy())   )
              !stop "EATEN"
            else
              !> If no food objects were encountered we still grow with zero
food gain.
              call proto_parents%individual(ind)%mass_grow(0.0_SRP)
              call proto_parents%individual(ind)%len_grow(0.0_SRP)
            end if

            call LOG_DBG("**** can see consp: " //
TOSTR(proto_parents%individual(ind)%perceive_consp%get_count() ) )
            if ( proto_parents%individual(ind)%has_consp() ) then
               call LOG_DBG("  coord(1)    >" //
&
```

```
TOSTR(proto_parents%individual(ind)%perceive_consp%conspecifics_seen(1)%l
ocation(.TRUE.)))
               call LOG_DBG("  iid         >" //
&

TOSTR(proto_parents%individual(ind)%perceive_consp%conspecifics_seen%get_
cid()))
            end if

            call LOG_DBG("**** can see pred:  " //
TOSTR(proto_parents%individual(ind)%perceive_predator%get_count() ) )
            if ( proto_parents%individual(ind)%has_pred() ) then
               call LOG_DBG("  coord(1)    =" //
&

TOSTR(proto_parents%individual(ind)%perceive_predator%predators_seen(1)%l
ocation(.TRUE.)))
               call LOG_DBG("  iid         =" //
&

TOSTR(proto_parents%individual(ind)%perceive_predator%predators_seen(1)%g
et_cid()))
               call LOG_DBG("  dist        =" //
&

TOSTR(proto_parents%individual(ind)%perceive_predator%predators_seen(1)%g
et_dist()))
            end if


          end do WALKS


!++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
          !> OUTPUT: Physically write the record to the disk
          call CSV_RECORD_WRITE(
record=lars_file_record_append_data_gos_label, &

csv_file_name=lars_output_filename_data_gos_label )

          call CSV_RECORD_WRITE(
record=lars_file_record_append_data_gos_arousal, &

csv_file_name=lars_output_filename_data_gos_arousal )

          call CSV_RECORD_WRITE(
record=lars_file_record_append_data_gos_repeated, &

csv_file_name=lars_output_filename_data_gos_repeated )

!++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


          call LOG_DBG("INFO: Subtracting cost of living for agent # " //
&
                    TOSTR(ind) // " and add weight and length to the
```

```
history.")

          !> Subtract the cost of living
          call proto_parents%individual(ind)%subtract_living_cost()

          call
add_to_history(proto_parents%individual(ind)%body_length_history, &
                              proto_parents%individual(ind)%body_length)

          call
add_to_history(proto_parents%individual(ind)%body_mass_history, &
                              proto_parents%individual(ind)%body_mass)

          if (proto_parents%individual(ind)%starved_death()) then
              call proto_parents%individual(ind)%dies_debug()
              call LOG_DELIMITER(LOG_LEVEL_SECTION)
              call LOG_DBG ("INFO: Agent dies due to starvation, ID: " //
&

TOSTR(proto_parents%individual(ind)%get_id()))
              call LOG_DBG ("      Body length: " //
&
                    TOSTR(proto_parents%individual(ind)%body_length) //
&
                    ", body mass: " //
&
                    TOSTR(proto_parents%individual(ind)%body_mass) //
&
                    ", maximum mass: " //
&

TOSTR(proto_parents%individual(ind)%body_mass_maximum) // &
                    ", birth mass : " //
&
                    TOSTR(proto_parents%individual(ind)%body_mass_birth)
)
              call LOG_DBG("       Energy :" //
&
                    TOSTR(proto_parents%individual(ind)%energy_current)
//    &
                    ", energy maximum: " //
&
                    TOSTR(proto_parents%individual(ind)%energy_maximum)
)
              call LOG_DELIMITER(LOG_LEVEL_SECTION)
            end if

    end do INDS


!++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    !> OUTPUT: Finally, we are closing the output files.
    call CSV_CLOSE( csv_file_name=lars_output_filename_data_gos_label )
    call CSV_CLOSE( csv_file_name=lars_output_filename_data_gos_arousal )
    call CSV_CLOSE( csv_file_name=lars_output_filename_data_gos_repeated )
```

```fortran
!+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  contains


!+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    ! OUTPUT: Produce a whole record with the names of the columns.
    function do_row_header(n_vars) result (string_record)
      integer, intent(in) :: n_vars
      character(len=2000) :: string_record

      !> Local vars
      integer :: i

      !> producing a whole record with column labels
      string_record = ""
      do i=1, n_vars
        call CSV_RECORD_APPEND( string_record, "VAR_" // TOSTR(i,n_vars) )
      end do

    end function do_row_header

  end subroutine LARS_TEST_LAB
```

A2. The Global Organismic State

```
  !> Find and set the global organismic state (GOS) based on the various
  !! available motivation  values.
  !! @note  GOS generation is a little changed in the new generation model.
  !!        1. We try to avoid constant switching of the GOS by requiring that
  !!           the difference between motivational components should exceed
  !!           some threshold value, if it does not, retain old GOS. So minor
  !!           fluctuations in the stimulus field are ignored. Threshold is
  !!           a dynamic parameter, so can also be zero.
  !!        2. The threshold is inversely related to the absolute value of
the
  !!           motivations compared, when the motivations are low, the
  !!           threshold is big, when their values are approaching 1, the
  !!           threshold approaches zero. So motivations have relatively
little
  !!           effects.
  subroutine gos_find_global_state(this)
    class(GOS_GLOBAL), intent(inout) :: this

    !> Local variables
    !> Arousal is the maximum level of motivation among all available new
    !! incoming motivations ones. But we still have the older/previous
"current"
    !! arousal value `%gos_arousal` until it is updated from the newly
incoming
    !! perceptions and motivations.
    real(SRP) :: arousal_new

    !> Dynamic threshold of GOS, the threshold a motivation has to exceed to
    !! win the competition with the current motivation.
    real(SRP) :: gos_dthreshold

    !> PROCNAME is the procedure name for logging and debugging (with
MODNAME).
    character(len=*), parameter :: PROCNAME = "(gos_find_global_state)"

    !> Arousal is the maximum level among all available motivations (**final**
    !! motivational components). This is the **new** state depending on all
    !! the currently incoming perceptions.
    arousal_new = this%motivations%max_final()

    !> The GOS competition threshold is a function of the current arousal
    !! level, if it is very low, we need a relatively high competing motivation
    !! to win competition, if it is high (1) then very small difference is
    !! enough. But note that this is the relative differences. So if we have
    !! a low motivation 0.1, we need 0.155 to win (threshold=0.55,
    !! 0.155=0.1+0.1Ã—0.55 ), but if we have high motivation 0.8, almost any
    !! exceeding  motivation (>0.808) will win. So we limit the possible
    !! effects of low motivations. We get the actual value as a nonparametric
    !! function, currently by nonlinear interpolation of the grid values
    !! defined by the `MOTIVATION_COMPET_THRESHOLD_CURVE_` parameter arrays.
    !! @plot `aha_gos_arousal_winthreshold.svg`
    gos_dthreshold = DDPINTERPOL(
```

```
MOTIVATION_COMPET_THRESHOLD_CURVE_ABSCISSA, &

MOTIVATION_COMPET_THRESHOLD_CURVE_ORDINATE, &
                                  this%gos_arousal )

    !> Save the interpolation plot in the debug mode using external command.
    !! @warning Involves **huge** number of plots, should normally be
    !!          disabled.
    call debug_interpolate_plot_save(
&
            grid_xx=MOTIVATION_COMPET_THRESHOLD_CURVE_ABSCISSA,
&
            grid_yy=MOTIVATION_COMPET_THRESHOLD_CURVE_ORDINATE,
&
            ipol_value=this%gos_arousal, algstr="DDPINTERPOL",
&
            output_file="plot_debug_arousal_gos_threshold_" //
&
                        TOSTR(Global_Time_Step_Model_Current) //
&
                        TAG_MMDD() // "_a_"// trim(this%individ_label()) //
&
                        "_" // RAND_STRING(LABEL_LENGTH,
LABEL_CST,LABEL_CEN) &
                        // PS )


    !> Now as we have the dynamic threshold, we can compare the current
    !! motivation level with the current (previous) arousal. If the motivation
    !! exceeds the current arousal by more than the threshold, the GOS
    !! changes to the new motivation. If not, we are still left with the
    !! previous GOS.
    AROUSAL_THRESHOLD: if (arousal_new - this%gos_arousal <
&
                                          gos_dthreshold * this%gos_arousal)
then
      !> If the maximum current arousal does not exceed the threshold,
      !! we are left with the old GOS. However, we reduce the current arousal
      !! spontaneously using a simple linear or some non-linear dissipation
      !! pattern using the `%gos_repeated` parameter that sets the number of
      !! repeated occurrences of the same (current) GOS.
      !! First, increment GOS repeat counter.
      this%gos_repeated = this%gos_repeated + 1
      !> And spontaneously decrease, **dissipate**, the current arousal
level.
      !! Spontaneous dissipation of arousal is implemented by multiplying the
      !! current level by a factor within the range [0.0..1.0] that can depend
      !! on the number of times this GOS is repeated.
      !! @note Note that the dissipation function is local to this procedure.
      !!        `arousal_decrease_factor_fixed` = fixed value
      !!        `arousal_decrease_factor_nonpar` = nonlinear,
nonparametric,
      !!        based on nonlinear interpolation.
      !! @plot `aha_gos_arousal_dissipation.svg`
      this%gos_arousal = this%gos_arousal *
&
```

```
arousal_decrease_factor_nonpar(this%gos_repeated)
    else AROUSAL_THRESHOLD
      !> If the maximum new arousal exceeds the threshold, we get to a
      !! **new GOS**. That is, the **highest** among the **new** competing
      !! motivations defines the new GOS.
      !! @note Use `associate`construct to set alias for long object
hierarchy.
      !! @note Note that `this%gos_repeated` is initialised to 1 at
`gos_reset`.
      associate ( MOT => this%motivations )
        !> Check **hunger**.
        GOS_IS_MAX: if (MOT%is_max_final(MOT%hunger)) then
          !> Reset all motivations to **non-dominant**.
          call this%gos_reset()
          !> Set new GOS for hunger...
          MOT%hunger%dominant_state = .TRUE.
          this%gos_main = MOT%hunger%label
          this%gos_arousal = MOT%hunger%motivation_finl
        !> Check **passive_avoidance**.
        else if (MOT%is_max_final(MOT%avoid_passive)) then GOS_IS_MAX
          !> Reset all motivations to **non-dominant**.
          call this%gos_reset()
          !> Set new GOS for passive_avoidance...
          MOT%avoid_passive%dominant_state = .TRUE.
          this%gos_main = MOT%avoid_passive%label
          this%gos_arousal = MOT%avoid_passive%motivation_finl
        !> Check **active_avoidance**.
        else if (MOT%is_max_final(MOT%avoid_active)) then GOS_IS_MAX
          !> Reset all motivations to **non-dominant**.
          call this%gos_reset()
          !> Set new GOS for active_avoidance...
          MOT%avoid_active%dominant_state = .TRUE.
          this%gos_main = MOT%avoid_active%label
          this%gos_arousal = MOT%avoid_active%motivation_finl
        !> Check **reproduction**.
        else if (MOT%is_max_final(MOT%reproduction)) then GOS_IS_MAX
          !> Reset all motivations to **non-dominant**.
          call this%gos_reset()
          !> Set new GOS for reproduction...
          MOT%reproduction%dominant_state = .TRUE.
          this%gos_main = MOT%reproduction%label
          this%gos_arousal = MOT%reproduction%motivation_finl
        end if GOS_IS_MAX
      end associate

    end if AROUSAL_THRESHOLD

    !> Add the current GOS parameters to the emotional memory stack
    !! @note Note that the memory stack arrays are defined in
    !!       APPRAISAL and cleaned/init in `init_appraisal`
    !! @note We can use the dedicated procedures. Here disabled so far to avoid
    !!       speed overhead.
    !call this%memory_motivations%gos_to_memory(
&
    !                v_gos_label=this%gos_main,
```

```
&
    !                 v_gos_arousal= this%gos_arousal,
&
    !                 v_gos_repeated=this%gos_repeated  )
    call add_to_history(this%memory_motivations%gos_main, this%gos_main)
    call add_to_history(this%memory_motivations%gos_arousal,
this%gos_arousal)
    call add_to_history(this%memory_motivations%gos_repeated,
this%gos_repeated)


    !> Finally recalculate the attention weights for all the states'
perception
    !! components. The dominant GOS state will now get its default attention
    !! weights whereas all non-dominant states will get modulated values, i.e.
    !! values recalculated from a non-linear interpolation based **attention
    !! modulation curve**.
    call this%attention_modulate()

    !! @note  Note that type-bound functions can be used (although this makes
    !!        sense only outside of this module to avoid a small function-call
    !!        overhead): `if ( this%motivations%hunger%is_dominant() )
then`. For the
    !!        motivational state label we can use the accessor function
    !!        `%label_is` : `return_gos =
this%motivations%hunger%label_is()` (it is
    !!        **mandatory** outside of this module as label is declared
    !!        `private`).
    if (this%motivations%hunger%dominant_state) then
      return_gos = this%motivations%hunger%label
    else if (this%motivations%avoid_passive%dominant_state) then
      return_gos = this%motivations%avoid_passive%label
    else if (this%motivations%avoid_active%dominant_state) then
      return_gos = this%motivations%avoid_active%label
    else if (this%motivations%reproduction%dominant_state) then
      return_gos = this%motivations%reproduction%label
    end if

  end function gos_global_get_label


!-----------------------------------------------------------------------
-----
  !> Calculate the overall level of arousal. Arousal is the current level
  !! of the dominant motivation that has brought about the current GOS at the
  !! previous time step.
  elemental function gos_get_arousal_level(this) result (arousal_out)
    class(GOS_GLOBAL), intent(in) :: this

    !> Arousal is the current level of motivation that has brought about GOS.
    real(SRP) :: arousal_out

    !> It is saved in this GOS-object component.
    arousal_out = this%gos_arousal

  end function gos_get_arousal_level
```

A3. The Attention Modulation Factor

```
  !> Modulate the attention weights to suppress all perceptions alternative
  !! to the current GOS. This is done using the attention modulation
  !! interpolation curve.
  !! @warning This subroutine is called from within `gos_find` and should not
  !!          be called separately.
  subroutine gos_attention_modulate_weights(this)
    class(GOS_GLOBAL), intent(inout) :: this

    !> Local variable, the weight given to the attention weight components
    !! of all the non-dominant motivation states. Based on nonlinear
    !! interpolation.
    real(SRP) :: percept_w

    !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - -
    !> **First**, we calculate the attention weight given to all non-dominant
    !! perceptions via nonlinear interpolation.
    percept_w = DDPINTERPOL( ATTENTION_MODULATION_CURVE_ABSCISSA,
&
                             ATTENTION_MODULATION_CURVE_ORDINATE,
&
                             this%gos_arousal )

    !> Save the interpolation plot in the debug mode using external command.
    !! @warning Involves **huge** number of plots, should normally be
    !!          disabled.
    call debug_interpolate_plot_save(
&
            grid_xx=ATTENTION_MODULATION_CURVE_ABSCISSA,
&
            grid_yy=ATTENTION_MODULATION_CURVE_ORDINATE,
&
            ipol_value=this%gos_arousal, algstr="DDPINTERPOL",
&
            output_file="plot_debug_attention_modulation_" //
&
                        TOSTR(Global_Time_Step_Model_Current) //
&
                        TAG_MMDD() // "_a_"// trim(this%individ_label()) //
&
                        "_" // RAND_STRING(LABEL_LENGTH,
LABEL_CST,LABEL_CEN) &
                        // PS )

    !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - -
    !> **Second**, we reset the attention weights for the **dominant GOS
    !! state** to their **default** parameter values whereas for all other
    !! states, to the **recalculated** `percept_w` modulated
    !! value.
    !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
- - -
    !> The **dominant** state is **hunger**:
    RESET_DOMINANT: if ( this%motivations%hunger%is_dominant() ) then

      !> @note Dominant is **hunger**.
      call this%motivations%hunger%attention_weight%attention_init
&
          (weight_light     = ATTENTION_WEIGHT_HUNGER_LIGHT,
&
           weight_depth     = ATTENTION_WEIGHT_HUNGER_DEPTH,
&
           weight_food_dir  = ATTENTION_WEIGHT_HUNGER_FOOD_DIR,
&
           weight_food_mem  = ATTENTION_WEIGHT_HUNGER_FOOD_MEM,
&
           weight_conspec   = ATTENTION_WEIGHT_HUNGER_CONSPEC,
&
           weight_predator  = ATTENTION_WEIGHT_HUNGER_PREDATOR,
&
           weight_stomach   = ATTENTION_WEIGHT_HUNGER_STOMACH,
&
           weight_bodymass  = ATTENTION_WEIGHT_HUNGER_BODYMASS,
&
           weight_energy    = ATTENTION_WEIGHT_HUNGER_ENERGY,
&
           weight_age       = ATTENTION_WEIGHT_HUNGER_AGE,
&
           weight_reprfac   = ATTENTION_WEIGHT_HUNGER_REPRFAC )

      call this%motivations%avoid_passive%attention_weight%attention_init
&
          (weight_light     = ATTENTION_WEIGHT_AVOID_PASS_LIGHT *
percept_w,   &
           weight_depth     = ATTENTION_WEIGHT_AVOID_PASS_DEPTH *
percept_w,   &
           weight_food_dir  = ATTENTION_WEIGHT_AVOID_PASS_FOOD_DIR *
percept_w,&
           weight_food_mem  = ATTENTION_WEIGHT_AVOID_PASS_FOOD_MEM *
percept_w,&
           weight_conspec   = ATTENTION_WEIGHT_AVOID_PASS_CONSPEC *
percept_w, &
           weight_predator  = ATTENTION_WEIGHT_AVOID_PASS_PREDATOR *
percept_w,&
           weight_stomach   = ATTENTION_WEIGHT_AVOID_PASS_STOMACH *
percept_w, &
           weight_bodymass  = ATTENTION_WEIGHT_AVOID_PASS_BODYMASS *
percept_w,&
           weight_energy    = ATTENTION_WEIGHT_AVOID_PASS_ENERGY *
percept_w,  &
           weight_age       = ATTENTION_WEIGHT_AVOID_PASS_AGE * percept_w,
&
           weight_reprfac   = ATTENTION_WEIGHT_AVOID_PASS_REPRFAC *
percept_w )

      call this%motivations%avoid_active%attention_weight%attention_init
&
```

```
          (weight_light     = ATTENTION_WEIGHT_AVOID_ACT_LIGHT * percept_w,
&
           weight_depth     = ATTENTION_WEIGHT_AVOID_ACT_DEPTH * percept_w,
&
           weight_food_dir = ATTENTION_WEIGHT_AVOID_ACT_FOOD_DIR *
percept_w, &
           weight_food_mem = ATTENTION_WEIGHT_AVOID_ACT_FOOD_MEM *
percept_w, &
           weight_conspec   = ATTENTION_WEIGHT_AVOID_ACT_CONSPEC *
percept_w,  &
           weight_predator = ATTENTION_WEIGHT_AVOID_ACT_PREDATOR *
percept_w, &
           weight_stomach   = ATTENTION_WEIGHT_AVOID_ACT_STOMACH *
percept_w,  &
           weight_bodymass = ATTENTION_WEIGHT_AVOID_ACT_BODYMASS *
percept_w, &
           weight_energy    = ATTENTION_WEIGHT_AVOID_ACT_ENERGY *
percept_w,   &
           weight_age       = ATTENTION_WEIGHT_AVOID_ACT_AGE * percept_w,
&
           weight_reprfac   = ATTENTION_WEIGHT_AVOID_ACT_REPRFAC *
percept_w )

      call this%motivations%reproduction%attention_weight%attention_init
&
          (weight_light     = ATTENTION_WEIGHT_REPRODUCE_LIGHT * percept_w,
&
           weight_depth     = ATTENTION_WEIGHT_REPRODUCE_DEPTH * percept_w,
&
           weight_food_dir = ATTENTION_WEIGHT_REPRODUCE_FOOD_DIR *
percept_w, &
           weight_food_mem = ATTENTION_WEIGHT_REPRODUCE_FOOD_MEM *
percept_w, &
           weight_conspec   = ATTENTION_WEIGHT_REPRODUCE_CONSPEC *
percept_w,  &
           weight_predator = ATTENTION_WEIGHT_REPRODUCE_PREDATOR *
percept_w, &
           weight_stomach   = ATTENTION_WEIGHT_REPRODUCE_STOMACH *
percept_w,  &
           weight_bodymass = ATTENTION_WEIGHT_REPRODUCE_BODYMASS *
percept_w, &
           weight_energy    = ATTENTION_WEIGHT_REPRODUCE_ENERGY *
percept_w,   &
           weight_age       = ATTENTION_WEIGHT_REPRODUCE_AGE * percept_w,
&
           weight_reprfac   = ATTENTION_WEIGHT_REPRODUCE_REPRFAC *
percept_w )

    !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - -
    !> The **dominant** state is **avoid_passive**:
    else if ( this%motivations%avoid_passive%is_dominant() ) then
RESET_DOMINANT

      call this%motivations%hunger%attention_weight%attention_init
&
```

```
            (weight_light     = ATTENTION_WEIGHT_HUNGER_LIGHT * percept_w,
&
             weight_depth     = ATTENTION_WEIGHT_HUNGER_DEPTH * percept_w,
&
             weight_food_dir = ATTENTION_WEIGHT_HUNGER_FOOD_DIR * percept_w,
&
             weight_food_mem = ATTENTION_WEIGHT_HUNGER_FOOD_MEM * percept_w,
&
             weight_conspec   = ATTENTION_WEIGHT_HUNGER_CONSPEC * percept_w,
&
             weight_predator = ATTENTION_WEIGHT_HUNGER_PREDATOR * percept_w,
&
             weight_stomach   = ATTENTION_WEIGHT_HUNGER_STOMACH * percept_w,
&
             weight_bodymass = ATTENTION_WEIGHT_HUNGER_BODYMASS * percept_w,
&
             weight_energy    = ATTENTION_WEIGHT_HUNGER_ENERGY * percept_w,
&
             weight_age       = ATTENTION_WEIGHT_HUNGER_AGE * percept_w,
&
             weight_reprfac   = ATTENTION_WEIGHT_HUNGER_REPRFAC * percept_w )

        !> @note Dominant **avoid_passive**.
        call this%motivations%avoid_passive%attention_weight%attention_init
&
            (weight_light     = ATTENTION_WEIGHT_AVOID_PASS_LIGHT,
&
             weight_depth     = ATTENTION_WEIGHT_AVOID_PASS_DEPTH,
&
             weight_food_dir = ATTENTION_WEIGHT_AVOID_PASS_FOOD_DIR,
&
             weight_food_mem = ATTENTION_WEIGHT_AVOID_PASS_FOOD_MEM,
&
             weight_conspec   = ATTENTION_WEIGHT_AVOID_PASS_CONSPEC,
&
             weight_predator = ATTENTION_WEIGHT_AVOID_PASS_PREDATOR,
&
             weight_stomach   = ATTENTION_WEIGHT_AVOID_PASS_STOMACH,
&
             weight_bodymass = ATTENTION_WEIGHT_AVOID_PASS_BODYMASS,
&
             weight_energy    = ATTENTION_WEIGHT_AVOID_PASS_ENERGY,
&
             weight_age       = ATTENTION_WEIGHT_AVOID_PASS_AGE,
&
             weight_reprfac   = ATTENTION_WEIGHT_AVOID_PASS_REPRFAC )

        call this%motivations%avoid_active%attention_weight%attention_init
&
            (weight_light     = ATTENTION_WEIGHT_AVOID_ACT_LIGHT * percept_w,
&
             weight_depth     = ATTENTION_WEIGHT_AVOID_ACT_DEPTH * percept_w,
&
             weight_food_dir = ATTENTION_WEIGHT_AVOID_ACT_FOOD_DIR *
percept_w,  &
             weight_food_mem = ATTENTION_WEIGHT_AVOID_ACT_FOOD_MEM *
```

```
percept_w, &
           weight_conspec  = ATTENTION_WEIGHT_AVOID_ACT_CONSPEC *
percept_w,  &
           weight_predator = ATTENTION_WEIGHT_AVOID_ACT_PREDATOR *
percept_w, &
           weight_stomach  = ATTENTION_WEIGHT_AVOID_ACT_STOMACH *
percept_w,  &
           weight_bodymass = ATTENTION_WEIGHT_AVOID_ACT_BODYMASS *
percept_w, &
           weight_energy   = ATTENTION_WEIGHT_AVOID_ACT_ENERGY *
percept_w,   &
           weight_age      = ATTENTION_WEIGHT_AVOID_ACT_AGE * percept_w,
&
           weight_reprfac  = ATTENTION_WEIGHT_AVOID_ACT_REPRFAC *
percept_w )

      call this%motivations%reproduction%attention_weight%attention_init
&
          (weight_light    = ATTENTION_WEIGHT_REPRODUCE_LIGHT * percept_w,
&
           weight_depth    = ATTENTION_WEIGHT_REPRODUCE_DEPTH * percept_w,
&
           weight_food_dir = ATTENTION_WEIGHT_REPRODUCE_FOOD_DIR *
percept_w, &
           weight_food_mem = ATTENTION_WEIGHT_REPRODUCE_FOOD_MEM *
percept_w, &
           weight_conspec  = ATTENTION_WEIGHT_REPRODUCE_CONSPEC *
percept_w,  &
           weight_predator = ATTENTION_WEIGHT_REPRODUCE_PREDATOR *
percept_w, &
           weight_stomach  = ATTENTION_WEIGHT_REPRODUCE_STOMACH *
percept_w,  &
           weight_bodymass = ATTENTION_WEIGHT_REPRODUCE_BODYMASS *
percept_w, &
           weight_energy   = ATTENTION_WEIGHT_REPRODUCE_ENERGY *
percept_w,   &
           weight_age      = ATTENTION_WEIGHT_REPRODUCE_AGE * percept_w,
&
           weight_reprfac  = ATTENTION_WEIGHT_REPRODUCE_REPRFAC *
percept_w )

    !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - -
    !> The **dominant** state is **avoid_active**:
    else if ( this%motivations%avoid_active%is_dominant() ) then
RESET_DOMINANT

      call this%motivations%hunger%attention_weight%attention_init
&
          (weight_light    = ATTENTION_WEIGHT_HUNGER_LIGHT * percept_w,
&
           weight_depth    = ATTENTION_WEIGHT_HUNGER_DEPTH * percept_w,
&
           weight_food_dir = ATTENTION_WEIGHT_HUNGER_FOOD_DIR * percept_w,
&
           weight_food_mem = ATTENTION_WEIGHT_HUNGER_FOOD_MEM * percept_w,
```

```
&
           weight_conspec  = ATTENTION_WEIGHT_HUNGER_CONSPEC * percept_w,
&
           weight_predator = ATTENTION_WEIGHT_HUNGER_PREDATOR * percept_w,
&
           weight_stomach  = ATTENTION_WEIGHT_HUNGER_STOMACH * percept_w,
&
           weight_bodymass = ATTENTION_WEIGHT_HUNGER_BODYMASS * percept_w,
&
           weight_energy   = ATTENTION_WEIGHT_HUNGER_ENERGY * percept_w,
&
           weight_age      = ATTENTION_WEIGHT_HUNGER_AGE * percept_w,
&
           weight_reprfac  = ATTENTION_WEIGHT_HUNGER_REPRFAC * percept_w
)

      call this%motivations%avoid_passive%attention_weight%attention_init
&
          (weight_light    = ATTENTION_WEIGHT_AVOID_PASS_LIGHT *
percept_w,   &
           weight_depth    = ATTENTION_WEIGHT_AVOID_PASS_DEPTH *
percept_w,   &
           weight_food_dir = ATTENTION_WEIGHT_AVOID_PASS_FOOD_DIR *
percept_w,&
           weight_food_mem = ATTENTION_WEIGHT_AVOID_PASS_FOOD_MEM *
percept_w,&
           weight_conspec  = ATTENTION_WEIGHT_AVOID_PASS_CONSPEC *
percept_w, &
           weight_predator = ATTENTION_WEIGHT_AVOID_PASS_PREDATOR *
percept_w,&
           weight_stomach  = ATTENTION_WEIGHT_AVOID_PASS_STOMACH *
percept_w, &
           weight_bodymass = ATTENTION_WEIGHT_AVOID_PASS_BODYMASS *
percept_w,&
           weight_energy   = ATTENTION_WEIGHT_AVOID_PASS_ENERGY *
percept_w,  &
           weight_age      = ATTENTION_WEIGHT_AVOID_PASS_AGE * percept_w,
&
           weight_reprfac  = ATTENTION_WEIGHT_AVOID_PASS_REPRFAC *
percept_w )

      !> @note Dominant is **avoid_active**.
      call this%motivations%avoid_active%attention_weight%attention_init
&
          (weight_light    = ATTENTION_WEIGHT_AVOID_ACT_LIGHT,
&
           weight_depth    = ATTENTION_WEIGHT_AVOID_ACT_DEPTH,
&
           weight_food_dir = ATTENTION_WEIGHT_AVOID_ACT_FOOD_DIR,
&
           weight_food_mem = ATTENTION_WEIGHT_AVOID_ACT_FOOD_MEM,
&
           weight_conspec  = ATTENTION_WEIGHT_AVOID_ACT_CONSPEC,
&
           weight_predator = ATTENTION_WEIGHT_AVOID_ACT_PREDATOR,
&
```

```
            weight_stomach   = ATTENTION_WEIGHT_AVOID_ACT_STOMACH,
&
            weight_bodymass = ATTENTION_WEIGHT_AVOID_ACT_BODYMASS,
&
            weight_energy    = ATTENTION_WEIGHT_AVOID_ACT_ENERGY,
&
            weight_age       = ATTENTION_WEIGHT_AVOID_ACT_AGE,
&
            weight_reprfac   = ATTENTION_WEIGHT_AVOID_ACT_REPRFAC )
      call this%motivations%reproduction%attention_weight%attention_init
&
           (weight_light     = ATTENTION_WEIGHT_REPRODUCE_LIGHT * percept_w,
&
            weight_depth     = ATTENTION_WEIGHT_REPRODUCE_DEPTH * percept_w,
&
            weight_food_dir = ATTENTION_WEIGHT_REPRODUCE_FOOD_DIR *
percept_w, &
            weight_food_mem = ATTENTION_WEIGHT_REPRODUCE_FOOD_MEM *
percept_w, &
            weight_conspec   = ATTENTION_WEIGHT_REPRODUCE_CONSPEC *
percept_w,  &
            weight_predator = ATTENTION_WEIGHT_REPRODUCE_PREDATOR *
percept_w, &
            weight_stomach   = ATTENTION_WEIGHT_REPRODUCE_STOMACH *
percept_w,  &
            weight_bodymass = ATTENTION_WEIGHT_REPRODUCE_BODYMASS *
percept_w, &
            weight_energy    = ATTENTION_WEIGHT_REPRODUCE_ENERGY *
percept_w,   &
            weight_age       = ATTENTION_WEIGHT_REPRODUCE_AGE * percept_w,
&
            weight_reprfac   = ATTENTION_WEIGHT_REPRODUCE_REPRFAC *
percept_w )

    !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -
    !> The **dominant** state is **reproduction**:
    else if ( this%motivations%reproduction%is_dominant() ) then
RESET_DOMINANT

      call this%motivations%hunger%attention_weight%attention_init
&
           (weight_light     = ATTENTION_WEIGHT_HUNGER_LIGHT * percept_w,
&
            weight_depth     = ATTENTION_WEIGHT_HUNGER_DEPTH * percept_w,
&
            weight_food_dir = ATTENTION_WEIGHT_HUNGER_FOOD_DIR * percept_w,
&
            weight_food_mem = ATTENTION_WEIGHT_HUNGER_FOOD_MEM * percept_w,
&
            weight_conspec   = ATTENTION_WEIGHT_HUNGER_CONSPEC * percept_w,
&
            weight_predator = ATTENTION_WEIGHT_HUNGER_PREDATOR * percept_w,
&
            weight_stomach   = ATTENTION_WEIGHT_HUNGER_STOMACH * percept_w,
```

```
&
            weight_bodymass = ATTENTION_WEIGHT_HUNGER_BODYMASS * percept_w,
&
            weight_energy   = ATTENTION_WEIGHT_HUNGER_ENERGY * percept_w,
&
            weight_age      = ATTENTION_WEIGHT_HUNGER_AGE * percept_w,
&
            weight_reprfac  = ATTENTION_WEIGHT_HUNGER_REPRFAC * percept_w )

     call this%motivations%avoid_passive%attention_weight%attention_init
&
           (weight_light    = ATTENTION_WEIGHT_AVOID_PASS_LIGHT *
percept_w,    &
            weight_depth    = ATTENTION_WEIGHT_AVOID_PASS_DEPTH *
percept_w,    &
            weight_food_dir = ATTENTION_WEIGHT_AVOID_PASS_FOOD_DIR *
percept_w,&
            weight_food_mem = ATTENTION_WEIGHT_AVOID_PASS_FOOD_MEM *
percept_w,&
            weight_conspec  = ATTENTION_WEIGHT_AVOID_PASS_CONSPEC *
percept_w,  &
            weight_predator = ATTENTION_WEIGHT_AVOID_PASS_PREDATOR *
percept_w,&
            weight_stomach  = ATTENTION_WEIGHT_AVOID_PASS_STOMACH *
percept_w,  &
            weight_bodymass = ATTENTION_WEIGHT_AVOID_PASS_BODYMASS *
percept_w,&
            weight_energy   = ATTENTION_WEIGHT_AVOID_PASS_ENERGY *
percept_w,   &
            weight_age      = ATTENTION_WEIGHT_AVOID_PASS_AGE * percept_w,
&
            weight_reprfac  = ATTENTION_WEIGHT_AVOID_PASS_REPRFAC *
percept_w )

     call this%motivations%avoid_active%attention_weight%attention_init
&
           (weight_light    = ATTENTION_WEIGHT_AVOID_ACT_LIGHT * percept_w,
&
            weight_depth    = ATTENTION_WEIGHT_AVOID_ACT_DEPTH * percept_w,
&
            weight_food_dir = ATTENTION_WEIGHT_AVOID_ACT_FOOD_DIR *
percept_w,  &
            weight_food_mem = ATTENTION_WEIGHT_AVOID_ACT_FOOD_MEM *
percept_w,  &
            weight_conspec  = ATTENTION_WEIGHT_AVOID_ACT_CONSPEC *
percept_w,   &
            weight_predator = ATTENTION_WEIGHT_AVOID_ACT_PREDATOR *
percept_w,  &
            weight_stomach  = ATTENTION_WEIGHT_AVOID_ACT_STOMACH *
percept_w,   &
            weight_bodymass = ATTENTION_WEIGHT_AVOID_ACT_BODYMASS *
percept_w,  &
            weight_energy   = ATTENTION_WEIGHT_AVOID_ACT_ENERGY *
percept_w,    &
            weight_age      = ATTENTION_WEIGHT_AVOID_ACT_AGE * percept_w,
&
```

```
            weight_reprfac  = ATTENTION_WEIGHT_AVOID_ACT_REPRFAC *
percept_w )

      !> @note Dominant **reproduction**.
      call this%motivations%reproduction%attention_weight%attention_init
&
           (weight_light    = ATTENTION_WEIGHT_REPRODUCE_LIGHT,
&
            weight_depth    = ATTENTION_WEIGHT_REPRODUCE_DEPTH,
&
            weight_food_dir = ATTENTION_WEIGHT_REPRODUCE_FOOD_DIR,
&
            weight_food_mem = ATTENTION_WEIGHT_REPRODUCE_FOOD_MEM,
&
            weight_conspec  = ATTENTION_WEIGHT_REPRODUCE_CONSPEC,
&
            weight_predator = ATTENTION_WEIGHT_REPRODUCE_PREDATOR,
&
            weight_stomach  = ATTENTION_WEIGHT_REPRODUCE_STOMACH,
&
            weight_bodymass = ATTENTION_WEIGHT_REPRODUCE_BODYMASS,
&
            weight_energy   = ATTENTION_WEIGHT_REPRODUCE_ENERGY,
&
            weight_age      = ATTENTION_WEIGHT_REPRODUCE_AGE,
&
            weight_reprfac  = ATTENTION_WEIGHT_REPRODUCE_REPRFAC )

    end if RESET_DOMINANT

  end subroutine gos_attention_modulate_weights
```

A4. R-script for statistical analysis

```
# Breakpoint linear regression, unconstrained, single breakpoint,
#   In this model x is ADF, y is AVERAGE GOS streak (average)
#
# Based on the method from:
#
https://www.r-bloggers.com/r-for-ecologists-putting-together-a-piecew
ise-regression/
#---------------------------------------------------------------------
-----------
# # SVN version info:
# $Id: script.breakpoint.R 3086 2017-03-20 19:02:56Z sbu062 $
#---------------------------------------------------------------------
-----------

#####################################################################
###########
# Function to perform a breakdown linear model and determine a breakdown
point.
# the optimal breakdown is determined using the standard parametric sigma
# (standard deviation of the residuals) or AIC.
# NOTE: In the function ADF is the independent variable (x) and
#       AVERAGE is the dependent variable (y))
breakdown.linear.model <- function(ADF, AVERAGE,
                                   search_min=0.4, search_max=0.99,
                                   min_sigma=TRUE,
                                   xlabel= "Predictor",
                                   ylabel= "Response")
{

  # Make a variable to keep range of breakpoints
  breaks <- ADF[which(ADF >= search_min & ADF <= search_max)]


#---------------------------------------------------------------------
---------
  # Iteratively search breakpoints for the model minimize residual MSE
or AIC
  mse <- numeric(length(breaks))  # Vector to keep residual MSE
  aics <- numeric(length(breaks)) # Vector to keep AIC values

  for(i in 1:length(breaks)){
   model.piecewise.part <- lm(AVERAGE ~ ADF*(ADF < breaks[i])
                                                         +
ADF*(ADF>=breaks[i]))
   # Calculate residual standard deviation (sigma)
   mse[i] <- summary(model.piecewise.part)[6] # obtained from summary
   #mse[i] <- sigma(model.piecewise.part)     # or 'sigma' function
   # Calculate AIC, Akaike Information Criterion value
```

```
    aics[i] <- AIC(model.piecewise.part)
  }

  # Print actual breakpoint vector to search the optimum within.
  print("The range of breakpoints to optimise:")
  print(breaks)

  # MSEs AICs are keept in these vectors
  mse <- as.numeric(mse) # require it to make mse a vector
  print("Output all values of 'sigma' and AIC:")
  print(mse)   # print sigmas
  print(aics)  # print AIC

  print("Minimum AIC for the broken model:")
  print(min(aics))

  # The best model and respectively the optimal breakpoint is that which
  # minimises the standard deviation of the residuals (MSE) or AIC.
  min_mse <- breaks[which(mse==min(mse))]
  min_aics <- breaks[which(aics==min(aics))]
  print ("ADF Breakpoint based on sigma and AIC:")
  print(min_mse)   # print these values
  print(min_aics)

  # The breakpoint can be based either on MSE or AIC
  if ( min_sigma ) {
    point <- min_mse
    print("Optimisation is based on 'sigma'.")
    }
  else {
    point <- min_aics
    print("Optimisation is based on AIC.")
    }
  print("The actual breakpoint value is:")
  print(point)

  # Run the final model
  model.piecewise <- lm(AVERAGE ~ ADF*(ADF < point) + ADF*(ADF > point))
  print("Final fitted model parameters:")
  print( summary(model.piecewise) )


#----------------------------------------------------------------------
---------
  # Plotting the two-part linear regression
  # 1. basic scatterplot
  plot(ADF,AVERAGE, ylim = c(0,30),  pch=16, xlab=xlabel, ylab=ylabel)
  # 2. first part of the linear curve with parameter estimates from model
  # summary
  curve((model.piecewise$coefficients[1] +
model.piecewise$coefficients[3]) +
        (model.piecewise$coefficients[2] +
model.piecewise$coefficients[5]) * x,
```

```
        add=T, from=0, to=point)
  # 3. second part of the linear curvem after the breakpoint...
  curve((model.piecewise$coefficients[1] +
model.piecewise$coefficients[4]) +
        model.piecewise$coefficients[2] * x,
        add=T, from=point, to=max(ADF))
  # 4. vertical breakpoint line
  abline(v=point, lty=3)


#--------------------------------------------------------------------
---------
  # Also plot the breakpoint minimum as bars of MSE or AIC
  print(mse)
  barplot(mse, names.arg = breaks,
            ylab="Standard deviation of residuals", xlab="Breakpoint")
  print(aics)
  barplot(aics, names.arg = breaks, ylab="AIC", xlab="Breakpoint")

}
####################################################################
###########

# Data analysis using this function

# Data are obtained from the CSV data file:
streaks <- read.csv("streaks4_switch.csv")

# Data is saved as 'streaks', attach first
attach(streaks)

# Do the data analysis: breakdown model
breakdown.linear.model(ADF, SWITCHES, 0.4, 1.0, FALSE, "ADF", "Number of
switches")

# Do additional data analysis: single line model
model.nobroken <- lm(SWITCHES ~ ADF)
summary(model.nobroken)
plot (ADF, SWITCHES, ylim=c(0,30), pch=16, ylab="Number of switches")
abline( summary(model.nobroken)$coefficients[1],
        summary(model.nobroken)$coefficients[2] )
print("AIC for the Single-line model:")
print(AIC(model.nobroken))

# Detach the working data frame
detach(streaks)
```

A5. Complete results

Probability of switching (%)			*Attention Modulation Factor*		
			Standard attention restriction	*No attention restriction*	*Linear attention restriction*
Arousal Dissipation Factor	*Constant*	*0,95*	4,2	19,5	0
		0,85	16,7	34,7	0
		0,20	30	34,7	8,4
	Function	*Slow*	3,3	15,4	0
		Intermediate	8,8	25,3	0
		Fast	30	38,1	0,6

Probability of re-evaluating (%)			*Attention Modulation Factor*		
			Standard attention restriction	*No attention restriction*	*Linear attention restriction*
Arousal Dissipation Factor	*Constant*	*0,95*	26,2	29	28,6
		0,85	44,5	49,2	46,3
		0,20	53	54,2	55,3
	Function	*Slow*	20	22,7	21,2
		Intermediate	37	38,5	36,7
		Fast	52,5	54,9	56,1

Switch ratio (switches / re-evaluations)			*Attention Modulation Factor*		
			Standard attention restriction	*No attention restriction*	*Linear attention restriction*
Arousal Dissipation Factor	*Constant*	*0,95*	0,16	0,67	0
		0,85	0,38	0,71	0
		0,20	0,57	0,64	0,15
	Function	*Slow*	0,17	0,68	0
		Intermediate	0,24	0,66	0
		Fast	0,57	0,69	0,1