

Lab4IQM Studio

User guide, version 1.0

Armando Jose Rodado Amaris

Lab for Integrated Quantitative Modelling · lab4iqm.com

October 2026

Abstract

Lab4IQM Studio is a workbench for mathematics, statistics and quantitative modelling inside Blender. You type a model in a short command language, or write it in Python with NumPy, and see it as an interactive 2D or 3D scene with sliders, animation and simulation. Figures go to papers as vector PDF, to 3D printers as STL and to the web as glTF, and LaTeX documents can be written and compiled in the same window, with figures that follow your models. Every figure in this guide was made by Lab4IQM's own code.

Contents

1	Installing	1
2	First steps	1
3	Drawing	2
4	Looks: colours, finishes, contours, light	3
5	Simulations	3
6	Data and statistics	4
7	Modules	5
8	Python scripts	6
9	Figures and documents	8
10	Where next	8
A	Command reference	8
A.1	Draw shapes and graphs	8
A.2	Change what is drawn	10
A.3	Views, figures and the look	11

A.4	Simulations	13
A.5	Data and statistics	14
A.6	Files, scripts and export	15
A.7	LaTeX documents	17
A.8	Help	18
A.9	Maths in expressions	19

1 Installing

1. In Blender (4.2 or newer), choose *Edit* → *Preferences* → *Get Extensions*, then the menu at the top right, *Install from Disk*, and pick **lab4iqm-1.0.0.zip**. Accept the permissions notice (network, only to download the Tectonic L^AT_EX engine when you ask for it; files, to save what you name).
2. Optional, for a dedicated application: install the application template **Lab4IQM-Studio-template-1.0.0.zip** with *File* → *Defaults* → *Install Application Template*, and make a launcher on a Mac with

```
osacompile -o ~/Desktop/"Lab4IQM Studio.app" \
-e 'do shell script "open -n -a Blender --args --app-template Lab4IQM_Studio"'
```

The launcher opens Blender with the Lab4IQM splash screen, its examples and a clean scene.

Press **N** over the 3D view to show the sidebar; the *Lab4IQM* tab holds the console, the object panel with sliders, the view panel and the examples.

2 First steps

Type a command in the console and press Enter:

```
plot(z=x^2-y^2)
plot(z=sin(a*sqrt(x^2+y^2))/sqrt(x^2+y^2+0.1), a=3, n=200)
```

Every name that is not a variable (here **a**) becomes a **slider** in the object panel. Objects are named **s1**, **c1**, ... and can be changed by name: **set(s2, a=5)**, **animate(s2, a=1..10, seconds=8)**, **color(s2, cmap=plasma)**, **delete(s1)**. Several commands can be pasted at once (the paste button runs the copied lines), separated by new lines or ;.

- **view2d()** looks straight down with rotation locked (graph paper); **view3d()** frees it again. **axes()** draws axes with ticks around everything.
- **hold(off)** makes each new shape replace the previous ones; **replace=s1** on a shape command swaps out one object.
- **help()** lists the commands by topic, **help(draw)** one topic and **help(plot)** one command with examples. The Examples panel runs ready-made models.

Your own functions. Type a definition and use the name wherever a formula goes:

```

f(x) = x^2 + sin(x)      (or simply f = x^2 + sin(x))
f(2)                    its value: sin(2) + 4 = 4.909...
plot(f)                  derivative(f=f)          taylor(f=f, n=5)
g(x, y) = x*y + 1        plot(z=g)                h(x) = f(2*x) + 1

```

Without brackets the variables are taken from the formula (x, y, z ; or t ; or u, v). `functions()` lists the definitions and `functions(forget=f)` removes one; the `.blend` file keeps them. Typing a definition again replaces it; a function built from another one keeps the version it was built from. Variables ($x, y, t, \dots$), constants and built-in functions cannot be redefined, and `a = 3` gives a named number instead of a slider.

3 Drawing

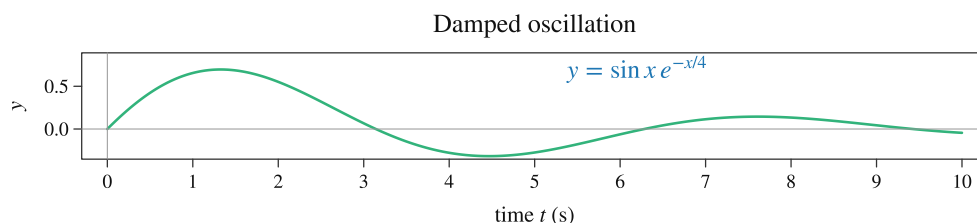
<code>plot(z=f(x,y)), plot(y=f(x))</code>	graphs of functions
<code>param(r=(x(u,v), y(u,v), z(u,v)))</code>	parametric surfaces and curves
<code>implicit(x^2+y^2+z^2 = 1)</code>	closed surfaces $F(x, y, z) = c$ (3D printable)
<code>field(F=(P, Q, R))</code>	vector fields as arrows
<code>label("\$E = mc^2\$", at=(1, 2))</code>	text and L ^A T _E X maths in the figure

Labels are typeset by Lab4IQM itself with the STIX fonts, so no T_EX installation is needed and they look the same in Blender, SVG and PDF. Sub- and superscripts, fractions, roots, sums and integrals with limits, growing brackets, Greek letters, accents and $\mathbb{R}$ all work; `$$...$$` gives display style. Axes take a title and names:

```

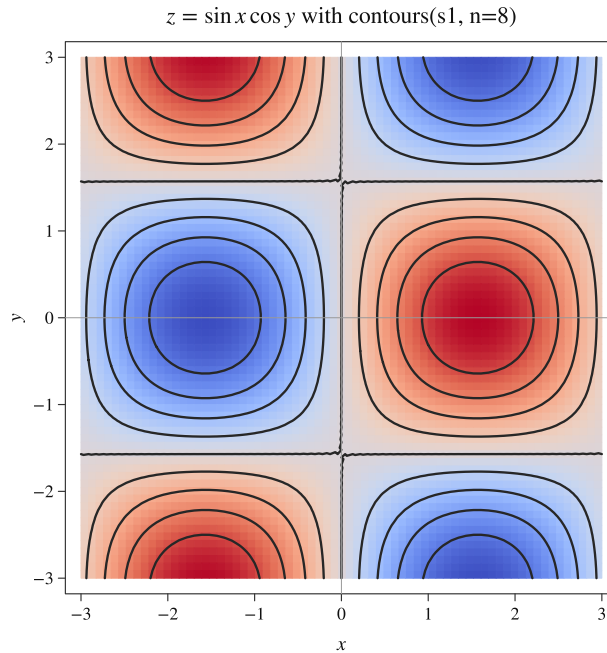
plot(y=sin(x)*exp(-x/4), x=0..10)
label("$y = \sin x\, e^{-x/4}$", at=(6.2, 0.6), color=blue)
axes(title="Damped oscillation", xlabel="time $t$ (s)", ylabel="$y$")
view2d()
export("damped.pdf")

```



4 Looks: colours, finishes, contours, light

`color(s1, cmap=coolwarm, by=x^2+y^2)` colours by any expression; `material(s1, glossy)` chooses a finish (matte, satin, glossy, ceramic, metal, glass); `lights(studio)`, `lights(soft)` or `lights(dramatic)` place lamps around the figure. `contours(s1, n=8)` draws level curves that follow sliders and animations, and seen from above they give a contour map:



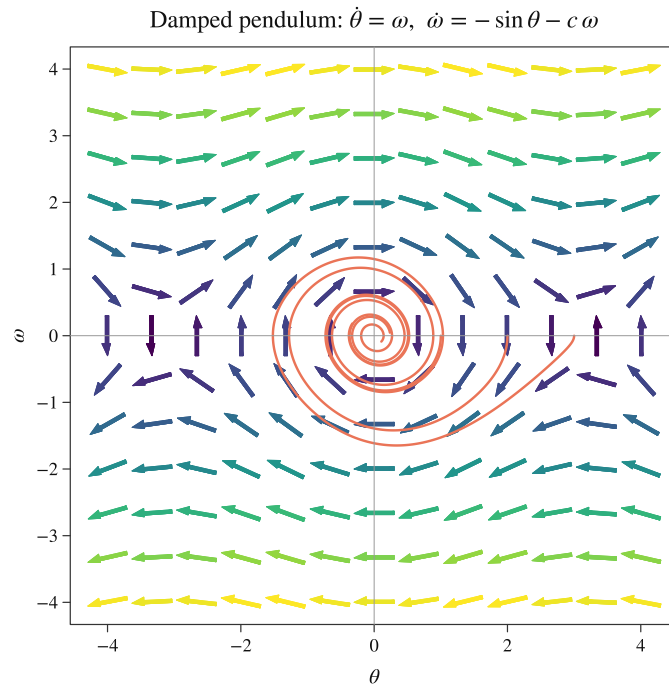
`render("figure.png")` makes a print-quality picture of the view with Cycles (real light, shadows, reflections and glass); `export("view.png")` is the instant alternative. `background("map.png", x=0..10)` puts your own picture under a 2D graph in data coordinates, and `background(color="#202830")` or `background(sky="studio.hdr")` sets the world around a 3D scene.

5 Simulations

<code>ode(F=(...), start=(...), t=0..20)</code>	ODEs (Runge–Kutta 4), one or many starts
<code>particles(F=(...), n=400)</code>	particles moving in a flow
<code>pde(ut=D*lap(u), u0=..., t=0..2)</code>	diffusion and reaction–diffusion on a grid
<code>pde(utt=c^2*lap(u), ...)</code>	waves

Simulations are computed once and cached; the `time` slider and `play=true` play them back. Parameters are sliders like everywhere else. For example, a damped pendulum in its phase plane:

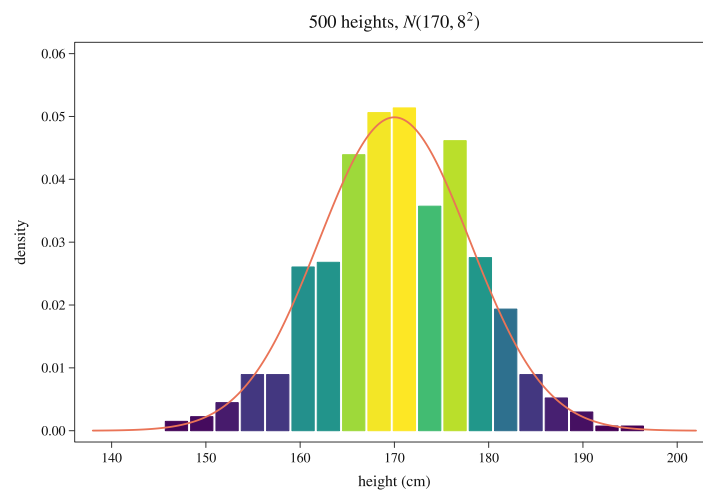
```
field(F=(y, -sin(x)), box=-4..4, normalize=true)
ode(F=(y, -sin(x) - c*y), start=((1,0), (2,0), (3,0)), t=0..20, c=0.2)
view2d()
```



6 Data and statistics

`points("data.csv", x=age, y=dose)` plots data; `hist`, `density`, `fit` and `summary` cover histograms, kernel densities, distributions (normal, uniform, exponential, gamma, beta, lognormal, t , χ^2 , Poisson, binomial, bivariate normal) and least-squares or Levenberg–Marquardt fits. Axes and exported figures are in data units:

```
hist(dist=normal, mu=170, sigma=8, n=500, density=true)
density(dist=normal, mu=170, sigma=8)
axes(xlabel="height (cm)", ylabel="density")
```



7 Modules

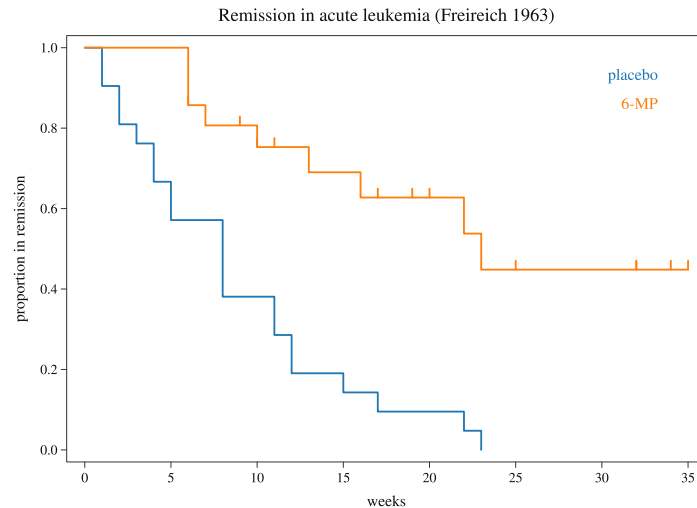
Modules add commands for one subject. Each is a small extension installed like Lab4IQM itself; then, as in Python or R, a file uses it after an import:

```
import biostat
```

The `.blend` file remembers its imports and imports them again when it is opened. `modules()` lists the modules installed and imported, `help(biostat)` the commands of one, and `modules(off=biostat)` stops using one in this file. A module adds a group to the Examples panel and brings example data, read as `"<module>:file.csv"`. The first module is **Biostatistics**:

<code>km(f, time=t, event=d, group=g)</code>	Kaplan–Meier, log-rank test, hazard ratio
<code>roc(f, score=(m1, m2), outcome=y)</code>	ROC curves, AUC (DeLong), best cut-off
<code>compare(f, value=v, group=g)</code>	box plots; t , Mann–Whitney, ANOVA, Kruskal–Wallis
<code>crosstab(f, row=a, col=b)</code>	χ^2 and Fisher tests, odds and risk ratios
<code>dose(f, dose=c, response=r)</code>	four-parameter logistic fit, EC50 or IC50

Here `f` is a CSV file in quotes, such as `"trial.csv"`. The second module, **Mathematical Oncology**, follows Chaplain & Preziosi, *Mathematical Oncology* (Springer, 2025): `growth`, `rates`, `growthfit`, `tumourvolume`, `contact` and `misperception` cover the growth laws of its Chapter 2; `oxygen`, `nutrient`, `spheroid`, `gif`, `glioma`, `clones` and `spread` the diffusion models of Chapter 3; `radius` and `mixture` the mass balance models of Chapter 4; and `nonnecrotic`, `necrotic`, `greenspan`, `chemomech`, `immunebasic`, `immune`, `turing`, `angiogenesis`, `sprouts`, `invasion`, `cells`, `upa`, `adhesion` and `metastasis` cover Chapters 5–11 (spheroids, mechanics, immunity, pre-patterns, angiogenesis, invasion and metastasis). `guide(oncology)` opens a companion that follows the chapters section by section. The third module, **Graph Theory**, follows West, *Introduction to Graph Theory*: `graph(petersen)` draws a graph (in 3D with `d3=true`) and `isomorphic`, `automorphisms`, `euler`, `bipartite`, `cuts`, `cycles`, `cliques`, `decompose`, `degrees`, `graphic`, `deck`, `orient` and `kings` cover its Chapter 1 and its exercises; `graphsaves` writes files for NetworkX. The fourth, **Computational Topology**, follows Edelsbrunner and Harer: `knot`, `link`, `tutte`, `surface`, `intersections`, `simplify`, `complex`, `rips`, `cech`, `delaunay`, `alpha`, `collapse` and `homology` draw knots, surfaces and complexes in 3D and compute their invariants, e.g. `homology(klein, coeff=all)`; `cohomology`, `pairing`, `alexander`, `critical` and `reeb` cover duality and Morse functions; `persistence`, `extended`, `spectral`, `vineyard`, `bottleneck` and `crofton` draw persistence diagrams, barcodes and vineyards and check the stability theorems. The fifth, **Differential Calculus**, follows OpenStax *Calculus Volume 1* on the core's symbolic engine: `function`, `limit`, `continuity`, `derivative`, `sketch`, `optimize`, `lhopital`, `newton`, `antiderivative` and others draw and compute exactly, e.g. `derivative(f=x^3*sin(x))` lists the rules it used; `practice` and `check` generate problems and check students' answers. The sixth, **Integral Calculus**, follows OpenStax *Calculus Volume 2*: `sigma`, `riemann`, `integral`, `average`, `areafunction`, `ftc`, `netchange` and `substitute` shade signed areas, evaluate integrals by the Fundamental Theorem with each rule shown and check them numerically; `between`, `revolve` (disks, washers and shells, drawn in 3D), `arclength`, `work`, `pump` and `centroid` cover the applications, and later chapters add techniques of integration, differential equations, `series`, `taylor`, `parametric`, `polar` and `conic`. The seventh, **Multivariable Calculus**, follows Volume 3 with vectors, lines, planes and quadric surfaces in 3D, space curves, partial derivatives, gradients, extrema, Lagrange multipliers, double and triple integrals, vector fields with the theorems of Green, Stokes and Gauss, and second-order differential equations. Results print in the console as a short report; the plots export like any figure.



```
import biostat
km("biostat:leukemia.csv", time=weeks, event=relapse, group=drug)
group          n  events  median  95% CI
placebo [blue]  21      21      8   (4, 12)
6-MP [orange]  21       9     23  (16, -)
Log-rank test: chi2 = 16.79, df = 1, p < 0.0001
Hazard ratio 6-MP vs placebo: 0.19 (95% CI 0.09 to 0.43)
```

In a script (Cmd+Return), `import biostat` gives the commands as functions, which return the report as text; column names and expressions are given in quotes:

```
import biostat
report = biostat.km("trial.csv", time="months", event="died", group="arm")
print(report)
```

Modules are free and listed at lab4iqm.com; anyone can write one (see `modkit/README.md` in the source).

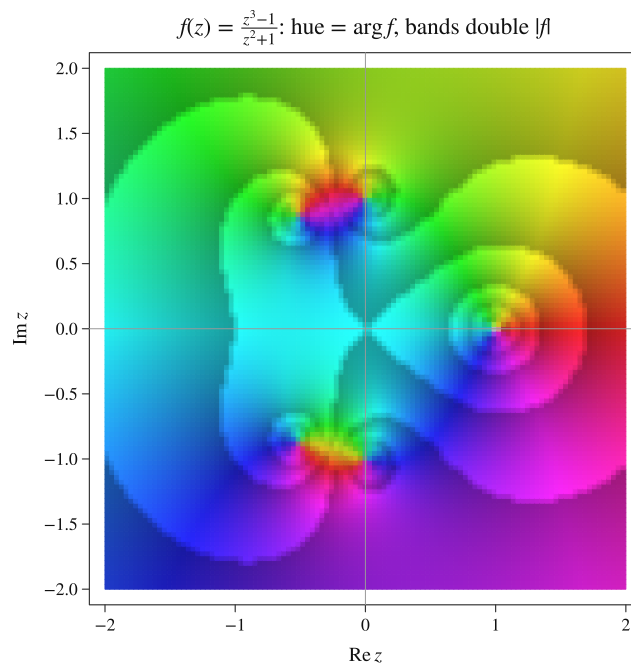
8 Python scripts

`code()` opens a code editor beside the 3D view (`code(window)`: in its own window, for a second screen). Scripts are ordinary Python with NumPy; **Cmd+Return** (Ctrl+Enter) runs them, `print()` output goes to the console and to `output()`, and errors are shown with their line numbers.

```
import numpy as np
import lab4iqm as L

L.clear()
def f(x, y):
    return np.sin(x) * np.cos(y)
L.surface(f, x=(-3, 3), y=(-3, 3), name="wave")
L.contours("wave", n=8)

L.figure("Complex")
L.complex_plot(lambda z: (z**3 - 1) / (z**2 + 1), re=(-2, 2), im=(-2, 2))
```

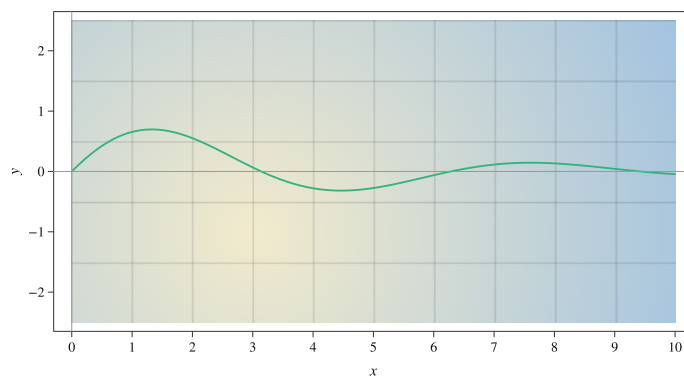


<code>L.surface, L.curve, L.parametric</code>	draw Python functions
<code>L.points(x, y, z)</code>	draw arrays of data
<code>L.complex_plot(f, kind=...)</code>	complex functions: "domain" or "landscape"
<code>L.riemann(p, sheets=2)</code>	Riemann surface of $w^n = p(z)$, with genus
<code>L.riemann(inverse=np.exp)</code>	inverse functions, e.g. $\log z$
<code>L.label, L.axes, L.export</code>	as the console commands
<code>L.run("..."), L.show(...)</code>	any console command; a line in the console

Write $\text{\LaTeX}$ in raw strings, `r"$\alpha$"`, so that Python keeps the backslashes. `name="..."` makes a script safe to run again: the old object is replaced. `save_script("waves.py")` and `open_script("waves.py")` keep scripts as files.

A background can be computed in the script itself, as a NumPy array or a Pillow image:

```
L.background(img, x=(0, 10))           # img: an (h, w, 3) array, row 0 at the top
```



9 Figures and documents

New Project (on the start screen, or the folder button with a plus in the View panel) or `project("paper1")` makes a folder with the `.blend` file and five folders, which you may use or ignore:

paper1/	paper1.blend			
	code/main.py	starter script	figures/	figures and pictures
	latex/main.tex	starter document	models/	STL, glTF
	data/	CSV files		

Inside a project, plain file names go to the folder for their type: `export("fig.pdf")` and `background("map.png")` use `figures/`, `export("shape.stl")` uses `models/`, `points("data.csv")` reads `data/`. Names with a folder part, and full paths, are used as they are. The figures a document includes are written to `figures/` too. `project()` shows the folders. To come back to a project, click its name under **Recent Projects** on the start screen, or use **Open Project...** (start screen or View panel) and choose its folder; this opens its `.blend` file, which brings back the figures, scripts and documents. Opening `paper1.blend` with **Open...** does the same.

Without a project, `folder("~/Documents/maths/paper1")` chooses where a project's files go (scripts, documents, figures, exports); it is saved with the `.blend` file. `export("figure.pdf")` and `export("figure.svg")` write the figure seen from above as vector graphics with L^AT_EX labels.

`doc("paper")` opens a L^AT_EX document in the code editor (a new one starts from a template). **Cmd+Return** compiles it in the background, with an installed T_EX (MacTeX, T_EX Live) or with Tectonic, which `latex(install)` downloads once. Errors appear with their line numbers, and the PDF is shown beside the 3D view (`pdf(window)` for a second screen). Lab4IQM figures enter the document by name:

```
\usepackage{lab4iqm}  
...  
\labfigure[width=0.8\linewidth]{Complex}
```

At every compile the figure tab is exported again: a vector PDF for a 2D graph, a rendered picture for 3D shapes (re-rendered only when the figure changed). Change the model, compile, and the paper follows.

10 Where next

The commands and the L. functions listed in the reference below form the **1.0 interface**: versions 1.x add to it and never remove or rename. Modules that add their own commands — Riemann surfaces and moduli, mathematical oncology, biostatistics — build on it. News, examples and modules: lab4iqm.com.

A Command reference

Every console command, by topic, as `help(topic)` and `help(command)` show it in the app.

A.1 Draw shapes and graphs

`help(draw)`

plot Graph of a function: a surface $z=f(x,y)$ or a curve $y=f(x)$.

```
plot(z=f(x,y), x=a..b, y=c..d, n=100, cmap=viridis)
plot(y=f(x), x=a..b)
plot(z=x^2+y^2)                                surface over x, y in -3..3
plot(z=sin(a*x)*cos(y), a=2)                    'a' becomes a slider in the panel
plot(z=x*y, x=-1..1, y=-2..2, n=200)
plot(y=sin(x), x=0..2*pi)                       curve in the xy-plane (try view2d())
plot(y=tan(x), ylim=-5..5)                       cut off values outside -5..5
x^2 - y^2                                         a bare expression is plotted too
Options: n= grid points per side, cmap= colour map, color= expression in
x, y, z used for colouring, zlim= / ylim= range to keep.
```

param Parametric surface (variables u, v) or curve (variable t).

```
param(r=(x(u,v), y(u,v), z(u,v)), u=a..b, v=c..d)
param(r=(x(t), y(t), z(t)), t=a..b)
param(r=(cos(u)*sin(v), sin(u)*sin(v), cos(v)), u=0..2*pi, v=0..pi)    sphere
param(r=((2+cos(v))*cos(u), (2+cos(v))*sin(u), sin(v)))                 torus
param(r=(cos(t), sin(t), t/5), t=0..6*pi)                             helix
param(r=(cos(3*t), sin(2*t)))                                           plane curve
Default ranges: u, v, t in 0..2*pi. Other names become sliders.
```

implicit Implicit surface $F(x,y,z) = c$, e.g. spheres, tori, level sets.

```
implicit(x^2+y^2+z^2 = 1, box=-2..2, n=60)
implicit(x^2+y^2+z^2 = 1)                                                sphere
implicit((sqrt(x^2+y^2)-R)^2 + z^2 = r^2, R=1.5, r=0.5, box=-2.5..2.5)    torus
implicit(x^4+y^4+z^4 - (x^2+y^2+z^2) = c, c=-0.2)                       a level set; c is a slider
implicit(z = x^2 - y^2)                                                  also works for graphs
A bare expression F means F = 0. The box is -2..2 unless given (box=, or
x=, y=, z= ranges). n= is the grid size per side (default 60, max 120).
If a closed surface does not fit in box=, the box grows automatically
(grow=false turns this off); surfaces that go on for ever keep the box.
x=, y=, z= ranges cut the surface on purpose and are never grown.
Holes or missing parts usually mean the box is too small or n too coarse.
```

field Vector field drawn as arrows, coloured by magnitude.

```
field(F=(P, Q, R), box=-2..2, n=7)
field(F=(P, Q))
field(F=(-y, x, 0))                                                    rotation about the z-axis
field(F=(x, y, z), n=6)                                              a source
field(F=(-y, x - k*y), k=0.3)                                         2D field in the xy-plane (try view2d())
field(F=(y, -sin(x)), box=-4..4, normalize=true)                    pendulum phase portrait
Options: n= arrows per side (3D default 7, 2D default 15), scale= arrow
length factor, normalize=true for equal lengths, box= / x= / y= / z= ranges.
```

label Write text or LaTeX maths in the figure.

```
label("$E = mc^2$", at=(1, 2))
label("Peak $x_0$", at=(0, 1.5), size=0.4, color=red)
label("$f(z) = \frac{z^3-1}{z^2+1}$", at=(0, 2.5))
label("Energy $E=\frac{1}{2}mv^2$", at=(-2, 1), align=left)
label("$\int_0^\infty e^{-x^2}dx = \frac{\sqrt{\pi}}{2}$", at=(0, -2), size=0.35)
```

Text outside `$...$` is plain; inside is LaTeX maths: `^` `_` `\frac` `\sqrt` `\sum` `\int` `\left(` `\right)` Greek letters `\mathbb{R}` `\text{...}` and so on. `$$...$$` gives display style (large fractions and sums with limits).
`at=(x, y)` or `(x, y, z)`; `size=` letter height in scene units (default 0.3);
`align=left|center|right`; `color=black, red, "#3366cc", ...` (default: light in the 3D view, black in exported figures). Labels lie flat in 2D view and stand upright in 3D, like the axis labels, and export to SVG and PDF.

functions List the functions defined in this file, or forget some.

```
functions()
functions(forget=f)
functions(forget=all)
```

Define a function by typing it: `f(x) = x^2 + sin(x)` or simply `f = x^2 + sin(x)` (without brackets the variables are taken from the formula: `x`, `y`, `z`; or `t`; or `u`, `v`).
Two variables: `g(x, y) = x*y + 1`. Then use the name wherever a formula goes:
`plot(f)` `derivative(f=f)` `taylor(f=f, n=5)` `f(2)` (its value)
`h(x) = f(x)^2 + 1` `plot(z=g)` `f` (shows the definition)
Typing a definition again replaces it. The `.blend` file keeps its functions.
Names that are variables (`x`, `y`, `z`, `t`, `u`, `v`, `n`, ...), constants (`pi`, `e`) or built-in functions (`sin`, `exp`, ...) cannot be defined. Letters that are not variables of the function (a in `f(x) = a*x^2`) stay free and become sliders where it is drawn.

A.2 Change what is drawn

```
help(change)
```

set Change the slider values of an object.

```
set(s1, a=2, b=0.5)
```

animate Animate parameters: keyframes from the first value to the second.

```
animate(s1, a=4..20, frames=60, start=1)
animate(s1, a=4..20) a goes from 4 to 20 over 60 frames
animate(s1, a=4..20, seconds=30) ... taking 30 seconds (slow it down at will)
animate(s1, a=4..20, seconds=30, fps=60) smoother: 60 frames per second (default:
scene's, 24)
animate(s1, a=4..20, bounce=true) 4 -> 20 -> 4: a seamless loop
animate(s1, a=0..2*pi, b=1..3, frames=120, start=1)
```

The value changes at a constant rate; `ease=true` gives a gentle start and stop.
The scene's frame range is set to the animation, so playback loops.
Then press Space with the mouse over the viewport to play.

color Recolour a surface with another colour map and/or colouring expression.

```
color(s1, cmap=plasma, by=x^2+y^2)
```

Colour maps: `viridis`, `plasma`, `inferno`, `magma`, `coolwarm`, `terrain`, `gray`, `rainbow`.
`by=` may use `x`, `y`, `z`, `r` (distance from origin) and the object's parameters; `by=z` is the default.

material Give objects a surface finish: matte, satin, glossy, ceramic, metal or glass.

```
material(s1, glossy)
material(glass)
material()
material(s1, glossy)    s1 shiny, with a clear coat
material(glass)         every Lab4IQM object in this figure as coloured glass
material(s1, satin)     back to the default
material()              list the finishes
Colours still come from the colour map. Glass and metal look their best
with lights(studio) and in render() pictures.
```

contours Draw level curves (lines of equal height) on a surface.

```
contours(s1)
contours(s1, n=12)
contours(s1, levels=(0, 0.5, 1))
contours(s1, off)
contours(s1)            about 8 round heights, e.g. 0.2, 0.4, ...
contours(s1, n=15)      more lines
contours(s1, levels=(-1, 0, 1)) exactly these heights
contours(s1, off)       remove them
The lines follow the surface when sliders or animations change it, and
export("fig.svg") draws them: seen from above this is a contour map.
```

hold Keep earlier objects when drawing a new one (on, the default) or replace them (off).

```
hold(on)
hold(off)
hold()
hold(off)              each new plot, field, ... replaces everything drawn before
hold(on)               new shapes are added next to the old ones (default)
hold()                show the current setting
To replace a single object instead, add replace=: field(F=(y, -x, 0), replace=f1)
The Hold button in the Console panel does the same as hold(on) / hold(off).
```

delete Delete the named objects.

```
delete(s1, c2)
```

clear Delete every object made by Lab4IQM.

```
clear()
```

list List the objects in the scene and their parameters.

```
list()
```

A.3 Views, figures and the look

```
help(view)
```

view2d Lock the view: top-down, orthographic, no rotation (pan and zoom only).

```
view2d()
```

view3d Unlock the view and return to a perspective 3D view.

```
view3d()
```

axes Draw axes with ticks and labels around everything in the scene.

```
axes()
axes(ticks=8)
axes(x=-5..5, y=0..10)
axes(title="$f(x)=x^2$", xlabel="$t$ (s)")
axes(off)
```

`axes()` fits the axes to all Lab4IQM objects; run it again after adding or changing objects. Give `x=`, `y=`, `z=` ranges to set them yourself, `ticks=` for roughly how many ticks per axis, and `axes(off)` to remove them. Labels lie flat in 2D view and stand upright in 3D view.

`title=` "...", `xlabel=` "...", `ylabel=` "...", `zlabel=` "..." name the figure and its axes; LaTeX maths works inside `$...$`, e.g. `xlabel="time $t$ (s)"`. `export("fig.svg")` and `export("fig.pdf")` use them too.

lights Light the figure with a set of lamps, sized to what is in it.

```
lights(studio)
lights(soft)
lights(dramatic)
lights(off)
```

`lights(studio)` key, fill and rim lights: clear shapes, soft shadows (the default)
`lights(soft)` large, even lights: good for colour maps
`lights(dramatic)` one low strong light: deep shadows, shows relief
`lights(off)` remove the lamps (back to the plain preview lighting)

The lamps are placed around everything in the figure when you call `lights()`; call it again after adding objects far away. `render()` uses them.

background Put your own picture behind a 2D graph, or set the colour or sky around the 3D scene.

```
background("paper.png", x=-3..3, y=-2..2)
background(color="#202830")
background(sky="studio.hdr")
background(off)
```

`background("map.png")` the picture covers the figure, keeping its
shape
`background("map.png", x=-3..3)` from `x = -3` to `3`; height from the picture's
shape
`background("map.png", x=0..10, y=0..5)` exactly this rectangle (may stretch the
picture)
`background("map.png", opacity=0.5)` half transparent
`background(color="#202830")` plain colour around the scene (also white,
black...)
`background(sky="studio.hdr", strength=1)` a 360-degree picture as the world (lights 3D
scenes)
`background(off)` remove the picture and go back to the usual

look

The picture sits just below $z = 0$ and takes no light, so its colours stay exact. It is saved inside the .blend file and appears in `export("fig.svg")` and `export("fig.pdf")`. From Python, `L.background(array)` takes a NumPy array or a Pillow image, so you can draw the background yourself.

figure Open a figure tab (its own scene and objects), or list the figure tabs.

```
figure("Figure 2")
figure()
figure("Complex")      switch to the figure tab "Complex", making it if needed
figure()               list the figure tabs
```

Each figure tab is a Blender scene, so the scene menu at the top right of the window switches between them. Objects, sliders, axes and the console history belong to their figure.

render A print-quality picture of the 3D view, with real light, shadows and reflections (Cycles).

```
render("picture.png")
render("big.png", width=3000, height=2000, samples=256)
render("picture.png")      the view as you see it, path-traced
render("hi.png", width=3000, height=2000)
render("fig.png", samples=256)    less noise, slower (default 96)
render("clear.png", transparent=true) no background
render("quick.png", engine=eevee) much faster, a little less exact
Uses the lamps from lights(); without them a studio set is used for the
picture only. It can take from seconds to a few minutes; Blender waits
until it is done. export("view.png") is the instant, as-seen alternative.
```

A.4 Simulations

help(simulate)

ode Solve $x' = F(x, t)$ and draw the trajectory (1, 2 or 3 unknowns x, y, z).

```
ode(F=(f, g, h), start=(x0, y0, z0), t=0..T)
ode(F=f(t, x), start=x0)
ode(F=(s*(y-x), x*(r-z)-y, x*y-b*z), start=(1,1,1), t=0..40, s=10, r=28, b=8/3)
    Lorenz
ode(F=(y, -sin(x) - c*y), start=((1,0), (2,0), (3,0)), t=0..20, c=0.2)    damped
    pendulum
ode(F=x*(1-x), start=(0.1, 0.5, 1.5), t=0..8)    logistic growth, x against t
Several starting points: start=((1,1,1), (1,1,1.001)). The 'time' slider
shows the trajectory up to that time; animate(c1, time=0..40, seconds=10)
draws it, or add play=true. Options: n= RK4 steps (default 4000).
With one unknown the curve is drawn as (t, x(t)) in the xy-plane.
```

particles Many particles carried by a flow $x' = F(x, t)$, as an animation.

```
particles(F=(f, g, h), n=300, box=-2..2, t=0..10)
particles(F=(-y, x, 0.2*sin(3*t)), n=400)    swirling cloud
particles(F=(s*(y-x), x*(r-z)-y, x*y-b*z), s=10, r=28, b=8/3, box=-15..15, t=0..5)
particles(F=(y, -sin(x)), box=-3..3, n=500)    pendulum phase plane (2D)
Particles start at random places in the box (seed= picks another random
set). They are coloured by speed. The animation starts at once (play=false
to switch off; seconds= sets its length). Options: steps= RK4 steps (600).
```

pde Solve a PDE on a grid (1D or 2D) and animate the solution u .

```
pde(utt=D*lap(u) + f(u), u0=g(x,y), t=0..T)
pde(utt=c^2*lap(u), u0=..., ut0=...)
pde(ut=D*lap(u), u0=exp(-4*(x^2+y^2)), D=0.5)           heat equation
pde(ut=D*lap(u) + r*u*(1-u), u0=exp(-10*(x^2+y^2)), D=0.05, r=1, box=-5..5, t=0..20)
    Fisher-KPP
pde(utt=c^2*lap(u), u0=exp(-10*(x^2+y^2)), c=1, boundary=zero)  wave equation
pde(ut=D*lap(u), u0=heaviside(1-abs(x)), D=0.2)              1D: no y in u0
Gray-Scott spots (two unknowns u, v; one line):
pde(ut=0.16*lap(u) - u*v^2 + F*(1-u), vt=0.08*lap(v) + u*v^2 - (F+k)*v,
    u0=1-0.5*heaviside(8-hypot(x,y)), v0=0.25*heaviside(8-hypot(x,y)), F=0.04, k=0.06,
    box=-40..40, t=0..4000, show=v, height=10)
Operators: lap(u) Laplacian, ddx(u), ddy(u) first derivatives; formulas may
also use x, y, t. A second unknown v needs vt= and v0=. The problem is 2D
when the formulas use y. boundary= free (no flux, default), zero or periodic.
The grid has n= points per side (2D: 80, 1D: 200). The time step is chosen
automatically (dt= to set it). height= scales u in the view.
```

A.5 Data and statistics

help(data)

points Scatter plot of data from a CSV file (2D with x, y ; 3D with x, y, z).

```
points(file="data.csv", x=age, y=dose, z=response)
points(file="data.csv")           first three numeric columns (3D)
points(file="data.csv", x=age, y=dose_mg)  2D scatter plot
points(file="data.csv", x=age, y=dose_mg, z=log(response))
points(data=((1, 2), (2, 3.5), (3, 3.9)))  numbers typed in
Column names are cleaned to identifiers: "dose (mg)" becomes dose_mg.
Each axis is scaled to fit the view and axes() labels data units; the next
statistics plot uses the same scale (rescale=true starts a new one, raw=true
draws in plain coordinates). Relative file names are looked up next to the
saved .blend file, or on the Desktop.
```

hist Histogram of data (bars), or of a random sample; 3D bars with $x=$ and $y=$.

```
hist(file="data.csv", x=age, bins=20)
hist(dist=normal, n=1000)
hist(file="data.csv", x=age)           bars of counts
hist(file="data.csv", x=age, density=true)  heights as density (area 1)
hist(file="data.csv", x=age, y=dose_mg)  3D histogram of two columns
hist(data=(2.1, 3.5, 3.7, 4.0, 5.2), bins=4)
hist(dist=normal, n=1000, mu=0, sigma=1)  random sample; mu, sigma are sliders
hist(dist=exp, n=5000, average=10)        means of 10 draws: central limit theorem
Distributions: normal, uniform, exp, gamma, beta, lognormal, t, chisq,
poisson, binomial. seed= picks another random sample.
```

density Smooth density of data (kernel estimate), or of a probability distribution.

```
density(file="data.csv", x=age)
density(dist=normal, mu=0, sigma=1)
density(file="data.csv", x=age)         kernel density estimate (curve)
density(file="data.csv", x=age, y=dose_mg)  2D density as a surface
```

`density(dist=normal, mu=0, sigma=1)` normal curve; mu, sigma are sliders
`density(dist=gamma, k=2, theta=1)` also uniform, exp, beta, lognormal, t,
 `chisq`
`density(dist=poisson, lam=4)` discrete: probability bars
`density(dist=bivnormal, rho=0.7)` bivariate normal surface (sx, sy, mx, my)
bw= sets the kernel bandwidth (default: Silverman's rule). To compare with a
histogram, draw `hist(..., density=true)` first so both share one scale.

fit Least-squares fit of a line, polynomial or any formula; prints the estimates.

`fit(file="data.csv", x=age, y=response)`
`fit(..., deg=2)`
`fit(..., model=K/(1+exp(-r*(x-m))))`
`fit(file="data.csv", x=dose_mg, y=response)` straight line
`fit(file="data.csv", x=dose_mg, y=response, deg=3)` cubic polynomial
`fit(file="data.csv", x=age, y=dose_mg, z=response)` plane $z = a + b x + c y$
`fit(file="data.csv", x=t, y=volume, model=V0*exp(r*x), V0=1, r=0.1)` any formula
 In `model=`, x (and y for surfaces) are the data; every other name is an
 unknown to estimate, starting from the value you give (default 1). The
 console shows each estimate with its standard error, R^2 and the RMSE.
 Draw the data first with `points(...)` so the fit shares its scale.

summary Descriptive statistics of every numeric column, and their correlations.

`summary(file="data.csv")`
`summary(data=(1, 2, 3))`

A.6 Files, scripts and export

`help(files)`

project Start a project: a folder with code, latex, figures, models and data, and the .blend file.

`project("paper1")`
`project("paper1", at=~ /Documents)`
`project()`
`project("paper1")` a new project folder paper1 in Documents (or the
 default folder): paper1.blend, code/main.py,
 latex/main.tex, figures/, models/, data/
`project("paper1", at=~ /research)` somewhere else
`project()` show this project's folders
 Inside a project, plain file names go to the folder for their type:
 .py -> code, .tex -> latex, .pdf .svg .png .jpg -> figures, .stl .glb -> models,
 .csv -> data (reading looks there too). Full paths work as they are.
 The figures documents include (\labfigure) are written to figures/.

folder Choose the project folder: where scripts, documents, figures and exports go.

`folder(~ /Documents/maths/paper1)`
`folder()`
`folder(..., default=true)`
`folder(reset)`
`folder(~ /Documents/maths/paper1)` files with plain names (`save_script("a.py")`,
 `doc("paper")`, `export("fig.pdf")`, data files)

	go to this folder; it is created if needed and saved with the .blend file
folder("...", default=true)	also the folder for new, unsaved projects
folder()	show the folder in use
folder(reset)	back to: next to the .blend file, else the Desktop

Full paths always work as they are. The View panel has a folder button.

export Save objects as STL (3D printing), glTF/GLB (web, 3D software), or SVG/PDF (2D figure).

export("shape.stl")	
export(s1, "sphere.stl", scale=20)	
export(s1, f1, "scene.glb")	
export("figure.svg")	
export("all.stl")	every Lab4IQM object in one file
export(s1, "sphere.stl", scale=20)	one object, 20x larger (slicers read mm)
export(s1, f1, "scene.glb")	glTF binary with colours, for the web
export("figure.svg")	2D vector figure seen from above, with axes
export("figure.pdf")	the same as a vector PDF (for LaTeX \includegraphics)
export(c1, "curve.svg", axes=false)	without the axes frame
export("view.png")	picture of the 3D view as you see it
export("view.png", width=3000, height=2000, transparent=true)	large, no background
export(s1, "shape.jpg")	only s1 in the picture

Relative names are saved next to the .blend file, or on the Desktop if it is not saved yet. The console shows the full path.

3D printers need closed solids: implicit() shapes are closed, $z = f(x,y)$ graphs are open surfaces with no thickness, and export says so.

code Open the Python code editor beside the view, or in its own window (for a second screen).

```
code()
code(window)
code(off)
code()      the code editor opens on the left of the 3D view (again: closes it)
code(window) the code editor opens in a new window, e.g. for a second monitor
code(off)   close the code editor beside the view
```

Scripts use the lab4iqm module: import lab4iqm as L, then L.surface(f, ...), L.curve(f, ...), L.parametric(f, ...), L.complex_plot(f, ...), L.points(...).

Run a script with Cmd+Return (Ctrl+Enter) or "Run in Lab4IQM" in the editor's header; print() output goes to the console and to output().

output Show everything your scripts printed, in a large scrollable panel or its own window.

```
output()
output(window)
output(off)
output(clear)
output()      the output panel opens below the code editor (again: closes it)
output(window) the output opens in a new window, e.g. for a second monitor
output(off)   close the output panel
output(clear) empty the output
```

The sidebar console shows only the last lines; the output keeps whole matrices and long results, and you can select and copy text from it.

save_script Save the script in the code editor as a .py file.

```
save_script("waves.py")
save_script()
save_script("waves.py", overwrite=true)
save_script("waves.py")      save it next to the .blend file (or on the Desktop)
save_script("~/Documents/maths/waves.py")  anywhere
save_script()                save again to the file it came from
An existing file is not replaced unless it is the script's own file or you
add overwrite=true. The script also stays inside the .blend file.
```

open_script Open a .py file in the code editor (it is not run until you run it).

```
open_script("waves.py")
open_script("waves.py")      from next to the .blend file (or the Desktop)
open_script("~/Documents/maths/waves.py")
Opening a file that is already open reloads it from disk. Run it with
Cmd+Return (Ctrl+Enter); save_script() writes your changes back.
```

new_script Start a new, empty script in the code editor.

```
new_script("riemann")
```

scripts List the scripts in this file and the .py files in the scripts folder.

```
scripts()
```

A.7 LaTeX documents

```
help(documents)
```

doc Open (or start) a LaTeX document in the code editor.

```
doc("paper.tex")
doc()
doc("paper.tex")      open paper.tex next to the .blend file (or on the Desktop),
                        or start it from a template if it does not exist yet
doc()                 show the last document again
Compile it with Cmd+Return (Ctrl+Enter) or compile(): the PDF opens in your
PDF viewer and updates at every compile. In the document,
\usepackage{lab4iqm} and \labfigure[width=0.8\linewidth]{Complex} put the
figure tab "Complex" in the document at each compile: a vector PDF for a 2D
graph, a rendered picture for 3D shapes (\labrender{Name} always renders).
```

compile Compile the LaTeX document to PDF (and re-export its Lab4IQM figures).

```
compile()
compile("paper.tex")
compile()              the document in the code editor (or the last one)
compile("paper.tex")  a document file
Uses Tectonic (latex(install) downloads it once), or MacTeX / TeX Live if
installed. Errors are listed with line numbers; the PDF opens in your
viewer. The first Tectonic compile downloads the LaTeX packages it needs
(an internet connection is needed once).
```

pdf Show the compiled PDF inside Lab4IQM (beside the view, or in its own window).

```
pdf()
pdf(3)
pdf(window)
pdf(open)
pdf(off)
pdf()          show the document's PDF beside the 3D view (again: hide it)
pdf(3)         go to page 3 (the arrows in the panel's header turn pages)
pdf(window)    show it in its own window, e.g. on a second screen
pdf(open)      open it in your usual PDF viewer instead
pdf(off)       hide the PDF panel
After each compile the panel shows the new PDF on the same page.
```

latex Show or choose the LaTeX engine; latex(install) downloads Tectonic.

```
latex()
latex(install)
latex(tectonic)
latex(system)
latex()        which engine compile() will use
latex(install) download Tectonic (about 21 MB), a complete LaTeX in one
                program; it fetches packages as documents need them
latex(tectonic) prefer Tectonic
latex(system)  prefer an installed TeX (MacTeX, TeX Live, MiKTeX)
```

A.8 Help

help(help)

help Show the commands by topic, the commands of one topic, or details of one command.

```
help()
help(draw)
help(plot)
```

guide Open the Lab4IQM Studio user guide, or a module's guide (shown beside the 3D view).

```
guide()
guide(oncology)
```

import Use a module's commands (in this file; the .blend remembers its imports).

```
import biostat
import biostat, oncology
import biostat          the Biostatistics commands: km, roc, compare, ...
Modules are installed like Lab4IQM (Get Extensions, or Install from Disk) and
wait until a file imports them, as in Python or R's library(). A .blend file
remembers its imports and loads them again when it is opened. modules() lists
the modules installed; modules(off=biostat) stops using one in this file.
From a script: L.run("import biostat").
```

modules List the Lab4IQM modules installed and imported; stop using one (off) or import it (on).

```
modules()
modules(off=biostat)
modules(on=biostat)
```

Modules add commands, help topics, examples and example data for a subject, for example Biostatistics. Install a module like the Lab4IQM extension (Get Extensions, or Install from Disk), then type `import biostat`.

```
modules(off=biostat)  stop using it in this file; modules(on=biostat) = import
    biostat
```

`help(<module id>)` lists a module's commands; "`<module id>:<file>`" reads one of its example data files, e.g. `km("biostat:leukemia.csv", ...)`.

To remove a module for good: Preferences > Add-ons, its menu, Uninstall.

A.9 Maths in expressions

Functions: `abs`, `acos`, `acosh`, `acot`, `acsc`, `arccos`, `arccot`, `arccsc`, `arcsec`, `arcsin`, `arctan`, `asec`, `asin`, `asinh`, `atan`, `atan2`, `atanh`, `cbrt`, `ceil`, `cos`, `cosh`, `cot`, `csc`, `ddx`, `ddy`, `exp`, `factorial`, `floor`, `heaviside`, `hypot`, `lap`, `ln`, `log`, `log10`, `log2`, `max`, `min`, `mod`, `round`, `sec`, `sign`, `sin`, `sinh`, `sqrt`, `tan`, `tanh`. Constants `pi`, `e`, `tau`; powers `x^2` or `x**2`; ranges `a..b`.