1. Introduction
If you make a journey to the core of a plasma simulation code, you will pass various layers. You will first encounter code that deals with the peculiarities of the plasma source under investigation. Next, you will travel through layers that implement more general physical concepts like the equations of Maxwell, Navier-Stokes and Stefan-Maxwell.

In many research papers about plasma simulation, the journey prematurely ends here. Most research is application-driven, and carried out both by and for plasma physicists. As a result, the mathematical foundation at the core of the simulation code is seldomly discussed and the only way to find out about its details is by contacting its author privately.

This situation is somewhat unfortunate. While the various plasma simulations differ in the details of the physical model that they implement, most commonalities can probably be found in this mathematical realm. The session ‘Plasma Modeling: Nuts and Bolts’ gives us a great opportunity to identify opportunities for sharing ideas, experiences—or even code.

2. LinSys overview
In this contribution we will discuss the LinSys library. This is a key infra-structural element of the Plasimo modelling framework [1] that is being developed at Eindhoven University of Technology. LinSys implements such general concepts as tensor values of arbitrary ranks, fields of such values, and systems of coupled equations involving such fields. LinSys has been implemented as a set of C++ class templates that take the fundamental data type — typically double or complex — and the fields’ domain types as template arguments. LinSys can be used with custom domains that must merely obey a small set of interface rules. As a result, LinSys has been successfully used with unstructured 2-dimensional domains and structured meshes in both 1-2 and 3-dimensions.

2. The Plasimo Grid
While we have been careful to design and implement LinSys in such a way that it can be used with custom domain definitions, an exposition of LinSys’ capabilities demands that we choose a particular domain. The demonstrations in the remainder of this text are based on Plasimo’s own numerical mesh code. All calculations in Plasimo are carried out in terms of ortho-curvilinear (OCL) computational coordinates. The advantage of using non-rectangular coordinate systems is that they allow the boundaries of the numerical mesh to coincide with the boundaries of the physical regions of interest. Using such boundary-fitted grids facilitates the imposition of boundary conditions and allows numerically well-behaved boundary discretisation schemes. For a detailed discussion of this topic we refer to [2].

![Figure 1](image-url)
to simulate the electromagnetic field in the Philips QL lamp with Plasimo \[3\]. Plasimo allows simulations to involve multiple, disjoint meshes and the figure shows that—in addition to the curved plasma region—a cylindrical mesh has been used to represent the ferrite core.

The Plasimo grid supports finite volume calculations. When a grid is created, it creates multiple meshes. One represents the nodal points at the centres of the grid cells, the second represents the locations of the centres of the control volume faces, where typically the perpendicular components of flux variables are stored.

While beneficial for many reasons, the usage of ortho-curvilinear coordinates comes at a price: the expressions for such operators as the gradient, curl and divergence that occur in physical equations are more elaborate than their counterparts in Cartesian coordinates. As an example, the Laplacian operator in Cartesian coordinates is given by

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2},$$

while in OCL coordinates \(c_i\) we have

$$\nabla^2 = \frac{1}{l_1l_2l_3} \left[ \frac{\partial}{\partial c_1} \left( \frac{l_2l_3}{l_1} \frac{\partial}{\partial c_1} \right) \right] + \frac{\partial}{\partial c_2} \left( \frac{l_1l_3}{l_2} \frac{\partial}{\partial c_2} \right) + \frac{\partial}{\partial c_3} \left( \frac{l_1l_2}{l_3} \frac{\partial}{\partial c_3} \right),$$

where the coefficients \(l_i\) represent the scale factors of the coordinate system.

Given such complexity, it is obvious that the discretisation of tensor equations in terms of OCL coordinates can be a tedious and error-prone task. And the discretisation of individual expressions is only part of the problem: additional complexity is induced by the fact that in many practical situations multiple equations need to be discretised and solved simultaneously. The fields that are involved in these equations may differ in rank: both scalar and vector or higher-order tensor fields may be involved. Moreover, the fields may be defined on different domains; as an example, some may be defined on the nodal domain, while others are defined on the control volume faces of the region of interest. The result is that quite some index gymnastics is typically required in the discretisation process. In the next section we will show how such situations can be conveniently dealt with using the facilities that LinSys has to offer.

3. LinSys in Detail

In order to understand how LinSys can facilitate various tasks in physics modelling, let us consider a simple example: the equation for the stationary temperature distribution with a linearised source term in a medium:

$$\nabla \cdot \mathbf{\Gamma} = Q_0 + \alpha T,$$  \hspace{1cm} (1)

where \(S = Q_0 + \alpha T\) represents the heat source, and the heat flux density vector \(\mathbf{\Gamma}\) is given by Fourier’s law (section VIII, item 98 in Ref. \[4\]):

$$\mathbf{\Gamma} = -\lambda \nabla T.$$  \hspace{1cm} (2)

In order to solve this equation on a numerical mesh, it is discretised. In this process, the equation is translated into a set of coupled algebraic equations that can be written as

$$Ax = b.$$  \hspace{1cm} (3)

Here \(x\) is the solution vector, that consists of the values of the components of the fields in all points of their domains. The system matrix \(A\) and source vector \(b\) must be assigned appropriate values in the discretisation procedure. Traditionally, code that implements the discretisation of a transport equation like the one above is highly domain- and dimension specific and bears no resemblance to the mathematical notation of the equation that it implements.

Let us now see how such calculation can be set up with LinSys. We will present only the key parts of the code here, and replace arguments to functions with dots when these arguments would merely distract the reader from the essentials. Let us first see how we can set up the system of equations:

```cpp
enum { Dimension=2 };
typedef plGrid<Dimension> Grid;
Grid grid (...);
System<double> system (...);
SystemField<double,Grid::Domain>
   T(system, grid.nodal.domain());
system.resize ();
```

The code starts with the construction of a grid, here in two dimensions. Secondly, a system of equations involving double-valued tensor fields is created. The constructor arguments instruct the system object which matrix solver it will be using. In the next line we create a system field \(T\) that represents the temperature. From the constructor arguments we can see that it is part of the system we have just created, and that it is defined on the nodal mesh. After we have created all fields that are part of the system, we call \(\text{resize}\) on the system, as a result of which the system will allocate memory for the matrix \(A\), the source vector \(b\) and the solution vector \(x\). In this case, the solution vector \(x\) consists (only) of the values of the temperature variable in the nodal points on which it is defined.

The most interesting part is the discretisation of equation\[1\] with the flux given by equation\[2\] With
LinSys, this can be achieved with a single line of C++ code that clearly express the programmer’s intent:

\[
\begin{align*}
-\text{div}(\lambda \ast \text{grad}(T)) &= Q_0 + \alpha \ast T \\
\end{align*}
\]

The term between braces is the equation that needs to be discretised. The second part of this line expresses that this equation applies to all the internal temperature values. In this code we have assumed that a scalar field \(\lambda\), which has been defined on the control volume faces, contains the heat conductivity coefficient.

We see that compound expressions can be built in terms of a small set of primitives, like addition, multiplication and the divergence and gradient operators, which are part of LinSys. Interestingly enough, this code does not depend on the dimension of the problem and makes no assumption about the coordinate system. Last but not least, the same notational convenience can be used to discretise \emph{sets of coupled equations}. All this can be achieved without performance penalty.

In LinSys, there is nothing special about boundary conditions: these are just equations. We may write

\[
(T=100.0). \text{foreach } (T, \text{boundary});
\]

to express that the Dirichlet boundary equation \(T = 100\) K must be discretised.

After discretising the equations, the system can be solved. This can be done with a single call to the member solve, that requires the required solver accuracy as argument:

\[
\text{system.solve}(\text{accuracy});
\]

After this call, the system field \(T\) contains the updated temperature values. It can be written to disk with the line

\[
T.\text{write field} \quad \text{"T.dat"};
\]

At this point is should be noted that the dimensionality of the system is only determined by the number \emph{Dimension} in the definition of the grid type in the first line of this example. That can take the values 1, 2 or 3. Indeed, all code is otherwise dimension-independent. In C++ language, the dimension of the problem is a \emph{template argument} of most grid-related code. In figure 2 we have shown a typical result for the temperature in 1 and two dimensions that has been obtained with the code above.

LinSys provides a number of dedicated operators beyond the standard ones. An example is the convection-diffusion discretiser that discretises the flux on the control volume faces using Scharfetter-Gummel’s exponential scheme \cite{5}. LinSys has an open design that allows users to add operators that are not yet available.

4. Microwave Simulations

Let us now look at the \emph{coupled} problem of calculating the electromagnetic field in microwave plasmas. In the case of harmonic excitation with angular frequency \(\omega\) and Ohmic currents, the Maxwell equations for a quasi-static situation can be written as

\[
\nabla \times E = k_0 h, \quad (4)
\]

\[
\nabla \times h = k_0 \epsilon_r E, \quad (5)
\]

Here the \emph{scaled magnetic field} \(h\) has been defined by dividing \(H\) by the \emph{complex impedance} \(-i \sqrt{\epsilon_0 / \mu_0}\):

\[
H = -i \sqrt{\epsilon_0 / \mu_0} h, \quad (6)
\]
the vacuum wave number has been defined as \( k_0 = \omega / c_0 = \omega \sqrt{\epsilon_0 \mu_0} \), and the complex permittivity includes the normal permittivity and a factor that is related to the conductivity \( \sigma \),

\[ \hat{\epsilon}_r = \epsilon_r + \frac{\sigma}{i \omega \epsilon_0} \quad (7) \]

Figure 3: The Yee algorithm for calculating electromagnetic fields requires that the magnetic field is stored on the nodal points of grid cells, whereas the tangential components are stored on the control volume boundaries.

The system (4-4) is usually solved with the Yee algorithm \[6\]. In this approach, the magnetic field is represented in the centres of control volume cells, while the tangential components of the electric field are defined on the cell faces. In Figure 3, this is illustrated for the case that the electric field has components in the coordinate directions 1 and 2, whereas the magnetic field only has a component in the direction 3.

The discretisation of the system of equations is child’s play with LinSys; it can be carried out with the coordinate system independent code

```c
( yee_curl ( lin (E)) + k0* lin (H)=0
). foreach (H);
( yee_curl ( lin (H)) + k0* eps_r_hat * lin (E)=0
). foreach (E, interior);
```

Figure 4 show a typical simulation result. This has been obtained for an empty cylindrical cavity with a radius of 15 cm and a length of 25 cm. The frequency of excitation is 2.45 GHz. The axis of symmetry is located at the bottom of the figure, the walls are perfect conductors, so no tangential electric field can be supported. In the centre of the wall at the top side, a slit of width 6.25 cm is present, where the tangential electric field has a fixed value of 1 V/m.

4. Conclusions and Outlook

In this paper we have introduced Plasimo’s LinSys library. LinSys facilitates setting up and discretising systems of equations involving field of arbitrary ranks and defined on arbitrary domains. Although we have shown some applications of LinSys for fields that have been defined on Plasimo’s domain types, the library is mostly domain-independent and can be used for arbitrary user-specified domains. We have demonstrated the application of LinSys to a heat transport problem, and to the problem of calculating the electromagnetic field in a microwave cavity. LinSys has also been successfully used to solve the problem of self-consistent ambipolar diffusion, which is governed by the Stefan-Maxwell equations.

Figure 4: Colour map of the amplitude of the reduced magnetic field in an empty cylindrical cavity. For this simulation, the magnetic field has an azimuthal component only. The axis of symmetry is located at the bottom of the figure. The other boundaries are perfect conductors, apart from a small slit that is located at the centre at the top side of the figure. See text for details.

References