

Interactive Learning

Oliver Rübenkönig
Jan G. Korvink

Mathematica provides a unique capability for interactive learning. The possibility to combine program code and explanations in an interactive environment is well suited for teaching. The Chair of Microsystem Simulation at the University of Freiburg has developed a wide range of interactive simulation tutorials that have been distributed under the GNU Free Documentation License (FDL). The tutorials cover finite difference, finite volume and finite element methods, and multigrid and iterative solvers. Topics such as sparse matrices and derivatives recovery are also explained. Students are led through the topics assuming little or no prior knowledge. We found that the students gained a good understanding by experimenting with available parameters. In a subsequent step the tutorials are used for verifying other program code.

■ Introduction

The *IMTEK Mathematica Supplement (IMS)* is a downloadable, open source add-on package for *Mathematica* [1]. The supplement provides several hundred functions in about 40 packages. Additionally, about 10 tutorials from the computer simulation area and one concise introductory *Mathematica* programming language tutorial are available, and parts from a computer science book [2] have been ported from Scheme.

We have chosen the GNU FDL so that the documentation is easily adapted for the needs of students or teachers. Also, the motivation for students to send suggestions and improvements for the tutorials is greater than when they are confronted with a (noninteractive) closed source form. Furthermore, we put great effort in motivating students to make suggestions on how to improve the tutorials. We try to exploit the fact that students themselves can explain teaching material to each other fairly well.

In this article we describe our experiences using *Mathematica* as a teaching tool for our subject. We also discuss the goals and philosophy of our teaching, as well as the different media we have tried for teaching: notebooks in lectures and as tutorials, video recordings, and HTML documents.

■ About Teaching

“Tell me and I’ll forget;
show me and I may remember;
involve me and I’ll understand.”
(Chinese Proverb)

First, we have to determine our primary and secondary teaching goals. Primary goals are concerned with complying with the syllabus. Secondary goals are concerned with conveying to students the skills they can use outside of the classroom.

The primary goals would be that students understand the subject in such a manner and depth that they can work in the area and have learned how to learn more.

The secondary goal would be that students know how to use a programming language to solve their engineering problems—and not only in the simulation domain. Being able to program forces students to think clearly about their ideas, understand how to convert their problems in the programming language, and in the process, or better yet due to this process, they inevitably have to think about the “details.” Since programming languages are not forgiving from a syntax point of view, the formulation must be clear.

Sometimes it is necessary to move in the wrong direction—only to then change course and thus have a learning experience. Students learn by making errors and errors give the reason for making changes.

There are several different teaching media.

1. Notebooks
2. HTML
3. Video

Notebooks can be used in a lecture, a lab, or at home. HTML and video are mainly used at home.

Usually the difference between lectures and tutorials is that a lecture is much more abstract than a tutorial. Additionally, when using an interactive computer algebra system (CAS) for presentation, the degree of interactivity is much less in a lecture than in a tutorial. The student has to work through a tutorial. In a lecture a student has a much more passive role. Also the difference in tone, which is much more relaxed for tutorials, helps students lose their fear of the abstract formulation of scientific facts.

■ Tutorials

Our tutorials come from several different areas.

1. Simulation (notebook, HTML, some video)
2. Technical Mechanics (notebook)
3. *Mathematica* (notebook, HTML, some video)

■ Video Lectures and Formats

We have also tried different video lecture types, which have several advantages.

1. Students can stop the lecture, think about the new material, and move on.
2. Students can learn when it is convenient for them to learn; they are not tied to any schedule.
3. Students can practice prior to exams. They can listen and watch the lecture over and over until they feel comfortable with the material.

Two different recording mechanisms, available at the University of Freiburg, were used to create the video lectures—first the Camtasia [3] suite and second LECTURNITY [4]. In both cases the presenter's screen and voice are recorded via a Tablet PC, and the presenter is not visible on the recording. In this article we have focused on screen recording mechanisms. Other formats, not considered here, capture the entire lecture as done at MIT [5] or Berkeley [6].

With the Camtasia suite a TSCC codec is used, which can be converted into AVI and RealMedia. As an example of such a screen recording, you can download the RealMedia tutorial, which is in German, from Simulation1_02.rm (47.6MB). The primary advantage is that the RealMedia file format is widely distributed. Secondly, it has a slightly better quality than the AVI format considered here.

A second possibility is provided by the authoring tool LECTURNITY. Again, a recording Tablet PC is used. A Java player is presently available; however, it does not work well under Linux.

■ Notebook Tutorials for the Simulation Curriculum

For students it is necessary to compute simple examples by hand. At some stage, however, much more insight may be gained by computing something that could not be done before; it would be optimal for students to visualize complex real-world examples. Such a success is very motivating. Secondly, we find the visualization of time-dependant problems by an animation is equally well suited for didactical purposes.

1. Derivatives Recovery
2. Finite Difference
3. Finite Volume
4. Finite Elements
5. Iterative Solvers
6. Multigrid Methods
7. Norms in Analysis
8. Partial Differential Equations

9. Shape Functions

10. Sparse Matrices

The simulation tutorials comprise about 180 printed pages. We try not to introduce too many new ideas in too short a time. We start from the simplest possible example and show students the limitations of the code developed so far in order to motivate a next step that fixes the previous problem.

■ Notebook Tutorial Examples

Next, we present two extracts from the *IMS* tutorial section—first the finite difference tutorial and second the finite volume tutorial.

□ Parabolic Finite Differences

Derivation

We look at the equation

$$u_t = u_{xx}. \quad (1)$$

The right-hand side at time step n can be approximated by the central difference

$$u_{xx} = \frac{u_{n,i-1} - 2u_{n,i} + u_{n,i+1}}{h^2}. \quad (2)$$

The left-hand side can be approximated by the *forward difference*. With n being the current time step and $n + 1$ the next time slice

$$u_t = \frac{u_{n+1,i} - u_{n,i}}{\tau}. \quad (3)$$

Putting things together we get

$$\frac{u_{n+1,i} - u_{n,i}}{\tau} = \frac{u_{n,i-1} - 2u_{n,i} + u_{n,i+1}}{h^2}. \quad (4)$$

We rearrange the equation to express explicitly the next time step $n + 1$

$$u_{n+1,i} = \frac{\tau}{h^2} (u_{n,i-1} - 2u_{n,i} + u_{n,i+1}) + u_{n,i}. \quad (5)$$

Initial conditions set the function values for the whole simulation domain for time step $n = 0$. Via the previous scheme we advance the initial conditions to time step $n = 1$. Proceeding in this manner, we compute all unknown values.

Rearranging and setting $s = \tau / h^2$ we get

$$u_{n+1,i} = s(u_{n,i-1} + u_{n,i+1}) + u_{n,i}(1 - 2s). \quad (6)$$

This is called an *explicit scheme* since the value for the next time step is computed explicitly from the previous time steps.

Implementation

Let us implement this explicit scheme and play around with it. As before, for the spatial discretization, we use the variable h . For the time discretization, we use the time step τ .

```
In[1]:= gridPoints = 11;
        tau = 1.0/200;

In[3]:= h = 1 / (gridPoints - 1);
        s = tau / h^2
```

Out[4]= 0.5

The initial conditions at the time step $n = 0$ are set to 20°C. In other words, for any spatial point x we *initially* have a temperature of 20°C.

```
In[5]:= u[0, x_] := u[0, x] = 20.0;
```

A noteworthy point is that we use *dynamic programming* to enhance the speed at which *Mathematica* will find the solution. Dynamic programming means that each computed value will be stored and thus remembered for later execution.

Next, we specify Dirichlet boundary conditions: both ends of the rod at spatial coordinate $x = 0$ and $x = 1$ are set to 100°C. In other words, for any time step the boundary has fixed values.

```
In[6]:= u[n_, 0] := u[n, 0] = 100.0;
        u[n_, 1/h] := u[n, 1/h] = 100.0;
```

The updating rule for the interior values

```
In[8]:= u[n_, i_] :=
        u[n, i] = Evaluate[s * (u[n-1, i+1] + u[n-1, i-1]) +
        (1 - 2*s)*u[n-1, i]];
```

The solution is now found using an outer product, where we only want to see a fraction of the time steps.

```
In[9]:= solution = Outer[u, Range[0, 1/tau/5], Range[0, 1/h]];
```

We map the function that plots a time slice over some selected time steps (i.e., time step 1, 11, 21, 31), which are stored in the solution vector.


```
In[13]:= gridPoints = 21;
         tau = 1.0/100;

In[15]:= h = 1 / (gridPoints - 1);
         s = tau / h^2
```

Out[16]= 4.

The s has now increased.

We set the initial and boundary conditions.

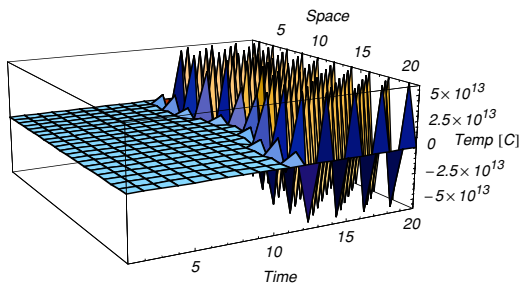
```
In[17]:= u[0, x_] := u[0, x] = 20;
         u[n_, 0] := u[n, 0] = 100;
         u[n_, 1/h] := u[n, 1/h] = 100;
```

The update rule

```
In[20]:= u[n_, i_] :=
         u[n, i] = Evaluate[s * (u[n-1, i + 1] + u[n - 1, i - 1]) +
         (1 - 2*s)* u[n - 1, i]];

In[21]:= res = Outer[u, Range[1/tau / 5], Range[0, 1/h]];

In[22]:= ListPlot3D[res, AxesLabel -> {"Space", "Time", "Temp [C]"},
         ViewPoint -> {2.884, -1.555, 0.845}]
```



This produces a disastrous result. The algorithm became unstable by increasing the number of grid points h and decreasing the number of time steps τ . In this case, for all values of $s \leq 1/2$ the algorithm is stable. Try it. This poses a serious problem: in order for the algorithm to stay stable, the time steps have to be increased to satisfy $\tau \geq 2 * h^2$. For 21 grid points that implies $2 * 20^2 = 800$ time steps.

The first question is where does this behavior come from? The second question is can we circumvent the instability problem related with explicit schemes? The first question will be answered now. The second one is dealt with in the next section, where so-called implicit schemes will emerge—a whole new class of parabolic equation solvers.

```
In[23]:= Remove["u*"]
```

□ Heat Equation with 1D Finite Volume

The Problem (Derivation)

We investigate the numerical approximation to the heat conduction equation that describes the heat distribution in the wall of a pipe. This example is inspired by [7].

Imagine a pipe with a thick wall (Figure 1). Inside this pipe we have a hot liquid. Outside at radius $r = b_2$ it is cold with $T = T_0$. In the inside at radius $r = b_1$ we have a prescribed heat flux q into the pipe's wall.

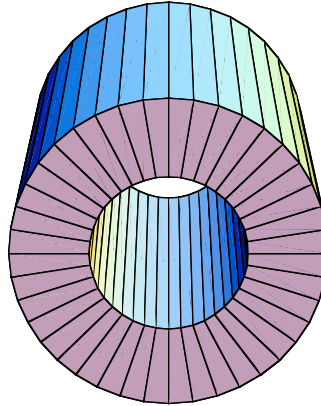


Figure 1. A pipe for transporting hot liquid in cold surroundings.

Figure 2 shows a cross-section of the pipe with the inner radius b_1 and the outer radius b_2 , the boundary conditions, and a thick line along which we compute the solution. It is assumed that this line could be any line from the inside to the outside; the solution of the PDE will not change.

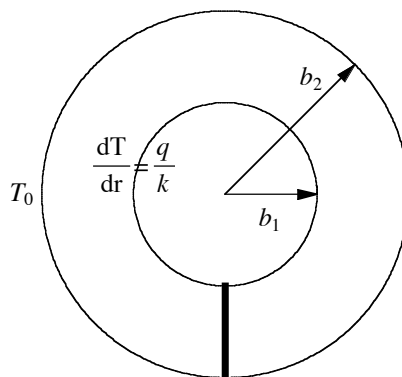


Figure 2. A cross-section of the pipe with the inner radius b_1 and the outer radius b_2 , the boundary conditions, and a thick line along which the distribution of heat in the wall of the pipe is computed.

The governing PDE is

$$\begin{aligned} \frac{1}{r} \frac{\partial}{\partial r} \left(r \sigma \frac{\partial \psi}{\partial r} \right) - \rho &= 0, \\ \frac{\sigma}{r} \frac{\partial \psi}{\partial r} + \sigma \frac{\partial^2 \psi}{\partial r^2} - \rho &= 0 \\ \Leftrightarrow \frac{\partial \psi}{\partial r} + r \frac{\partial^2 \psi}{\partial r^2} &= \frac{\rho}{\sigma} r, \end{aligned} \quad (7)$$

where $f(r) = r \frac{\rho}{\sigma}$ is the inhomogeneity or load of the PDE.

The analytical solution for this is

$$\text{In[24]:= anaSol = DSolve} \left[\left\{ \partial_r \psi[r] + r \partial_{rr} \psi[r] == \frac{\rho}{\sigma} r, \psi'[1] == J, \psi[2] == 0 \right\}, \psi[r], r \right]$$

$$\text{Out[24]=} \left\{ \left\{ \psi[r] \rightarrow \frac{-4 \rho + r^2 \rho + 2 \rho \text{Log}[2] - 4 J \sigma \text{Log}[2] - 2 \rho \text{Log}[r] + 4 J \sigma \text{Log}[r]}{4 \sigma} \right\} \right\}$$

For the numerical solution of the PDE we replace ψ with u

$$\begin{aligned} \frac{\partial \psi}{\partial r} + r \frac{\partial^2 \psi}{\partial r^2} &= \frac{\rho}{\sigma} r \\ \Leftrightarrow \frac{\partial u}{\partial r} + r \frac{\partial^2 u}{\partial r^2} &= \frac{\rho}{\sigma} r. \end{aligned} \quad (8)$$

Meshing

We first mesh the 1D domain $r \in [1, 2]$ with n equally spaced intervals of length h ; the so-called *control volume* j extends from $r_j - \frac{h}{2}$ to $r_j + \frac{h}{2}$ (Figure 3).

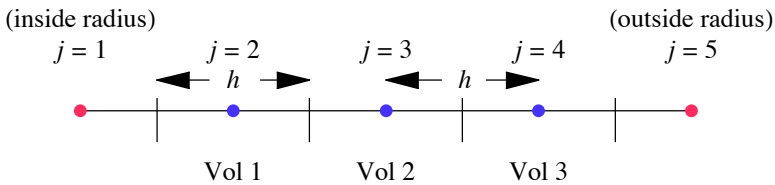


Figure 3. The one-dimensional simulation domain with two red boundary points. The blue points represent additional internal points at which the heat distribution is computed. The points are equally spaced with a length of h . Each internal point is surrounded by a control volume. The control volume of the internal points is also of size h . The control volume of the boundary points is of size $h/2$.

Interior Points

Here we just suppose $u(r)$ changes linearly between any two adjacent discretized points

$$\int_{r_j - \frac{b}{2}}^{r_j + \frac{b}{2}} \left(\frac{\partial u}{\partial r} + r \frac{\partial^2 u}{\partial r^2} \right) dr = \int_{r_j - \frac{b}{2}}^{r_j + \frac{b}{2}} f(r) dr \quad (9)$$

$$\Leftrightarrow \left(r \frac{\partial u}{\partial r} \right)_{r_j + \frac{b}{2}} - \left(r \frac{\partial u}{\partial r} \right)_{r_j - \frac{b}{2}} = \frac{r_j b \rho}{\sigma}.$$

Now we can apply finite difference type discretized derivatives

$$\left(\frac{\partial u}{\partial r} \right)_{r_j + \frac{b}{2}} = \frac{u_{j+1} - u_j}{b}, \quad (10)$$

$$\left(\frac{\partial u}{\partial r} \right)_{r_j - \frac{b}{2}} = \frac{u_j - u_{j-1}}{b}.$$

$$\Leftrightarrow \left(r_j + \frac{b}{2} \right) (u_{j+1} - u_j) - \left(r_j - \frac{b}{2} \right) (u_j - u_{j-1}) = \frac{r_j b^2 \rho}{\sigma} \quad (11)$$

$$\Leftrightarrow \left(-\frac{b}{2} + r_j \right) u_{j-1} - 2 r_j u_j + \left(\frac{b}{2} + r_j \right) u_{j+1} = \frac{r_j b^2 \rho}{\sigma}.$$

Boundary Conditions

The Dirichlet boundary condition:

$$u_{r=2} = 0. \quad (12)$$

For the Neumann boundary condition $\left(\frac{\partial u}{\partial r} \right)_{r=1} = \mathcal{J}$

$$\int_1^{1 + \frac{b}{2}} \left(\frac{\partial u}{\partial r} + r \frac{\partial^2 u}{\partial r^2} \right) dr = \left(r \frac{\partial u}{\partial r} \right)_{1 + \frac{b}{2}} - \left(r \frac{\partial u}{\partial r} \right)_1 = \int_1^{1 + \frac{b}{2}} f(r) dr$$

$$\Leftrightarrow \int_1^{1 + \frac{b}{2}} f(r) dr + \left(r \frac{\partial u}{\partial r} \right)_1 = \left(r \frac{\partial u}{\partial r} \right)_{1 + \frac{b}{2}} \quad (13)$$

$$\Leftrightarrow \frac{2 b \left(\mathcal{J} + \frac{b(4+b)\rho}{8\sigma} \right)}{2 + b} = u_{r=1+b} - u_{r=1}.$$

Building the Matrix (Implementation)

Physical properties

```
In[25]:= J = -1; ρ = 0; σ = 1;
```

```
In[26]:= gridPoints = 5;
h = 1. / (gridPoints - 1);
```

Empty global matrix and empty global load vector

```
In[28]:= matrix = Table[0.0, {gridPoints}, {gridPoints}];
loadVector = Table[0.0, {gridPoints}];
```

Neumann boundary conditions as in equation (13)

```
In[30]:= matrix[[1, 1]] = -1.0;
matrix[[1, 2]] = 1.0;
loadVector[[1]] =  $\frac{2h \left( J + \frac{h(4+h)\rho}{8\sigma} \right)}{2+h};$ 
```

Dirichlet boundary conditions as in equation (12)

```
In[33]:= matrix[[gridPoints, gridPoints]] = 1.0;
loadVector[[gridPoints]] = 0;
```

Assembly of the global system as in the right-hand side of equation (11)

```
In[35]:= Do[
matrix[[pos, pos - 1]] = -h/2 + (1 + (pos - 1) h);
matrix[[pos, pos]] = -2 (1 + (pos - 1) h);
matrix[[pos, pos + 1]] = h/2 + (1 + (pos - 1) h)
, {pos, 2, gridPoints - 1}];
```

Assembly of the global load as in the left-hand side of equation (11)

```
In[36]:= Do[loadVector[[pos]] =  $\frac{(1 + (pos - 1) h) h^2 \rho}{\sigma}$ ,
, {pos, 2, gridPoints - 1}];
```

Solution

Solution of the linear system of equations

```
In[37]:= res = LinearSolve[matrix, loadVector]
```

```
Out[37]:= {0.69122, 0.468998, 0.287179, 0.133333, 0.}
```

Analytical versus Numerical Solution

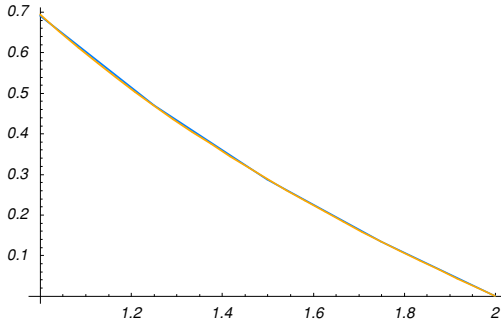
We prepare the data for postprocessing. The coordinates of the mesh points are joined with the result at these points. Next, an analytical solution to the problem at hand is found.

```
In[38]:= numSol = Transpose[Join[{Range[1, 2, h]}, {res}]];
{anaSolFunc} =
 $\psi /. DSolve[\{\partial_r \psi[r] + r \partial_{rr} \psi[r] == \frac{\rho}{\sigma} r, \psi'[1] == J, \psi[2] == 0\}, \psi, r]$ 
```

```
Out[39]:= {Function[{r}, Log[2] - Log[r]]}
```

```
In[40]:= Needs["Graphics']"]
```

```
In[41]:= DisplayTogether[ {
  ListPlot[numSol, PlotJoined → True,
    PlotRange → All, PlotStyle → Hue[.6]],
  Plot[anaSolFunc[r], {r, 1, 2}, PlotStyle → Hue[.1]]
}]
```



This result is informative if we need the quantitative distribution of the heat in the pipe. If we are interested in the quality of the solution, the plot is not very informative. In this case we wish to know how close the numerical solution is to the analytical solution. To this end we evaluate the analytical function at the mesh points.

```
In[42]:= anaRes = anaSolFunc[#] & /@ Range[1, 2, h];
```

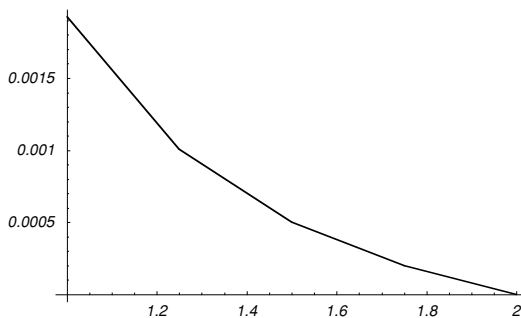
We can then compute the norm.

```
In[43]:= Norm[anaRes - res, 2]
```

```
Out[43]= 0.00224014
```

A better graphical representation is the difference of the analytical solution at the mesh points to the computed result. The error of the numerical solution compared to the analytical solution

```
In[44]:= ListPlot[Transpose[Join[{Range[1, 2, h]}, {anaRes - res}]],
  PlotJoined → True]
```



Experiment with a different number of grid points.

■ Conclusion

In a teaching environment that frequently needs to show parts of program code, we find that using *Mathematica* is of great use—as long as students have control over the speed of the presented material. When a notebook or video is presented, which is full of program code, students may not gain much from the presentation—more than one line of code is often tedious for students. Depending on the level of fluency in the language itself, students face two problems: understanding the contents of the lecture and understanding the program code. We have had good experience with supplementing lectures with tutorials, which may paraphrase the content of the lecture. Tutorials provide a convenient way in which the student can set his own pace of learning, possibly at home. It always fascinates students to see live graphics, and this motivation factor should not be underestimated.

■ Acknowledgment

The authors would like to thank Kai Kratt for investigating exporting the notebook tutorials to HTML and fine-tuning the interactive web tutorials. This project was funded by [8].

■ References

- [1] O. Rübenkönig and J. Korvink, *IMTEK Mathematica Supplement (IMS)*, (2002–2005) www.imtek.uni-freiburg.de/simulation/mathematica/IMSweb.
- [2] H. Abelson and G. Sussman, *Structure and Interpretation of Computer Programs*, 2nd ed., Cambridge, Massachusetts: The MIT Press, 1996.
- [3] *Camtasia Studio*, Version 1.0, Okemos, MI: TechSmith Corporation (Feb 2, 2006) www.techsmith.com/products/studio/default.asp.
- [4] *LECTURNITY*, Version Player 1.6.0, Saarbrücken, Germany: imc AG (Feb 2, 2006) www.im-c.de/lecturnity/en/index.htm.
- [5] MIT OpenCourseWare (MIT OCW), Cambridge: Massachusetts Institute of Technology (Jan 1, 2007) www.ocw.mit.edu/index.html.
- [6] *webcast.berkeley*, Berkeley: University of California (Jan 1, 2007) webcast.berkeley.edu.
- [7] D. R. J. Owen and E. Hinton, *A Simple Guide to Finite Elements*, Swansea, UK: Pineridge Press, 1980.
- [8] *eLectures*, Freiburg, Germany: Fakultät für Angewandte Wissenschaften, University of Freiburg (Oct 26, 2005) electures.informatik.uni-freiburg.de:8484/catalog/project/ss2005.jsp.

About the Authors

Oliver Rübenkönig is a Ph.D. student at the University of Freiburg, Germany. Rübenkönig received his Diploma in Microsystem Engineering in 2001. He uses *Mathematica* extensively in research and teaching and has developed many student exercises and some courses.

Jan G. Korvink obtained his M.Sc. in computational mechanics from the University of Cape Town in 1987 and his Ph.D. in applied computer science from the ETH Zurich in 1993. After his graduate studies, Korvink joined the Physical Electronics Laboratory of the ETH Zurich, where he established and led the Modeling Group. He then moved to the Albert Ludwig University in Freiburg, Germany, where he holds a Chair position in microsystem technology and runs the Laboratory for Microsystem Simulation. Currently, Korvink is dean of the Faculty of Applied Science. He has written more than 130 journal and conference papers in the area of microsystem technology and co-edits the review journal *Applied Micro and Nanosystems*. His research interests include the modeling, simulation, and low-cost fabrication of microsystems.

Oliver Rübenkönig

Jan G. Korvink

Lab for Simulation, Department of Microsystems Engineering (*IMTEK*)

University of Freiburg

Germany

ruebenko@imtek.uni-freiburg.de