The econ-viz Package: Publication-Quality Microeconomics Diagrams

Pin Yue Sung[*†] Ling Tak Douglas Chung[*]

2026-09-26 · v1.12.0 · ·

Contents

1 Introduction **2**
1.1 Scope 2
1.2 Reading guide 2
2 Installation **2**
2.1 Requirements 2
2.2 Installing uv 3
2.3 Installing econ-viz 3
2.4 Optional dependencies 3
2.5 Installing the command-line tool 3
2.6 Development setup 4
2.7 Verifying the installation 4
3 Quick start **4**
3.1 A minimal example 4
3.2 Step by step 5
4 Canvas **6**
4.1 Constructor 6
4.2 Drawing methods 7
4.3 Styles 11
4.4 Method chaining 13
5 Figures and demand diagrams **13**
5.1 Multi-panel figures 14
5.2 Paths 15
5.3 Demand diagrams 15
5.4 Price-effect decomposition 17
5.5 Edgeworth box 18
6 Core models **19**
6.1 Smooth preferences 19
6.2 Kinks and corners 21
6.3 Income and reference points 23
7 Advanced models **25**
8 Analysis **26**
8.1 Comparative statics 26
8.2 Slutsky matrix 27
8.3 Homogeneity 28
8.4 Returns to scale 28
9 Themes and configuration **29**
9.1 Built-in themes 29
9.2 Custom themes 30
9.3 Configuration files 31
10 Export **32**
10.1 Output formats 32
10.2 Animated GIFs 32
10.3 Interactive window 32
11 Command-line interface **32**
11.1 Help and models 33
11.2 Plotting 33
11.3 Creating a configuration file 36
11.4 Demand formulas 36
12 Animation **36**
13 Interactive widgets **38**
13.1 Notebook setup 38
13.2 Widgets or GIFs 39
14 LaTeX parsing **39**
14.1 Supported forms 39
14.2 Errors 40
14.3 In the command-line tool 40
Change history **41**
Command Index **44**
References **45**

[*]Department of International Business, National Chengchi University, Taipei, Taiwan.
[†]Corresponding author. Email: contact@econ-viz.org

1 Introduction

The econ-viz package is a Python toolkit for producing publication-quality microeconomics diagrams: indifference curves, budget constraints, consumer equilibria, demand curves and exchange economies. It draws in the conventions of textbook figures—first-quadrant axes with arrow tips, LaTeX-style labels and no numeric ticks—and exports to PNG, PDF, SVG, TikZ or animated GIF.

1.1 Scope

econ-viz is organised in five parts: utility models, equilibrium solving, utility levels, drawing layers, and output formats. Models and solutions remain usable on their own, and the same model can be passed to a canvas, multi-panel figure, demand diagram, or analysis API.

1.2 Reading guide

The manual is organised in four parts (Table 1):

Table 1 Chapter guide

Topic	Subtopic	Description	Section
Drawing & export	Canvas & layers	Axes, curves, equilibria	Section 4
	Figures & demand	Panels & demand	Section 5
	Themes & config	Colours, styles, config	Section 9
	Output formats	Image files and TikZ	Section 10
Models & analysis	Built-in models	Utility functions	Section 6
	Custom & many-good	Custom, *N* goods	Section 7
	Comparative statics	Derivatives & Slutsky	Section 8
Animation & interaction	GIF animation	Sweeps as GIFs	Section 12
	Jupyter widgets	Sliders in notebooks	Section 13
Other inputs	Command line	Plots from the shell	Section 11
	LaTeX parser	Models from formulas	Section 14

On first use, read Section 3 and Section 4 in order. To look up a particular API, go straight to its section or to the command index.

2 Installation

2.1 Requirements

econ-viz requires Python 3.10 or later[1]. This chapter manages packages and projects with uv[2] and gives the matching pip command where useful.

[1] The Python website provides installers for every operating system: `https://www.python.org/downloads/`.

[2] uv is a fast Python package and project manager by Astral that can also manage Python versions; see its documentation for installation and usage: `https://docs.astral.sh/uv/`.

Installing the package also installs NumPy, SciPy, matplotlib and SymPy, for array computation, numerical solving, plotting and symbolic algebra[3].

2.2 Installing uv

The commands in this manual use uv. Existing projects can keep using pip, pipx or Poetry the package API does not depend on the tool that installed it.

Run the installer for your operating system.

macOS, Linux

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

Windows

```
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps
1 | iex"
```

2.3 Installing econ-viz

Create a project and add econ-viz as a dependency:

```
uv init my-diagrams
cd my-diagrams
uv add econ-viz
```

Run scripts inside the project environment with `uv run`:

```
uv run python main.py
```

Save the basic example of Section 3 as `main.py` in the project directory, then run the command above. `uv add` records the dependency and `uv run` uses the project's Python environment; install and run in the same environment.

To pin the version this manual describes, give it when adding the dependency:

```
uv add "econ-viz==1.12.0"
```

In an existing Python virtual environment, install it with pip:

```
python -m pip install -U econ-viz
```

The package is named `econ-viz`; its Python import name is `econ_viz`:

```
from econ_viz import Canvas, solve
```

An `import` statement cannot contain a hyphen. On `ModuleNotFoundError`, check that the interpreter running the script is the one the package was installed into.

2.4 Optional dependencies

Animation and Jupyter widgets need optional dependencies. Install the group you need:

`animation` GIF export with `Animator` (Section 12); pulls in Pillow.

`interactive` Notebook widgets with `WidgetViewer` (Section 13); pulls in ipywidgets and IPython.

`all` Every optional dependency.

```
uv add "econ-viz[animation]"    # GIF export (Pillow)
uv add "econ-viz[interactive]"  # notebook widgets
uv add "econ-viz[all]"          # all extras
```

Installing an extra does not run an animation or open a notebook by itself; call the API as shown in the relevant chapter.

2.5 Installing the command-line tool

To use only the command-line interface, install econ-viz as a standalone tool (Section 11):

```
uv tool install econ-viz
```

[3] Ordinary Python plotting needs no LaTeX installation; a LaTeX distribution is only required to compile exported TikZ code.

This installs the tool in an environment of its own. To import the package in your own Python code, still run `uv add econ-viz` in that project; calling `uv run econ-viz` inside the project keeps the CLI and your code on the same version.

2.6 Development setup

```
git clone https://github.com/EconViz/econ-viz.git
cd econ-viz
uv sync --all-extras
```

`uv sync --all-extras` installs the development dependencies and every optional dependency. Then run the tests:

```
uv run pytest
```

2.7 Verifying the installation

```
uv run econ-viz --version   # econ-viz 1.12.0
uv run econ-viz help
```

The version in the comment is only an example; the output reflects the installed version. To check that Python can import the drawing and solving API:

```
uv run python -c "from econ_viz import Canvas, solve; print('OK')"
```

On a server or any environment without a display, write files with `save()` or the CLI's `--output`. `show()` needs an interactive plotting backend, so a window that fails to open does not mean the installation failed.

3 Quick start

3.1 A minimal example

This chapter uses the complete Cobb-Douglas consumer problem:

$$\max_{x,y} u(x,y) = x^{\frac{1}{2}} y^{\frac{1}{2}}$$
$$\text{s.t. } 2x + 3y = 30$$

The script solves the optimum and draws the indifference map, budget set, and equilibrium shown in Figure 1.

Example 1

```
from econ_viz import Canvas, levels, solve
from econ_viz.models import CobbDouglas

model = CobbDouglas(alpha=0.5, beta=0.5)
eq    = solve(model, px=2.0, py=3.0, income=30.0)
lvls  = levels.around(eq.utility, n=5)

cvs = Canvas(
    x_max=20, y_max=15,
    x_label="x", y_label="y",
    title=r"Cobb-Douglas $x^{0.5} y^{0.5}$"
)
cvs.add_utility(model, levels=lvls)
cvs.add_budget(2.0, 3.0, 30.0, fill=True)
cvs.add_equilibrium(eq, show_ray=True)
cvs.save("cobb_douglas.png")
```

Running `main.py` writes `cobb_douglas.png` to the current working directory. The optimum is $x^* = 7.5$, $y^* = 5$; spending is $2 \times 7.5 + 3 \times 5 = 30$, exactly the income. The budget line meets the axes at $\frac{I}{p_x} = 15$ and $\frac{I}{p_y} = 10$.

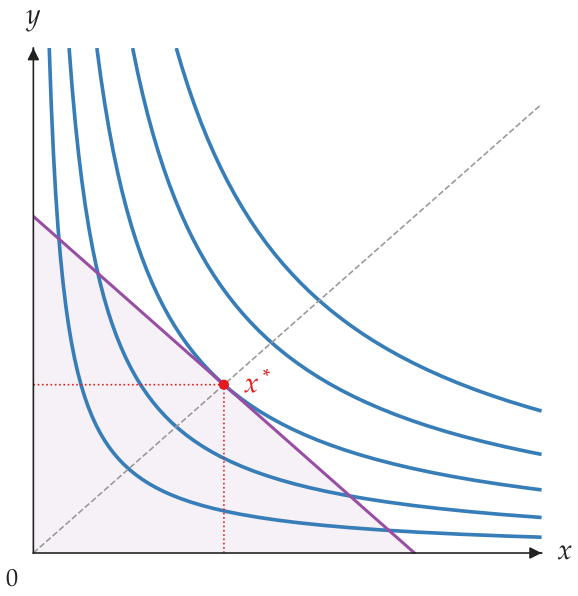

Figure 1
The consumer-equilibrium diagram.

3.2 Step by step

Every diagram follows the same steps: build a model, solve for the equilibrium, pick utility levels, create a canvas, add layers and export.

`econ_viz.models`

```
from econ_viz.models import CobbDouglas
model = CobbDouglas(alpha=0.5, beta=0.5)
```

Pick a utility function. Section 6 lists every built-in model; any Python function can also be wrapped as a model (Section 7).

`solve` `solve(⟨model⟩, px=⟨float⟩, py=⟨float⟩, income=⟨float⟩)`

Solve the consumer's problem and return an immutable `Equilibrium`.

`x` *float* Optimal quantity x^*.

`y` *float* Optimal quantity y^*.

`utility` *float* Utility at the optimum.

`bundle_type` *str* Solution type: `"interior"` for an interior solution, `"boundary"` when the numerical solution sits at a quantity lower bound (e.g. Stone-Geary subsistence), `"kink"` for a Leontief kink and `"corner"` for a perfect-substitutes corner.

```
from econ_viz import solve
eq = solve(model, px=2.0, py=3.0, income=30.0)
print(eq.x, eq.y, round(eq.utility, 3))

# 7.5 5.0 6.124
```

`levels.around` `levels.around(⟨utility⟩, n=⟨int⟩)`

Generate n utility levels centred on a given utility. Figure 1 uses `n=5`, so it draws five indifference curves.

`levels.around()` keeps the given utility among the levels, so the equilibrium lies on one of the curves. Passing `levels=5` instead lets the canvas pick levels from the sampled utility range, which need not include the equilibrium utility. To compare the same preferences across several figures, compute `lvls` once and pass it to each figure.

`Canvas` `Canvas(x_max=⟨float⟩, y_max=⟨float⟩, ...)`

Create the canvas and set the axis ranges; the other options are described in Section 4.

The axis ranges only control what is shown; they play no part in `solve()`. If the equilibrium is missing from the figure, check whether `eq.x` or `eq.y` lies outside the range and raise the axis limits.

`Canvas.add_*` Add indifference curves, a budget line or the equilibrium to the same canvas. The calls can be chained (Section 4.4). `add_equilibrium()` uses an existing solution and does not solve again; after changing prices, income or the model, call `solve()` again and build a new figure.

`Canvas.save` `Canvas.show` `save()` writes the file in the format given by its extension (Section 10); `show()` opens a matplotlib window. Output paths are relative to the working directory of the script; create any subdirectory beforehand.

Checking inputs and failed solves

Prices and income must be positive; otherwise `solve()` raises `InvalidParameterError`[4].

When the solver does not converge, `solve()` raises `OptimizationError` rather than returning an invalid equilibrium.

```
from econ_viz import InvalidParameterError, OptimizationError, solve

try:
    eq = solve(model, px=2.0, py=3.0, income=30.0)
except (InvalidParameterError, OptimizationError) as error:
    print(error)
else:
    print(round(eq.x, 3), round(eq.y, 3), eq.bundle_type)
```

Numerical solutions carry small errors: a result may be close to, but not exactly, `7.5`. Compare with a tolerance when checking results, and round with `round()` only for display, so early rounding does not distort later calculations.

4 Canvas

`Canvas` builds on a matplotlib `Figure` and `Axes` and provides layers for indifference curves, budget lines, equilibria and more.

4.1 Constructor

`Canvas`

```
from econ_viz import ArrowStyle, Canvas, Stroke, themes

cvs = Canvas(
    x_max=20,
    y_max=15,
    x_label="x",
    y_label="y",
    title=r"Cobb-Douglas $x^{0.5} y^{0.5}$",
    dpi=300,
    x_label_pos="right",   # "top", "right" or "bottom"
    y_label_pos="top",     # "left", "top" or "right"
    font="DejaVu Sans",
    math_font="stix",
    axis_stroke=Stroke(width=1.0, arrow=ArrowStyle.TRIANGLE),
    theme=themes.default,
)
```

Updated: v1.7.0 Create a canvas and set its axis ranges, axis labels, fonts, line styles and theme. Visual settings not passed explicitly come from the active `Config` and `Theme`.

[4] Smooth models are solved with sequential least squares programming (SLSQP).

`x_max`
float · default `10`
Upper bound of the horizontal axis.

`y_max`
float · default `10`
Upper bound of the vertical axis.

`x_label`
str · default `"X"`
Label at the tip of the horizontal axis.

`y_label`
str · default `"Y"`
Label at the tip of the vertical axis.

`title`
str | *None* · default `None`
Figure title.

`dpi`
int · default `300`
Raster export resolution, clamped to 1–1200.

`x_label_pos`
str · default `"right"`
Position of the horizontal label relative to the arrow; a `LabelPosition` is also accepted (Table 2).

Table 2 `x_label_pos` values

Value	Position
`"top"`	Above the arrow
`"right"`	Right of the arrow
`"bottom"`	Below the arrow

`y_label_pos`
str · default `"top"`
Position of the vertical label relative to the arrow (Table 3).

Table 3 `y_label_pos` values

Value	Position
`"left"`	Left of the arrow
`"top"`	Above the arrow
`"right"`	Right of the arrow

`font`
str | *list* · default `None`
Font for all text, or a list of candidate fonts. It applies to this canvas only and leaves matplotlib's global configuration alone.

`math_font`
str · default `None`
matplotlib math font set: `dejavusans`, `dejavuserif`, `cm`, `stix`, `stixsans` and so on.

`axis_stroke`
Stroke · default `theme`
Width, style, colour and arrowhead of both axes (Section 4.3).

`x_axis_stroke`
Stroke · default `None`
Override for the horizontal axis.

`y_axis_stroke`
Stroke · default `None`
Override for the vertical axis.

`theme`
Theme · default `themes.default`
Colour and style theme (Section 9).

4.2 Drawing methods

The `add_*` methods add a layer to the canvas and return the same `Canvas`, so calls can be chained (Section 4.4).

`add_utility`

```
cvs.add_utility(
    func,
    levels=3,          # count or list of levels
    color=None,        # default: theme.ic_color
    linewidth=None,    # default: theme.ic_linewidth
```

```
    show_rays=False,
    show_kinks=False,
    kink_radius=1.0,
    show_bliss=True,   # star at the bliss point (Satiation)
    show_ic_labels=False,
    stroke=None,
    ray_stroke=None,
    ic_label=None,
    highlight_level=None,
    secondary_stroke=None,
    label_style="numeric",
)
```

Draw indifference curves for a utility model.

`levels`
int | list · default `3`
: A number of curves, or a list of utility values such as `levels.around(eq.utility, n=5)` to place them around the optimum.

`highlight_level`
float | None · default `None`
: Focal utility level. The closest curve is drawn in the main style.

`secondary_stroke`
Stroke | None · default `None`
: Style of the other curves; defaults to `theme.secondary_ic_stroke`.

`show_ic_labels`
bool · default `False`
: Label the curves. Labels follow the curve's tangent and stay inside the plotting area (Figure 3).

`label_style`
str · default `"numeric"`
: `"numeric"` shows utility values; `"ordinal"` numbers the curves $u_1, u_2, \dots$ in increasing utility.

```
lvls = levels.around(eq.utility, n=5)
cvs.add_utility(
    model,
    levels=lvls,
    highlight_level=eq.utility,
    show_ic_labels=True,
    label_style="ordinal",
)
```

add_budget

```
cvs.add_budget(
    px, py, income,
    color=None,
    linewidth=None,
    linestyle="-",
    label=None,        # legend label (LaTeX)
    fill=False,        # shade feasible set
    fill_alpha=None,   # default: theme.budget_fill_alpha
    stroke=None,
)
```

Draw the budget line $p_x x + p_y y = I$.

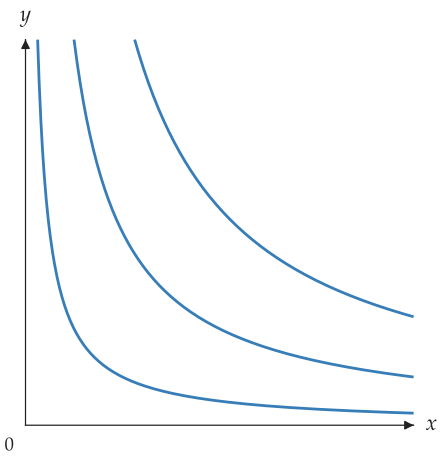

Figure 2
Three indifference curves.

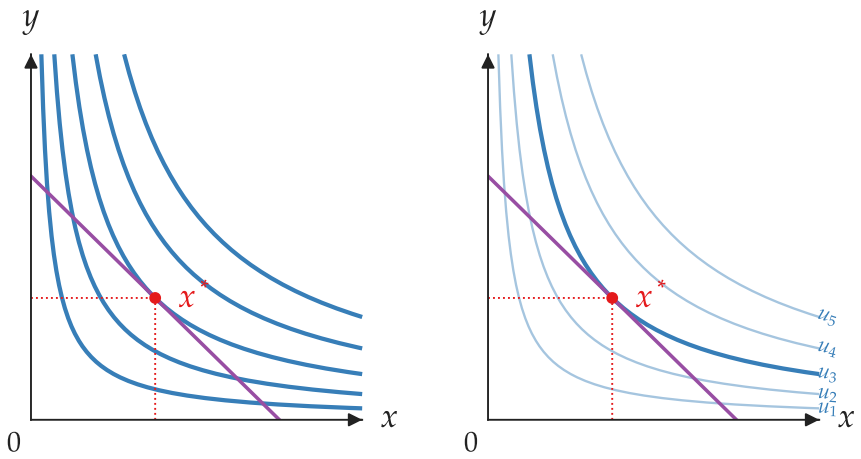

Figure 3
Focal and secondary curves.

`fill`
bool | Fill · default `False`
`True` shades the feasible set with the theme's fill; pass a `Fill` to set its colour and opacity.

`stroke`
Stroke | None · default `None`
Colour, width, style and opacity of the line in one object; takes precedence over `color`, `linewidth` and `linestyle` (Section 4.3).

`label`
str | None · default `None`
Legend text, shown once `show_legend()` is called.

`add_equilibrium`

```
cvs.add_equilibrium(
    eq,                 # result of solve()
    color=None,
    markersize=None,
    label="x^*",
    drop_dashes=True,   # dashed lines to axes
    show_ray=False,     # expansion path
    drop_stroke=None,
    ray_stroke=None,
    marker=None,
)
```

Mark the optimal bundle returned by `solve()`.

`drop_dashes`
bool · default `True`
Drop lines to both axes.

`show_ray`
bool · default `False`
Draw the expansion path through the origin and the optimum.

`marker`
Marker | None · default `None`
Style of the equilibrium marker (Section 4.3).

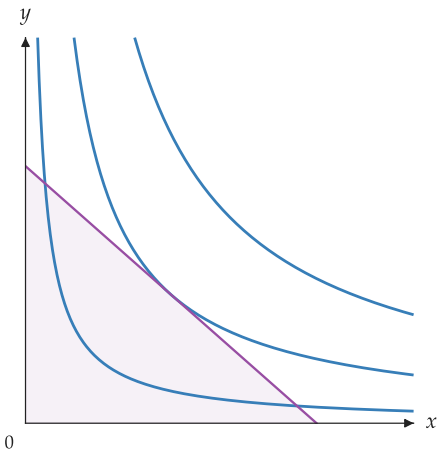

Figure 4
Budget line and shading.

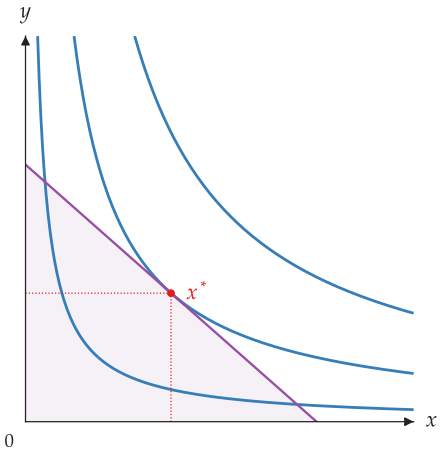

Figure 5
Equilibrium and drop lines.

`add_ray`

```
cvs.add_ray(
    slope,              # dy/dx
    color=None,
    linewidth=None,
    stroke=None,
)
```

Draw a ray from the origin with slope `slope` ($\mathrm{d}y/\mathrm{d}x$), for an expansion path or a fixed consumption ratio.

`add_point`

```
cvs.add_point(
    x, y,
    label=None,
    color=None,
    markersize=None,
    offset=None,        # label offset (pt)
    marker=None,
)
```

Mark any point, such as a bundle to compare with the optimum.

`label`
str | *Label* | *None* · default `None`
Marker text, as a string or a `Label`.

`offset`
tuple | *None* · default `None`
Offset of the text from the point, in points.

`marker`
Marker | *None* · default `None`
Style of the point marker.

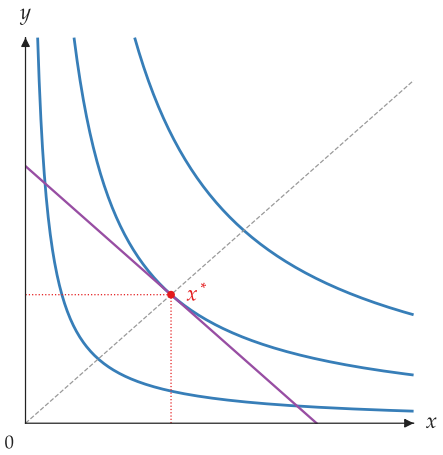

Figure 6
Expansion-path ray.

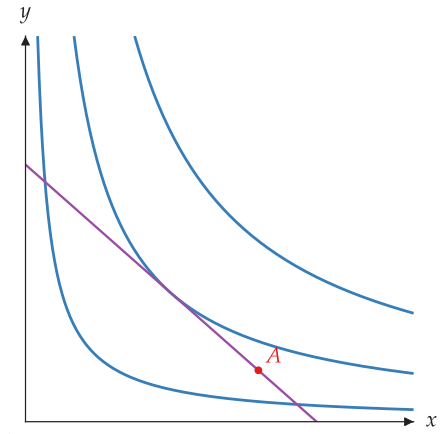

Figure 7
A labelled point A.

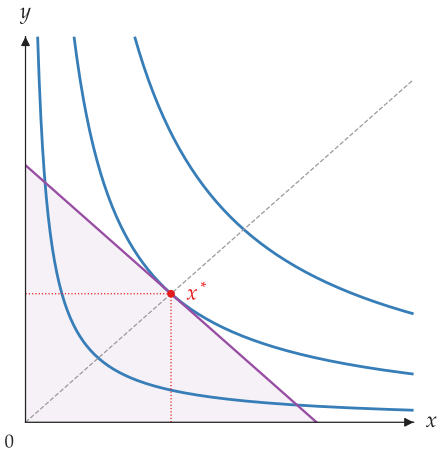

Figure 8
The complete canvas.

show
save

```
cvs.show()                # interactive window
cvs.save("figure.png")    # .png / .pdf / .svg / .tex
```

`show()` opens an interactive window. `save()` writes the file in the format given by its extension (Section 10) and releases the matplotlib figure, so call it last.

4.3 Styles

`LineStyle` defines line patterns, `ArrowStyle` arrowheads, and `Stroke` bundles width, pattern, colour and arrowhead for axes, budget lines, paths and drop lines.

Line styles

`LineStyle` `LineStyle.SOLID, DASHED, DOTTED, DASHDOT`

New: v1.7.0 Solid, dashed, dotted and dash-dot lines (Figure 9). Axes accept the enum or the matching string; other lines use `Stroke(style=...)`.

```
import matplotlib.pyplot as plt

from econ_viz import Canvas, LineStyle

styles = [
    LineStyle.SOLID,
    LineStyle.DASHED,
    LineStyle.DOTTED,
    LineStyle.DASHDOT,
]
```

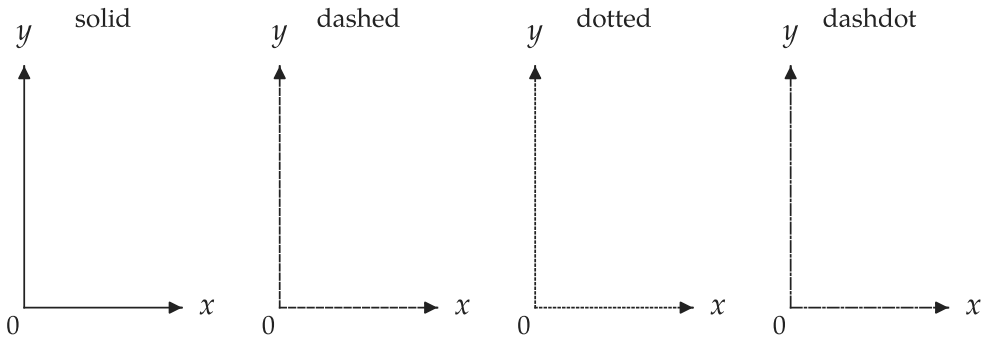

Figure 9
The four line styles.

```
fig, axes = plt.subplots(1, 4, figsize=(8.5, 2.2))
for ax, style in zip(axes, styles):
    Canvas(
        title=style.value,
        x_line_style=style,
        y_line_style=style,
        fig=fig,
        ax=ax,
    )

# Axis labels sit outside the arrowheads; give four panels more room.
fig.subplots_adjust(wspace=0.62)
fig.savefig("line_styles.png", dpi=160, transparent=True)
```

Arrow styles

`ArrowStyle` `ArrowStyle.SIMPLE`, `TRIANGLE`, `FANCY`, `WEDGE`

New: v1.7.0 Four arrowhead styles (Figure 10). Axes use `x_arrow_style` and `y_arrow_style`; other lines use `Stroke(arrow=...)`.

Stroke

`Stroke` `Stroke(width=`⟨*float*⟩`, style=`⟨*LineStyle*⟩`, color=`⟨*str*⟩`, arrow=`⟨*ArrowStyle*⟩`, opacity=`⟨*float*⟩`)`

New: v1.7.0 *Updated: v1.9.0* Width, pattern, colour, arrowhead and opacity of one line. Fields left unset inherit from the active theme. `opacity` ranges from 0 to 1.

Axis

`Axis` `Axis(label=None, label_position=None, stroke=None)`

New: v1.8.0 Label, label position and line of one axis in one object. `Canvas`, `Figure`, `DemandDiagram` and `EdgeworthBox` all accept `x_axis` and `y_axis`. When shorthand arguments are passed as well, fields set in the `Axis` win.

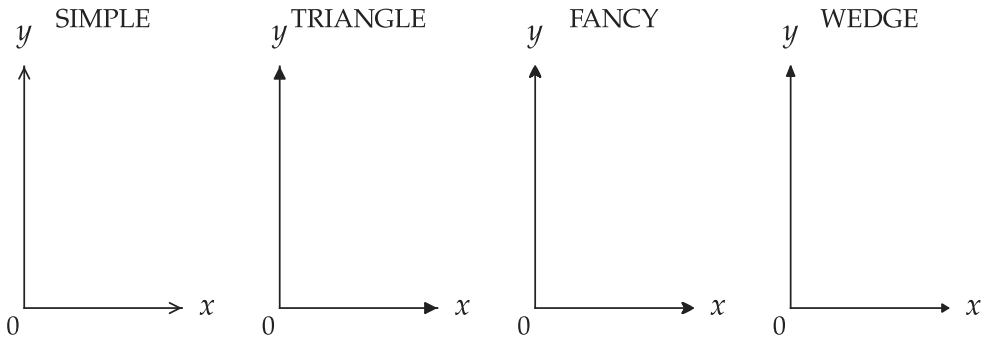

Figure 10
The four arrow styles.

```python
from econ_viz import Axis, Canvas, Label, Stroke

cvs = Canvas(
    x_axis=Axis(
        label=Label(text="x_1", fontsize=16),
        label_position="bottom",
        stroke=Stroke(width=1.2),
    ),
    y_axis=Axis(label="x_2"),
)
```

Marker and Label

`Marker` `Marker(color=None, size=None, shape=None, opacity=None)`

New: v1.8.0 *Updated: v1.9.0* Colour, size, shape and opacity of a point marker. `shape` takes a matplotlib marker string such as `"o"`, `"s"`, `"^"`, `"D"` or `"*"`.

`Label` `Label(text=None, position=None, offset=None, color=None, fontsize=None, visible=None, opacity=None)`

New: v1.8.0 *Updated: v1.9.0* Text, position, offset, colour, size, visibility and opacity of a label. The position is a side or a corner; `Label(visible=False)` hides the text.

```python
from econ_viz import Label, Marker

cvs.add_equilibrium(
    eq,
    marker=Marker(shape="s", size=8),
    label=Label(position="bottom-left", offset=8),
)
cvs.add_point(
    12, 2,
    marker=Marker(color="black", shape="D"),
    label=Label(text="A", position="left", fontsize=14),
)
```

Fill

`Fill` `Fill(color=None, opacity=None)`

New: v1.8.0 *Updated: v1.9.0* Colour and opacity of a shaded area. `alpha` is a compatible shorthand for `opacity`; one call must not pass different values for both.

```python
from econ_viz import Fill

cvs.add_budget(
    2, 3, 30,
    fill=Fill(color="lightgrey", opacity=0.4),
)
```

4.4 Method chaining

The `add_*` methods can be chained, ending with `save()`:

```python
Canvas(x_max=20, y_max=15) \
    .add_utility(model, levels=lvls) \
    .add_budget(2.0, 3.0, 30.0, fill=True) \
    .add_equilibrium(eq, show_ray=True) \
    .save("figure.png")
```

5 Figures and demand diagrams

On top of `Canvas`, econ-viz builds these larger diagrams:

- `Figure` for multi-panel layouts;

- `PricePath` and `IncomePath` for budget and equilibrium sweeps;
- `DemandDiagram` for linked goods-space and Marshallian-demand views;
- `decompose_price_effect` for the Hicks and Slutsky decompositions;
- `EdgeworthBox` for two-consumer exchange diagrams.

5.1 Multi-panel figures

`Figure` `Figure(⟨layout⟩, x_max=⟨float⟩, y_max=⟨float⟩, ..., shared_x=False, shared_y=False)`

New: v1.2.0 Compose several canvases into one figure, for before-and-after comparisons, decompositions or classroom slides. It takes the same styling arguments as `Canvas`. `fig[idx]` returns the panel `Canvas`, so the drawing API is unchanged once the layout exists.

Example 2

```
from econ_viz import Figure, Layout, Legend, levels, solve
from econ_viz.models import CobbDouglas

fig = Figure(
    Layout.SIDE_BY_SIDE,
    x_max=20,
    y_max=15,
    x_label="x",
    y_label="y",
    title="Before / After Price Change",
    shared_y=True,
)

cases = [
    (CobbDouglas(alpha=0.5, beta=0.5), 2.0, 3.0, 30.0, r"Before: $p_x=2$"),
    (CobbDouglas(alpha=0.3, beta=0.7), 4.0, 3.0, 30.0, r"After: $p_x=4$"),
]

for idx, (model, px, py, income, title) in enumerate(cases):
    eq = solve(model, px=px, py=py, income=income)
    panel = fig[idx]
    panel.ax.set_title(title)
    panel.add_utility(model, levels=levels.around(eq.utility, n=5), label="IC")
    panel.add_budget(px, py, income, fill=True, label="BC")
    panel.add_equilibrium(eq, show_ray=True)

fig[0].show_legend(legend=Legend(position="upper right"))
fig.save("figure_side_by_side.png")
```

`Layout` *New: v1.2.0* The available layouts: `SINGLE`, `STACKED`, `SIDE_BY_SIDE`, `TOP_TWO_BOTTOM_ONE`, `TOP_ONE_BOTTOM_TWO`, `GRID_2X2` and `GRID_3X3`.

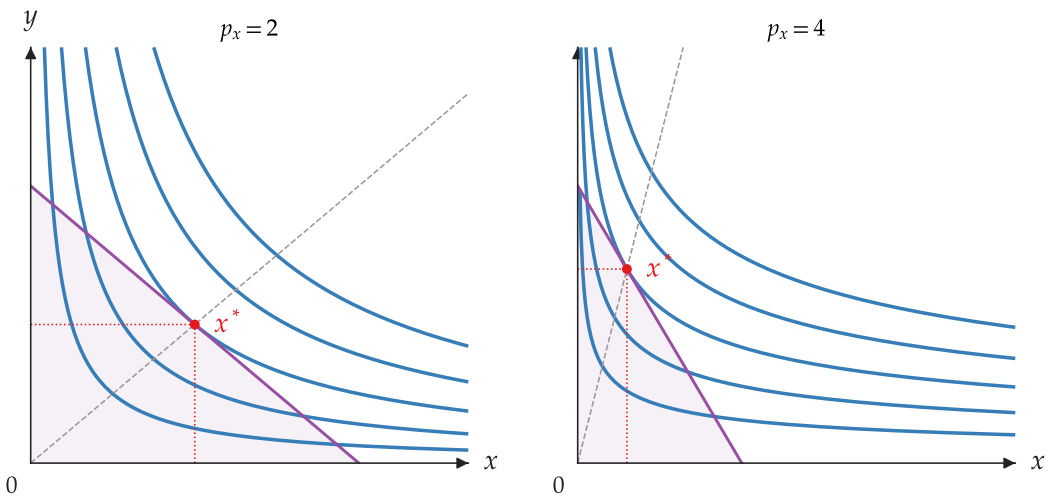

Figure 11
Price change, before and after.

5.2 Paths

PricePath
IncomePath

```
from econ_viz import IncomePath, LinearBudget, PricePath
from econ_viz.models import CobbDouglas

model = CobbDouglas(alpha=0.5, beta=0.5)
budget = LinearBudget(px=2.0, py=2.0, income=40.0)

price_path = PricePath(
    model,
    budget=budget,
    price="px",
    price_range=(0.8, 6.0),
    n=40,
)
income_path = IncomePath(
    model,
    budget=budget,
    income_range=(20.0, 80.0),
    n=30,
)
```

New: v1.2.0 Sweep one budget parameter while repeatedly solving the consumer's problem. Use a path to draw price- and income-consumption curves with `Canvas.add_path`, to feed a `Demand Diagram`, or to inspect how bundles move as prices or income vary.

Canvas.add_path

```
from econ_viz import Canvas, levels

eq = price_path.equilibria[len(price_path.equilibria) // 2]
lvls = levels.around(eq.utility, n=5)

Canvas(x_max=25, y_max=20) \
    .add_utility(model, levels=lvls) \
    .add_path(price_path, label="PCC") \
    .save("price_path.png")
```

New: v1.2.0 Draw a path directly on a canvas when a full demand diagram is not needed. For Cobb-Douglas preferences the price-consumption curve is horizontal (Figure 12): spending on y does not depend on p_x.

5.3 Demand diagrams

DemandDiagram

```
DemandDiagram(⟨price path⟩, title=None, ...)
.add_marshallian_panel(price_markers=None, show_pcc=False,
show_demand_guides=True)
```

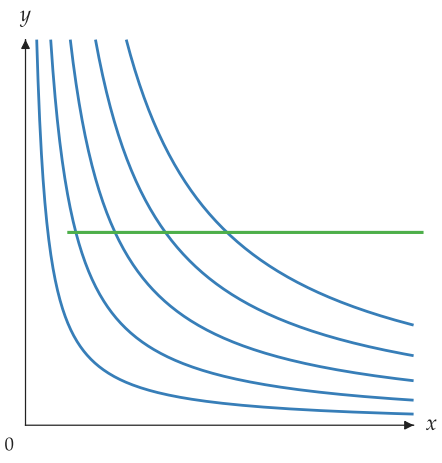

Figure 12
The price-consumption curve as p_x varies.

New: v1.2.0 A stacked two-panel figure: indifference curves, budget lines and equilibria in the top panel, the corresponding Marshallian demand curve in the bottom one (Figure 13).

- `DemandDiagram` expects a `PricePath`.
- Smooth, kinked and corner demand are handled differently, so that the bottom panel stays economically meaningful.
- `show_pcc=True` overlays the price-consumption curve in goods space.

Example 3

```
from econ_viz import DemandDiagram, LinearBudget, PricePath
from econ_viz.models import CobbDouglas

model = CobbDouglas(alpha=0.5, beta=0.5)
budget = LinearBudget(px=2.0, py=2.0, income=40.0)
path = PricePath(
    model,
    budget=budget,
    price="px",
    price_range=(0.8, 6.0),
    n=40,
)

fig = DemandDiagram(path, title="Demand: Cobb-Douglas")
fig.add_marshallian_panel(
    price_markers=[1.5, 4.0],
    show_pcc=False,
    show_demand_guides=True,
)
fig.save("demand_cobb_douglas.png")
```

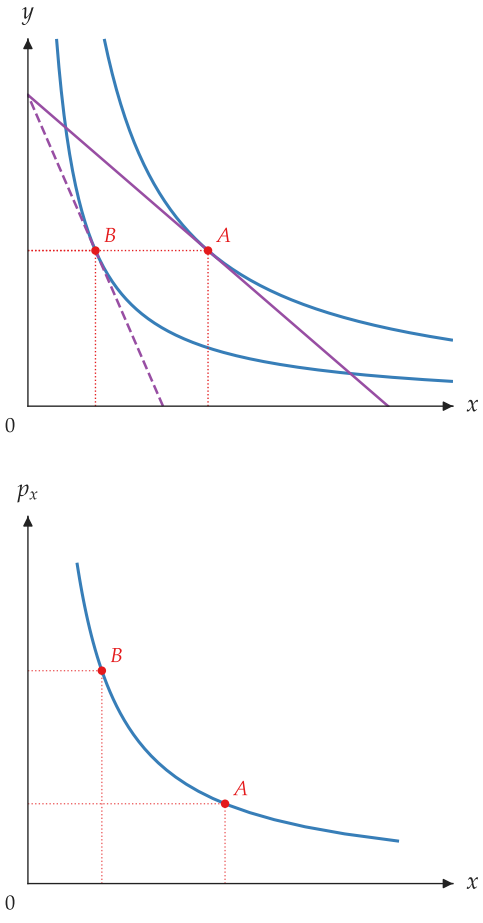

Figure 13
Demand curve at the optima.

5.4 Price-effect decomposition

`decompose_price_effect` `decompose_price_effect(⟨model⟩, px=(⟨old⟩, ⟨new⟩), py=⟨float⟩, income=⟨float⟩, method=⟨method⟩)`

New: v1.5.0 Separate a price change into substitution and income effects. Choose `Decomposition Method.HICKS` to hold the original utility fixed (Hicks, 1939), or `SLUTSKY` (Slutsky, 1915) to keep the original bundle affordable. The result has these fields:

`A, B, C` The original, compensated and final bundles.

`substitution_effect` Substitution effect.

`income_effect` Income effect.

`total_effect` Total effect.

`compensated_income` Income after compensation.

`Canvas.add_decomposition` Draw the three bundles A, B and C, the budget lines, the indifference curves and the effect arrows. A Hicks decomposition draws the curves through A and C; a Slutsky decomposition also draws the one through B.

New: v1.5.0
Updated: v1.9.0

`show_curves` Draw the decomposition's indifference curves; set to `False` when you draw your own.
bool · default `True`

`curve_stroke` Style of the indifference curves.
Stroke | *None* · default `None`

`curve_label` Show and style the U_0, U_1 and U_B labels.
Label | *None* · default `None`

`substitution, income` Style of the substitution and income effects (see `Effect` below).
Effect | *None* · default `None`

`point_marker` Markers of the bundles.
Marker | *None* · default `None`

`point_label` Text of the bundles.
Label | *None* · default `None`

`legend` Position and style of the legend (see `Legend` below).
Legend | *None* · default `None`

`Effect` `Effect(color=None, y=None, label=None, label_position="right", label_offset=4, opacity=None)`

New: v1.8.0 Colour, height of the arrow below the horizontal axis, label and opacity of the substitution or income effect. The line itself can still be overridden with `substitution_stroke` or `income_stroke`.
Updated: v1.9.0

`Legend` `Legend(position="auto", fontsize=None, frame=None, columns=None, visible=None, opacity=None)`

New: v1.9.0 `"auto"` picks the inner corner that hides the least of the plot; when all four corners overlap it, the legend moves to the right. Inner corners such as `"upper left"` and outer positions `"top"`, `"bottom"`, `"left"` and `"right"` can also be given. `Legend(visible=False)` hides the legend.

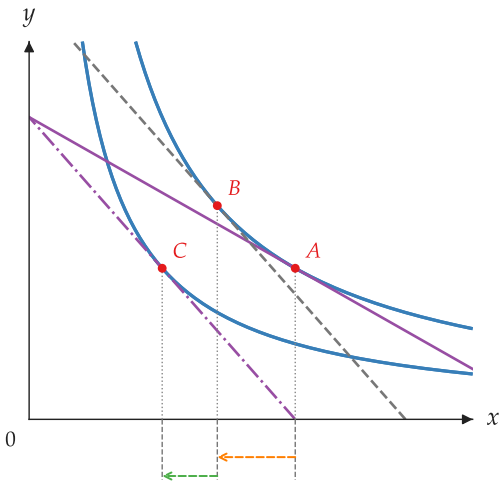

Figure 14
Hicks decomposition.

Example 4

```python
from econ_viz import Canvas, DecompositionMethod, Effect, Label, Legend
from econ_viz.models import CobbDouglas
from econ_viz.optimizer import decompose_price_effect

model = CobbDouglas(alpha=0.5, beta=0.5)
result = decompose_price_effect(
    model,
    px=(2.0, 4.0),
    py=3.0,
    income=60.0,
    method=DecompositionMethod.HICKS,
)

(
    Canvas(x_max=25, y_max=25, title="Hicks decomposition")
    .add_decomposition(
        result,
        show_arrows=True,
        label_effects=True,
        show_x_projections=True,
        substitution=Effect(label="SE"),
        income=Effect(label="IE", opacity=0.8),
        curve_label=Label(position="right"),
        legend=Legend(position="bottom"),
    )
    .save("hicks.png")
)
```

5.5 Edgeworth box

`EdgeworthBox` `EdgeworthBox(`⟨*utility A*⟩`,` ⟨*utility B*⟩`, total_x=`⟨*float*⟩`, total_y=`⟨*float*⟩`, ...)`

New: v1.3.0

A two-consumer exchange economy (Edgeworth, 1881). Consumer A is measured from the lower-left origin and consumer B from the upper-right origin. Layers are added with `add_endowment`, `add_contract_curve`, `add_core`, `add_price_line` and `add_walrasian_equilibrium`.

Use `method="mrs"` for the contract curve of smooth preferences, and `method="pareto"` for models with kinks, corners or custom piecewise utility functions.

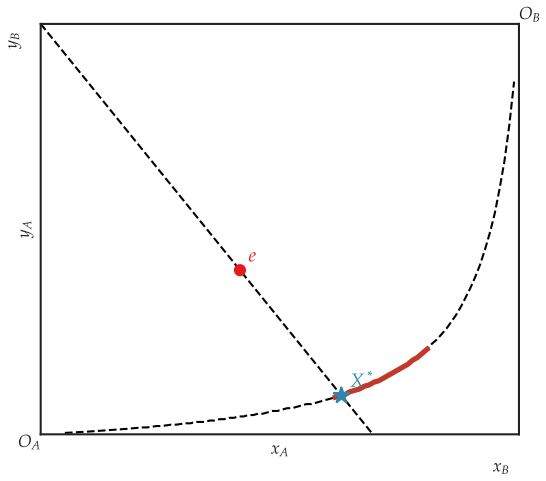

Figure 15
Contract curve and equilibrium.

Example 5

```python
from econ_viz import EdgeworthBox
from econ_viz.models import CobbDouglas

box = EdgeworthBox(
    CobbDouglas(alpha=0.8, beta=0.2),
    CobbDouglas(alpha=0.2, beta=0.8),
    total_x=12.0,
    total_y=10.0,
    title="Asymmetric Cobb-Douglas",
)

(
    box.add_endowment(5.0, 4.0)
    .add_contract_curve(n=100, method="mrs")
    .add_core()
    .add_price_line(px=1.2, py=1.0)
    .add_walrasian_equilibrium(px=1.2, py=1.0)
    .show_legend(loc="center left", bbox_to_anchor=(1.02, 0.5))
    .save("edgeworth.png")
)
```

6 Core models

The core models cover smooth, kinked, linear, income-dependent, and reference-point preferences. They all live in `econ_viz.models`. This chapter gives each model's function, parameters, distinct behaviour, and figure.

6.1 Smooth preferences

These models produce smooth indifference curves and normally have an interior optimum when prices and income are positive.

Cobb-Douglas

`CobbDouglas` `CobbDouglas(alpha=0.5, beta=0.5)`

The standard model for smooth, strictly convex preferences (Cobb & Douglas, 1928):

$$u(x, y) = x^{\alpha} y^{\beta}.$$

The exponents control the relative weight placed on each good. The indifference curves approach both axes without touching them.

`alpha` Weight on good x.
default `0.5`

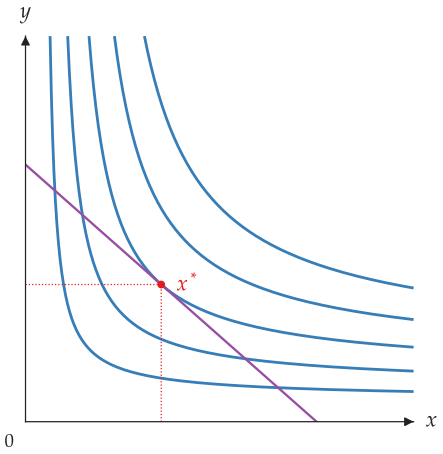

Figure 16
CES with $\rho = -0.5$.

`beta` Weight on good y.
default `0.5`

CES

`CES` `CES(alpha=0.5, beta=0.5, rho=0.5)`

Constant elasticity of substitution (Arrow et al., 1961): the ease of substitution varies while preferences stay smooth,

$$u(x,y) = (\alpha x^\rho + \beta y^\rho)^{1/\rho}.$$

The elasticity of substitution is $\sigma = 1/(1-\rho)$. As ρ changes, CES approaches Cobb-Douglas ($\rho \to 0$), Leontief ($\rho \to -\infty$) or perfect substitutes ($\rho \to 1$).

`alpha` Weight on good x.
default `0.5`

`beta` Weight on good y.
default `0.5`

`rho` Substitution parameter, $\rho \neq 1$.
default `0.5`

Translog

`Translog` `Translog(alpha_0=0.0, alpha_x=0.5, alpha_y=0.5, beta_xx=0.0, beta_yy=0.0, beta_xy=0.0)`

New: v1.1.0 A flexible log-quadratic model for smooth preferences (Christensen et al., 1975):

$$\ln u(x,y) = \alpha_0 + \alpha_x \ln x + \alpha_y \ln y + \frac{1}{2}\beta_{xx}(\ln x)^2 + \frac{1}{2}\beta_{yy}(\ln y)^2 + \beta_{xy} \ln x \ln y.$$

The quadratic and interaction terms let the curvature vary across the consumption space. With all β coefficients zero it reduces to a Cobb-Douglas-style log-linear form.

`alpha_0` Log-utility intercept.
default `0.0`

`alpha_x` First-order weight on $\ln x$.
default `0.5`

`alpha_y` First-order weight on $\ln y$.
default `0.5`

`beta_xx` Curvature in x.
default `0.0`

`beta_yy` Curvature in y.
default `0.0`

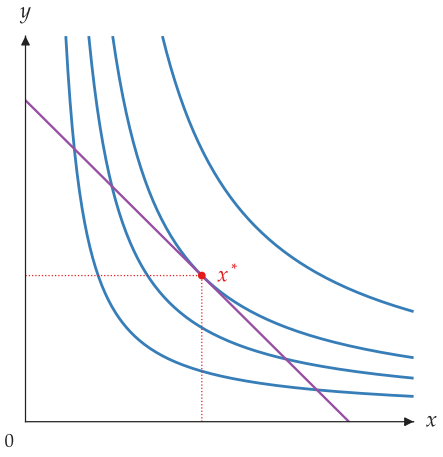

Figure 17
Translog with $\beta_{xy} = 0.12$.

`beta_xy` Interaction between x and y.
default `0.0`

6.2 Kinks and corners

These models show how non-smooth or linear preferences move the optimum to a kink or a corner.

Perfect complements

`Leontief` `Leontief(a=1.0, b=1.0)`

Goods consumed in fixed proportions:

$$u(x, y) = \min(ax, by).$$

The indifference curves are L-shaped, and the optimum lies at the kink where the two weighted quantities are equal.

`a` Weight on good x.
default `1.0`

`b` Weight on good y.
default `1.0`

Perfect substitutes

`PerfectSubstitutes` `PerfectSubstitutes(a=1.0, b=1.0)`

A constant rate of trade-off between the goods:

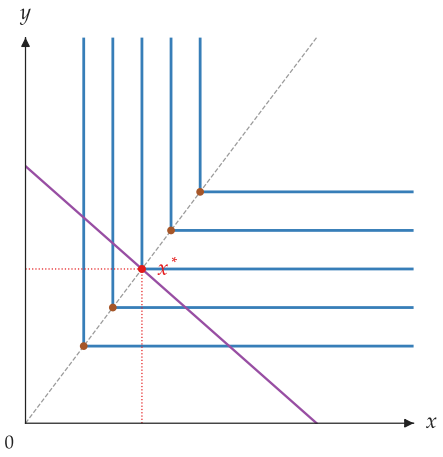

Figure 18
Leontief, with the ray through the kinks.

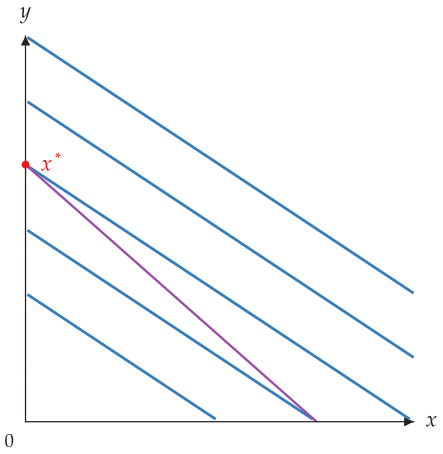

Figure 19
Perfect substitutes: a corner solution on the y-axis.

$$u(x, y) = ax + by.$$

The indifference curves are straight lines. The consumer normally buys only the good with the greater marginal utility per dollar, a corner solution.

`a` Marginal utility of good x.
default `1.0`

`b` Marginal utility of good y.
default `1.0`

Maximum utility

`maximum` `CustomUtility(func=lambda x, y: np.maximum(a * x, b * y))`

Maximum utility keeps only the larger weighted quantity in each bundle:

$$u(x, y) = \max(ax, by).$$

Each indifference curve has two arms that extend towards the axes. The preferences are non-convex, so on a linear budget the optimum normally lies at the intercept with the greater weighted utility. There is no dedicated class; the model is built with `CustomUtility` (Section 7).

`a` Weight on good x.
default `1.0`

`b` Weight on good y.
default `1.0`

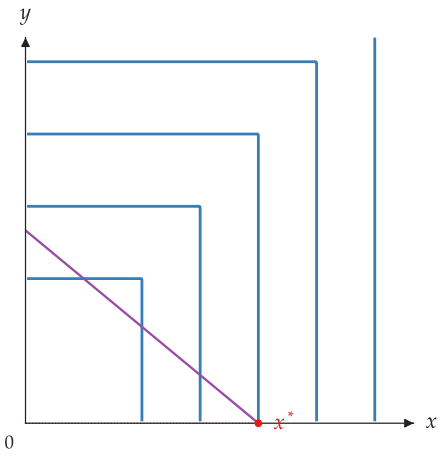

Figure 20
Maximum utility.

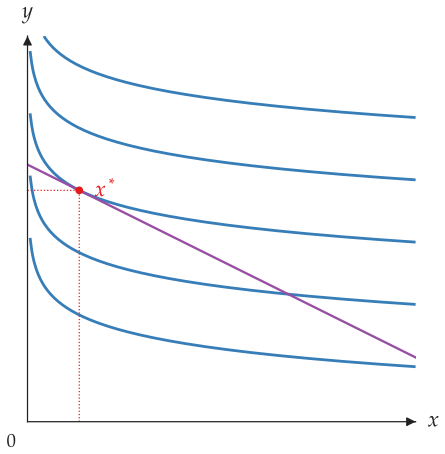

Figure 21
Quasi-linear preferences: vertically parallel indifference curves.

6.3 Income and reference points

These models add special income effects, subsistence requirements or a preferred consumption point.

Quasi-linear

`QuasiLinear` `QuasiLinear(v_func=numpy.log, linear_in="y")`

One good enters utility linearly:

$$u(x, y) = f(x) + y.$$

Once the solution is interior, the non-linear good has no income effect. `linear_in` reverses the roles of x and y.

`v_func` default `numpy.log` — Increasing, concave function f.

`linear_in` default `"y"` — Good that enters linearly.

Haagsma

`Haagsma` `Haagsma(alpha_x=1.0, alpha_y=2.0, gamma_x=2.0, gamma_y=27.0)`

New: v1.9.0

The Haagsma (2012) utility function,

$$u(x, y) = \alpha_x \ln(x - \gamma_x) - \alpha_y \ln\big(\gamma_y - y\big).$$

It is defined for $x > \gamma_x$ and $0 \le y < \gamma_y$, and requires $0 < \alpha_x < \alpha_y$. Marshallian demand for x falls with income, and x is a Giffen good when $\gamma_y p_y < I < \gamma_y p_y + \gamma_x p_x$.

Once income reaches $\gamma_y p_y + \gamma_x p_x$, utility has no finite maximum on the budget line and the model raises `InvalidParameterError`.

`alpha_x` default `1.0` — Weight on good x; must be less than `alpha_y`.

`alpha_y` default `2.0` — Weight on good y.

`gamma_x` default `2.0` — Lower bound on the quantity of x.

`gamma_y` default `27.0` — Upper bound on the quantity of y.

`demand(px, py, income)` — Closed-form interior solution.

`is_giffen(px, py, income)` — Whether x is a Giffen good at the given budget.

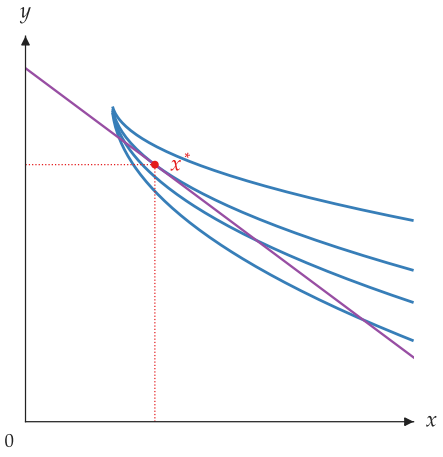

Figure 22
Haagsma with $\gamma_y = 10$.

Stone-Geary

`StoneGeary` `StoneGeary(alpha=0.5, beta=0.5, bar_x=1.0, bar_y=1.0)`

Cobb-Douglas with minimum consumption requirements (Geary, 1950; Stone, 1954):

$$u(x,y) = (x - \overline{x})^{\alpha}(y - \overline{y})^{\beta}.$$

The consumer first covers the subsistence quantities $\overline{x}$ and $\overline{y}$, then allocates the remaining income like a Cobb-Douglas consumer.

`alpha` default `0.5` — Weight on supernumerary x.

`beta` default `0.5` — Weight on supernumerary y.

`bar_x` default `1.0` — Subsistence quantity $\overline{x}$.

`bar_y` default `1.0` — Subsistence quantity $\overline{y}$.

Satiation

`Satiation` `Satiation(bliss_x=5.0, bliss_y=5.0, a=1.0, b=1.0)`

A bliss point that maximises utility:

$$u(x,y) = -a(x - x^*)^2 - b(y - y^*)^2.$$

Utility falls in every direction away from (x^*, y^*), so the indifference curves are closed ellipses and monotonicity does not hold.

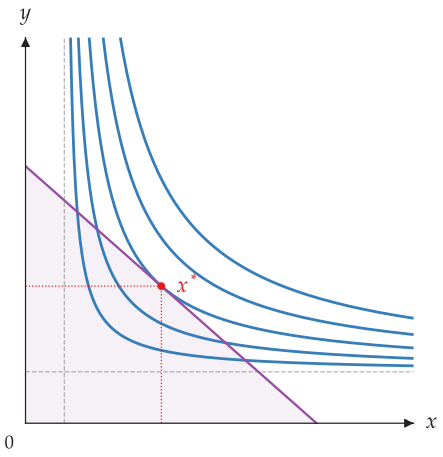

Figure 23
Stone-Geary with $\overline{x} = \overline{y} = 2$.

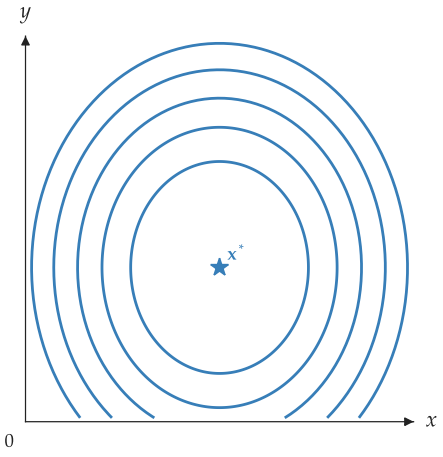

Figure 24
Satiation around the bliss point $(6, 4)$.

`bliss_x` Bliss-point coordinate x^*.
default `5.0`

`bliss_y` Bliss-point coordinate y^*.
default `5.0`

`a` Curvature along the x-axis.
default `1.0`

`b` Curvature along the y-axis.
default `1.0`

7 Advanced models

Advanced models extend the built-in families with user-defined functions or more than two goods.

Custom utility

`CustomUtility` `CustomUtility(func=⟨callable⟩, name=⟨str⟩)`

Wrap any vectorised Python callable as an econ-viz model. The callable must accept two NumPy arrays and return an array of the same shape. The example below uses

$$u(x, y) = \ln x + \ln y.$$

`func` Vectorised utility function of x and y.

`name` Display name of the model.

```python
import numpy as np
from econ_viz import Canvas, levels, solve
from econ_viz.models import CustomUtility

model = CustomUtility(
    func=lambda x, y: np.log(x) + np.log(y),
    name="log+log",
)
eq = solve(model, px=2.0, py=3.0, income=30.0)

(
    Canvas(x_max=20, y_max=15, title="Custom Utility")
    .add_utility(model, levels=levels.around(eq.utility, n=5))
    .add_budget(2.0, 3.0, 30.0)
    .add_equilibrium(eq)
    .save("custom.png")
)
```

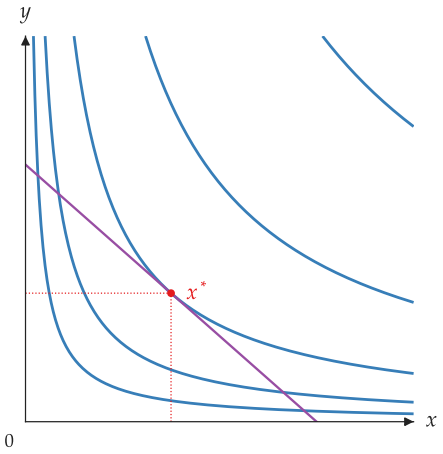

Figure 25
A custom model, $\ln x + \ln y$.

Many-good Cobb-Douglas

`MultiGoodCD` `MultiGoodCD(⟨shares⟩)`

`MultiGoodCD.freeze` `.freeze(⟨good⟩=⟨quantity⟩, ...)`

Cobb-Douglas preferences over N goods:

$$u(x_1, \ldots, x_N) = \prod_{i=1}^{N} x_i^{\alpha_i}.$$

`freeze()` fixes every good except x and y and returns a `CustomUtility` that can be drawn on a two-dimensional canvas.

`shares` Mapping from each good's name to its exponent α_i.

`freeze(...)` Fixed quantities of the goods other than x and y.

```
from econ_viz import Canvas, levels, solve
from econ_viz.models import MultiGoodCD

model = MultiGoodCD({"x": 0.3, "y": 0.3, "z": 0.4})
two_good_model = model.freeze(z=10.0)
eq = solve(two_good_model, px=2.0, py=3.0, income=30.0)

(
    Canvas(x_max=20, y_max=15, title="Multi-Good Cobb-Douglas")
    .add_utility(
        two_good_model,
        levels=levels.around(eq.utility, n=5),
    )
    .add_budget(2.0, 3.0, 30.0, fill=True)
    .add_equilibrium(eq)
    .save("multigood.png")
)
```

8 Analysis

The analysis API provides comparative statics of Marshallian demand, the Slutsky matrix, and checks for homogeneity and homotheticity of utility functions.

8.1 Comparative statics

`comparative_statics` `comparative_statics(⟨model⟩, px=⟨float⟩, py=⟨float⟩, income=⟨float⟩)`

New: v1.1.0 Estimate the six partial derivatives of two-good Marshallian demand with respect to p_x, p_y and income by finite differences, returned as a `ComparativeStatics`.

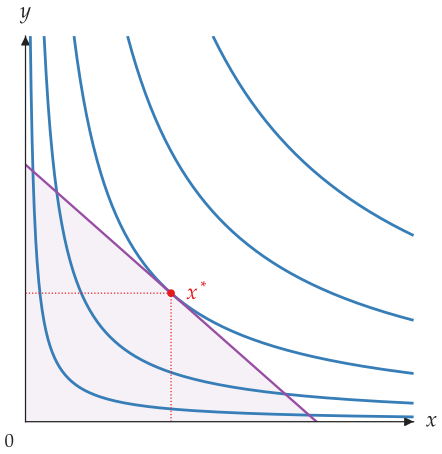

Figure 26
A three-good Cobb-Douglas model with $z = 10$ fixed.

- Central finite differences around `solve(...)`, with a default relative step of `1e-3`.
- Warns when the own-price derivative is positive or the income derivative negative, the signatures of Giffen and inferior goods.

`dx_dpx` *float* — $\partial x^*/\partial p_x$.

`dx_dpy` *float* — $\partial x^*/\partial p_y$.

`dx_dI` *float* — $\partial x^*/\partial I$.

`dy_dpx` *float* — $\partial y^*/\partial p_x$.

`dy_dpy` *float* — $\partial y^*/\partial p_y$.

`dy_dI` *float* — $\partial y^*/\partial I$.

```
from econ_viz.models import CobbDouglas
from econ_viz.optimizer import comparative_statics

model = CobbDouglas(alpha=0.4, beta=0.6)
cs = comparative_statics(model, px=2.0, py=3.0, income=60.0)

print(round(cs.dx_dpx, 1), round(cs.dx_dpy, 1), round(cs.dx_dI, 1))
print(round(cs.dy_dpx, 1), round(cs.dy_dpy, 1), round(cs.dy_dI, 1))

# -6.0 0.0 0.2
# 0.0 -4.0 0.2
```

8.2 Slutsky matrix

`slutsky_matrix` `slutsky_matrix(⟨model⟩, px=⟨float⟩, py=⟨float⟩, income=⟨float⟩)`

Updated: v1.2.3

The two-good Slutsky substitution matrix, computed from Marshallian demand derivatives and income effects. The result checks symmetry, negative semidefiniteness and homogeneity, and warns when a condition fails. `as_array()` returns it as a NumPy array.

```
from econ_viz import slutsky_matrix
from econ_viz.models import CobbDouglas

S = slutsky_matrix(
    CobbDouglas(alpha=0.4, beta=0.6),
    px=2.0, py=3.0, income=60.0,
)
```

```
print(round(S.s_xx, 1), round(S.s_xy, 1))
print(round(S.s_yx, 1), round(S.s_yy, 1))
print(S.as_array().round(1))

# -3.6 2.4
# 2.4 -1.6
# [[-3.6  2.4]
#  [ 2.4 -1.6]]
```

8.3 Homogeneity

`HomogeneityAnalyzer` `HomogeneityAnalyzer(⟨model⟩)`

New: v1.1.0 Check homogeneity and homotheticity of a utility function, and degree-zero homogeneity of demand.

`degree()` Estimate the degree of homogeneity; see Section 8.4.

`euler_check(x, y)` Euler-theorem residual at a bundle.

`is_homothetic()` Whether the MRS is invariant to proportional scaling.

`demand_degree_zero(px, py, income)` Whether Marshallian demand is homogeneous of degree zero.

```
from econ_viz.analysis import HomogeneityAnalyzer
from econ_viz.models import CobbDouglas

analyzer = HomogeneityAnalyzer(CobbDouglas(alpha=0.4, beta=0.6))
result = analyzer.degree()

print(round(result.degree, 6))
print(result.returns_to_scale)
print(round(analyzer.euler_check(3.0, 4.0), 6))
print(analyzer.is_homothetic())
print(analyzer.demand_degree_zero(px=2.0, py=3.0, income=60.0))

# 1.0
# ReturnsToScale.CONSTANT
# 0.0
# True
# True
```

8.4 Returns to scale

`degree()` returns a `HomogeneityResult` with these fields:

`degree` *float* Estimated degree of homogeneity.

`is_homogeneous` *bool* Whether the function is homogeneous.

`returns_to_scale` *ReturnsToScale* Returns-to-scale class, one of the values below.

For Cobb-Douglas utility the degree is $\alpha + \beta$:

$$u(\lambda x, \lambda y) = \lambda^{\alpha+\beta} u(x, y).$$

`INCREASING` $\alpha + \beta > 1$, increasing returns.

`CONSTANT` $\alpha + \beta = 1$, constant returns.

`DECREASING` $\alpha + \beta < 1$, decreasing returns.

`NOT_HOMOGENEOUS` No consistent degree of homogeneity.

```
def shifted_utility(x, y):
    return x**0.4 * y**0.6 + 1.0


models = [
```

```
    CobbDouglas(alpha=0.7, beta=0.6),
    CobbDouglas(alpha=0.4, beta=0.6),
    CobbDouglas(alpha=0.2, beta=0.5),
    shifted_utility,
]

for model in models:
    result = HomogeneityAnalyzer(model).degree()
    degree = None if result.degree is None else round(result.degree, 1)
    print(degree, result.returns_to_scale.name)

# 1.3 INCREASING
# 1.0 CONSTANT
# 0.7 DECREASING
# None NOT_HOMOGENEOUS
```

9 Themes and configuration

A `Theme` defines the default look of a diagram; a `Config` loads theme overrides and font settings from a TOML file. `Canvas`, `Figure`, `DemandDiagram`, price-effect decompositions and `EdgeworthBox` share these style objects.

9.1 Built-in themes

themes

```
from econ_viz import Canvas, themes

cvs = Canvas(theme=themes.paper)
```

Updated: v1.11.0

The Python API provides the themes below. Theme objects are immutable; to change a field, create a new `Theme` or override the field with a `Config`.

Table 4 Built-in themes

Name	Character
`default`	Colour-blind-friendly default palette
`colorblind`	Alias with the same palette as `default`
`nord`	Cool theme on the Nord palette
`paper`	Transparent background and fine lines for print
`monochrome`	Black and white; layers told apart by line style and marker
`presentation`	Larger labels, lines and markers for projection
`dark`	Dark background with matching foreground colours

The Python API and the `base` key of a configuration file accept all seven names[5].

The default theme takes its colours from a colour-blind-friendly palette (thriveth, 2014). `themes.COLORBLIND_CYCLE_HEX` and `themes.COLORBLIND_CYCLE_RGB` give the palette as hexadecimal and RGB values. Economics teaching leans heavily on diagrams, so meaning should never rest on colour alone, or visually impaired students cannot follow it (Kugler & Andrews, 1996).

[5] Passed directly to `--theme` on the command line, only the six themes other than the `colorblind` alias are accepted.

9.2 Custom themes

`Theme`

```
from econ_viz import Theme

my_theme = Theme(
    name="custom",
    background_color="white",
    label_scale=1.0,
    axis_color="#333333",
    label_color="#333333",
    ic_color="#2563eb",
    secondary_ic_color="#93c5fd",
    secondary_ic_opacity=0.45,
    budget_color="#dc2626",
    eq_color="#16a34a",
)
```

Updated: v1.12.0

A `Theme` holds default colours, widths and styles for diagram elements. Fields not given keep the class defaults; style arguments passed explicitly to a drawing method take precedence over the theme.

`name`
str
Theme identifier.

`background_color`
str | *None* · default `None`
Background of the figure and axes; `None` keeps it transparent.

`label_scale`
float · default `1.0`
Multiplier for ordinary label sizes.

`axis_color`
str · default `"#222222"`
Axis colour.

`label_color`
str · default `"#222222"`
Colour of axis labels and origin text.

`ic_color`
str · default `"#377EB8"`
Indifference-curve colour.

`ic_linewidth`
float · default `1.8`
Indifference-curve width.

`secondary_ic_color`
str | *None* · default `None`
Colour of non-focal indifference curves; `None` uses `ic_color`.

`secondary_ic_linewidth`
float · default `1.0`
Width of non-focal indifference curves.

`secondary_ic_opacity`
float · default `0.45`
Opacity of non-focal indifference curves.

`path_color`
str · default `"#4DAF4A"`
Colour of PCC and ICC paths.

`budget_color`
str · default `"#984EA3"`
Budget-line colour.

`budget_fill_alpha`
float · default `0.08`
Opacity of the budget-set shading.

`eq_color`
str · default `"#E41A1C"`
Colour of the equilibrium and its drop lines.

`eq_markersize`
float · default `4.0`
Equilibrium marker size.

`Theme` also holds colours and widths for rays, kinks, compensated budget lines and price-effect arrows. `axis_stroke`, `drop_stroke`, `projection_stroke`, `guide_stroke` and `box_stroke` hold the defaults of other lines; markers, labels, fills and legends come from `Marker`, `Label`, `Fill` and `Legend` properties (Section 4.3).

9.3 Configuration files

Config.load
Config.use
Config.reset

```
from econ_viz import Config

Config.load("econ-viz.toml").use()
# diagrams created from here on use these settings

Config.reset()
# back to the built-in defaults
```

New: v1.10.0 `Config.load()` reads a TOML file and returns a `Config`; `use()` makes it the default for later diagrams, and `reset()` restores the built-in defaults.

Tutorial: one style for every diagram

Figures in one set of lecture notes or one paper usually share their colours and line styles. Rather than repeating the same arguments on every `Canvas`, write the style once in an `econ-viz.toml` at the project root, where both Python and the command line read it.

1. **Create a template.** Run the following in the project root. It writes a commented `econ-viz.toml` listing the names and fields each section accepts:
   ```
   econ-viz init
   ```
   Every setting in the template starts commented out; keep only what you change and delete the rest.
2. **Pick a base theme.** `base` names the built-in theme to start from (Table 4); everything else overrides it:
   ```
   base = "paper"
   ```
3. **Override colours and styles.** Write only the fields you change:
   ```
   [color]
   ic = "#1B4F72"
   eq = "#C0392B"

   [stroke.budget]
   width = 1.6
   style = "dashed"

   [marker.eq]
   size = 5

   [label.point]
   fontsize = 11
   position = "right"
   ```
   Section names map to theme properties: `ic` under `[color]` sets `ic_color`, and `[stroke.budget]` sets `theme.budget_stroke`. The fields inside a section are the arguments of `Stroke`, `Marker`, `Label`, `Fill` or `Legend` (Section 4.3). Fonts go under `[font]` as `text` and `math`. Fields you leave out keep the base theme's values.
4. **Use it from Python.** Load it once, before creating any diagram:
   ```
   from econ_viz import Config

   Config.load().use()   # reads ./econ-viz.toml by default
   ```
   Every `Canvas`, `Figure`, `DemandDiagram` and `EdgeworthBox` created afterwards uses these settings; `Config.reset()` restores the defaults.
5. **Use it from the command line.** Pass the file with `--config`:
   ```
   econ-viz plot --config econ-viz.toml --model cobb-douglas \
     --px 2 --py 3 --income 30 --output figure.png
   ```

Settings apply in this order, highest first: arguments passed to `Canvas` or a drawing method, then the configuration file, then the base theme. For example, `Canvas(theme=themes.default)` ignores the file's theme.

A mistake in the file makes `Config.load()` raise `InvalidParameterError` with the valid names. For instance, writing `[stroke.budgt]` for `[stroke.budget]` gives:

```
econ-viz.toml: [stroke.budgt]: no such setting (choose: axis, box,
budget, ...)
```

10 Export

10.1 Output formats

Canvas.save `cvs.save(⟨path⟩)`

Write the figure to `path`; the format follows the extension. `Figure`, `DemandDiagram` and `EdgeworthBox` have the same method.

- PNG: raster, at the canvas `dpi` (default 300).
- PDF, SVG: vector output that stays sharp when printed or scaled.
- TeX: TikZ code (`.tex`) for inclusion in LaTeX documents; adjust it with the keyword arguments `tikz_scale` and `tikz_standalone`.

```
cvs.save("figure.png")   # PNG, canvas DPI (default 300)
cvs.save("figure.pdf")   # PDF (vector)
cvs.save("figure.svg")   # SVG (vector)
cvs.save("figure.tex")   # TikZ
```

The `dpi` parameter of `Canvas` only affects raster output; lower it for faster previews:

```
cvs = Canvas(x_max=20, y_max=15, dpi=150)   # lower DPI, faster preview
```

10.2 Animated GIFs

Export a GIF with `Animator.save()`; Section 12 has complete examples.

```
from econ_viz.animation import Animator

Animator(draw_frame, frames=frames).save("animation.gif", fps=12, dpi=120)
```

10.3 Interactive window

`cvs.show()` opens a live matplotlib window instead of saving. In the CLI, omitting `--output` has the same effect (Section 11.2):

```
econ-viz plot --model cobb-douglas --px 2 --py 3 --income 30
```

11 Command-line interface

The command-line interface plots diagrams, runs batches and prints demand formulas. Install it as a standalone tool with `uv tool install econ-viz`, or prefix each command with `uv run` inside a uv project (Section 2.5).

Table 5 CLI commands

Command	Description
`econ-viz help [⟨command⟩]`	Show help for the CLI or a specific command
`econ-viz models`	List all supported utility models
`econ-viz plot ...`	Generate and export a diagram
`econ-viz solve-tex ...`	Print a closed-form Marshallian demand as TeX
`econ-viz init [path]`	Create an `econ-viz.toml` configuration template

11.1 Help and models

`econ-viz help` `econ-viz help [⟨command⟩]`

Without an argument, list all commands; with one, show its options.

```
econ-viz help          # all commands
econ-viz help plot     # plot options
econ-viz help models   # models options
```

`econ-viz models` List the model names and parameters the CLI supports. `--model` takes kebab-case names, which differ from the Python class names (Table 6). The parameter options are described in Section 11.2.

Table 6 CLI models

Model	`--model` value	Required	Optional
Cobb-Douglas	`cobb-douglas`	`alpha`, `beta`	–
CES	`ces`	`rho`	`alpha`, `beta`
Perfect complements	`leontief`	`a`, `b`	–
Perfect substitutes	`perfect-substitutes`	`a`, `b`	–
Satiation	`satiation`	`bliss-x`, `bliss-y`	`a`, `b`
Quasi-linear	`quasi-linear`	–	`v-func`, `linear-in`
Stone-Geary	`stone-geary`	–	`alpha`, `beta`, `bar-x`, `bar-y`
Translog	`translog`	–	`alpha-0`, `alpha-x`, `alpha-y`, `beta-xx`, `beta-yy`, `beta-xy`

11.2 Plotting

`econ-viz plot` `econ-viz plot --model ⟨name⟩ ⟨options⟩`
`econ-viz plot --latex ⟨expression⟩ ⟨options⟩`

Give the model with `--model`, or the utility function with `--latex`, not both. With `--output` the diagram is written to a file; without it an interactive window opens.

The budget line and equilibrium need both prices and income. `--no-budget`, `--no-equilibrium` and `--no-curves` each omit one layer and can be combined (Table 7).

Table 7 Layer options

Case	Result
`--px`, `--py` and `--income` all given	Budget line drawn and equilibrium solved
Any price or income missing	Indifference curves only, no budget line or equilibrium
`--no-equilibrium` added	No solve and no equilibrium, even with prices and income
`--no-budget` added	Budget line hidden, even with prices and income
`--no-curves` with prices and income	Budget line and equilibrium only, no indifference curves

```
# named model
econ-viz plot --model cobb-douglas --alpha 0.5 --beta 0.5 ...

# LaTeX expression
econ-viz plot --latex "x^{0.4} y^{0.6}" ...
```

Example 6

```
# Cobb-Douglas, shaded budget set
econ-viz plot --model cobb-douglas --alpha 0.5 --beta 0.5 \
              --px 2 --py 3 --income 30 \
              --fill --output cobb_douglas.png

# LaTeX input, Nord theme, expansion path
econ-viz plot --latex "x^{0.4} y^{0.6}" \
              --px 2 --py 3 --income 30 \
              --theme nord --show-ray \
              --output cd_latex.png

# perfect complements, larger canvas
econ-viz plot --model leontief --a 1 --b 2 \
              --px 2 --py 3 --income 30 \
              --x-max 20 --y-max 15 \
              --output leontief.png

# CES, curves only
econ-viz plot --model ces --rho -0.5 \
              --x-max 20 --y-max 15 --n-curves 6 \
              --no-budget --no-equilibrium \
              --output ces.png

# satiation (bliss point)
econ-viz plot --model satiation --bliss-x 6 --bliss-y 4 \
              --x-max 12 --y-max 10 \
              --no-budget --no-equilibrium \
              --output satiation.png

# no --output: open a window
econ-viz plot --model cobb-douglas --px 2 --py 3 --income 30
```

Model selection

`--model, -m` Model name; `econ-viz models` lists them all.

`--latex, -l` LaTeX expression for Cobb-Douglas, perfect complements, perfect substitutes or CES (Section 14).

Prices and income

`--px` Price of good x.

`--py` Price of good y.

`--income` Consumer income.

Model parameters

`--alpha` Alpha parameter (Cobb-Douglas, CES). default `0.5`

`--beta` Beta parameter (Cobb-Douglas, CES). default `0.5`

`--a` Parameter a (perfect complements, perfect substitutes, satiation). default `1.0`

`--b` Parameter b (perfect complements, perfect substitutes, satiation). default `1.0`

`--rho` Substitution parameter (CES). default `0.5`

`--bliss-x` Bliss-point x-coordinate (satiation). default `5.0`

`--bliss-y` Bliss-point y-coordinate (satiation). default `5.0`

Quasi-linear, Stone-Geary and Translog take these further options:

`--v-func` Non-linear function of the quasi-linear model, `log` or `sqrt`.
default `log`

`--linear-in` Good that enters the quasi-linear model linearly, `x` or `y`.
default `y`

`--bar-x` Stone-Geary subsistence quantity $\overline{x}$.
default `1.0`

`--bar-y` Stone-Geary subsistence quantity $\overline{y}$.
default `1.0`

`--alpha-0` Translog intercept.
default `0.0`

`--alpha-x` Translog first-order weight on $\ln x$.
default `0.5`

`--alpha-y` Translog first-order weight on $\ln y$.
default `0.5`

`--beta-xx` Translog curvature in x.
default `0.0`

`--beta-yy` Translog curvature in y.
default `0.0`

`--beta-xy` Translog interaction between x and y.
default `0.0`

```
econ-viz plot --model stone-geary --bar-x 2 --bar-y 2 \
              --px 2 --py 3 --income 30 \
              --x-max 20 --y-max 15 --output stone_geary.png
```

Subsistence spending here is $2 \times 2 + 3 \times 2 = 10$, below the income of 30. With an income of 10 or less, the solve fails because the budget lies outside the model's domain.

Canvas and layers

`--x-max` Horizontal axis limit.
default `10`

`--y-max` Vertical axis limit.
default `10`

`--x-label` Horizontal axis label.
default `x`

`--y-label` Vertical axis label.
default `y`

`--title` Figure title.

`--theme` Theme name: `default`, `nord`, `paper`, `monochrome`, `presentation` or `dark` (Section 9).
default `default`

`--config` Path to a TOML configuration file. Command-line options take precedence over the file.

`--n-curves` Number of indifference curves.
default `5`

`--dpi` Raster output resolution.
default `300`

`--fill` Shade the feasible set below the budget line.

`--show-ray` Draw the expansion-path ray through the optimum.

`--no-budget` Omit the budget line.

`--no-equilibrium` Omit the equilibrium point.

`--no-curves` Omit the indifference curves.

`--output, -o` Output file (`.png`, `.pdf`, `.svg`, `.tex`); omit it to open an interactive window.
For batch runs, give each parameter set its own file name and create the output directory first. `--x-max` and `--y-max` do not follow the prices; when an intercept or the optimum falls outside the canvas, raise the limits as well.

11.3 Creating a configuration file

`econ-viz init` `econ-viz init [⟨path⟩] [--force]`

New: v1.10.0 Write a commented `econ-viz.toml` template. Without a path it goes in the current directory; an existing file is overwritten only with `--force`.

```
econ-viz init
econ-viz init config/figures.toml
econ-viz plot --config econ-viz.toml --model cobb-douglas \
              --px 2 --py 3 --income 30 --output figure.png
```

11.4 Demand formulas

`econ-viz solve-tex` `econ-viz solve-tex --model ⟨name⟩ ⟨options⟩`

Print the closed-form Marshallian demand as TeX, for any document that supports TeX math.

```
# numeric parameters
econ-viz solve-tex --model cobb-douglas --alpha 0.4 --beta 0.6

# symbolic parameters
econ-viz solve-tex --model cobb-douglas --symbolic-params

# custom price and income symbols
econ-viz solve-tex --model leontief --a 2 --b 3 \
                   --px-symbol p_1 --py-symbol p_2 --income-symbol M
```

Supports `cobb-douglas`, `leontief`, `perfect-substitutes` and their LaTeX expressions. The command prints the formula only and writes no figure.

`--symbolic-params` Keep model parameters as symbols, such as α and β; without it the given values are substituted.

`--px-symbol` Symbol for the price of x. default `p_x`

`--py-symbol` Symbol for the price of y. default `p_y`

`--income-symbol` Symbol for income. default `I`

12 Animation

`Animator` calls a drawing function over a sequence of parameter values and exports the returned `Canvas` or `Figure` objects as a GIF. It needs the `animation` extra (Section 2.4).

`Animator` `Animator(⟨draw⟩, frames=⟨values⟩).save(⟨path⟩, fps=12, dpi=120, loop=0)`

New: v1.4.0 Call `draw(value)` for every value in `frames` and write the canvases as the frames of a GIF.

- `save()` uses Pillow, so no `ffmpeg` dependency is required.
- Frames are composited onto white before export, which prevents frame-stacking artefacts.
- `fps`, `dpi` and `loop` set the frame rate, resolution and loop count.

Example 7

```python
import numpy as np

from econ_viz import Canvas, levels, solve
from econ_viz.animation import Animator
from econ_viz.models import CobbDouglas

def draw(px: float) -> Canvas:
    model = CobbDouglas(alpha=0.5, beta=0.5)
    eq = solve(model, px=px, py=2.0, income=20.0)
    lvls = levels.around(eq.utility, n=5)

    return (
        Canvas(
            x_max=14, y_max=12,
            x_label="X_1", y_label="X_2",
            title="Price sweep"
        )
        .add_utility(model, levels=lvls)
        .add_budget(px=px, py=2.0, income=20.0, fill=True)
        .add_equilibrium(eq, show_ray=True, drop_dashes=True)
    )

Animator(draw, frames=np.linspace(1.0, 6.0, 45)).save(
    "price_sweep.gif",
    fps=12,
    dpi=120,
)
```

`examples/scripts/animation.py` provides four examples[6]:

- Parameter sweep: prices and income fixed, a utility-function parameter varies.
- Price sweep: utility function and p_y fixed, p_x varies and the budget line rotates, as in the example above.
- Income sweep: utility function and prices fixed, income varies and the budget line shifts.
- Budget-only sweep: the budget line alone, without the utility layers.

The four differ only in which values `draw()` holds fixed and which one it varies. For the parameter sweep, let the `draw()` above take `alpha` and fix the price at $p_x = 2$:

```python
def draw(alpha: float) -> Canvas:
    model = CobbDouglas(alpha=alpha, beta=1.0 - alpha)
    eq = solve(model, px=2.0, py=2.0, income=20.0)
    ...

Animator(draw, frames=np.linspace(0.1, 0.9,
45)).save("parameter_sweep.gif")
```

For the income sweep, `draw()` takes `income` and passes it to `add_budget()` and `solve()`. For the budget line alone, leave out `add_utility()` and `add_equilibrium()`.

Figure 27 to Figure 29 each show four frames.

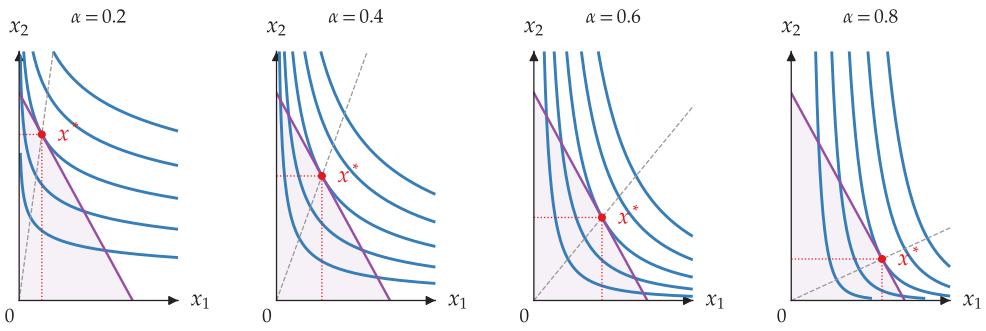

Figure 27
Parameter sweep: α varies with $\beta = 1 - \alpha$.

[6] Each example covers Cobb-Douglas, CES, perfect substitutes and perfect complements.

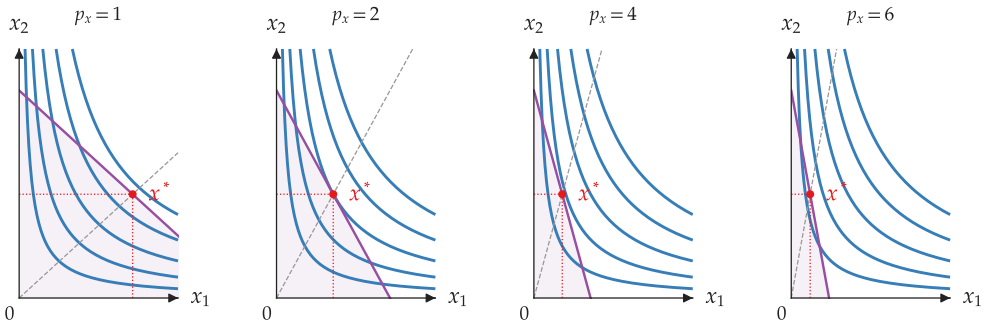

Figure 28
Price sweep with $p_y = 2$ and $I = 20$ fixed.

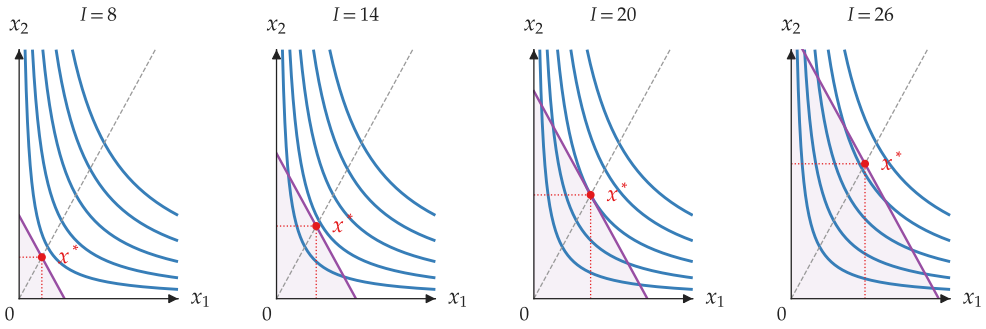

Figure 29
Income sweep with $p_x = p_y = 2$ fixed.

13 Interactive widgets

`WidgetViewer` provides sliders and numeric input boxes in Jupyter and redraws the figure when a parameter changes. It needs the `interactive` extra (Section 2.4).

`WidgetViewer` `WidgetViewer(⟨draw⟩, ⟨name⟩=(⟨min⟩, ⟨max⟩, ⟨step⟩), ...).show()`

New: v1.4.0

Pass a drawing function `draw` and give each parameter's range as `(min, max, step)`. Each parameter gets a linked `FloatSlider` and `FloatText`; when a value changes, `draw` is called again and its figure replaces the old one in the cell.

`draw` follows the same contract as for `Animator` (Section 12), but may take several parameters. Starting from the price-sweep example there, let `draw()` also take `alpha`:

Example 8

```
from econ_viz.interactive import WidgetViewer

def draw(alpha: float, px: float) -> Canvas:
    model = CobbDouglas(alpha=alpha, beta=1.0 - alpha)
    ...  # rest as in the animation example

WidgetViewer(
    draw,
    alpha=(0.2, 0.8, 0.05),
    px=(1.0, 6.0, 0.25),
).show()
```

13.1 Notebook setup

In a fresh Jupyter or Colab runtime, run the steps in this order[7]:

1. Run the install cell.
2. If the install upgraded ipywidgets, traitlets or IPython, restart the runtime.
3. After the restart, continue from the import cell without reinstalling.

[7] The Playground notebook in the econ-viz repository, `notebook/econ-viz Playground.ipynb`, follows this flow; open it in Jupyter, VS Code or Colab. It does not reinstall when econ-viz is already present.

13.2 Widgets or GIFs

Use `WidgetViewer` when users should adjust the parameters themselves; use `Animator` to export a GIF when a fixed sweep should play in slides or on a web page (Section 12).

14 LaTeX parsing

`parse_latex` `parse_latex(⟨expression⟩)`

Parse a LaTeX utility function into a model, usable with `solve()`, `Canvas` or any other API that takes a model.

```
from econ_viz import parse_latex

model = parse_latex(r"x^{0.4} y^{0.6}")  # CobbDouglas(alpha=0.4,
beta=0.6)
model = parse_latex(r"\min(2x, 3y)")     # Leontief(a=2.0, b=3.0)
model = parse_latex(r"2x + 3y")          # PerfectSubstitutes(a=2.0,
b=3.0)
```

14.1 Supported forms

Table 8 lists the accepted forms. Omitted coefficients and exponents default to 1.

Table 8 Utility functions recognised by `parse_latex`.

Family	Pattern	Example
Cobb-Douglas	`x^{α} y^{β}` or `x^α y^β`	`x^{0.3} y^{0.7}`
Perfect complements	`\min(ax, by)` or `min(ax, by)`	`\min(2x, y)`
Perfect substitutes	`ax + by`	`3x + 1.5y`
CES	`(αx^ρ + βy^ρ)^{1/ρ}`	`(0.4x^{-0.5}+0.6y^{-0.5})^{-2}`

A leading `U =` or `u(x, y) =` may be omitted:

```
U(x,y) = x^{0.5} y^{0.5}
U = x^{0.5} y^{0.5}
x^{0.5} y^{0.5}
```

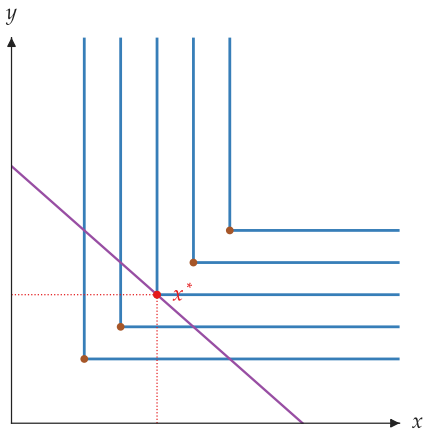

Figure 30 `parse_latex(r"\min(2x, 3y)")`.

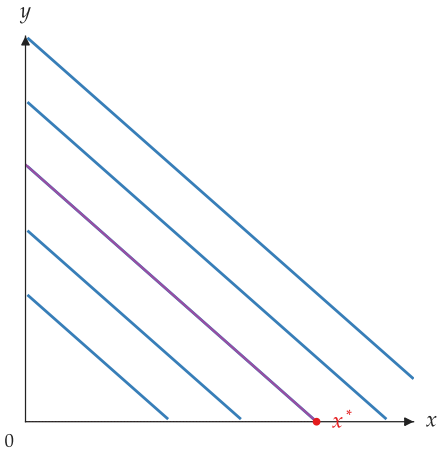

Figure 31 `parse_latex(r"2x + 3y")`.

14.2 Errors

Unparseable input raises `econ_viz.exceptions.ParseError`, whose message lists the supported forms:

```
from econ_viz import parse_latex
from econ_viz.exceptions import ParseError

try:
    model = parse_latex(r"x^2 + y^2")
except ParseError as e:
    print(e)

# Unrecognised LaTeX utility function: 'x^2 + y^2'
# Supported forms:
#   Cobb-Douglas       : x^{alpha} y^{beta}
#   Leontief           : \min(ax, by)
#   Perfect Substitutes: ax + by
#   CES                : (alpha x^{rho} + beta y^{rho})^{1/rho}
```

14.3 In the command-line tool

`econ-viz plot --latex` uses the same parser and accepts the same input (Section 11.2):

```
econ-viz plot --latex "x^{0.4} y^{0.6}" --px 2 --py 3 --income 30 -o
out.png
```

Change history

Release history of econ-viz. Documentation-only changes are not listed; each entry is tagged where the feature it describes is documented, following l3doc convention, so the page number is the real page.

v1.12.0 (2026-09-26)

General: Focal and secondary indifference-curve levels, with numeric or ordinal labels ····· 2

`Canvas.add_utility`: Focal utility level, secondary curve style and ordinal labels ····· 6

`Theme`: Colour, width and opacity of secondary indifference curves ····· 29

v1.11.0 (2026-09-26)

General: Complete theme system: one visual language across canvases, multi-panel figures, demand diagrams, decompositions and Edgeworth boxes ····· 2

`Canvas`: Backgrounds, labels, lines and markers all take their defaults from the theme ····· 6

`Figure`: Panel backgrounds, labels, lines and markers fully follow the theme ····· 14

`EdgeworthBox`: Box frame, contract curve, core, endowment, price line and Walrasian equilibrium fully follow the theme ····· 18

`Theme`: Complete theme system: background colour, label scale and defaults for every economic layer ····· 29

`themes`: New `paper`, `monochrome`, `presentation` and `dark` themes ····· 29

`Config`: Configuration files accept every built-in theme and keep the same visual meaning across diagram types ····· 31

`econ-viz plot`: `--theme` accepts every built-in CLI theme ····· 32

v1.10.1 (2026-09-26)

General: Ships `py.typed`; CI runs Ruff and Mypy ····· 2

v1.10.0 (2026-09-26)

`econ-viz`: Python 3.10 reads TOML configuration files with tomli ····· 2

`Figure`: Falls back to the active `Config` when no theme or font is given ····· 14

`EdgeworthBox`: Falls back to the active `Config` when no theme or font is given ····· 18

`Config`: `Config`, `econ-viz.toml`, `econ-viz init` and `plot --config` ····· 31

`econ-viz init`: Configuration-template command and `plot --config` ····· 32

v1.9.0 (2026-09-26)

`Canvas`: Axes, origin, titles and other text elements accept a `Label` ····· 6

`Stroke`: New `opacity` field ····· 11

`DemandDiagram`: Supports `Legend`, `Marker`, `Label` and per-layer `Stroke` ····· 15

`Canvas.add_decomposition`: Draws the indifference curves through A, B and C by default, with curve labels and `Legend` ····· 17

Effect arrows below the horizontal axis point A→B and B→C; zero effects draw no arrow ···· 17

`Haagsma`: Haagsma utility model with a closed-form solution, for inferior- and Giffen-good analysis ····· 23

v1.8.0 (2026-09-26)

`Canvas`: `Axis`, `Marker`, `Label` and `Fill` style objects ····· 6

`Stroke`: Every line style is a `Stroke`; the individual style arguments remain as shorthands ····· 11

`Canvas.add_decomposition`: `Effect`, `Marker` and `Label` style objects ····· 17

`EdgeworthBox`: Markers, text and axes accept `Marker`, `Label` and `Axis` ····· 18

v1.7.0 (2026-09-25)

General: CI tests Python 3.10–3.13, enforces branch coverage, and smoke-tests runnable examples ····· 2

Tagged releases publish only after tests pass ···· 2

`econ-viz`: Package management and builds migrated to uv ····· 2

All example scripts run from a clean checkout · 4

`econ-viz --version` prints the installed package version ····· 32

`Canvas`: Per-figure text and math fonts without changing matplotlib's global configuration ····· 6

Independent axis-label positions, line styles and arrowhead styles for `Canvas` and `Figure` ····· 11

`Stroke`: One `Stroke` controls line width, style, colour and arrowheads across canvases, figures, demand diagrams and Edgeworth boxes ····· 11

`CobbDouglas`: Fixed utility domains, asymmetric CES expansion paths, Cobb-Douglas limit cases, and satiation optima that leave income unspent · 19

Fixed contour levels when utility is near zero or negative ····· 19

`comparative_statics`: Comparative statics no longer step outside the domain near zero prices or income, or near subsistence constraints ····· 26

v1.6.0 (2026-04-24)

General: Generated files under `examples/output/` are no longer tracked ····· 2
`Canvas`: Default indifference-curve line width reduced from `2.0` to `1.8` ····· 11
`themes.default`: Colour-blind-friendly default palette, with `COLORBLIND_CYCLE_RGB` and `COLORBLIND_CYCLE_HEX` ····· 29
`Canvas.save`: TikZ axis regression tests no longer depend on colour indices ····· 32
`Animator`: Animation examples moved to `examples/scripts/animation.py` ····· 36
v1.5.0 (2026-04-21)
General: New example scripts for decomposition, demand diagrams, PCC/ICC paths, multi-panel layouts and TikZ export ····· 2
Regression tests for TikZ export and price-effect decomposition ····· 2
`econ-viz`: Notebook install flow on Colab is restart-safe ····· 4
`Figure`: Better placement and visibility of axis labels in shared-panel layouts ····· 14
`decompose_price_effect`: Price-effect decomposition: `decompose_price_effect()`, `PriceEffectDecomposition` and `Canvas.add_decomposition()`, drawing bundles A/B/C with substitution and income effects ····· 17
Decomposition API importable from the package root ····· 17
`Canvas.save`: Pure TikZ export backend; `Canvas.save()` and `Figure.save()` accept `.tex` ····· 32
v1.4.0 (2026-04-09)
General: Tests for `Animator` and `WidgetViewer` · 2
`econ-viz[extras]`: Optional extras `animation`, `interactive` and `all` ····· 3
`Animator.save`: Frames are composited onto an opaque background before export, for more reliable GIFs ····· 32
`Animator`: `Animator` produces GIF parameter sweeps with Pillow, without `ffmpeg` ····· 36
GIF examples for utility-parameter, price, income and budget-only sweeps ····· 36
`WidgetViewer`: `WidgetViewer` provides slider controls in Jupyter notebooks ····· 38
Numeric input boxes for notebook widgets ····· 38
v1.3.2 (2026-04-03)
General: Publish workflow skips versions already on PyPI ····· 2
`EdgeworthBox`: Contract curve and price line default to black dashed lines ····· 18
v1.3.1 (2026-04-03)
General: Package-root exports load lazily, with the public API unchanged ····· 2
`Canvas`: Canvas rendering split into `canvas.renderers` and `canvas.primitives` ····· 11
`EdgeworthBox`: Edgeworth internals split into compute, state and plotter modules ····· 18
`models.registry`: Model registry (`models.registry`) used by the command-line tool to build models ····· 19
`CobbDouglas`: Contour-level rules gathered in `contours.level_policies` ····· 19
`Canvas.save`: `Canvas`, `Figure` and `EdgeworthBox` share one figure exporter (`io.exporter`) ····· 32
`econ-viz`: CLI errors raise `CliConfigError`, with exit handling centralised in `cli.main` ····· 32
v1.3.0 (2026-04-03)
General: `examples/edgeworth_box.py` covering common utility-function combinations ····· 2
Dedicated Edgeworth tests ····· 2
`EdgeworthBox`: `EdgeworthBox` and `Equilibrium FocusConfig` for two-consumer exchange diagrams ····· 18
Contract-curve, core and Walrasian-equilibrium overlays, with equilibrium-focused indifference curves ····· 18
v1.2.3 (2026-03-31)
`slutsky_matrix`: `SlutskyMatrix` checks symmetry, negative semidefiniteness and homogeneity, and warns when a condition fails · 27
`econ-viz models`: CLI supports `QuasiLinear`, `StoneGeary` and `Translog` ····· 32
v1.2.2 (2026-03-31)
General: Dedicated PCC/ICC example generator and tests ····· 2
`PricePath`: PCC/ICC paths are smoothed by default, with slightly extended ends and no markers unless requested ····· 15
PCC/ICC paths use their own colours; more padding in the goods space of demand diagrams ····· 15
v1.2.0 (2026-03-30)
`Layout`: Multi-panel `Figure` layouts and the `Layout` enum ····· 14
`PricePath`: `PricePath`, `IncomePath` and `Canvas.add_path()` for PCC/ICC plots ····· 15
`DemandDiagram`: `DemandDiagram`, linking goods space and Marshallian demand ····· 15
v1.1.0 (2026-03-30)
`Translog`: `Translog` model, with legends and indifference-curve labels ····· 19
`comparative_statics`: `comparative_statics` helper ····· 26

`HomogeneityAnalyzer`: `HomogeneityAnalyzer` and `ReturnsToScale` in the analysis module ····· 28
v1.0.2 (2026-03-29)
`econ-viz`: NumPy upper bound removed to avoid conflicts on Colab ····· 2
`Canvas`: Math axis labels are no longer double-wrapped ····· 11
v1.0.1 (2026-03-29)
`econ-viz`: Pin `numpy<2` to avoid ABI mismatches on Colab and local installs ····· 2
Relaxed Python and NumPy bounds; `pytest` pinned to 8.x ····· 2
v1.0.0 (2026-03-28)
General: Initial release ····· 41

Command Index

A
add_budget ... 8
add_equilibrium ... 9
add_point ... 10
add_ray ... 10
add_utility ... 7
Animator ... 36
ArrowStyle ... 12
Axis ... 12
C
Canvas ... 5, 6
Canvas.add_* ... 6
Canvas.add_decomposition ... 17
Canvas.add_path ... 15
Canvas.save ... 6, 32
Canvas.show ... 6
CES ... 20
CobbDouglas ... 19
comparative_statics ... 26
Config.load ... 31
Config.reset ... 31
Config.use ... 31
CustomUtility ... 25
D
decompose_price_effect ... 17
DemandDiagram ... 15
E
econ-viz help ... 33
econ-viz init ... 36
econ-viz models ... 33
econ-viz plot ... 33
econ-viz solve-tex ... 36
econ_viz.models ... 5
EdgeworthBox ... 18
Effect ... 17
F
Figure ... 14
Fill ... 13
H
Haagsma ... 23
HomogeneityAnalyzer ... 28
I
IncomePath ... 15
L
Label ... 13
Layout ... 14
Legend ... 17
Leontief ... 21
levels.around ... 5
LineStyle ... 11
M
Marker ... 13
maximum ... 22
MultiGoodCD ... 26
MultiGoodCD.freeze ... 26
P
parse_latex ... 39
PerfectSubstitutes ... 21
PricePath ... 15
Q
QuasiLinear ... 23
S
Satiation ... 24
save ... 11
show ... 11
slutsky_matrix ... 27
solve ... 5
StoneGeary ... 24
Stroke ... 12
T
Theme ... 30
themes ... 29
Translog ... 20
W
WidgetViewer ... 38

References

Arrow, K. J., Chenery, H. B., Minhas, B. S., & Solow, R. M. (1961). Capital-labor substitution and economic efficiency. *Review of Economics and Statistics*, *43*(3), 225–250.

Christensen, L. R., Jorgenson, D. W., & Lau, L. J. (1975). Transcendental logarithmic utility functions. *American Economic Review*, *65*(3), 367–383.

Cobb, C. W., & Douglas, P. H. (1928). A theory of production. *American Economic Review*, *18*(1), 139–165.

Edgeworth, F. Y. (1881). *Mathematical psychics*. C. Kegan Paul.

Geary, R. C. (1950). A note on "A constant-utility index of the cost of living". *Review of Economic Studies*, *18*(1), 65–66.

Haagsma, R. (2012). A convenient utility function with Giffen behaviour. *ISRN Economics*, *2012*, Article 608645. https://doi.org/10.5402/2012/608645

Hicks, J. R. (1939). *Value and capital*. Clarendon Press.

Kugler, P., & Andrews, K. (1996). Graphical analysis and the visually impaired in undergraduate economics courses. *The Journal of Economic Education*, *27*(3), 224–228. https://doi.org/10.1080/00220485.1996.10844911

Slutsky, E. (1915). Sulla teoria del bilancio del consumatore. *Giornale degli Economisti e Rivista di Statistica*, *51*, 1–26.

Stone, R. (1954). Linear expenditure systems and demand analysis: An application to the pattern of British demand. *Economic Journal*, *64*(255), 511–527.

thriveth. (2014, January 22). *A color blind/friendly color cycle for Matplotlib line plots* [Source code]. https://gist.github.com/thriveth/8560036