Application of Verified Optimization Techniques to Parameter Identification for Solid Oxide Fuel Cells

Stefan Kiel, Ekaterina Auer, and Andreas Rauh

06.06.2012
Table of Contents

1 Solid Oxide Fuel Cells

2 Parameter Identification

3 Optimization Algorithm

4 Improving Performance with GPU

5 Conclusions and Outlook
Solid oxide fuel cells (SOFCs)

- Convert chemical energy into electricity
- Decentralized energy supply
- Complex control strategies
Project VeriCell

A joint project between the Universities of Rostock and Duisburg-Essen

A goal in Duisburg
Development of a flexible software environment capable of simulating the SOFC stack and testing control strategies....

...that take into account
  - Instationary operating points
  - Uncertainties due to measurement errors, ...

→ Allow for using control-oriented mathematical SOFC models developed in Rostock.
Semi-discretization of the stack module into finite volume elements

PDEs → ODEs

Currently 3 models

$1 \times 1 \times 1$: 1 State
$3 \times 1 \times 1$: 3 States
$3 \times 3 \times 1$: 9 States
$\dot{\theta}_{FC} = \dot{m}_{H_2} \cdot \left( p_{\Delta H,2} \cdot \theta_{FC}^2 + p_{\Delta H,1} \cdot \theta_{FC} + p_{\Delta H,0} \right) + 6 \cdot p_A \cdot (\theta_A - \theta_{FC}) + (\theta_{AG} - \theta_{FC})$

$\cdot \left( \dot{m}_{H_2} \cdot \left( p_{H_2,2} \cdot \theta_{FC}^2 + p_{H_2,1} \cdot \theta_{FC} + p_{H_2,0} \right) + \dot{m}_{H_2O} \cdot \left( p_{H_2O,2} \cdot \theta_{FC}^2 + p_{H_2O,1} \cdot \theta_{FC} + p_{H_2O,0} \right) \right)$

$+ \dot{m}_{N_2} \cdot \left( p_{N_2,A,2} \cdot \theta_{FC}^2 + p_{N_2,A,1} \cdot \theta_{FC} + p_{N_2,A,0} \right) + \dot{I}_{FC} \cdot p_{el} - \dot{m}_A \cdot (\theta_{FC} - \theta_{CG})$

$\cdot \left( 77 \cdot p_{N_2,C,0}/100 + 11 \cdot p_{O_2,0}/50 + 77 \cdot p_{N_2,C,1} \cdot \theta_{FC}/100 \right)$

$+ 11 \cdot p_{O_2,1} \cdot \theta_{FC}/50 + 77 \cdot p_{N_2,C,2} \cdot \theta_{FC}^2/100 + 11 \cdot p_{O_2,2} \cdot \theta_{FC}^2/50)$

$= f(\theta_{FC}, p)$

No analytical solution

The temperature $\theta_{FC}$ is the variable, the rest are parameters

Parameters

- thermal resistances of the stack materials
- dependencies of heat capacities on the temperature
- heat produced on the surface of each individual cell
Solving the ODEs

“Verified Approximation” (Euler’s method)

\[ y_k := y_{k-1} + h \cdot f(y_{k-1}, p) \]

(Denote \( \theta_{FC} \) by \( y \).)

Verified Solution

Enclose the exact result numerically through a verified IVP-Solver (VNODE-LP, ValEncIA-IVP, ...)

Solid Oxide Fuel Cells
Solving the ODEs (2)

Pros & Cons

Euler’s method
- Simple
- Not verified
- Derivatives available (AD)
- Imposes overestimation
- *Easily portable to the GPU*

IVP-Solver
- Verifies the whole model
- Derivatives require solving an extra ODE
- Increased computational effort

→ Currently, we stick to Euler’s method.
Parameter Identification

Goal

Parametrize the model in a robust and accurate way

We identify 6 parameters

\[ p = [p_{H_2,0} \ p_{H_2,O,0} \ p_{N_2,A,0} \ p_{N_2,C,0} \ p_{O_2,0} \ p_{\Delta H,0}] \]

The other parameters are replaced by approximations obtained by floating-point methods.

Identification is based on a set of reference measurements

(from the SOFC test rig of the University of Rostock)
Parameter Identification

Quadratic Error Function

\[ \Phi(p) = \sum_{k=1}^{T} (y(t_k, p) - y_m(t_k))^2 \]

with

\[ p \rightarrow \text{Parameters to identify} \]
\[ y(t_k, p) \rightarrow \text{Simulated temperature at time } t \]
\[ y_m(t_k) \rightarrow \text{Measured temperature at time } t \]
\[ T \rightarrow \text{Number of measurements (19963)} \]
\[ t_{k-1} - t_k = 1s \rightarrow \text{Step size 1s} \]

(\( \Phi(p) \) is written down the \( 1 \times 1 \times 1 \) model.)
Problem Statement

Optimization problem

$$\min_{p \in p_0} \Phi(p)$$

Bound constraint problem ($p_0 \in \mathbb{R}^6$, wid $p_0 = 2.0$)
Initial vector for $p_0$ derived by floating-point methods

Difficulties

- Objective function is computationally expensive
- Calculating derivatives is really slow (even with fadbad++)
- Considerable overestimation

→ Individual strategies for dealing with the problem (derivative free).
**Consistent States**

**Consistent parameter vectors**
A state vector $p$ is consistent if $\forall t \in \{0, ..., T\}$:

$$[y(t, p)] \subseteq y_m(t) + [\Delta y_m]$$

with the worst-case measurement error $[\Delta y_m] = [-15, 15]$ holds.

**Inconsistent parameter vectors**
A state vector $p$ is inconsistent if $\exists t \in \{0, ..., T\}$:

$$[y(t, p)] \cap (y_m(t) + [\Delta y_m]) = \emptyset$$
**UniVerMeC**

**Unified Framework for Verified Geometric Computations**

- **core**  Adapter for underlying arithmetic libraries
- **functions** Uniform representation for functions
- **objects** Implicit surfaces, CSG models, polyhedrons
- **decomp** Spatial decomposition, Multisection schemes
- **algorithms** Distance computation, Global optimization, ....
**Application to Parameter Identification**

**Algorithms**
Global optimization algorithm

**Decomposition**
Ratz multisection scheme

**Abstract Functions**
Function representation
(Affine linearization)
(Automatic differentiation)

**Adapters**
C-XSC (interval arithmetic)
YalAA (affine arithmetic)
Optimization Algorithm

Branch & Bound

Basic pattern

1. $p \leftarrow \mathcal{L}$
2. Discard $p$ if it is infeasible
3. Discard $p$ if $\Phi(p) > \overline{D}$
4. Contract $p$
5. Update of $\overline{D}$
6. Add $p$ to $\mathcal{L}_{\text{final}}$ if termination criteria are satisfied
7. Split $p$ and add new boxes to $\mathcal{L}$

Main data structures

- Two lists containing parts of the search space (boxes)
- Ordered working list $\mathcal{L}$
- Solution list $\mathcal{L}_{\text{final}}$

Termination criteria

- $\text{wid } p \leq \epsilon_p, \epsilon_p > 0$
- $\text{wid } (\Phi(p)) \leq \epsilon_\Phi, \epsilon_\Phi > 0$

- Finds the minimum in the specified starting box
- Based on Hansen’s interval optimization algorithm
Parallelization

A lot of local work (objective evaluation, midpoint test, ...)

Workload sharing problem

The B&B tree is unbalanced

No apriori partitioning into subproblems

Parallel CPU version with OpenMP

We share $\mathcal{L}$ between all threads
→ Only suitable for shared-memory
→ Bottle neck for the algorithm (critical section)
Configurable Algorithm in UniVerMeC

Divided into phases

- Phase A
- Phase Pos. Infeas.
- Phase B
- Phase Feasible
- Phase C
- Phase Strict. Feas.
- Phase D
- Phase Split

Phases consist of strategy elements

- Phases can be reconfigured (iterations, minimum box width on $\mathcal{L}$)

Special phases for strategy changes and end of the algorithm

- They can
  - discard a box
  - contract a box
  - restart the main loop and add a box back to $\mathcal{L}$
**Configurable Algorithm in UniVerMeC**

**Divided into phases**

- **Phase A**
  - Phase Pos. Infeas.
  - Phase B
  - Phase Feasible
  - Phase C
  - Phase Strict. Feas.
  - Phase D
  - Phase Split

**Configuration**

- **Phase A**
  - Midpoint Test

- **Phase Feasible**
  - Update upper bound

- **Phase D**
  - Linearization and pruning based on the consistency constraint

- **Phase Split**
  - Calculate bound on $\Phi(p)$
  - Check for (in)consistent states
Optimization Algorithm

Quality Measure

Identified candidate intervals $p$ are characterized by

$$e = \sqrt{\frac{1}{T} \sum_{k=1}^{T} (y_{k-1} - y_m(t_k) + f(y_{k-1}, \text{mid}(p)))^2}$$

This measure is

- practice-motivated
- similar to the root mean square error measure

Candidate $p$ with lowest $e$ is chosen as solution, if there is no other way to proof that $p$ is the optimum (minimum).
CPU Results

### Difference between measured and simulated temperature

![Graph showing the difference between measured and simulated temperature over time.](image_url)

<table>
<thead>
<tr>
<th></th>
<th>floating point</th>
<th>UniVerMeC</th>
</tr>
</thead>
<tbody>
<tr>
<td>error in K</td>
<td>5.17</td>
<td>7.68</td>
</tr>
<tr>
<td>Wall time (h)</td>
<td>≈ 6</td>
<td>≈ 0.70</td>
</tr>
</tbody>
</table>
GPGPU

History

GPUs are highly specialized units for rendering

Earlier: fixed-function pipeline

Computer graphic becomes more and more sophisticated

→ Programmable units (Shader)

General Purpose Computation on Graphics Processing Unit

Use of GPUs also for non-graphic tasks (e.g. scientific computing)

Cheap computational power (for the right problem)
The **Stream Processing Model** consists of:

- many data elements (*the stream*)
- a program to apply to the elements (*the kernel*)

This is done in parallel on a special co-processor (GPU, FPGA)

Applicable when the kernel:

- has to be applied to many elements
- is parallelizable (results do not depend on each other)
- is computationally expensive (much more arithmetic than IO operations!)
GPU Architecture

GPU consists of **Streaming Multiprocessors** (SMs)
Can execute many threads simultaneously (hundreds)
**SIMT (Single Instruction, Multiple Threads) architecture** (≈ to SIMD)
More basic than the CPU (e.g. no branch-prediction)
→ GPUs are cheap streamprocessors

**Organization**

Threads are organized into a grid
Blocks of threads (max. 1024 threads per block)
These blocks are distributed to a SM
Synchronization only possible inside the same block
GPGPU program consists of a CPU and a GPU part

- CPU starts the GPU part
- Data transfer is expensive (long latency)
- CPU can proceed while GPU computes (asynchronous calls)

→ Parts not well suited for the GPU are executed on the CPU
Perform as many computations as possible on transferred data
**OpenCL**

Open standard (Khronos group)
Supported by all major vendors
Tool support still immature
Requires only single precision, no rounding modes besides to nearest (→ optional extension for double arithmetic available)

→ We are using CUDA.

**CUDA**

Proprietary NVIDIA specific programming model
Better tool support
Nearly IEEE 754 compliant (on newer cards)
Rudimentary interval library available

→ We are using CUDA.
Improving Performance with GPU

Floating-Point Considerations

**Consumer GPUs are optimized for single precision throughput**

<table>
<thead>
<tr>
<th></th>
<th>1.x</th>
<th>2.0</th>
<th>2.1</th>
<th>3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single prec.</td>
<td>8</td>
<td>32</td>
<td>48</td>
<td>192</td>
</tr>
<tr>
<td>Double prec.</td>
<td>1</td>
<td>16</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

*Operations per clock cycle per SM for NVIDIA GPUs*

Lower memory usage for single prec. (decreases memory transfer time)

Preliminary test showed that single precision is not sufficient (Rounding errors are too much amplified by the Euler iteration...)

Interval evaluations of $\Phi(p)$ for the $3 \times 3 \times 1$ model
Use of GPU in Parameter Identification

\[ \Phi(p) = \sum_{k=1}^{T} (y(t_k, p) - y_m(t_k))^2 \]

Value of \( y(t_k, p) \) depends on \( y(t_{k-1}, p) \)

A single evaluation cannot be parallelized

But we can evaluate \( \Phi(p) \) over different subdivision intervals in parallel!
Objective Function Evaluation Benchmark

Reference system

Intel Core i7-860 (2.8 GHz)
gcc 4.7 on Linux
GeForce GTX 560 Ti
CUDA 4.2

Results

1 × 1 × 1: GPU 30× faster
3 × 3 × 1: GPU 114× faster
(employs much more arithmetic operations on the same input data!)
Integration into the Optimization Algorithm

GPU and CPU run in parallel
Working list $\mathcal{L}$ is in host memory
One CPU thread feeds the GPU with data
Other CPU threads work normally

$\rightarrow$ Currently, only bounds on $\Phi$ are derived using the GPU
Improving Performance with GPU

GPU Results ($1 \times 1 \times 1$)

**Difference between measured and simulated temperature**

![Graph showing the difference between measured and simulated temperature over time.]

<table>
<thead>
<tr>
<th>Error measure</th>
<th>Wall time</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>7.42944</td>
</tr>
<tr>
<td>CPU (OpenMP)</td>
<td>7.68</td>
</tr>
</tbody>
</table>
Conclusions

- Interval branch and bound optimization algorithms are well suited for parallelization.
- Usage of GPU caused a speed up of 18 (against the parallel CPU version) for the $1 \times 1 \times 1$ model in the parameter identification algorithm.
- Higher speedup is expected for higher dimensional models.
- The runtime for the parameter identification could be significantly decreased.

Future Work

- Apply the algorithm to the $3 \times 3 \times 1$ model.
- Replace the Euler iteration by an IVP solver (fully verified approach).
- Improve GPU kernel speed by making better use of GPU memory hierarchies.
Thank You for Your Attention!
E. Auer, S. Kiel, and A. Rauh.
Verified parameter identification for solid oxide fuel cells.
Accepted.

E. Hansen and G. W. Walster.
*Global Optimization Using Interval Analysis.*

NVIDIA.

Interval methods for control-oriented modeling of the thermal behavior of high-temperature fuel cell stacks.
Accepted.