PyX – A Graphics Package for PostScript Creation

André Wobst

5. March 2004

DANTE 2004

Darmstadt

Index

anciently: PyX := Postscript + Python + TEX

nowadays: PyX – python graphics package

- Motivation
- Short introduction to Python
- Paths as mathematical objects
- Output of paths: decorators
- TEX-interface for the creation of text
- Complex graphical tasks (x-y-graphs etc.)
- Evolution of PyX

Motivation

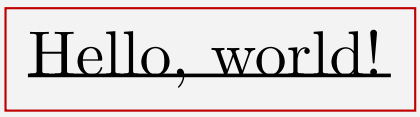
- Programmable graphical system
- Creation of freely scaleable vector graphics
- Extensive elementary graphical capabilities
- Support for complex graphical tasks
- Based on a suitable existing language
  (both for PyX and for the user)
- TEX for typesetting (integration as good as possible)
- Take other solutions into account

Hello, world!

Example `hello.py`:

```
from pyx import *
c = canvas.canvas()
c.text(0, 0, "Hello, world!")
c.stroke(path.line(0, 0, 2, 0))
c.writetofile("hello")
```

Execution of "`python hello.py`" creates "`hello.eps`":



Python 1

- Interpreted language (with byte code)
- Few characters with special meaning (good readablility)
- strongly typed, but dynamically during execution
- Numbers: `int`, `long` (infinite precision), `float` (8 bytes), `complex`
- Strings: `s = "Hello, world!"` → `print s, s[0]`
- Lists: `l = [1, 2, 3]` → `print l[0], l[1:2]`
- Dictionaries: `d = {"a": 1, "b": 2}` → `print d["a"]`
- Program blocks by indentation

  "We will perhaps eventually be writing only small modules which are identified by name as they are used to build larger ones, so that devices like indentation, rather than delimiters, might become feasible for expressing local structure in the source language."

  – Donald E. Knuth, "Structured Programming with goto Statements" Computing Surveys **6**, 261 (1974)

wobsta.de

Python 2

- `if-else-elif`-construct (logic also with integers):

```
if i or j:
    print "i or j not equal zero"
else:
    print "i and j are both zero"
```

- `for`-loop (range-function creates a sequence of integers):

```
for i in range(10):
    print i
```

- `while`-loop (len-function calculates length of a sequence):

```
while len(l):
    print l[0]
    l = l[1:]
```

- `for`- and `while`-loops support `continue`, `break` and `else`
- Exception handling by `try-except`-blocks

wobsta.de

Python 3

- Functions:
```
def faculty(n):
    if not n:
        return 1
    return n*faculty(n-1)
```
- Named and predefined arguments (like `keyval.sty`):
```
def f(a, b="default", c=[1, 2, 3]): ...
```
- Objects:
```
class A: # inheritance via class B(A): ...
    def show(self, a, b, c): ...
```
- Special methods: `__init__(self, ...)`, `__add__(self, other)`
- Instances:
```
a = A()
a.show(1, 2, 3) # identical to A.show(a, 1, 2, 3)
```


RyX

- RyX is GPL
- Available at http://pyx.sourceforge.net/, current version 0.5.1
- Python package (runs at Python 2.0 and above)
  Python uses PSF Licence (no copyleft, GPL compatible)
  Python is available on a huge variety of platforms
- Uses TeX (or LaTeX) for typesetting
  based on dvi-Output and Type1-fonts
- `from pyx import *` loads the following modules (among others):

`path, trafo`	paths, linear transformations
`canvas, deco`	drawing area, decorators of paths
`style, color`	attributes for the decoration of paths
`text`	TeX-interface
`box, connector, graph`	complex graphical tasks

Coordinate system/units

- Origin is bottom left
- standard unit: 1 cm
- context-sensitive scaling:
  - unscaleable – `t` (for true)
  - regular coordinates – `u` (for user)
  - distances, symbol sizes – `v` (for visual)
  - line width – `w` (for width)
  - (`x` for TEX might be added in the future)
- Automatic rectangular boundary (bounding box)
  (does not take into account the line width)
- Optional centering the output on a paper format while
  keeping the output useable in LATEX and others

Paths

- mathematical objects (no withs)
- Composed out of path elements
- Path elements: `moveto`, `lineto`, `curveto`, `arc`, `closepath`, `...`
- Creation of paths by classes of the path module:

```
p = path(moveto(0, 0), lineto(1, 0))
p.append(lineto(1, 1))
p2 = line(0, 0, 1, 0) # also rect, curve, circle
```

- Operations on (norm-)paths:

`arclength`	path lengths
`lentopar`	parameter as a function of the length
`intersect`	intersection of paths (parameter)
`at`	position at the path (parameter)
`split`	path segments (parameter)
`transformed`	transformed path

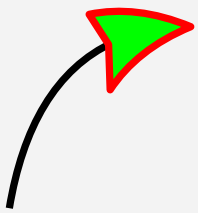
Decorators

- Combine paths and stroke attributes (line attributes, colors, . . .)
- Might modify paths and/or add features
- Canvas method `draw` applies decorators:
  `c.draw(p, [deco.stroked, deco.earrow.normal])`
- Stroke- and fill-decorators available as canvas methods:
  `c.stroke(p, [deco.earrow.normal])`
  `c.fill(path.rect(0, 0, 5, 5))`
- Draw attributes can be applied to `draw`, `stroke`, and `fill`
  or can be passed to decorators:

```
c.stroke(p, [color.rgb.blue, style.linestyle.dashed])
c.draw(p, [deco.filled([color.rgb.red]),
           deco.stroked([color.rgb.green])])
```

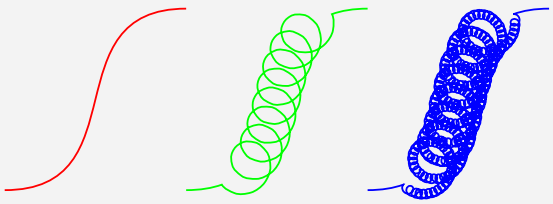

Example: arrow

```
from pyx import *
c=canvas.canvas()
p=path.curve(0, 0, 0.05, 0.3, 0.2, 0.5, 0.5, 0.5)
a=deco.earrow.Large([deco.filled([color.rgb.green]),
    deco.stroked([color.rgb.red, style.linejoin.round])])
c.stroke(p, [a])
c.writetofile("arrow")
```



Example: spring

```
from pyx06pre import *
c=canvas.canvas()
p = path.curve(0, 0, 1.5, 0, 0.5, 2, 2, 2)
c.stroke(p, [color.rgb.red])
c.stroke(p.transformed(trafo.translate(2, 0)),
         [color.rgb.green, deco.wriggle()])
c.stroke(p.transformed(trafo.translate(4, 0)),
         [color.rgb.blue, deco.wriggle(),
          deco.wriggle(loops=250, radius=0.05)])
c.writeEPSfile("wriggle")
```



Creation of Text

- TEX or LATEX is started as a separate process and monitored (several instances possible, but usually not necessary)
- Text to be pocessed is put into a box; box extents are returned using stdout; box contents is written to a separate DVI page
- DVI is read when needed (use `--ipc` option or finish TEX)
- RYX creates canvas instances out of the DVI pages
- Exculsive use of Type1 fonts (using `psfonts.map` and so on)
- Virtual font support
- Font reduction of used glyphs (by C extension module)
- Support of well defined `\special`-commands; pyx.def for `graphic[s/x]` and `color`-package
- Markers for positions in the text (by `\special`)

TEX details 1

```
from pyx import *
text.set(texdebug="text.tex")
c = canvas.canvas()
c.text(0, 0, "Hello, world!")
c.writetofile("text")
```

- text module defines a class `texrunner` together with an instance `defaulttexrunner` thereof
- `text.set` for constructor paramters (LATEX-class etc.)
- preamble mode: `text.preamble(r"\usepackage{graphicx}")`
- `c.text(...)` is identical to:
  `c.insert(text.text(...)) # or even more precise`
  `c.insert(text.defaulttexrunner.text(...))`
- Example above creates runable file `text.tex`

TEX details 2

```
This is TeX, Version 3.14159 (Web2C 7.4.5)
(./text.tex
! Undefined control sequence.
l.5 \raiseerror
                %
PyXInputMarker:executeid=1:
PyXInputMarker:executeid=2:
PyXInputMarker:executeid=3:
PyXInputMarker:executeid=4:
PyXBox:page=1,lt=0.0pt,rt=55.58344pt,ht=6.94444pt,dp=1.94444pt:
[80.121.88.1]
PyXInputMarker:executeid=5:
 )
(see the transcript file for additional information)
Output written on text.dvi (1 page, 192 bytes).
Transcript written on text.log.
```

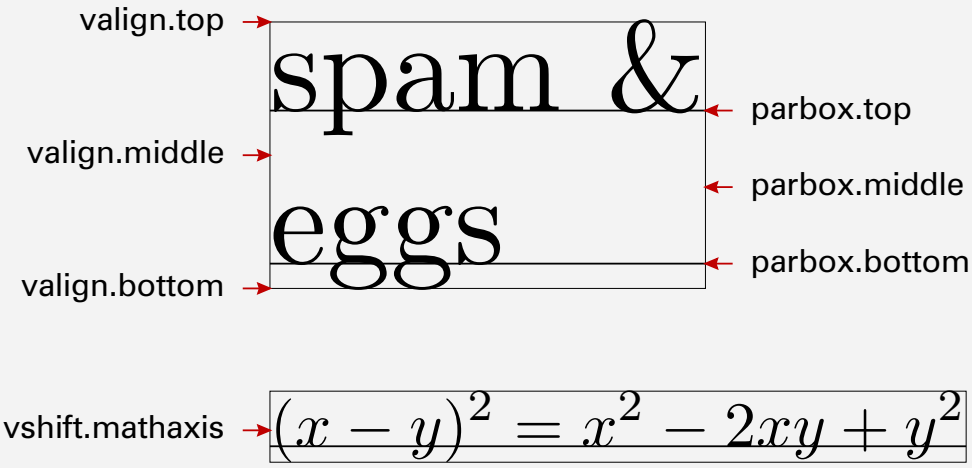
TEX details 3

```
% ...
\newbox\PyXBox%
\long\def\ProcessPyXBox#1#2{%
\setbox\PyXBox=\hbox{{#1}}%
\immediate\write16{PyXBox:page=#2,...,ht=\the\ht\PyXBox,...:}%
% ...
\ht\PyXBoxOpt%
{\count0=80\count1=121\count2=88\count3=#2\shipout\box\PyXBox}}%
\def\PyXInput#1{\immediate\write16{PyXInputMarker:executeid=#1:}}%
\PyXInput{1}%
% ...
\ProcessPyXBox{Hello, world!%
}{1}%
\PyXInput{5}%
\end%
```


Text attributes

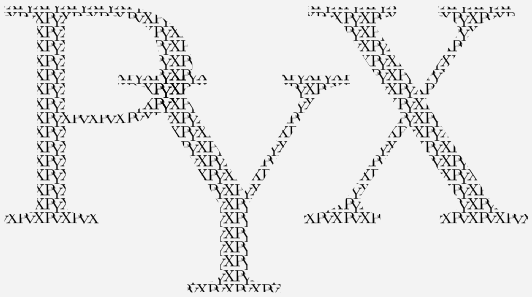
- Horizontal alignment: `text.halign.[left|center|right]`
- Vertical alignment: `text.valign.[baseline|top|middle|bottom]`
- Vertical box: `text.parbox(<Breite>)`; additional parameter `baseline` with variants `top`, `middle`, and `bottom`
- Vertical shift by `text.vshift` using a percentage of a text height or the mathematical axis (`text.vshift.mathaxis`)
- Font size `text.size.large` etc.
- Mathematical mode by `text.mathmode`
- Any personal TEX/LATEX-construct
- Additional fill attributes and transformations

Example: text alignment



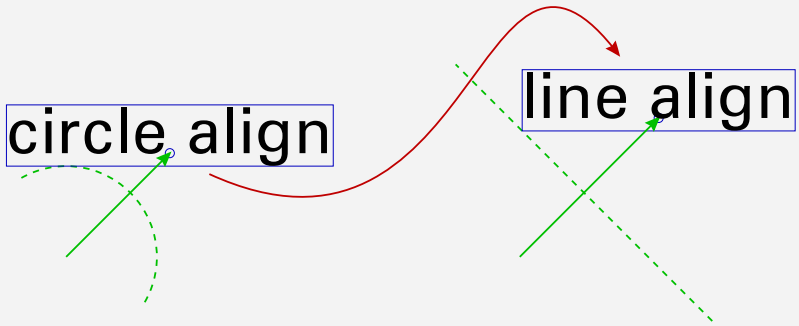
Example: text with pattern

```
from pyx import *
p = canvas.pattern()
p.text(0, 0, r"\PyX", [trafo.scale(0.5)])
c = canvas.canvas()
c.text(0, 0, r"\PyX", [trafo.scale(10), p])
c.writetofile("pattern")
```



Boxes and connectors

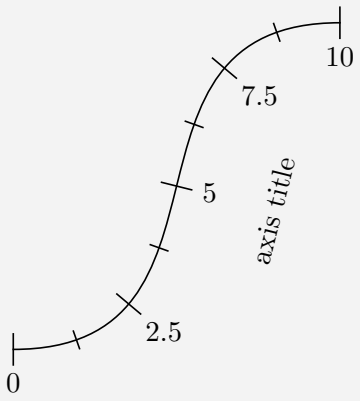
- Boxes are drawing areas with boundary (currently only convex polygons; clipping by boxes in the future, which is currently a canvas attribute)
- Box center for alignment and connectors



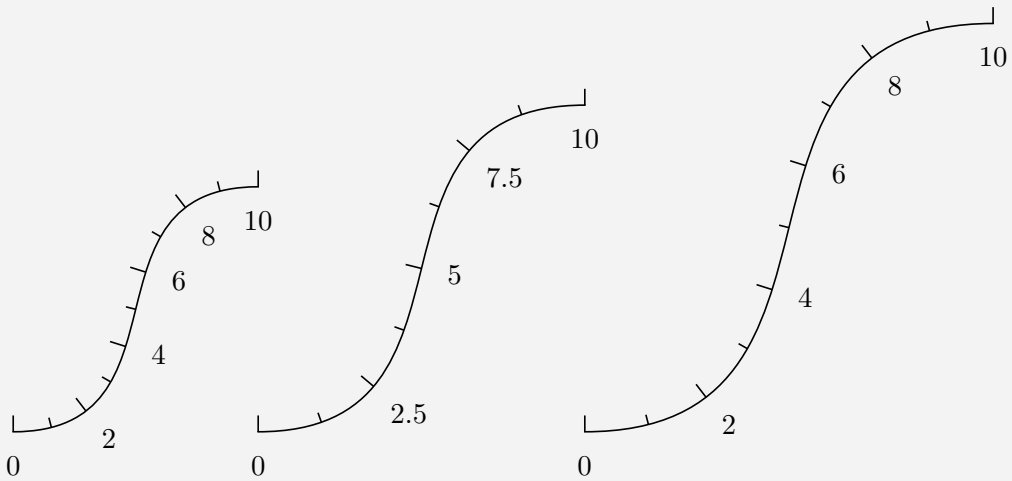
Axes

- Elementary component of graphs
- Automatic axes partitioning taking into account number of ticks and distances (extents) of labels
- Mixing of manual set axis ticks and automatically created axis partitioning
- Axis partitioning by rational number arithmetics
- Flexible axis labeling
- Axes with break points, axes for bar graphs
- Along arbitrary paths
- extendable to true time axes (experimental)
- Sources of the following examples at http://pyx.sourceforge.net/

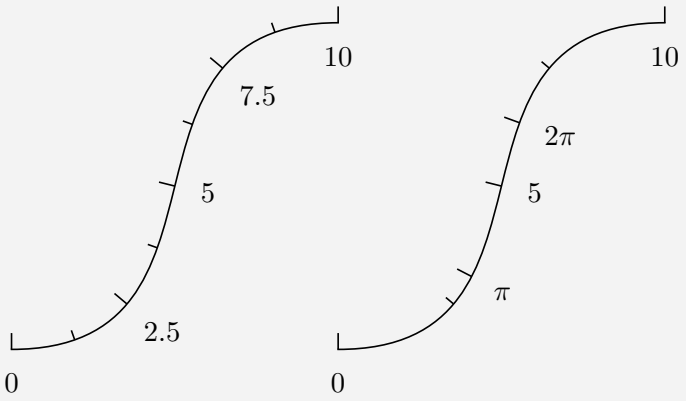
Example: path with axis



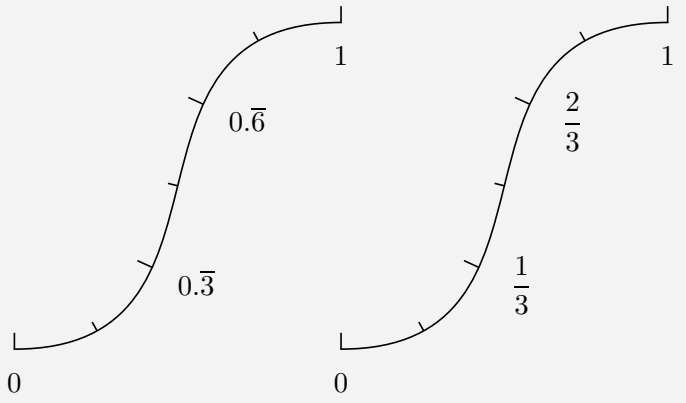
Example: axis rating



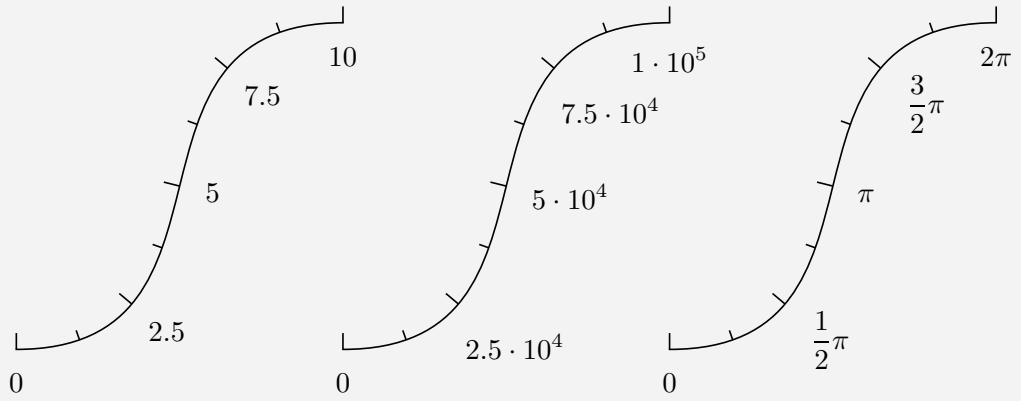
Example: manual ticks



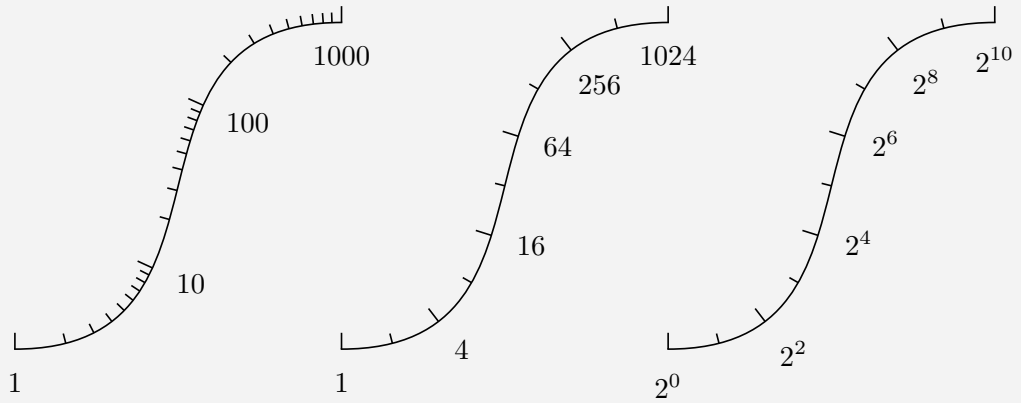
Example: rational arithmetic



Example: axis labels



Example: logarithmic axes

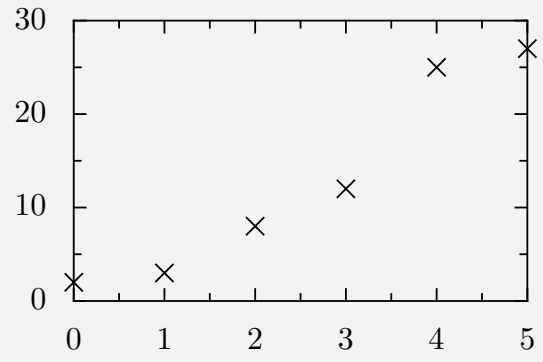


X-Y-Graphs

- Composed out of components, which are mostly independend of specific graph geometries
- Arbitrary number of axes; linkable axes even between graphs
- Data from files, from a function of a parametric function
- Styles: symbols, lines, texts, arrows with size and direction, error bars, colored rectangles, bars and a variety of combinations thereof
- Different data with different styles within a graph
- Data keys

Example: simple graph

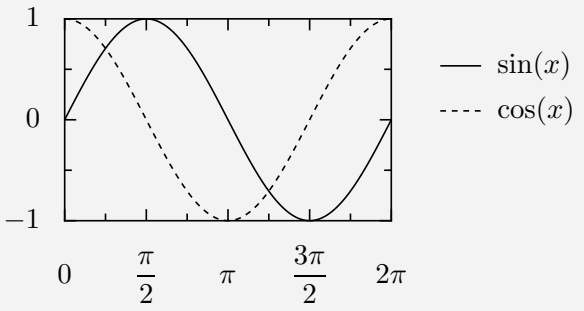
```
from pyx06pre import *
g = graph.type.graphxy(width=5)
g.plot(graph.data.data("minimal.dat", x=1, y=2))
g.writeEPSfile("minimal")
```



RyX – ein Graphikpaket zur PostScript-Erzeugung
André Wobst, 5. März 2004, DANTE 2004, Darmstadt

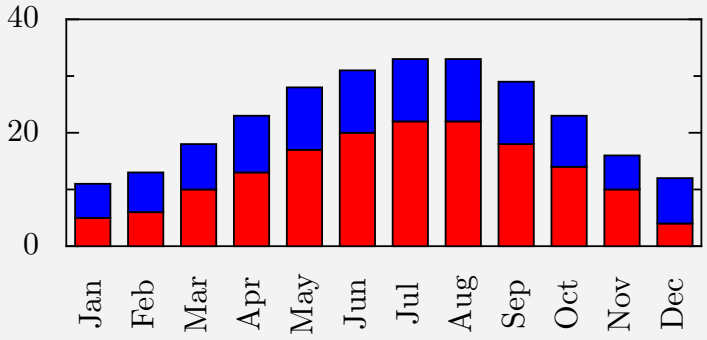
Example: function graph

```
from math import pi
from pyx06pre import *
g=graph.type.graphxy(width=4, key=graph.key.key(hinside=0),
    x=graph.axis.linaxis(min=0, max=2*pi, divisor=pi,
        texter=graph.texter.rationaltexter(enumsuffix=r"\pi")))
g.plot(graph.data.function("y=sin(x)", title=r"$\sin(x)$"))
g.plot(graph.data.function("y=cos(x)", title=r"$\cos(x)$"))
g.writeEPSfile("piaxis")
```



Example: bar graph

```
from pyx06pre import *
p=graph.painter.baraxispainter(nameattrs=[trafo.rotate(90)])
g=graph.type.graphxy(width=7, height=2.5,
                     x=graph.axis.baraxis(painter=p))
g.plot(graph.data.data("bar.dat", xname=1, y=2, ystack1=3),
       graph.style.bar())
g.writeEPSfile("bar")
```



Evolution

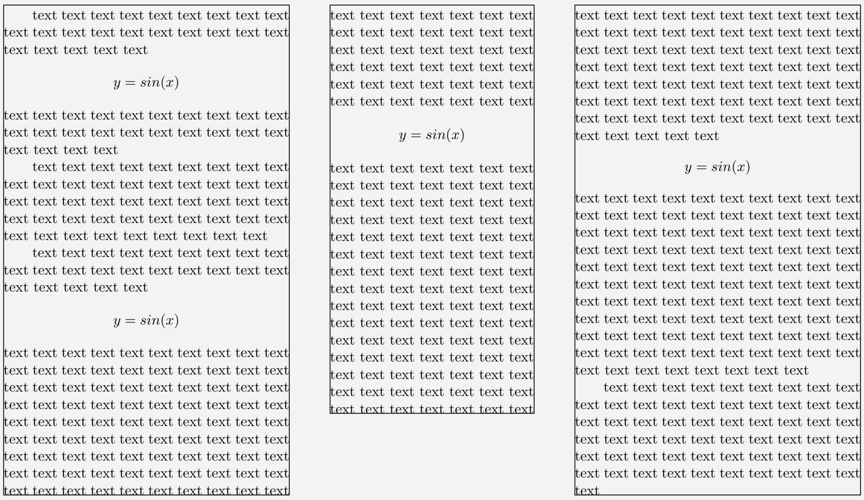
- Initiated and base development: Jörg Lehmann, André Wobst
- Other current developers: Michael Schindler, Gert-Ludwig Ingold
- Project start (CVS) at 01.09.2000
- SourceForge.net project since 30.01.2002
- First release 07.10.2002 (version 0.1)
- Current version 0.5.1 from 22.01.2004
- Currently and in close future alpha releases
- Forthcoming version 0.6 in (hopefully) a few weeks

Goals

- Stabilizing the interfaces
- Development on self-evident functionality:
  for example graph geometries, boxes, decorators
- Development on less obvious functionality:
  for example filling text in boxes (first experiments in CVS)
- Additional output media:
  PDF (first experiments in CVS) and SVG
- Documentation, examples, tests

RYX – ein Graphikpaket zur PostScript-Erzeugung
André Wobst, 5. März 2004, DANTE 2004, Darmstadt

More than TeX …



Questions

Questions?

contact@wobsta.de
http://www.wobsta.de/