Watbal3

```
import pylab
```

Layered Water Balance Model

Soil Water Fluxes

The 1-D finite elements formulation of Richards' equation is frequently formulated as:

$$Q_{bl} = 1/2(K_u(j) + K_u(j+1))(\frac{h(j) - h(j+1)}{1/2(\delta_z(j) + \delta_z(j+1)} - 1)$$

where Q_{bl} is the (upwards) flow across the bottom boundary of soil layer j, h is the average capillary pressure in a layer (here expressed as a positive value) and δ_z is the thickness of a layer.

```
var('j')
delzvec(j) = function('delzvec',j)         # thickness of layer j
ksatvec(j) = function('ksatvec',j)         # saturated hydraulic conductivity in
layer j
thetavec(j) = function('thetavec',j)       # volumetric soil moisture in layer j
suvec(j) = function('suvec',j)    # Saturation degree in layer j
pcapvec(j) = function('pcapvec',j)             # Matric suction in layer j
kunsatvec(j) = function('kunsatvec',j)     # Unsaturated hydraulic conductivity in
layer j
qblvec(j) = function('qblvec',j)           # Flow across the bottom boundary of
layer j (upwards)
```

```
# Classical formulation of flow across the bottom boundary of layer j (upwards)
eq_qbl = qblvec(j) == 1/2*(kunsatvec(j) + kunsatvec(j+1))*(-1 + (pcapvec(j) -
pcapvec(j+1))/(1/2*(delzvec(j) + delzvec(j+1))))
eq_qbl
```

```
qblvec(j) == -1/2*(2*(pcapvec(j + 1) - pcapvec(j))/(delzvec(j + 1)
delzvec(j)) + 1)*(kunsatvec(j + 1) + kunsatvec(j))
```

```
# Infiltration capacity
qinfc = (pcapvec(1)/(1/2*delzvec(1)) + 1)*ksatvec(1)
qinfc
```

```
(2*pcapvec(1)/delzvec(1) + 1)*ksatvec(1)
```

Matrix pressure head

```
thetarvec(j) = function('thetarvec',j)         # Residual soil moisture in layer j
thetasvec(j) = function('thetasvec',j)         # Saturated soil moisture in layer j
avgvec(j) = function('avgvec',j)               # van Genuchten parameter alpha in
layer j
mvgvec(j) = function('mvgvec',j)               # van Genuchten Parameter m in
layer j
```

```
nvgvec(j) = function('nvgvec',j)             # van Genuchten parameter n in
layer j
```

```
eq_theta_su = thetavec(j) == thetarvec(j) - suvec(j)*thetarvec(j) +
suvec(j)*thetasvec(j)
eq_theta_su
```

```
thetavec(j) == -suvec(j)*thetarvec(j) + suvec(j)*thetasvec(j) +
thetarvec(j)
```

```
# Infiltration capacity
qinfc = (pcapvec(1)/(1/2*delzvec(1)) + 1)*ksatvec(1)
qinfc
```

```
(2*pcapvec(1)/delzvec(1) + 1)*ksatvec(1)
```

Matrix pressure head after van Genuchten

```
thetarvec(j) = function('thetarvec',j)       # Residual soil moisture in layer j
thetasvec(j) = function('thetasvec',j)       # Saturated soil moisture in layer j
avgvec(j) = function('avgvec',j)             # van Genuchten parameter alpha in
layer j
mvgvec(j) = function('mvgvec',j)             # van Genuchten Parameter m in
layer j
nvgvec(j) = function('nvgvec',j)             # van Genuchten parameter n in
layer j
```

where 1/avgvec has dimension of length, while mvg and nvg are related by:

```
eq_mvg = mvgvec(j) == 1 - (1/nvgvec(j))
eq_mvg
```

```
mvgvec(j) == -1/nvgvec(j) + 1
```

eq_h can be solved for su(j) to obtain:

```
eq_suh = suvec(j) == (1 / (1 + (avgvec(j) * pcapvec(j))^nvgvec(j)))^mvgvec(j)
eq_suh
```

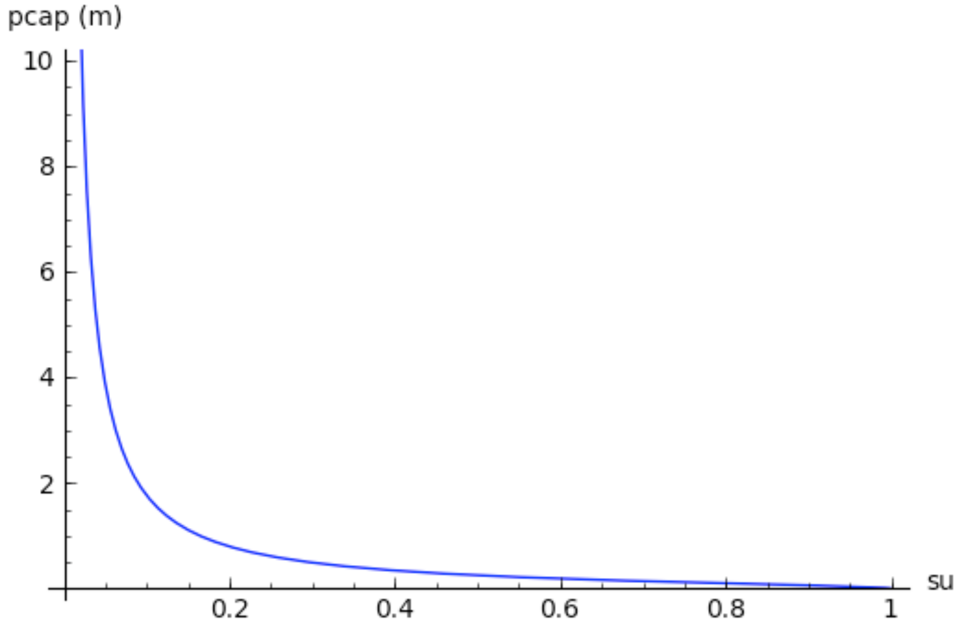
```
suvec(j) == (1/((pcapvec(j)*avgvec(j))^nvgvec(j) + 1))^mvgvec(j)
```

```
#parameters for sandy loam
pars_sl = {avgvec(j):7.5, nvgvec(j):1.89,ksatvec(j):1.228*10^-5,
thetarvec(j):0.065, thetasvec(j):0.41}
pars_sl[mvgvec(j)] = eq_mvg.rhs().subs_expr(pars_sl)
pars_sl[delzvec(j)] = 0.5
pars_sl
```

```
{delzvec(j): 0.500000000000000, avgvec(j): 7.50000000000000,
thetarvec(j): 0.0650000000000000, mvgvec(j): 0.470899470899471,
nvgvec(j): 1.89000000000000, thetasvec(j): 0.410000000000000,
ksatvec(j): 0.0000122800000000000}
```

```
curve = eq_h.subs(eq_mvg).subs_expr(pars_sl).subs_expr(suvec(j) == x)
print curve.rhs()
p1 = plot(curve.rhs(), (x,0.01,1))
p1.axes_labels(['su','pcap (m)'])
p1.axes_range(0,1,0,10)
#p1.show(frame=True, axes=False)
p1
```

```
0.133333333333333*(1/x^2.12359550561798 - 1)^0.529100529100529
```



The relative conductivity is given as:

```
krwvec(j) = suvec(j)^(1/2)*(1 - (1 - suvec(j)^(1/mvgvec(j)))^mvgvec(j))^2
krwvec(j)
```

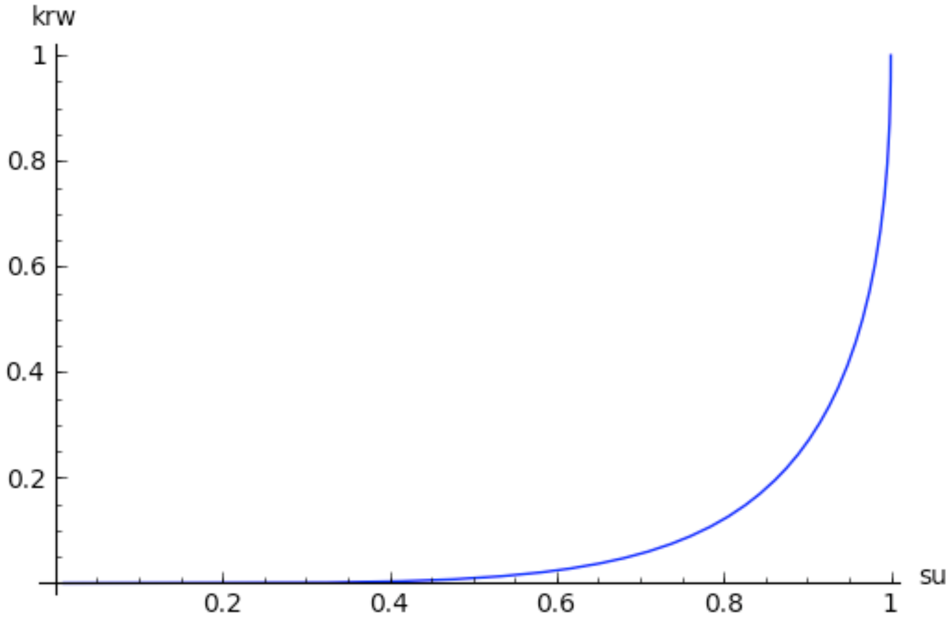
```
((-suvec(j)^(1/mvgvec(j)) + 1)^mvgvec(j) - 1)^2*sqrt(suvec(j))
```

```
eq_kunsat = kunsatvec(j) == ksatvec(j)*krwvec(j)
eq_kunsat
```

```
kunsatvec(j) == ((-suvec(j)^(1/mvgvec(j)) + 1)^mvgvec(j) -
1)^2*ksatvec(j)*sqrt(suvec(j))
```

```
curve = krwvec(j).subs_expr(eq_mvg).subs_expr(pars_sl).subs_expr(suvec(j) == x)
print curve
p1 = plot(curve, (x,0.01,1))
p1.axes_labels(['su','krw'])
p1.axes_range(0,0.99,0,1)
p1
```

```
((-x^2.12359550561798 + 1)^0.470899470899471 - 1)^2*sqrt(x)
```



Seapage face flow

In the original VOM, seapage face flow was formulated as a function of the saturated area fraction (omgo = 1 - omgu) and ys, where

```
omgo == 1/2*(2*cz - 2*zr - sqrt((2*cz - zr)^2 - 2*(2*cz - zr)*ys +
2*ys*zr - zr^2))/(cz - zr)
```

or

```
omgu == yu/(cz - zr)
ys == cz - omgu * yu
```

```
var('yu omgo omgu ys cz zr zm cgs g0 spgfcf')
eq_omgu = omgu == yu/(cz - zr)
eq_ys = ys == cz - omgu * yu
```

```
soln = solve(eq_omgu.subs(solve(eq_ys,yu)[0]),omgu)
soln
```

```
[omgu == -sqrt(cz/(cz - zr) - ys/(cz - zr)), omgu == sqrt(cz/(cz -
zr) - ys/(cz - zr))]
```

```
eq_omgu1 = soln[1]; eq_omgu1
```

```
omgu == sqrt(cz/(cz - zr) - ys/(cz - zr))
```

```
eq_omgo = omgo == 1 - eq_omgu1.rhs()
eq_omgo
```

```
omgo == -sqrt(cz/(cz - zr) - ys/(cz - zr)) + 1
```

```
eq_omgo1 = omgo == 1/2*(2*cz - 2*zr - sqrt((2*cz - zr)^2 - 2*(2*cz - zr)*ys +
2*ys*zr - zr^2))/(cz - zr)
eq_spgfcf = spgfcf == ksatvec(j) * omgo / (cos(g0) * cgs) * 1/2 * (ys - zr)
```

```
assume(cz > zr, cz > ys, ys > zr)
```

```
eq_spgfcf.subs(eq_omgo).expand().factor()
```

```
spgfcf == 1/2*(sqrt(cz - zr) - sqrt(cz - ys))*(ys -
zr)*ksatvec(j)/(sqrt(cz - zr)*cgs*cos(g0))
```

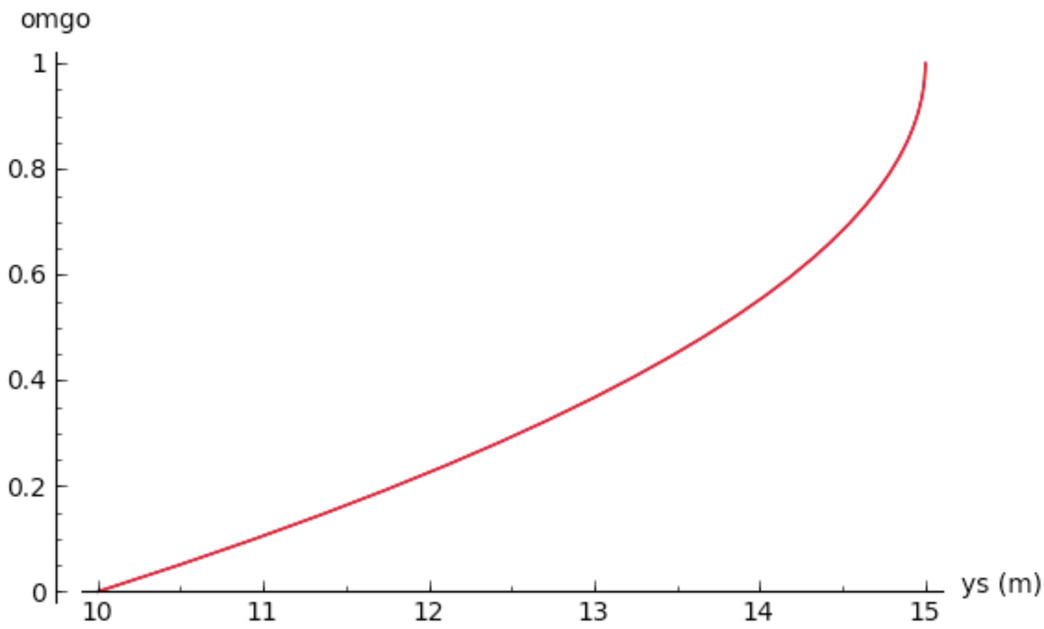
```
eq_spgfcf.subs(eq_omgo1)
```

```
spgfcf == 1/4*(ys - zr)*(2*cz - 2*zr - sqrt((2*cz - zr)^2 - 2*(2*c:
- zr)*ys + 2*ys*zr - zr^2))*ksatvec(j)/((cz - zr)*cgs*cos(g0))
```

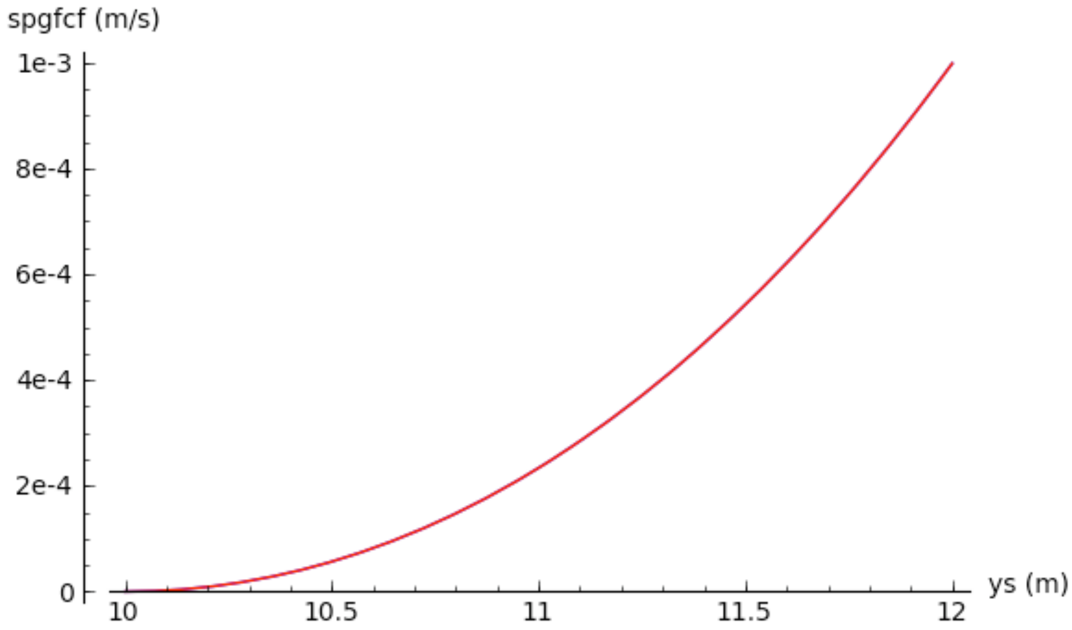
For Howard Springs, this gave the following relationship between the thickness of the saturated zone (ys) and runoff (spgfcf):

```
pars1 = {}
pars1[zr] = 10
pars1[cz] = 15
pars1[cgs] = 10
pars1[g0] = 0.033
pars1[ksatvec(j)] = 1.23e-5
```

```
P = plot(1 - eq_omgu1.rhs().subs(pars1),(ys,10,15))
P += plot(eq_omgo.rhs().subs(pars1),(ys,10,15), rgbcolor = 'red')
P.axes_labels(['ys (m)','omgo'])
P
```




```
P = plot(3600*eq_spgfcf.rhs().subs(eq_omgo).subs(pars1), (ys,10,12))
P += plot(3600*eq_spgfcf.rhs().subs(eq_omgo1).subs(pars1), (ys,10,12), rgbcolor =
'red')
P.axes_labels(['ys (m)','spgfcf (m/s)'])
P
```



A more intuitive way of formulating runoff for a horizontal catchment with vertical channel walls would be a function of the head above channel elevation (centre of gravity = 0.5*(ys-zr)) and hydrol. cond. multiplied by the flow cross-section divided by a mean distance to channel:

```
eq_spgfcf1 = ksatvec(j) * 1/2 * (ys - zr)^2 / (cgs * cos(g0))
eq_spgfcf1.subs(pars1)
```

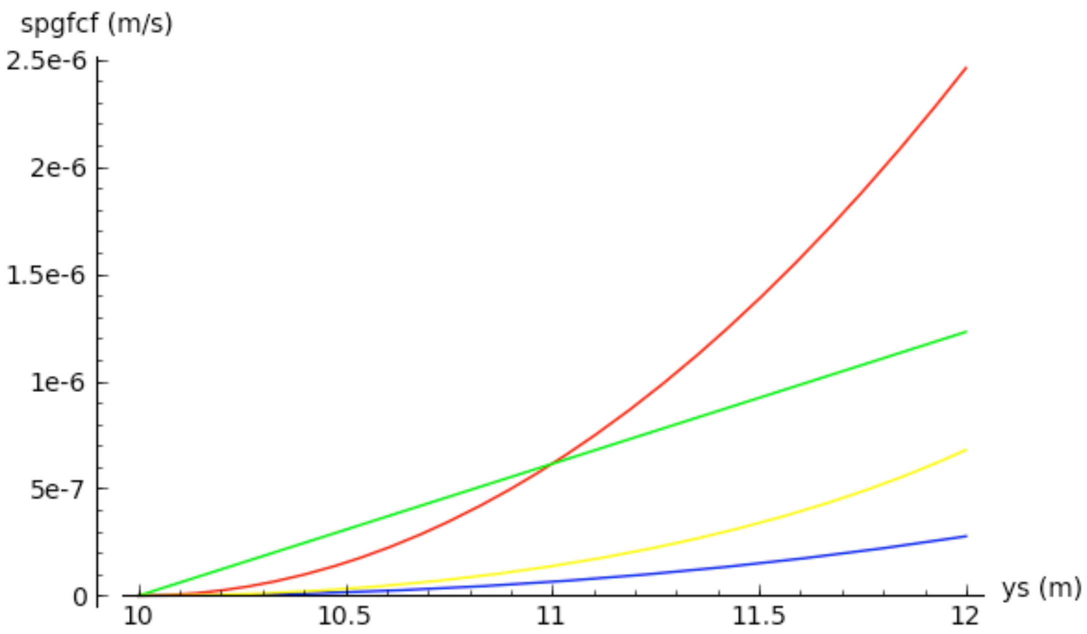
```
(6.15335019513449e-7)*(ys - 10)^2
```

```
eq_spgfcf2 = ksatvec(j) * 1/2 * (ys - zr) / (cgs * cos(g0))
eq_spgfcf2.subs(pars1)
```

```
(6.15335019513449e-7)*ys - 6.15335019513449e-6
```

```
# In Globe version 580
eq_spgfcf3 = 0.5 * (ys - zr) * (cz - zr - sqrt(-2 * ys * cz + 2 * ys * zr + cz ^
2 - zr ^ 2)) * ksatvec(j)/ ((cz - zr) * cgs * cos(g0))
```

```
P = plot(eq_spgfcf.rhs().subs(eq_omgo).subs(pars1),(ys,10,12))
P += plot(eq_spgfcf1.subs(pars1),(ys,10,12), rgbcolor = 'red')
P += plot(eq_spgfcf2.subs(pars1),(ys,10,12), rgbcolor = 'green')
P += plot(eq_spgfcf3.subs(pars1),(ys,10,12), rgbcolor = 'yellow')
P.axes_labels(['ys (m)','spgfcf (m/s)'])
P
```



All of the alternative formulations would lead to larger runoff than the on in the original VOM based on Reggiani (2000). Therefore, we will adopt the original formulation:

```
eq_spgfcf.subs(eq_omgo)
```

```
spgfcf == -1/2*(sqrt(cz/(cz - zr) - ys/(cz - zr)) - 1)*(ys -
zr)*ksatvec(j)/(cgs*cos(g0))
```

Equilibrium soil moisture (θ)

No fluxes between soil layers occur if the folowing $h(j)$ profile is achieved:

```
soln = solve(0 == eq_qbl.rhs(),pcapvec(j))
soln
```

```
[pcapvec(j) == 1/2*delzvec(j + 1) + 1/2*delzvec(j) + pcapvec(j + 1
```

```
eq_heq = soln[0]
```

For sandy loam, the equilibrium su above a saturated layer would thus be:

```
eq_suh.subs(eq_heq.subs_expr(pcapvec(j + 1) == 0, delzvec(j + 1) ==
delzvec(j))).subs(pars_sl)
```

```
suvec(j) == 0.297131909518541
```

```
# Equilibrium suvec in the old code
dummydepth =  ys + delzvec(j) / 2
dummysu = (1 + (avgvec(j) * (dummydepth - ys)) ^ nvgvec(j)) ^ (-mvgvec(j))
```

```
dummysu.subs(pars_sl)
```

```
0.504217476108434
```

In the old code, equilibrium pcap in the layer above the saturated one was assumed to be

=0.5*delzvec(j), while in the new code it is = delzvec(j):

```
eq_heq.subs_expr(pcapvec(j + 1) == 0, delzvec(j + 1) == delzvec(j)).subs(pars_sl)
```

```
pcapvec(j) == 0.500000000000000
```

Position of the water table

Note that the finite elements formulation of Richards' equation leads to an equilibrium above a saturated layer if $(\frac{h(j+1)-h(j)}{1/2(\delta_z(j)+\delta_z(j+1)} - 1) = 0$

where $h(j+1) = 0$. This would imply that water would flow into the saturated layer if $h(j)$ was any lower. This again would imply that water could flow from an unsaturated layer into a layer that is already saturated! Equilibrium would be reached at:

```
pcapeqvec(j) = function('pcapeqvec',j)          # Equilibrium capillary head
```

```
soln = solve(0 == eq_qbl.rhs().subs_expr(pcapvec(j+1) == 0),pcapvec(j))
soln
```

```
[pcapvec(j) == 1/2*delzvec(j + 1) + 1/2*delzvec(j)]
```

```
eq_pcapeq = pcapeqvec(j) == soln[0].rhs()
eq_pcapeq
```

```
pcapeqvec(j) == 1/2*delzvec(j + 1) + 1/2*delzvec(j)
```

```
eq_pcapeq.rhs().subs_expr(delzvec(j+1) == delzvec(j)).subs(pars_sl)
```

```
0.500000000000000
```

Equilibrium su at which there would be no flow into the saturated layer below:

```
sueqvec(j) = function('sueqvec(j)',j)
eq_sueq1 = eq_suh.subs(pcapvec(j) == eq_pcapeq.rhs())
eq_sueq1
```

```
suvec(j) == (1/((1/2*(delzvec(j + 1) +
delzvec(j))*avgvec(j))^nvgvec(j) + 1))^mvgvec(j)
```

Equilibrium θ at which there would be no flow into the saturated layer below:

```
thetaeqvec(j) = function('thetaeqvec',j)
eq_thetaeq1 = eq_theta_su.subs_expr(eq_suh.subs(pcapvec(j) == pcapeqvec(j)))
eq_thetaeq1
```

```
thetavec(j) == -(1/((avgvec(j)*pcapeqvec(j))^nvgvec(j) +
1))^mvgvec(j)*thetarvec(j) + (1/((avgvec(j)*pcapeqvec(j))^nvgvec(j
+ 1))^mvgvec(j)*thetasvec(j) + thetarvec(j)
```

In sandy loam, if all layers are of the same thickness, this would give:

```
eq_thetaeq1.subs_expr(eq_pcapeq).subs_expr(delzvec(j + 1) ==
delzvec(j)).subs(pars_sl)
```

```
thetavec(j) == 0.167510508783897
```

This would be equivalent to a saturation degree of:

```
eq_suh.subs(pcapvec(j) == eq_pcapeq.rhs()).subs_expr(delzvec(j + 1) ==
delzvec(j)).subs(pars_sl)
```

```
suvec(j) == 0.297131909518541
```

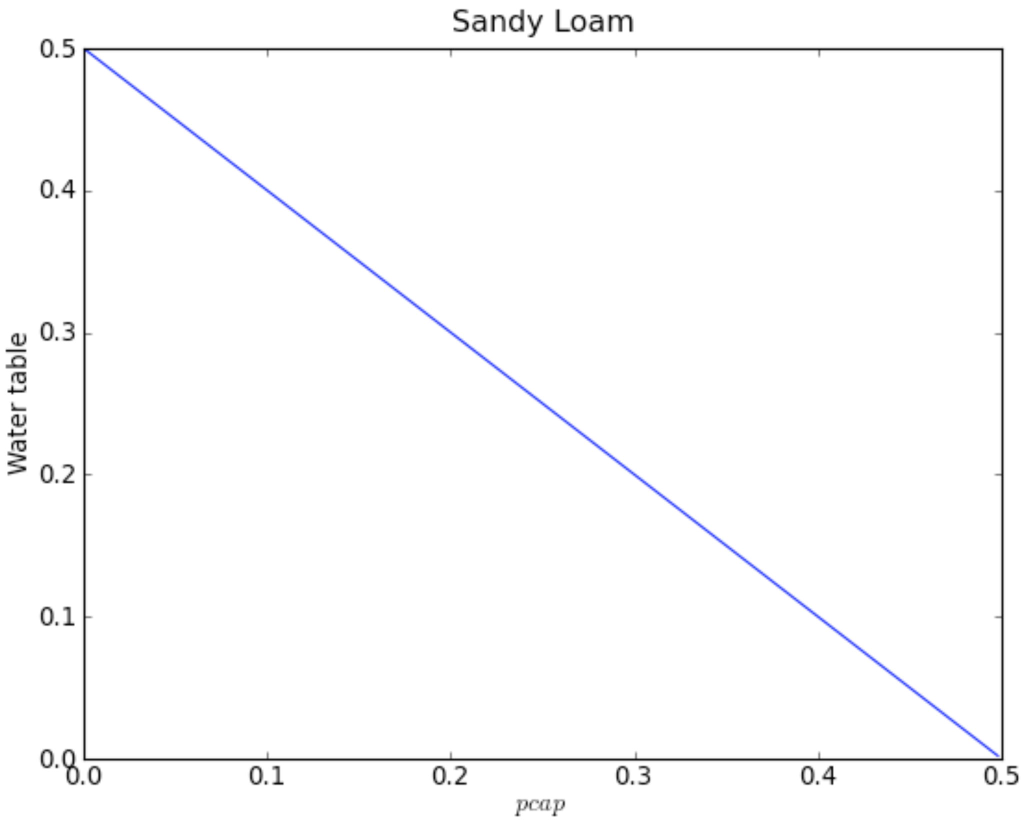
For consistency with the formulation of unsaturated flow, we will formulate the position of the water table as a linear function between 0 at $pcap = pcap_{eq}$ and δ_z at $pcap = 0$.

```
wt1(j) = function('wt1', j)   # Thickness of saturated fraction in layer j
eq_wt1 = wt1(j) == delzvec(j) - delzvec(j)/eq_pcapeq.rhs()*pcapvec(j)
eq_wt1
```

```
wt1(j) == -2*delzvec(j)*pcapvec(j)/(delzvec(j + 1) + delzvec(j)) +
delzvec(j)
```

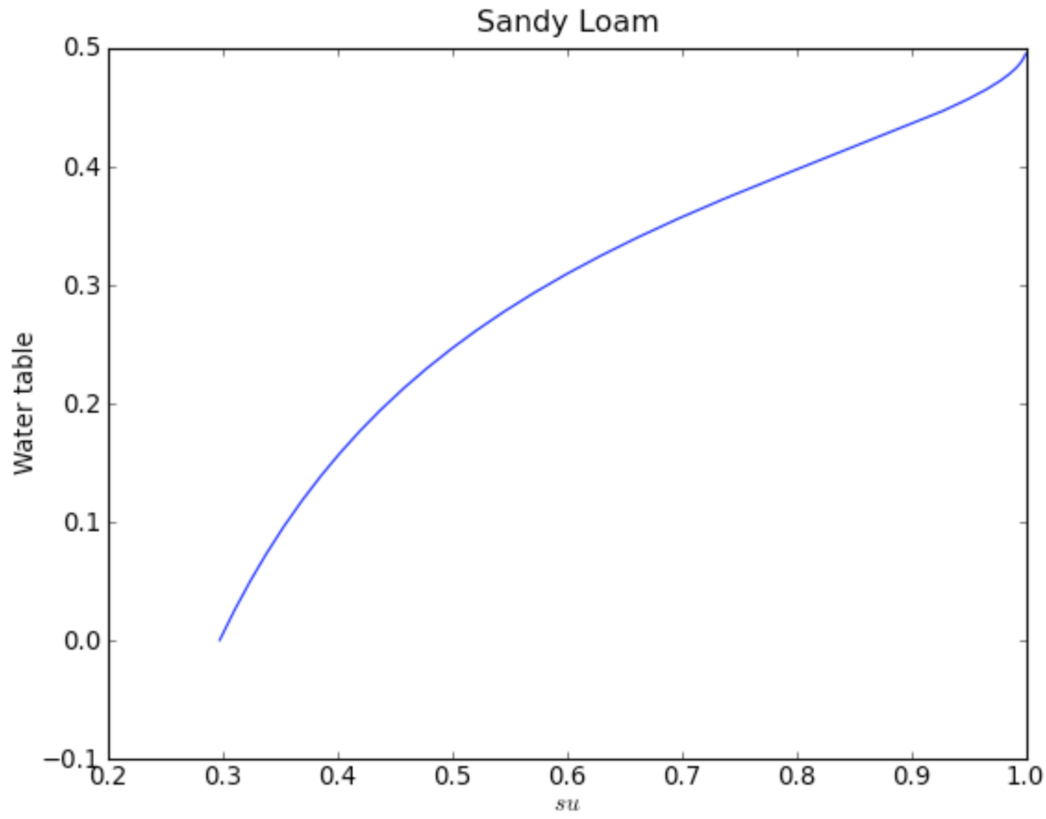
This would give the following relation between $pcap$ and the position of the water table in the bottom layer:

```
%hide
pcapvals = srange(0,eq_pcapeq.rhs().subs_expr(delzvec(j+1) ==
delzvec(j)).subs(pars_sl).n(),0.001)
wtvals = [eq_wt1.rhs().subs_expr(delzvec(j+1) ==
delzvec(j)).subs(pars_sl).subs_expr(pcapvec(j) == dummy).n() for dummy in
pcapvals]

pylab.clf()
pylab.figure(1)
pylab.plot(pcapvals,wtvals)
#pylab.ylim(0.45,0.5)
pylab.title('Sandy Loam')
pylab.xlabel("$pcap$") # label the axes
pylab.ylabel("Water table")
pylab.savefig('foo.png',dpi = 80) # fire!
pylab.close()
```



As a function of su:

```
%hide
suvals = srange(eq_suh.rhs().subs_expr(pcapvec(j) ==
delzvec(j)).subs(pars_sl).n(),1,0.001)
wtsuvals = [eq_wt1.rhs().subs_expr(delzvec(j+1) ==
delzvec(j)).subs_expr(eq_h).subs_expr(suvec(j) == dummy).subs(pars_sl).n() for
dummy in suvals]

pylab.figure(1)
pylab.plot(suvals,wtsuvals)
#pylab.ylim(0.45,0.5)
pylab.title('Sandy Loam')
pylab.xlabel("$su$") # label the axes
pylab.ylabel("Water table")
pylab.savefig('foo.png',dpi = 80) # fire!
pylab.close()
```



As a function of θ:

```
eq_wt1.subs_expr(eq_h).subs_expr(eq_su_theta)
```

```
wt1(j) == -2*((-(thetavec(j) - thetarvec(j))/(thetarvec(j) -
thetasvec(j)))^(-1/mvgvec(j)) -
1)^(1/nvgvec(j))*delzvec(j)/((delzvec(j + 1) +
delzvec(j))*avgvec(j)) + delzvec(j)
```

```
thetavals = srange(eq_suh.rhs().subs_expr(pcapvec(j) ==
delzvec(j)).subs(pars_sl).n(),1,0.001)
```

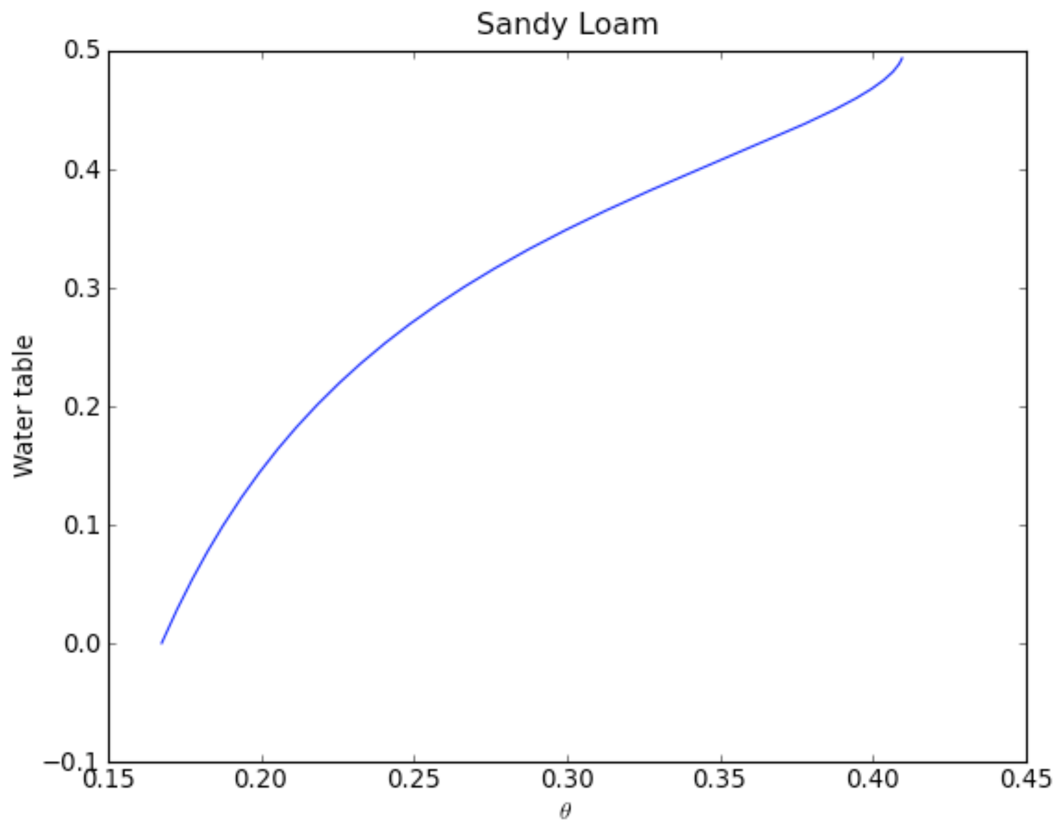
```
eq_wt1.rhs().subs_expr(eq_h).subs_expr(eq_su_theta).subs_expr(delzvec(j+1) ==
delzvec(j)).subs(pars_sl)
```

```
-0.133333333333333*(1/(2.89855072463768*thetavec(j) -
0.188405797101449)^2.12359550561798 - 1)^0.529100529100529 +
0.500000000000000
```

```
var('dummy')
val_thetaeq = (eq_thetaeq1.subs_expr(eq_pcapeq).subs_expr(delzvec(j + 1) ==
delzvec(j)).subs(pars_sl).rhs()).n()
val_thetas = thetasvec(j).subs(pars_sl).n()
thetavals = srange(val_thetaeq,val_thetas,0.001)
```

```
wtthetavals =
[eq_wt1.rhs().subs_expr(eq_h).subs_expr(eq_su_theta).subs_expr(delzvec(j+1) ==
delzvec(j)).subs(pars_sl).subs_expr(thetavec(j) == dummy).n() for dummy in
thetavals]
```

```
%hide
pylab.figure(1)
pylab.plot(thetavals,wtthetavals)
#pylab.ylim(0.45,0.5)
pylab.title('Sandy Loam')
pylab.xlabel("$\\theta$") # label the axes
pylab.ylabel("Water table")
pylab.savefig('foo.png',dpi = 80) # fire!
pylab.close()
```



The non-linearity between θ and the water table will lead to a stepwise change of the recession curve as the water table moves from one layer to the next. The steps would become less visible as the layer thickness is decreased.

Change in state variables

The soil moisture in each layer changes as a function of the fluxes. In the top soil layer, soil evaporation and infiltration have to be considered in addition to root water uptake and unsaturated flow, while in the bottom layer seepage face flow has to be considered. To relate volumetric fluxes to changes in the saturation degree, we need to consider the $su(\theta)$ function and the thickness of each layer.

```
eq_su_theta
```

$$\mathrm{suvec}(j) == -(\mathrm{thetavec}(j) - \mathrm{thetarvec}(j))/(\mathrm{thetarvec}(j) - \mathrm{thetasvec}(j))$$

```
iovec(j) = function('iovec',j)      # input - output in each soil layer
ruptkvec(j) = function('ruptkvec',j)   # root water uptake by perennial veg
```

```
ruptkgvec(j) = function('ruptkgvec',j)   # root water uptake by seasonal veg
var('esoil')   # soil evaporation
dsuvec(j) = function('dsuvec',j)  # change in saturation degree per time
```

```
dthetavec(j) = iovec(j)/delzvec(j)
```

```
var('theta thetar thetas')
sutheta = eq_su_theta.rhs().subs_expr(thetavec(j) == theta, thetasvec(j) ==
thetas, thetarvec(j) == thetar)
sutheta
```

```
-(theta - thetar)/(thetar - thetas)
```

```
diff(sutheta,theta)
```

```
-1/(thetar - thetas)
```

```
eq_dsuvec = dsuvec(j) == dthetavec(j)/(thetasvec(j) - thetarvec(j))
eq_dsuvec
```

```
dsuvec(j) == -iovec(j)/((thetarvec(j) - thetasvec(j))*delzvec(j))
```

```
eq_theta_su
```

```
thetavec(j) == -suvec(j)*thetarvec(j) + suvec(j)*thetasvec(j) +
thetarvec(j)
```

Overflowing layers

If a saturated layer is between another saturated layer above and an unsaturated layer below, it can happen that the equations will still lead to a net water uptake into the saturated layer:

```
eq_qbottom = eq_qbl.subs_expr(eq_h, eq_h(j = j + 1),eq_kunsat, eq_kunsat(j = j +
1))
eq_qbottom
```

```
qblvec(j) == 1/2*(2*((suvec(j)^(-1/mvgvec(j)) -
1)^(1/nvgvec(j))/avgvec(j) - (suvec(j + 1)^(-1/mvgvec(j + 1)) -
1)^(1/nvgvec(j + 1))/avgvec(j + 1))/(delzvec(j + 1) + delzvec(j)) -
1)*(((-suvec(j + 1)^(1/mvgvec(j + 1)) + 1)^mvgvec(j + 1) -
1)^2*ksatvec(j + 1)*sqrt(suvec(j + 1)) + ((-suvec(j)^(1/mvgvec(j))
1)^mvgvec(j) - 1)^2*ksatvec(j)*sqrt(suvec(j)))
```

```
eq_qtop = (eq_qbl.subs_expr(eq_h, eq_h(j = j + 1),eq_kunsat, eq_kunsat(j = j +
1))).subs_expr(j == j - 1)
eq_qtop
```

```
qblvec(j - 1) == -1/2*(2*((suvec(j)^(-1/mvgvec(j)) -
1)^(1/nvgvec(j))/avgvec(j) - (suvec(j - 1)^(-1/mvgvec(j - 1)) -
1)^(1/nvgvec(j - 1))/avgvec(j - 1))/(delzvec(j - 1) + delzvec(j)) -
1)*(((-suvec(j - 1)^(1/mvgvec(j - 1)) + 1)^mvgvec(j - 1) -
1)^2*ksatvec(j - 1)*sqrt(suvec(j - 1)) + ((-suvec(j)^(1/mvgvec(j))
1)^mvgvec(j) - 1)^2*ksatvec(j)*sqrt(suvec(j)))
```

```
eq_qbl.subs_expr(pcapvec == x).rhs()
```

```
1/2*(2*(x - pcapvec(j + 1))/(delzvec(j + 1) + delzvec(j)) -
1)*(kunsatvec(j + 1) + kunsatvec(j))
```

```
net_qbl(j) = eq_qbottom.rhs() - eq_qtop.rhs()
net_qbl(j)
```

```
1/2*(2*((suvec(j)^(-1/mvgvec(j)) - 1)^(1/nvgvec(j))/avgvec(j) -
(suvec(j - 1)^(-1/mvgvec(j - 1)) - 1)^(1/nvgvec(j - 1))/avgvec(j -
1))/(delzvec(j - 1) + delzvec(j)) + 1)*(((-suvec(j - 1)^(1/mvgvec(
- 1)) + 1)^mvgvec(j - 1) - 1)^2*ksatvec(j - 1)*sqrt(suvec(j - 1)) 
((-suvec(j)^(1/mvgvec(j)) + 1)^mvgvec(j) -
1)^2*ksatvec(j)*sqrt(suvec(j))) + 1/2*(2*((suvec(j)^(-1/mvgvec(j))
1)^(1/nvgvec(j))/avgvec(j) - (suvec(j + 1)^(-1/mvgvec(j + 1)) -
1)^(1/nvgvec(j + 1))/avgvec(j + 1))/(delzvec(j + 1) + delzvec(j)) 
1)*(((-suvec(j + 1)^(1/mvgvec(j + 1)) + 1)^mvgvec(j + 1) -
1)^2*ksatvec(j + 1)*sqrt(suvec(j + 1)) + ((-suvec(j)^(1/mvgvec(j))
1)^mvgvec(j) - 1)^2*ksatvec(j)*sqrt(suvec(j)))
```

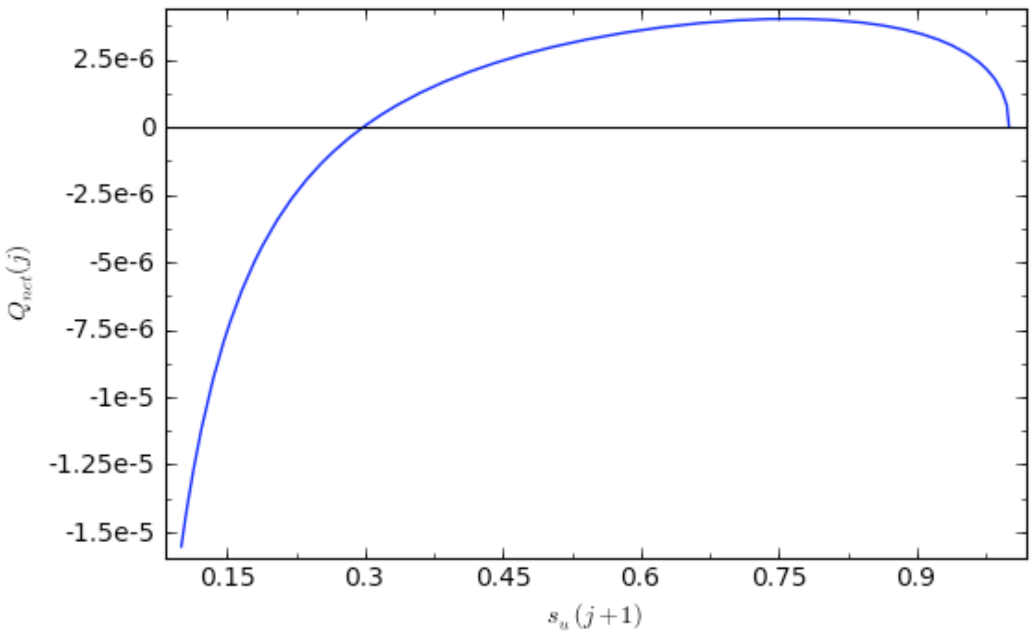
```
var('delz')
test = (net_qbl(j).subs_expr(mvgvec(j - 1) == mvgvec(j), mvgvec(j + 1) ==
mvgvec(j), nvgvec(j - 1) == nvgvec(j), nvgvec(j + 1) == nvgvec(j), avgvec(j - 1)
== avgvec(j),avgvec(j + 1) == avgvec(j), ksatvec(j - 1) == ksatvec(j), ksatvec(j
+ 1) == ksatvec(j), delzvec(j - 1) == delz, delzvec(j + 1) == delz, suvec(j - 1)
== 1, suvec(j) == 1, suvec(j + 1) == x))
test
```

```
(0^mvgvec(j) - 1)^2*ksatvec(j) - 1/2*(2*((x^(-1/mvgvec(j)) -
1)^(1/nvgvec(j))/avgvec(j) - 0^(1/nvgvec(j))/avgvec(j))/(delz +
delzvec(j)) + 1)*(((-x^(1/mvgvec(j)) + 1)^mvgvec(j) -
1)^2*sqrt(x)*ksatvec(j) + (0^mvgvec(j) - 1)^2*ksatvec(j))
```

```
test.subs_expr(pars_sl)(delz = 0.5)(x = 0.9)
```

```
3.50069401157976e-6
```

```
P = plot(test.subs_expr(pars_sl)(delz = 0.5),(x,0.1,1),frame = True)
P.axes_labels(['$s_u(j + 1)$','$Q_{net}(j)$'])
P
```



The above plot demonstrates that there would be a net flow of water into the saturated layer j if layer $j+1$ has saturation contents between 0.3 and <1.0. This is caused by the fact that the decrease in unsaturated hydraulic conductivity with decreasing soil moisture offsets the increase in matric potential gradient, so that the downwards flow out of the saturated layer initially decreases with decreasing soil moisture below.

```
P = plot((eq_kunsat.subs_expr(eq_suh).subs_expr(pars_sl).subs_expr(pcapvec(j) ==
```

```
x).rhs()),(x,0.0001,1))
P.axes_labels(['pcap (m)', 'kunsat (m/s)'])
P
```