Algorithms in MasterSim

Andreas Nicolai

May 20, 2016

1 Master-Algorithms

All algorithms are called with current time point t, step size to be taken h and vector of current solution $\mathbf{y_t}$ with the mapping $\mathbf{u_s} = \mathbf{P_s}(\mathbf{y})$ for the inputs to all slaves s.

1.1 Gauss-Jacobi

Algorithm 1: Gauss-Jacobi Algorithm

Input: t, h, $\mathbf{y_t}$ vector with reals at time level t
Output: $\mathbf{y_{t+h}}$ vector with solution at time level $t+h$

```
begin
    for cycle ∈ cycles do
        for slave ∈ cycle.slaves do
            Set slave inputs using inputs from variable vector y_t
            Advance slave to time level t + h
            Retrieve outputs and update y_{t+h} (overwriting vector elements)
```

1.2 Iterative Gauss-Seidel

Needs temporary vector $\mathbf{y^m_{t+h}}$ for iterative quantities. This algorithm expects FMU states from time t and matching inputs $\mathbf{u_t} = \mathbf{P}(\mathbf{y_t})$ have been stored.

Algorithm 2: Gauss-Seidel Algorithm

Input: t, h, $\mathbf{y_t}$ vector with reals at time level t
Output: $\mathbf{y_{t+h}}$ vector with solution at time level $t + h$

```
begin
    y_{t+h} := y_t
    for cycle ∈ cycles do
        while iteration < maxIterations do
            Store iterative solution for convergence test
            y^m_{t+h} := y_{t+h}
            if iteration > 1 then
                Restore slave states

            for slave ∈ cycle.slaves do
                Set slave inputs using inputs from variable vector y_{t+h}
                Advance slave to time level t + h
                Retrieve outputs and store in vector y_{t+h} (partially overwriting vector elements)
            if cycle.slaves.count() == 1 then
                No need to iterate when only one slave in cycle
                break
            if h < h_limit then
                If time step is too low, skip iteration (to get past discontinuities)
                break
            Compute Weighted-Root-Mean-Square norm of differences
            res = WRMS (y^m_{t+h}, y_{t+h})
            if res < 1 then
                Converged
                break
        if iteration >= maxIterations then
            Max. iteration count exceeded, algorithm not converged
            return IterationLimitExceeded
```

1.3 Newton

This algorithm expects FMU states from time t and matching inputs $\mathbf{u_t} = \mathbf{P}(\mathbf{y_t})$ have been stored. Needs temporary vector $\mathbf{y^m_{t+h}}$ for iterative quantities, vector $\mathbf{r}$ for residuals.

Newton algorithm is based on rearranged fix point iteration scheme (1), which is transformed into the

Newton step equation (2). This gives the correction to the current value estimate (3).

$$y := S(y) \tag{1}$$

$$0 = y - S(y) = G(y)$$

$$\left.\frac{\partial G}{\partial y}\right|_m \Delta y^{m+1} = -G(y^m)$$

$$\left.\frac{\partial G}{\partial y}\right|_m \Delta y^{m+1} = y^m - S(y^m) \tag{2}$$

$$y^{m+1} = y^m + \Delta y^{m+1} \tag{3}$$

Jacobian matrix elements are computed via difference quotiont approximation (4).

$$\begin{aligned}\frac{\partial G_i}{\partial y_j} &\simeq \frac{G_i(\mathbf{y}+\varepsilon\mathbf{e}_j) - G_i(\mathbf{y})}{\varepsilon} \\ &= \frac{[\mathbf{y}+\varepsilon\mathbf{e}_j]_i - S_i(\mathbf{y}+\varepsilon\mathbf{e}_j) - y_i + S_i(\mathbf{y})}{\varepsilon} \\ &= \frac{y_i + \varepsilon\delta_{ij} - S_i(\mathbf{y}+\varepsilon\mathbf{e}_j) - y_i + S_i(\mathbf{y})}{\varepsilon} \\ &= \delta_{ij} - \frac{S_i(\mathbf{y}+\varepsilon\mathbf{e}_j) - S_i(\mathbf{y})}{\varepsilon}\end{aligned} \tag{4}$$

Algorithm 3: Newton Algorithm

Input: t, h, $\mathbf{y_t}$ vector with reals at time level t
Output: $\mathbf{y_{t+h}}$ vector with solution at time level $t+h$

```
begin
    for cycle ∈ cycles do
        while iteration < maxIterations do
            for slave ∈ cycle.slaves do
                Set slave inputs using inputs from variable vector y^i_{t+h}
                Advance slave to time level t + h to get Sy = S(y^i_{t+h})
                Retrieve outputs and store in vector r
            Compute residuals r := r − y^i_{t+h}
            if iteration == 1 then
                Setup Jacobian J
            Extract all r_i that belong to the cycle and put them into vector rhs
            rhs := variableMap(r)
            Solve equation system, rhs now holds Δy
            rhs := J^{-1} rhs
            Compute WRMS norm
            δ = ||Δy||_WRMS
            Compute new solution
            y^{i+1}_{t+h} = y^i_{t+h} + variableMapping(Δy)
            if δ < 1 then
                break
```