

How to enter Partial Differential Equations in Natural (Textbook) Notation into COMSOL Multiphysics®

B. FLUCHE
Berlin, Germany

Abstract

COMSOL Multiphysics® ships the Weak Form PDE interface in its base package, so in principle any licence can solve an arbitrary partial differential equation without an add-on module. In practice almost nobody does, because the interface expects a weak form and deriving one by hand is a specialist skill. This paper describes a simulation app, built entirely in the Application Builder in Java, whose objective is to remove that step: the user types a system of equations the way a textbook prints it — $\nabla^2 u = f$, $\nabla \times E = -\partial B / \partial t$, $\nabla \cdot E = \rho$ — and the app parses it, generates the corresponding weak form, builds the geometry, physics, mesh and study, solves, and plots the result together with derived quantities. The same input path reaches the heat equation, the wave equation, advection–diffusion, a fourth-order biharmonic system, scalar and vector eigenvalue spectra, magnetostatics, incompressible flow with both driven and lid-driven boundaries, and — as the most demanding case — the complete Maxwell system with permittivity, permeability, conductivity, impressed current and a charge density carried as a live global ODE. An integrated test suite of 308 assertions runs from a button inside the application. The paper reports on the architecture, the verification numbers, and — with equal emphasis — what the app deliberately does not do: it gives direct access to the Weak Form PDE interface through PDE input in textbook notation, but it does **not** replace the AC/DC, RF, Wave Optics or CFD Modules of COMSOL.

Keywords: weak form, Application Builder, simulation apps, code generation, Maxwell's equations

Objective: entering PDEs in natural form

The Weak Form PDE interface is part of the COMSOL Multiphysics® base package. Any licence can therefore, in principle, solve any partial differential equation that can be written as a weak statement. The obstacle is not the licence. It is the notation. A user who wants to solve a Poisson problem in a weak form must supply not $\nabla^2 u = f$ but its weak counterpart, and a user who wants a coupled first-order system must supply a variational statement of it that is also stable. Deriving that by hand, correctly, is a skill that most of the engineers and scientists who need the answer do not have and should not have to acquire.

The objective of the work reported here is exactly this one step, and nothing more ambitious: let the user write the equation as it is printed in a textbook and let the application do the rest. A simulation app built in the Application Builder accepts a system of equations in natural notation, parses it, generates the weak form, and then drives the entire COMSOL workflow — geometry, physics, boundary conditions, mesh, study, solver, post-processing — without the user ever opening the Model Builder. The design target throughout has been textbook notation in, solved fields out.

Natural notation here means the operators as they are written on paper. The input palette offers the Laplacian ∇^2 , the divergence $\nabla \cdot$, the curl $\nabla \times$ and the gradient ∇ ; the time derivatives $\partial / \partial t$ and $\partial^2 / \partial t^2$ and

the eigenvalue λ ; the spatial derivatives $\partial / \partial x$, $\partial / \partial y$, $\partial / \partial z$; and two composite operators, $\nabla \cdot (\kappa \nabla)$ for a diffusion term with a variable coefficient and $\nabla \times \nabla \times$ for the double curl. A palette button inserts the operator; the user may equally type it. Figure 1 shows the palette as the application presents it.

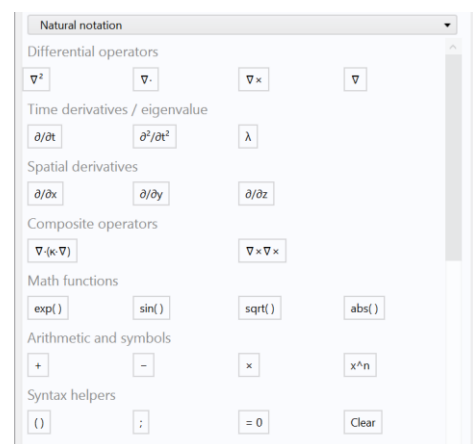


Figure 1. The textbook notation palette. Every operator the parser understands is on one page: ∇^2 , $\nabla \cdot$, $\nabla \times$ and ∇ ; the time derivatives $\partial / \partial t$ and $\partial^2 / \partial t^2$ and the eigenvalue λ ; the spatial derivatives; the composite operators $\nabla \cdot (\kappa \nabla)$ and $\nabla \times \nabla \times$; the usual mathematical functions; and the syntax helpers, including the semicolon that separates the equations of a coupled system.

A second input route is offered for users who think in terms of COMSOL's own coefficient form rather than in operators. The application exposes the coefficient-form template directly, with one field per coefficient, and generates the same kind of model from it. Figure 2 shows that second palette. The two routes share the geometry, boundary-condition, solve and post-processing stages; they differ only in how the equation reaches the parser.

Figure 2. The coefficient-form palette, the alternative input route. The template at the top is COMSOL's own coefficient form, and each coefficient has its own field: the mass and damping coefficients on the time derivatives, the diffusion coefficient c , the two convection vectors α and β , the absorption a , the source f and the conservative flux source γ . Here c has been given the name ρ and the damping coefficient the value 1.

Two design commitments have shaped everything else. The first is that the application refuses visibly rather than answering quietly and wrongly: where a formulation would be unstable, or a structure is not one the app recognises, it declines and states the reason in the review card. The second is that every capability is either enforced by an automated regression suite or explicitly labelled as verified by hand.

The weak form in plain terms

Why COMSOL asks for a weak form at all

A finite-element program cannot work with $\nabla^2 u = f$ as written. The standard way out is more than two centuries old and is not specific to COMSOL: instead of demanding that the equation hold at every point, demand that it holds on average against every member of a family of test functions.

In practice, one multiplies the residual by a test function v , integrates over the domain, and then

integrates by parts. Integration by parts is the whole trick: it moves one derivative off the unknown and onto the test function, so that second derivatives never have to exist. For $\nabla^2 u = f$ the result is the familiar statement that the integral of $-\nabla u \cdot \nabla v + f v$ must vanish for every admissible v . That is the weak form, and in COMSOL syntax it reads

```
-(test(ux)*ux + test(uy)*uy) + f*test(u)
```

which is precisely the string the user would otherwise have to write by hand. The parser produces it — and its variants with variable coefficients, transport terms, reaction terms and time derivatives of first and second order — by rewriting a parsed expression tree term by term, not by matching patterns in the input string.

Inside the application

Forms, methods and libraries

The application is an ordinary Application Builder project with three kinds of content, and Figure 3 shows all three. The forms are the pages the user sees, one per stage of the workflow: equation input, parameters, geometry, boundary conditions, solve and results, plus the two palettes of Figures 1 and 2. The methods are the actions: parse, create geometry, react to a geometry change, build the model, update the boundary selection, add a boundary condition, refresh the boundary-condition log, solve, update the time plot, toggle the plot, and export image or data. The libraries are the part that carries the intelligence, and they are the subject of the next section.

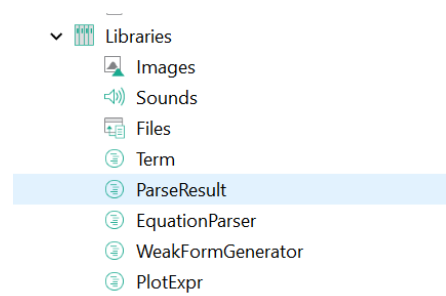


Figure 3. The Libraries branch of the Application Builder tree. Five entries — *Term*, *ParseResult*, *EquationParser*, *WeakFormGenerator* and *PlotExpr* — are the app's own Java classes and hold everything that understands mathematics; *Images*, *Sounds* and *Files* are Application Builder built-ins and are unused here.

The five library classes

Everything that understands mathematics lives in five Java classes in the Libraries branch. They depend on nothing outside `java.util`, and they compile outside COMSOL. That last property

matters more than it first appears: the equation layer can be compiled and tested without a COMSOL licence, and a large part of the verification programme rests on that.

Term is the atom. One instance is one additive term of one equation, reduced to four fields: a type, a sign, a coefficient and a variable. The types are the operators of the palette — first and second time derivative, Laplacian, gradient, divergence, curl, the three spatial derivatives — plus LINEAR for a term such as $k \cdot u$ and SOURCE for everything that does not contain the unknown. Reducing an equation to a list of Terms is what makes the later stages structural rather than textual: no stage after the parser looks at the user's string again.

ParseResult is what one parse produces: the term lists, one per equation component, together with the detected dependent variables, the parameter list, the recognised structure flags, and the generated weak forms for every route the system admits — stationary, Galerkin transient, per-step transient, eigenvalue. It also carries the refusals. When a route is not available, ParseResult holds the reason as a sentence, and the app displays that sentence instead of silently offering a study type that would not be valid. The class also owns the classifier, described below, so that classification is defined in exactly one place.

EquationParser turns a string into a ParseResult. It normalises the input (newlines and semicolons are both equation separators), protects mathematical function calls and power expressions behind placeholders so that they survive term splitting, expands the vector operators into components, flattens parenthesised sums, splits into terms at the top level only, and classifies each term. It also contains the recogniser for coupled vector systems, which turns $\nabla \cdot$ and $\nabla \times$ equations in several declared vector unknowns into residual atoms and assembles the least-squares and Galerkin forms from them. Where it cannot classify a term, it records the term and refuses; it does not drop it.

WeakFormGenerator performs the rewriting described in the previous section: each Term becomes its weak counterpart in COMSOL syntax, a Laplacian becoming a test-gradient product, a source becoming a test-function product, and so on. It is a pure function of the term list and the spatial dimension, which is why the same generator serves the scalar, coupled and vector paths.

PlotExpr is the mirror image of the parser and serves the post-processing stage. The solver knows the fields only as slots $u_1, u_2, u_3, \dots$; the user thinks in terms of E, B and u . PlotExpr translates a plot expression written in field notation into slot notation — a component, a magnitude, a derivative of a magnitude, a curl component, a divergence, a

Poynting component — and passes anything it does not recognise through unchanged, so that a raw COMSOL expression still works. Asked to plot a bare vector, it refuses and names the alternatives.

The processing chain

Around those five classes the methods form a chain of four stages. The parser produces the typed term lists. A classifier maps the parsed structure onto a physics class — scalar, coupled scalar, single curl, curl-curl, curl-curl eigenvalue, Maxwell (E, B), Navier-Stokes 2D — and from that class derives the element type, the boundary-condition palette and the study type. The generator rewrites the terms into COMSOL syntax. Geometry is then created directly in the application, chosen from a small set of 2D shapes (rectangle, circle, ellipse, ring) and 3D shapes (box, sphere, cylinder). Boundary conditions — Dirichlet, Neumann, Robin, slip/no-slip for fluid dynamics, and tangential for vector fields — are set next. The final stage is solving and post-processing; the complete workflow runs inside the application, and the Model Builder is never needed.

What the application uses from COMSOL

The app uses only interfaces present in the base package: the Weak Form PDE interface in one-, two- and three-component form, the Global ODEs and DAEs interface for the continuity equation, the Coefficient Form PDE interface for the coefficient-form route, and the Stationary, Time Dependent, Time Discrete and Eigenvalue studies. No AC/DC, RF, Wave Optics, CFD, MEMS or other add-on module is used at any point. This is the central practical claim of the work: the physics reported below is accessible to any COMSOL licence holder.

The target system: Maxwell's equations

The system that defines the upper end of the scope of the app is the complete system of Maxwell equations with constant permittivity and permeability and with conductivity, entered by the user as a single string — $\nabla \cdot E = \rho, \nabla \cdot B = 0, \nabla \times E = -\partial B / \partial t$ and $\nabla \times B = \mu \epsilon \cdot \partial E / \partial t + \sigma E + J$ — which in the ASCII of the input field is

```
div(E)=rho_n ; div(B)=0 ;
curl(E)=-dt(B) ;
curl(B)=me*dt(E)+sg*E+J_e
```

with me for the product $\mu \epsilon$, sg for the conductivity, J_e for an impressed current density and rho_n for a normalised charge density. The two vector unknowns are declared as E, B in the field below the equation box; that declaration is what tells the parser that E and B are vectors rather than scalars, which no notation can express by itself. Figure 4

shows the input.

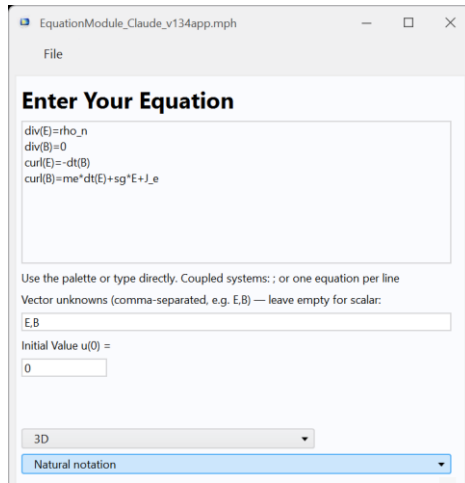


Figure 4. The complete system of Maxwell equations with constant material properties as it is typed. Four equations in textbook notation, the two vector unknowns declared as E , B , three dimensions, textbook-notation input mode. No weak form is written by the user at any point; the app generates all four routes — stationary, Galerkin transient, per-step transient and eigenvalue — and reports which of them this system admits.

The app solves this on a dimensionless code axis with the wave speed set to unity, so that one time unit is one domain length of light path. All numerical values reported below are on that axis; converting them to SI is a similarity scaling that the app does not currently apply, and that is stated again among the limitations.

The charge density is not merely a source term. Charge conservation, $\partial \rho / \partial t + \nabla \cdot \mathbf{J} = 0$, is carried as a global ODE degree of freedom that the app creates itself, so that in a conducting medium the charge relaxes as $\exp(-\sigma t / \epsilon)$ while the field equations are integrated. The continuity equation is therefore a live, regression-tested part of the solved system rather than a condition imposed on it. The app also detects the one combination that is legal to solve but unphysical — a non-zero conductivity together with a constant non-zero charge density — and says so.

Results: how far one parser reaches

The Maxwell system is the highlight, and it is reported first. It would be a misleading highlight on its own, though: the point of a general equation module is that the same input path, the same five classes and the same suite also carry the ordinary problems that most users have.

The complete Maxwell system

The target system is solved end to end. In the lossless cavity the total field energy is conserved

over a hundred steps, the spurious content stays below 10^{-3} of the total, and the modal projection at two time units matches the analytic sign and magnitude. With finite conductivity the energy decays instead, and with the charge density active the global ODE reproduces the analytic relaxation: at half a relaxation time the solved charge is 0.606524 against $\exp(-0.5) = 0.606531$.

Cavity eigenmodes

The PEC unit-cube cavity is the sharpest available test of the spatial operator, because its spectrum is known exactly: the admissible wavenumbers are $\pi \sqrt{n}$ for those integers n whose index triple contains at least two non-zero entries. The app reproduces the levels at $n = 2, 3, 5$ and 6 — the lowest of them to a relative deviation of about $3 \cdot 10^{-4}$ — and, equally important, does not produce the inadmissible $n = 1$ and $n = 4$ levels. The computed values are collected in Table 1. The boundary penalty enforcing both the tangential- E and the normal- B condition is applied as a mesh-scaled term, so that the penalty-to-stiffness ratio stays constant under refinement.

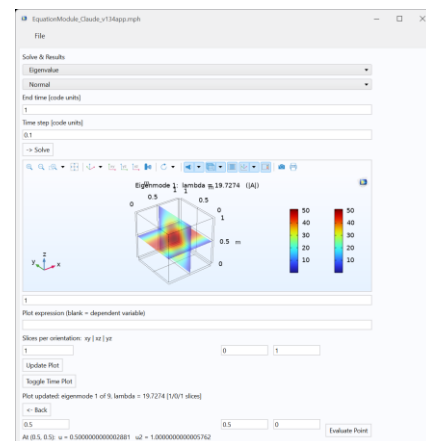


Figure 5. Lowest computed eigenmode of the PEC unit-cube cavity, shown as the field magnitude on three slice planes. It is the $n = 2$ level, whose analytic wavenumber is $\pi \sqrt{2}$.

The transient scalar family: heat, diffusion, reaction and waves

Heat conduction, diffusion, reaction–diffusion and the scalar wave equation were each run against a closed-form reference in two and, where applicable, three dimensions. On the unit square with a $\sin(\pi x) \sin(\pi y)$ initial condition and homogeneous Dirichlet boundaries, the heat equation $\partial u / \partial t = \nabla^2 u$ decays as $\exp(-2\pi^2 t)$, and the app follows that decay across a fall of three orders of magnitude, with a deviation of about -1.6% at $t = 0.1$ that is a time-stepping error and not a spatial one. The same holds in three dimensions, where the rate is $\exp(-3\pi^2 t)$. The diffusion case was run with the

dependent variable named c rather than u , to exercise the variable-name handling, and agrees with its reference to a few thousandths of a per cent. The reaction–diffusion case reaches its predicted plateau. The wave equation $\partial^2 u / \partial t^2 = \nabla^2 u$ inverts completely at the half period and recovers full amplitude at the full period. The wave case also measures the time integration itself: over three time units the computed period runs about 0.7 % long. This is not a defect of the formulation but the cost of COMSOL's default time stepping, which the app does not currently tighten; exposing those settings is the clearest single improvement available to the app.

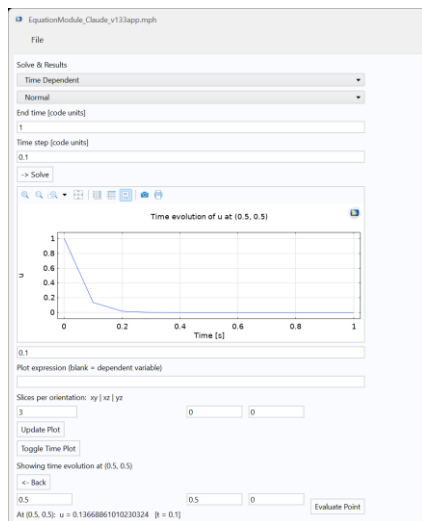


Figure 6. The heat equation $\partial u / \partial t = \nabla^2 u$ on the unit square with a $\sin(\pi x)\sin(\pi y)$ initial condition and homogeneous Dirichlet boundaries: the decay of the centre value against time, taken from the application's own time-evolution plot. Everything in the window — the study type, the end time, the time step, the probe point — was set from inside the app.

Fourth order: the biharmonic system

Fourth-order problems are entered as a coupled pair, which is how they are solved in practice: $\nabla^2 u = w$ and $\nabla^2 w = f$, two second-order equations sharing an auxiliary unknown, separated by a semicolon in the input field. Nothing in the parser is specific to the biharmonic operator — it is the ordinary coupled-scalar path — and the solution matches its closed-form reference to better than a thousandth of a per cent for both fields. The same path carries coupled systems of two, three and four fields, verified against linear solutions of known slope.

Transport and the boundary layer

Advection–diffusion, $\nabla^2 u - \mathbf{b} \cdot \nabla u = 0$ with a given velocity vector, exercises the composite convection operator: the app expands $\mathbf{b} \cdot \nabla u$ into components and exposes b_x , b_y and b_z as parameters. On the

unit square in a transport-dominated setting the app reproduces the analytic centre value to five significant digits, and the shape of the boundary layer is asserted at a second point. The variable-coefficient diffusion operator $\nabla \cdot (\kappa \nabla u)$, which is a single palette button, is verified separately in two and three dimensions.

Magnetostatics, and the value of running the test

The gauged single-curl system $\nabla \times \mathbf{H} = \mathbf{J}$ with $\nabla \cdot \mathbf{H} = 0$ is solved in both two and three dimensions. At the centre of the unit cube the computed field components agree with the exact ± 0.25 to a few parts in 10^5 , the curl reproduces its unit value to the same accuracy, and the divergence gauge holds below 10^{-8} . On the boundary, where the Dirichlet condition is imposed, the same quantities agree with the exact values to the printed digits; the interior agreement quoted above is the honest measure of the discretisation error.

Incompressible flow by structural recognition

Some systems are better solved by built-in recognition than by a generated weak form, and incompressible flow is one of them. The app therefore contains a structural promoter: when the parsed system has the shape of two momentum equations sharing a pressure-like unknown plus a divergence-free constraint, it is recognised as two-dimensional Navier–Stokes and handed to a curated equal-order formulation with pressure stabilisation. The recognition is structural, not lexical — a pure function of the parsed component lists — and both the recognition and its near-misses are covered by the suite: the same shape in three dimensions, and the same two momentum equations without the continuity constraint, are deliberately not promoted. The distinction is worth stating precisely, because it is easy to overclaim. The system typed by the user is recognised from natural notation — that is the contribution — but what is then solved is a curated, stabilised recipe rather than the generated weak form, and the app says so in the review card. Two references are used. In plane Poiseuille flow (Figure 7) the centreline velocity, the parabolic shape and the pressure are compared with the closed-form solution. The lid-driven cavity (Figure 8) uses the same equations with a tangential velocity on the top boundary and no-slip on the other three; only the boundary conditions differ, and they are set on the boundary-condition page of the app. The flow answers with the single recirculating eddy that is the classical benchmark of Ghia et al. at Reynolds number 100. On the mesh of Figure 8 the centre velocity lies 3.7 % below the tabulated value; on the coarser mesh pinned in the regression suite the gap is 9.1 %. Both are mesh and stabilisation effects, and both are reported as such rather than tuned away.

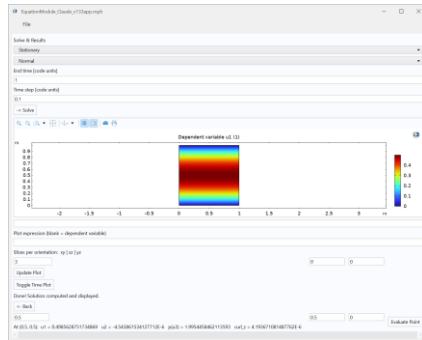


Figure 7. Plane Poiseuille flow, solved from the structurally recognised two-dimensional Navier–Stokes system with $\rho = \mu = 1$, a parabolic inlet profile, no-slip walls and a free outlet. The point evaluation at the bottom of the window reports the centreline velocity, the transverse velocity, the pressure and the vorticity: three of the four have closed-form references, and one is a derived quantity rather than a solved one.

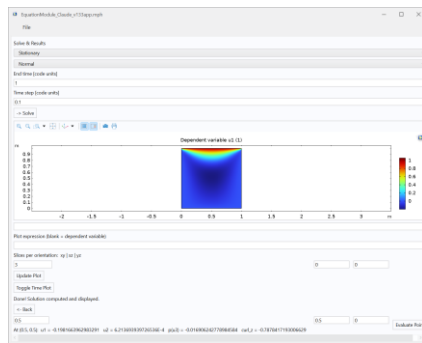


Figure 8. The same Navier–Stokes system, the same recognition path, one change in the boundary conditions: a tangential velocity on the top boundary and no-slip on the other three. The horizontal velocity component shows the driven layer under the lid and the recirculating eddy filling the cavity. The point evaluation reports the centre velocity, which is compared in Table 1 with the value tabulated by Ghia et al. [3] for Reynolds number 100.

Derived quantities

The app plots not only the solved fields but quantities derived from them — the current density σE , the displacement field ϵE , the magnetic field B/μ , the Poynting vector $E \times H$, and the curl or divergence of any solved vector field. Also, for a scalar field u derived quantities can be plotted — e.g. for a 2D problem the total gradient $T = \sqrt{u_x^2 + u_y^2}$ as a parameter for shape recognition of or inside the domain.

Non-linear systems

Non-linear scalar problems — Burgers' equation, the Allen–Cahn equation, the Bratu problem, and the general family with u^2 , $\exp(u)$ or $\sin(u)$ source terms — are parsed and generate correct weak forms in two and three dimensions, with the study type inferred from the presence and order of time

derivatives.

System	Analytic reference and agreement
Maxwell (E,B), lossless cavity	energy invariant conserved; spurious $< 10^{-3}$
Charge relaxation, $\sigma \neq 0$	$\exp(-\sigma t/\epsilon)$; 0.606524 vs 0.606531
PEC cavity eigenmodes	$\pi\sqrt{n}$, $n = 2, 3, 5, 6$; no inadmissible levels
Helmholtz eigenvalues, 2D	$2\pi^2 = 19.7392$; degeneracies reproduced
Magnetostatics, 2D and 3D	± 0.25 ; interior -0.249986 , curl 0.999965
Navier–Stokes, Poiseuille	parabolic profile; pressure 1.99545 vs 2
Navier–Stokes, lid-driven cavity	$-0.198166 / -0.187178$ vs Ghia -0.20581
Advection–diffusion	0.0758575 vs 0.0758577
Derived-quantity error	field error; amplification 1.00
Heat equation, 2D and 3D	0.136689 vs $\exp(-2\pi^2 t) = 0.138911$; over 3 decades
Diffusion, variable named c	closed form; 0.0015 – 0.0033 %
Wave equation, 2D and 3D	period 1.414214; $+0.68$ %
Biharmonic system	closed form; 0.00025 % / 0.00012 %
Poisson, circular domain	$(1 - r^2)/4$; centre value 0.25

Table 1. Verified systems and their analytic references

What the application is not: it does not replace the modules

The value of this app is that a single PDE or a whole system can be solved by entering it in textbook notation. But it is not, and is not intended to be, a substitute for COMSOL's physics modules, and the difference is not a matter of maturity that further work would close. A physics module is a curated package of boundary conditions, material models, mesh strategies, solver settings and post-processing that encodes decades of engineering practice. This app generates a weak form. Those are different things.

No engineered electromagnetic boundary conditions. The app offers Dirichlet, Neumann, Robin, zero flux, tangential and velocity conditions. It offers no ports, no wave excitation with mode analysis, no terminals or lumped elements, no impedance or transition conditions and no perfectly matched layers. Radiating problems, S-parameter sweeps, coaxial feeds, antenna far fields and capacitance matrices are what the AC/DC, RF and

Wave Optics modules exist for, and none of them is reachable here. The unbounded domain is the sharpest example: without a PML or an absorbing condition, only a closed region with an energy-conserving or prescribed boundary can be solved.

No turbulence modelling. The flow that this app solves is laminar, steady, incompressible and two-dimensional, from a curated equal-order recipe with pressure stabilisation. There are no turbulence models, no wall functions or wall treatment, no turbulent transport, no multiphase flow and no conjugate heat transfer. The CFD and Heat Transfer modules are not approximated here; the Ghia comparison above, 9 % from the tabulated value, is a fair indication of the register the app plays in.

No material libraries and no engineering post-processing. Coefficients are numbers the user types. There is no material database, no temperature- or frequency-dependent material model, and no built-in evaluation of forces, losses, impedances, quality factors or scattering parameters. The app plots fields and the derived quantities listed above.

The honest statement of the scope is therefore this: the app makes arbitrary partial differential equations accessible on a base licence, which is useful for teaching, for exploratory modelling, for problems that no module covers, and for anyone who wants to see the weak form of their own equation. Where a module exists for the problem at hand, the module is the better tool.

Further limitations of the present version

Units. The app solves on a dimensionless code axis. Reported times, frequencies and field magnitudes are in code units, and converting them to SI is a similarity scaling that the app does not perform. A scaling layer is planned; it is not implemented.

The vector wave equation. The second-order vector formulation is not yet implemented. A separate Model Builder study established that COMSOL will accept a second time derivative in a weak form on Nedge elements under a Time Dependent study, which is not documented, and sized the changes the app would need. The capability is proven feasible and is deliberately outside the claim of this paper. The scalar wave equation, by contrast, is implemented and verified.

Curated versus parsed. Two-dimensional Navier–Stokes and time-harmonic Maxwell in A-formulation are solved by curated recipes. In the Navier–Stokes case the system is recognised from natural notation first; in the A-formulation case entry is by keyword.

Suite-gated versus hand-verified. Fourteen systems are enforced by the regression suite on every run. The transient scalar family, the non-linear family, the biharmonic system, Poisson with

a source on a circular domain, and one coefficient-form case are verified against analytic references but are not yet suite-gated. Promoting them is the most valuable remaining work on the verification side.

Solver strategy. Curl–curl problems use a direct solver; no iterative solver with an auxiliary-space preconditioner is offered, which bounds the mesh sizes that are practical.

Conclusions

A simulation app built entirely in the Application Builder can remove the weak-form derivation step for a substantial class of partial differential equations and can do so on a base COMSOL Multiphysics® licence. The same input path — one palette, one parser, five Java classes — carries the transient scalar family, a fourth-order biharmonic system, scalar and vector eigenvalue spectra, magnetostatics, incompressible flow by structural recognition, non-linear problems, and the complete Maxwell system with conductivity, impressed current and a conserved charge density, all listed in Table 1. An integrated suite of 308 assertions enforces fourteen of them on every run; the remainder are verified against analytic references and labelled accordingly.

Future work is ordered by value: the second-order vector formulation, the scaling layer that would allow SI input and output, exposure of the transient solver settings, and promotion of the hand-verified systems into the regression suite. What is deliberately not on that list is any attempt to reproduce what the physics modules do. The app is a door into the Weak Form PDE interface, and it is most useful when it is understood as exactly that.

References

- [1] COMSOL AB, COMSOL Multiphysics® Reference Manual, version 6.4, Stockholm, 2025.
- [2] COMSOL AB, Introduction to the Application Builder, version 6.4, Stockholm, 2025.
- [3] U. Ghia, K. N. Ghia and C. T. Shin, High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method, *J. Comput. Phys.* 48, 387–411, 1982.

Acknowledgements

This application was developed using Anthropic Claude Opus 4.x/5.0. The programming of the Java code was entirely done by this powerful AI system. I also thank Dr. Sven Friedel (COMSOL Switzerland) for the critical and constructive proofreading of this paper before being submitted.