PyNomo Documentation

Release 0.3.0

Ron Doerfler, Joe Marasco, Leif Roschier

February 01, 2016

CONTENTS

pyNomo is a python library for making nomographs (or nomograms) that are graphical calculators. Nomographs are defined as a python script that consists in most part of dictionaries.

CHAPTER

ONE

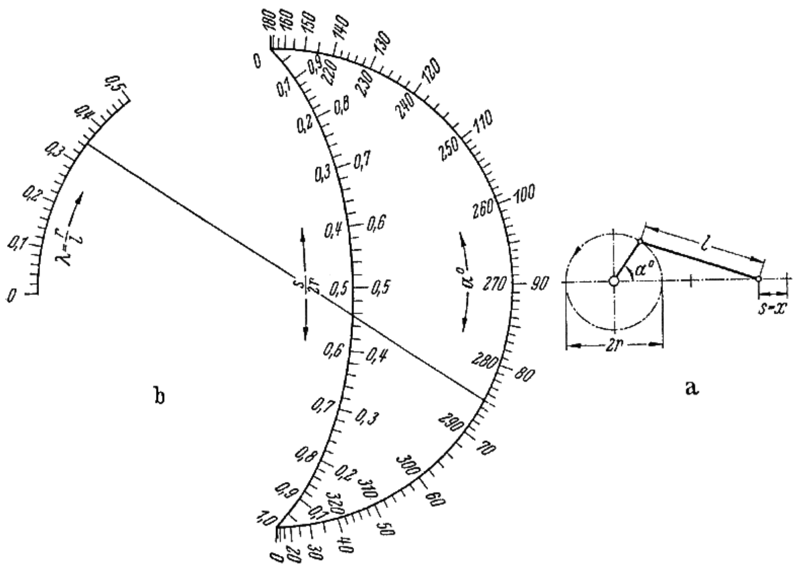
INTRODUCTION

1.1 What is a Nomogram and Why Would It Interest Me?

A *nomogram* or *nomograph* is a diagram that provides an easy, graphical way of calculating the result of a mathematical formula. Sometimes also called an *alignment chart*, a nomogram consists of a set of numbered scales, usually one for each variable in the formula, arranged so that a straightedge can be placed across known values to find the unknown value that solves the formula. Since an equation in two variables is usually represented by a graph, most nomograms represent formulas that involve three or more variables.

These graphical calculators were invented in 1880 by Philbert Maurice d'Ocagne and used extensively for many years to provide engineers with fast graphical calculations of complicated formulas to a practical precision. Electronic calculators and computers have made nomograms much less common today, but when a fast, handy calculator of a particular formula is needed they can be very useful. The cost to produce one is a sheet of paper, and they are fun to design, easy to use, and can be beautiful designs that engage people.

For example, here's a nomogram from 1920 that relates the variables l, s, r and α for a slider-crank mechanism:



The equation that this solves is quite complicated:

$s = r(1 - \cos\alpha) + l(1 - (1 - \lambda^2 \sin^2\alpha)^{1/2})$ where $\lambda = r/l$

There is a sample isopleth line on the nomogram that solves the equation for one set of values, scaled by r. For a value $\lambda = r/l = 0.35$ and an angle $\alpha = 75°$, we find that $s/2r \approx 0.455$, where we read off the same sides of the $s/2r$ and α scales. Note that in practice this nomogram would be drawn by a draftsman to a much larger scale for greater precision.

Try it out yourself! Pick a radius r, a length $l \geq 2r$ and an angle α, and find s on your calculator. Imagine an engineer solving this by hand for various parameters before calculators were invented. Then solve it on the nomogram here with a straightedge and compare your answers. When you're finished, choose values of r, l and s and solve for α. You'll realize that a nomogram can solve even for implicit variables that cannot be isolated on one side of the equation!

How in the world was this nomogram designed? Somehow this layout of scales solves the equation for every combination of its values using just a straightedge. For the nine most common functional relationships, PyNomo generates vector-image nomograms in PDF form using simple but customizable scripts in which you provide the functions of the variables. Beyond this, experienced designers can use a tenth PyNomo option to draw nomograms with arbitrarily complicated layouts such as this one, and even linear and circular slide rules.

Designing nomograms is an enjoyable pursuit, much more so than in the past since PyNomo can provide the expert knowledge and also serve as the technical draftsman. And as described below, nomograms are very useful for a variety of applications even today.

1.2 Uses of Nomograms

Nomograms have been used in an extensive array of applications. A sample includes

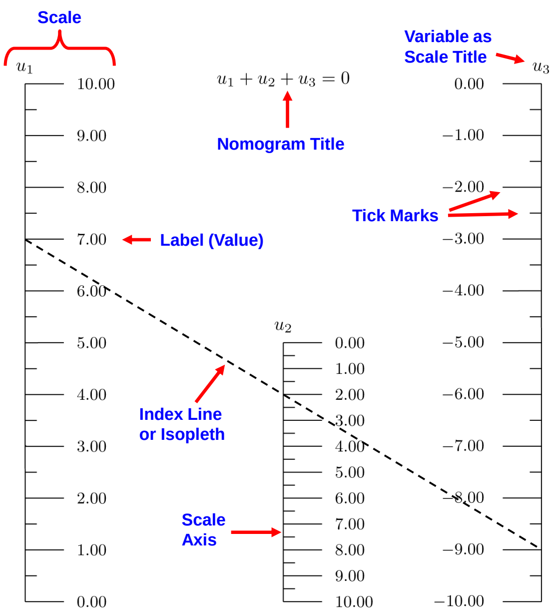
- The original application by d'Ocagne, the automation of complicated "cut and fill" calculations for earth removal during the construction of the French national railway system. This was an important proof of concept, because the calculations are non-trivial and the results translated into significant savings of time, effort, and money.
- The design of channels, pipes and weirs for regulating the flow of water.
- The work of Lawrence Henderson, in which nomograms were used to correlate many different aspects of blood physiology. It was the first major use of nomograms in the United States and also the first medical nomograms anywhere. Nomograms continue to be used extensively in medical fields.
- Ballistics calculations prior to fire control systems, where calculating time was critical.
- Machine shop calculations, to convert blueprint dimensions and perform calculations based on material dimensions and - properties. These nomograms often included markings for standard dimensions and for available manufactured parts.
- Statistics, for complicated calculations of properties of distributions and for operations research including the design of acceptance tests for quality control.
- Operations Research, to obtain results in a variety of optimization problems.
- Chemistry and chemical engineering, to encapsulate both general physical relationships and empirical data for specific compounds.
- Aeronautics, in which nomograms were used for decades in the cockpits of aircraft of all descriptions. As a navigation and flight control aid, nomograms were fast, compact and easy-to-use calculators.

- Astronomical calculations, as in the post-launch orbital calculations of Sputnik 1 by P.E. Elyasberg.[1]
- Engineering work of all kinds: Electrical design of filters and transmission lines, mechanical calculations of stress and loading, optical calculations, and so forth.
- Military, where complex calculations need to be made in the field quickly and with reliability not dependent on electrical devices.

Nomograms serve a dual purpose: they allow nitty-gritty fast computation—answers in the form of unambiguous numbers—and at the same time provide tremendous insight through the relationship of the various scales, their labeling, limits, and gradations. The better nomograms are self-documenting. They provide a visual model of a system and manifest a wonderful ability to imply interrelationships and cross-variable sensitivities. As the mathematician and computer scientist Richard Hamming remarked, “The purpose of computing is insight, not numbers.”

1.3 Parts of Nomograms

There are few parts to a nomogram, but it is important to know them as they will be referenced throughout the documentation. We will introduce the terms with most common type of nomogram consisting of three parallel straight scales. This form is used to solve an equation in which functions of three variables sum to zero. The simplest such formula is $u_1 + u_2 + u_3 = 0$ for the three variables u_1, u_2 and u_3. An example of this type of nomogram is shown below, annotated with terms used to describe the parts of a nomogram. In general the scales can be functions of u_1, u_2 and u_3, but here the scales are simply the variables. The nomogram solves the equation for any variable given values of the other two variables, with the sample isopleth here representing the solution for values 7, 2 and -9.



Nomograms should be self-contained, that is, anyone can understand what the nomogram solves and how to use it with only passing knowledge of what they are. This means that there should be a sample isopleth to guide the user. If the application is not obvious, it should be listed in the title, and perhaps a figure relating the variables to the application may be called for. The equation being solved should also be listed on the nomogram; we can assure you there are few more tedious tasks than reverse-engineering a decades-old nomogram back to its defining equation.

1.4 What Can PyNomo Do For Me?

PyNomo allows us to design nearly all nomograms, even grid and compound nomograms for equations of more than three variables, with very little mathematics background. A knowledge of algebra is necessary in order to first arrange the equation into one of the ten standard types of equation that PyNomo supports (nine specific types and one general type).

Then a PyNomo script is written for the nomogram type that fits the relationships among the variable functions. Perhaps two functions are multiplied and one divided in your equation, or perhaps the relationships are more complicated. Typically this involves looking through the table of formats for the types of equations PyNomo supports and choosing one that matches your equation. Then a sample script from a standard example of that type is copied and edited to use the functions in your equation. Copying and modifying a standard example as a starting point is easy and fast–we all do that.

The script is run and a PDF file is automatically created with the nomogram laid out for printing. Once you start making nomograms you may want to customize how they look—the spacing of tick marks on the scales, the scale titles, the location of the nomogram title, and so forth. You may want to draw a sample isopleth and add color to the scales and their labels. PyNomo offers many such features, and this documentation tries to cover them all, but don't be put off by these extra details sprinkled throughout the examples here. They may make the scripts appear more complicated, but they are totally optional and can be ignored until the day you decide you really would like that one scale to be red. That's the point where you look in the documentation for scale parameters that involve color.

Explore the tutorials and you will find yourself amazed that you are creating nomograms that really do work. There are also sections of this documentation that deal with more advanced topics such as designing nomograms for very complicated equations using determinant equations, applying transformations and projections to twist and stretch nomograms to square them up for more precise use, and even using PyNomo to create linear and circular slide rules.

CHAPTER

TWO

TUTORIAL

2.1 Tutorial 1: Hello World

Todo

Here simplest tutorial...

2.2 Tutorial 2: More complex example...

Todo

Here more complex tutorial... (15 minutes?)

CHAPTER THREE

INSTALLATION

pyNomo is a python library and thus requires working python installation on the computer. pyNomo stands on the shoulders of (read: requires) the python packages: numpy, scipy and pyx that requires LaTeX-installation. From version > 0.3.0 pynomo is compatible both with python 2 and python 3.

For editing pyNomo scripts any text browser works but integrated development environment (IDE) for python can speed up developments. Good free IDE alternatives are for example PyCharm community edition and spyder.

3.1 Python 2.7.x OSX Installation

In OSX Macports is an effective tool to manage open-source software. In the following a MacPorts environment is set for Python and pyNomo. *sudo* runs the commands as super-user and requires it's password to be given.

First install python 2.7

```
$ sudo port install python27
```

One can list available python versions on the system with command

```
$ sudo port select --list python
```

Select MacPorts python 2.7

```
$ sudo port select --set python python27
```

Install python package index tool (pip)

```
$ sudo port install py27-pip
```

and set it active

```
$ port select --set pip pip27
```

Now python environment should be correct to be run from `/opt/local/Library/...`. Now install other required packages.

```
$ sudo port install py27-numpy
$ sudo port install py27-scipy
$ sudo port install py27-pyx
$ sudo pip install pynomo
```

3.2 Python 3.5 OSX Installation

In OSX Macports is an effective tool to manage open-source software. In the following a MacPorts environment is set for Python and pyNomo. *sudo* runs the commands as super-user and requires it's password to be given.

First install python 3.5

```
$ sudo port install python35
```

One can list available python versions on the system with command

```
$ sudo port select --list python
```

Select MacPorts python 2.7

```
$ sudo port select --set python3 python35
```

Install python package index tool (pip)

```
$ sudo port install py35-pip
```

and set it active (this sets it system-wide, if you are using also python2, consider twice)

```
$ port select --set pip pip35
```

Now python environment should be correct to be run from `/opt/local/Library/...`. Now install other required packages.

```
$ sudo port install py35-numpy
$ sudo port install py35-scipy
$ sudo port install py35-scipy
$ sudo port install texlive
$ sudo port install texlive-fonts-recommended
```

If you set pip active in whole system, run:

```
$ sudo pip install pynomo
$ sudo pip install --allow-external pyx pyx
```

If not, check where pip is located and run for example (check your pip path)

```
$ sudo /opt/local/bin/pip install pynomo
$ sudo /opt/local/bin/pip install --allow-external pyx pyx
```

3.3 Python 2.7.x Linux installation

In Debian Linux distribution and in its derivatives (for example Ubuntu and Raspbian) pynomo can be installed using *apt-get* with the following commands. *sudo* runs the commands as super-user and requires it's password to be given.

```
$ sudo apt-get -y install python
$ sudo apt-get -y install python-pyx
$ sudo apt-get -y install python-pip
$ sudo apt-get -y install python-numpy
$ sudo apt-get -y install python-scipy
$ sudo pip install pynomo
```

3.4 Python 3 Linux installation

In Debian Linux distribution and in its derivatives (for example Ubuntu and Raspbian) pynomo can be installed using *apt-get* with the following commands. *sudo* runs the commands as super-user and requires it's password to be given.

```
$ sudo apt-get -y install python3
$ sudo apt-get -y install python3-pip
$ sudo apt-get -y install python3-numpy
$ sudo apt-get -y install python3-scipy
$ sudo apt-get -y install texlive-latex-base
$ sudo apt-get -y install texlive-fonts-recommended
$ sudo pip install pynomo
$ pip3 install --allow-external pyx pyx
$ pip3 install pynomo
```

3.5 Python 2.7.x Windows installation

1. Download and install python 2.7.x from www.python.org/downloads/ .
2. Download and install MIKTeX LaTeX -distribution from http://miktex.org/download.
3. Download and install numpy from sourceforge.net/projects/numpy.
4. Download and install scipy from sourceforge.net/projects/scipy.

pyx (python graphics package) installation is more tricky. Either

- Download pyx 0.12.1 (python graphics package) from http://sourceforge.net/projects/pyx/files/pyx/0.12.1/PyX-0.12.1.tar.gz/download
- Uncompress the file *PyX-0.12.1.tar.gz* using for example 7-zip.
- Open command prompt (cmd) and go to the uncompressed folder that contains file *setup.py*.
- run command `python setup.py install`

or cross your fingers and just run:

```
> pip install --allow-external pyx pyx
```

on command prompt with administrative rights.

Finally pyNomo is installed either by downloading installer from http://sourceforge.net/projects/pynomo/ and by running it. Other choice to try is to run:

```
> pip install pynomo
```

on command line. Tedious, huh! If you find simpler Windows recipe, please email it to the maintainer of the project.

3.6 Python 3.5 Windows installation

1. Download and install python 3.5.x from www.python.org/downloads/ .
2. Download and install MIKTeX LaTeX -distribution from http://miktex.org/download.

3. Download and install numpy from sourceforge.net/projects/numpy.
4. Download and install scipy from sourceforge.net/projects/scipy.

pyx (python graphics package) installation is more tricky. Either

- Download pyx 0.14 (python graphics package) from https://downloads.sourceforge.net/project/pyx/pyx/0.14/PyX-0.14.tar.gz
- Uncompress the file *PyX-0.14.tar.gz* using for example 7-zip.
- Open command prompt (cmd) and go to the uncompressed folder that contains file *setup.py*.
- run command `python setup.py install`

or cross your fingers and just run:

```
> pip install --allow-external pyx pyx
```

on command prompt with administrative rights.

Finally pyNomo is installed either by downloading installer from http://sourceforge.net/projects/pynomo/ and by running it. Other choice to try is to run:

```
> pip install pynomo
```

on command line. Tedious, huh! If you find simpler Windows recipe, please email it to the maintainer of the project.

3.7 Python 2.7.x Docker installation

Docker is a platform to create a sandboxed virtualized environments. In the following example *Dockerfile* a virtualized Ubuntu is created that has pyNomo installed with all requirements:

```
FROM ubuntu

# Install required packages:
# python, pyx, pip, numpy, scipy, pynomo and their requirements
RUN apt-get update
RUN apt-get -y upgrade
RUN DEBIAN_FRONTEND=noninteractive apt-get -y install python
RUN DEBIAN_FRONTEND=noninteractive apt-get -y install python-pyx
RUN DEBIAN_FRONTEND=noninteractive apt-get -y install python-pip
RUN DEBIAN_FRONTEND=noninteractive apt-get -y install python-numpy
RUN DEBIAN_FRONTEND=noninteractive apt-get -y install python-scipy
RUN DEBIAN_FRONTEND=noninteractive pip install pynomo

# Add /app directory and make it working dir
RUN mkdir -p /app
ADD . /app
WORKDIR /app

# Set the default command to execute -> "python my_pynomo_file.py"
CMD ["python", "my_pynomo_file.py"]
```

Docker container (environment) *my_pynomo_docker* is built in the directory */my_directory_path* that has the file *Dockerfile* with command

```
$ docker build -t my_pynomo_docker .
```

Once environment is built and *my_pynomo_file.py* is in directory *'/my_directory_path/pdf_py_dir/* one can run

```
$ docker run -i -v /my_directory_path/pdf_py_dir:/app my_pynomo_docker
```

that runs command `python  my_pynomo_file.py` inside */app directory* of container that is mapped to directory */my_directory_path/pdf_py_dir* of the host system. That way a folder is used to share the script file and the generated pdf file between host system and the container (virtualized Linux environment).

3.8 Python 3 Docker installation

Docker is a platform to create a sandboxed virtualized environments. In the following example *Dockerfile* a virtualized Ubuntu is created that has pyNomo installed with all requirements:

```
FROM ubuntu

# Install required packages
RUN apt-get update && apt-get -y install -y \
    python3 \
    python3-pip \
    python3-numpy \
    python3-scipy \
    texlive-latex-base \
    texlive-fonts-recommended
RUN DEBIAN_FRONTEND=noninteractive pip3 install --allow-external pyx pyx

# Add our python app code to the image
RUN mkdir -p /app
ADD . /app
WORKDIR /app

CMD ["bash"]
```

Docker container (environment) *my_pynomo_docker* is built in the directory */my_directory_path* that has the file *Dockerfile* with command

```
$ docker build -t my_pynomo_docker .
```

Once environment is built and *my_pynomo_file.py* is in directory *'/my_directory_path/pdf_py_dir/* one can run

```
$ docker run -it -v /my_directory_path/pdf_py_dir:/app my_pynomo_docker
```

that opens terminal in */app directory* of container that is mapped to directory */my_directory_path/pdf_py_dir* of the host system. There one can run own scripts like:

```
$ python3 my_script.py
```

That way a folder is used to share the script file and the generated pdf file between host system and the container (virtualized Linux environment).

CHAPTER FOUR

BIG PICTURE OF NOMOGRAPH CONSTRUCTION

Nomographs of PyNomo are constructed by writing a python script that defines the nomograph parameters and initializes class `Nomographer(parameters)` to build the nomograph.

Nomograph is constructed by defining axes that are used to build blocks. If there are more than one block, they are aligned with each other in order to construct the nomograph.

A simple example of pseudocode of typical PyNomo structure is the following:

```
from pynomo.nomographer import *  # this loads the needed pynomo class
# define block 1
axis_params_1_for_block_1 = {...}
axis_params_2_for_block_1 = {...}
axis_params_3_for_block_1 = {...}
block_1 = {...}

# define block 2
axis_params_1_for_block_2 = {...}
axis_params_2_for_block_2 = {...}
axis_params_3_for_block_2 = {...}
block_2 = {...}

# define nomograph
main_params = { 'filename': 'filename_of_nomograph.pdf',  # filename of output
                'block_params': [block_1,block_2],        # the blocks make the nomograph
                'transformations':[('scale paper',)],     # these make (projective) transformations for the canves
                }
# create nomograph
Nomographer(main_params)
```

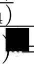
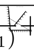
It is to be noted that nomograph is defined as python dictionaries that constitute one main dictionary that is passed to Nomographer class.

4.1 Axes

A nomograph consist in simple terms of axes (or scales) that are positioned in a way to fulfil the equation to be graphed. Axes (or grids or graphs) are the leafs that build the tree of a nomograph. Defining axes and their appearance is major work in nomograph construction. Different possibilities are illustrated in Axes chapter.

4.2 Blocks

Blocks relate axes to each other. Each block fulfils some equation where axes are the variables. The following blocks below with corresponding equations are the core of PyNomo. These are used as easy building blocks for nomograph construction. If these do not suffice one can build as complex nomograph as one wishes by using determinants in type 9.

Type 1	$F_1(u_1)+F_2(u_2)+F_3(u_3)=0$	
Type 2	$F_1(u_1)=F_2(u_2)F_3(u_3)$	
Type 3	$F_1(u_1)+F_2(u_2)+\cdots+F_N(u_N)=0$	
Type 4	$\frac{F_1(u_1)}{F_2(u_2)}=\frac{F_3(u_3)}{F_4(u_4)}$	
Type 5	$F_1(v)=F_2(x,u)$.	
Type 6	$u=u$	
Type 7	$\frac{1}{F_1(u_1)}+\frac{1}{F_2(u_2)}=\frac{1}{F_3(u_3)}$	
Type 8	$y=F(u)$	
Type 9	$\begin{vmatrix} F_1(u_1[,v_1]) & G_1(u_1[,v_1]) & H_1(u_1[,v_1]) \\ F_2(u_2[,v_2]) & G_2(u_2[,v_2]) & H_2(u_2[,v_2]) \\ F_3(u_3[,v_3]) & G_3(u_3[,v_3]) & H_3(u_3[,v_3]) \end{vmatrix} = 0$	
Type 10	$F_1(u)+F_2(v)F_3(w)+F_4(w)=0$	

4.3 Combination of blocks

If a nomograph consists of many equations that are aligned, a compound nomograph is constructed. Chapter compound nomograph discusses block aligment in detail.

4.4 Transformations

Scales shall be transformed in order to use given space (paper) optimally. Chapter Transformations discusses transformations.

CHAPTER

FIVE

AXES

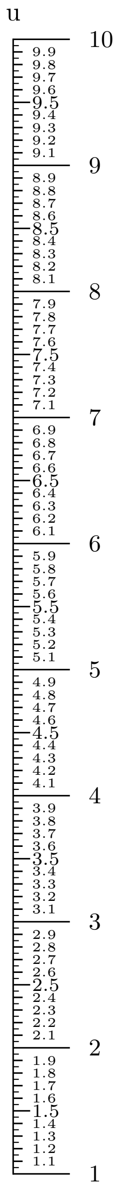
5.1 Axes by example

Axes are fundamental building blocks of nomographs. The following code uses minimal axis definion `N_params` that is rendered as a linear scale illustrated below. The range of values axis represents is defined with keywords `u_min` and `u_max`. `title` sets title string for the axis. Key part of the nomograph is the functional form of the axis. In the example below it is defined with keyword `function` and is given as a function. Different types of blocks assume different keywords of axis functions. For example types 1, 2 and 3 take keyword `function` but type 9 takes either `f`, `g`, `h` or `f_grid`, `g_grid`, `h_grid` keywords. So one have to define axis parameters compatible with the used block type. In the examples below Type 8 is used as block to taking axis definition because it is the simplest one.

5.1.1 Linear scale (`'scale_type':'linear'`)

Here we start with the simplest axis. It has by default scale `'scale_type':'linear'` that is simple linear scale.

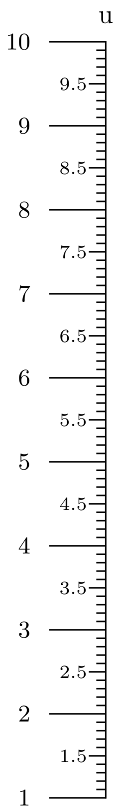
```
# ex_axes_1.py

import sys
sys.path.insert(0, "..")
from pynomo.nomographer import *

# axis definitions
N_params = {'u_min': 1.0,            # axis start value
            'u_max': 10.0,           # axis stop value
            'function': lambda u: u, # axis function
            'title': 'u',            # axis titles
            }

# block definitons defining one block of type 8
block_params = {'block_type': 'type_8',
                'f_params': N_params,
                'width': 5.0,
                'height': 15.0,
                }

# nomograph generation definitions
main_params = {'filename': 'ex_axes_1.pdf',
               'paper_height': 15.0,
               'paper_width': 5.0,
               'block_params': [block_params],
               'transformations': [('scale paper',)]
              }

# actual code that builds the nomograph
Nomographer(main_params)
```



Because the example above looked little too busy or packed, we reduce the ticks by using only three different tick levels `'tick_levels':3` and two tick text levels `'tick_text_levels':2`. Tick side relative to the final drawing is set to left using `'tick_side':'left'`.

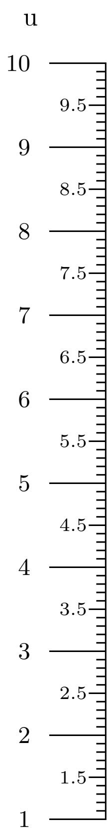
```
# ex_axes_2.py

N_params = {'u_min': 1.0,
            'u_max': 10.0,
            'function': lambda u: u,
            'title': 'u',
            'tick_levels': 3,        # <-
            'tick_text_levels': 2,   # <-
            'tick_side': 'left',     # <-
            }

block_params = {'block_type': 'type_8',
                'f_params': N_params,
                'width': 5.0,
                'height': 10.0,
                }

main_params = {'filename': 'ex_axes_2.pdf',
               'paper_height': 10.0,
               'paper_width': 5.0,
```

```
                'block_params': [block_params],
                'transformations': [('scale paper',)]
               }

Nomographer(main_params)
```



Title position can be shifted in both x- and y-directions. In the following we shift it using key-values `'title_x_shift':-1.0` and `'title_y_shift':0.5`. Units are here centimeters.

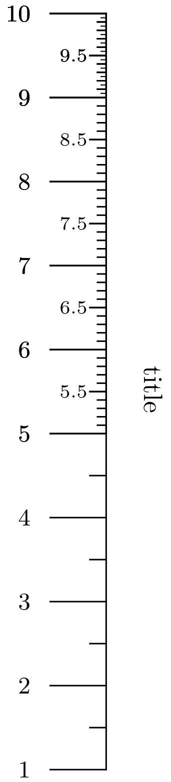
```
# ex_axes_3.py

N_params = {'u_min': 1.0,
            'u_max': 10.0,
            'function': lambda u: u,
            'title': 'u',
            'tick_levels': 3,
            'tick_text_levels': 2,
            'tick_side': 'left',
            'title_x_shift': -1.0,    # <-
            'title_y_shift': 0.5      # <-
            }

block_params = {'block_type': 'type_8',
                'f_params': N_params,
                'width': 5.0,
                'height': 10.0,
                }

main_params = {'filename': 'ex_axes_3.pdf',
               'paper_height': 10.0,
               'paper_width': 5.0,
               'block_params': [block_params],
               'transformations': [('scale paper',)]
               }

Nomographer(main_params)
```



Sometimes single level of axis definitions is not enough. We might want to add more ticks in some additional range of the axis. Keyword `'extra_params'` helps here. Value for this key is an array of dictionaries that modify given params in the given range set by `u_min` and `u_max`. In the following example we define additional ranges with more ticks in ranges 5.0..10.0 and 9.0..10.0. We also draw title this time to center using `'title_draw_center:True`.

```
# ex_axes_4.py

N_params = {'u_min': 1.0,
            'u_max': 10.0,
            'function': lambda u: u,
            'title': 'title',
            'tick_levels': 2,
            'tick_text_levels': 1,
            'tick_side': 'left',
            'title_draw_center': True,              # <-
            'extra_params': [{'u_min': 5.0,         # <- range 1
                              'u_max': 10.0,        # <-
                              'tick_levels': 3,     # <-
                              'tick_text_levels': 2, # <-
                              },                    # <-
                             {'u_min': 9.0,         # <- range 2
                              'u_max': 10.0,        # <-
                              'tick_levels': 4,     # <-
                              'tick_text_levels': 2, # <-
                              }                     # <-
                             ]                      # <-
            }
block_params = {'block_type': 'type_8',
                'f_params': N_params,
                'width': 5.0,
                'height': 10.0,
                }
main_params = {'filename': 'ex_axes_4.pdf',
               'paper_height': 10.0,
               'paper_width': 5.0,
               'block_params': [block_params],
               'transformations': [('scale paper',)]
```

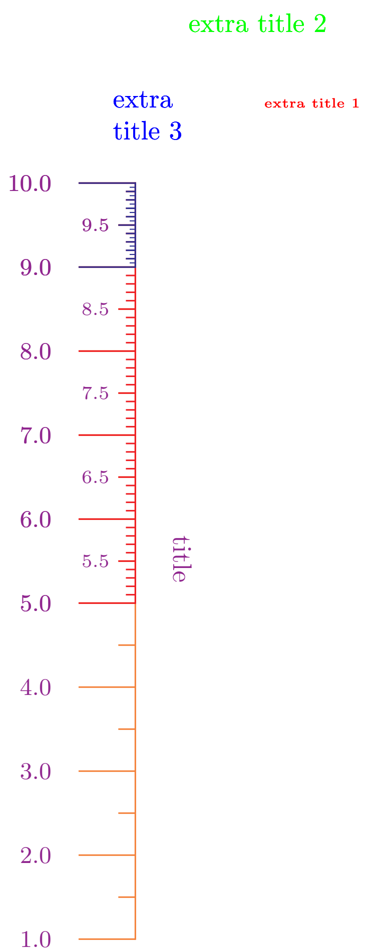
```
                }
Nomographer(main_params)
```



Color can be used to tune visual appearance of the axis. In the following example we tune colors with self-explaining keywords `'axis_color'`, `'text_color'` and `'title_color'`. Additional titles are set by using keyword `'extra_titles'` with value of an array of dictionaries that can take keywords `'dx'` and `'dy'` as relative position to main title. Value of keyword `'text'``sets the title text and ``'pyx_extra_defs'` can be used to give additional parameters for pyx rendering that is only option in current release. In the example numbers are formatted to have one three digits before comma and and one digit after comma using `'text_format':r"$%3.1f$ "`.

```
# ex_axes_4_1.py

N_params = {'u_min': 1.0,
            'u_max': 10.0,
            'function': lambda u: u,
            'title': 'title',
            'tick_levels': 2,
            'tick_text_levels': 1,
            'tick_side': 'left',
            'title_draw_center': True,
            'text_format': r"$%3.1f$ ",                     # <- format numbers as %3.1f
            'axis_color': color.cmyk.Orange,
            'text_color': color.cmyk.Plum,
            'title_color': color.cmyk.Plum,
            'extra_params': [{'u_min': 5.0,
                              'u_max': 10.0,
                              'tick_levels': 3,
                              'tick_text_levels': 2,
                              'axis_color': color.cmyk.Red,
                              },
                             {'u_min': 9.0,
                              'u_max': 10.0,
                              'tick_levels': 4,
                              'tick_text_levels': 2,
                              'axis_color': color.cmyk.Blue,
```

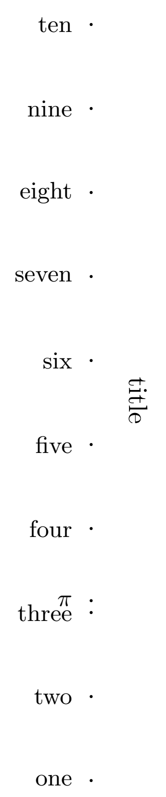
```
                              }
                            ],
            'extra_titles': [{'dx': 1.0,                                    # <- 1st extra title
                              'dy': 1.0,                                    # <-
                              'text': 'extra title 1',                      # <-
                              'width': 5,                                   # <-
                              'pyx_extra_defs': [color.rgb.red, text.size.tiny]   # <-
                              },
                             {'dx': 0.0,                                    # <- 2nd extra title
                              'dy': 2.0,                                    # <-
                              'text': 'extra title 2',                      # <-
                              'width': 5,                                   # <-
                              'pyx_extra_defs': [color.rgb.green]           # <-
                              },
                             {'dx': -1.0,                                   # <- 3rd extra title
                              'dy': 1.0,                                    # <-
                              'text': r"extra \par title 3",                # <- \par = newline
                              'width': 5,                                   # <-
                              'pyx_extra_defs': [color.rgb.blue]            # <-
                              }]
            }
block_params = {'block_type': 'type_8',
                'f_params': N_params,
                'width': 5.0,
                'height': 10.0,
                }
main_params = {'filename': 'ex_axes_4_1.pdf',
               'paper_height': 10.0,
               'paper_width': 5.0,
               'block_params': [block_params],
               'transformations': [('scale paper',)]
               }
Nomographer(main_params)
```



5.1.2 Manual point scale (`'scale_type':'manual point'`)

Sometimes axes have to be defined manually. One option is to use manual point scale type with `'scale_type':'manual point'` and define the points as a dict to keyword `'manual_axis_data'`.

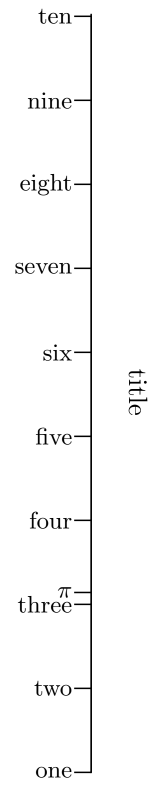
```
# ex_axes_5.py

N_params = {'u_min': 1.0,
            'u_max': 10.0,
            'function': lambda u: u,
            'title': 'title',
            'tick_levels': 2,
            'tick_text_levels': 1,
            'tick_side': 'left',
            'title_draw_center': True,
            'scale_type': 'manual point',        # <- use manual points
            'manual_axis_data': {1.0: 'one',     # <- give point values as keys
                                 2.0: 'two',     # <- and texts as values
                                 3.0: 'three',
                                 3.1415: r'$\pi$',
                                 4.0: 'four',
                                 5.0: 'five',
                                 6.0: 'six',
                                 7.0: 'seven',
                                 8.0: 'eight',
                                 9.0: 'nine',
                                 10.0: 'ten'}
            }
block_params = {'block_type': 'type_8',
                'f_params': N_params,
                'width': 5.0,
                'height': 10.0
                }
main_params = {'filename': 'ex_axes_5.pdf',
               'paper_height': 10.0,
               'paper_width': 5.0,
               'block_params': [block_params],
               'transformations': [('scale paper',)]
               }
Nomographer(main_params)
```



5.1.3 Manual line scale ('scale_type':'manual line')

Similarly other option is to use manual line scale type with 'scale_type':'manual line' that draws main scale line and ticks. Drawn ticks are defined as a dict to keyword 'manual_axis_data' as above example.

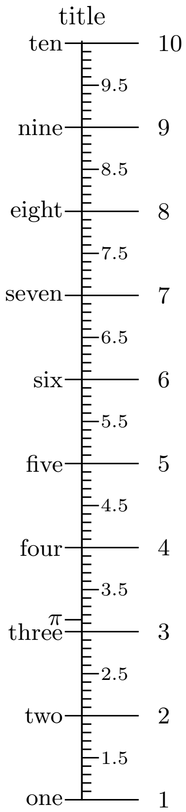
```
# ex_axes_6.py

N_params = {'u_min': 1.0,
            'u_max': 10.0,
            'function': lambda u: u,
            'title': 'title',
            'tick_levels': 2,
            'tick_text_levels': 1,
            'tick_side': 'left',
            'title_draw_center': True,
            'scale_type': 'manual line',     # <-
            'manual_axis_data': {1.0: 'one',
                                 2.0: 'two',
                                 3.0: 'three',
                                 3.1415: r'$\pi$',
                                 4.0: 'four',
                                 5.0: 'five',
                                 6.0: 'six',
                                 7.0: 'seven',
                                 8.0: 'eight',
                                 9.0: 'nine',
                                 10.0: 'ten'}
            }
block_params = {'block_type': 'type_8',
                'f_params': N_params,
                'width': 5.0,
                'height': 10.0,
                }
main_params = {'filename': 'ex_axes_6.pdf',
               'paper_height': 10.0,
               'paper_width': 5.0,
               'block_params': [block_params],
               'transformations': [('scale paper',)]
               }
Nomographer(main_params)
```



Combining manual lines and a linear scale.

```
# ex_axes_7.py

N_params = {'u_min': 1.0,
            'u_max': 10.0,
            'function': lambda u: u,
            'title': 'title',
            'tick_levels': 2,
            'tick_text_levels': 1,
            'tick_side': 'left',
            'scale_type': 'manual line',
            'manual_axis_data': {1.0: 'one',
                                 2.0: 'two',
                                 3.0: 'three',
                                 3.1415: r'$\pi$',
                                 4.0: 'four',
                                 5.0: 'five',
                                 6.0: 'six',
                                 7.0: 'seven',
                                 8.0: 'eight',
                                 9.0: 'nine',
                                 10.0: 'ten'},
            'extra_params': [{'u_min': 1.0,
                              'u_max': 10.0,
                              'scale_type': 'linear',
                              'tick_levels': 3,
                              'tick_text_levels': 2,
                              'tick_side': 'right',
                              }]
            }
block_params = {'block_type': 'type_8',
                'f_params': N_params,
                'width': 5.0,
                'height': 10.0,
                }
main_params = {'filename': 'ex_axes_7.pdf',
               'paper_height': 10.0,
               'paper_width': 5.0,
               'block_params': [block_params],
               'transformations': [('scale paper',)]
```

```
            }
Nomographer(main_params)
```

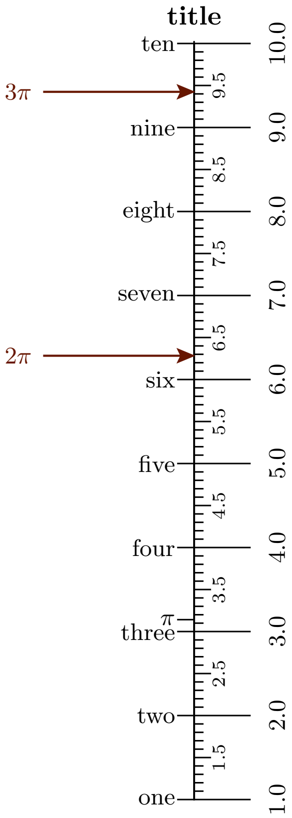


5.1.4 Manual arrows (`'scale_type':'manual arrow'`)

Manual arrows can be used to point values in the scale using arrows.

```
# ex_axes_7_1.py

N_params = {'u_min': 1.0,
            'u_max': 10.0,
            'function': lambda u: u,
            'title': r'\bf title',
            'tick_levels': 2,
            'tick_text_levels': 1,
            'tick_side': 'left',
            'scale_type': 'manual line',
            'manual_axis_data': {1.0: 'one',
                                 2.0: 'two',
                                 3.0: 'three',
                                 3.1415: r'$\pi$',
                                 4.0: 'four',
                                 5.0: 'five',
                                 6.0: 'six',
                                 7.0: 'seven',
                                 8.0: 'eight',
                                 9.0: 'nine',
                                 10.0: 'ten'},
            'extra_params': [{'u_min': 1.0,
                              'u_max': 10.0,
                              'scale_type': 'linear',
                              'tick_levels': 3,
                              'tick_text_levels': 2,
                              'tick_side': 'right',
                              'extra_angle': 90.0,
                              'text_horizontal_align_center': True,
                              'text_format': r"$%2.1f$"},
```

```
                              {'scale_type': 'manual arrow',          # <-
                               'manual_axis_data': {6.2830: r'$2\pi$',
                                                    9.4245: r'$3\pi$'},
                               'arrow_color': color.cmyk.Sepia,
                               'arrow_length': 2.0,
                               'text_color': color.cmyk.Sepia,
                               }]
             }
block_params = {'block_type': 'type_8',
                'f_params': N_params,
                'width': 5.0,
                'height': 10.0,
                }
main_params = {'filename': 'ex_axes_7_1.pdf',
               'paper_height': 10.0,
               'paper_width': 5.0,
               'block_params': [block_params],
               'transformations': [('scale paper',)]
               }
Nomographer(main_params)
```

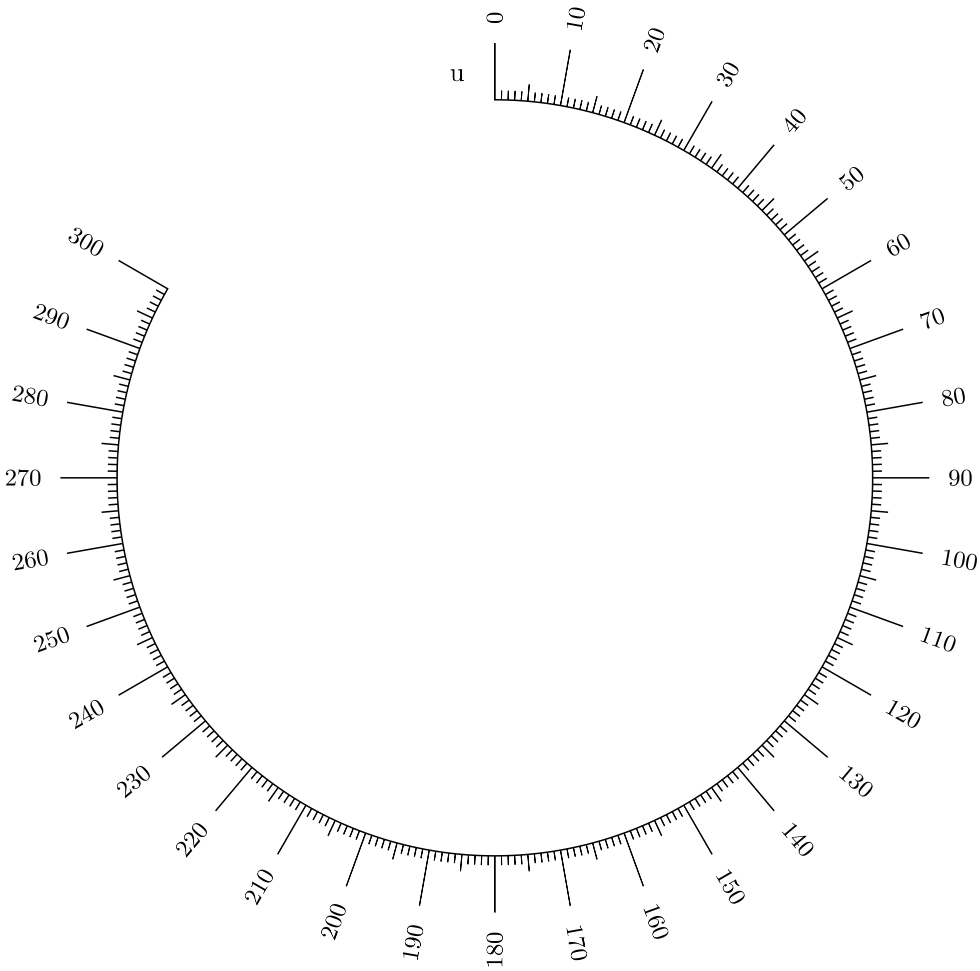


5.1.5 Manual function ('function_x' and 'function_y')

If one wants to explicitely draw scale in xy-scace, parameters 'function_x' and 'function_y' can be used in conjuction with block type 8. In the following example circular scale is drawn.

```
# ex_axes_8.py

N_params = {'u_min': 0.0,
            'u_max': 300.0,
            'function_x': lambda u: 3 * sin(u / 180.0 * pi),
            'function_y': lambda u: 3 * cos(u / 180.0 * pi),
            'title': 'u',
            'tick_levels': 3,
            'tick_text_levels': 1,
            'title_x_shift': -0.5,
```

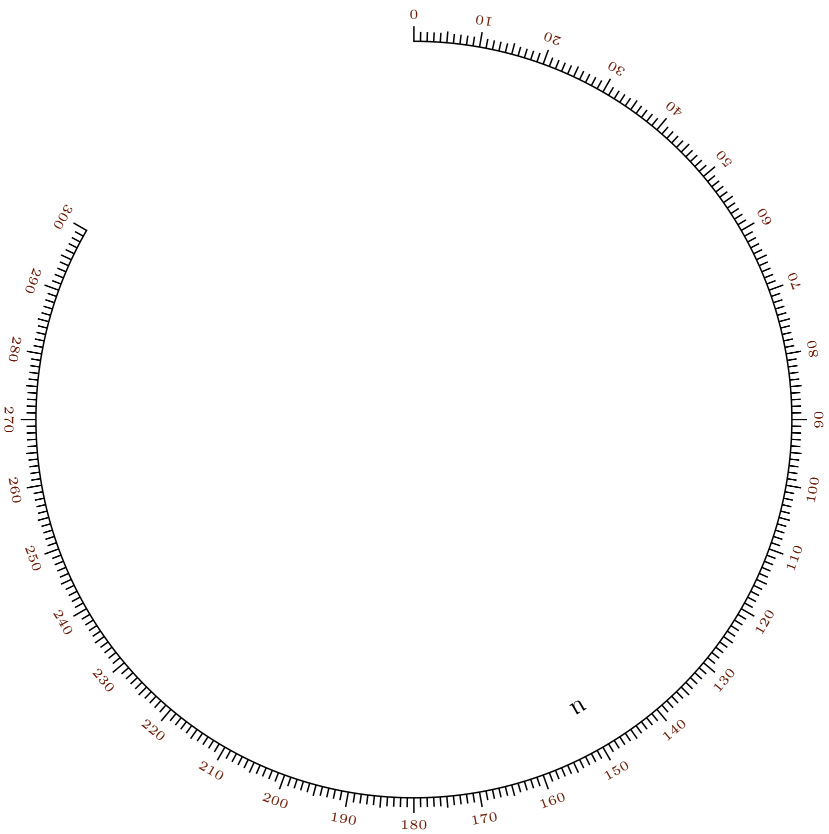
```
            }
block_params = {'block_type': 'type_8',
                'f_params': N_params,
                'width': 5.0,
                'height': 15.0,
                }
main_params = {'filename': 'ex_axes_8.pdf',
               'paper_height': 10.0,
               'paper_width': 10.0,
               'block_params': [block_params],
               'transformations': [('scale paper',)]
               }
Nomographer(main_params)
```



In the following we fine-tune the appearance of the scale. Tick lengths are explicitly given with params 'grid_length_x' (note name with bad logic), text sizes are tuned with params 'text_size_x' and distance of text to the scale is set using 'text_distance_x'. 'full_angle' parameter allows text to be drawn also upside down and text angle is rotated with 'extra_angle'.

```
# ex_axes_8_1.py

N_params = {'u_min': 0.0,
            'u_max': 300.0,
            'function_x': lambda u: 3 * sin(u / 180.0 * pi),
            'function_y': lambda u: 3 * cos(u / 180.0 * pi),
            'title': 'u',
            'tick_levels': 3,
            'tick_text_levels': 1,
            'title_x_shift': -0.5,
            'grid_length_0': 0.8/4,
```

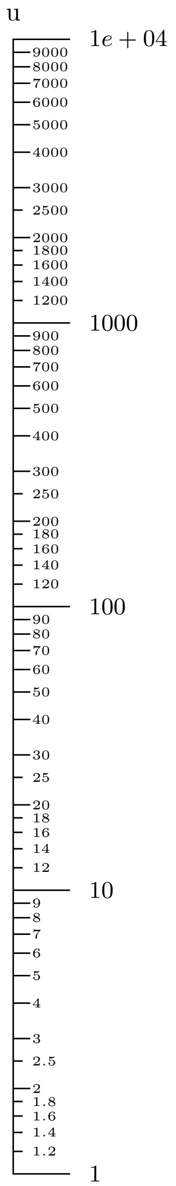
```
            'grid_length_1': 0.6/4,
            'grid_length_2': 0.5/4,
            'grid_length_3': 0.4/4,
            'grid_length_4': 0.3/4,
            'text_size_0': text.size.tiny,
            'text_size_1': text.size.tiny,
            'text_size_2': text.size.tiny,
            'text_size_3': text.size.tiny,
            'text_size_4': text.size.tiny,
            'text_distance_0': 1.2/4,
            'text_distance_1': 1.1/4,
            'text_distance_2': 1.0/4,
            'text_distance_3': 1.0/4,
            'text_distance_4': 1.0/4,
            'title_distance_center': 0.7,
            'title_opposite_tick': True,
            'title_draw_center': True,
            'text_format': "$%3.1f$",
            'full_angle': True,
            'extra_angle': 90.0,
            'text_horizontal_align_center': True,
            'text_format': r"$%2.0f$",
            'text_color': color.cmyk.Sepia,
            }
block_params = {'block_type': 'type_8',
                'f_params': N_params,
                'width': 5.0,
                'height': 15.0,
                }
main_params = {'filename': 'ex_axes_8_1.pdf',
               'paper_height': 10.0,
               'paper_width': 10.0,
               'block_params': [block_params],
               'transformations': [('scale paper',)]
               }
Nomographer(main_params)
```



5.1.6 Linear scale ('scale_type':'log')

Often one needs to use logarithmic functions in scales and 'scale_type':'log' makes some optimizations for this kind of scale appearance.

```
# ex_axes_9.py

N_params = {'u_min': 1.0,
            'u_max': 10000.0,
            'function': lambda u: log(u),
            'title': 'u',
            'scale_type': 'log',
            }
block_params = {'block_type': 'type_8',
                'f_params': N_params,
                'width': 5.0,
                'height': 15.0,
                }
main_params = {'filename': 'ex_axes_9.pdf',
               'paper_height': 15.0,
               'paper_width': 5.0,
               'block_params': [block_params],
               'transformations': [('scale paper',)]
              }
Nomographer(main_params)
```



5.1.7 Smart scales (`'scale_type':'smart linear'`, `'scale_type':'smart log'`)

Linear and log scales just plot ticks and texts as given with params `'tick_levels'` and `'tick_text_levels'`. Often this approach generates busy scales with overlapping texts and too dense ticks. Better approach is to use smart linear scales `'scale_type':'smart linear'` or smart log scales `'scale_type':'smart log'` These scales check that tick and text distances does not go below given thresholds (`'tick_distance_smart'` and `'text_distance_smart'`. TODO: example to use smart scales.

5.2 Common axis params

Table 5.1: Common axis params

parameter	default value	explanation
'ID'	'none'	**String.** To identify the axis.
'tag'	'none'	**String.** To align blocks w.r.t each other along axes with same tag.
'dtag'	'none'	**String.** To double-align blocks w.r.t each other along axes with same tag.
'title'	''	**String.** Axis title.
'title_x_shift'	0.0	**Float.** Title shift in x-direction.
'title_y_shift'	0.25	**Float.** Title shift in y-direction.
'scale_type'	'linear'	**String.** Scale type. Can be 'linear': linear scale. 'log': logarithmic scale. 'smart linear': linear scale with equal spacings. 'smart log': logarithmic scale with equal spacings, can also have negative values. 'manual point': Points and corresponding text positions are given manually in 'manual axis data'. No line is drawn. 'manual line': Ticks and corresponding text positions are given manually in 'manual axis data'.
'tick_levels'	4	**Integer.** How many levels (minor, minor-minor, etc.) of ticks are drawn. Largest effect to 'linear' scale.
'tick_text_levels'	'3'	**Integer.** How many levels (minor, minor-minor, etc.) of texts are drawn. Largest effect to 'linear' scale.
'tick_side'	'right'	**String.** Tick and text side in final paper. Can be: 'right''or '''left'
'reference'	False	**Boolean.** If axis is treated as reference line that is a turning point.
'reference_padding'	'0.2'	**Float.** Fraction of reference line over other lines.
'manual_axis_data'	{}	**Dict.** Manually set tick/point positions and text positions. Could be for example:{1:'1', 3.14:r'π', 5:'5', 7:'seven', 10:'10'}
'title_draw_center'	False	**Boolean.** Title is drawn to center of line.
'title_distance_center'	'type_9'	**String.** To double-align blocks w.r.t each other along axes with same tag.
'title_opposite_tick'	True	**Boolean.** Title in opposite direction w.r.t ticks.
'align_func'	lambda u:u	**func(u).** function to align different scales.
'align_x_offset'	0.0	**Float.** If axis is aligned with other axis, this value x offsets final scale.
'align_y_offset'	0.0	**Float.** If axis is aligned with other axis, this value y offsets final scale.
'text_format'	r'$%4.4g$ '	**String.** Format for numbers in scale.

Continued on next page

Table 5.1 – continued from previous page

parameter	default value	explanation
'extra_params'	[{},...]	**Array of Dicts.** List of dictionary of params to be drawn additionally.
'text_distance_#'	x.x	**Float.** where #=0,1,2,3 or 4. Distance of text from scale line. Number corresponds to the level, where 0 is the major tick and 4 is the most minor ticks.
'grid_length_#'	x.x	**Float.** where #=0,1,2,3 or 4. Length of the tick. Number corresponds to the level, where 0 is the major tick and 4 is the most minor ticks.
'text_size_#'	x.x	**Float.** where #=0,1,2,3 or 4. Text size. For example: `text.size.small`, `text.size.scriptsize` or `text.size.tiny`. Number corresponds to the level, where 0 is the major tick and 4 is the most minor ticks.
'text_size_log_#'	x.x	**Float.** where #=0,1 or 2. Text size. For example: `text.size.small`, `text.size.scriptsize` or `text.size.tiny` . Number corresponds to the level, where 0 is the major tick and 2 is the most minor ticks.
'full_angle'	False	**Boolean.** If true, text can be upside down, otherwise +- 90 degrees from horizontal. Good foor example for full circle scales.
'extra_angle'	0.0	**Boolean.** Title is drawn to center of line.
'title_draw_center'	False	**Float.** Angle to rotate tick text from horizontal along tick.
'text_horizontal_align_center'	False	**Boolean.** Aligns tick text horizontally to center. Good when text rotated 90 degrees.
'turn_relative'	False	**Boolean.** Side left or right is relative according to traveling of scale from min to max.
'arrow_size'	0.2	**Float.** Used with arrow scale.
'arrow_length'	1.0	**Float.** Used with arrow scale..
'arrow_color'	color.rgb.black	**Color.** Used with arrow scale.
'axis_color'	color.rgb.black	**Color.** Color of axis.
'text_color'	color.rgb.black	**Color.** Color of tick texts.
'extra_titles'	[]	**Array.** List of extra title dicts for scale. Could be i.e.“[{‘dx’:1.0, ‘dy’:1.0, ‘text’:’extra title 1’, ‘width’:5, ‘pyx_extra_defs’: [color.rgb.red,text.size.Huge]}, {‘text’: ‘extra title 2’}]“.

Continued on next page

Table 5.1 – continued from previous page

parameter	default value	explanation
'base_start'	None	**None/Float.** Defines number with 'base_stop' (instead of 'u_min' or 'u_max') to find major tick decades.
'base_stop'	None	**None/Float.** Defines number with 'base_start' (instead of 'u_min' or 'u_max') to find major tick decades.
'tick_distance_smart'	.05	**Float**. Minimum distance between smart ticks.
'text_distance_smart'	.25	**Float**. Minimum distance between smart texts.

CHAPTER

SIX

BLOCKS

Every block in pynomo represents some equation. The blocks and their functions are listed in the following table.

Type 1	$F_1(u_1) + F_2(u_2) + F_3(u_3) = 0$	Three parallel lines
Type 2	$F_1(u_1) = F_2(u_2)F_3(u_3)$	"N" or "Z"
Type 3	$F_1(u_1) + F_2(u_2) + \cdots + F_N(u_N) = 0$	N parallel lines
Type 4	$\frac{F_1(u_1)}{F_2(u_2)} = \frac{F_3(u_3)}{F_4(u_4)}$	"Proportion"
Type 5	$F_1(v) = F_2(x, u)$.	"Contour"
Type 6	$u = u$	"Ladder"
Type 7	$\frac{1}{F_1(u_1)} + \frac{1}{F_2(u_2)} = \frac{1}{F_3(u_3)}$	"Angle"
Type 8	$y = F(u)$	"Single"
Type 9	$\begin{vmatrix} F_1(u_1[, v_1]) & G_1(u_1[, v_1]) & H_1(u_1[, v_1]) \\ F_2(u_2[, v_2]) & G_2(u_2[, v_2]) & H_2(u_2[, v_2]) \\ F_3(u_3[, v_3]) & G_3(u_3[, v_3]) & H_3(u_3[, v_3]) \end{vmatrix} = 0$	"General"
Type 10	$F_1(u) + F_2(v)F_3(w) + F_4(w) = 0$	One curved line

6.1 Type 1

Type 1 is three parallel lines that have functional relationship:

$$F_1(u_1) + F_2(u_2) + F_3(u_3) = 0$$

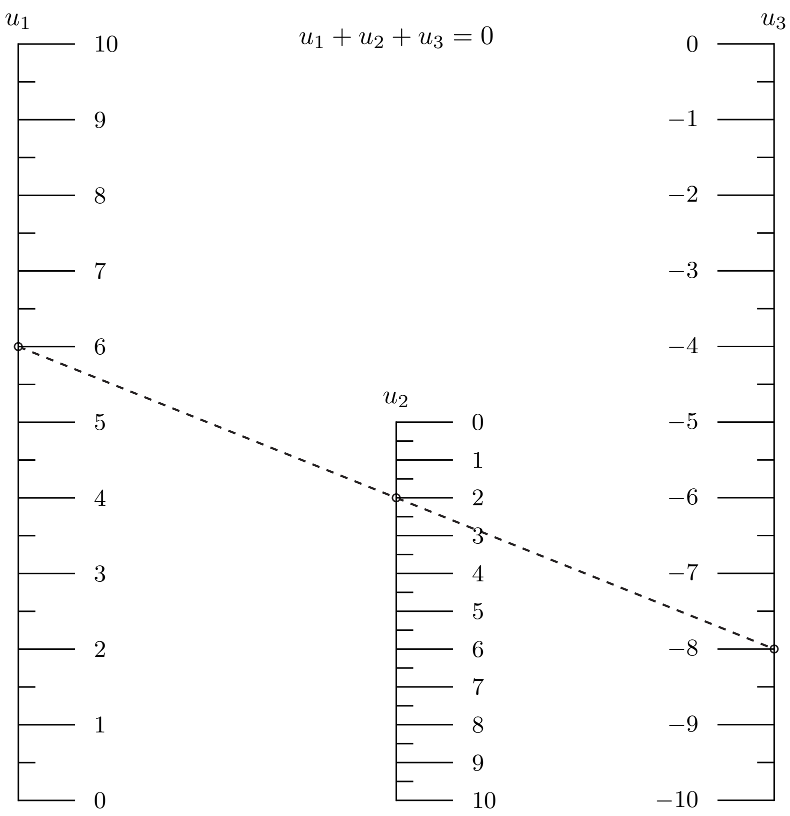
Note, that this kind of function can be transformed to many forms by using type 8 that is a equation given in determinant form. Use of this nomograph is given by the following simple example.

6.1.1 Simple example

This simple example plots nomograph for equation:

$$u_1 + u_2 + u_3 = 0.$$

Generated nomograph



Source code of simple example of type 1

```
"""
    ex_type1_nomo_1.py

    Simple nomogram of type 1: F1+F2+F3=0
"""
import sys
sys.path.insert(0, "..")
#sys.path[:0] = [".."]
from pynomo.nomographer import *

N_params_1={
        'u_min':0.0,
        'u_max':10.0,
        'function':lambda u:u,
        'title':r'$u_1$',
        'tick_levels':2,
        'tick_text_levels':1,
                }

N_params_2={
        'u_min':0.0,
        'u_max':10.0,
        'function':lambda u:u,
        'title':r'$u_2$',
        'tick_levels':2,
        'tick_text_levels':1,
                }

N_params_3={
        'u_min':0.0,
        'u_max':-10.0,
        'function':lambda u:u,
        'title':r'$u_3$',
```

```
        'tick_levels':2,
        'tick_text_levels':1,
                }


block_1_params={
             'block_type':'type_1',
             'width':10.0,
             'height':10.0,
             'f1_params':N_params_1,
             'f2_params':N_params_2,
             'f3_params':N_params_3,
             'isopleth_values':[[6,2,'x']],
             }

main_params={
              'filename':'ex_type1_nomo_1.pdf',
              'paper_height':10.0,
              'paper_width':10.0,
              'block_params':[block_1_params],
              'transformations':[('rotate',0.01),('scale paper',)],
              'title_str':r'$u_1+u_2+u_3=0$',
              'debug':False,
              }
Nomographer(main_params)
```

6.1.2 Parameters for type 1

Axis parameters

Table 6.1: Specific axis parameters for type 1

parameter key	default value	type, explanation
'function'	–	**func(u).** Function in equation For example `lambda u: u`
'u_min'	–	**Float.** Minimum value of function variable.
'u_max'	–	**Float.** Maximum value of function variable.

See *Common axis params* for other parameters.

Block parameters

Table 6.2: Specific block parameters for type 9

parameter	default value	explanation
'block_type'	'type_1'	**String.** This is type 1 block
'width'	10.0	**Float.** Block width (to be scaled)
'height'	10.0	**Float.** Block height (to be scaled)
'f1_params'	–	**Axis params Dict.** Axis params for function f1
'f2_params'	–	**Axis params Dict.** Axis params for function f2
'f3_params'	–	**Axis params Dict.** Axis params for function f3
'mirror_x'	False	**Boolean.** If x-axis is mirrored
'mirror_y'	False	**Boolean.** If y-axis is mirrored
'proportion'	1.0	**Float.** Factor for spacings between lines
'isopleth_values'	[[]]	** List of list of isopleth values.** Unknown values are given with strings, e.g. 'x'. An example:`[[0.8, 0.1, 'x'], ['x', 0.2, 1.0]]`

General parameters

See *Main params* for top level main parameters.

6.2 Type 2

Type 1 is "N" or "Z" nomograph that have functional relationship:

$$F_1(u_1) = F_2(u_2)F_3(u_3)$$

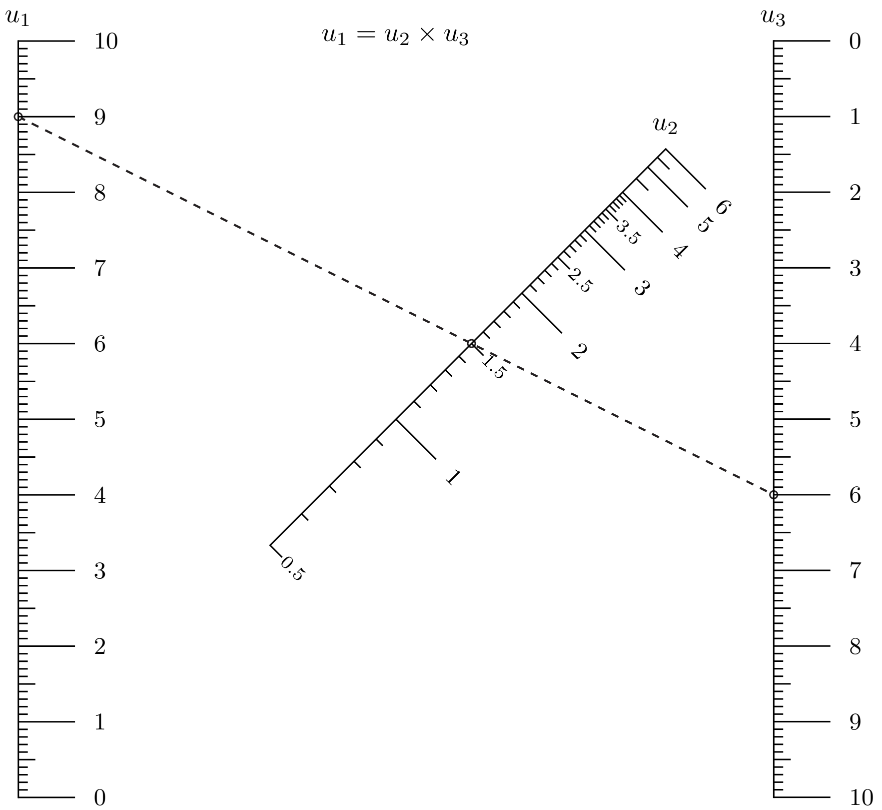
Use of this nomograph is given by the following simple example.

6.2.1 Simple example

This simple example plots nomograph for equation:

$$u_1 = u_2 u_3$$

Generated nomograph



Source code of simple example of type 2

```
"""
    ex_type2_nomo_1.py

    Simple nomogram of type 2: F1=F2*F3
"""
import sys
sys.path.insert(0, "..")
from pynomo.nomographer import *

N_params_1={
        'u_min':0.0,
        'u_max':10.0,
        'function':lambda u:u,
        'title':r'$u_1$',
        'tick_levels':3,
        'tick_text_levels':1,
                }

N_params_2={
        'u_min':0.5,
        'u_max':6.0,
        'function':lambda u:u,
        'title':r'$u_2$',
        'tick_levels':3,
        'tick_text_levels':2,
        'scale_type':'linear smart',
                }

N_params_3={
        'u_min':0.0,
        'u_max':10.0,
        'function':lambda u:u,
        'title':r'$u_3$',
```

```
        'tick_levels':3,
        'tick_text_levels':1,
                }


block_1_params={
             'block_type':'type_2',
             'width':10.0,
             'height':10.0,
             'f1_params':N_params_1,
             'f2_params':N_params_2,
             'f3_params':N_params_3,
             'isopleth_values':[[9,1.5,'x']],
             }

main_params={
              'filename':'ex_type2_nomo_1.pdf',
              'paper_height':10.0,
              'paper_width':10.0,
              'block_params':[block_1_params],
              'transformations':[('rotate',0.01),('scale paper',)],
              'title_str':r'$u_1=u_2\times u_3$'
              }
Nomographer(main_params)
```

6.2.2 Parameters for type 2

Axis parameters

Table 6.3: Specific axis parameters for type 2

parameter key	default value	type, explanation
'function'	–	**func(u).** Function in equation For example `lambda u: u`
'u_min'	–	**Float.** Minimum value of function variable.
'u_max'	–	**Float.** Maximum value of function variable.

See *Common axis params* for other parameters.

Block parameters

Table 6.4: Specific block parameters for type 2

parameter	default value	explanation
'block_type'	'type_2'	**String.** This is type 2 block
'width'	10.0	**Float.** Block width (to be scaled)
'height'	10.0	**Float.** Block height (to be scaled)
'f1_params'	–	**Axis params Dict.** Axis params for function f1
'f2_params'	–	**Axis params Dict.** Axis params for function f2
'f3_params'	–	**Axis params Dict.** Axis params for function f3
'mirror_x'	False	**Boolean.** If x-axis is mirrored
'mirror_y'	False	**Boolean.** If y-axis is mirrored
'proportion'	1.0	**Float.** Factor for spacings between lines
'isopleth_values'	[[]]	** List of list of isopleth values.** Unknown values are given with strings, e.g. 'x'. An example:`[[0.8, 0.1, 'x'], ['x', 0.2, 1.0]]`

General parameters

See *Main params* for top level main parameters.

6.3 Type 3

Type 3 has N parallel lines that have functional relationship:

$$F_1(u_1) + F_2(u_2) + \cdots + F_N(u_N) = 0$$

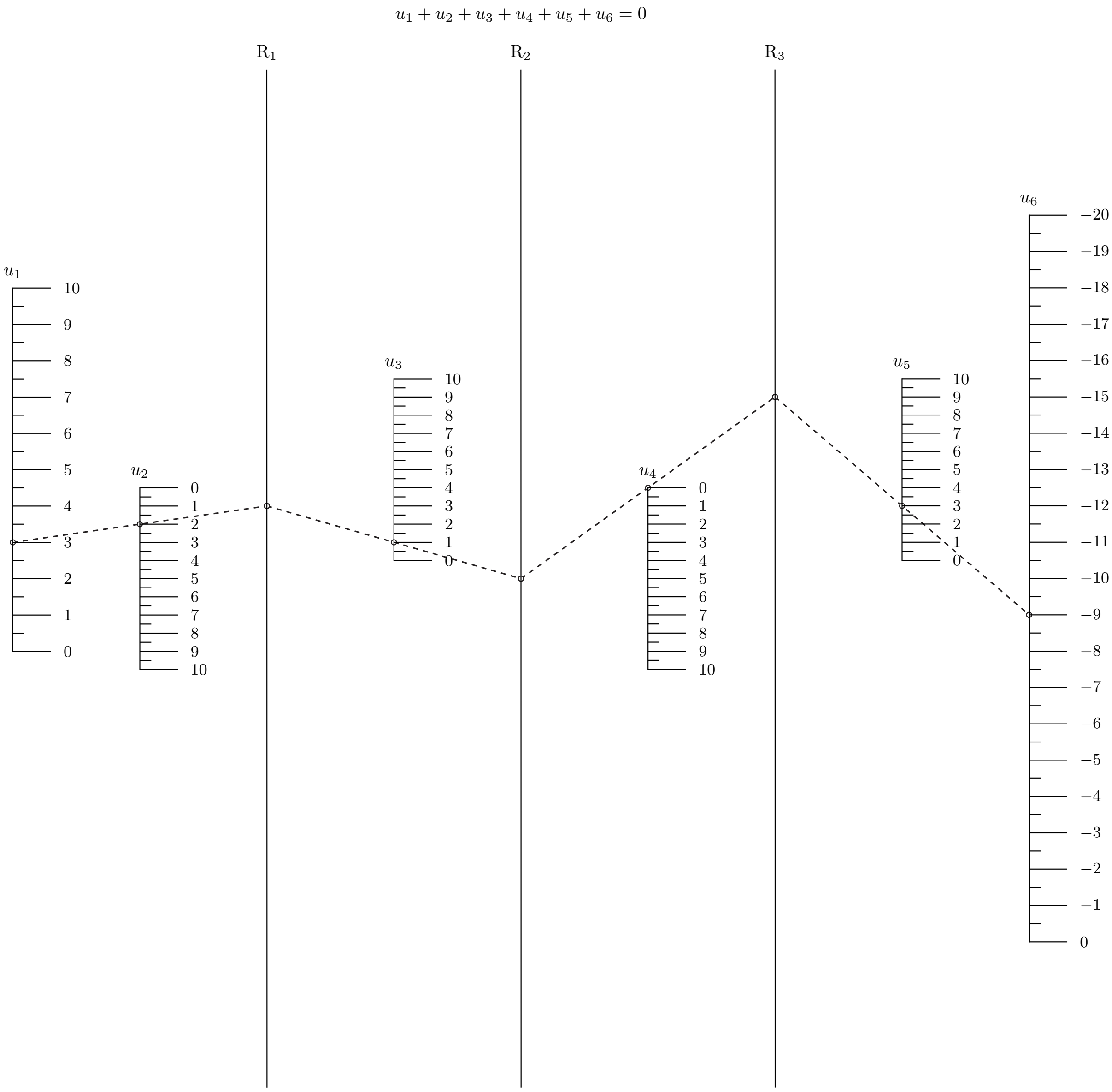
Use of this nomograph is given by the following simple example.

6.3.1 Simple example

This simple example plots nomograph for equation:

$$u_1 + u_2 + u_3 + u_4 + u_5 + u_6 = 0$$

Generated nomograph



Source code of simple example of type 2

```
"""
    ex_type3_nomo_1.py

    Simple nomogram of type 3: F1+F2+...+FN=0
    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
"""
import sys
sys.path.insert(0, "..")
from pynomo.nomographer import *

N_params_1={
        'u_min':0.0,
        'u_max':10.0,
        'function':lambda u:u,
        'title':r'$u_1$',
        'tick_levels':2,
        'tick_text_levels':1,
                }
N_params_2={
```

```
        'u_min':0.0,
        'u_max':10.0,
        'function':lambda u:u,
        'title':r'$u_2$',
        'tick_levels':2,
        'tick_text_levels':1,
                }
N_params_3={
        'u_min':0.0,
        'u_max':10.0,
        'function':lambda u:u,
        'title':r'$u_3$',
        'tick_levels':2,
        'tick_text_levels':1,
                }
N_params_4={
        'u_min':0.0,
        'u_max':10.0,
        'function':lambda u:u,
        'title':r'$u_4$',
        'tick_levels':2,
        'tick_text_levels':1,
                }
N_params_5={
        'u_min':0.0,
        'u_max':10.0,
        'function':lambda u:u,
        'title':r'$u_5$',
        'tick_levels':2,
        'tick_text_levels':1,
                }
N_params_6={
        'u_min':-20.0,
        'u_max':0.0,
        'function':lambda u:u,
        'title':r'$u_6$',
        'tick_levels':2,
        'tick_text_levels':1,
        'tick_side':'right',
                }

block_1_params={
             'block_type':'type_3',
             'width':10.0,
             'height':10.0,
             'f_params':[N_params_1,N_params_2,N_params_3,
                         N_params_4,N_params_5,N_params_6],
             'isopleth_values':[[3,2,1,0,3,'x']],
             }

main_params={
              'filename':'ex_type3_nomo_1.pdf',
              'paper_height':20.0,
              'paper_width':20.0,
              'block_params':[block_1_params],
              'transformations':[('rotate',0.01),('scale paper',)],
              'title_str':r'$u_1+u_2+u_3+u_4+u_5+u_6=0$',
              'title_y':21.0,
              }
Nomographer(main_params)
```

6.3.2 Parameters for type 3

Axis parameters

Table 6.5: Specific axis parameters for type 3

parameter key	default value	type, explanation
'function'	–	**func(u).** Function in equation For example `lambda u: u`
'u_min'	–	**Float.** Minimum value of function variable.
'u_max'	–	**Float.** Maximum value of function variable.

See *Common axis params* for other parameters.

Block parameters

Table 6.6: Specific block parameters for type 3

parameter	default value	explanation
'block_type'	'type_3'	**String.** This is type 3 block
'width'	10.0	**Float.** Block width (to be scaled)
'height'	10.0	**Float.** Block height (to be scaled)
'f_params'	–	**List of Axis params Dict.** List of Axis params.
'mirror_x'	False	**Boolean.** If x-axis is mirrored
'mirror_y'	False	**Boolean.** If y-axis is mirrored
'reference_padding'	0.2	**Float.** Additional length to reference axes.
'reference_titles'	[]	**Array of Strings.** List of reference line titles. For example *['R_1','R_2','R_3']'.*
'reference_color'	color.rgb.black	**Color.** Color of reference lines.
'isopleth_values'	[[]]	** List of list of isopleth values.** Unknown values are given with strings, e.g. 'x'. An example:`[[0.8,'x',0.7,7.0,9.0],[0.7,0.8,'x',5.0,4.44]]`

General parameters

See *Main params* for top level main parameters.

6.4 Type 4

Type 4 is proportion nomograph that have functional relationship:

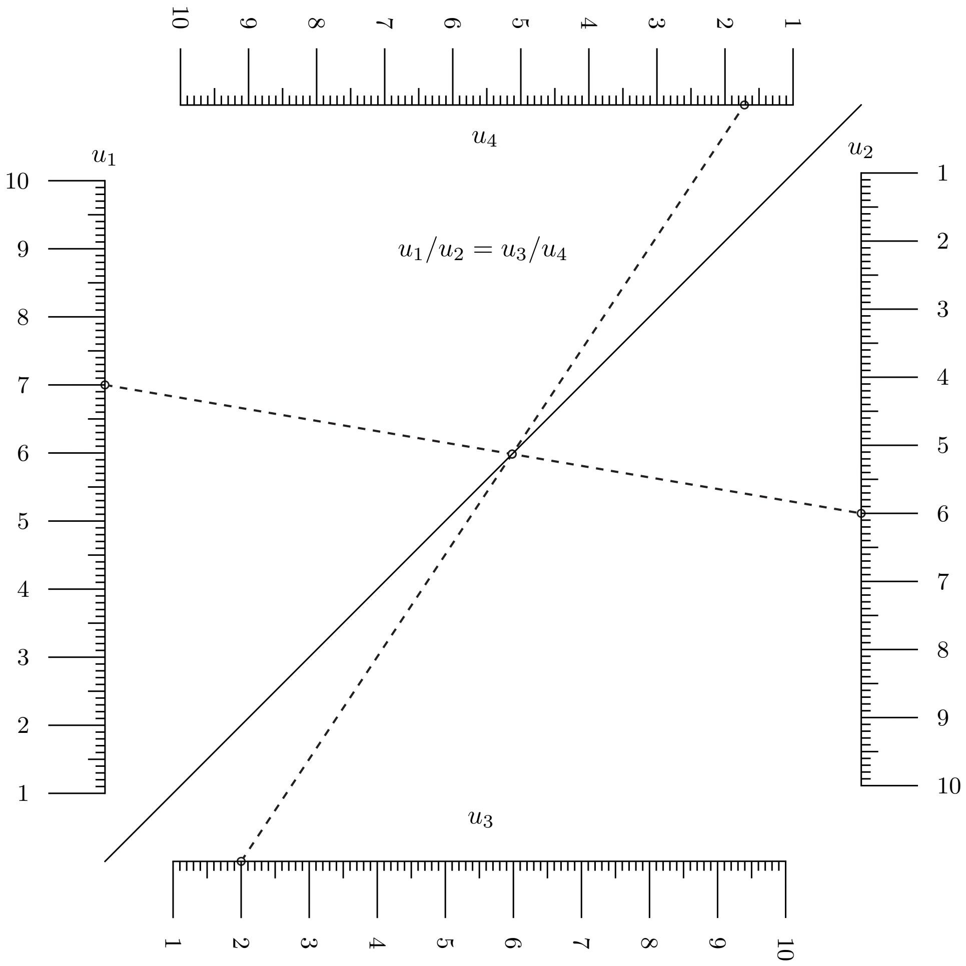
$$\frac{F_1(u_1)}{F_2(u_2)} = \frac{F_3(u_3)}{F_4(u_4)}$$

6.4.1 Simple example

This simple example plots nomograph for equation:

$$u_1/u_2 = u_3/u_4$$

Generated nomograph



Source code of simple example of type 4

```
"""
    ex_type4_nomo_1.py

    Simple nomogram of type 4: F1/F2=F3/F4
"""
import sys
sys.path.insert(0, "..")
from pynomo.nomographer import *

N_params_1={
        'u_min':1.0,
        'u_max':10.0,
        'function':lambda u:u,
        'title':r'$u_1$',
        'tick_levels':3,
        'tick_text_levels':1,
        'tick_side':'left',
```

```
                }
N_params_2={
        'u_min':1.0,
        'u_max':10.0,
        'function':lambda u:u,
        'title':r'$u_2$',
        'tick_levels':3,
        'tick_text_levels':1,
        'tick_side':'right',
                }
N_params_3={
        'u_min':1.0,
        'u_max':10.0,
        'function':lambda u:u,
        'title':r'$u_3$',
        'tick_levels':3,
        'tick_text_levels':1,
        'tick_side':'right',
        'title_draw_center':True,
        'title_opposite_tick':False,
                }
N_params_4={
        'u_min':1.0,
        'u_max':10.0,
        'function':lambda u:u,
        'title':r'$u_4$',
        'tick_levels':3,
        'tick_text_levels':1,
        'tick_side':'left',
        'title_draw_center':True,
        'title_opposite_tick':False,
                }

block_1_params={
                'block_type':'type_4',
                'f1_params':N_params_1,
                'f2_params':N_params_2,
                'f3_params':N_params_3,
                'f4_params':N_params_4,
                'isopleth_values':[[7,6,2,'x']],
                             }

main_params={
              'filename':'ex_type4_nomo_1.pdf',
              'paper_height':10.0,
              'paper_width':10.0,
              'block_params':[block_1_params],
              'transformations':[('rotate',0.01),('scale paper',)],
              'title_str':r'$u_1/u_2=u_3/u_4$',
              'title_y':8.0,
              }
Nomographer(main_params)
```

6.4.2 Parameters for type 4

Axis parameters

Table 6.7: Specific axis parameters for type 4

parameter key	default value	type, explanation
'function'	–	**func(u).** Function in equation For example `lambda u: u`
'u_min'	–	**Float.** Minimum value of function variable.
'u_max'	–	**Float.** Maximum value of function variable.

See *Common axis params* for other parameters.

Block parameters

Table 6.8: Specific block parameters for type 4

parameter	default value	explanation
'block_type'	'type_4'	**String.** This is type 4 block
'width'	10.0	**Float.** Block width (to be scaled)
'height'	10.0	**Float.** Block height (to be scaled)
'f1_params'	–	**Axis params Dict.** Axis params for function f1
'f2_params'	–	**Axis params Dict.** Axis params for function f2
'f3_params'	–	**Axis params Dict.** Axis params for function f3
'f4_params'	–	**Axis params Dict.** Axis params for function f4
'mirror_x'	False	**Boolean.** If x-axis is mirrored
'mirror_y'	False	**Boolean.** If y-axis is mirrored
'padding'	0.9	**Float.** How much axis extend w.r.t. width/height.
'float_axis'	'F1 or F2'	**Strings.** If given 'F1 or F2', then scaling is according to them, otherwise according to F3 and F4.
'reference_color'	color.rgb.black	**Color.** Color of reference lines.
'isopleth_values'	[[]]	** List of list of isopleth values.** Unknown values are given with strings, e.g. 'x'. An example:`[[0.8,'x',0.7,0.5],[0.7,0.8,'x',0.3]]`

General parameters

See *Main params* for top level main parameters.

6.5 Type 5

Type 5 is graphing block that has functional relationship:

$$F_1(u) = F_2(x, v).$$

This type of block is used commonly in nomographs that have an equation in form

$$f_a(a_1, a_2, a_3, ...) = f_b(u, v)$$

and :math:'f_b(u,v) cannot be represented as line-nomograph. Typically equation above is written as pair of equations:

$$f_a(a_1, a_2, a_3, ...) = x$$

and

$$f_b(u, v) = x.$$

This equation is written in form

$$F_1(u) = F_2(x, v).$$

in order to construct this contour block. In reality block consists of horizontal lines:

$$F_1(u) = y$$

and contour lines

$$F_2(x, v) = y,$$

where x and y are the coordinates of canvas. Coordinate x is reference with name wd in block parameters and it holds

$$x = f_{wd}(wd).$$

Note: Type 5 is a very complex (say stupid) way to make basic graphs. In the future versions of pynomo a more simple way for graphs will be implemented.

6.5.1 Simple example

In the following example

$$F_1(u) = u$$

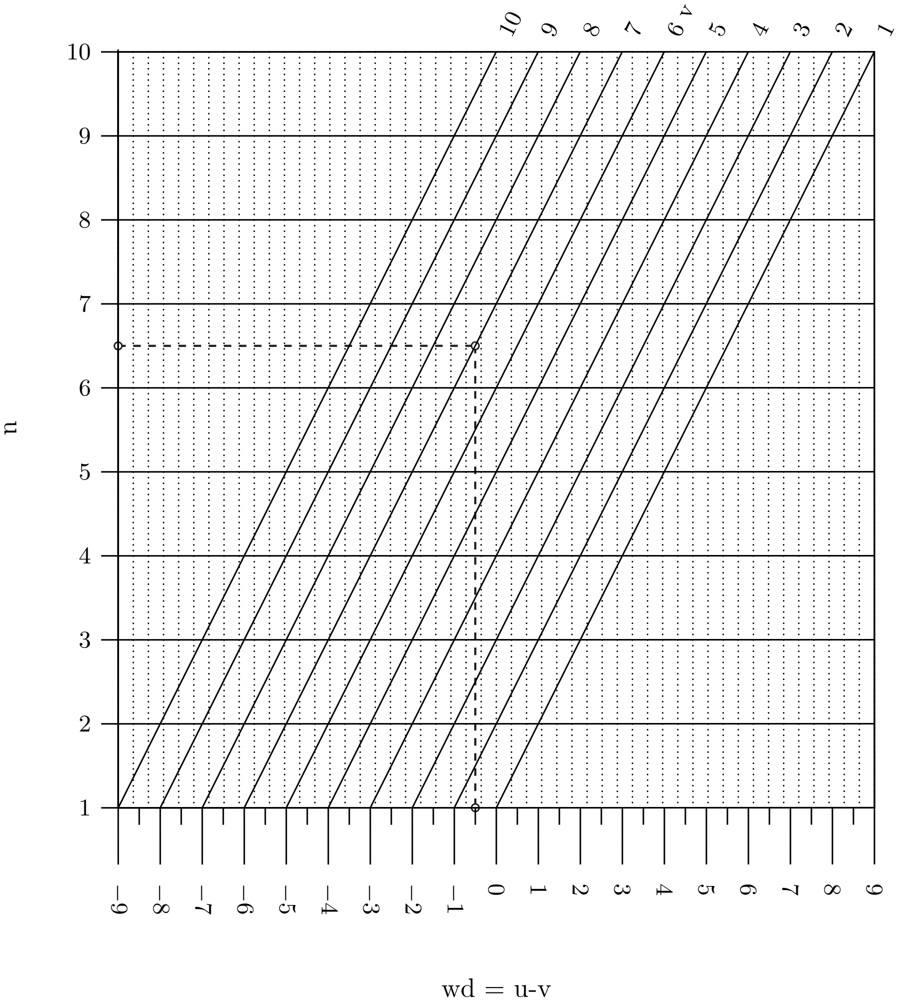
and

$$F_2(wd, v) = wd + v.$$

Thus the original equation is

$$wd = u - v.$$

Generated nomograph



Source code of simple example of type 5

```
"""
    ex_type5_nomo_1.py

    Simple nomogram of type 5.
"""
import sys
sys.path.insert(0, "..")
from pynomo.nomographer import *



block_params={
   'block_type':'type_5',
   'u_func':lambda u:u,
   'v_func':lambda x,v:x+v,
   'u_values':[1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0],
   'v_values':[1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0],
   'wd_tick_levels':2,
   'wd_tick_text_levels':1,
   'wd_tick_side':'right',
   'wd_title':'wd = u-v',
   'u_title':'u',
   'v_title':'v',
   'wd_title_opposite_tick':True,
   'wd_title_distance_center':2.5,
```

```
    'isopleth_values':[[6.5,7,'x']],
  }


main_params={
              'filename':'ex_type5_nomo_1.pdf',
              'paper_height':10.0,
              'paper_width':10.0,
              'block_params':[block_params],
              'transformations':[('rotate',0.01),('scale paper',)]
              }

Nomographer(main_params)
```

6.5.2 Parameters for type 5

Axis parameters

No specific axis parameters. Everything is defined in block.

Block parameters

Table 6.9: Specific block parameters for type 4

parameter	default value	explanation
'block_type'	'type_5'	**String.** This is type 5 block.
'width'	10.0	**Float.** Block width (to be scaled)
'height'	10.0	**Float.** Block height (to be scaled)
'mirror_x'	False	**Boolean.** If x-axis is mirrored
'mirror_y'	False	**Boolean.** If y-axis is mirrored
'u_func'	–	**func(u).** u function. For example `lambda u: u`
'v_func'	–	**func(u,v).** v function. For example `lambda x,v: x+v`
'wd_func'	–	**func(wd).** wd func. For example `lambda wd: wd`
'wd_func_inv'	–	**func(wd).** Inverse of wd-func. For example `lambda wd: wd`
'u_values'	–	**List of Floats.** List of plotted u values. For example *[1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0]‘.*
'u_tag'	'none'	**String.** To align blocks w.r.t each other along axes with same tag.
'u_title'	''	**String.** Axis title.
'u_title_x_shift'	0.0	**Float.** Title shift in x-direction.
'u_title_y_shift'	0.25	**Float.** Title shift in y-direction.

Continued on next page

Table 6.9 – continued from previous page

parameter	default value	explanation
'u_scale_type'	'linear'	**String.** Scale type. Can be 'linear': linear scale. 'log': logarithmic scale. 'smart linear': linear scale with equal spacings. 'smart log': logarithmic scale with equal spacings, can also have negative values. 'manual point': Points and corresponding text positions are given manually in 'manual axis data'. No line is drawn. 'manual line': Ticks and corresponding text positions are given manually in 'manual axis data'.
'u_tick_levels'	4	**Integer.** How many levels (minor, minor-minor, etc.) of ticks are drawn. Largest effect to 'linear' scale.
'u_tick_text_levels'	'3'	**Integer.** How many levels (minor, minor-minor, etc.) of texts are drawn. Largest effect to 'linear' scale.
'u_tick_side'	'right'	**String.** Tick and text side in final paper. Can be: 'right''or '''left'
'u_reference'	False	**Boolean.** If axis is treated as reference line that is a turning point.
'u_reference_padding'	'0.2'	**Float.** Fraction of reference line over other lines.
'u_manual_axis_data'	{}	**Dict.** Manually set tick/point positions and text positions. Could be for example:"{1:'1', 3.14:r'pi', 5:'5',7:'seven', 10:'10'}'
'u_title_draw_center'	False	**Boolean.** Title is drawn to center of line.
'u_title_distance_center'	'type_9'	**String.** To double-align blocks w.r.t each other along axes with same tag.
'u_title_opposite_tick'	True	**Boolean.** Title in opposite direction w.r.t ticks.
'u_align_func'	lambda u:u	**func(u).** function to align different scales.
'u_align_x_offset'	0.0	**Float.** If axis is aligned with other axis, this value x offsets final scale.
'u_align_y_offset'	0.0	**Float.** If axis is aligned with other axis, this value y offsets final scale.
'u_text_format'	r'$%4.4g$ '	**String.** Format for numbers in scale.
'u_extra_params'	[{},...]	**Array of Dicts.** List of dictionary of params to be drawn additionally.
'u_text_distance_#'	x.x	**Float.** where #=0,1,2,3 or 4. Distance of text from scale line. Number corresponds to the level, where 0 is the major tick and 4 is the most minor ticks.

Continued on next page

Table 6.9 – continued from previous page

parameter	default value	explanation
'u_grid_length_#'	x.x	**Float.** where #=0,1,2,3 or 4. Length of the tick. Number corresponds to the level, where 0 is the major tick and 4 is the most minor ticks.
'u_text_size_#'	x.x	**Float.** where #=0,1,2,3 or 4. Text size. For example: `text.size.small`, `text.size.scriptsize` or `text.size.tiny`. Number corresponds to the level, where 0 is the major tick and 4 is the most minor ticks.
'u_text_size_log_#'	x.x	**Float.** where #=0,1 or 2. Text size. For example: `text.size.small`, `text.size.scriptsize` or `text.size.tiny` . Number corresponds to the level, where 0 is the major tick and 2 is the most minor ticks.
'u_full_angle'	False	**Boolean.** If true, text can be upside down, otherwise +- 90 degrees from horizontal. Good foor example for full circle scales.
'u_extra_angle'	0.0	**Boolean.** Title is drawn to center of line.
'u_text_horizontal_align_center'	False	**Boolean.** Aligns tick text horizontally to center. Good when text rotated 90 degrees.
'u_axis_color'	color.rgb.black	**Color.** Color of axis.
'u_text_color'	color.rgb.black	**Color.** Color of tick texts.
'v_values'	–	**List of Floats.** List of plotted v values. For example *[1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0]*'.
'v_title'	''	**String.** Axis title.
'v_title_draw_center'	False	**Boolean.** Title is drawn to center of line.
'v_title_distance_center'	'type_9'	**String.** To double-align blocks w.r.t each other along axes with same tag.
'v_title_opposite_tick'	True	**Boolean.** Title in opposite direction w.r.t ticks.
'wd_tag'	'none'	**String.** To align blocks w.r.t each other along axes with same tag.
'wd_title'	''	**String.** Axis title.
'wd_title_x_shift'	0.0	**Float.** Title shift in x-direction.
'wd_title_y_shift'	0.25	**Float.** Title shift in y-direction.

Continued on next page

Table 6.9 – continued from previous page

parameter	default value	explanation
'wd_scale_type'	'linear'	**String.** Scale type. Can be 'linear': linear scale. 'log': logarithmic scale. 'smart linear': linear scale with equal spacings. 'smart log': logarithmic scale with equal spacings, can also have negative values. 'manual point': Points and corresponding text positions are given manually in 'manual axis data'. No line is drawn. 'manual line': Ticks and corresponding text positions are given manually in 'manual axis data'.
'wd_tick_levels'	4	**Integer.** How many levels (minor, minor-minor, etc.) of ticks are drawn. Largest effect to 'linear' scale.
'wd_tick_text_levels'	'3'	**Integer.** How many levels (minor, minor-minor, etc.) of texts are drawn. Largest effect to 'linear' scale.
'wd_tick_side'	'right'	**String.** Tick and text side in final paper. Can be: 'right''or '''left'
'wd_reference'	False	**Boolean.** If axis is treated as reference line that is a turning point.
'wd_reference_padding'	'0.2'	**Float.** Fraction of reference line over other lines.
'wd_manual_axis_data'	{}	**Dict.** Manually set tick/point positions and text positions. Could be for example:{1:'1', 3.14:r'π', 5:'5', 7:'seven', 10:'10'}
'wd_title_draw_center'	False	**Boolean.** Title is drawn to center of line.
'wd_title_distance_center'	'type_9'	**String.** To double-align blocks w.r.t each other along axes with same tag.
'wd_title_opposite_tick	True	**Boolean.** Title in opposite direction w.r.t ticks.
'wd_align_func'	lambda u:u	**func(u).** function to align different scales.
'wd_align_x_offset'	0.0	**Float.** If axis is aligned with other axis, this value x offsets final scale.
'wd_align_y_offset'	0.0	**Float.** If axis is aligned with other axis, this value y offsets final scale.
'wd_text_format'	r'$%4.4g$ '	**String.** Format for numbers in scale.
'wd_extra_params'	[{},...]	**Array of Dicts.** List of dictionary of params to be drawn additionally.
'wd_text_distance_#'	x.x	**Float.** where #=0,1,2,3 or 4. Distance of text from scale line. Number corresponds to the level, where 0 is the major tick and 4 is the most minor ticks.

Continued on next page

Table 6.9 – continued from previous page

parameter	default value	explanation
'wd_grid_length_#'	x.x	**Float.** where #=0,1,2,3 or 4. Length of the tick. Number corresponds to the level, where 0 is the major tick and 4 is the most minor ticks.
'wd_text_size_#'	x.x	**Float.** where #=0,1,2,3 or 4. Text size. For example: `text.size.small`, `text.size.scriptsize` or `text.size.tiny`. Number corresponds to the level, where 0 is the major tick and 4 is the most minor ticks.
'wd_text_size_log_#'	x.x	**Float.** where #=0,1 or 2. Text size. For example: `text.size.small`, `text.size.scriptsize` or `text.size.tiny` . Number corresponds to the level, where 0 is the major tick and 2 is the most minor ticks.
'wd_full_angle'	False	**Boolean.** If true, text can be upside down, otherwise +- 90 degrees from horizontal. Good foor example for full circle scales.
'wd_extra_angle'	0.0	**Boolean.** Title is drawn to center of line.
'wd_text_horizontal_align_center'	False	**Boolean.** Aligns tick text horizontally to center. Good when text rotated 90 degrees.
'wd_axis_color'	color.rgb.black	**Color.** Color of axis.
'wd_text_color'	color.rgb.black	**Color.** Color of tick texts.
'isopleth_values'	[[]]	** List of list of isopleth values.** Unknown values are given with strings, e.g. 'x'. An example:`[[0.8,'x',0.7],[0.7,0.8,'x']]`

General parameters

See *Main params* for top level main parameters.

6.6 Type 6

Type 6 is ladder nomograph:

$$u = u.$$

In practice this means that if one axis has for example y-position as

$$y = f_1(u)$$

and it was desirable to have

$$y = f_2(u)$$

in order to connect blocks together, one uses ladder to make the transformation.

Note: Ladders are not beautiful and should be used only when no other solution exist.

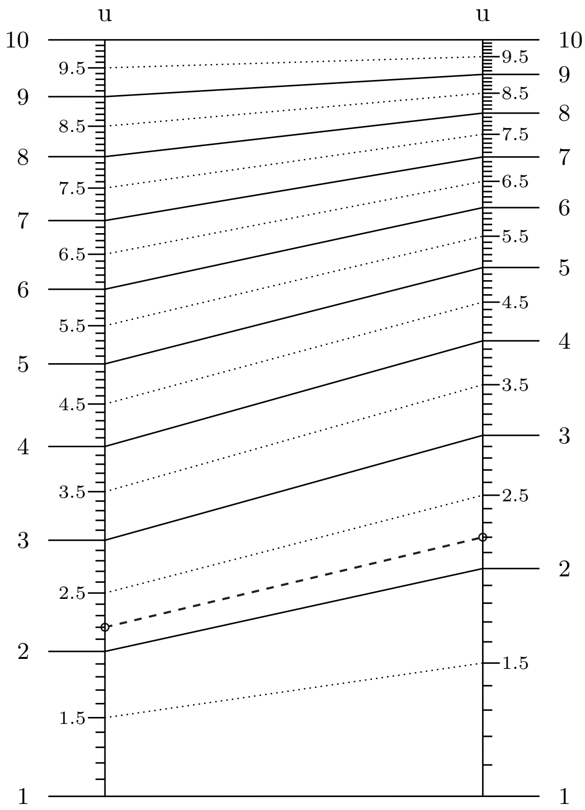
6.6.1 Simple example

This simple example plots nomograph for equation:

$$u = u,$$

where linear scale is converted to a logarithmic scale.

Generated nomograph



Source code of simple example of type6

```
"""
    ex_type6_nomo_1.py

    Simple nomogram of type 6.
"""
import sys
sys.path.insert(0, "..")
from pynomo.nomographer import *

N_params_1={
        'u_min':1.0,
        'u_max':10.0,
        'function':lambda u:u**0.5,
        'title':'u',
```

```
        'tick_levels':3,
        'tick_text_levels':2,
        'tick_side':'left',
        }

N_params_2={
        'u_min':1.0,
        'u_max':10.0,
        'function':lambda u:log(u),
        'title':'u',
        'tick_levels':3,
        'tick_text_levels':2,
        }

block_params={
              'block_type':'type_6',
              'f1_params':N_params_1,
              'f2_params':N_params_2,
              'width':5.0,
              'height':10.0,
              'isopleth_values':[[2.2,'x']],
              #'curve_const':0.01
                     }

main_params={
              'filename':'ex_type6_nomo_1.pdf',
              'paper_height':10.0,
              'paper_width':5.0,
              'block_params':[block_params],
              'transformations':[('rotate',0.01),('scale paper',)]
              }

Nomographer(main_params)
```

6.6.2 Parameters for type 6

Axis parameters

Table 6.10: Specific axis parameters for type 6

parameter key	default value	**type**, explanation
'function'	–	**func(u).** Function in equation For example lambda u: u
'u_min'	–	**Float.** Minimum value of function variable.
'u_max'	–	**Float.** Maximum value of function variable.

See *Common axis params* for other parameters.

Block parameters

Table 6.11: Specific block parameters for type 6

parameter	default value	explanation
'block_type'	'type_6'	**String.** This is type 6 block.
'type'	'parallel'	**String.** Can be either 'parallel'``or ``'orthogonal'.
'x_empty'	0.2	**Float.** If orthogonal, how much fractional space before start of x-axis.
'y_empty'	0.2	**Float.** If orthogonal, how much fractional space before start of y-axis.
'curve_const'	0.0	**Float.** Sets the lenght of angle of Bezier curve. low value = straigh line, high value = curved line.
'width'	10.0	**Float.** Block width (to be scaled)
'height'	10.0	**Float.** Block height (to be scaled)
'f1_params'	–	**Axis params Dict.** Axis params for function f1
'f2_params'	–	**Axis params Dict.** Axis params for function f2
'mirror_x'	False	**Boolean.** If x-axis is mirrored
'mirror_y'	False	**Boolean.** If y-axis is mirrored
'ladder_color'	color.rgb.black	**Color.** Ladder color.
'isopleth_values'	[[]]	** List of list of isopleth values.** Unknown values are given with strings, e.g. 'x'. An example:[[0.8,'x'],[0.7,'x']]

General parameters

See *Main params* for top level main parameters.

6.7 Type 7

Type 7 is "angle" nomograph that has functional relationship:

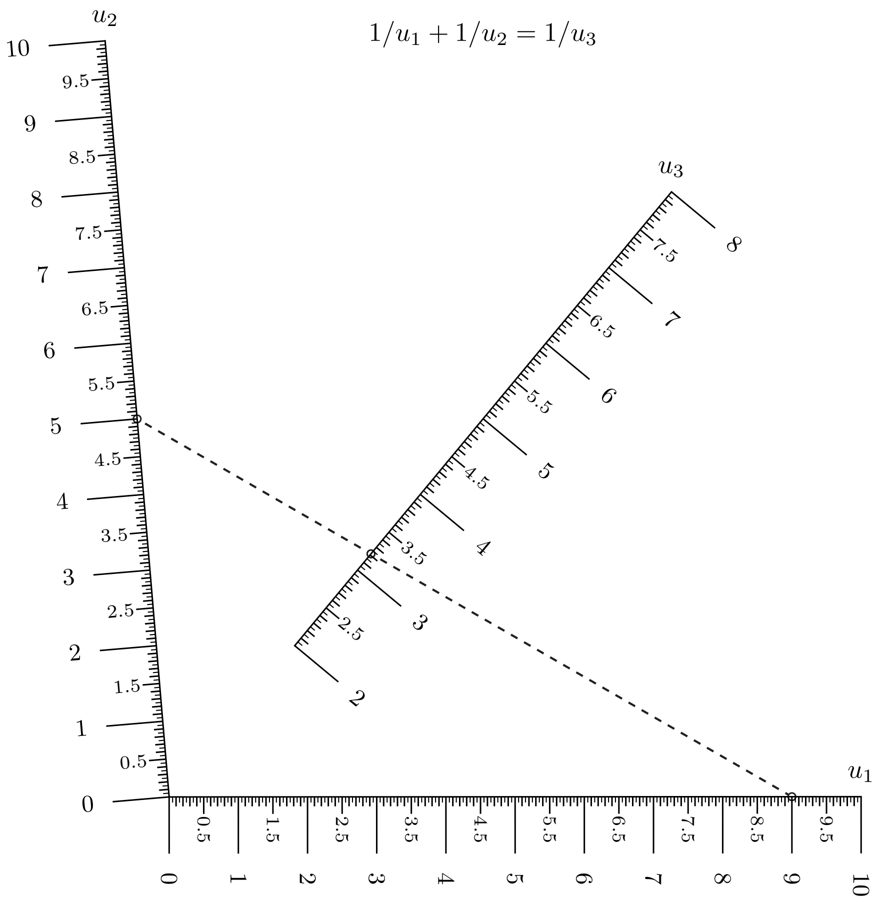
$$\frac{1}{F_1(u_1)} + \frac{1}{F_2(u_2)} = \frac{1}{F_3(u_3)}$$

6.7.1 Simple example

This simple example plots nomograph for equation:

$$1/u_1 + 1/u_2 = 1/u_3$$

Generated nomograph



Source code of simple example of type 2

6.7.2 Parameters for type 7

Axis parameters

Table 6.12: Specific axis parameters for type 7

parameter key	default value	**type**, explanation
`'function'`	–	**func(u).** Function in equation For example `lambda u: u`
`'u_min'`	–	**Float.** Minimum value of function variable.
`'u_max'`	–	**Float.** Maximum value of function variable.

See *Common axis params* for other parameters.

Block parameters

Table 6.13: Specific block parameters for type 7

parameter	default value	explanation
'block_type'	'type_4'	**String.** This is type 7 block
'width'	10.0	**Float.** Block width (to be scaled)
'height'	10.0	**Float.** Block height (to be scaled)
'f1_params'	–	**Axis params Dict.** Axis params for function f1
'f2_params'	–	**Axis params Dict.** Axis params for function f2
'f3_params'	–	**Axis params Dict.** Axis params for function f3
'mirror_x'	False	**Boolean.** If x-axis is mirrored
'mirror_y'	False	**Boolean.** If y-axis is mirrored
'angle_u'	45.0	**Float.** Angle between u1 and u3. Note: later transformations may alter the angle.
'angle_v'	45.0	**Float.** Angle between u2 and u3. Note: later transformations may alter the angle.
'isopleth_values'	[[]]	** List of list of isopleth values.** Unknown values are given with strings, e.g. 'x'. An example:`[[0.8,'x',0.7],[0.7,0.8,'x']]`

General parameters

See *Main params* for top level main parameters.

6.8 Type 8

Type 8 is single nomograph:

$$y = F(u)$$

or

$$x = F_x(u),$$

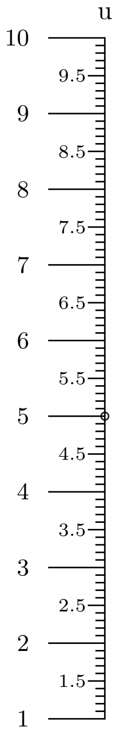
$$y = F_y(u).$$

x and y are coordinates of canvas. Often this block is used for construction of dual-scales to existing scales.

6.8.1 Simple example

This simple example plots single vertical scale.

Generated nomograph



Source code of simple example of type 8

```
"""
    ex_type8_nomo_1.py

    Simple nomogram of type 8.
"""
import sys
sys.path.insert(0, "..")
from pynomo.nomographer import *

N_params_1={
        'u_min':1.0,
        'u_max':10.0,
        'function':lambda u:u,
        'title':'u',
        'tick_levels':3,
        'tick_text_levels':2,
        'tick_side':'left',
        }

block_params={
              'block_type':'type_8',
              'f_params':N_params_1,
              'width':5.0,
              'height':10.0,
              'isopleth_values':[[5]]
                     }

main_params={
              'filename':'ex_type8_nomo_1.pdf',
              'paper_height':10.0,
              'paper_width':5.0,
              'block_params':[block_params],
              'transformations':[]
              }

Nomographer(main_params)
```

6.8.2 Parameters for type 8

Axis parameters

Table 6.14: Specific axis parameters for type 8

parameter key	default value	type, explanation
'function'	–	**func(u).** Function in equation. For example `lambda u: u`.
'u_min'	–	**Float.** Minimum value of function variable.
'u_max'	–	**Float.** Maximum value of function variable.
'function_x'	–	**func(u).** x-position in function. If used 'function_y' must be defined. For example `lambda u: u`.
'function_y'	–	**func(u).** y-position in function. If used 'function_x' must be defined. Overrides 'function'. For example `lambda u: u`.

See *Common axis params* for other parameters.

Block parameters

Table 6.15: Specific block parameters for type 8

parameter	default value	explanation
'block_type'	'type_8'	**String.** This is type 8 block
'width'	10.0	**Float.** Block width (to be scaled)
'height'	10.0	**Float.** Block height (to be scaled)
'f1_params'	–	**Axis params Dict.** Axis params for function f1
'f2_params'	–	**Axis params Dict.** Axis params for function f2
'f3_params'	–	**Axis params Dict.** Axis params for function f3
'f4_params'	–	**Axis params Dict.** Axis params for function f4
'mirror_x'	False	**Boolean.** If x-axis is mirrored
'mirror_y'	False	**Boolean.** If y-axis is mirrored
'padding'	0.9	**Float.** How much axis extend w.r.t. width/height.
'float_axis'	'F1 or F2'	**Strings.** If given 'F1 or F2', then scaling is according to them, otherwise according to F3 and F4.
'reference_color'	color.rgb.black	**Color.** Color of reference lines.
'isopleth_values'	[[]]	** List of list of isopleth values.** Unknown values are given with strings, e.g. 'x'. An example:`[[0.8,'x',0.7,0.5], [0.7,0.8,'x',0.3]]`

General parameters

See *Main params* for top level main parameters.

6.9 Type 9

Type 9 is "general determinant" nomograph that has functional relationship:

$$\begin{vmatrix} F_1(u_1[,v_1]) & G_1(u_1[,v_1]) & H_1(u_1[,v_1]) \\ F_2(u_2[,v_2]) & G_2(u_2[,v_2]) & H_2(u_2[,v_2]) \\ F_3(u_3[,v_3]) & G_3(u_3[,v_3]) & H_3(u_3[,v_3]) \end{vmatrix} = 0.$$

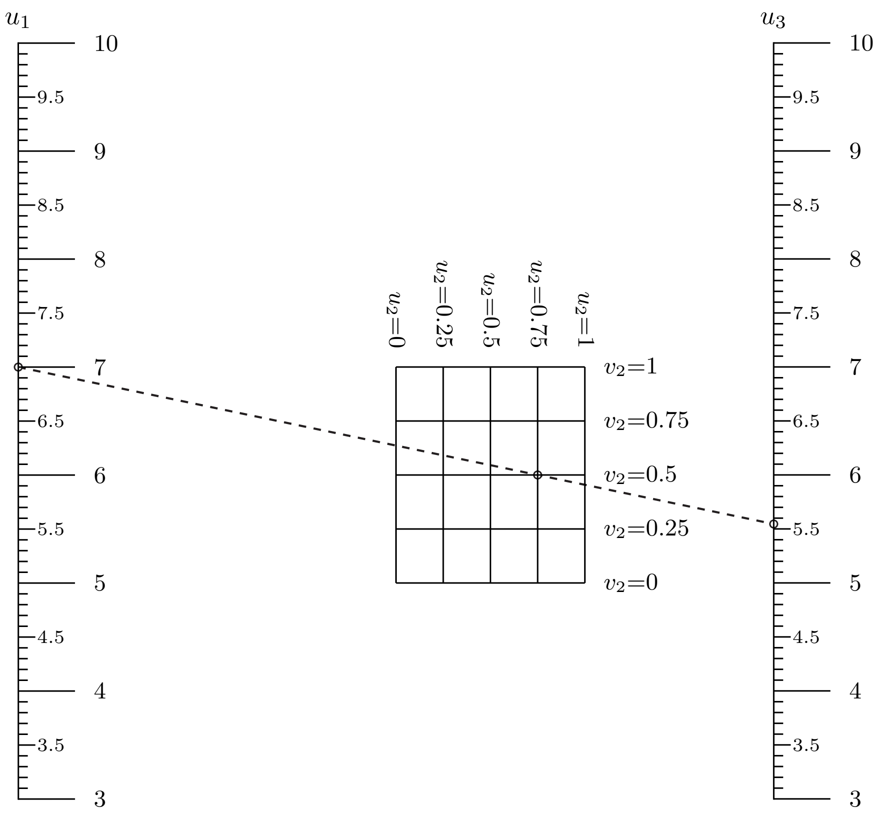
This is the basic building block for line nomographs. Notation $u[,v]$ is to be understood such that if v is defined´, a grid is constructed for the row, otherwise a normal scale with variable u.

6.9.1 Simple example

This simple example plots nomograph for equation in determinant form:

$$\begin{vmatrix} 0 & u_1 & 1 \\ u_2+2 & 2v_2+5 & 1 \\ 4 & u_3 & 1 \end{vmatrix} = 0$$

Generated nomograph



Source code of simple example of type 9

```
"""
    ex_type9_nomo_1.py

    Simple nomogram of type 9: determinant
"""
import sys
sys.path.insert(0, "..")
from pynomo.nomographer import *

N_params_1={
            'u_min':3.0,
            'u_max':10.0,
            'f':lambda u:0,
            'g':lambda u:u,
            'h':lambda u:1.0,
            'title':r'$u_1$',
            'scale_type':'linear',
            'tick_levels':3,
            'tick_text_levels':2,
            'grid':False}

N_params_2={
        'u_min':0.0, # for alignment
        'u_max':1.0,  # for alignment
        'f_grid':lambda u,v:u+2.0,
        'g_grid':lambda u,v:2*v+5.0,
        'h_grid':lambda u,v:1.0,
        'u_start':0.0,
        'u_stop':1.0,
        'v_start':0.0,
        'v_stop':1.0,
        'u_values':[0.0,0.25,0.5,0.75,1.0],
        'v_values':[0.0,0.25,0.5,0.75,1.0],
        'grid':True,
        'text_prefix_u':r'$u_2$=',
        'text_prefix_v':r'$v_2$=',
        }

N_params_3={
            'u_min':3.0,
            'u_max':10.0,
            'f':lambda u:4.0,
            'g':lambda u:u,
            'h':lambda u:1.0,
            'title':r'$u_3$',
            'scale_type':'linear',
            'tick_levels':3,
            'tick_text_levels':2,
            'grid':False
            }

block_params={
             'block_type':'type_9',
             'f1_params':N_params_1,
             'f2_params':N_params_2,
             'f3_params':N_params_3,
             'transform_ini':False,
             'isopleth_values':[[7,[0.75,0.5],'x']]
             }

main_params={
              'filename':'ex_type9_nomo_1.pdf',
              'paper_height':10.0,
              'paper_width':10.0,
              'block_params':[block_params],
              'transformations':[('rotate',0.01),('scale paper',)]
              }
Nomographer(main_params)
```

6.9.2 Parameters for type 9

Axis parameters

Table 6.16: Specific axis parameters for type 9 grid axis

parameter key	default value	type, explanation
'grid'	–	**Bool.** True because this is grid.
'f'	–	**func(u,v).** F function in determinant. For example lambda u,v:u+v
'g'	–	**func(u,v).** G function in determinant. For example lambda u,v:u+v
'h'	–	**func(u,v).** H function in determinant. For example lambda u,v:u+v
'u_start'	–	u start when drawing v=const line
'u_stop'	–	u stop when drawing v=const line
'v_start'	–	v start when drawing u=const line
'v_stop'	–	v stop when drawing u=const line
'u_values'	–	List of grid lines u=const. For example [0.0,0.25,0.5,0.75,1.0]
'v_values'	–	List of grid lines v=const. For example “[0.0,0.25,0.5,0.75,1.0]‘
'text_prefix_u'	–	Text prefix for u before value
'text_prefix_v'	–	Text prefix for v before value
'v_texts_u_start'	False	If v-texts are in u start side
'v_texts_u_stop'	True	If v-texts are in u stop side
'u_texts_v_start'	False	If u-texts are in v start side
'u_texts_v_stop'	True	If u-texts are in v stop side
'u_line_color'	color.rgb.black	**Color.** u line color
'v_line_color'	color.rgb.black	**Color.** v line color
'u_text_color'	color.rgb.black	**Color.** u text color
'v_text_color'	color.rgb.black	**Color.** v text color
'text_distance'	0.25	**Float.** Text distance
'circles'	False	**Boolean.** If marker circles to crossings
'extra_params'	–	**List of Dicts.** List of params to be drawn.

See *Common axis params* for other parameters.

Block parameters

Table 6.17: Specific block parameters for type 9

parameter	default value	explanation
'block_type'	'type_9'	**String.** This is type 9 block
'width'	10.0	**Float.** Block width (to be scaled)
'height'	10.0	**Float.** Block height (to be scaled)
'f1_params'	–	**Axis params Dict.** Axis params for function f1
'f2_params'	–	**Axis params Dict.** Axis params for function f2
'f3_params'	–	**Axis params Dict.** Axis params for function f3
'mirror_x'	False	**Boolean.** If x-axis is mirrored
'mirror_y'	False	**Boolean.** If y-axis is mirrored
'transform_ini'	False	**Boolean.** If row 1 and row 3 end and start are to be transformed to be in rectangle corners. If True, be sure that 'u_min_trafo' and 'u_max_trafo' are defined.
'isopleth_values'	[[]]	** List of list of isopleth values.** Grid values are given with tuple (a,b) and are not solved. Unknown values are given with strings, e.g. 'x'. An example:`[[0.8,(0.1,0.2),'x'], ['x',(0.1,0.2),1.0]]`

General parameters

See *Main params* for top level main parameters.

6.10 Type 10

Type 10 is nomograph that has one curved line. It has functional relationship:

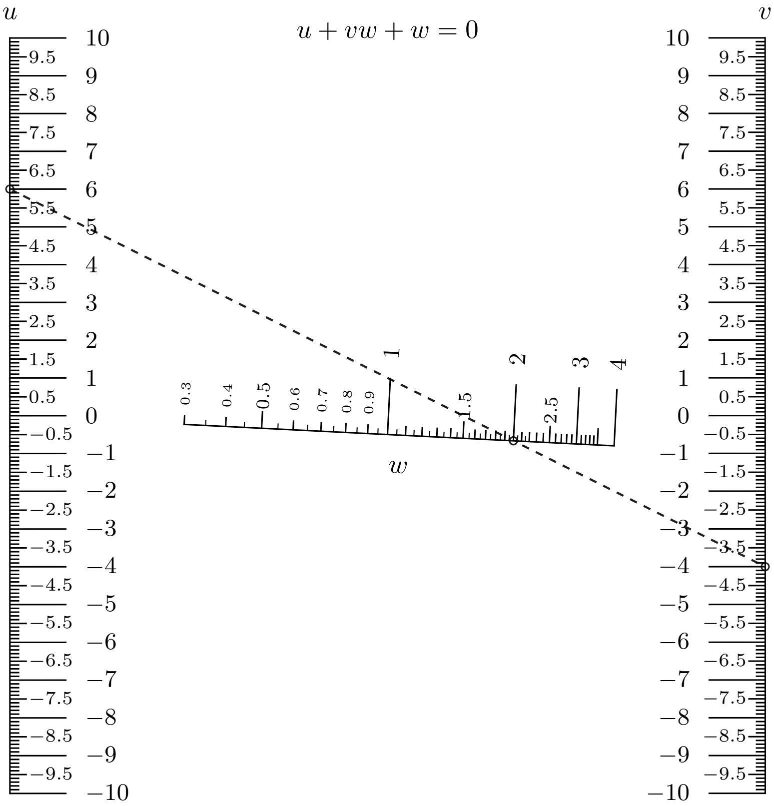
$$F_1(u) + F_2(v)F_3(w) + F_4(w) = 0.$$

6.10.1 Simple example

This simple example plots nomograph for equation:

$$u + vw + w = 0.$$

Generated nomograph



Source code of simple example of type 10

```
"""
    ex_type10_nomo_1.py

    Simple nomogram of type 7: F1(u)+F2(v)*F3(w)+F4(w)=0
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
"""
import sys
sys.path.insert(0, "..")
from pynomo.nomographer import *

N_params_1={
        'u_min':-10.0,
        'u_max':10.0,
        'function':lambda u:u,
        'title':r'$u$',
        'tick_levels':3,
        'tick_text_levels':2,
                }

N_params_2={
        'u_min':-10.0,
        'u_max':10.0,
        'function':lambda u:u,
        'title':r'$v$',
        'tick_levels':3,
        'tick_text_levels':2,
        'tick_side':'left',
                }

N_params_3={
        'u_min':0.3,
        'u_max':4.0,
        'function_3':lambda u:u,
```

```
        'function_4':lambda u:u,
        'title':r'$w$',
        'tick_levels':4,
        'tick_text_levels':3,
        'scale_type':'linear smart',
        'title_draw_center':True,
                }

block_1_params={
             'block_type':'type_10',
             'width':10.0,
             'height':10.0,
             'f1_params':N_params_1,
             'f2_params':N_params_2,
             'f3_params':N_params_3,
             'isopleth_values':[[6,-4,'x']]
             }

main_params={
              'filename':'ex_type10_nomo_1.pdf',
              'paper_height':10.0,
              'paper_width':10.0,
              'block_params':[block_1_params],
              'transformations':[('rotate',0.01),('scale paper',)],
              'title_str':r'$u+vw+w=0$'
              }
Nomographer(main_params)
```

6.10.2 Parameters for type 10

Axis parameters

Table 6.18: Specific axis parameters for type 10

parameter key	default value	type, explanation
'function'	–	**func(u).** Function in the equation for F_1 and $F_2 . For example “lambda u : u‘$
'function_3'	–	**func(u).** Function in the equation for F_3. For example `lambda u: u`
'function_4'	–	**func(u).** Function in the equation for F_4. For example `lambda u: u`
'u_min'	–	**Float.** Minimum value of function variable.
'u_max'	–	**Float.** Maximum value of function variable.

See *Common axis params* for other parameters.

Block parameters

Table 6.19: Specific block parameters for type 10

parameter	default value	explanation
'block_type'	'type_10'	**String.** This is type 10 block
'width'	10.0	**Float.** Block width (to be scaled)
'height'	10.0	**Float.** Block height (to be scaled)
'f1_params'	–	**Axis params Dict.** Axis params for function f1
'f2_params'	–	**Axis params Dict.** Axis params for function f2
'f3_params'	–	**Axis params Dict.** Axis params for function f3
'f4_params'	–	**Axis params Dict.** Axis params for function f4
'mirror_x'	False	**Boolean.** If x-axis is mirrored
'mirror_y'	False	**Boolean.** If y-axis is mirrored
'padding'	0.9	**Float.** How much axis extend w.r.t. width/height.
'float_axis'	'F1 or F2'	**Strings.** If given 'F1 or F2', then scaling is according to them, otherwise according to F3 and F4.
'reference_color'	color.rgb.black	**Color.** Color of reference lines.
'isopleth_values'	[[]]	** List of list of isopleth values.** Unknown values are given with strings, e.g. 'x'. An example:[[0.8,'x',0.7,0.5],[0.7,0.8,'x',0.3]]

General parameters

See *Main params* for top level main parameters.

CHAPTER SEVEN

BLOCK ALIGNMENT

Todo

Here discussion about block alignment, double alignment, tags, ...

CHAPTER EIGHT

TRANSFORMATIONS

Todo

Here discussion about transformations. Why and how they are made.

CHAPTER NINE

TOP LEVEL PARAMETERS

Main params define the top level properties of the nomograph. TODO: some explanations.

9.1 Main params

Table 9.1: General params

parameter	default value	explanation
'filename'	'pynomo_default.pdf'	**String.** Filename of generated file. .pdf and .eps formats supported.
'paper_height'	20.0	**String.** Height of paper (roughly, ticks and texts extend this).
'paper_width'	20.0	**String.** Width of paper (roughly, ticks and texts extend this).
'block_params'		**Array of Blocks.** List of blocks that make the nomograph.
'transformations'	[('rotate', 0.01), ('scale paper')]	**Array of tuples.** List of transformations to transform nomograph.
'title_str'	''	**String.** Title string of nomograph.
'title_x'	paper_width/2.0	**Float.** Title x-position.
'title_y'	paper_height	**Float.** Title y-position.
'title_box_width'	paper_width/2.2	**Float.** Title box width.
'title_color'	'color.rgb.black'	**Color.** Title color.
'make_grid'	False	**Boolean.** If True, draws grid to help position texts, etc.
'pre_func'	None	**func(context).** PyX function(canvas) to draw under nomograph. Function definition could be:
'post_func'	None	**func(context).** PyX function(canvas) to draw over nomograph. Definiton same as for 'pre_func'.
'debug'	False	**Boolean.** If True, prints dicts of definions.
'extra_texts'	[]	**List of Dicts defining texts.** Defines extra texts. Could be for example:
'isopleth_params'	[{}]	**List of Dicts.** Defines appearance of isopleths. Could be for example:

CHAPTER

TEN

EXAMPLES

In the following are listed examples to show nomographs possibilities. Also is explained the background for the cases and underlying math for the nomograph construction. Source code shows the implementation.

10.1 Example: Amortized loan calculator

10.1.1 Theory and background

This approach of constructing an amortized loan calculator is similar to one in Ref. [1]_

Equation for amortized loan [2]_ is:

$\frac{a}{A} = \frac{\frac{p}{100\times 12}}{1-\frac{1}{(1+\frac{p}{100\times 12})^{12n}}},$

where A is the amount of loan, a is monthly payment amount, p interest rate per year (monthly interest rate is taken as $p/12$) [3] and n is number of years for payment.

This equation of four variables is probably impossible to present with line and grid nomographs. For this reason a "Type 5" contour nomogram is constructed of the right hand side of the equation and left hand equation is just N-nomogram (Type 2). The two equations for nomogram construction are:

$x = \frac{a}{A}$

and

$x = \frac{\frac{p}{100\times 12}}{1-\frac{1}{(1+\frac{p}{100\times 12})^{12n}}}.$

In practice x is the x-coordinate of the canvas where nomogram is constructed.

Right hand side of equation

By defining coordinates x and y:

$x = \frac{\frac{p}{100\times 12}}{1-\frac{1}{(1+\frac{p}{100\times 12})^{12n}}},$

$y = 12n,$ we may solve y in terms of x and n:

[3] http://en.wikipedia.org/wiki/Annual_percentage_rate#Does_not_represent_the_total_cost_of_borrowing

$y = \frac{\log(\frac{x}{x-\frac{p}{100\times 12}})}{\log(1+\frac{p}{100\times 12})}$

The previous two equations are of correct form

$y = f_1(v)$

and

$y = f_2(x, u)$

for type 5 nomogram. For compressing time axis (y-axis), we transform $y \rightarrow \log y$ and find

$y = \log\left(\frac{\log(\frac{x}{x-\frac{p}{100\times 12}})}{\log(1+\frac{p}{100\times 12})}\right)$

$y = \log(12n)$.

Left hand side of equation

Left hand side of equation

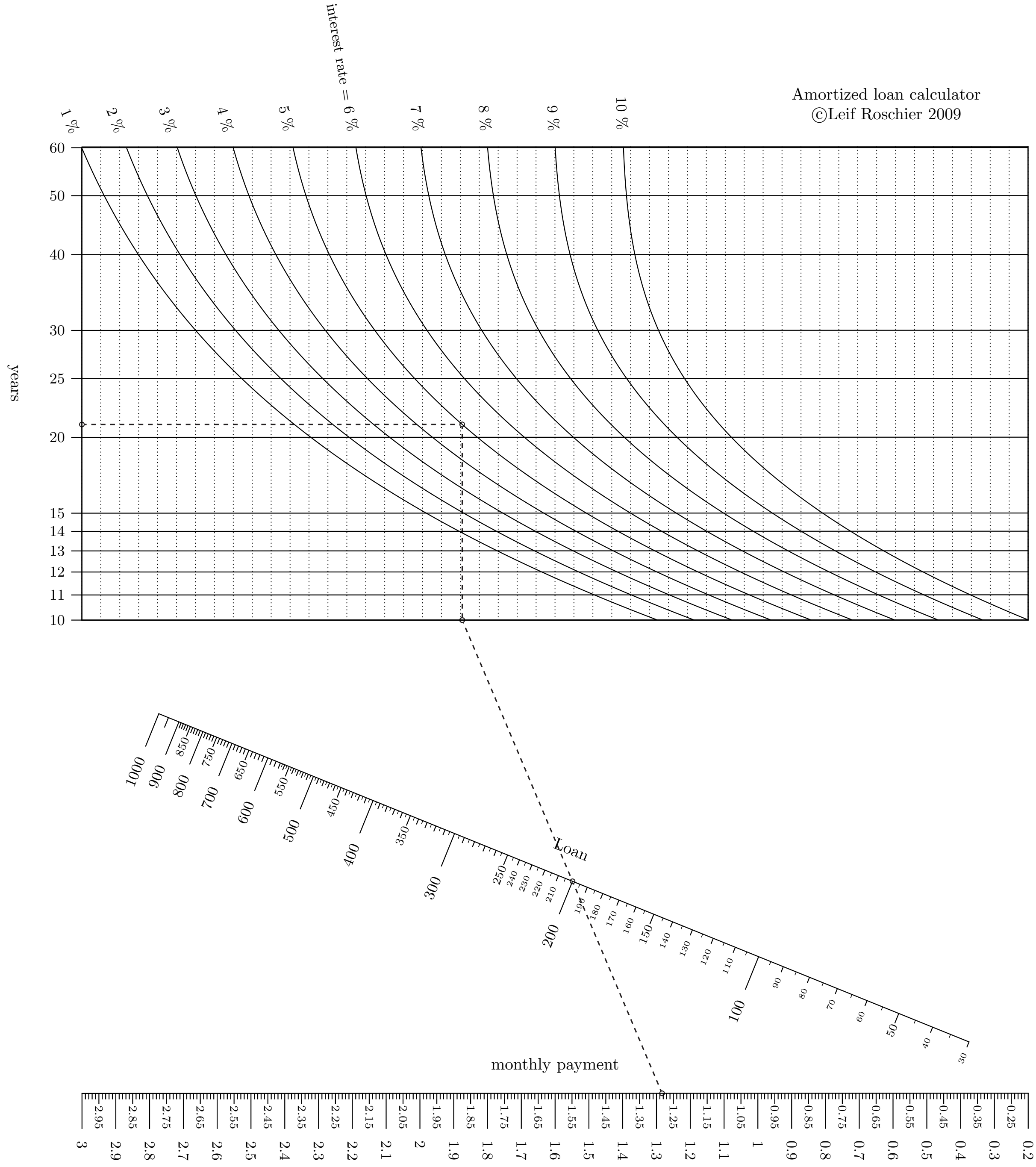
$x = \frac{a}{A}$

is just N-nomogram

$F_1(u_1) = F_2(u_2)F_3(u_3)$

References

10.1.2 Generated nomograph



10.1.3 Source code

```
"""
    ex_amortized_loan.py

    Amortized loan calculator
"""
import sys
sys.path.insert(0, "..")
from pynomo.nomographer import *

# Type 5 contour
```

```python
def f1(x,u):
    return log(log(x/(x-u/(100.0*12.0)))/log(1+u/(100.0*12.0)))

block_1_params={
             'width':10.0,
            'height':5.0,
            'block_type':'type_5',
            'u_func':lambda u:log(u*12.0),
            'v_func':f1,
            'u_values':[10.0,11.0,12.0,13.0,14.0,15.0,20.0,25.0,30.0,40.0,50.0,60.0],
            'v_values':[1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0],
            'wd_tag':'A',
            'u_title':'years',
            'v_title':r'interest rate = ',
            'u_text_format':r"$%3.0f$ ",
            'v_text_format':r"$%3.0f$ \%% ",
            'isopleth_values':[[21,5,'x']]
              }

# this is non-obvious trick to find bottom edge coordinates of the grid in order
# to align it with N nomogram
block1_dummy=Nomo_Block_Type_5(mirror_x=False)
block1_dummy.define_block(block_1_params)
block1_dummy.set_block()

# Let's define the N-nomogram
N_params_3={
        'u_min':block1_dummy.grid_box.params_wd['u_min'],
        'u_max':block1_dummy.grid_box.params_wd['u_max'],
        'function':lambda u:u,
        'title':'',
        'tag':'A',
        'tick_side':'right',
        'tick_levels':2,
        'tick_text_levels':2,
        'reference':False,
        'tick_levels':0,
        'tick_text_levels':0,
        'title_draw_center':True
                }
N_params_2={
        'u_min':30.0,
        'u_max':1000.0,
        'function':lambda u:u,
        'title':'Loan',
        'tag':'none',
        'tick_side':'left',
        'tick_levels':4,
        'tick_text_levels':3,
        'title_draw_center':True,
        #'text_format':r"$%3.0f$ ",
        'scale_type':'linear smart',
                }
N_params_1={
        'u_min':0.2,
        'u_max':3.0,
        'function':lambda u:u,
        'title':'monthly payment',
        'tag':'none',
        'tick_side':'right',
        'tick_levels':3,
        'tick_text_levels':2,
        'title_draw_center':True
                }

block_2_params={
              'block_type':'type_2',
              'width':10.0,
              'height':20.0,
              'f1_params':N_params_1,
              'f2_params':N_params_2,
              'f3_params':N_params_3,
```

```
              'isopleth_values':[['x',200,'x']]
              }

main_params={
               'filename':'amortized_loan.pdf',
               'paper_height':20.0,
               'paper_width':20.0,
               'block_params':[block_1_params,block_2_params],
               'transformations':[('rotate',0.01),('scale paper',)],
                 'title_str':r'Amortized loan calculator    \copyright    Leif Roschier  2009',
                 'title_x': 17,
                 'title_y': 21,
                 'title_box_width': 5
               }
Nomographer(main_params)
```

10.2 Example Photography exposure

10.2.1 Theory and background

This example illustrates how exposure in photography depends on factors: latitude, time of day, day of year, weather, composition. It relates these to camera settings: film speed (e.g. ISO 100), aperture and shutter speed. The mathematical approach and model is taken from book written by V. Setälä. **[1]_** This book illustrates the approach as nomographs but they are different compared with the one generatated here. Book uses shadow length, but we break shadow length into time, date and latitude via solar zenith angle.

The basic equation in Setälä (pp.492-494) can be extracted and written as

$$FS - L - A - W + C + T = 0 \tag{10.1}$$

where parameters of (10.1) are listed below:

FS	Film speed	DIN value that equals $10\log(S) + 1$,where S is ISO FILM speed
T	shutter time	$10\log\left(\frac{t}{1/10}\right)$
A	aperture	$10\log\left(\frac{N^2}{3.2^2}\right)$
L	shadow length (in steps)	two times (shadow length)/(person length) $= 2\arctan(\phi)$, where ϕ is solar zenith angle.
W	weather	Clear sky, Cumulus clouds: 0, Clear sky: 1, Sun through clouds: 3, Sky light gray: 6, Sky dark gray: 9, Thunder-clouds cover sky: 12
C	Composition	Person under trees: -6, Inside forest : -4, Person in shadow of wall : -1, Person at open place; alley under trees : 2, Buildings; street : 5, Landscape and front matter : 7, Open landscape : 9, Snow landscape and front matter; beach : 11,Snow field; open sea : 13, Clouds : 15

It is to be noted that Setälä has stops ten times base-10 logarithmic. Today we think stops in base-2 logarithmic.

Shadow lenght

Calculation of shadow length as a function of day of year, time of day and latitude is according to **[2]_**. Following equations are used. For fractional year (without time information) we take

$\gamma = (day - 1 + 0.5)2\pi/365.$

For time offset (eqtime) we use equation (in minutes)

$TO = 229.18(0.000075 + 0.001868\cos(\gamma) - 0.032077\sin(\gamma) - 0.014615\cos(2\gamma) - 0.040849\sin(2\gamma))$

to calculate that error is below 17 minutes for time axis. We assume that sun is at heightest point at noon and this is the error and approximation. We calculate stops in logarithmic scale and in this case we do not need very accurate equations for time. For declination we use equation

and for hour angle

$ha = (60h + \overline{TO})/4 - 180.$

Solar zenith angle (ϕ), latitude (LAT), declination (D) and hour angle (ha) are connected with equation:

$\cos(\phi) = \sin(LAT)sin(D) + \cos(LAT)\cos(D)\cos(ha).$

This is in our desired form as a function of hour (h), day (day), latitude (LAT), solar zenith angle (ϕ):

$\cos(\phi) = \sin(LAT)sin(D(\gamma(day))) + \cos(LAT)\cos(D(\gamma(day)))\cos(ha(h))$.

In practice illuminance of flat surface on earth depends on solar zenith angle as $\cos(\phi)$. Setälä uses shadow length that is easily measurable, but scales incorrectly, as value is proportional to $\tan(\phi)$. Also Setälä sums linear value with logarithmic ones as a practical approximation. To correct these assumptions, here we assume that values for shadow length 1 and 10 for Setälä are reasonable, and an equation that scales logarithmically is found:

$L = 0.33766 - 13.656\log 10(\cos(\phi))$

that gives $L = 1$ for $\phi = 26.565 = \arctan(1/2)$ and $L = 10$ for $\phi = 78.69 = \arctan(10/2).$

10.2.2 Construction of the nomograph

The presented equation is the following:

$$FS - \{0.33766 - 13.656\log_{10}[\sin(LAT)\sin(D(\gamma(day))) + \cos(LAT)\cos(D(\gamma(day)))\cos(ha(h))]\} \\ -A - W + C + T = 0.$$

In order to construct the nomograph, we split the equation into four blocks and an additional block to present values as EV100.

Table 10.1: Main equation split into blocks for the nomograph.

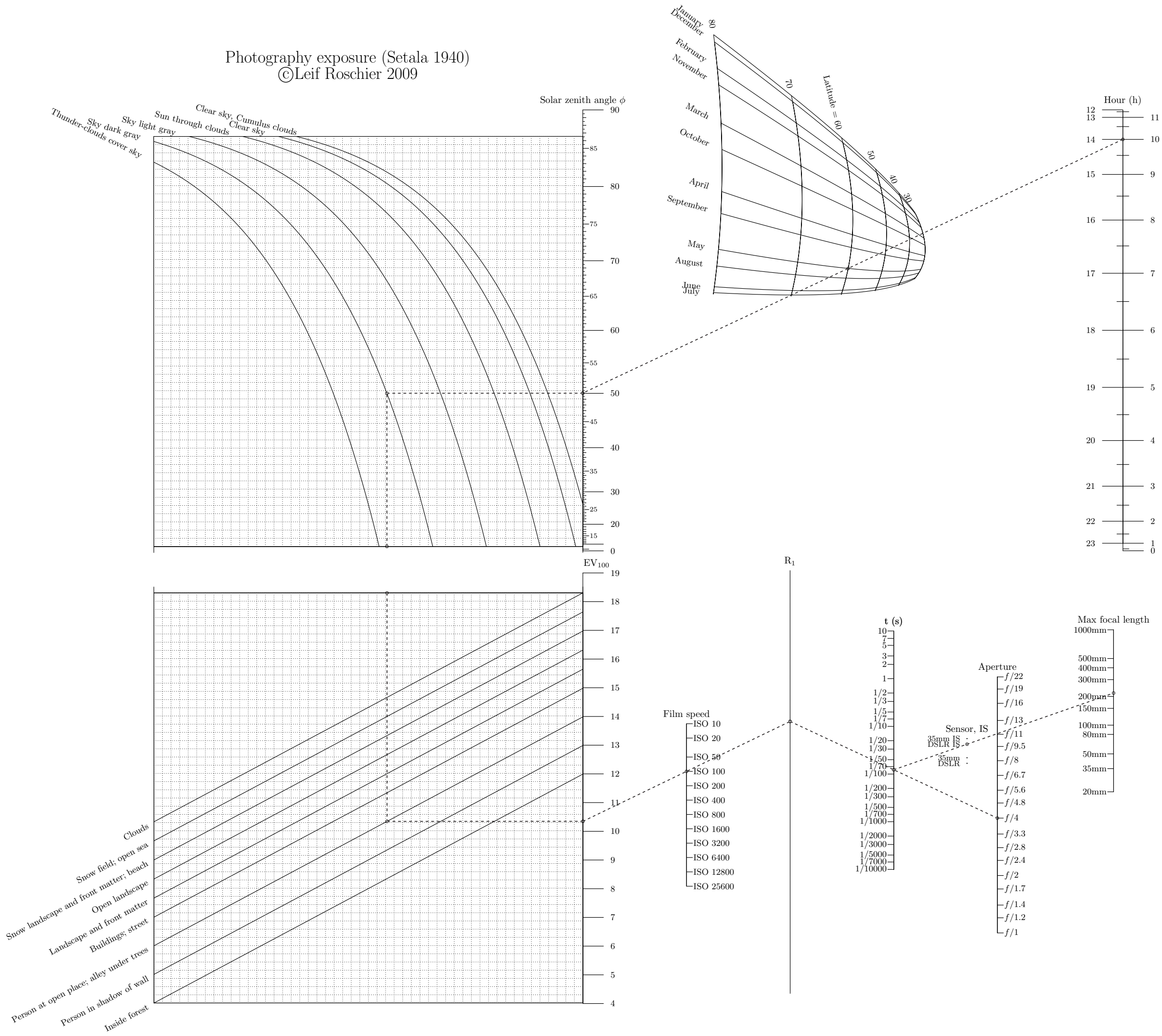
Explanation	Type
$$x_1 \equiv \cos(\phi) = \sin(LAT) sin(D(\gamma(day))) + \cos(LAT)\cos(D(\gamma(day)))\cos(ha(h))$$ formed into determinant: $$\begin{vmatrix} 0 & \cos(\phi) & 1 \\ \frac{\cos(LAT)\cos(D(\gamma(day)))}{1+(\cos(LAT)\cos(D(\gamma(day))))} & \frac{\sin(LAT)\sin(D(\gamma(day)))}{1+(\cos(LAT)\cos(D(\gamma(day))))} & 1 \\ 1 & -\cos(ha(h)) & 1 \end{vmatrix} = 0$$	Type 9
$$C_1 \equiv L + W = 0.006918 - 13.656\log_{10}(x_1) + W$$ split into two equations for contour construction: $$y_1 = C_1$$ $$y_1 = 0.006918 - 13.656\log_{10}(x_1) + W$$	Type 5
$$C_2 \equiv L + W + C = C_1 + C$$ split into two equations for contour construction: $$y_2 = C_2$$ $$y_2 = C_1 + C$$	Type 5
$$C_2 = FS - A + T$$ equals $$C_2 - (10\log_{10}(S) + 1.0) + 10\log_{10}\left(\frac{N^2}{3.2^2}\right) - 10\log_{10}\left(\frac{1/t_i}{1/10}\right) = 0,$$ where $$t_i \equiv 1/t$$ is inverse shutter time.	Type 3

Continued on next page

Table 10.1 – continued from previous page

Explanation	Type
Additional EV100 scale by using relation $$C_2 = (-EV_{100} + 13.654)/0.3322$$	Type 8
Maximum focal length calculator according to equation $$t_i/f = FL$$ written as $$-10\log_{10}\left(\frac{1/t_i}{1/10}\right) - 10\log_{10}\left(\frac{f}{10}\right) - 10\log_{10}(FL) = 0$$ in order to align correctly with previous equation. The values for the factor f are: DSLR (3/2), 35mm (1), DSLR image stabilization (3/8) and 35mm image stabilization (1/8).	Type 1

10.2.3 Generated nomograph



10.2.4 Source code

```
"""
    ex_photo_exposure.py

    Photgraph exposure.
"""
import sys
sys.path.insert(0, "..")
from pynomo.nomographer import *
"""
functions for solartime taken from solareqns.pdf from
http://www.srrb.noaa.gov/highlights/sunrise/solareqns.PDF
"""


# fractional year
def gamma(day):
    return 2 * pi / 365.0 * (day - 1 + 0.5)
# equation of time


def eq_time(day):
    gamma0 = gamma(day)
    return 229.18 * (0.000075 + 0.001868 * cos(gamma0) - 0.032077 * sin(gamma0)\
                     - 0.014615 * cos(2 * gamma0) - 0.040849 * sin(2 * gamma0))
```

```python

# mean correction, with constant correction we make less than 17 minutes  error
# in time axis
temp_a = arange(0, 365.0, 0.1)
temp_b = eq_time(temp_a)
correction = mean(temp_b) # this is 0.0171885 minutes


# declination
def eq_declination(day):
    g0 = gamma(day)
    return 0.006918 - 0.399912 * cos(g0) + 0.070257 * sin(g0) - 0.006758 * cos(2 * g0)\
            + 0.000907 * sin(2 * g0) - 0.002697 * cos(3 * g0) + 0.00148 * sin(3 * g0)


def f1(dummy):
    return 0.0


def g1(fii):
    return cos(fii*pi/180.0)


def f2(lat, day):
    dec = eq_declination(day)
    return (cos(lat * pi / 180.0) * cos(dec)) / (1.0 + (cos(lat * pi / 180.0) * cos(dec)))


def g2(lat, day):
    dec = eq_declination(day)  # in radians
    return (sin(lat * pi / 180.0) * sin(dec)) / (1.0 + (cos(lat * pi / 180.0) * cos(dec)))


def f3(dummy):
    return 1


def g3(h):
    hr = (h * 60.0 + correction) / 4.0 - 180.0
    return -1.0 * cos(hr * pi / 180.0)

days_in_month = (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
times1=[]
for idx in range(0, 12):
    times1.append(sum(days_in_month[0:idx])+1)

time_titles = ['January', 'February', 'March', 'April', 'May', 'June',
               'July', 'August', 'September', 'October', 'November', 'December']

phi_params = {'u_min': 0.0,
              'u_max': 90.0,
              'u_min_trafo': 0.0,
              'u_max_trafo': 90.0,
              'f': f1,
              'g': g1,
              'h': lambda u: 1.0,
              'title': r'Solar zenith angle $\phi$',
              'title_x_shift': 0.0,
              'title_y_shift': 0.25,
              'scale_type': 'linear smart',
              'tick_levels': 4,
              'tick_text_levels': 2,
              'tick_side': 'right',
              'tag': 'phi',
              'grid': False,
              }
time_params = {'u_min': 0.0,
               'u_max': 23.0,
               'u_min_trafo': 0.0,
               'u_max_trafo': 12.0,
               'f': f3,
               'g': g3,
               'h':lambda u: 1.0,
```

```
                'title': r'Hour (h)',
                'title_x_shift': 0.0,
                'title_y_shift': 0.25,
                'scale_type': 'linear',
                'tick_levels': 2,
                'tick_text_levels': 1,
                'tick_side': 'right',
                'tag': 'none',
                'grid': False,
                }
lat_day_params = {'ID': 'none',  # to identify the axis
                  'tag': 'none',  # for aligning block wrt others
                  'title': 'Grid',
                  'title_x_shift': 0.0,
                  'title_y_shift': 0.25,
                  'title_distance_center': 0.5,
                  'title_opposite_tick': True,
                  'u_min': 20.0,  # for alignment
                  'u_max': 80.0,  # for alignment
                  'f_grid': f2,
                  'g_grid': g2,
                  'h_grid': lambda u, v: 1.0,
                  'u_start': 30.0,
                  'u_stop': 80.0,
                  'v_start': times1[0],  # day
                  'v_stop': times1[-1],
                  'u_values': [30.0, 40.0, 50.0, 60.0, 70.0, 80.0],
                  'u_texts': ['30', '40', '50', 'Latitude = 60', '70', '80'],
                  'v_values': times1,
                  'v_texts': time_titles,
                  'grid': True,
                  'text_prefix_u': r'',
                  'text_prefix_v': r'',
                  'text_distance': 0.5,
                  'v_texts_u_start': False,
                  'v_texts_u_stop': True,
                  'u_texts_v_start': False,
                  'u_texts_v_stop': True,
                  }
block_params = {'block_type': 'type_9',
                'f1_params': phi_params,
                'f2_params': lat_day_params,
                'f3_params': time_params,
                'transform_ini': True,
                'isopleth_values': [['x', [60, times1[4]], 14.0]]
                }


# limiting functions are to avoid NaN in contour construction that uses optimization
def limit_xx(x):
    x1 = x
    return x1


def limit_x(x):
    x1 = x
    return x1

const_A = 0.33766
const_B = -13.656

block_params_weather = {'block_type': 'type_5',
                        'u_func': lambda u: u,
                        'v_func':lambda x, v: const_A + const_B * log10(limit_x(x)) + v,
                        'u_values': [1.0, 25.0],
                        'u_manual_axis_data': {1.0: '',
                                               25.0: ''},
                        'v_values': [0.0, 1.0, 3.0, 6.0, 9.0, 12.0],
                        'v_manual_axis_data': {0.0: ['Clear sky, Cumulus clouds',
                                                     {'x_corr': 0.5,
                                                      'y_corr': 0.0,
                                                      'draw_line': False}],
```

```
                                               1.0: 'Clear sky',
                                               3.0: 'Sun through clouds',
                                               6.0: 'Sky light gray',
                                               9.0: 'Sky dark gray',
                                               12.0: 'Thunder-clouds cover sky'},
                        'v_text_distance': 0.5,
                        'wd_tick_levels': 0,
                        'wd_tick_text_levels': 0,
                        'wd_tick_side': 'right',
                        'wd_title': '',
                        'manual_x_scale': True,
                        'x_min': 0.06,
                        'x_max': 0.99,
                        'u_title': '',
                        'v_title': '',
                        'wd_title_opposite_tick': True,
                        'wd_title_distance_center': 2.5,
                        'wd_align_func': lambda L: acos(limit_xx(10.0**((L - const_A) / const_B))) * 180.0 / pi,  # phi as L
                        'wd_func': lambda L: 10.0**((L - const_A) / const_B),  # x as L
                        'wd_func_inv': lambda x: const_A+const_B * log10(x),  # L as x
                        'wd_tag': 'phi',
                        'mirror_y': True,
                        'mirror_x': False,
                        'width': 10.0,
                        'height': 10.0,
                        'u_scale_opposite': True,
                        'u_tag': 'AA',
                        'horizontal_guides': True,
                        'isopleth_values': [['x', 9.0, 'x']],
                        }
block_params_scene = {'block_type': 'type_5',
                      'u_func': lambda u: u,
                      'v_func': lambda x, v: x + v,
                      'u_values': [1.0, 25.0],
                      'u_manual_axis_data': {1.0: '',
                                             25.0: ''},
                      'u_tag': 'AA',
                      'wd_tag': 'EV',
                      'v_values': [-4.0, -1.0, 2.0, 5.0, 7.0, 9.0, 11.0, 13.0, 15.0],
                      'v_manual_axis_data': {-6.0: 'Person under trees',
                                             -4.0: 'Inside forest',
                                             -1.0: 'Person in shadow of wall',
                                             2.0: 'Person at open place; alley under trees',
                                             5.0: 'Buildings; street',
                                             7.0: 'Landscape and front matter',
                                             9.0: 'Open landscape',
                                             11.0: 'Snow landscape and front matter; beach',
                                             13.0: 'Snow field; open sea',
                                             15.0: 'Clouds',
                                             },
                      'wd_tick_levels': 0,
                      'wd_tick_text_levels': 0,
                      'wd_tick_side': 'right',
                      'wd_title': '',
                      'u_title': '',
                      'v_title': '',
                      'wd_title_opposite_tick': True,
                      'wd_title_distance_center': 2.5,
                      'mirror_x': True,
                      'horizontal_guides': True,
                      'u_align_y_offset': -0.9,
                      'isopleth_values': [['x', 2.0, 'x']],
                      }
camera_params_1 = {'u_min': -10.0,
                   'u_max': 15.0,
                   'function': lambda u: u,
                   'title': r'',
                   'tick_levels': 0,
                   'tick_text_levels': 0,
                   'tag': 'EV',
                   }
camera_params_2 = {'u_min': 10.0,
```

```
                    'u_max': 25600.0,
                    'function': lambda S: -(10 * log10(S) + 1.0),
                    'title': r'Film speed',
                    'manual_axis_data': {10.0: 'ISO 10',
                                         20.0: 'ISO 20',
                                         50.0: 'ISO 50',
                                         100.0: 'ISO 100',
                                         200.0: 'ISO 200',
                                         400.0: 'ISO 400',
                                         800.0: 'ISO 800',
                                         1600.0: 'ISO 1600',
                                         3200.0: 'ISO 3200',
                                         6400.0: 'ISO 6400',
                                         12800.0: 'ISO 12800',
                                         25600.0: 'ISO 25600',
                                         },
                    'scale_type': 'manual line'
                    }
camera_params_3 = {'u_min': 0.1,
                    'u_max': 10000.0,
                    'function': lambda t: -10 * log10((1.0 / t) / (1.0 / 10.0)) - 30,
                    'manual_axis_data': {1/10.0: '10',
                                         1/7.0: '7',
                                         1/5.0: '5',
                                         1/3.0: '3',
                                         1/2.0: '2',
                                         1.0: '1',
                                         2.0: '1/2',
                                         3.0: '1/3',
                                         5.0: '1/5',
                                         7.0: '1/7',
                                         10.0: '1/10',
                                         20.0: '1/20',
                                         30.0: '1/30',
                                         50.0: '1/50',
                                         70.0: '1/70',
                                         100.0: '1/100',
                                         200.0: '1/200',
                                         300.0: '1/300',
                                         500.0: '1/500',
                                         700.0: '1/700',
                                         1000.0: '1/1000',
                                         2000.0: '1/2000',
                                         3000.0: '1/3000',
                                         5000.0: '1/5000',
                                         7000.0: '1/7000',
                                         10000.0: '1/10000',
                              },
                    'scale_type': 'manual line',
                    'title': r't (s)',
                    'text_format': r"1/%3.0f s",
                    'tag': 'shutter',
                    'tick_side': 'left',
                    }
camera_params_4 = {'u_min': 1.0,
                    'u_max': 22.0,
                    'function': lambda N: 10 * log10((N / 3.2)**2) + 30,
                    'manual_axis_data': {1.0: '$f$/1',
                                         1.2: '$f$/1.2',
                                         1.4: '$f$/1.4',
                                         1.7: '$f$/1.7',
                                         2.0: '$f$/2',
                                         2.4: '$f$/2.4',
                                         2.8: '$f$/2.8',
                                         3.3: '$f$/3.3',
                                         4.0: '$f$/4',
                                         4.8: '$f$/4.8',
                                         5.6: '$f$/5.6',
                                         6.7: '$f$/6.7',
                                         8.0: '$f$/8',
                                         9.5: '$f$/9.5',
                                         11.0 :'$f$/11',
```

```
                                        13.0 :'$f$/13',
                                        16.0 :'$f$/16',
                                        19.0 :'$f$/19',
                                        22.0 :'$f$/22',
                                        },
                    'scale_type': 'manual line',
                    'title': r'Aperture',
                    }
block_params_camera = {'block_type': 'type_3',
                       'width': 10.0,
                       'height': 10.0,
                       'f_params': [camera_params_1, camera_params_2, camera_params_3,
                                    camera_params_4],
                       'mirror_x': True,
                       'isopleth_values': [['x', 100.0, 'x', 4.0]],
                       }


def old_EV(EV):  # C2(EV100) in wiki
    return (-EV + 13.654) / 0.3322

EV_para = {'tag': 'EV',
           'u_min': 4.0,
           'u_max': 19.0,
           'function': lambda u: old_EV(u),
           'title': r'EV$_{100}$',
           'tick_levels': 1,
           'tick_text_levels': 1,
           'align_func': old_EV,
           'title_x_shift': 0.5,
           'tick_side': 'right',
           }
EV_block = {'block_type': 'type_8',
            'f_params': EV_para,
            'isopleth_values': [['x']],
            }
# maximum focal length
FL_t_para={'u_min': 0.1,
           'u_max': 10000.0,
           'function': lambda t:-10 * log10((1.0 / t) / (1.0 / 10.0)) - 30,
           'scale_type': 'linear',
           'tick_levels': 0,
           'tick_text_levels': 0,
           'title': r't (s)',
           'text_format': r"1/%3.0f s",
           'tag': 'shutter',
           }
FL_factor_params_2 = {'u_min': 1.0/4.0,
                      'u_max': 3.0/2.0,
                      'function': lambda factor: -10 * log10(factor / 10.0) + 0,
                      'title': r'Sensor, IS',
                      'scale_type': 'manual point',
                      'manual_axis_data': {1.0/(2.0/3.0): 'DSLR',
                                           1.0/(1.0): '35mm',
                                           1.0/(8.0/3.0): 'DSLR IS',
                                           1.0/(4.0): '35mm IS',
                      },
                      'tick_side':'left',
                      'text_size_manual': text.size.footnotesize,  # pyx directive
                      }
FL_fl_params = {'u_min': 20.0,
                'u_max': 1000.0,
                'function': lambda FL:-10 * log10(FL) + 30,
                'title': r'Max focal length',
                'tick_levels': 3,
                'tick_text_levels': 2,
                'tick_side': 'left',
                'scale_type': 'manual line',
                'manual_axis_data': {20.0: '20mm',
                                     35.0: '35mm',
                                     50.0: '50mm',
                                     80.0: '80mm',
```

```
                                    100.0: '100mm',
                                    150.0: '150mm',
                                    200.0: '200mm',
                                    300.0: '300mm',
                                    400.0: '400mm',
                                    500.0: '500mm',
                                    1000.0: '1000mm'}
                }

FL_block_params = {'block_type': 'type_1',
                   'width': 12.0,
                   'height': 10.0,
                   'f1_params': FL_t_para,
                   'f2_params': FL_factor_params_2,
                   'f3_params': FL_fl_params,
                   'mirror_x': True,
                   'proportion': 0.5,
                   'isopleth_values': [['x', 1.0/(8.0/3.0), 'x']],
                   }

main_params = {'filename': ['ex_photo_exposure.pdf', 'ex_photo_exposure.eps'],
               'paper_height': 35.0,
               'paper_width': 35.0,
               'block_params': [block_params, block_params_weather, block_params_scene,
                                block_params_camera, EV_block, FL_block_params],
               'transformations': [('rotate', 0.01), ('scale paper',)],
               'title_x': 7,
               'title_y': 34,
               'title_box_width': 10,
               'title_str': r'\LARGE Photography exposure (Setala 1940) \par \copyright Leif Roschier  2009 '
              }
Nomographer(main_params)
```

CHAPTER

ELEVEN

LITERATURE

11.1 List of relevant books

Todo

Here links to literature...

11.2 Sources in the web

Todo

Here links to web resources...

11.3 Scientific articles

Todo

Here links to peer-reviewed scientific articles related to nomography and some discussion why the link.

CHAPTER

TWELVE

APPENDIX

12.1 Comparison of Nomogram to Computer Application

Characteristic	Computer	Nomogram
Hardware Requirements	Computer, specialized calculator, or smartphone	Any straightedge, pencil
Software Requirements	Application encapsulating the relevant relationships	Graphical representation of the relevant relationships
Infrastructure Requirements	Computing resources, perhaps Internet access; if smartphone, appropriate app	Ambient light
Energy Needs	Electrical outlet or batteries	Ambient light
Learning Curve	Knowing what to punch in, plus learning curve for software, hardware, and infrastructure	Knowing how to connect two points, and how to interpolate a point on a scale
Documentation	What documentation? Where?	Self-documenting
Tool Distribution	Likely Internet access	Single sheet of paper
Results Distribution	Need printer or Internet connection	Hand-carry or fax document
Cost	Variable	Cost of duplicating and transmitting a single page
Accuracy (decimal places)	As many as you want… if the software provides them	As many as you need… given the precision of the input
Speed: As fast as…	…your hardware	…you can draw a straight line
Sensitivity Analysis	Repeated data sets	Examination of graphic
Implicit Solution	Usually difficult or impossible	Automatic
Common Failure Mode	Punch in wrong numbers	Can't find glasses
Need to Calculate	None	None
Third World Use	Problematic depending on computing accessibility	Works so long as pencil and paper are available
GIGO Susceptibility	High; May be hard to detect	Garbage In is clearly documented
Permanence	Need a printer	Creates a written record as part of usage pattern

Continued on next page

Table 12.1 – continued from previous page

Characteristic	Computer	Nomogram
Trust Factor	Did the programmer get it right?	Did the nomographer get it right?
Communication	Single number output	Graphical interactivity
Pizzazz Factor	High: Very modern	Low: Old-fashioned slide-rule-like technology

CHAPTER THIRTEEN

LICENSE

PyNomo is open source and free software licensed under the GNU GPL Version 3.