4.1 Equation and soil properties

a) Obtain the equation for the pressure head

Richard's Equation:

$$\varphi \frac{\partial \theta(p)}{\partial t} = \nabla \cdot (k(p) \nabla (p - \rho g z))$$

first we define our pressure head $h = \frac{p}{\rho g}$ and replace p accordingly:

$$\varphi \frac{\partial \theta(h\rho g)}{\partial t} = \nabla \cdot (K(h\rho g) \nabla (h\rho g - \rho g z))$$

now we pull out the constants ρg, define $K(h) = k(\rho g h)$ and apply the product rule:

$$\varphi \rho g \frac{\partial \theta(h)}{\partial h} \frac{\partial h}{\partial t} = \rho g \nabla \cdot (K(h) \nabla (h - z))$$

we define $C(h)$ as $\varphi \frac{\partial \theta(h)}{\partial h}$, resolve the later gradient term and devide by ρg:

$$C(h) \frac{\partial h}{\partial t} = \nabla \cdot (K(h)(\nabla h - 1))$$

and finaly we resolve the paranthesis on the right hand side:

$$C(h) \frac{\partial h}{\partial t} = \nabla \cdot (K(h) \nabla h) - \frac{\partial K(h)}{\partial z}$$

Now we have the pressure head based equation. Depending on the formulation of Richard's Equation (compare slides with wikipedia) the $\frac{\partial K}{\partial z}$ term ends up positive or negative.

b) What properties determine how fast a wetting front expands?

In the diffusion equation ($\frac{\partial \phi(\mathbf{r},t)}{\partial t} = \nabla \cdot \left[D(\phi, \mathbf{r}) \nabla \phi(\mathbf{r}, t) \right]$) the rate of dissipation depends on the diffusion coefficient. By comparing it to Richard's Equation we see that the rate of wetting depends on the permeability coefficient $K(h)$, the specific moister capacity $C(h)$ and the porosity φ. The diffusion coefficient can be compared to $\frac{K}{C\varphi}$.

c) Derive a consistent zero-flux boundary condition at $z = a$.

$$Q = -KA\nabla(h + z) \stackrel{!}{=} 0$$
$$-KA\nabla(h + z) = 0$$
$$-KA(\nabla h + 1) = 0$$

K and A won't be zero, so:

$$\nabla h = -1$$

Thus a fixed gradient of -1 equals enforces the zero-flux boundary condition.

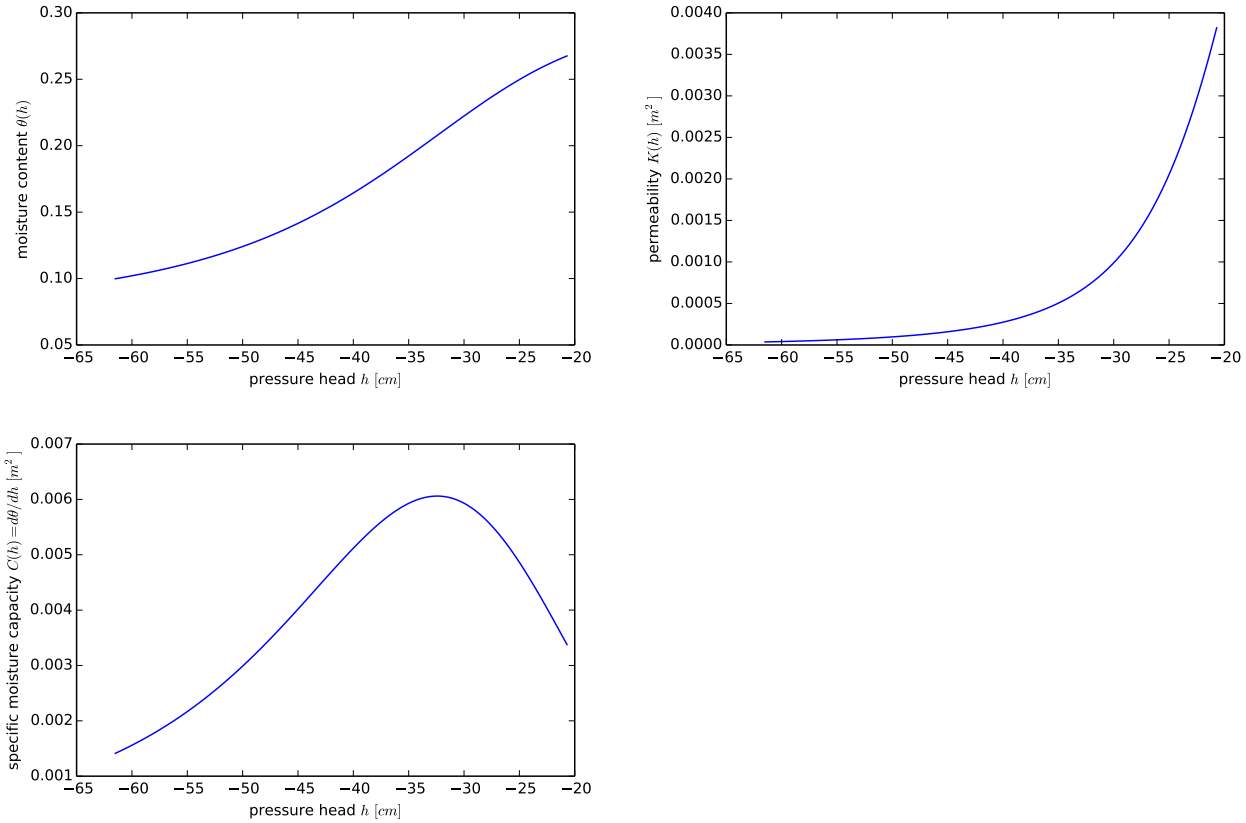


Figure 1: Figures for θ, C and K.

d) Generate figures as a function of the pressure head in the range $h \in [-20.7, -61.5]$ cm

See Figure 1.

4.2 Finite-difference discretization in space: Darcy equation for the pressure head

a) Discretize in space for node z_i

I chose to put K and h values all in the same place, so that to evaluate the K function, h does not need to be interpolated. If I had used a staggered grid scheme, only two K values would have been required to solve for one node. See Figure 2.

Starting from $C(h)\frac{\partial h}{\partial t} = \nabla \cdot (K(h)\nabla h) + \frac{\partial K(h)}{\partial z}$, we set the left term to zero (steady state) and then discretize it at the central point of z_i. The cell height is dz.

$$0 = \frac{K_{i+1/2}\left(\frac{h_{i+1}-h_i}{dz}\right) - K_{i-1/2}\left(\frac{h_i-h_{i-1}}{dz}\right)}{dz} + \frac{K_{i+1} - K_{i-1}}{2dz}$$

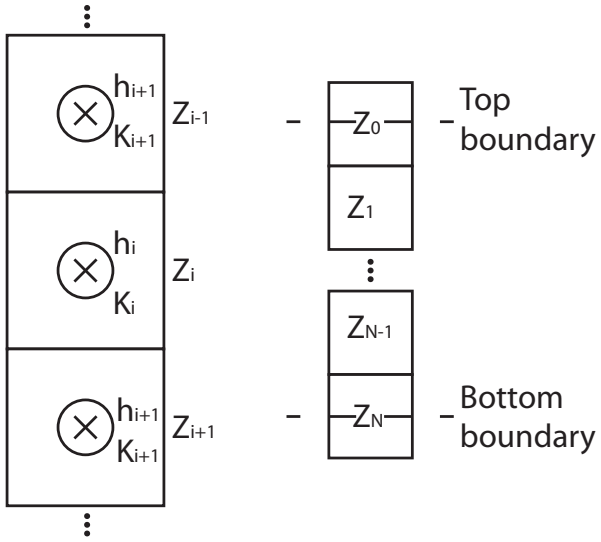


Figure 2: FD Grid

now we replace $K_{i+1/2}$ and $K_{i-1/2}$ by interpolation terms:

$$0 = \frac{\frac{K_i+K_{i+1}}{2}\left(\frac{h_{i+1}-h_i}{dz}\right) - \frac{K_i+K_{i-1}}{2}\left(\frac{h_i-h_{i-1}}{dz}\right)}{dz} + \frac{K_{i+1}-K_{i-1}}{2dz}$$

simplifying it to:

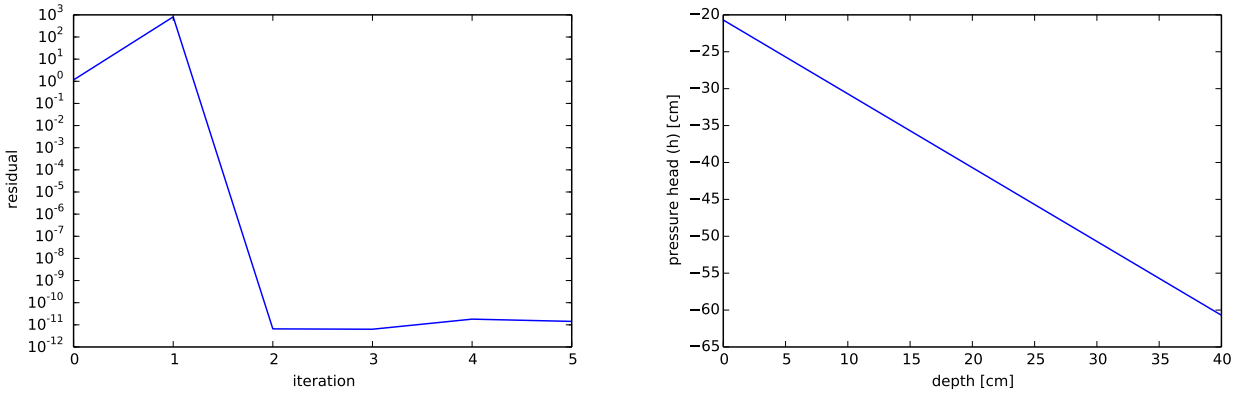
$$0 = \frac{1}{2dz^2}\left[h_{i-1}\left(K_{i-1}+K_i\right) - h_i\left(K_{i-1}+2K_i+K_{i+1}\right) + h_{i+1}\left(K_i+K_{i+1}\right) + K_{i+1}dz - K_{i-1}dz\right]$$

$$0 = h_{i-1}\left(K_{i-1}+K_i\right) - h_i\left(K_{i-1}+2K_i+K_{i+1}\right) + h_{i+1}\left(K_i+K_{i+1}\right) + dz(K_{i+1}-K_{i-1})$$

b) Write the system in matrix form. Constant pressure head at top ($h = -20.7\,\mathbf{cm}$) and impermeable bottom.

The impermeable boundary condition (Neumann) requires that $h_N = h_{N-1}$ and the constant pressure head (Dirichlet) leads to $h_0 = h$:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ c_{i-1} & c_i & c_{i+1} & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & c_{i-1} & c_i & c_{i+1} & \ddots & 0 & 0 & 0 & 0 \\ 0 & 0 & c_{i-1} & c_i & \ddots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ddots & c_i & c_{i+1} & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & c_{i-1} & c_i & c_{i+1} & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & c_{i-1} & c_i & c_{i+1} \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & -1/2 & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \\ \vdots \\ h_{N-3} \\ h_{N-2} \\ h_{N-1} \\ h_N \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} h \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} - dz \begin{bmatrix} 0 \\ K_2 - K_0 \\ K_3 - K_1 \\ K_4 - K_2 \\ \vdots \\ K_{N-2} - K_{N-4} \\ K_{N-1} - K_{N-3} \\ K_N - K_{N-2} \\ 1 \end{bmatrix}}_{b}$$



(a) Plot of residual change in regard to iterations. (b) Plot of pressure head over depth.

Figure 3: Simulation results according to 4.2.

with $c_i = -K_{i-1} - 2K_i - K_{i+1}$, $c_{i+1} = K_i + K_{i+1}$ and $c_{i-1} = K_{i-1} + K_i$. h_N referes to bottom most cell and h_0 to top most cell.

c) Implement a Picard (fixed-point) method.

```
dz = 1
h = array([-20.7]+([-61.5] * (40/dz)))
r = []
A, b = build_soe(h, dz)
r.append(sum(abs(A.dot(h)-b)))
for i in range(45):
    A, b = build_soe(h, dz)
    h_ = solve(A,b)
    r.append(sum(abs(abs(h_)-abs(h))))
    h = h_
```

`numpy.linalg.solve` uses the LAPACK library to solve $Ax = b$. `build_soe` generates the matrix A and right-hand-side b described in b) from h and can be found in Listing 1. This code reaches machine precision after less then 45 iterations for the given problem.
A plot of residual r can be found in Figure 3a.

c) Compute the steady pressure-head distribution and show it in a figure as a function of depth $b - z$. Interpret the result.

See Figure 3b for plot. We see that the pressure head increases, due to gravitational effects linearly to fulfill the Neumann boundary condition of $\nabla h = -1$ at the bottom.

Listing 1: Matrix A and vector b creation routine.

```
def build_soe(h0, dz=1.0):
    a = 0.0 # cm
    b = 40.0 # cm
    N = int((b-a)/dz)
    h = -20.7 # cm (for top boundary condition)

    if type(h0) is float:
        h0 = [h0] * (N+1)

    mat = []
    for i in range(0, N+1):
        row = [0.0] * (N+1)

        if i == N+0:
            # boundary condition! (bottom)
            row[-1] = 1.0
            row[-2] = -1.0
        elif i == 0:
            # boundary condition! (top)
            row[0] = -1.0
        else:
            row[i] = -K(h0[i-1])-2.0*K(h0[i])-K(h0[i+1]) # c_i
            row[i-1] = K(h0[i])+K(h0[i-1]) # c_{i-1}
            row[i+1] = K(h0[i+1])+K(h0[i]) # c_{i+1}

        mat.append(row)
    mat = array(mat)

    vec = [0.0] * (N+1)
    for i in range(0, N+1):
        if i == N+0:
            # boundary condition! (bottom)
            vec[i] = -dz
        elif i == 0:
            # boundary condition! (top)
            vec[i] = -h
        else:
            # -dz*(K_{i+1}-K_{i-1})
            vec[i] = -dz*(-K(h0[i-1])+K(h0[i+1]))
    vec = array(vec)

    return mat, vec
```

4.3 Numerical solution of Richards equation

a) When using the explicit Euler method. Argue of what order this restriction is. Does it change with time?

The explicit Euler method (or Forward Euler) is a very stiff problem and thus requires that the CFL condition is fulfilled:

$$\frac{u\Delta t}{\Delta x} \leq C_{\max}$$

with $C_{\max} = 1/2$.

To estimate the magnitude of velocity u, we can use the kinematic viscosity:

$$u = \frac{\nu}{\Delta x}$$

which in turn gives us:

$$\Delta t \leq \frac{\Delta x^2}{2\nu}$$

b) Develop a solver for the h-based Richard's equation with the implicit Euler method.

Now we take the time derivative back into the equation:

$$C(h)\frac{\partial h}{\partial t} = \nabla \cdot (K(h)\nabla h) + \frac{\partial K(h)}{\partial z}$$

C_i is reated the same as K_i is, with $C_i = \frac{\alpha\beta h(\theta_r - \theta_s)|h_i^n|^{\beta-2}}{(\alpha+|h_i^n|^\beta)^2}$.
We extend the above found steady-state discretization for the time-step n to $n+1$:

$$C_i\frac{h_i^{n+1} - h_i^n}{dt} = \frac{1}{2dz^2}\left[h_{i-1}^{n+1}c_{i-1} - h_i^{n+1}c_i + h_{i+1}^{n+1}c_{i+1}\right] + \frac{1}{2dz}\left[(K_{i+1} - K_{i-1})\right]$$

multiplying by dt and moving everything to the right:

$$0 = -C_i h_i^{n+1} + \frac{dt}{2dz^2}\left[h_{i-1}^{n+1}c_{i-1} - h_i^{n+1}c_i + h_{i+1}^{n+1}c_{i+1}\right] + \frac{dt}{2dz}\left[(K_{i+1} - K_{i-1})\right] + C_i h_i^n$$

$$0 = \frac{dt}{2dz^2}\left[h_{i-1}^{n+1}c_{i-1} - h_i^{n+1}\left(\frac{2dz^2}{dt}C_i + c_i\right) + h_{i+1}^{n+1}c_{i+1}\right] + \frac{dt}{2dz}\left[(K_{i+1} - K_{i-1})\right] + C_i h_i^n$$

$$0 = \left[h_{i-1}^{n+1}c_{i-1} - h_i^{n+1}\left(\frac{2dz^2}{dt}C_i + c_i\right) + h_{i+1}^{n+1}c_{i+1}\right] + dz\left[(K_{i+1} - K_{i-1})\right] + \frac{2dz^2}{dt}C_i h_i^n$$

$$\left[h_{i-1}^{n+1}c_{i-1} - h_i^{n+1}\left(\frac{2dz^2}{dt}C_i + c_i\right) + h_{i+1}^{n+1}c_{i+1}\right] = -dz\left[(K_{i+1} - K_{i-1})\right] - \frac{2dz^2}{dt}C_i h_i^n$$

This can now be use to create a matrix and right-hand-side with both (Dirichlet) boundary conditions ($h_{top} = -20.7$ and $h_{bottm} = 61.5$). The grid looks just like before.

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ c_{i-1} & c_i & c_{i+1} & \dots & 0 & 0 & 0 \\ 0 & c_{i-1} & c_i & \ddots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ddots & c_i & c_{i+1} & 0 \\ 0 & 0 & 0 & \dots & c_{i-1} & c_i & c_{i+1} \\ 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} h_0^{n+1} \\ h_1^{n+1} \\ h_2^{n+1} \\ \vdots \\ h_{N-2}^{n+1} \\ h_{N-1}^{n+1} \\ h_N^{n+1} \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} h_{top} \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ h_{bottom} \end{bmatrix} - dz \begin{bmatrix} 0 \\ K_2 - K_0 \\ K_3 - K_1 \\ \vdots \\ K_{N-1} - K_{N-3} \\ K_N - K_{N-2} \\ 0 \end{bmatrix} - \frac{2dz^2}{dt} C_i \begin{bmatrix} 0 \\ h_1^n \\ h_2^n \\ \vdots \\ h_{N-2}^n \\ h_{N-1}^n \\ 0 \end{bmatrix}}_{b}$$

with $c_i = -K_{i-1} - 2K_i - K_{i+1} - \frac{2dz^2}{dt} C_i$, $c_{i+1} = K_i + K_{i+1}$ and $c_{i-1} = K_{i-1} + K_i$. h_N refers to bottom most cell and h_0 to top most cell.

c) Implement this method and solve Richards' equation

With $t_{\text{end}} = 360\,\text{s}$, $\Delta z = 1$, $h(a,t) = -61.5\,\text{cm}$ and $h(b,t) = -20.7\,\text{cm}$. For plots see Figure 4

The changes in `build_soe` were very straight forward and can easily be reproduced using the modified described matrix described in 4.3 c) and Listing 1. The time iteration code can be found in Listing 2.

d) Is there the same amount of water in the domain in all cases?

If we integrate $\theta(z, 360\,\text{s})$ we see that the amount of water in the domain is slightly different: $\int \theta_{\Delta t=10\,\text{s}}(z, 360\,\text{s})dz = 5.482$, $\int \theta_{\Delta t=30\,\text{s}}(z, 360\,\text{s})dz = 5.478$, $\int \theta_{\Delta t=120\,\text{s}}(z, 360\,\text{s})dz = 5.508$.

4.4 Mass Conservation

$\text{MB}(t^{n+1}) = \frac{\text{total additional mass in the domain}}{\text{total net flux into the domain}}$

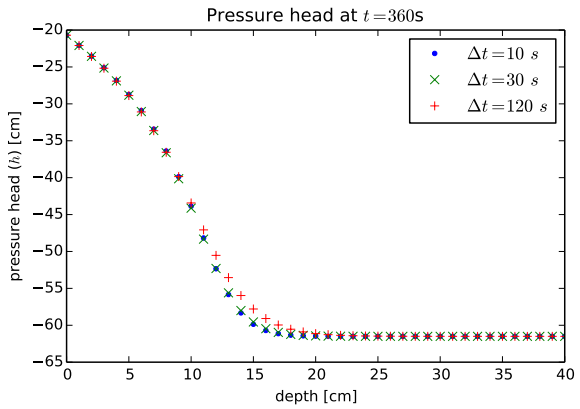
a) In the case that the pressure head is fixed at top an bottom, show $\text{MB}(t^{n+1})$

The total mass in the domain can be found by integrating $\theta(z,t)$ over z, since we have discrete θ values, we can use a sum over the internal values:

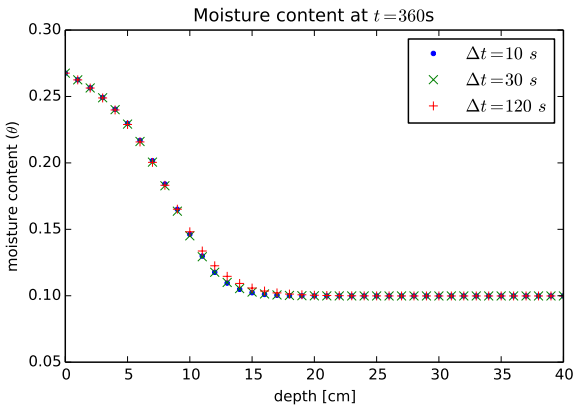
$$m(t) = \int \theta(z,t)dz = \sum_{i=0}^{N-1} \theta_i^t \Delta z$$

to get the change from $t = 0$ to $t = n + 1$ we do the following:

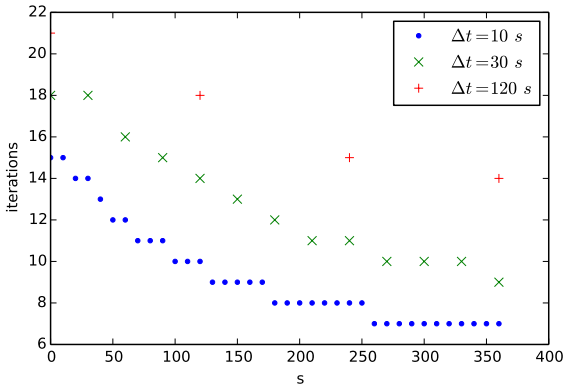
$$\Delta m = m_{n+1} - m_0 = \sum_{i=0}^{N-1} \theta_i^{n+1} \Delta z - \sum_{i=0}^{N-1} \theta_i^0 \Delta z = \sum_{i=0}^{N-1} (\theta_i^{n+1} - \theta_i^0) \Delta z$$



(a) Plot of pressure head over depth.



(b) Plot of moisture content over depth.



(c) Plot of required iterations per timestep.

Figure 4: Simulation results according to 4.3 c).

Listing 2: Implicit euler time-step simulation.

```
def simulate(dz=1, dt=10.0, max_iter=50, eps=10e-7):
    h = array([-20.7]+([-61.5] * (40/dz)))
    h_timeline = []
    r = []
    iters = []
    steps = []
    t = 0
    while t <= 360:
        h_old = h
        A, b = build_soe(h, h_old, dz=dz, dt=dt)
        r = sum(abs(A.dot(h)-b))

        iterations = 0
        while r > eps:
            h_ = solve(A,b) # from numpy.linalg
            r = sum(abs(abs(h_)-abs(h)))
            h = h_
            A, b = build_soe(h, h_old=h_old, dz=dz, dt=dt)

            iterations += 1
            if iterations > max_iter:
                print "did not converge"
                break

        if iterations > max_iter:
            print "stopped at t=",t,'s'
            break

        h_timeline.append(h)
        iters.append(iterations)
        steps.append(t)
        t += dt

    return h_timeline, iters, steps
```

This is equivalent to the given numerator of $\mathrm{MB}(t^{n+1})$.

Now to the denominator. To get the total net flux into the domain, we need to calculate the fluxes at both edges at a time step t:

$$Q^t = (K^t \nabla(h^t + z))_{\text{at bottom}} - (K^t \nabla(h^t + z))_{\text{at top}}$$

$$Q^t = K^t_{N-1/2} \left(\frac{h^t_{N-1} - h^t_{N-1}}{\Delta z} + 1 \right) - K^t_{1/2} \left(\frac{h^t_1 - h^t_0}{\Delta z} + 1 \right)$$

to get from here to the total flux we need to integrate again:

$$Q^{\text{tot}} = \int_0^{t^{n+1}} Q^t dt = \sum_{t=0}^{t=t^{n+1}} Q^t \Delta t$$

$$\mathrm{MB}(t^{n+1}) = \frac{\Delta m}{Q^{\text{tot}}}$$

b) Is the scheme developed in the previous task mass-conservative?

In Figure 5 we can clearly see that the scheme is not mass conserving. I get's very close (95 % mass balance ratio) with $\Delta t = 1\,\mathrm{s}$, but decreases rapidly with increased time steps. I added more Δts to be able to show a better curve.

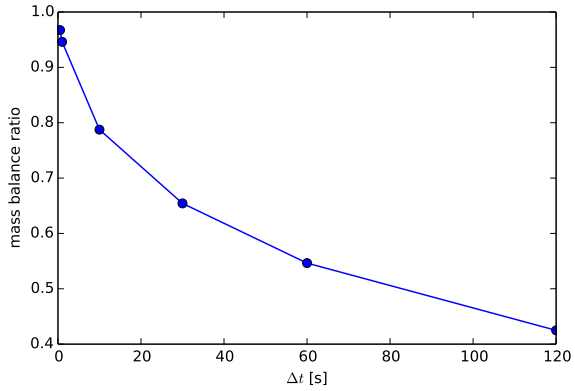


Figure 5: Mass balance in regard to time-step length

4.5 Mixed Richards' equation: Improved Picard scheme

a)

We start with the given mixed-form equation:

$$\frac{\theta^{n+1,m+1} - \theta^n}{\Delta t} - \nabla \cdot \left(K^m \nabla h^{m+1} \right) - \frac{\partial K^m}{\Delta z} = 0$$

now replace $\theta^{n+1,m+1}$ with the taylor expansion truncated at first order:

$$\theta^{n+1,m+1} = \theta^{n+1,m} + C(h)^{n+1,m} \underbrace{(h^{n+1,m+1} - h^{n+1,m})}_{\delta h}$$

and get:

$$\frac{\theta^{n+1,m} + C(h^{n+1,m})\delta h - \theta^n}{\Delta t} - \nabla \cdot \left(K^m \nabla h^{m+1}\right) - \frac{\partial K^m}{\Delta z} = 0$$

$$\frac{C(h^{n+1,m})\delta h}{\Delta t} = \nabla \cdot \left(K^{n+1,m} \nabla h^{n+1,m+1}\right) + \frac{\partial K^{n+1,m}}{\Delta z} - \frac{\theta^{n+1,m} - \theta^n}{\Delta t}$$

Using finite difference method with the same grid as in task 4.2:

$$C_i^{n+1,m} \frac{(h_i^{n+1,m+1} - h_i^{n+1,m})}{dt} =$$
$$\frac{1}{2dz^2}\left[h_{i-1}^{n+1,m+1}\left(K_{i-1}^{n+1,m+1} + K_i\right) - h_i^{n+1,m+1}\left(K_{i-1}^{n+1,m} + 2K_i^{n+1,m} + K_{i+1}^{n+1,m}\right) + h_{i+1}^{n+1,m}\left(K_i^{n+1,m} + K_{i+1}^{n+1,m}\right)\right] +$$
$$\frac{K_{i+1}^{n+1,m} - K_{i-1}^{n+1,m}}{2dz} - \frac{\theta_i^{n+1,m} - \theta_i^n}{dt}$$

with $c_{i-1} = K_{i-1}^{n+1,m} + K_i$, $c_i = -K_{i-1}^{n+1,m} - 2K_i^{n+1,m} - K_{i+1}^{n+1,m} - \frac{C_i^{n+1,m+1} 2dz^2}{dt}$ and $c_{i+1} = K_i^{n+1,m} + K_{i+1}^{n+1,m}$ we get:

$$-C_i^{n+1,m} \frac{h_i^{n+1,m}}{dt} = \frac{1}{2dz^2}\left[h_{i-1}^{n+1,m+1} c_{i-1} + h_i^{n+1,m+1} c_i + h_{i+1}^{n+1,m+1} c_{i+1})\right] + \frac{K_{i+1}^{n+1,m} - K_{i-1}^{n+1,m}}{2dz} - \frac{\theta_i^{n+1,m} - \theta_i^n}{dt}$$

$$\frac{1}{2dz^2}\left[h_{i-1}^{n+1,m+1} c_{i-1} + h_i^{n+1,m+1} c_i + h_{i+1}^{n+1,m+1} c_{i+1})\right] = -\frac{K_{i+1}^{n+1,m} - K_{i-1}^{n+1,m}}{2dz} + \frac{\theta_i^{n+1,m} - \theta_i^n}{dt} - C_i^{n+1,m} \frac{h_i^{n+1,m}}{dt}$$

multiplying by $2dz^2$:

$$\left[h_{i-1}^{n+1,m+1} c_{i-1} + h_i^{n+1,m+1} c_i + h_{i+1}^{n+1,m+1} c_{i+1})\right] =$$
$$-dz(K_{i+1}^{n+1,m} - K_{i-1}^{n+1,m}) + \frac{2dz^2}{dt}\left(\theta_i^{n+1,m} - \theta_i^n - C_i^{n+1,m} h_i^{n+1,m}\right)$$

We see that the equation is mostly the same as in Task 4.3, the only difference being that the right-hand-side is extended by the term $\frac{2dz^2}{dt}(\theta_i^{n+1,m} - \theta_i^n)$. This term will ensure the mass conservation.

b) Repeat Task 4.3 c) above using the new method.

See Figure 6.

I would have expected better mass conservation with the mixed form, but I suppose that small time steps are already very good at conserving mass and too large time steps are just too extreme.

c) What solver would you recommend? Why?

I would use the mixed form, since it is as computationally expensive as the h-based one, but provides better mass conservation for larger time steps.

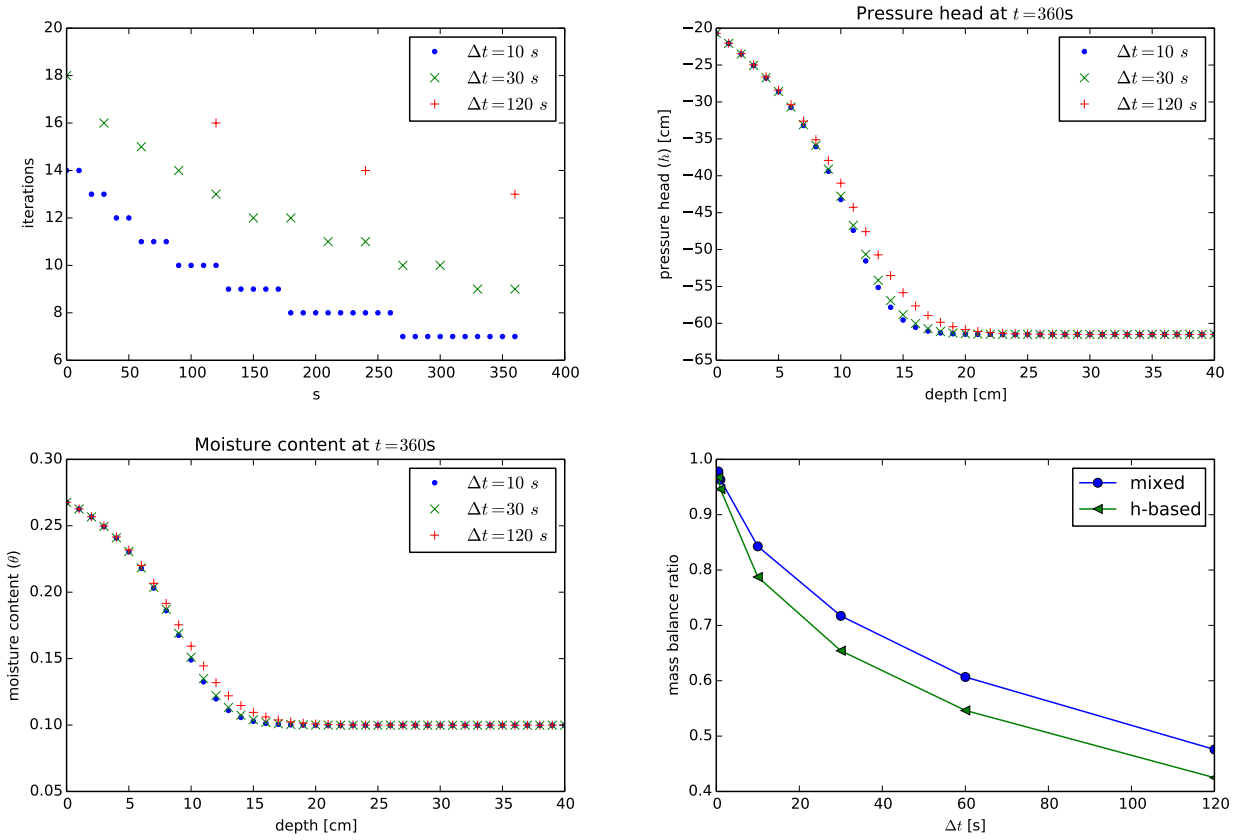


Figure 6: Simulation results with mixed-form equation.