S10_Model_Hybridization

March 19, 2021

1 Hybridizing Models in VESIcal

One of the advantages of implementing the solubility models in a generic python module is the flexibility this affords the user in changing the way solubility models are defined and used. In particular, the structure allows any combination of pure-fluid models to be used together in modelling mixed-fluids, and fugacity or activity models can be quickly changed without modifying code. This allows advanced users to see how changing a fugacity or activity model implemented in any particular solubility model would affect model results.

To run this notebook, first VESIcal must be imported and a sample defined:

[1]:
```
import sys
sys.path.append('../../../../')

import VESIcal as v

mysample = v.Sample({'SiO2':  77.3,
             'TiO2':   0.08,
             'Al2O3': 12.6,
             'Fe2O3':  0.207,
             'Cr2O3':  0.0,
             'FeO':    0.473,
             'MnO':    0.0,
             'MgO':    0.03,
             'NiO':    0.0,
             'CoO':    0.0,
             'CaO':    0.43,
             'Na2O':   3.98,
             'K2O':    4.88,
             'P2O5':   0.0,
             'H2O':    6.5,
             'CO2':    0.05})
```

1.1 Using Model objects directly

The calculations shown in this manuscript utilise the python-class Calculation interfaces. When the class is called, the required model is usually selected from the default models using the model name as a string, e.g.:

[2]:
```
calculation = v.calculate_dissolved_volatiles(sample=mysample, pressure=1000.0,␣
↪X_fluid=0.1, model='ShishkinaIdealMixing')
```

```
../../../../VESIcal/calculate_classes.py:52: RuntimeWarning: SiO2 (77.3 wt%)
exceeds the upper limit of 65.0 wt% suggested by Shishkina et al. for their H2O
model. The Shishkina et al. (2014) model should be used to model mixed fluids
with caution. The mixed fluid model does not recreate the experimental
observations. SiO2 (77.3 wt%) is outside the calibration range of Shishkina et
al. carbon (calculated from the max and minimum concentrations in the
calibration dataset +-5%; 40.0-57.0 wt%).
  w.warn(self.calib_check,RuntimeWarning)
```

When the `calculate_dissolved_volatiles` class is initiated, it retrieves a pre-defined model object instance. However, creating model objects directly affords greater control over how the calculation is performed. A model object for a pure fluid can be created by:

[3]:
```
model_object = v.models.shishkina.carbon()
```

Any method that is used during solubility calculations can now be accessed directly. For example, the compositional dependence of CO_2 solubility is captured by the π^* parameter in the –SHISHKINA– parameterisation. The value of this parameter is calculated everytime a solubility calculation is performed using the `ShishkinaCarbon` model, but is not accessible through the `Calculation` class interfaces. However, the method that calculates π^* can be called directly from the model object:

[4]:
```
model_object.PiStar(sample=mysample)
```

[4]:
```
0.11957696507035985
```

The available methods can be found when using Jupyterlab or ipython by pressing the tab key after typing `model_object.`. Calculation methods can also be called directly from the model object, without using the `Calculation` class interface:

[5]:
```
model_object.calculate_dissolved_volatiles(sample=mysample, pressure=1000.0)
```

[5]:
```
0.011596717182072776
```

This is computationally faster than using the `Calculation` interface, but does not automatically pre-process the sample composition, or run calibration checks. Alternatively, the `model_object` can be used with the `Calculation` class interface by passing the object in place of a string for the `model` variable:

[6]:
```
calculation = v.calculate_dissolved_volatiles(sample=mysample, pressure=1000.0,␣
↪model=model_object)
calculation.result
```

```
../../../../VESIcal/calculate_classes.py:52: RuntimeWarning: SiO2 (77.3 wt%) is
outside the calibration range of Shishkina et al. carbon (calculated from the
```

```
max and minimum concentrations in the calibration dataset +-5%; 40.0-57.0 wt%).
  w.warn(self.calib_check,RuntimeWarning)
```

[6]: 0.011596717182072776

1.2 Changing Model fugacity and activity models

This functionality is more powerful when the user makes changes to components of the model. For example, when every model object is initialized in VESIcal, it has a fugacity and activity model associated with it. Where models parameterise solubility as a function of pressure (or partial pressure) directly, as done by –SHISHKINA–, this is equivalent to assuming the fugacity is that of an ideal gas. By retrieving the fugacity model from the `model_object` we created above, we can see that this is the case:

[7]:
```
model_object.fugacity_model
```

[7]: <VESIcal.fugacity_models.fugacity_idealgas at 0x7f85af65cc10>

Other models, e.g., –DIXON–, parameterise solubility as a function of fugacity, calculated using an equation of state for the vapour phase. The default fugacity model for `DixonCarbon` is the –KERRICK AND JACOBS–, and is set when the model is initialized:

[8]:
```
model_object = v.models.dixon.carbon()
model_object.fugacity_model
```

[8]: <VESIcal.fugacity_models.fugacity_MRK_co2 at 0x7f85af601ed0>

However, if we wanted to see how the calculation results would change were the –REDLICH-KWONG– model used instead, we can change this component of the model:

[9]:
```
model_object.set_fugacity_model(v.fugacity_models.fugacity_RK_co2())
model_object.fugacity_model
```

[9]: <VESIcal.fugacity_models.fugacity_RK_co2 at 0x7f85af606090>

Any calculations now performed using `model_object` will use fugacities calculated with –REDLICH-KWONG– in place of –KERRICK AND JACOBS–. Each model object also has an activity model associated with it. This allows for non-ideal solution of vapour species in the melt. Whilst none of the models presently within VESIcal use non-ideal activities, this would permit models such as –DUAN– to be implemented within the VESIcal framework in the future.

1.3 Defining and using MixedFluid model objects

The model objects for mixed fluids have a similar structure, with one major difference. A `MixedFluid` model object is a generic model which may be implemented with any of the pure-fluid models within VESIcal. The default `MixedFluid` model object for –SHISHKINA– is defined by:

[10]:
```
mixed_model = v.model_classes.MixedFluid({'CO2':v.models.shishkina.carbon(),
                                          'H2O':v.models.shishkina.water()})
```

As with the pure-fluid model objects, calculations can be performed directly using the model object, e.g.:

[11]:
```
mixed_model.calculate_equilibrium_fluid_comp(sample=mysample, pressure=1000.0)
```

[11]:
```
{'CO2': 0.006038102616413603, 'H2O': 0.9939618973835864}
```

or by supplying the `Calculate` class interface with `mixed_model` as the value of `model`:

[12]:
```
calculation = v.calculate_equilibrium_fluid_comp(sample=mysample, pressure=1000.
↪0, model=mixed_model)
calculation.result
```

```
../../../../VESIcal/calculate_classes.py:52: RuntimeWarning: SiO2 (77.3 wt%) is
outside the calibration range of Shishkina et al. carbon (calculated from the
max and minimum concentrations in the calibration dataset +-5%; 40.0-57.0
wt%).SiO2 (77.3 wt%) exceeds the upper limit of 65.0 wt% suggested by Shishkina
et al. for their H2O model.
  w.warn(self.calib_check,RuntimeWarning)
```

[12]:
```
{'CO2': 0.006038102616413603, 'H2O': 0.9939618973835864}
```

If we wanted to change the fugacity (or activity) models used in the calculation, we must access the pure-fluid model objects stored within the mixed-fluid model object:

[13]:
```
mixed_model.models[0].fugacity_model
```

[13]:
```
<VESIcal.fugacity_models.fugacity_idealgas at 0x7f85af606ad0>
```

[14]:
```
mixed_model.models[0].set_fugacity_model(v.fugacity_models.fugacity_KJ81_co2())
mixed_model.models[0].fugacity_model
```

[14]:
```
<VESIcal.fugacity_models.fugacity_KJ81_co2 at 0x7f85af606550>
```

The `MixedFluid` model object also allows different solubility models to be combined, for example if we wanted to use the –ALLISON– CO2 solubility model in conjunction with a water solubility model we could define our own `MixedFluid` model object:

[15]:
```
mixed_model = v.model_classes.MixedFluid({'CO2':v.models.allison.carbon(),
                                          'H2O':v.models.iaconomarziano.
↪water()})
```

[ ]:
```

```