indicates that these two tasks are actually closely related. Remember, the balance equation

$$\text{balance}(x) = f(x)$$

offers us two different ways to get to a solution. Either we solve the equation $0 = f(x)$ directly or we approach the steady state by a sub-simulation on the differential equation $\text{der}(x) = f(x)$. On the first look, this looks like two separate tasks. However, let us analyze how we would perform such a sub-simulation in practice.

After all, this is a special case: we do not perform a usual simulation; we want to perform a simulation for the sole purpose to approach the steady state with the time[3] $t \to \infty$. Which integration method would we choose for that?

Since the differential equation $\text{der}(x) = f(x)$ is supposed to describe a stable system, an implicit method is a strong favorite. Any explicit integration method would be limited in its step-size in order to maintain stability and approaching infinity with steps of finite width is an unpromising endeavor.

Since we do not care about the precise trajectory leading to the steady state and since the steady state solution itself is insensitive to the local integration error, there is no reason to choose any higher-order method. Order 1 is completely sufficient.

So our method of choice would be to perform Backward Euler with as large steps as possible. Hence, let us look at one integration step of this method going from t to $t+h$:

$$x_{t+h} = x_t + h \cdot \text{der}(x_{t+h})$$

or in our current example:

$$x_{t+h} = x_t + h \cdot f(x_{t+h})$$

Being an implicit method, we have to solve the system of equations: $0 = g(x_{t+h})$ with $g(x_{t+h})$ being defined as:

$$g(x_{t+h}) = x_t - x_{t+h} + h \cdot f(x_{t+h})$$

Evidently for $h \to \infty$, solving $g(x_{t+h})$ becomes equivalent to solving $f(x)$ directly. Now it becomes clear how the solver dynamics can support us to find the solution of $f(x)$. Instead of solving the system $f(x)$, we can solve $g(x)$ and in this way, we have won one important degree of freedom: we can choose h.

In this way, we have transformed the problem into a numerical continuation problem [1]. In general, a continuation problem results from transforming a function $F(x)$ to $F'(x,\lambda)$ with $\lambda \in [0,1]$ where $F'(x,1) = F(x)$ and $F'(x,0)$ is easy to solve. Many solutions methods have been developed for this kind of problem and they are already applied by many M&S Frameworks, mostly to solve initialization problems in a more robust way [9] for instance by using homotopy [8].

[3] Please note, the time t does not represent the main simulation time here but the time of the nested sub-simulation. The complete time-span of the sub-simulation represents only one instant in the main simulation.

To use numerical continuation solvers not only for initialization problems but also during simulation is also not a completely new idea. Artificial time integration is not uncommon to find solutions for PDEs [2]. The main difference to classic continuation problem in our case is that is not bounded by 1 but is free to go to infinity. Hence we have to adapt the continuation solver. The following paragraph sketches an algorithm that is a variant of the simplest kind of numerical continuation: the natural parameter continuation where h is our continuation parameter.

In case h is too large and our guess value for x_{t+h} is outside the convergence area, we can choose h small enough to be located in the convergence area again. And with each solution of $g(x)$, we step a little closer to the final solution of $f(x)$. In this way, we have found a robust way to solve our non-linear system of equations. Figure 5 depicts the corresponding algorithm of the balance dynamics solver.

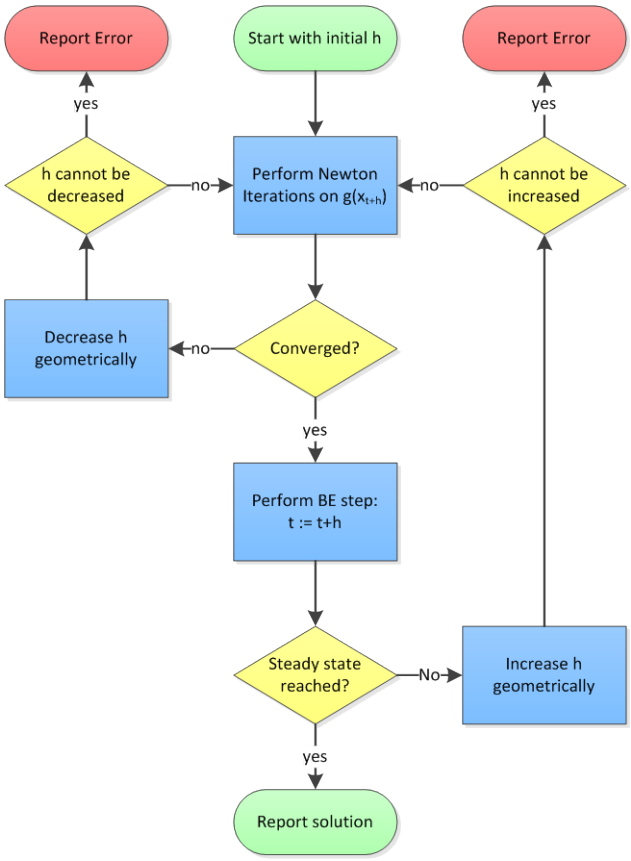

Figure 5: Algorithm for the balance dynamics solver

This algorithm becomes part of the main simulation loop and replaces the former direct solver for $0 = f(x)$. It is hence performed at each integration step of the main simulation task. It is not necessarily slower than the direct solver for $f(x)$. Having a high initial value for the sub-simulation step-size h and a good guess value for x_{t+h}, not many more iterations would be required than for a direct solution of $0 = f(x)$. A call to the direct solver is thus not required.