Parallelizing the Distributed Hydrologic Model MOBIDIC

Term Project

Aldrich Castillo
Ph.D. candidate in Environmental Engineering

Massachusetts Institute of Technology
May 2010
1. INTRODUCTION

Hydrologic models are employed in many applications such as regional flood forecasting, soil moisture analysis for agricultural applications, water resources management, and sustainability studies. As opposed to conventional lumped models, distributed models provide a physics-based representation of the hydrologic processes occurring in a basin by taking into account the spatial variability of the natural properties of the basin, and the spatial, in addition to the temporal, variability of meteorological forcings. By accounting the geospatial variability of the basin, physically based models provide better representations of hillslope and channel dynamics, which in turn results to improve modeling of state variables such as soil moisture or the flow at any point in a river network. Dramatic advances in Geographic Information System (GIS), weather forecasting, remote sensing, and of course, computer software and hardware, now allows highly complex yet robust hydrologic models. Unfortunately, while there have been considerable efforts directed into the research and development of distributed models, such as the ongoing Distributed Model Intercomparison Project spearheaded by the National Weather Service (NWS), the abovementioned advances are yet to be substantially utilized by current models.

I am a 2nd year Ph.D. student doing research on Integrated Modeling of Surface and Subsurface Hydrology. The main model I am using, and helping to develop, is the Modello Bilancio Idrologico DIstributo e Continuo (MOBIDIC), a fully-distributed raster-based hydrologic model. MOBIDIC is written in MATLAB with more than a hundred subroutines and a total of about a few thousand lines of codes. While MOBIDIC is fully functional, and have been successfully applied in several basins including the
current use in the experimental system of flood forecasting of the Arno River Basin Authority, Italy (Campo et al., 2006), it is still under development. One of the impediments of MOBIDIC is its long runtime. Since MOBIDIC is a fully-distributed model, it deals with large matrices and thus, is very computationally intensive. For example, the auto-calibration of a 30km by 30km watershed using 30m spatial discretization (1000 by 1000 grids), for 3 months period with hourly time step, with 5 parameters to be calibrated, and using a 3.0-GHz 4GB RAM desktop, takes about 3 days.

Prior to taking this class, I have no previous experience with high performance computing in general, and of parallel computing in particular. However, I believed that these can significantly speedup MOBIDIC. In addition to possible speedup, parallel computing can also increase the maximum allowable size of the problem i.e. minimize RAM requirement.

For the term paper, I focused on the main subroutine of MOBIDIC, called `mobidic_sid.m`, and its child functions, which perform the hydrologic computations and accounts for at least 90 percent of the total model runtime. I have three main goals for this class project. The first is to apply the lessons I learned in the class about general computing to improve the performance of mobidic_sid.m (i.e. speedup before parallelizing the subroutine). The second goal is to extract the child functions performing the mass and energy balances and assess the speedup through parallel computing. The last, but probably the most important goal is to have a better understanding of MOBIDIC from a computational perspective, learn what can and should be done to improve the performance, and take away lessons learned for future research efforts.
2. DISTRIBUTED HYDROLOGIC MODELING AND MOBIDIC

The hydrologic cycle is a complex system of multiple interdependent processes. As shown in Figure 1, this cycle involves the continuous movement of mass and energy that have natural spatial and temporal variability.

Figure 1. The hydrologic cycle (Vivoni, 2002)

Figure 2. A conceptual diagram of MOBIDIC (Castelli, 2008)
MOBIDIC is a raster-based hydrologic balance model that represents the hydrologic cycle by simulating a system of reservoirs and the mass and energy fluxes between them, Figure 2. The model is based on a distributed mass balance of water and energy at the surface and near surface ground (top soil), and a lumped (catchment-scale) mass balance of water at the deeper ground. The different roles of gravity and capillary forces in moving water through the soil are adequately modeled by employing a conceptual compartmentalization of each computational unit i.e. grid block of soil between a gravitational compartment (large pore) and a capillary compartment (small pore). This compartmentalization is based on an arbitrarily threshold pore size of 60 $\mu$m (Campo et al., 2006). The volumes of water per unit area in the gravitational and capillary components, $W_g$ and $W_c$, should not exceed the maximum capacity values $W_{g\,\max}$ and $W_{c\,\max}$, and are governed by the following mass balance equations:

$$\frac{dW_g}{dt} = I_{nf} - S_{per} - Q_d - S_{as}$$

$$\frac{dW_c}{dt} = S_{as} - E_T$$

where, $I_{nf}$ is infiltration rate, $S_{per}$ is percolation, $Q_d$ is hypodermic flow, $S_{as}$ is adsorption from the gravitational to the capillary soil reservoir and $E_T$ is evapotranspiration. The adsorption $S_{as}$ is assumed to be a linear irreversible process, and proportional to a ‘bulk suction head’ through an absorption parameter $\kappa$, as shown in Equation 3. Evapotranspiration $E_T$ comes from water in the capillary reservoir only (i.e. root uptake overwhelms other evapotranspiration processes) and is equal to the potential value $E_{T\,pot}$ until the soil dries out, see Equation 4.
\[ S_{ax} = \kappa \left( 1 - \frac{W_c}{W_{c\,max}} \right) \] (3)

\[ E_T = \begin{cases} E_{T\,pot} & \text{if } W_c > 0 \\ 0 & \text{if } W_c = 0 \end{cases} \] (4)

Following the conceptualization explained above, the storage capacities \(W_{g\,max}\) and \(W_{c\,max}\) can then be defined as the maximum water content above the field capacity, and the maximum water content between the field capacity and the wilting point, respectively (Campo et al., 2006). Water in the gravitational reservoir is transported toward the groundwater as percolation \(S_{per}\), or toward downhill computational cells as hypodermic flow \(Q_d\). Both these fluxes behave linearly with the gravitational water content as shown in Equations 5 and 6, where \(\gamma\) is the percolation parameter and \(\beta\Delta x\) is the hypodermic flow velocity (with \(\Delta x\) as the horizontal spacing of the cells).

\[ S_{per} = \gamma W_g \] (5)

\[ Q_d = \beta W_g \] (6)

Infiltration, \(I_{nf}\), is modeled in MOBIDIC by Equation 7. \(P\) is the precipitation rate on the cell and \((Q_d, R_H, R_{H\,up})\) are the hypodermic flow, Horton runoff and Dunne runoff contributions from upstream cells, respectively. Notice that the nonlinearities of infiltration are accounted by limiting both its rate through the saturation hydraulic conductivity \(K_s\) and its accumulation through the soil gravitational capacity \(W_{g\,max}\). To account for rainfall variability, \([P + ((Q_d + R_H + R_{H\,up})_{up})]\) is treated as an exponentially-distributed random variable (Campo et al., 2006). Furthermore, the Horton \(R_H\) and Dunne \(R_D\) runoff rates are expressed by Equations 8 and 9.
\[ I_{nf} = \begin{cases} 
(P + ((Q_d + R_H + R_D)_{up}) \{1 - \exp \left[ \frac{-K_s}{P + ((Q_d + R_H + R_D)_{up})} \right] \} & \text{if } W_g < W_{g \text{ max}} \\
0 & \text{if } W_g = W_{g \text{ max}} 
\end{cases} \] (7)

\[ R_H = \begin{cases} 
(P + ((Q_d + R_H + R_D)_{up}) \exp \left[ \frac{-K_s}{P + ((Q_d + R_H + R_D)_{up})} \right] & \text{if } W_g < W_{g \text{ max}} \\
0 & \text{if } W_g = W_{g \text{ max}} 
\end{cases} \] (8)

\[ R_D = \begin{cases} 
0 & \text{if } W_g < W_{g \text{ max}} \\
(P + ((Q_d + R_H + R_D)_{up}) - (\beta + \gamma)W_{g \text{ max}} & \text{if } W_g = W_{g \text{ max}} 
\end{cases} \] (9)

The upstream hypodermic flow entering a cell is the sum of hypodermic flows from all upstream cells. Downhill routing of surface runoff is done through the use of a linear filter representing the effect of water storage in the hillslope and in streams \( W_s \), Equation 10. Here, \( \phi_{ch} \) is the fraction of runoff that is available for infiltration in downhill cells, while the remaining part \( (1 - \phi_{ch}) \) contributes to the channel flow.

\[ \frac{dW_s}{dt} = -\alpha_sW_s + R_H + R_D + \phi_{ch}(\alpha_sW_s)_{up} \] \hspace{1cm} (10)

Parameters that control such schemes are still defined at the cell scale with empirical relations such as,

\[ \alpha_s = \frac{\alpha_0}{\Delta x} \sqrt{A_i} \] \hspace{1cm} (11)

where \( A \) is the contributing area to each cell and \( i \) is topographic slope.

MOBIDIC outputs include discharge flows in each pixel that is on the stream network, the soil moisture content in terms of saturation percentage of each computational cell (capillary volume, gravitational volume, groundwater for each grid) and potential and actual evapotranspiration.
3. METHODOLOGY

My attempt to parallelize MOBIDIC follows the schematic diagram shown in Figure 3 which is a modification of a recommended procedure by ISI (undated). Before actually writing the codes, a conceptual (i.e. schematic) diagram of the model should be developed. Fortunately, MOBIDIC is an existing model and I did not have to code from scratch. Unfortunately however, since many subroutines have been added and changes made to the original MOBIDIC which was developed about a decade ago, the current version is not very organized and its performance is not optimal. Thus, I spent considerable time in the sub-loop of profiling and revising the code, as indicated by the feedback arrow on Figure 3.

![Schematic diagram of parallelizing MOBIDIC](image)

Figure 3. Schematic diagram of parallelizing MOBIDIC (based partly on ISI, undated).

Since MOBIDIC is a large and complex model, I focused on the main subroutine mobidic_sid.m and its 62 child functions. In order to test the extracted subroutine mobidic_sid.m, I ran simulations for the Blue River Basin located in the Southern part of
Oklahoma, see Figure 4. The data and parameters for this study basin have all been prepared before as part of my Ph.D. research. The Blue River Basin encompasses about 1200 square kilometer. For profiling purposes, I used a 120 m grid size (although the highest resolution of data is 30 m), and simulated 1 month with hourly time step.

![Figure 4. The location of the Blue River Basin in Oklahoma and its digital elevation model together with weather stations (black) and USGS stream-flow gauges (red).](image)

Profiling of the code was done using **MATLAB Profiler™** which automatically accounts the time used by each line of a code, each sub-routine and sub-sub-routine called, and each line in those (sub-) sub-routines, as shown by Figure 5. It also counts the number of times a function or line is executed. The results of the Profiler were then used to improve the performance of mobidic_sid.m. The speedup obtained even before parallelizing was accomplished by tediously going through each line of the codes and applying various techniques such as i) vectorization; ii) elimination of loops and conditional statements; and iii) revising, rearranging; and simplifying the algorithm. The
new code is again profiled and this loop is repeated until the performance is acceptable. Moreover, the Profiler also provided insight as to which sections/subroutines can be speedup by using a parallel computer.

![Profiler Report](image)

Figure 5. Matlab Profiler™ Report for the original version of mobidic_sid.m in a simulation of the Blue River Basin
The subroutine mobidic_sid.m is now ready to be implemented in a parallel computer. However, since this subroutine still has 62 sub-sub-routines, I reduce the goal to parallelize the actual water and energy balance subroutine (I named it mobidic_sid_starp.m) which in total includes 6 functions shown in Figure 6.

Figure 6. The subroutine for mass and energy balance to be implemented in parallel

The parallel version is implemented using Star-P 2.6.0. The parallelism can be achieved by either data-parallel method using ‘*p’ or task-parallel using ‘ppeval’. I realized that for calibration, task-parallel is probably more appropriate. Task-parallel might also have some advantage when a basin is divided in many independent sub-basins and an algorithm for allocating these sub—basins to different processors optimally has been developed. However, for real-time simulation (not calibration), it is easier to simply use data-parallel using ‘*p’. To test the speedup using parallel computing, a synthetic basin is used. I use a square domain with the number of grid blocks on one dimension varying from 24 to 3000.

For the Star-P parallel computations, I used the cluster on starp.csail.mit.edu. The cluster is an SGI Altix with 6 nodes, each with two 1.3 GHz Itanium 2’s and 2GB of RAM, for a total of 12 computational processors and 12GB of RAM. For the Star-P client-side, I used my personal laptop which is an IBM ThinkPad SL410 64-bit with two
2.1GHz Genuine Intel® Core™ Duo T6570 processors (although I only utilized 1 of the cores because I don’t have Parallel MATLAB), with 4GB of RAM (3.84 usable). The operating system of my laptop is Microsoft Windows 7.

4. RESULTS AND ANALYSIS

a. Before Parallelizing

Table 1 shows the runtimes of mobidic_sid.m and its main child functions, which represent almost 100 percent of the entire runtime. The original version of mobidic_sid.m took 1354 seconds (22.6 minutes) to complete a 30-day simulation of the Blue River Basin using a spatial and temporal discretization of 120m, and 1 hour, respectively. The total speedup obtained by the final version is 79 percent. The remaining computational time (21%) is mainly used by saving and writing data to hard-disk.

Table 1. Comparison of runtimes of the original and revised versions of mobidic_sid.m and its child functions.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mobidic_sid</td>
<td>744</td>
<td>1354</td>
<td>1026</td>
<td>814</td>
<td>21%</td>
<td>314.8</td>
<td>212.4</td>
<td>79%</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>go_route_ord</td>
<td>744</td>
<td>323.9</td>
<td>322.4</td>
<td>57</td>
<td>83%</td>
<td>9.3</td>
<td>8.3</td>
<td>97%</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>forcing_map</td>
<td>4464</td>
<td>326.6</td>
<td>320.9</td>
<td>91.9</td>
<td>72%</td>
<td>91.9</td>
<td>90</td>
<td>72%</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>glob_route</td>
<td>744</td>
<td>544.2</td>
<td>195.9</td>
<td>139</td>
<td>58%</td>
<td>33.7</td>
<td>24.3</td>
<td>88%</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>energy_balance</td>
<td>806</td>
<td>34.4</td>
<td>20.4</td>
<td>16.9</td>
<td>33%</td>
<td>11.8</td>
<td>8.9</td>
<td>56%</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>mass_balance</td>
<td>744</td>
<td>11.8</td>
<td>11.8</td>
<td>10.5</td>
<td>11%</td>
<td>10.5</td>
<td>10.5</td>
<td>11%</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>calc_forcing_day</td>
<td>744</td>
<td>176.8</td>
<td>2</td>
<td>68.2</td>
<td>25%</td>
<td>68.2</td>
<td>1.5</td>
<td>25%</td>
<td></td>
</tr>
</tbody>
</table>
Figure 7. Comparison of runtimes of the original and revised versions of mobidic_sid.m and its child functions.

b. Speedup by Parallelizing

The Star-P version of the mass and energy balance subroutine of mobidic_sid.m (named mobidic_sid_starp.m) and its child functions are attached in the Appendix section. Figure 8 shows the speedup obtained by parallelizing this subroutine for different domain size and varying number of processors. For small domains (about up to a grid of 240 by 240), the speedup is very minimal. This is because of the overhead cost of the parallel implementation i.e. rearranging/redistributing the data, latency, and communication time between processors. As the domain size becomes larger, the overhead time becomes less significant and most of the total model runtime is used by actual mathematical operations. Theoretically, using 2 processors has a maximum speedup of 50%, while the actual speedup obtained for large basins is about 40%. Using 12 processors has a maximum speedup of 91.6% while the actual obtained
speedup obtained for large basins is about 85%. Since some sections of the subroutine remained as serial operations, the actual speedups are actually very good.

![Graph showing speedup for different domain sizes and number of processors.](image)

Figure 8. Speedup obtained by parallelizing the mass and energy balance subroutine of mobidic_sid.m (mobidic_sid_starp.m) for different domain size and varying number of processors

5. SUMMARY AND RECOMMENDATIONS

Distributed hydrologic models provide a physically based representation of the hydrologic processes occurring in a basin by considering the spatial variability of the basin, and the spatial and temporal variability of meteorological forcings. The paper gave an overview of Distributed Hydrologic Modeling in general, and of MOBIDIC, a fully-distributed raster-based hydrologic balance model, in particular. The first part of the project was aimed at speeding up the subroutine without parallel computing. This was done iteratively by profiling the code using Matlab Profiler and revising the code. The speedup obtained for a simulation of the Blue River Basin in Oklahoma is about 80%. Next, the mass and energy balance subroutine was extracted and made parallel using Star-P and ran on a 12 processor Altix machine on starp.csail.mit.edu. The speedup was
measured for different domain size and different number of processors. The speedup obtained by parallelizing is minimal for small domains but increases as the domain size becomes larger. Considering that there is some overhead cost of parallelism and some sections of the codes have to stay as serial operations, the actual speedups obtained for the large domains are very close to the maximum theoretical speedups.

Initial efforts were made to make some parts of MOBIDIC data-parallel. Future works include developing a task-parallel version esp. for calibration purposes. As this research is ongoing, the author believes that the insights and lessons learned in the accomplishment of this term project and the other concepts and methods learned in Course 18.337 will be very useful in the accomplishments of the overall goals of this research.

7. REFERENCES


Castelli, F. (2008), Integrated water resources management and flood control with hydrologic modeling: from distributed parameters to distributed results, Slide Presentation, NUS, Singapore

Interactive Supercomputing, Inc. (ISI), undated, Star-P’s Power Tools for Parallelism

Vivoni, E. (2002), TIN and similarity in landscape processes, Conference Proceeding, AGU Meeting, Fall of 2002
7. APPENDICES

**Function mobidic_sid_starp()**

%MIT Course 18.337
%Written by Aldrich Castillo
%MOBIDIC modified subroutine for 3D water and energy balance for a
%hypothetical square basin.

pprofile on;

%SPATIAL DISCRETIZATION
nx=3000; %number of computational grid blocks along x
ny=nx; %number of computational grid blocks along y
x=1:1:nx;
y=1:1:ny;
[xx,yy]=meshgrid(x*p,y*p);
z=[0 .05 .20 0.6]'; %depth at which soil moistures are measured [m]
l=length(z)-1; %number of soil layers
dz=z(2:end)-z(1:l);
Dz=z(end)-z(1); %total soil depth [m]

%Multipliers
Dzeros=zeros(ny,nx*p);
Deye=ones(ny,nx*p);
D2=1-0.1*rand(ny,nx*p);

%SITE PROPERTIES
blat=34.31; %latitude [deg]
blon=-96.55; %longitude [deg]
alb=0.15*D2; %albedo
ev0=(1e-5)/24; %potential ET [m/hr]
CH=1.e-6; %turbulent heat coeff
nis=1e-6;
Tcost=290; %temperature of deep aquifer
ff=pi/12/3600;
pr=1000;

%SOIL PROPERTIES: sandy loam with 3 layers
ks0=1*4.42e-2/3600*D2; %hydraulic conductivity [m/s]
ks=ks0; kmax=ks; ksmin=Dzeros;
gam=0.6-5*D2; %percolation
kap=1.2e-7*D2; %adsorption
bet=1.0e-6*D2; %hypodermic flow
alpsur=5.0e-6*D2; %surface flow
cha=0;
f0=0;

theta_i=[1 0 0.05 0.093 0.091]';
theta=dz'*theta_i(4:end)/Dz;
dt=1; %[hr]

theta_sat=0.42; %soil moisture at saturation
theta_fld=0.27; %field capacity;
theta_r=0.09; %residual

Wc0=Dz*(theta_fld-theta_r)*Deye; %MOBIDIC gravitational reservoir capacity [m]
Wg0=Dz*(theta_sat-theta_fld)*Deye; %MOBIDIC capillary reservoir capacity [m]
Wg=Dzeros;
Wmax=0.01; %Max ponded height [m]
Wc=Dz*(theta-theta_r)*Deye;

k=find(Wc>Wc0);
if ~isempty(k)
    Wc(k)=Wc(k)-Wc0(k);
    Wc(k)=Wc(k)-Wg(k);
end
Ws=(Wg*Wg0).*(Wg-Wg0); %discard excess water for initial state
Wg=Wg-Ws;
Ws(Ws>Wsmax)=Wsmax; %Initial ponded height [m]

%WEATHER FILE
W=load('C:\RESEARCH\SHAW\Tests_different_climates\blue\dura.WEA');
%Columns 1)day; 2)hour; 3)year; 4)hourly air temperature (°C); 5) wind speed
%(mph); 6) relative humidity (%); 7) precip (in); 8) density of newly fallen
%snow (specified optionally), (g/cm3); 9) incoming solar radiation (W/m2)
W(W==99999)=NaN;
Tair=273.15+W(:,4); %air temp [°C]
U=0.447*W(:,5); %wind speed (originally in mph) [m/s]
qair=W(:,6)/100; %humidity of air
qair(qair>1)=1;
Rs=3600*W(:,9); %incoming solar radiation [W/m2]
Pprev=[0 0; W(:,2) W(:,7)]; Pprev(end,:)=[];
Pprev(Pprev(:,1)==23,2)=0;
P=(W(:,7)-Pprev(:,2))*(2.54e-2/dt*3600); %precipitation [m/s]
pneg=find(P<0);
if ~isempty(pneg)
P(pneg)=W(pneg,7)*2.54e-2/(dt*3600);
end
W(:,3:end)=[];
clear Pprev;

%SOIL TEMPERATURE
TS=load('C:\RESEARCH\SHAW\Tests_different_climates\blue\dura.TMP');
TSt=TS(:,1)+TS(:,2)./24;
TS(TS==99999)=NaN;
TS=273.15+TS;
aa_t=0; %amplitude of temperature = Tmax-Tmin

%INITIALIZATION
pir=Dzeros; %initial upstream runoff [m/s]
pid=Dzeros; %initial upstream hypodermic flow [m/s]

%MAIN LOOP
for t=1:3
    tic
    %Temporal interpolation of soil temperature
    time=W(t,1)+W(t,2)/24;
tl=(find(TSt>time,1,'first'));
tb=tl-1;
Td0_temp=TS(tb)+(TS(tl)-TS(tb))*(time-TSt(tb))/(TSt(tl)-TSt(tb));
Td0=Td0_temp*Deye;
tim_hday=24*3600*mod(time,1);
[hrise,hset]=solarhours(blat,blon,floor(time));
hrise=hrise*3600; hset=hset*3600;
pp_t=pi/2-pi/6+ff.*ctim_hday;
pp_r=pi/2+ff.*ctim_hday;
t_end=hrise-cttim_hday;
step=hrise-cttim_hday;

    C_T=Tair(t)*Deye;
    Tm=C_T;
    Ut=U(t)*Deye;
    qair_t=qair(t)*Deye;
    Pt=P(t)*Deye;

    [Ts,Td,etp]=energybalance(ff,aa_t,0,pp_t,pp_r,C_T,0,Td0,Tm,Ut,pr,...
        qair_t,t_end,step,CH,alb,ks,nis,Tcost,1);
    evp=((~isnan(etp)).*etp+(isnan(etp)).*evr0)./(dt*3600);
\[ W_g, W_c, W_s, \text{pir}, \text{pid}, \text{evr}, \text{pe} = \text{massbalance}(W_g, W_c, W_s, P_t, \text{pir}, \text{pid}, \text{evr}, ...) \]

\[ k_s, k_{\text{max}}, k_{\text{min}}, W_{g0}, W_{c0}, \text{cha}, \text{gam}, \text{kap}, \text{bet}, \text{alpsur}, f_0, 3600 \times dt; \]

\[ \text{thetatot} = D_z \times \theta_r + W_g + W_c; \]
\[ \text{SMavg} = \text{thetatot} \times /D_z; \]
\[ \text{SM}_{W_g} = W_g \times /D_z; \]
\[ \text{SM}_{W_c} = W_c \times /D_z; \]
\[ \text{toc} \]
\[ \text{end} \]

ppprofile off
ppprofile report

---

**function** \[ W_{gu}, W_{cu}, W_{su}, \text{flr}, \text{fld}, \text{evr}, \text{pe} = \text{massbalance}(W_g, W_c, W_s, \text{pip}, \text{pir}, ...) \]

\% 
Integrate the watershed soil mass-balance

Balance of gravitational storage
\[ dW_g /dt = \text{iflt} - \text{pe} - \text{as} - \text{dipo} - \text{su} \]
\[ \text{iflt} = \text{ppr} - \text{ec} + \text{pid} \text{ (infiltration)} \]
\[ \text{ec} = \text{ppr} \times \exp(- (1 - f_0) k_s / \text{ppr}) \text{ (Hortonian runoff)} \]
\[ \text{ppr} = \text{pip} + (1 - \text{cha}) \times \text{pir} \]
\[ \text{pe} = \text{gam} \times W_g \text{ (percolation)} \]
\[ \text{as} = \text{kap} \times (1 - W_c / W_{c0}) \text{ (adsorption)} \]
\[ \text{dipo} = \text{bet} \times W_g \text{ (hypodermic flow)} \]
\[ \text{su} = \text{(Dunne runoff, from limiting } W_g \text{ to } W_{g0}) \]

Balance of capillary storage
\[ dW_c /dt = \text{as} - \text{evr} \]
\[ \text{evr} = \min(\text{evp}, W_c / dt) \text{ (evapotranspiration)} \]

Balance of surface storage
\[ dW_s /dt = \text{cha} \times \text{pir} + \text{ec} + \text{su} - \text{alpsur} \times W_s \]

Input:
\[ W_g = \text{matrix of initial value of } W_g \text{ [m]} \]
\[ W_c = \text{matrix of initial value of } W_c \text{ [m]} \]
\[ W_s = \text{matrix of initial value of } W_s \text{ [m]} \]
\[ \text{pip} = \text{matrix of forcing local precipitation} \]
\[ \text{pir} = \text{matrix of forcing upstream runoff} \]
\[ \text{pid} = \text{matrix of upstream hypodermic flow [m/s]} \]
\[ \text{evp} = \text{matrix of potential evapotranspiration [m/s]} \]
\[ k_s = \text{matrix of saturated infiltration velocity [m/s]} \]
\[ k_{\text{min}} = \]
\[ k_{\text{max}} = \]
\[ W_{g0} = \text{matrix of maximum capacity in } W_g \text{ [m]} \]
\[ W_{c0} = \text{matrix of maximum capacity in } W_c \text{ [m]} \]
\[ \text{cha} = \text{matrix of fraction of channelized runoff [-]} \]
\[ \text{gam} = \text{parameter for percolation [1/s]} \]
\[ \text{kap} = \text{parameter for adsorption [1/s]} \]
\[ \text{bet} = \text{parameter for hypodermic flow [1/s]} \]
\[ \text{alpsur} = \text{parameter for surface flow [1/s]} \]
\[ f_0 = \]
\[ dt = \text{timestep [s]} \]

Output:
\[ W_{gu} = \text{matrix of updated value of } W_g \text{ [m]} \]
\[ W_{cu} = \text{matrix of updated value of } W_c \text{ [m]} \]
\[ W_{su} = \text{matrix of updated value of } W_s \text{ [m]} \]
\[ \text{flr} = \text{flow exiting each cell within the step as runoff [m/s]} \]
\[ \text{fld} = \text{flow exiting each cell within the step as hypodermic [m/s]} \]
% percolation [m/s]

Wgu = Wg;
ppr = (l-f0) .* ks./ppr;
ec = (ppr > 0) .* (ppr .* exp(-1.*ks./ppr));
iflt = ppr - ec + pid;

% Linear absorption model
as = kap.* (1-Wc./Wc0);
k = find((as-evp)*dt > (Wc0-Wc));
if ~isempty(k)
    as(k) = (Wc0(k)-Wc(k))/dt+evp(k);
end
kl= as*dt < Wgu;
k2= as*dt>=Wgu & (as-iflt)*dt<Wgu;
k3= (as-iflt)*dt >= Wgu;
Wgu= Wgu - dt*as.*kl;
iflt=iflt+(as+Wgu/dt).*k2;
Wgu=Wgu.*(k2==0);
as=as+(as+Wgu/dt+iflt).*k3;
Wgu=Wgu.*(k3==0);
iflt=iflt.*(k3==0);

% Capillary balance
Wcu = Wc + dt*as;
k = find(evp > Wcu/dt);
evr=evp;
evr(k) = Wcu(k)/dt;
evr = evr.*(evr>0);
Wcu= Wcu - dt*evr;

% Gravitational soil balance
dipo=0.*Wgu;
pe=dipo;
su=dipo;
A=gam+bet;
k = find(Wgu >= Wg0);
if ~isempty(k)
    Wgu(k) = 0.999999*Wg0(k);
end
tp=1./A.*log((Wgu-iflt./A)./(Wg0-iflt./A));
itp=(imag(tp)==0 & tp > 0 & tp < dt);
k = itp==0;
pe=pe+k.*(-pe+iflt+1-exp(-dt*A)).*(Wgu-iflt./A)/dt;
dipo=dipo+k.*(-dipo+bet.*pe./A);
pe=pe+k.*(-pe+pe.*(A-bet)/A);
Wgu=Wgu+k.*(-Wgu+iflt./A+ (Wgu-iflt./A).*exp(-A.*dt));
su=su.*k.*su;
k = itp==1;
pe=pe+k.*(-pe+(iflt+1-exp(-A.*tp)).*(Wgu-iflt./A)/tp).*tp./dt+A/dt.*(dt-tp).*Wg0;
dipo=dipo+k.*(-dipo+pe.*bet./A);
pe=pe+k.*(-bet*pe./A);
su=su+k.*(-su+iflt-pe-dipo-(Wg0-Wgu)/dt);
Wgu=Wgu+k.*(-Wgu+Wg0);

% Total hypodermic flow
fld = dipo;
% Surface storage balance and runoff
qq=(cha.*pir+ec+su);
qqsalp=qq./alpsur;
Wsu=qqsalp.*(Wsu-qqsalp).*exp(-alpsur*dt);
flr=qq*(Wsu-qqsalp).*((1-exp(-alpsur*dt))/dt);  

function [Ts,Td,.ev]=energybalance(ff,aa_t,aa_r,pp_t,pp_r,C_T,C_R,...
         Td_ini,Tm,U,pr,Hair,t_end,step,C_h,alb,ks,ni_s,Tcost,etrsuetp)
%{
% ff= Vector of diurnal frequencies
Scalar or Matrix (if multiple modes, 3-D matrix)
   aa_t = air temperature amplitudes [K]
   aa_r = solar radiation amplitudes [Wm^-2]
   pp_t = air temperature phases
   pp_r = radiation phases
   C_T = constant part air temperature [K]
   C_R = constant part solar radiation [Wm^-2]
Scalar or Matrix
   U = windspeed [ms^-1]
   pr = air pressure [mb]
   Hair = air relative humidity [0-1]
   Td_ini = Td temperature at t=0 (I.C. at t=24.00) [K]
   Tm = mean air temp. of the previous day (for linearization) [K]
   Ts = soil temperature at surface [K]
   Td = soil temperature at depth dz [K]
%}  
dz=sqrt(1/ff(1)*ni_s);%0.05;%%
alpha=sqrt(365);%%% alpha=dz_deep/dz;

%BOUNDARY & INITIAL CONDITIONS
%Tcost=constant deep ground temperature; %280 %in K

%THERMODYNAMIC AND ELECTROMAGNETIC CONSTANTS
sigma=5.6697e-8; %Stefan Boltzmann

%SOIL PARAMETERS
%ni_s=Ks./(rhos.*cs) thermal diffusivity (m2/s) %1e-6; %Stull
%Ks=soil thermal conductivity (J/s/m/K)
eps_soi=0.98; %soil emissivity

%WATER
rhow=1000;

%AIR PARAMETERS
rhoair=1.225;
Lv=2.5e6; % Latent heat of vaporization
cp=1004; % Specific heat at constant pressure (J K^-1 kg^-1)
eps_air=0.87; % air emissivity

%SOLUTION OF CONSTANT PART
TmC=Tm-273.15;
Tm3=Tm.^3;
Tm4=Tm3.*Tm;
KH=C_h.*U;
rhoaircpKH=rhoair*cp*KH;
P622=.622*rhoair*Lv*etrsuetp./pr.*KH;
es4Tm3=4*sigma*eps_soi*Tm3;
ea4Tm3=4*sigma*eps_air*Tm3;
es1=6.112*exp((17.67*TmC.)/(TmC+243.5));
es2=es1.*(17.67./(Tm-29.65)-17.67*TmC.)/((Tm-29.65).^2));
ea1=es1.*Hair;
ea2=es2.*Hair;
%compute time-independent coefficients of constant part solution

den=ks./dz-es4Tm3-rhoaircpKH*P622.*es2;
alphadz2=(1+alpha)*dz^2;

D0= 2*ni_s*(Tcost./(alpha*alphadz2)-(P622.*Tm.*(es2-ea2)+3*sigma*(eps_soi-eps_air)*Tm4+P622.*(ea1-es1))./(den.*alphadz2));

D1=2*ni_s.*(1-alb)./((den.*alphadz2));

D2=2*ni_s.*(P622.*ea2+rhoaircpKH+ea4Tm3)./(den.*alphadz2);

PP=2*ni_s*(-alpha*(ks./((den.*dz))-1-alpha)./(alpha*alphadz2));

%start sub-step loop

for tt=0:step:t_end;
    eP=exp(PP);
    Td1=(D1.*C_R+D2.*C_T)./PP.*(ePt-1);
    Ts1=-(ks./dz.*Td1+variaroba)./(ks./dz-es4Tm3-rhoaircpKH*P622.*es2);

    %end of constant part solution

    Td2=0*Td1;
    Ts2=0*Ts1;

    for i=1:length(ff);
        w=ff(i);A_T=aa_t(:,:,i);A_R=aa_r(:,:,i);p_T=pp_t(:,:,i);p_R=pp_r(:,:,i);
        PPw=PP.*w;
        PPw2=PPw.*w;
        wttp_T=w*tt+p_T;
        wttp_R=w*tt+p_R;
        D1_S=D1.*A_R;
        D2_S=D2.*A_T;
        Td02=(PPw.*(D1_S.*(ePt.*cos(p_R)-cos(wttp_R)) + D2_S.*ePt.*cos(p_T) - cos(wttp_T)))+eP.*(Td02+Pw^2) + ePt.*((PP2+PPw2) + PP2.*((D2_S.*sin(p_T)+D1_S.*sin(p_R)) - PP2.*((D2_S.*sin(wttp_T)+D1_S.*sin(wttp_R)))./(PP3+PPw2));
        Ts02=-(ks./dz.*Td02+A_R.*sin(wttp_R).*(1-alb)... +A_T.*sin(wttp_T).*ea4Tm3+rhoaircpKH*P622.*ea2)... +satspum(Ts,pr,0)-satspum(aa_t.*sin(w.*tt+pp_t)+C_T,pr,0).*Hair);

    Td2=Td2+Td02;
    Ts2=Ts2+Ts02;
end

Td=Td1+Td2;
Ts=Ts1+Ts2;

ev_p=ev_p0;
ev_p0=rhoair*lv*etrsuetp.*KH.*(satspum(Ts,pr,0)-satspum(aa_t.*sin(w.*tt+pp_t)+C_T,pr,0).*Hair);

ev=evp+(evp_old+evp0).*((step/2)); %step in sec

end
evp = evp./(rhow*Lv);
evp = evp.*(evp>=0);
evp0 = (7e-3)/24;
evp(evp>evp0) = evp0;

function q = satspum(T, P, dT)
% T in °K;
% P in mb
% dT = dewpoint depression in °K (or °C)
ep = 0.622;
Tc = T - 273.15 - dT;
es = 6.112*exp(17.67*Tc./(Tc+243.5));
if dT
  es = es - (0.00066*(1+0.00115*Tc)).*P.*dT;
end
q = ep*es./(P - (1 - ep)*es);

function [hrise, hset] = solarhours(lat, lon, jday)
% Compute hours of sunrise and sunset for Julian day jday at location (lat, lon)

hpre = 0;
hpos = 12;
while (hpos-hpre) > 1/60
  hrise = (hpos+hpre)/2;
  [az, el] = solarpos(hrise, jday, lat, lon);
  if el > 0
    hpos = hrise;
  else
    hpre = hrise;
  end
end

hpre = 12;
hpos = 24;
while (hpos-hpre) > 1/60
  hset = (hpos+hpre)/2;
  [az, el] = solarpos(hset, jday, lat, lon);
  if el > 0
    hpre = hset;
  else
    hpos = hset;
  end
end

function [az, el] = solarpos(hour, day, lat, lon)
% [az, el] = solar azimuth and elevation (deg)
% hour = local time (may be decimal)
% day = Julian day
% lat, lon = geographic position (deg N, deg E)
delta = 23.45*pi/180*cos(2*pi/365*((172-mod(day,365))));
dt1 = round(lon/15) - lon/15;
az = (hour+12-dt1)*15;
az = mod(az, 360);
sinalp = sin(delta).*sin(lat*pi/180)+cos(delta).*cos(lat*pi/180).*cos(az*pi/180);
el = 180/pi*asin(sinalp);
az = mod(180+az, 360);