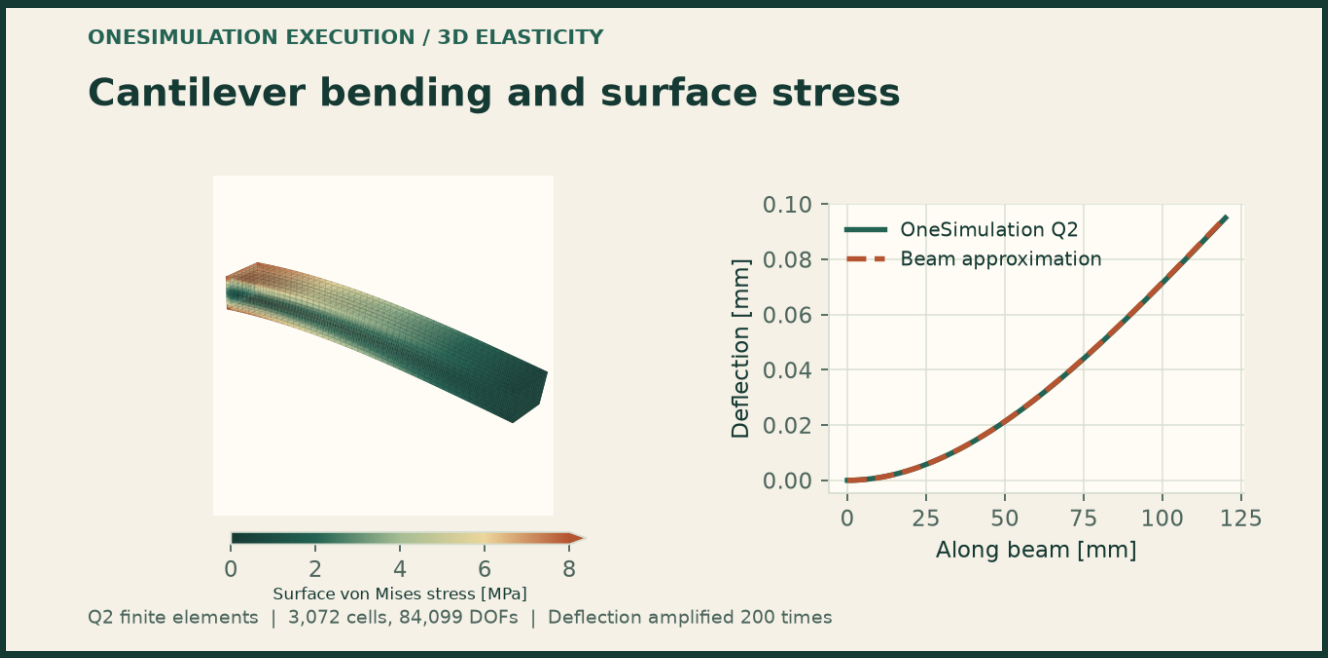
ONESIMULATION / REPOSITORY SHOWCASE

A Computational Atlas

Physics, numerical methods and CAD in Python



16 PHYSICS PACKAGES / 48 DIMENSIONAL CASE STUDIES

FEM, FVM, FDM / TIME INTEGRATION / CAD / PYTHON NOTEBOOK

View OneSimulation Runtime Alpha on Gumroad

Contents

Select a section to jump to its first page.

ORIENTATION

Orientation and reading guide 3

Runtime Alpha and delivery scope 5

PHYSICS ATLAS

Acoustics 6

Biophysics 9

Chemistry and species transport 12

Electromagnetics 15

Electrostatics 18

Geotechnics 21

Heat transfer 24

Hydrodynamics 27

Magnetohydrodynamics 30

Magnetostatics 33

Multibody dynamics 36

Multiphysics coupling 39

Optics 42

Plasma physics 45

Solid mechanics 48

Lumped systems and networks 51

Model and service catalog 54

ONESIMULATION RESULTS

OneSimulation runs and recorded fields 58

Advanced recorded studies 70

DISCRETIZATION

FEM, FVM and FDM 74

TIME INTEGRATION

Time integration 82

SOLVERS

Direct, iterative and nonlinear solvers 94

CAD AND MESHING

CAD, geometry and meshing 99

NOTEBOOK WALKTHROUGH

Complete Python notebook 104

Result queries, visualization and export 109

CUSTOM EQUATIONS

Define and solve your own equation 112

PARALLEL COMPUTING

Parallel computing and HPC 117

APPENDIX

Publication coverage by namespace 120

Appendix and export inventory 122

A library for computational experiments

OneSimulation brings geometry, material laws, physics, numerical schemes and field output into a Python workflow.

Define Build a geometry and select domains and boundaries. Specify material and physical laws, initial values and boundary data.

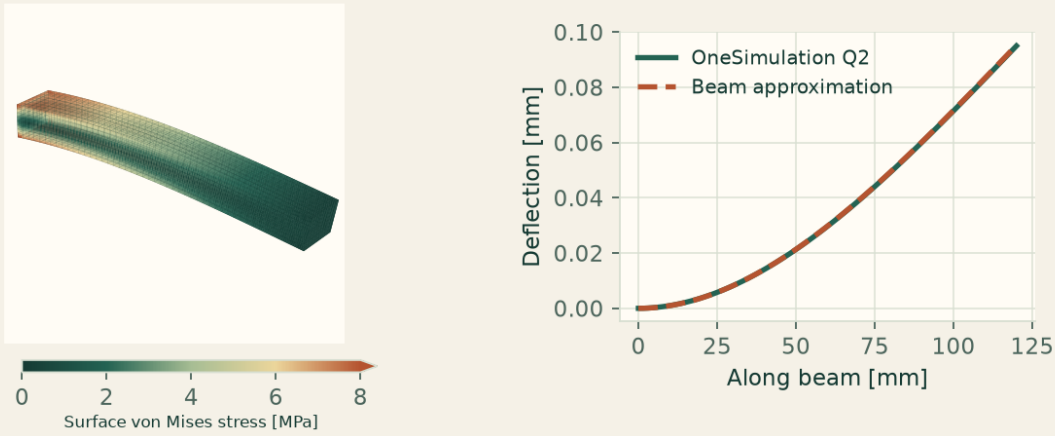
Discretize Choose the mesh and representation for the unknowns. Select a time integrator for transient problems and a compatible algebraic solver for the resulting equations.

Solve Run the stationary or transient study. Validate the declared model first, then examine the numerical provider and diagnostics recorded by the run.

Interrogate Sample fields along lines and planes, calculate derived quantities or integrals, compare to a reference, and export figures or data.

ONESIMULATION EXECUTION / 3D ELASTICITY

Cantilever bending and surface stress



Q2 finite elements | 3,072 cells, 84,099 DOFs | Deflection amplified 200 times

A repository demonstration. A 3D elastic volume yields displacement, a sampled stress surface, and a comparison against a beam approximation.

API IN PRACTICE

Compose the model, mesh and numerical method explicitly. This is the study entry point used by the stationary cantilever example.

```
from onesim.simulation import Simulation

simulation = Simulation(problem=problem, mesh=mesh, numerics=numerics)
simulation.validate().require_valid()
result = simulation.solve_stationary()
```

How to read the evidence

The atlas separates a useful physical benchmark from a verified code path. Both matter when choosing a method and interpreting a result.

Label	Meaning
OneSimulation execution	The companion scripts executed the project source through its declared model, study or numerical provider. Metrics accompany these figures.
Recorded OneSimulation execution	The figure is redrawn from existing OneSimulation samples or mesh data. The book identifies the source dataset and observable.
Independent reference	An exact relation, manufactured field, reduced calculation or numerical comparison in the book's own scripts. Use it as a benchmark, not as a runtime result.
API inventory	A declaration or package export is present in the inspected source. A model, mesh, field space and method must still form an executable combination.

> **Dimensions** The atlas has a 1D, 2D and 3D presentation for every physics package. Some are actual spatial fields. Others show a line trace, plane, trajectory, graph layout or an analytical limit. Dimensionality is described on each page.

> **API excerpts** Each case shows a small part of the Python setup. Names such as box, material or result refer to objects you supply. The excerpt shows the relevant declarations or field queries. See the full notebook for a complete study.

> **Project and distribution** The atlas covers the repository. Gumroad lists the packaged Runtime Alpha and its supported delivery scope. Access to an API name is different from qualification of all models, dimensions and methods.

Runtime Alpha and its example studies

OneSimulation Multiphysics Alpha is distributed as an installable Python runtime. This atlas also covers the broader repository.

Example study of Runtime Alpha 0.2.1a2	What it demonstrates
Radiative cooling	Thermal mass, radiative boundary law and an analytical reference
Damped oscillator	Lumped mechanics, energy and time integration
3D cantilever	Quadratic FEM, displacement, stress and line queries
Circulating heat transport	Temperature transport with prescribed flow and 2D field views
3D magnetostatics	Resolved coil, edge elements and magnetic induction

> **Where the examples are** The example studies are free to read and download in the online example library at johmey.github.io/OneSimulation-Showcase. The runtime package itself contains no notebooks.

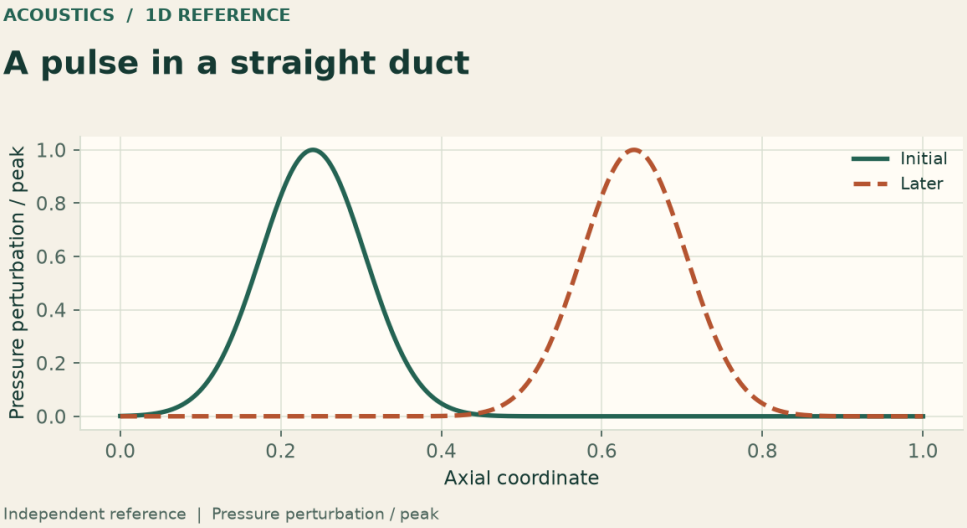
> **Runtime and license** The runtime uses standard 64-bit CPython 3.13 on macOS Apple Silicon, Linux x86-64 or Windows x64. Activation and periodic license renewal require internet access. Availability, plans and prices are listed on the product page.

> **Additional repository examples** The Sod, rotating-disc VSI and ECR studies in this atlas have separate recorded evidence. They are additional repository examples and use functions beyond Runtime Alpha 0.2.1a2.

OneSimulation Multiphysics Alpha on Gumroad

A pulse in a straight duct

Pressure and particle velocity are the state variables of a linear acoustic wave. The package also exposes ultrasonic propagation and reduced models for pressure-driven or radiating mechanical modes.



Independent reference. Pressure perturbation / peak.

$$p(x,t) = g(x - ct), \qquad v(x,t) = p/(\rho_0 c)$$

Setup Launch a Gaussian pressure pulse in a uniform duct. A matched velocity perturbation excites the right-going characteristic. The comparison snapshots are taken before any wall interaction.

Read the result The peak translates with constant width. A numerical overlay should track both its position and the pressure-to-velocity ratio.

OneSimulation connection LinearAcoustics and its propagation provider exercise traveling waves, periodic integrals and reflective or imposed-flux boundaries.

API IN PRACTICE

Set the pressure pulse and its matched right-going velocity on a Line named duct. pulse(x) is the initial Gaussian in Pa.

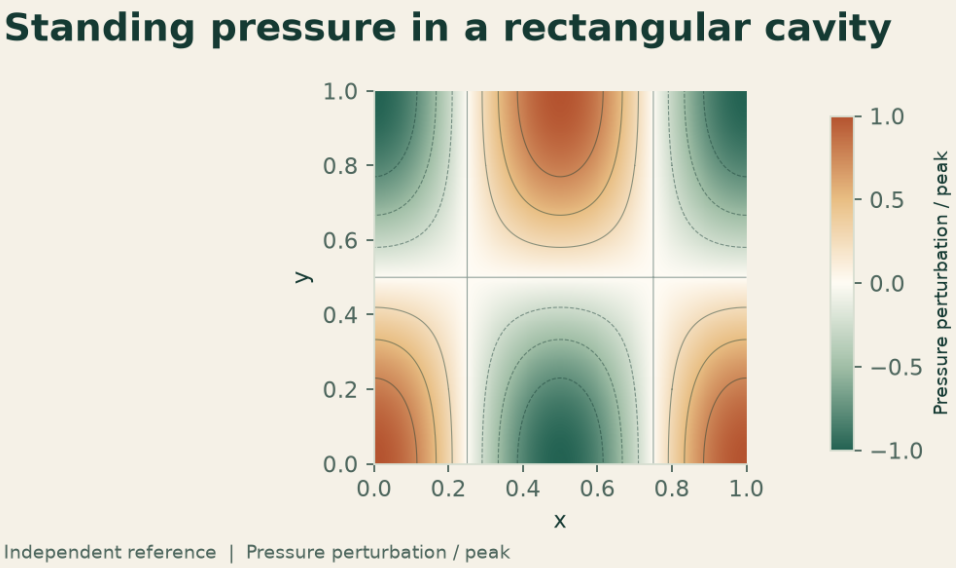
```
from onesim.physics.acoustics import LinearAcoustics
from onesim.simulation.initial_conditions import InitialValue

air = LinearAcoustics(domain=duct.line, density=1.2, sound_speed=343)
initial = (InitialValue(air.pressure, pulse),
           InitialValue(air.velocity, lambda x: pulse(x) / (1.2 * 343)))
```

Selected API names PressureDrivenMode, RadiatingMode, UltrasonicAcoustics, LinearAcoustics, AcousticsSpatialOperator, AcousticModelKind, AcousticNodeKind, AcousticSystemService. The full export list appears in the appendix.

Standing pressure in a rectangular cavity

ACOUSTICS / 2D REFERENCE



Independent reference. Pressure perturbation / peak.

$$p(x, y, t) = \cos(2\pi x)\cos(\pi y)\cos(\omega t), \quad \omega = c\pi\sqrt{5}$$

Setup Use a unit rectangle with rigid walls and excite the (2, 1) Neumann mode. Plot the spatial mode at maximum pressure.

Read the result The normal pressure gradient vanishes at the walls. Sign changes mark nodal lines and separate regions in opposite phase.

OneSimulation connection A modal reference for pressure and boundary checks. The 1D spatial-method tests do not establish this rectangular cavity configuration.

API IN PRACTICE

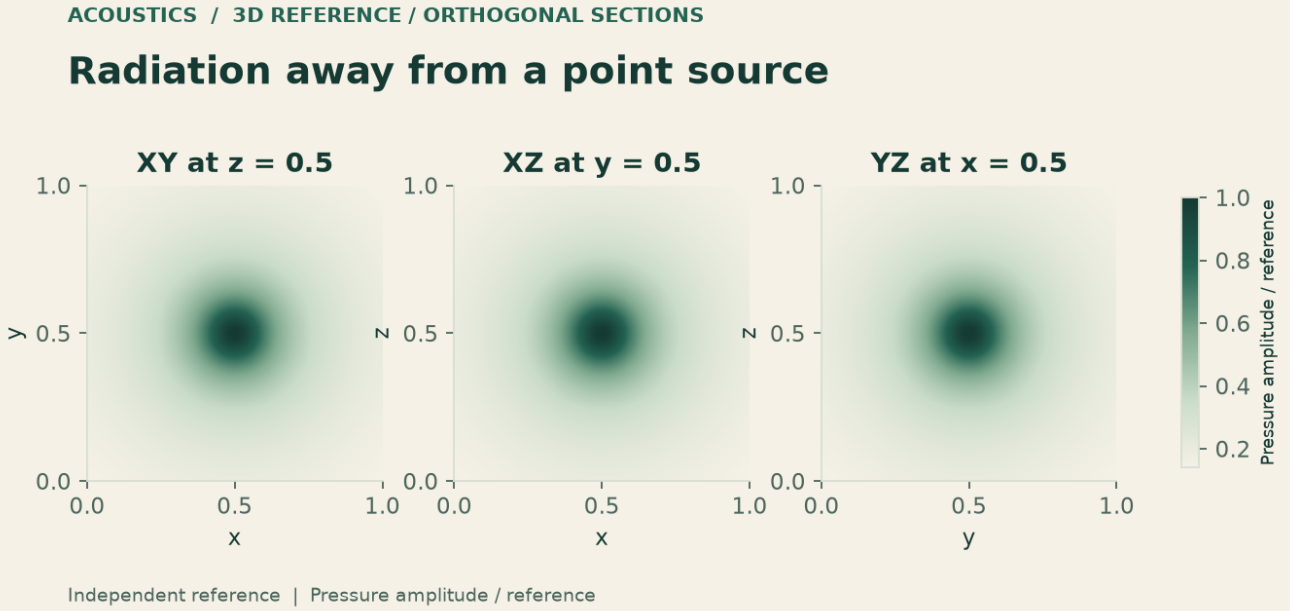
Declare the rigid-wall condition on an acoustic duct. This is the boundary idiom for the cavity analogy. The 2D mode is analytical.

```
from onesim.physics.acoustics import LinearAcoustics
from onesim.simulation.problem_definition.boundaries import ReflectingWall

air = LinearAcoustics(domain=duct.line, density=1.2, sound_speed=343)
rigid_end = ReflectingWall(boundary=duct.boundaries.end, physics=air)
```

Repository cross-reference test_acoustic_method_flexibility, test_acoustic_mode_input_flexibility.

Radiation away from a point source



Independent reference. Pressure amplitude / reference.

$$|p(r)| \propto (r^2 + a^2)^{-1/2}$$

Setup Evaluate a regularized free-field pressure envelope around a compact source, with normalized core radius 0.10. Three planes through the center expose its radial structure.

Read the result Equal-amplitude contours are circular in each central slice. Far from the core, geometric spreading approaches inverse distance.

OneSimulation connection RadiatingMode exposes a reduced radiation model. The plotted envelope is a free-field reference and has no room boundaries or full-wave solver behind it.

API IN PRACTICE

Use a LumpedDomain named body for the reduced radiation source. The free-field envelope is a separate calculation.

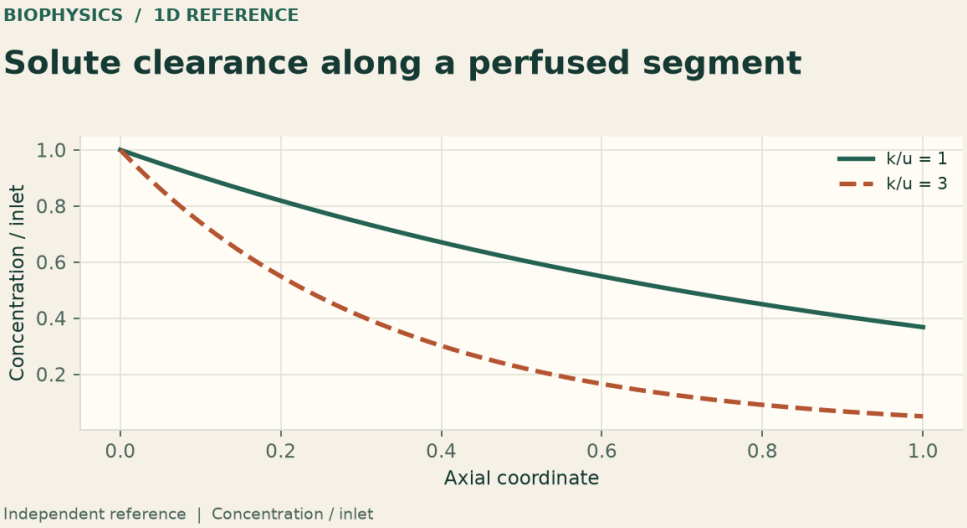
```
from onesim.physics.acoustics import RadiatingMode

mode = RadiatingMode(domain=body.system, mass=0.1, stiffness=200,
                     area=0.005, radiation_resistance=0.02)
power_to_record = mode.radiated_power
```

Repository cross-reference test_acoustic_method_flexibility, test_acoustic_mode_input_flexibility.

Solute clearance along a perfused segment

The package represents biological exchange through components such as passive membranes, tissue elements, compliant vessels and organ clearance. Reduced states and input ports are central to these models.



Independent reference. Concentration / inlet.

$$u\,C_x = -kC, \qquad C/C_0 = \exp(-kx/u)$$

Setup Prescribe a constant mean speed and a first-order uptake rate along a unit vessel. Compare normalized uptake lengths corresponding to k/u of 1 and 3.

Read the result The steeper curve represents stronger clearance or slower transit. The axial balance is a useful check before connecting multiple tissue and vessel compartments.

OneSimulation connection AxialTissue, CompliantVessels and OrganClearance are reduced models. This advection-clearance reference is spatial, unlike a well-mixed organ state.

API IN PRACTICE

Set volume and clearance on a well-mixed compartment. Its concentration is a state readout for a reduced uptake model.

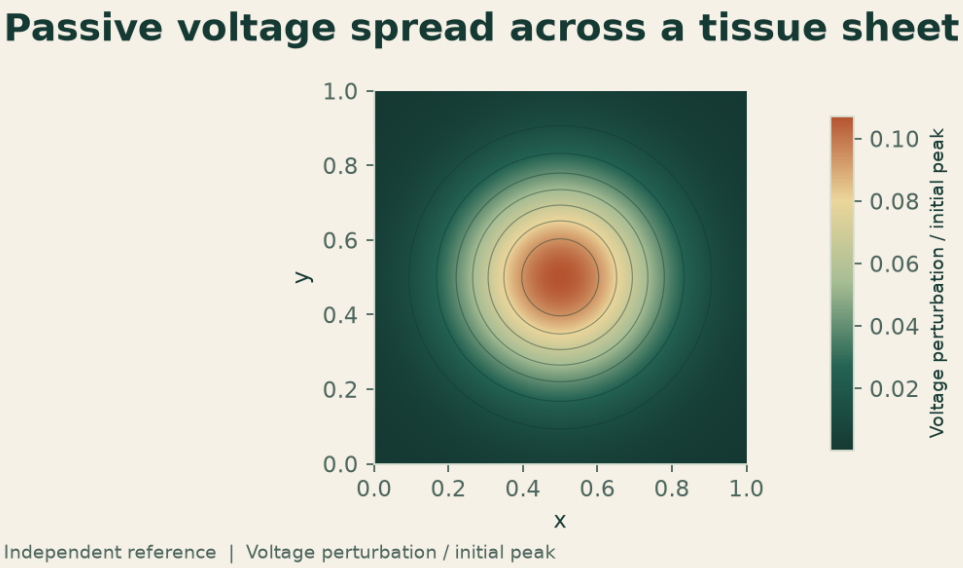
```
from onesim.cad.geometry import LumpedDomain
from onesim.physics.biophysics import OrganClearance

organ = LumpedDomain()
uptake = OrganClearance(domain=organ.system, volume=1e-4, clearance=2e-6)
concentration_to_record = uptake.concentration
```

Selected API names OrganClearance, AxialTissue, TissueElement, PassiveMembrane, CompliantVessels, VesselSegment. The full export list appears in the appendix.

Passive voltage spread across a tissue sheet

BIOPHYSICS / 2D REFERENCE



Independent reference. Voltage perturbation / initial peak.

$$V - V_{\text{rest}} \propto s^{-2}\exp[-r^2/(2s^2) - t/\tau]$$

Setup Place a localized depolarization on a homogeneous sheet and evaluate a spreading, leaking Gaussian. Width and decay represent the diffusion and membrane time scales.

Read the result The perturbation spreads laterally as its peak falls. A passive response has no regenerative action potential.

OneSimulation connection PassiveMembrane provides a reduced bioelectrical component. A spatial monodomain solver and electrophysiology calibration are outside this example.

API IN PRACTICE

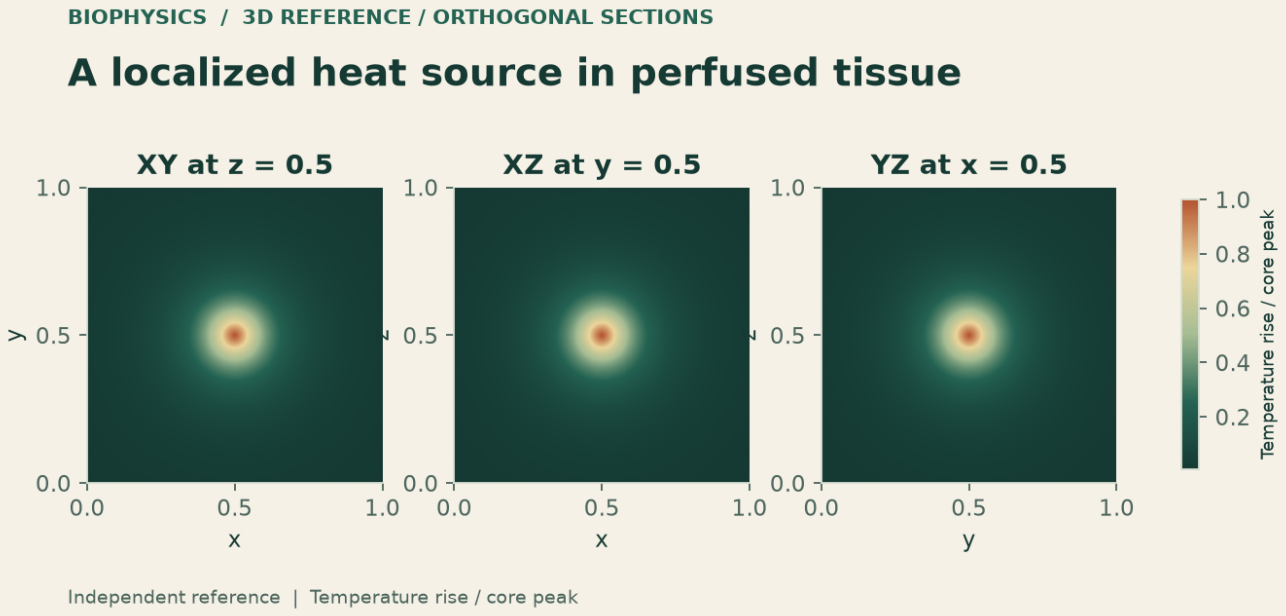
Set a membrane's capacitance, leakage and resting potential on a LumpedDomain named tissue. The sheet adds spatial transport beyond this component.

```
from onesim.physics.biophysics import PassiveMembrane
from onesim.simulation.initial_conditions import InitialValue

membrane = PassiveMembrane(domain=tissue.system, capacitance=0.01,
                           conductance=0.1, reversal_potential=-0.07)
initial_voltage = InitialValue(membrane.voltage, -0.06)
```

Repository cross-reference test_biophysics_input_flexibility.

A localized heat source in perfused tissue



Independent reference. Temperature rise / core peak.

$$\Delta T(r) \propto \frac{\exp(-r/L)}{\sqrt{r^2 + a^2}}$$

Setup Take a steady, homogeneous screening model with perfusion length L = 0.23 and core regularization a = 0.09. Plot three slices through the idealized source.

Read the result Perfusion localizes the warm region. The decay length controls penetration, while the core regularization prevents a singular plot.

OneSimulation connection The BioheatProvider is a separate execution surface. This screened field illustrates the continuum limit and carries no patient-specific interpretation.

API IN PRACTICE

Choose perfusion and an external heat input on the exported bioheat provider. This is its reduced cell path, separate from the 3D reference.

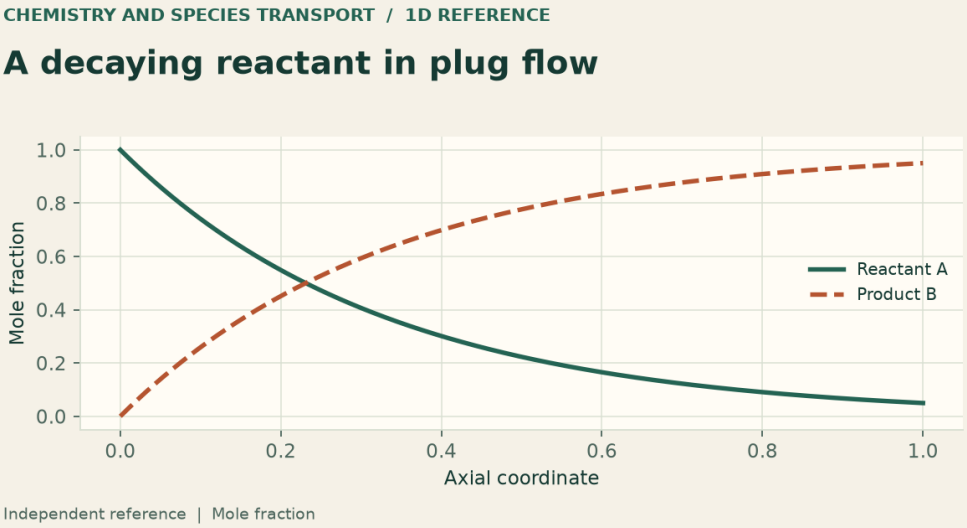
```
from onesim.physics.biophysics import BioheatProvider

bioheat = BioheatProvider(cell_count=64, conductivity=0.5,
                          perfusion_capacity_rate=2500.0,
                          blood_temperature=310.15, external_heat=1000.0)
```

Repository cross-reference test_biophysics_input_flexibility.

A decaying reactant in plug flow

Species inventories, reaction rates and Jacobians can feed a coupled problem. The package exposes a reaction-network IR, species transport, and reduced electrochemical and battery components.



Independent reference. Mole fraction.

$$Y_A = \exp(-kx/u), \qquad Y_B = 1 - Y_A$$

Setup Feed pure A at the inlet of a constant-speed reactor. Convert A to B in a first-order reaction with normalized rate k/u = 3.

Read the result Both fractions stay nonnegative, their sum remains one, and reaction progress grows with distance.

OneSimulation connection SpeciesTransport and the kinetics interfaces support source terms and coupled tangents. The example reduces the species balance to an exact steady plug-flow check.

API IN PRACTICE

Declare an advected and decaying reactant on the reactor Line. Add the product through a reaction or source port to model the A-to-B balance.

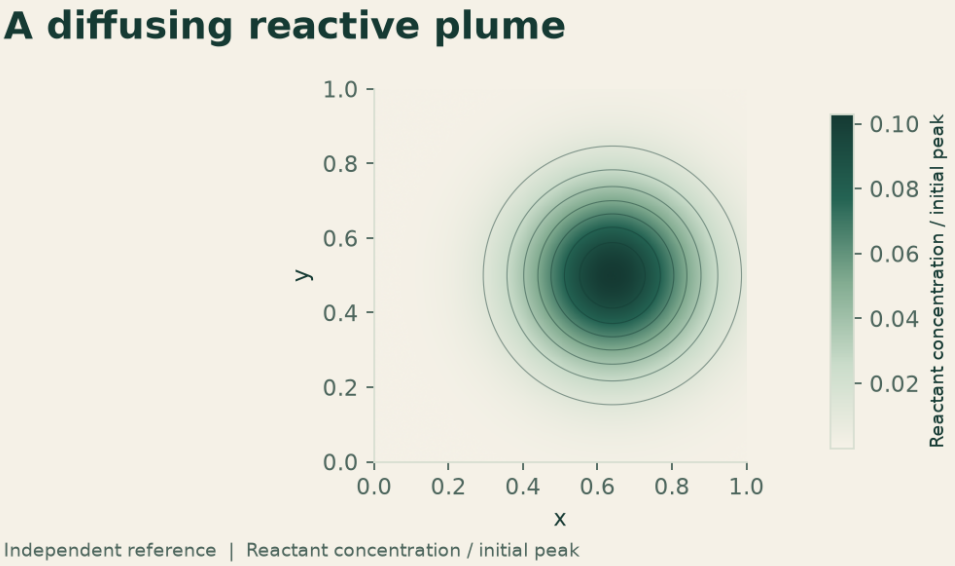
```
from onesim.physics.chemistry import SpeciesTransport, TransportSpecies

reactant = TransportSpecies("A", diffusivity=0.0, decay_rate=3.0)
transport = SpeciesTransport(domain=reactor.line, species=(reactant,),
                             velocity=1.0)
concentration_to_record = transport.field("A")
```

Selected API names ElectrochemicalReaction, BatteryChemistry, LumpedElectrode, LumpedBattery, ReactionKinetics, Chemistry, ChemistryKinetics, ChemistryEvaluation, ReactionNetworkIR. The full export list appears in the appendix.

A diffusing reactive plume

CHEMISTRY AND SPECIES TRANSPORT / 2D REFERENCE



Independent reference. Reactant concentration / initial peak.

$$C_A(r,t) = C_0(s_0/s)^2 \exp[-r^2/(2s^2) - kt]$$

Setup Release a localized reactant in a uniform sheet. Evaluate the two-dimensional diffusion-reaction Green function for a Gaussian initial condition, centered downstream of its origin.

Read the result The footprint broadens while the integral decays exponentially. A pure-diffusion check would keep the integral constant.

OneSimulation connection An analytical target for a spatial species balance. The repository tests include nonlinear boundary fluxes and full species/source tangents.

API IN PRACTICE

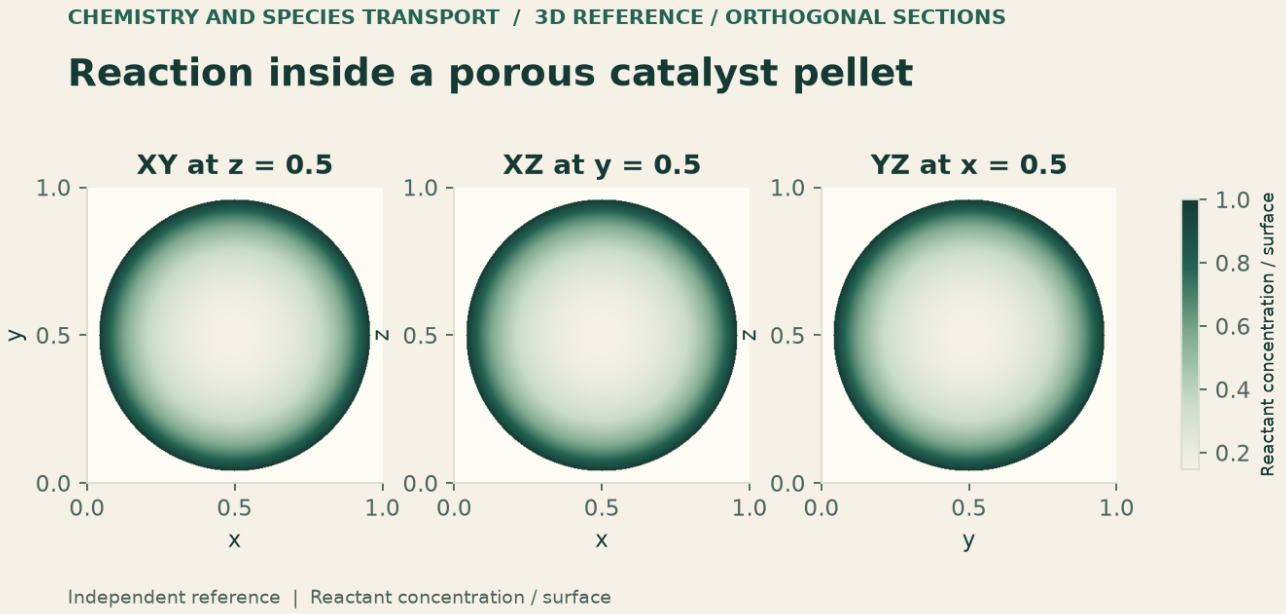
Attach diffusivity and decay to the transported field. This excerpt selects the line version of the balance used in the plume reference.

```
from onesim.physics.chemistry import SpeciesTransport, TransportSpecies
from onesim.simulation.initial_conditions import InitialValue

tracer = TransportSpecies("A", diffusivity=0.05, decay_rate=0.35)
model = SpeciesTransport(domain=channel.line, species=(tracer,), velocity=0.4)
initial_plume = InitialValue(model.field("A"), initial_concentration)
```

Repository cross-reference test_kinetics_input_flexibility, test_species_transport_flexibility.

Reaction inside a porous catalyst pellet



Independent reference. Reactant concentration / surface.

$$C(r)/C_s = \frac{\sinh(\phi r/R)}{(r/R)\sinh\phi}$$

Setup Set concentration at the surface of a spherical pellet and use a first-order consumption law. Set the Thiele modulus to 4 and use the regular center limit.

Read the result Reactant is depleted at the center. The three sections show the active outer shell and the spherical symmetry of this benchmark.

OneSimulation connection This continuum diffusion-reaction reference motivates combining transport and reaction laws. It is distinct from LumpedBattery and does not qualify a specific 3D battery model.

API IN PRACTICE

Specify the pellet's diffusion and consumption coefficients on the species declaration. A volume discretization and surface trace would be separate selections.

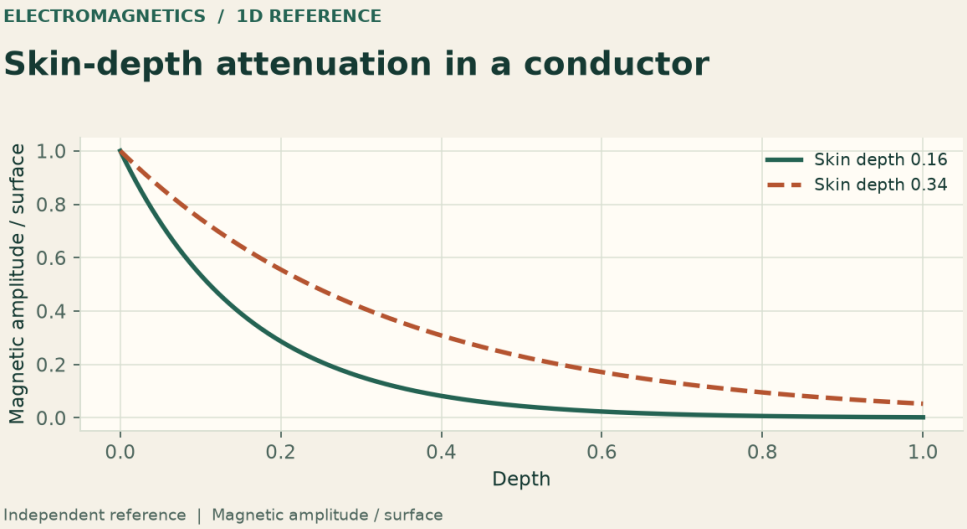
```
from onesim.physics.chemistry import TransportSpecies

radius, diffusivity = 0.02, 1e-6
rate = 4.0**2 * diffusivity / radius**2
reactant = TransportSpecies("A", diffusivity=diffusivity, decay_rate=rate)
```

Repository cross-reference test_kinetics_input_flexibility, test_species_transport_flexibility.

Skin-depth attenuation in a conductor

The package spans reduced circuits and field models. FrequencyDomainMaxwell, waveguide ports, magnetic diffusion and finite-integration paths serve different formulations.



Independent reference. Magnetic amplitude / surface.

$$|H(x)|/H_s = \exp(-x/\delta), \qquad \delta = \sqrt{2/(\omega\mu\sigma)}$$

Setup Drive a conducting half-space harmonically. Compare two normalized skin depths, 0.16 and 0.34, on the same depth axis.

Read the result The smaller skin depth concentrates current near the surface. The exponential is the amplitude envelope of a complex field that also rotates in phase with depth.

OneSimulation connection MagneticDiffusion provides a field-model entry point. The half-space relation is a benchmark for boundary and frequency conventions.

API IN PRACTICE

Set permeability and conductivity on a conductor Line and select the induction field. The harmonic envelope is the reference for the driven case.

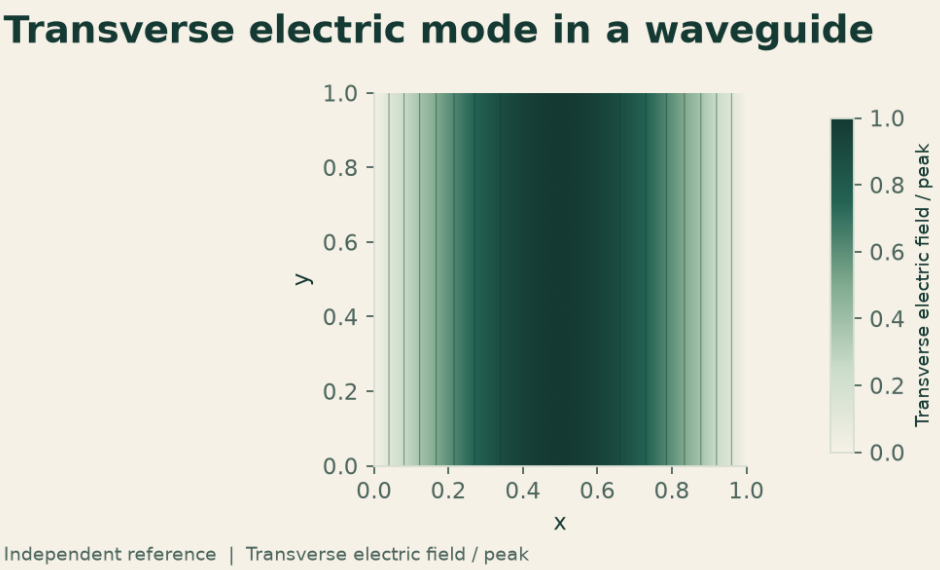
```
from onesim.physics.electromagnetics import MagneticDiffusion

metal = MagneticDiffusion(domain=conductor.line, permeability=1.2566e-6,
                          conductivity=5.8e7)
induction_to_record = metal.magnetic_induction
```

Selected API names FrequencyDomainMaxwell, WaveguidePort, CircularWaveguideMode, PerfectElectricConductor, ImpedanceBoundary, WaveguideModeBasis, SeriesRLC, AveragedBuckConverter, DCMotor. The full export list appears in the appendix.

Transverse electric mode in a waveguide

ELECTROMAGNETICS / 2D REFERENCE



Independent reference. Transverse electric field / peak.

$$E_y(x, y)/E_0 = \sin(\pi x/a)$$

Setup Sample the TE10 cross-section of an ideal rectangular perfectly conducting guide. The plotted field is the transverse electric component for a single mode.

Read the result The electric amplitude vanishes at the two sidewalls where that component is tangential. The broad central band is characteristic of the lowest rectangular TE mode.

OneSimulation connection WaveguidePort and FrequencyDomainMaxwell implement modal driving. The existing public waveguide checks include circular TE11, with a different modal basis.

API IN PRACTICE

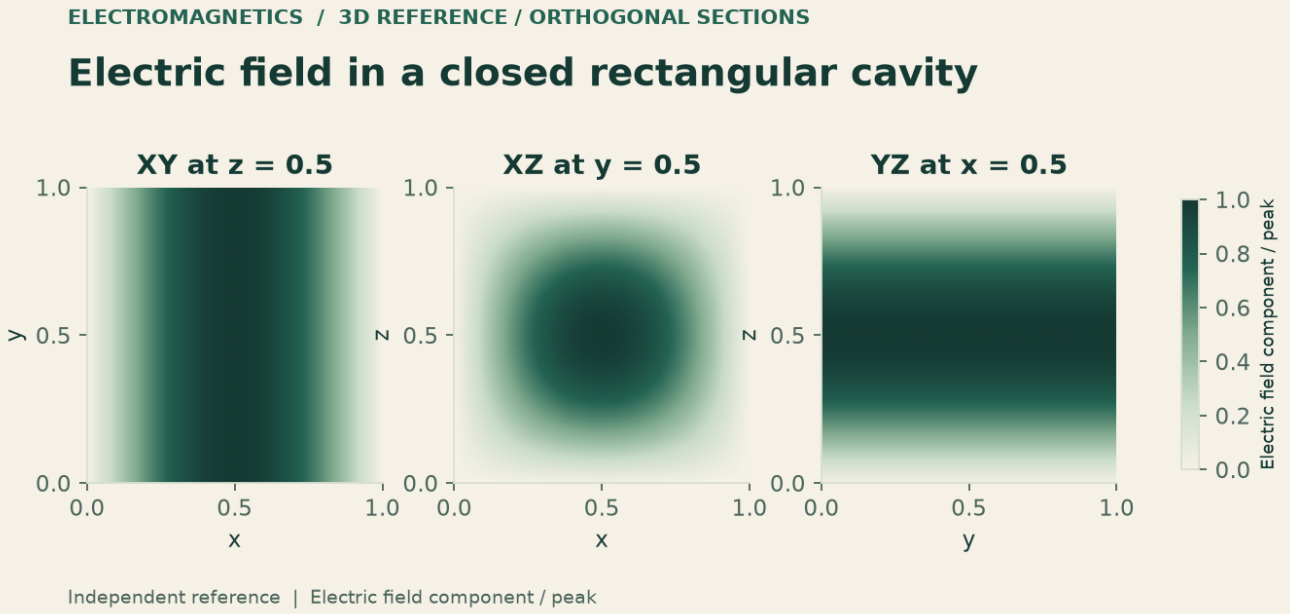
Attach a modal drive to the bottom face of a Cylinder named guide. This is the public circular TE11 port, corresponding to the modal drive idea in the TE10 reference.

```
from onesim.physics.electromagnetics import CircularWaveguideMode, WaveguidePort

mode = CircularWaveguideMode("TE11")
drive = WaveguidePort(boundary=guide.boundaries.bottom, physics=maxwell,
                      mode=mode, incident_power=2.0)
```

Repository cross-reference test_public_maxwell, test_finite_integration_electromagnetics, test_magnetic_diffusion_flexibility.

Electric field in a closed rectangular cavity



Independent reference. Electric field component / peak.

$$E_y/E_0 = \sin(\pi x/a)\sin(\pi z/d)$$

Setup Use a separable TE101-type standing mode with y-directed electric field and perfect electric conductor walls.

Read the result The slices expose nodes on the x and z boundaries. The y-normal end faces permit a nonzero normal component.

OneSimulation connection A closed-cavity field reference for a curl-conforming formulation. Actual Maxwell tests check port power, phasor conventions, heating and field reconstruction.

API IN PRACTICE

Assign a conducting wall and a curl-conforming field space to a Box model. A driven Maxwell study also needs a source and frequency.

```
from onesim.physics.electromagnetics import PerfectElectricConductor
from onesim.numerics.fem import FiniteElement, Nedelec

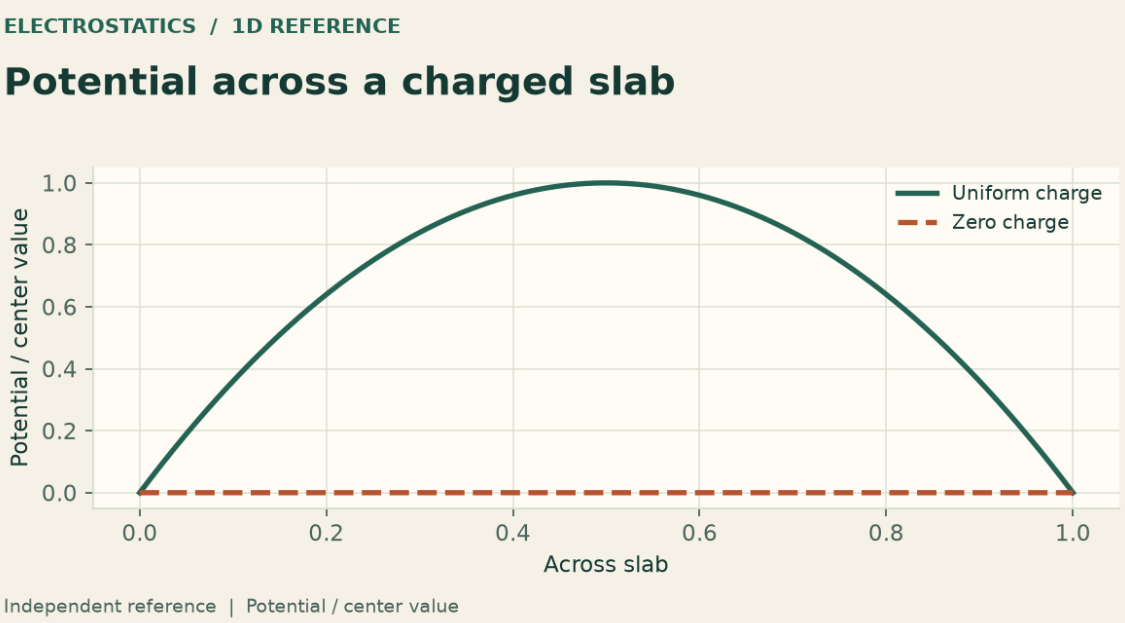
wall = PerfectElectricConductor(boundary=cavity.boundaries.all, physics=maxwell)
space = FiniteElement(element=Nedelec(order=1))
electric_field = maxwell.electric_field
```

Repository cross-reference test_public_maxwell, test_finite_integration_electromagnetics, test_magnetic_diffusion_flexibility.

Potential across a charged slab

PoissonEquation combines potential, source and boundary conditions. Scalar potential is a convenient way to compare discretizations because the field follows from its gradient.

ELECTROSTATICS / 1D REFERENCE



Independent reference. Potential / center value.

$$-\phi'' = 8, \quad \phi(0) = \phi(1) = 0, \quad \phi = 4x(1-x)$$

Setup Use uniform normalized space charge and zero voltage on both parallel faces. Plot the exact potential alongside the zero-charge potential.

Read the result Curvature follows the charge sign. The maximum lies at the center and the electric field reverses there.

OneSimulation connection PoissonEquation and FixedPotential supply the scalar model and Dirichlet values. The numerical methods chapter also solves a manufactured Poisson case.

API IN PRACTICE

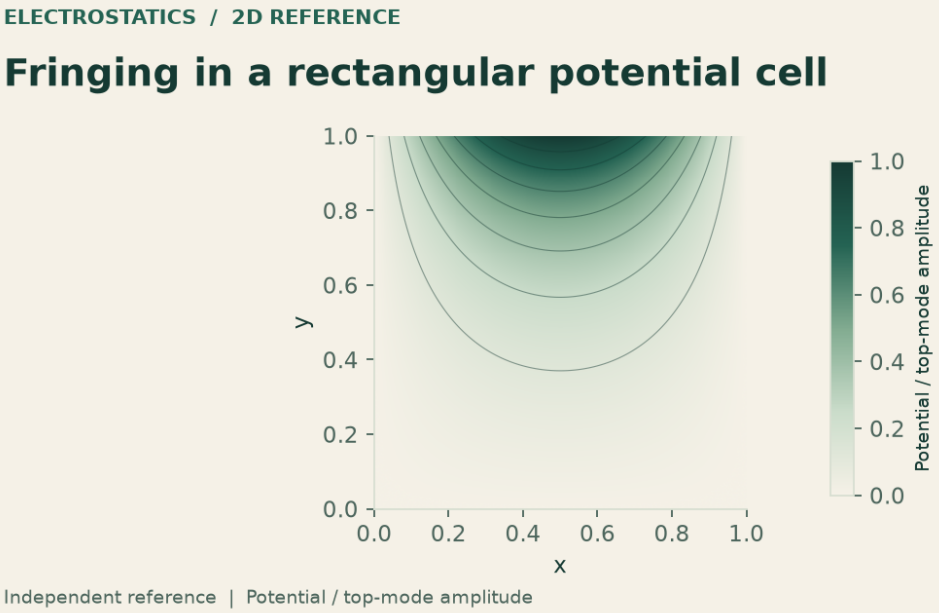
On the slab Line, prescribe the charge and ground both ends. epsilon is the permittivity property evaluated by the model.

```
from onesim.physics.electrostatics import PoissonEquation, FixedPotential

poisson = PoissonEquation(domain=slab.line, permittivity=epsilon, charge_density=8)
ground = FixedPotential(boundary=slab.boundaries.all, physics=poisson, value=0)
potential_to_record = poisson.potential
```

Selected API names NormalElectricDisplacement, PoissonEquation, FixedPotential, ZeroNormalElectricDisplacement. The full export list appears in the appendix.

Fringing in a rectangular potential cell



Independent reference. Potential / top-mode amplitude.

$$\phi(x, y) = \sin(\pi x)\sinh(\pi y)/\sinh(\pi)$$

Setup Ground three sides of a unit rectangle. Prescribe the sinusoidal voltage on its top edge and set interior charge to zero.

Read the result The harmonic boundary pattern decays into the interior. The side edges remain at the prescribed ground potential.

OneSimulation connection A Laplace benchmark for FixedPotential. Compare potential and normal electric displacement separately when changing methods.

API IN PRACTICE

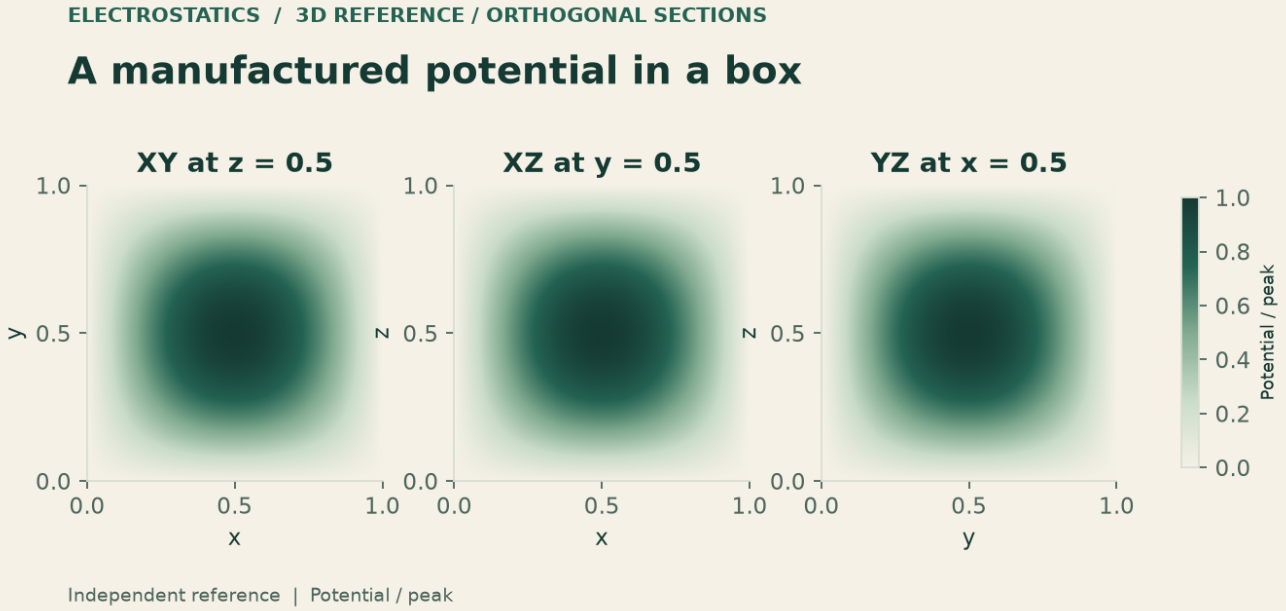
Attach the sinusoidal voltage function to a selected top edge. Supply a compatible planar potential model and the remaining ground conditions separately.

```
import numpy as np
from onesim.physics.electrostatics import FixedPotential

top_trace = FixedPotential(boundary=top_edge, physics=poisson,
                           value=lambda x, y, z: np.sin(np.pi * x))
```

Repository cross-reference test_scalar_method_flexibility, test_cell_projection_poisson.

A manufactured potential in a box



Independent reference. Potential / peak.

$$\phi = \sin(\pi x)\sin(\pi y)\sin(\pi z), \quad -\nabla^2\phi = 3\pi^2\phi$$

Setup Apply the manufactured volume charge on a unit box with zero potential on all six faces.

Read the result The central maximum and zero boundary values are visible on all three slices. This source is defined from the solution so any spatial error can be measured.

OneSimulation connection Suitable for a 3D scalar Poisson verification study. Mesh, boundary treatment and solver determine the convergence actually observed.

API IN PRACTICE

Use a unit-permittivity property epsilon and the sine product phi_exact(x, y, z) to supply the manufactured charge on the Box.

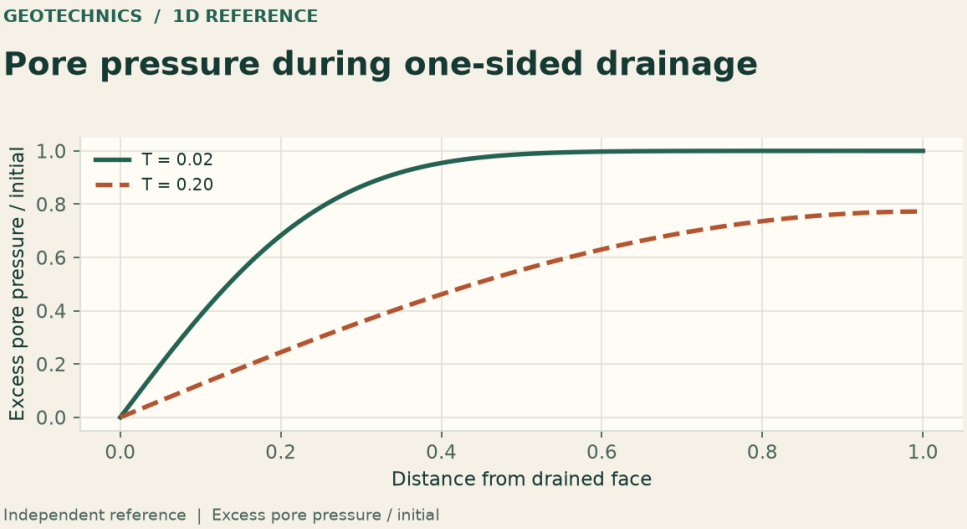
```
from onesim.physics.electrostatics import PoissonEquation, FixedPotential
from math import pi

poisson = PoissonEquation(domain=box.volume, permittivity=epsilon,
    charge_density_source=lambda x, y, z: 3 * pi**2 * phi_exact(x, y, z))
walls = FixedPotential(boundary=box.boundaries.all, physics=poisson, value=0)
```

Repository cross-reference test_scalar_method_flexibility, test_cell_projection_poisson.

Pore pressure during one-sided drainage

GroundwaterFlow, TerzaghiConsolidation, BiotConsolidation and BuckleyLeverettFlow separate hydraulic diffusion, deformation, storage and saturation transport.



Independent reference. Excess pore pressure / initial.

$$p/p_0 = \sum_{n \geq 0} \frac{2}{\lambda_n} \sin(\lambda_n x) e^{-\lambda_n^2 T}, \quad \lambda_n = (n + 1/2)\pi$$

Setup Drain the face at x = 0 and seal x = 1. Compare dimensionless times 0.02 and 0.20 for a layer that starts with uniform excess pressure.

Read the result Pressure falls first near the drain. The far face has zero slope, consistent with zero normal flux.

OneSimulation connection TerzaghiConsolidation and drained/impermeable boundaries are the natural model vocabulary for this reference.

API IN PRACTICE

Apply separate drain and sealed-end conditions to a Terzaghi model on the layer Line. Add the initial excess pore pressure to the problem.

```
from onesim.physics.geotechnics import DrainedBoundary, ImpermeableBoundary
from onesim.simulation.initial_conditions import InitialValue

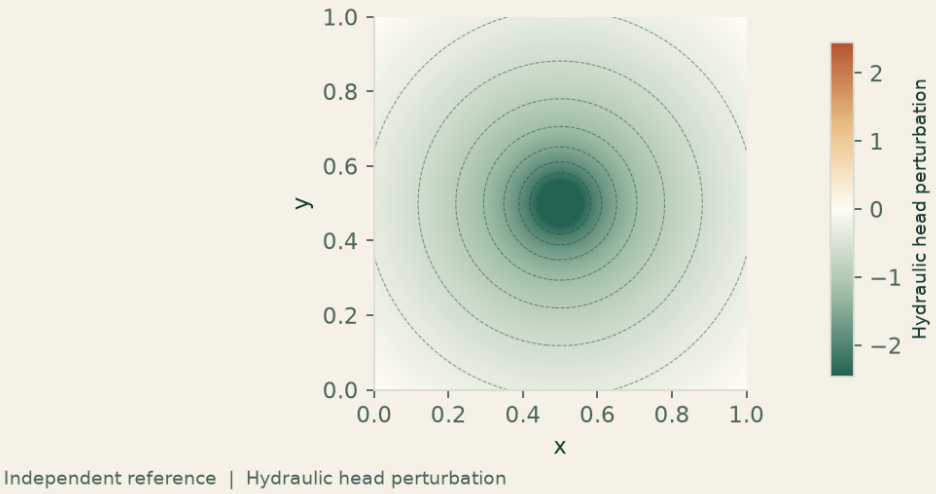
drain = DrainedBoundary(boundary=layer.boundaries.start, physics=terzaghi)
seal = ImpermeableBoundary(boundary=layer.boundaries.end, physics=terzaghi)
p0 = InitialValue(terzaghi.pore_pressure, 10_000)
```

Selected API names GroundwaterFlow, BiotConsolidation, TerzaghiConsolidation, PrescribedPorePressure, PoreFluidFlux, DrainedBoundary, ImpermeableBoundary, BuckleyLeverettFlow, QuadraticCoreyFractionalFlow. The full export list appears in the appendix.

Hydraulic head near an extraction well

GEOTECHNICS / 2D REFERENCE

Hydraulic head near an extraction well



Independent reference. Hydraulic head perturbation.

$$h(r) - h(R) = \frac{Q}{2\pi T}\log(r/R)$$

Setup Evaluate a steady confined-aquifer drawdown with circular outer reference head and a finite well radius. Normalize Q/(2 pi T) to 1.

Read the result Head is lowest close to the well. Logarithmic contours are widely spaced at large radius and crowd near the extraction point.

OneSimulation connection GroundwaterFlow tests use actual mesh coordinates and independently selected gradient and diffusion operations. The radial plot is a confined-flow analytical limit.

API IN PRACTICE

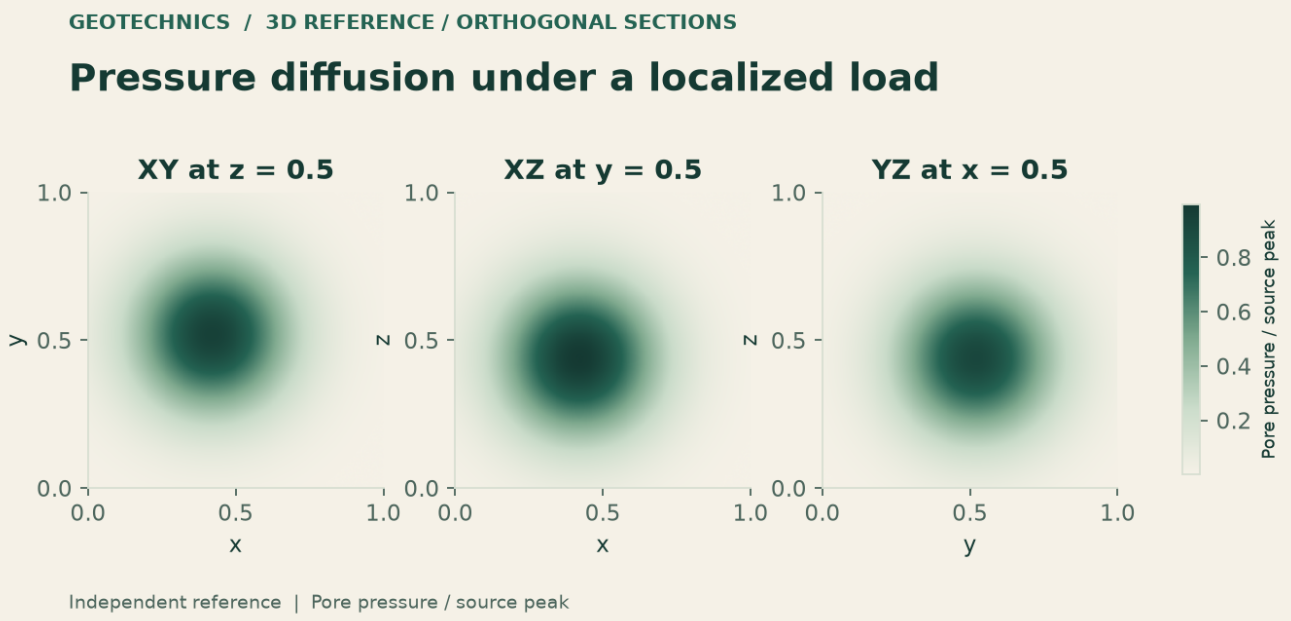
Declare the hydraulic storage and conductivity on a reduced aquifer Line. Use the head and Darcy flux as separate checks against an analytical limit.

```
from onesim.physics.geotechnics import GroundwaterFlow

aquifer = GroundwaterFlow(domain=transect.line, conductivity=0.02,
                          specific_storage=0.2)
fields = (aquifer.head, aquifer.darcy_flux)
```

Repository cross-reference test_public_groundwater, test_public_biot_consolidation, test_public_consolidation_waterflood.

Pressure diffusion under a localized load



Independent reference. Pore pressure / source peak.

$$p(\mathbf{x}, t) \propto s^{-3}\exp(-|\mathbf{x}-\mathbf{x}_0|^2/(2s^2))$$

Setup Use a localized excess-pressure disturbance in a homogeneous three-dimensional diffusion limit. Plot orthogonal sections of the expanding plume.

Read the result The plume spreads from its off-center origin. Under the infinite-domain approximation the pressure integral is constant, even as peak pressure falls.

OneSimulation connection BiotConsolidation resolves coupled displacement and pressure, and repository volume tests check storage and Darcy terms. This plot isolates the pressure diffusion limit.

API IN PRACTICE

Declare the coupled 3D solid and pore fluid. The diffusion plot isolates one limit of this model. A full study also needs its mechanical and hydraulic boundaries.

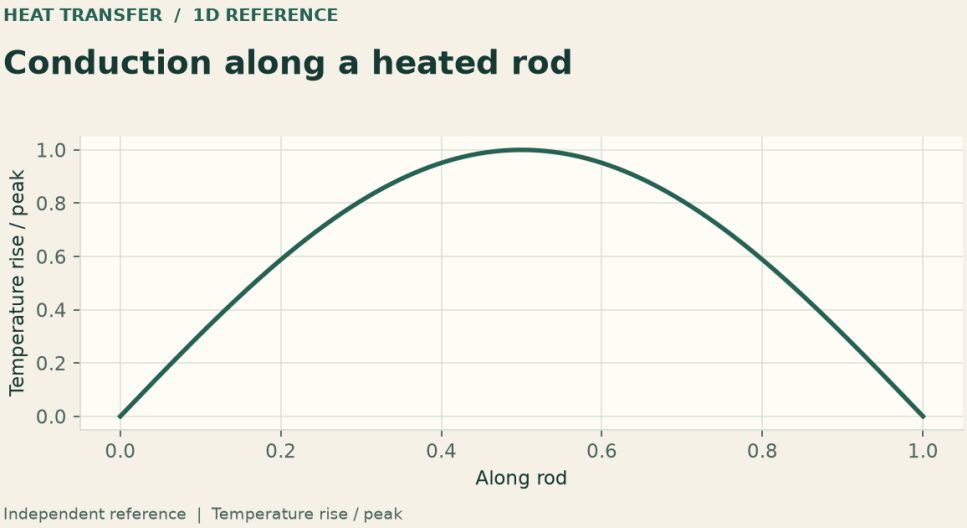
```
from onesim.physics.geotechnics import BiotConsolidation

soil = BiotConsolidation(domain=box.volume, youngs_modulus=1e7,
    poisson_ratio=0.3, biot_coefficient=0.9, specific_storage=1e-8,
    permeability=1e-13, fluid_viscosity=1e-3)
fields = (soil.displacement, soil.pore_pressure)
```

Repository cross-reference test_public_groundwater, test_public_biot_consolidation, test_public_consolidation_waterflood.

Conduction along a heated rod

Conduction, convection boundary conditions, phase change, transport and concentrated radiation cooling share a temperature-centered workflow, with different choices of state and numerical balance.



Independent reference. Temperature rise / peak.

$$-T'' = \pi^2 \sin(\pi x), \qquad T = \sin(\pi x)$$

Setup Ground both end temperatures relative to a reference and prescribe a sinusoidal volumetric heating profile. Use nondimensional conductivity 1.

Read the result The source and curvature have consistent signs. The maximum occurs midway between the fixed-temperature ends.

OneSimulation connection HeatConduction and FixedTemperature appear in the public stationary tests. The same elliptic structure underlies the FEM, FVM and FDM comparison.

API IN PRACTICE

Give a conduction model the manufactured heat source. The excerpt uses a slender 3D Box named rod with a material supplying conductivity.

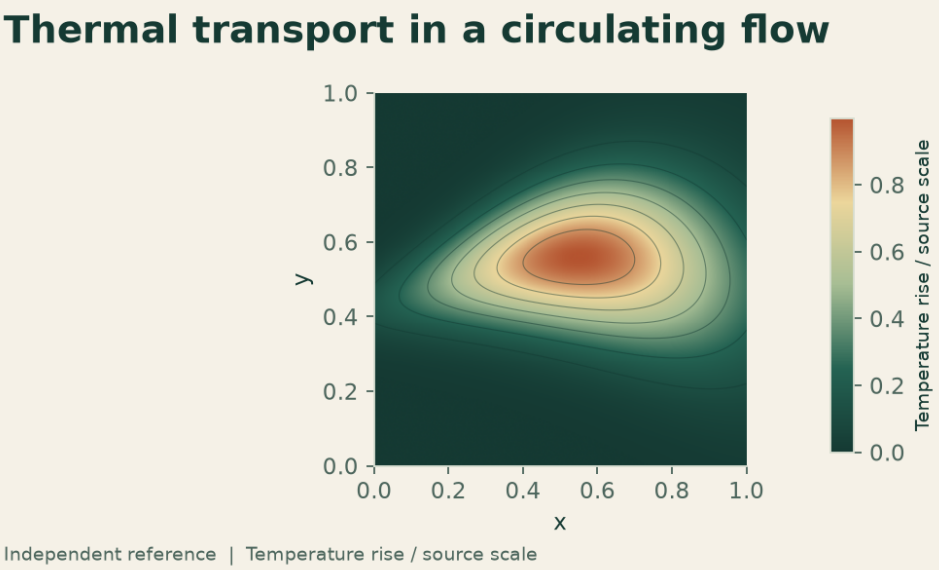
```
import numpy as np
from onesim.physics.heat_transfer import HeatConduction

rod_heat = HeatConduction(domain=rod.volume, material=solid,
    heating_source=lambda x, y, z: np.pi**2 * np.sin(np.pi * x))
temperature_to_record = rod_heat.temperature
```

Selected API names HeatFlux, HeatConduction, FixedTemperature, ConvectiveHeatFlux, RadiationCooling, ThermalGreyDiffuseRadiationBoundary, PhaseChangeHeating, ThermalPhaseChangeLawSet, HeatAdvectionDiffusion. The full export list appears in the appendix.

Thermal transport in a circulating flow

HEAT TRANSFER / 2D REFERENCE



Independent reference. Temperature rise / source scale.

$$\rho c_p(\partial_t T + \mathbf{u}\cdot\nabla T) = k\nabla^2 T + q$$

Setup Use a stylized unit plate with a prescribed heated streak and growing transverse width. The recorded circulating heat calculation appears on a separate runtime plate.

Read the result The plume is elongated downstream. Changing the Peclet number changes the relative importance of drift and diffusion.

OneSimulation connection HeatAdvectionDiffusion accepts a flow field and thermal data. The repository includes a separate circulation notebook and tests for volumetric convection.

API IN PRACTICE

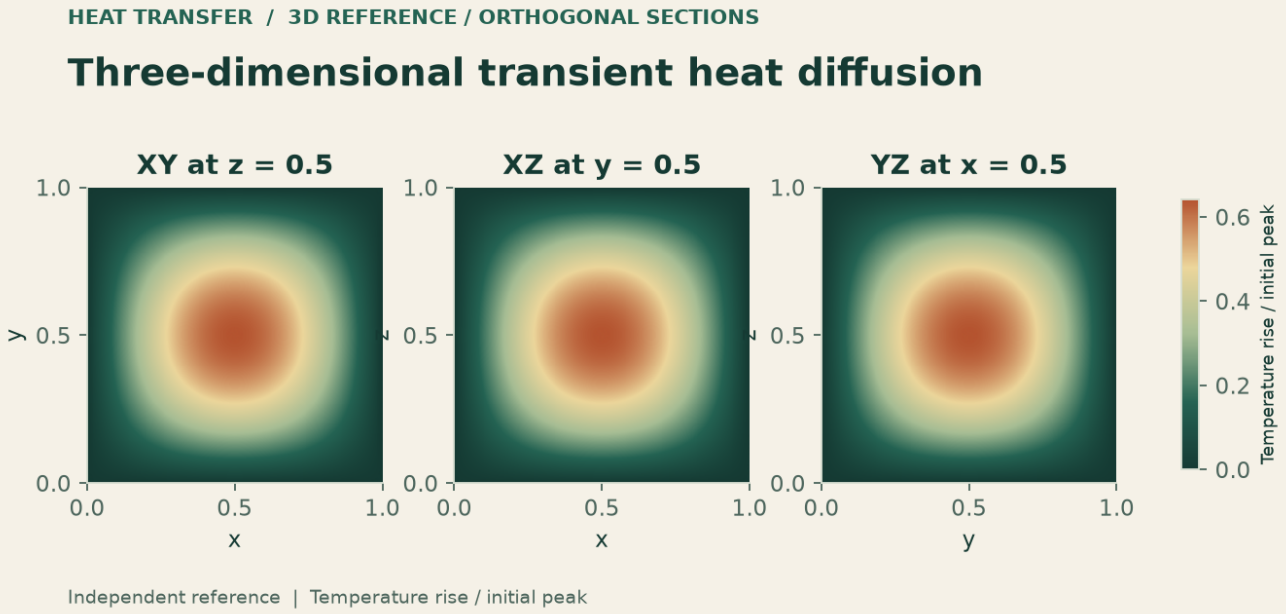
Attach a prescribed velocity and heating function to a thin Box. This is the declaration used by the separate circulating-heat run.

```
from onesim.physics.heat_transfer import ConvectiveHeatTransfer
from onesim.meshing import StructuredHexahedralMesher

heat = ConvectiveHeatTransfer(domain=plate.volume, material=solid,
                              velocity=velocity, heating=heating)
mesh = StructuredHexahedralMesher(cells=(48, 32, 1)).generate(plate)
```

Repository cross-reference test_public_stationary, test_public_lumped_heat_solids, test_convective_heat_volume.

Three-dimensional transient heat diffusion



Independent reference. Temperature rise / initial peak.

$$T/T_0 = e^{-3\pi^2 \alpha t} \sin(\pi x) \sin(\pi y) \sin(\pi z)$$

Setup Initialize the fundamental mode on a unit cube, with zero temperature rise on its boundary. Evaluate at dimensionless time alpha t = 0.015.

Read the result The mode preserves its shape while its amplitude decays. Orthogonal slices expose the symmetry and fixed boundaries.

OneSimulation connection This exact mode can check 3D transient HeatConduction. Runtime qualification depends on the selected mesh, boundary treatment and time integrator.

API IN PRACTICE

Initialize the mode as a rise above 300 K. mode(x, y, z) is the sine product. Fix the boundary at the reference temperature.

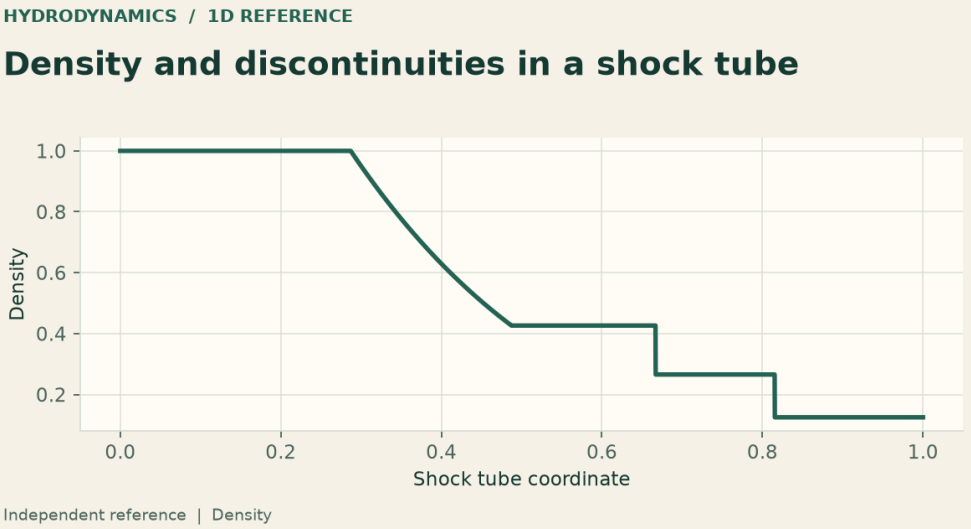
```
from onesim.physics.heat_transfer import HeatConduction, FixedTemperature
from onesim.simulation.initial_conditions import InitialValue

heat = HeatConduction(domain=box.volume, material=solid)
initial = InitialValue(heat.temperature, lambda x, y, z: 300 + mode(x, y, z))
wall = FixedTemperature(boundary=box.boundaries.all, physics=heat, value=300)
```

Repository cross-reference test_public_stationary, test_public_lumped_heat_solids, test_convective_heat_volume.

Density and discontinuities in a shock tube

The package represents incompressible and compressible flow, shallow water, phase transport, Darcy flow and selected viscoelastic or reactive mixture models.



Independent reference. Density.

$$\partial_t U + \partial_x F(U) = 0, \qquad U = (\rho, \rho u, E)$$

Setup Use the Sod initial states (rho, u, p) = (1, 0, 1) and (0.125, 0, 0.1), with gamma = 1.4 and diaphragm at x = 0.5. Sample the exact Riemann solution at t = 0.18.

Read the result From left to right, the density profile resolves the rarefaction, contact and shock. A numerical run will spread or sharpen those transitions according to its reconstruction and limiter.

OneSimulation connection CompressibleEuler, Rusanov and TransmissiveBoundary provide a public conservative configuration. The repository's Sod test references a separate verification package.

API IN PRACTICE

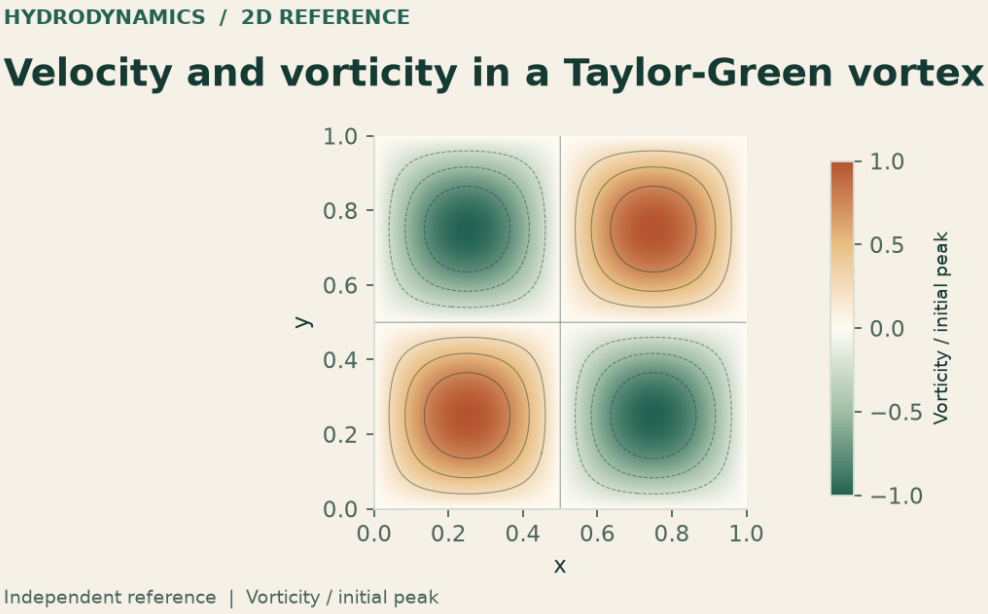
Specify the ideal-gas closure and a Rusanov flux for the shock tube Line. The left and right conservative states enter as InitialValue declarations.

```
from onesim.physics.hydrodynamics import CompressibleEuler, IdealGasEquationOfState
from onesim.numerics.fvm import FiniteVolume, Rusanov

gas = CompressibleEuler(domain=tube.line,
                        equation_of_state=IdealGasEquationOfState(gamma=1.4))
space = FiniteVolume(convection=Rusanov())
```

Selected API names IncompressibleNavierStokes, RotatingWall, NoSlipWall, FixedVelocity, FlowTraction, PressureTraction, CompressibleEuler, EulerInvariantDomain, TransmissiveBoundary. The full export list appears in the appendix.

Velocity and vorticity in a Taylor-Green vortex



Independent reference. Vorticity / initial peak.

$$u = \sin(2\pi x)\cos(2\pi y), \quad v = -\cos(2\pi x)\sin(2\pi y)$$

Setup Use a periodic square and plot the normalized initial vorticity of the divergence-free velocity field. A viscous run has exponential decay in this two-dimensional solution.

Read the result Four alternating vortices fill the square. Periodicity and incompressibility are immediate checks for the spatial operator.

OneSimulation connection IncompressibleNavierStokes is exercised with actual Cartesian momentum and pressure gauge tests. This field is an independent periodic velocity reference.

API IN PRACTICE

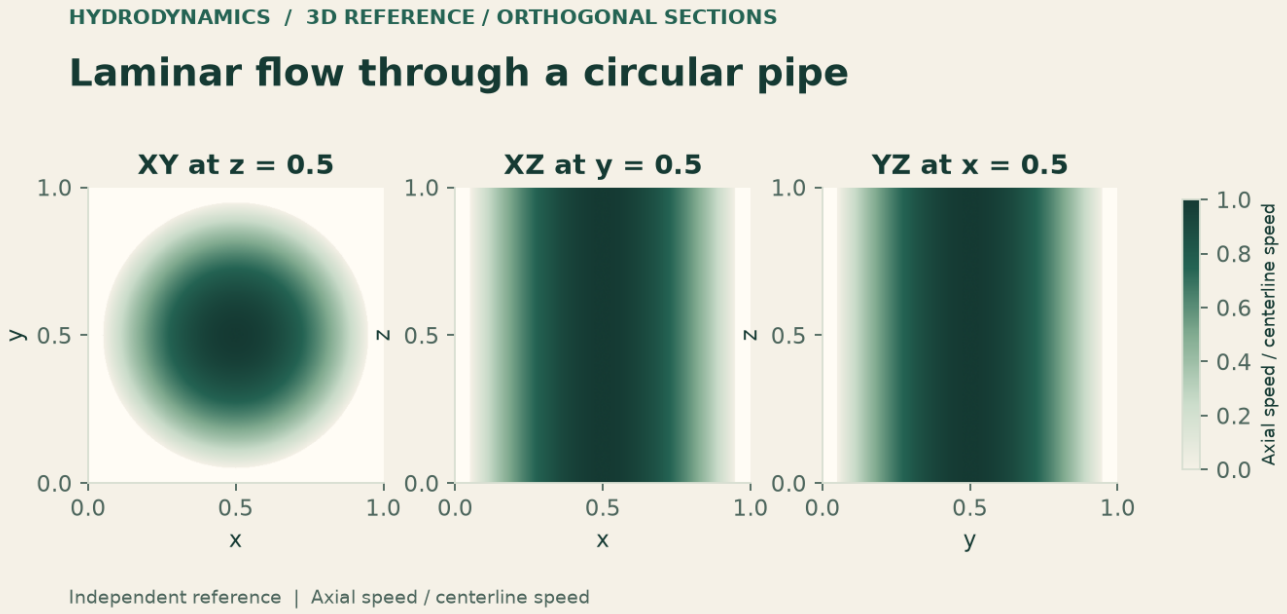
Initialize the velocity mode on a Cartesian volume. Use the 2D vortex as a plane-invariant reference and specify compatible periodic pairs in the study.

```
from onesim.physics.hydrodynamics import IncompressibleNavierStokes
from onesim.simulation.initial_conditions import InitialValue

flow = IncompressibleNavierStokes(domain=box.volume, material=fluid)
initial_vortex = InitialValue(flow.velocity, taylor_green_velocity)
```

Repository cross-reference test_public_incompressible_flow, test_public_compressible_euler, test_shallow_water_sources.

Laminar flow through a circular pipe



Independent reference. Axial speed / centerline speed.

$$u_z/U_{\max} = 1 - (r/R)^2$$

Setup Take a pressure-driven fully developed pipe and mask points outside its circular cross-section. The axial speed is constant along the pipe at a fixed radius.

Read the result The end section is parabolic in radius. Longitudinal sections show no axial variation and zero velocity at the no-slip wall.

OneSimulation connection An analytical benchmark for a three-dimensional incompressible model with a cylindrical domain. It checks a different regime from a transient vortex or shock.

API IN PRACTICE

Assign no slip to the cylinder's lateral wall. Define the inlet and outlet pressures separately to drive the pipe.

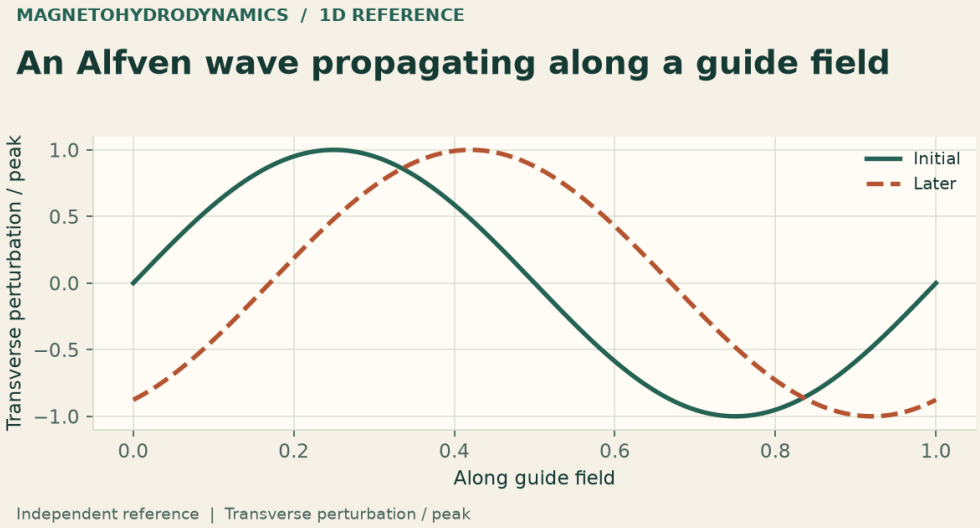
```
from onesim.physics.hydrodynamics import IncompressibleNavierStokes, NoSlipWall

flow = IncompressibleNavierStokes(domain=pipe.volume, material=fluid)
wall = NoSlipWall(boundary=pipe.boundaries.outer_wall, physics=flow)
fields = (flow.velocity, flow.pressure)
```

Repository cross-reference test_public_incompressible_flow, test_public_compressible_euler, test_shallow_water_sources.

An Alfven wave propagating along a guide field

TransverseMHDWaves represents perturbations about a background state. The package also provides inductionless momentum, two-fluid exchange and radiation-matter components.



Independent reference. Transverse perturbation / peak.

$$u_y = U\sin[2\pi(x - v_A t)], \qquad v_A = B_0/\sqrt{\mu\rho}$$

Setup Compare the initial sinusoidal perturbation with a later state after a normalized shift of 0.17 wavelengths. Take ideal MHD and a uniform guide field.

Read the result The wave translates without changing its amplitude. Coupled magnetic and velocity perturbations travel together in a selected polarization.

OneSimulation connection TransverseMHDWaves is exercised with both polarizations, oriented waves and selected FD, FV and DG operators.

API IN PRACTICE

Set the guide field and medium on a Line. Use velocity_y and magnetic_y to declare the matching initial polarization.

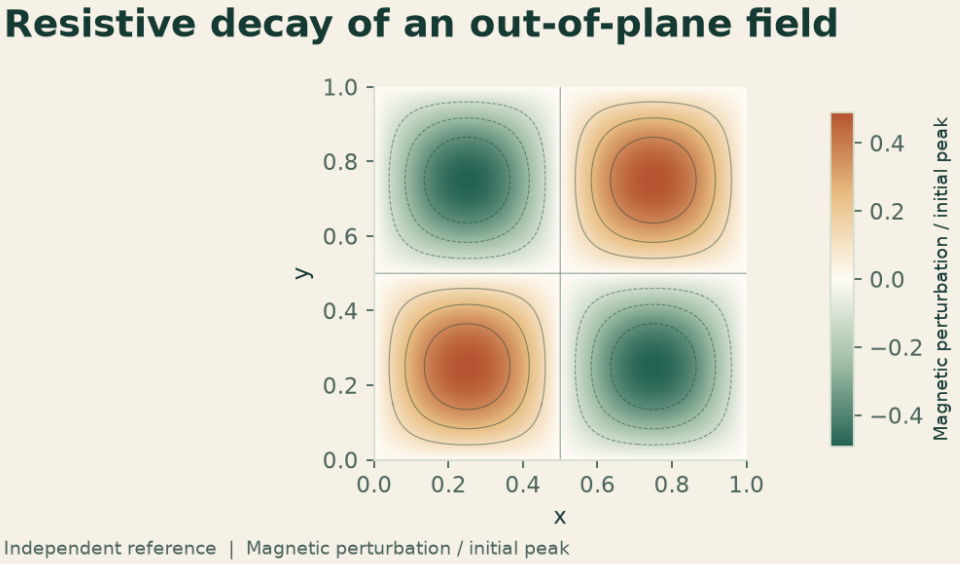
```
from onesim.physics.magnetohydrodynamics import TransverseMHDWaves

wave = TransverseMHDWaves(domain=guide.line, density=1.0,
                          longitudinal_field=1.0, permeability=1.0)
fields = (wave.velocity_y, wave.magnetic_y)
```

Selected API names TransverseMHDWaves, MAGNETOHYDRODYNAMICS, TwoFluidExchange, InductionlessMomentum, InductionlessNoSlipWall, RadiationMatterExchange. The full export list appears in the appendix.

Resistive decay of an out-of-plane field

MAGNETOHYDRODYNAMICS / 2D REFERENCE



Independent reference. Magnetic perturbation / initial peak.

$$B_z/B_0 = e^{-8\pi^2\eta t}\sin(2\pi x)\sin(2\pi y)$$

Setup Use a periodic square with no advection to isolate the magnetic diffusion term. Take normalized eta t = 0.009.

Read the result Alternating lobes keep their phase pattern while resistive dissipation reduces amplitude.

OneSimulation connection A diffusion-limit reference connected to nonideal MHD operators. It does not exercise the Hall term or fluid-field coupling by itself.

API IN PRACTICE

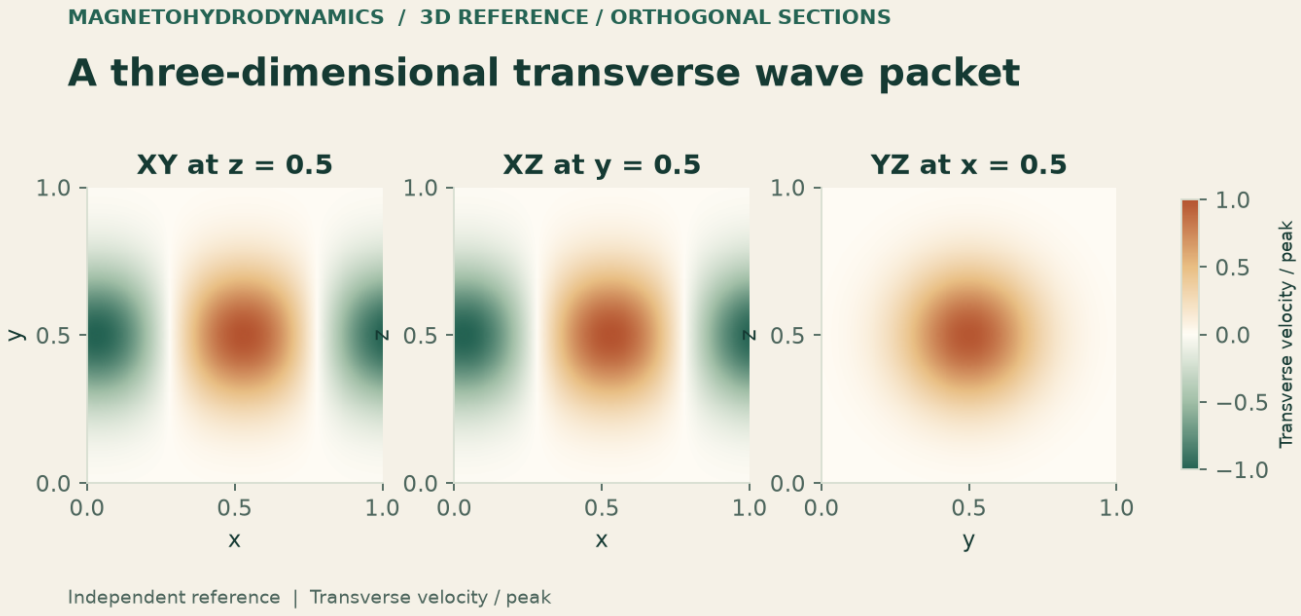
Set magnetic diffusivity on the reduced wave operator. The 2D reference isolates resistive decay of a spatial mode.

```
from onesim.physics.magnetohydrodynamics import TransverseMHDWaves

wave = TransverseMHDWaves(domain=guide.line, density=1.0,
    longitudinal_field=1.0, permeability=1.0, magnetic_diffusivity=0.009)
induction_to_record = wave.magnetic_z
```

Repository cross-reference test_transverse_mhd_method_flexibility, test_homogeneous_mhd_input_flexibility.

A three-dimensional transverse wave packet



Independent reference. Transverse velocity / peak.

$$u_y/U = \sin[2\pi(x - x_0)]\exp[-((y - y_0)^2 + (z - z_0)^2)/s^2]$$

Setup Use a prescribed transverse envelope around an x-directed guide field. Slice the kinematic packet through its transverse center.

Read the result Longitudinal slices show phase bands. The cross-section shows the transverse localization. This scalar component alone does not specify a solenoidal full MHD state.

OneSimulation connection A visualization of the perturbation one could target in a volume study. The demonstrated repository wave path is a reduced transverse model.

API IN PRACTICE

Pass the axial packet as a profile on the line model's transverse velocity. The analytical volume applies a separate transverse envelope.

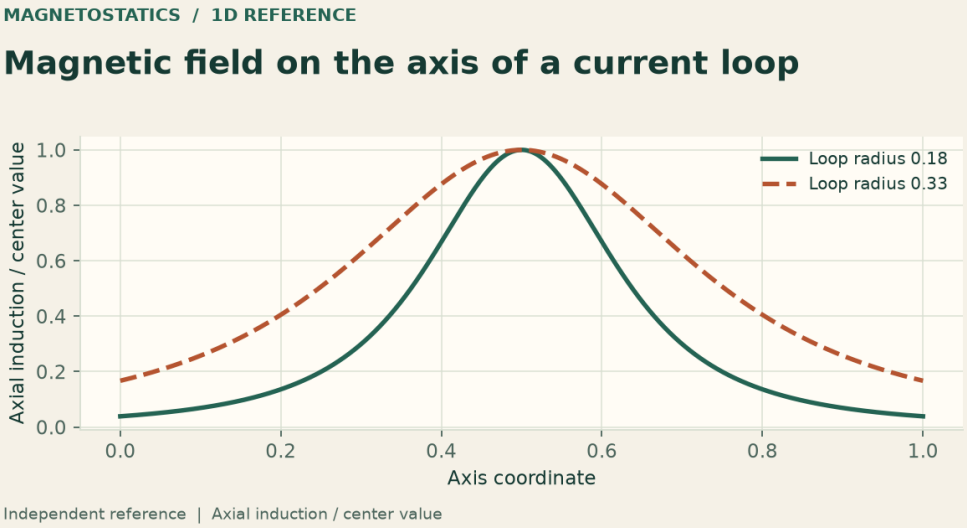
```
from onesim.simulation.initial_conditions import InitialValue

transverse_packet = InitialValue(wave.velocity_y, axial_packet)
paired_induction = InitialValue(wave.magnetic_y, axial_induction)
```

Repository cross-reference test_transverse_mhd_method_flexibility, test_homogeneous_mhd_input_flexibility.

Magnetic field on the axis of a current loop

Magnetostatics represents magnetic vector potential and magnetic induction, with gauge choices, material reluctance and current excitation. The coil study exposes a volume field in three dimensions.



Independent reference. Axial induction / center value.

$$B_z(z)/B_z(0) = [1 + (z/R)^2]^{-3/2}$$

Setup Sample along the axis of a thin circular loop and compare radii at fixed normalized on-axis center field.

Read the result The axial induction peaks at the center and decays cubically far from the loop. This line measurement is embedded in a three-dimensional field.

OneSimulation connection SolenoidCurrent can excite a resolved current distribution. The thin-loop law is useful for axis checks and has a different source geometry from a thick coil.

API IN PRACTICE

Represent a thin loop with a one-turn source of small but finite cross-section. Use axis_points to sample the field after solving the coil study.

```
from onesim.physics.magnetostatics import SolenoidCurrent

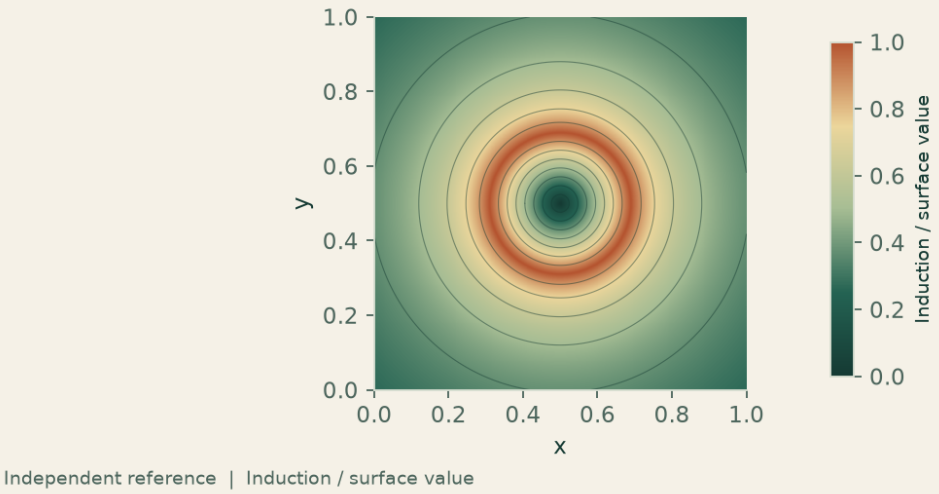
loop = SolenoidCurrent(inner_radius=0.049, outer_radius=0.051, length=0.002,
                       turns=1, current=1.0)
axis_induction = result.field(coil_model.magnetic_flux_density).evaluate(axis_points)
axial_component = axis_induction[:, 2]
```

Selected API names Magnetostatics, MagneticVectorPotential, CoulombGauge, SolenoidCurrent, MagneticInsulation, MagneticPotentialBoundary, MagneticFieldBoundary, MagneticReluctanceBoundary. The full export list appears in the appendix.

Cross-section around a straight conductor

MAGNETOSTATICS / 2D REFERENCE

Cross-section around a straight conductor



Independent reference. Induction / surface value.

$$B_\theta/B(R) = r/R \ (r \le R), \qquad B_\theta/B(R) = R/r \ (r > R)$$

Setup Use a long conductor carrying uniform current with normalized radius 0.19. Plot induction magnitude in the perpendicular plane.

Read the result The induction grows linearly inside, peaks at the surface and decreases outside. The corresponding vector direction is azimuthal.

OneSimulation connection A cross-section reference for magnetic-field sampling and current normalization. This example uses Ampere's law and omits finite-length effects.

API IN PRACTICE

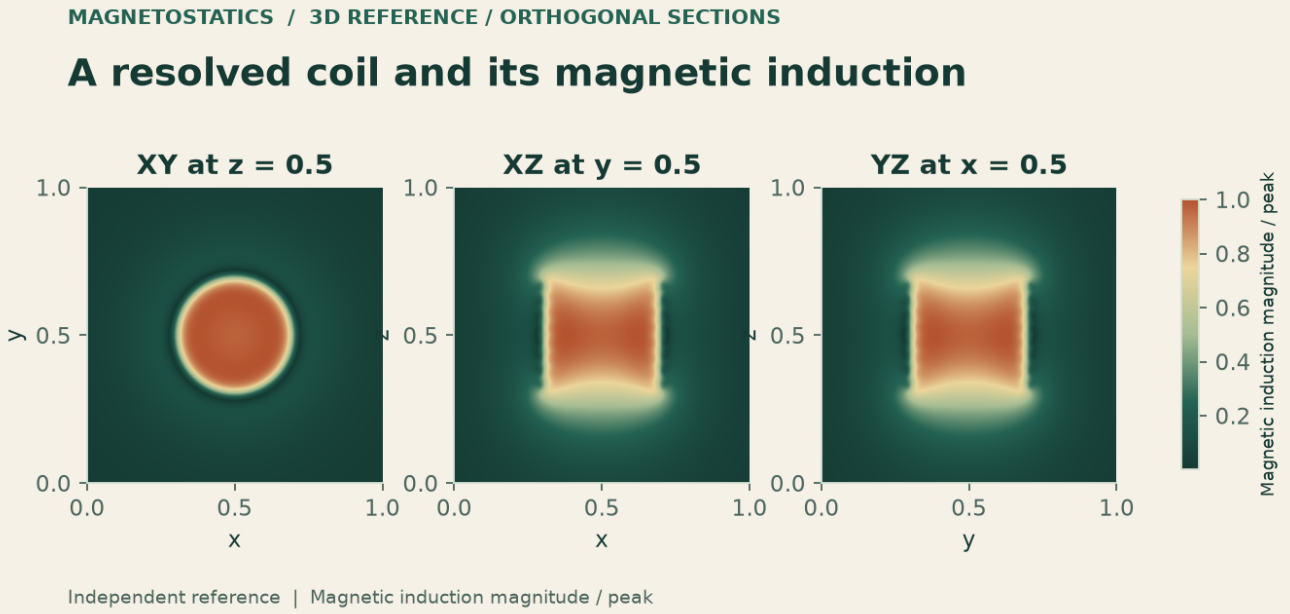
Sample a solved magnetic field on a perpendicular section. section_points contains the world coordinates of the desired plane.

```
import numpy as np

B = result.field(conductor_model.magnetic_flux_density)
section_magnitude = np.linalg.norm(B.evaluate(section_points), axis=1)
```

Repository cross-reference test_public_magnetostatics, test_magnetic_boundary_ports, test_magnetic_gauge_ports.

A resolved coil and its magnetic induction



Independent reference. Magnetic induction magnitude / peak.

$$\nabla \times (\nu \nabla \times \mathbf{A}) = \mathbf{J}, \qquad \mathbf{B} = \nabla \times \mathbf{A}$$

Setup Use a coil in a surrounding volume. For the analytical illustration, superpose circular current loops with finite radius and multiple axial stations.

Read the result Central slices show the high-induction region in and near the coil. A separate runtime plate presents a computed FEM field from the supplied repository example.

OneSimulation connection Magnetostatics, SolenoidCurrent, Nedelec and CoulombGauge connect current, edge-space potential and magnetic induction.

API IN PRACTICE

Build the coil source directly and select the gauge for the vector potential. The volume mesh must resolve coil.support.

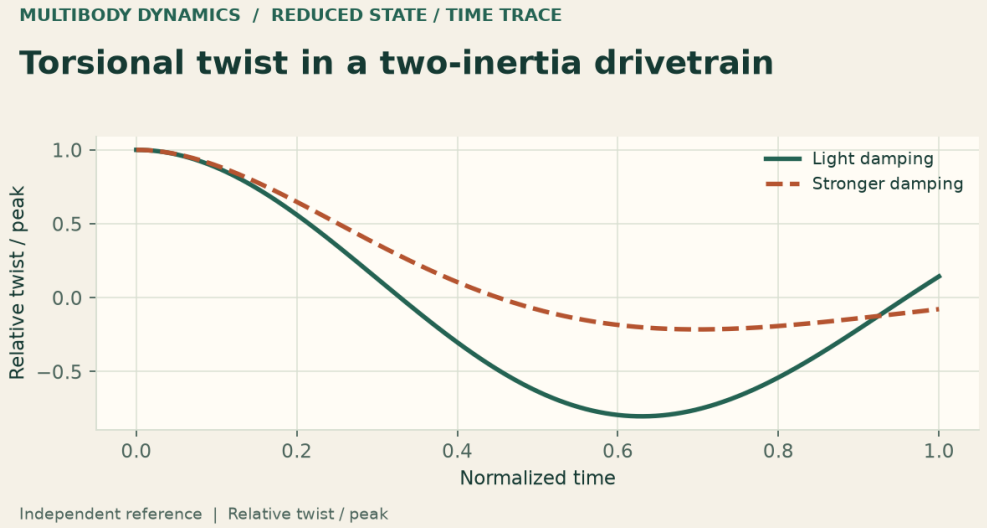
```
from onesim.physics.magnetostatics import SolenoidCurrent
from onesim.physics.magnetostatics import CoulombGauge, MagneticVectorPotential

coil = SolenoidCurrent(inner_radius=0.02, outer_radius=0.03, length=0.02,
                       center=(0, 0, 0.04), turns=10, current=1.0)
formulation = MagneticVectorPotential(gauge=CoulombGauge())
```

Repository cross-reference test_public_magnetostatics, test_magnetic_boundary_ports, test_magnetic_gauge_ports.

Torsional twist in a two-inertia drivetrain

The package exposes mechanism and rigid-body models such as DrivenFourBar, EulerTop, SimplePendulum, linear joints and torsional drivetrains. A trajectory may occupy a 3D space while the system evolves only a handful of coordinates.



Independent reference. Relative twist / peak.

$$I_{\text{eff}}\ddot{\theta} + c\dot{\theta} + k\theta = 0$$

Setup Release an initial relative shaft twist. Compare lightly and strongly damped responses in the relative mode with normalized natural frequency 5.

Read the result Higher damping removes torsional ringing faster. The relative mode is distinct from rigid rotation of both inertias together.

OneSimulation connection TorsionalDrivetrain and LinearJointSystem expose reduced mechanical state. This is a single-coordinate reduction.

API IN PRACTICE

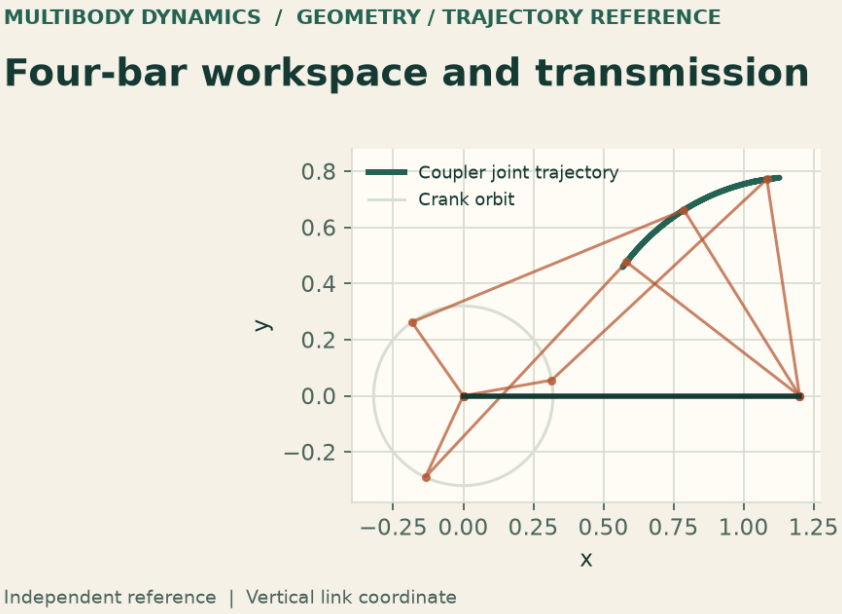
Declare two inertias and their connecting shaft on a LumpedDomain named machine. Stiffness and damping are independent inputs.

```
from onesim.physics.multibody import TorsionalDrivetrain

shaft = TorsionalDrivetrain(domain=machine.system, inertias=(1.0, 1.0),
                            shaft_stiffness=(12.5,), shaft_damping=(0.35,))
energy_to_record = shaft.energy
```

Selected API names LinearJointSystem, DrivenFourBar, TorsionalDrivetrain, EulerTop, SimplePendulum. The full export list appears in the appendix.

Four-bar workspace and transmission



Independent reference. Vertical link coordinate.

$$|\mathbf{B}-\mathbf{A}| = b, \qquad |\mathbf{B}-\mathbf{D}| = c$$

Setup Close a planar four-bar by intersecting two circles for each crank angle. Use ground length 1.2, crank 0.32, coupler 1.05 and rocker 0.78. Follow one assembly branch.

Read the result The coupler joint traces a curved arc. The selected crank positions also show how link orientation changes around the loop.

OneSimulation connection DrivenFourBar provides a mechanism model. The plotted geometry is an independent closure calculation for the specified lengths.

API IN PRACTICE

Declare the lengths and assembly branch on the mechanism's LumpedDomain. The geometry of the linkage follows from its input angle.

```
from onesim.physics.multibody import DrivenFourBar

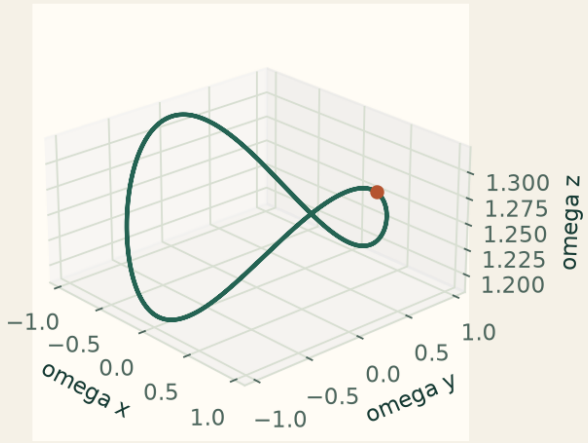
linkage = DrivenFourBar(domain=mechanism.system, ground_length=1.2,
    crank_length=0.32, coupler_length=1.05, rocker_length=0.78,
    drive_speed=1.0, assembly_branch="open")
angle_to_record = linkage.input_angle
```

Repository cross-reference test_multibody_input_flexibility.

Body-frame motion of a torque-free rigid top

MULTIBODY DYNAMICS / GEOMETRY / TRAJECTORY REFERENCE

Body-frame motion of a torque-free rigid top



Independent reference | Angular velocity magnitude / initial

Independent reference. Angular velocity magnitude / initial.

$$\mathbf{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times (\mathbf{I}\boldsymbol{\omega}) = 0$$

Setup Integrate Euler's rigid-body equations for principal inertias 1, 2 and 3 and initial angular velocity (0.9, 0.35, 1.3). Draw the trajectory in body-frame angular-velocity space.

Read the result The path stays on intersecting constant-energy and constant-angular-momentum surfaces up to integration error.

OneSimulation connection EulerTop represents a 3D rigid-body model. The plot shows phase-space coordinates, with no deforming 3D continuum or contact mesh.

API IN PRACTICE

Set the three principal inertias on a rigid body's LumpedDomain. Zero applied torque gives the two invariants to record.

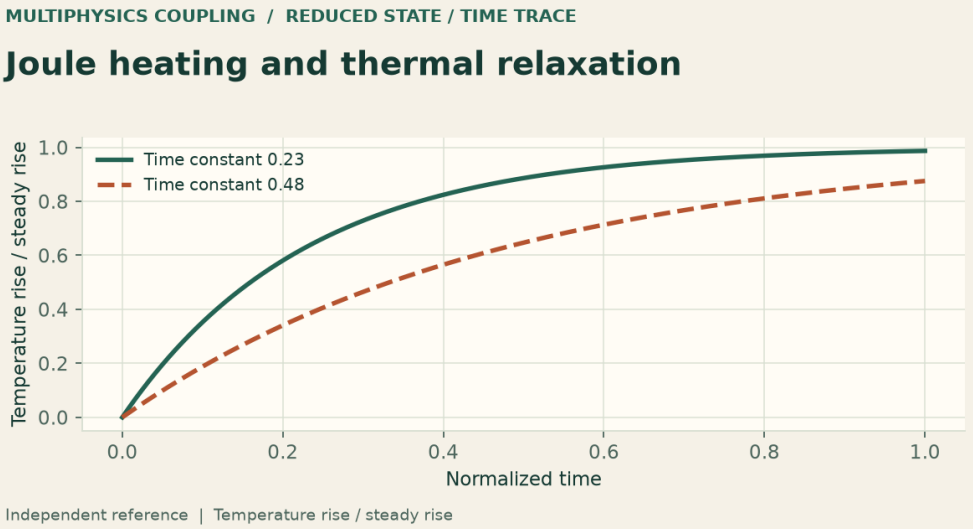
```
from onesim.physics.multibody import EulerTop

top = EulerTop(domain=body.system, inertia=(1.0, 2.0, 3.0))
invariants = (top.energy, top.momentum_norm)
```

Repository cross-reference test_multibody_input_flexibility.

Joule heating and thermal relaxation

Coupling carries physical data between models. Examples include electrical heating, fluid-column and piston dynamics, voice coils, thermal exchange, and an adiabatic reaction.



Independent reference. Temperature rise / steady rise.

$$C\dot{T} = I^2R - G(T - T_\infty)$$

Setup Drive a resistor with constant current and compare thermal relaxation times 0.23 and 0.48. Start at ambient and normalize by each steady rise.

Read the result The electrical input sets the heating power while the thermal capacitance sets the transient. The result approaches a finite steady value.

OneSimulation connection HeatedResistor and ThermalExchange illustrate coupled concentrated states. This is a reduced thermal coordinate rather than a 1D heat field.

API IN PRACTICE

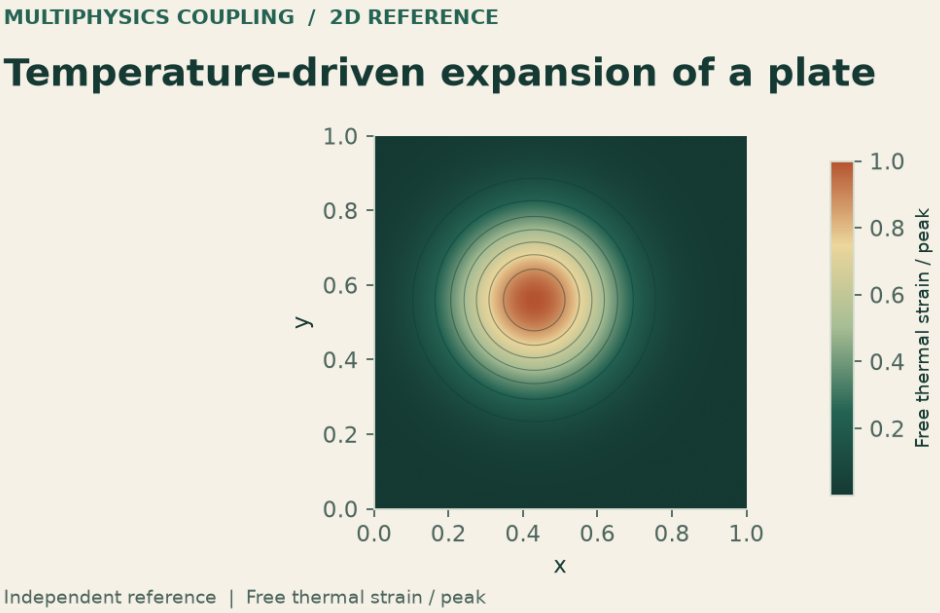
Set voltage and resistance on the electrical heating model. For constant resistance choose V = I R. A thermal conductance supplies the relaxation path.

```
from onesim.physics.multiphysics import HeatedResistor

heater = HeatedResistor(domain=body.system, resistance=10.0,
                        heat_capacity=5.0, applied_voltage=2.0)
temperature_to_record = heater.temperature
```

Selected API names MULTIPHYSICS, NUMERICS_BINDING, HeatedResistor, FluidColumnPiston, VoiceCoil, ThermalExchange, AdiabaticReaction. The full export list appears in the appendix.

Temperature-driven expansion of a plate



Independent reference. Free thermal strain / peak.

$$\boldsymbol{\varepsilon}_{\mathrm{th}} = \alpha(T - T_0)\mathbf{I}$$

Setup Use a Gaussian hot patch on a plate and sample the local free thermal strain associated with its temperature field.

Read the result The largest eigenstrain is at the heated center. Mechanical stress still requires equilibrium and displacement constraints.

OneSimulation connection Field coupling and IsotropicThermalExpansion can pass temperature into a structural calculation. The map shows the driving strain field.

API IN PRACTICE

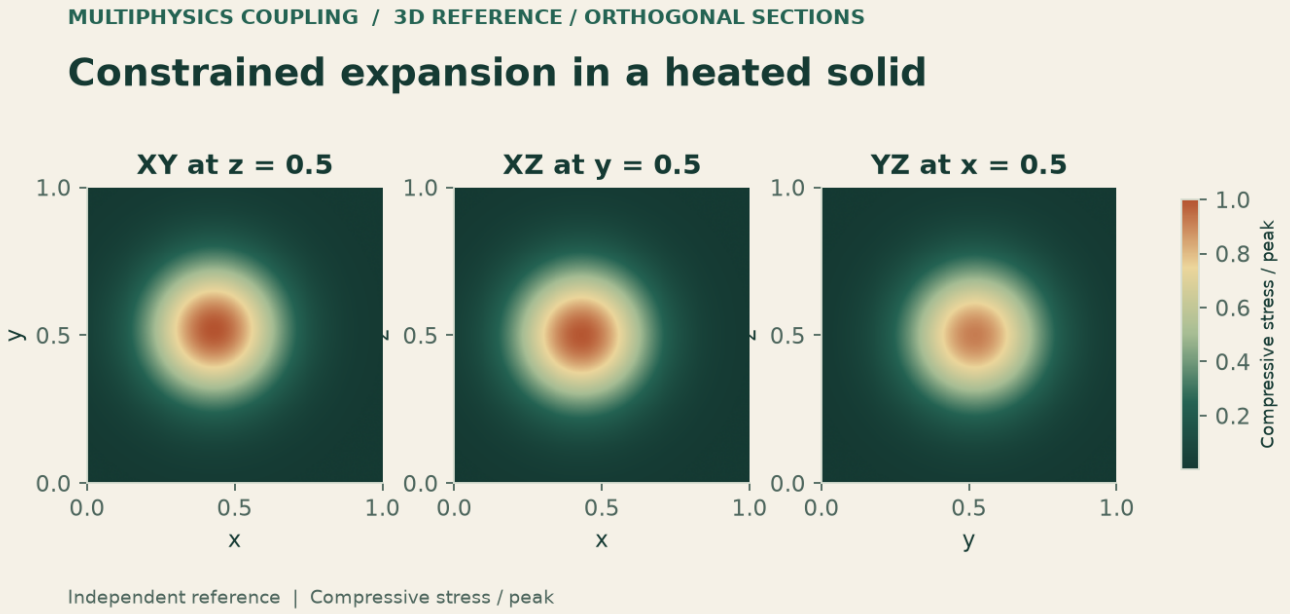
Select the thermal expansion law on a thin solid volume. Its temperature input can receive the solved heat field.

```
from onesim.physics.solid_mechanics import LinearElasticity
from onesim.physics.solid_mechanics import IsotropicThermalExpansion

elasticity = LinearElasticity(domain=plate.volume, material=solid,
    thermal_strain=IsotropicThermalExpansion(reference_temperature=300))
```

Repository cross-reference test_field_coupling, test_mixed_flow_heat_strategies, test_public_biot_consolidation.

Constrained expansion in a heated solid



Independent reference. Compressive stress / peak.

$$\sigma_{xx} = -E\alpha(T - T_0) \quad \text{(uniaxial restrained limit)}$$

Setup Prescribe a temperature rise with an off-center hot region in a box. Map the local restrained stress scale using normalized E alpha = 1.

Read the result The hottest region supplies the largest compressive scale. A full 3D elasticity solve redistributes stresses through compatibility and boundary conditions.

OneSimulation connection A field-transfer and constitutive benchmark for thermoelasticity. The local formula isolates a limit and does not represent the full 3D stress solution.

API IN PRACTICE

Connect the heat field to the elasticity temperature input. heat and elasticity are volume model declarations and transfer evaluates the source finite-element field.

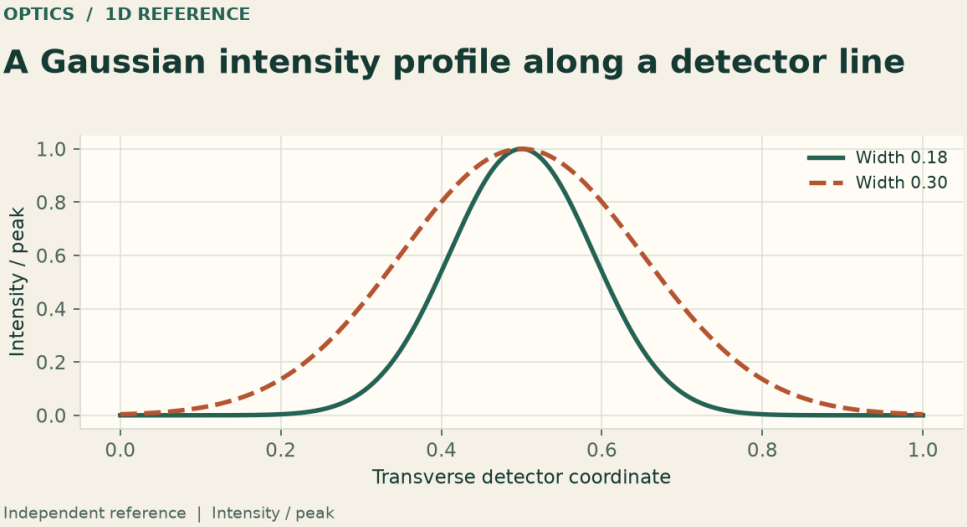
```
from onesim.simulation.field_connections import FieldCoupling
from onesim.numerics.field_transfer import FiniteElementEvaluation

transfer = FiniteElementEvaluation()
thermal_load = FieldCoupling(source=heat.temperature,
                             target=elasticity.temperature, transfer=transfer)
```

Repository cross-reference test_field_coupling, test_mixed_flow_heat_strategies, test_public_biot_consolidation.

A Gaussian intensity profile along a detector line

Paraxial and photon-envelope models expose optical propagation and sources. Complex amplitude, phase and intensity are distinct outputs with different numerical checks.



Independent reference. Intensity / peak.

$$I(x)/I_0 = \exp(-2x^2/w^2)$$

Setup Sample a Gaussian beam on a transverse detector line for normalized widths 0.18 and 0.30, centered at x = 0.5.

Read the result The wider beam has a flatter peak and spreads its intensity over a larger interval. In a power-conserving comparison the peak must also be rescaled.

OneSimulation connection ParaxialEnvelope and ComplexParaxialEnvelope carry amplitude; intensity is formed from its modulus squared.

API IN PRACTICE

Initialize the complex field amplitude on a detector Line. The amplitude width here yields the intensity width in the plot.

```
import numpy as np
from onesim.simulation.initial_conditions import InitialValue

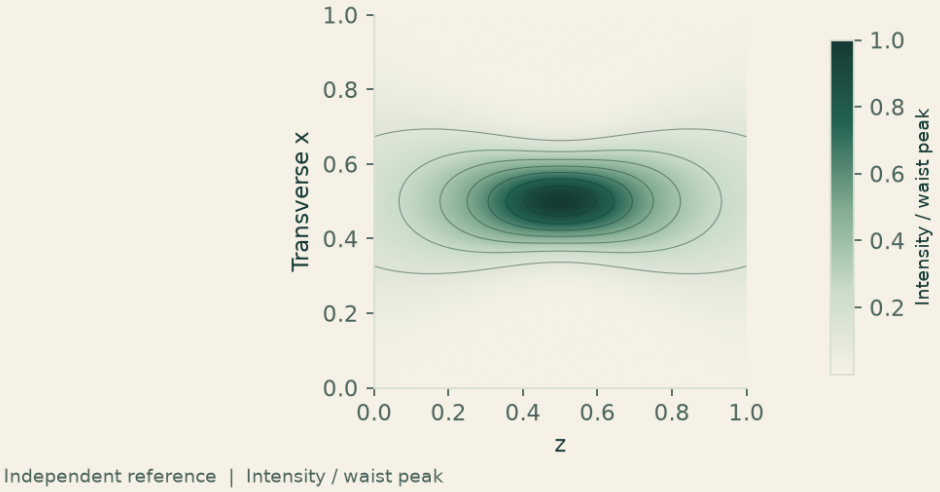
amplitude = lambda x: np.exp(-((x - 0.5) / 0.18)**2)
beam_initial = InitialValue(beam.envelope, amplitude)
```

Selected API names ComplexParaxialEnvelope, ParaxialEnvelope, PhotonEnvelope. The full export list appears in the appendix.

Diffraction and the beam waist

OPTICS / 2D REFERENCE

Diffraction and the beam waist



Independent reference | Intensity / waist peak

Independent reference. Intensity / waist peak.

$$I(x,z)/I_0 = [w_0/w(z)]^2 e^{-2x^2/w(z)^2}, \quad w(z) = w_0\sqrt{1+(z/z_R)^2}$$

Setup Plot a longitudinal section through a Gaussian beam that focuses at z = 0.5. Use normalized waist 0.16 and Rayleigh range 0.25.

Read the result The beam narrows near the focus and widens on either side. Peak intensity falls away from the waist.

OneSimulation connection This exact paraxial envelope is an independent benchmark. Repository tests cover complex amplitudes, selected methods, boundary traces and output tangents.

API IN PRACTICE

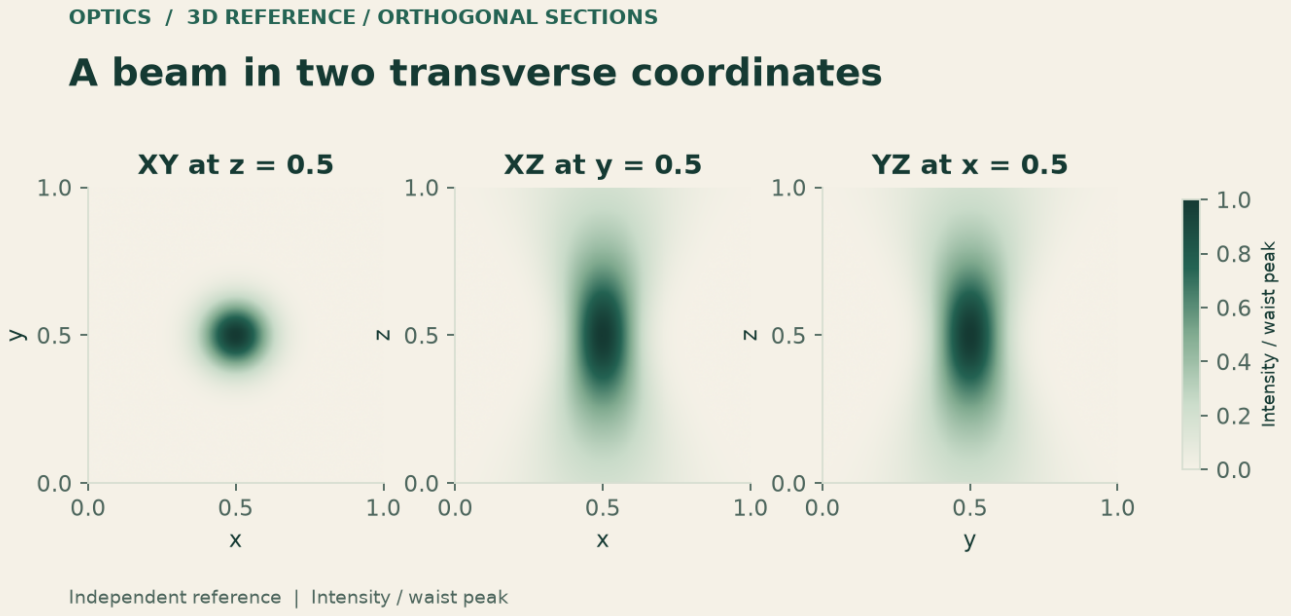
Declare the complex paraxial envelope on a line grid. Stacking transverse profiles is the visualization used for the independent waist reference.

```
from onesim.physics.optics import ComplexParaxialEnvelope

beam = ComplexParaxialEnvelope(domain=transverse.line, wavenumber=100.0,
    group_speed=1.0, permittivity=1.0, reference_index=1.0)
amplitude_to_record = beam.envelope
```

Repository cross-reference test_optics_method_flexibility, test_continuous_optics_flexibility, test_optics_time_flexibility.

A beam in two transverse coordinates



Independent reference. Intensity / waist peak.

$$I(x, y, z)/I_0 = [w_0/w(z)]^2 \exp[-2(x^2 + y^2)/w(z)^2]$$

Setup Evaluate a Gaussian field in a propagation volume. A transverse slice through the waist shows the radial spot and longitudinal slices show its divergence.

Read the result The same beam can be read as a sequence of transverse planes or a 3D intensity volume. It remains a paraxial envelope model.

OneSimulation connection A propagation volume is different from a full three-dimensional vector Maxwell solve. Use the envelope interfaces for paraxial physics.

API IN PRACTICE

Read intensity from the envelope model as a derived field. This is the reduced propagation observable used to interpret the 3D reference slices.

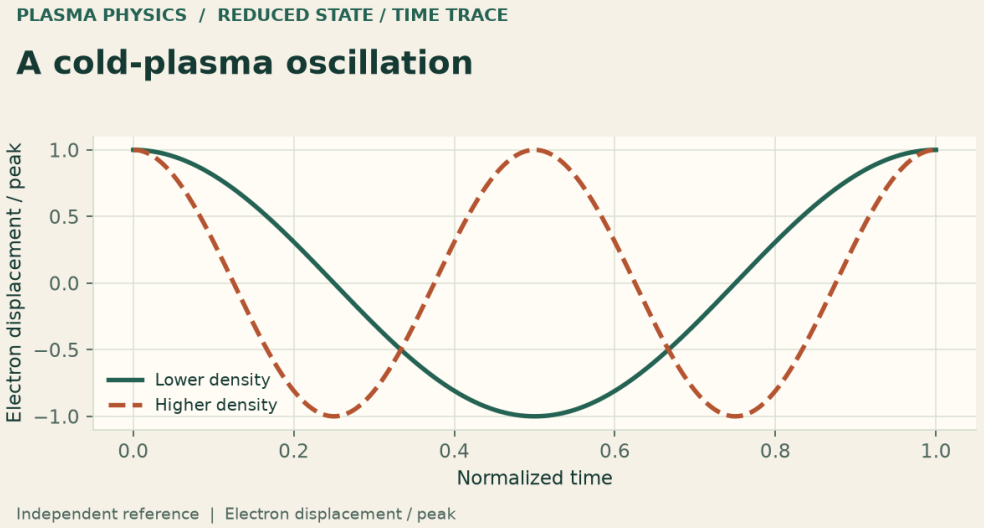
```
from onesim.physics.optics import ParaxialEnvelope

beam = ParaxialEnvelope(domain=transverse.line, wavenumber=100.0,
                        group_speed=1.0, permittivity=1.0)
intensity_to_record = beam.intensity
```

Repository cross-reference test_optics_method_flexibility, test_continuous_optics_flexibility, test_optics_time_flexibility.

A cold-plasma oscillation

Charged-particle models, cold-plasma response, drift-diffusion, electron energy and a native ECR pathway expose progressively richer plasma states.



Independent reference. Electron displacement / peak.

$$\ddot{x} + \omega_p^2 x = 0, \qquad \omega_p^2 = n_0 e^2/(\varepsilon_0 m_e)$$

Setup Perturb a uniform electron population against an immobile ion background. Compare normalized oscillation frequencies 2 pi and 4 pi on the same time interval.

Read the result Higher background electron density raises the plasma frequency. The cold collisionless limit retains constant amplitude.

OneSimulation connection ColdPlasmaOscillation exposes the reduced model. The plotted coordinate is a displacement evolved in time, with no resolved sheath.

API IN PRACTICE

Set density and zero collision frequency on a reduced electron model. Record velocity and electric field to check the oscillation frequency.

```
from onesim.physics.plasma import ColdPlasmaOscillation

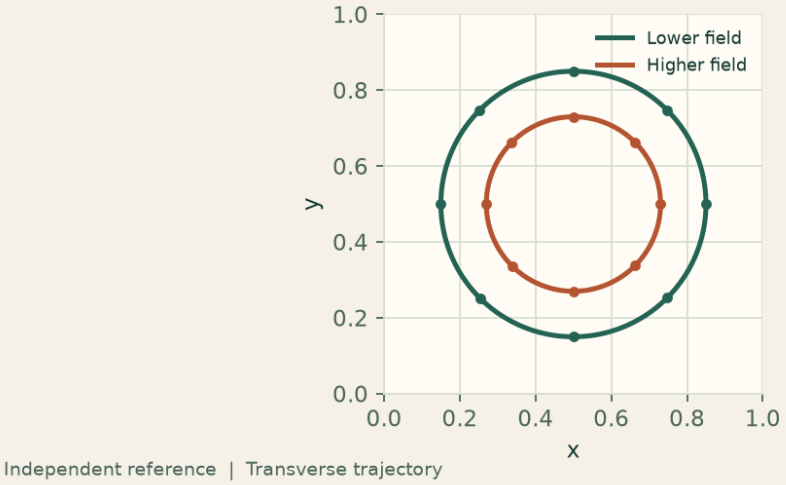
plasma = ColdPlasmaOscillation(domain=body.system, density=1e15,
    electron_mass=9.109e-31, elementary_charge=1.602e-19,
    permittivity=8.854e-12, collision_frequency=0.0)
fields = (plasma.velocity, plasma.electric_field)
```

Selected API names NativeEcrPotentialGauge, NativeEcrPlasma, EcrExcitation, NativeEcrDiscretization, MagnetizedColdPlasma, DriftDiffusionPlasma, MeanElectronEnergyEquation, MagnetizedElectronTransport, IonMobilityModel. The full export list appears in the appendix.

Gyromotion in a uniform magnetic field

PLASMA PHYSICS / GEOMETRY / TRAJECTORY REFERENCE

Gyromotion in a uniform magnetic field



Independent reference | Transverse trajectory

Independent reference. Transverse trajectory.

$$\omega_c = |q|B/m, \qquad r_L = v_\perp/\omega_c$$

Setup Track one idealized charged particle in the plane perpendicular to a uniform magnetic field. Compare two gyroradii and indicate phase along each orbit.

Read the result A stronger field yields a smaller orbit at fixed transverse speed. In the uniform field, speed and orbital radius remain constant.

OneSimulation connection ChargedParticle and UniformGuidingCenter expose reduced particle and guiding-center descriptions. The figure is an exact single-particle reference.

API IN PRACTICE

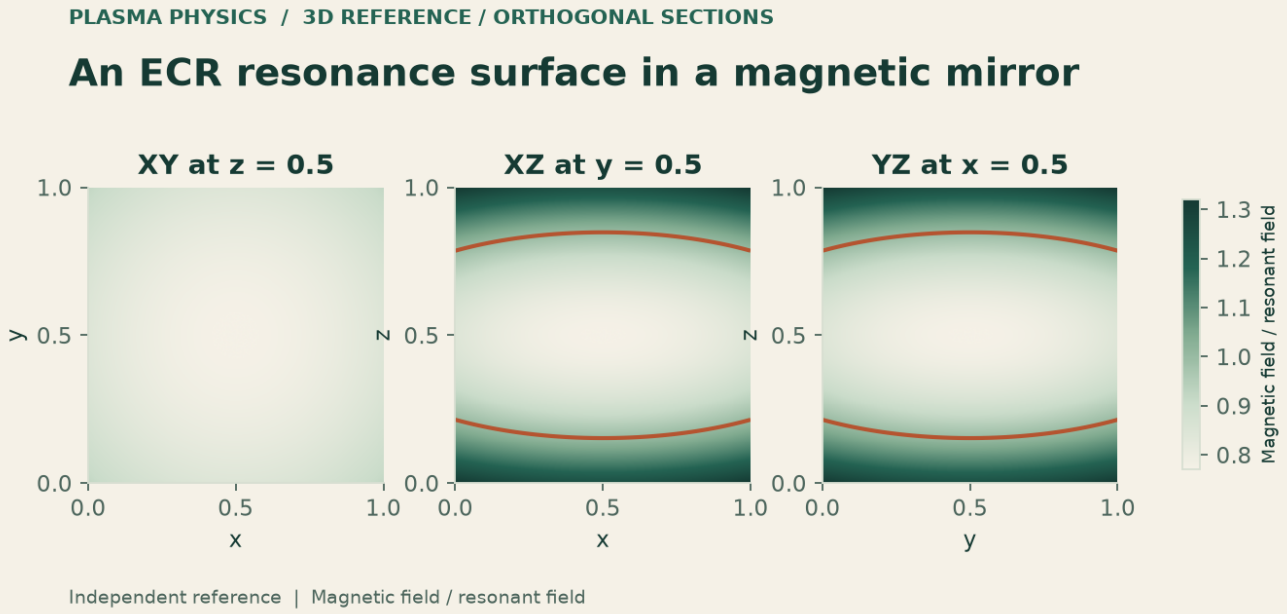
Apply a uniform magnetic field and initialize transverse velocity on the particle's LumpedDomain. Record the two position components for the orbit.

```
from onesim.physics.plasma import ChargedParticle
from onesim.simulation.initial_conditions import InitialValue

ion = ChargedParticle(domain=body.system, mass=1.673e-27,
                      charge=1.602e-19, magnetic=(0, 0, 0.02))
kick = InitialValue(ion.velocity_x, 1e4)
```

Repository cross-reference test_public_plasma_balance, test_plasma_fe_flexibility, test_plasma_volume_mesh_flexibility.

An ECR resonance surface in a magnetic mirror



Independent reference. Magnetic field / resonant field.

$$\omega_{\mathrm{drive}} = |e|B/m_e$$

Setup Construct a normalized magnetic-field magnitude with axial mirror variation and weaker radial variation. The contour at B/Bres = 1 locates the resonant surface.

Read the result The slices reveal two mirror-side resonance regions. The field magnitude alone cannot predict deposited RF power, ionization or steady plasma density.

OneSimulation connection NativeEcrPlasma connects electromagnetic excitation, species, energy and transport. This prescribed field highlights one diagnostic used in such a setup.

API IN PRACTICE

Declare the drive for the native ECR model. The geometry, chemistry, transport and wall conditions remain separate inputs.

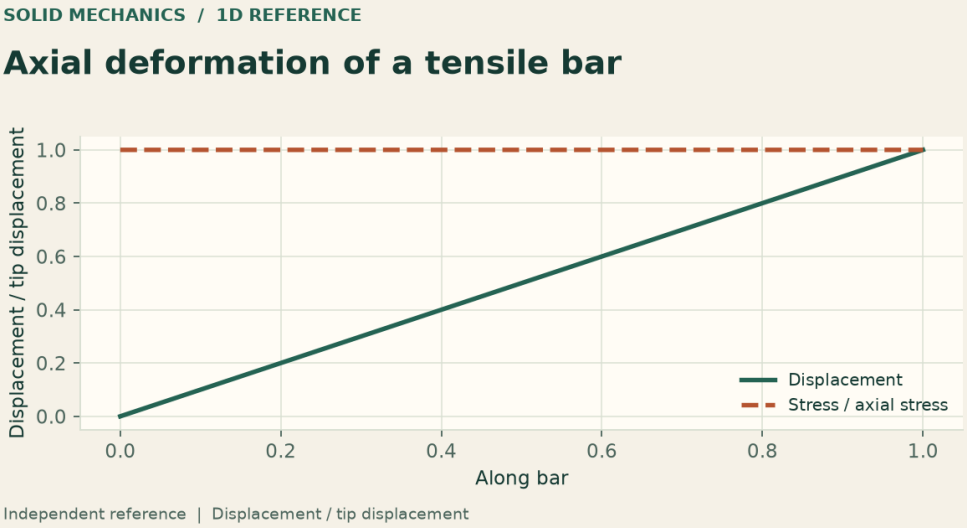
```
from onesim.physics.plasma import EcrExcitation, NativeEcrPlasma

drive = EcrExcitation(frequency=2.45e9, power=100.0,
                      impressed_current_density=rf_current)
model = NativeEcrPlasma(domain=chamber.volume, chemistry=chemistry,
    transport_model=transport, excitation=drive, walls=walls)
```

Repository cross-reference test_public_plasma_balance, test_plasma_fe_flexibility, test_plasma_volume_mesh_flexibility.

Axial deformation of a tensile bar

Linear elasticity, thermal expansion and boundary tractions cover spatial structural problems. The package also includes oscillation, viscoelastic relaxation, impact, fatigue and cohesive opening models.



Independent reference. Displacement / tip displacement.

$$u(x) = Fx/(EA), \qquad \sigma = F/A$$

Setup Clamp a uniform bar at x = 0 and apply a centered axial load at x = 1. Compare displacement with the strain-driven stress scale.

Read the result The displacement increases linearly and the stress is constant away from end effects.

OneSimulation connection LinearElasticity and FixedDisplacement provide the spatial vocabulary. StructuralOscillator addresses the separate concentrated dynamics case in the notebook.

API IN PRACTICE

Put a fixed displacement at one end of a slender volume and convert force to traction on the opposite face. elastic is its LinearElasticity model.

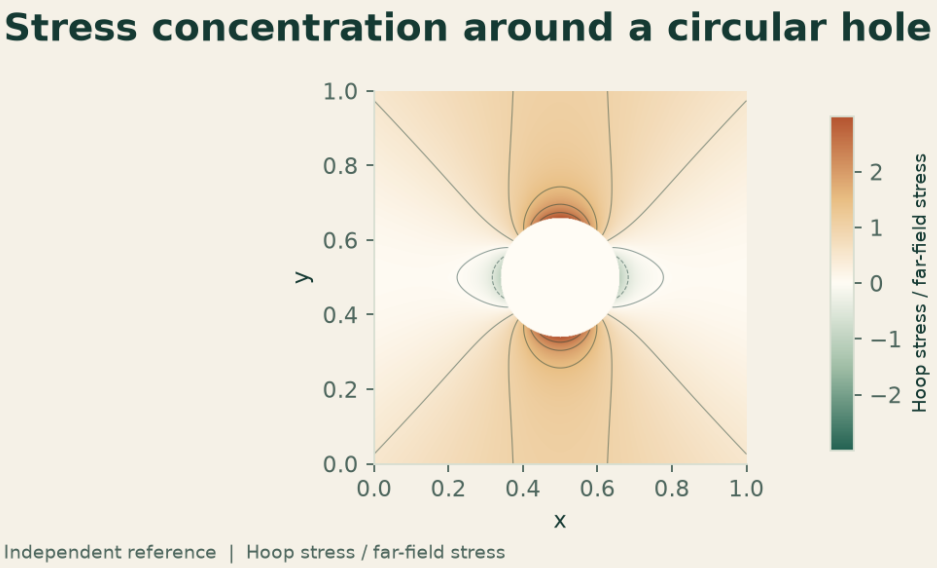
```
from onesim.physics.solid_mechanics import FixedDisplacement, Traction

clamp = FixedDisplacement(boundary=bar.boundaries.x_min,
                          physics=elastic, value=(0, 0, 0))
pull = Traction(boundary=bar.boundaries.x_max, physics=elastic,
                value=(force / area, 0, 0))
```

Selected API names Traction, LinearElasticity, IsotropicThermalExpansion, FixedDisplacement, TractionFree, StructuralOscillator, MaxwellRelaxation, FatigueCrackGrowth, ParisLaw. The full export list appears in the appendix.

Stress concentration around a circular hole

SOLID MECHANICS / 2D REFERENCE



Independent reference. Hoop stress / far-field stress.

$$\sigma_{\theta\theta}(R,\theta)/\sigma_\infty = 1 - 2\cos(2\theta)$$

Setup Evaluate the Kirsch elastic solution for a plate in uniaxial tension. Mask the circular hole and use radius 0.16 in normalized coordinates.

Read the result Hoop stress reaches three times the far-field tension at the transverse edge of the hole, with compressive stress near the loading-axis edge.

OneSimulation connection A plane elasticity reference for testing mesh concentration and stress recovery near curved boundaries.

API IN PRACTICE

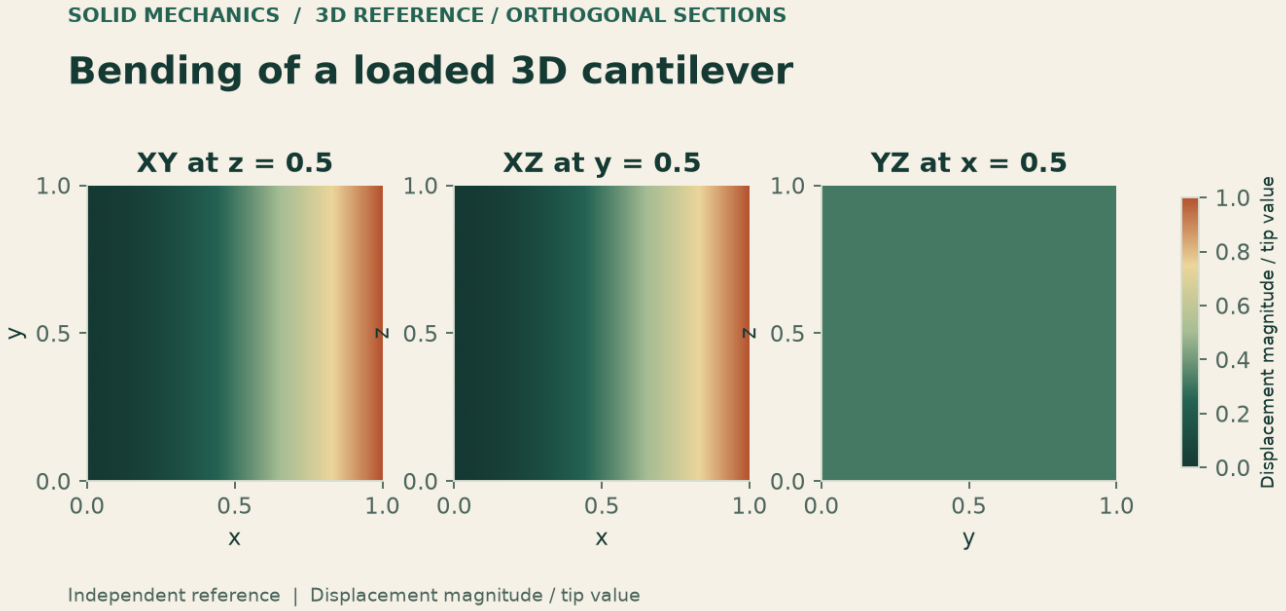
Prescribe far-field tension on a selected loaded edge. After solving the perforated plate, sample the stress near the hole.

```
from onesim.physics.solid_mechanics import Traction

tension = Traction(boundary=loaded_edge, physics=elastic, value=(1e6, 0, 0))
stress = result.field(elastic.stress)
near_hole_stress = stress.evaluate(probe_points)
```

Repository cross-reference test_public_stationary, test_public_lumped_heat_solids, test_p2_equilibrium.

Bending of a loaded 3D cantilever



Independent reference. Displacement magnitude / tip value.

$$u_z(x) = -\frac{Fx^2(3L - x)}{6EI}$$

Setup Clamp a 120 by 18 by 12 mm solid and apply a distributed tip load of 30 N. The reference shows the Euler-Bernoulli centerline shape on three sections.

Read the result The maximum deflection lies at the free end. The handbook also includes the actual Q2-FEM mesh, von Mises field and line-sample comparison.

OneSimulation connection LinearElasticity, StructuredHexahedralMesher and SparseDirect are used in the repository's 3D cantilever demonstration.

API IN PRACTICE

Select Q2 displacement elements and apply 30 N over the tip face of the cantilever. The dimensions below are in meters.

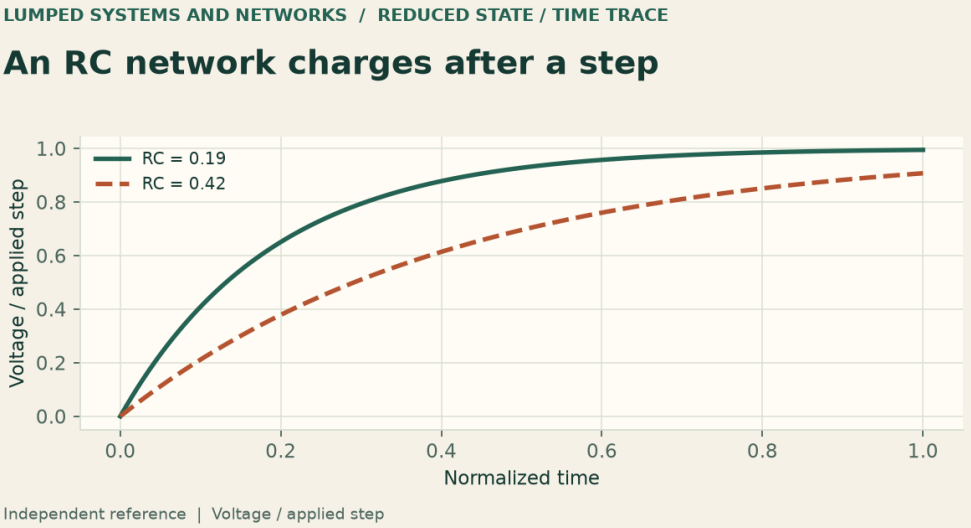
```
from onesim.physics.solid_mechanics import Traction
from onesim.numerics.fem import FiniteElement, Lagrange

tip = Traction(boundary=beam.boundaries.x_max, physics=elastic,
               value=(0, 0, -30 / (0.018 * 0.012)))
space = FiniteElement(element=Lagrange(order=2))
```

Repository cross-reference test_public_stationary, test_public_lumped_heat_solids, test_p2_equilibrium.

An RC network charges after a step

Thermal, hydraulic, pneumatic, process, electrical and signal components can form reduced systems. State dimension counts unknowns; a network diagram may still use spatial coordinates for readability.



Independent reference. Voltage / applied step.

$$C\dot{V} = (V_{\text{in}} - V)/R, \qquad V/V_{\text{in}} = 1 - e^{-t/(RC)}$$

Setup Start a capacitor at zero voltage and compare normalized time constants 0.19 and 0.42 after a unit input step.

Read the result The smaller time constant charges faster. At one time constant each response reaches about 63.2 percent of its steady value.

OneSimulation connection ResistorCapacitor is a reduced electrical component. This chart represents one voltage state and time, without a spatial electrical field.

API IN PRACTICE

Set resistance, capacitance and a constant source. Initialize the capacitor at zero voltage to get the step response.

```
from onesim.physics.systems import ResistorCapacitor
from onesim.simulation.initial_conditions import InitialValue

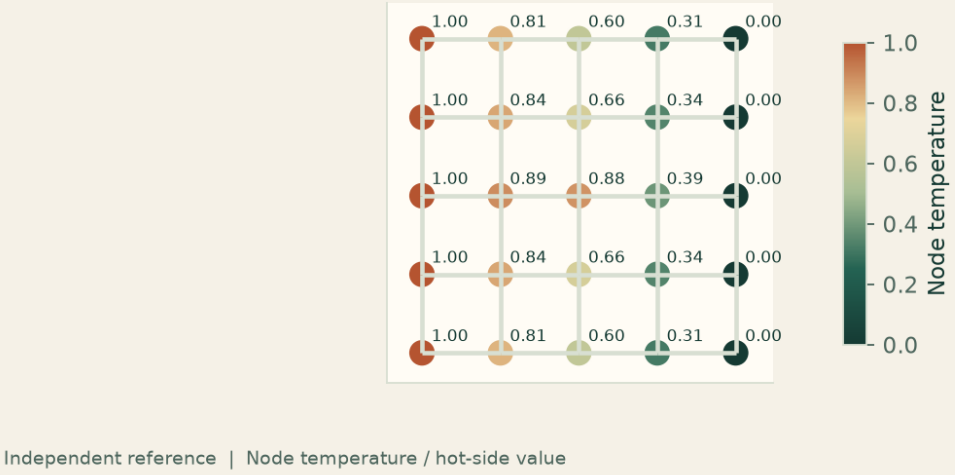
rc = ResistorCapacitor(domain=circuit.system, resistance=1000,
                       capacitance=0.00019, supply_voltage=1.0)
initial_charge = InitialValue(rc.voltage, 0.0)
```

Selected API names SignalBlock, HydraulicBranch, HydraulicNode, PneumaticBranch, PneumaticChamber, ProcessStream, ProcessUnit, ThermalLink, ThermalNode. The full export list appears in the appendix.

Temperature imbalance in a planar thermal network

LUMPED SYSTEMS AND NETWORKS / GEOMETRY / TRAJECTORY REFERENCE

Temperature imbalance in a planar thermal network



Independent reference | Node temperature / hot-side value

Independent reference. Node temperature / hot-side value.

$$\sum_j G_{ij}(T_j - T_i) + Q_i = 0$$

Setup Lay out a 5 by 5 graph with unit links, hot and cold end columns, and a localized central heater. Solve the interior node equations directly.

Read the result The graph highlights lateral heat spreading around the source. Edges carry conductance, and labels denote node temperatures.

OneSimulation connection ThermalNode and ThermalLink are the corresponding graph concepts. The layout is a network, with no continuum shape functions.

API IN PRACTICE

Define one boundary node, one heated node and their conductance. Repeat node and link declarations to assemble the lattice.

```
from onesim.physics.systems import ThermalNode, ThermalLink

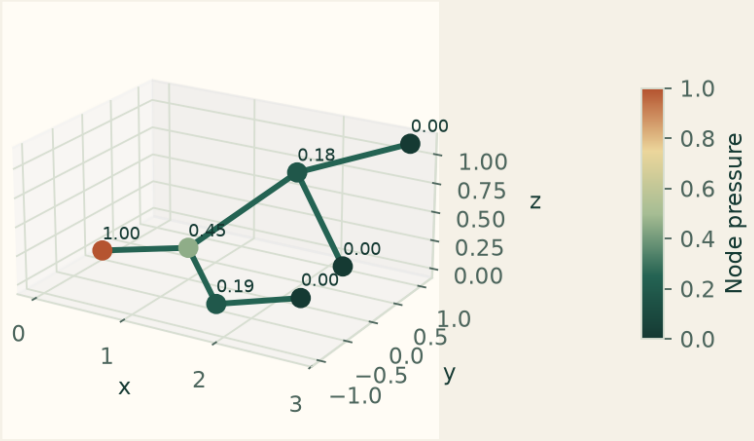
hot = ThermalNode("hot", imposed_temperature=1.0)
center = ThermalNode("center", heat_source=0.9)
edge = ThermalLink("hot-center", "hot", "center", conductance=1.0)
```

Repository cross-reference test_system_network_input_flexibility, test_lumped_solver_matrix.

Pressure distribution in a spatial pipe graph

LUMPED SYSTEMS AND NETWORKS / GEOMETRY / TRAJECTORY REFERENCE

Pressure distribution in a spatial pipe graph



Independent reference | Node pressure / inlet pressure

Independent reference. Node pressure / inlet pressure.

$$\sum_j (p_i - p_j)/R_{ij} = q_i$$

Setup Construct a branching 3D network with one unit-pressure inlet and three zero-pressure outlets. Solve the linear resistive flow balance at its junctions.

Read the result Pressure falls along each route, and branch resistances redistribute flow. Segment geometry visualizes topology, while the equations live at nodes.

OneSimulation connection HydraulicNode and HydraulicBranch are reduced system components. The 3D layout illustrates connectivity and supplies no 3D CFD claim.

API IN PRACTICE

Declare one inlet, one junction and the connecting resistance. Coordinates for the 3D drawing are separate from network state.

```
from onesim.physics.systems import HydraulicNode, HydraulicBranch

inlet = HydraulicNode("inlet", imposed_pressure=1.0)
junction = HydraulicNode("junction")
pipe = HydraulicBranch("supply", "inlet", "junction", resistance=1.0)
```

Repository cross-reference test_system_network_input_flexibility, test_lumped_solver_matrix.

Waves, living systems and reacting matter

The 1D, 2D and 3D plates illustrate representative problems. These are the additional model and service families to look for in each package.

Physics package	Model and service families
Acoustics	Linear propagation, damped ultrasound, pressure-driven oscillators, structural and vibroacoustic radiation.
Biophysics	Passive membranes, bioheat, axial tissue, compliant circulation, vessel exchange and organ clearance.
Chemistry	Species transport, reaction-network IR, kinetics and Jacobians, redox electrodes and reduced batteries.
Electromagnetics	Frequency-domain Maxwell, waveguide modes and ports, impedance boundaries, magnetic diffusion, series RLC, averaged buck converters and DC motors.

> **Check the execution path** A field declaration, a reduced model and a provider configuration serve different roles. Check the source API index and the selected model, carrier and numerical provider for a supported combination.

Potential, porous media, heat and flow

The 1D, 2D and 3D plates illustrate representative problems. These are the additional model and service families to look for in each package.

Physics package	Model and service families
Electrostatics	Poisson potential, charge sources, electric-field reconstruction, potential and electric-displacement boundaries.
Geotechnics	Groundwater and Darcy flux, Terzaghi and Biot consolidation, pore storage, rock and soil service families, Buckley–Leverett transport and Corey fractional flow.
Heat transfer	Steady and transient conduction, fixed temperature, flux and Robin conditions, radiation, convective transport and enthalpy-based phase change.
Hydrodynamics	Incompressible Navier–Stokes, Euler, well-balanced shallow water, phase transport, Darcy pressure, reacting mixtures, viscoelastic stress and axisymmetric flow.

> **Check the execution path** A field declaration, a reduced model and a provider configuration serve different roles. Check the source API index and the selected model, carrier and numerical provider for a supported combination.

Coupled fields and mechanical motion

The 1D, 2D and 3D plates illustrate representative problems. These are the additional model and service families to look for in each package.

Physics package	Model and service families
Magnetohydrodynamics	Transverse waves, ideal and nonideal terms, resistive and viscous effects, Hall and inductionless paths, two-fluid exchange and radiation–matter exchange.
Magnetostatics	Magnetic vector potential, Coulomb gauge, resolved solenoid current, potential and field traces, insulation and reluctance conditions.
Multibody dynamics	Linear joints, pendulum, driven four-bar, torsional drivetrain, Euler top, kinematics and rigid-body services.
Multiphysics	Thermoelastic transfer, thermal exchange, Joule heating, piston and fluid coupling, voice coil, magnetomechanical and reactive couplings.

> **Check the execution path** A field declaration, a reduced model and a provider configuration serve different roles. Check the source API index and the selected model, carrier and numerical provider for a supported combination.

Photons, plasma, solids and networks

The 1D, 2D and 3D plates illustrate representative problems. These are the additional model and service families to look for in each package.

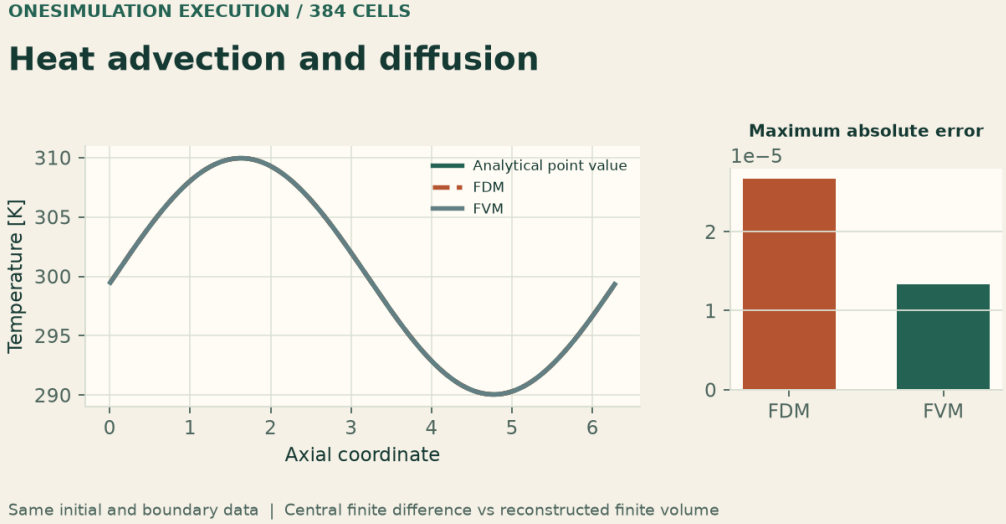
Physics package	Model and service families
Optics	Real and complex paraxial envelopes, sources and refractive inputs, intensity, photon density and photon flux.
Plasma	Charged particle and guiding center, cold-plasma oscillations and dielectric response, waterbags, drift–diffusion, electron energy, wall laws and ECR coupling.
Solid mechanics	Linear elasticity, thermal expansion, concentrated oscillation, Maxwell relaxation, Paris-law growth, Hertz impact and cohesive opening.
Systems and networks	Electrical, thermal, hydraulic and pneumatic networks, accumulators, gas chambers, stirred process volumes, controls and signal flow.

> **Check the execution path** A field declaration, a reduced model and a provider configuration serve different roles. Check the source API index and the selected model, carrier and numerical provider for a supported combination.

A temperature wave in a periodic line

A single heat model, two numerical representations.

ONESIMULATION EXECUTION / 384 CELLS



OneSimulation source execution. The plotted point reference differs slightly from an exact cell average.

The same HeatAdvectionDiffusion declaration, 384 uniform cells, periodic pair and RK4 steps of 0.00025 s are used in both runs. The finite-difference scheme selects central derivatives. The finite-volume scheme selects a two-point diffusive flux, monotone upwind and MUSCL reconstruction.

Method	Maximum absolute error at t = 0.1
FDM	2.67e-05
FVM	1.33e-05

> **Compare the right quantity** FDM point values are compared to the exact point values. FVM values are compared to exact cell averages. The maximum absolute error uses the native representation.

API IN PRACTICE

Use one heat problem and line mesh. Select the central or conservative spatial operator independently.

```
from onesim.numerics.fdm import FiniteDifference, CentralLaplacian
from onesim.numerics.fvm import FiniteVolume, MUSCL, MonotoneUpwind
from onesim.numerics.fvm import TwoPointDiffusion

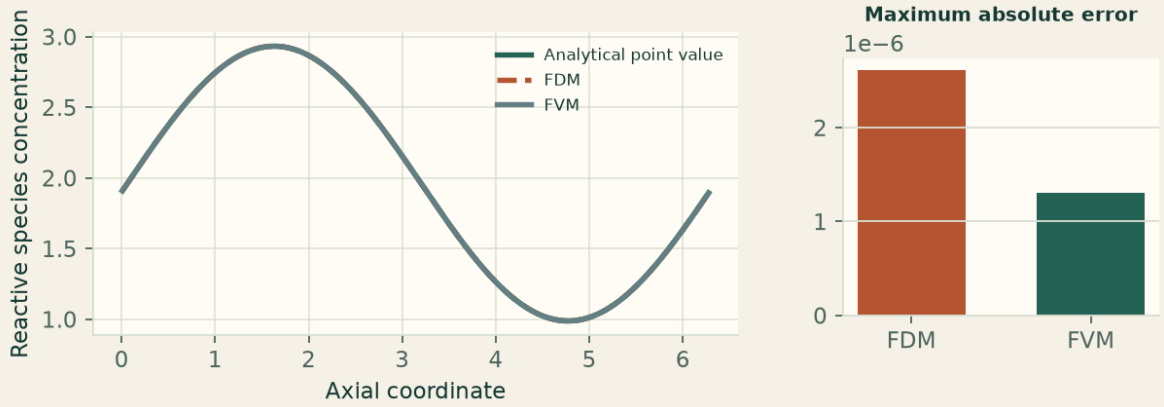
fd = FiniteDifference(diffusion=CentralLaplacian())
fv = FiniteVolume(diffusion=TwoPointDiffusion(), convection=MonotoneUpwind(),
                  reconstruction=MUSCL())
```

One tracer, one reacting species

Species storage, diffusion, advection and decay on the same mesh.

ONESIMULATION EXECUTION / 384 CELLS

Transport of two species



Same initial and boundary data | Central finite difference vs reconstructed finite volume

OneSimulation source execution. The plotted point reference differs slightly from an exact cell average.

SpeciesTransport carries both a passive tracer and a first-order decaying species. The plate displays the reactive field. Its volume-weighted integral matches the correct analytical integral within the exhibit tolerance.

Method	Maximum absolute error at t = 0.1
FDM	2.61e-06
FVM	1.31e-06

> **Compare the right quantity** FDM point values are compared to the exact point values. FVM values are compared to exact cell averages. The maximum absolute error uses the native representation.

API IN PRACTICE

Declare both species on one line. Each name gives a separate field to initialize, force or record.

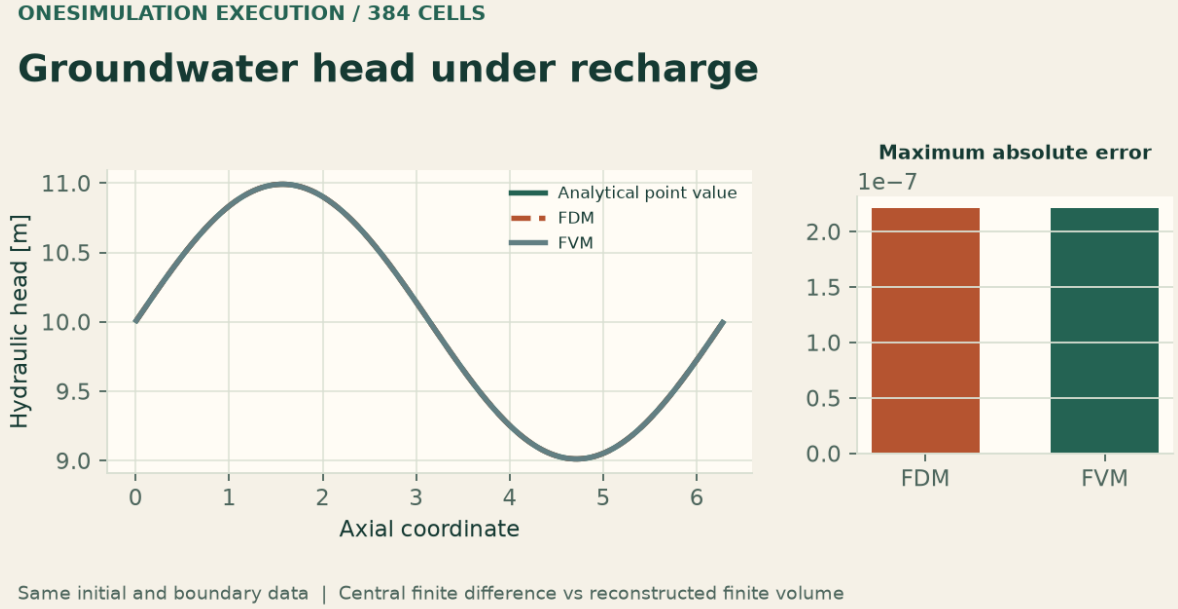
```
from onesim.physics.chemistry import SpeciesTransport, TransportSpecies

tracer = TransportSpecies("tracer", diffusivity=0.03)
reactive = TransportSpecies("reactive", diffusivity=0.08, decay_rate=0.2)
model = SpeciesTransport(domain=line.line, species=(tracer, reactive), velocity=0.6)
```

Hydraulic head with a recharge source

Check the forcing through the physical integral as well as the profile.

ONESIMULATION EXECUTION / 384 CELLS



OneSimulation source execution. The plotted point reference differs slightly from an exact cell average.

GroundwaterFlow uses periodic data, conductivity 0.02, storage 0.2 and recharge 0.004. A uniform mean rises under the recharge while the Fourier perturbation diffuses.

Method	Maximum absolute error at t = 0.1
FDM	2.21e-07
FVM	2.21e-07

> **Compare the right quantity** FDM point values are compared to the exact point values. FVM values are compared to exact cell averages. The maximum absolute error uses the native representation.

API IN PRACTICE

Pass recharge directly to the groundwater model and record both its head and flux.

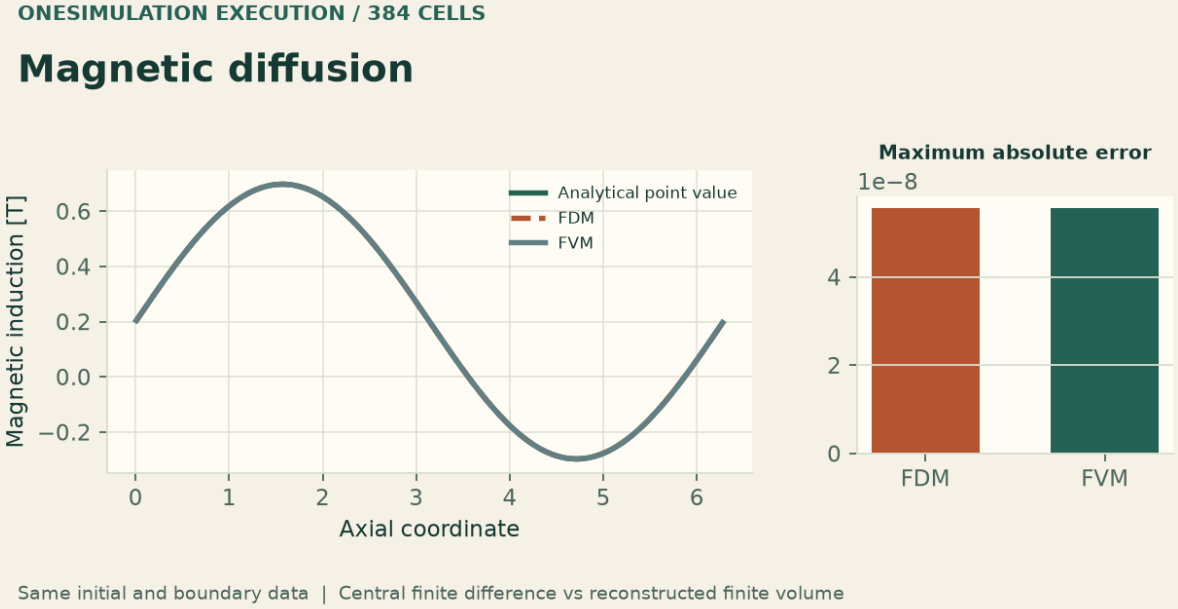
```
from onesim.physics.geotechnics import GroundwaterFlow

aquifer = GroundwaterFlow(domain=line.line, conductivity=0.02,
                          specific_storage=0.2, recharge=0.004)
outputs = (aquifer.head, aquifer.darcy_flux)
```

Magnetic diffusion with two spatial methods

Compare the same initial induction under pure diffusion.

ONESIMULATION EXECUTION / 384 CELLS



OneSimulation source execution. The plotted point reference differs slightly from an exact cell average.

MagneticDiffusion uses permeability 1 and conductivity 20 in this benchmark. The periodic induction perturbation decays around a fixed mean.

Method	Maximum absolute error at t = 0.1
FDM	5.55e-08
FVM	5.55e-08

> **Compare the right quantity** FDM point values are compared to the exact point values. FVM values are compared to exact cell averages. The maximum absolute error uses the native representation.

API IN PRACTICE

Pass the same material constants to the model in both runs. Only Numerics changes between the runs.

```
from onesim.physics.electromagnetics import MagneticDiffusion
from onesim.numerics import Numerics

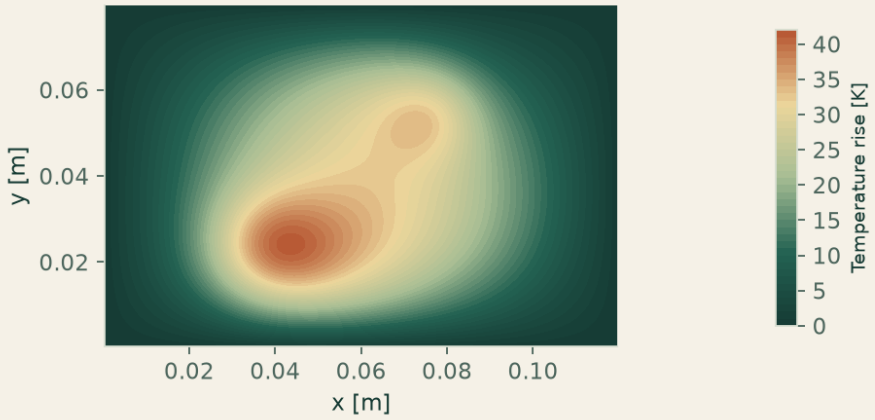
model = MagneticDiffusion(domain=line.line, permeability=1.0, conductivity=20.0)
fd_run = Numerics(spatial=fd, time_integrator=rk4)
fv_run = Numerics(spatial=fv, time_integrator=rk4)
```

Heat carried by a circulating flow

A one-layer 3D calculation produces a midplane map that reads as a 2D thermal field.

ONESIMULATION EXECUTION / CIRCULATING HEAT

A heated region carried by prescribed circulation



144 by 96 by 1 finite volumes | Temperature rise relative to the 293.15 K edge value

OneSimulation source execution. The map uses native midplane control-volume temperatures.

Geometry and boundary data The box uses a 144 by 96 by 1 structured mesh. Side boundaries are fixed at 293.15 K and the two thin faces have zero normal heat flux.

Read the field The sampled peak rise is 41.60 K. The footprint reflects both transport by the prescribed velocity and diffusion.

Scope This plate shows a thermal solve driven by prescribed flow. It does not claim a simultaneous Navier-Stokes solve for the circulation.

API IN PRACTICE

Supply the prescribed velocity and heating as callables. After solving, evaluate the temperature field at the mesh centers.

```
from onesim.physics.heat_transfer import ConvectiveHeatTransfer

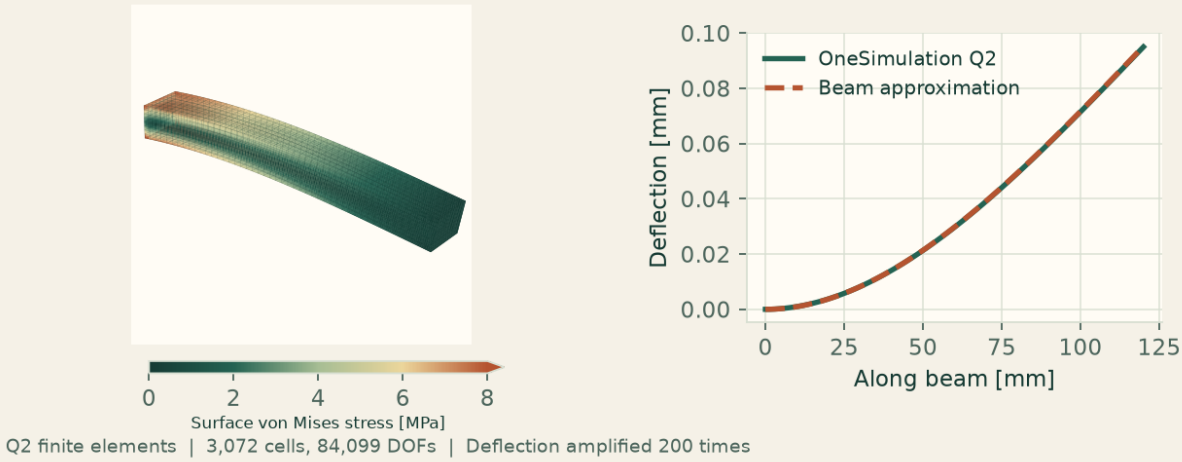
model = ConvectiveHeatTransfer(domain=plate.volume, material=solid,
                               velocity=velocity, heating=heating)
temperature = result.field(model.temperature).evaluate(centers)
```

A load, a mesh and a bending field

A 30 N distributed tip load bends a 120 by 18 by 12 mm cantilever.

ONESIMULATION EXECUTION / 3D ELASTICITY

Cantilever bending and surface stress



Recorded OneSimulation result. The color scale describes surface von Mises stress. The curve compares displacement along the centerline.

Run configuration LinearElasticity, Q2 hexahedral elements and a direct sparse solver. The left face is fixed. Constant illustrative material values are E = 70 GPa and Poisson ratio 0.30.

Quantity	Value
FEM tip deflection	0.095013 mm
Euler-Bernoulli tip deflection	0.095238 mm
Relative difference	0.24%

Interpretation The beam formula is a slender-beam approximation. A sampled stress near a clamped edge is mesh-sensitive.

API IN PRACTICE

Get displacement and von Mises stress from the same result. Query each at its own world-coordinate probe points.

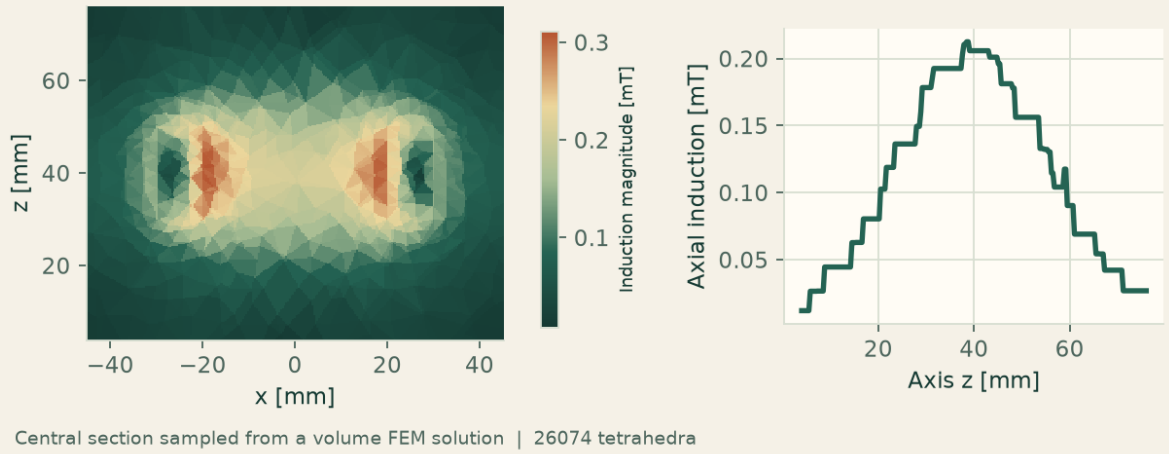
```
u = result.field(elastic.displacement)
stress = result.field(elastic.von_mises_stress)
tip_deflection = u.evaluate([[0.12, 0.009, 0.006]])[0, 2]
surface_stress = stress.evaluate(surface_points)
```

Where the induction field concentrates

A resolved solenoid current drives a vector-potential solve on a surrounding tetrahedral volume.

ONESIMULATION EXECUTION / 3D MAGNETOSTATICS

The induction field of a resolved coil



Central section sampled from a volume FEM solution | 26074 tetrahedra

Recorded OneSimulation result. Field magnitude on the central plane and axial induction along the coil axis.

Run configuration Magnetostatics uses edge-space potential and a Coulomb-gauge formulation. The plotted induction is reconstructed from the potential.

Read the field The volume mesh has 26074 tetrahedra. The maximum magnitude among plotted plane samples is 0.3110 mT.

Compare with care The independent thin-loop and regularized loop-stack references in the atlas illustrate spatial trends. A finite-volume source and a bounded FEM domain have different near-field behavior.

API IN PRACTICE

Use an edge element for the potential, then sample the reconstructed induction. section_points contains points on the center plane.

```
from onesim.numerics.fem import FiniteElement, Nedelec

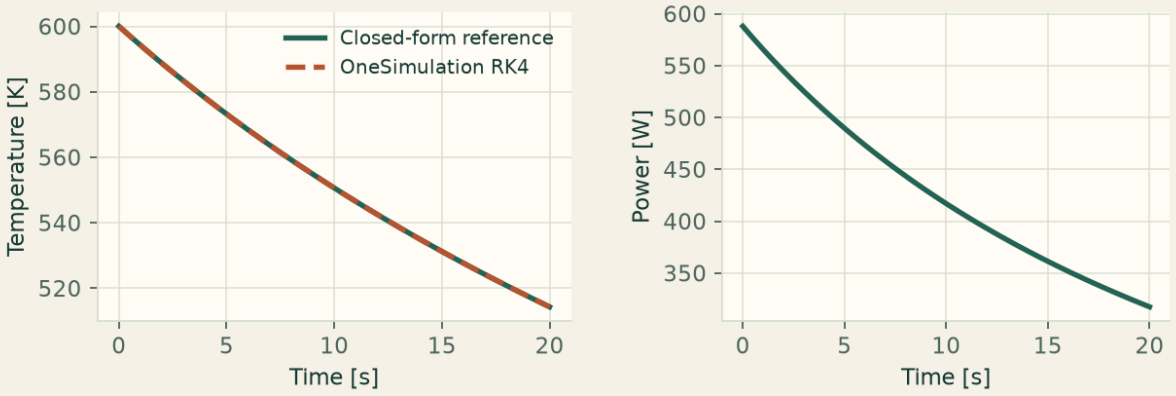
space = FiniteElement(element=Nedelec(order=1))
B = result.field(coil_model.magnetic_flux_density)
section = B.evaluate(section_points)
```

A thermal mass cools by radiation

One of the five delivery notebooks uses a concentrated thermal state to turn a nonlinear boundary law into a testable transient.

ONESIMULATION EXECUTION / THERMAL RADIATION

Temperature and emitted power fall together



Concentrated thermal model | Idealized 0 K surroundings | RK4 step 0.05 s

OneSimulation source execution. Temperature and emitted power are recovered from the same time history.

Compare with the closed form The 600 K initial state cools to 514.292 K in 20 s. The maximum temperature difference from the analytical reference is 7.96e-13 K for these parameters.

API IN PRACTICE

Specify emissivity, surroundings, heat capacity and area as separate inputs. The analytical check uses idealized 0 K surroundings.

```
from onesim.physics.heat_transfer import RadiationCooling
from onesim.physics.heat_transfer import ThermalGreyDiffuseRadiationBoundary

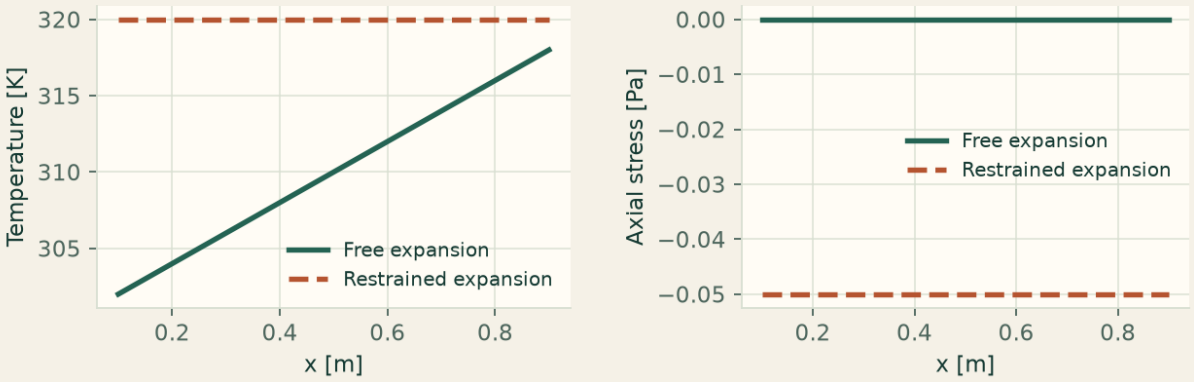
wall = ThermalGreyDiffuseRadiationBoundary(
    condition_id="radiation", emissivity=0.8, ambient_temperature=0.0)
model = RadiationCooling(domain=body.system, heat_capacity=100,
                         radiating_area=0.1, radiation=wall)
```

Temperature becomes a mechanical load

A P1 conduction field is evaluated in a P2 elasticity calculation. The two fields keep their own discretizations.

ONESIMULATION EXECUTION / ONE-WAY THERMAL-TO-STRUCTURAL COUPLING

A temperature field drives elastic stress



416 tetrahedra | P1 thermal field transferred to P2 elasticity

OneSimulation source execution. Two manufactured cases contrast free expansion and full restraint.

What the constraint changes Free expansion yields negligible stress along the probe. With all faces restrained and a uniform 20 K rise, the axial stress is −0.05 Pa for the illustrative material used in this verification.

API IN PRACTICE

Connect the solved temperature to the elasticity model's input. Provide the per-model spaces through the Simulation declaration.

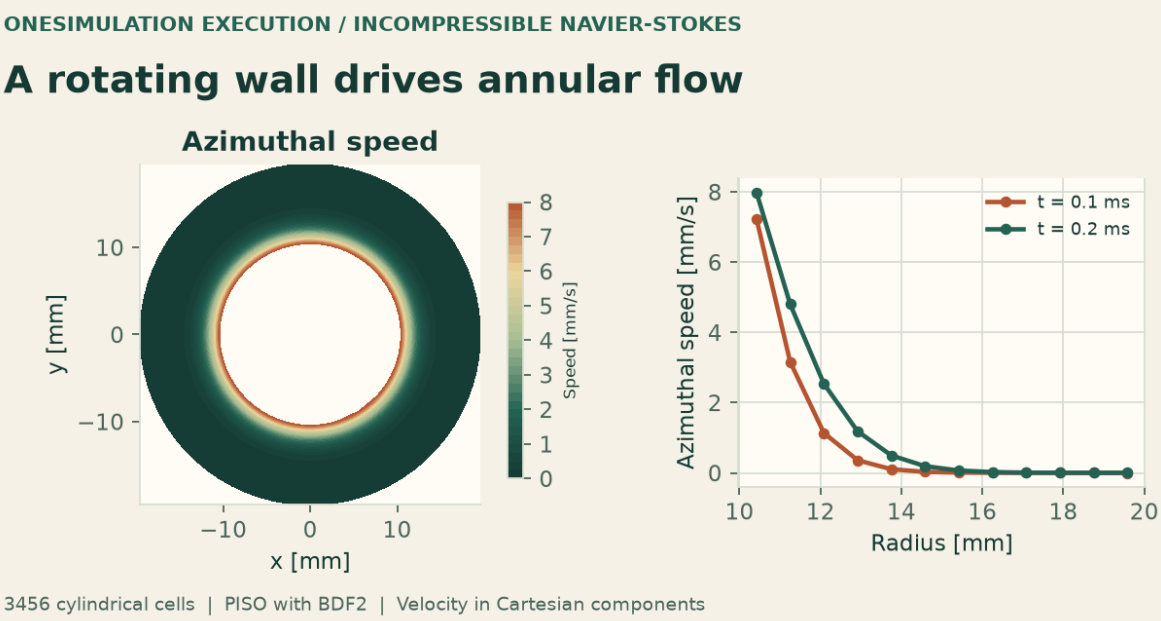
```
from onesim.numerics.field_transfer import FiniteElementEvaluation
from onesim.simulation.field_connections import FieldCoupling

thermal_load = FieldCoupling(source=heat.temperature,
    target=solid.temperature, transfer=FiniteElementEvaluation())
```

A rotating wall drives a 3D flow

A cylindrical finite-volume mesh and PISO pressure correction evolve Cartesian momentum in an annulus with periodic axial ends.

ONESIMULATION EXECUTION / INCOMPRESSIBLE NAVIER-STOKES



OneSimulation source execution. The view shows azimuthal velocity recovered from the Cartesian field at cell centers.

Gauge and solver The inner wall supplies rotation and the outer wall is stationary. The 3456-cell case uses BDF2 and retains its volume-mean pressure gauge at 2.0 Pa.

API IN PRACTICE

Select the pressure and momentum solvers separately. Put the coupling into Numerics and the mean-value gauge into Problem.

```
from onesim.numerics.coupled_systems import PISO
from onesim.simulation.field_constraints import MeanValue

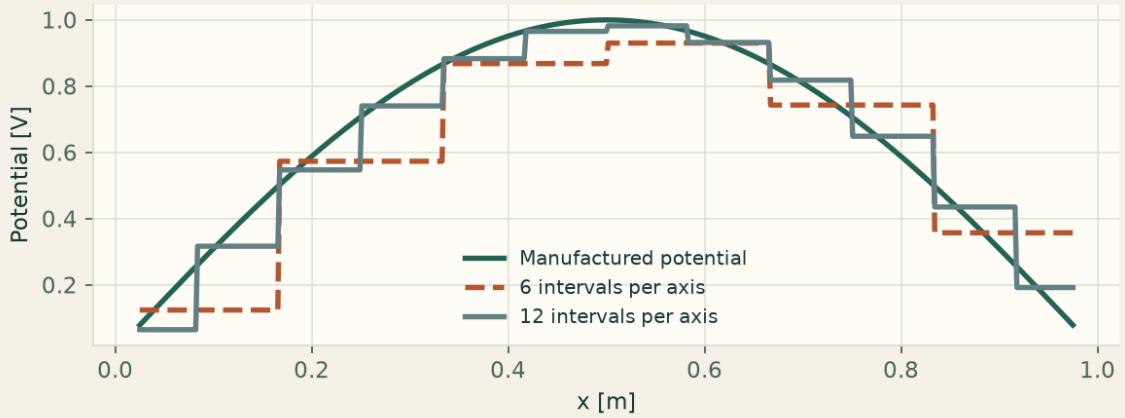
coupling = PISO(momentum_solver=momentum, pressure_solver=pressure,
                pressure_correctors=2)
gauge = MeanValue(flow.pressure, 2.0)
```

Ground a box and solve its volume charge

The mixed electrostatic solve recovers a potential and a compatible normal-flux field with a cellwise charge balance.

ONESIMULATION EXECUTION / MIXED ELECTROSTATICS

Potential converges while each cell balances charge



Grounded unit box | Mixed normal-flux space | Field sampled on the central line

OneSimulation source execution. Two mesh resolutions are compared with the same manufactured potential.

> **Compare on native supports** RMS errors at the respective native coordinates were 0.00793 V and 0.00201 V. The mixed divergence check passed on both meshes.

API IN PRACTICE

Supply an explicit permittivity property and charge source, then ground all faces. Select the mixed finite-element space in Numerics.

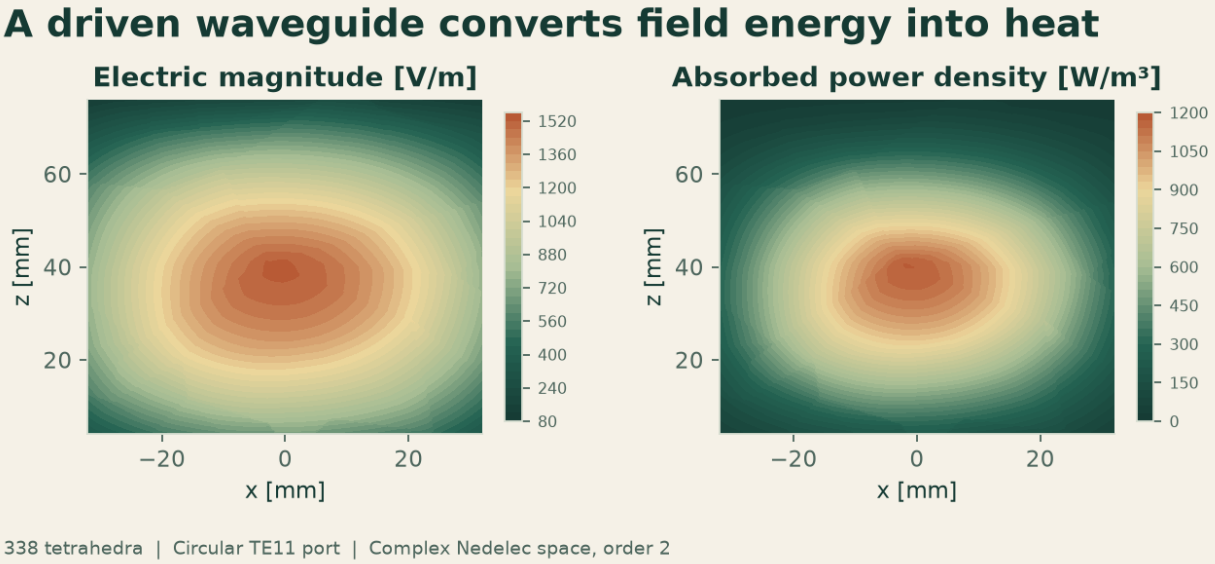
```
from onesim.physics.electrostatics import PoissonEquation, FixedPotential

model = PoissonEquation(domain=box.volume, permittivity=epsilon,
                        charge_density=charge)
wall = FixedPotential(physics=model, boundary=box.boundaries.all, value=0)
```

A waveguide field carries power and heat

A circular TE11 port drives a complex electric field. The same solve exposes magnetic flux and absorbed power density.

ONESIMULATION EXECUTION / FREQUENCY-DOMAIN MAXWELL



OneSimulation source execution. A physical cut samples electric magnitude and dissipated power density on the center plane.

> **Check port power** The incident 2.0 W splits into 1.8139 W reflected and 0.1861 W absorbed for this conducting test medium.

API IN PRACTICE

Bind the incident mode to the guide's bottom face. Select an edge space for the complex electromagnetic field.

```
from onesim.physics.electromagnetics import CircularWaveguideMode, WaveguidePort
from onesim.numerics.fem import FiniteElement, Nedelec

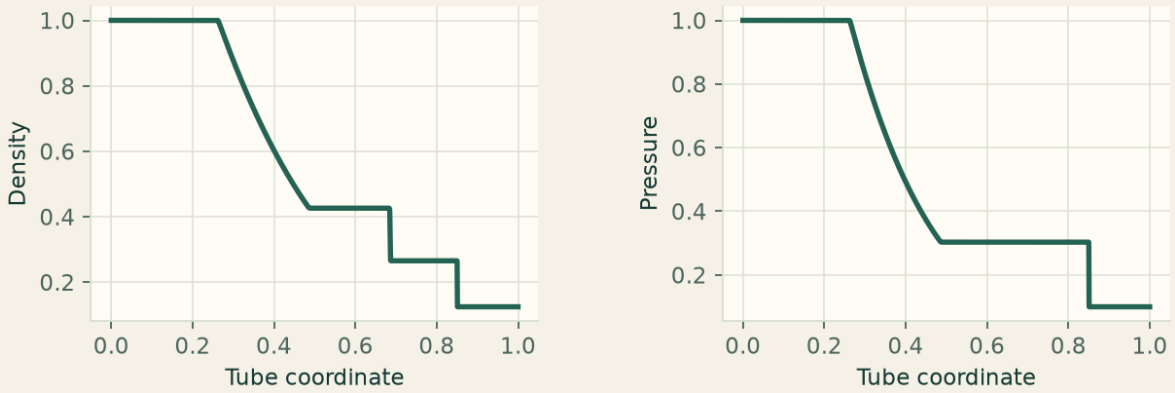
port = WaveguidePort(boundary=guide.boundaries.bottom, physics=maxwell,
                     mode=CircularWaveguideMode("TE11"), incident_power=2)
space = FiniteElement(element=Nedelec(order=2))
```

Resolve the Sod shock tube

A conservative Euler balance resolves the wave pattern with WENO5-JS flux reconstruction and classical RK4.

RECORDED ONESIMULATION EXECUTION / COMPRESSIBLE EULER

A rarefaction, a contact and a shock



6,400 points | WENO5-JS with global Lax-Friedrichs splitting | Classical RK4

OneSimulation WENO5 and RK4 at t = 0.2. The 6,400-point run took 16,000 steps.

> **Check physical balances** Density and pressure stayed positive at all 21 sampled states. Absolute balance errors for mass and energy were 2.22e-16 and 0 in the recorded normalization.

API IN PRACTICE

Select the flux difference and split explicitly for the Euler model on its line mesh.

```
from onesim.numerics.fdm import FiniteDifference, WENOFluxDifference
from onesim.numerics.fdm import GlobalLaxFriedrichsSplit, WENO5Flux

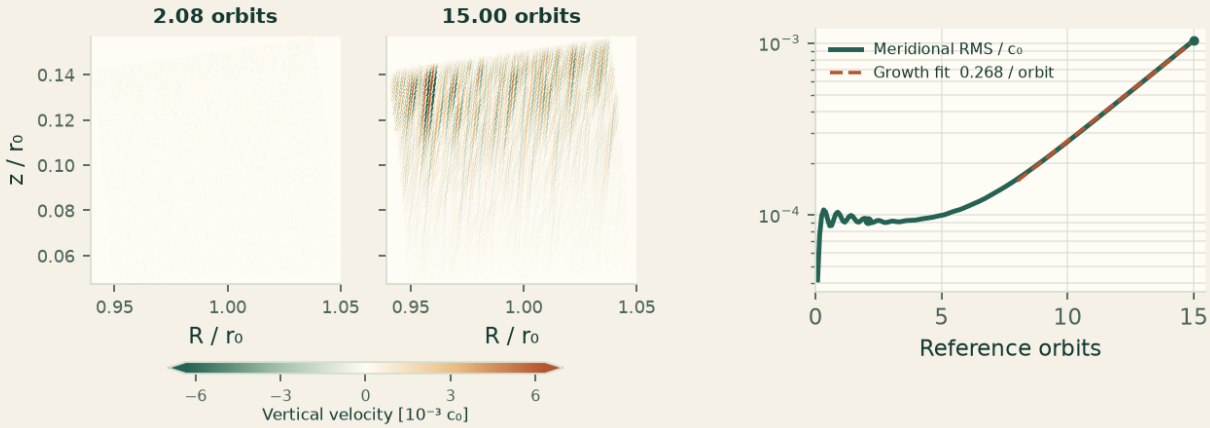
space = FiniteDifference(gradient=WENOFluxDifference(
    flux_splitting=GlobalLaxFriedrichsSplit(), reconstruction=WENO5Flux()))
```

Vertical shear instability in a rotating disc

A seeded angular-momentum perturbation develops into inclined bands of vertical motion in a locally isothermal, axisymmetric disc.

ONESIMULATION EXECUTION / ROTATING HYDRODYNAMICS

Vertical shear drives growing flow bands



OneSimulation 256 × 256 run. Both meridional sections use the same vertical-velocity scale. The right panel shows the mass-weighted meridional speed from the accepted states.

Measure the growth From orbit 8 to 15.00, the RMS amplitude grows 6.4×. A fit to log amplitude gives 0.268 per orbit with $R^2 = 0.9998$. The interior vertical RMS grows 16.7× between the two plotted states.

Model and scope The run uses staggered meridional momentum, QUICK and BDF2 with a 0.5 code-time step cap. The exhibit covers the growth phase in an axisymmetric model. Nonlinear saturation would require a longer run.

API IN PRACTICE

Compose the physical, domain and spatial declarations with the axisymmetric interface. Then obtain the background fields for the initial state.

```
from onesim.physics.hydrodynamics.shared.geometry.execution.axisymmetric import (
    AxisymmetricFlow,
)

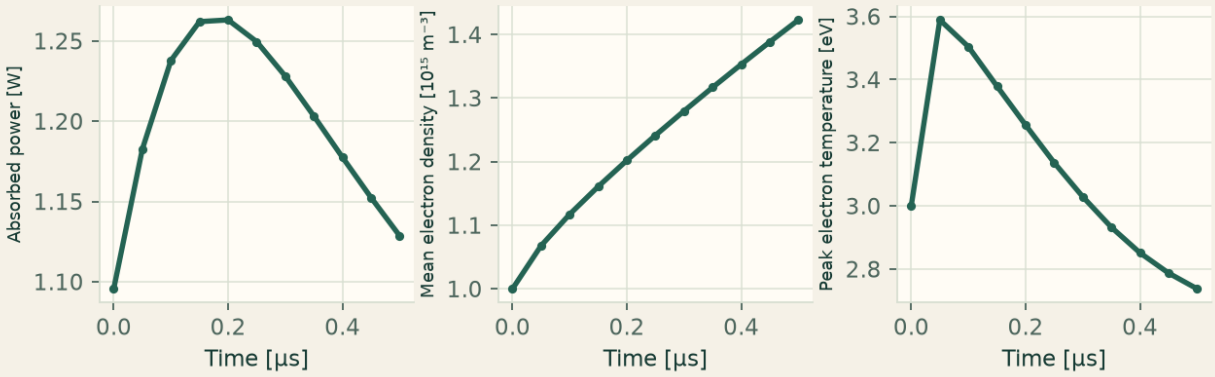
flow = AxisymmetricFlow(physical, domain, spatial, name="Rotating disc")
background = flow.background_fields(native=True)
```

Follow microwave absorption in an ECR model

A rectangular waveguide launches the TE10 mode into a metal chamber. The evolving plasma dielectric determines how much power is absorbed.

RECORDED ONESIMULATION EXECUTION / ECR PLASMA

Microwave absorption feeds plasma evolution



24,576 tetrahedra | 10 accepted steps | Powered TE_{10} port

Accepted states through 0.5 μs. The port flux and volume absorption balance to 2.71e-10 W.

> **Metal walls and feed window** The 80 × 40 mm aperture launches 5 W at 2.45 GHz. The grounded metal uses perfect electric conductor conditions for Maxwell and effective sheath losses for particles. An ideal insulating window seals the aperture. The relative charge-balance defect is 9.49e-13.

API IN PRACTICE

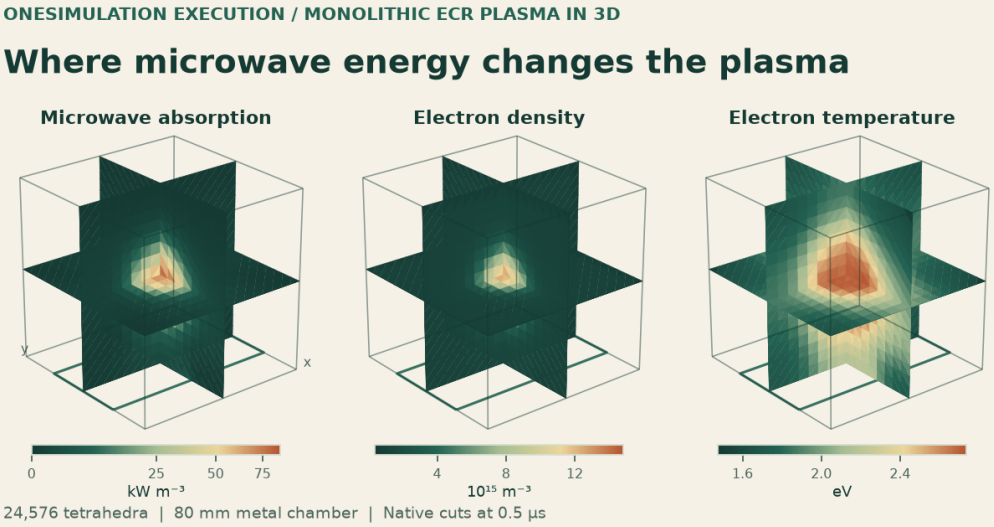
Bind a powered TE10 mode to the mesh faces of the axial aperture.

```
from onesim.physics.electromagnetics.waves.models import high_frequency_em

port = high_frequency_em.RectangularWaveguidePort(
    faces=port_faces, origin=(0., 0.02, 0.), normal=(0., 0., -1.),
    width_direction=(1., 0., 0.), width=0.08, height=0.04, power=5.0,
)
```

A coupled plasma discharge in three dimensions

The 2.45 GHz Maxwell field, electrostatic potential, species densities and electron energy are solved together at each time step.



Three central sections show native cell values at 0.5 µs. The green outline marks the axial waveguide aperture.

Volume diagnostic	Computed value
Mesh and real unknowns	24,576 tetrahedra and 184,928 unknowns
Grid intervals per axis	16 × 16 × 16 at 5 mm spacing
Integrated microwave absorption	1.129 W
Peak electron density	14.84 × 10^15 m^-3 · 33.75 mm from wall
Peak electron temperature	2.74 eV · 28.75 mm from wall

Interior resonance An axial mirror and transverse quadrupole enclose the 87.52 mT resonance in the chamber. The seeded transient uses an illustrative collisional gas at 62.1 Pa.

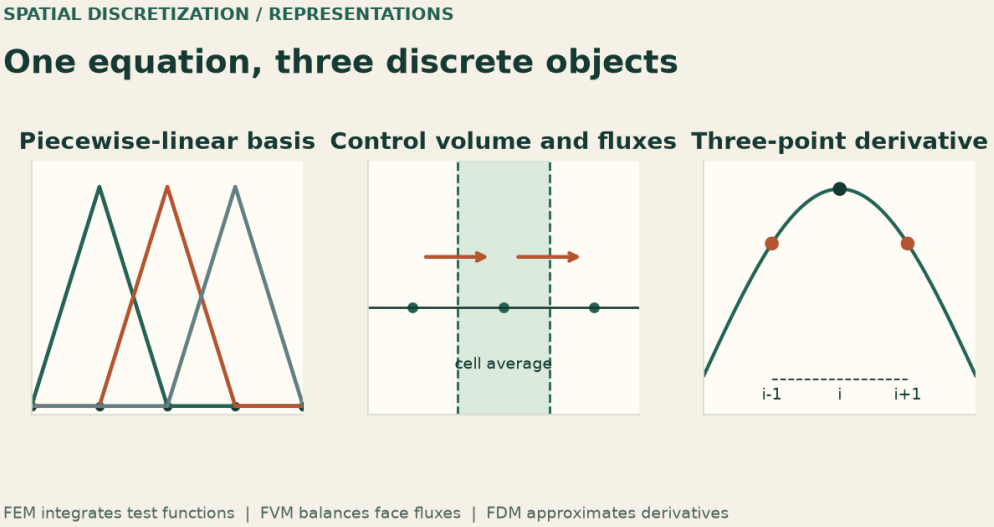
API IN PRACTICE

Pass the port and metal edge constraints to the native Maxwell coupling. Use the first-order edge-element provider for maxwell.

```
from onesim.physics.plasma.field_coupling.binding.coupling import (
    frequency_transient as ft,
)
microwave = ft.FrequencyTransientCoupling(
    frequency=2.45e9, power=port.power, port=port,
    pec_edge_ids=metal_edge_ids, maxwell_api=maxwell, fallback="raise",
)
```

Three ways to turn a field into algebra

A change of spatial method changes what the unknowns represent and how the equations couple them.



Schematic illustration. A nodal basis, a conservative volume and a local stencil each encode a different discrete object.

Family	Discrete state	Key check
FEM	Coefficients of basis functions	Weak residual, trace constraints, reconstruction
FVM	Cell averages or conserved cell integrals	Cancellation of shared face fluxes
FDM	Point or staggered samples	Stencil consistency and boundary closure

> **Implementation** Select the actual spatial declaration in Numerics and check what the physical model and mesh can bind. FEM, FVM and FDM can solve related equations while using different outputs, constraints and stability conditions.

API IN PRACTICE

The declarations expose the choice of element, flux or stencil in the study configuration.

```
from onesim.numerics.fem import FiniteElement, Lagrange
from onesim.numerics.fvm import FiniteVolume, TwoPointDiffusion
from onesim.numerics.fdm import FiniteDifference, CentralLaplacian

fem = FiniteElement(element=Lagrange(order=1))
fvm = FiniteVolume(diffusion=TwoPointDiffusion())
fdm = FiniteDifference(diffusion=CentralLaplacian())
```

Finite elements integrate a weak form

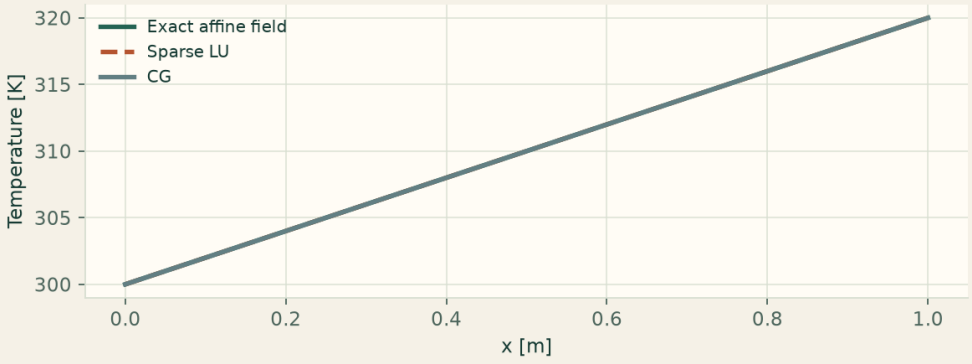
Approximate the field with basis functions and require the residual to vanish against test functions.

$$\int_\Omega k\nabla u_h \cdot \nabla v_h \, dx = \int_\Omega f v_h \, dx + \int_{\Gamma_N} g_N v_h \, ds$$

In the scalar diffusion form above, g_N is the prescribed conormal k times the outward derivative. Each element contributes a local stiffness matrix and load. Global assembly shares continuous degrees of freedom across neighboring elements, with boundary terms or constraints chosen for the problem.

ONESIMULATION EXECUTION / P1 FINITE ELEMENTS

A linear temperature field in a tetrahedral cube



Centerline x at y = z = 0.5 m | 2123 tetrahedra | Affine Dirichlet boundary on all faces

OneSimulation source execution. A P1 finite-element temperature field on a tetrahedral cube recovers the affine reference with two solver selections.

Choose the space Lagrange for continuous scalar or vector fields, DiscontinuousLagrange for discontinuous Galerkin paths, Nedelec for curl-conforming edge fields, and TaylorHood for selected mixed velocity-pressure formulations.

Assembly and stabilization Quadrature, mass treatment, boundary enforcement and stabilization affect the discrete problem. Source exports include SUPG, GLS, CIP, symmetric interior penalty, mixed constraints and HcurlNitsche.

API IN PRACTICE

Select the basis order in the field space, then combine it with a direct linear solver for the stationary heat study.

```
from onesim.numerics import Numerics
from onesim.numerics.fem import FiniteElement, Lagrange
from onesim.numerics.linear import SparseDirect

space = FiniteElement(element=Lagrange(order=1))
numerics = Numerics(spatial=space, linear_solver=SparseDirect())
```

Finite volumes balance each cell

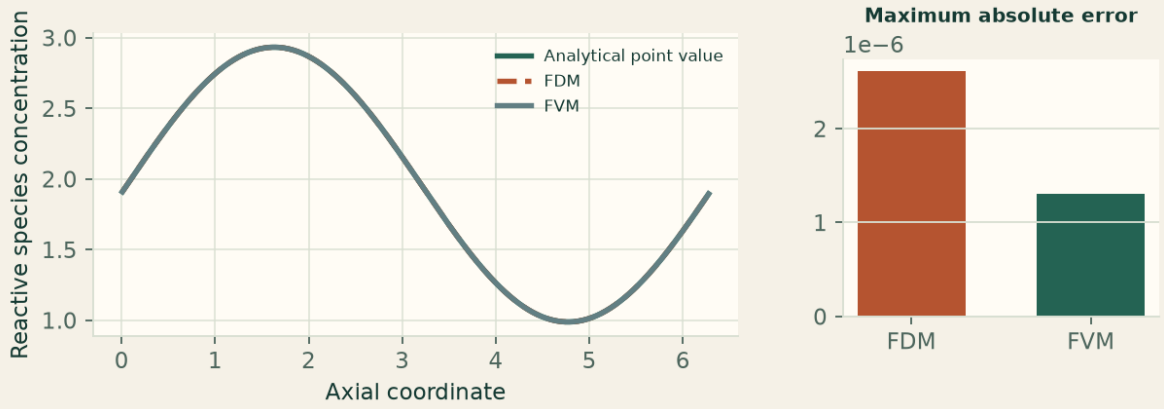
Integrate the conservation law over a cell and exchange flux through its faces.

$$\frac{d}{dt}(|K|\bar{U}_K) = -\sum_{f \subset \partial K} |f|\hat{F}_f + \int_K S\,dx$$

A single numerical flux appears with opposite sign in its adjacent cells, so internal fluxes cancel in a global balance. The reconstruction supplies interface states and a limiter can control oscillations around sharp fronts.

ONESIMULATION EXECUTION / 384 CELLS

Transport of two species



Same initial and boundary data | Central finite difference vs reconstructed finite volume

OneSimulation source execution with FiniteVolume and FiniteDifference selected on the same periodic species problem.

Public components FiniteVolume, MonotoneUpwind, LinearUpwind, Rusanov, TwoPointDiffusion, MUSCL and limiters. ScharfetterGummel treats drift-diffusion fluxes. RhieChow is a coupling option for selected collocated flow paths.

What to verify Use cell volumes and face normals from the actual mesh. Check integral conservation, boundary flux signs, admissible states and the right reconstruction of point fields.

API IN PRACTICE

Select the interface flux, reconstruction and limiter as separate controls for the conservative transport problem.

```
from onesim.numerics.fvm import FiniteVolume, MUSCL, MinmodLimiter, Rusanov

space = FiniteVolume(convection=Rusanov(), reconstruction=MUSCL(),
                     limiter=MinmodLimiter())
```

Finite differences act on sampled fields

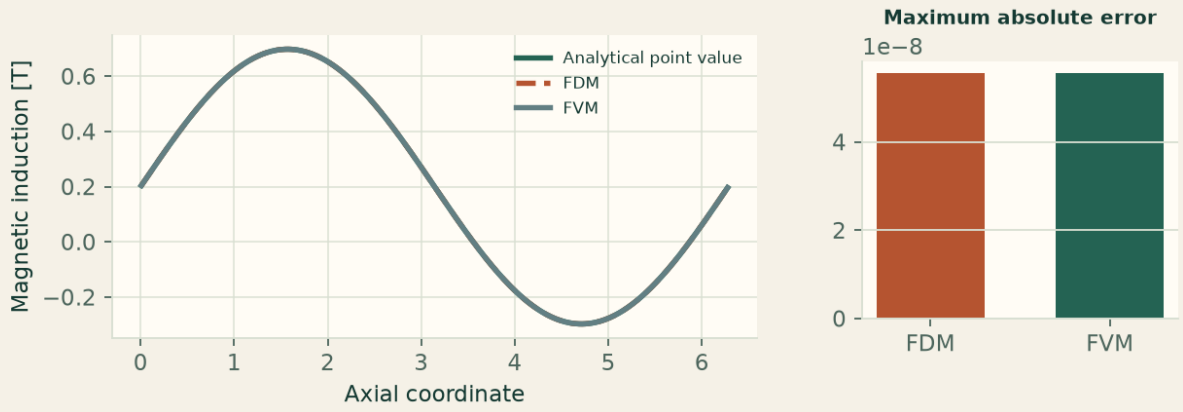
A stencil combines neighboring values to approximate a derivative at a point.

$$u''(x_i) \approx (u_{i-1} - 2u_i + u_{i+1})/h^2$$

On a uniform grid this centered second derivative is second-order accurate for a sufficiently smooth field. Near boundaries, one-sided or boundary-flux operators enforce the requested traces. Nonuniform grids require coefficients derived for the actual spacing.

ONESIMULATION EXECUTION / 384 CELLS

Magnetic diffusion



Same initial and boundary data | Central finite difference vs reconstructed finite volume

OneSimulation source execution. Central finite differences and finite volumes solve the same periodic magnetic diffusion model.

Public components FiniteDifference, CentralDifference, CentralLaplacian, PolynomialDifference, PolynomialLaplacian, SummationByPartsDifference, BoundaryFluxDifference and conservative WENO flux-difference declarations.

What to verify Confirm whether values live at points, centers or staggered locations. A stencil's interior order does not settle the accuracy of its boundary closure.

API IN PRACTICE

Make the Laplacian explicit on a scalar diffusion study. Use the model binder to check that the mesh can supply the needed stencil.

```
from onesim.numerics import Numerics
from onesim.numerics.fdm import CentralLaplacian, FiniteDifference

numerics = Numerics(spatial=FiniteDifference(diffusion=CentralLaplacian()),
                    time_integrator=rk4)
```

DISCRETIZATION / SAME-PROBLEM EXPERIMENT

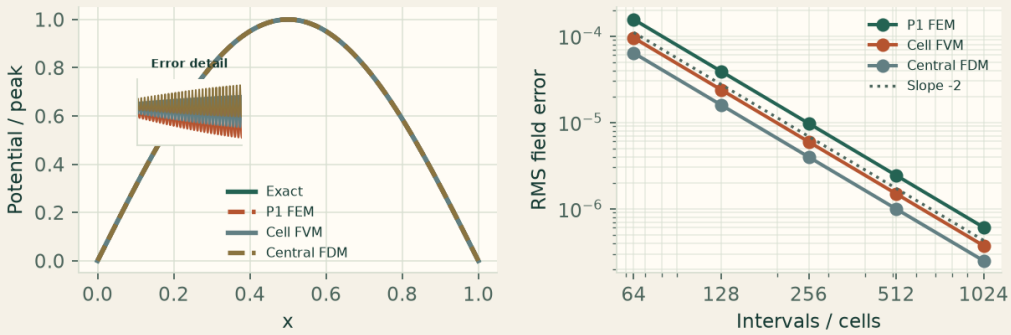
Solve the same Poisson equation three ways

Identical forcing and boundary values make the representation and convergence differences visible.

$$-u'' = \pi^2 \sin(\pi x), \quad u(0) = u(1) = 0, \quad u(x) = \sin(\pi x)$$

INDEPENDENT NUMERICAL BENCHMARK / POISSON EQUATION

Convergence under mesh refinement



Same source and boundary values | RMS error of piecewise-linear reconstruction on a common grid

Independent benchmark. Continuous P1 FEM with quadrature, cell-centered FVM with two-point fluxes, and three-point central FDM.

Intervals	P1 FEM RMS	Cell FVM RMS	Central FDM RMS
64	0.000156	9.53e-05	6.35e-05
128	3.89e-05	2.38e-05	1.59e-05
256	9.72e-06	5.95e-06	3.97e-06
512	2.43e-06	1.49e-06	9.93e-07
1024	6.08e-07	3.72e-07	2.49e-07

Fair comparison The table uses piecewise-linear reconstruction onto a common grid. The FVM balance integrates its source over each cell. FEM integrates the source against basis functions. Error constants are specific to this smooth problem and output reconstruction.

API IN PRACTICE

Give the model the shared source and permittivity property before selecting a method. The plot uses independent assemblies.

```
from onesim.physics.electrostatics import PoissonEquation
model = PoissonEquation(domain=line.line, permittivity=epsilon,
                        charge_density=source)
```

Higher-order and coupled formulations

The source separates a physical balance from its numerical realization so admissible replacements can be tested explicitly.

Path	What changes	Example check
Discontinuous Galerkin	Element-local polynomial state and interface terms	Pulse transport with limiter and numerical flux
Mixed flow and elasticity	Different field spaces and block constraints	Pressure gauge or volume storage under load
Curl and cochain methods	Edge or face topology and metric operators	Discrete curl, Stokes relation and field energy
Conservative high-resolution FD	A numerical flux difference with reconstruction	Shock or advected sharp-front behavior
Moving meshes	Volume, boundary motion and state transfer	Geometric conservation and history reconstruction

Code entry points Look in numerics.fem, numerics.fvm, numerics.fdm, numerics.field_transfer, numerics.coupled_systems and the meshing adaptation APIs. Select the provider and check the study's reported execution.

Useful diagnostics Compare spatial refinement, algebraic residual, conservation, boundary data, admissibility, point versus volume output and temporal error separately.

API IN PRACTICE

For a compatible transport model, choose a discontinuous polynomial field and a numerical face flux in one declaration.

```
from onesim.numerics.fem import FiniteElement, DiscontinuousLagrange
from onesim.numerics.fvm import Rusanov

space = FiniteElement(element=DiscontinuousLagrange(order=2),
                      convection=Rusanov())
```

Topology, fields and metric are separate choices

The repository includes mixed finite elements, finite-integration and cochain paths in addition to the classical spatial families.

Spatial surface	What the unknowns preserve	Executable check
Mixed normal-flux FEM	Normal flux and scalar balance use different spaces.	Mixed diffusion and polynomial flux reconstruction.
Edge-based FEM	Tangential traces and discrete curl carry electromagnetic fields.	Nedelec waveguide power, impedance conditions and reconstruction.
Finite integration and cochains	Incidence and orientation carry the topology. A Hodge operator supplies the material and metric.	Discrete Stokes, selected Hodge operators and Fresnel reflection.
Conservative transport	Face fluxes, invariant-domain controls and limiters address advection and discontinuities.	Sod, bounded phase transport and well-balanced shallow water.
ALE and remeshing	Moving geometry carries physical measures. A selected transfer maps field state and supported histories.	Moving MAC fields, curved volume projection and remeshing output.

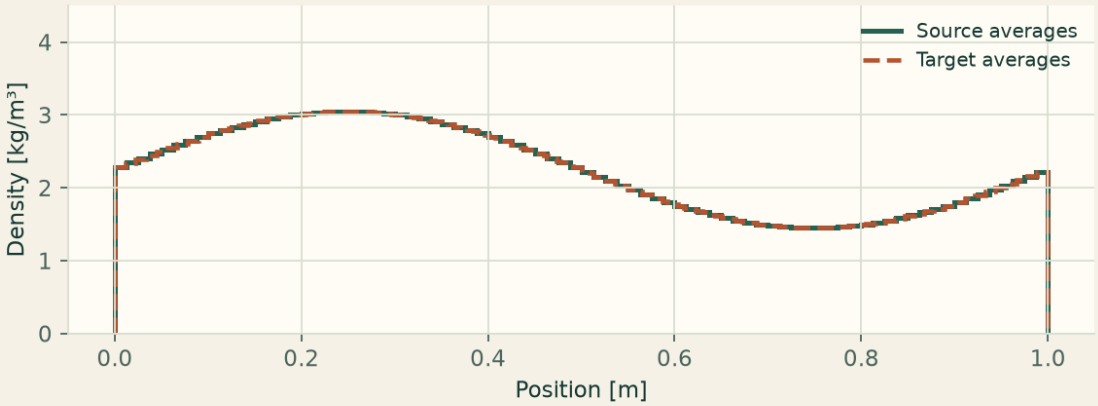
> **Controls remain explicit** Choose quadrature, boundary enforcement, initial projection and stabilization with the selected field space. Public declarations cover continuous and discontinuous bases, flux spaces and curl-conforming spaces. Compatibility follows the model and carrier.

Change cell partitions without losing the integral

Conservative remapping moves cell averages between different supports. Field evaluation and conservative projection address different physical goals.

ONESIMULATION EXECUTION / CONSERVATIVE FIELD TRANSFER

Remap cell averages while preserving the integral



Native cell averages are shown as steps | Source and target have different cell boundaries

OneSimulation source execution. The width-weighted source and target integrals agree.

Measure the transfer The source has 80 cells and the target has 128. Both width-weighted integrals are 2.25 kg/m² for the unit-area line example.

API IN PRACTICE

Declare the transfer on the two edge arrays, then apply it to the actual source averages.

```
from onesim.numerics.field_transfer import operators

mapping = operators.conservative_remap(
    "density", "source", "target", source_edges=old_edges,
    target_edges=new_edges, source_unit="kg/m^3", target_unit="kg/m^3")
target = operators.apply(mapping, source_averages)
```

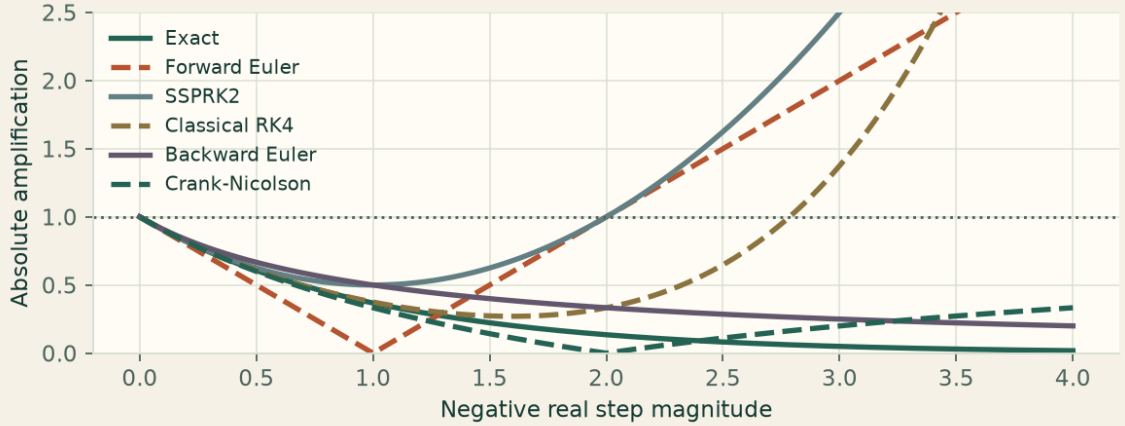
Explicit and implicit steps

After spatial discretization, a transient problem evolves a finite-dimensional state. Some models also carry algebraic constraints.

$$M\dot{y} = F(t, y), \qquad R(t, y, \dot{y}) = 0$$

TIME INTEGRATION / EXACT SCALAR TEST

Amplification explains a step-size limit



Test equation y' = lambda y | Horizontal coordinate is -lambda times step for a negative real lambda

Independent scalar test. An amplification factor with magnitude above one grows a decaying exact mode.

Explicit Evaluate known stages. Cost can be low per stage, while wave speeds, diffusion or reaction eigenvalues constrain the usable step.

Implicit Solve for a stage or new state. A stable method can use larger steps on stiff modes, while phase, accuracy and nonlinear convergence still matter.

Split or constrained systems IMEX treats designated terms differently. DAE and monolithic routes require the residual, constraints and compatible stage solves.

API IN PRACTICE

Change the time declaration in the study rather than embedding the integrator in the physics. An implicit path also needs a compatible stage solve.

```
from onesim.numerics.time import ForwardEuler, BackwardEuler

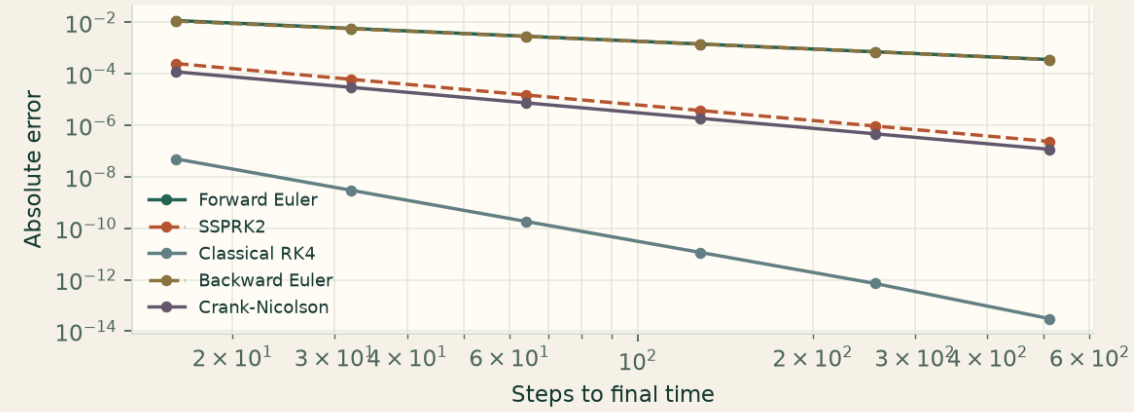
explicit_step = ForwardEuler()
implicit_step = BackwardEuler()
```

Temporal refinement and phase error

An integrator's formal order is visible only when temporal error dominates and the solution is smooth.

TIME INTEGRATION / TEMPORAL REFINEMENT

Accuracy depends on the selected method and step



Independent fixed-step calculation | Absolute error at time 1 for y' = -y, y(0) = 1

Independent scalar refinement. Error is measured at the same final time for each method.

Different questions Stability asks whether an error grows. Convergence asks whether the discrete trajectory approaches the exact one. Phase and damping matter even for stable solutions.

In OneSimulation ForwardEuler, ClassicalRungeKutta4, BackwardEuler and BDF are typed declarations. TimeMethod selects named generic routes such as ssprk2 and implicit_rk_midpoint. AdaptiveTimeStep configures step control rather than supplying a new integration formula.

Practical refinement Hold the spatial representation fixed first. Run at successively smaller steps. For multistep methods, account for startup accuracy and checkpoint history.

API IN PRACTICE

Build one simulation with a selected integrator. Repeat the construction for a different method and compare at the same final time.

```
from onesim.numerics import Numerics
from onesim.numerics.time import ClassicalRungeKutta4, TimeMethod

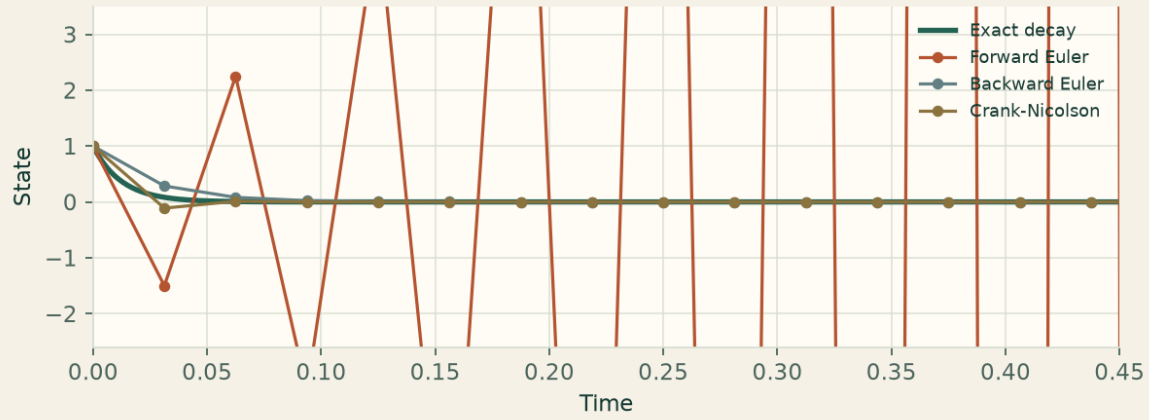
fourth_order = Numerics(spatial=space, time_integrator=ClassicalRungeKutta4())
second_order = Numerics(spatial=space, time_integrator=TimeMethod("ssprk2"))
```

Stable implicit updates on a stiff mode

For negative real stiffness, forward Euler becomes unstable beyond its linear stability interval.

TIME INTEGRATION / STIFFNESS

An explicit and an implicit step on a stiff decay



Here h = 0.03125 and lambda = -80 | Unstable explicit samples leave the vertical plotting range

Independent scalar example at the same fixed step. The exact decay, backward Euler and Crank-Nicolson show different damping.

$$R_{FE}(z) = 1 + z, \quad R_{BE}(z) = 1/(1 - z), \quad R_{CN}(z) = (1 + z/2)/(1 - z/2)$$

What the plot shows Backward Euler strongly damps the stiff mode. Crank-Nicolson is stable on the negative real axis but can alternate signs when the step is large relative to the decay time.

Connect to a model Check the coupled stage residual and algebraic constraints. Implicit does not mean the nonlinear solve will converge for any initial guess.

API IN PRACTICE

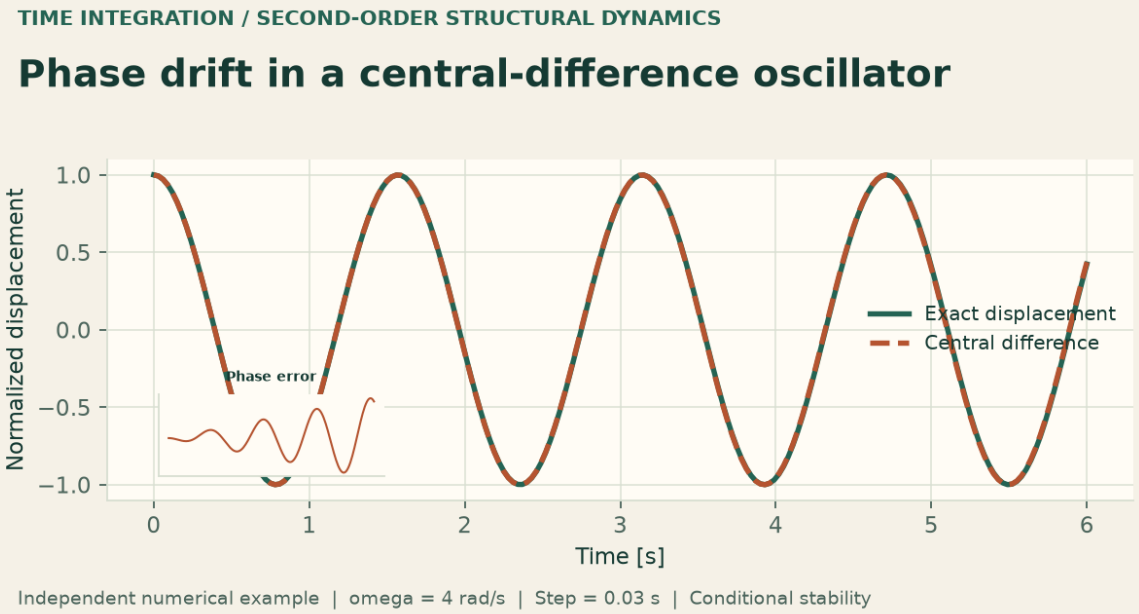
Select an implicit update and give the study a nonlinear stage solver. The spatial model and boundary data remain separate.

```
from onesim.numerics import Numerics
from onesim.numerics.time import TimeMethod
from onesim.numerics.nonlinear import Newton

numerics = Numerics(spatial=space, time_integrator=TimeMethod("crank_nicolson"),
                    nonlinear_solver=Newton(linear_solver=linear))
```

Centered updates and the wave time scale

Second-order displacement dynamics use a distinct execution surface in the catalog.



Independent central-difference example. The amplitude stays bounded while the discrete phase drifts.

$$u_{n+1} = 2u_n - u_{n-1} - h^2\omega^2 u_n$$

Stability For the undamped scalar oscillator the usual strict stability range is 0 < h times omega < 2. Highest spatial frequencies often set the usable structural step.

Related methods Leapfrog advances a first-order history formulation on its supported surface. Central difference is a second-order-state method. The notebook uses classical RK4 on a two-variable first-order system.

API IN PRACTICE

The second-order kernel advances a position and velocity pair directly. This example takes one step for the harmonic oscillator above.

```
from onesim.numerics.time.methods.kernels import CentralDifferenceIntegrator

stepper = CentralDifferenceIntegrator()
rhs = lambda state, time: (state[1], -16.0 * state[0])
position, velocity = stepper.advance((1.0, 0.0), rhs, 0.0, 0.12)
```

Explicit methods

Canonical method IDs, formal orders and execution surfaces from the inspected source.

Method ID	Order	Execution surface	Worked-use example
forward_euler	1	generic-object, universal-array	Start y = 1 on y' = -y. At h = 0.1 the first step is 0.9.
classical_rk4	4	generic-object, universal-array	Advance the damped oscillator with four stages. Verify phase and energy against the notebook reference.
ssprk2	2	generic-object, universal-array	Use the two-stage update on a monotone advective flux under its SSP step restriction.
embedded_heun_euler	2	generic-object, universal-array	Compare the Heun correction with the Euler predictor to estimate a local error on smooth decay.
adams_bashforth2	2	generic-object, universal-array	After one startup step, use the two most recent slopes to advance a smooth prescribed forcing.
leapfrog	2	generic-object, universal-array	Use equal steps and a consistent startup for centered wave evolution. Inspect its computational mode.
central_difference	2	second-order-array	Advance second-order oscillator displacement using its previous two states. Inspect the phase plot.
predictor_corrector_abam2	2	generic-object, universal-array	Predict a decay step from two slopes and evaluate a trapezoidal correction at the predicted state.

Selection The method catalog describes capabilities. A given state representation, model binder and provider may support a narrower set of methods.

API IN PRACTICE

For a supported first-order model, pass the catalog ID as a TimeMethod. The second-order central difference example uses its separate kernel.

```
from onesim.numerics.time import TimeMethod

stepper = TimeMethod("embedded_heun_euler")
```

Implicit single-step and split methods

Canonical method IDs, formal orders and execution surfaces from the inspected source.

Method ID	Order	Execution surface	Worked-use example
backward_euler	1	generic-object, index-one-dae, universal-array	For y' = -80y at h = 0.03125, the update multiplies y by 1/3.5.
crank_nicolson	2	generic-object, index-one-dae	On y' = -y with h = 0.1 the first step is 0.95/1.05.
theta_075	1	generic-object, index-one-dae	Blend 25% old and 75% new slope on a diffusion mode to damp more than trapezoidal weighting.
implicit_rk_midpoint	2	generic-object	Solve a single midpoint stage. On a linear decay it gives the same amplification as trapezoidal integration.
dirk_sdirk2	2	generic-object	Use two implicit stages with a shared diagonal coefficient on a stiff diffusion mode.
imex_sdirk2	2	partitioned-imex	Split nonstiff advection and stiff reaction. Attach the split right-hand sides and stage solver.

Selection The method catalog describes capabilities. A given state representation, model binder and provider may support a narrower set of methods.

API IN PRACTICE

Select a named implicit method and supply its stage solver in Numerics. A split IMEX route uses distinct explicit and implicit inputs.

```
from onesim.numerics.time import TimeMethod

stepper = TimeMethod("dirk_sdirk2")
```

Implicit history methods

Canonical method IDs, formal orders and execution surfaces from the inspected source.

Method ID	Order	Execution surface	Worked-use example
adams_moulton_2	2	generic-object	Solve the second-order implicit corrector on a smooth scalar decay.
adams_moulton_3	3	generic-object	Carry two prior history entries to correct a smooth time-dependent forcing.
adams_moulton_4	4	generic-object	Use the longer history on a smooth trajectory and refine time to check fourth-order behavior.
bdf2_gear	2	generic-object	Use one stored older state after startup to advance stiff decay.
bdf3_gear	3	generic-object	Start from three consistent states on smooth diffusion decay and check the history error.
bdf4_gear	4	generic-object	Refine a smooth transient with sufficient startup accuracy to expose the multistep order.
bdf5_gear	5	generic-object	Try a smooth slow mode and compare behavior for a complex eigenvalue before adopting the step.
bdf6_gear	6	generic-object	Use six consistent states in a smooth test. Check stability and sensitivity to history explicitly.
bdf2	2	core-implicit-delegate, index-one-dae	Restart a provider-owned implicit evolution and verify that its previous state and time step are preserved.

Selection The method catalog describes capabilities. A given state representation, model binder and provider may support a narrower set of methods.

API IN PRACTICE

Use the typed BDF declaration for startup and step growth. Select an Adams-Moulton corrector by its catalog ID.

```
from onesim.numerics.time import BDF, BackwardEuler, TimeMethod

gear = BDF(order=2, startup=BackwardEuler(), maximum_step_ratio=1.5)
corrector = TimeMethod("adams_moulton_3")
```

Linearized and provider-owned methods

Canonical method IDs, formal orders and execution surfaces from the inspected source.

Method ID	Order	Execution surface	Worked-use example
rosenbrock_euler	1	generic-object	Linearize a stiff scalar reaction. Solve once with the shifted Jacobian.
rosenbrock_wanner_euler	1	generic-object	Advance a stiff reaction with a linearly implicit Wanner-Euler route and inspect the Jacobian solve.
monolithic_backward_eul er	1	monolithic-residual	Step a coupled differential-algebraic model with the residual and algebraic rows in one solve.

Selection The method catalog describes capabilities. A given state representation, model binder and provider may support a narrower set of methods.

API IN PRACTICE

Declare the Rosenbrock method by name and pair it with a backend that can solve its linearized stages.

```
from onesim.numerics.time import TimeMethod

stepper = TimeMethod("rosenbrock_euler")
```

Compare the selected methods on one decay

The repository test adapter advances y' = -y from y(0) = 1 with ten steps of 0.1. The exact final state is 0.3678794412.

Selected method	Final state	Absolute error
forward_euler	0.3660323413	0.00185
classical_rk4	0.3678794412	3.09e-11
ssprk2	0.3678856187	6.18e-06
embedded_heun_euler	0.3678856187	6.18e-06
adams_bashforth2	0.3678762835	3.16e-06
leapfrog	0.3679443266	6.49e-05
predictor_corrector_abam2	0.3678763759	3.07e-06
backward_euler	0.3697112123	0.00183
crank_nicolson	0.3678763755	3.07e-06
theta_075	0.3687949407	0.000915
implicit_rk_midpoint	0.3678763755	3.07e-06

Interpretation These are OneSimulation source executions on a scalar test through the declared time adapter. The results reflect startup and solver choices as well as the formula.

Specialized routes Central difference, partitioned IMEX, Rosenbrock methods and provider-owned DAE or monolithic methods have additional state or solver contracts. Their use examples appear in the catalog.

API IN PRACTICE

A study consumes the method declaration through Numerics. Reuse the physical problem to compare the selected integration formulas.

```
from onesim.numerics import Numerics
from onesim.numerics.time import TimeMethod

selected = Numerics(spatial=space, time_integrator=TimeMethod("ssprk2"))
```

Compare the selected methods on one decay

The repository test adapter advances y' = -y from y(0) = 1 with ten steps of 0.1. The exact final state is 0.3678794412.

Selected method	Final state	Absolute error
dirk_sdirk2	0.3678779521	1.49e-06
adams_moulton_2	0.3678763755	3.07e-06
adams_moulton_3	0.3678794564	1.52e-08
adams_moulton_4	0.3678794411	9.53e-11
bdf2_gear	0.3678948653	1.54e-05
bdf3_gear	0.3678795319	9.08e-08
bdf4_gear	0.3678794405	7.21e-10
bdf5_gear	0.3678794412	5.64e-12
bdf6_gear	0.3678794412	2.64e-12
BDF(order=2)	0.3678948653	1.54e-05

Interpretation These are OneSimulation source executions on a scalar test through the declared time adapter. The results reflect startup and solver choices as well as the formula.

Specialized routes Central difference, partitioned IMEX, Rosenbrock methods and provider-owned DAE or monolithic methods have additional state or solver contracts. Their use examples appear in the catalog.

API IN PRACTICE

A study consumes the method declaration through Numerics. Reuse the physical problem to compare the selected integration formulas.

```
from onesim.numerics import Numerics
from onesim.numerics.time import TimeMethod

selected = Numerics(spatial=space, time_integrator=TimeMethod("ssprk2"))
```

History, adaptivity and study-specific routes

A robust run preserves the state and the algorithm's history together.

BDF declaration BDF accepts orders 1 through 6, explicit startup selection and a maximum step ratio. Check the specific provider for supported order, variable step and restart combinations. BDF2 variable-step growth is bounded by the declaration's zero-stability guard.

Adaptive stepping Compare the accepted solution with an embedded, defect or split error estimate on the applicable surface. Reject and retry a step when needed, with limits for step size and rejection count.

Specialized study vocabulary The separate dynamic-study enum names forward_euler, rk45, bdf, multirate and index_one_dae. RK45 denotes an adaptive explicit Runge-Kutta route in that study layer. Multirate and index-one DAE name execution strategies rather than single universal tableaus.

Provider ports Index-one theta, native cell DAE, monolithic residual, weak-source IMEX and split IMEX have different contracts. Preserve mass terms, algebraic rows, stage solves and any nonlinear or linearized solver controls.

Worked restart test Advance a periodic diffusion or damped oscillator case, checkpoint partway, restore the same selected configuration and compare subsequent states and output times against an uninterrupted run.

API IN PRACTICE

Declare the startup method and step bounds separately. Pass both to a provider that accepts adaptive BDF.

```
from onesim.numerics.time import AdaptiveTimeStep, BDF, BackwardEuler

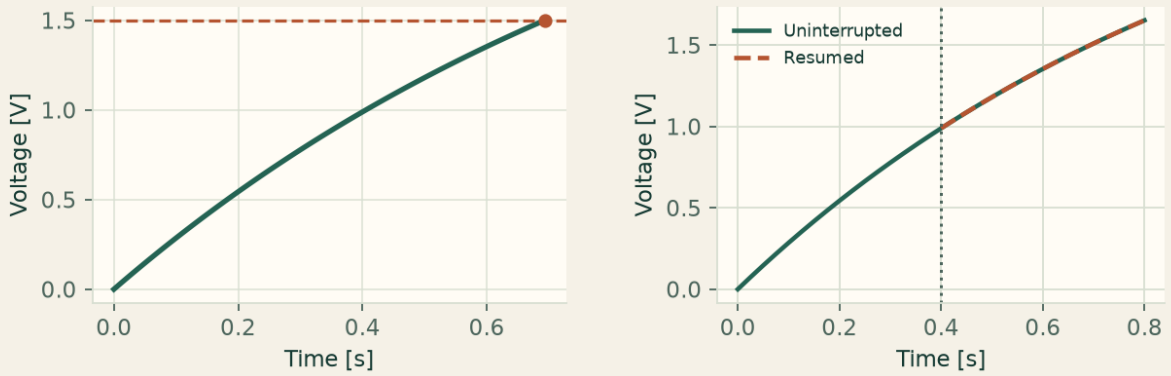
stepper = BDF(order=2, startup=BackwardEuler(), maximum_step_ratio=1.5)
control = AdaptiveTimeStep(initial=0.01, minimum=1e-4, maximum=0.05)
```

Stop on an event and continue from history

The same RC model tests adaptive event localization and checkpointed BDF2 continuation.

ONESIMULATION EXECUTION / TIME EVENT AND CHECKPOINT

Stop on a field threshold, resume with history



RC circuit | Adaptive threshold time checked against ln(2) | BDF2 checkpoint at 0.4 s

OneSimulation source execution. The threshold run and the checkpoint comparison use separate time configurations.

Check the accepted state The adaptive solver reached the 1.5 V event at 0.6931472 s, compared with ln(2) s. The resumed and uninterrupted BDF2 runs agree at the final time to 0 V in this run.

API IN PRACTICE

Use a field event for terminal conditions. For continuation, pass the saved study checkpoint through the execution options.

```
from onesim.simulation import ExecutionOptions, FieldThreshold

event = FieldThreshold(field=rc.voltage, value=1.5,
                       event_id="half-charge", direction=1)
stop = ExecutionOptions(events=(event,))
resume = ExecutionOptions(restart_checkpoint=first.last_checkpoint)
```

Separate outputs and restart data Stored output frames support field queries and visualization. A compatible checkpoint carries integration history and execution identity for supported continuations.

From discretization to sparse equations

Spatial assembly and implicit stages create systems that can have very different algebraic structure.

$$Ax = b, \qquad J(u_k)\,\delta u_k = -R(u_k), \qquad u_{k+1} = u_k + \lambda_k \delta u_k$$

Structure	Useful options	Decision point
Real symmetric positive definite	SparseDirect or ConjugateGradient	CG requires the appropriate operator and preconditioner properties.
Nonsymmetric real	SparseDirect or GMRES	Restart and preconditioning affect iteration counts.
Indefinite saddle point	General sparse direct or block-preconditioned Krylov	Handle the pressure or gauge constraint explicitly.
Complex frequency-domain	ComplexSparseDirect or supported complex Krylov	Preserve complex arithmetic and phasor conventions.
Nonlinear residual	Newton variants, FixedPoint or JFNK	Separate the outer stopping test from the linear solve tolerance.

Preconditioning ILU and algebraic multigrid approximate inverse action in different ways. Block and Schur selections use field structure. Choose a compatible backend and inspect what was consumed.

API IN PRACTICE

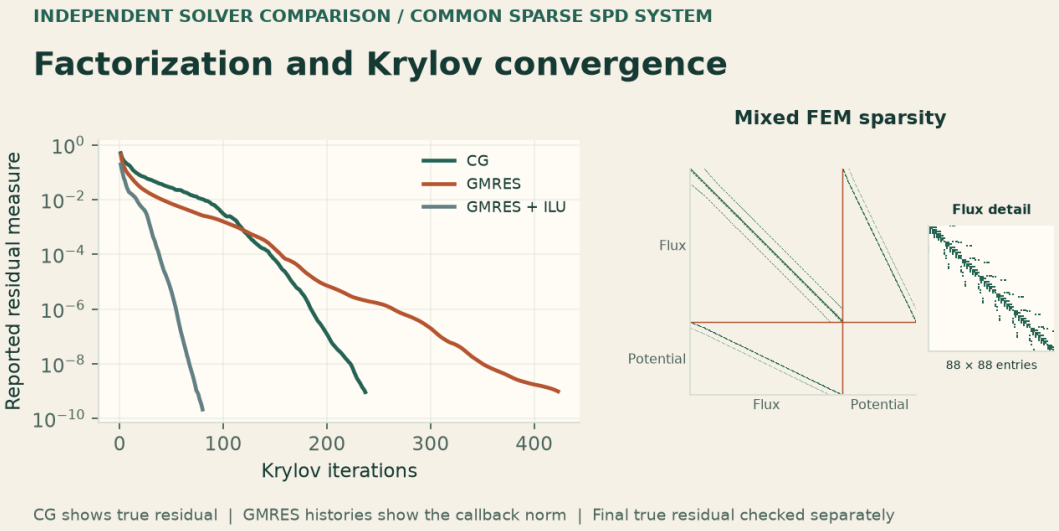
Choose a direct, SPD iterative or preconditioned general solver based on the assembled operator.

```
from onesim.numerics.linear import SparseDirect, ConjugateGradient, GMRES, ILU

direct = SparseDirect()
spd = ConjugateGradient(relative_tolerance=1e-9)
krylov = GMRES(preconditioner=ILU(), restart=80, relative_tolerance=1e-9)
```

Sparse LU and Krylov on the same matrix

A 5184-unknown five-point Poisson system with 25632 nonzeros, one right-hand side and deterministic forcing.



SciPy histories for the Poisson system appear on the left. The right panel shows 228,960 nonzeros from the 31,968-unknown OneSimulation mixed FEM assembly. The off-diagonal blocks couple flux and potential. The inset shows 88 rows and columns of the flux block.

Method	Iterations	True relative residual
CG	237	9.2e-10
GMRES	423	9.75e-10
GMRES + ILU	80	3.31e-10
Sparse LU	Factorization	2.16e-15

What changes Sparse LU stores triangular factors and can reuse them for additional right-hand sides. CG uses SPD structure. GMRES builds a Krylov basis and can restart; ILU trades extra setup and storage for faster convergence on this matrix.

API IN PRACTICE

Swap the linear solver in Numerics while keeping the assembled problem and field space fixed. The chart uses an independent matrix benchmark.

```
from onesim.numerics import Numerics
from onesim.numerics.linear import SparseDirect, GMRES, ILU

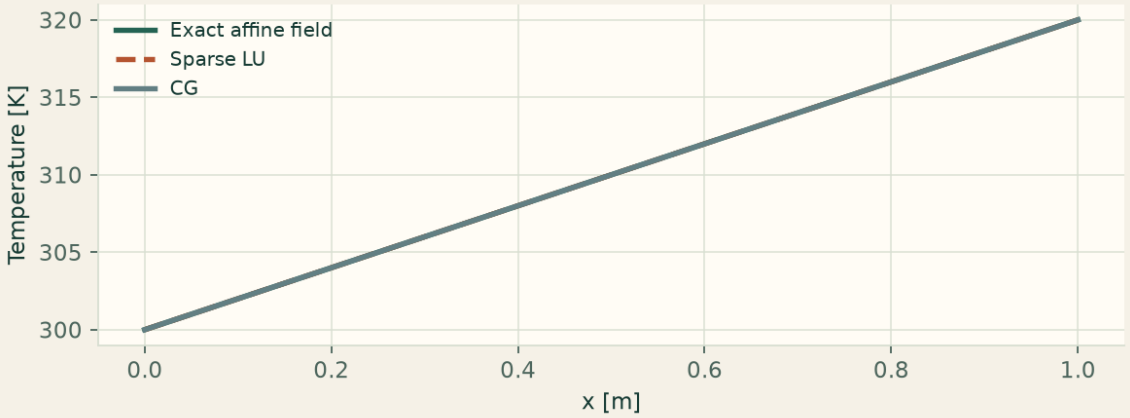
direct = Numerics(spatial=space, linear_solver=SparseDirect())
iterative = Numerics(spatial=space, linear_solver=GMRES(preconditioner=ILU()))
```

Direct and iterative choices in a real study

A 3D finite-element conduction problem makes the effect of the selected linear solver observable without changing the model.

ONESIMULATION EXECUTION / P1 FINITE ELEMENTS

A linear temperature field in a tetrahedral cube



Centerline x at y = z = 0.5 m | 2123 tetrahedra | Affine Dirichlet boundary on all faces

OneSimulation source execution. The same tetrahedral P1 problem is solved with SparseDirect and ConjugateGradient.

```
from onesim.numerics.linear import ConjugateGradient, SparseDirect

solvers = [SparseDirect(), ConjugateGradient(relative_tolerance=1e-11)]
for solver in solvers:
    numerics = Numerics(spatial=space, linear_solver=solver)
    result = Simulation(problem=problem, mesh=mesh,
                        numerics=numerics).solve_stationary()
```

> **Interpretation** Both solutions recover the affine exact field on the refined mesh. That agreement verifies the model and the selection, while the separate Krylov comparison exposes convergence behavior.

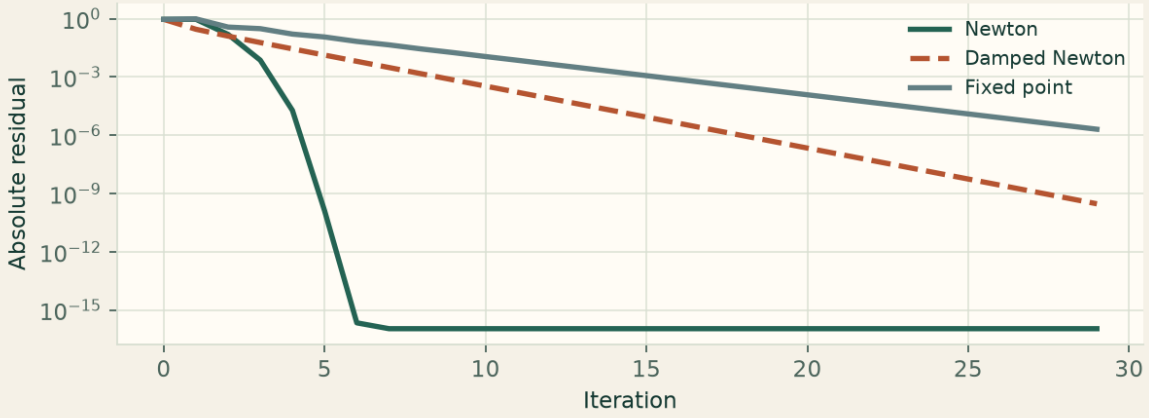
Newton, damping and fixed-point iteration

Solve a scalar residual first to understand the stopping test, then apply the same concepts to coupled fields.

$$R(u) = u^3 + u - 1, \qquad R'(u) = 3u^2 + 1$$

INDEPENDENT NONLINEAR COMPARISON / SCALAR RESIDUAL

How nonlinear methods approach the root



Same initial state | Damping trades progress per iteration for smaller corrections

Independent comparison for a smooth scalar problem. The plot starts all three iterations from the same state.

OneSimulation exports Newton, DampedNewton, TrustRegionNewton, QuasiNewton, FixedPoint, JFNK and JacobianFreeNewtonKrylov expose different nonlinear configurations. Verify the residual, Jacobian action and line-search or trust-region choices for the provider.

Krylov inside Newton JFNK can avoid forming a dense Jacobian, while still benefiting from a useful preconditioner. Check Jacobian-vector products and residual scales against independent directional derivatives.

API IN PRACTICE

Set the nonlinear strategy independently of the field equations. linear is the compatible solver for each Newton correction.

```
from onesim.numerics.nonlinear import Newton, DampedNewton, FixedPoint

newton = Newton(linear_solver=linear)
damped = DampedNewton(linear_solver=linear, line_search_reduction=0.52)
fixed_point = FixedPoint(relative_tolerance=1e-8)
```

What a solver report can tell you

A small residual is one part of a verification workflow. It measures the assembled equation at the current representation.

Convergence norms Compare the final unpreconditioned residual, scaling, termination reason and requested tolerance. A preconditioned callback norm does not have the same meaning as the physical residual.

Matrix ownership Use CG on supported positive-definite operators. A constraint saddle point may need a gauge, augmentation or block-preconditioned solve. RightGMRES and FlexibleGMRES require providers that implement their requested algorithms.

Factor and preconditioner lifecycle A changing matrix can invalidate a reusable factorization or preconditioner. Check whether the graph, numeric entries, boundary rows or field ordering have changed.

Physical checks Reconstruct the fields, apply boundary and flux checks, and compare quantities of interest across spatial and temporal refinement. Mesh error can dominate algebraic error.

Timing and scale The comparison in this book uses one deterministic matrix and exposes convergence. Representative time and memory comparisons require a workload, hardware, several repeats and a separate account of setup cost.

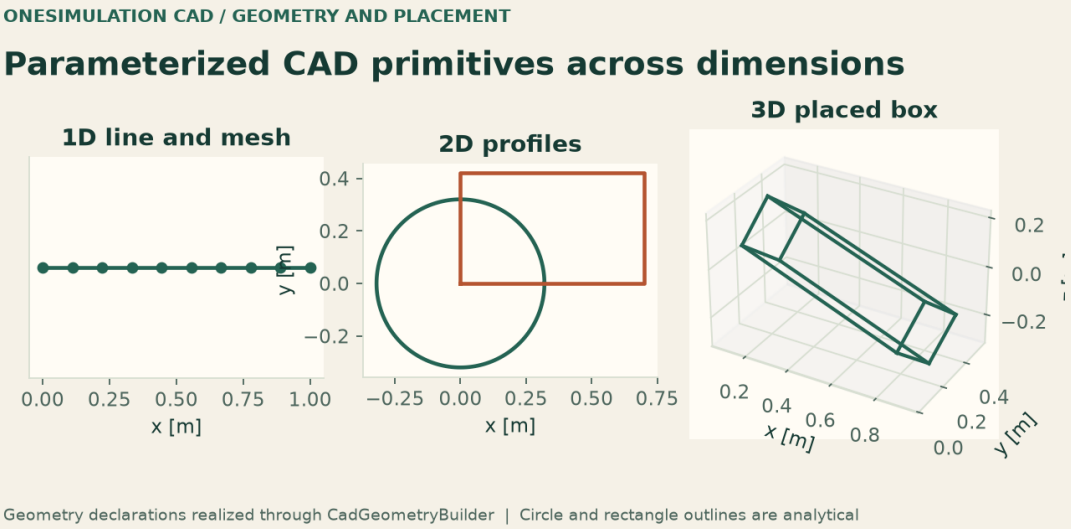
API IN PRACTICE

Inspect the result's solver report after solving, alongside the physical field residual and balance checks.

```
result = simulation.solve_stationary()
report = result.result_set.solver_report
print(report["provider_id"])
```

Geometry that carries simulation intent

A parametric CAD declaration defines spatial measures, local frames and boundary identity before a discretization is built.



OneSimulation source execution for geometry realization and line meshing. The profile curves show their exact declared shapes.

Dimension	Selected declarations	Examples of downstream use
1D	Line	Axial coordinates, endpoint faces, structured and nonuniform line cells
2D	Circle, Rectangle, Plane	Planar domains, cross-sections and geometry frames
3D	Box, Cylinder, AnnularCylinder, curved cell declarations	Volumes, named faces, mesh generation and field support

> **Placement** The box example converts local corners to world coordinates using its declared frame. Face normals and bounds must use that same transform.

API IN PRACTICE

Declare dimensions and placement in world coordinates, then read the realized geometry measures.

```
from onesim.cad import CadGeometryBuilder
from onesim.cad.d3 import Box

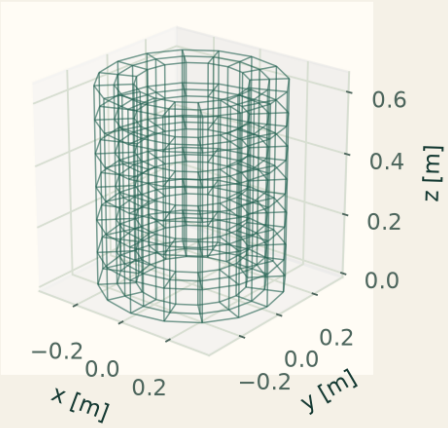
box = Box(length=0.8, width=0.33, height=0.21, axis=(1, 2, 3))
built = CadGeometryBuilder().build(box)
volume = built.data.metrics.volume
```

Cut a bore and retain the construction

An annular solid is built from two cylinders and a Boolean subtraction, then discretized as a conforming ring mesh.

ONESIMULATION CAD / BOOLEAN HISTORY AND BOUNDARY-AWARE MESH

A bore cut through a cylinder



1. **Outer cylinder**
2. **Inner tool cylinder**
3. **Boolean subtraction**
4. **Structured annular mesh**

Boolean construction realized by the CAD kernel | 192 structured annular hexahedra

OneSimulation source execution. The annular wireframe comes from actual mesh connectivity.

```
from onesim.cad import CadGeometryBuilder
from onesim.cad.d3 import AnnularCylinder
from onesim.meshing import StructuredAnnularHexahedralMesher

part = AnnularCylinder(inner_radius=.19, outer_radius=.32, length=.65)
built = CadGeometryBuilder().build(part)
mesh = StructuredAnnularHexahedralMesher(
    radial_cells=2, azimuthal_cells=16, axial_cells=6,
    geometry_order=1,
).generate(part)
```

Check the construction The CAD kernel reports volume 0.135387 cubic meters. The demonstration mesh contains 192 cells. The inner and outer surfaces serve different physical boundary roles.

Selections, assemblies and exchange

Geometry operations and identity let a script recover meaningful topology after construction or placement.

Concern	Source surfaces	How to inspect it
Feature history	FeatureHistoryExecutor, CadGeometryBuilder	Read construction nodes, realized body references and metrics.
Placement and joints	CadPlacementService, JointDatumFactory	Compare transformed points, bounds and normal directions.
Selections	SelectionQuery, resolve_selection, named boundaries	Check the returned entity set and the owning geometry revision.
Repair and exchange	Edge-healing proposals and exchange backends	Inspect proposed topology changes and select the supported format/backend explicitly.
Mesh handoff	Structured and tetrahedral mesh generators	Compare cell count, boundary tags, physical measure and element quality.

Physical boundary selection Apply temperature on a selected face, a mechanical support on another, or a prescribed flux on an annular wall. Track both the physical sign and the selected entity.

Interoperability The source provides CAD and exchange components. Available format and kernel operations depend on the selected backend and deployment.

API IN PRACTICE

Form a boundary selection directly from named faces. Assign the same condition to the union in one declaration.

```
from onesim.physics.heat_transfer import FixedTemperature

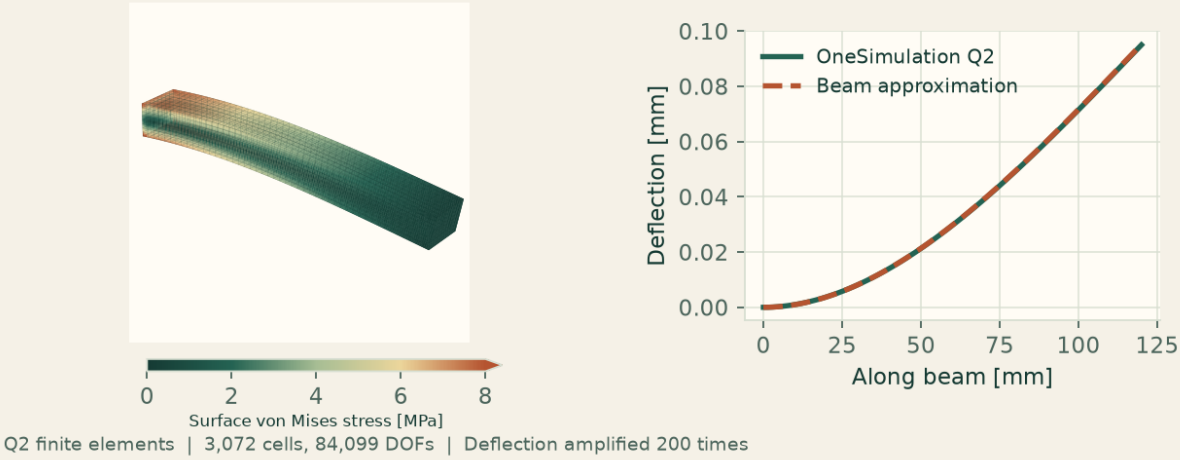
sides = box.boundaries.x_min | box.boundaries.x_max
ambient = FixedTemperature(boundary=sides, physics=heat, value=293.15)
```

From geometry to a field space

A correct geometry is the first link in a chain that also includes cell maps, orientation and basis functions.

ONESIMULATION EXECUTION / 3D ELASTICITY

Cantilever bending and surface stress



Q2 finite elements | 3,072 cells, 84,099 DOFs | Deflection amplified 200 times

Recorded OneSimulation example. Structured geometry and boundary identity support a higher-order field solve and surface stress reconstruction.

Geometry order and field order A curved or higher-order geometry mapping describes the cell shape. The field space describes the numerical approximation on that mapping. Check these orders and quadrature separately.

Orientation Cell Jacobians, facet normals and edge orientations determine volume, flux and curl conventions. Check for inverted cells and mismatched boundary tags.

Moving and adapted meshes Geometry and state transfer must preserve relevant integral, constraint and history information. A new mesh generally requires a new field-space binding.

API IN PRACTICE

Choose the hexahedral mesh resolution separately from the displacement approximation order.

```
from onesim.meshing import StructuredHexahedralMesher
from onesim.numerics.fem import FiniteElement, Lagrange

mesh = StructuredHexahedralMesher(cells=(16, 3, 3)).generate(beam)
space = FiniteElement(element=Lagrange(order=2))
```

Give the same geometry different physical roles

A geometric selection identifies support. A material law and a boundary condition identify the physical quantity on that support.

Layer	Source surface	Representative use
Geometry and topology	CAD construction, placement, history, repair, selections and exchange namespaces	Use a named face after placement or subtract a bore and preserve the inner wall.
Mesh lifecycle	Generation, boundary mapping, quality, refinement, adaptation, partitioning and exchange	Use a structured or tetrahedral carrier, inspect physical measure and transfer state after remeshing.
Materials	Fluids, solids, gases, electromagnetic response, thermal and transport laws	Select conductivity, viscosity, elasticity or complex constitutive data independently of the geometry.
Inputs and constraints	Typed fields, expressions, initial conditions and algebraic constraints	Apply a world-coordinate expression with units or fix a mean pressure or potential.

API IN PRACTICE

Give the thermal trace physical units and named coordinates. Pass the same expression object to the boundary declaration.

```
from onesim.simulation.problem_definition import Expression
from onesim.physics.heat_transfer import FixedTemperature

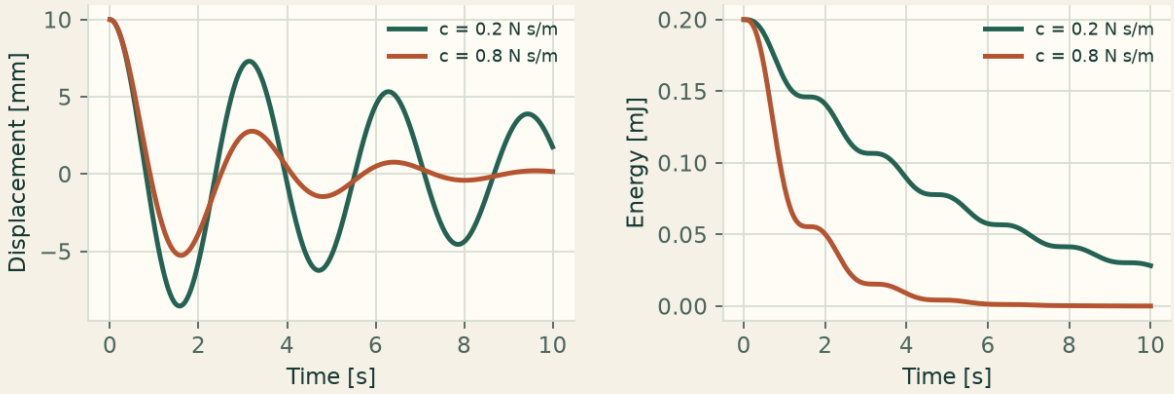
trace = Expression("base+slope*x", unit="K", coordinates=("x",),
                   parameters={"base": (300, "K"), "slope": (20, "K/m")})
boundary = FixedTemperature(boundary=face, physics=heat, value=trace)
```

From imports to an exported result

A complete English Jupyter notebook uses the public API to model a damped spring, validate the problem, run RK4 and export the results.

ONESIMULATION EXECUTION / CONCENTRATED DYNAMICS

Damping changes displacement and energy



Two actual RK4 runs | m = 1 kg, k = 4 N/m, step = 0.005 s, initial displacement = 10 mm

Two OneSimulation source runs at c = 0.2 and 0.8 N s/m. The energy plot makes the damping change easy to see.

$$m\ddot{u} + c\dot{u} + ku = 0, \qquad E = \tfrac{1}{2}m\dot{u}^2 + \tfrac{1}{2}ku^2$$

Numerical check The maximum displacement errors for the two traces are 6.07e-12 m and 1.53e-12 m against the closed-form underdamped solution.

Try it Use the complete companion notebook Damped_Oscillator.ipynb or copy the next four listings into separate cells. Change one input at a time and repeat the comparison.

API IN PRACTICE

Change the damping directly on the physical model. The following listings show every step from this declaration to the exported fields.

```
from onesim.physics.solid_mechanics import StructuralOscillator

model = StructuralOscillator(domain=body.system, mass=1.0,
                             stiffness=4.0, damping=0.8)
```

1. Imports and physical parameters

Use a Python environment with OneSimulation, NumPy and Matplotlib installed. The Runtime distribution uses its normal activation. All physical inputs in this notebook use SI units.

```
from pathlib import Path

import matplotlib.pyplot as plt
import numpy as np

from onesim.cad.geometry import LumpedDomain
from onesim.meshing import LumpedMesh
from onesim.numerics import Numerics
from onesim.numerics.discretization import LumpedParameters
from onesim.numerics.time import ClassicalRungeKutta4
from onesim.physics.solid_mechanics import StructuralOscillator
from onesim.results.storage import MemoryOutputSink, Output
from onesim.simulation import Problem, Simulation
from onesim.simulation.initial_conditions import InitialValue

mass = 1.0                  # kg
stiffness = 4.0             # N/m
damping = 0.2               # N s/m
initial_displacement = 0.01 # m
initial_velocity = 0.0      # m/s
time_step = 0.005           # s
end_time = 10.0             # s

assert mass > 0 and stiffness > 0
assert 0 < damping < 2 * np.sqrt(mass * stiffness)
```

2. Build and run the simulation

The domain, physical model, initial conditions, mesh, numerical method and output requests are separate declarations. The output cadence is 0.02 s, while the integrator advances in 0.005 s steps.

```
domain = LumpedDomain()
model = StructuralOscillator(
    domain=domain.system,
    mass=mass, stiffness=stiffness, damping=damping,
)
problem = Problem(
    name="Damped oscillator",
    geometry=domain,
    physics=(model,),
    initial_conditions=(
        InitialValue(model.displacement, initial_displacement),
        InitialValue(model.velocity, initial_velocity),
    ),
)
simulation = Simulation(
    problem=problem,
    mesh=LumpedMesh(domain),
    numerics=Numerics(
        spatial=LumpedParameters(),
        time_integrator=ClassicalRungeKutta4(),
    ),
)
simulation.validate().require_valid()

sink = MemoryOutputSink()
simulation.run(
    start_time=0.0, end_time=end_time, time_step=time_step,
    output=Output(
        fields=(model.displacement, model.velocity, model.mechanical_energy),
        interval=0.02, sink=sink,
    ),
)
```

3. Recover the fields and check the reference

Compare the full displacement time series with the underdamped solution. The energy identity checks that the displacement and velocity correspond to the energy field saved by the model.

```
time = np.asarray([float(s.time) for s in sink.snapshots])

def series(field):
    return np.asarray([
        np.asarray(s.fields[field.field_id].values).reshape(-1)[0]
        for s in sink.snapshots
    ])

displacement = series(model.displacement)
velocity = series(model.velocity)
energy = series(model.mechanical_energy)

alpha = damping / (2 * mass)
omega = np.sqrt(stiffness / mass - alpha**2)
a = initial_displacement
b = (initial_velocity + alpha * a) / omega
reference = np.exp(-alpha * time) * (
    a * np.cos(omega * time) + b * np.sin(omega * time)
)

max_error = np.max(np.abs(displacement - reference))
assert max_error < 2e-8
assert np.isclose(time[-1], end_time)
assert np.all(np.diff(energy) <= 1e-12)
assert np.allclose(
    energy, 0.5 * mass * velocity**2 + 0.5 * stiffness * displacement**2,
)
print(f"Maximum displacement error: {max_error:.3e} m")
```

4. Export CSV and SVG

The CSV keeps the units in its headers. The SVG records the figure in a format that can be resized for a report. Try increasing damping to 0.8 N s/m and rerun the notebook.

```
out = Path("onesimulation_results")
out.mkdir(exist_ok=True)
np.savetxt(
    out / "damped_oscillator.csv",
    np.column_stack([time, displacement, reference, velocity, energy]),
    delimiter=",",
    header="time_s,displacement_m,reference_m,velocity_m_s,energy_J",
    comments="",
)

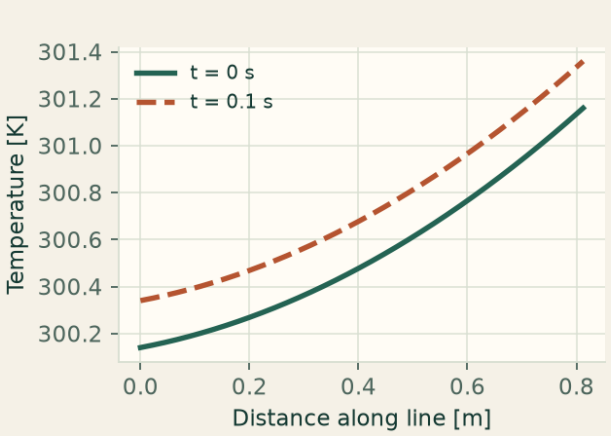
paper, forest, rust = "#f5f1e7", "#246353", "#b55430"
fig, (ax_u, ax_e) = plt.subplots(
    2, 1, figsize=(8, 6), sharex=True, layout="constrained",
)
fig.patch.set_facecolor(paper)
for axis in (ax_u, ax_e):
    axis.set_facecolor("#fffcf5")
    axis.grid(alpha=0.25)
ax_u.plot(time, 1e3 * reference, color=forest, label="Analytical reference")
ax_u.plot(time, 1e3 * displacement, ".", color=rust, label="OneSimulation")
ax_u.set_ylabel("Displacement [mm]")
ax_u.legend()
ax_e.plot(time, 1e3 * energy, color=forest)
ax_e.set(xlabel="Time [s]", ylabel="Mechanical energy [mJ]")
fig.savefig(out / "damped_oscillator.svg")
plt.show()
```

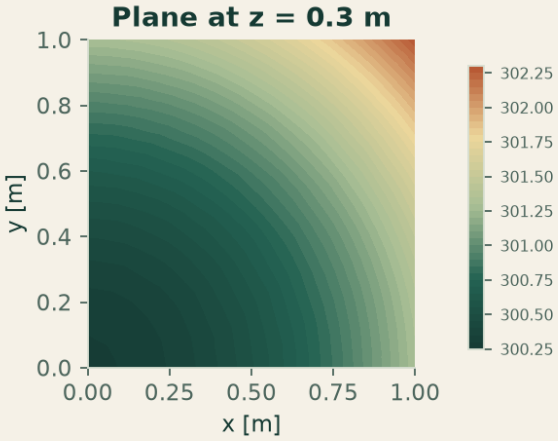
Ask a solved field a physical question

The same solution supplies values at world points, line profiles, plane cuts and integration over a named support.

ONESIMULATION EXECUTION / NATIVE RESULT QUERIES

One P2 field, two physical views





416 tetrahedra | Profile and plane evaluated from the native P2 basis

OneSimulation source execution. The line and plane use native finite-element queries and an accepted time.

> **Measure an integral** The manufactured unit cube at t = 0.1 s gives a temperature volume integral of 301.2 K m³ and net outward heat flux of -30 W. Divide a volume integral by its physical volume for a mean.

API IN PRACTICE

Keep the model's field reference, select an accepted time and sample on a physical support.

```
from onesim.cad.d2 import Plane

temperature = result.field(heat.temperature)
cut = Plane(normal=(0, 0, 1), origin=(0, 0, 0.3))
samples = temperature.sample(along=cut, time=0.1)
figure = temperature.plot(slice=cut, time=0.1)
```

Keep the data, view and restart choices explicit

Physical quantities can remain in memory, stream to a file sink, or be passed to a visualization or later study.

Function	Supported route	What to retain
Physical queries	Point evaluation, profiles, planes, components, normal fluxes and derived histories	Coordinates, field support, units and accepted time.
Visualization	Geometry, mesh, fields, networks, time series, animation and tables	View, time, component and complex representation. Interactive views use optional dependencies.
Output and persistence	Selected fields through memory or HDF5 sinks, frame retention, saved mesh sequences, versions and result archives	Sampling cadence, final frame and artifact identity.
Execution	Local worker submission, progress, diagnostics, checkpoint and restart	Provider, numerical configuration and necessary state history.
Reuse and comparison	Pass a solved field into an independent study or use a selected transfer between two supports	Region, geometry version, units and interpolation or conservation policy.

API IN PRACTICE

Choose which fields and times the run will export. This sink writes snapshots under the named directory.

```
from onesim.results.storage import HDF5OutputSink, Output

sink = HDF5OutputSink("field-output")
output = Output(fields=(heat.temperature, heat.heat_flux),
                interval=0.1, sink=sink)
```

Choose how linked models advance

OneSimulation distinguishes a physical field connection, a numerical transfer and the strategy used to advance the connected systems.

Strategy or port	What it controls	Repository evidence
One-way field connection	One source drives a receiving field or input port on its selected space.	P1 temperature transfer into P2 elasticity.
Sequential and partitioned	Order, exchange and iteration of separate solves, including time-dependent feedback.	Thermal exchange, flow–heat and partitioned rollback.
Monolithic and algebraic closure	A single residual includes the linked fields and constraints.	Coupled plasma and electromagnetic state, or an algebraic field feedback.
Quasi-stationary coupling	A stationary field is refreshed within a transient evolution.	Replay local integrators from the accepted state and reject failed trial updates.
Incompressible coupling	Velocity and pressure use a selected correction or coupled solve.	PISO, BDF2 momentum, volume pressure gauge and retained restart history.

API IN PRACTICE

Select a transfer independently of the coupling strategy. Declare the source and the receiving input on their actual field owners.

```
from onesim.numerics.field_transfer import ConservativeVolumeTransfer
from onesim.simulation.field_connections import FieldCoupling

link = FieldCoupling(source=source_field, target=receiver_input,
                     transfer=ConservativeVolumeTransfer())
```

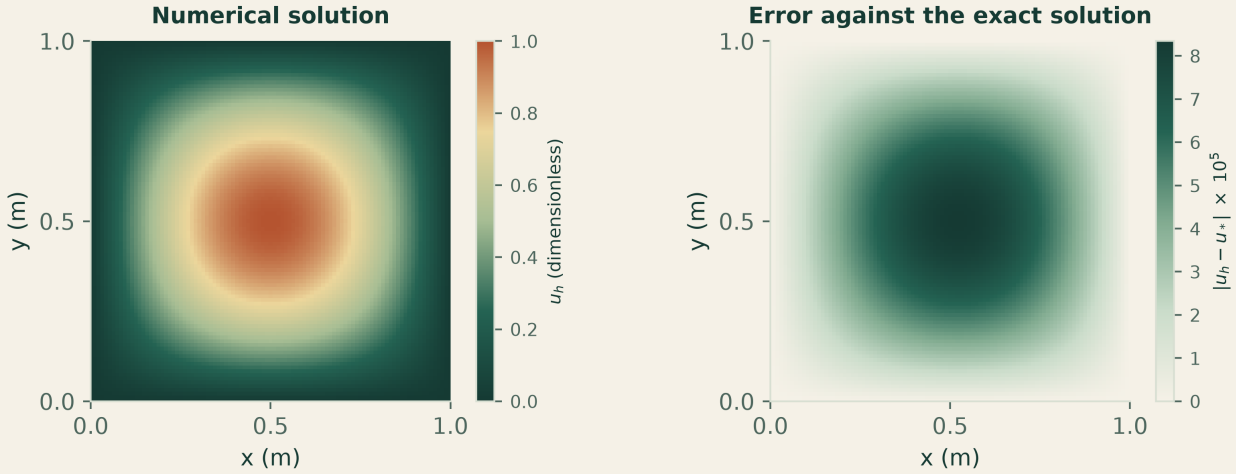
Your equation. OneSimulation's numerics.

Supply a mathematical model of your own, solve it through the shared numerical framework, and test the result against an independent reference.

$$-\nabla\cdot[D(x)\nabla u] + \kappa u = f, \qquad D(x) = D_0(1 + x/L)$$

YOUR EQUATION / COMPUTED 2D FIELDS

A supplied equation becomes a solved field



96 x 96 cells / 9,025 solved unknowns / zero Dirichlet boundary / CG + Jacobi

Actual OneSimulation solution on 96 x 96 cells. Both panels use the native 97 x 97 nodal field. This is a two-dimensional PDE; no analytical field is substituted for the numerical solution.

A model you define The coefficient, reaction rate, forcing and boundary value are supplied as EquationFormula objects. No built-in thermal or other physics model is instantiated.

A framework that solves it The recorded execution plan binds conservative flux differences to a sparse CSR matrix, Conjugate Gradient and Jacobi preconditioning. The fine solve has 9,025 interior unknowns and 44,745 nonzero matrix entries.

API IN PRACTICE

Select the solve directly after defining the equation on the next page.

```
result = solve_coefficient_form(
    equation, resolution=96, parameter_values=parameters,
    linear_method="cg", preconditioner="jacobi",
    relative_tolerance=1e-10, absolute_tolerance=1e-12,
    maximum_iterations=2000,
)
```

1. Declare the equation and its coefficients

These are the executed OneSimulation imports and declarations. The square is [0, 1] m by [0, 1] m; u is dimensionless and is prescribed to zero on every edge.

```
import numpy as np
from onesim.simulation.problem_definition.declarations.physics.equations \
    .equation_solver import (
    solve_coefficient_form,
)
from onesim.simulation.problem_definition.declarations.physics.models \
    .contracts.models.custom_models import (
    EquationFormula,
    EquationModel,
    EquationModelForm,
)

parameters = {"D0": 1.0, "L": 1.0, "kappa": 2.0}  # m^2/s, m, 1/s
source = (
    "(2*D0*(pi/L)**2*(1+x/L)+kappa)*sin(pi*x/L)*sin(pi*y/L)"
    "-D0*pi/L**2*cos(pi*x/L)*sin(pi*y/L)"
)
equation = EquationModel(
    equation_id="user.variable_diffusion",
    name="Your own reaction-diffusion equation",
    form=EquationModelForm.COEFFICIENT_PDE,
    dependent_variables=("u",),
    domain_ids=("unit_square",),
    dimension=2,
    coordinate_system="cartesian",
    formulas=(
        EquationFormula("diffusion", "D0*(1+x/L)", "m^2/s"),
        EquationFormula("absorption", "kappa", "1/s"),
        EquationFormula("source", source, "1/s"),
        EquationFormula("dirichlet", "0", "1"),
    ),
)
```

> **Units and geometry** D0 = 1 m^2/s, kappa = 2 1/s and L = 1 m. The coefficient-form helper assembles the unit square. Changing L alone does not resize its mesh; keep L = 1 for this reference case.

> **Source interface** These explicit module paths identify the working coefficient-form interface in the checked source version. The executable companion uses the same imports and declarations.

2. Derive a result you can check

A manufactured solution turns an unfamiliar equation into a controlled verification problem. Derive the forcing, pass only that forcing to the solver, and compare the returned field afterwards.

$$u_*(x,y) = \sin(\pi x/L)\sin(\pi y/L), \qquad u_*|_{\partial\Omega} = 0$$

$$f = [2D_0(\pi/L)^2(1 + x/L) + \kappa]u_* - \frac{D_0\pi}{L^2}\cos(\pi x/L)\sin(\pi y/L)$$

> **Keep the product rule** The second term comes from the spatial derivative of D(x). Omitting it changes the PDE and destroys this reference comparison.

> **Choose a compatible solver** Positive diffusion, nonnegative absorption and prescribed boundary values produce a symmetric positive-definite discrete system in this case. CG is appropriate here; another equation may require a different solver.

```
result = solve_coefficient_form(
    equation, resolution=96, parameter_values=parameters,
    linear_method="cg", preconditioner="jacobi",
    relative_tolerance=1e-10, absolute_tolerance=1e-12,
    maximum_iterations=2000,
)
axis = np.asarray(result["coordinates"])
x, y = np.meshgrid(axis, axis, indexing="ij")
u = np.asarray(result["solution"])
exact = np.sin(np.pi*x) * np.sin(np.pi*y)
error = u - exact
assert u.shape == (97, 97)
assert np.max(np.abs(error)) < 1e-4
assert result["solver"]["relative_residual"] < 2e-10
```

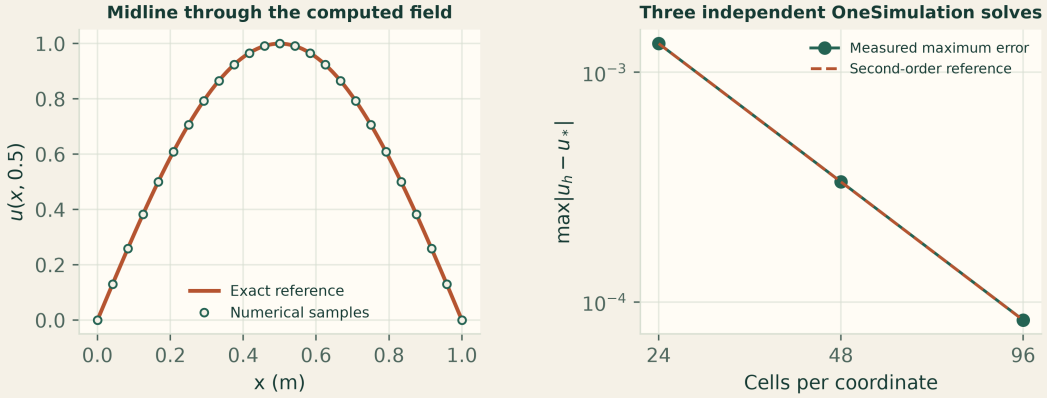
> **Measured fine-grid result** Maximum error 8.3428e-5; relative L2 error 8.2925e-5. The linear solver takes 138 iterations and reports a relative residual of 4.0647e-11.

3. Refine the problem, not the picture

Three independent solves establish spatial convergence. Rendering a denser image would not provide this evidence.

YOUR EQUATION / INDEPENDENT VERIFICATION

Refine the mesh. Measure the error.



Observed orders: 1.998 and 2.000 / fine-grid maximum error: 8.343e-05

The numerical midline agrees with the analytical curve. Doubling the cell count along each axis reduces the maximum field error by approximately four.

Cells	Unknowns	Maximum error	Observed order
24 x 24	529	1.3335e-03	--
48 x 48	2,209	3.3376e-04	1.9983
96 x 96	9,025	8.3428e-05	2.0002

Two different checks The solver residual measures the accuracy of the algebraic solve. The error against u_* and the observed orders 1.9983 and 2.0002 measure the spatial approximation.

API IN PRACTICE

Repeat the same declaration while changing only numerical resolution.

```
for cells in (24, 48, 96):
    result = solve_coefficient_form(
        equation, resolution=cells, parameter_values=parameters,
        linear_method="cg", preconditioner="jacobi",
        relative_tolerance=1e-10, absolute_tolerance=1e-12,
        maximum_iterations=2000,
    )
```

Reproduce it Custom_Equation.ipynb includes the complete calculation, assertions and plots. Custom_Equation.py writes all three native fields and the numerical provenance. The executed notebook and script produced bitwise-identical solution arrays.

Take the numerical framework beyond the built-in models

Choose the extension level that matches your equation. Keep the physical definition, spatial operator and numerical method explicit.

You supply	OneSimulation provides
EquationModel and EquationFormula for an executable coefficient-form case	Structured assembly, sparse operator, selected stationary solver and convergence diagnostics.
A Python right-hand side or nonlinear residual through the operator interfaces	Compatible time integration, nonlinear stages and linear-solver machinery.
PhysicsModel, typed fields, PhysicalEquation and a model runtime binder	Integration into the common problem, mesh, solve and result lifecycle.

What this example establishes A user-defined variable-coefficient reaction-diffusion PDE executes through OneSimulation in 2D. Its fine-grid field and second-order refinement have been measured.

The boundary of the formula helper This stationary coefficient-form route supports scalar Cartesian 1D/2D problems with positive diffusion and nonnegative absorption. It does not turn every arbitrary PDE string into an FEM, FVM or FDM method. More general equations need a suitable numerical operator and binding.

A separate 3D extension check The repository integration test test_custom_model.py binds a user-declared diffusion model to quadratic 3D finite elements and SparseDirect, then verifies its solution. This demonstrates the model/binder extension path; it is separate from the 2D formula example.

Try a controlled change Vary D0 or kappa while retaining the supplied source expression. The manufactured exact solution remains the same, so the numerical effect can be checked. Re-derive the reference and boundary values when changing the equation itself.

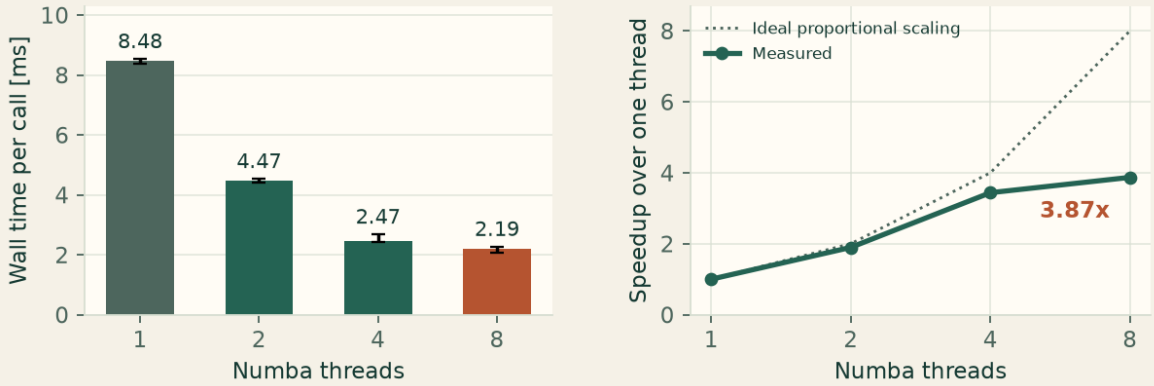
Source and Runtime scope The example was executed against the source checkout on 11 November 2026. Check the imports and capabilities in your installed release. The Cookbook gives the derivation, complete code, diagnostics and reproducibility steps.

Measured CPU thread scaling

OneSimulation runs selected numerical kernels across CPU threads. A finite-volume flux-divergence benchmark measures the effect on one Mac.

SHARED MEMORY / MEASURED SOURCE EXECUTION

One finite-volume kernel, four thread counts



262,144 cells, three fields | Numba 0.66.0 | Nine timed batches per thread count

Measured median wall time. Error bars span nine batches of eight calls after warm-up. Speedup is relative to the same compiled kernel with one thread.

Threads	Median per call	Speedup
1	8.48 ms	1.00x
2	4.47 ms	1.90x
4	2.47 ms	3.44x
8	2.19 ms	3.87x

Numerical parity All four thread counts produced bit-identical arrays. A separate 512-cell periodic heat-transport study through the local worker also gave bit-identical temperatures at one and four configured threads.

API IN PRACTICE

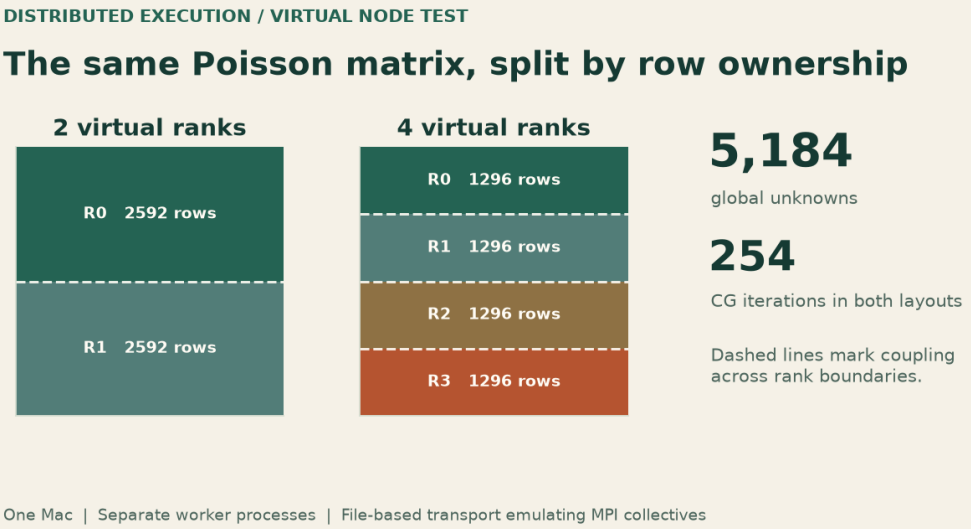
Set the worker's CPU thread budget before submitting the study. run_options contains the inputs to study.run.

```
from onesim.compute import ComputeResources, LocalCompute
from onesim.compute.scheduling.submission import submit

target = LocalCompute(resources=ComputeResources(threads_per_rank=4))
job = submit(study, compute=target, **run_options)
```

One solve across multiple worker processes

A 72 by 72 Poisson system exercises both the OneSimulation task map and a collective reference with rank-owned matrix rows.



Each virtual rank is a separate worker process on the same Mac. Boundary rows couple to columns owned by another rank.

Execution path	Ranks	Rel. residual	Rel. error vs serial
OneSimulation task map	2	8.63e-11	3.97e-11
OneSimulation task map	4	8.63e-11	3.97e-11
Rank-local collective reference	2	8.63e-11	3.97e-11
Rank-local collective reference	4	8.63e-11	3.97e-11
Native PETSc / MPIUNI	1	8.63e-11	3.97e-11

Worker evidence The task map distributed 254 matrix-vector products. A deliberate exception on rank 1 reached the coordinator and the job reported failure. The collective reference used a separate CG implementation with local CSR rows and global reductions.

API IN PRACTICE

Map the owned row intervals within a worker operation. Each callback returns its block contribution for assembly by the coordinator.

```
from onesim.compute.resources.compute_context import current_context

context = current_context()
blocks = context.map(compute_owned_rows, row_intervals)
```

Bind the communicator and the owned rows

For a distributed solve, OneSimulation binds the compute target and matrix ownership. PETSc supplies the collective sparse assembly and Krylov solver.

Route	Execution model	Verification in this atlas
Task mapping	The coordinator dispatches ordered work to MPI ranks.	Two and four local workers with emulated transport.
Collective PETSc	Each rank supplies its CSR rows and local right-hand side on the same communicator.	Native PETSc 3.25.3 in MPIUNI with one rank. Two- and four-rank arithmetic checked with the collective reference.
Slurm	The launcher requests nodes, ranks and threads, then starts the worker through srun.	Source deployment path. Native multi-node execution and scaling were outside these measurements.

API IN PRACTICE

Inside a collective worker, bind the rank's DistributedCsrShard and local right-hand side to the communicator supplied by the launch.

```
from onesim.numerics.linear.selection.execution.backends.petsc.ksp_linear_solver import (
    PetscKspLinearSolver,
)

report = PetscKspLinearSolver().solve_distributed(
    shard, local_rhs, configuration, comm=context.communicator,
)
```

API IN PRACTICE

Select a site-specific shared work directory and the worker factory for the collective study. The declaration below requests two nodes.

```
from onesim.compute import ComputeResources, SlurmCluster
from onesim.compute.scheduling.submission import submit

resources = ComputeResources(nodes=2, ranks_per_node=2, threads_per_rank=4)
target = SlurmCluster(shared_path, resources=resources, mpi_mode="collective")
job = submit(compute=target, worker_factory=bind_rank_study)
```

> **Distribution scope** This chapter covers the OneSimulation repository. Cluster deployment is outside the qualified scope of the Gumroad Runtime Alpha package.

Build a model

The publication audit inspected the nine scientific API roots and 76 direct subpackages of the current checkout.

onesim namespace	Direct source subpackages	Find it in the atlas
cad	d1, d2, d3, exchange, geometry, operations, selection	CAD and meshing
meshing	adaptation, boundaries, exchange, generation, meshes, partitioning, quality, refinement	CAD and meshing, Discretization
materials	electromagnetic, fluids, gases, library, mixtures, properties, solids, thermal, transport	CAD and meshing, Physics atlas
physics	acoustics, biophysics, chemistry, electromagnetics, electrostatics, geotechnics, heat_transfer, hydrodynamics, magnetohydrodynamics, magnetostatics, multibody, multiphysics, optics, plasma, solid_mechanics, systems	Physics atlas, Model and service catalog, OneSimulation results

Read the inventory The companion namespace-inventory.json preserves the inspected export names by import path. An importable namespace can contain specialized provider or extension interfaces even when it declares no top-level exports.

Evidence levels Source API coverage, recorded studies, fresh source runs and the qualified Alpha download answer different questions. Match the model, boundary data, space, integrator and provider for each intended study.

Select a solve and inspect its results

The publication audit inspected the nine scientific API roots and 76 direct subpackages of the current checkout.

onesim namespace	Direct source subpackages	Find it in the atlas
numerics	coupled_systems, discretization, fdm, fem, field_transfer, fvm, linear, nonlinear, time	Discretization, Time integration, Solvers
simulation	checkpoints, composition, execution, field_connections, field_constraints, initial_conditions, problem_definition, state, studies	Notebook walkthrough, Time integration
compute	cpu, distributed, gpu, resources, scheduling	Parallel computing
results	assessment, comparisons, derived, fields, integration, sampling, storage	Notebook walkthrough / result queries and storage
visualization	animation, fields, geometry, meshes, networks, tables	Notebook walkthrough / storage and views

> **Read the inventory** The companion namespace-inventory.json preserves the inspected export names by import path. An importable namespace can contain specialized provider or extension interfaces even when it declares no top-level exports.

> **Evidence levels** Source API coverage, recorded studies, fresh source runs and the qualified Alpha download answer different questions. Match the model, boundary data, space, integrator and provider for each intended study.

Physics API inventory

Names below are public exports of the inspected package snapshot. Provider and result classes appear alongside model declarations.

physics.acoustics PressureDrivenMode, RadiatingMode, UltrasonicAcoustics, LinearAcoustics, AcousticsConfiguration, AcousticsPublicApi, AcousticsProvider, AcousticsResult, AcousticsSpatialOperator, AcousticsState, StructuralAcousticsConfiguration, StructuralAcousticsPublicApi, StructuralAcousticsProvider, StructuralAcousticsResult, StructuralAcousticsState, UltrasoundConfiguration, UltrasoundPublicApi, UltrasoundProvider, UltrasoundResult, UltrasoundState, VibroacousticsConfiguration, VibroacousticsPublicApi, VibroacousticsProvider, VibroacousticsResult, VibroacousticsState, AcousticFeatureSpec, AcousticModelKind, AcousticNodeKind, AcousticSystemSpec, AcousticSystemService, AcousticSystemValidation, adapt_acoustics_result, adapt_structural_acoustics_result, adapt_ultrasound_result, adapt_vibroacoustics_result

physics.biophysics OrganClearance, AxialTissue, TissueElement, PassiveMembrane, CompliantVessels, VesselSegment, BioelectricalConfiguration, BioelectricalPublicApi, BioheatProvider, BioheatResult, BiophysicsProviderRegistry, DEFAULT_BIOPHYSICS_PROVIDER_REGISTRY, CirculatoryConfiguration, CirculatoryPublicApi, OrganConfiguration, OrganPublicApi, TissueConfiguration, TissuePublicApi, adapt_bioelectrical_result, adapt_bioheat_result, adapt_biophysics_result, adapt_circulatory_result, adapt_organ_result, adapt_tissue_result

physics.chemistry TransportSpecies, SpeciesTransport, ElectrochemicalReaction, BatteryChemistry, LumpedElectrode, LumpedBattery, ReactionKinetics, Chemistry, ChemistryKinetics, ChemistryEvaluation, ReactionNetworkIR, SpeciesIR, ReactionIR, ChemistryJacobian

Physics API inventory

Names below are public exports of the inspected package snapshot. Provider and result classes appear alongside model declarations.

physics.electromagnetics FrequencyDomainMaxwell, WaveguidePort, CircularWaveguideMode, PerfectElectricConductor, ImpedanceBoundary, WaveguideModeBasis, SeriesRLC, AveragedBuckConverter, DCMotor, MagneticDiffusion, cold_electron_permittivity_property, operators

physics.electrostatics NormalElectricDisplacement, PoissonEquation, FixedPotential, ZeroNormalElectricDisplacement

physics.geotechnics GroundwaterFlow, GroundwaterConfiguration, GroundwaterPublicApi, GroundwaterProvider, GroundwaterResult, PoromechanicsConfiguration, PoromechanicsPublicApi, PoromechanicsProvider, PoromechanicalState, PoromechanicalResult, ReservoirConfiguration, ReservoirPublicApi, ReservoirProvider, ReservoirState, ReservoirResult, adapt_geotechnics_result, adapt_groundwater_result, adapt_poromechanical_result, adapt_reservoir_result, adapt_rock_mechanics_result, adapt_soil_mechanics_result, RockMechanicsConfiguration, RockMechanicsPublicApi, RockMechanicsProvider, RockMechanicsState, RockMechanicsResult, SoilMechanicsConfiguration, SoilMechanicsPublicApi, SoilMechanicsProvider, SoilMechanicsState, SoilMechanicsResult, GeotechnicsProviderRegistry, DEFAULT_GEOTECHNICS_PROVIDER_REGISTRY, BiotConsolidation, TerzaghiConsolidation, PrescribedPorePressure, PoreFluidFlux, DrainedBoundary, ImpermeableBoundary, BuckleyLeverettFlow, QuadraticCoreyFractionalFlow, SaturationInlet, SaturationOutflow

Physics API inventory

Names below are public exports of the inspected package snapshot. Provider and result classes appear alongside model declarations.

physics.heat_transfer HeatFlux, HeatConduction, FixedTemperature, ConvectiveHeatFlux, RadiationCooling, ThermalGreyDiffuseRadiationBoundary, PhaseChangeHeating, ThermalPhaseChangeLawSet, HeatAdvectionDiffusion, advanced_thermal, configuration, errors, heat_conduction, heat_transport, public_api, validation_report, ConvectiveHeatTransfer, RadiativeHeatFlux

physics.hydrodynamics IncompressibleNavierStokes, RotatingWall, NoSlipWall, FixedVelocity, FlowTraction, PressureTraction, CompressibleEuler, EulerInvariantDomain, TransmissiveBoundary, IdealGasEquationOfState, ShallowWater, ShallowWaterInvariantDomain, PhaseAdvection, PhaseFractionInvariantDomain, ReactiveMixtureTransport, OldroydBShear, ViscoelasticStressLaw, DarcyPressureFlow, DarcyPressureBoundary, DarcyFluxBoundary, CahnHilliard, DoubleWellFreeEnergy, PhaseFlux, CompositionGradient, TopographicShallowWater, WetDryShallowWaterDomain, CahnHilliard, DoubleWellFreeEnergy, PhaseFlux, CompositionGradient

physics.magnetohydrodynamics TransverseMHDWaves, MAGNETOHYDRODYNAMICS, MagnetohydrodynamicPublicApi, adapt_ambipolar_diffusion_result, adapt_built_in_mhd_result, adapt_cgl_mhd_result, adapt_drift_diffusion_result, adapt_electromagnetic_plasma_result, adapt_electrostatic_plasma_result, adapt_glm_mhd_result, adapt_hall_mhd_result, adapt_mhd_2d_result, adapt_mhd_3d_result, adapt_mhd_result, adapt_multi_fluid_mhd_result, adapt_multiphase_mhd_result, adapt_porous_mhd_result, adapt_quasistatic_mhd_result, adapt_radiation_exchange_result, adapt_radiation_mhd_result, adapt_reactive_mhd_result, adapt_reactive_source_result, adapt_relativistic_mhd_result, adapt_two_fluid_mhd_result, TwoFluidExchange, InductionlessMomentum, InductionlessNoSlipWall, RadiationMatterExchange

Physics API inventory

Names below are public exports of the inspected package snapshot. Provider and result classes appear alongside model declarations.

physics.magnetostatics Magnetostatics, MagneticVectorPotential, CoulombGauge, SolenoidCurrent, MagneticInsulation, MagneticPotentialBoundary, MagneticFieldBoundary, MagneticReluctanceBoundary

physics.multibody LinearJointSystem, DrivenFourBar, TorsionalDrivetrain, EulerTop, SimplePendulum, DrivetrainConfiguration, DrivetrainPublicApi, DrivetrainProvider, DrivetrainState, DrivetrainResult, MultibodyProviderRegistry, DEFAULT_MULTIBODY_PROVIDER_REGISTRY, JointSystemsConfiguration, JointSystemsPublicApi, JointSystemProvider, JointSystemState, JointSystemResult, KinematicsConfiguration, KinematicsPublicApi, KinematicsProvider, KinematicsState, KinematicsResult, MechanismsConfiguration, MechanismsPublicApi, MechanismProvider, MechanismState, MechanismResult, RigidBodyDynamicsConfiguration, RigidBodyDynamicsPublicApi, RigidBodyDynamicsProvider, RigidBodyState, RigidBodyDynamicsResult, FlexibleBodyProvider, FlexibleBodyResult, PendulumProvider, PendulumResult, adapt_drivetrain_result, adapt_flexible_body_result, adapt_joint_system_result, adapt_kinematics_result, adapt_mechanism_result, adapt_multibody_result, adapt_pendulum_result, adapt_rigid_body_dynamics_result

physics.multiphysics ElectroThermalConfiguration, ElectroThermalPublicApi, ElectroThermalProvider, ElectroThermalState, ElectroThermalResult, MultiphysicsProviderRegistry, DEFAULT_MULTIPHYSICS_PROVIDER_REGISTRY, FluidStructureConfiguration, FluidStructurePublicApi, FluidStructureProvider, FluidStructureState, FluidStructureResult, GeneralCoupledSubsystemsConfiguration, GeneralCoupledSubsystemsPublicApi, GeneralCoupledSubsystemsProvider, GeneralCoupledSubsystemsState, GeneralCoupledSubsystemsResult, MagnetoMechanicsConfiguration, MagnetoMechanicsPublicApi, MagnetoMechanicsProvider, MagnetoMechanicsState, MagnetoMechanicsResult, ReactiveCouplingConfiguration, ReactiveCouplingPublicApi, ReactiveCouplingProvider, ReactiveCouplingState, ReactiveCouplingResult, ThermoMechanicsConfiguration, ThermoMechanicsPublicApi, ThermoMechanicsProvider, ThermoMechanicsState, ThermoMechanicsResult, MULTIPHYSICS, NUMERICS_BINDING, adapt_electro_thermal_result, adapt_fluid_structure_result, adapt_general_coupled_subsystems_result, adapt_magneto_mechanics_result, adapt_reactive_coupling_result, adapt_thermo_mechanics_result, HeatedResistor, FluidColumnPiston, VoiceCoil, ThermalExchange, AdiabaticReaction

Physics API inventory

Names below are public exports of the inspected package snapshot. Provider and result classes appear alongside model declarations.

physics.optics ComplexParaxialEnvelope, ParaxialEnvelope, PhotonEnvelope, OpticsConfiguration, OpticsPublicApi, OpticsProvider, OpticsResult, OpticsState, PhotonicsConfiguration, PhotonicsPublicApi, PhotonicsProvider, PhotonicsResult, PhotonicsState, adapt_optics_result, adapt_photonics_result

physics.plasma NativeEcrPotentialGauge, NativeEcrPlasma, EcrExcitation, NativeEcrDiscretization, NativeEcrStateTransfer, MagnetizedColdPlasma, DriftDiffusionPlasma, MeanElectronEnergyEquation, MagnetizedElectronTransport, IonMobilityModel, ElectronCollisionModel, Maxwellian, AbsorbingPlasmaWall, NoParticleFlux, PlasmaTransportCoefficients, PlasmaWallFlux, create_plasma_factories, create_plasma_runtime, ChargedParticle, UniformGuidingCenter, ColdPlasmaOscillation, FreeStreamingWaterbag, IsothermalElectronWaterbag, operators

physics.solid_mechanics Traction, LinearElasticity, IsotropicThermalExpansion, FixedDisplacement, TractionFree, StructuralOscillator, MaxwellRelaxation, FatigueCrackGrowth, ParisLaw, HertzImpact, HertzContactLaw, BodyForce, CohesiveOpening, BilinearCohesiveLaw

Physics API inventory

Names below are public exports of the inspected package snapshot. Provider and result classes appear alongside model declarations.

> **physics.systems** ControlSignalFlowConfiguration, ControlSignalFlowPublicApi, ControlSignalFlowProvider, ControlSignalFlowResult, ControlSignalFlowState, SignalBlock, HydraulicBranch, HydraulicNode, HydraulicSystemsConfiguration, HydraulicSystemsPublicApi, HydraulicSystemsProvider, HydraulicSystemsResult, HydraulicSystemsState, PneumaticBranch, PneumaticChamber, PneumaticSystemsConfiguration, PneumaticSystemsPublicApi, PneumaticSystemsProvider, PneumaticSystemsResult, PneumaticSystemsState, ProcessModelsConfiguration, ProcessModelsPublicApi, ProcessModelsProvider, ProcessModelsResult, ProcessModelsState, ProcessStream, ProcessUnit, ThermalLink, ThermalNetworksConfiguration, ThermalNetworksPublicApi, ThermalNetworksProvider, ThermalNetworksResult, ThermalNetworksState, ThermalNode, ZeroDOneDModelsConfiguration, ZeroDOneDModelsPublicApi, ZeroDOneDModelsProvider, ZeroDOneDResult, ZeroDOneDState, ElectricalBranch, ElectricalNetworksConfiguration, ElectricalNetworksProvider, ElectricalNetworksPublicApi, ElectricalNetworksResult, ElectricalNode, FieldSystemCouplingProvider, FieldSystemCouplingResult, CoupledSystemModelSpec, FieldSystemCouplingSpec, SystemComponentKind, SystemComponentSpec, SystemConnectionKind, SystemConnectionSpec, SystemModelKind, SystemsProviderRegistry, DEFAULT_SYSTEMS_PROVIDER_REGISTRY, adapt_control_signal_flow_result, adapt_electrical_networks_result, adapt_field_system_coupling_result, adapt_hydraulic_systems_result, adapt_pneumatic_systems_result, adapt_process_models_result, adapt_systems_result, adapt_thermal_networks_result, adapt_zero_d_one_d_result, HydraulicAccumulator, ThermalCapacitance, ResistorCapacitor, FirstOrderControl, WellMixedTank, SealedGasChamber

Numerical declarations

Selections, providers and data objects compose through typed declarations. A compatible model binding determines executable combinations.

numerics.time TimeEventDefinition, TimeMethod, ClassicalRungeKutta4, ForwardEuler, BackwardEuler, BDF, AdaptiveTimeStep, TimeStepLimits, TimeExecutionBinding

numerics.linear GMRES, RightGMRES, FlexibleGMRES, ConjugateGradient, SparseDirect, ComplexSparseDirect, ILU, AlgebraicMultigrid, BlockPreconditioner, SchurPreconditioner

numerics.nonlinear Newton, DampedNewton, TrustRegionNewton, QuasiNewton, FixedPoint, JFNK, JacobianFreeNewtonKrylov

numerics.fdm FiniteDifference, CentralDifference, CentralLaplacian, PolynomialDifference, PolynomialLaplacian, SummationByPartsDifference, HydrostaticFluxDifference, BoundaryFluxDifference, GlobalLaxFriedrichsSplit, WENO5Flux, WENOFluxDifference

Numerical declarations

Selections, providers and data objects compose through typed declarations. A compatible model binding determines executable combinations.

numerics.fvm MonotoneUpwind, FiniteVolume, Rusanov, LinearUpwind, CentralDifference, LeastSquaresGradient, RhieChow, ScharfetterGummel, ImplicitLocalSources, DifferentiableSamples, MimeticCurl, TwoPointDiffusion, MUSCL, ConvexInvariantDomain, MinmodLimiter, MonotonizedCentralLimiter, VanLeerLimiter, HydrostaticReconstruction, WetDryActiveSet, SuperbeeLimiter, VenkatakrishnanLimiter, BarthJespersenLimiter, TvbLimiter, BoundaryLeastSquaresGradient, CentralElasticStress, WetDryTraceScaling, Eno2Reconstructor, Eno3Reconstructor, Weno3Reconstructor, Weno5Reconstructor, CellGradientReconstruction, FiniteIntegration, WhitneyHodge, OrientedCellComplex, OrthogonalHodge

numerics.fem HcurlNitsche, ConstraintAugmentation, FiniteElement, Lagrange, DiscontinuousLagrange, Nedelec, ConsistentMassStability, SUPG, GLS, CIP, DGPolynomialLimiter, StateLimiterPort, ScalarInterval, DGJumpPenalty, SymmetricInteriorPenalty, InvariantDomainGraphViscosity, RowSumLumping, OrderedFields, TaylorHood, GradDiv, MixedMomentumContribution, Nitsche, RaviartThomas, MixedDiffusionElement, BrezziDouglasMarini, GalerkinDiffusion, VolumeCIP, InterpolatedFluxDivergence, PSPG, MixedContinuityContribution, EqualOrderVelocityPressure, HydrostaticGraphViscosity, GalerkinAdvection, DissipativeMassStability

numerics.discretization count, Numerics, NumericalExecutionPlan, NumericalIR, NodalInterpolation, CellAverageProjection, L2Projection, LumpedParameters, OperatorConditions, OperatorFacetTopology, EvaluationCacheControl, NumericalCompatibilityError, MethodProviderOffer, evaluate_method_provider, CapabilitySet, RequirementSet

numerics Numerics, NumericalIR, NumericalExecutionPlan, fem, fvm, fdm, linear, nonlinear, time, discretization, field_transfer, coupled_systems

Numerical declarations

Selections, providers and data objects compose through typed declarations. A compatible model binding determines executable combinations.

numerics.coupled_systems AlgebraicClosure, MonolithicCoupling, PartitionedCoupling, SequentialCoupling, QuasiStationaryCoupling, PISO, CoupledVelocityPressure

numerics.field_transfer GlobalBroadcastTransfer, SpatialAverageTransfer, GlobalValueTransfer, FiniteElementEvaluation, CellToQuadratureProjection, FieldInterpolation, ElectricFieldToFaces, PeriodicPotentialTrace, ConservativeVolumeTransfer, operators

CAD and mesh exports

Selections, providers and data objects compose through typed declarations. A compatible model binding determines executable combinations.

> **cad.d1** Line

> **cad.d2** Circle, Rectangle, Plane

> **cad.d3** QuadraticTetrahedron, QuadraticHexahedron, TrilinearHexahedron, Box, Cylinder, AnnularCylinder

> **cad.exchange** Subpackage declarations and backend namespaces.

CAD and mesh exports

Selections, providers and data objects compose through typed declarations. A compatible model binding determines executable combinations.

> **cad.operations** FeatureExecutionResult, FeatureHistoryExecutor, CadPlacementService, JointDatumFactory, EdgeHealingCandidate, EdgeHealingProposal, approve_edge_healing, propose_edge_healing, SelectionQuery, SelectionQueryResult, resolve_selection

> **cad** CadGeometryBuilder, d1, d2, d3, geometry, operations, selection, exchange

> **cad.geometry** LumpedDomain, Body, Edge, Face, Geometry, Topology, Vertex

> **cad.selection** Subpackage declarations and backend namespaces.

CAD and mesh exports

Selections, providers and data objects compose through typed declarations. A compatible model binding determines executable combinations.

> **meshing** NodalMeshMotion, AffineMeshMotion, PrescribedMovingMesh, StructuredAnnularHexahedralMesher, QuadraticTetrahedralMesher, QuadraticHexahedralMesher, TrilinearHexahedralMesher, StructuredHexahedralMesher, LumpedMesh, StructuredTetrahedralMesher, TetrahedralMesher, LocalRefinement, PeriodicPair, StructuredCylindricalMesher, StructuredLineMesher, NonuniformLineMesher, meshes, generation, refinement, boundaries, adaptation, partitioning, quality, exchange

Simulation, materials and output

Selections, providers and data objects compose through typed declarations. A compatible model binding determines executable combinations.

simulation FieldThreshold, Problem, Simulation, SimulationValidation, SimulationDependencies, SimulationCompiler, ExecutionPort, ModelDeclaration, ModelRuntimeBinder, NativeModelOperator, PhysicsModel, PhysicalEquation, FieldDefinition, FieldRef, ExecutionOptions, SimulationIssue, ModelCapability, problem_definition, initial_conditions, field_constraints, field_connections, composition, studies, execution, state, checkpoints

materials IsotropicSolid, NewtonianFluid, NeutralGas, VacuumPermittivity, VacuumPermeability, PropertyLaw, fluids, solids, gases, electromagnetic, thermal, transport, mixtures, properties, library

results SimulationResult, SolvedField, ResultFieldBinding, SimulationResultSet, ResultFieldData, fields, sampling, integration, derived, storage, comparisons, assessment

compute CPUBackend, ComputeResources, LocalCompute, SlurmCluster, cpu, gpu, distributed, resources, scheduling

Simulation, materials and output

Selections, providers and data objects compose through typed declarations. A compatible model binding determines executable combinations.

visualization ALL_RESULT_VISUALIZATION_TYPES, ANIMATION_AXES, ANIMATION_EXPORT_FORMATS, AnimationKeyframe, CAMERAS, COMPLEX_REPRESENTATIONS, CitationRecord, DisplayLayer, FIELD_SLICE_VISUALIZATION_TYPES, FieldSliceVisualizationArtifact, InteractionCommand, InteractiveOptions, NetworkArc, NetworkLayout, NodePlacement, PALETTE_DARK, PALETTE_LIGHT, RESULT_VISUALIZATION_TYPES, ReactionNetwork, ReactionNode, ReadabilityReport, RenderOptions, ResultAnimationArtifact, ResultTemplateArtifact, ResultVisualizationArtifact, SCALAR_SCALES, SelectionArtifact, SpeciesNode, Table, ViewModel, analytic_isosurface, animation, animation_axis_values, animation_frame_indices, balance_contributions, build_interactive_page, build_reaction_network, compute_layout, contact_flux_plot, create_mesh_sequence_widget, create_result_explorer, diagnostic_plot, diagnostic_slice_plot, dispatch, explore, field, geometry, geometry_preview, geometry_widget, line_plot, mesh, mesh_interior, mesh_preview, mesh_sequence_line_plot, mesh_sequence_plot, mesh_sequence_series, mesh_sequence_series_plot, mesh_sequence_slice_plot, mesh_sequence_widget, mesh_widget, network_from_species, network_payload, persist_selection, plot, plot_contact_flux, plot_line_profile, plot_mesh_sequence_series, rate_law_formulas, reaction_rate_expression, reaction_table, readability_report, render_reaction_network, result, result_animation, result_scene, result_widget, running_pages, save_reaction_network, scene, selected_entity_ids, serve_interactive_page, slice_plot, solver_diagnostics, source_table, species_table, stop_interactive_page, time_series, time_series_plot, widget, meshes, fields, networks, tables

Sources and reproducibility

The source package contains the editorial data, LaTeX chapters, generated figures, notebook and benchmark scripts.

Physics references physics_cases.py describes setup, equations and interpretation. reference_fields.py and make_figures.py generate the independent plots.

Numerical benchmarks numerical_exhibits.py defines the three-way Poisson comparison, time-step studies, Krylov comparison and nonlinear residual examples. numerical-metrics.json records computed values.

Parallel execution hpc/ contains the original measurements and the source tests for the threaded kernel, local study and virtual ranks. hpc_exhibits.py redraws the figures.

Coverage audit coverage_exhibits.py redraws the recorded advanced studies and verifies radiation, native field queries, thermoelasticity, conservative remapping and BDF2 restart. coverage-metrics.json records the checks.

OneSimulation exhibits runtime_exhibits.py runs the oscillator, four 1D scalar comparisons, a 3D stationary heat case and CAD examples. Recorded figures use the repository marketing samples for the cantilever, coil and circulating heat model.

Catalog sources api-inventory.json records package exports. time-catalog.json is generated from the executable time method catalog in onesim.numerics.time. Repository test basenames in the atlas identify nearby checks and examples.

Release scope See the Gumroad product description for the Runtime distribution, price, system requirements and license terms. The repository atlas documents a broader source surface.

OneSimulation Runtime Alpha on Gumroad