Inhaltsverzeichnis

Ebene Geometrie

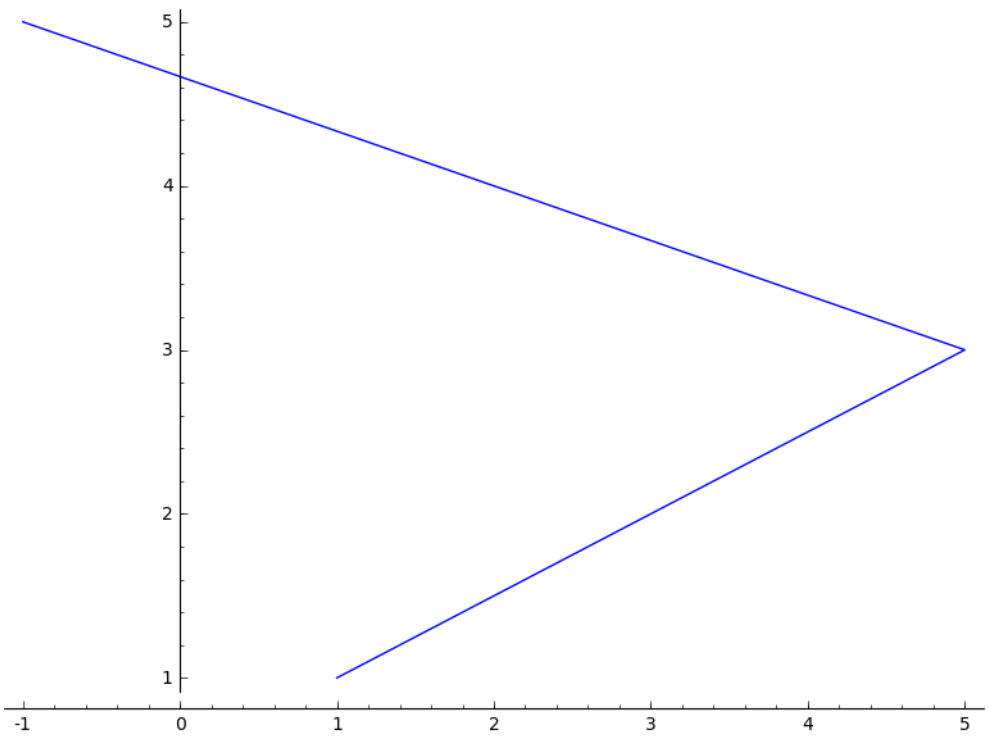
Wir zeichnen ein Dreieck mit seinen Schwerlinien.

In [1]:
```
A=[1,1]
B=[5,3]
C=[-1,5]
```

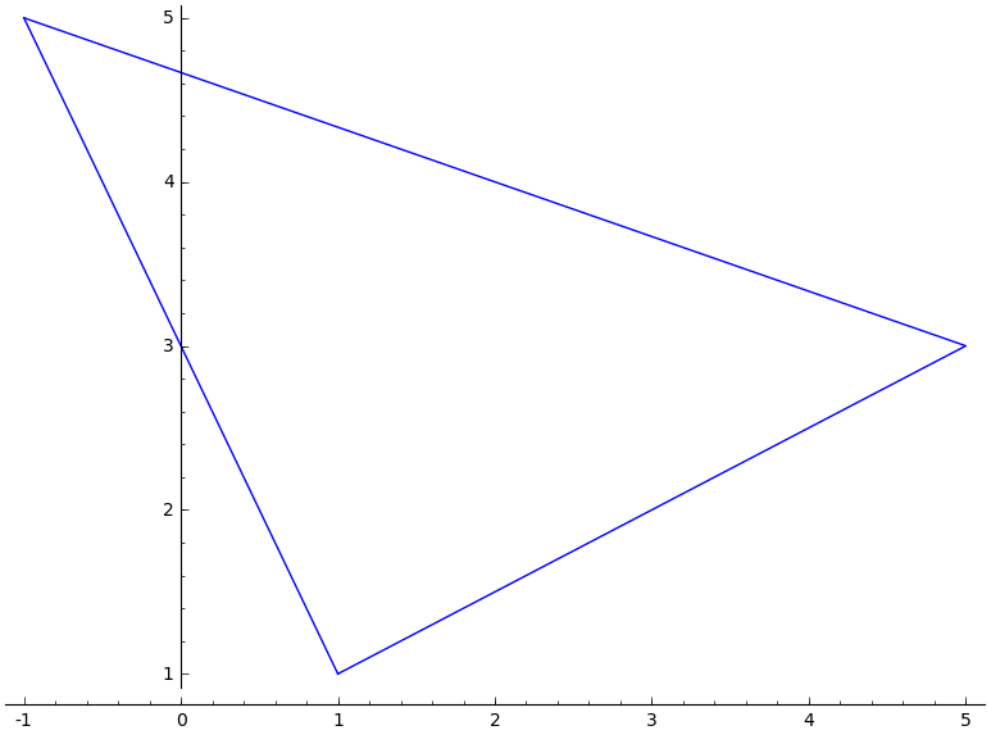
Die Befehle **line**, **point**, **circle** etc. erzeugen die entsprechenden graphischen Objekte und man kann das Bild der Reihe nach aufbauen. Koordinaten können als Listen, Tupel oder Vektoren übergeben werden.

In [2]:
```
g = line([A,B,C])
g
```

Out[2]:



In [3]:
```
g = g+line([A,C])
g.show()
```



Die Schwerlinien verbinden die Mittelpunkte der Seiten mit den gegenüberliegenden Scheiteln.

In [4]:
```
(A+B)/2
```

```
---------------------------------------------------------------
---------------
TypeError                                 Traceback (most r
ecent call last)
<ipython-input-4-f6a0bdb91ec5> in <module>()
----> 1 (A+B)/Integer(2)

/usr/opt/Sage-8.3-amd64/local/lib/python2.7/site-packages/s
age/rings/integer.pyx in sage.rings.integer.Integer.__div__
(build/cythonized/sage/rings/integer.c:13451)()
   1938             return y
   1939
-> 1940         return coercion_model.bin_op(left, right, o
perator.div)
   1941
   1942     cpdef _div_(self, right):

/usr/opt/Sage-8.3-amd64/local/lib/python2.7/site-packages/s
age/structure/coerce.pyx in sage.structure.coerce.CoercionM
odel_cache_maps.bin_op (build/cythonized/sage/structure/coe
rce.c:10874)()
   1225         # We should really include the underlying e
rror.
   1226         # This causes so much headache.
-> 1227         raise bin_op_exception(op, x, y)
   1228
   1229     cpdef canonical_coercion(self, x, y):

TypeError: unsupported operand parent(s) for /: '<type 'lis
t'>' and 'Integer Ring'
```

Listenaddition ist nicht das richtige. Wir müssen die Punkte in Vektoren umwandeln.

In [6]:
```
A=vector(A)
B=vector(B)
C=vector(C)
A,B,C
```

Out[6]:
```
((1, 1), (5, 3), (-1, 5))
```

In [7]:
```
parent(A)
```

Out[7]:
```
Ambient free module of rank 2 over the principal ideal doma
in Integer Ring
```

In [9]:
```
(A+B)/2
```

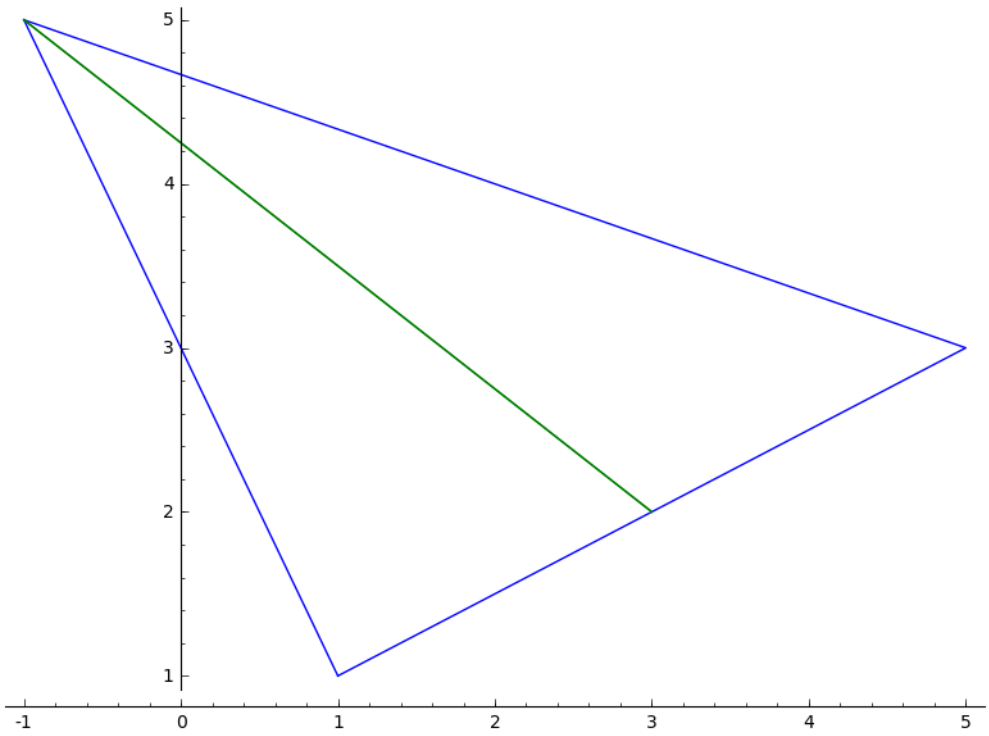
Out[9]:
```
(3, 2)
```

In [10]:
```
parent(_)
```

Out[10]:
```
Vector space of dimension 2 over Rational Field
```

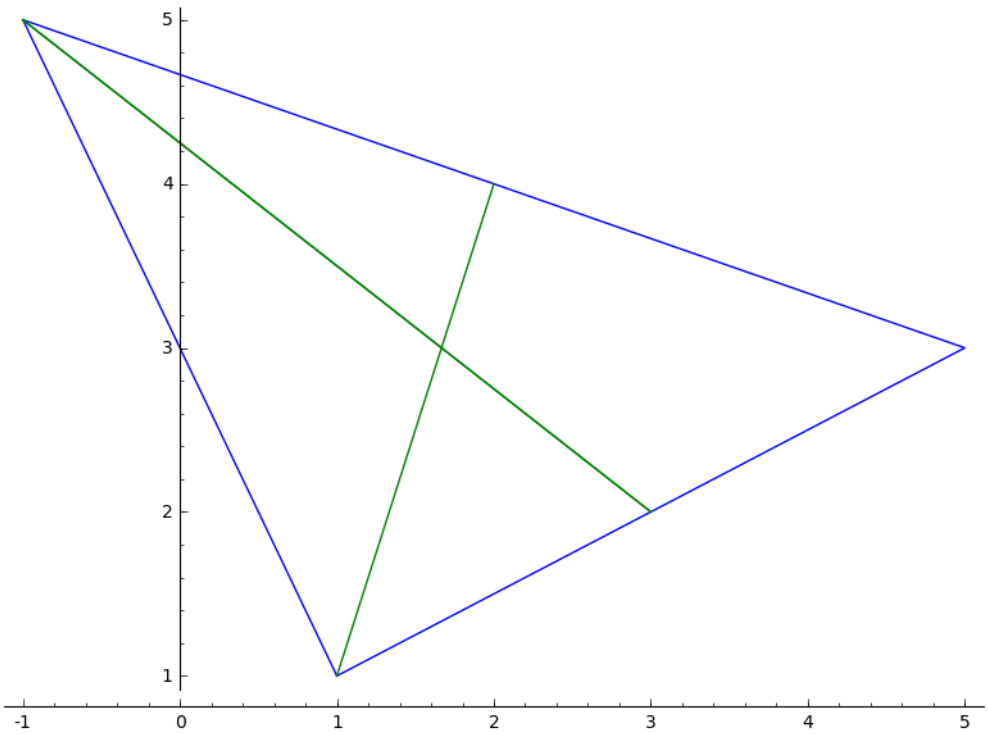
In [11]:
```
g = g + line([(A+B)/2,C],color='green')
g
```

Out[11]:



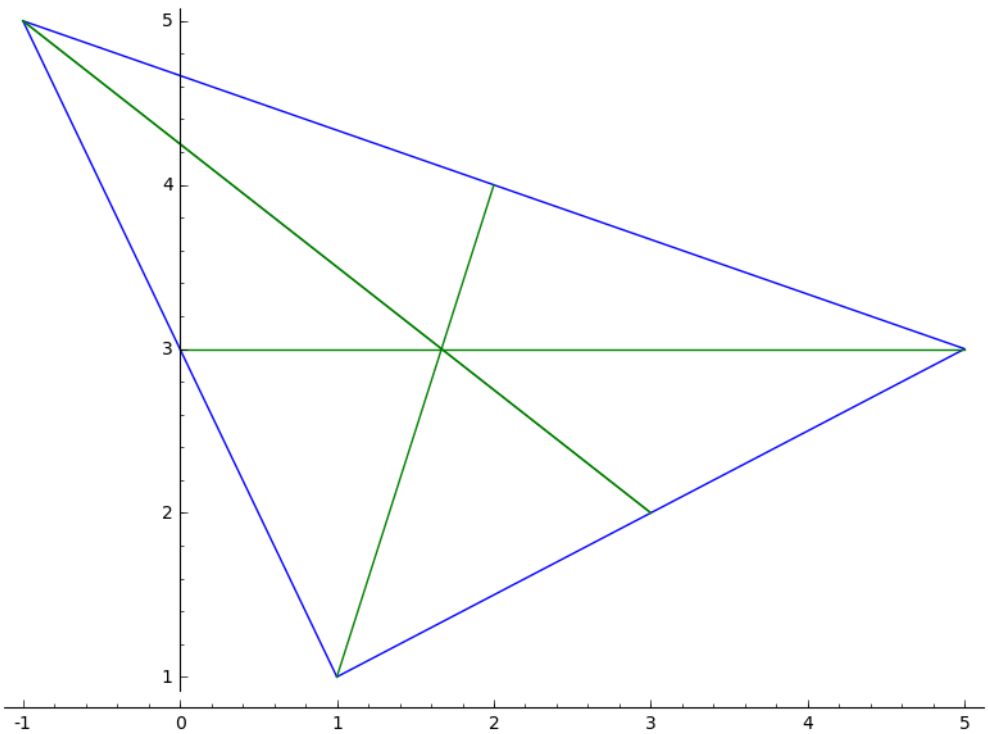
In [12]:
```
g = g+line([(B+C)/2,A],color='green')
g
```

Out[12]:



In [13]:
```
g = g+line([(C+A)/2,B],color='green')
g
```

Out[13]:



Um den Schwerpunkt zu bestimmen, müssen die Schwerlinien geschnitten werden.

In [14]:
```
var('s,t')
```

Out[14]:
```
(s, t)
```

Die Schwerlinien in Parameterdarstellung.

In [15]:
```
sA = A + s * ((B+C)/2-A)
sA
```

Out[15]:
```
(s + 1, 3*s + 1)
```

In [16]:
```
sB = B + t * ((A+C)/2-B)
sB
```

Out[16]:
```
(-5*t + 5, 3)
```

Gleichungen mit Vektoren kann man nicht direkt lösen:

In [17]:
```
solve(sA-sB,[s,t])
```

```
------------------------------------------------------------
---------------
TypeError                                 Traceback (most r
ecent call last)
<ipython-input-17-8d35ec2b0aba> in <module>()
----> 1 solve(sA-sB,[s,t])

/usr/opt/Sage-8.3-amd64/local/lib/python2.7/site-packages/s
age/symbolic/relation.pyc in solve(f, *args, **kwds)
   1048
   1049     if not isinstance(f, (list, tuple)):
-> 1050         raise TypeError("The first argument must be
a symbolic expression or a list of symbolic expressions.")
   1051
   1052     # f is a list of such expressions or equations

TypeError: The first argument must be a symbolic expression
or a list of symbolic expressions.
```

wir müssen sie zuerst in Listen zurückverwandeln.

In [18]:
```
list(sA-sB)
```

Out[18]:
```
[s + 5*t - 4, 3*s - 2]
```

In [19]:
```
st = solve(list(sA-sB),[s,t])
st
```

Out[19]:
```
[[s == (2/3), t == (2/3)]]
```

Wie immer wird eine Liste von Lösungen zurückgegeben, auch wenn die Lösung eindeutig ist.

In [20]:
```
S = sA.subs(st[0])
S
```

Out[20]:
```
(5/3, 3)
```

Probe

In [22]:
```
sB.subs(st[0])
```

Out[22]:
```
(5/3, 3)
```

oder mit dictionary

In [23]:
```
st = solve(list(sA-sB), [s,t], solution_dict=True)
st
```

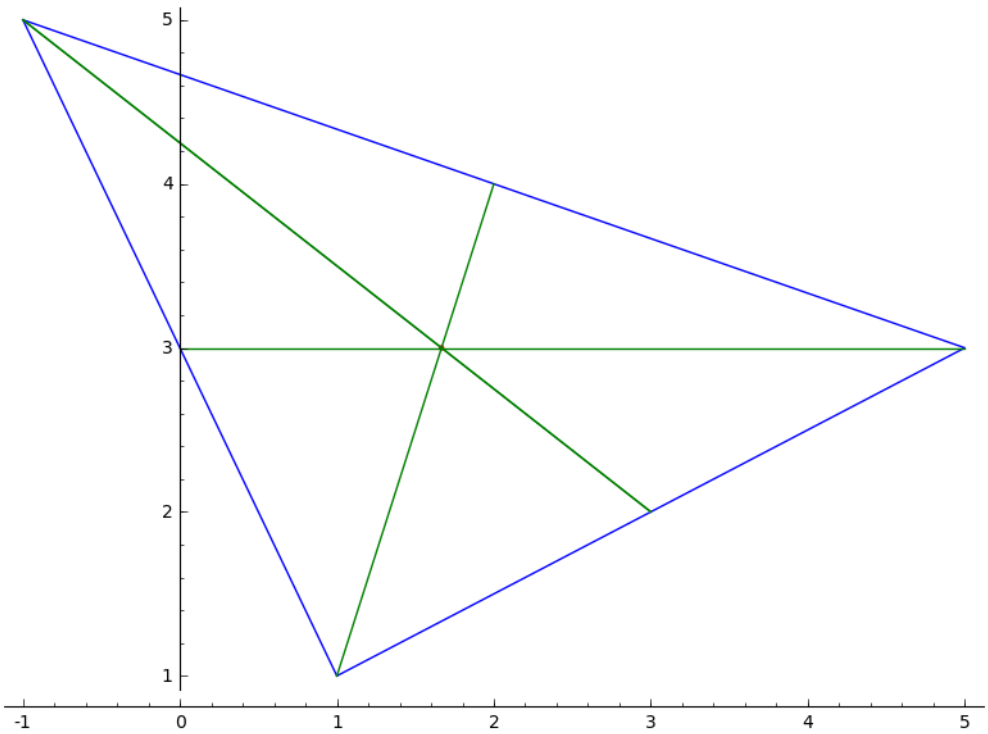
Out[23]:
```
[{t: 2/3, s: 2/3}]
```

In [24]:
```
S = sA.subs(st[0])
S
```

Out[24]:
```
(5/3, 3)
```

In [25]:
```
g += point(S, color = 'red')
g
```

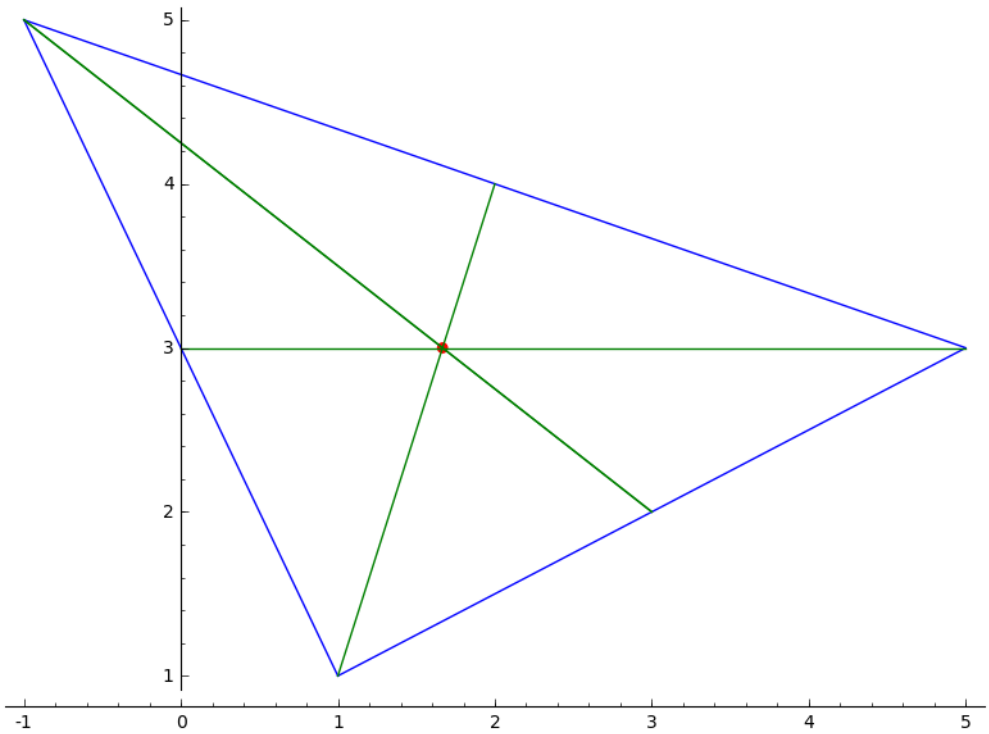
Out[25]:



Um ihn besser sichtbar zu machen, ein Kreis.

In [27]:
```
g=g+ point(S, size=40, color='red')
g
```

Out[27]:



Automatische Farben

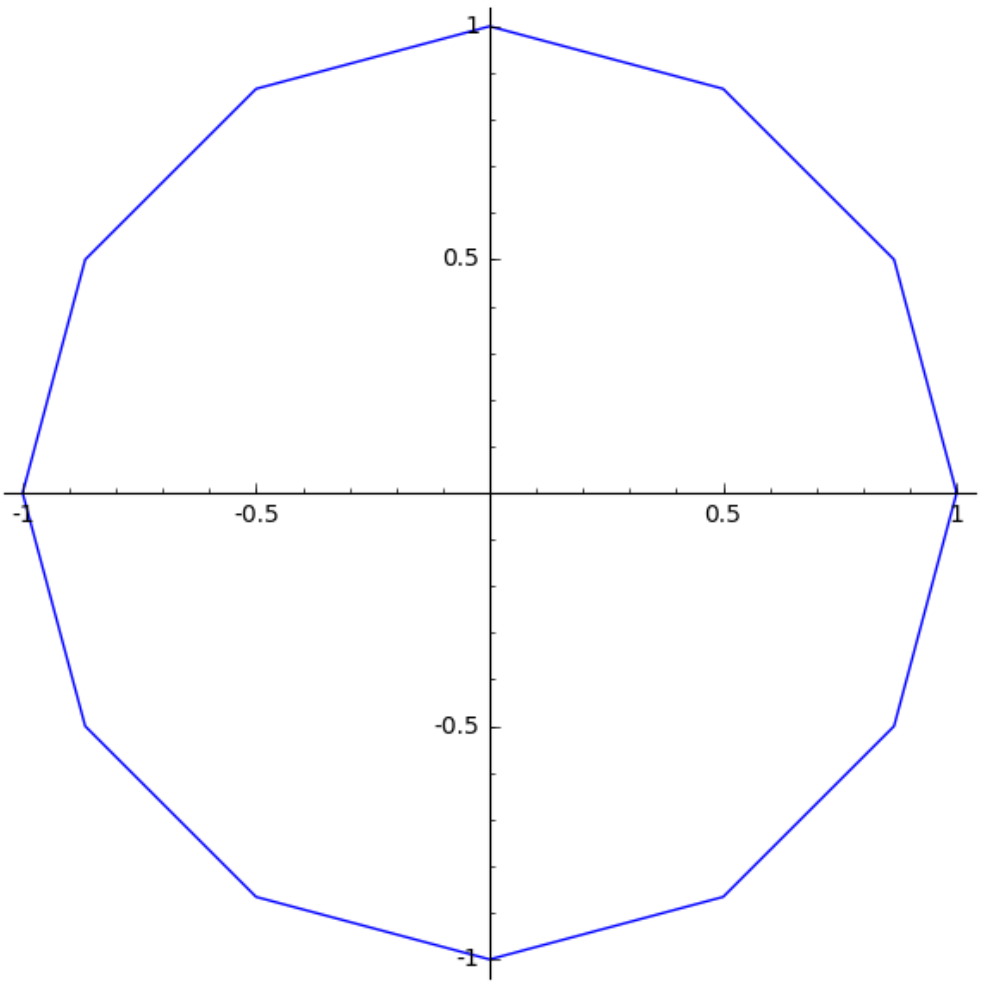
Am Beispiel eines n-Ecks

In [33]:
```
def ngon(n):
    return [[cos(2*pi*k/n),sin(2*pi*k/n)] for k in range(n+
1)]
```

In [34]:
```
ngon(12)
```

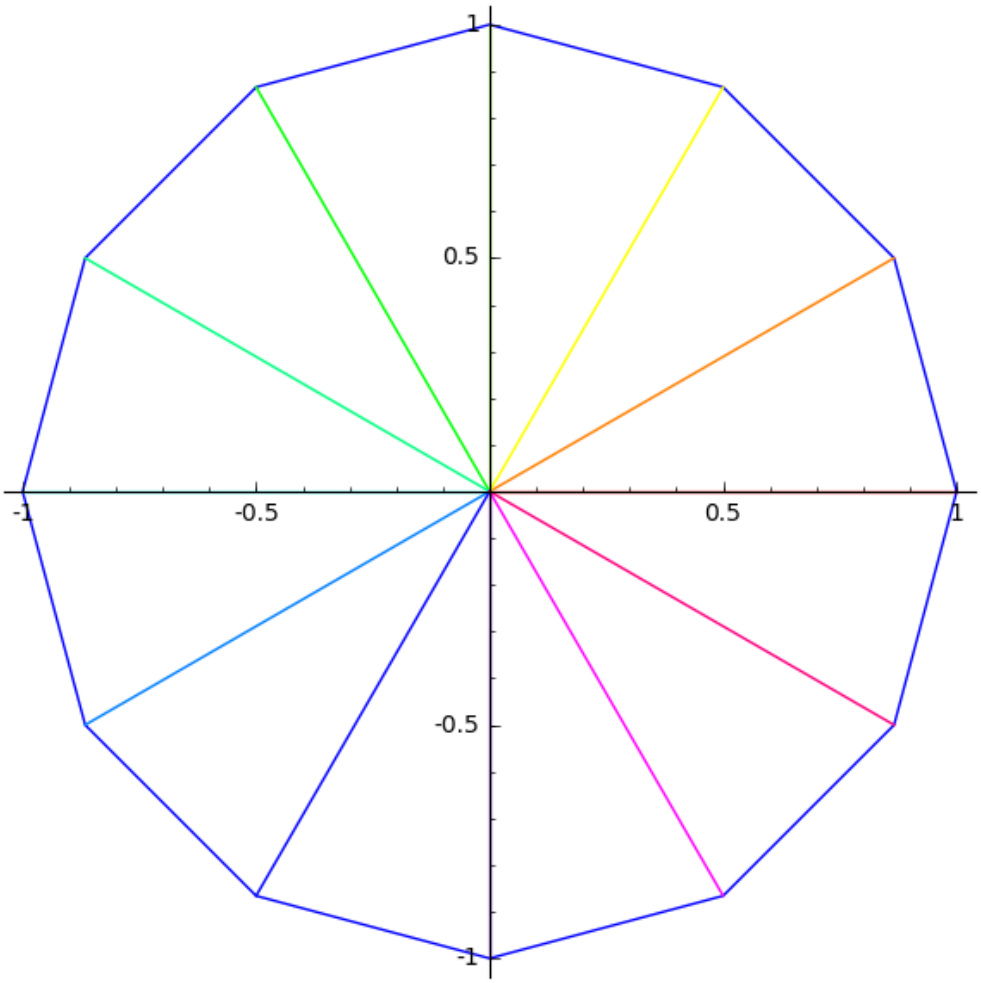
Out[34]:
```
[[1, 0],
 [1/2*sqrt(3), 1/2],
 [1/2, 1/2*sqrt(3)],
 [0, 1],
 [-1/2, 1/2*sqrt(3)],
 [-1/2*sqrt(3), 1/2],
 [-1, 0],
 [-1/2*sqrt(3), -1/2],
 [-1/2, -1/2*sqrt(3)],
 [0, -1],
 [1/2, -1/2*sqrt(3)],
 [1/2*sqrt(3), -1/2],
 [1, 0]]
```

In [40]:
```
g12 = line(ngon(12),aspect_ratio=1)
g12.show()
```

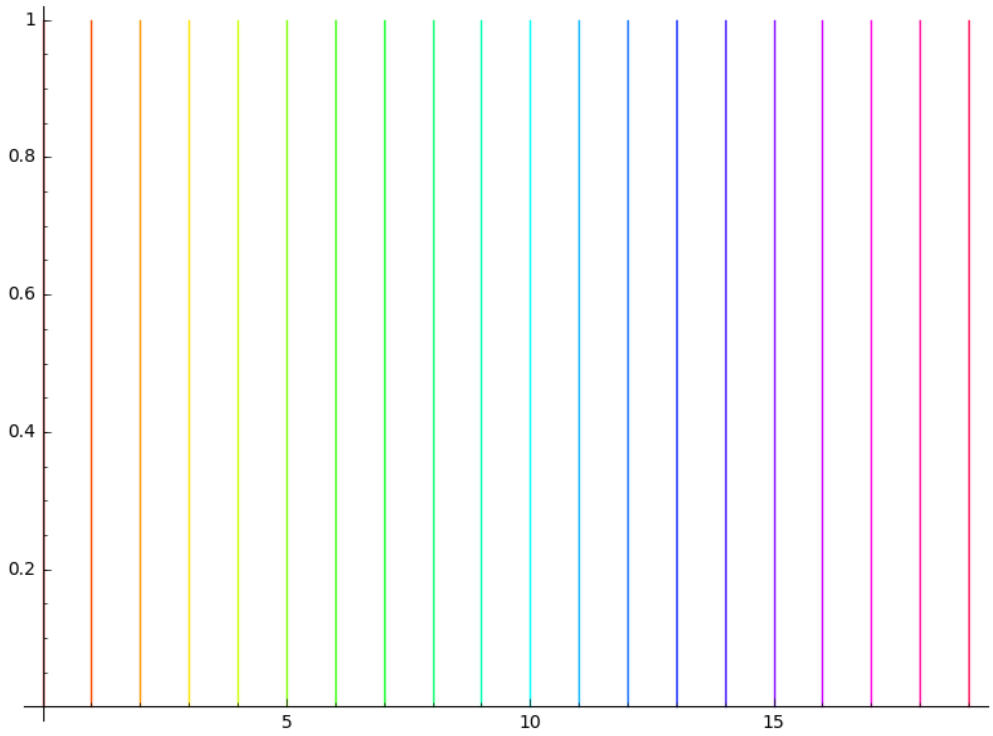


In [42]:
```
for k in range(12):
    g12 += line ([[0,0],[cos(2*pi*k/12),sin(2*pi*k/12)]], h
ue=k/12)
```

In [43]:
```
g12.show()
```



In [44]:
```
col=Graphics()
for k in range(20):
    col += line([[k,0],[k,1]],hue=k/20)
col.show()
```



Lineare Algebra: Matrizen und lineare Gleichungen.

Matrizen werden intern als Listen von Zeilen behandelt und sind entsprechend zu erzeugen:

In [45]:
```
A=matrix ( [[1,2,3], [4,5,6]])
A
```

Out[45]:
```
[1 2 3]
[4 5 6]
```

In [46]:
```
parent(A)
```

Out[46]:
```
Full MatrixSpace of 2 by 3 dense matrices over Integer Ring
```

In [47]:
```
A[1,1]
```

Out[47]:
```
5
```

Sage beginnt bei 0 zu zählen!

In [48]:
```
A[0,0]
```

Out[48]: 1

In [49]:
```
B=matrix(3,2 ,[ 1,2,3,4,5,6])
B
```

Out[49]:
```
[1 2]
[3 4]
[5 6]
```

Matrixprodukt mit *

In [50]:
```
A*B
```

Out[50]:
```
[22 28]
[49 64]
```

Große Matrizen können aus Funktionen erzeugt werden. Auch hier NB: sage beginnt bei 0 zu zählen!

In [51]:
```
n = 10
H  = matrix (n,n, lambda i,j: 1/(i+j+1))
H
```

Out[51]:
```
[   1  1/2  1/3  1/4  1/5  1/6  1/7  1/8  1/9 1/10]
[ 1/2  1/3  1/4  1/5  1/6  1/7  1/8  1/9 1/10 1/11]
[ 1/3  1/4  1/5  1/6  1/7  1/8  1/9 1/10 1/11 1/12]
[ 1/4  1/5  1/6  1/7  1/8  1/9 1/10 1/11 1/12 1/13]
[ 1/5  1/6  1/7  1/8  1/9 1/10 1/11 1/12 1/13 1/14]
[ 1/6  1/7  1/8  1/9 1/10 1/11 1/12 1/13 1/14 1/15]
[ 1/7  1/8  1/9 1/10 1/11 1/12 1/13 1/14 1/15 1/16]
[ 1/8  1/9 1/10 1/11 1/12 1/13 1/14 1/15 1/16 1/17]
[ 1/9 1/10 1/11 1/12 1/13 1/14 1/15 1/16 1/17 1/18]
[1/10 1/11 1/12 1/13 1/14 1/15 1/16 1/17 1/18 1/19]
```

Alternative mit einer *List Comprehension*

In [298]:
```
H  = matrix ([[1/(i+j+1) for j in range(n)] for i in range
(n)])
H
```

Out[298]:

Matrizen können auch symbolische Einträge (oder Elemente anderer Ringe) haben, allerdings müssen alle Einträge aus dem gleichen Ring kommen.

In [52]:
```
A = matrix (2,2,[1,x,0,1])
A
```

Out[52]:
```
[1 x]
[0 1]
```

In [53]:
```
parent(A)
```

Out[53]:
```
Full MatrixSpace of 2 by 2 dense matrices over Symbolic Rin
g
```

In [54]:
```
A*A
```

Out[54]:
```
[  1 2*x]
[  0   1]
```

In [55]:
```
A^2
```

Out[55]:
```
[  1 2*x]
[  0   1]
```

In [56]:
```
A^3
```

Out[56]:
```
[  1 3*x]
[  0   1]
```

In [57]:
```
var('k')
A^k
```

Out[57]:
```
[  1 k*x]
[  0   1]
```

In [58]:
```
identity_matrix(5)
```

Out[58]:
```
[1 0 0 0 0]
[0 1 0 0 0]
[0 0 1 0 0]
[0 0 0 1 0]
[0 0 0 0 1]
```

In [59]:
```
zero_matrix(2,5)
```

Out[59]:
```
[0 0 0 0 0]
[0 0 0 0 0]
```

Die in der Numerik wichtigen dünnbesetzten (engl. *sparse*) Matrizen kann man platzsparend mit dictionaries definieren.

```
In [60]: B = matrix (30, 30, {(0,0):1, (3,4):1})
         B
```

```
Out[60]: 30 x 30 sparse matrix over Integer Ring (use the '.str()' m
         ethod to see the entries)
```

```
In [72]: show(B)
```

$$\left(\begin{array}{cccccccccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right.$$

In [73]:
```
parent(B)
```

Out[73]:
```
Full MatrixSpace of 30 by 30 sparse matrices over Integer R
ing
```

In [74]:
```
B[0,0]
```

Out[74]:
```
1
```

In [75]:
```
B[0,1]
```

Out[75]:
```
0
```

Intern werden Matrizen als Listen von Zeilen gespeichert.

In [76]:
```
A[0]
```

Out[76]:
```
(1, x)
```

In [77]:
```
A.row(0)
```

Out[77]:
```
(1, x)
```

In [78]:
```
A.column(0)
```

Out[78]:
```
(1, 0)
```

Zuweisung einzelner Elemente ändert die ganze Matrix.

In [79]:
```
var('y')
A[1,1] = y
```

In [80]:
```
A
```

Out[80]:
```
[1 x]
[0 y]
```

In [81]:
```
A[0,0]=0.5
A
```

Out[81]:
```
[0.500000000000000                 x]
[                 0                 y]
```

In [82]:
```
A.parent()
```

Out[82]:
```
Full MatrixSpace of 2 by 2 dense matrices over Symbolic Rin
g
```

In [83]:
```
A[1,1]="a"
A
```

Out[83]:
```
[0.500000000000000                  x]
[                 0                 a]
```

In [84]:
```
A[1,1]
```

Out[84]:
```
a
```

Beim Zuordnen neuer Einträge darf der Bereich nicht verlassen werden.

In [85]:
```
B = matrix ([[1,2],[3,4]])
B.parent()
```

Out[85]:
```
Full MatrixSpace of 2 by 2 dense matrices over Integer Ring
```

In [86]:
```
B[0,0]=1/2
```

```
---------------------------------------------------------------
---------------
TypeError                                 Traceback (most r
ecent call last)
<ipython-input-86-b6b1b8fa0708> in <module>()
----> 1 B[Integer(0),Integer(0)]=Integer(1)/Integer(2)

/usr/opt/Sage-8.3-amd64/local/lib/python2.7/site-packages/s
age/matrix/matrix0.pyx in sage.matrix.matrix0.Matrix.__seti
tem__ (build/cythonized/sage/matrix/matrix0.c:8914)()
   1421
   1422         if single_row and single_col and not no_col
_index:
-> 1423             self.set_unsafe(row, col, self._coerce_
element(value))
   1424             return
   1425

/usr/opt/Sage-8.3-amd64/local/lib/python2.7/site-packages/s
age/matrix/matrix0.pyx in sage.matrix.matrix0.Matrix._coerc
e_element (build/cythonized/sage/matrix/matrix0.c:10391)()
   1526         if isinstance(x, Element) and (<Element>
x)._parent is self._base_ring:
   1527             return x
-> 1528         return self._base_ring(x)
   1529
   1530     ##############################################
############

/usr/opt/Sage-8.3-amd64/local/lib/python2.7/site-packages/s
age/structure/parent.pyx in sage.structure.parent.Parent.__
call__ (build/cythonized/sage/structure/parent.c:9727)()
    918         if mor is not None:
    919             if no_extra_args:
--> 920                 return mor._call_(x)
    921             else:
    922                 return mor._call_with_args(x, args,
kwds)

/usr/opt/Sage-8.3-amd64/local/lib/python2.7/site-packages/s
age/rings/rational.pyx in sage.rings.rational.Q_to_Z._call_
(build/cythonized/sage/rings/rational.c:35641)()
   4186         """
   4187         if not mpz_cmp_si(mpq_denref((<Rational>x).
value), 1) == 0:
-> 4188             raise TypeError("no conversion of this
rational to integer")
   4189         cdef Integer n = Integer.__new__(Integer)
   4190         n.set_from_mpz(mpq_numref((<Rational>x).val
ue))

TypeError: no conversion of this rational to integer
```

Dazu muß die Matrix zuerst umgewandelt werden.

In [217]:
```
B1 = matrix(QQ,B)
B1
```

Out[217]:
```
[1 2]
[3 4]
```

In [218]:
```
parent(B1)
```

Out[218]:
```
Full MatrixSpace of 2 by 2 dense matrices over Rational Fie
ld
```

In [219]:
```
B1[0,0]=1/2
B1
```

Out[219]:
```
[1/2   2]
[  3   4]
```

Matlab-Notation

In [97]:
```
A[:,1]
```

Out[97]:
```
[x]
[a]
```

In [98]:
```
A[1,:]
```

Out[98]:
```
[0 a]
```

Zurück zur Hilbertmatrix. Wir lösen eine lineare Gleichung $Hx = b$. Zunächst die rechte Seite als einspaltige Matrix.

In [99]:
```
b = matrix(n,1, lambda i,j:1/(i+1)^2)
b
```

Out[99]:
```
[    1]
[  1/4]
[  1/9]
[ 1/16]
[ 1/25]
[ 1/36]
[ 1/49]
[ 1/64]
[ 1/81]
[1/100]
```

Mehrere Möglichkeiten.

In [100]:
```
H^-1*b
```

Out[100]:
```
[ 1451/252]
[      -99]
[     1188]
[    -8008]
[  63063/2]
[-378378/5]
[   112112]
[-700128/7]
[ 196911/4]
[ -92378/9]
```

In [101]:
```
H\b
```

Out[101]:
```
[ 1451/252]
[      -99]
[     1188]
[    -8008]
[  63063/2]
[-378378/5]
[   112112]
[-700128/7]
[ 196911/4]
[ -92378/9]
```

In [102]:
```
H.inverse()*b
```

Out[102]:
```
[ 1451/252]
[      -99]
[     1188]
[    -8008]
[  63063/2]
[-378378/5]
[   112112]
[-700128/7]
[ 196911/4]
[ -92378/9]
```

In [103]:
```
H.solve_right(b)
```

Out[103]:
```
[ 1451/252]
[      -99]
[     1188]
[    -8008]
[  63063/2]
[-378378/5]
[   112112]
[-700128/7]
[ 196911/4]
[ -92378/9]
```

Oder als Vektor.

In [104]:
```
b = vector ( [ 1/i^2 for i in [1..n] ])
b
```

Out[104]:
```
(1, 1/4, 1/9, 1/16, 1/25, 1/36, 1/49, 1/64, 1/81, 1/100)
```

In [105]:
```
H^-1*b
```

Out[105]:
```
(1451/252, -99, 1188, -8008, 63063/2, -378378/5, 112112, -7
00128/7, 196911/4, -92378/9)
```

In [106]:
```
H.solve_right(b)
```

Out[106]:
```
(1451/252, -99, 1188, -8008, 63063/2, -378378/5, 112112, -7
00128/7, 196911/4, -92378/9)
```

In [107]:
```
H\b
```

Out[107]:
```
(1451/252, -99, 1188, -8008, 63063/2, -378378/5, 112112, -7
00128/7, 196911/4, -92378/9)
```

Vorsicht bei singulären Matrizen.

In [108]:
```
A = matrix( [[1,2,3], [2,4,5], [0,0,1]] )
A
```

Out[108]:
```
[1 2 3]
[2 4 5]
[0 0 1]
```

In [109]:
```
A.rank()
```

Out[109]:
```
2
```

Es werden nicht alle Lösungen geliefert!

In [110]:
```
A \ vector([0,0,0])
```

Out[110]:
```
(0, 0, 0)
```

Um alle Lösungen zu erhalten, muß man den Kern der Matrix kennen.

In [111]:
```
A.right_kernel()
```

Out[111]:
```
Free module of degree 3 and rank 1 over Integer Ring
Echelon basis matrix:
[ 2 -1  0]
```

Das ist ein Vektorraum. (In diesem Fall nur ein Modul, weil die ganzen Zahlen keinen Körper bilden)

Lineare Algebra: Vektorräume

In [127]:
```
V = QQ^5
V
```

Out[127]:
```
Vector space of dimension 5 over Rational Field
```

Erzeugen von Elementen.

In [128]:
```
v1 = V([1,1,1,0,0])
v2 = V([1,-1,1,0,0])
v3 = V([1,0,1,0,0])
```

In [129]:
```
v1
```

Out[129]:
```
(1, 1, 1, 0, 0)
```

In [130]:
```
parent(v1)
```

Out[130]:
```
Vector space of dimension 5 over Rational Field
```

Der von den Vektoren erzeugte Unterraum. Es wird sofort eine Basis gebildet.

In [131]:
```
U = V.subspace([v1,v2,v3])
U
```

Out[131]:
```
Vector space of degree 5 and dimension 2 over Rational Fiel
d
Basis matrix:
[1 0 1 0 0]
[0 1 0 0 0]
```

In [132]:
```
U.basis()
```

Out[132]:
```
[
(1, 0, 1, 0, 0),
(0, 1, 0, 0, 0)
]
```

Feststellen, ob ein Vektor im Unterraum liegt.

In [133]:
```
v4 = V([0,0,0,1,0])
v4 in U
```

Out[133]:
```
False
```

In [134]:
```
v1 in U
```

Out[134]:
```
True
```

Bestimmung der Koordinaten bezüglich der Basis von U.

In [135]:
```
U.coordinates(v1)
```

Out[135]:
```
[1, 1]
```

Manipulation von Unterräumen (Summe, Durchschnitt, Vergleich).

In [136]:
```
W = V.subspace([V(v4)])
W
```

Out[136]:
```
Vector space of degree 5 and dimension 1 over Rational Fiel
d
Basis matrix:
[0 0 0 1 0]
```

In [137]:
```
W.is_subspace(U)
```

Out[137]:
```
False
```

In [138]:
```
W.is_subspace(V)
```

Out[138]:
```
True
```

In [139]:
```
U+W
```

Out[139]:
```
Vector space of degree 5 and dimension 3 over Rational Fiel
d
Basis matrix:
[1 0 1 0 0]
[0 1 0 0 0]
[0 0 0 1 0]
```

In [140]:
```
U.is_subspace(U+W)
```

Out[140]:
```
True
```

In [141]:
```
U.intersection(W)
```

Out[141]:
```
Vector space of degree 5 and dimension 0 over Rational Fiel
d
Basis matrix:
[]
```

Wir haben oben den Kern einer Matrix betrachtet.

In [142]:
```
A=matrix([[1,2,3],[2,4,5],[0,0,1]])
A
```

Out[142]:
```
[1 2 3]
[2 4 5]
[0 0 1]
```

In [143]:
```
A.right_kernel()
```

Out[143]:
```
Free module of degree 3 and rank 1 over Integer Ring
Echelon basis matrix:
[ 2 -1  0]
```

Wir verpflanzen die Matrix in einen Körper.

In [147]:
```
A1 = matrix(QQ, A)
parent(A1)
```

Out[147]:
```
Full MatrixSpace of 3 by 3 dense matrices over Rational Fie
ld
```

In [148]:
```
A1.right_kernel()
```

Out[148]:
```
Vector space of degree 3 and dimension 1 over Rational Fiel
d
Basis matrix:
[   1 -1/2    0]
```

Der Bildraum.

In [149]:
```
A1.image()
```

Out[149]:
```
Vector space of degree 3 and dimension 2 over Rational Fiel
d
Basis matrix:
[1 2 0]
[0 0 1]
```

Vorsicht, das ist der Bildraum der Abbildung $x \mapsto xA$, die Zeilenvektoren auf Zeilenvektoren abbildet!

In [150]:
```
A1.row_space()
```

Out[150]:
```
Vector space of degree 3 and dimension 2 over Rational Fiel
d
Basis matrix:
[1 2 0]
[0 0 1]
```

Um den Bildraum der Abbildung $x \mapsto Ax$ zu bekommen, muß man den Spaltenraum erzeugen:

In [151]:
```
A1.column_space()
```

Out[151]:
```
Vector space of degree 3 and dimension 2 over Rational Fiel
d
Basis matrix:
[ 1  0  2]
[ 0  1 -1]
```

In diesem Fall sind Kern und Bildraum komplementär.

In [152]:
```
A1.right_kernel() + A1.column_space()
```

Out[152]:
```
Vector space of degree 3 and dimension 3 over Rational Fiel
d
Basis matrix:
[1 0 0]
[0 1 0]
[0 0 1]
```

Das ganze geht auch über endlichen Körpern.

In [153]:
```
K = GF(3)
K
```

Out[153]:
```
Finite Field of size 3
```

In [154]:
```
V = K^3
V
```

Out[154]:
```
Vector space of dimension 3 over Finite Field of size 3
```

Auch der Vektorraum hat nur endlich viele Elemente.

In [155]:
```
V.list()
```

Out[155]:
```
[(0, 0, 0),
 (1, 0, 0),
 (2, 0, 0),
 (0, 1, 0),
 (1, 1, 0),
 (2, 1, 0),
 (0, 2, 0),
 (1, 2, 0),
 (2, 2, 0),
 (0, 0, 1),
 (1, 0, 1),
 (2, 0, 1),
 (0, 1, 1),
 (1, 1, 1),
 (2, 1, 1),
 (0, 2, 1),
 (1, 2, 1),
 (2, 2, 1),
 (0, 0, 2),
 (1, 0, 2),
 (2, 0, 2),
 (0, 1, 2),
 (1, 1, 2),
 (2, 1, 2),
 (0, 2, 2),
 (1, 2, 2),
 (2, 2, 2)]
```

1. Man kann auch direkt darüber iterieren, ohne die Liste zu erzeugen (dahinter steckt ein sogenannter *Generator*, mehr dazu später).

In [156]:
```
for v in V:
    print v
```

```
(0, 0, 0)
(1, 0, 0)
(2, 0, 0)
(0, 1, 0)
(1, 1, 0)
(2, 1, 0)
(0, 2, 0)
(1, 2, 0)
(2, 2, 0)
(0, 0, 1)
(1, 0, 1)
(2, 0, 1)
(0, 1, 1)
(1, 1, 1)
(2, 1, 1)
(0, 2, 1)
(1, 2, 1)
(2, 2, 1)
(0, 0, 2)
(1, 0, 2)
(2, 0, 2)
(0, 1, 2)
(1, 1, 2)
(2, 1, 2)
(0, 2, 2)
(1, 2, 2)
(2, 2, 2)
```

Entsprechendes geht auch für die rationalen Zahlen, allerdings bricht die Schleife nicht ab, wenn man keine Vorsorge trifft.

In [157]:
```
k=0
for v in QQ^3:
   if k > 20:
       break
   k += 1
   print v
```

```
(0, 0, 0)
(1, 0, 0)
(-1, 0, 0)
(1/2, 0, 0)
(-1/2, 0, 0)
(2, 0, 0)
(-2, 0, 0)
(1/3, 0, 0)
(-1/3, 0, 0)
(3, 0, 0)
(-3, 0, 0)
(2/3, 0, 0)
(-2/3, 0, 0)
(3/2, 0, 0)
(-3/2, 0, 0)
(1/4, 0, 0)
(-1/4, 0, 0)
(4, 0, 0)
(-4, 0, 0)
(3/4, 0, 0)
(-3/4, 0, 0)
```

In [158]:
```
(QQ^3).list()
```

```
---------------------------------------------------------------
---------------
AttributeError                            Traceback (most r
ecent call last)
<ipython-input-158-6704d9b8d769> in <module>()
----> 1 (QQ**Integer(3)).list()

/usr/opt/Sage-8.3-amd64/local/lib/python2.7/site-packages/s
age/structure/category_object.pyx in sage.structure.categor
y_object.CategoryObject.__getattr__ (build/cythonized/sage/
structure/category_object.c:7996)()
    853             AttributeError: 'PrimeNumbers_with_cate
gory' object has no attribute 'sadfasdf'
    854         """
--> 855         return self.getattr_from_category(name)
    856
    857     cdef getattr_from_category(self, name):

/usr/opt/Sage-8.3-amd64/local/lib/python2.7/site-packages/s
age/structure/category_object.pyx in sage.structure.categor
y_object.CategoryObject.getattr_from_category (build/cython
ized/sage/structure/category_object.c:8159)()
    868                 cls = self._category.parent_class
    869
--> 870             attr = getattr_from_other_class(self, c
ls, name)
    871             self.__cached_methods[name] = attr
    872             return attr

/usr/opt/Sage-8.3-amd64/local/lib/python2.7/site-packages/s
age/cpython/getattr.pyx in sage.cpython.getattr.getattr_fro
m_other_class (build/cythonized/sage/cpython/getattr.c:246
8)()
    387         dummy_error_message.cls = type(self)
    388         dummy_error_message.name = name
--> 389         raise AttributeError(dummy_error_message)
    390     cdef PyObject* attr = instance_getattr(cls, nam
e)
    391     if attr is NULL:

AttributeError: 'FreeModule_ambient_field_with_category' ob
ject has no attribute 'list'
```

Zurück zum endlichen Vektorraum. Auch die Anzahl der Unterräume ist endlich.

In [159]:
```
V
```

Out[159]:
```
Vector space of dimension 3 over Finite Field of size 3
```

In [160]:
```
V2=V.subspaces(2)
V2
```

Out[160]:
```
<generator object subspaces at 0x7f4ef8110be0>
```

In [161]:
```
V2l = list(V2)
V2l
```

```
Out[161]: [Vector space of degree 3 and dimension 2 over Finite Field
          of size 3
           Basis matrix:
           [1 0 0]
           [0 1 0], Vector space of degree 3 and dimension 2 over Fin
          ite Field of size 3
           Basis matrix:
           [1 0 0]
           [0 1 1], Vector space of degree 3 and dimension 2 over Fin
          ite Field of size 3
           Basis matrix:
           [1 0 0]
           [0 1 2], Vector space of degree 3 and dimension 2 over Fin
          ite Field of size 3
           Basis matrix:
           [1 0 1]
           [0 1 0], Vector space of degree 3 and dimension 2 over Fin
          ite Field of size 3
           Basis matrix:
           [1 0 1]
           [0 1 1], Vector space of degree 3 and dimension 2 over Fin
          ite Field of size 3
           Basis matrix:
           [1 0 1]
           [0 1 2], Vector space of degree 3 and dimension 2 over Fin
          ite Field of size 3
           Basis matrix:
           [1 0 2]
           [0 1 0], Vector space of degree 3 and dimension 2 over Fin
          ite Field of size 3
           Basis matrix:
           [1 0 2]
           [0 1 1], Vector space of degree 3 and dimension 2 over Fin
          ite Field of size 3
           Basis matrix:
           [1 0 2]
           [0 1 2], Vector space of degree 3 and dimension 2 over Fin
          ite Field of size 3
           Basis matrix:
           [1 0 0]
           [0 0 1], Vector space of degree 3 and dimension 2 over Fin
          ite Field of size 3
           Basis matrix:
           [1 1 0]
           [0 0 1], Vector space of degree 3 and dimension 2 over Fin
          ite Field of size 3
           Basis matrix:
           [1 2 0]
           [0 0 1], Vector space of degree 3 and dimension 2 over Fin
          ite Field of size 3
           Basis matrix:
           [0 1 0]
           [0 0 1]]
```

In [162]:
```
len(V2l)
```

Out[162]: 13

Generatoren

Ein *Generator* ist ein Spezialfall eines Iterators (http://de.wikipedia.org/wiki/Iterator). Wichtige Anwendungen sind: 'lazy evaluation', Schleifen, elegantes Aufsammeln von Zwischenergebnissen und insbesondere rekursive Erzeugungmathematischer Objekte.

'lazy evaluation'

Praktisch gesehen ist ein Generator eine Art Liste, deren Elemente spontan bei Bedarf erzeugt werden und dann verfallen. Einfache Generatoren kann man ähnlich wie Listen erzeugen, mit runden anstatt eckigen Klammern.

In [163]:
```
g = (1..4)
g
```

Out[163]: <generator object at 0x7f4ef8134c30>

Der Generator wird duch die Methode **.next()** in Bewegung versetzt. Er gibt daraufhin einen Wert aus und bleibt bis zum nächsten Aufruf von **.next()** stehen (mehr dazu unten). Von außen betrachtet verhält er sich wie ein Kaugummiautomat, wobei **.next()** dem Einwurf einer Münze entspricht.

In [164]:
```
g.next()
```

Out[164]: 1

In [165]:
```
g.next()
```

Out[165]: 2

In [166]:
```
g.next()
```

Out[166]: 3

In [167]:
```
g.next()
```

Out[167]: 4

Wenn alle Werte ausgegeben sind, bricht der Generator ab.

```
In [168]: g.next()
```

```
---------------------------------------------------------------
---------------
StopIteration                             Traceback (most r
ecent call last)
<ipython-input-168-7dbbdfed0980> in <module>()
----> 1 g.next()

StopIteration:
```

So wie ein Kaugummi nicht von alleine in den Automaten zurückkehrt, sondern letzterer neu befüllt werden muß, kann ein Generator kann nur einmal von vorne nach hinten durchlaufen werden und nicht in einen vorhergehenden Zustand zurückversetzt werden. Dafür muß ein neuer Generator initialisiert werden.

```
In [169]: g = (1..4)
```

```
In [170]: g.next()
Out[170]: 1
```

Man kann auch alle Kaugummis auf einen Schlag herausholen durch direke Umwandlung in eine Liste:

```
In [171]: list(g)
Out[171]: [2, 3, 4]
```

```
In [172]: g.next()
```

```
---------------------------------------------------------------
---------------
StopIteration                             Traceback (most r
ecent call last)
<ipython-input-172-7dbbdfed0980> in <module>()
----> 1 g.next()

StopIteration:
```

Schleifen

Generatoren spielen eine wichtige Rolle in Schleifen, insbesondere, wenn dabei sonst eine lange Liste erzeugt werden müßte.

In [173]:
```
for i in (1..4):
    print i
```

```
1
2
3
4
```

Ein Generator kann auch unendlich groß sein. In diesem Fall ist es nicht ratsam, ihn in eine Liste umzuwandeln, weil sage nicht von vornherein erkennen kann, ob ein Generator nach endlich vielen Schritten abbricht oder nicht. (Dieses Frage ist nach A.Turing algorithmisch nicht entscheidbar).

In [174]:
```
nn = (1..)
```

In [175]:
```
nn.next()
```

Out[175]: 1

In [176]:
```
nn.next()
```

Out[176]: 2

In [177]:
```
for i in nn:
    print i
    if i > 10:
        break
```

```
3
4
5
6
7
8
9
10
11
```

In [178]:
```
nn.next()
```

Out[178]: 12

Aus bereits vorhandenen können mit *List Comprehension* neue Iteratoren erzeugt werden.

In [179]:
```
a = (1..5)
```

In [180]:
```
b = (i^2 for i in a)
b
```

Out[180]:
```
<generator object <genexpr> at 0x7f4ef8109d20>
```

dabei läuft im Hintergrund der ursprüngliche Generator mit:

In [181]:
```
a.next()
```

Out[181]:
```
1
```

In [182]:
```
b.next()
```

Out[182]:
```
4
```

In [183]:
```
a.next()
```

Out[183]:
```
3
```

In [184]:
```
b.next()
```

Out[184]:
```
16
```

In [185]:
```
a.next()
```

Out[185]:
```
5
```

In [186]:
```
b.next()
```

```
---------------------------------------------------------------
---------------
StopIteration                             Traceback (most r
ecent call last)
<ipython-input-186-2742f26def7c> in <module>()
----> 1 b.next()

StopIteration:
```

In [187]:
```
a.next()
```

```
---------------------------------------------------------------
---------------
StopIteration                             Traceback (most r
ecent call last)
<ipython-input-187-f3a3911740c6> in <module>()
----> 1 a.next()

StopIteration:
```

ansonsten läßt sich ein Generator wie eine Liste behandeln.

In [188]:
```
reduce(operator.add, (1..5))
```

Out[188]: 15

Generatoren Programmieren

Ein Generator kann wie eine Funktion programmiert werden, indem anstelle von **return** der Befehl **yield** gesetzt wird. Letzterer hat zwei Funktionen:

- Anhalten der Funktion
- Zurückgeben eines Werts.

Dabei wird der aktuelle Zustand festgehalten und beim nächsten Aufruf von `.next()` an dieser Stelle fortgesetzt bis zum nächsten `yield` oder bis zum Ende des Codes.

In [189]:
```
def abc():
    yield 'a'
    yield 'b'
    yield 'c'
```

Dabei ist zu beachten, daß **abc()** nicht der Generator an sich ist, sondern eine Funktion, die bei jedem Aufruf neuen Generator erzeugt und initialisiert.

In [190]:
```
g = abc()
g
```

Out[190]: <generator object abc at 0x7f4ef8109b90>

Der Aufruf von **.next()** bewirkt, daß der Generator gestartet wird und bis zum ersten **yield** läuft. Der entsprechende Wert wird zurückgegeben und der Generator hält an.

In [191]:
```
g.next()
```

Out[191]: 'a'

Beim zweiten Aufruf macht er an der gleichen Stelle weiter, bis er auf das nächste **yield** trifft.

In [192]:
```
g.next()
```

Out[192]: 'b'

usw.

In [193]:
```
g.next()
```

Out[193]:
```
'c'
```

Bis das Ende des Codes erreicht wird.

In [194]:
```
g.next()
```

```
---------------------------------------------------------------
---------------
StopIteration                             Traceback (most r
ecent call last)
<ipython-input-194-7dbbdfed0980> in <module>()
----> 1 g.next()

StopIteration:
```

Will man einen Generator unter gewissen Bedingungen vorzeitig abbrechen, wird an der entsprechenden Stelle ein **return** eingefügt.

In [195]:
```
def abc(cnicht):
    yield 'a'
    yield 'b'
    if cnicht:
        return
    yield 'c'
```

Ist die Bedingung erfüllt, bricht der Generator an der entsprechenden Stelle ab.

In [196]:
```
g = abc(True)
```

In [197]:
```
g.next()
```

Out[197]:
```
'a'
```

In [198]:
```
g.next()
```

Out[198]:
```
'b'
```

```
In [199]: g.next()
```

```
---------------------------------------------------------------
---------------
StopIteration                             Traceback (most r
ecent call last)
<ipython-input-199-7dbbdfed0980> in <module>()
----> 1 g.next()

StopIteration:
```

Wenn nicht, wird **return** übersprungen.

```
In [200]: g = abc(False)
```

```
In [201]: g.next()
Out[201]: 'a'
```

```
In [202]: g.next()
Out[202]: 'b'
```

```
In [203]: g.next()
Out[203]: 'c'
```

Noch einmal kurz

```
In [204]: [t for t in abc(True)]
Out[204]: ['a', 'b']
```

```
In [205]: [t for t in abc(False)]
Out[205]: ['a', 'b', 'c']
```

Aufsammeln von Ergebnissen

Generatoren sind auch nützlich, um im Verlauf von Rechnungen Zwischenergebnisse auszuwerfen, ohne dabei explizit eine Liste mitzuführen. Wir betrachten noch einmal das Primzahlzwillingsproblem aus der Übung. **p** durchläuft alle Primzahlen kleiner als die obere Schranke **n** und jedes Mal, wenn ein Zwilling auftaucht, wird das Paar ausgeworfen.

In [206]:
```
def primzw(n=oo):
    p=3
    while p<n:
        q = next_prime(p)
        if q == p+2:
            yield [p,q]
        p=q
```

Der Vorteil ist Übersichtlichkeit und geringstmöglicher Speicherbedarf (allerdings auf Kosten der Laufzeit).

In [207]:
```
list(primzw(100))
```

Out[207]:
```
[[3, 5], [5, 7], [11, 13], [17, 19], [29, 31], [41, 43], [5
9, 61], [71, 73]]
```

Dabei kann im Prinzip die Anzahl der Zwillinge offen gelassen werden:

In [208]:
```
pp = primzw(oo)
```

Um zum Beispiel die ersten 10 Paare zu erzeugen, schreiben wir einen zweiten Generator um den ersten herum, der die Paare mitzählt:

In [209]:
```
def firstn(n,g):
    for i in range(n):
        yield g.next()
```

Und wir bekommen das gewünschte Resultat ohne darüber nachdenken zu müssen, wie groß wir die obere Grenze wählen müssen.

In [210]:
```
list(firstn(10,pp))
```

Out[210]:
```
[[3, 5],
 [5, 7],
 [11, 13],
 [17, 19],
 [29, 31],
 [41, 43],
 [59, 61],
 [71, 73],
 [101, 103],
 [107, 109]]
```

Rekursive Generatoren

Besonders elegant werden Generatoren, wenn sie rekursiv angewandt werden. Als Beispiel erzeugen wir rekursiv alle Elemente der kartesischen Potenz einer Menge. Die kartesische Potenz kann induktiv definiert werden, was sich unmittelbar in einen Generator übersetzen läßt:

In [211]:
```
def cart(S,n):
    if n == 1:
        for s in S:
            yield [s]
    else:
        for s in S:
            for p in cart(S,n-1):
                yield [s]+p
```

In [212]:
```
list(cart([1,2],4))
```

Out[212]:
```
[[1, 1, 1, 1],
 [1, 1, 1, 2],
 [1, 1, 2, 1],
 [1, 1, 2, 2],
 [1, 2, 1, 1],
 [1, 2, 1, 2],
 [1, 2, 2, 1],
 [1, 2, 2, 2],
 [2, 1, 1, 1],
 [2, 1, 1, 2],
 [2, 1, 2, 1],
 [2, 1, 2, 2],
 [2, 2, 1, 1],
 [2, 2, 1, 2],
 [2, 2, 2, 1],
 [2, 2, 2, 2]]
```

Ganz analog kann man alle Permutationen der Elemente einer Liste erzeugen, indem jeweils ein Element an die Spitze gestellt und der Rest permutiert wird.

In [213]:
```
def perm(S):
    n = len(S)
    if n==0: yield []
    else:
        for i in range(n):
            for p in perm(S[:i]+S[i+1:]):
                yield [S[i]] + p
```

In [214]:
```
list(perm([1,2,3]))
```

Out[214]:
```
[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3,
2, 1]]
```

Einfacher zu lesen und zu Programmieren wird es mit **Set**:

In [215]:
```
def permset(S):
    S1=Set(S)
    n = len(S1)
    if n == 0:
        yield []
    else:
        for s in S1:
            for p in perm(S1.difference([s])):
                yield [s]+p
```

In [216]:
```
list(permset([1,2,3]))
```

Out[216]:
```
[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3,
2, 1]]
```