Isotope Tracking

Applications

After a new product application at time $t+1$, the top soil carbon isotope signature ($\delta^{13}C$) for a soil layer k, is updated by balancing mass terms. Note that for pesticide applications only the first layer ($k = z0$) is considered, such that:

$$\delta^{13}C_{k(t+1)} = \frac{1}{M_{k,tot(t+1)}} \left(\delta^{13}C_{k(t)} \cdot M_{k(t)} + \delta^{13}C_{app(t+1)} \cdot M_{app(t+1)} \right)$$

$$M_{k,tot(t+1)} = M_{k(t)} + M_{app(t+1)}$$

where $M_{k(t)}$ is the pesticide mass (μg) for the layer k present before application *app*.

Non-reactive transport

For each non-fractionating mass transfer process $\delta^{13}C$ is updated also by balancing mass terms for each cell:

$$\delta^{13}C_{k(t+1)} = \frac{1}{M_{k,tot(t+1)}} \left(\delta^{13}C_{k(t)} \cdot M_{k(t)} + \delta^{13}C_{gain(t+1)} \cdot M_{gain(t+1)} - \delta^{13}C_{loss(t+1)} \cdot M_{loss(t+1)} \right)$$

$$M_{k,tot(t+1)} = M_{k(t)} + M_{gain(t+1)} - M_{loss(t+1)}$$

Update at each cell for each layer is computed by the follwing function, where the relevant layer and processes is selected:

```
def update_layer_delta(model, layer, process, mass_process, mass_before_transport):
    if layer == 0:
      delta_layer = model.delta_z0
      delta_layer_above = None
      mass_layer = model.pestmass_z0
    elif layer == 1:
        delta_layer = model.delta_z1
        delta_layer_above = model.delta_z0
        mass_layer = model.pestmass_z1
    elif layer == 2:
        delta_layer = model.delta_z2
        delta_layer_above = model.delta_z1
        mass_layer = model.pestmass_z2

    if process == "volat":
        pass
    elif process == "runoff":
        pass
    elif process == "leach":
        pass
    elif process == "latflux":
        pass
    else:
        raise NotImplementedError
```

```
    return "updated delta for delta_layer"
```

For brevity, only two examples are shown. For volatilization, the process process=**"volat"** is chosen and the updated $\delta^{13}C$ for a soil layer k is returned as,

```
def update_layer_delta(model, layer, process, mass_process, mass_before_transport):
    if layer == 0:
      delta_layer = model.delta_z0
      delta_layer_above = 0
      mass_layer = model.pestmass_z0
    elif layer == 1:
        delta_layer = model.delta_z1
        delta_layer_above = model.delta_z0
        mass_layer = model.pestmass_z1
    elif layer == 2:
        delta_layer = model.delta_z2
        delta_layer_above = model.delta_z1
        mass_layer = model.pestmass_z2

    if process == "volat":
        mass_loss = mass_process["mass_loss"]
        mass_gain = 0
        delta_gain = 0
        delta_loss = delta_layer

    elif process == "runoff":
        pass
    elif process == "leach":
        pass
    elif process == "latflux":
        pass
    else:
        raise NotImplementedError
    if process == "latflux":
        pass
    else:
        mass_tot = mass_before_transport + mass_gain - mass_loss
        delta_int = ((1/mass_tot) *
                     (delta_layer * mass_before_transport +  # initial
                      delta_gain * mass_gain -  # mass_in
                      delta_loss * mass_loss))  # mass_out
    return delta_int
```

For lateral flux, update follows the same approach as that for water mass exchange across cells. The pesticide mass at cell j after lateral flux is given by:

$$M_{j,tot(t+1)} = M_{j(t)} + \sum_{i=1}^{N(t)} M_{loss,i(t)} - M_{loss,j(t)}$$

The mass gain at cell j is given by the mass from upstream cells i contributing to downstream cells and given by:

$$\sum_{i=1}^{N(t)} M_{loss,i(t)} = \frac{W_j \sum_{i=1}^{N(t)} max[C_{i,aq} \cdot \Big(c_z(SW_i - SW_{fc,i})\Big),\ 0]}{\sum_{i=1}^{N(t)} W_i}$$

Loss at cell j is then given by:

$$M_{loss,j(t)} = C_{j,aq} \cdot \Big(c_z(SW_j - SW_{fc,j})\Big)$$

Considering the isotope mass balance, while simplfying for the relative wetness index at cell j, we obtain:

$$W_{j/i} = \frac{W_j}{\sum_{i=1}^{N(t)} W_i}$$

$$\delta^{13}C_{j(t+1)} = \frac{1}{M_{j,tot(t+1)}} \Big(\delta^{13}C_{j,(t)} \cdot M_{j(t)} + W_{j/i} \sum_{i=1}^{N(t)} max[\delta^{13}C_i \cdot C_{i,aq} \cdot \Big(c_z(SW_i - SW_{fc,i})\Big),\ 0] - \delta^{13}C_{j(t)} \cdot M_{loss,j(t)}\Big)$$

The *update_layer_delta* functions, takes process=**"latflux"** and implements the above equations as:

```
def update_layer_delta(model, layer, process, mass_process, mass_before_transport):
    if layer == 0:
      delta_layer = model.delta_z0
      delta_layer_above = 0
      mass_layer = model.pestmass_z0
    elif layer == 1:
        delta_layer = model.delta_z1
        delta_layer_above = model.delta_z0
        mass_layer = model.pestmass_z1
    elif layer == 2:
        delta_layer = model.delta_z2
        delta_layer_above = model.delta_z1
        mass_layer = model.pestmass_z2

    if process == "volat":
        pass
    elif process == "runoff":
        pass
    elif process == "leach":
        pass
    elif process == "latflux":
        mass_latflux = mass_process["net_mass_latflux"]  # mg
        mass_loss = mass_process["cell_mass_loss_downstream"]
        mass_gain = mass_process["upstream_mass_inflow"]
        mass_tot = mass_before_transport + mass_gain - mass_loss
        # Proof of first fraction, i.e., f1 > 1
        f1 = mass_before_transport / mass_tot
        # Proof of 2nd (inflow) fraction, f2 < 1
        # f2 = accuflux(model.ldd, mass_gain)/accuflux(model.ldd, mass_tot)
        f3 = mass_loss / mass_tot  # Proof of third (leaving) mass fraction, f3 < 1
    else:
        raise NotImplementedError
    if process == "latflux":
        delta2_f2 = accuflux(model.ldd_subs, mass_gain*delta_layer)/accuflux(model.ldd_subs, mass_tot)
```

```
        delta_int = (delta_layer * f1) + delta2_f2 - (delta_layer * f3)
    else:
        pass

    return delta_int
```

Full function

This is my change.

```
def update_layer_delta(model, layer, process, mass_process, mass_before_transport):
    if layer == 0:
      delta_layer = model.delta_z0
      delta_layer_above = 0
      mass_layer = model.pestmass_z0
    elif layer == 1:
        delta_layer = model.delta_z1
        delta_layer_above = model.delta_z0
        mass_layer = model.pestmass_z1
    elif layer == 2:
        delta_layer = model.delta_z2
        delta_layer_above = model.delta_z1
        mass_layer = model.pestmass_z2
    if process == "volat":
        mass_loss = mass_process["mass_loss"]
        mass_gain = 0
        delta_gain = 0
        delta_loss = delta_layer
    elif process == "runoff":
        mass_loss = mass_process["mass_runoff"]
        mass_gain = 0
        delta_gain = 0
        delta_loss = delta_layer
    elif process == "leach":
        mass_leached = mass_process["mass_leached"]  # mg
        mass_loss = mass_leached
        mass_gain = 0
        delta_gain = delta_layer_above
        delta_loss = delta_layer
    elif process == "latflux":
        mass_latflux = mass_process["net_mass_latflux"]  # mg
        mass_loss = mass_process["cell_mass_loss_downstream"]
        mass_gain = mass_process["upstream_mass_inflow"]
        mass_tot = mass_before_transport + mass_gain - mass_loss
        # Proof of first fraction, i.e., f1 > 1
        f1 = mass_before_transport / mass_tot
        # Proof of 2nd (inflow) fraction, f2 < 1
        # f2 = accuflux(
        #   model.ldd, mass_gain)/accuflux(model.ldd, mass_tot)
        # Proof of third (leaving) mass fraction, f3 < 1
        f3 = mass_loss / mass_tot
    else:
        raise NotImplementedError
    if process == "latflux":
```

```python
        delta2_f2 = accuflux(model.ldd_subs, mass_gain*delta_layer)/accuflux(model.ldd_subs, mass_tot)
        delta_int = (delta_layer * f1) + delta2_f2 - (delta_layer * f3)
    else:
        mass_tot = mass_before_transport + mass_gain - mass_loss
        delta_int = ((1/mass_tot) *
                     (delta_layer * mass_before_transport +  # initial
                      delta_gain * mass_gain -  # mass_in
                      delta_loss * mass_loss))  # mass_out
    # return {"delta_int": delta_int, "mass_layer": mass_layer}
    return delta_int
```

References