lsg05

Aufgabe 1

```
reset()
var('x')
L1 = [x^(2*n-1) for n in [1..5]]
L2 = [x^(n.factorial()) for n in [1..5]]
L3 = L1 + L2
```

```
[f.diff() for f in L3]
```

```
[1, 3*x^2, 5*x^4, 7*x^6, 9*x^8, 1, 2*x, 6*x^5, 24*x^23, 120*x^119]
```

```
[f.integrate() for f in L3]
```

```
[1/2*x^2, 1/4*x^4, 1/6*x^6, 1/8*x^8, 1/10*x^10, 1/2*x^2, 1/3*x^3,
1/7*x^7, 1/25*x^25, 1/121*x^121]
```

```
L = [f(x=3) for f in L3]; L
```

```
[3, 27, 243, 2187, 19683, 3, 9, 729, 282429536481,
1797010299914431210413179829509605039731475627537851106401]
```

```
m = max(L)
L.remove(m); L
```

```
[3, 27, 243, 2187, 19683, 3, 9, 729, 282429536481]
```

Aufgabe 2

```
reset()
L = [16, 81, 125, 512, 729, 4096, 19683, 78125, 262144, 390625, 505,
22343243, 512]
```

```
L = [n for n in L if n%3 != 0]; L
```

```
[16, 125, 512, 4096, 78125, 262144, 390625, 505, 22343243, 512]
```

```
L = [n for n in L if n%2 != 0]; L
```

```
[125, 78125, 390625, 505, 22343243]
```

```
L = [n for n in L if n%5 == 0]; L
```

```
[125, 78125, 390625, 505]
```

Aufgabe 3

```
reset()
var('x')
A = matrix([[19,-2,4],[4,10,-2],[4,-8,25]])
B = matrix([[1,-3,3],[3,-5,3],[6,-6,4]])
C = matrix([[-3,1,-1],[-7,5,-1],[-6,6,-2]])
E = identity_matrix(3)
Ms = [A,B,C]
```

```
show( [(M-x*E).det() for M in Ms] )
```

$$\left[-(x-19)((x-25)(x-10)-16)+8x-72, -(x-1)((x-4)(x+5)\right.$$

```
[M.eigenvalues() for M in Ms]
```

```
[[27, 18, 9], [4, -2, -2], [4, -2, -2]]
```

```
show( A.eigenvectors_right() )
```

$$\left[(27,[(1,0,2)],1),\left(18,\left[\left(1,\frac{1}{2},0\right)\right],1\right),\left(9,\left[\left(0,1,\frac{1}{2}\right)\right],1\right)\right.$$

```
show( B.eigenvectors_right() )
```

$$[(4,[(1,1,2)],1),(-2,[(1,0,-1),(0,1,1)],2)]$$

```
show( C.eigenvectors_right() )
```

$$[(4,[(0,1,1)],1),(-2,[(1,1,0)],2)]$$

```
def diagonalisierbar(X):
   for E in X.eigenvectors_right():
       geom_vielfachheit = len(E[1])
       alg_vielfachheit = E[2]
       if geom_vielfachheit != alg_vielfachheit:
           return False
   return True
```

```
[diagonalisierbar(M) for M in Ms]
```

```
[True, True, False]
```

```
var('a')
D = matrix([[cos(a), -sin(a)], [sin(a), cos(a)]]); D
```

```
[ cos(a) -sin(a)]
[ sin(a)  cos(a)]
```

D.eigenvalues()

```
[-I*sin(a) + cos(a), I*sin(a) + cos(a)]
```

Also reelle Eigenwerte für a=0 und a=pi bzw. a=k*pi, k in N.

Also bei einer Punktspiegelung um den Ursprung oder der Selbstabbildung.

Der Grund liegt in der Definition der Eigenwerte im Reellen: f(v)=lambda*v. Also ändern die Eigenvektoren durch die Abbildung f nur ihre Länge, aber nicht ihre Richtung, dies ist bei einer Drehung nur für eine Drehung um 0° oder 180° möglich

Aufgabe 4

```
reset()
V = matrix([[2,0,1],[0,2,2],[2,0,3]])
W = matrix([[1,0,1],[1,0,0],[1,1,1]])
```

W.inverse()*V

```
[ 0  2  2]
[ 0  0  2]
[ 2 -2 -1]
```

Aufgabe 5

```
reset()
var('x,y')
A = matrix([[0,2,-2],[3,x,4],[-1,y,1]])
B = matrix([[1,8,x],[3,2,y],[6,4,4]])
C = A*B; C
```

```
[            -6             -4        2*y - 8]
[      3*x + 27       2*x + 40 x*y + 3*x + 16]
[       3*y + 5        2*y - 4    y^2 - x + 4]
```

solve(det(C)==0, y)

```
[y == -1/3*x - 7/3, y == 2]
```

solve(det(C)==0, x)

```
[x == -3*y - 7]
```

expand(det(C(y=2)))

```
0
```

expand(det(C(x=-3*y-7)))

```
0
```

Aufgabe 6

[len(filter(is_prime, [n^2+n+m^2 for n in [1..100]])) for m in [0..41]]

```
[1, 32, 0, 13, 0, 28, 0, 21, 0, 22, 0, 33, 0, 26, 0, 5, 0, 17, 0,
26, 0, 15, 0, 33, 0, 13, 0, 9, 0, 12, 0, 44, 0, 11, 0, 19, 0, 23, 0,
14, 0, 23]
```

```
for m in [0..41]:
   c = 0
   for n in [1..100]:
      if (n^2+n+m^2).is_prime():
         c = c+1
   print (m,c)
```

```
(0, 1)
(1, 32)
(2, 0)
(3, 13)
(4, 0)
(5, 28)
(6, 0)
(7, 21)
(8, 0)
(9, 22)
(10, 0)
(11, 33)
(12, 0)
(13, 26)
(14, 0)
(15, 5)
(16, 0)
(17, 17)
(18, 0)
(19, 26)
(20, 0)
(21, 15)
(22, 0)
(23, 33)
```

```
(24, 0)
(25, 13)
(26, 0)
(27, 9)
(28, 0)
(29, 12)
(30, 0)
(31, 44)
(32, 0)
(33, 11)
(34, 0)
(35, 19)
(36, 0)
(37, 23)
(38, 0)
(39, 14)
(40, 0)
(41, 23)
```

Aufgabe 7

```
reset()
x = [1]
for n in [0..9]:
   x.append(x[n] - (x[n]^2-2)/2/x[n])
show(x)
```

$$\left[1, \frac{3}{2}, \frac{17}{12}, \frac{577}{408}, \frac{665857}{470832}, \frac{886731088897}{627013566048}, \frac{157258404803291863333}{111198484434986813793}\right.$$

```
x = [1.]
for n in [0..9]:
   x.append(x[n] - (x[n]^2-2)/2/x[n])
x
```

```
[1.00000000000000, 1.50000000000000, 1.41666666666667,
1.41421568627451, 1.41421356237469, 1.41421356237310,
1.41421356237309, 1.41421356237310, 1.41421356237309,
1.41421356237310, 1.41421356237309]
```

Aufgabe 8

```
reset()
def fak(n):
   if n==0:
      return 1
   else:
```

```
        return n*fak(n-1)
```

```
fak(5)
```

```
120
```

```
fak(1000)
```

```
WARNING: Output truncated!
full_output.txt


Traceback (click to the left of this block for traceback)
...
RuntimeError: maximum recursion depth exceeded in cmp
```

full_output.txt

```
def fak2(n):
    x = 1
    for k in [1..n]:
        x = x * k
    return x
```

```
fak2(5)
```

```
120
```

```
fak2(1000)
```

```
40238726007709377354370243392300398571937486421071463254379991042993\
85123986290205920442084869694048004799886101971960586316668729948085\
58901323829669944590997424504087073759918823627727188732519779505950\
99527612087497546249704360141827809464649629105639388743788648733711\
91810458257836478499770124766328898359557354325131853239584630755574\
09114262417474349347553428646576611667797396668820291207379143853719\
58824980812686783837455973174613608537953452422158659320192809087829\
73084313928444032812315586110369768013573042161687476096758713483120\
25478589320767169132448426236131412508780208000261683151027341827977\
70478463586817016436502415369139828126481021309276124489635992870511\
49649754199093422215668325720808213331861168115536158365469840467089\
75602900950537616475847728421889679646244945160765353408198901385442\
48798495995331910172335555660213945039973628075013783761530712776192\
68490343526252000158885351473316117021039681759215109077880193931781\
14194545257223865541461062892187960223838971476088506276862967146674\
69756291123408243920816015378088989396451826324367161676217916890977\
99119037540312746222899880051954444142820121873617459926429565817466\
28302955570299024324153181617210465832036786906117260158783520751516\
28422554026517048330422614397428693306169089796848259012545832716822\
64580665267699586526822728070757813918581788896522081643483448259932\
66043367660176999612831860788386150279465955131156552036093988180612\
```

```
13855860030143569452722420634463179746059468257310379008402443243846\
56572450144028218852524709351906209290231364932734975655139587205596\
54228749774011413346962715422845862377387538230483865688976461927383\
81490014076731044664025989949022222176590433990188601856652648506179\
97023561938970178600408118897299183110211712298459016419210688843871\
21855646124960798722908519296819372388642614839657382291123125024186\
64935314397013742853192664987533721894069428143411852015801412334482\
80150513996942901534830776445690990731524332782882698646027898643211\
39083506217095002597389863554277196742822248757586765752344220207573\
63056949882508796892816275384886339690995982628095612145099487170124\
45164612603790293091208890869420285106401821543994571568059418727489\
98094254742173582401063677404595741785160829230135358081840096996372\
52423056085590370062427124341690900415369010593398383577793941097002\
77534720000000000000000000000000000000000000000000000000000000000000\
00000000000000000000000000000000000000000000000000000000000000000000\
00000000000000000000000000000000000000000000000000000000000000000000\
0000000000000000000000000000000000000000000000000000
```

```
def fak3(n):
    return prod([1..n])
```

```
fak3(5)
```

```
120
```