Super sampling and transformative feedback in Yampa

Jonathan Thaler
School of Computer Science
University of Nottingham
jonathan.thaler@nottingham.ac.uk

Abstract

TODO: this is a small paper which describes the experiments and extensions of super-sampling and transformative feedback in Yampa. If it is going to be published is not clear, maybe it has not enough impact / novelty but still it is a good write-up exercise and can then be used as a reference for users of these features in Yampa (and Yampa in general).

TODO: we need a nice and tiny example which is easy to describe and requires both features: super-sampling and transformative feedback

Index Terms

Functional Reactive Programming, Super-Sampling

I. Introduction

TODO: need a nice and small example, SIR is too complex and don't want to touch on ABS

II. Transformative Feedback

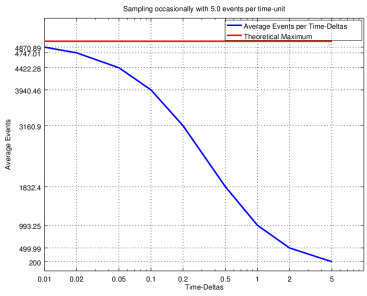
TODO: shortly describe what we want to do:

run a SF i [a] within embed or reactimate which in turn runs a collection of signal-functions: SF (ai, i) ao. In each step the ao then determines the ai for the next step and ao is transformed into an a.

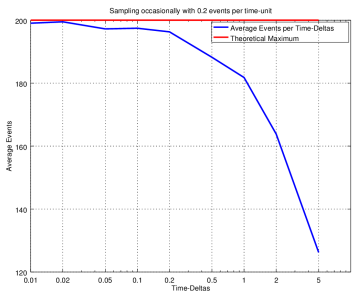
III. Sampling the system

When sampling the system, the correct Δt must be selected which depends on the highest frequency which occurs in a time-reactive function in the whole system. For example in the SIR model we want infected agents to make on average contact with $\beta = 5$ other agents per time-unit, which means with a frequency of $\frac{1}{5}$. This functionality is built on Yampas function *occasionally* which behaviour we investigated under differing Δt with the above frequency. In this investigation we simply sampled occasionally with different Δt for a duration of $t = 1,000$ and the event-frequency of $\frac{1}{5}$. The result can be seen in Figure **??** and is quite striking. The plot clearly shows that occasionally needs a quite high sampling frequency even for a comparatively low event-frequency, which becomes of course worse for higher event-frequencies.

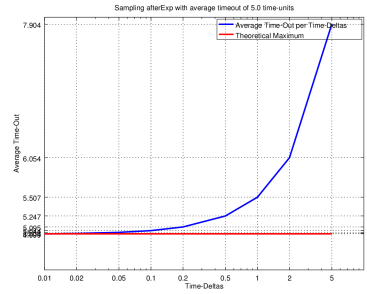
The other time-reactive function which occurs in the SIR model is the timed transition from infected to recovered which occurs on average with an exponential random-distribution after $\delta = 15$. This functionality is built on a custom implementation of Yampas *after* which creates an event after a time-out of the passed in time-duration drawn from an exponential random-distribution. Clearly this function has different semantics as although it also continuously emit events over time - *NoEvent* before the time was hit, and *Event b* after the time hit - the relevant point is that it switches to Event at some discrete point in time. This is implemented as simply adding up the Δt until the accumulator is greater of equal than the previously drawn exponential time-out. We also investigated the behaviour of this function under varying Δt using a time-out of $\delta = 15$. Our approach was to sample the *afterExp* until an event occurs and then see when it has occurred. We run this with 10,000 replications with different random-number seeds and average the resulting times. The results can be seen in Figure **??**. The result is striking in another way: this function seems to be pretty invariant to the time-deltas, for obvious reasons: we are basically just interested in the "after"-condition of the whole semantics whereas in occasionally we are interested in the "repeatedly"-conditions. If we under-sample the *afterExp* then we can be off by one Δt. If we under-sample *occasionally* we keep loosing events - the less difference between Δt and event-frequency, the more events we lose. Of course *afterExp* can also be used for very short time-outs e.g. $\frac{1}{5}$. We have investigated the behaviour of this function for various Δt as well as seen in Figure **??**. Here the result is much more striking and shows that *afterExp* is vulnerable to small time-outs as well as *occasionally*. To show that *occasionally* is not vulnerable to very low frequencies of e.g. one event every 5 time-steps we plotted the behaviour of this under varying time-steps in Figure **??**. The result shows that for low frequencies occasionally works fine with larger Δt.



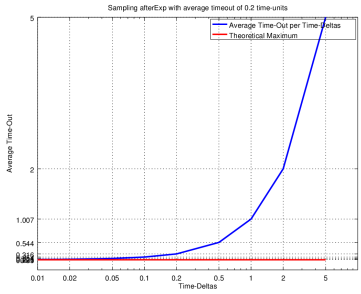
(a) Sampling *occasional* with a frequency of $\frac{1}{5}$ (average of 5 events per time-unit). The theoretical average is 5000 events within this time-frame.



(b) Sampling *occasional* with a frequency of 5 (average of 0.2 events per time-unit). The theoretical average is 200 events within this time-frame.



(c) Sampling *afterExp* with an average time-out of 5.



(d) Sampling *afterExp* with an average time-out of 0.2.

Fig. 1: Sampling the *afterExp* and *occasional* functions to visualise the influence of sampling frequencies on the occurrence of the respective events. Δt are [5, 2, 1, $\frac{1}{2}$, $\frac{1}{5}$, $\frac{1}{10}$, $\frac{1}{20}$, $\frac{1}{50}$, $\frac{1}{100}$]. The experiments for *afterExp* used 10,000 replications. The experiments for *occasional* ran for $t = 1,000$ with 100 replications.

A. Increasing the frequency

Using these observation we run simulations with varying Δt with $\Delta = 0.5$, $\Delta = 0.2$ and $\Delta = 0.1$ with the results visible in Figures **??**, **??** and **??** but still when decreasing Δt we don't approach the SD dynamics. As previously mentioned the agent-based approach is a discrete one which means that with increasing number of agents, the discrete dynamics approximate the continuous dynamics of the SD simulation. We run further simulations with $\Delta = 0.1$ but with varying agent numbers to see the influence with the results seen in Figure **??**.

Although increasing the number of agents improves our approximation, still the dynamics of 10,000 Agents are still nowhere close to the SD dynamics. This is because as opposed to SD, which is deterministic, the agent-based approach is inherently a stochastic one as we continuously draw from random-distributions which drive our state-transitions. What we see in Figure **??** is then just a single run where the dynamics would result in slightly different shapes when run with a different random-number generator seed. The agent-based approach thus generates a distribution of dynamics over which ones needs to average to arrive at the correct solution.

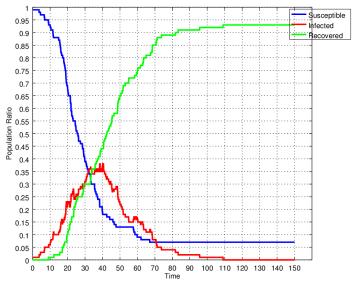
B. Super Sampling

In Yampa there exists a function *embed* which allows to run a given signal-function with provided Δt but the problem is that this function does not really help because it does not return a signal-function. What we need is a signal-function which takes the number of super-samples *n*, the signal-function *sf* to sample and returns a new signal-function which performs super-sampling on it:

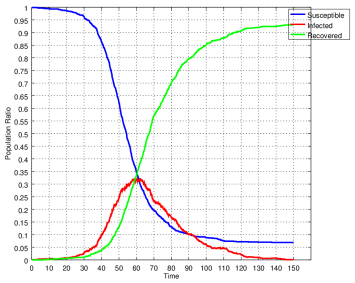
```
superSampling :: Int -> SF a b -> SF a [b]
```

It does this by evaluating *sf* for *n* times, each with $\Delta t = \frac{\Delta t}{n}$ and the same input argument *a* for all *n* evaluations. At time 0 no super-sampling is done and just a single output of *sf* is calculated. A list of *b* is returned with length of *n* containing the result of the *n* evaluations of *sf*. If 0 or less super samples are requested exactly one is calculated.

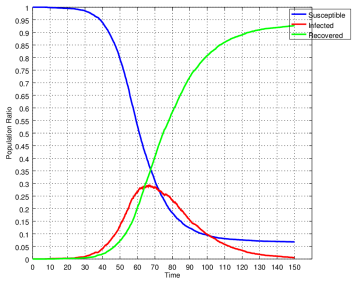
We ran tests super-sampling both *occasionally* Figure **??**, Figure **??** and *afterExp* Figure **??**, Figure **??**. They work the same way as above except that now $\Delta t = 1.0$ but using increasing numbers of super-samples. The results are as expected: as the number of super-samples increase, so increases the accuracy.



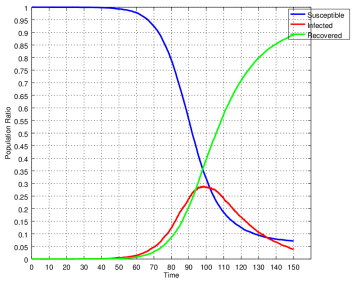
(a) 100 Agents



(b) 1,000 Agents

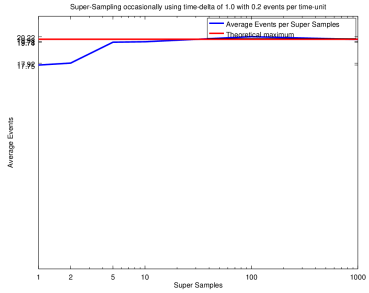


(c) 5,000 Agents

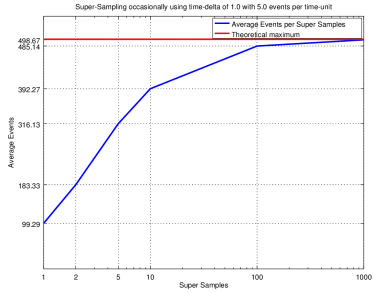


(d) 10,000 Agents

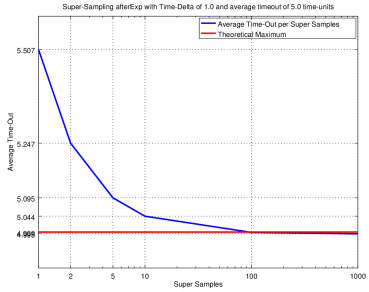
Fig. 2: Varying agent numbers with same model-parameters except population size. All simulations run for 150 time-steps with $\Delta t = 0.1$



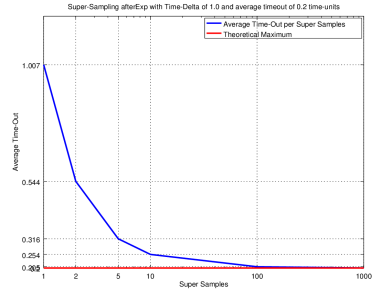
(a) Super-Sampling the *occasional* function with event-frequency of 5 (average of 0.2 events per time-unit). The theoretical average is 20 event within this time-frame.



(b) Super-Sampling the *occasional* function with event-frequency of $\frac{1}{5}$ (average of 5 events per time-unit). The theoretical average is 500 event within this time-frame.



(c) Super-Sampling the *afterExp* function with average time-out of 5.



(d) Super-Sampling the *afterExp* function with average time-out of 0.2.

Fig. 3: Super-Sampling the *afterExp* and *occasional* functions to visualize the influence of increasing number of super-samples on the average occurrence of the respective events. The $\Delta t = 1.0$ in both cases with super-samples of [1, 2, 5, 10, 100, 1000]. The experiments for *afterExp* used 10,000 replications. The experiments for *occasional* ran for $t = 100$ with 100 replications.

At first this might not seem to be a real win as we still need to calculate a big number of samples every time. The big win comes though when these super-sampled signal-functions are embedded in a larger system which could run on a comparatively low frequency of $\Delta t = 1.0$. So we are then increasing the sampling-frequency just where we need it and keep the frequency low where it is not required.

We are using super-sampling in our SIR implementation to increase performance. We do this by setting $\Delta t = 1.0$ and super-sampling the relevant functions with time-semantics which are *transitionAfterExp* and *sendMessageOccationallySrc*. For both we provide in our EDSL versions which support super-sampling:

```
sendMessageOccasionallySrcSS :: RandomGen g => g -> Double -> Int -> MessageSource
                             -> SF (AgentOut, e) AgentOut

transitionAfterExpSS :: RandomGen g => g -> Double -> Int
                     -> AgentBehaviour -> AgentBehaviour -> AgentBehaviour
```

Both now take an additional parameter which determines the number of super-samples to be calculated. According to the above observations of the *occasionally* and *afterExp* functions which are the foundations of both of the functions we sample *sendMessageOccasionallySrcSS* with 20 super-samples and *transitionAfterExpSS* with 2. This will ensure that by using $\Delta t = 1.0$ we only calculate t steps when running a simulation for t time but that we sample our relevant functions with enough resolution to capture its frequencies. Optimally we should increase the number of super-samples for *sendMessageOccasionallySrcSS* to about 100. This will result in lower performance as *every* agent will perform this super-sampling. So in the end it is a struggle of performance vs. sufficiently close approximation. We define the number of super-samples in lines 29 and 32 and use the functions in line 95 and 104 of Appendix **??**.

C. Extensions

TODO: how do we sample the t+dt space can be various: random, uniform (as above), triangular,... the problem is that we must make an assumption about the shape of the function. probably random-noise sampling is best as done in computer-graphics? This needs a thorough research and experiments.