Lehrstuhl für Elektrische Energiesysteme

Friedrich-Alexander-Universität Technische Fakultät

An Accessible PyTorch Implementation of Automatic Differentiation for Power System Model Parameter Identification and Optimization

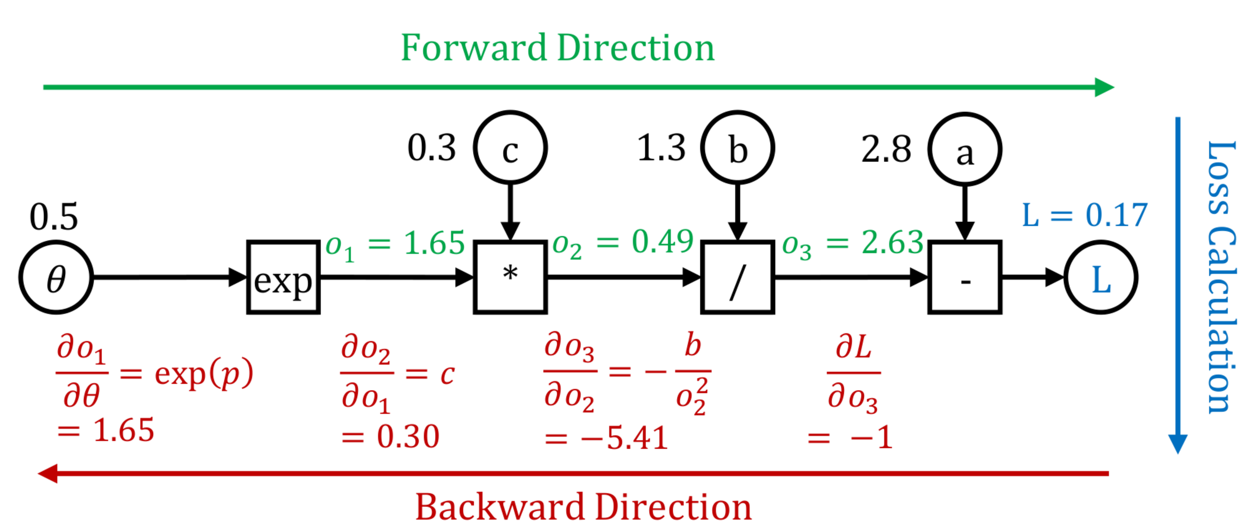
1st Georg Kordowich, 2nd Johann Jaeger

Friedrich-Alexander-Universität Erlangen-Nürnberg – Germany

Automatic Differentiation

Computation of Gradients

- Conventional computation: $\frac{\partial f(x)}{\partial x} \approx \frac{f(x+h)-f(x)}{h}$
- Automatic differentiation (AD) applies chain rule of differentiation: $\frac{\partial L}{\partial p} = \frac{\partial L}{\partial o_1}\frac{\partial o_1}{\partial o_2}\frac{\partial o_2}{\partial o_3} \cdots \frac{\partial o_{n-1}}{\partial o_n}\frac{\partial o_n}{\partial p}$



Dynamic RMS Simulations

Phasor Based Simulation

- Power system can be described by a set of differential algebraic equations:
  $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{y})$
  $0 = g(\boldsymbol{x}, \boldsymbol{y})$
- Simulation consists of locally differentiable operation ($+, -, *, /, ln(x), e^x, \ldots$)

→ Automatic differentiation is applicable to power system simulations

→ A framework enabling automatic differentiation is necessary for the accesibility of the approach

Simulate for t timesteps

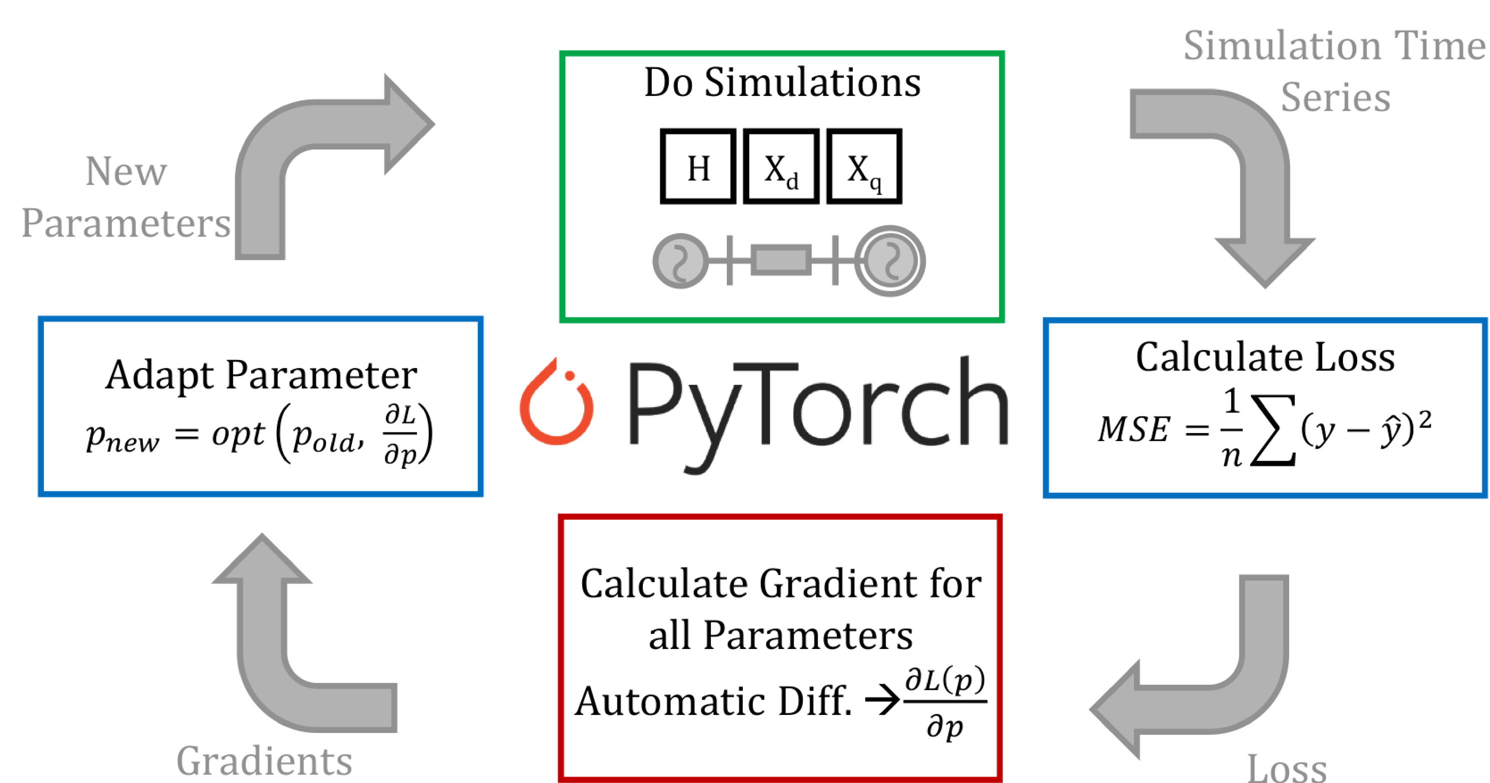
Calculate Algebraic Equations $y_t = \mathrm{h}(x_t)$

Calculate Differential Equations $\dot{x}_t = \mathrm{f}(x_t, y_t)$

Integration of State Variables $x_{t+1} = x_t + \dot{x}_t \cdot \Delta t$

AD for Power Systems

Gradient Descent based Optimization Process



A PyTorch Based Framework for Automatic Differentiation

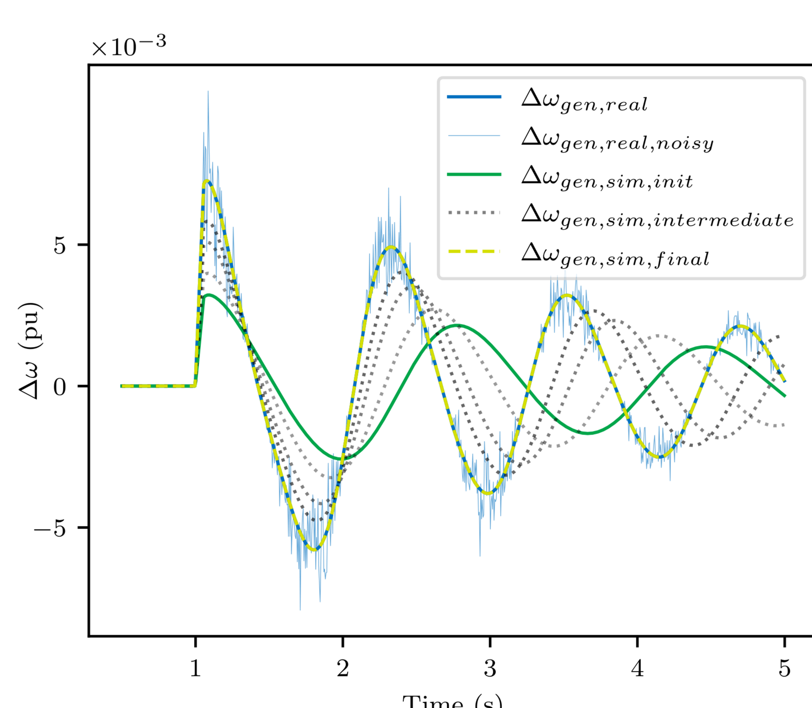
- Determines the gradient $\frac{\partial L}{\partial \theta}$ of parameters θ with respect to a loss function L.
- Dynamic power system simulation is implemented in Python using PyTorch as an AD tool
- Flexible and modular
- Calculation of gradients in one line of code
- Optimization using pre-defined PyTorch optimizers is possible

```
# addition of a bus to the model
sim.add_bus(Bus(name='Bus 0', v_n=24))
# addition of a short circuit event
sim.add_sc_event(start_time=1,
                 end_time=1.05, bus='Bus 1')
...

# one optimization step
t, result = sim.run()  # Simulation
loss = loss_function(result,
                     target)  # Loss calculation
loss.backward()  # Gradient computation
optimizer.step()  # Gradient descent
```

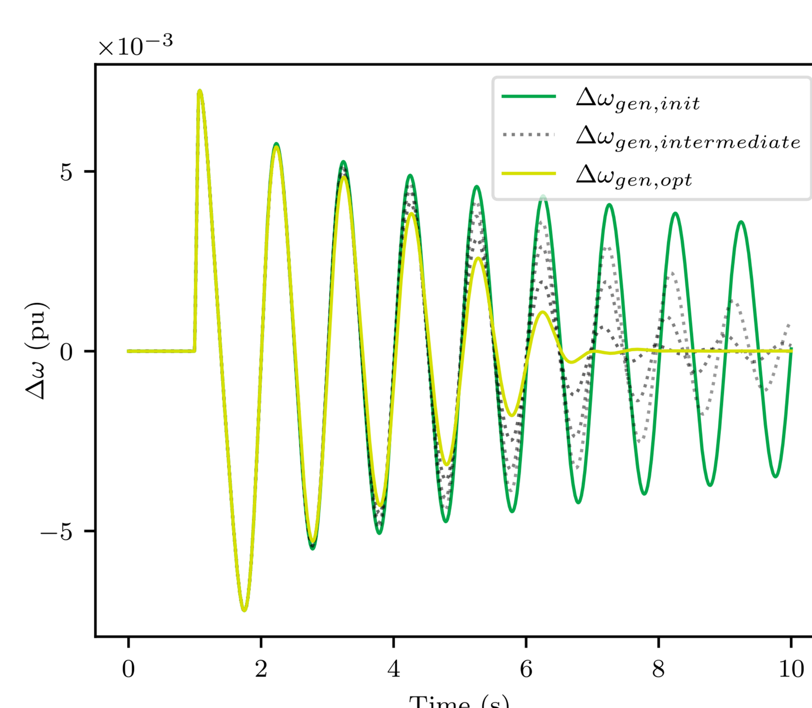
Exemplary Applications

Parameter Identification



Which simulation parameters describe a real-world system best?

Parameter Optimization



Which controller parameters lead to a sufficiently damped system?

Conclusions

Advantages

+ Scalable gradient calculation
+ Accessible implementation
+ Inherently vectorized approach
+ Full control over simulation
+ Simple integration of ANNs

Disadvantages

- Gradient descent can get stuck in local optima
- Python is inherently slower than more low level languages