http://pyx.sourceforge.net/

PyX 0.6.3
Examples

Jörg Lehmann <joergl@users.sourceforge.net>
André Wobst <wobsta@users.sourceforge.net>

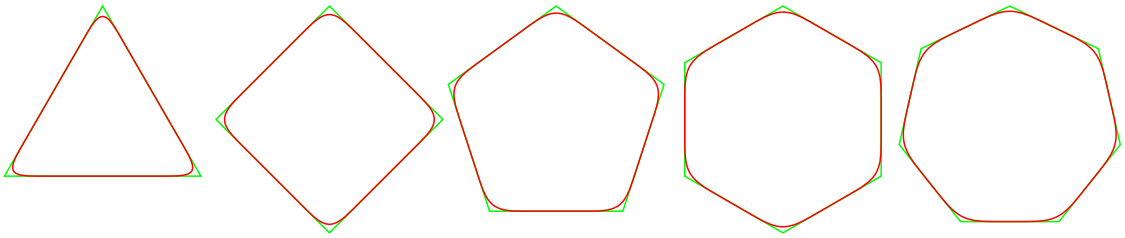
April 27, 2004

Abstract

This is an automatically generated document from the PyX examples directory. For each PyX example file its name, source code and the corresponding postscript output are shown. Please pay attention to source code comments within the examples for further information.

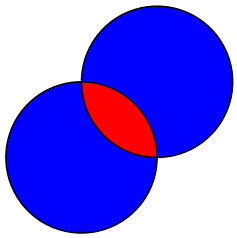
box

```
from math import sin, cos, pi
from pyx import *

r = 1.5

# create a box list of regular polygons
boxes = [box.polygon([(-r*sin(i*2*pi/n), r*cos(i*2*pi/n))
                      for i in range(n)])
         for n in range(3, 8)]
# tile with spacing 0 horizontally
box.tile(boxes, 0, 1, 0)

c = canvas.canvas()
for b in boxes:
    # plot the boxes path
    c.stroke(b.path(), [color.rgb.green])
    # a second time with bezier rounded corners
    c.stroke(b.path(bezierradius=0.5), [color.rgb.red])
c.writeEPSfile("box")
```



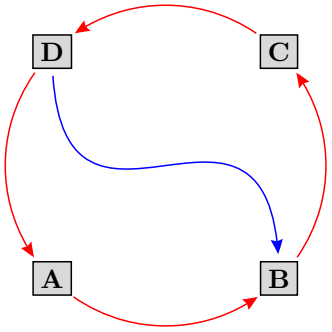
circles

```
from pyx import *

circ1 = path.normpath(path.circle(0, 0, 1)) # you don't really need normpath,
circ2 = path.normpath(path.circle(1, 1, 1)) # but its better to have it once
                                            # for those operations
(circ1a, circ1b), (circ2a, circ2b) = circ1.intersect(circ2)
intersection = (circ1.split([circ1a, circ1b])[0]
                << circ2.split([circ2b, circ2a])[0])
intersection.append(path.closepath())

union = (circ1.split([circ1a, circ1b])[1]
         << circ2.split([circ2b, circ2a])[1])
union.append(path.closepath())

c = canvas.canvas()
c.fill(union, [color.rgb.blue])
c.fill(intersection, [color.rgb.red])
c.stroke(circ1)
c.stroke(circ2)
c.writeEPSfile("circles")
```



connect

```
from pyx import *
from pyx.connector import arc, curve

unit.set(uscale=3)

c = canvas.canvas()

textattrs = [text.halign.center, text.vshift.middlezero]
A = text.text(0, 0, r"\bf A", textattrs)
B = text.text(1, 0, r"\bf B", textattrs)
C = text.text(1, 1, r"\bf C", textattrs)
D = text.text(0, 1, r"\bf D", textattrs)

for X in [A, B, C, D]:
    c.draw(X.bbox().enlarged(0.1).path(),
        [deco.stroked(), deco.filled([color.grey(0.85)])])
    c.insert(X)

for X,Y in [[A, B], [B, C], [C, D], [D, A]]:
    c.stroke(arc(X, Y, boxdists=0.2), [color.rgb.red, deco.earrow.normal])

c.stroke(curve(D, B, boxdists=0.2, relangle1=45, relangle2=-45, relbulge=0.8),
         [color.rgb.blue, deco.earrow.normal])

c.writeEPSfile("connect")
```

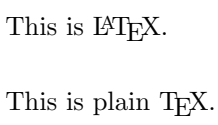


hello

```
from pyx import *

c = canvas.canvas()
c.text(0, 0, "Hello, world!")
c.stroke(path.line(0, 0, 2, 0))
c.writeEPSfile("hello")
```

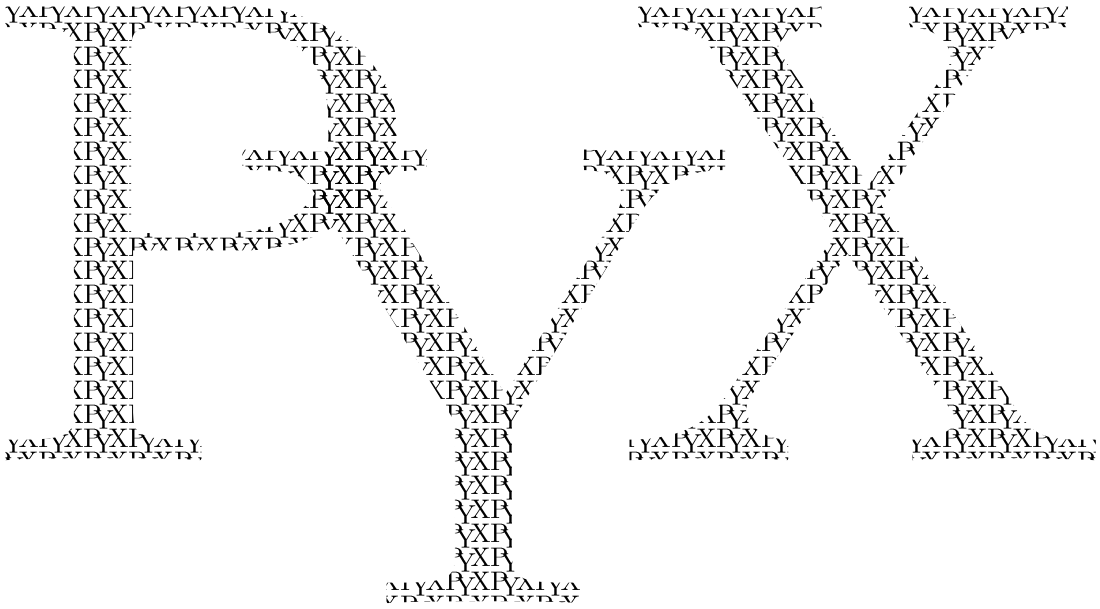
latex

```
from pyx import *

# set properties of the defaulttexrunner, e.g. switch to LaTeX
text.set(mode="latex")

c = canvas.canvas()
# the canvas, by default, uses the defaulttexrunner from the text module
# (this can be changed by the canvas method settexrunner)
c.text(0, 0, r"This is \LaTeX.")

# you can have several texrunners (several running TeX/LaTeX instances)
plaintex = text.texrunner() # plain TeX instance
c.insert(plaintex.text(0, -1, r"This is plain \TeX."))

c.writeEPSfile("latex")
```



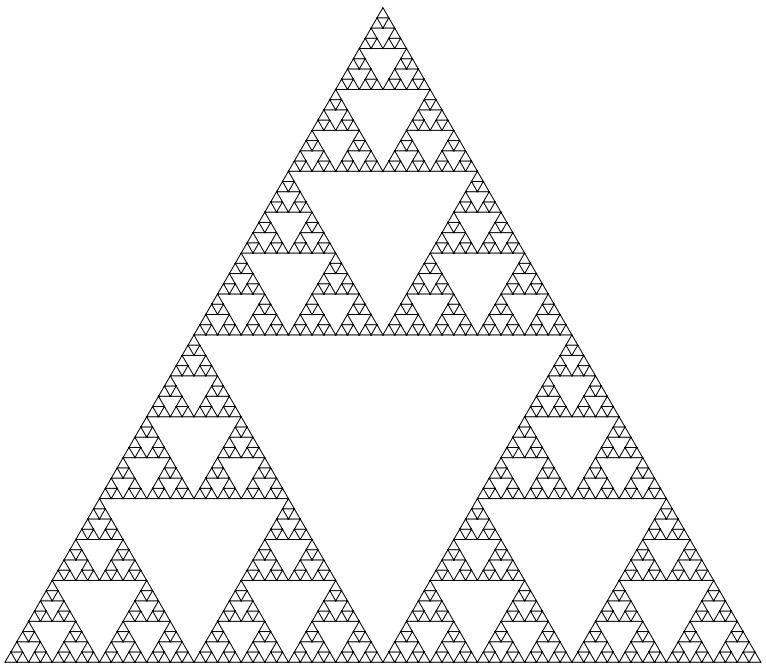
pattern

```
from pyx import *

p = canvas.pattern()
p.text(0, 0, r"\PyX")

c = canvas.canvas()
c.text(0, 0, r"\PyX", [trafo.scale(25), p])
c.writeEPSfile("pattern")
```



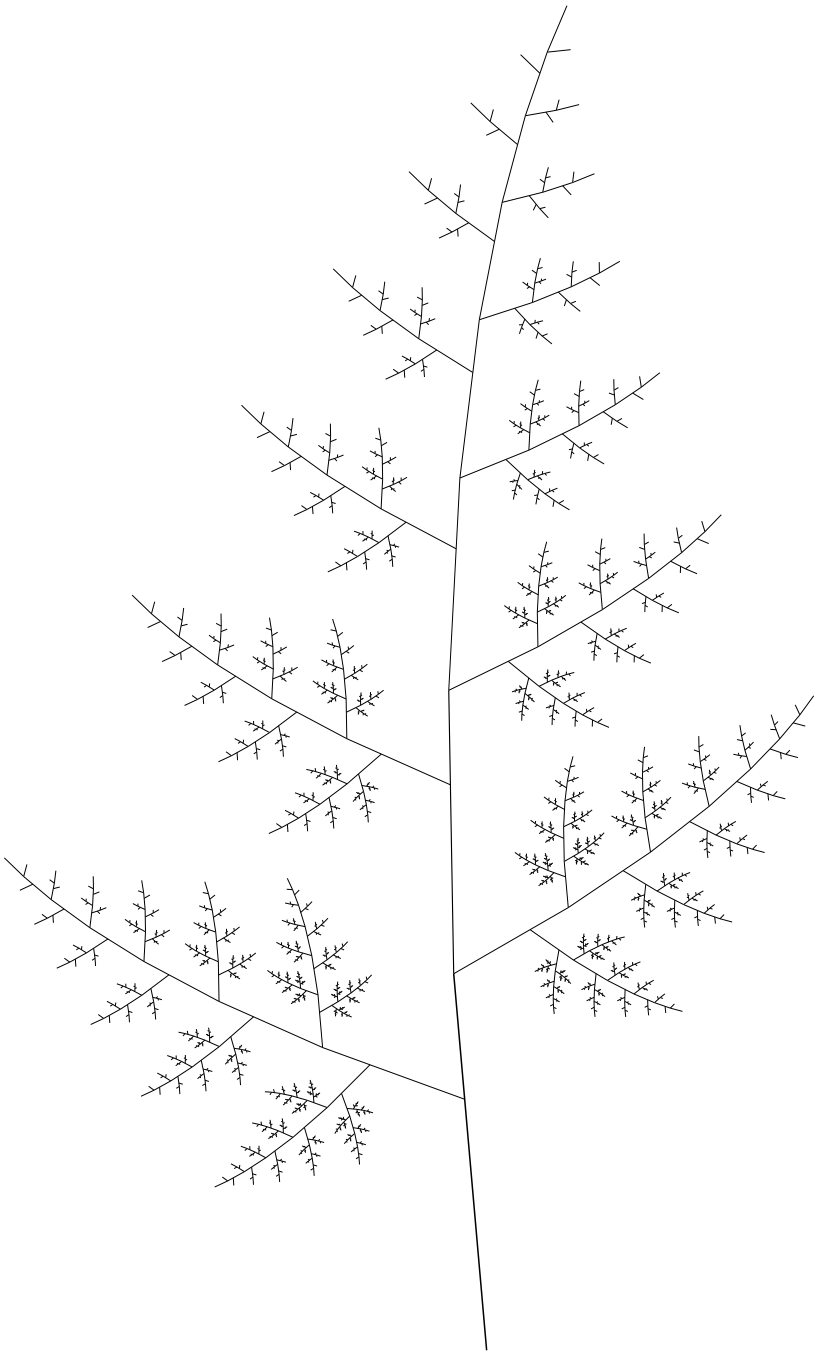
sierpinski

```
# Sierpinski triangle
# contributed by Gerhard Schmid

from math import sqrt
from pyx import *

# triangle geometry
l = 10
h = 0.5 * sqrt(3) * l

# base triangle path
p = path.path(path.moveto(0, 0),
              path.lineto(l, 0),
              path.lineto(0.5 * l, h),
              path.closepath())

for i in range(6):
    # path is scaled down ...
    p = p.transformed(trafo.scale(0.5))
    # ... and three times plotted (translated accordingly)
    p += ( p.transformed(trafo.translate(0.5 * l, 0)) +
           p.transformed(trafo.translate(0.25 * l, 0.5 * h)) )

c = canvas.canvas()
c.stroke(p, [style.linewidth.Thin])
c.writeEPSfile("sierpinski", paperformat="a4")
```



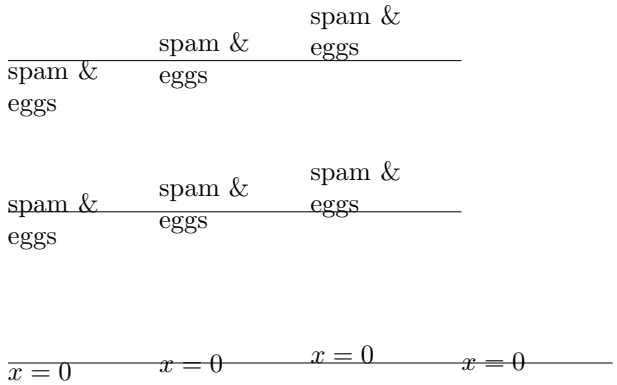
tree

```
# -*- coding: ISO-8859-1 -*-
# fractal tree
# contributed by Gerhard Schmid and André Wobst

from pyx import *

# base tree length
l = 5

# base transformations for the left , center, and right part of the tree
ltrafo = trafo.rotate(65).scaled(0.4).translated(0, l * 2.0 / 3.0)
ctrafo = trafo.rotate(-4).scaled(0.75).translated(0, l)
rtrafo = trafo.mirror(90).rotated(-65).scaled(0.35).translated(0, l)

def tree(depth):
    "return transformations for a recursive tree of given depth"
    r = [trafo.rotate(5)]
    if depth > 0:
        subtree = tree(depth - 1)
        r.extend([t*ltrafo for t in subtree])
        r.extend([t*ctrafo for t in subtree])
        r.extend([t*rtrafo for t in subtree])
    return r

c = canvas.canvas()
for t in tree(7):
    # apply the transformation to a "sub"-canvas and insert it into the "main" canvas
    c.insert(canvas.canvas([t])).stroke(path.line(0, 0, 0, l))
c.writeEPSfile("tree", paperformat="a4")
```



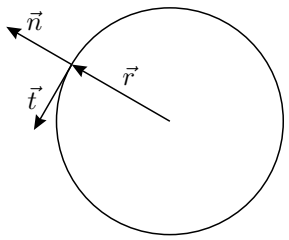
valign

```
from pyx import *

c = canvas.canvas()

# apply global TeX setting
text.preamble(r"\parindent=0pt")
w = 1.2 # an appropriate parbox width

# vertical alignments by margins
c.stroke(path.line(0, 4, 6, 4), [style.linewidth.THin])
c.text(0, 4, r"spam \& eggs", [text.parbox(w), text.valign.top])
c.text(2, 4, r"spam \& eggs", [text.parbox(w), text.valign.middle])
c.text(4, 4, r"spam \& eggs", [text.parbox(w), text.valign.bottom])

# vertical alignments by baselines
c.stroke(path.line(0, 2, 6, 2), [style.linewidth.THin])
c.text(0, 2, r"spam \& eggs", [text.parbox(w, baseline=text.parbox.top)])
c.text(2, 2, r"spam \& eggs", [text.parbox(w, baseline=text.parbox.middle)])
c.text(4, 2, r"spam \& eggs", [text.parbox(w, baseline=text.parbox.bottom)])

# vertical shifts
c.stroke(path.line(0, 0, 8, 0), [style.linewidth.THin])
c.text(0, 0, r"x=0", [text.mathmode, text.vshift.topzero])
c.text(2, 0, r"x=0", [text.mathmode, text.vshift.middlezero])
c.text(4, 0, r"x=0", [text.mathmode, text.vshift.bottomzero])
c.text(6, 0, r"x=0", [text.mathmode, text.vshift.mathaxis])

c.writeEPSfile("valign")
```



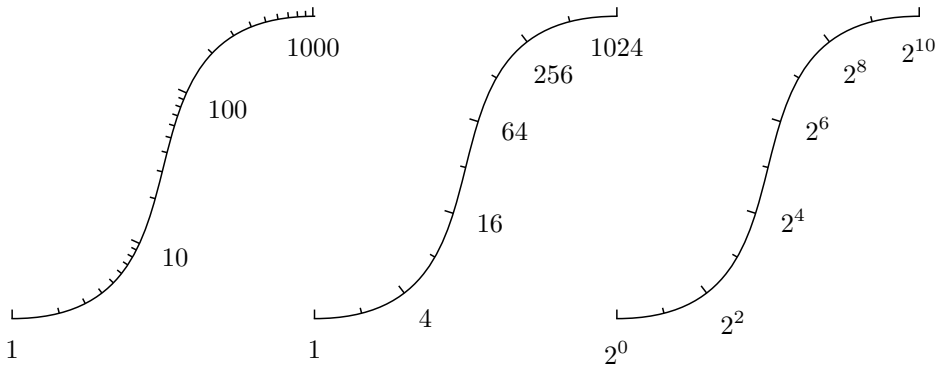
vector

```
from math import pi, sin, cos
from pyx import *

def vector(x1, y1, x2, y2, t, pos=0.5, distance=0.1):
    c = canvas.canvas()
    c.stroke(path.line(x1, y1, x2, y2), [deco.earrow.normal])
    textbox = text.text((1-pos)*x1 + pos*x2, (1-pos)*y1 + pos*y2, t,
                        [text.halign.center, text.vshift.mathaxis])
    if distance < 0:
        textbox.linealign(-distance, y1 - y2, x2 - x1)
    else:
        textbox.linealign(distance, y2 - y1, x1 - x2)
    c.insert(textbox)
    return c

r = 1.5
a = 150

c = canvas.canvas()
dx, dy = cos(a * pi / 180), sin(a * pi / 180)
x, y = r * dx, r * dy
c.stroke(path.circle(0, 0, r))
c.insert(vector(0, 0, x, y, r"$\vec r$"))
c.insert(vector(x, y, x - dy, y + dx, r"$\vec t$", pos=0.7))
c.insert(vector(x, y, x + dx, y + dy, r"$\vec n$", pos=0.7))
c.writeEPSfile("vector")
```



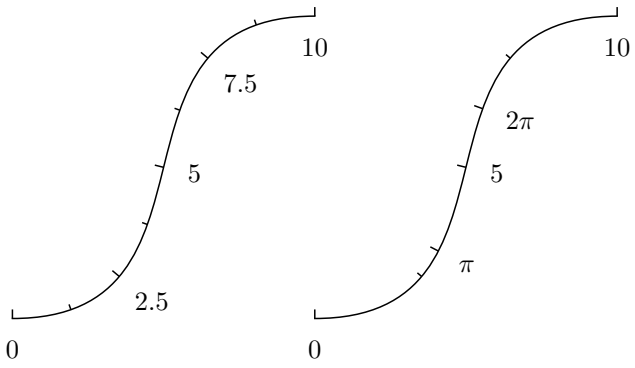
axis/log

```
# Certainly logarithmic axes are supported in PyX. By playing with
# partitioners and texters, you can easily change the base.
#
# It is left as an exercise to the reader to create a automatic
# partitioner for logarithmic axes with base 2.

import math
from pyx import *
from pyx.graph import axis

p = path.curve(0, 0, 3, 0, 1, 4, 4, 4)

log2parter = axis.parter.log([axis.parter.preexp([axis.tick.rational(1)], 4),
                              axis.parter.preexp([axis.tick.rational(1)], 2)])
log2texter = axis.texter.exponential(nomantissaexp=r"{2^{%s}}",
                                     mantissamax=axis.tick.rational(2))

c = canvas.canvas()
c.insert(axis.pathaxis(p, axis.log(min=1, max=1024)))
c.insert(axis.pathaxis(p.transformed(trafo.translate(4, 0)),
                       axis.log(min=1, max=1024, parter=log2parter)))
c.insert(axis.pathaxis(p.transformed(trafo.translate(8, 0)),
                       axis.log(min=1, max=1024, parter=log2parter,
                                texter=log2texter)))
c.writeEPSfile("log")
```



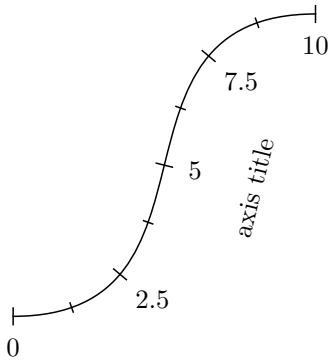
axis/manualticks

```
# Ticks can be set manually in combination with automatically created
# ticks. Note that the rating takes into account the manual ticks as
# well.

import math
from pyx import *
from pyx.graph import axis

p = path.curve(0, 0, 3, 0, 1, 4, 4, 4)

myticks = [axis.tick.tick(math.pi, label="\pi", labelattrs=[text.mathmode]),
           axis.tick.tick(2*math.pi, label="2\pi", labelattrs=[text.mathmode])]

c = canvas.canvas()
c.insert(axis.pathaxis(p, axis.linear(min=0, max=10)))
c.insert(axis.pathaxis(p.transformed(trafo.translate(4, 0)),
                       axis.linear(min=0, max=10, manualticks=myticks)))
c.writeEPSfile("manualticks")
```



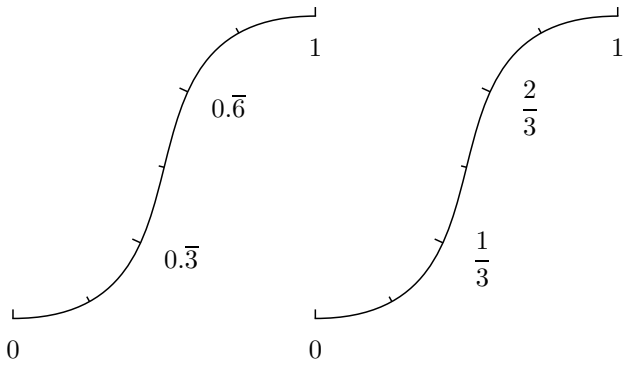
axis/painter

```
# Axis painters performs the painting of an axis. By default, ticks
# are stroked inside of the graph (in the path examples there is no
# graph but don't mind) while labels and the axis title are plotted
# outside. The axis title is rotated along the axis (without writing it
# upside down), while the tick labels are not rotated. The axis painters
# takes a variety of keyword arguments to modify the default
# behaviour.

from pyx import *
from pyx.graph import axis

ap = axis.painter.regular(outerticklength=axis.painter.ticklength.normal)

c = axis.pathaxis(path.curve(0, 0, 3, 0, 1, 4, 4, 4),
                  axis.linear(min=0, max=10, title="axis title",
                              painter=ap))
c.writeEPSfile("painter")
```



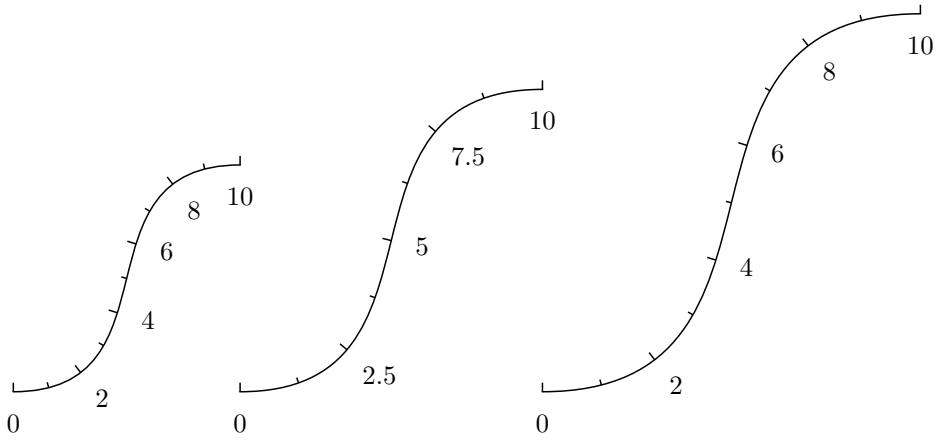
axis/parter

```
# Partitioners (in the code the short form parter is used) take
# care of calculating appropriate tick positions for a given axis
# range. Automatic partitioners create several tick lists , which are
# rated by an axis rater instance afterwards while manual partitioners
# create a single tick list only, which thus doesn't need to be rated.
#
# Note that the partitioning uses fractional number arithmetics. For
# that, tick instances can be initialized with floats using a fixed
# precision but also with strings as shown.

import math
from pyx import *
from pyx.graph import axis

p = path.curve(0, 0, 3, 0, 1, 4, 4, 4)

myparter = axis.parter.linear(["1/3", "1/6"])

c = canvas.canvas()
c.insert(axis.pathaxis(p, axis.linear(min=0, max=1, parter=myparter)))
c.insert(axis.pathaxis(p.transformed(trafo.translate(4, 0)),
                       axis.linear(min=0, max=1, parter=myparter,
                                   texter=axis.texter.rational())))
c.writeEPSfile("parter")
```



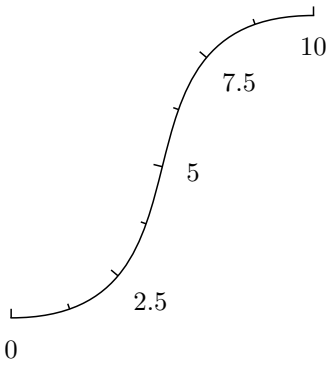
axis/rating

```
# In the example below, several axes with the same parameters are
# plotted on a path scaled in 3 different sizes. Note that the axes
# adjust the ticks appropriately to the available space. The rating
# mechanism takes into account the number of ticks and subticks, but
# also the distances between the labels. Thus, the example in the
# middle has less ticks than the smallest version, because there is
# not enough room for labels with a decimal place.
#
# The rating mechanism is configurable and exchangeable by the axis
# keyword argument "rater". Instead of reconfiguring the rating
# mechanism, simple adjustments to favour more or less ticks are
# possible by the axis keyword argument "density".

from pyx import *
from pyx.graph import axis

p = path.curve(0, 0, 3, 0, 1, 4, 4, 4)

c = canvas.canvas()
c.insert(axis.pathaxis(p.transformed(trafo.translate(-4, 0).scaled(0.75)),
                       axis.linear(min=0, max=10)))
c.insert(axis.pathaxis(p, axis.linear(min=0, max=10)))
c.insert(axis.pathaxis(p.transformed(trafo.scale(1.25).translated(4, 0)),
                       axis.linear(min=0, max=10)))
c.writeEPSfile("rating")
```



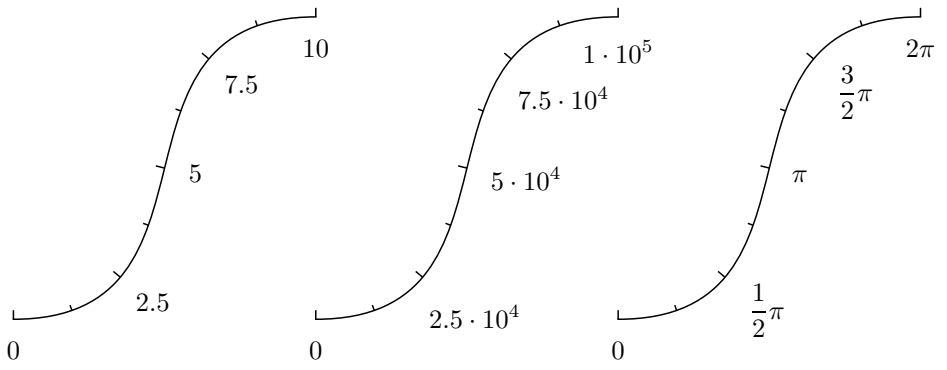
axis/simple

```
# This is the basic example how to draw an axis along an arbitrary
# path. The function pathaxis from the graph.axis module takes a path
# and returns a canvas. Different from the typical usecase in graphs,
# we must fix the axis range by appropriate min and max arguments,
# because of missing data. In graphs the range can be adjusted
# automatically.

from pyx import *

c = graph.axis.pathaxis(path.curve(0, 0, 3, 0, 1, 4, 4, 4),
                        graph.axis.linear(min=0, max=10))
c.writeEPSfile("simple")
```



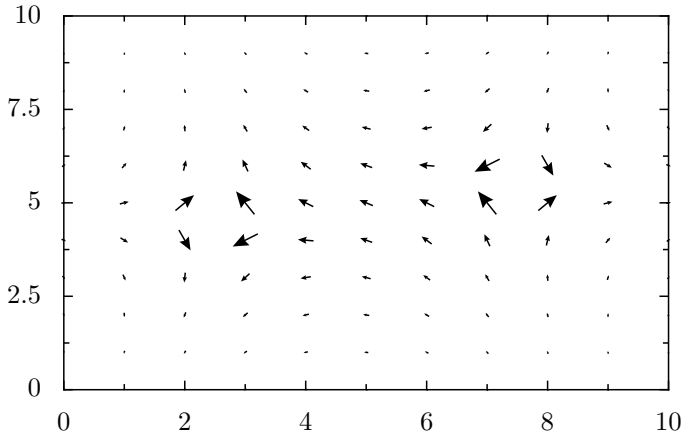
axis/texter

```
# Texters create the label strings written to the ticks. There are
# texters available for decimal numbers without and with an
# exponential part as well as fractions. Internally, the partitioning
# is based on fractions to avoid any rounding problems.
#
# Although we could modify axis.linear into a piaxis "inplace", we
# define a special piaxis below to give an impression, how easy
# alternative default settings can be implemented. A more advanced
# task would be to add an appropriate special partitioner for a
# piaxis.

import math
from pyx import *
from pyx.graph import axis

class piaxis(axis.linear):

    def __init__(self, divisor=math.pi,
                 texter=axis.texter.rational(suffix="\pi"), **kwargs):
        axis.linear.__init__(self, divisor=divisor, texter=texter, **kwargs)


p = path.path(path.moveto(0, 0), path.curveto(3, 0, 1, 4, 4, 4))

c = canvas.canvas()
c.insert(axis.pathaxis(p, axis.linear(min=0, max=10)))
c.insert(axis.pathaxis(p.transformed(trafo.translate(4, 0)),
                       axis.linear(min=0, max=1e5)))
c.insert(axis.pathaxis(p.transformed(trafo.translate(8, 0)),
                       piaxis(min=0, max=2*math.pi)))
c.writeEPSfile("texter")
```



graphs/arrows

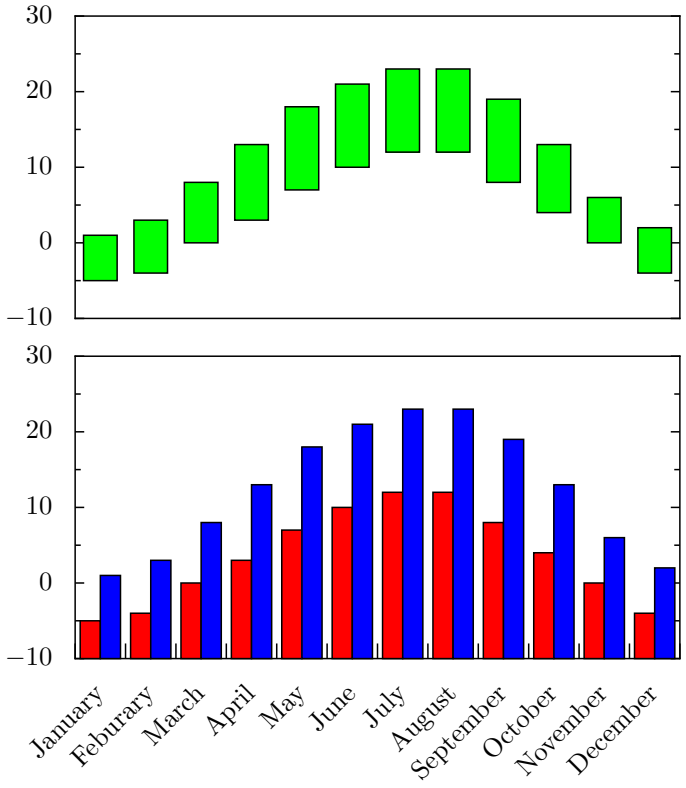
```
from math import pi, atan2
from pyx import *

z1 = 2.5 + 4.5j
z2 = 7.5 + 5.5j

# we abuse a parametric function below, so we express everything in terms of a parameter k
x = lambda k: int(k)/11
y = lambda k: int(k)%11
z = lambda k: x(k) + y(k) * 1j
f = lambda k: 1/(z(k)-z1)/(z(k)-z2)                    # function to be plotted
s = lambda k: 5*abs(f(k))                              # magnitude of function value
a = lambda k: 180/pi*atan2(f(k).imag, f(k).real)       # direction of function value

g = graph.graphxy(width=8,
                  x=graph.axis.linear(min=0, max=10),
                  y=graph.axis.linear(min=0, max=10))
g.plot(graph.data.paramfunction("k", 0, 120,
                                "x,␣y,␣size,␣angle␣=␣x(k),␣y(k),␣s(k),␣a(k)",
                                points=121, context=locals()),# access extern
       graph.style.arrow())                                   # variables&functions
g.writeEPSfile("arrows")                                      # by passing a context
```



graphs/bar

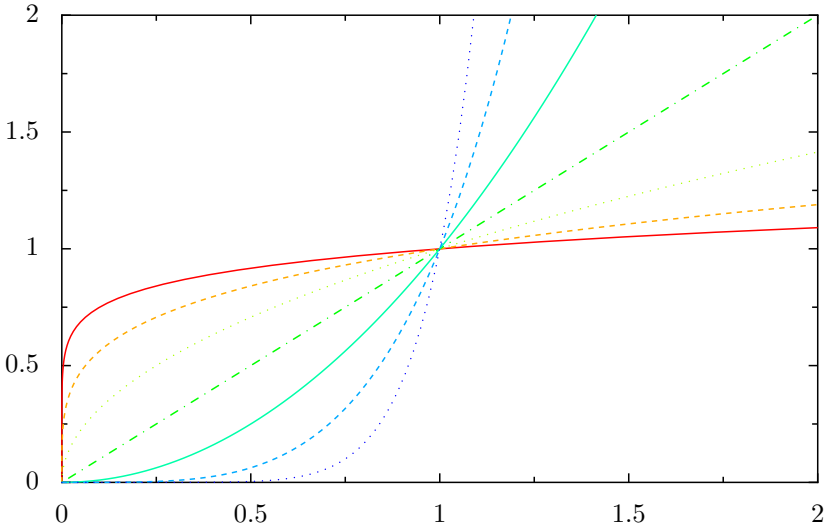
```
from pyx import *
from pyx.graph import axis

# we prepare some stuff first
bap = axis.painter.bar # just an abbreviation
a1 = axis.bar(painter=bap(nameattrs=None)) # for single bars
a2 = axis.bar(painter=bap(nameattrs=[trafo.rotate(45),
                                    text.halign.right],
                                    innerticklength=0.2),
                          subaxis=axis.bar(dist=0)) # for several bars
nofirst = [attr.changelist([None, color.rgb.green])] # special draw attrs

c = canvas.canvas() # we draw several plots, thus we create a main canvas
g = c.insert(graph.graphxy(ypos=4.5, width=8, height=4, x=a1))
g.plot(graph.data.file("bar.dat", xname=1, y=2, ystack1=3),
       graph.style.bar(barattrs=nofirst))
g2 = c.insert(graph.graphxy(width=8, x=a2, height=4))
g2.plot([graph.data.file("bar.dat", xname=1, y=2),
         graph.data.file("bar.dat", xname=1, y=3)],
        graph.style.bar())
c.writeEPSfile("bar")
```

```
# bar.dat looks like :
# #month   min max
# January    -5   1
# Feburary   -4   3
   ...
```



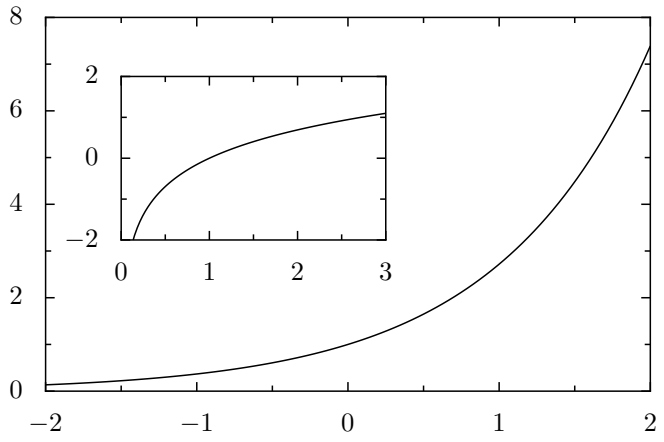
graphs/change

```
from pyx import *

g = graph.graphxy(width=10,
                  x=graph.axis.linear(min=0, max=2),
                  y=graph.axis.linear(min=0, max=2))
g.plot([graph.data.function("x=y**(2**(3-%i))" % i) for i in range(3)] +
       [graph.data.function("y=x**(2**%i)" % i) for i in range(4)],
       graph.style.line([color.palette.Rainbow]))
g.writeEPSfile("change")
```



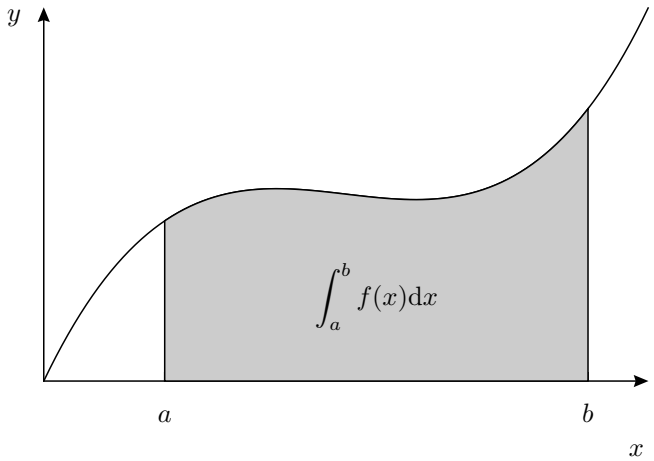
graphs/inset

```
from pyx import *

g = graph.graphxy(width=8, x=graph.axis.linear(min=-2, max=2))
g.plot(graph.data.function("y=exp(x)"))
g2 = g.insert(graph.graphxy(width=3.5, xpos=1, ypos=2,
                            x=graph.axis.linear(min=0, max=3),
                            y=graph.axis.linear(min=-2, max=2)))
g2.plot(graph.data.function("y=log(x)"))
g.writeEPSfile("inset")
```



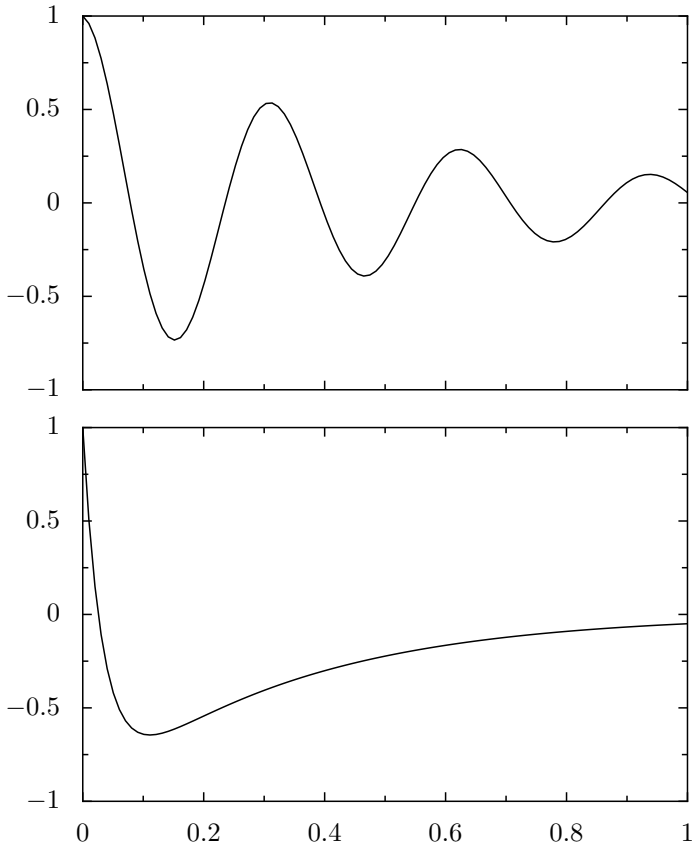
graphs/integral

```
from pyx import *

a, b = 2, 9 # integral area

p = graph.axis.painter.regular(basepathattrs=[deco.earrow.normal],
                               titlepos=0.98, titledirection=None)
ticks = [graph.axis.tick.tick(a, label="$a$"),
         graph.axis.tick.tick(b, label="$b$")]
g = graph.graphxy(width=8, x2=None, y2=None,
                  x=graph.axis.linear(title="$x$", min=0, max=10,
                                      manualticks=ticks,
                                      parter=None, painter=p),
                  y=graph.axis.linear(title="$y$", parter=None, painter=p))
d = g.plot(graph.data.function("y=(x-3)*(x-5)*(x-7)"))
g.finish()
p = d.path # the path is available after the graph is finished

pa = g.xgridpath(a)
pb = g.xgridpath(b)
(splita,), (splitpa,) = p.intersect(pa)
(splitb,), (splitpb,) = p.intersect(pb)
area = (pa.split([splitpa])[0] <<
        p.split([splita, splitb])[1] <<
        pb.split([splitpb])[0].reversed())
area.append(path.closepath())
g.stroke(area, [deco.filled([color.gray(0.8)])])
g.text(g.pos(0.5 * (a + b), 0)[0], 1,
       r"\int_a^b f(x){\rm d}x", [text.halign.center, text.mathmode])
g.writeEPSfile("integral")
```



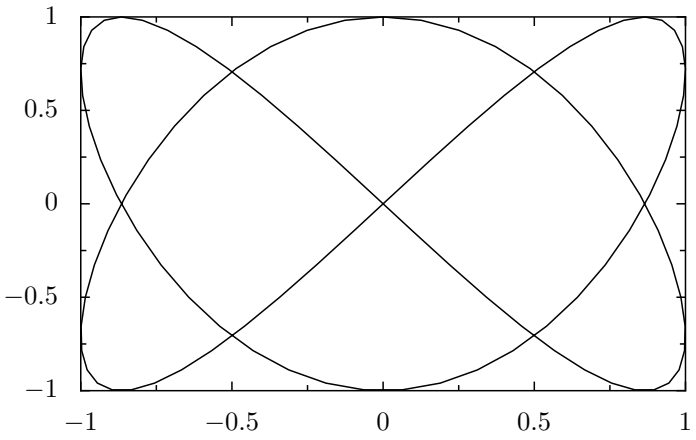
graphs/link

```
import math
from pyx import *

c = canvas.canvas()

g1 = c.insert(graph.graphxy(width=8,
                            x=graph.axis.linear(min=0, max=1)))
g1.plot(graph.data.function("y=2*exp(-30*x)-exp(-3*x)"))

g2 = c.insert(graph.graphxy(width=8, ypos=g1.height+0.5,
                            x=g1.axes["x"].createlinkaxis()))
g2.plot(graph.data.function("y=cos(20*x)*exp(-2*x)"))

c.writeEPSfile("link")
```



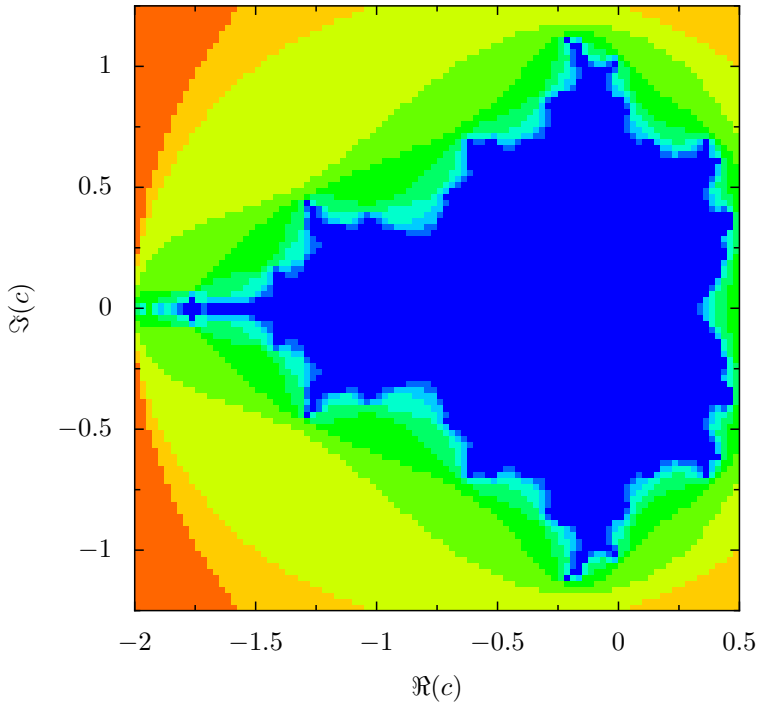
graphs/lissajous

```
from math import pi
from pyx import *

g = graph.graphxy(width=8)
g.plot(graph.data.paramfunction("k", 0, 2*pi, "x, y = sin(2*k), cos(3*k)"))
g.writeEPSfile("lissajous")
```



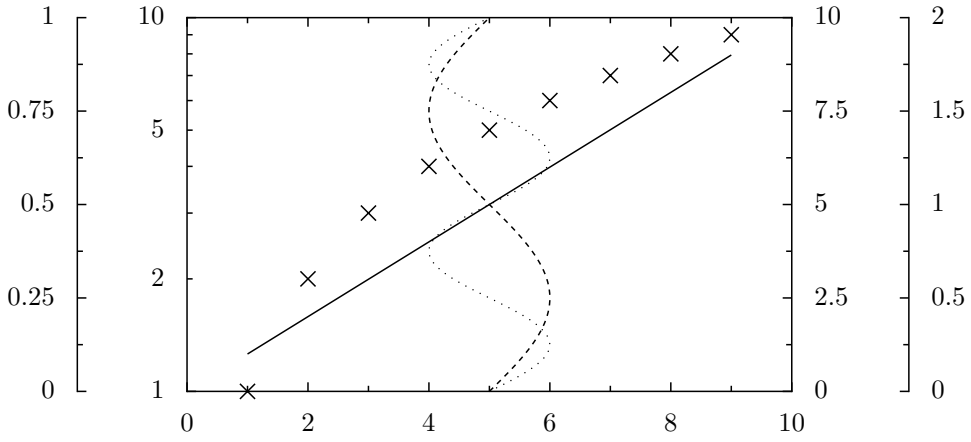
graphs/mandel

```
# contributed by Stephen Phillips

from pyx import *

# Mandelbrot parameters
re_min = -2
re_max = 0.5
im_min = -1.25
im_max = 1.25
gridx = 100
gridy = 100
max_iter = 10

# Set-up
re_step = (re_max - re_min) / gridx
im_step = (im_max - im_min) / gridy
d = []

# Compute fractal
for re_index in range(gridx):
    re = re_min + re_step * (re_index + 0.5)
    for im_index in range(gridy):
        im = im_min + im_step * (im_index + 0.5)
        c = complex(re, im)
        n = 0
        z = complex(0, 0)
        while n < max_iter and abs(z) < 2:
            z = (z * z) + c
            n += 1
        d.append([re - 0.5 * re_step, re + 0.5 * re_step,
                  im - 0.5 * im_step, im + 0.5 * im_step,
                  float(n)/max_iter])

# Plot graph
g = graph.graphxy(height=8, width=8,
                  x=graph.axis.linear(min=re_min, max=re_max, title=r'$\Re(c)$'),
                  y=graph.axis.linear(min=im_min, max=im_max, title=r'$\Im(c)$'))
g.plot(graph.data.list(d, xmin=1, xmax=2, ymin=3, ymax=4, color=5),
       graph.style.rect(color.palette.Rainbow))
g.dodata() # plot data first, then axes
g.writeEPSfile('mandel')
```


1
0.5
0
−0.5
−1
−2
−1.5
−1
−0.5
0
0.5
ℑ(c)
ℜ(c)

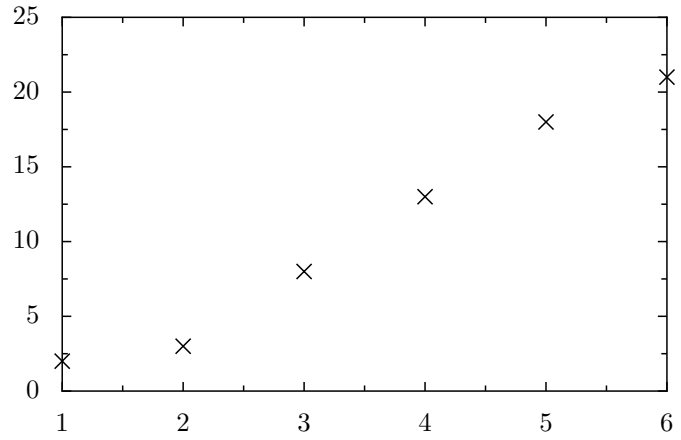
graphs/manyaxes

```
import math, random
from pyx import *

# a xy-graph has linear x and y axes by default
# they might be overwritten and futher axes might be added as well
g = graph.graphxy(width=8, y=graph.axis.log(), y2=graph.axis.lin(),
                  y3=graph.axis.lin(min=0, max=1),
                  y4=graph.axis.lin(min=0, max=2))

# we generate some data and a function with multiple arguments
d = [[i, math.exp(0.8*i+random.random())] for i in range(1,10)]
f = lambda x, a: x*a

g.plot(graph.data.list(d, x=0, y=1))
g.plot(graph.data.function("y2=f(x, 1)", context=locals()))

g.plot(graph.data.function("x=5+sin(2*pi*y3)"))
g.plot(graph.data.function("x=5+sin(2*pi*y4)"))

g.writeEPSfile("manyaxes")
```



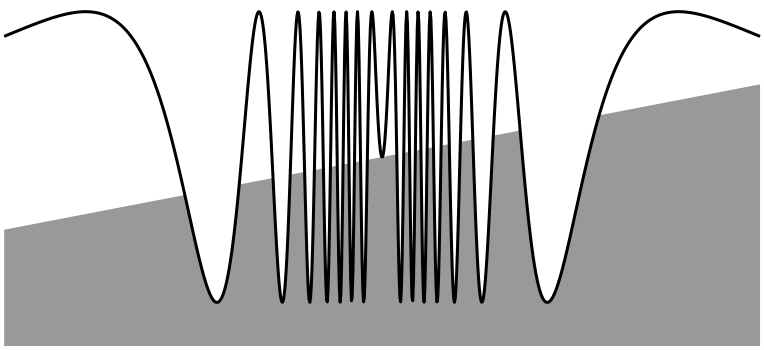
graphs/minimal

```
from pyx import *

g = graph.graphxy(width=8)
g.plot(graph.data.file("minimal.dat", x=1, y=2))
g.writeEPSfile("minimal")

# the file minimal.dat looks like:
# 1 2
# 2 3
# 3 8
# 4 13
# 5 18
# 6 21

# graph styles can be modified by a second parameter to the plot method:
# g.plot(graph.data.file("minimal.dat", x=1, y=2), graph.line())
```



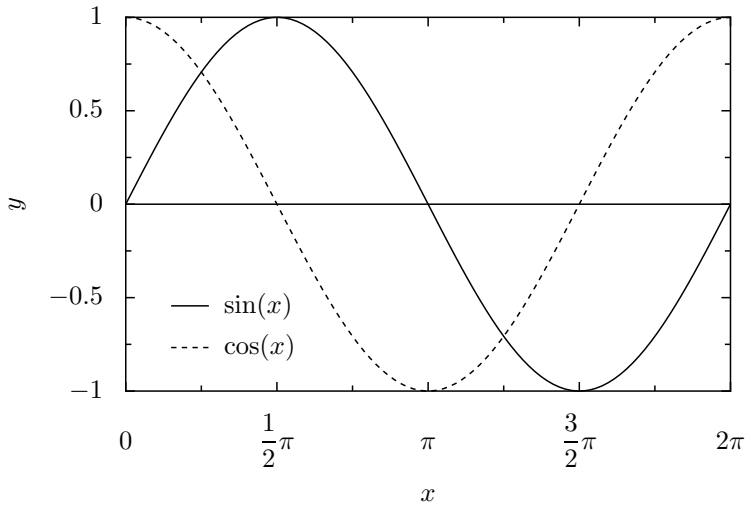
graphs/partialfill

```
# contributed by Michael Schindler

from pyx import *

# get the lines from the graph
xax = graph.axis.linear(min=-1, max=1.0, painter=None)
yax = graph.axis.linear(min=-1.3, max=1.3, painter=None)
g = graph.graphxy(width=10, ratio=2, x=xax, y=yax)
fline = g.plot(graph.data.function("y=sin(1.0/(x**2+0.02122))", points=1000))
horiz = g.plot(graph.data.function("y=0.5*x", points=2))
g.finish()

# convert paths to normpaths (for efficiency reasons only)
fline = path.normpath(fline.path)
horiz = path.normpath(horiz.path)
# intersect the lines
splith, splitf = horiz.intersect(fline)

# create gray area (we do not use simple clipping)
area = horiz.split([splith[0]])[0]
for i in range(0, len(splith)-2, 2):
    area = area.joined(fline.split([splitf[i], splitf[i+1]])[1])
    area = area.joined(horiz.split([splith[i+1], splith[i+2]])[1])
area = area.joined(fline.split([splitf[-2], splitf[-1]])[1])
area = area.joined(horiz.split([splith[-1]])[1])
area.append(path.lineto(*g.vpos(1, 0)))
area.append(path.lineto(*g.vpos(0, 0)))
area.append(path.closepath())

c = canvas.canvas()

# draw first the area, then the function
c.fill(area, [color.gray(0.6)])
c.stroke(fline, [style.linewidth.Thick, style.linejoin.round])

c.writeEPSfile("partialfill")
```



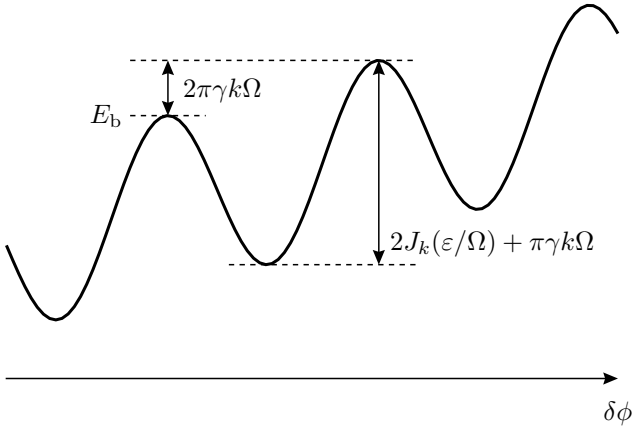
graphs/piaxis

```
from math import pi
from pyx import *
from pyx.graph import axis

g = graph.graphxy(width=8, key=graph.key.key(pos="bl"),
        x=axis.linear(min=0, max=2*pi, title="$x$", divisor=pi,
                      texter=axis.texter.rational(suffix=r"\pi")),
        y=axis.linear(title="$y$"))

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.finish()
g.stroke(g.ygridpath(0))

g.writeEPSfile("piaxis")
```



graphs/washboard

```
# contributed by Sigmund Kohler

from math import pi, cos
from pyx import *
from pyx.deco import barrow, earrow
from pyx.style import linewidth, linestyle
from pyx.graph import graphxy
from pyx.graph.axis import linear
from pyx.graph.axis.painter import regular
from pyx.graph.style import line
from pyx.graph.data import function

mypainter = regular(basepathattrs=[earrow.normal], titlepos=1)
def mycos(x): return -cos(x)+.10*x

g = graphxy(height=5, x2=None, y2=None,
            x=linear(min=-2.5*pi, max=3.3*pi, parter=None,
                     painter=mypainter, title=r"$\delta\phi$"),
            y=linear(min=-2.3, max=2, painter=None))
g.plot(function("y=mycos(x)", context=locals()),
       line(lineattrs=[linewidth.Thick]))
g.finish()

x1, y1 = g.pos(-pi+.1, mycos(-pi+.1))
x2, y2 = g.pos(-.1, mycos(-.1))
x3, y3 = g.pos(pi+.1, mycos(pi+.1))

g.stroke(path.line(x1-.5, y1, x1+.5, y1), [linestyle.dashed])
g.stroke(path.line(x1-.5, y3, x3+.5, y3), [linestyle.dashed])
g.stroke(path.line(x2-.5, y2, x3+.5, y2), [linestyle.dashed])
g.stroke(path.line(x1, y1, x1, y3), [barrow.normal, earrow.normal])
g.stroke(path.line(x3, y2, x3, y3), [barrow.normal, earrow.normal])
g.text(x1+.2, 0.5*(y1+y3), r"$2\pi\gamma_k\Omega$", [text.vshift.middlezero])
g.text(x1-.6, y1-.1, r"$E_{\rm b}$", [text.halign.right])
g.text(x3+.15, y2+.20, r"$2J_k(\varepsilon/\Omega)+\pi\gamma_k\Omega$")

g.writeEPSfile("washboard")
```


$2\pi\gamma k\Omega$
$E_{\rm b}$
$2J_k(\varepsilon/\Omega) + \pi\gamma k\Omega$
$\delta\phi$