Raster fluxes

Flow dynamics of overland flow, ground water flow, surface-ground water integration, and associated solute constituents

Overland (surface) flow

Alternatiing Direction Explicit function

/***/

Flux_SWater

/* Surface water fluxing routine
This is the alternating direction function that first fluxes water in the E/W direction and then in the N/S direction. It sweeps from left to right, then goes back from right to left, then goes from top to bottom and finally from bottom to top. This alternates every hyd_iter.

Surface water model boundary exchanges may only occur across cells designated with designated attribute in the boundary condition map file.
*/

/* Note that surface water variable is actually updated in the Flux_SWstuff function */

step_Cell = sq_celWid * sfstep/CELL_SIZE; /* constant used in surface water flux equations */

/* check the donor and recipients cells for a) on-map, b) the cell attribute that allows sfwater boundary flow from the system and c) the attribute that indicates levee presence:
the levee attribute of 1 (bitwise) allows flow to east;
attribute of 2 (bitwise) allows flow to south (levee atts calc'd by WatMgmt.c) */

/* as always, x is row, y is column! */

main north-to-south

west-to-east

call water fluxes

call solute fluxes

east-to-west

call water fluxes

call solute fluxes

main west-to-east

north-to-south

call water fluxes

call solute fluxes

south-to-north

call water fluxes

call solute fluxes

Flow equation function

```
for(north-south spatial loop)
   if (every_other_iteration) {  /* alternate loop directions every other hyd_iter (it) */
      for(west-east spatial loop)  /* loop from west to east */
         if( donor-recipient condition met  )
            FF = Flux_SWcells(arguments);
                                  /* FF units = m */
            if (FF != 0.0) Flux_SWstuff (arguments);
   else {
   for(east-west spatial loop)   /* loop from east to west */
      if( donor-recipient condition met  )
         FF = Flux_SWcells(arguments);
                                  /* FF units = m */
         if (FF != 0.0) Flux_SWstuff (arguments);

for(west-east spatial loop)
   if (every_other_iteration) {  /* alternate loop directions every other hyd_iter (it) */
   for(north-south spatial loop)  /* loop from north to south */
      if( ( donor-recipient condition met  )
            FF = Flux_SWcells(arguments);
                                  /* FF units = m */
         if (FF != 0.0) Flux_SWstuff (arguments);
   else {
   for(south-north spatial loop)  /* loop from south to north */
      if( ( donor-recipient condition met   )
         FF = Flux_SWcells(arguments);
                                  /* FF units = m */
         if (FF != 0.0) Flux_SWstuff (arguments);


/*****************************************************************************************/
Flux_SWcells

/* Surface water flux eqns.
```

no code “un-generator” for this Fluxes.c source file; here we have extracted the principal dynamic functions, removing some C arguments, debug code etc

Application of Manning's eqn to calculate flux between two adjacent cells (i0,i1) and (j0,j1)
Returns height flux. Flux is positive if flow is from i to j.
Checks for available volume, and that flow cannot make the head in recepient cell higher
than in the donor one. */

mean Manning's coef

```
M = (MC[cellLoci] + MC[T(j0,j1)])/2.;
```

```
/* code omitted that determines the head and tail water heads for model domain boundaries */
```

determine head and tail water conditions at domain boundary

```
dh = Hi - Hj;              /* dh is "from --> to" */
adh = ABS (dh);

if (dh > 0)
{
```

calculate flow (F) for positive head diff

```
        if (SWater[cellLoci] < DetentZ) return 0.0;
        F = (M != 0) ?
                (pow(adh,alpha2) * base_flux_rate / M  * pow(SWater[cellLoci],alpha1)*step_Cell) :
                (0.0);
    /* ensure adequate volume avail */
        F =  ( F > ramp(SWater[cellLoci] - DetentZ) ) ? (ramp(SWater[cellLoci] - DetentZ)) : (F);
```

ensure flow constraints are met

```
         /* check to ensure no flip/flops associated with depth */
    if ( ( Hi - F ) < ( Hj + F ) )  F = Min ( dh/2.0, ramp(SWater[cellLoci] - DetentZ) );
}
else
{
```

calculate flow (F) for negative head diff

```
        if (SWater[cellLocj] < DetentZ) return 0.0;
        /* F is negative in this case */
        F = (M != 0) ?
                ( - pow(adh,alpha2) * base_flux_rate / M  * pow(SWater[cellLocj],alpha1)*step_Cell) :
                (0.0);
    /* ensure adequate volume avail */
```

no code "un-generator" for this Fluxes.c source file; here we have extracted the principal dynamic functions, removing some C arguments, debug code etc

ensure flow constraints are met

```
            F =  ( -F > ramp(SWater[cellLocj] - DetentZ) ) ? (-ramp(SWater[cellLocj] - DetentZ)) : (F);
             /* check to ensure no flip/flops associated with depth */
            if ( ( Hi - F ) > ( Hj + F ) )  F = - Min ( adh/2.0, ramp(SWater[cellLocj] - DetentZ) );
        }

        return (F);       /* returns height flux */
```

return calculated flow value

```
/*****************************************************************************************/
```

Flux_SWstuff

Solute flux equation function

```
/* Flux of surface water solutes (nutrients, salinity, etc), and update surface water variable, budget calcs.
   These are units of solute mass being fluxed from i0,i1 to j0,j1
   Flux & SURFACE_WAT are in units of height (m);
   for mass balance checks, the order of STUF1-3 is salt, nitrogen, phosphorus */
```

mass of solutes to flux with surface water

```
 if (Flux >0.0) {
    m1 = (SURFACE_WAT[cel_i]>0.0) ? (STUF1[cel_i]*Flux/SURFACE_WAT[cel_i]) : (0.0);
    m2 = (SURFACE_WAT[cel_i]>0.0) ? (STUF2[cel_i]*Flux/SURFACE_WAT[cel_i]) : (0.0);
    m3 = (SURFACE_WAT[cel_i]>0.0) ? (STUF3[cel_i]*Flux/SURFACE_WAT[cel_i]) : (0.0);
 }
 else  {
    m1 = (SURFACE_WAT[cel_j]>0.0) ? (STUF1[cel_j]*Flux/SURFACE_WAT[cel_j]) : (0.0);
    m2 = (SURFACE_WAT[cel_j]>0.0) ? (STUF2[cel_j]*Flux/SURFACE_WAT[cel_j]) : (0.0);
    m3 = (SURFACE_WAT[cel_j]>0.0) ? (STUF3[cel_j]*Flux/SURFACE_WAT[cel_j]) : (0.0);
 }
```

update mass of solutes in cell

```
 STUF1[cel_j] += m1;  /* add the masses of solutes */
 STUF2[cel_j] += m2;
 STUF3[cel_j] += m3;
 STUF1[cel_i] -= m1;
```

no code “un-generator” for this Fluxes.c source file; here we have extracted the principal dynamic functions, removing some C arguments, debug code etc

update surface water depth in cell

budget calculations omitted (have no effect on model dynamics)

Groundwater flow & surface-ground integration

Alternatiing Direction Explicit function

main north-to-south

west-to-east

call water fluxes

```
 STUF2[cel_i] -= m2;
 STUF3[cel_i] -= m3;
 SURFACE_WAT[cel_j] += Flux; /* now update the surfwater depths */
 SURFACE_WAT[cel_i] -= Flux;

 * warning conditions etc omitted */


 /*****comments not in source code *******/
/* omit lines of code that account for water and solute budgets  (have no effect on model dynamics)*/
 /*****comments not in source code *******/




/*****************************************************************************************/
Flux_GWater

/* Groundwater fluxing routine
   This is the alternating direction function that first fluxes water in the E/W direction and then in the N/S
   direction. It sweeps from left to right, then goes back from right to left, then goes from
   top to bottom and finally from bottom to top. This alternates every hyd_iter.
   Water is fluxed with mass conservation, allowing outflow from (no inflow to) the model system.
   Order of constituents is salt, N, and P*/

 /* ix is row, iy is col! */
/* we only check the donor cell for on-map, allowing losses from the system */
 for(north-south spatial loop)
    if (every_other_iteration) {  /* alternate loop directions every other hyd_iter (it) */
    for(west-east spatial loop)  /* loop from west to east */
          /* allow boundary flow if donor cell is marked (5 or 10) */
          if (donor-recipient condition met )
          Flux_GWcells(arguments);
```

no code “un-generator” for this Fluxes.c source file; here we have extracted the principal dynamic functions, removing some C arguments, debug code etc

east-to-west

call water fluxes

main west-to-east

north-to-south

call water fluxes

south-to-north

call water fluxes

```
    else {
    for(east-west spatial loop)   /* loop from east to west */
          /* allow boundary flow if donor cell is marked (5 or 10) */
       if ( donor-recipient condition met )
          Flux_GWcells(arguments);

 for(west-east spatial loop)
 {
    if (every_other_iteration) {  /* alternate loop directions every other hyd_iter (it) */
    for(north-south spatial loop)  /* loop from north to south */
          /* allow boundary flow if donor cell is marked (5 or 10) */
       if (donor-recipient condition met)
          Flux_GWcells (arguments);
    else {
    for(south-north spatial loop)  /* loop from south to north */
          /* allow boundary flow if donor cell is marked (5 or 10) */
       if (donor-recipient condition met )
          Flux_GWcells(arguments);
```

Flow equation and surface-ground-water integration function

```
/*****************************************************************************************/
Flux_GWcells

/* Ground water flux eqns, incl. integration of groundwater with surface water.
   Application of Darcy's eqn to calculate flux between two adjacent cells (i0,i1) and (j0,j1)
   Flux is positive if flow is from i to j.
   Checks for available volume, and that flow cannot make the head in recepient cell higher
   than in the donor one.
   Integrates the vertical exchange among surface, unsaturated, and saturated water;
   updates all hydro and solute state variables.
   Calcs the budget sums for water and solutes.  */

    if (in model domain ) {
       GW_head_i = SatWat[i] / poros[HAB[i]];
```

determine heads within

no code “un-generator” for this Fluxes.c source file; here we have extracted the principal dynamic functions, removing some C arguments, debug code etc

model domain

omitted code to determine heads, solute conc, etc in boundary cells outside of domain

average hydraulic conductivity

head difference

```
      tot_head_i = GW_head_i + SfWat[i];
   if (in model domain ) {
      GW_head_j = SatWat[j] / poros[HAB[j]];
      tot_head_j = GW_head_j + SfWat[j];

 /*****comments not in source code *******/
/* omit lines of code that develop boundary condition head, solute concentratios, etc */
 /*****comments not in source code *******/

   AvgRate = ( rate[i] + rate[j] )/2.0; /* average hyd conductivity of the two cells */

/* hydraulic head difference, positive flux from i to j */
   dh = tot_head_i - tot_head_j;


      /* determine donor and recipient cells based on head difference,
        and provide the sign of the flux;
        The surface water detention depth also used here as threshold
        head difference for fluxing (currently global, 1 cm)*/
   if (dh > DetentZ)
   {
         cell_don=i;
         cell_rec=j;
      sign = 1.0;
   }
   else if (dh < -DetentZ)
   {
         cell_don=j;
         cell_rec=i;
      sign = -1.0;
   }
   else
      return; /* no flux (or surface-groundwater integration) under minimal head diffs */
```

no code “un-generator” for this Fluxes.c source file; here we have extracted the principal dynamic functions, removing some C arguments, debug code etc

calculate potential horizontal flux

start of do-while loop to integrate horizontal fluxes with changes in storage capacities in entire water column

modify horizontal flux depending on unsaturated capacity

post-flux capacites

```
/* Potential horizontal flux eqn (Darcy's eqn simplified to square cells).
   This is the maximum height (m) of water vol to flux under fully saturated condition.
   (alpha3 is the third "flux_parm" read from the Driver.parm file (a calib parm, normally 1.0)).
   Note that the Min check is not important under current implementations, as SatWat includes water
   down to the base datum, which is currently 6 meters below sea level (actually, 1929 NGVD) */

    Flux = Min(Abs(dh) * alpha3 * AvgRate * SatWat[cell_don] / CELL_SIZE * gwstep , SatWat[cell_don]);

/**** this do-while routine (1) integrates the surface, saturated, and unsaturated water,
  and (2) checks to ensure the heads do not reverse in a time step due to large fluxes.
  If heads do reverse, the total Flux is decremented until there is no reversal */
/****/
    do {
       /* The total potential flux is apportioned to (1) the horizontal component
          that fluxes to an adjacent cell and (2) the vertical component that
          remains in the donor cell after the horizontal outflow from a donor cell.
          Thus, an unsaturated zone is created, or increased in size, with loss of
          saturated water from the donor cell; this lateral gravitational flow leaves behind
          the field capacity moisture in an unsat zone. (If donor-cell surface water is present,
          it potentially will replace the unsaturated soil capacity within the same time
          step in this routine).*/
       fluxTOunsat_don = Flux * (poros[HAB[cell_don]] - sp_yield[HAB[cell_don]])  ;
       fluxHoriz = Flux - fluxTOunsat_don;

/**** given the total potential groundwater Flux, get the donor cell's
     new **post-flux** capacities */
       UnsatZ_don  =   elev[cell_don] - (SatWat[cell_don]-Flux) / poros[HAB[cell_don]] ; /* unsat zone depth */

       UnsatCap_don =  UnsatZ_don * poros[HAB[cell_don]]; /* maximum pore space capacity (UnsatCap)
                                                             in the depth (UnsatZ) of new unsaturated zone */
       UnsatPot_don  = UnsatCap_don - (Unsat[cell_don]+fluxTOunsat_don); /* (height of) the volume of pore space
```

no code “un-generator” for this Fluxes.c source file; here we have extracted the principal dynamic functions, removing some C arguments, debug code etc

surface to saturated /unsaturated fluxes

```
                                                    (soil "removed") that is unoccupied in the unsat zone */
        /* determining the pathway of flow (to sat vs. unsat) of surface water depending on depth
           of an unsat zone relative to the surface water.  With a relatively deep unsat zone, this
           downflow tends to zero (infiltration occurs within the vertical hydrology module of StellaEqns.c) */
      Sat_vs_unsat  = 1/Exp(100.0*Max((SfWat[cell_don]-UnsatZ_don),0.0));

   /* sf-unsat-sat fluxes */
      if (SfWat[cell_don] > 0.0) { /* surface water downflow is assumed to be as fast
                                      as horizontal groundwater outflows */
         sfTOsat_don  =
            ( (1.0-Sat_vs_unsat)*UnsatPot_don>SfWat[cell_don] ) ?
            ( SfWat[cell_don] ) :
            ( (1.0-Sat_vs_unsat)*UnsatPot_don);
        /* with downflow of surface water into an unsat zone, the proportion of that height
           that is made into saturated storage is allocated to the sat storage variable */
         if (sfTOsat_don >=  UnsatPot_don) {
               sfTOsat_don = UnsatPot_don ; /* downflow constrained to the avail soil capacity */
               unsatTOsat_don = Unsat[cell_don]; /* allocate unsat to sat in this case */
            }
         else {
            unsatTOsat_don = (UnsatZ_don > 0.0) ?
               ( (sfTOsat_don/poros[HAB[cell_don]] ) / UnsatZ_don * Unsat[cell_don] ) :
               (0.0); /*  allocate to saturated storage whatever proportion of
                        unsat zone that is now saturated by sfwat downflow */
         }
      }
      else { /* w/o surface water, these vertical fluxes are zero */
         sfTOsat_don = unsatTOsat_don = 0.0;
      }
/***** potential new head in donor cell will be ... */
      H_pot_don = (SatWat[cell_don] - fluxTOunsat_don - fluxHoriz + sfTOsat_don + unsatTOsat_don )
         / poros[HAB[cell_don]] +(SfWat[cell_don] - sfTOsat_don) ;
```

new donor head potential

no code “un-generator” for this Fluxes.c source file; here we have extracted the principal dynamic functions, removing some C arguments, debug code etc

recipient capacities

```
/**** recipient cell's **pre-flux** capacities */
      UnsatZ_rec  =   elev[cell_rec] - SatWat[cell_rec] / poros[HAB[cell_rec]] ; /* unsat zone depth */
      UnsatCap_rec =  UnsatZ_rec * poros[HAB[cell_rec]]; /* maximum pore space capacity (UnsatCap)
                                                  in the depth (UnsatZ) of new unsaturated zone */
      UnsatPot_rec  = UnsatCap_rec - Unsat[cell_rec]; /* (height of) the volume of pore space (soil "removed")
                                                  that is unoccupied in the unsat zone */
    /* sf-unsat-sat fluxes */
      horizTOsat_rec = fluxHoriz; /* lateral inflow to soil into saturated storage */
      satTOsf_rec = Max(fluxHoriz - UnsatPot_rec, 0.0); /* upwelling beyond soil capacity */
      unsatTOsat_rec = (UnsatZ_rec > 0.0) ? /* incorporation of unsat moisture into sat storage with
                                    rising water table due to horiz inflow */
        ( ((horizTOsat_rec-satTOsf_rec)/poros[HAB[cell_rec]] ) / UnsatZ_rec * Unsat[cell_rec] ) :
        (0.0);
```

new recipient head potential

```
/**** potential new head in recipient cell will be ... */
      H_pot_rec = (SatWat[cell_rec] + horizTOsat_rec + unsatTOsat_rec - satTOsf_rec)
        / poros[HAB[cell_rec]] + (SfWat[cell_rec] + satTOsf_rec) ;
```

ensure flow constraints are met

```
/**** check for a head reversal - if so, reduce flux to achieve equilibrium (donor >= recip) */
      if ( (Flux != 0.0) && ((H_pot_rec - H_pot_don) > MinCheck) ) {
        revers++;
        Flux *= 0.90;
        equilib = 0;
          /* if the unsat water storage causes an unfixable (very rare(/never?)) head reversal due to the
             sfwat and gwater integration alone, set the Flux to zero, then allow the vertical
             sf/ground integration to be done with no horiz flux and be done with it */
        if (revers>4) {
          Flux =  0.0;
          }
      }
      else {
        equilib = 1;
      }
```

no code “un-generator” for this Fluxes.c source file; here we have extracted the principal dynamic functions, removing some C arguments, debug code etc

bottom of do-while loop for ground-surface water integration

determine solute flux associated with horizontal water fluxes

determine solute flux associated with vertical water fluxes

```
    } while  (!equilib); /* end of routine for integrating groundwater-surface water and preventing head reversals */

/**********
  finished calc'ing the water fluxes
  **********/
/* calc the flux of the solutes between sed/soil across cells in horiz dir,
 but don't update the state vars until the conc's for vertical flows have been calc'd */
    sedwat_don = SatWat[cell_don] + Unsat[cell_don];
    sed_stuf1fl_horz = (sedwat_don > 0.0) ? (gwSTUF1[cell_don] / sedwat_don*fluxHoriz) : (0.0);
    sed_stuf2fl_horz = (sedwat_don > 0.0) ? (gwSTUF2[cell_don] / sedwat_don*fluxHoriz) : (0.0);
    sed_stuf3fl_horz = (sedwat_don > 0.0) ? (gwSTUF3[cell_don] / sedwat_don*fluxHoriz) : (0.0);

/* pass along the solutes between sed/soil and surface water in vertical dir;
   for "simplicity", this does not account for the phosphorus active-zone conc. gradient,
   but assumes the homogeneity of conc within the entire soil column */
    if (sfTOsat_don > 0.0) {
        sf_stuf1fl_don = (SfWat[cell_don] > 0.0) ? (swSTUF1[cell_don] / SfWat[cell_don]*sfTOsat_don) : (0.0);
        sf_stuf2fl_don = (SfWat[cell_don] > 0.0) ? (swSTUF2[cell_don] / SfWat[cell_don]*sfTOsat_don) : (0.0);
        sf_stuf3fl_don = (SfWat[cell_don] > 0.0) ? (swSTUF3[cell_don] / SfWat[cell_don]*sfTOsat_don) : (0.0);
        gwSTUF1[cell_don] += sf_stuf1fl_don;
        gwSTUF2[cell_don] += sf_stuf2fl_don;
        gwSTUF3[cell_don] += sf_stuf3fl_don;
        swSTUF1[cell_don] -= sf_stuf1fl_don;
        swSTUF2[cell_don] -= sf_stuf2fl_don;
        swSTUF3[cell_don] -= sf_stuf3fl_don;

    }
    if (satTOsf_rec > 0.0) {
        sedwat_rec = SatWat[cell_rec] + Unsat[cell_rec];
        sed_stuf1fl_rec = (sedwat_rec > 0.0) ? (gwSTUF1[cell_rec] / sedwat_rec*satTOsf_rec) : (0.0);
        sed_stuf2fl_rec = (sedwat_rec > 0.0) ? (gwSTUF2[cell_rec] / sedwat_rec*satTOsf_rec) : (0.0);
        sed_stuf3fl_rec = (sedwat_rec > 0.0) ? (gwSTUF3[cell_rec] / sedwat_rec*satTOsf_rec) : (0.0);
```

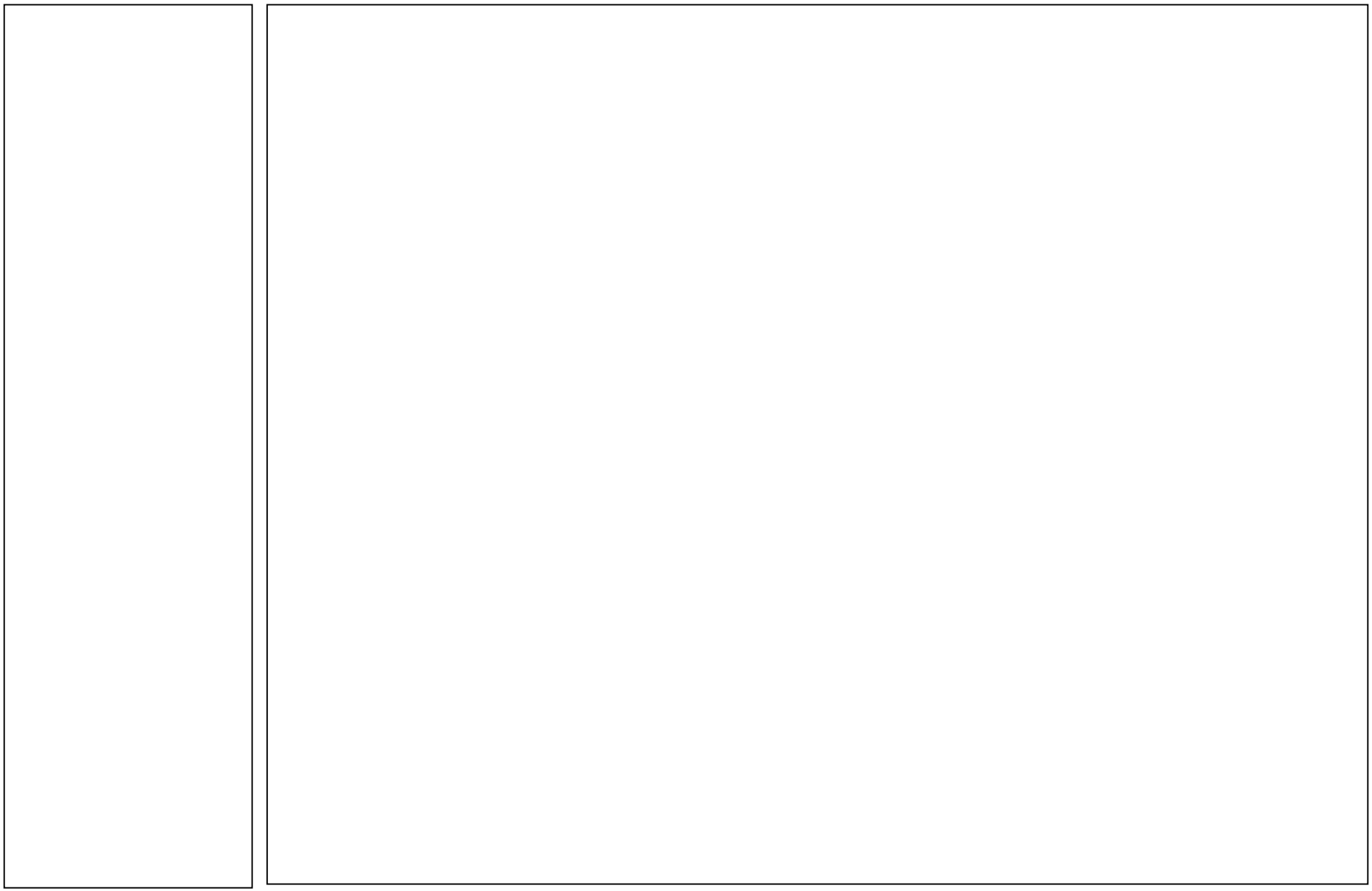
no code “un-generator” for this Fluxes.c source file; here we have extracted the principal dynamic functions, removing some C arguments, debug code etc

	gwSTUF1[cell_rec] -= sed_stuf1fl_rec;
	gwSTUF2[cell_rec] -= sed_stuf2fl_rec;
	gwSTUF3[cell_rec] -= sed_stuf3fl_rec;
	swSTUF1[cell_rec] += sed_stuf1fl_rec;
	swSTUF2[cell_rec] += sed_stuf2fl_rec;
	swSTUF3[cell_rec] += sed_stuf3fl_rec;
	}

update groundwater solutes	/* update the groundwater solutes due to horiz flows */
	gwSTUF1[cell_don] -= sed_stuf1fl_horz;
	gwSTUF2[cell_don] -= sed_stuf2fl_horz;
	gwSTUF3[cell_don] -= sed_stuf3fl_horz;
	gwSTUF1[cell_rec] += sed_stuf1fl_horz;
	gwSTUF2[cell_rec] += sed_stuf2fl_horz;
	gwSTUF3[cell_rec] += sed_stuf3fl_horz;

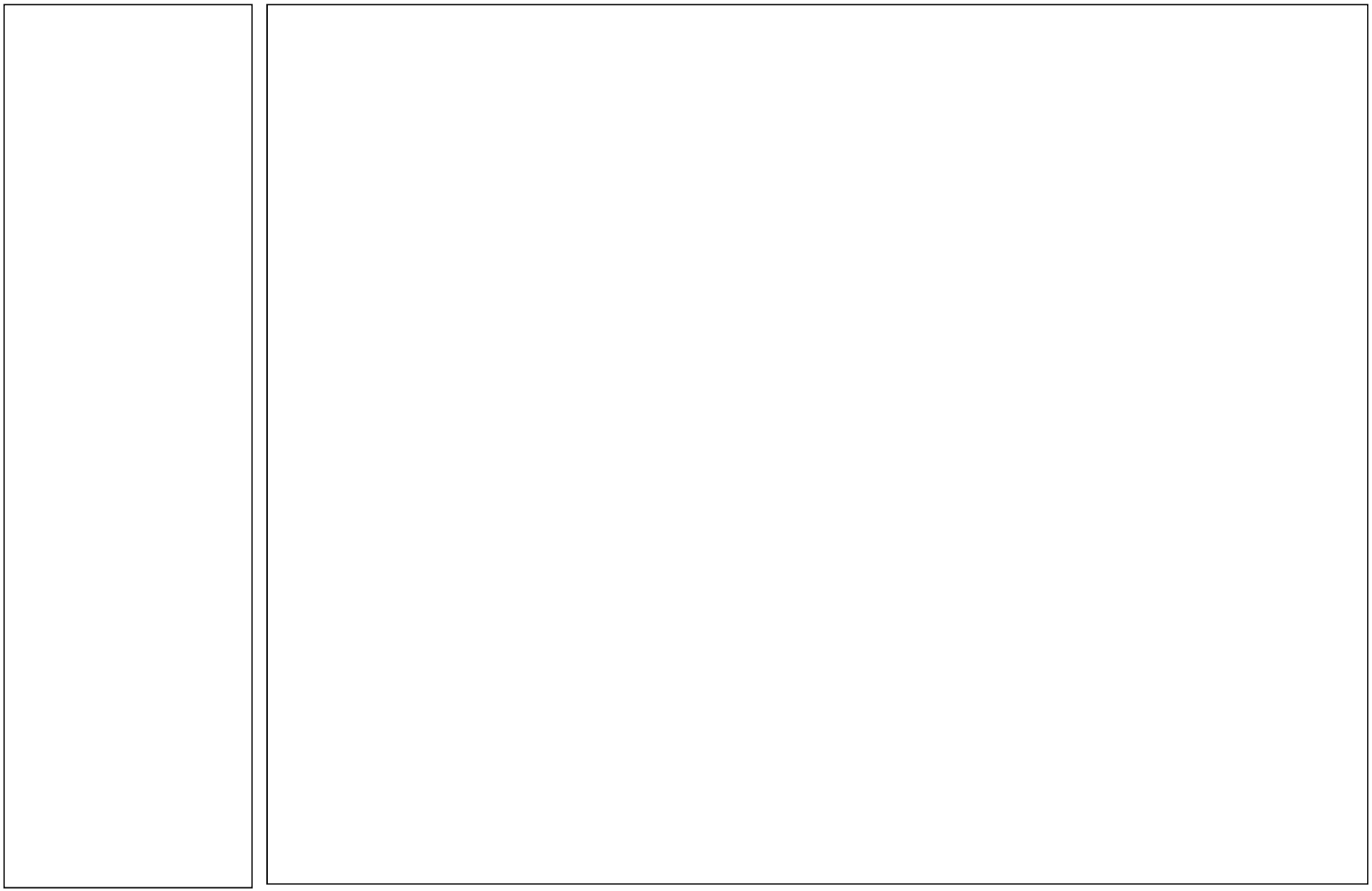
update water storages	/* update the hydro state variables */ SfWat[cell_don] += (-sfTOsat_don); Unsat[cell_don] += (fluxTOunsat_don - unsatTOsat_don) ; SatWat[cell_don] += (sfTOsat_don + unsatTOsat_don - fluxTOunsat_don - fluxHoriz); SfWat[cell_rec] += (satTOsf_rec); Unsat[cell_rec] += (-unsatTOsat_rec); SatWat[cell_rec] += (horizTOsat_rec + unsatTOsat_rec - satTOsf_rec); /* (horizTOsat_rec + satTOsf_rec) = fluxHoriz */
budget calculations omitted (have no effect on model dynamics)	/*****comments not in source code *******/ /* omit lines of code that account for water and solute budgets (have no effect on model dynamics)*/ /*****comments not in source code *******/

no code “un-generator” for this Fluxes.c source file; here we have extracted the principal dynamic functions, removing some C arguments, debug code etc



no code “un-generator” for this Fluxes.c source file; here we have extracted the principal dynamic functions, removing some C arguments, debug code etc

no code “un-generator” for this Fluxes.c source file; here we have extracted the principal dynamic functions, removing some C arguments, debug code etc



no code “un-generator” for this Fluxes.c source file; here we have extracted the principal dynamic functions, removing some C arguments, debug code etc

no code “un-generator” for this Fluxes.c source file; here we have extracted the principal dynamic functions, removing some C arguments, debug code etc