Sage

Kovács Kristóf, Magyar András, Simon András

2023. november 22.

A Sage egy számítógépes algebrai rendszer, ami azt jelenti, hogy numerikus és szimbolikus számításokat is tud végezni. A Python programozási nyelvre épül, ezért programozható.

További információ Sage-ről:

- Paul Zimmermann & al., Computational Mathematics with SageMath https://www.sagemath.org/sagebook/english.html
- Sage dokumentáció: https://doc.sagemath.org/
- Egyebek: http://www.gregory-bard.com/Sage.html

Hogyan lehet kipróbalni?

- https://sagecell.sagemath.org/
- https://cocalc.com/ (itt regisztrálni kell, cserébe el lehet menteni amin dolgozunk — mint Overleaf-ben)
- (még nem működik) be lehet jelentezni leibniz-re, nyitni egy terminált, és abban

  ```
  leibniz:~$ ssh tarski
  ...@tarski's password: XXXXX
  tarski:~$ sage
  ```

  ahol `XXXXX` a password-ünk. (leibniz-en is el lehet indítani sage-et, de ott egy (túl) régi verzió működik)

1. Ízelítő a lehetőségekből

```
sage: 1+1
2

sage: factor(123456)
2^6 * 3 * 643
```

```
sage: factorial(42)
1405006117752879898543142606244511569936384000000000

sage: factor(x^2-1)
(x + 1)*(x - 1)
```

Ez egy tipikus szimbolikus számítás volt.

```
sage: solve(x^2-1==0,x)
[x == -1, x == 1]

sage: m = matrix([[1,1,1],[1,2,3],[3,2,1]])
sage: m

[1 1 1]
[1 2 3]
[3 2 1]

sage: m+m

[2 2 2]
[2 4 6]
[6 4 2]

sage: m^2

[ 5 5 5]
[12 11 10]
[ 8 9 10]

sage: diff(sin(x),x)
cos(x)

sage: integral(sin(x),x)
-cos(x)
```

Mi a programozhatóság, és miért hasznos?

Egy egyszerű példa. Fermat a 17. században azt sejtette, hogy minden $F_n = 2^{2^n} + 1$ alakú szám (ahol $n \in \mathbb{N}$) prímszám. Ha lett volna Sage, könnyen találhatott volna ellenpéldát:

```
sage: for n in range(1,60):
          if not(is_prime(2^(2^n)+1)):
               print(f"n={n} ellenpélda: 2^(2^{n})+1 = {factor(2^(2^n)+1)}")
               break
n=5 ellenpélda: 2^(2^5)+1 = 641 * 6700417
```

Tehát F_5 összetett szám. (Valójában még mindig nem ismert, hogy létezik-e egyáltalán olyan $n > 5$, amelyre F_n *prím*.)
Másik példa:

```
sage: [[Subsets(n, m).cardinality() for m in range(n+1)] for n in range(10)]
```

```
[[1],
 [1, 1],
 [1, 2, 1],
 [1, 3, 3, 1],
 [1, 4, 6, 4, 1],
 [1, 5, 10, 10, 5, 1],
 [1, 6, 15, 20, 15, 6, 1],
 [1, 7, 21, 35, 35, 21, 7, 1],
 [1, 8, 28, 56, 70, 56, 28, 8, 1],
 [1, 9, 36, 84, 126, 126, 84, 36, 9, 1]]
```

Ez a Pascal-háromszög első tiz sora: `Subsets(n,m)` az $\{1, 2, ..., n\}$ halmaz összes m méretű részhalmazát adja vissza. Írhattuk volna ezt is:

```
sage: [[binomial(n,m) for m in range(0,n+1)] for n in range(10)]
```

2. Alapok

`=` hozzárendelésre (nem összehasonlításra) szolgál:

```
sage: 3 = 3
ValueError

sage: a = 3
sage: a
3

sage: type(a)
<type 'sage.rings.integer.Integer'>
```

Ez nem `a`, hanem `a` értékének típusa:

```
sage: a = "Hello world!"
sage: type(a)
<type 'str'>
```

Térjünk vissza

```
sage: a = 3
```

-hoz.

Összehasonlításra a `==` (és `<=`, `>=`, `<`, `>`, `!=`) predikátumokat használjuk:

```
sage: a == 3
True

sage: a > 3
False
```

Néhány matematikai művelet:

```
sage: 2^16
65536
```

```
sage: 2**16
65536

sage: 13/4
13/4

sage: 13//4 #integer quotient
3

sage: 13%4 # (13 modulo 4)
1
```
(A következő három csak a parancssori verzióban működik.)
```
sage: _ #az utolsó eredmény
1

sage: ___ #az utolsó előtti előtti eredmény
3

sage: __ #az utolsó előtti eredmény
1
```
Egyszerre több eredményt is megkaphatunk:
```
sage: a == 3, a > 3
(True, False)

sage: type(_)
<type 'tuple'>
```
És egyszerre több, egymás után végrehajtandó utasítást is kiadhatunk:
```
sage: a = 3; a
3
```

Operátor	**Leírás**
`or`	logikai vagy
`and`	logikai és
`not`	logikai nem (tagadás)
`in`, `not in`	(nem) eleme
`is`, `is not`	identitásteszt
`>`, `<=`, `>`, `>=`, `==`, `!=`	összehasonlítás
`+`, `-`	összeadás, kivonás
`*`, `/`, `%`	szorzás, osztás, maradék
`**`, `^`	hatványozás

Itt minél később jön egy operátor, annál magasabb a prioritása. Tehát például
```
sage: 2^3*4 == (2^3)*4
True
```

de

```
sage: 2^3*4 == 2^(3*4)
False

sage: sqrt(2) #a Sage pontos
sqrt(2)

sage: sqrt(2.0) #de csak akkor, ha módjában áll
1.41421356237310

sage: n(sqrt(2)) #és még olyankor is pontatlanná tehetjük
1.41421356237310
```

Az `n()` függvény numerikus közelítést ad vissza.

```
sage: sqrt(2)^2 #a Sage pontos
2
```

Mindjárt visszatérünk a közelítésekre, de előbb megjegyezzük, hogy az eddig használt függvényeknek van metódus megfelelőjük is, így írhatunk ilyeneket is:

```
sage: 2.sqrt(), 2.sqrt().n(), sqrt(2).n()
(sqrt(2), 1.41421356237310, 1.41421356237310)
```

Még néhány példa a numerikus közelítésre (és metódusokra):

```
sage: pi #egzakt
pi

sage: n(pi) #numerikus közelítés
3.14159265358979

sage: n(pi,digits=50) #...50 jegy pontossággal
3.14159265358979323846264338327950288419716939937 51
```

Ezt írhattuk volna így is: `pi.n(digits=50)`.

```
sage: sin(pi/3) #egzakt
1/2*sqrt(3)

sage: n(sin(pi/3)) #közelítés
0.866025403784439

sage: sin(n(pi/3)) #közelítés
0.866025403784439
```

Általános segítségért írjuk be a `?` parancsot. Ha segítséget szeretnénk kapni egy Sage függvényhez, írjuk be a nevét, egy kérdőjellel kiegészítve, pl. `sin?` vagy `diff?`. A metódusokkal sajnos kicsit bonyolultabb a helyzet.

Amikor a Sage parancssori verziójába beírunk valamit, mint például `sin?`, előfordulhat, hogy a program elindít egy lapozó programot, ami megjeleníti a kért segítséget. Írjon be egy szóközt a következő oldalra görgetéshez, írja be a `h` parancsot, hogy segítséget kapjon a lapozóprogrammal kapcsolatban, és írja be a `q` parancsot a kilépéshez, vagyis a `sage:` prompthoz való visszatéréshez. Az `up/down` és a `PgUp/PgDn` billentyűk valószínűleg működni fognak.

A parancssori verzióban használhatjuk a `TAB`-kiegészítést!

2.1. Alapvető algebra és kalkulus

Amikor szimbolikus számításokat végzünk, szimbolikus változókra van szükségünk, szemben a szokásos változókkal. Az egyik különbség az, hogy a közönséges változókat a Sage mindig kiértékeli, míg a szimbolikus változókat nem. És ez utóbbi az, amit általában a matematikában akarunk, mert amikor $x^2 + 1$-t írunk, akkor a polinomot akarjuk, nem azt a számot, amelyet az x-et kiértékelve kapunk. Hasonlóképpen, a $\sin' x$ (`diff(sin(x),x)`) esetében nem érdekel, hogy mi az x értéke, a $\sin x$ *függvényt* akarjuk deriválni.[1]

A `x` alapértelmezés szerint szimbolikus változó:

```
sage: type(x)
<type 'sage.symbolic.expression.Expression'>
```

szemben minden mással, pl.:

```
sage: type(y)
NameError:  name 'y' is not defined
```

De `y`-t is szimbolikus változóvá alakíthatjuk, így:

```
sage: var('y')
y
```

és ezután már

```
sage: type(y)
<type 'sage.symbolic.expression.Expression'>
```

Valójában ezt több változóra is megtehetjük egyszerre, így:

```
sage: var('m xx yy zz')
(m, xx, yy, zz)
```

Matematikai („hívható szimbolikus”) függvények definiálása

```
sage: f(y) = y^3 ; f
y |-> y^3
```

[1] Valójában az úgynevezett „szimbolikus változók” megtalálhatók a közönséges programozási nyelvekben, például a Pythonban is. Egy függvény definiálásakor a változói szintén nem értékelődnek ki, csak „helyfenntartók”.

```
sage: f(3)
27
```
`f` fenti definíciójának mellékhatása, hogy `y` szimbolikus változóvá alakul. Így
```
sage: f(2*y)
8*y^3
```
is működik, akárcsak
```
sage: f(2*x^2)
8*x^6
```
de `f(2*z)` nem, hacsak `z` nem alakult már korábban szimbolikus változóvá.

Néhány más dolog a függvényekkel kapcsolatban:
```
sage: diff(f)
y |-> 3*y^2

sage: diff(f,2) #deriváld kétszer
y |-> 6*y
```
De `diff(f,y)` és `diff(f,y,2)`, vagy méginkább `diff(f(y),y)` és `diff(f(y),y,2)` talán világosabb[2]. És mindenképpen meg kell nevezni a változót többváltozós függvénynél[3] , mint ebben a példában:
```
sage: g(x,y) = y*exp(x*y)
sage: diff(g,y)
(x, y) |-> x*y*e^(x*y) + e^(x*y)
```
vagy
```
sage: g.diff(y)
(x, y) |-> x*y*e^(x*y) + e^(x*y)
```
vagy akár
```
sage: g(x,y).diff(y)
x*y*e^(x*y) + e^(x*y)
```

Sage-el kereshetünk primitív függvényt:
```
sage: integral(f(y),y)
1/4*y^4
```
vagy integrálhatunk:
```
sage: integral(f(y), y, 1, 4)
255/4
```
azaz $\int_1^4 f(y)\,dy = \frac{255}{4}$, de határértéket is számolhatunk:
```
sage: g(y) = (1 + pi/y)^y
sage: limit(g(y), y=infinity)
e^pi
```
vagy
```
sage: g(y).limit(y=infinity)
e^pi
```

[2] Másrészt viszont `diff(f)` és `diff(f,y)` maguk is („hívható, szimbolikus”) függvények, míg `diff(f(y),y)` csak kifejezés. Ez a gyakorlatban azt jelenti, hogy `diff(f,y)(42)`-t írhatunk, `diff(f(y),y)(42)`-t viszont nem, helyette `diff(f(y),y).subs(y=42)`-t kell — a `.subs()` metódusról rövidesen lesz szó.

[3] kivéve, ha a Jacobi-mátrixát akarjuk megkapni

És a Sage félre is vezethet bennünket:

```
sage: limit(1/y, y = 0)
Infinity
```

noha tudja, hogy a baloldali határérték *nem* ∞:

```
sage: limit(1/y, y = 0, dir = '-')
-Infinity
```

Mellesleg `infinity` írható `oo`-ként is.

Ha csak egy dolgot akarunk csinálni egy függvénnyel, akkor nem szükséges definiálni. Például ahelyett, hogy definiálnánk `f(y) = y^3`-öt és aztán integrálnánk (mint fent), mondhatjuk egyszerűen, hogy

```
sage: integral(y^3,y,1,4)
255/4
```

És még ha több dolgot is akarunk vele tenni, a parancssori kliensben könnyen előhívhatjuk a korábbi parancsokat, nemcsak a kurzor-billentyűkkel, de inkrementális visszafelé kereséssel is, amit a `Ctrl-r` billentyűkombináció indít, akárcsak a parancssorban. Ezért nem életszükséglet, hogy minden függvényt definiáljunk, amivel hosszasan dolgozni akarunk. Másrészt viszont úgy tűnik, hogy a `TAB`-kiegészítés jobban működik a definiált neveken. Például a `TAB` hatására `f.in` beírása után ezek a lehetőségek jelennek meg:

```
f.integral f.inverz_laplace
f.integrate f.inverse_mod
```

`y^3.in` beírása után meg semmi. Próbáljuk ki! Hasonló a helyzet metódusok help-jével.

A kifejezésekkel való foglalkozás közben néha egy változót (vagy akár más, bonyolultabb részkifejezést) szeretnénk valamilyen kifejezéssel helyettesíteni. Ezt a `subs()` metódussal tehetjük meg. Néhány példa:

```
sage: ((x+cos(y))^3+x*y).subs(y=z+1)
(x + cos(z + 1))^3 + x*(z + 1)
```

Ez tipikus. Ne felejtsük el a zárójeleket a helyettesíteni kívánt kifejezés körül:

```
sage: (x+cos(y))^3+x*y.subs(y=z+1)
(x + cos(y))^3 + x*(z + 1)
```

Itt az első `y` békén maradt, mert a Sage nem tudta, hogy része annak a kifejezésnek, amelyen `subs()`-nak hatnia kellene.

Kevésbé jellemző, amikor egy bonyolultabb részkifejezés helyére akarunk helyettesíteni. Ebben az esetben `==`-t kell használni `=` helyett:

```
sage: ((x+cos(y))^3+x*y).subs(x*y==z+1)
(x + cos(y))^3 + z + 1
```

vagy egy *szótár*ban (ez egy Python adatstruktúra) is megadhatjuk, hogy mit akarunk mi helyébe helyettesíteni:

```
sage: ((x+cos(y))^3+x*y).subs({x*y:  z+1})
(x + cos(y))^3 + z + 1
```

Két másik gyakran használt metódus kifejezések manipulálására az `expand()` és a `collect()`:

```
sage: ((x+cos(y))^3+x*y).expand()
x^3 + 3*x^2*cos(y) + 3*x*cos(y)^2 + cos(y)^3 + x*y

sage: ((x+cos(y))^3+x*y).expand().collect(x)
x^3 + 3*x^2*cos(y) + cos(y)^ 3 + (3*cos(y)^2 + y)*x
```

A `collect()` metódus hívásának eredménye az argumentumaként megadott változó polinomja.

Egyenletek megoldása

A `solve()` függvényt már láttuk működés közben.

```
sage: solve(x^2-1==0,x)
[x == -1, x == 1]
```

Ez is működik:

```
sage: solve(x^2-1,x)
[x == -1, x == 1]
```

mert ha csak egy kifejezést adunk meg egyenlet helyett, akkor azt feltételezi, hogy annak keressük a gyökeit. A `solve()` a megoldások listáját[4] adja vissza:

```
sage: type(_)
<class 'sage.structure.sequence.Sequence_generic'>
```

Ez egy speciális lista, de azért lista:

```
sage: isinstance(solve(x^2-1,x), list)
True
```

Ezért az egyes megoldásokat elérhetjük, ha az eredményt `[i]`-vel indexeljük; így például

`sage: sols = solve(x^2-1,x)` után írhatjuk, hogy

```
sage: sols[1]
x == 1
```

Vegyük figyelembe, hogy az indexelés 0-val kezdődik. És ha az értékre van szükségünk (mint általában), például azért, hogy ellenőrizzük, valóban megoldás-e (lásd fent a `subs()` metódust!), akkor ennek az egyenletnek a jobb oldalát vesszük:

```
sage: sols[1].rhs()
1
```

A `solve()` mindig megpróbál pontos megoldásokat találni (legalábbis magányos egyenletek esetén). Ha nem talál, akkor visszaadja az eredeti egyenletet, néha enyhén álcázott alakban:

```
sage: solve(x^7-3*x^2+1,x)
[0 == x^7 - 3*x^2 + 1]
```

Ha ez történik, használhatjuk a `find_root` parancsot[5] *valós* gyökök keresésére:

```
sage: find_root(x^7-3*x^2+1,-100,100)
-0,5715684166370613
```

[4] A lista objektumok gyűjteménye (egy Python adatstruktúra); az egyes objektumok természetes számokkal való indexeléssel érhetők el.

[5] vagy a `solve()` függvény `to_poly_solve` keyword argumentumát, ld. nemsokára!

ami egy megoldás numerikus közelítését adja vissza. Az utolsó két argumentum határozza meg azt az intervallumot, amelyen a gyököt keresi. Tehát egy, az előzőtől különböző gyök megtalálásához próbálkozhatunk ezzel:

```
sage: find_root(x^7-3*x^2+1,0,100)
1,1792979467976112
```

A `solve()` segítségével egyenlőtlenségeket is meg lehet oldani:

```
sage: solve(x^2>8,x)
[[x < -2*sqrt(2)], [x > 2*sqrt(2)]]
```

Szerepelhetnek a megoldandó egyenletben paraméterek is (amik persze valójában pont ugyanolyan változók, mint `x`):

```
sage: var('x a b c')
sage: solve(a*x^2 + b*x + c,x)
[x == -1/2*(b + sqrt(b^2 - 4*a*c))/a, x == -1/2*(b - sqrt(b^2 - 4*a*c))/a]
```

Ezért aztán ugyanezt az egyenletet meg tudjuk oldani a többi változóra is, bár ebben az esetben ez nem túl hasznos:

```
sage: solve(a*x^2 + b*x + c==0,b)
[b == -(a*x^2 + c)/x]
```

Több egyenlet (egyenletrendszer) és egyenlőtlenség egyidejű megoldása is lehetséges:

```
sage: solve([x+y==6, x*y==4], x, y)
[[x == -sqrt(5) + 3, y == -4/(sqrt(5 ) - 3)], [x == sqrt(5) + 3, y
== 4/(sqrt(5) + 3)]]
```

Az első argumentum ebben az esetben a megoldani kívánt egyenletek (és egyenlőtlenségek) listája, a maradék argumentumok pedig az „ismeretlenek”.

Példa arra, amikor egyenlőtlenség is szerepel a listában:

```
sage: solve([x^2 -1==0, x<0],x)
[[x == -1]]
```

`solve()` néha nem ad vissza minden megoldást:

```
sage: solve(sin(x),x)
[x == 0]
```

Ilyenkor erőltethetjük egy kicsit:

```
sage: solve(sin(x),x, to_poly_solve='force')
[x == pi*z35]
```

`z35` egy frissen generált szimbólum, és tetszőleges egész számot jelent. Másik példa:

```
sage: solve(abs(x)==1,x)
[abs(x) == 1]
```

de

```
sage: solve(abs(x)==1,x,to_poly_solve='force')
[x == -1, x == 1]
```

További lehetőségekért nézzük meg, hogy mit ír a help vagy a dokumentáció a `solve()`-ról.

Feltevések

```
sage: solve(x^2==1,x)
[x == -1, x == 1]

sage: assume(x>0); solve(x^2==1,x)
[x == 1]

sage: assumptions()
[x > 0]

sage: forget(x>0); solve(x^2==1,x)
[x == -1, x == 1]
```

További ilyenek:
`sage: assume(y,'integer')` vagy akár
`sage: assume(y,'odd')`

Lineáris algebra

Azt már láttuk, hogyan lehet egyenlet(rendszer)eket megoldani, és természetesen ezek az egyenletek lehetnek lineárisak.

```
sage: solve([2*x + 4*y == 14,3*x + 2*y == 13],x,y)
[[x == 3, y == 2]]
```

Ennek egy megoldása volt, de itt egy másik, aminek végtelen sok:

```
sage: solve([4*x - 2*y + 3*z == 1, 8*x - 4*y + 6*z == 2, 12*x - 6*y
+ 9*z == 3 ],x,y,z)
[[x == -3/4*r3 + 1/2*r4 + 1/4, y == r4, z == r3]]
```

Itt `r3` és `r4` frissen generált szimbólumok, amik tetszőleges valós számot jelentenek.

Ha nincs megoldás, akkor `solve()` az üres listát adja vissza:

```
sage: solve([x + y == 0, x + y == 1],x,y)
[]
```

Lineáris egyenletrendszereket mátrix alakban is meg tudunk oldani. Nézzük az előbbiek közül azt, aminek végtelen sok megoldása volt. Mátrixot legegyszerűbben úgy lehet megadni, hogy a `matrix()` függvényt a sorok listájával hívjuk meg, ahol minden sor a koordinátáinak listája:

```
sage: A = matrix([[4,-2,3],[8,-4,6],[12,-6,9]]) ; A

[ 4 -2 3]
[ 8 -4 6]
[12 -6 9]
```

De biztonságosabb ezt így írni:
`sage: A = matrix(QQ,[[4,-2,3],[8,-4,6],[12,-6,9]])` mert így tudatjuk Sage-el, hogy nem csak $\mathbb{Z}$-ben (ahonnan az együtthatók jönnek), hanem $\mathbb{Q}$-ban is végezhet számításokat. (`QQ` helyett `RR`-et írhatnánk, de az a pontosság rovására menne.)

```
sage: v = vector([1,2,3]).column() ; v
```

```
[1]
[2]
[3]
```

sage: `A.solve_right(v)` (vagy `A \ v`)

```
[1/4]
[0]
[0]
```

Ez egy partikuláris megoldás, és valóban az, mert
sage: `A*(A \ v)`

```
[1]
[2]
[3]
```

Ha minden megoldást akarunk, ehhez hozzá kell adnunk `A` magtere egy bázisának összes lineáris kombinációját[6]. Egy bázist megkapunk így:
sage: `k = A.right_kernel() ; k`

```
Vector space of degree 3 and dimension 2 over Rational Field
Basis matrix:
[1 0 -4/3]
[0 1 2/3]
```

Ez mátrixnak tűnik, de nem az. (Ez kiderül `type(k)`-val.) Így kaphatunk belőle mátrixot:
sage: `m=k.matrix() ; m, type(m)`

```
[1 0 -4/3]
[0 1 2/3]
<type 'sage.matrix.matrix_rational_dense.Matrix_rational_dense'>
```

ami azért jó, mert így hozzá tudunk férni a soraihoz:
sage: `m[0] , m[1]`
`((1, 0, -4/3), (0, 1, 2/3))`
amiknek lineáris kombinációit kell a fenti partikuláris megoldáshoz adni, hogy az összes megoldást megkapjuk.
Néhány további dolog mátrixokkal kapcsolatban: `A` továbbra is a fent használt mátrix:
sage: `A`

```
[ 4 -2  3]
[ 8 -4  6]
[12 -6  9]
```

[6] Ha $Ax_0 = b$, akkor az $Ax = b$ egyenlet összes megoldásainak halmaza $x_0 + \operatorname{Ker} A (= \{ x_0 + x : Ax = 0 \})$.

Egy mátrix adott sorának vagy oszlopának lekérdezése:

```
sage: A.row(1)
(8, -4, 6)
```

vagy

```
sage: A[1]
(8, -4, 6)
```

és

```
sage: A.column(1)
(-2, -4, -6)
```

A mátrixot kiegészíthetjük extra oszlopokkal

```
sage: A.augment(v)
```

```
[ 4 -2  3  1]
[ 8 -4  6  2]
[12 -6  9  3]
```

és redukált lépcsős alakra hozhatjuk:

```
sage: A.augment(v).echelon_form()
```

```
[1 -1/2  3/4  1/4]
[0    0    0    0]
[0    0    0    0]
```

Ebből már látszik, hogy

```
sage: A.rank()
1
```

Ha `B` invertálható mátrix, mint pl. ez:

```
sage: B = matrix(QQ,[[2,4,0],[0,-1,1],[1,1,8]]); rank(B)
3
```

akkor invertálhatjuk egyszerűen így:

```
sage: B, B^(-1), B^(-1)*B
```

```
(
[2  4  0]  [ 9/14  16/7  -2/7]  [1 0 0]
[0 -1  1]  [-1/14  -8/7   1/7]  [0 1 0]
[1  1  8], [-1/14  -1/7   1/7], [0 0 1]
)
```

`A.eigenvalues()`, `A.eigenvectors_right()` és `A.charpoly()` az `A` mátrix sajátértékeit, sajátvektorait és karakterisztikus polinomját adja vissza.[7]

```
sage: A.charpoly()
x^3 - 9*x^2

sage: A.eigenvalues()
[9, 0, 0]
```

[7] Emlékeztető: A sajátértékei a $\det(A-\lambda I) = 0$ *karakterisztikus egyenlet* gyökei ($\det(A-\lambda I)$ neve: A *karakterisztikus polinom*ja.) A λ sajátértékhez tartozó sajátaltér: az $A-\lambda I$ mátrixú homogén egyenletrendszer megoldásai, azaz $\mathrm{Ker}(A-\lambda I)$.

```
sage: A.eigenvectors_right()
```

```
[(9,
  [
  (1, 2, 3)
  ],
  1),
 (0,
  [
  (1, 0, -4/3),
  (0, 1, 2/3)
  ],
  2)]
```

És valóban:

```
sage: A*x*vector([1,2,3]), A*(x*vector([1, 0, -4/3]) + y*vector([0,
1, 2/3]))
```

```
((9*x, 18*x, 27*x), (0, 0, 0))
```

De még ha hiányoznának is Sage-ből ezek a függvények, akkor sem lennénk teljesen elveszve:

```
sage: I3 = identity_matrix(QQ,3) ; I3
```

```
[1 0 0]
[0 1 0]
[0 0 1]
```

```
sage: x * I3
```

```
[x 0 0]
[0 x 0]
[0 0 x]
```

Úgyhogy megkaphatjuk `A` karakterisztikus polinomját így:

```
sage: det(A - x*I3)
-((x + 4)*(x - 9) + 36)*(x - 4) + 20*x
```

ami némi egyszerűsítés után talán már ismerős néhány sorral feljebbről:

```
sage: _.simplify_full()
-x^3 + 9*x^2
```

`A` sajátértékeit így kapjuk meg:

```
sage: solve(det(A - x*I3),x)
[x == 0, x == 9]
```

Ebben az esetben a gyökök multiplicitása is érdekelhet bennünket:

```
sage: solve(det(A - x*I3),x,multiplicities=True)
([x == 0, x == 9], [2, 1])
```

ami azt jelenti, hogy a 0 kettős gyök. És most már megkaphatjuk a különböző sajátértékekhez tartozó sajátalterek egy bázisát:

```
sage: (A - x*I3).subs(x=9).right_kernel()
```

```
Vector space of degree 3 and dimension 1 over Symbolic Ring
Basis matrix:
[1 2 3]
```

```
sage: (A - x*I3).subs(x=0).right_kernel()
```

```
Vector space of degree 3 and dimension 2 over Symbolic Ring
Basis matrix:
[1 0 -4/3]
[0 1  2/3]
```

Néhány további dolog vektorokkal kapcsolatban:

```
sage: v = vector([1,2,3]) ; w = vector([3,2,1]); v, w
((1, 2, 3), (3, 2, 1))
```

Vektor hossza:

```
sage: v.norm()
sqrt(14)
```

Skaláris szorzat:

```
sage: v.dot_product(w)
10
```

Vektoriális szorzat:

```
sage: v.cross_product(w)
(-4, 8, -4)

sage: v.cross_product(w).dot_product(v),v.cross_product(w).dot_product(w)
(0,0)
```

Számelmélet

Már láttuk, hogy a számok (és a polinomok) faktorizálhatók a `factor()` segítségével:

```
sage: factor(123456)
2^6 * 3 * 643
```

`divisors()` egy szám osztóinak listáját adja vissza:

```
sage: divisors(12)
[1, 2, 3, 4, 6, 12]
```

Ha csak egy szám egy másik számmal való osztásának maradéka érdekel, a Python `%` művelete is megfelel:

```
sage: 13%11
2
```

De a Sage-nek van saját `mod()` függvénye is, ami a megfelelő maradékosztály egy reprezentánsát adja vissza:

```
sage: a = mod(13,11) ; a
2
```

Itt derül ki a különbség:

```
sage: a.parent()
Ring of integers modulo 11
```
és
```
sage: a^4
5
```
ami rendben is van, mert $2^4 \pmod{11} = 5$. Tehát például $5^{-1} \mod 26$-ot kiszámolhatjuk így:
```
sage: m = mod(5,26)^(-1); m
21
```
És valóban,
```
sage: m*mod(5,26)
1
```

`gcd()` a két argumentumának, vagy az egyetlen argumentumaként megadott lista elemeinek legnagyobb közös osztóját számítja ki:
```
sage: gcd(123,240)
3

sage: gcd([123,240,500])
1
```
`lcm()` (legkisebb közös többszörös) hasonlóan működik.

Az Euler féle φ függvényt (egy természetes szám alatti, hozzá relatív prímek száma) az `euler_phi()` segítségével számíthatjuk ki:
```
sage: euler_phi(8)
4
```
a 8-nál kisebb páratlan számok száma.

`solve_mod()` egy egyenlet vagy egyenletek egy listájának összes megoldását modulo valamilyen egész adja vissza (a modulus a második argumentuma). Az egyenletekben csak egy- vagy többváltozós polinomok szerepelhetnek. Például a $12x \equiv 15 \pmod{21}$ lineáris kongruencia megoldásához a következőt írjuk:
```
sage: solve_mod(12*x==15,21)
[(3,), (10,), (17,)]
```
Vagy: a fenti $5^{-1} \mod 26$-ot így is kiszámíthattuk volna:
```
sage: solve_mod(5*x == 1, 26)
[(21,)]
```
Példa nemlineáris kongruencia megoldására: $12x^5 \equiv 15 \pmod{21}$
```
sage: solve_mod(12*x^5==15,21)
[(12,), (19,), (5,)]
```
és valóban, például
```
sage: mod(12*5^5,21)
15
```

A különböző modulusú lineáris kongruenciák szimultán megoldására használhatjuk a `crt()` függvényt (`crt` a „Chinese Remainder Theorem” rövidítése). Például

$$x \equiv 3 \pmod{8} \qquad x \equiv 4 \pmod{9} \qquad x \equiv 5 \pmod{25}$$

megoldásához ezt írjuk
```
sage: crt([3,4,5],[8,9,25])
355
```
és ez az eredmény azt jelenti, hogy az összes megoldás halmaza $\{n : n \equiv 355 \pmod{8 \cdot 9 \cdot 25}\}$.

Polinomok

Ha azt akarjuk, hogy `x` ne egyszerűen egy matematikai változó legyen, hanem egy határozatlan egy polinomgyűrűben, akkor
`sage: x = polygen(QQ, 'x')` vagy
`sage: x = polygen(CC, 'x')` vagy
`sage: x = polygen(Integers(8), 'x')` vagy
`sage: x = polygen(GF(5), 'x')` stb. kell írnunk, attól függően, hogy melyik gyűrűt ($\mathbb{Q}[x]$, $\mathbb{C}[x]$,$\mathbb{Z}_8[x]$ vagy $\mathbb{Z}_5[x]$) szeretnénk, ahelyett, hogy egyszerűen `var('x')`-et (ami interaktív használatban egyenértékű `x = var('x')`-el) írnánk.

Ezt a következőképpen ellenőrizhetjük:
```
sage: y = var('y'); q = (2*y+1)*(y+2)*(y^4-1); y.parent(),q.parent(),q
(Symbolic Ring, Symbolic Ring, (y^4 - 1)*(2*y + 1)*(y + 2)))
```
de
```
sage: y = polygen(QQ,'y'); q = (2*y+1)*(y+2)*(y^4-1); y.parent(),q.parent(),q
(Univariate Polynomial Ring in y over Rational Field, Univariate Polynomial
Ring in y over Rational Field, 2*y^6 + 5*y^5 + 2*y^4 - 2*y^2 - 5*y
- 2)
```
vagy
```
sage: y = polygen(GF(5),'y'); q = (2*y+1)*(y+2)*(y^4-1); y.parent(),q.parent(),q
(Univariate Polynomial Ring in y over Finite Field of 5 size, Univariate
Polynomial Ring in y over Finite Field of size 5, 2*y^6 + 2*y^4 + 3*y^2
+ 3)
```

Példa. Hány *különböző* gyöke van y^4-2-nek mint $\mathbb{Q}$, $\mathbb{C}$, $\mathbb{R}$ és $\mathbb{Z}_8$ feletti polinomnak?
```
sage: rings = [QQ, RR, CC, Integers(8)]

sage: for r in rings:
....:     y = polygen(r,'y')
....:     q = y^4 - 2
....:     print(r, len(q.roots(multiplicities=False)))
....:
(Rational Field, 0)
(Real Field with 53 bits of precision, 2)
(Complex Field with 53 bits of precision, 4)
(Ring of integers modulo 8, 0)
```

Azért különböző gyökök, mert a `.roots(multiplicities=False)` metódus a különböző gyökök listáját adja vissza. A `multiplicities=False` ún. keyword argumentum nélkül (vagy a `.roots(multiplicities=True)` hívással) azon `(r,`

`n)` párok listáját adja vissza, ahol `r` a gyök, `n` pedig a multiplicitása, így a lista hossza ekkor is a különböző gyökök száma.[8] Például,

```
sage: x = polygen(QQ,'x'); p = (x-1)^2 * (x+3); p, p.roots()
(x^3 + x^2 - 5*x + 3, [(-3, 1), (1, 2)])
```

Természetesen összeadhatunk, szorozhatunk stb. polinomokat. Például,

```
sage: y = polygen(Integers(3),'y'); p = 2*y+1; p+p, p*p, p^2, p/p
(y + 2, y^2 + y + 1, y^2 + y + 1, 1)
```

Maradékos osztás:

```
sage: x = polygen(QQ,'x'); p = x^3+1; q = 2*x-1; p//q, p%q
(1/2*x^2 + 1/4*x + 1/8, 9/8)
```

ami azt jelenti, hogy $x^3 + 1 = (2x - 1)\left(\frac{1}{2}x^2 + \frac{1}{4}x + \frac{1}{8}\right) + \frac{9}{8}$.
Legnagyobb közös osztó:

```
sage: x = polygen(QQ,'x'); p = x^4-4*x^3-x^2+16*x-12; q = x^2-2*x-3;
p.gcd(q)
x - 3
```

Faktorizáció:

```
sage: x = polygen(GF(2),'x'); p = x^5+3*x-1 ; p.factor()
(x^2 + x + 1) * (x^3 + x^2 + 1)
```

de

```
sage: x = polygen(ZZ,'x'); p = x^5+3*x-1 ; p.factor()
x^5+3*x-1
```

és ennek így is kell lennie, mert

```
sage: p.is_irreducible()
True
```

Példa. Melyek a $2x^4 - 5x^3 - 8x^2 + 17x - 6$ polinom racionális gyökei?

```
sage: x = polygen(QQ,'x'); p = 2*x^4 - 5*x^3 - 8*x^2 + 17*x - 6; p.factor()
(2) * (x - 3) * (x - 1) * (x - 1/2) * (x + 2)
```

vagy

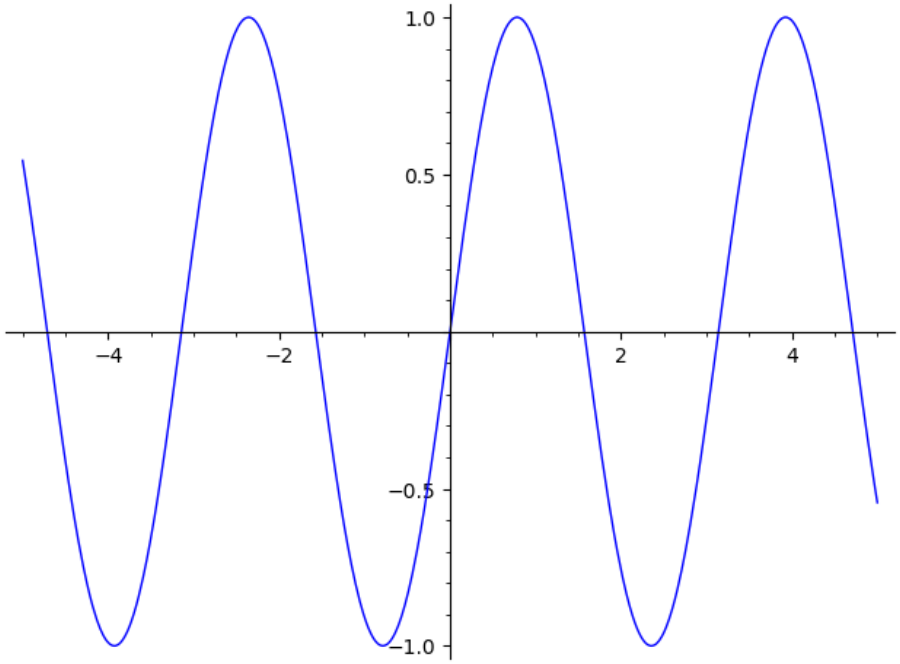
```
sage: p.roots()
[(3, 1), (1, 1), (1/2, 1), (-2, 1)]
```

(a párok második tagjai a multiplicitások).

[8] Azért használtunk itt `multiplicities=True`-t, mert Sage nem tudja kiszámítani a $\mathbb{Z}_n$ feletti polinomok gyökeinek multiplicitásait ha n összetett. Ez nem az ő hibája.

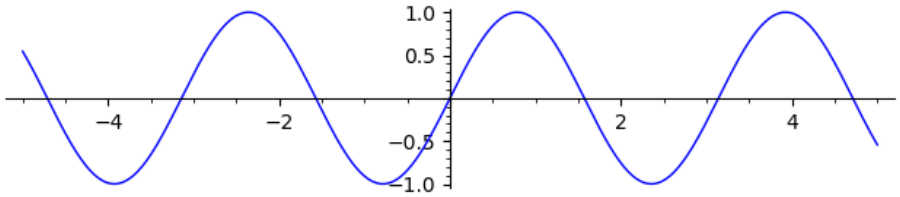
Függvényábrázolás

A `plot(f(x),(x,a,b))` parancs az $[a, b]$ intervallumra megszorított f függvény grafikonját rajzolja meg. A grafikon kinézetét keyword argumentumokkal lehet befolyásolni. (Ha ezeket nem adjuk meg, `plot()` az alapértelmezett értékükkel dolgozik.) Az `aspect_ratio` például az egységnégyzet magasság/szélesség arányát



1. ábra. `plot(sin(2*x),(x,-5,5))`

írja le. Tehát ahhoz, hogy a függőleges egység kétszer akkora legyen, mint a vízszintes, `aspect_ratio=2`-t kell megadni. Az 1. ábrán az alapértelmezett `aspect_ratio='automatic'` hatása látható, a 2. ábrán pedig `aspect_ratio=1`-é.

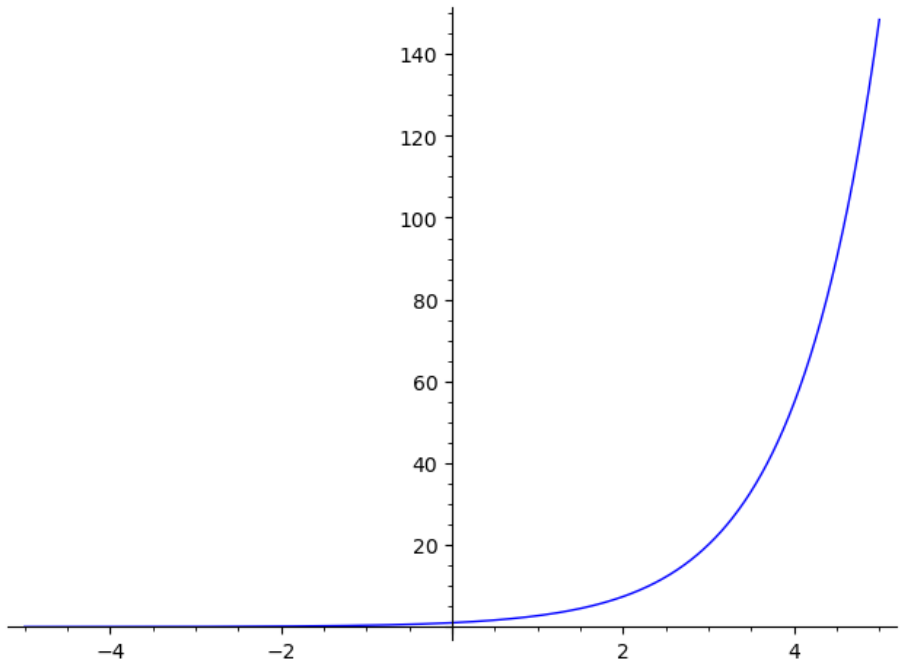


2. ábra. `plot(sin(2*x),(x,-5,5), aspect_ratio=1)`

Így lehet megnézni az `aspect_ratio` hatását:

```
sage: l = [plot(sin(2*x),(x,-5,5),aspect_ratio=2, color='red'), plot(sin(2*x),(x,-5,5),
aspect_ratio=1)]
sage: an = animate(l); an.show(iterations=3, delay=100)
```

Az automatikus `aspect_ratio` jól tud jönni, pl. ebben:

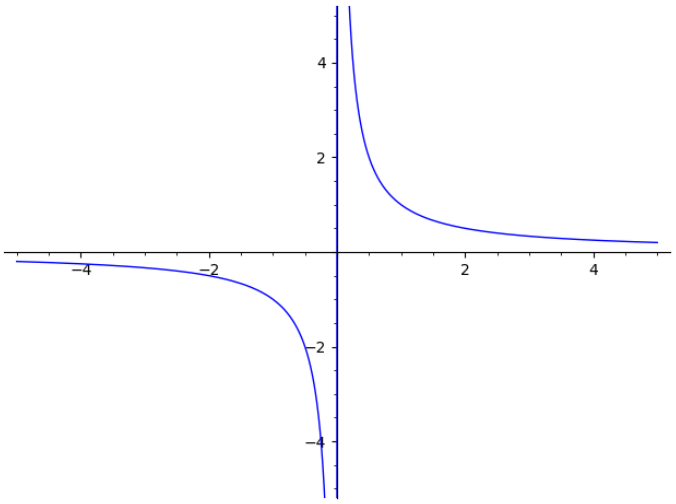


3. ábra. `plot(exp(x),(x,-5,5))`

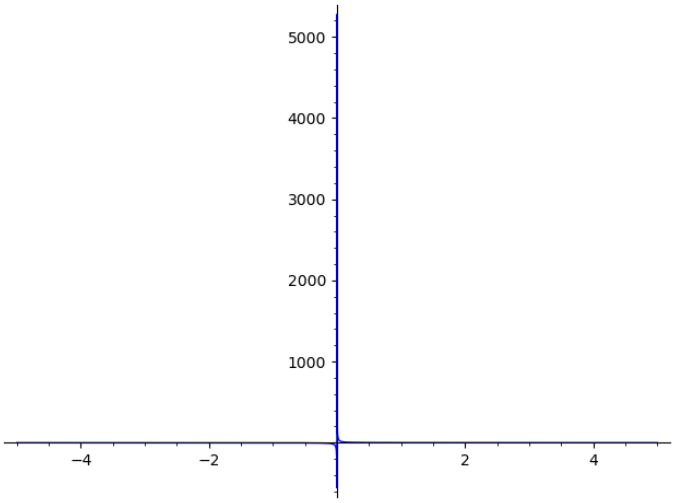
Ha csak néhány „kilógó” függvényérték a probléma, hasznosak lehetnek az `ymin` és `ymax` keyword argumentumok. Például mennyivel informatívabb $1/x$ 4. ábrán látható grafikonja, mint amit az 5. mutat.

Be lehet satírozni pl. az x-tengely és a grafikon közti részt (ld. 6. ábra!) a `fill` keyword argumentumot használva. De az x-tengely (vagyis az $y = 0$ egyenes) helyett választhatunk más vízszintes egyenest: `fill=42` hatására az $y = 42$ egyenes és a grafikon közti rész lesz besatírozva, ld. a 7. ábrát!

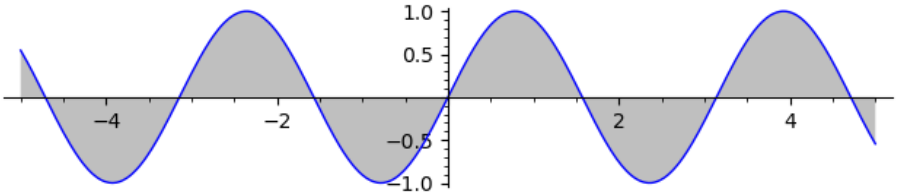
Oda lehet írni, hogy melyik függvényt ábrázolja a kép `legend_label` segítségével (ld. a 8. ábrát!). Ennek értéke egy LaTeX parancs, de ha abban '\' is van, akkor `'parancs'` helyett `r'parancs'`-ot kell írni (erre azért van szükség, mert Python egyébként azt hinné, hogy mondjuk `\sin`-ban \-el az `s`-et akarjuk escape-elni). Ez különösen hasznos, ha egy ábrába (mint a 9.-be) két grafikont teszünk. Ebben két további érdekesség van. Az egyik a szín megadása (`color='red'`). A másik egy hasznos Python-specialitás: mivel a `'` karakterre szükség volt a LaTeXstringen belül (a deriválás jelzésére), a Python string-et dupla idézőjelek



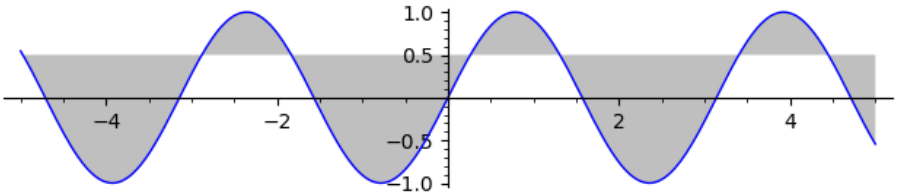
4. ábra. `plot(1/x,(x,-5,5), ymax=5, ymin=-5)`



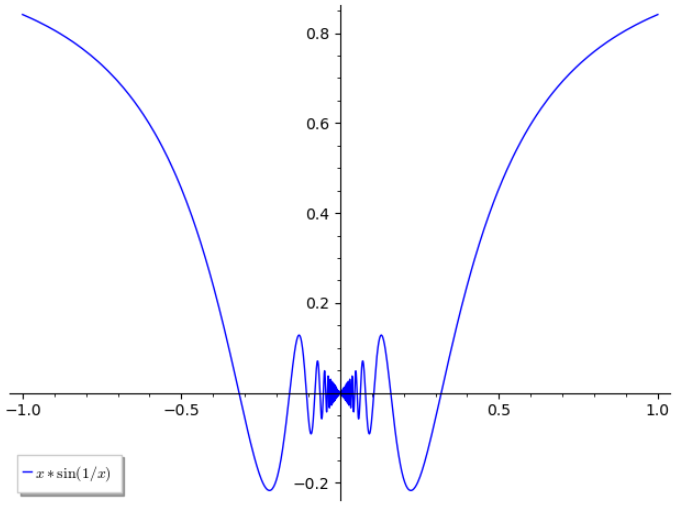
5. ábra. `plot(1/x,(x,-5,5))`



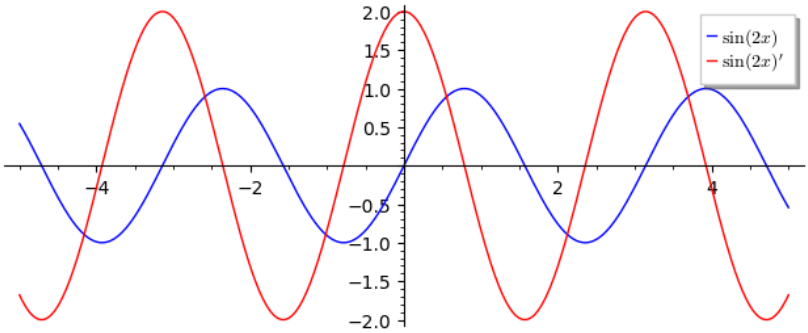
6. ábra. `plot(sin(2*x),(x,-5,5), fill='axis',aspect_ratio=1)`



7. ábra. `plot(sin(2*x),(x,-5,5), fill=0.5,aspect_ratio=1)`



8. ábra. `plot(x*sin(1/x),(x,-1,1), legend_label=r'$x\sin(1/x)$')`



9. ábra. `f(x) = sin(2*x); P1 = plot(f(x),(x,-5,5), aspect_ratio=1, legend_label=r'$\sin(2x)$'); P2 = plot(diff(f(x),x),(x,-5,5), aspect_ratio=1, color='red', legend_label=r"$\sin(2x)'$"); P1 + P2`

(") közé kellett írjuk: `"$\sin(2x)'$"`.

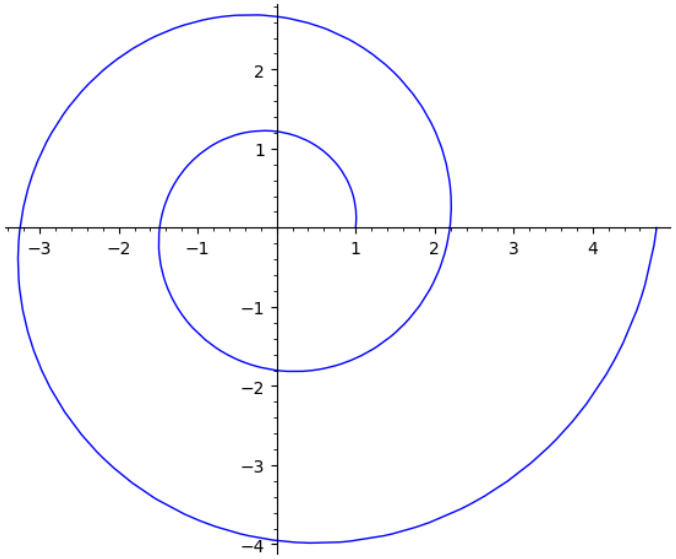
Polárkoordinátarendszerben is ábrázolhatunk függvényt, a `polar=True` és `aspect_ratio=1` keyword argumentumok megadásával. Így készült a 10. ábra. (Ez egyébként az egyik `tikzlab`-beli feladat.) De készülhetett volna a `polar_plot()` parancs segítségével is, így:

```
sage: polar_plot(exp(x/8),(x,0, 4*pi))
```

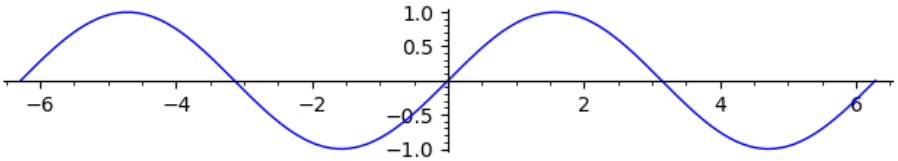
Paraméteres görbék

Azaz amelyek (ha síkgörbék, akkor) két koordináta-függvénnyel vannak megadva. Ilyenekkel Ti*k*Z-ben is találkoztunk amikor függvényt ábrázoltunk, csak ott esetleg első koordináta-függvénynek az identitásfüggvényt választottuk, ahogy a 11. ábrán. De nem muszáj íly módon „szimulálnunk” `plot()`-ot, ld. a 12., 13. és 14. ábrát!

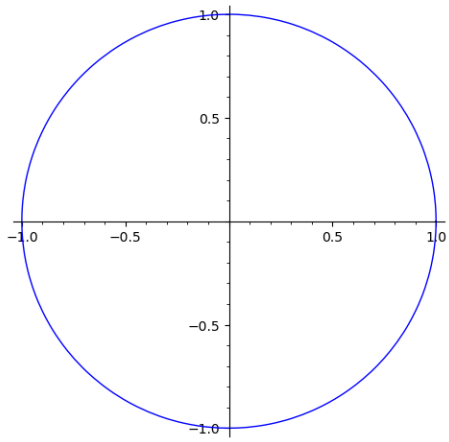
Egy zárt görbe belsejét be is satírozhatjuk, de itt a `fill` értéke csak `True` (vagy `False`, ami az alapértelmezés) lehet. Ld. a 15. ábrát!



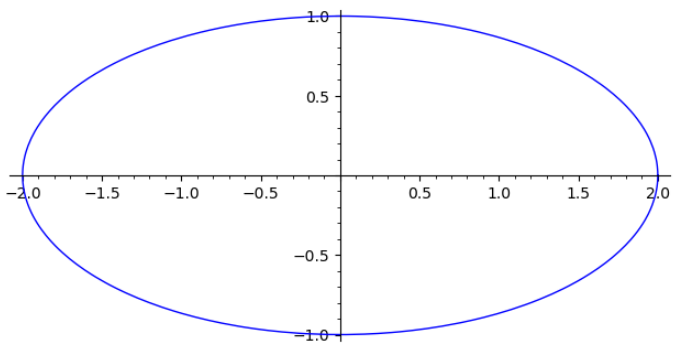
10. ábra. `plot(exp(x/8),(x,0, 4*pi), polar=True, aspect_ratio=1)`



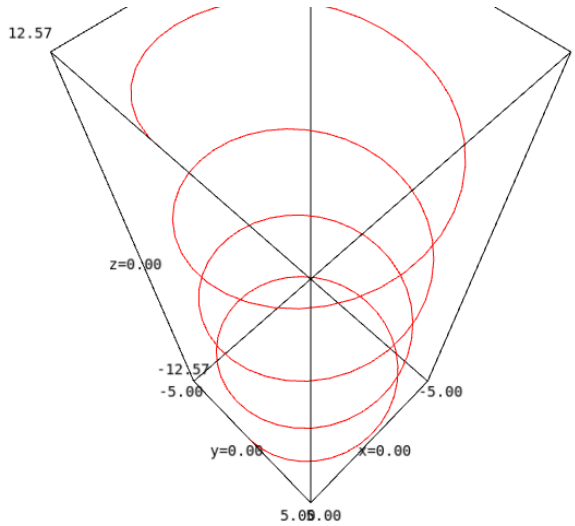
11. ábra. `parametric_plot((x,sin(x)),(x,-2*pi,2*pi),aspect_ratio=1)`



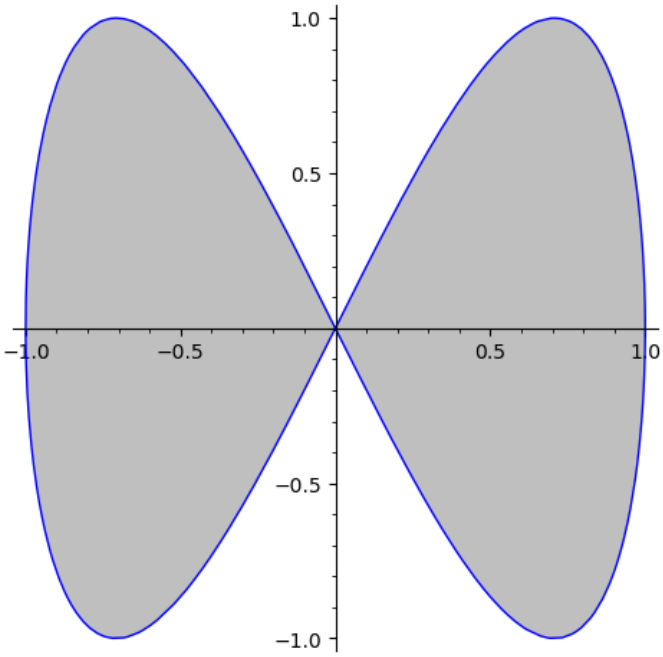
12. ábra. `parametric_plot((cos(x),sin(x)),(x,0,2*pi),aspect_ratio=1)`



13. ábra. `parametric_plot((2*cos(x),sin(x)),(x,0,2*pi),aspect_ratio=1)`



14. ábra. `parametric_plot((5*cos(x),5*sin(x),x),(x,-4*pi,4*pi),plot_points=150, color="red")`



15. ábra. `parametric_plot((sin(x),sin(2*x)),(x, 0, 2*pi),fill=True)`

3. Függelék

Négyzetes mátrix sajátértéke és sajátvektora

Definíció. λ sajátértéke az A négyzetes mátrixnak[9], ha $\exists x \neq 0$ amire $Ax = \lambda x$; ilyenkor x sajátvektora A-nak λ sajátértékkel. (λ lehet 0, x nem! Ha 0 lehetne sajátvektor, akkor mindig az volna, 0 sajátértékkel.) A λ sajátértékhez tartozó sajátaltér: 0 és sajátvektorok λ sajátértékkel.

Példa. A identitás-mátrixnak 1 az egyetlen sajátértéke, és az egész tér az ehhez tartozó sajátaltér. A 0-mátrixnak 0 az egyetlen sajátértéke, és az egész tér az ehhez tartozó sajátaltér. A síkon az origó körüli $\pi/2$ szögű forgatás mátrixának nincs sajátértéke.

Hogy lehet ezeket kiszámolni?

$$\begin{aligned} \lambda \text{ sajátértéke } A\text{-nak} &\iff (\exists x \neq 0) Ax = \lambda x \\ &\iff (\exists x \neq 0)(A - \lambda I)x = 0 \iff \det(A - \lambda I) = 0. \end{aligned}$$

[9] Valójában az általa kódolt lineáris transzformációknak, de Sage-ben azokkal nem tudunk számolni

Ezért A sajátértékei a $\det(A - \lambda I) = 0$ *karakterisztikus egyenlet* gyökei. $\det(A - \lambda I)$ neve: *A karakterisztikus polinom*ja. A λ sajátértékhez tartozó sajátaltér: az $A - \lambda I$ mátrixú homogén egyenletrendszer megoldásai, azaz $\mathrm{Ker}(A - \lambda I)$.