Notes

Fred Phillips

August 1, 2014

1 Current State

In a typical simulation bacteria can grow with a doubling time of 1 day which matches what Charles has told us. The bacteria grow by consuming *Iron* and *O2*. As they grow they produce *biomass* and *H3O+*. The H3O+ dissolves the rock (represented by *capsule*), which releases Iron. See *Reactions* and the attached reactions table for more info.

We were expecting to see a self-limiting growth where the bugs would grow to some thickness which would prevent those at the top from receiving Iron and the ones at the bottom from receiving O2. We are not seeing this because the Iron and O2 can diffuse all the way through the biofilm, the diffusivity coefficients for the iron and O2 are known physical values that can't be changed.

1.1 Issues & Suggestions

1.1.1 Growth

The bugs continue to grow indefinitely, Charles said in the field the biofilms are observed to be only around 3 bugs thick. Must be some other mechanism stopping their growth other than diffusion of iron and O2. Figures 1-4 show how there is more than enough Iron and O2 present throughout the domain to allow the bugs to grow.

1.1.2 Rock dissolution rate

There is a general open problem of the rate of bug growth relative to the rate of rock dissolution. As can be seen in Figure 5, if the bugs grow too fast they get shoved into the rock. And if the rock dissolves too fast, the bugs can get left behind and end up floating above the rock.

I think the former is more likely to be a problem in the future, now we have the actual rate parameters nailed down. It might be solved by increasing the number of shoves per timestep in the protocol file, or just decreasing the agent timestep so not too many bugs grow at once. When fully relaxed the bugs shouldn't overlap with each other or the rock.

I don't think the floating bugs will be problem now, I think it was just an artefact of an incorrect (too high) rock dissolution rate.

1.1.3 Diffusion of Iron

Using a simple 1D reaction-diffusion solver it can be shown that a source on the boundary between a low diffusivity area, and a high diffusivity area with a zero flux boundary at the bottom and an open (concentration = 0) boundary at the top causes high concentrations to be found in the low diffusivity area, see Figure 6.

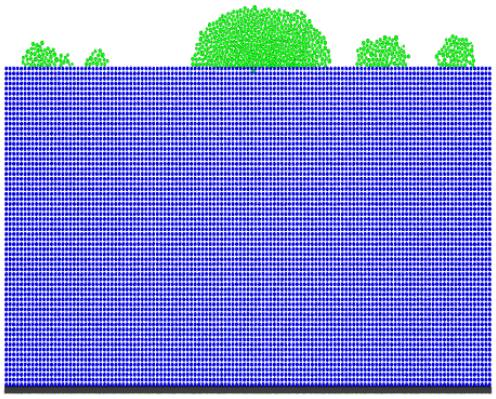

Figure 1: Bugs at timestep 35

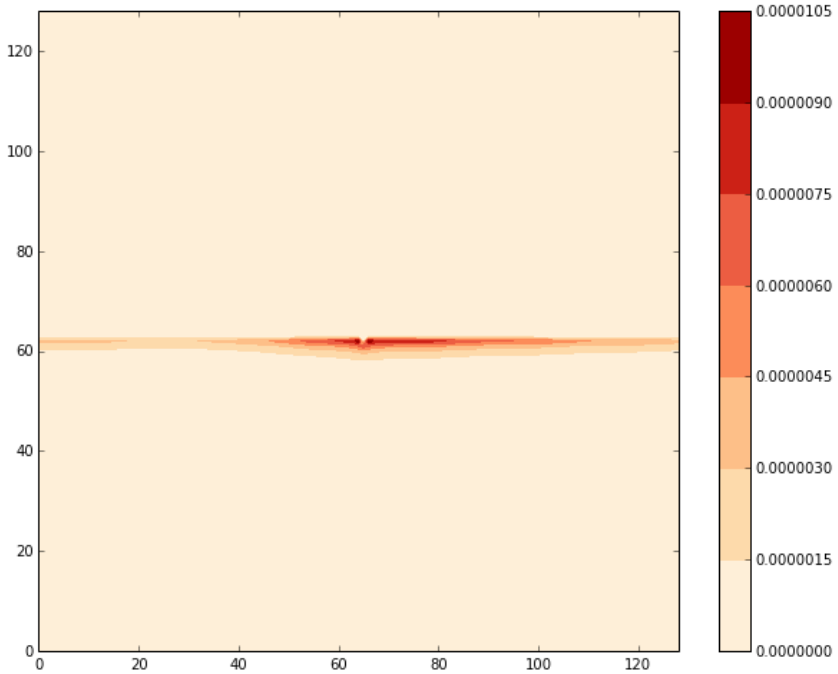

Figure 2: Iron concentration at timestep 35

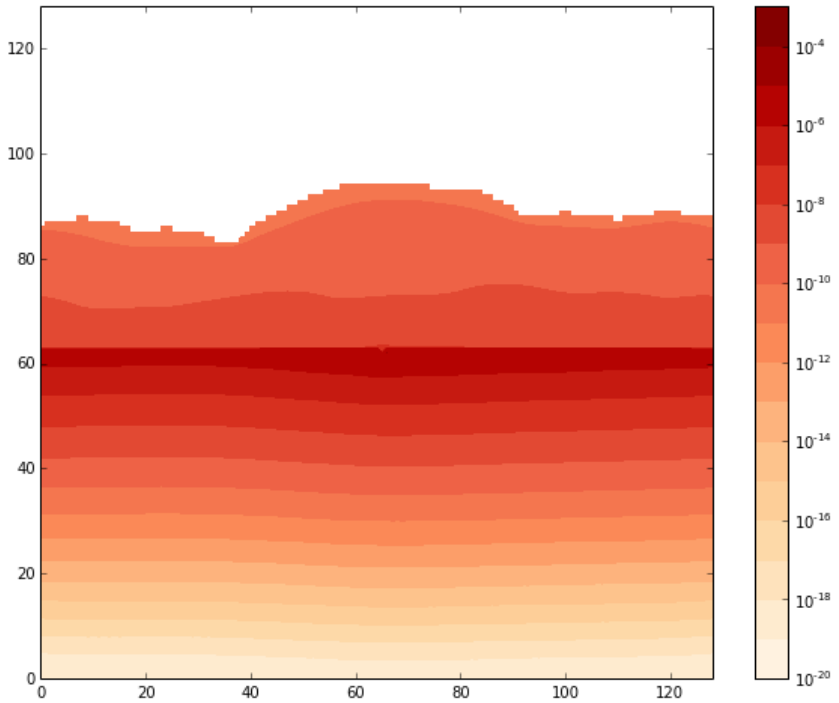

Figure 3: Log plot of Iron concentration at timestep 35

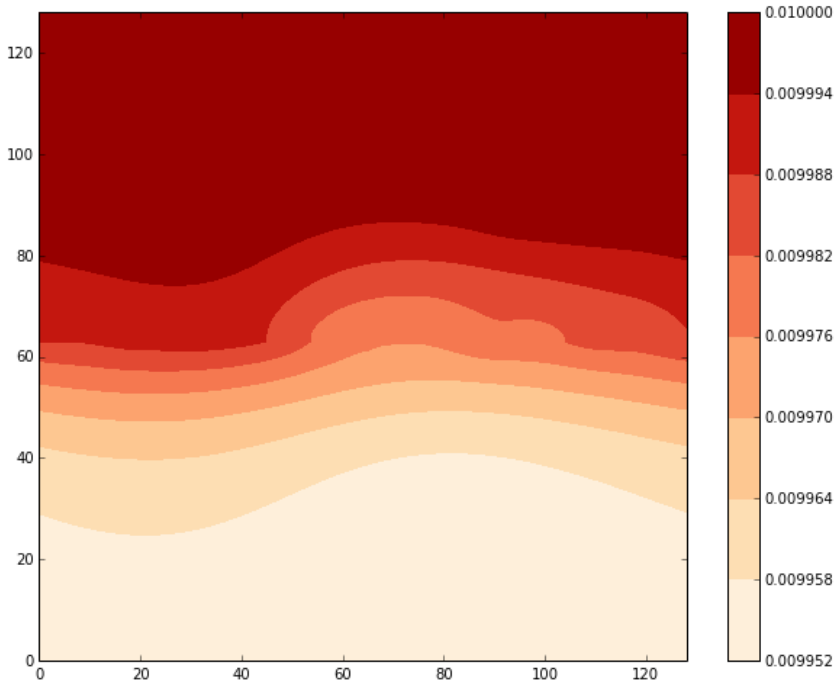

Figure 4: O2 concentration at timestep 35

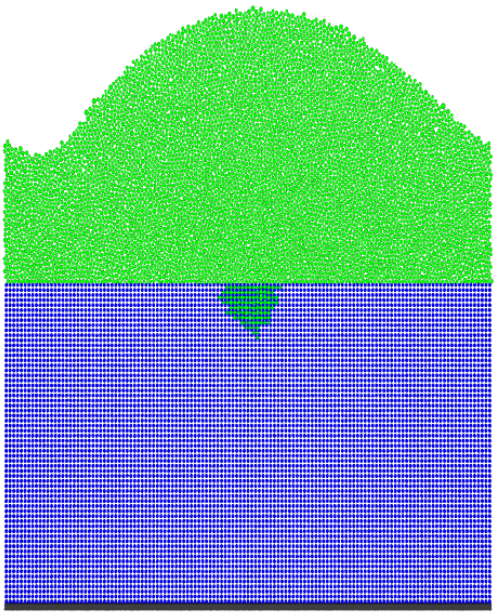

Figure 5: Bugs growing too fast and overlapping with rock

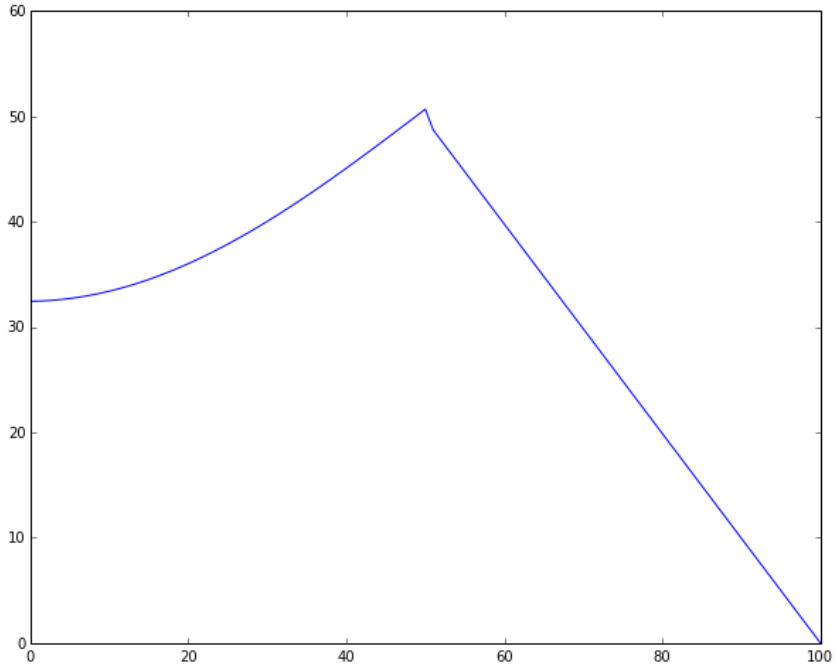

Figure 6: Diffusion of concentration when point source is located just inside ($x = 50$) an area of low diffusion ($x = 1$ to 50)

1.1.4 Self-limiting growth & Diffusion solver

We think that the limiting growth problem and the Iron buildup in the rock could be solved by using a time-dependent diffusion solver (ie not solving for steady state).

As it stands at the moment the solutes are solved to steady state, at the beginning of each timestep based on the current distribution of biomass, then the bugs are grown using the new solute concentrations.

I think the best way to go about turning this into a time-dependent solver would be to use some sort of leapfrog style algorithm where the RD equation is solved over some finite time, then the bugs are grown for the same amount of time. I don't think you can do this using the multi-grid solver, as I can't work out how to know how much time it integrates over (probably as close to infinity as they can get!). I think by creating a new class based on `DiffusionSolver` will be able to do it, using a slightly simpler integration method. Then if you set the global timestep to be equal to the agent timestep, and ensure this solver integrates for that length of time `Simulator.step()` shouldn't need much changing to make things solve in the right order, using the right intermediate values.

2 Setup

Rock is represented by a uniform square lattice of equally sized particles of EPS (capsule). The rock fills the domain from the substrate up to a specified height (I use 0.5 of the domain height). This lattice is generated using `createagentlattice.py` (see *Utilities and Scripts*).

Bacteria are placed on the surface of the rock by specifying the `initArea` in the protocol file. In `protocol_files/generated_rock_states` there is a protocol file with *only* the rock lattice, and another with 186 bacteria in a few clumps.

If there are only a few bacteria (about 10) it can take a while for them to produce enough H3O+ to start visibly growing.

2.1 Getting Up and Running

I've created a GitHub repository (`https://github.com/fophillips/bacteria-on-rocks`) which will let you get running a simulation from where I left off. First you want to clone the repository to your own machine:

```
% git clone https://github.com/fophillips/bacteria-on-rocks.git
% cd bacteria-on-rocks
```

this will create a directory `bacteria-on-rocks` in your current working directory, and change into it. In here you will find a file structure that looks like:

```
% tree -L 2
.
|-- iDynoMiCS
|-- protocol_files
|   |-- generated_rock_states
|   |-- bacteria_on_rock.xml
|   `-- random.state
|-- reactions.tbl
|-- results_files
|   `-- bacteria_on_rock(20140722_1354)
`-- utils
    |-- createagentlattice.py
    |-- generate_povray_video
    |-- pyDynoMiCS
    `-- reaction-diffusion
```

If you run

```
% git submodule init
% git submodule update
```

This will download the latest version of iDynoMiCS as well as the utilities in `utils`. Now we need to compile this version of iDynoMiCS

```
% cd iDynoMiCS
% ant
```

Now we can return to our base directory and run our protocol file using this version of iDynoMiCS

```
% cd ..
% ./iDynoMiCS/scripts_to_start_idynomics/RunIdyno.py -s ./iDynoMiCS/bin
    protocol_files/bacteria_on_rock.xml
```

this will generate results and put them in `results_files`. There is already one set of results files in there from this protocol file, with images and videos in `images`.

3 Reactions

Parameters and references are in reactions.tbl in the bacteria-on-rocks GitHub.

- **AbioticWeather** The weathering caused by physical means unrelated to the bacteria. The reaction is catalysed by the rock, and the rate has linear form wrt the concentration of the *Weather* solute.
- **BacterialGrowth** The uptake of *Iron* and *O2* creates *biomass* and *H3O+*. The parameters of the MonodKinetic for Iron uptake are from Casey's experiments. The H3O+ acts as an inhibiting solute to try and achieve a minimum pH of 6. If the pH is not limited in this way the growth accelerates indefinitely as the H3O+ concentration grows.
- **DissolveRock** The biotic dissolution of rock by H3O+ based on experimental data. Uses a new kinetic factor called *AcidKinetic* and a reaction type called *AcidReactionFactor*. This is so the reaction can be defined by the equation referenced in the reactions table. The reaction has been changed slightly to have a rate of 0 at ph7 and above, and the terms that depend on the OH- concentration have been removed. Add equation. Because the iron diffuses into the rock, and is artificially absorbed, the yield of this reaction is affected. Instead of using the correct stochiometric coefficient, we have to make it artificially high, until we see the correct rate of growth from the bacteria.
- **DegradeIron** The hack to stop the Iron building up in the rock. Rock absorbs the iron at an arbitrary rate. The rate here affects the total amount of iron outside of the rock too. There are HUGE amounts of iron absorbed, which greatly skews the yield of DissolveRock.
- **Starvation** This is currently turned off. This is to kill the bugs off which aren't growing. I haven't researched what sort of parameters this should use. Otherwise biofilm grows indefinitely, could be replaced by detachment.

3.1 BacterialGrowth Yields

Casey: cupriavidus is heterotroph, uses carbon as energy source.

Mass based equation for carbohydrate metabolism[1]:

$$CH_2O + 0.309O_2 + 0.085NH_4^+ + 0.289HCO_3^- \Leftrightarrow 0.535C_5H_7O_2N + 0.633CO_2 + 0.515H_2O$$

Assume consumption of 1 mole of HCO_3^- equates to production of one mole of H_3O^+. 0.289g of HCO_3^- is 2.64e-3 moles, so 0.093g of H_3O^+.

We use the mass based yields with iDynoMiCS.

4 Utilities and Scripts

4.1 generate_povray_video

This might be useful for Gavin, I based it off Gavin's script. I keep one copy of it in `~/Code/utils/` and just run:

`~/Code/utils/generate_povray_video path/to/results/povray.zip`

and it creates an `images` directory in the results directory, generates all the images and creates the AVI movie and also a GIF which you can use on a webpage or presentation, then cleans up all the .pov files.

4.2 createagentlattice.py

This is a Python script to generate the rock configurations. It is run with

```
python ~/Code/utils/createagentlattice.py MyRock 0.5 > rock_agent_state.xml
```

Where `MyRock` is the species you want to generate and `0.5` is the fraction of the domain height you want to fill.

4.3 pyDynoMiCS

This is what I use for making the solute contour plots etc. It turns the iDynoMiCS results files into matrices of numbers.

See the README at `https://github.com/fophillips/pyDynoMiCS` to see how to use it.

4.4 reaction-diffusion

This is a simple 1D reaction diffusion solver (`https://github.com/fophillips/reaction-diffusion`) It has hardcoded boundary conditions of zero-flux at $x = 0$ and concentration=0 at $x = L$. See the README on the GitHub for usage info.

5 Changes to code

See `https://github.com/fophillips/iDynoMiCS/compare/no-move#files_bucket` for a line-by-line run through of changes.

- Changed `ParticulateEPS` to not move if it is shoved
- Changed `Domain.refreshBioFilmGrids()` to use different diffusivities for different species.

[1] Equation 3.7 Filipe, C. D. M., & C P Leslie Grady, J. (1998). Biological Wastewater Treatment, Second Edition, Revised and Expanded. CRC Press.

- iDynoMiCS has a class called `ReactionFactor` which describes a single reaction, that is made of many kinetic factors and is heavily reliant on a catalysing particle. I have written a new class based on this called `AcidReactionFactor` which depends on the interfacial surface area, instead of a catalysing particle.
- Added `AcidKinetic` to calculate the rock dissolution rate as a function of H3O+ concentration.