Discussion paper: variable naming schemes

May 22, 2014

Scheme A: pv, cv, and the likes

How

```
FV/cells: cv(...)
FV/block: pv(...)
LB/cells: cv(...)
```

Pro

- short and easy to remember
- this is the way the variables are actually used
- cv/pv is consistent in the FV block (where it is most relevant)

Con

- `cv` might be misleading for FV when using species etc.
- `cv` is probably very confusing for LBM novices

Scheme B: variable

How

```
FV/cells: variable(...)
FV/block: pv(...)
LB/cells: variable(...)
```

Pro

- generic names match anything
- for outside access it is clear which method to call if accessing variables (unified access)

Con

- generic names are not self-descriptive
- unified access can easily be achieved through traits class as well
- for e.g. DG block the concept of 1 variable/cell breaks down anyways

Scheme C: Q

How

```
FV/cells: Q(...)
FV/block: P(...)
LB/cells: macroscopic(...)
```

Pro

- very short and to the point
- macroscopic is maybe a more precise name in the LBM context

Con

- very short names make it harder to grep
- `Q` might be not 100% correct whn using species

Scheme D: mixit

How

```
FV/cells: state(...)
FV/block: pv(...)
LB/cells: macroscopic(...)
```

Pro

- all names are as restrictive as possible (but not more)

Con

- no apparent connection between the names

Note on DG cells

DG cells (called *elements*) have more than one DOF, so it is hard to generalize variable access to the DG block. In any case, `cv` is probably the most apt name for the variables, since the used DG method only solves PDEs that can be written in conservative form.

If really taking into account the DG block, the best interface would be one that does not return a value for a cell, but for a coordinate (e.g. `var_at_x(x, v)`.