SageMath ile Matematik

Sinan Kapçak

College of Engineering and Technology,
American University of the Middle East, Kuwait

Alin'e

İçindekiler

Önsöz

Jimmy Wales kurucularından olduğu Vikipedi'yi şöyle tanımlıyor: "*Dünya üzerindeki her insana kendi dilinde, en üst kalitede, bedava bir ansiklopedi oluşturma ve dağıtma uğraşısı*". Son birkaç on yılda insanlık için bundan daha değerli az şey yapılmıştır diye düşünüyorum. Linus Torvalds tarafından yaratılan Linux işletim sistemi de benzer şekilde herkesin ücretsiz kullanabildiği ve binlerce insanın katkısıyla zenginleşen, büyüyen bir işletim sistemi.

SageMath (ya da kısaca Sage) de tıpkı Vikipedi ve Linux'ın yaratıcıları gibi iyi kalpli ve paylaşımcı biri olan William Stein adlı matematikçinin insanlığa armağanı. Programın resmi sitesinde misyon "*Magma, Maple, Mathematica ve Matlab'a geçerli bir alternatif olarak, özgür ve açık kaynak bir yazılım yaratmak*" şeklinde ifade ediliyor. Yani SageMath bu ticari yazılımlara alternatif olarak, herkesin ulaşabileceği ücretsiz ve güçlü bir yazılım olma peşinde. İlk versiyonunun çıktığı 2005'ten günümüze, matematiğin pek çok alanında bu hedefine ulaştığını söylemek hiç de yanlış olmaz. Programın Python tabanlı olması da son yıllarda yaygınlaşmasına yardımcı olmuştur elbet. Ne de olsa Python popülerliğini her geçen gün artırıyor ve öğrenmesi kolay programlama dilleri listelerinde başı çekiyor.

Sage için pek çok dilde kaynak bulmak mümkün. Birkaç tane hayli kapsamlı İngilizce kitap var [2, 10, 6, 1] ve bazılarının elektronik kopyaları herkese açık [2, 10]. Ancak Türkçe kaynak ne yazık ki çok yetersiz. Okumakta olduğunuz bu kitaptan başka Türkçe bir SageMath kitabı şu an itibarıyla yok. Bu kitabın amacı SageMath gibi bir girişimi Türkçe olarak öğrenci, öğretmen ve araştırmacılara sunmak ve bu matematik programının ülkemizde de yaygınlaşmasını sağlamaktır.

Kitap, programlamaya giriş niteliğinde olduğundan bu konuda hiçbir ön bilgi gerektirmiyor. Ortaöğretim düzeyinde matematik bilgisi, Sage'in temelleri ve programlamaya dair bölümlerin anlaşılması için yeterli olacaktır. Diğer bazı bölümler için tabii ki ilgili matematik alanına hakimiyet gereklidir.

Programı hiçbir ödeme yapmadan indirip bilgisayarınıza kurmanız yüzde yüz yasal. Dahası; kurulumla uğraşmak istemiyorsanız çevrimiçi kullanım seçeneklerini inceleyebilirsiniz. Detaylar 1.4 numaralı alt bölümde.

Bu kitap hiçbir maddi kazanç beklentisi olmadan yazılmıştır. Elektronik kopyası ücretsiz olarak `www.nesinkoyleri.org/e-kutuphane` veya `www.k-interact.net` adreslerinden indirilebilir. Dileyenler için kitabın basılı kopyası da bulunmaktadır.

Teşekkür

SageMath yazılımının yaratıcısı, bütün SageMath kitaplarının görünmez yazarı William Stein'a teşekkürler!

Bilgisayar ve programlama üzerine ilk derslerimi aldığım hocalarım Timur Karaçay ve merhum Doğan Çoker'e; matematik yazılımının ne demek olduğunu kendisinden aldığım ve benim için bir dönüm noktası olan dersle öğrendiğim, kitabın yazımında da yapıcı tavsiyeleriyle destek olan değerli hocam Ünal Ufuktepe'ye ne kadar teşekkür etsem azdır. Matematik yazılımı alanında gelişimime büyük katkısı olan Erasmus proje danışmanım Andrés Iglesias Prieto'ya da teşekkürü bir borç bilirim.

Bana bu kitabın hedefine ulaşması için emin ellerde olduğu hissini veren, dijital ve basılı olarak yayımlanmasına destek olan Ali Nesin'e ve basım sürecindeki yardımlarından dolayı Nesin Yayınevi genel yayın yönetmeni Esin Pervane'ye çok teşekkürler!

Bu süreçte gerek içerik gerekse nitelik anlamında yardım, öneri ve eleştirilerinden dolayı Ağacık Zafer, Ali Manzak, Ali Nabavi, Arash Pourkia, Fatih Doğan, G. Yazgı Tütüncü, Göknur Yapakçı, Kerem Kaşkaloğlu, Mustafa Zeki, Orhan Dönmez, Osman Can Hatipoğlu, Selçuk Baktır, Server Kasap, Tuncay Candan, Uğur Akduman, Uğur Madran ve Zekeriya Uykan'a ayrı ayrı teşekkürlerimi sunarım.

Kitabı baştan sona her zamanki incelikli ve eleştirel bakışıyla okuyup önerileriyle onu daha anlaşılır hale getiren dostum Harun Barış Çolakoğlu'na; yoğun çalışma temposunda değerli zamanını ayırarak deneyim dolu önerileriyle bu kitabı zenginleştirip daha ulaşılır hale getiren Murat Alp'e sonsuz teşekkürler!

Annem Emine Selamet, babam Mehmet, abim Cemal Bilge, ikiz kardeşlerim Beyzagül ve Şeydagül'e her zamanki sevgi ve desteklerinin yanında kitaba katkı sağlayan yorum, öneri ve eleştirilerinden dolayı teşekkürler!

Yapıcı önerileriyle harikalar yaratan, bu yoğun hazırlık sürecinde gösterdiği sabır ve anlayışla her zaman yanımda olan sevgili eşim Aslı'ya ve farkında bile olmadan beni motive eden küçücük kızımız Alin'e teşekkürler!

Sinan Kapçak
Egaila, Kuveyt
Ekim 2022

Bölüm 1

SageMath'e Başlarken

SageMath [7, 8, 4] bir matematik yazılımı, daha teknik bir deyişle bir bilgisayar cebir sistemidir. Günümüzde bu tip yazılımlar sayısal ve sembolik ileri düzey işlemler ve hesaplamalar yapabilmenin yanında iki ve üç boyutlu grafikleri görüntüleyebilir, animasyon üretebilir ve programlamada kullanılabilirler. Bu bölümde SageMath yazılımının ne olduğunu, onu cazip kılan bazı özellikleri, programa ulaşmak için en kestirme yolu ve diğer kullanım seçeneklerini göreceksiniz.

1.1 *SageMath* Nedir?

Programın adı *SageMath*; ancak *Sage* kullanımı da bir o kadar yaygın. 'Sage' İngilizce bir kelime. Anlamı 'ada çayı'. Aynı zamanda 'bilge', 'bilgece' anlamına da geliyor. Aslında, *System for Algebra and Geometry Experimentation* (cebir ve geometri deneyimi için sistem) ifadesinin kısaltılmış hali. 2005 yılında, Amerikalı bir matematikçi olan William Stein'ın Harvard Üniversitesi'nde bulunduğu sırada başlattığı bir yazılım. Çok sayıda halihazırdaki açık kaynak paketi kullanıyor ve mottosunda da vurguladığı gibi, tekerleği tekrar icat etmektense arabayı oluşturmayı hedefliyor. Mathematica, Matlab, Maple, Magma, Mathcad gibi ücretli programlara ücretsiz bir alternatif olarak doğmuş ve hızlı bir şekilde çok etkili bir noktaya gelmiş olan SageMath, şu an itibarıyla analiz, diferensiyel denklemler, doğrusal cebir, geometri, soyut cebir, kombinatorik, çizge kuramı, nümerik analiz, sayılar teorisi ve istatistik gibi matematiğin pek çok alanını kapsayan özelliklere sahip.

1.2 Neden *SageMath*?

SageMath hem ücretsiz hem kapsamlı hem de ileri düzey bir matematik yazılımıdır. Herhangi bir web tarayıcısından bir dakika içerisinde ulaşılabilecek olan bu yazılımı geçerli kılan bir başka sebep de Python tabanlı olması. Python, kullanımı kolay ve son yıllarda en çok yaygınlaşan programlama dillerinden biri. Öyle ki dünyada pek çok üniversitede programlamaya giriş dersleri Python ile verilmekte. SageMath'ı öğrenirken aynı

zamanda çok geçerli bir dil öğrenecek ve özellikle programlama alanında Python kaynaklarından destek alma şansınız da olacaktır.

Program Windows, Linux ve MacOS işletim sistemlerinde çalıştığı gibi, hiçbir kurulum yapmadan çevrimiçi (online) olarak da kullanılabilir.

1.3 İlk Etkileşim: SageMathCell

SageMath'e ulaşmanın en hızlı yolu olan

`sagecell.sagemath.org`

adresine hemen gidelim ve yazılımı çevrimiçi olarak kullanmaya başlayalım. Bunu herhangi bir ağ tarayıcısında adres çubuğuna yukarıdaki adresi yazarak yapabilirsiniz. Karşınıza çıkacak olan web sayfasında, kod yazmak için boş bir hücre, yazdıklarınızı çalıştırmak içinse üzerinde *Evaluate* (hesapla) yazan bir buton göreceksiniz.

Örneğin hücreye `2+2` yazıp *Evaluate* butonuna tıklayalım:

```
2 + 2
```

```
4
```

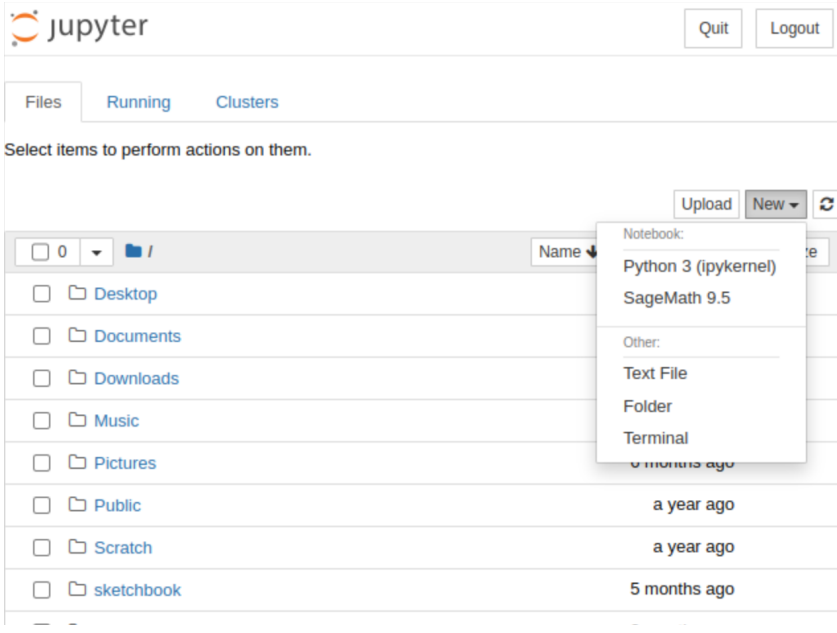
SageMathCell hızlı erişim ve paylaşım özelliğinden (4.10 numaralı alt bölüm) dolayı kullanışlı bir platform. Ancak birden fazla hücre kullanamamak ve çalışmamızı kaydedememek oldukça kısıtlayıcı.

1.4 Diğer Kullanım Seçenekleri

Çevrimiçi diğer bir seçenek olan CoCalc (`cocalc.com`) platformu yazılımın her an güncel versiyonunu sunması gibi bir avantajın yanında dosyalarınızı kaydetme şansı da veriyor. Ancak belli bir bellek ve hesaplama kaynağının aşımında CoCalc ücretli bir opsiyon öneriyor. Eğer hiçbir ücret ödemek istemiyorum diyorsanız SageMath yazılımını kişisel bilgisayarınıza kurmanızı öneririm. Bu aynı zamanda en stabil sonucu verecektir. Programın kurulumu ile ilgili detayları Ek A'da bulabilirsiniz.

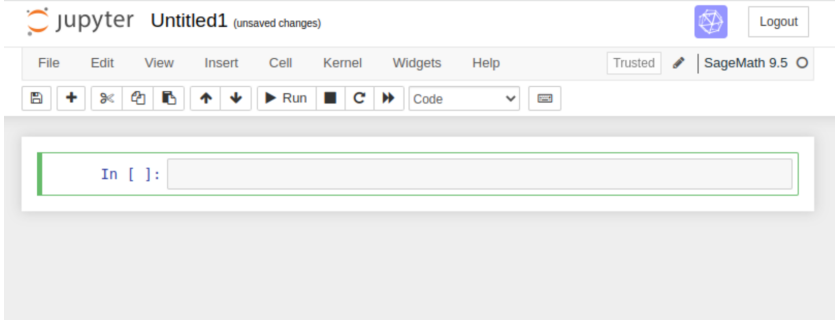
Jupyter Not Defteri

Jupyter Not Defteri `.ipynb` uzantılı özel bir dosya türü. Ek A'daki yönlendirmelerle programı kurup çalıştırdıktan sonra aşağıdaki gibi Jupyter arayüzünü göreceksiniz. Sağ üstteki New butonuna tıklayıp SageMath x.x'i seçiyoruz (buradaki x.x, kurmuş olduğunuz SageMath'in versiyonudur).

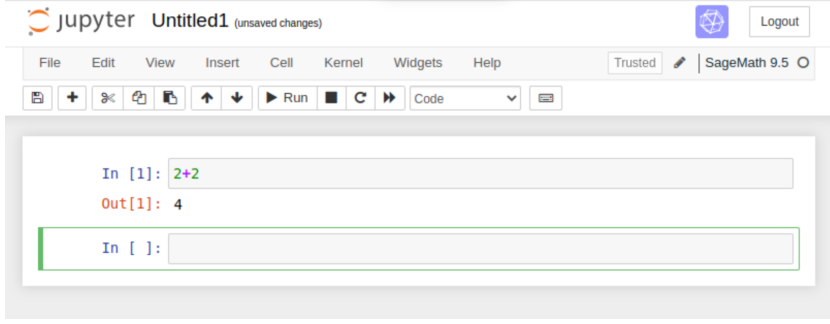


SageMath x.x seçimimizi yaptıktan sonra, varsayılan internet tarayıcısıyla aşağıdaki gibi bir Jupyter Not Defteri (.ipynb dosyası) açılacaktır. SageMath kullanıma hazırdır.

Not 1.4.1 Dosyanızı farklı bir klasöre kaydetmek isterseniz SageMath x.x seçimini yapmadan önce dilediğiniz klasörü seçebilirsiniz.



Bir girdiyi çalıştırmak için programdaki Run tuşunu ya da shift+enter kombinasyonunu kullanırız. Hesaplama bitince de çıktıyı görürüz. Aşağıda `2+2` girdisini (`In[1]`) çalıştırıyoruz ve çıktı (`Out[1]`) olarak `4` elde ediyoruz.



Not 1.4.2 Jupyter Not Defteri kullanarak programlama dosyalarını interaktif ders notu olarak kaydetme şansımız var. Hem de LaTeX desteğiyle.

Not 1.4.3 İkinci bir dosya türü de yaygın olarak kullanılan Sage Not Defteri. Bu dosya `.sagews` uzantılı olup Jupyter Not Defteri'ne benzer bir şekilde hücrelere ayrılmıştır.

Not 1.4.4 İnternet tabanlı bir bulut bilişim platformu olan CoCalc, SageMath projesinin bir parçasıdır. Sage Not Defteri (`.sagews`) ve Jupyter Not Defteri'nin (`.ipynb`) yanında LaTeX dosyasını (`.tex`) da destekleyen CoCalc, gerçek zamanlı ortak çalışma özelliği ve ders yönetim sistemleri de sunuyor.

Bölüm 2

Temel Kavramlar I

Bu bölümde matematik ve programlamada kullanılan temel SageMath komutlarını göreceksiniz. Özellikle ülkemizde ortaöğretim düzeyi matematik derslerindeki pek çok işlem ve fonksiyonun SageMath ile nasıl ifade edileceğine yer verilecek. Aynı zamanda kodlamanın temel kavram ve operatörlerinden bazılarının SageMath'teki kullanımına da değineceğiz.

2.1 Aritmetik

Doğal olarak artı (+) ve eksi (−), sırasıyla toplama ve çıkarma işlemleri için kullanılıyor. Çarpma işlemi için yıldız işaretini, başka bir deyişle asteriski (∗), bölme işlemi için de eğik çizgiyi (/) kullanıyoruz. Birkaç örnek verelim:

```
3 + 7 * 8
```

```
59
```

```
(3 - 7) / 8
```

```
-1/2
```

```
7/3 - 1/3
```

```
2
```

Aynı satırda birden fazla işlem yapılabilir, sadece aralarına virgül koymamız gerekir:

```
1+2*3-4 , (1+2)*3-4 , 1+2*(3-4) , (1+2)*(3-4)
```

```
(3, 5, -1, -3)
```

Not 2.1.1 SageMath dört işlemde sıranın çok belirgin olarak verilmediği durumlarda soldan sağa doğru işlem yapar. Örnek olarak:

```
8/2/2/2
```

```
1
```

Bir sayının üssünü hesaplamak için şapka işaretini (^) kullanıyoruz. Bunun yerine çift yıldız (∗∗) da kullanılabilir. Örneğin 19^{10} sayısını hesaplamak için `19^10` veya `19**10` ifadelerinden birini kullanabiliriz:

```
19^10
```

```
6131066257801
```

Çok daha büyük sayıların hızlıca hesaplanabileceğini belirtelim. Mesela, hemen $2^{1000000}$ sayısını deneyebilirsiniz.

2.2 Nümerik Gösterim

Şimdi de aşağıdaki işlemi inceleyelim:

```
1/125
```

```
1/125
```

SageMath burada kesir gösterimini olduğu gibi korumayı tercih ediyor. Sayı değerini nümerik olarak görmek istersek aşağıdaki gibi `n` komutunu kullanabiliriz. Bu komut *numerical* (Türkçesi *nümerik*) kelimesinin baş harfinden geliyor.

```
n(1/125)
```

```
0.00800000000000000
```

`n` komutu `.n()` formunda aşağıdaki gibi de kullanılabilir:

```
(1/125).n()
```

```
0.00800000000000000
```

Not 2.2.1 `n` komutuna bir alternatif, yukarıdaki iki kullanım biçimini de kabul eden `numerical_approx` (*nümerik yaklaşım* anlamına gelen *numerical approximation* ifadesinin kısaltılmış hali) komutudur.

```
numerical_approx(1/125) , 1/125.numerical_approx()
```

```
(0.00800000000000000, 0.00800000000000000)
```

Not 2.2.2 Tam da bu noktada birkaç yerel hususa değinelim:

1. Ülkemizde ondalık işareti olarak virgül kullanırız. Örneğin iki buçuk sayısını 2,5 şeklinde yazarız. SageMath ondalık işareti olarak nokta kullanıyor. Onun anladığı iki buçuk sayısı 2.5 şekinde yazılır. Kitap boyunca hem tutarlı olmak hem de kafa karışıklığına mahal vermemek için, ondalık ifadeleri virgül yerine nokta ile kullanacağız.
2. SageMath için 0.97 ile .97 aynı şeyi ifade eder. Yani ondalık işaretinden önceki sıfırı yazmasak da olur. Bu kullanım şekli, hesap makinesi ve bilgisayarın yaygınlaşmasıyla çoğu ülkede standartlaşmış durumda.
3. Ülkemizde binlik işareti olarak isteğe bağlı nokta ya da boşluk kullanıyoruz. Yani günlük hayatta büyük sayıları yazarken daha açık seçik olmak adına basamakları üçer üçer grupluyoruz. Örneğin bir milyon sayısını 1,000,000 ya da 1 000 000 şeklinde yazmak gibi. Ancak SageMath'te binlik işareti kullanılmaz; yani bir milyon sayısını 1000000 şeklinde yazmalıyız.

Büyük Sayılar İçin Nümerik Gösterim

Nümerik yaklaşık değer büyük sayılar için de kullanılabilir. Örneğin oldukça büyük olduğunu bildiğimiz

$$9^{9^9}$$

sayısını biraz daha elle tutulur hale getirmek istersek:

```
n(9^9^9)
```

```
4.28124773175747e369693099
```

Yani bu büyük sayının yaklaşık ifadesi şöyleymiş:

$$4.28124773175747 \times 10^{369693099}$$

Not 2.2.3 Yukarıdaki çıktıda gördüğümüz `e` harfi *exponent* (yani *üs*) kelimesinden geliyor ve onu "10 üzeri" şeklinde anlamalıyız. Örneğin `5.12e30` sayısının anlamı 5.12×10^{30}. Benzer şekilde `8.557e-12` demek, 8.557×10^{-12} demek.

Not 2.2.4 a^{b^c} ifadesinin $a^{(b^c)}$ anlamına geldiğini hatırlatalım. Aslında, daha genel olarak, birden fazla sayıda kuvvet içeren durumlarda kuvvet işlemi üstten hesaplanmaya başlar. SageMath de `a^b^c` komutunu aynı şekilde algılıyor.

Birkaç Örnek

Şimdi de aşağıdaki sayıların nümerik ifadelerini dört işlem ve kuvvet kullanarak hesaplayalım:

$$\sqrt[5]{11}$$

```
n( 11^(1/5) )
```

```
1.61539426620218
```

$$\frac{1}{12^{13}}$$

```
n( 1/(12^13) )
```

```
9.34638789871792e-15
```

$$\left[1-\left(\frac{1}{32.3}-0.1967\right)^{-3}\right]^{\frac{2}{3}}$$

```
( 1 - ( 1/( 32.3 ) - 0.1967 )^(-3) )^(2/3)
```

```
36.5139846153917
```

Not 2.2.5 Yukarıdaki son işlemle ilgili birkaç şey:

1. Ondalık sayılarla işlem yaptığımızdan, `n` komutunu kullanmaya gerek duymadan otomatik olarak nümerik bir sonuç elde ettik. Benzer şekilde, $\sqrt[5]{11}$ sayısını nümerik olarak hesaplamak için, `n` komutu kullanmak yerine 11 sayısını 11.0 şeklinde yazabiliriz. Hatta, sadece nokta koyarak, yani `11.^(1/5)` şeklinde yazmak yeterli olacaktır.
2. Hesaplayacağımız ifade köşeli parantez içermesine karşın SageMath'te bunu normal parantez olarak yazdık. Nümerik hesaplarda sadece normal parantez kullanmalıyız. Denemeler yaparak SageMath'in diğer parantezleri bu tip işlemlerde kabul etmediğini görebilirsiniz.

Tek ve Çift Sayılar

Bir tam sayının *çift* (İngilizcesi *even*) ya da *tek* (İngilizcesi *odd*) olduğunu SageMath'e aşağıdaki gibi soruyoruz:

```
is_even(4)
```

```
True
```

```
is_odd(4)
```

```
False
```

Yukarıda 4 sayısı için "çift mi?" ve "tek mi?" sorularına sırasıyla `True` (*doğru*) ve `False` (*yanlış*) yanıtlarını aldık.

Bölüm ve Kalan

Bir bölme işleminde bölümü bulmak için "//" komutu kullanılır:

```
17 // 5
```

```
3
```

Kalanı bulmak için de yüzde işaretini (%) kullanıyoruz. Aşağıda 17'nin 5'e bölümünden kalanı buluyoruz. Diğer bir deyişle 17'nin mod 5'teki ifadesini:

```
17 % 5
```

```
2
```

Bölüm ve kalanı sırasıyla tek seferde bulmak için `divmod` komutunu da kullanabiliriz:

```
divmod(17,5)
```

```
(3, 2)
```

Çarpanlar ve Asal Çarpanlar

Bölenlerle ilgili diğer iki komut ise `divisors` ve `prime_divisors`, yani sırasıyla *bölenler* (çarpanlar) ve *asal bölenler* (asal çarpanlar). Kullanımları şöyle:

```
divisors(12)
```

```
[1, 2, 3, 4, 6, 12]
```

```
prime_divisors(12)
```

```
[2, 3]
```

Bir tam sayı için, "asal mı?" sorusunu `is_prime` komutuyla sorabiliriz. Aşağıda 1273 sayısı için bunu yaptık ve aldığımız yanıt `False` (*yanlış*).

```
is_prime(1273)
```

```
False
```

Yani 1273 asal değilmiş.

Şimdi de bir tam sayıyı asal çarpanlarına nasıl ayıracağımızı görelim. Bunun için *çarpan* anlamına gelen `factor` komutunu kullanıyoruz. Aşağıda iki örnek var:

```
factor(1273)
```

```
19 * 67
```

```
factor(2^111+1)
```

```
3^2 * 1777 * 3331 * 17539 * 25781083 * 107775231312019
```

Görüldüğü üzere SageMath büyük sayıları da asal çarpanlarına ayırmada maharetli.

2.3 OBEB - OKEK

Ortak bölenlerin en büyüğü (OBEB veya EBOB) için kullandığımız komut `gcd` (*greatest common divisor*):

```
gcd(12,16)
```

```
4
```

```
gcd(12345678,23456789)
```

```
1
```

Ortak katların en küçüğü (OKEK veya EKOK) içinse `lcm` (*least common multiple*) komutunu kullanıyor SageMath:

```
lcm(6,8)
```

```
24
```

Aşağıdaki örnek ise $2^6 3^7$ ile $2^7 5^6$ sayılarının en küçük ortak katının asal çarpanlara ayrılmış halini veriyor:

```
factor(lcm(2^6*3^7,2^7*5^6))
```

```
2^7 * 3^7 * 5^6
```

İkiden fazla sayının OBEB ya da OKEK'ini köşeli parantez kullanarak buluruz. OBEB için bir örnek aşağıda:

```
gcd( [ 37^10+1 , 53^10+1 , 11^10+1 ] )
```

```
122
```

2.4 Faktöriyel

Faktöriyel aşağıdaki örneklerdeki gibi kullanılır. Matematiksel gösterimi ünlem işareti (!) olsa da, SageMath bu işareti faktöriyel için kullanmaz.

```
factorial(3)
```

```
6
```

```
factorial(20)
```

```
2432902008176640000
```

2.5 Yorum/Açıklama Ekleme

Bazı durumlarda bir satırdaki belli ifadeleri derleme dışı bırakmak isteyebiliriz. Bunu çoğunlukla ya kod ile ilgili bir açıklama eklemek ya da belli satırlardaki komutları iptal etmek için kullanırız. Böyle durumlar için komut # işareti. Aşağıdaki örnekte # işaretinden sonrası derleme işlemine katılmıyor. Program o kısım yokmuş gibi davranıyor.

```
# bu satır derleme dışı kalacaktır
3+2 # bu satır 3 ile 2'yi topluyor
# 3+5
```

```
5
```

Bazen birkaç satırı aynı anda derleme dışı bırakmak ya da onlarca satır alabilecek açıklamalar yapmak isteyebiliriz. Bunun için açıklamaya 3 tane tırnak işaretiyle başlayıp, açıklamayı yine 3 tane tırnak işaretiyle bitirmemiz gerekir. Örneğin:

```
""" birkaç satır derleme dışı kalacaktır
birkaç satır derleme dışı kalacaktır
birkaç satır derleme dışı kalacaktır
birkaç satır derleme dışı kalacaktır """
3+2
```

```
5
```

Not 2.5.1 Aşağıda komut ve sayıları kesmeyecek şekilde bırakılan boşlukların sonucu etkilemediğini görüyoruz:

```
( 11 /  8    +  6  )  .   n  (   )
```

```
7.37500000000000
```

2.6 Karekök

Karekök işlemi için kullanılan SageMath komutu `sqrt`. Komut, Türkçe anlamı *karekök* olan İngilizce *square root* kelimesinin kısaltılmasıyla oluşturulmuş.

```
sqrt(64)
```

```
8
```

```
sqrt(10)
```

```
sqrt(10)
```

Neden bu yanıtı aldık? $\sqrt{10}$ irrasyonel bir sayı olduğundan bu sayıyı ondalık olarak yazmaya kalkarsak sonsuz basamak kullanmamız gerekir. Yani SageMath ya da başka bir programın bize bu sayının gerçek değerini ondalık sayı olarak vermesi olanaksız. SageMath bize en doğru sonucu vermek için $\sqrt{10}$ sayısını olduğu gibi bıraktı. Ama nümerik olarak yaklaşık değerini nasıl bulabileceğimizi biliyoruz:

```
n(sqrt(10))
```

```
3.16227766016838
```

Nümerik gösterimde varsayılan toplam basamak sayısı 15'tir. Diyelim ki biz 50 basamak görmek istiyoruz. Bunun için, Türkçe karşılığı *basamaklar* olan `digits` komutunu kullanırız:

```
n( sqrt(10) , digits = 50 )
```

```
3.1622776601683793319988935444327185337195551393252
```

Bir milyon basamak görmek de mümkün; bunu siz deneyebilirsiniz. Sadece 4 basamak görmek istersek:

```
n( sqrt(10) , digits = 4 )
```

```
3.162
```

Yukarıda noktadan bağımsız bir şekilde toplam 4 basamak görüyoruz. Bazen noktanın sağında belli sayıda basamak görünsün isteyebiliriz. Bunun için `round` (yani *yuvarla*) komutu kullanılabilir:

```
round( sqrt(10) , 4 )
```

```
3.1623
```

Not 2.6.1 `round` komutu noktadan sonra en fazla 15 basamağı gösterir.

Not 2.6.2

1. SageMath büyük/küçük harfe genel olarak duyarlıdır. Örneğin kök almak için kullanılan `sqrt` komutunun yerine `Sqrt` veya `SQRT` komutlarını kullanamayız. Birkaç istisnai durumdan ikisi şöyle: `true` ile `True` ve `false` ile `False` aynı işlevi görür.

2. Bir satırı uygun herhangi bir yerinden kesip alt satırda devam etmek için "\" işaretini kullanabilirsiniz. Ancak bunu sayıları, metinleri ve komutları bölmeden yapmamız gerekir.

2.7 Mutlak Değer, İşaret ve Tam Değer Fonksiyonu

Mutlak değer için kullanılan komut `abs`; İngilizce karşılığı olan *absolute value* ifadesinin kısaltılmış hali.

```
abs(-7) , abs(0) , abs(1/9)
```

```
(7, 0, 1/9)
```

İşaret kelimesinin İngilizcesi *sign*. Latince, işaret demek olan *signum* kelimesinden gelmiş. SageMath işaret fonksiyonu için `sgn` komutunu kullanıyor:

```
sgn(-7) , sgn(0) , sgn(1/9)
```

```
(-1, 0, 1)
```

Tam değer fonksiyonu tam sayıları olduğu gibi bırakırken tam sayı dışındaki reel sayılar için, bulunduğu tam sayı aralığının taban değerini verir. SageMath de tam değer fonksiyonu için *taban* anlamına gelen `floor` komutunu kullanır:

```
floor(7.59)
```

```
7
```

Bir de tavan anlamına gelen *ceiling* fonksiyonu var; bu da sayının içinde bulunduğu tamsayı aralığının tavan değerini veriyor. Onun için de `ceil` komutunu kullanıyoruz. Mesela:

```
ceil(7.59)
```

```
8
```

2.8 Üç Matematiksel Sabit: π, e ve i

π sayısı için `pi`, e sayısı için de `e` komutunu kullanıyoruz. Gelin bu iki irrasyonel sayının ilk 100 basamağını görelim:

```
n(pi , digits=100)
```

```
3.14159265358979323846264338327950288419716939937510582097494459230781640\
6286208998628034825342117068
```

```
n(e , digits=100)
```

```
2.71828182845904523536028747135266249775724709369995957496696762772407663\
0353547594571382178525166427
```

Kompleks (karmaşık) sayıları ifade etmek için kullanılan i sayısı için `i` ya da `I` komutunu kullanıyoruz. Hızlı bir örnek:

```
I^2
```

```
-1
```

Bu tam da i sayısının tanımı aslında. Yani karesi -1 olan sayı. Matematiksel ifade ile

$$i^2 = -1.$$

Örnek olarak aşağıdaki ifadeyi hesaplayalım:

$$\frac{(2+4i)(4-2i)}{(i-1)^6}.$$

```
(2+4*I)*(4-2*I)/(I-1)^6
```

```
-2*I + 3/2
```

Şimdi de bu üç sabiti içeren aşağıdaki ifadeye bakalım:

```
e^(I*pi)+1
```

```
0
```

SageMath'in verdiği bu cevap, matematiğin en estetik formüllerinden biri olarak kabul edilen aşağıdaki Euler Formülü'nü bilmeyenler için şaşırtıcı gelebilir:

$$e^{i\pi} + 1 = 0$$

Bu noktada $\exp(a)$ gösteriminin e^a ile aynı şey olduğunu ve SageMath'in de bunu bildiğini belirtelim (`exp` komutu Türkçesi *üstel* demek olan *exponential* ifadesinin kısaltılmasıyla oluşturulmuş.).

```
exp(2)
```

```
e^2
```

2.9 Logaritmik İşlemler

Doğal logaritma, yani tabanı e sayısı olan logaritmayı belirtmek için `ln` (Latincesi olan *logarithmus naturalis* ifadesinin kısaltılması) ya da `log` komutunu kullanıyoruz:

```
ln(e)
```

```
1
```

```
log(e)
```

```
1
```

Not 2.9.1 Ülkemizde ortaöğretim matematik eğitiminde log deyince bayağı (adi) logaritma, yani 10 tabanında logaritmanın anlaşıldığını hatırlatalım.

Herhangi bir tabanda logaritmayı aşağıdaki şekilde anlatırız. Örneğin 10 tabanında 1000'in logaritması:

```
log(1000 , 10)
```

```
3
```

Aşağıdaki de 2 tabanında 1024'ün logaritması:

```
log(1024 , 2)
```

```
10
```

2 tabanında 1025'in logaritmasının nümerik yaklaşık değeri de şöyle:

```
n( log(1025 , 2) )
```

```
10.0014081943928
```

log ve exp işlemlerinin birbirlerinin ters işlemi olduğunu biliyoruz. Aşağıdaki örnekler bunu destekliyor.

```
log(exp(7))
```

```
7
```

```
exp(log(7))
```

```
7
```

2.10 Trigonometrik İşlemler

En yaygın işlemlerden bir diğeri de trigonometrik işlemlerdir. Aşağıdaki tablo trigonometrik fonksiyonların SageMath komutlarını veriyor:

Fonksiyon	Komut
Sinüs	`sin(x)`
Kosinüs	`cos(x)`
Tanjant	`tan(x)`
Kotanjant	`cot(x)`
Sekant	`sec(x)`
Kosekant	`csc(x)`

Not 2.10.1 Matematiksel bir detay: Aksi belirtilmedikçe trigonometrik ifadeler *radyan* olarak alınır. Şöyle ki; $\sin 5$ denince 5 radyanlık açının sinüsünü anlamalıyız. Ya da bir fonksiyon olarak verildiğinde; mesela $y = \tan x$ ifadesindeki x açısı radyan olarak verilmiştir.

Birkaç örnek:

```
sin(pi/2)
```

```
1
```

```
cos(pi)-cos(3.14)
```

```
-1.26827246049732e-6
```

Yukarıdaki sonucun sıfıra çok yakın negatif bir sayı olması şaşırtıcı değil.

```
tan(pi/5)
```

```
sqrt(-2*sqrt(5) + 5)
```

```
csc(1).n()
```

```
1.18839510577812
```

2.11 Maksimum/Minimum Değer

Bir sayı grubundaki en büyük (*max*) ve en küçük (*min*) değeri şöyle buluyoruz:

```
max( 11/12 , 12/13 , 13/14 )
```

```
13/14
```

```
max( e^pi , pi^e )
```

```
e^pi
```

```
min( -3 , -4 , 6 , 0 )
```

```
-4
```

```
min(cos(1),cos(2))
```

```
cos(2)
```

2.12 Sadeleştirme, Düzenleme ve Gösterim

Sık kullanılan birkaç komut: `simplify`, `full_simplify`, `expand`, `factor`, `print` ve `show`. Bu komutlar nümerik ve sembolik ifadeleri sadeleştirmek/düzenlemek ve göstermek için kullanılır.

Bir ifadeyi sadeleştirmek için kullanılan komutlardan biri `simplify` (*sadeleştir*). Aşağıda iki kullanım şekli var:

```
( abs(1-sqrt(2)) ).simplify()
```

```
sqrt(2) - 1
```

```
simplify( abs(1-sqrt(2)) )
```

```
sqrt(2) - 1
```

`simplify` komutunun yetersiz kaldığı pek çok durum vardır. Daha sık kullanılan ve çok daha etkili olan diğer bir sadeleştirme komutu da `full_simplify` ya da eşdeğeri olan `simplify_full`. Ancak bu komut yukarıdaki iki kullanım biçiminden sadece ilkini kabul eder. Örneğin

$$\left(1+\sqrt{7}\right)^{10}+\left(1-\sqrt{7}\right)^{10}$$

ifadesini sadeleştirmek istersek:

```
( (1+sqrt(7))^10+(1-sqrt(7))^10 ).full_simplify()
```

```
414976
```

Yukarıdaki işlemin sonucunu ifadeyi genişleterek de bulabiliriz. Bunun için `expand` (*genişlet*) komutunu kullanırız:

```
( (1+sqrt(7))^10+(1-sqrt(7))^10 ).expand()
```

```
414976
```

Asal çarpanlara ayırmak için kullandığımız `factor` komutu, sadece sayılar için değil, daha genel ifadeleri de çarpanlara ayırmak için kullanılır. Mesela, komutu aşağıdaki polinoma uygulayalım:

$$2x^7-11\,x^6-6\,x^5-2\,x^3+11\,x^2+6\,x$$

```
( 2*x^7 - 11*x^6 - 6*x^5 - 2*x^3 + 11*x^2 + 6*x ).factor()
```

```
(x^2 + 1)*(2*x + 1)*(x + 1)*(x - 1)*(x - 6)*x
```

Bir sonucu ekrana yazdırmak için `print` komutunu kullanabiliriz:

```
print( (x^3+1)/sqrt(1-x^2) )
```

```
(x^3 + 1)/sqrt(-x^2 + 1)
```

Yukarıdaki ifadeyi daha açık seçik bir şekilde görmek için de `show` komutu var:

```
show( (x^3+1)/sqrt(1-x^2) )
```

$$\frac{x^3+1}{\sqrt{-x^2+1}}$$

`show` komutu için bir örnek daha:

```
show( cot(pi/16) )
```

$$\sqrt{2}+\sqrt{2\sqrt{2}+4}+1$$

Not 2.12.1 Sadeleştirme en sık kullanılan işlemlerden biridir. Yukarıda gördüğümüz `simplify` ve `simplify_full` komutlarının yanında, farklı matematiksel ifadeler için bazı sadeleştirme komutlarını da belirtmek uygun olacaktır:

Matematiksel İfade	Sadeleştirme Komutu
Trigonometrik	`simplify_trig()`
Rasyonel	`simplify_rational()`
Faktöriyel	`simplify_factorial()`
Logaritma	`simplify_log()`
Reel	`simplify_real()`
Köklü	`canonicalize_radical()`

2.13 Değişken Tanımlama

Önceki alt bölümde x değişkeniyle yapılan işlemlerden birini y değişkenini kullanarak yapmaya çalışalım:

```
show( (y^3+1)/sqrt(1-y^2) )
```

```
NameError: name 'y' is not defined
```

Evet, y'nin tanımlı olmadığı mesajını alıyoruz. SageMath x değişkenini otomatik bir şekilde değişken olarak tanıyorken, başka bir değişken kullanmak istersek onu bizzat tanıtmamız gerekir. Bu aşamada, SageMath ile olan diyaloğumuz biraz değişiyor. Artık sadece sorular sormak yerine ona bilgi verip ilgili sorular sormaya başlıyoruz. Şimdi y değişkenini tanımlayalım:

```
y = var("y")
```

y değişkenini tanımlamak için şu komutlardan herhangi biri de aynı işi görecektir: `y = var('y')` veya `var('y')` veya `var("y")`. Kullanılan `var` komutu, *değişken* anlamındaki *variable* kelimesinin kısaltılması.

Aynı komutu şimdi bir kez daha deneyip SageMath'in artık *y* değişkenini tanıdığını görelim:

```
show( (y^3+1)/sqrt(1-y^2) )
```

$$\frac{y^3+1}{\sqrt{-y^2+1}}$$

Aynı anda birden fazla değişken tanımlamak için aşağıdaki dört yöntemden herhangi birini kullanabiliriz (tırnak işareti yerine kesme işareti de kullanılabilir):

```
z,t,x1,x2,sayi = var("z, t, x1, x2, sayi")
```

```
z,t,x1,x2,sayi = var("z t x1 x2 sayi")
```

```
var("z, t, x1, x2, sayi")
```

```
var("z t x1 x2 sayi")
```

Not 2.13.1

1. Değişken isimleri boşluk içermez. Mesela, `birinci eleman` diye bir değişken oluşturamayız ama `birinci_eleman` şeklinde kullanmak mümkün.
2. Değişken isimleri rakam ile başlayamaz. Yani `x2` diye bir değişken olur ama `2x` diye bir değişken tanımlanamaz.
3. Büyük küçük harf duyarlılığı vardır. Yani `x2` ve `X2` değişkenlerini SageMath farklı değişkenler olarak algılar.

Tanımlanan bir değişkeni SageMath'in unutmasını sağlamak için `restore` komutunu kullanmak yeterli. Aşağıda *y* değişkeni için bunu yapıyoruz:

```
restore("y")
```

2.14 Atama Operatörü

Aşağıdaki örneği inceleyelim:

```
a=2
a+5
```

```
7
```

İlk satırda a'nın 2 olduğunu söylüyoruz, sonra da (ikinci satıra enter ile geçerek) $a+5$ ifadesinin değerini soruyoruz. Değer atamak için kullandığımız atama operatörü eşitlik işareti (=).

Benzer bir örnek:

```
a=2
b=10
a*b+1
```

```
21
```

Birden fazla değişkene değer ataması yaparken aşağıdaki gibi daha kolay bir yöntem de kullanabiliriz:

```
a,b,c,d = 1,2,3,4
a+b+c+d
```

```
10
```

Not 2.14.1 Bir değişkene değer atayacağımız zaman değişken sola, değer de sağa yazılır: Örneğin `a=7` gibi. `7=a` ifadesi hata verecektir.

Not 2.14.2 Bir değişkene bir değer atandığında, onu ayrıca `var` komutu ile tanımlamaya gerek kalmaz. Değer atadıktan sonra kullanılan `var` komutu değişkenin değerini unutmasına sebep olur:

```
a=5
var("a")
a
```

```
a
```

Çok sık kullanacağımız `print` ve `show` komutlarını hatırlayalım:

```
a=2
print(sqrt(a))
```

```
sqrt(2)
```

```
a=2
show(sqrt(a))
```

$$\sqrt{2}$$

Değer atamaya dair bir başka örnek:

```
# üçgen alanı
taban = 6-sqrt(3)
yükseklik = 3+sqrt(3)
alan = taban*yükseklik/2
show(alan)
```

$$-\frac{1}{2}\left(\sqrt{3}+3\right)\left(\sqrt{3}-6\right)$$

Burada, daha anlaşılır bir sonuç elde etmek için sadeleştirme (`full_simplify`) ya da genişletme (`expand`) komutlarından birini kullanabiliriz.

```
#üçgen alanı
taban = 6-sqrt(3)
yükseklik = 3+sqrt(3)
alan = taban*yükseklik/2
show(alan.expand())
```

$$\frac{3}{2}\sqrt{3}+\frac{15}{2}$$

Aynı satırda birden fazla ifade kullanmak için noktalı virgül kullanılabilir. Küçük bir örnek:

```
a=2; b=a^2; show(b^2)
```

16

Görüldüğü gibi değişkenlere değer atamak çok kolay. Atamayı iptal etmek için `restore` komutunu kullanabiliriz. Aşağıda K'ya 12 değeri atanıyor, daha sonra bu atama iptal ediliyor. K çağrılınca da SageMath onun tanımlanmadığını söylüyor:

```
K=12
restore("K")
print(K)
```

```
NameError: name 'K' is not defined
```

Tüm değişken atamalarını iptal etmek için `reset()` komutu kullanılabilir.

Programlama konusunda yeniyseniz aşağıdaki örnekleri başlangıçta biraz yadırgayabilirsiniz. Ama alışmanız hızlı olacaktır.

```
a=2
a=a+1
show(a)
```

3

İlk satırda a için 2 sayısı atanıyor. İkinci satırdaysa mevcut a değişkenine 1 eklenerek yeni a değeri atanıyor. Son satırda da SageMath'in bize a değişkenini göstermesini istiyoruz ve en güncel a değeri olan 3 cevabını alıyoruz.

Not 2.14.3 Yukarıda `a = a+1` komutu yerine `a += 1` de kullanılabilir. Sezgisel olarak daha anlaşılır olduğundan en azından başlangıç örneklerinde ilk ifadeyi kullanacağız.

Benzer bir örnek aşağıda:

```
a=10
a=a+1
a=a+1
a=a+1
print(a)
```

13

Aşağıdaki örnekte a'ya sırasıyla bazı değerler veriliyor ve doğal olarak ekrana son alınan değer yazılıyor. Çünkü SageMath yukarıdan aşağıya doğru girilenleri okuyor ve en güncellenmiş bilgiyi ciddiye alıyor.

```
a=6
a=8
a=2.5
a=11
print(a)
```

11

Not 2.14.4 Değer atamaya dair önemli bir nokta var. Aşağıdaki komutlara bakalım:

```
pi=-100
show(pi)
```

-100

Neler oluyor! π sayısı bir anda -100 oldu. Bunun sebebi SageMath'in bize halihazırda tanımlı sabitlere başka değerler atama hakkı veriyor olması. Bu karmaşaya girmemek için, bilinen sabitleri değişkenleriniz olarak kullanmaktan kaçının. Örneğin a, b, c, d, e, f gibi harflere değer atanmak istendiğinde matematiksel sabit olan e'ye, istenmeden farklı bir değer atanırsa, bu durum karmaşa yaratabilir. Eğer böyle bir atama yaptıysanız, `restore` komutu bunu geri alacaktır. Mesela π'yi geri kazanmak için `restore("pi")` komutunu kullanmak yeterli. Bir başka yöntem de `reset("pi")` komutu. Bütün değişken atamalarını iptal etmek istersek `reset()` komutunu kullanabileceğimizi görmüştük.

Not 2.14.5 Bir değişkenin veri tipini `type` ya da `parent` komutlarıyla öğrenebiliriz. Aşağıdaki örnekte *a* değişkeninin tam sayı (integer) veri tipinde olduğunu görüyoruz.

```
a=2
parent(a)
```

```
Integer Ring
```

Programlamada en yaygın veri tipleri şunlar: tam sayı (int), gerçel sayı yaklaşık değeri için kullanılan kayan-noktalı sayılar (float), harf dizinleri (str), liste (list), demet (tuple), sözlük (dict) ve de mantıksal tip olarak bool (bool).

2.15 Kıyaslama Operatörleri

Kıyaslama operatörleri eşitlik ve eşitsizliklerin doğruluğunu kontrol eder. Eşit, küçük, büyük, küçük eşit ve büyük eşit operatörleri olarak sırasıyla `==`, `<`, `>`, `<=` ve `>=` ifadelerini kullanıyoruz. Ayrıca eşit olmama durumunu `!=` operatörüyle belirtiyoruz. Birkaç örnek görelim:

```
2 == 2
```

```
True
```

```
2 == 3
```

```
False
```

```
2 > 5
```

```
False
```

```
2 <= 5
```

```
True
```

```
2 != 2
```

```
False
```

Sadece `True` (*doğru*) ve `False` (*yanlış*) değerlerini alan ifadelere *boolean ifade* diyoruz. Kıyaslama operatörleri de buna bir örnektir ve `bool` komutu kullanarak da sorgulama yapılabilir. Bu ifade *Boole Cebiri*'nin kurucusu olan matematikçi, filozof ve mantıkçı George Boole'ün isminden gelmektedir.

```
bool(2 != 2)
```

```
False
```

Bir örnek daha inceleyelim:

```
x=4
y=5
x+2*y == 14
```

```
True
```

Bu son örnekte hem atama (=) hem de kıyaslama (==) operatörü kullandık. Yukarıdaki üç satırın Türkçesi şöyle: "x dörttür. y beştir. $x + 2y$ on dört müdür?". Yani `True` cevabı kıyaslamayı sorguladığımız için veriliyor.

Bazı durumlarda daha karmaşık kıyaslamalar yapmak zorunda kalabiliriz. Mesela birden fazla ifadenin hepsinin aynı anda doğru olması ya da en az birinin yanlış olması gibi sorgulamalar. Bunun için mantıksal operatörlere ihtiyaç duyarız.

2.16 Mantıksal Operatörler

Programlama dillerinde akışı kontrol etmek ve kıyaslamalar yapabilmek için mantıksal operatörlere ihtiyaç duyarız. Temel mantıksal operatörler şöyle: `and` (*ve*), `or` (*veya*), `not` (*değil*). Bir yerlerden *ve*, *veya*, *değil* bağlaçlarını, doğruluk tablosunu ya da $D \wedge Y = Y$ gibi önermeleri hatırlıyor olabilirsiniz (doğru için D, yanlış için Y, *ve* için $\wedge$ sembolünü kullandık). Bu işlemleri ve daha fazlasını SageMath ile yapalım:

```
True and True
```

```
True
```

```
True and False
```

```
False
```

```
2==2 and 3==4
```

```
False
```

```
2==2 or 3==4
```

```
True
```

```
a=2
a^2==4 and a^3==10
```

```
False
```

```
a=2
a^2==4 or a^3==10
```

```
True
```

```
not 2>8
```

```
True
```

Biraz daha karmaşık bir örnek:

```
(2 != 4 and 2>3) or ( is_prime(13) and not 3<2)
```

```
True
```

2.17 Matematiksel Fonksiyon Tanımlama

Aşağıda $f(x) = 3x - 1$ fonksiyonunu tanımlıyoruz:

```
f(x)=3*x-1
```

Tanımladık. Şimdi bu fonksiyonu kullanalım:

```
f(10)
```

```
29
```

Fonksiyon dediğimiz şey aslında bir kural. Mesela $f(x) = 3x - 1$ fonksiyonu "3 ile çarp 1 çıkar" kuralıdır. Fakat bir de yine fonksiyon adı verilen ve programlamada kullanılan yapılar var. Aslında onlar da daha genel kurallar zinciri ve 6.2 numaralı alt bölümün konusu.

Örnek 2.17.1 Aşağıdaki fonksiyonu SageMath'te tanımlayalım:

$$A(t) = 1 - e^{-|t|}$$

```
var("t")
A(t)=1-exp(-abs(t))
```

Aşağıda, ilk olarak A fonksiyonunun t'yi neye dönüştürdüğünü, yani fonksiyonun kuralını görüyoruz:

```
show(A)
```

$t \mapsto -e^{(-|t|)} + 1$

Burada da SageMath'in bize $A(t)$'yi göstermesini istiyoruz:

```
show(A(t))
```

$-e^{(-|t|)} + 1$

Ve bazı sayısal örnekler:

```
A(0)
```

```
0
```

```
A(ln(3/4)).show()
```

$\frac{1}{4}$

```
A(5)
```

```
-e^(-5) + 1
```

```
A(5).n()
```

```
0.993262053000915
```

Örnek 2.17.2 Şimdi de aşağıdaki fonksiyonu tanımlayalım:

$$h(x) = \frac{x+2}{x-2}$$

```
h(x)=(x+2)/(x-2)
```

Bu fonksiyonun, kendisiyle 4 defa bileşkesini, yani aşağıdaki ifadeyi bulalım:

$$h\,(h\,(h\,(h\,(h\,(x)))))$$

```
h(h(h(h(h(x))))).simplify_full().show()
```

$-\frac{7\,x-58}{29\,x-94}$

Alıştırma 2.17.1 $f(x) = 2 - \frac{3}{x}$ *ve* $g(x) = 3 - \frac{2}{x}$ *fonksiyonları veriliyor.*

$$f\,(f\,(g(g(x))))$$

bileşke fonksiyonunu SageMath ile hesaplayın.

2.18 Sembolik İfadeler

SageMath, kıyaslama operatörü (==) kullanarak yazdığımız `1+1 == 2` ifadesine hemen `True` (*doğru*) diye cevap veriyor. Peki `x+x == 2*x` ifadesi için ne diyecek bakalım:

```
x+x == 2*x
```

```
2*x == 2*x
```

`True` ya da `False` gibi bir sonuç elde etmedik. Bu ifadenin her x değeri için doğru olduğunu, yani eşitliğin sağ ve sol taraflarının özdeş olduğunu biliyoruz. SageMath de bunu biliyor. Burada, daha önce 2.15 numaralı alt bölümde bahsetmiş olduğumuz `bool` komutunu kullanarak bu sorgulamayı yapabiliriz.

```
bool( x+x == 2*x )
```

```
True
```

Yerine Koyma

En yaygın işlemlerden bir diğeri de yerine koyma işlemidir. Sayısal bir örnek:

$$S = w^3 - 1$$

denkleminde $w = 10$ alarak S'yi hesaplayalım:

```
var("w")
S = w^3-1
S(w=10)
```

```
999
```

Yerine koyma için iki yöntem daha var. `subs` ve `substitute` (*yerine koy*) komutları aşağıdaki gibi kullanılabilir:

```
var("w")
S = w^3-1
S.subs(w=10)
```

```
999
```

```
var("w")
S = w^3-1
S.substitute(w=10)
```

```
999
```

Birden fazla değişken için benzer bir örnek:

```
var("x y")
K = 2*x + 3*y
K( x = 3 , y = -1 )
```

```
3
```

Yerine koyma işleminde sembolik ifadelerle de işlem yapılabilir:

```
var("x y a b")
ifade = 2*x^4 + 3*y^3
ifade.subs(x = 1-a^2, y = a^2-b ).expand()
```

```
2*a^8 - 5*a^6 - 9*a^4*b + 12*a^4 + 9*a^2*b^2 - 3*b^3 - 8*a^2 + 2
```

Not 2.18.1 Bu noktada, `f(x)=...` ile `f=...` tanımlamaları arasındaki farkı söyleyelim. Aslında ikisi ile de yerine koyma, limit, türev, integral, denklem çözme ve grafik çizme işlemleri yapılabilir. Ama yine de SageMath'e göre ilki bir fonksiyon, diğeri değil. İkincisine sembolik bir atama diyebiliriz. Bu farkın kendini gösterdiği bir duruma örnek olarak aşağıdaki tanımlamaları düşünelim:

```
g1(x)=7
g2=7
```

Şimdi `g1(10)` ve `g2(10)` komutlarını deneyin. İlki bir fonksiyon (sabit fonksiyon) olduğundan `g1(10)` komutuna yanıt alabiliyoruz. Ancak `g2` sadece bir sabit olduğundan `g2(10)` ifadesi hata vermektedir.

Adım Adım Denklem Çözmek

Bir denklemin her iki tarafını aynı anda, kağıt kalem kullanmadan adım adım yeni işlemlere tabi tutabiliriz. Küçük bir örnek:

```
denklem = 3*x+1==6*x-5
show(denklem)
denklem=denklem-1
show(denklem)
denklem=denklem-6*x
show(denklem)
denklem=denklem/-3
show(denklem)
```

$3x + 1 = 6x - 5$
$3x = 6x - 6$
$-3x = -6$
$x = 2$

Notlar

Bu bölümü birkaç not ile kapatıyoruz:

Not 2.18.2 Kullandığınız SageMath yazılımının versiyonunu `version()` komutuyla öğrenebilirsiniz:

```
version()
```

```
'SageMath version 9.7, Release Date: 2022-09-19'
```

Not 2.18.3 Bir komutu yazarken tab tuşuna basıp olası diğer komutları da görebilirsiniz. Bu çok etkili özellik SageMathCell'de yok. CoCalc ve tabii ki de bilgisayarda kurulu SageMath'te var. Örnek olarak `fac` yazıp tab tuşuna basın ve `fac` ile başlayan iki olası komutu görün: `factor` ve `factorial`.

Not 2.18.4 Bir komutla ilgili yardım almanın bir yolu komutun sonuna tek ya da çift soru işareti koymak: `factor?` veya `factor??` gibi. Çift soru işareti daha detaylı yardım, açıklama ve örnekler veriyor. Ancak bu belgelerin İngilizce olduğunu belirtelim.

Not 2.18.5 Aşağıda daha önce gördüğümüz bazı komutların alternatif kullanımı için birkaç örnek var. Aslında bu alternatif kullanımı kabul eden pek çok komut mevcut.

```
a=12
a.factor(), a.factorial(), a.divisors(), (a^55+5).is_prime(), a.sqrt()
```

```
(2^2 * 3, 479001600, [1, 2, 3, 4, 6, 12], True, 2*sqrt(3))
```

Bu komut alternatifinin avantajı tab tuşu ile birlikte etkili bir kullanımının olması. Noktayı koyduktan sonra tab tuşuna basınca mevcut nesne için olası komutları görebiliyoruz.

Bölüm 3

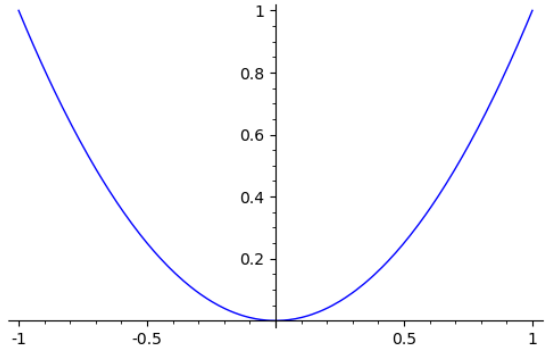
Grafik I

Bu bölümde temel grafik işlemlerini inceleyeceğiz. Asıl odağımız `plot` komutu ve bu komut için yaygın kullanılan seçenekler olacaktır. Daha ileri düzey grafik işlemleri 5 numaralı bölümde.

3.1 `plot` Komutu ile İlk Grafik

Hızlı bir giriş yapalım ve $y = x^2$ fonksiyonunun grafiğini çizdirelim:

```
plot(x^2)
```



Yukarıda SageMath otomatik olarak, grafiğini çizdireceğimiz fonksiyonun x'e bağlı olduğunu ve x aralığının $(-1, 1)$ olduğunu varsayıyor. Siz de benzer örneklerle bunu deneyebilirsiniz. Şimdi `plot` komutunun bazı yaygın kullanılan seçeneklerini inceleyelim.

Not 3.1.1 x'ten farklı bir değişken kullanmak istersek, bunu `var` komutuyla tanımlamamız gerekir. Örneğin:

```
var("t")
plot(t^2)
```

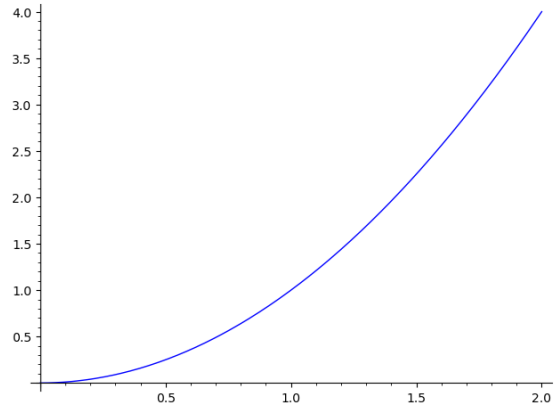
3.2 Grafik Aralıkları

Mesela yukarıdaki fonksiyonun grafiğini $(0, 2)$ aralığında çizdirelim. Bunun için aşağıdakilerden herhangi biri kullanılabilir.

```
plot( x^2 , 0 , 2 )
```

```
plot( x^2, (0,2) )
```

```
plot( x^2, (x,0,2) )
```

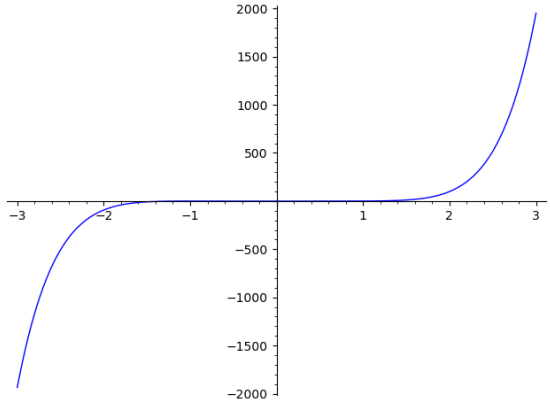


Görüldüğü üzere bir $y = f(x)$ fonksiyonu için x aralığını istediğimiz gibi giriyoruz, SageMath de bu aralığa karşılık gelen y değerlerini göstermeye çalışıyor. Ama bazı durumlarda y değerleri çok küçük ya da çok büyük, grafiğin detayları ise anlaşılmaktan çok uzak olabilir. Örneğin

$$y = (x^5 + 1)(x^2 - 1)$$

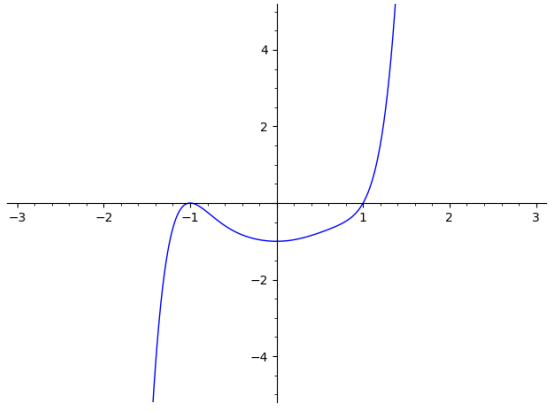
fonksiyonunu $(-3, 3)$ aralığında çizdirelim:

```
plot( (x^5+1)*(x^2-1), (x,-3,3) )
```



Yukarıdaki grafikte y aralığına bir sınırlama koymadığımızdan, -2000 ile 2000 gibi sayılar arasında bir görüntü elde ettik. Bu şekilde, fonksiyonun kök değerlerine yakın bölgelerindeki davranışını anlamak olanaksız. Aşağıda y aralığını sınırladığımızda çok daha anlaşılır bir grafik elde ediyoruz.

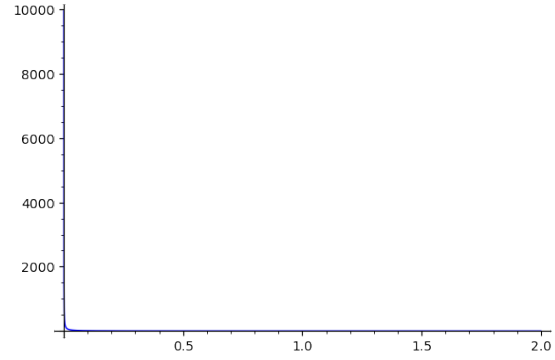
```
plot( (x^5+1)*(x^2-1), (x,-3,3), ymax=5 , ymin=-5 )
```



Not 3.2.1 Yatay eksen aralığı için de aralık vermek yerine `xmin=-3` ve `xmax=3` komutları kullanılabilir.

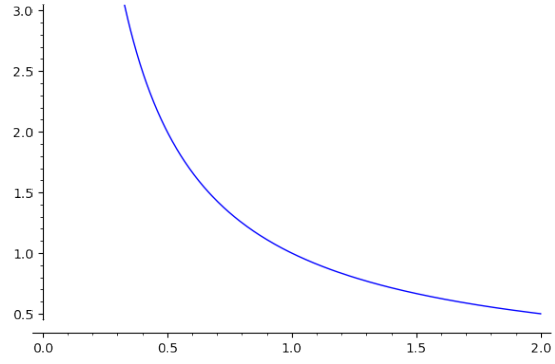
Benzer bir örnek daha: $y = \frac{1}{x}$ fonksiyonunu $(0, 2)$ aralığında çizdirelim:

```
plot(1/x, (x,0,2) )
```



Yukarıdaki bu anlaşılmaz grafiği elde etmemizin sebebi, fonksiyonun $x = 0$'a yakın pozitif değerlerine karşılık çok büyük değerler alıyor olmasıdır. Kimse böyle bir grafik görmek istemez. Aşağıda en büyük y değerini 3 olarak sınırladığımızda çok daha açık bir grafik elde ediyoruz.

```
plot(1/x, (x,0,2), ymax=3 )
```

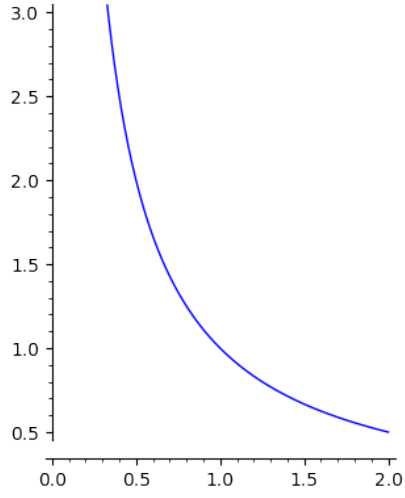


Elde ettiğimiz bu grafik daha anlaşılır olsa da en ve boy ölçeklendirmelerinin farklı olduğunu görüyoruz. Bunu, SageMath otomatik olarak anlaşılır bir oran olarak öneriyor. Ama bazı durumlarda grafiği 1:1 oranıyla ya da farklı oranlarda görmek isteyebiliriz. Bunu nasıl yapacağımız aşağıdaki alt bölümde veriliyor:

3.3 Boy/En Oranı

Boy/en oranını kendimiz girmek istersek `aspect_ratio` komutunu aşağıdaki gibi kullanabiliriz:

```
plot(1/x, (x,0,2), ymax=3, aspect_ratio=1 )
```

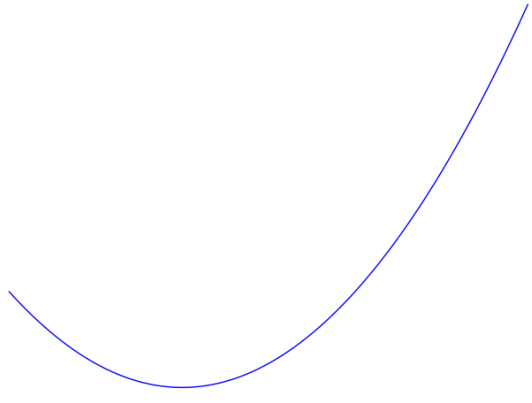


Yukarıdaki örnekte boy/en oranı 1 olarak seçilmiştir. 1 yerine 1'den küçük ve büyük pozitif sayılar yazarak bu iki farklı durumu gözlemleyebilirsiniz.

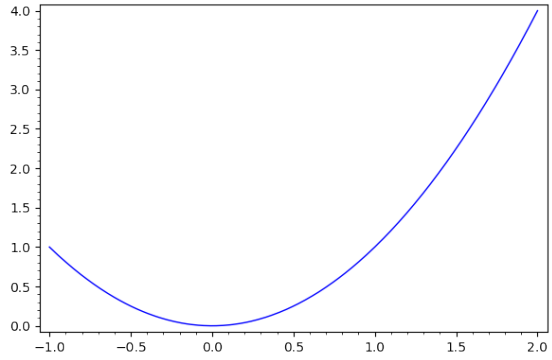
3.4 Eksenler ve Çerçeve

Bazı durumlarda eksenleri (*axes*) ve/veya çerçeveyi (*frame*) göstermek veya gizlemek isteyebilirsiniz. Aşağıdaki örnekler ilgili komutlarla verilmiştir. Anlaşılacağı üzere göstermek için `True`, gizlemek için `False` kullanıyoruz.

```
plot(x^2, (x,-1,2), axes=False )
```



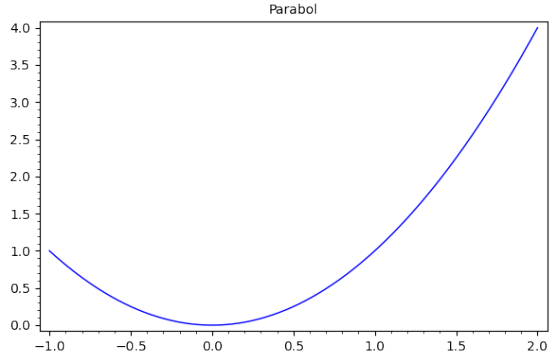
```
plot(x^2, (x,-1,2), axes=False, frame=True )
```



3.5 Grafik Başlığı

Şimdi de yukarıdaki grafiğe bir başlık ekleyelim:

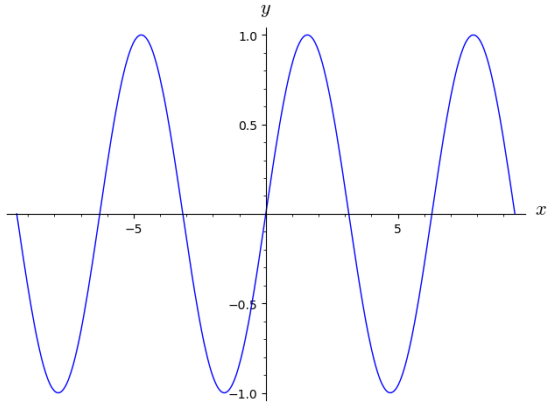
```
plot(x^2, (x,-1,2), axes=False, frame=True, title="Parabol" )
```



3.6 Eksenleri Etiketleme

Grafiklere eksen etiketi eklemek istersek bunu `axes_labels` komutuyla yapabiliriz. Aşağıdaki örnekte x ve y eksen etiketleriyle $y = \sin x$ fonksiyonunun grafiği verilmiştir. Matematiksel LaTeX ifadesini iki dolar işareti ($) arasına yazdığımıza dikkat edin.

```
plot( sin(x) , (x,-3*pi,3*pi), axes_labels=["$x$","$y$"] )
```



3.7 Grafik Boyutu

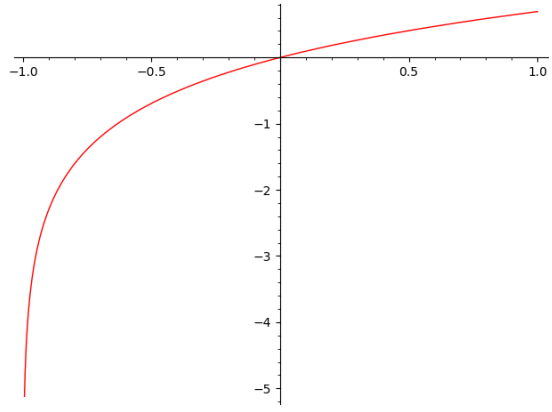
Grafik boyutunu *figure size* ifadesinin kısaltılmış hali olan `figsize` komutu ile giriyoruz. Aşağıdaki komutta grafik boyutuyla oynayıp SageMath'in bu sayılardan ne anladığını kendiniz keşfedebilirsiniz.

```
plot( x^2, figsize=7 )
```

3.8 Renk Seçenekleri

SageMath renk yönetimi konusunda oldukça tatmin edici. Kırmızı (red), mavi (blue), sarı (yellow), yeşil (green), turuncu (orange) gibi yaygın olarak kullanılan renkleri sadece renklerin isimlerini tırnak içinde girerek çok kolay bir şekilde grafiğe atayabiliriz. Örneğin:

```
plot( log(1+x), color="red" )
```



RGB (red, green, blue) renk modelinde, bu üç rengin her biri için 256 (0'dan 255'e) farklı değer vardır ve bunların her birinin farklı seçimi farklı bir renk vereceğinden, toplamda $256 \times 256 \times 256 = 16,777,216$ farklı renk elde edilebilir. Tabii ki her renge ait bir isim olmadığından bazı renkleri sayılarla ifade etmek akıllıca olacaktır. SageMath R, G ve B değerleri için [0,1] aralığında sayılar kullanarak renkleri oluşturur. Sıfır değeri o rengin hiç olmaması, 1 değeri ise tam olması demektir. Mesela tam bir kırmızı elde etmek için, RGB değerlerini $(1, 0, 0)$ olarak seçmeliyiz. Böylece R tam olacak, G ve B hiç olmayacak.

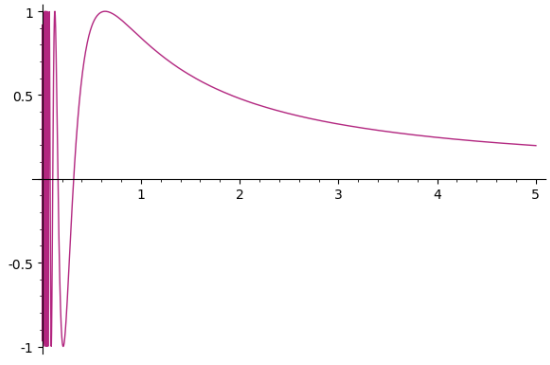
Yukarıdaki grafiği elde etmenin bir başka yöntemi aşağıdaki satır. Kırmızı rengini `"red"` şeklinde değil de, $(1, 0, 0)$ ifadesi ile tanımlıyoruz:

```
plot( log(1+x), color=(1,0,0) )
```

Ara değerleri kullanarak karışım denemeleri yapabilirsiniz. Ancak unutmamak gerekir ki bunlar pigment değil, optik renkler. Yani yağlı boyada olduğu gibi karışım sonuçları beklemeyin. Pigment ve optik renkler ana ve ara renkleri de dahil olmak üzere oldukça farklı yapı ve işleyişteler. Konumuzun dışına çıkmadan devam edelim. İsterseniz Renk Teorisi üzerine daha fazla bilgi için bir Google araması yapabilirsiniz.

Aşağıdaki örnekte RGB değerlerini $(0.7, 0.15, 0.5)$ şeklinde seçiyoruz:

```
plot( sin(1/x), (x,0,5) , color=(.7,.15,.5) )
```



RGB renk modeli çok fazla seçenek sunsa da uygun renk aramak çok da kolay olmayacaktır. Bunun için SageMath bize daha kısıtlı sayıda ama kullanışlı bazı ön tanımlı renkler öneriyor. Aşağıdaki gibi `colors` komutunu çalıştırınca renk ismi ve RGB koduyla birlikte uzunca bir renk listesi göreceksiniz.

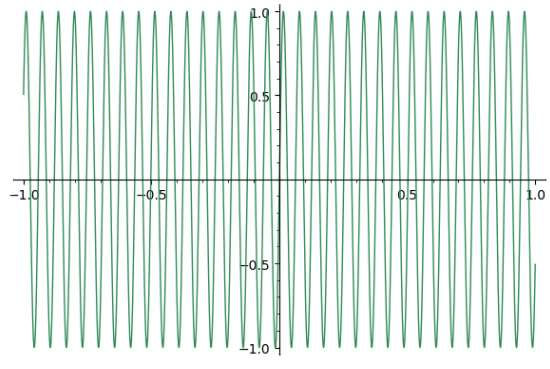
```
colors
```

Sadece renk isimlerinin listesini görmek için:

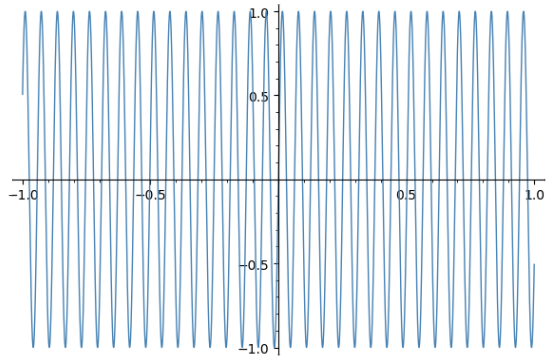
```
sorted(colors)
```

Aşağıdaki iki örnekte renk listesinden seçilen iki farklı renk ismi kullanıldı. İstersenız RGB kodunu da kullanabilirsiniz.

```
plot( sin(100*x), color="seagreen" )
```




```
plot( sin(100*x), color="steelblue" )
```



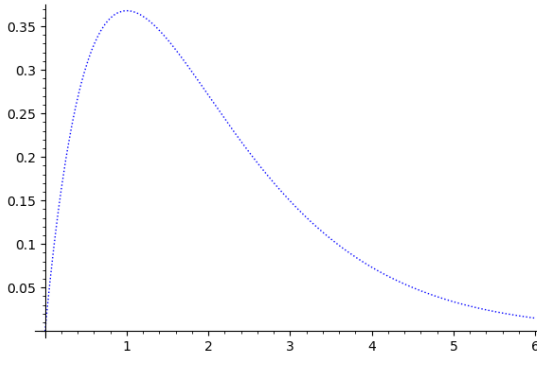
Not 3.8.1 Bir diğer renk modeli de HSV, yani Hue-Saturation-Value (Ton-Doygunluk-Değer). Bu modeli grafik seçeneklerinde `color` yerine `hue` yazarak ve benzer şekilde [0, 1] aralığında üçlüler alarak test edebilirsiniz. Bazı durumlarda bu modelin kontrolü RGB modelinden daha kolay. Aşağıdaki gibi denemeler yapabilirsiniz.

```
plot( sin(100*x), hue=(.1,.6,.8) )
```

3.9 Çizgi Stili

Grafiklerde `linestyle` komutu kullanılarak farklı çizgi stilleri seçilebilir.

```
plot( x*exp(-x), (x,0,6), linestyle = ":" )
```

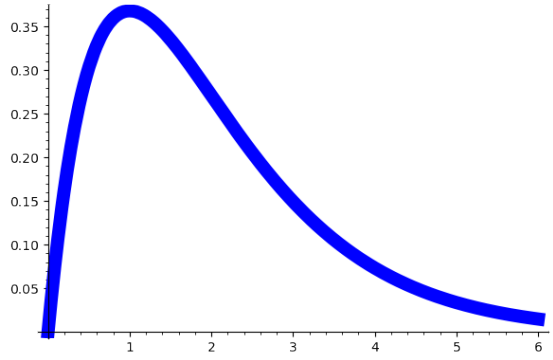


Yukarıdaki `":"` (iki nokta) ifadesi grafiği nokta nokta çizdiriyor. Onun yerine `"-."` (bir tire, bir nokta) ya da `"--"` (çift tire) ifadelerini koyarak farklı stiller görebilirsiniz. Varsayılan stil, yani normal çizgi stili için şunu kullanıyoruz: `"-"` (tire).

3.10 Çizgi Kalınlığı

Çizgi kalınlığı `thickness` (*kalınlık*) komutuyla aşağıdaki gibi belirtilir:

```
plot( x*exp(-x), (x,0,6), thickness=10 )
```



Not 3.10.1 Pek çok komut için kullandığımız alternatif yapıyı burada `plot` komutu için de kullanabiliriz. Aşağıdaki iki kod aynı işlevi görmektedir:

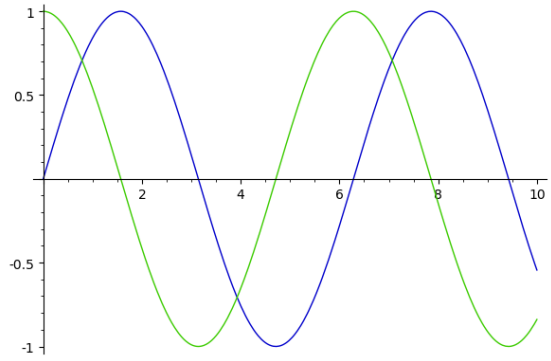
```
f=x*exp(-x)
plot( f , (x,0,6), linestyle = ":", thickness=10 , color="green" )
```

```
f=x*exp(-x)
f.plot( (x,0,6), linestyle = ":", thickness=10 , color="green" )
```

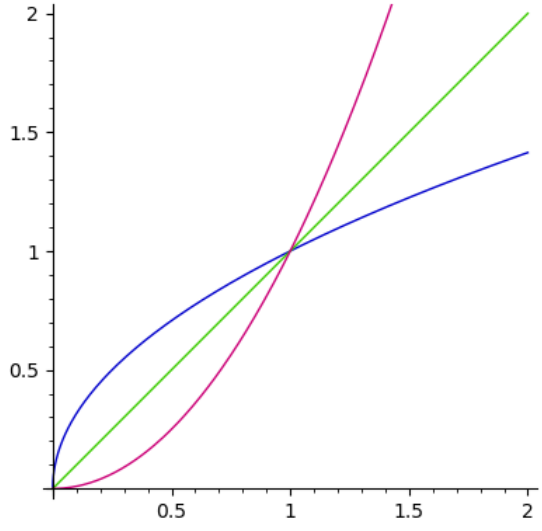
3.11 Grafik Katmanları

En fazla ihtiyaç duyulan durumlardan biri de birden fazla grafiği aynı koordinat sisteminde görmektir. Aşağıda birkaç açıklayıcı örnek verilmiştir:

```
plot( [ sin(x) , cos(x) ] , (x,0,10) )
```

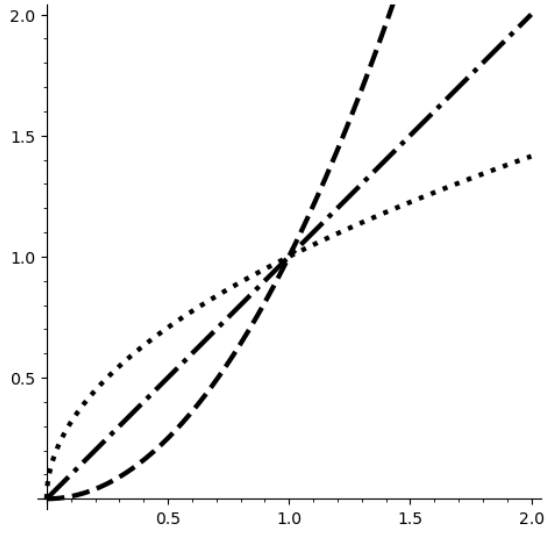



```
plot( [ sqrt(x) , x , x^2 ] , (x,0,2) , ymax=2, aspect_ratio=1 )
```



2.14.3 numaralı notta belirtilen += komutunun kullanımı grafikler için de geçerlidir. Yani aşağıda kullanıldığı gibi, mevcut grafik değişkenine yeni grafik ekleyerek güncelleme yapabiliyoruz.

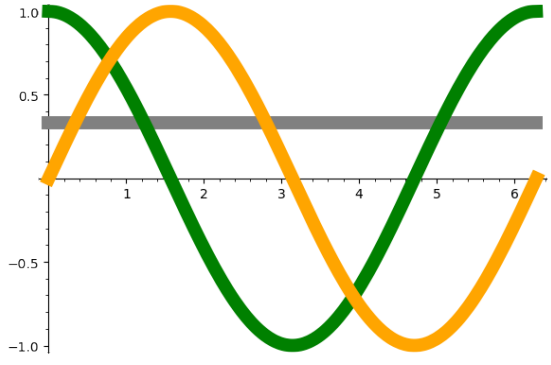
```
grafik = plot( sqrt(x), (x,0,2), color="black", thickness= 3, linestyle=
    ":")
grafik += plot( x, (x,0,2), color= "black" , thickness=3 ,linestyle=
    "-.")
grafik += plot( x^2 , (x,0,2), color= "black" , thickness=3 , linestyle
    = "--")
show( grafik, ymax=2 , aspect_ratio=1)
```



Üst Üste Grafikler

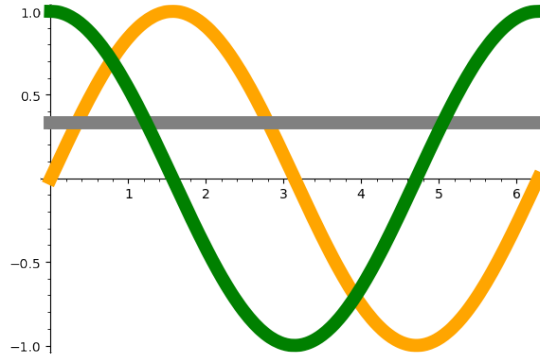
Grafik katmanlarının hangi sırayla üst üste geleceklerini `zorder` komutuyla belirtiyoruz. Kullanılan sayının büyük olması üstte olmasıyla özdeşleştirilmiş. Aşağıdaki örnekte 20, 15, 10 sırası kullanılarak en üstte `grafik1`, ortada `grafik2`, en altta da `grafik3` gösteriliyor:

```
grafik1= plot( sin(x), (x,0,2*pi),color="orange",thickness=10, zorder=20)
grafik2= plot( cos(x), (x,0,2*pi),color="green", thickness=10, zorder=15)
grafik3= plot( 1/3, (x,0,2*pi),color="gray", thickness=10, zorder=10)
show( grafik1 + grafik2 + grafik3 )
```



Aşağıdaki örnekte de `grafik2` en üstte, `grafik3` ortada ve `grafik1` en altta görüntüleniyor:

```
grafik1= plot( sin(x), (x,0,2*pi),color="orange",thickness=10, zorder=6)
grafik2= plot( cos(x) , (x,0,2*pi),color="green", thickness=10, zorder=8)
grafik3= plot( 1/3 , (x,0,2*pi),color="gray", thickness=10, zorder=7)
show( grafik1 + grafik2 + grafik3 )
```

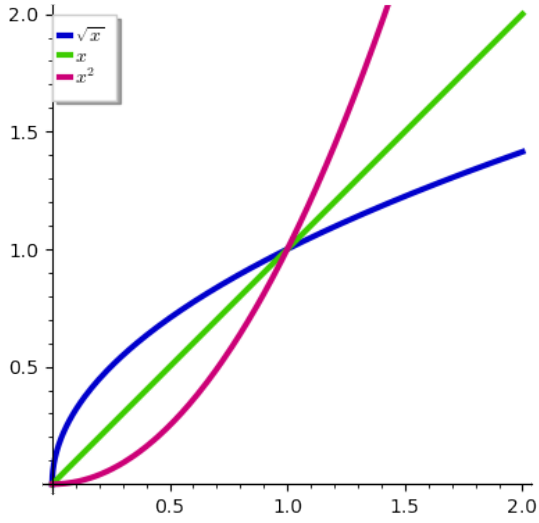


Görüldüğü üzere, `zorder` komutuna atanan sayının kaç olduğu değil, diğerleriyle büyüklük kıyası önemlidir. Büyükten küçüğe giden `zorder` sayılarına karşılık yukarıdan aşağıya giden grafik katmanları gelmektedir.

3.12 Gösterge Ekleme

Birden fazla grafik ögesinin olduğu durumlarda, gösterge kullanmak genellikle açıklayıcı olur. Bunun için `legend_label` komutunu grafik listesiyle aynı sırada olacak şekilde yine bir liste olarak giriyoruz.

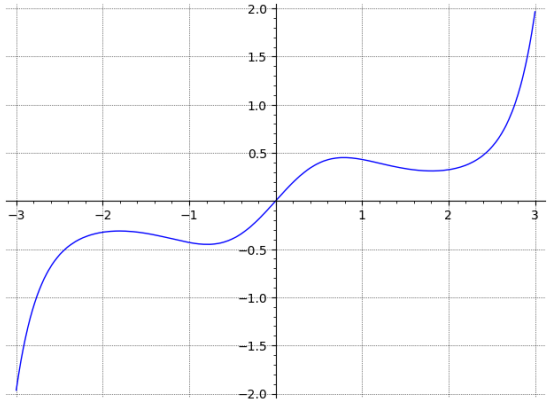
```
plot( [ sqrt(x) , x , x^2 ] , (x,0,2) , ymax=2, thickness=3,
    aspect_ratio=1 , legend_label=[ r"$\sqrt{x}$", "$x$" , "$x^2$" ] )
```



3.13 Izgara Çizgileri

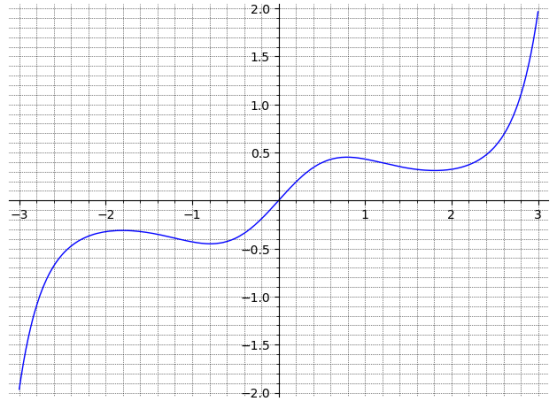
Grafiğe ızgara (grid) çizgileri eklemek için `gridlines` komutunu kullanıyoruz. Aşağıdaki örnekleri inceleyelim:

```
plot( x*exp(-x*sin(x)), (x,-3,3), gridlines=True )
```



Daha sık aralıklı ızgara çizgileri için:

```
plot( x*exp(-x*sin(x)), (x,-3,3), gridlines="minor" )
```



Eğer ızgara çizgisinin sadece belli değerlerde görünmesini istiyorsanız:

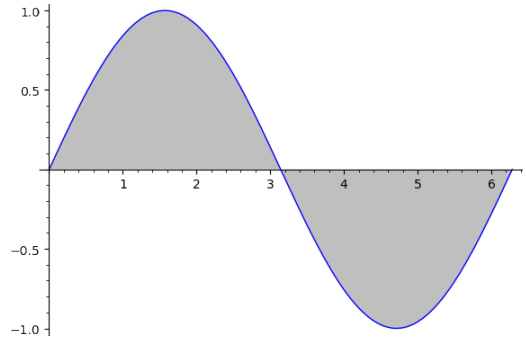
```
plot( x*exp(-x*sin(x)), (x,-3,3), gridlines=[ [1,1.2],[-1,0,1] ] )
```

Bu durumda, ızgara çizgileri yatay eksen için sadece 1 ve 1.2 noktalarından, dikey eksen için de -1, 0 ve 1 noktalarından geçecektir.

3.14 Grafik Bölgesini Tarama

Bir grafiğin yatay eksen ile arasında kalan bölgeyi taramak için seçeneklere `fill=True` ifadesini eklemek yeterlidir:

```
plot( sin(x), (x, 0, 2*pi), fill=True)
```



Dolgu rengini `fillcolor` komutu ile belirleyebilirsiniz. Aşağıdaki kod satırını çalıştırıp fonksiyonun siyaha, dolgu renginse yeşile boyandığını görünüz:

```
plot( sin(x), (x, 0, 2*pi), fill=True, color="black", fillcolor="green")
```

Bölüm 4

Temel Kavramlar II

2 numaralı bölümün devamı niteliğinde olan bu bölümde SageMath ve matematiğin daha ileri düzey kavramları verilecektir.

4.1 Harf Dizinleri

Programlamada, metin de diyebileceğimiz harf dizinleri (string) sık sık kullanılsa da bizim odaklanacağımız sadece birkaç durum olacak. Bunları örneklerle görelim:

Bir metni ekrana yazdırmak için, `print` ve `show` komutlarını kullanabiliriz. İlgili metni iki kesme işareti ya da tırnak işareti arasına alıyoruz. Örneğin:

```
print("SageMath ile Matematik")
```

```
SageMath ile Matematik
```

```
show("SageMath ile Matematik")
```

SageMath ile Matematik

Metin içinde kesme veya tırnak işaretleri kullanılacaksa, hemen öncesinde kaçış karakteri (ters eğik çizgi) kullanabiliriz:

```
show("Mozart\'ın \"Figaro\'nun Düğünü\" operasının ilk gösterimi
    1786\'daymış.")
```

```
Mozart'ın "Figaro'nun Düğünü" operasının ilk gösterimi 1786'daymış.
```

Şimdi de, metin ve değişken içeren bir cümleyi ekrana yazdıralım. Bunun için iki yöntem göreceğiz. Biri `f"..."` komutunu kullanıp değişkenleri küme parantezine almak:

```
isim= "Hamit"
meyve= "ayva"
kilo= 2
print(f"{isim} pazardan {kilo} kilo {meyve} aldı.")
```

```
Hamit pazardan 2 kilo ayva aldı.
```

Diğer yöntem de metinleri tırnak içinde yazıp metin ve değişkenleri virgül ile ayırmak:

```
isim= "Hamit"
meyve= "ayva"
kilo= 2
print(isim,"pazardan",kilo,"kilo",meyve,"aldı.")
```

```
Hamit pazardan 2 kilo ayva aldı.
```

Metin, matematiksel ifade ve de değişkeni aynı çıktıda görmek isteyeceğimiz durumlarda LaTeX ifadesini `LatexExpr(r"...")` komutu ile gösterebiliriz.

```
x=sqrt(2)
show( LatexExpr(r"x="),x," ise,",LatexExpr(r"\ x^2="),x^2,"\'dir." )
```

$x = \sqrt{2}$ ise, $x^2 = 2$'dir.

4.2 Liste

Sonlu ve sıralı elemanlar topluluğuna *liste* diyoruz. Listede elemanlar birden çok sayıda kullanılabilir. Bir liste tanımlayalım:

```
L = [1,4,2,-5,0,127,0,11,127,5]
```

Görüldüğü üzere, listeleri oluştururken köşeli parantez kullanıyoruz. İlk olarak bu listenin uzunluğunu bulalım. Uzunluktan kastımız kaç elemana sahip olduğudur:

```
len(L)
```

```
10
```

Listemizde 10 eleman varmış. Kullanılan komut İngilizcede uzunluk anlamına gelen *length* kelimesinin ilk üç harfi.

SageMath indisleme yaparken 0'dan başlar. Yani listedeki ilk elemanın ne olduğunu soracak olursak, aşağıdaki komutu yazmamız gerekir:

```
L[0]
```

```
1
```

Şimdi de indisi 3 olan elemandan (dahil), indisi 6 olan elemana (hariç) kadar olanları listeleyelim:

```
L[3:6]
```

```
[-5, 0, 127]
```

Listelerde çok etkili bir adresleme de negatif sayılarla yapılıyor. Liste sonundaki eleman `L[-1]` ile çağrılıyor.

```
L[-1]
```

```
5
```

Benzer şekilde `L[-2]`, `L[-3]` gibi komutları denemek size kalsın.

Şimdi de listenin sonuna 55 sayısını ekleyelim. Bunun için *eklemek, ek yapmak* anlamına gelen `append` komutunu kullanıyoruz. Aşağıdaki gibi:

```
L.append(55)
L
```

```
[1, 4, 2, -5, 0, 127, 0, 11, 127, 5, 55]
```

Diyelim ki `L` listesinden baştan dördüncü eleman olan -5 sayısını çıkarıp yerine 100 yazmak istiyoruz. Bunun için -5'in indisi olan 3 sayısını kullanıyor ve aşağıdaki gibi değişikliği yapıyoruz:

```
L[3]=100
L
```

```
[1, 4, 2, 100, 0, 127, 0, 11, 127, 5, 55]
```

Eğer arada bir yerlere yeni bir eleman eklemek istersek *arasına koymak, eklemek* anlamındaki `insert` komutunu aşağıdaki gibi kullanabiliriz:

```
L.insert(1,13)
L
```

```
[1, 13, 4, 2, 100, 0, 127, 0, 11, 127, 5, 55]
```

Listedeki 4 indisli elemanı silmek için `del` komutu (*silmek* anlamındaki *delete* kelimesinin kısaltılmış hali) kullanılabilir:

```
del L[4]
L
```

```
[1, 13, 4, 2, 0, 127, 0, 11, 127, 5, 55]
```

Şimdi de birden fazla listeyi nasıl birleştirebileceğimizi görelim. Bunun için, önce yeni bir liste tanımlayalım:

```
K=["abc","klm",3,4,5]
```

Aşağıdaki komut iki listeyi verilen sırayla birleştiriyor.

```
L+K
```

```
[1, 13, 4, 2, 0, 127, 0, 11, 127, 5, 55, "abc" , "klm" , 3, 4, 5]
```

Aşağıda sık kullanılan bazı listeleri pratik bir şekilde nasıl elde edebileceğimizi görüyoruz. Örneğin 0'dan 7'ye kadar olan sayıları sırasıyla eleman kabul eden bir liste tanımlayalım:

```
M=list(range(8))
M
```

```
[0, 1, 2, 3, 4, 5, 6, 7]
```

Burada, `range` komutu 0'dan başlayarak ilk 8 tam sayıyla liste oluşturuyor. Komutun buradaki kelime anlamı, *menzil, sıra* ya da *dizmek*. `list` komutunu kullanarak da *listele* demek istiyoruz.

Aşağıda elemanları 6'dan (dahil) 66'ya (hariç) kadar sırasıyla 10'ar 10'ar sayarak elde edilen bir liste var:

```
list(range( 6,66,10 ))
```

```
[6, 16, 26, 36, 46, 56]
```

Elemanları 6'dan (dahil) 13'e (hariç) kadar 1.6'lık ondalık adımlarla elde edilen listeyi de aşağıdaki gibi `srange` komutuyla elde edebiliriz:

```
list(srange(6,13,1.6))
```

```
[6.00000000000000,
 7.60000000000000,
 9.20000000000000,
 10.8000000000000,
 12.4000000000000]
```

Sık kullanılan pratik bir liste elde etme yöntemi de şöyle: Diyelim ki 3 ve 7 de dahil olmak üzere, 3'ten 7'ye kadar olan tam sayıları sırasıyla eleman kabul eden bir liste oluşturmak istiyoruz:

```
[3..7]
```

```
[3, 4, 5, 6, 7]
```

Henüz `for` döngüsünü (6.4 numaralı alt bölüm) görmemiş olsak da, çok temel bir liste üretme yönteminde kullanıldığı için burada ona değinmek uygun olacaktır. Örneğin aşağıdaki kod 1'den 5'e kadar olan sayıların karelerini içeren bir liste üretecektir:

```
[ i^2 for i in [1..5] ]
```

```
[1, 4, 9, 16, 25]
```

Son olarak kelime anlamı fermuar olan `zip` komutunu görelim:

```
x=[1..10]
y=[101..110]
```

```
list(zip(x,y))
```

```
[(1, 101),
 (2, 102),
 (3, 103),
 (4, 104),
 (5, 105),
 (6, 106),
 (7, 107),
 (8, 108),
 (9, 109),
 (10, 110)]
```

Liste sayısı sadece iki olunca bir fermuar yapısı oluşsa da ikiden fazla liste için de aynı komut kullanılabiliyor. Üçüncü bir `z` listesi tanımlayıp `list(zip(x,y,z))` komutunu deneyebilirsiniz.

Not 4.2.1 (Demet) Listeye çok benzer bir başka yapı da *demet (tuple)* dediğimiz veri yapısı. Parantez şekli listelerde kullandığımızdan farklı olarak, normal parantez: `A = (1,2,3,4)` gibi. Demetler *immutable* yani *değişmez* veri yapısı olarak bilinir. Listelerde özgürce yaptığımız, örneğin terim ekleme (`append`) gibi değiştirme işlemleri demetlerde yapılamaz. Demet olarak oluşturulabilecek matematiksel yapılardan biri vektördür.

4.3 Eşitlik Çözmek

Eşitlik çözmek için `solve` (yani, *çöz*) komutunu nasıl kullandığımızı aşağıdaki örnekte görebilirsiniz:

```
solve(2*x+1 == 7 , x )
```

```
[x == 3]
```

Görüldüğü gibi, eşitliği "==" komutunu kullanarak yazıyoruz ve bu eşitliği x değişkenine göre çözmek istediğimizi belirtiyoruz.

Benzer bir örnek:

```
solve( x^2+3*x-5==0, x )
```

```
[x == -1/2*sqrt(29) - 3/2, x == 1/2*sqrt(29) - 3/2]
```

Yukarıdaki denklemin çözümünü daha açık bir şekilde görmek istersek, çözümü bir değişkene atayıp `show` komutunu kullanabiliriz:

```
cozum = solve( x^2+3*x-5==0, x )
show(cozum)
```

$$\left[x = -\frac{1}{2}\sqrt{29} - \frac{3}{2}, x = \frac{1}{2}\sqrt{29} - \frac{3}{2}\right]$$

Elde Edilen Çözümü Kullanmak

Yukarıda elde ettiğimiz çözümlerin bir liste olarak verildiğini görüyoruz. Böylece bu çözümler gerektiği yerde liste işlemleriyle kullanılabilir.

```
cozum[0]
```

```
x == -1/2*sqrt(29) - 3/2
```

```
cozum[1]
```

```
x == 1/2*sqrt(29) - 3/2
```

Diyelim ki yukarıda elde ettiğimiz kökleri aşağıdaki G fonksiyonu için kullanmak istiyoruz.

$$G(x) = x^4 + 2\,x^3 - 15\,x^2 - 16\,x + 36$$

```
G(x) = x^4 + 2*x^3 - 15*x^2 - 16*x + 36
G(x).subs( cozum[0] ).simplify_full()
```

```
1
```

```
G(x) = x^4 + 2*x^3 - 15*x^2 - 16*x + 36
G(x).subs( cozum[1] ).simplify_full()
```

```
1
```

Not 4.3.1 (Denklemin Solu ve Sağı) Bir denklemin solundaki ya da sağındaki ifadeyi sırasıyla `lhs` (left hand side) ve `rhs` (right hand side) komutlarıyla belirtebiliriz.

```
denklem= x^2==7*x
show( denklem.lhs() )
show( denklem.rhs() )
```

x^2
$7x$

Örnek 4.3.1 Bu defa biraz daha sembolik bir şekilde, denklem çözmeye devam edelim.

$$2a + bc^2 = 100$$

denklemini sırasıyla a, b ve c'ye göre çözelim (yani yukarıdaki denklemde sırasıyla a, b ve c'yi çekelim demek istiyoruz). Tabii öncelikle değişkenleri `var` komutu ile tanımlamalıyız:

```
var(" a b c ")
cozum_a=solve( 2*a+b*c^2==100,a )
cozum_b=solve( 2*a+b*c^2==100,b )
cozum_c=solve( 2*a+b*c^2==100,c )
```

Şimdi de çözümleri `show` komutunu kullanarak görelim:

```
show(cozum_a)
```

$$\left[a = -\frac{1}{2}\, bc^2 + 50\right]$$

```
show(cozum_b)
```

$$\left[b = -\frac{2\,(a-50)}{c^2}\right]$$

```
show(cozum_c)
```

$$\left[c = -\sqrt{-\frac{2\,a}{b} + \frac{100}{b}}, c = \sqrt{-\frac{2\,a}{b} + \frac{100}{b}}\right]$$

Alıştırma 4.3.1 (İkinci Dereceden Denklemin Kökleri) *Aşağıdaki ikinci dereceden denklemin köklerini veren formülü SageMath kullanarak bulun:*

$$ax^2 + bx + c = 0$$

Not 4.3.2 Üçüncü ve dördüncü dereceden polinom denklemleri yazıp çözmeyi deneyebilirsiniz. Hatta formüllerini benzer şekilde bulabilirsiniz. Ama beşinci ya da daha yüksek dereceler için o kadar şanslı olmayabilirsiniz. Çünkü o tip denklemler için genel bir formül yok; sadece bazı özel durumlar için çözümler var. Tabii nümerik yaklaşımla oldukça etkili sonuçlar bulabiliyoruz. Bunu 4.5 numaralı alt bölümde göreceğiz.

Denklem Sistemi

Şimdi de basit iki örnekle, birden fazla denklemin (yani denklem sisteminin) ortak çözümünü nasıl bulabileceğimizi görelim:

```
var("y")
cozum=solve( [x^2==2*y , x == y+1 ], [x,y] )
show(cozum)
```

$[[x = (-i + 1), y = (-i)], [x = (i + 1), y = i]]$

```
var("y")
cozum=solve( [ 2*x^2+3*y^2==5 , 2*x+3*y==2 ] , [x,y] )
show(cozum)
```

$$\left[\left[x = -\frac{3}{10}\sqrt{14} + \frac{2}{5}, y = \frac{1}{5}\sqrt{14} + \frac{2}{5}\right], \left[x = \frac{3}{10}\sqrt{14} + \frac{2}{5}, y = -\frac{1}{5}\sqrt{14} + \frac{2}{5}\right]\right]$$

Not 4.3.3 `solve` komutu çözümü kusursuz vermeye çalışır. Eğer denklem(ler) doğrusal ise SageMath bize her zaman tam sonucu verecektir. Çözüm olmadığı durumda ise çözümün olmadığını söyleyecektir (7.14 numaralı alt bölüm). Doğrusal olmayan denklem ya da denklem sistemlerinin tam çözümlerine matematiksel yazılımlar pek çok durumda yanıt veremez; ancak uygun şartlarda nümerik yanıtlar verebilirler (4.5 numaralı alt bölüm). Bu yetersiz olma durumu yazılımlardan çok, matematiğin doğasıyla ilgilidir.

4.4 Eşitsizlik Çözmek

Eşitsizlikler de benzer şekilde `solve` komutuyla çözülüyor:

```
solve( -x < 8, x )
```

```
[[x > -8]]
```

Aşağıda $x \geq \frac{1}{x}$ eşitsizliğini çözüyoruz:

```
solve( x >= 1/x, x )
```

```
[[x >= -1, x < 0], [x >= 1]]
```

Yani çözüm şu şekilde: $[-1, 0) \cup [1, \infty)$

4.5 Nümerik Olarak Kök Bulma

Bu alt bölümde `solve` komutunun yetersiz kaldığı durumlar için nümerik yolla kök bulma yöntemine odaklanacağız. Örneğin aşağıdaki denklemi çözmeye çalışalım:

$$x = e^{-x^2}$$

```
solve( x==exp(-x^2), x )
```

```
[x == e^(-x^2)]
```

SageMath, denklemi olduğu gibi bıraktı. Bu durumda kökün nümerik yaklaşık değerini bulmak isteyebiliriz. Bunun için `find_root` (yani, *kök bul*) komutunu kullanırız. Ancak kök için, içinde bulunduğunu düşündüğümüz uygun bir tahmini aralık girmemiz gerekir. Açık bir şekilde yukarıdaki denklemin kökleri $y = x$ ve $y = e^{-x^2}$ fonksiyonlarının grafiklerinin kesiştiği noktaların apsisleri, yani x değerleridir. Aşağıdaki kod ile iki grafiği aynı koordinat sisteminde $(-2, 2)$ aralığında çizdirin ve kesişim noktasını görün.

```
plot( [exp(-x^2) , x ] , (x,-2,2) )
```

Kök için aralık önerisi olarak örneğin $(0, 2)$ alınabilir.

```
find_root(x==exp(-x^2), 0,2)
```

```
0.6529186404191585
```

Unutmayalım ki bu sonuç sadece yaklaşık bir değer olmakla birlikte, girilen aralıkta ancak kök olduğu durumda yanıt alabileceğiz. Ayrıca birden fazla kökün olduğu aralıklarda SageMath sadece bir kök verecektir. Yani aralıkları biraz dikkatli seçmeliyiz.

Benzer ve biraz daha detaylı bir örneği 8.4 numaralı alt bölümde bulabilirsiniz.

Aşağıda daha karmaşık denklemlerin köklerini yaklaşık olarak buluyoruz. Bu aşamada, denklemlerin sol ve sağ taraflarının grafiklerini aynı koordinat sisteminde çizdirmek, uygun aralık seçimi için etkili olacaktır.

```
find_root( sin(x+1) == x , -2 , 2 )
```

```
0.9345632107520242
```

```
find_root( x*log(x) == 1 , 1 , 2 )
```

```
1.7632228343518965
```

Alıştırma 4.5.1 *Aşağıdaki denklemin tüm reel köklerinin, mümkünse tam sonuçlarını, değilse yaklaşık değerlerini bulun:*

$$1 - (1 - x)\left(2 - x^2\right)\left(3 - x^3\right) = 0$$

Alıştırma 4.5.2 *π ile 2π arasında $\cos x = \frac{1}{3}$ denklemini sağlayan tüm x çözümlerinin yaklaşık değerini bulun.*

4.6 Varsayım Ekleme

Bazen belli varsayımlar altında işlemlerimizi yapmak isteyebiliriz; örneğin x sayısının negatif olduğunu varsaymak gibi. Bunu `assume` (yani *varsay*) komutuyla yapıyoruz. Bir örnek verelim:

```
assume(x<0)
solve(x^2==4,x)
```

```
[x == -2]
```

Girilen bir varsayımı SageMath'in unutması için `forget` (Türkçesi *unut*) komutunu kullanıyoruz:

```
forget(x<0)
solve(x^2==4,x)
```

```
[x == -2, x == 2]
```

4.7 Küme

Kümeler, listelerden farklı olarak eleman tekrarının ve sıranın önemsiz olduğu yapılar. Aşağıda bir A kümesi tanımlıyoruz. Küme oluşturmak için küme parantezi kullandığımıza dikkat edin.

```
A={8,8,8,1,2,3,4}
A
```

```
{1, 2, 3, 4, 8}
```

Görüldüğü üzere üç tane 8 kullanmamıza rağmen, SageMath bunlardan sadece birini alıp elemanları kendine uygun bir sıraya koyuyor. Burada sıranın bir öneminin olmadığını biliyoruz tabii ki.

Aşağıdaki küme sayı ve metin içeriyor:

```
B = {"bir","iki", 3 , "dört" }
B
```

```
{3, 'bir', 'dört', 'iki'}
```

En yaygın küme işlemlerinden birleşim, kesişim, fark ve simetrik fark işlemleri, sırasıyla `union()`, `intersection()`, `difference()` ve `symmetric_difference()` komutlarıyla yapılıyor:

```
A={2,4,6,8,10}
B={1,2,3,4,8,12}
A.union(B), A.intersection(B), A.difference(B), A.symmetric_difference(B)
```

```
({1, 2, 3, 4, 6, 8, 10, 12}, {2, 4, 8}, {6, 10}, {1, 3, 6, 10, 12})
```

Bir A kümesinin bütün alt kümelerini görmek için `list(subsets(A))` komutunu kullanmak yeterli.

```
D={"a","b","c"}
list(subsets(D))
```

```
[[], ['b'], ['a'], ['b', 'a'], ['c'], ['b', 'c'], ['a', 'c'], ['b', 'a',
    'c']]
```

Son olarak, bir listeyi bir kümeye nasıl dönüştürebileceğimizi görelim. Buradaki kasıt, tekrarlanan elemanlardan sadece bir tane olacak şekilde, listenin elemanlarını kullanarak sıranın önemli olmadığı bir yapı elde etmek.

```
G = [3,3,4,-3,1,-3]
H = set(G)
H
```

```
{-3, 1, 3, 4}
```

Not 4.7.1 Yukarıda bir liste olan G ve bir küme olan H için `G[1]` ve `H[1]` komutlarını deneyin. `G[1]` komutu listenin indisi 1 olan elemanını veriyorken, `H[1]` komutunun bir anlamının olmadığı açıktır.

4.8 Kombinasyon ve Permütasyon

Kombinasyon "kaç farklı seçim" sorusuyla ilgilenir. Aşağıdaki formül n'nin r'li kombinasyonunu verir:

$$C(n,r) = \frac{n!}{r!(n-r)!}$$

Kombinasyonu `binomial` komutu ile aşağıdaki gibi hesaplayabiliriz:

```
binomial(10,3)
```

```
120
```

Permütasyon ise "kaç farklı sıralama" sorusuna yanıt verir. n'nin r'li permütasyonu aşağıdaki formülle hesaplanır:

$$P(n,r) = \frac{n!}{(n-r)!}$$

Yukarıdaki iki formülden

$$P(n,r) = C(n,r)\,r!$$

olduğu açıktır. Örneğin 10'un 3'lü permütasyonu aşağıdaki gibi hesaplanabilir:

```
binomial(10,3)*factorial(3)
```

```
720
```

Not 4.8.1 Daha pratik olabilecek başka bir yöntem de kombinasyon için `comb`, permütasyon için de `perm` fonksiyonlarını aralarına virgül koyarak aşağıdaki gibi çağırmakla mümkün.

```
from math import comb, perm
```

Tamamdır. Şimdi `comb` ve `perm` komutlarını özgürce kullanabiliriz.

```
comb(10,3)
```

```
120
```

```
perm(10,3)
```

```
720
```

4.9 "Rastgele" Seçim

Bu alt bölümde belli bir kümeden rastgele eleman nasıl seçebileceğimizi çeşitli durumlar için inceleyeceğiz.

$[0, 1)$ Aralığı

$[0, 1)$ aralığından rastgele bir ondalık sayı (aslında kayan-noktalı sayı) seçmek için `random()` komutunu kullanıyoruz.

```
random()
```

```
0.6279331104946685
```

Tam Sayı Seçimi

Bir aralıktan rastgele tam sayı seçmek için `randint` komutunu kullanıyoruz. Aşağıda $\{1, 2, 3, \ldots, 9, 10\}$ kümesinden rastgele bir seçim yapılıyor. 1 ve 10 sayılarının da seçim için dahil edildiklerini vurgulayalım.

```
randint(1,10)
```

```
8
```

Ondalık Sayı Seçimi

12 ile 18 arasından rastgele bir ondalık sayı (aslında kayan-noktalı sayı) seçmek için aşağıdaki yöntem kullanılabilir:

```
12+6*random()
```

```
14.347287715005278
```

Yukarıda `6*random()` ifadesi 0-6 aralığında ondalık sayılar üretir (6 iki sayının farkından elde edildi). Bu seçimi 12 sayısına ekleyerek 12-18 aralığında bir sayı üretiyoruz.

Sonlu Bir Kümeden Rastgele Seçim

Son olarak, verilen bir listeden rastgele bir seçim yapalım. Bunun için `choice` komutunu kullanıyoruz.

```
choice( [1,2,3,"yes",6,7,"no",8,8,8] )
```

```
7
```

4.10 SageMath Çalışmanızı Paylaşmak

SageMath'te yazılmış bir kodu hızlı bir şekilde paylaşmanın bir yolu, ilk olarak

`sagecell.sagemath.org`

adresinde kodu *Evaluate* tuşuna basarak çalıştırıp, ekranın sağında *Share* (*paylaş*) tuşuna basmak. Aşağıdaki görseldeki gibi bir şey göreceksiniz. Permalink (kalıcı bağlantı), Short temporary link (kısa ve geçici bağlantı) ve ayrıca geçici bağlantının kare kodu da aşağıdaki gibi verilmekte:

4.11 SageMath ve LaTeX

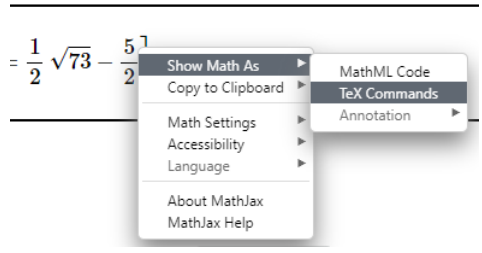
SageMath kullanarak elde ettiğiniz bir çıktıyı LaTeX ortamında nasıl kullanabileceğimizi görelim:

Aşağıda $x^2 + 5x - 12 = 0$ denkleminin çözümü önce `kokler` adında bir değişkene atanıyor, sonra da `show` komutuyla bir çıktı elde ediliyor:

```
kokler = solve(x^2+5*x-12==0,x)
show(kokler)
```

$$\left[x = -\frac{1}{2}\sqrt{73} - \frac{5}{2}, x = \frac{1}{2}\sqrt{73} - \frac{5}{2}\right]$$

Çıktıya sağ tıklayarak *TeX Commands* seçeneğiyle LaTeX komutunu elde edebilirsiniz.



Aynı örnek için bir diğer yöntem olan `latex` komutu da aşağıdaki gibi kullanılabilir.

```
kokler = solve(x^2+5*x-12==0,x)
latex(kokler)
```

```
\left[x = -\frac{1}{2} \, \sqrt{73} - \frac{5}{2}, x = \frac{1}{2} \,
    \sqrt{73} - \frac{5}{2}\right]
```

Not 4.11.1 Eğer bir grafik çıktısını LaTeX ortamında kullanmak istiyorsanız grafiği kaydetmeniz gerekecektir. Bunu nasıl yapacağınız 5.12 numaralı alt bölümde.

Bölüm 5

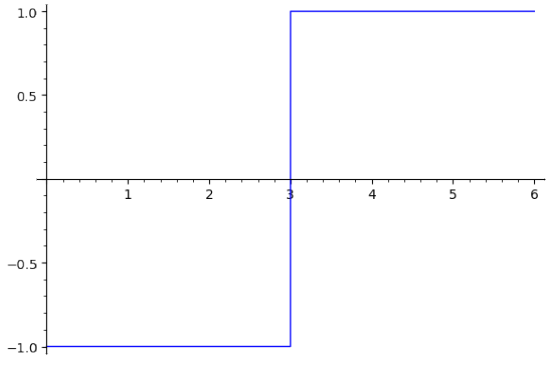
Grafik II

Bu bölüm temel grafik bilgilerini gördüğümüz 3 numaralı bölümün devamı niteliğinde olacak. Burada üç boyutlu interaktif grafikler de dahil olmak üzere, nispeten ileri düzey grafikler için komut ve seçeneklere odaklanacağız.

5.1 Süreksiz Bazı Fonksiyonların Grafikleri

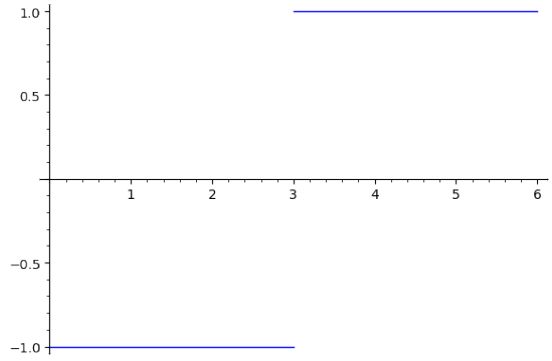
Süreksiz fonksiyonların grafiklerinin bilgisayar çizimleri süreksizlik noktalarında çoğunlukla kusurlu olur. Örneğin $f(x) = \operatorname{sgn}(x-3)$ fonksiyonunun grafiğini SageMath ile çizecek olursak, fonksiyonun süreksiz olduğu noktada, yani $x = 3$ noktasında aşağıdaki gibi beklenmedik dikey bir doğru parçası görürüz.

```
plot( sgn(x-3) , (x,0,6) )
```



Bu durumda `exclude` (yani *dışla*) komutunu aşağıdaki gibi $x = 3$ noktasını dışlamak için kullanabiliriz.

```
plot( sgn(x-3) , (x,0,6) , exclude= [3] )
```



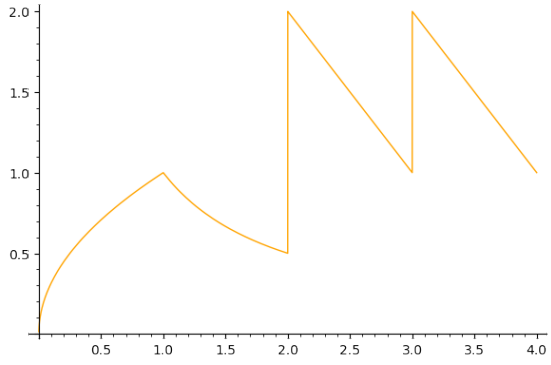
Parçalı fonksiyonlarda da sıçramalar yaygın olduğundan, gerekli durumlarda aynı komut kullanılabilir.

Parçalı Fonksiyon

SageMath parçalı fonksiyon için `piecewise` komutunu kullanır. Aşağıda f parçalı fonksiyonunu ve bu fonksiyonu nasıl tanımlayıp çizdirebileceğimizi görüyoruz:

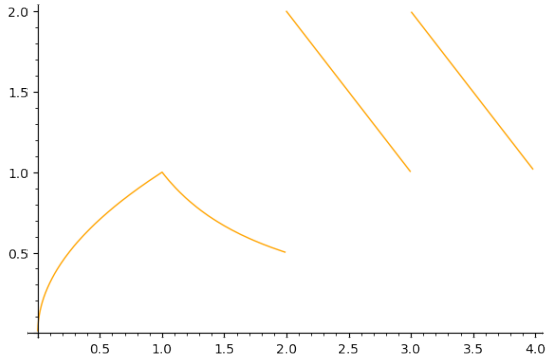
$$f(x) = \begin{cases} \sqrt{x} & 0 < x < 1 \\ \frac{1}{x} & 1 \leq x \leq 2 \\ 4 - x & 2 < x < 3 \\ 5 - x & 3 < x < 4 \end{cases}$$

```
f(x) = piecewise( [ [(0,1), sqrt(x)], [[1,2],1/x], [(2,3),4-x],
    [(3,4),5-x] ] )
show( plot( f, (x,0, 4 ) , color= "orange" ))
```



Benzer şekilde süreksizlik noktalarını, yani $x = 2$ ve $x = 3$ noktalarını dışlayalım:

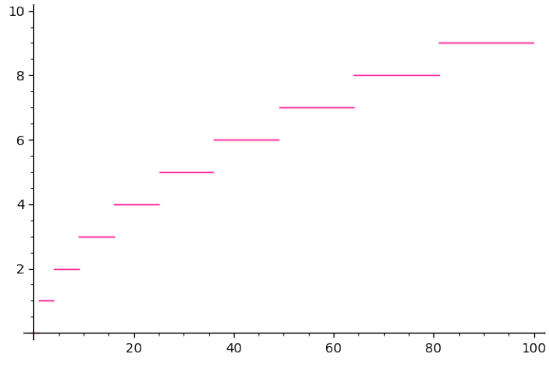
```
f(x) = piecewise( [ [(0,1), sqrt(x)], [[1,2],1/x], [(2,3),4-x],
    [(3,4),5-x] ] )
show( plot( f, (x,0, 4 ) , color= "orange" , exclude=[2,3] ))
```



Dışlanan noktaların bir liste olarak girildiğine dikkat edin. Gerekirse uzunca bir dışlama listesi oluşturabiliriz. Örnek olarak pek çok süreksizlik noktası olan aşağıdaki fonksiyonu inceleyelim:

$$f(x) = \left[\!\left[\sqrt{x}\right]\!\right]$$

```
plot( floor(sqrt(x) ) , x, 0,100, exclude= [ n^2 for n in [1..10] ] ,
    color="deeppink" )
```

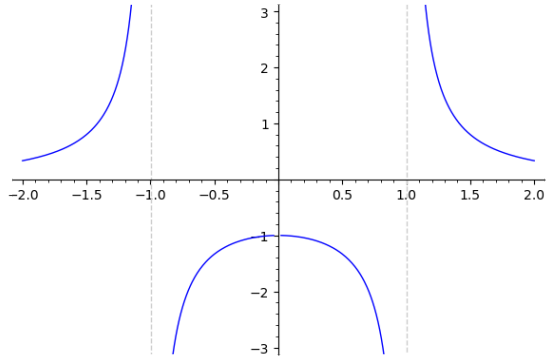


Not 5.1.1 Parçalı fonksiyon tanımlamak için kullanılan `piecewise` komutu yarı açık aralıkları direkt olarak kabul etmese de bu tip aralıkları iki kümenin birleşimi şeklinde yazarak tanımlamak mümkün. Örneğin $[a, b)$ yarı açık aralığı bir nokta ile bir açık aralığın birleşimi gibi düşünülebilir: $[a, b) = [a, a] \cup (a, b)$. Parçalı fonksiyon tanımlamanın daha kapsamlı bir yöntemini 6.3 numaralı alt bölümde göreceğiz.

Dikey Asimptot İçeren Grafikler

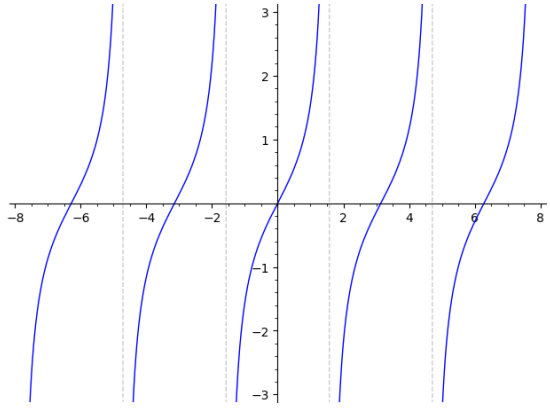
SageMath dikey asimptot içeren grafiklere asimptot çizgisini ekleyebiliyor. Bunun için, gerekli dikey sınırları belirttikten sonra şu komutu ekliyoruz: `detect_poles="show"`. Aşağıdaki örneğe bakalım:

```
plot( 1/(x^2-1),(x,-2,2), ymax=3,ymin=-3, detect_poles="show")
```



İkinci bir örnek olarak, bol bol dikey asimptotu olan $y = \tan x$ fonksiyonunun grafiğini çizelim:

```
plot( tan(x) ,(x,pi/2,11*pi/2), ymax=3,ymin=-3, detect_poles="show")
```



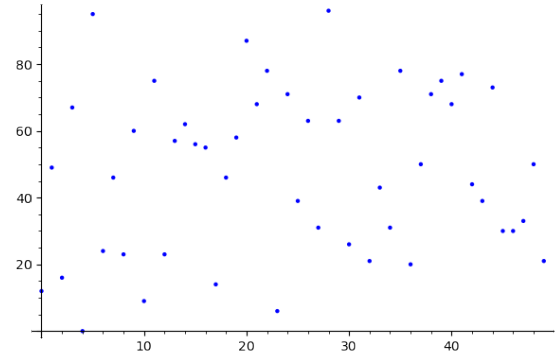
Not 5.1.2 Maalesef SageMath'in yatay, eğik ve eğri asimptotları çizecek bir komutu yok (SageMath versiyon 9.6). Bunları `linestyle`, `color` ve belki de `thickness` gibi makyaj komutlarıyla kendiniz oluşturabilirsiniz.

5.2 Liste Grafiği

Bir sayı listesinin grafiğini `list_plot` komutuyla çizdiriyoruz. SageMath bunu yatay eksen indisler, dikey eksen de indislere karşılık gelen liste elemanları olacak şekilde yapıyor. Aşağıda `veri` adlı bir liste ve bu listenin grafiği var:

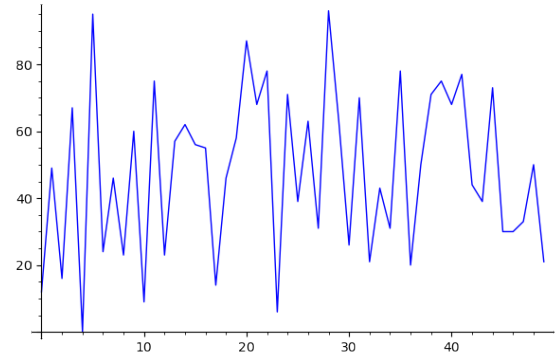
```
veri = [12, 49, 16, 67, 0, 95, 24, 46, 23, 60, 9, 75, 23, 57, 62, 56,
    55, 14, 46, 58, 87, 68, 78, 6, 71, 39, 63, 31, 96, 63, 26, 70, 21,
    43, 31, 78, 20, 50, 71, 75, 68, 77, 44, 39, 73, 30, 30, 33, 50, 21]
```

```
list_plot(veri)
```



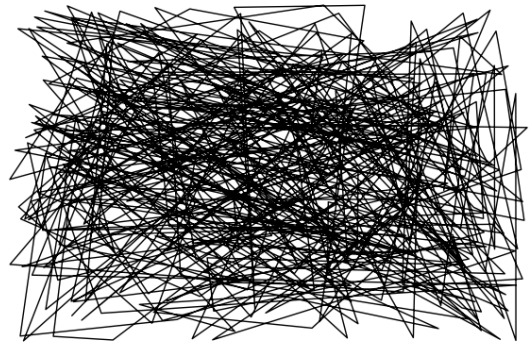
Grafiğin davranışını daha iyi anlamak için ardışık noktaları birleştirebiliriz. Bunun için `plotjoined=True` seçeneğini kullanırız.

```
list_plot(veri , plotjoined=True)
```



`list_plot` komutunu sadece sayı listesi için değil, düzlemdeki bir nokta listesi için de kullanabiliriz. Aşağıda, düzlemde rastgele 400 noktanın birleştirilmesiyle oluşan bir grafik elde ediyoruz:

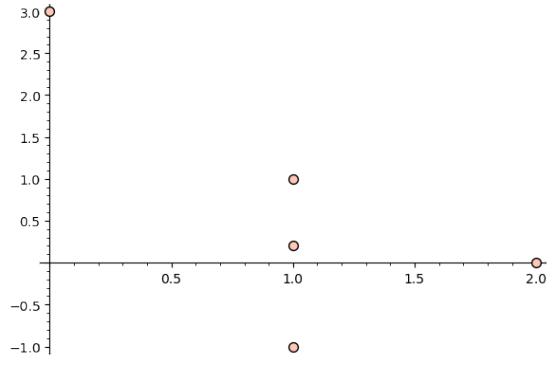
```
noktalar=[ (random(),random()) for i in [1..400] ]
list_plot(noktalar, color='black', axes=False, plotjoined=True )
```



5.3 Saçılım Grafiği

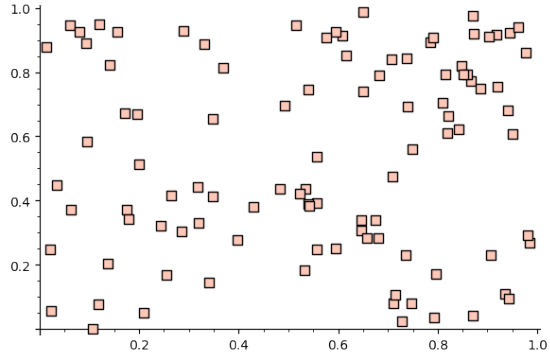
Bir nokta listesinin saçılım grafiğini liste grafiğine benzer bir şekilde aşağıdaki gibi elde edebiliriz:

```
noktalar=[ (0,3),(2,0),(1,1),(1,-1),(1,.2) ]
scatter_plot( noktalar )
```



İşaretçinin şeklini kare olarak değiştirebiliriz. Bunun için `marker="s"` komutunu kullanırız. Burada "s" *square* (kare) kelimesinin baş harfinden gelmektedir.

```
noktalar=[ (random(),random()) for i in [1..100]]
scatter_plot(noktalar , marker="s" )
```

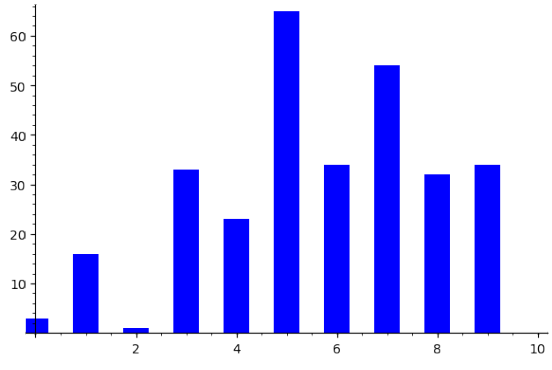


Not 5.3.1 Daire ve kare dışındaki işaretçi seçeneklerini `plot??` komutunu çalıştırarak görebilirsiniz.

5.4 Çubuk Grafik

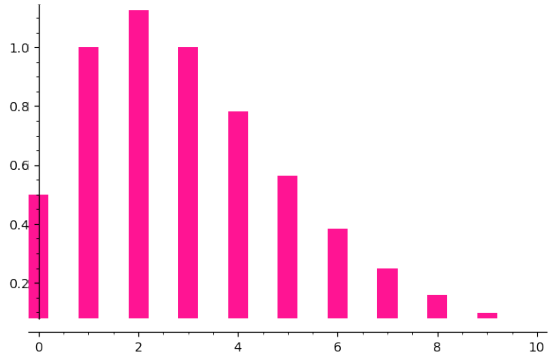
Bir sayı listesi için çubuk grafik, `bar_chart` komutuyla elde edilir:

```
L= [3,16,1,33,23,65,34,54,32,34]
bar_chart( L )
```



Çubukların genişliğini genişlik anlamındaki `width` komutuyla yönetebiliriz:

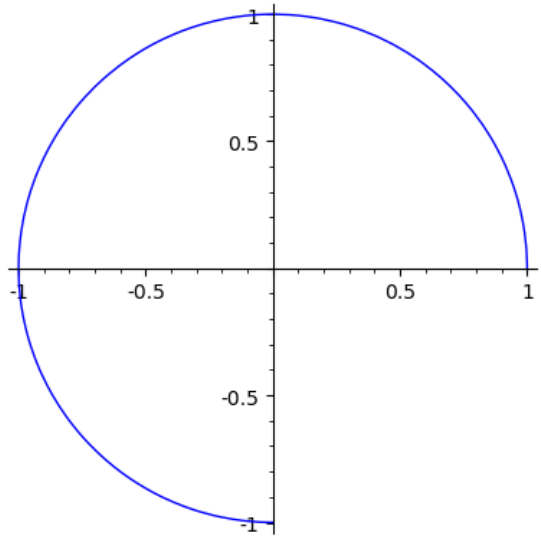
```
bar_chart( [ x^2*2^-x for x in [1..10] ], width=0.4 , color="deeppink")
```



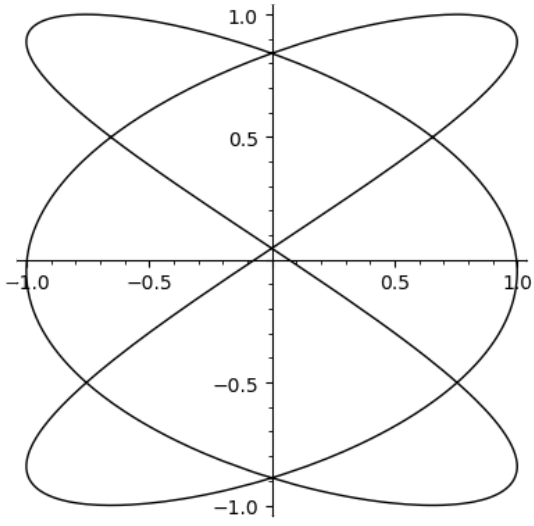
5.5 Parametrik Fonksiyonun Grafiği

Düzlem üzerinde t parametresine bağlı parametrik bir eğri çizdirmek istersek önce `var("t")` komutuyla t değişkenini tanımlarız; sonra da `parametric_plot` komutunu aşağıdaki örneklerdeki gibi kullanırız:

```
var("t")
parametric_plot( (cos(t),sin(t) ) , (t,0,3/2*pi) )
```




```
var("t")
parametric_plot( (sin(3*t),sin(2*t+1) ) , (t,0,2*pi) , color="black" )
```



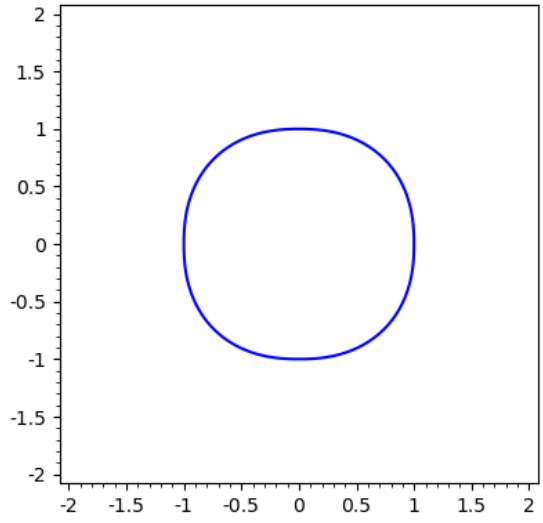
5.6 Kapalı Fonksiyonun Grafiği

Kapalı fonksiyonlarla sık sık karşılaşırız. Örneğin bir çember denklemi olan $x^2 + y^2 = 1$ bir kapalı fonksiyondur. Kapalı fonksiyonu çizdirmek için `implicit_plot` komutunu kullanırız.

Aşağıda çember denklemini biraz değiştirerek farklı bir kapalı fonksiyon ve grafiğini elde ediyoruz:

$$|x|^{2.3} + |y|^{2.3} = 1$$

```
var("y")
implicit_plot( (abs(x))^(2.3) + (abs(y))^(2.3) == 1, (x,-2,2), (y,-2,2) )
```



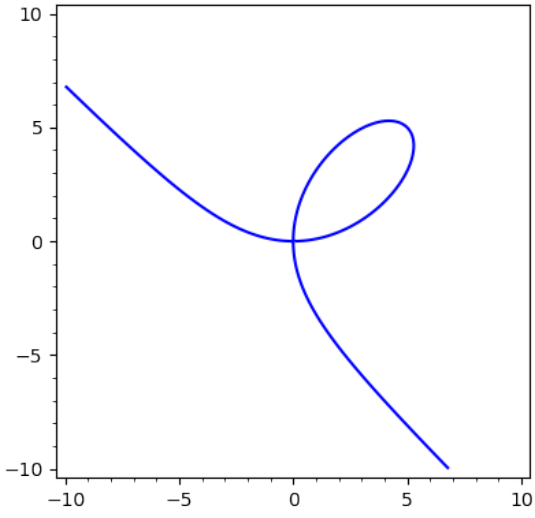
Alıştırma 5.6.1 *Aşağıdaki denklem için farklı pozitif a değerleri vererek grafikleri inceleyin.*

$$|x|^a + |y|^a = 1$$

Kapalı fonksiyon için bir örnek daha:

$$x^3 + y^3 = 10xy$$

```
var("y")
implicit_plot( x^3+y^3==10*x*y , (x,-10,10), (y,-10,10) )
```



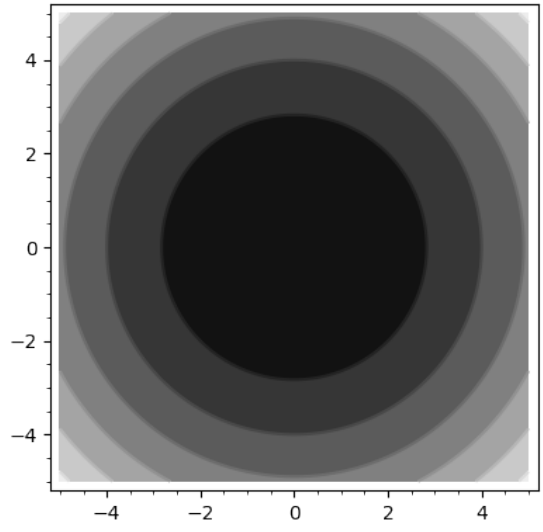
5.7 Kontur Grafiği

Kontur grafiği $z = f(x, y)$ fonksiyonunun seviye eğrilerinden oluşan bir grafiktir. Yani c bir sabit olmak üzere $f(x, y) = c$ formundaki eğrilerin aynı bir düzlemdeki grafiğidir. Örnek olarak aşağıdaki paraboloidi ele alalım ve bu yüzeyin bir kontur grafiğini çizdirelim:

$$z = x^2 + y^2$$

Kullanacağımız komut `contour_plot`:

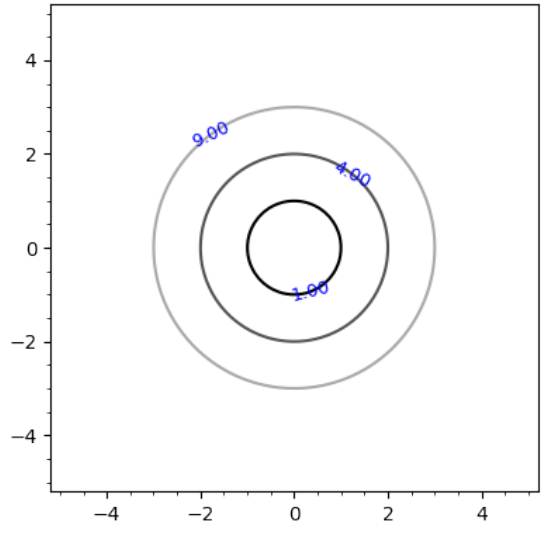
```
var("y")
contour_plot( x^2+y^2 , (x,-5,5), (y,-5,5) )
```



Görüldüğü üzere çeşitli *c* değerleri için eğriler (bu durum için çemberler) çizilip, eğrilerin araları da *c* değerlerine göre dolgu rengi ile boyandı.

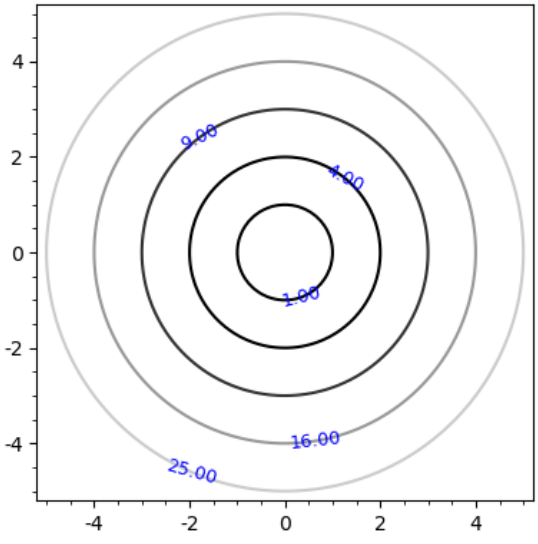
Aşağıdaki örnekte ise *c*'nin sadece 1, 4 ve 9 değerlerini almasını istiyoruz, dolgu rengi kullanmak istemiyoruz ve *c* sayısı için etiketleme istiyoruz:

```
var("y")
contour_plot( x^2+y^2 , (x,-5,5), (y,-5,5), contours=[1,4,9],
    fill=False, labels=True )
```



Aşağıda c değerlerinin liste şeklinde verildiği bir örnek var:

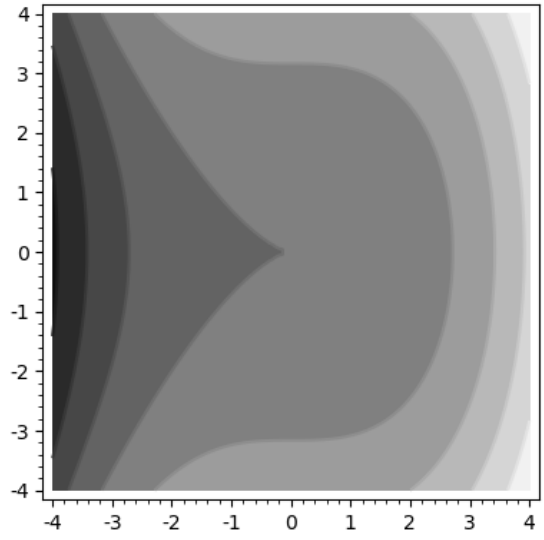
```
var("y")
kontur=[ n^2 for n in [1..5] ]
contour_plot( x^2+y^2 , (x,-5,5), (y,-5,5), contours=kontur,fill=False,
    labels=True )
```



Çizgi Sıklığı ve Renk Haritası

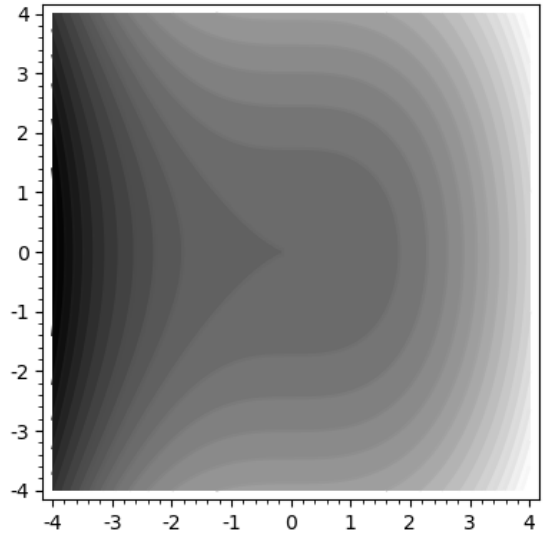
Şimdi de $z = \frac{1}{2}x^3 + y^2$ fonksiyonu için bir kontur grafiği çizelim:

```
x,y = var("x,y")
contour_plot((x^3/2 + y^2), (x,-4,4), (y,-4,4))
```



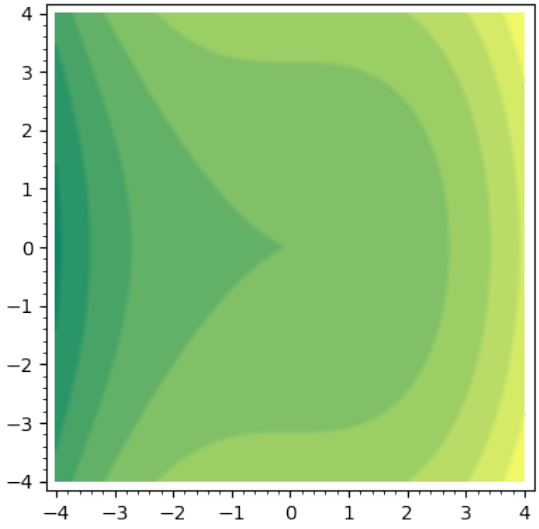
Çizgi sıklığını arttırmak için `contours` komutunu aşağıdaki gibi kullanıyoruz:

```
x,y = var("x,y")
contour_plot((x^3/2 + y^2), (x,-4,4), (y,-4,4) , contours=30 )
```



Son olarak da *color map* (renk haritası) seçeneğiyle farklı renk gruplarını nasıl seçeceğimize dair bir örnek verelim:

```
x,y = var("x,y")
contour_plot((x^3/2 + y^2), (x,-4,4), (y,-4,4) , cmap="summer" )
```

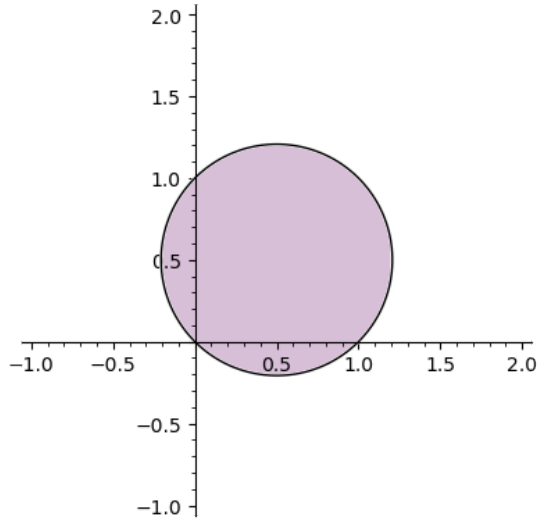


Renk haritası seçeneklerini `sorted(colormaps)` komutunu çalıştırıp görebilirsiniz.

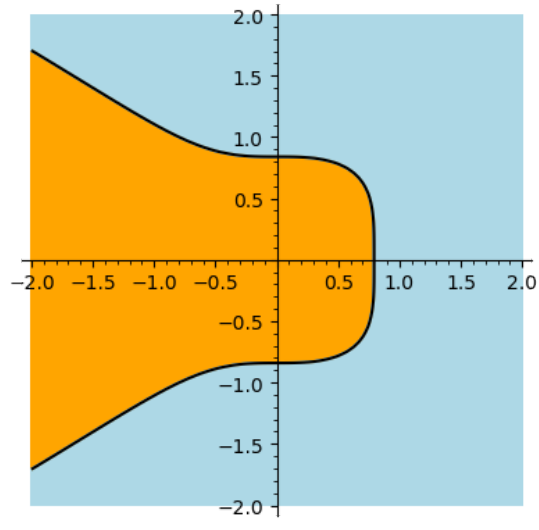
5.8 Eşitsizlik Grafikleri

Eşitsizlik ya da eşitsizlik sistemi grafikleri için `region_plot` komutunu kullanırız. Aşağıda birkaç örnek göreceksiniz. Örneklerde kullanılan grafik seçenekleri şöyle: `incol` (iç kısmın rengi), `outcol` (dış kısmın rengi), `bordercol` (sınır çizgisinin rengi) ve `borderwidth` (sınır çizgisinin kalınlığı). Bazı renk seçeneklerine daha önce de belirttiğimiz gibi `colors` komutunu çalıştırarak ulaşabilirsiniz.

```
var("y")
region_plot( x+y > x^2+y^2 , (x,-1,2), (y,-1,2), incol="thistle",
    borderwidth=1, bordercol= "black")
```



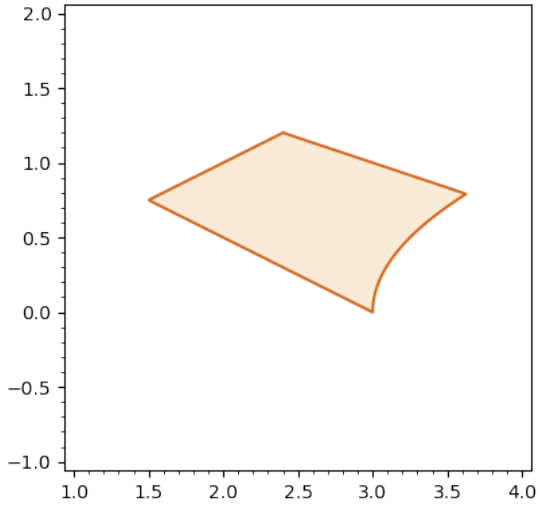

```
var("y")
region_plot(x^3 + y^4 < 1/2, (x,-2,2), (y,-2,2), incol="orange",
    bordercol="black", outcol="lightblue")
```



Birden fazla eşitsizliğin, yani eşitsizlik sisteminin ortak çözümünün grafiği aşağıdaki gibi elde edilebilir:

```
var("y")
```

```
region_plot( [ x + 3*y < 6 , x - y^2 < 3 , -x + 2*y < 0 , -x - 2*y < -3
    ], (x,1,4), (y,-1,2), incol="antiquewhite", bordercol="chocolate",
    plot_points=400 , axes=False , frame=True )
```

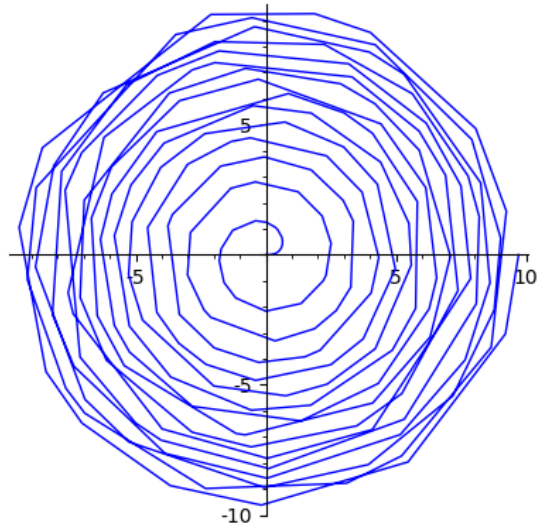


5.9 Polar Fonksiyonun Grafiği

Polar fonksiyonun grafiği `polar_plot` komutu ile çizdirilir. Aşağıdaki fonksiyonu ele alalım:

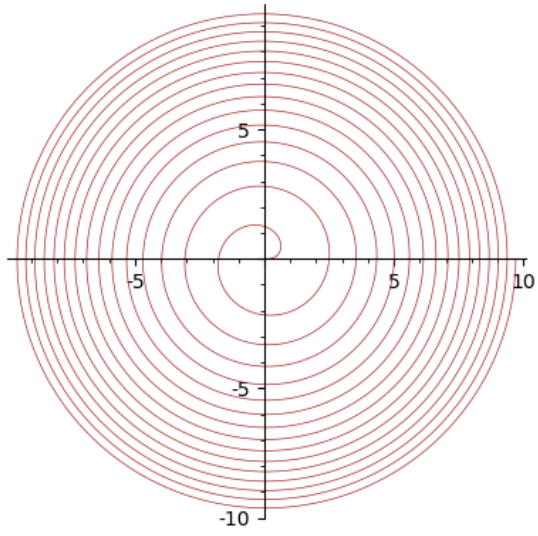
$$r(\theta) = \sqrt{\theta}, \quad 0 < \theta < 30\pi$$

```
var("theta")
polar_plot( sqrt(theta), (theta, 0, 30*pi) )
```



Yukarıdaki grafikte ters giden bir şeyler var. Düzgün yapıda bir grafik beklerken çok fazla sayıda kırılma görüyoruz. Bunun sebebi, SageMath'in varsayılan olarak 200 nokta birleştirerek grafik çizimini yapması. Pek çok durumda bu sayı yeterli. Yukarıdaki örnekteki gibi yetersiz olduğu durumlarda nokta sayısını kendimiz arttırarak daha kaliteli sonuç elde edebiliriz. Aşağıda `plot_points=2000` ifadesiyle nokta sayısını arttırıyor, biraz da makyajla daha güzel bir grafik elde ediyoruz:

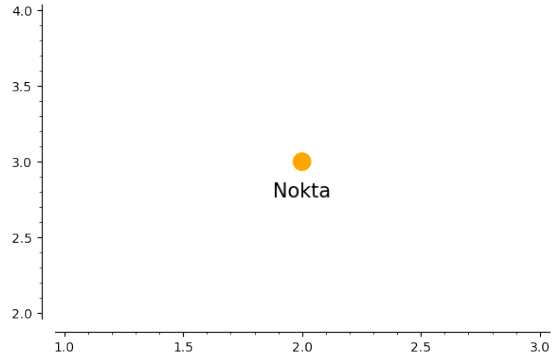
```
var("t")
polar_plot( sqrt(t), (t, 0, 30*pi) , color="brown", plot_points=2000,
    thickness=.5 )
```



5.10 Grafiğe Metin Eklemek

Bazı durumlarda grafiğe metin ya da matematiksel bir ifade eklemek isteyebiliriz. Bunun için `text` komutunu kullanırız. Aşağıdaki örnekte boyutu 15, rengi siyah olacak şekilde "Nokta" metnini koordinat düzleminde $(2, 2.8)$ noktasında göstermek istiyoruz. Ayrıca `point` komutuyla da turuncu bir nokta ekliyoruz.

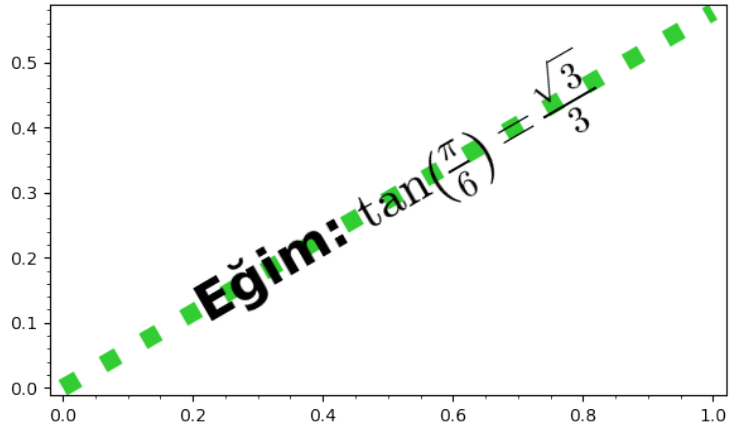
```
metin = text("Nokta", (2,2.8) , fontsize=15, color="black" )
nokta = point( (2,3) , size=200 , color="orange" )
show(metin + nokta , xmin=1, xmax=3, ymin=2, ymax=4 )
```



Aşağıda biraz daha karmaşık bir örnek var. Bu defa metin LaTeX formunda yazılmış matematiksel ifadeler içerdiği gibi 30 derecelik bir eğimle yazılıyor. Buna ek olarak,

yine 30 derecelik bir eğimle, yani eğimi tan 30° olan bir doğru da çiziliyor. Renk, boyut, vb. diğer seçenekleri incelemek size kalsın.

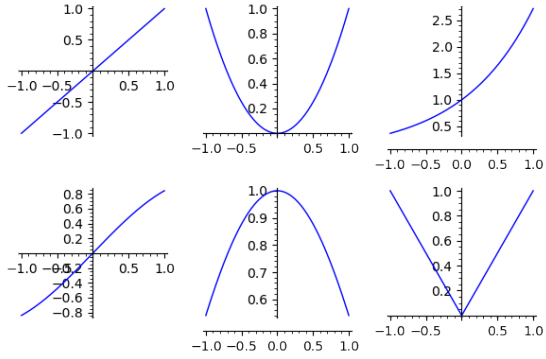
```
metin = text( r" Eğim:
    $\tan{\left(\frac{\pi}{6}\right)}=\frac{\sqrt{3}}{3}$",
    (3/6,sqrt(3)/6), fontsize=30, fontweight='bold', color="black",
    rotation=30 )
dogru = plot( sqrt(3)/3*x , (x,0,1) , aspect_ratio=1 , color="limegreen"
    , linestyle=":" , thickness=5 )
show(metin + dogru , frame=True , axes=False )
```



5.11 Grafik Tablosu

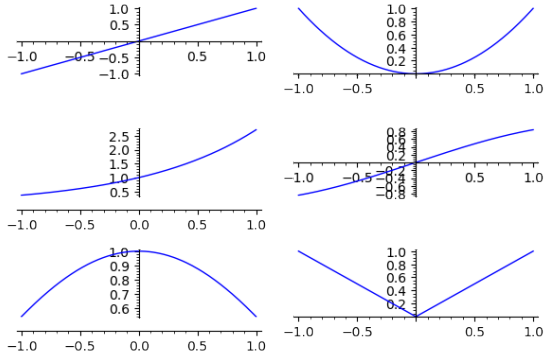
Birden fazla grafik ögesini bir grafik tablosuna dönüştürebiliriz. Bunu girdileri grafikler olan bir matris yapısı oluşturarak yaparız. Aşağıdaki örnekleri inceleyin:

```
p1 = plot(x)
p2 = plot(x^2)
p3 = plot(exp(x))
p4 = plot(sin(x))
p5 = plot(cos(x))
p6 = plot(abs(x))
P= [ [p1,p2,p3], [p4,p5,p6] ]
graphics_array(P)
```

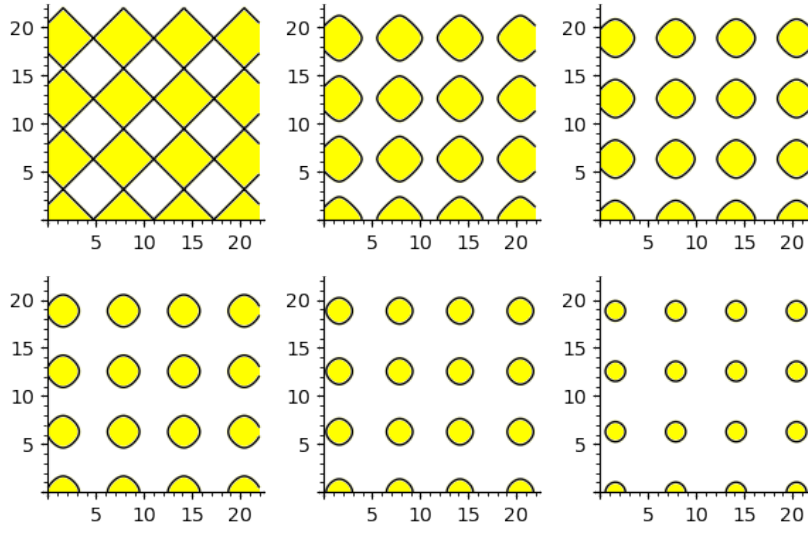


Grafik tablosu için bir diğer yöntem de önce grafikleri liste olarak yazıp sonra da satır sayısını `graphics_array` komutunda belirtmek. Örnekleri inceleyiniz.

```
p1 = plot(x)
p2 = plot(x^2)
p3 = plot(exp(x))
p4 = plot(sin(x))
p5 = plot(cos(x))
p6 = plot(abs(x))
PP=[ p1,p2,p3,p4,p5,p6 ]
graphics_array(PP,3)
```



```
PP=[ region_plot( sin(x) + cos(y) > i*.3 , (x,0,7*pi), (y,0,7*pi),
    incol="yellow", bordercol="black", borderstyle="-" ,borderwidth=1 )
    for i in [0..5] ]
graphics_array(PP,2)
```



Alıştırma 5.11.1 (Lissajous Eğrileri) *Parametrik olarak verilen aşağıdaki eğri ailesine Lissajous eğrileri denir:*

$$x(t) = A\sin(at + \delta), \quad y(t) = B\sin(bt)$$

$A = B = 1$ *ve* $\delta = \frac{\pi}{2}$ *alarak* (a, b) *ikilisi için* $(1,1)$, $(1,2)$, $(1,3)$, $(2,3)$, $(3,4)$, $(3,5)$, $(4,5)$ *ve* $(5,6)$ *değerlerini kullanarak 8 tane Lissajous eğrisiyle iki sütunlu bir grafik tablosu oluşturun.*

5.12 Grafiği Dosya Olarak Kaydetmek

Elde ettiğiniz bir grafiği dosya olarak kaydetmek için hızlı bir yöntem, grafiğin üzerine sağ tıklayarak kaydetme seçeneğini kullanmak. Dosya uzantısı konusunda seçici iseniz vektör grafik de dahil olmak üzere pek çok dosya uzantısına olanak veren `save` komutunu aşağıdaki gibi kullanabilirsiniz.

```
var("t")
grafik = parametric_plot( (sin(3*t),sin(2*t+1) ) , (t,0,2*pi) ,
    color="black" )
grafik.save( "parametrik.png" )
```

Not 5.12.1 Vektör grafik elde etmek için `.png` uzantısı yerine, mesela `.svg` uzantısını kullanabilirsiniz. Grafik dosyası, mevcut SageMath dosyasının bulunduğu klasöre kaydedilecektir. Çevrimiçi `sagecell` kullanıyorsanız, dosyaya ulaşacağınız bir bağlantı göreceksiniz.

5.13 Üç Boyutlu ve Etkileşimli Grafikler

Bu alt bölümde üç boyutlu grafiklerden en sık kullanılan üç tanesine odaklanıyoruz: `plot3d`, `implicit_plot3d`, ve `parametric_plot3d`. Hepsinin ortak özelliği üç boyutlu etkileşimli grafik göstermeleri. Yani bu şekilde elde edeceğiniz grafik sabit bir resim değil; fare kullanarak onu farklı açılardan görebilirsiniz. Farenin sol tuşu yardımıyla grafiği çevirebilir, fare tekerleği ile grafiğe yakınlaşıp uzaklaşabilirsiniz.

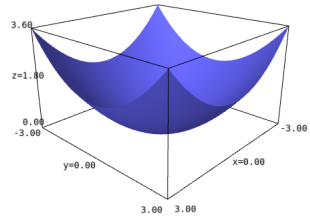
`plot3d` Komutuyla $z = f(x, y)$ Yüzeyi

Burada `plot3d` komutunu kullanacağız. Aşağıda

$$z = \frac{1}{5}\left(x^2 + y^2\right)$$

yüzeyinin etkileşimli bir grafiğini çizdiriyoruz.

```
var("y")
plot3d( (x^2+y^2)/5, (x,-3,3),(y,-3,3) )
```

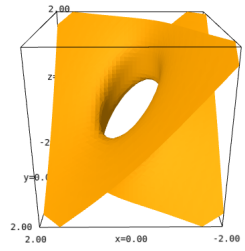


`implicit_plot3d` Komutuyla $F(x, y, z) = 0$ Yüzeyi

`implicit_plot3d` komutunun amacı $F(x, y, z) = 0$ gibi üç değişkenli bir kapalı fonksiyonun belirttiği yüzeyi görüntülemek. Örnek fonksiyon ve grafiği aşağıda:

$$(x + y + z)^4 = x^2 + y^2 - \frac{1}{2}$$

```
var("x y z")
implicit_plot3d( (x+y+z)^4 == x^2+y^2-1/2 ,(x,-2,2), (y,-2,2), (z,-2,2),
    color="orange")
```



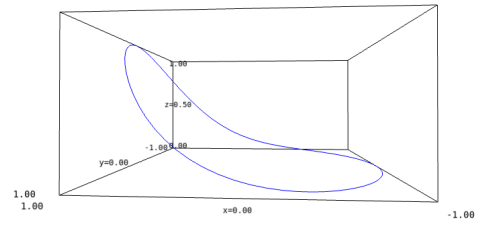
`parametric_plot3d` Komutuyla $\mathbf{r}(t) = \langle f(t), g(t), h(t) \rangle$ Eğrisi

Burada t parametresine bağlı parametrik

$$\mathbf{r}(t) = \left\langle \cos t, \sin t, e^{-t^2} \right\rangle$$

eğrisinin grafiğini `parametric_plot3d` komutunu kullanarak elde ediyoruz:

```
t = var("t")
parametric_plot3d((cos(t), sin(t), exp(-t^2)), (t,-pi,pi))
```

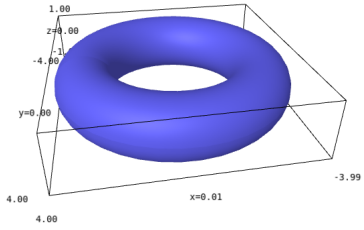


`parametric_plot3d` Komutuyla $\mathbf{S}(u, v) = \langle f(u, v), g(u, v), h(u, v) \rangle$ Yüzeyi

Burada yine parametrik bir fonksiyon var; ancak iki parametre kullanıyoruz ve böylece yüzey elde ediyoruz:

$$\mathbf{S}(u, v) = \langle (3 + \cos u) \cos v, (3 + \cos u) \sin v, \sin u \rangle$$

```
var("u v")
parametric_plot3d( ( ( 3+cos(u) )*cos(v), (3+cos(u))*sin(v) , sin(u) ) ,
    (u,0,2*pi), (v,0,2*pi) )
```



5.14 Yaygın Kullanılan Bazı Grafikler

5.1 numaralı tabloda ok, çember, elips, doğru parçası, çokgen, küre ve vektörün grafikleri için SageMath komutları, örnek kullanım ve açıklamalar veriliyor.

Obje	Örnek Kullanım	Açıklama
Ok	`arrow((0,0), (1,1))`	$(0,0)$ noktasından $(1,1)$ noktasına ok çizer.
Çember	`circle((3,2), 1)`	Merkezi $(3,2)$, yarıçapı 1 olan çemberi çizer.
Elips	`ellipse((0,0),2,3)`	Merkezi $(0,0)$, yarı küçük eksen uzunluğu 2, yarı büyük eksen uzunluğu 3 olan elipsi çizer.
Doğru Parçası	`line([(0,0), (1,1)])`	$(0,0)$ noktasından $(1,1)$ noktasına doğru parçası çizer. İkiden fazla nokta ile de ekleme yapılabilir.
Çokgen	`polygon([(0,0),(1,1),(-2,2),(0,1)])`	Köşeleri $(0,0)$, $(1,1)$, $(-2,2)$ ve $(0,1)$ noktaları olan çokgen çizer.
Küre	`sphere((0,0,0),size=1 )`	Merkezi $(0,0,0)$, yarıçapı 1 birim olan küreyi çizer.
Vektör	`plot(vector([1,2,3]))`	$\langle 1,2,3\rangle$ vektörünü orijini başlangıç kabul ederek çizer.

Tablo 5.1: Yaygın Kullanılan Bazı Grafik Objeleri, Örnek Kullanımları ve Açıklamaları

Bölüm 6

SageMath Programlamaya Giriş

Bu bölüme kadar, halihazırda tanımlı SageMath komutlarıyla çalıştık. Mesela 1273 sayısının asal olmadığını `is_prime(1273)` komutunu çalıştırarak gördük. e^π sayısının π^e sayısından büyük olduğunu `max(e^pi,pi^e)` komutuyla anladık. Ne var ki, örneğin, verilen bir a sayısı için a ile a^2 arasında kaç tane asal sayı olduğunu veren bir komut yok. Peki böyle bir komut yaratmak mümkün mü? Bu bölümde işte bu tip sorulara ve çok daha fazlasına cevap verecek olan programlamanın temel elemanlarını göreceğiz.

Tahmin edersiniz ki yalnızca hazır komutları kullanmak oldukça kısıtlayıcıdır. Programlamayı öğrenmek, sadece piyano tuşlarına basabilen birinin artık beste yapmaya başlamasına benzetilebilir.

6.1 Girintiler

SageMath için bir satırın hangi girinti ile başladığı çok önemli. Girintiler o satırın hangi bloğa ait olduğunu bize söyler. Bu işleyiş organize bir şekilde çalışmamıza da yardımcı olur. Şu ana kadarki örneklerimizde satıra hep sıfır girinti ile başladığımızdan, bunun önemine dair bir durum ile karşılaşmadık. Aşağıdaki alt bölümde, fonksiyon tanımlarken girinti kullanmaya ilk kez ihtiyaç duyacağız.

6.2 Fonksiyon Tanımlama

Matematiksel fonksiyonları nasıl tanımlayacağımızı 2.17 numaralı alt bölümde görmüştük. Şimdi de daha genel ve matematiksel fonksiyonu da kapsayan, programlamadaki fonksiyon yapısını inceleyeceğiz. Tanımlamaya *define* (yani, *tanımla*) kelimesinin kısaltılmış hali olarak kullanılan `def` komutu ile başlıyoruz. Aşağıdaki basit örnek, girilen sayının karesini veren, `karesini_bul` adında bir fonksiyon tanımlıyor:

```
def karesini_bul(x):
    return(x^2)
```

Evet, `karesini_bul` fonksiyonunu tanımladık. İlk satırı tamamlayıp iki noktadan (:) sonra enter'a basınca imleç otomatik olarak girintili bir şekilde (4 boşluk bırakarak) satır başına geçti. Bu aşamada fonksiyonun x değişkenine bağlı olarak ne yapması gerektiğini söylüyoruz. Çıktı olarak x^2 sonucunu görmek istediğimizi *cevap* ya da *geri dönüş* anlamındaki `return` komutunu kullanarak belirtip ilk fonksiyonumuzu tanımladık.

Örneğin 5 değeri için fonksiyonu çalıştıralım ve çıktıyı görelim:

```
karesini_bul(5)
```

```
25
```

Yukarıda tanımladığımız fonksiyon, kenar uzunluğu girilen bir karenin alanını veren bir fonksiyon olarak aşağıdaki şekilde de ifade edilebilir. Kenar uzunluğuna bağlı bu fonksiyonun adı `karealani` olsun:

```
def karealani(kenar):
    return(kenar^2)
```

```
karealani(5)
```

```
25
```

Bir de çıktı olarak metin de içeren durumu görelim. Yine, aynı basit örnekle, bu defa `KAREALANI` adında bir fonksiyon ile yapalım bunu:

```
def KAREALANI(kenar):
    print("Bir kenarı", kenar , "birim olan karenin alanı" , kenar^2 ,
        "birimkaredir." )
```

`print` komutunu kullanarak ekrana yazdırmak istediğimiz metin ve değişkenleri giriyoruz. Bu ifadeleri virgül ile ayırdığımıza dikket edin (4.1 numaralı alt bölüm). Şimdi kare alanını bulmaya hazırız. Kenarı 5 birim olan bir karenin alanını bulmak için aşağıdaki komutu çalıştırmak yeterli:

```
KAREALANI(5)
```

```
Bir kenarı 5 birim olan karenin alanı 25 birimkaredir.
```

Benzer şekilde birden fazla girdisi olan bir fonksiyon da aşağıdaki gibi tanımlanabilir:

```
def dikdortgenalani(kenar1,kenar2):
    print("Kenarlari",kenar1,"ve",kenar2,"birim olan dikdörtgenin
        alani",kenar1*kenar2,"birimkaredir." )
```

Kenar uzunlukları 34 ile 32 olan dikdörtgenin alanını bulalım:

```
dikdortgenalani(34,32)
```

```
Kenarlari 34 ve 32 birim olan dikdörtgenin alani 1088 birimkaredir.
```

Fonksiyon kullanan bir örnek daha inceleyelim:

Örnek 6.2.1 Saniye olarak girilen bir süreyi, saat-dakika-saniye (SDS) formuna çeviren bir program yazalım. Tabii ki, dakika ve saniye değerlerinin 60'dan küçük negatif olmayan tam sayılar olduğunu göz önünde bulundurarak. Mesela 77 saniye dediğimiz şey 0 saat 1 dakika 17 saniyedir. 3666 saniye de, 1 saat 1 dakika 6 saniyedir. Bu basit problemde bölüm (// komutu) ve kalanın (% komutu) hesaplanacağı bölme işlemleri yapacağımız açıktır. Bu komutlar 2.1 numaralı alt bölümde işlenmişti.

```
def SDS(toplam_saniye):
    saat=toplam_saniye//3600
    dakika=(toplam_saniye % 3600)//60
    saniye=toplam_saniye%60
    print(saat ,"saat", dakika,"dakika" , saniye, "saniye")
```

Program tamam. Hemen kullanalım:

```
SDS(3666)
```

```
1 saat 1 dakika 6 saniye
```

```
SDS(10^10)
```

```
2777777 saat 46 dakika 40 saniye
```

Fonksiyon tanımlama pratik kullanımından dolayı oldukça etkili bir şey. Kitabın kalan kısmında onlarca fonksiyon tanımlayacağız ama şimdilik programlamanın diğer önemli kavramlarını tanımaya devam edelim.

6.3 Koşullu Önermeler: `if, else, elif` Komutları

`if` komutunun kelime anlamı, *eğer*. Bu komut, belli bir koşul sağlanıyorsa yapılması gerekenleri belirtmek için kullanılır. Koşul sağlanmıyorsa ne olacak? SageMath o `if` yapısına dair hiçbir şey yapmayacak. Yani, `if` komutu sadece koşulun sağlandığı durum ile ilgilenir. Basit bir `if` uygulaması aşağıda:

```
if 1==1:
    print("Merhaba!")
```

```
Merhaba!
```

```
if 1==2:
    print("Merhaba!")
```

Yukarıdaki örnekte 1, 2'ye eşit olmadığından bir çıktı alamadık. Neredeyse aynı örnekler aşağıda. `if` komutunda da iki noktadan (:) sonra girinti kullandığımıza dikkat edin.

```
if True:
    print("Merhaba!")
```

```
Merhaba!
```

```
if False:
    print("Merhaba!")
```

Bir örnek daha:

```
a=2
if a>0:
    print("a pozitif bir sayı")
```

```
a pozitif bir sayı
```

Aşağıda `if-else` (yani, *eğer-değilse*) yapısını kullanıyoruz. Bu durumda `else` komutu bir yukarısındaki `if` koşulunun sağlanmadığı bütün durumlar için ne yapılması gerektiğini söyler.

```
a=-6
if a>0:
    print("a pozitif bir sayı")
else:
    print("a pozitif değil")
```

```
a pozitif değil
```

Burada `else` komutu $a > 0$ durumunun sağlanmadığı bütün durumlar, yani $a \leq 0$ durumunda neler olacağını söylüyor.

Not 6.3.1 Art arda birden fazla `if` yapısı kullanıldığında `else` komutu sadece sonuncu `if` koşulunu dışlar. Aşağıdaki örneği inceleyiniz:

```
a=1
if a==1:
    print("A")
if a==2:
    print("B")
else:
    print("C")
```

```
A
C
```

Şimdi de bir örnek üzerinden `if-elif-else` (`elif` komutu *else* ve *if* kelimelerinin birleşmesiyle oluşturulmuş) yapısının işlevselliğini görelim. Diyelim ki en küçük 10 pozitif tam sayı A grubunda, sonraki 10 tam sayı B grubunda, sonraki tüm tam sayılar ise C grubunda olsun. Girilen bir sayının hangi grupta olduğunu söyleyen bir program yazalım:

```
k=12
if k>20:
    print("C grubunda")
elif k>10:
    print("B grubunda")
elif k>0:
    print("A grubunda")
else:
    print("Hiçbir grupta değil")
```

```
B grubunda
```

Burada `elif` komutu kendinden önce gelen tüm koşulların dışındaki durumlara yeni bir koşul getiriyor.

Şimdi de yukarıdaki hikayemizi bir fonksiyon ile ifade edip ona daha pratik bir kullanım kazandıralım:

```
def grup_bul(k):
    if k>20:
        print("C grubunda")
    elif k>10:
        print("B grubunda")
    elif k>0:
        print("A grubunda")
    else:
        print("Hiçbir grupta değil")
```

Evet, fonksiyon tanımlamamızı yaptık. Şimdi 12 sayısının hangi grupta olduğunu sorgulayalım:

```
grup_bul(12)
```

```
B grubunda
```

Örnek 6.3.1 (Parçalı Fonksiyon) Benzer bir yapıyı aşağıdaki parçalı fonksiyonu tanımlamak için kullanabiliriz:

$$F(x) = \begin{cases} 1 - x^2, & x \leq 1 \\ 2 - x, & x > 1 \end{cases}$$

```
def F(x):
    if x<=1:
        return(1-x^2)
    else:
        return(2-x)
```

Fonksiyonu tanımladık. Şimdi de fonksiyonun bazı noktalardaki değerlerini bulalım:

```
F(-3)
```

```
-8
```

```
F(1)
```

```
0
```

```
F(3)
```

```
-1
```

Alıştırma 6.3.1 *5.1 numaralı bölümde* `piecewise` *komutu ile tanımlanan parçalı fonksiyonu* `if, elif` *ve* `else` *komutlarını kullanarak tanımlayın.*

Örnek 6.3.2 (Üçgenin Kenar Uzunlukları) Girilen üç pozitif sayının bir üçgenin kenar uzunlukları olmak için uygun olup olmadığını veren bir fonksiyon tanımlayalım. Bunun için Üçgen Eşitsizliği'ne ihtiyacımız olacak. Biliyoruz ki a, b, c pozitif sayıları

$$|b - c| < a < b + c$$

koşulunu sağlıyorsa, a, b, c bir üçgenin kenar uzunlukları olmak için uygun sayılardır. Koşulu sağlamıyorlarsa da uygun değillerdir.

```
def Ucgen_mi(a,b,c):
    if abs(b-c) < a < b+c :
        print( a , b , c , "üçgen oluşturur.")
    else:
        print( a , b , c , "üçgen oluşturmaz.")
```

```
Ucgen_mi(3,4,5)
```

```
3 4 5 üçgen oluşturur.
```

Not 6.3.2 Yukarıdaki üçgen örneğinde $|b - c| < a < b + c$ koşulu yerine

$$|a - c| < b < a + c \quad \text{veya} \quad |a - b| < c < a + b$$

koşullardan birini de kullanabilirsiniz. Bu üç koşulun eşdeğer olduğunu eşitsizlikleri açarak görebilirsiniz. Sadece birini kullanmak yeterli. (Üç eşitsizliğin eşdeğer olduğunu görmenin bir başka yolu da şöyle: Bir koşul doğruysa üçgen oluşur; üçgen oluşursa diğer iki koşul da doğrudur. Böylece bir koşul doğruysa diğer ikisi de doğrudur.)

Örnek 6.3.3 6.2 numaralı alt bölümde oluşturduğumuz `dikdortgenalani` fonksiyonunu hatırlayın. Aynı fonksiyonu `if` kullanarak, kenar uzunluklarının pozitif olmadığı durumda uyarı veren bir fonksiyona dönüştürelim:

```
def dikdortgenalani(kenar1,kenar2):
    if kenar1 <= 0 or kenar2 <= 0:
        print("Kenar uzunlukları pozitif olmalı!")
    else:
        print("Kenarları",kenar1,"ve",kenar2,"birim olan dikdörtgenin
            alanı",kenar1*kenar2,"birimkaredir." )
```

```
dikdortgenalani(5 , 0)
```

```
Kenar uzunluklari pozitif olmali!
```

Alıştırma 6.3.2 *Girilen bir sayının aynı zamanda hem asal hem de 100'den büyük olduğu durumda ekrana "Hem asal hem 100'den büyük."; diğer bütün durumlarda "Ya asal değil ya da 100'den büyük değil." ifadelerini yazdıran bir fonksiyon oluşturun.*

6.4 `for` Döngüsü

`for` (*için*) döngüsünü bir işlem ya da işlem grubunu istediğimiz belli değerler için tekrar tekrar yapmak istediğimizde kullanırız. Mesela, i değişkeninin sırasıyla 1, 2 ve 3 değerleri için i^2 değerini ekrana yazdıran bir kod yazalım:

```
for i in [1,2,3]:
    print(i^2)
```

```
1
4
9
```

Yukarıdaki iki kod satırının Türkçesi şu: i sayısının [1,2,3] listesindeki her bir elemanı için sırasıyla i^2 değerini ekrana yazdır. Yine, iki noktadan (:) sonra girintili devam ediyoruz (bu konuda SageMath rehberlik ediyor bize).

Çok benzer bir örnek daha:

```
for k in range(5):
    print(k)
```

```
0
1
2
3
4
```

Yukarıda `range(5)` komutu kullanılıyor. Biliyoruz ki bu, `[0,1,2,3,4]` listesinden başka bir şey değil ve bunu `[0..4]` şeklinde de yazabiliriz.

Şimdi de 1'den 40'a kadar olan sayılardan asal olanları veren bir program yazalım:

```
for p in [1..40]:
    if is_prime(p)==True:
        print(p)
```

```
2
3
5
7
11
13
17
19
23
29
31
37
```

Yukarıda hem `for` hem de `if` komutu kullanılıyor. İlk yapılan şu: `p`'yi 1 al, asal ise ekrana yazdır. Daha sonra, `p`'yi 2 al, asal ise ekrana yazdır. Bu benzer şekilde 40'a kadar devam edecek ve son olarak 40 sayısına aynı sorgulama yapılacak ve döngü sonlanacak.

Not 6.4.1 İkinci satırda `if is_prime(p)==True:` ifadesini kullandık. Onun yerine `if is_prime(p):` demek de yeterli.

Yukarıdaki asal sayılardan `append` komutu kullanarak `asallar` diye bir liste oluşturalım:

```
asallar=[ ]
for p in [1..40]:
    if is_prime(p):
        asallar.append(p)
asallar
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]
```

Yukarıda ilk olarak `asallar` adında boş bir liste tanımlıyor, daha sonra asalları `append` komutu ile bu listeye ekliyoruz. İlk yapılan şu: `p`'yi 1 al, asal ise `asallar` listesine ekle. Daha sonra, `p`'yi 2 al, asal ise `asallar` listesine ekle. Benzer şekilde devam edip son olarak 40 sayısına aynı sorgulama yapılıyor. İşimiz bitince de son satırda liste ekrana yazdırılıyor. Bunun için girinti kullanmadığımıza dikkat edin.

`for` Komutuyla Liste Oluşturmak

`for` komutu kullanarak pratik bir şekilde listeler yaratabiliriz. Örneğin:

```
L=[ i^2 for i in [1..5] ] ; L
```

```
[1, 4, 9, 16, 25]
```

İkinci bir örnek olarak

$$\left\{\frac{3k+2}{2k+3}\right\}$$

dizisi için k'nın 1'den 100'e kadar olan bütün terimlerinin sayısal değerlerini listeleyelim:

```
[ ((3*k+2)/(2*k+3)).n() for k in [1..100]]
```

Bunun sonucu olarak 100 eleman içeren bir liste göreceksiniz. Diyelim ki büyük sayılar için bu verilen ifadenin nasıl değerler aldığını merak ediyoruz:

```
[ ((3*k+2)/(2*k+3)).n() for k in [10^5-5 .. 10^5]]
```

```
[1.49998749956248,
 1.49998749968749,
 1.49998749981250,
 1.49998749993750,
 1.49998750006250,
 1.49998750018750]
```

Elde ettiğimiz listenin verilen dizinin limiti hakkında ipucu verdiğine dikkat edin.

Benzer şekilde, bu kestirme yöntem ile şartlı ifadeler de kullanabiliriz. 1'den 40'a kadar olan asal sayılar örneğimizi bu teknikle tekrarlayalım:

```
[ p for p in [1..40] if is_prime(p)==True ]
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]
```

Şimdi de yukarıdaki son örneği kullanıp sayı aralığına bağlı `asal_bul` adında daha genel bir fonksiyon tanımlayalım:

```
def asal_bul(a,b):
    asallar=[p for p in [a..b] if is_prime(p)==True ]
    print(asallar)
```

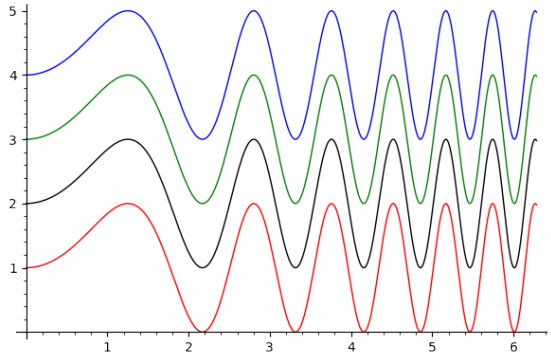
Tanımlamayı yaptık. Mesela, 10^{10} ile $10^{10}+100$ arasındaki asalları listeleyelim:

```
asal_bul( 10^10 , 10^10+100 )
```

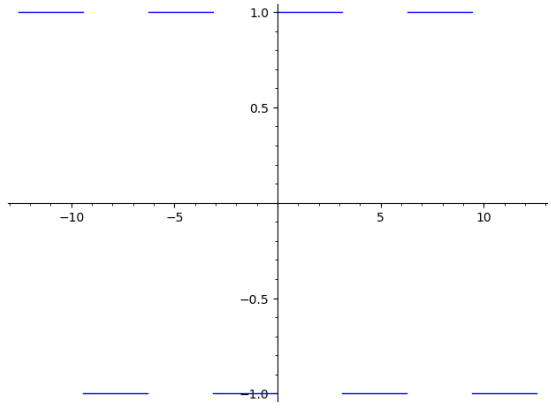
```
[10000000019, 10000000033, 10000000061, 10000000069, 10000000097]
```

Aşağıda grafik içeren iki örnek var:

```
plot( [ sin(x^2)+i for i in [1..4] ], (x,0,2*pi), color=["red", "black",
    "green","blue" ])
```




```
plot( sgn(sin(x)) , x, -4*pi, 4*pi, exclude= [ n*pi for n in [-4..4] ] )
```



Örnek 6.4.1 (Rekürsif Fonksiyon) 100 ile başlayıp tekrar tekrar şu işlemi, elde ettiğimiz sayılara uygulayalım: 3 ile çarp, 10 ekle, 4'e böl. Bu işlemi 30 defa tekrarlayıp tüm sayıları içeren bir liste oluşturalım. İlk olarak birer liste ve fonksiyon yaratıp `append` komutunu kullanacağız:

```
L=[100.]
f(x)=(3*x+10)/4
for i in [1..30]:
    L.append( f( L[-1] ) )
show(L)
```

```
[100.000000000000, 77.5000000000000, 60.6250000000000, 47.9687500000000,
    38.4765625000000, 31.3574218750000, 26.0180664062500,
    22.0135498046875, 19.0101623535156, 16.7576217651367,
    15.0682163238525, 13.8011622428894, 12.8508716821671,
```

```
12.1381537616253, 11.6036153212190, 11.2027114909142,
10.9020336181857, 10.6765252136393, 10.5073939102294,
10.3805454326721, 10.2854090745041, 10.2140568058780,
10.1605426044085, 10.1204069533064, 10.0903052149798,
10.0677289112349, 10.0507966834261, 10.0380975125696,
10.0285731344272, 10.0214298508204, 10.0160723881153]
```

Yukarıda neler olduğuna dair biraz açıklama: 100 ile başlayıp istenen işlemlerle elde edilen tüm sayıları içeren bir L listesi elde etmek istiyoruz. Bunun için işe sadece 100 sayısını içeren bir L listesi tanımlayarak başlıyoruz. Verilen işlemleri uygulayıp sonradan `append` komutu ile 30 defa ekleme yapacağız.

Başlangıçta listenin tek elemanı 100'dür, son eleman olan `L[-1]` de 100 olacaktır. Böylece ilk eklenen sayı $f(100) = 77.5$. İlk eklenen sayıyla birlikte, L listesi [100.0, 77.5] listesine dönüşür. Listenin sonundaki, yani son eklenen sayıyı kullanmak için `L[-1]` ifadesini kullandığımıza dikkat edin. Yani sırada bu işlem silsilesini 77.5 sayısına uygulamak var. Bu şekilde, istenen sayı listesini elde ediyoruz.

Not 6.4.2

1. Programa `L=[100.]` ile başlıyoruz. 100 sayısının sonundaki nokta ile ondalık ifadeler elde ettiğimizi hatırlayın. Noktayı kaldırıp kodu tekrar çalıştırın. Nokta olmayınca, kesin değerler veren kesirli ifadeler korunacaktır. Bazı durumlarda kesirli yapıyı korumak işlemleri ağırlaştırabilir.

2. Son satırdaki `show(L)` ifadesini girintili yazarsak ne olur? Deneyin ve olanları anlamaya çalışın.

3. Yukarıdaki sonucun 10 sayısına yaklaştığını gözlemleyebiliriz. Bu davranışı grafikte görmek için, `list_plot(L)` komutunu kullanabilirsiniz:

```
list_plot(L)
```

Aşağıdaki kod aynı L listesini oluşturmak için başka bir yöntem. Daha pratik olan bu kodu yorumlamak da size kalsın.

```
L=[ ]
x=100.
for i in [1..30]:
    L.append( x )
    x=(3*x+10)/4
show(L)
```

Yukarıda elde ettiğimiz listeler L adında bir listeye atanıyor. Eğer amacımız bir liste ataması yapmak değil de sadece sayıları listelemek ise:

```
x=100.
for i in [1..30]:
    x=(3*x+10)/4
    print( x )
```

Alıştırma 6.4.1 $f^n = \underbrace{f \circ f \circ \cdots \circ f}_{n \text{ tane } f}$ *olmak üzere,*

$$f(x) = \frac{x-1}{x+3}$$

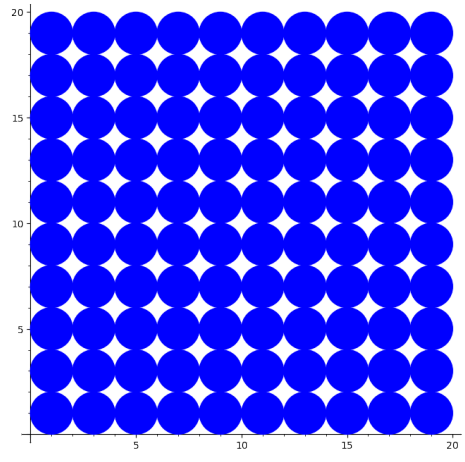
fonksiyonu için $f^{1000}\left(\frac{1}{2}\right)$ *değerini bulun.*

Aşağıdaki örnekte iç içe `for` döngüleri var:

Örnek 6.4.2 Koordinat düzleminin birinci bölgesinde, 10×10 olacak şekilde 100 tane birbirine teğet daire çizmek istiyoruz. Yarıçapları 1'er birim olsun.

Bir tane daire çizmek kolay: Mesela `circle((3,2),5)` komutu, merkezi $(3,2)$ noktasında olup yarıçapı 5 birim olan çemberi çizer. Çemberin içini de `fill=True` komutuyla doldururuz. Ama 100 tane daire için bu işlemi 100 defa yapmak iyi fikir değil, tabii ki. Kullanacağımız şey, sıkıcı pek çok işten bizi kurtaran `for` komutu.

```
graf=plot([ ])
for i in srange(1,20,2):
    for j in srange(1,20,2):
        daire=circle((i,j), 1, fill=True)
        graf += daire
show(graf)
```



Bu programda boş bir grafik ile işe başlıyoruz. `srange` komutunu 1'den 20'ye ikişer ikişer saymak için kullanıyoruz. 10 tane `i` değerinin her biri için 10 tane `j` değeri elde ediyoruz. Her (i,j) ikilisi için (i,j) merkezli birim daireyi çizdiriyoruz ve çizilen ögeyi grafiğe ekliyoruz. Son olarak (19,19) noktası için de bu işlem yapılıp döngü tamamlanıyor ve grafik gösteriliyor.

Alıştırma 6.4.2 (Kronecker Çarpımı) *A ve B matrislerinin boyutları sırasıyla $m \times n$ ve $p \times q$ olsun. $A \otimes B$ Kronecker çarpımı aşağıdaki $pm \times qn$ boyutundaki blok matrisi olarak tanımlanır:*

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix}$$

Örneğin;

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \otimes \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 2 & 4 \\ 3 & 4 & 6 & 8 \\ 5 & 6 & 10 & 12 \\ 3 & 6 & 4 & 8 \\ 9 & 12 & 12 & 16 \\ 15 & 18 & 20 & 24 \end{pmatrix}.$$

Girilen A ve B matrisleri için $A \otimes B$ Kronecker çarpımını veren `Kronecker(A,B)` formunda bir fonksiyon oluşturun (SageMath ile matris tanımlama 7.13 numaralı alt bölümde).

Not 6.4.3 6.4.2 numaralı alıştırmada tanımlanan Kronecker çarpımı için aslında SageMath'in halihazırda `A.tensor_product(B)` şeklinde bir komutu olduğunu belirtelim.

`for` döngüsünde `break` ve `continue` komutları

`for` döngüsü içinde `break` (burada *kesmek, bitirmek* anlamında) komutunu döngüyü sonlandırmak için kullanıyoruz. Birkaç örnek verelim:

```
for sayi in [4,5,6]:
    print(sayi)
    break
```

```
4
```

Yukarıdaki örnekte `print(sayi)` ile `break` ifadelerinin yerini değiştirip programı tekrar deneyin ve farkı anlamaya çalışın.

Aşağıda `break` kullanan açıklayıcı iki örnek var:

```
for sayi in [3,7,0,6,-4,6,-3,10,11]:
    if sayi < 0:
        break
    print(sayi)
```

```
3
7
0
6
```

```
for sayi in [3,7,0,6,-4,6,-3,10,11]:
    if sayi < 0:
        print(sayi)
        break
    print(sayi)
```

```
3
7
0
6
-4
```

Bir de `continue` (yani, *devam et*) komutu için hızlı bir örnek verelim. Çıktıda negatif olmayan tüm sayıları listeliyoruz. Yani negatif olan durumları atlayarak döngüye devam ediyoruz.

```
for sayi in [3,7,0,6,-4,6,-3,10,11]:
    if sayi < 0:
        continue
    print(sayi)
```

```
3
7
0
6
6
10
11
```

6.5 `while` Döngüsü

`while` (-iken) döngüsü, belirlenmiş bir koşul bozulana kadar bir işlem ya da işlem grubunu uygulamak için kullanılır. Koşul her sağlandığında döngüdeki işlemler bir kez yapılır. Koşul bozulduğunda döngü sonlanır.

Dünyanın en basit `while` döngüsünü görelim ve anlamaya çalışalım:

```
a=5
while a<10:
    print(a)
    a=a+1
```

```
5
6
7
8
9
```

Yukarıdaki programın Türkçesi şu: *a* sayısı 5'tir (ilk satır). *a* sayısı 10'dan küçük olduğu sürece aşağıdaki girintili bloktaki komutları sırasıyla tekrar tekrar uygula. Yani *a* sayısını ekrana yazdır, sonra *a*'nın değerini 1 artır. *a* değeri 10'dan küçük olduğu sürece "ekrana yazdır, 1 artır" işlemini tekrarla. Peki, ne zaman duracak bu döngü? *a* 10 değerini aldığında $a < 10$ testinden geçmeyeceğinden döngü orada duracak ve 10 sayısını ekrana yazdıramayacak. Testten geçen son sayı 9 olacaktır.

Şimdi de aşağıdaki kodun yukarıdakinden neden farklı olduğunu anlamaya çalışın.

```
a=5
while a<10:
    a=a+1
    print(a)
```

```
6
7
8
9
10
```

Örnek 6.5.1 Aşağıdaki kod ilk 20 asal sayıyı liste olarak veriyor. Öncelikle `asallar` adında boş bir liste oluşturuyoruz. Bu listeye ilk 20 asalı `sayaç` diye bir değişken kullanarak adım adım ekleyeceğiz. `sayaç` 1 ile başlayıp 20 ile bitecek. Bir de asal sayı adayımız olan `p` değişkeni var, onu da yine 1'den başlatıyoruz. `while` döngüsü şunu yapıyor: `sayaç` 21'den küçük olduğu sürece, eğer `p` asal ise `p`'yi `asallar` listesine ekleyip `sayaç` değerini 1 arttır. Ardından, bir sonraki asal sayı adayını test etmek için `p` değerini 1 arttır. Benzer şekilde yeni değerler için `sayaç` hala 21'den küçükse işlemleri yeni `p` asal sayı adayı için tekrarla. `sayaç` 20'yi geçtiğinde döngü sonlanacaktır ve `asallar` listesi görüntülenecektir.

```
asallar=[ ]
sayaç=1
p=1
while sayaç<21:
    if is_prime(p):
        asallar.append(p)
        sayaç=sayaç+1
    p=p+1
asallar
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
 71]
```

Not 6.5.1 Aslında SageMath'te asal sayıların listesini temsil eden ve asalları listenin indisiyle çağırmamıza imkan veren bir komut var: `Primes()`. Örneğin `Primes()[0]` listenin ilk elemanı olan 2 asalını verecektir. Yukarıda elde ettiğimiz listeyi `Primes()` komutunu kullanarak pratik bir şekilde oluşturabiliriz:

```
[ Primes()[i] for i in range(20) ]
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
    71]
```

Örnek 6.5.2 Aşağıdaki toplamı düşünün:

$$1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\frac{1}{6}+\cdots$$

Örneğin ilk iki terimi toplarsak $1+\frac{1}{2}=\frac{3}{2}=1.5$ elde ederiz. İlk 5 terimin toplamı ise

$$1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}=\frac{137}{60}\approx 2.2833.$$

Sorumuz şu: Bu şekilde en az kaç tane terimi toplarsak sonuç 10'u geçer? Ama, bi dakika! Belki de bu toplam 10 sayısına hiç ulaşmayacak! Neyse, biz şimdilik 10'a ulaşacağını varsayıp sadece algoritmamıza odaklanalım:

```
toplam=0
k=1
while toplam<10:
    toplam=toplam+1/k
    k=k+1
print(k-1)
print(toplam.n())
```

```
12367
10.0000430082758
```

Görünen o ki 12367 tane terim toplarsak 10'dan biraz büyük bir sayı buluyoruz. Programı yorumlayalım: Toplamı 0 (`toplam=0`), k değerini de 1 (`k=1`) alarak işe başladık. `toplam=0`, 10'dan küçük olduğundan, `toplam` değişkenine $\frac{1}{k}$ yani 1 ekliyoruz. Sonra da k'yı 1 arttırıyoruz. Güncellenmiş değerler: `toplam=1`, `k=2`. Toplam 10'dan küçük olduğu sürece, k değerleri için $\frac{1}{k}$ eklemeleri yapıp k sayısını 1 arttırıp yeni toplamı tekrar test ediyoruz. Toplam 10'u geçince de döngü duruyor, bir hayli fazla döngüden sonra.

Yukarıda gördüğünüz toplam, meşhur *harmonik seri*:

$$\sum_{k=1}^{\infty}\frac{1}{k}=1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\frac{1}{6}+\cdots.$$

Bu toplamın sonsuz olduğu ilk olarak 14. yüzyılda ispatlanmış. Sonrasında da pek çok ispatı yapılmış.

Örnek 6.5.3 Şimdi de SageMath, 6-6 gelene kadar zar atsın, 6-6 gelince de dursun. Gelen zar çiftlerini de ekrana yazdıralım.

```
zartoplam=3
while zartoplam != 12:
    zar1, zar2 = randint(1,6), randint(1,6)
    zartoplam=zar1+zar2
    print( (zar1,zar2) )
```

```
[4, 6]
[3, 3]
[4, 1]
[2, 2]
[6, 5]
[5, 3]
[5, 5]
[1, 6]
[4, 5]
[6, 6]
```

Başlarken, `zartoplam` değişkenine 12'den farklı herhangi bir değer atıyoruz ki döngü başlasın. `randint(1,6)` komutunun [1,6] aralığındaki tam sayılardan "rastgele" bir tam sayı üretmek için kullanıldığını belirtelim. Döngüyü yorumlamak size kalsın.

Alıştırma 6.5.1 $f(x) = \frac{x+2}{x-2}$ *fonksiyonuna kaç defa bileşke uygularsak*

$$-\frac{283\,x - 882}{441\,x - 1606}$$

fonksiyonunu buluruz?

Alıştırma 6.5.2 $f(x) = x^{20}\sin^{30} x$ *fonksiyonu için* $f^{(k)}(0) \neq 0$ *koşulunu sağlayan en küçük k doğal sayısı kaçtır? (*$f^{(k)}$ *ifadesi k'ıncı mertebeden türev anlamındadır. Türev ile ilgili komutlar için 7.3 numaralı alt bölüme bakabilirsiniz.)*

Örnek 6.5.4

$$A_n = \underbrace{1999\ldots9991}_{n \text{ tane } 9}$$

formundaki sayıları düşünelim. $n > 200$ koşulunu sağlayan en küçük A_n asal sayısında (eğer böyle bir asal varsa) kaç tane 9 olduğunu bulalım. Öncelikle A_n dizisini aşağıdaki gibi yazalım:

$$A_n = 1 + 10^{n+1} + 9\sum_{k=1}^{n} 10^k$$

Şimdi de n sayısını 200'den başlatarak bir `while` döngüsü yaratalım. Döngüyü başlatmak için işe asal olmayan bir A (1 alalım) sayısıyla başlıyoruz:

```
A=1
n=200
while is_prime ( A )==False:
    n=n+1
    A = 1 + 10^(n+1) + 9*( sum( 10^k for k in [1..n] ) )
show(n)
```

729

n değeri sayıda kaç tane 9 olduğunu temsil ediyor. $A = 1$ asal olmadığından testten geçecek ve $n = 200 + 1$ değeri ile A sayısı hesaplanacak. Döngü başa dönecek ve A sayısının asal olup olmadığı test edilecek. Asal değilse bir sonraki sayı olan $n = 202$ için A hesaplanacak. Ta ki A asal olana kadar bu böyle devam edecek. Döngü bittiğinde de kaç tane 9 olduğu ekrana yazdırılacak.

Not 6.5.2 Yukarıdaki örnekte A_n dizisini sonlu toplam kullanmadan,

$$A_n = 2 \times 10^{n+1} - 9$$

formülüyle de ifade edebiliriz.

Alıştırma 6.5.3 $x_1 = 1$, $x_2 = 12$, $x_3 = 123$, $x_4 = 1231$, $x_5 = 12312$, $x_6 = 123123$, $x_7 = 1231231$, … *biçiminde, basamakları sadece 1, 2, 3 rakamlarından oluşan sayı dizisini düşünün.* $n > 100$ *koşulunu sağlayan en küçük* x_n *asal sayısı kaç basamaklıdır?*

Alıştırma 6.5.4

$$B_n = \underbrace{111\ldots11}_{n \text{ tane}}$$

formundaki asal sayılardan en az 1000 basamaklı en küçük asal sayı kaç basamaklıdır?

`while` döngüsünde `break` ve `continue` komutları

Pozitif tam sayılardan asal olmayan en az ilk kaç tanesini toplarsak toplam 1000'den büyük olur? Aşağıda buna cevap veren bir program var. Komutları inceleyin. Zaten, pek çoğu tanıdık komutlar. Sadece, `continue` komutunun belli durumları atlamak için kullanıldığına, `break` komutunun da işlemi sonlandırdığına dikkat edin.

```
a=0
b=0
while True:
    a=a+1
    if is_prime(a)==True:
        continue
    b=b+a
    if b>1000:
        print(a)
        break
```

52

Alıştırma 6.5.5 *6.5.4 numaralı örnekteki problemi* `break` *komutu kullanarak çözün.*

6.6 İnteraktif Araç Yaratmak: @interact Komutu

SageMath'deki en etkili, kullanışlı ve de eğlenceli komutlardan biri `@interact` komutu. Adı üzerinde; bu komutu interaktif (etkileşimli) araçlar oluşturmak için kullanacağız. Bunun güzel bir tarafı da yaratılan aracı kodlama bilmeyenlerin de kullanabilmesi.

Çok basit bir örnekle başlayalım. Aşağıdaki girdi kutusundaki kodu olduğu gibi kopyalayıp SageMath'te çalıştırın.

```
@interact
def _( a=input_box(default=5) ):
    print(a^2)
```

Bu, olabilecek en basit etkileşimli kodlardan biri. Kullanıcı bir sayı giriyor ve yanıt olarak girilen sayının karesini elde ediyor.

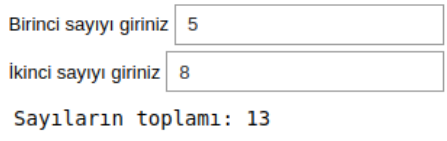
Kodu biraz inceleyelim: `@interact` komutu ile başlıyoruz. Daha sonra fonksiyon tanımlar gibi `def` komutunu kullanıyoruz, sonra da fonksiyonun ismi geliyor. Fonksiyona bir isim verebiliriz, ancak bu isim herhangi bir yerde kullanılmayacağından, genellikle fonksiyon ismi olarak "_" işaretini kullanıyoruz. Bir `a` değişkeni tanımlıyoruz ve varsayılan değer olarak 5'i atıyoruz.

Yukarıdaki örneğe bazı seçenekler ekleyelim: İlk olarak `label` komutu ile girdi kutucuğunu tanımlayan ifadeyi yazabiliriz.

```
@interact
def _( a=input_box(label="Bir sayı giriniz",default=5) ):
    print(a^2)
```

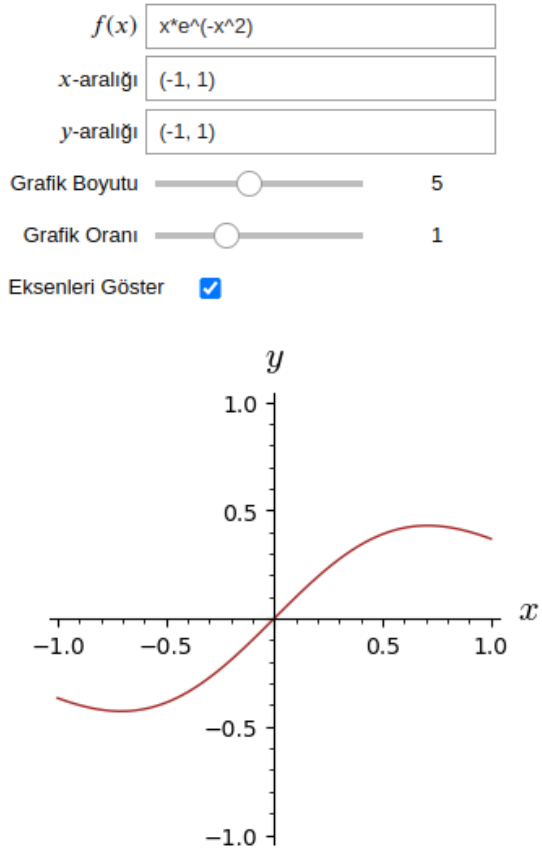
Örnek 6.6.1 Şimdi de iki tane sayı girişini kabul edelim ve bunların toplamını veren etkileşimli bir araç oluşturalım. Aşağıda görüldüğü üzere birden fazla girdi olan tanımlamalarda her bir girdi maddesini alt satıra geçerek tanımlamak kodun açık seçik görünmesi açısından önemli. Aşağı satıra geçmek için enter'a basınca, uygun girinti konusunda SageMath burada bize yardımcı oluyor.

```
@interact
def _(
    a=input_box(label="1. sayıyı giriniz" , default=5),
    b=input_box(label="2. sayıyı giriniz" , default=8)
     ):
    print("Sayıların toplamı:" , a+b)
```



Örnek 6.6.2 x ve y aralıklarını kullanıcının girdiği, verilen bir fonksiyonun grafiğini çizen bir örneği inceleyelim:

```
var("x y")
@interact
def _(f=input_box(label="$f(x)$",default= x*exp(-x^2),width=50 ),
     aralik_x=input_box(label="$x$-aralığı",default= (-1,1),width=25),
     aralik_y=input_box(label="$y$-aralığı",default= (-1,1),width=25 ),
     boyut=slider( [1..10] ,label="Grafik Boyutu",default= 5 ),
     oran=slider( [ 1/4, 1/2, 1 , 2 , 4 , 8 , 16 ] ,label="Grafik
         Oranı",default= 1 ),
     eksen_check=("Eksenleri Göster", True)
    ):
    a=aralik_x[0]
    b=aralik_x[1]
    c=aralik_y[0]
    d=aralik_y[1]
    graf=plot( f, color="brown", ymin=c, ymax=d, xmin=a , xmax=b,
        aspect_ratio=oran, figsize=boyut )
    if eksen_check==True:
       show( graf , frame=False, axes=True, axes_labels=["$x$","$y$"] )
    if eksen_check==False:
       show( graf , frame=True, axes=False, axes_labels=["$x$","$y$"] )
```



Yukarıdaki kodlar için birkaç açıklama/yorum:

1. $f(x)$ fonksiyonu ve eksen aralıkları için toplam üç tane `input_box` oluşturuyoruz. Girdi kutucukları için kutucuğun boyutunu `width` komutuyla belirliyoruz.

2. Grafiğin boyut bilgisi için bir kaydırgaç kullanıyoruz. Bunu `slider` komutu ile yapıyoruz. Boyut bilgisi [1,10] aralığından seçilerek `boyut` değişkenine atanıyor. Kaydırgaç kullanmak istemezseniz daha özgür bir girdi hakkı veren `input_box` kullanmak burada da mümkün tabii ki.

3. Boy/en oranı da benzer şekilde `oran` değişkenine atanıyor. Bu defa değerler açık seçik verilmiş bir listeden seçiliyor.

4. Eksenleri gösterip gizlemek için onay kutusunu (checkbox) yukarıdaki gibi oluşturuyoruz. Varsayılan tercih `True`. Yani eksenler başlangıçta görünüyor.

5. Bu maddeye kadar kullanıcıdan alınacaklar alındı. Şimdi de x-aralığı (a, b), y-aralığı (c, d) olacak şekilde a, b, c, d atamasını yapıyoruz.

6. Grafiği koşullu bir şekilde (`if` kullanarak) gösteriyoruz. Bunu yaparken eksenler ve çerçeve seçeneklerini onay kutusuna göre seçiyoruz.

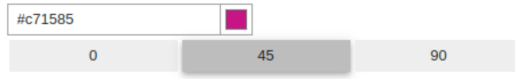
Aşağıda tamamen aynı sonucu veren ama biraz daha pratik yazılmış bir kod var. Bunu inceleyip yorumlamak da size kalsın.

```
var("x y")
@interact
def _(f=input_box(label="$f(x)$",default= x*exp(-x^2),width=50 ),
     aralik_x=input_box(label="$x$-aralığı",default= (-1,1),width=25),
     aralik_y=input_box(label="$y$-aralığı",default= (-1,1),width=25 ),
     boyut=slider( [1..10] ,label="Grafik Boyutu",default= 5 ),
     oran=slider( [ 2^k for k in [-2..4] ] ,label="Grafik
         Oranı",default= 1 ),
     eksen_check=("Eksenleri Göster", True)
    ):
    a,b,c,d = aralik_x[0] , aralik_x[1] , aralik_y[0] , aralik_y[1]
    graf=plot( f, color="brown", ymin=c, ymax=d, xmin=a , xmax=b,
        aspect_ratio=oran, figsize=boyut )
    show( graf , frame=(not eksen_check), axes=eksen_check,
        axes_labels=["$x$","$y$"] )
```

Renk seçicisi ve selektör içeren hızlı bir örnekle bu alt bölümü noktalıyoruz.

Örnek 6.6.3 Aşağıdaki interaktif araç metin ve matematiksel ifadeden oluşan görseli kullanıcının açı seçimine bağlı olarak pozitif yönde çeviriyor:

```
@interact
def _(
renk = color_selector( widget = "colorpicker" , label = " " , default =
    "mediumvioletred" ),
cevir = selector([ 0, 45, 90] , buttons=True , label=" " , default=45 )
):
    metin = text( r" kök iki $\sqrt{2}$" , (0,0) , axes=False,
        fontsize=30 , fontweight="bold" , color=renk , rotation=cevir )
    show(metin)
```



Bazı detaylar:

1. Renk seçimini nasıl etkin hale getirdiğimizi bu örnekte görüyoruz. Kullanıcı istediği rengi renk seçicisiyle belirliyor ve bu bilgi `renk` değişkenine atanıyor. Varsayılan renk olarak `mediumvioletred` rengini kullandık. Nereden mi bulduk bu rengi? `colors` komutu ile! SageMath'e `colors` yazıp çalıştırın, onlarca renk tercihi göreceksiniz. Daha kapsamlı renk seçenekleri 3.8 numaralı alt bölümde.

2. Bir de selektör kullanımı var bu örnekte. Seçeneklerimizi `selector` komutu ile giriyoruz. Bu örneğe özel olarak 0, 45 ve 90 seçeneklerini kullanıcıya gösterip, onun seçimini `cevir` değişkenine atıyoruz. Seçenekleri açılır liste şeklinde göstermek istersek `buttons=True` ifadesini hiç kullanmayabilir ya da `True` yerine `False` yazabiliriz.

3. Daha genel bir seçim grubu oluşturabiliriz, yani sadece sayı değil, metin de kullanılabilir. Genel olarak, `if cevir=="45":` şeklinde bir şartlı yapı çok daha kullanışlı olacaktır.

4. Ne `renk` ne de `cevir` için etiketleme kullandık. İkisinin de ne ifade ettiği kullanıcı için açık seçik olduğundan etikete gerek duymadık.

5. Kalan kısımda orijine matematiksel bir ifade de içeren bir metin yerleştiriyoruz. Eksenleri gizliyoruz. Diğer detayları inceleyip keşfedin.

Alıştırma 6.6.1 *$x \geq 0$ olmak üzere*

$$\sqrt{ax} = \sin\left(\frac{x}{a}\right)$$

denklemini $a > 0$ parametresine bağlı olarak grafik kullanarak inceleyin. Bunun için a parametresini kaydırgaç ile kontrol edeceğiniz interaktif bir araç yaratın. Örneğin $a = 1$ olduğunda denklemin sadece $x = 0$ kökünün olduğunu göreceksiniz.

a. $a = 0.23$ *için denklemin kaç farklı kökü vardır.*

b. Denklemin 3 *farklı kökünün olduğu bir durum var mıdır? Varsa, buna uygun bir a değeri belirtin.*

c. Denklemin 1000 *tane farklı kökünün olduğu bir durum mümkün mü? Açıklayın.*

6.7 Animasyon

Bu alt bölümde SageMath ile nasıl animasyon oluşturacağımızı göreceğiz. SageMath için animasyon demek herhangi bir nesnenin parametreye bağlı değişimini göstermek demektir. Örneğin

$$y = \sin kx$$

fonksiyonunu düşünün. Her farklı k değeri farklı bir grafik verecektir. Mesela, $k = 0$ değerinden başlayıp $k = 5$ değerine kadar 0.2'lik sıçramalarla pek çok grafik elde edebiliriz. Bu grafikleri adeta çizgi film oluşturuyor gibi art arda koyacak olursak bir animasyon elde ederiz. İşte SageMath bizim için aşağıda tam da bunu yapıyor. Komutun adı `animate`, yani *canlandır*. Aslında yaptığımız şey parametreye bağlı bir liste oluşturup bunu `animate` komutu ile canlandırmak. Aşağıdaki kodu çalıştırıp animasyonu görün.

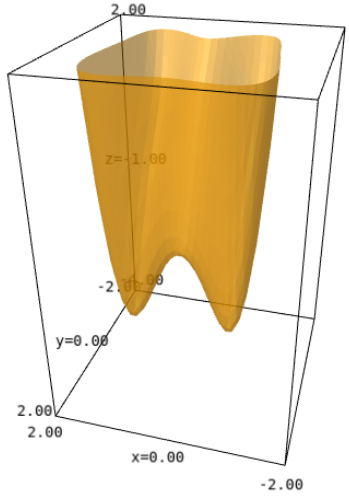
```
animate( [ plot( (sin(k*x)), aspect_ratio=1, xmin=-3, xmax=3, ymin=-1.1,
    ymax=1.1 ) for k in srange(0,5,.2) ] )
```

Oranlama (`aspect_ratio`) ve x,y aralıklarını girmek, görsel kontrolu sağlamak için genellikle iyi fikir.

Örnek 6.7.1 (Kontur Grafik Animasyonu) Şimdi de

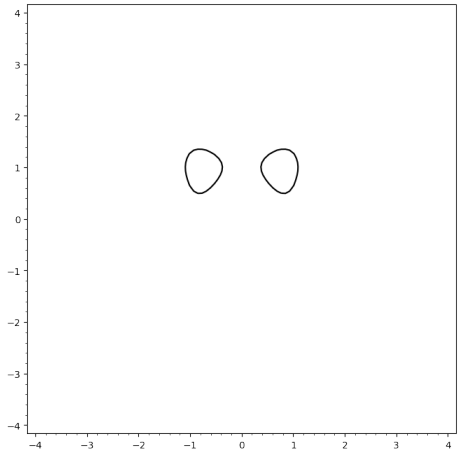
$$f(x, y) = x^4 + y^4 + 2\left(x^2 - 1\right)^2 - (y + 1)^2$$

fonksiyonunu ve bu fonksiyonun aşağıdaki grafiğini düşünelim:



Grafiği verilen bu yüzeyi yatay bir düzlemle kesip, ara kesitini alırsak seviye eğrilerinden birini buluruz. Örneğin aşağıdaki kapalı fonksiyon bunlardan biri; elde edilen grafik de denklemin hemen altında:

$$x^4 + y^4 + 2\left(x^2 - 1\right)^2 - (y+1)^2 = -1.5$$



Şimdi bunu pek çok düzlem ile yapıp pek çok seviye eğrisi bulalım ve bütün çıktıları kullanarak bir animasyon elde edelim. Bu aslında kontur grafiklerin animasyonu demek. Kontur grafik ile ilgili detayları 5.7 numaralı bölümde bulabilirsiniz.

```
animate( [ contour_plot( x^4 + y^4 + 2*(x^2-1)^2-(y+1)^2 , (x,-4,4),
    (y,-4,4), contours=[k], fill=false ) for k in srange(-3,5,.3) ] )
```

Görüntüleyeceğiniz animasyon k değerinin -3'ten 5'e kadar 0.3'lük sıçramalarla oluşacak $z = k$ düzlemlerinin verilen yüzeyle ara kesiti olan eğrilerin oluşturduğu bir animasyon olacaktır.

Not 6.7.1 Bu animasyon manyetik rezonans (MR) tomografinin görüntülerini andırıyor. Onlar da aslında düzemsel kesitler gibi düşünülebilir.

Örnek 6.7.2 (Küçülüp Küçülen Sayılar) Aşağıdaki kodu çalıştırın ve animasyonu görün. Bunu yorumlamak da okuyucuya kalsın.

```
animate( [ text( round(k,1),(0,0), axes=False, fontsize=10*k,
    color="gray" ) for k in [ 5-j for j in srange(0,5,.2) ] ] )
```

Alıştırma 6.7.1 *Birer derecelerle saat yönünde dönen, ve döndüğü açıyı yazan sayıların animasyonunu yapın. Örneğin* 12 *derece saat yönünde dönmüş bir* 12 *sayısı bu animasyonun karelerinden biri.*

Alıştırma 6.7.2 $y = \sin x$ *eğrisi boyunca hareket eden bir parabol animasyonu programlayın.*

Bölüm 7

SageMath ile İleri Matematik

Bu bölümde tek ve çok değişkenli kalkülüs, diferensiyel denklemler ve lineer cebir gibi alanlarda SageMath kullanımını ve bazı önemli komutları inceleyeceğiz.

7.1 Yaygın Kullanılan Fonksiyonlar

Aşağıdaki tablolarda sık kullanılan bazı fonksiyonların SageMath komutları verilmiştir:

Fonksiyon	**Komut**
Karekök	`sqrt(x)`
Mutlak Değer	`abs(x)`
Üstel	`exp(x)`
Doğal Logaritma	`ln(x)` veya `log(x)`
a Tabanında Logaritma	`log(x , a)`

Fonksiyon	**Komut**
Sinüs	`sin(x)`
Kosinüs	`cos(x)`
Tanjant	`tan(x)`
Kotanjant	`cot(x)`
Sekant	`sec(x)`
Kosekant	`csc(x)`

Fonksiyon	Komut	Fonksiyon	Komut
Arksinüs	`arcsin(x)`	Sinüs hiperbolik	`sinh(x)`
Arkkosinüs	`arccos(x)`	Kosinüs hiperbolik	`cosh(x)`
Arktanjant	`arctan(x)`	Tanjant hiperbolik	`tanh(x)`
Arkkotanjant	`arccot(x)`	Kotanjant hiperbolik	`coth(x)`
Arksekant	`arcsec(x)`	Sekant hiperbolik	`sech(x)`
Arkkosekant	`arccsc(x)`	Kosekant hiperbolik	`csch(x)`

Not 7.1.1 Ters trigonometrik fonksiyonlar için fonksiyonların başına `arc` yerine `a` eklesek de olur. `asin`, `atan` gibi. Küçük bir örnek:

```
asin(1) , acos(1) , atan(1) , acot(1) , asec(1) , acsc(1)
```

```
(1/2*pi, 0, 1/4*pi, 1/4*pi, 0, 1/2*pi)
```

7.2 Limit

Basit bir limit problemiyle başlayalım:

$$\lim_{x\to 1} \frac{x^3-1}{x-1}$$

```
limit( (x^3- 1) / (x - 1), x = 1)
```

```
3
```

Limit almak için `limit` veya `lim` komutunu kullanıyoruz. Yukarıdaki örnekte x, 1'e gidiyor; bunu `x = 1` şeklinde ifade ediyoruz. Bir örnek daha:

$$\lim_{x\to 0} \frac{1-e^{-x}}{x}$$

```
lim( (1-exp(-x)) / x , x = 0)
```

```
1
```

Şimdi de biraz karmaşık görünen aşağıdaki limiti düşünelim:

$$\lim_{x\to 0} \frac{\sin\left(x^{18}-x^{20}\right)}{1-\cos x^9}$$

```
lim( sin(x^18-x^20)/(1-cos(x^9)) , x = 0)
```

```
2
```

SageMath hızlı bir şekilde sonucun 2 olduğunu söylüyor. Normal şartlarda bu problem için 18 defa L'Hospital Kuralı uygulamamız gerekir ki bu da pek çok türev almak ve karmaşık işlemler yapmak demek.

Sağdan limit için `dir = "plus"`, `dir = "right"` ya da `dir = "+"`; soldan limit için de `dir = "minus"`, `dir = "left"` ya da `dir = "-"` kullanılabilir. Buradaki `dir` ifadesi *direction* (yani *yön*) kelimesinin kısaltılmış hali. Hızlı bir örnek:

$$\lim_{x\to 0^+} \ln x$$

```
limit( ln(x), x = 0, dir="plus")
```

```
-Infinity
```

Bir örnek daha:

$$\lim_{x\to 1^+} \frac{(x-1)^x}{\sin(\pi x)}$$

```
limit( (x-1)^x / sin(pi*x), x = 1, dir="+" )
```

```
-1/pi
```

Aşağıdaki örnekte tam değer fonksiyonunun 5 noktasındaki soldan ve sağdan limitini buluyoruz:

$$\lim_{x\to 5^-} [\![x]\!] \quad \text{ve} \quad \lim_{x\to 5^+} [\![x]\!]$$

```
lim( floor(x) , x=5, dir="-" ) , lim( floor(x) , x=5, dir="+" )
```

```
(4, 5)
```

Bağımsız değişkenin sonsuza gittiği limitleri de aşağıdaki örneklerdeki gibi hesaplayabiliriz. Sonsuz (∞) ifadesini `Infinity`, `infinity` ya da simgesel olarak sonsuz işaretine biraz benzeyen yanyana çift küçük o harfi, yani `oo` şeklinde yazıyoruz.

Basit bir örnek:

$$\lim_{x\to\infty} \frac{4x-11}{3x+13}$$

```
lim( (4*x-11) / (3*x+13) , x = oo)
```

```
4/3
```

Son bir limit örneği:

$$\lim_{x\to\infty} \frac{\sqrt{x}\ln x}{\sqrt{1+x^2}}$$

```
lim( sqrt(x)*log(x) / sqrt(x^2+1), x = oo)
```

```
0
```

Not 7.2.1 Limitleri alınan yukarıdaki fonksiyonların grafiklerini uygun aralıklarda çizdirip limit değerlerini gözlemleyebilirsiniz.

Alıştırma 7.2.1 *Aşağıdaki limitin değerini SageMath ile hesaplayın:*

$$\lim_{x\to 0^+} x^{1-\cos x}$$

7.3 Türev ve Kısmi Türev

Türev almak için kullanılan komutlar `diff` ya da `derivative`. Örneğin $y = \sin x$ fonksiyonunun x değişkenine göre türevini alalım:

```
diff(sin(x) , x )
```

```
cos(x)
```

Yüksek mertebeden türev de benzer şekilde hesaplanıyor. Aşağıda $y = \cos(\ln x)$ fonksiyonunun 7'nci mertebeden türevini alıyoruz ve sonucu daha anlaşılır görmek için `show` komutunu kullanıyoruz.

```
diff( cos( log(x) ) , x , 7 ) .show()
```

$$\frac{1050\,\cos(\log(x))}{x^7} + \frac{730\,\sin(\log(x))}{x^7}$$

Kısmi türev de aynı komut ile hesaplanır. Örneğin aşağıdaki çok değişkenli fonksiyonun y değişkenine göre kısmi türevini hesaplayalım:

$$f(x,y) = \frac{xy}{2x-3y}$$

```
var("y")
diff((x*y)/(2*x-3*y) , y ).show()
```

$$\frac{x}{2\,x-3\,y} + \frac{3\,xy}{(2\,x-3\,y)^2}$$

Son olarak da

$$g(x,y) = xe^{-x-2y}$$

fonksiyonunun x değişkenine göre 100'üncü mertebeden kısmi türevinin $(-2,1)$ noktasındaki değerini bulalım.

```
var("y")
x_turev = diff(x*exp(-x-2*y) , x , 100 )
show(x_turev)
x_turev.subs(x=-2, y=1 )
```

$x e^{(-x-2\,y)} - 100\, e^{(-x-2\,y)}$

```
-102
```

Alıştırma 7.3.1

$$F(u, v, w) = \log(u + \log(v + \log(w)))$$

foksiyonunun w değişkenine göre dördüncü mertebeden kısmi türevinin $(1, 1, 1)$ *noktasındaki değerini bulun.*

Alıştırma 7.3.2 $x > 0$ *olmak üzere,* x^x *ifadesinin en küçük değerini yaklaşık olarak bulun. 4.5 numaralı alt bölüm kök bulmada yardımcı olabilir.*

7.4 Belirsiz İntegral

İntegral için `integrate` (*integralini al*) veya `integral` (*integral*) komutlarını kullanıyoruz. Aşağıdaki belirsiz integrali hesaplayalım:

$$\int \frac{dx}{1+x^2}$$

```
integrate(1/(1+x^2), x)
```

```
arctan(x)
```

Görüldüğü üzere fonksiyonu girdikten sonra integrali x değişkenine göre almak istediğimizi belirtiyoruz. SageMath cevapta belirsiz intagraldeki $+C$ sabitini göstermemeyi tercih ediyor.

Biraz daha karmaşık bir örnek:

$$\int \sqrt{1-\sqrt{x}}\, dx$$

```
integral( sqrt(1-sqrt(x)), x).simplify_full().show()
```

$\frac{4}{15}\left(3\,x - \sqrt{x} - 2\right)\sqrt{-\sqrt{x}+1}$

Bir de aşağıdaki integrali $s = 1, 2, 3$ için SageMath yardımıyla hesaplayalım:

$$\int e^{x^s}\, dx$$

```
A = [ integral( exp(x^s), x ) for s in [1,2,3] ]
show(A)
```

$$\left[e^x, -\frac{1}{2} i \sqrt{\pi}\,\mathrm{erf}\,(i\,x), -\frac{x\Gamma\left(\frac{1}{3},-x^3\right)}{3\left(-x^3\right)^{\frac{1}{3}}}\right]$$

Yukarıda elde ettiğimiz sonuç söz konusu üç durumu listeliyor. $s = 1$ durumu çok masum bir integral verse de diğer iki durum temel fonksiyonlarla ifade edilemeyen integral sonuçları veriyor: $s = 2$ durumu *hata fonksiyonu* (*error function*) denilen erf fonksiyonu ile, $s = 3$ durumu da Γ fonksiyonu (*gama fonksiyonu*) ile ifade ediliyor.

Şimdi de çok değişkenli fonksiyonlar için iki örnek inceleyelim. İşleyiş tamamen aynı; sadece komutu birden fazla kez kullanıyoruz.

$$\iint \cos\left(\ln\left(xy\right)\right)\,dx\,dy$$

```
var("x y")
integral(integral(cos(log(x*y)), x),y).simplify_full().show()
```

$$\frac{1}{2}\,xy\sin\left(\log\left(xy\right)\right)$$

$$\iiiint wx^2yz^3\,dx\,dy\,dw\,dz$$

```
var("x y z w")
integral(integral(integral(integral( w*x^2*y*z^3 , x),y),w),z).show()
```

$$\frac{1}{48}\,w^2x^3y^2z^4$$

7.5 Belirli İntegral

Basit bir belirli integral örneği ile başlayalım:

$$\int_0^1 x^2\,dx$$

```
integrate(x^2, x, 0, 1 )
```

```
1/3
```

İntegral aralığını nasıl girdiğimizi inceleyin. Bir örnek daha:

$$\int_0^1 x^3 e^{1+x^2}\, dx$$

```
integrate(x^3*exp(1+x^2), x, 0, 1 )
```

```
1/2*e
```

Aşağıdaki örnekte $b > 0$ ve $c > 0$ varsayımlarını eklemek için 4.6 numaralı bölümde gördüğümüz `assume` komutunu kullanıyoruz:

$$\int_{-c}^{c} \frac{dx}{x^2 + 2kx + k^2 + b}$$

```
var("c b k")
assume(b>0)
assume(c>0)
integral(1/( x^2+2*k*x+(k^2+b)), x , -c, c).show()
```

$$\frac{\arctan\left(\frac{c+k}{\sqrt{b}}\right)}{\sqrt{b}} - \frac{\arctan\left(-\frac{c-k}{\sqrt{b}}\right)}{\sqrt{b}}$$

Alıştırma 7.5.1 *Aşağıda integrali alınan fonksiyon* $(0, 1)$ *aralığında pozitif olduğundan, belirli integralin sonucu da pozitif olacaktır. İntegrali SageMath ile hesaplayıp eşitsizliği yorumlayın:*

$$\int_0^1 \frac{x^4(1-x)^4}{1+x^2} dx > 0$$

Alıştırma 7.5.2 (Katenari) *Zincir eğrisi de denilen katenari, iki ucundan tutulan bir zincirin kendi ağırlığının etkisiyle oluşturduğu eğriye verilen ad. Bu eğri aşağıdaki hiperbolik fonksiyon ile tanımlanıyor:*

$$y = a \cosh\left(\frac{x}{a}\right)$$

Farklı a değerleri için bu eğrinin grafiğini çizin ve eğri uzunluğu formülünü kullanarak $a = 2$ *için* $0 \le x \le 3$ *aralığında eğri uzunluğunu hesaplayın.*

İki ya da çok katlı belirli integraller de aynı işlemin tekrarıyla hesaplanır. Üç katlı bir integral içeren bir örnek verelim:

$$\int_0^1 \int_0^{1-x} \int_0^{1-x-y} xyz\,dz\,dy\,dx$$

```
var("x y z")
integrate( integrate( integrate( x*y*z, (z, 0, 1-x-y) ), (y, 0, 1-x) ),
    (x, 0, 1) )
```

1/720

7.6 Has Olmayan İntegral

Has olmayan integralleri de belirli integrallere benzer şekilde hesaplıyoruz. Sonsuzu (∞) belirtmek için `infinity`, `Infinity` veya `oo` komutlarını kullandığımızı hatırlatalım. Örnekleri inceleyiniz:

$$\int_0^{\infty} e^{-x}\,dx$$

```
integrate(exp(-x), x, 0, infinity)
```

1

$$\int_0^1 \frac{dx}{\sqrt{x}}$$

```
integrate(1/sqrt(x), x, 0, 1)
```

2

$$\int_{-\infty}^{\infty} \frac{dx}{1+x^4}$$

```
integral(1/(1+x^4), x , -oo, oo).show()
```

$\frac{1}{2}\sqrt{2}\pi$

$$\int_0^{\infty} \frac{e^{-x}}{\sqrt{x}}\,dx$$

```
sonuc = integrate(exp(-x)/sqrt(x), x, 0, infinity)
show(sonuc)
```

$\sqrt{\pi}$

Alıştırma 7.6.1 *$\alpha \in \mathbb{R}$ olmak üzere, aşağıdaki has olmayan integrali hesaplayın:*

$$\int_{-\infty}^{\infty} \frac{dx}{x^2 + 2\alpha x + \alpha^2 + 1}$$

Alıştırma 7.6.2 *Aşağıdaki has olmayan integrali $k = 1, 2, 3, 4, 5$ için hesaplayın:*

$$\int_{-\infty}^{\infty} \frac{\sin^k x}{x^k}\, dx$$

Alıştırma 7.6.3 *Aşağıdaki integrali hesaplayın:*

$$\int_{0}^{\infty} \frac{x^2 + 3x + 1}{x^4 + 1}\, dx$$

7.7 Nümerik İntegral

Belirli integrali hesaplamak bilinen integral yöntemleriyle her zaman mümkün olmayabilir. Bu durumlarda `integral_numerical` komutunu aşağıdaki gibi kullanarak integralin nümerik yaklaşık değerini hesaplayabiliriz:

$$\int_{0}^{\pi} \frac{\sin x}{x}\, dx$$

```
integral_numerical(sin(x)/x, 0, pi)
```

```
(1.8519370519824667, 2.05606315526737e-14)
```

Yukarıda SageMath'in verdiği ilk değer integral sonucunun yaklaşık değerini, ikinci değer de yaklaşık hatayı vermekte. Hata sıfıra oldukça yakın.

7.4 numaralı bölümde $s = 1, 2, 3$ için incelediğimiz integrallerin aşağıdaki belirli integral halini nümerik olarak hesaplayalım:

$$\int_{0}^{1} e^{x^s}\, dx$$

```
[ integral_numerical( exp(x^s), 0, 1 ) for s in [1,2,3] ]
```

```
[(1.718281828459045, 1.9076760487502454e-14),
 (1.4626517459071817, 1.6238696453143372e-14),
 (1.3419044179774198, 1.4898131816847513e-14)]
```

Not 7.7.1 Yukarıdaki integraller verilen fonksiyonların $(0, 1)$ aralığındaki grafiklerinin altındaki alanı verir. Bu üç fonksiyonun grafiklerini verilen aralıkta çizdirip kıyaslamanız cevapları daha anlaşılır yapacaktır.

Alıştırma 7.7.1 *Optikte kullanılan Fresnel integralleri aşağıdaki gibi tanımlanır:*

$$S(x) = \int_0^x \sin\left(t^2\right)\, dt, \quad C(x) = \int_0^x \cos\left(t^2\right)\, dt$$

$S(1)$ *ve* $C(1)$ *değerlerini nümerik olarak bulun.*

7.8 Diziler

Dizi en somut ve SageMath'e tanıdık bir ifadeyle bir listedir. Aşağıda 10 terimli bir dizi örneği var:

$$\left\{ \frac{n}{n+1} \right\}_{n=1}^{10}$$

Bu dizinin terimlerini listeleyelim:

```
[ k/(k+1) for k in [1..10] ]
```

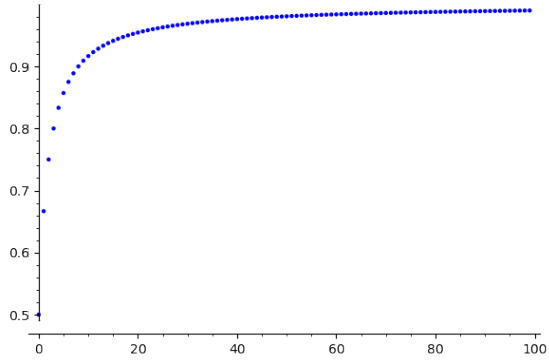
```
[1/2, 2/3, 3/4, 4/5, 5/6, 6/7, 7/8, 8/9, 9/10, 10/11]
```

Şimdi de matematiksel olarak daha ilgi çekici olan aşağıdaki sonsuz terimli diziyi düşünelim:

$$\left\{ \frac{n}{n+1} \right\}_{n=1}^{\infty}$$

Sonsuz terimli bir dizi için önemli sorulardan biri çok büyük n değerleri için dizinin terimlerinin belli bir sayıya yaklaşıp yaklaşmadığı, yani limitinin olup olmadığıdır. Bunu nümerik olarak görmek istersek dizinin belli sayıda terimini yukarıda yaptığımız gibi listeleyebiliriz. SageMath ile limit hesaplamadan önce, ilk 100 terimin grafiksel ifadesini görelim:

```
list_plot( [k/(k+1) for k in [1..100] ] )
```



Yukarıdaki grafik, dizinin terimlerinin 1'e çok yaklaşacağı, yani dizinin limitinin 1 olduğu hissini veriyor. Bunu limit alarak doğrulayabiliriz. Diziler için limit tıpkı fonksiyonlarda yaptığımız gibi hesaplanır. Komut `lim` veya `limit`:

$$\lim_{k\to\infty} \frac{k}{k+1}$$

```
var("k")
lim( k/(k+1) , k=oo )
```

```
1
```

Yukarıdaki sonucun grafik ile örtüştüğünü görüyoruz.

Not 7.8.1 Bir dizinin davranışını grafik yardımıyla daha iyi anlamak için birkaç ipucu:

1. Yukarıdaki grafiği $y = 1$ doğrusunun grafiği ile aynı koordinat sisteminde çizdirerek daha ikna edici bir sonuç elde edebilirsiniz. Birden fazla grafik ögesi kullanmak için 3.11 numaralı bölüme bakabilirsiniz.

2. Davranışı yukarıdaki grafik kadar belirgin olmayan dizilerde `list_plot` komutuna `plotjoined=True` seçeneğini ekleyebiliriz. Böylece art arda gelen noktaları bir doğru parçasıyla birleştirip daha belirgin bir resim elde etmiş oluruz. Örnek için aşağıdaki satırı çalıştırabilirsiniz.

   ```
   list_plot( [ (-1)^n*n/(n+1) for n in [1..50] ] , plotjoined=True )
   ```

3. Dizinin büyük sayılardaki davranışını anlamak için, grafik büyük sayılardan başlatılabilir. Mesela, 10^{10}'dan $10^{10} + 100$'e.

İki limit örneği daha verelim:

$$\lim_{k\to\infty} \left(\frac{1}{k}\right)^{\frac{1}{k}}$$

```
var("k")
lim( (1/k)^(1/k) , k=oo )
```

```
1
```

$$\lim_{k\to\infty} \frac{k}{\sqrt[k]{k!}}$$

```
var("k")
limit( k/(factorial(k))^(1/k), k=oo )
```

```
e
```

7.9 Sonlu Toplam

Bir dizinin sonlu sayıda elemanlarının toplamını `sum` (yani, *toplam*) komutuyla hesaplayabiliriz. Örneğin:

$$\sum_{k=1}^{5} k^2$$

```
var("k")
sum( k^2 , k , 1 , 5 )
```

```
55
```

Aşağıda $\left\{ \frac{1}{n^2} \right\}$ dizisinin ilk 20000 teriminin toplamının yaklaşık değerini buluyoruz:

```
var("i")
toplam=sum( 1/i^2 , i , 1, 20000 )
n(toplam)
```

```
1.64488406809821
```

7.10 Seriler

Sonsuz terimli dizilerin terimlerinin toplamına *seri* dendiğini hatırlayınız. SageMath sonsuz toplamı da sonlu toplama benzer bir şekilde hesaplıyor. Örneğin

$$\sum_{k=1}^{\infty} \frac{1}{k^2}$$

toplamına bakalım:

```
sum(1/k^2, k, 1, oo)
```

```
1/6*pi^2
```

Bu sonuç serinin yakınsak olduğunu söylüyor bize. Ayrıca bu sonucun nümerik ifadesini bularak, önceki bölümde elde ettiğimiz 20000 terimin toplamına oldukça yakın olduğunu görebilirsiniz.

Aşağıda, 6.5 numaralı alt bölümde bahsettiğimiz Harmonik Seri var. SageMath bunun bir ıraksak seri olduğunu "Sum is divergent" cevabıyla ifade ediyor.

$$\sum_{k=1}^{\infty} \frac{1}{k}$$

```
sum(1/k, k, 1, oo)
```

```
ValueError: Sum is divergent.
```

Bir örnek daha:

$$\sum_{m=1}^{\infty} \frac{m^7}{7^m}$$

```
var("m")
sum(m^7/7^m, m, 1, infinity)
```

```
285929/11664
```

7.11 Taylor ve Maclaurin Serileri

Reel (ya da kompleks) bir f fonksiyonunun, a bir reel (ya da kompleks) sayı olmak üzere, a noktasında sonsuz defa türevlenebildiğini varsayalım. f fonksiyonunun a noktasındaki *Taylor serisi* aşağıdaki gibi tanımlanır:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

Bu seriye $a = 0$ özel durumunda *Maclaurin serisi* de denir. İlk $n+1$ terimden oluşan sonlu toplama n'inci Taylor polinomu diyoruz ve bu polinomu `taylor` komutunu kullanarak elde ediyoruz. Aşağıda komutun iki farklı kullanım biçimi var.

İlk olarak, örneğin

$$f(x) = e^{-x}$$

fonksiyonunun $x = 0$ noktasında 9'uncu dereceden Taylor polinomunu bulalım:

```
taylor( exp(-x), x, 0,9 )
```

```
-1/362880*x^9 + 1/40320*x^8 - 1/5040*x^7 + 1/720*x^6 - 1/120*x^5 +
    1/24*x^4 - 1/6*x^3 + 1/2*x^2 - x + 1
```

Şimdi de

$$f(x) = x^2 \cos x$$

fonksiyonunun $x = \pi$ noktasında 4'üncü dereceden Taylor polinomunu buluyoruz:

```
f=x^2*cos(x)
show(f.taylor(x,pi,4))
```

$$-\frac{1}{24}(\pi - x)^4\left(\pi^2 - 12\right) - \pi(\pi - x)^3 + \frac{1}{2}(\pi - x)^2\left(\pi^2 - 2\right) - \pi^2 + 2\pi(\pi - x)$$

7.12 Vektör İşlemleri

Aşağıda $\mathbf{v} = \langle 1, 2, 3 \rangle$ ve $\mathbf{w} = \langle 0, -2, 1 \rangle$ vektörlerini tanımlıyoruz:

```
v=vector( [1,2,3] )
w=vector( [0,-2,1] )
```

Tanımladık. Şimdi bu iki vektör ile çeşitli işlemler yapalım. Mesela, $\mathbf{v} + \mathbf{w}$ ve $2\mathbf{v} - 3\mathbf{w}$ lineer kombinasyonlarını hesaplayalım:

```
v+w
```

```
(1, 0, 4)
```

```
2*v-3*w
```

```
(2, 10, 3)
```

İç Çarpım

İç (skaler) çarpım için `dot_product` komutunu ya da * (asterisk) işaretini kullanıyoruz. Örneğin $\mathbf{v} \cdot \mathbf{w}$ işlemi aşağıdaki gibi yapılabilir:

```
v.dot_product(w)
```

```
-1
```

Vektörel Çarpım

Vektörel çarpım için `cross_product` komutunu kullanıyoruz. Aşağıda sırasıyla $\mathbf{v} \times \mathbf{w}$ ve $\mathbf{w} \times \mathbf{v}$ işlemleri var. Burada sıranın önemli olduğunu hatırlatalım.

```
v.cross_product(w)
```

```
(8, -1, -2)
```

```
w.cross_product(v)
```

```
(-8, 1, 2)
```

Vektör Uzunluğu

Bir **v** vektörünün uzunluğunu, yani |**v**| değerini `norm` komutunu aşağıdaki şekillerde kullanarak bulabiliriz:

```
v.norm()
```

```
sqrt(14)
```

```
norm(v)
```

```
sqrt(14)
```

7.13 Matris İşlemleri

Bir matrisi tanımlamak için `matrix` komutunu kullanırız. Örneğin aşağıdaki A matrisini tanımlayıp `show` komutuyla gösterelim:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

```
A = matrix([[1,2,3],[4,5,6]])
show(A)
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Matris tanımlamanın bir başka yöntemi de önce matrisin boyutunu sonra da art arda satırları girmek. Örneğin:

```
A=matrix( 2,3, [1,2,3,4,5,6] )
show(A)
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Özel olarak kare matrisler için sadece satır sayısını belirtmek yeterlidir. Örneğin `matrix(2,2, [1,2,3,4])` matrisi `matrix(2, [1,2,3,4])` şeklinde de ifade edilebilir.

Şimdi de toplama, çıkarma, skalerle çarpma ve matris çarpımı işlemlerini inceleyelim. Bunun için önce iki matris tanımlayalım:

```
A=matrix( 2, [4,-1,1,0] )
B=matrix( 2, [-1,1,4,0] )
```

A ve B matrislerini tanımladık. Şimdi aşağıdaki işlemleri takip edin:

```
show(A)
```

$$\begin{pmatrix} 4 & -1 \\ 1 & 0 \end{pmatrix}$$

```
show(B)
```

$$\begin{pmatrix} -1 & 1 \\ 4 & 0 \end{pmatrix}$$

```
show(A+B)
```

$$\begin{pmatrix} 3 & 0 \\ 5 & 0 \end{pmatrix}$$

```
show(A-3*B)
```

$$\begin{pmatrix} 7 & -4 \\ -11 & 0 \end{pmatrix}$$

```
show(A*B)
```

$$\begin{pmatrix} -8 & 4 \\ -1 & 1 \end{pmatrix}$$

```
show( A^2-B^2 )
show( (A-B)*(A+B) )
```

$$\begin{pmatrix} 10 & -3 \\ 8 & -5 \end{pmatrix}$$

$$\begin{pmatrix} 5 & 0 \\ -9 & 0 \end{pmatrix}$$

Alıştırma 7.13.1 *Aşağıda verilen A ve B matrisleri için* $(A+B)^2$ *ve* $A^2+2AB+B^2$ *işlemlerini yapın ve sonucu yorumlayın:*

$$A=\begin{pmatrix} 2 & 5 \\ -1 & 1 \end{pmatrix}, \quad B=\begin{pmatrix} 3 & 1 \\ -2 & 0 \end{pmatrix}$$

Alıştırma 7.13.2

$$K=\begin{pmatrix} -1 & -1 \\ 3 & 2 \end{pmatrix}$$

matrisi veriliyor. SageMath'te

a. K matrisini tanımlayın.

b. K^{97} *matrisini hesaplayın.*

c. $\sum_{i=1}^{1000} K^i$ *toplamını hesaplayın.*

Aşağıda *birim matris*in kullanıldığı birkaç örnek var. Göreceğiniz üzere $k \times k$ boyutunda birim matris için `matrix.identity(k)` komutu kullanılır:

```
show(matrix.identity(2))
```

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Birim matrisi matris işlemleri içinde kullanıyorsak 1 olarak yazabiliriz:

```
A=matrix( 2, [4,-1,1,0] )
show(A+matrix.identity(2))
show(A+1)
```

$$\begin{pmatrix} 5 & -1 \\ 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 5 & -1 \\ 1 & 1 \end{pmatrix}$$

Benzer şekilde, aşağıda 7 dediğimiz şey de birim matrisin 7 katı anlamına geliyor:

```
A=matrix( 2, [4,-1,1,0] )
show(A-7)
```

$$\begin{pmatrix} -3 & -1 \\ 1 & -7 \end{pmatrix}$$

```
show(A^2-4*A+1)
```

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Şimdi de aşağıdaki M matrisi ile bazı yaygın kullanılan işlemleri yapalım:

$$M = \begin{pmatrix} 1 & 0 & -2 \\ 0 & 1 & 3 \\ 0 & 1 & -1 \end{pmatrix}$$

```
M=matrix( 3, [ 1,0,-2,0,1,3,0,1,-1 ] )
```

M matrisinin devriği (transpozu), yani M^T:

```
show( M.transpose() )
```

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ -2 & 3 & -1 \end{pmatrix}$$

M matrisinin izi (*trace*), yani $\mathrm{tr}(M)$:

```
show( M.trace() )
```

1

M matrisinin determinantı, yani $\det(M)$:

```
show( M.det() )
```

-4

M matrisinin ters matrisi, yani M^{-1}:

```
show( M.inverse() )
```

$$\begin{pmatrix} 1 & \frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{1}{4} & \frac{3}{4} \\ 0 & \frac{1}{4} & -\frac{1}{4} \end{pmatrix}$$

M matrisinin indirgenmiş eşelon formu:

```
show( M.rref() )
```

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

M matrisinin rankı, yani Rank(M):

```
show( M.rank() )
```

3

M matrisinin ek (adjoint) matrisi, yani Ek(M):

```
show( M.adjugate() )
```

$$\begin{pmatrix} -4 & -2 & 2 \\ 0 & -1 & -3 \\ 0 & -1 & 1 \end{pmatrix}$$

M matrisinin karakteristik polinomu:

```
show( M.charpoly() )
```

$$x^3 - x^2 - 4x + 4$$

M matrisinin özdeğerleri (eigenvalues):

```
show( M.eigenvalues() )
```

$$[2, 1, -2]$$

Son olarak, daha kapsamlı bir şekilde özdeğerler (eigenvalues) ve onlara karşılık gelen özvektörler (eigenvectors) ve cebirsel çokluklarını (algebraic multiplicities) birlikte veren bir komut:

```
show( M.eigenvectors_right() )
```

$$\left[\left(2, \left[\left(1, -\frac{3}{2}, -\frac{1}{2}\right)\right], 1\right), (1, [(1,\ 0,\ 0)], 1), \left(-2, \left[\left(1, -\frac{3}{2}, \frac{3}{2}\right)\right], 1\right)\right]$$

Yukarıdaki son çıktıda, M matrisinin 2, 1 ve -2 olmak üzere üç tane özdeğerinin olduğunu görüyoruz. Bununla birlikte her bir özdeğerin yanında ona karşılık gelen özvektörü ve cebirsel çokluğu da veriliyor. Örneğin 2 özdeğerine karşılık gelen özvektör $\left\langle 1, -\frac{3}{2}, -\frac{1}{2} \right\rangle$ ve bu özdeğerin cebirsel çokluğu 1.

7.14 Doğrusal Denklem Sistemleri

SageMath ve benzeri matematik yazılımlarının becerikli oldukları alanlardan biri de doğrusal (lineer) denklem sistemi çözmektir. Aşağıdaki örnekler üzerinden denklem sistemlerini SageMath ile nasıl çözeceğimizi görelim.

Örnek 7.14.1

$$x + y = 7$$
$$x - y = 11$$

```
var("y")
solve( [ x+y==7, x-y==11], (x,y) )
```

[[x== 9, y== -2]]

Örnek 7.14.2

$$3x + y - 11z = 700$$
$$x - 9y - 2z = 600$$

```
var("y z")
cozum=solve( [ 3*x+y-11*z==700, x-9*y-2*z==600 ], (x,y,z) )
show(cozum)
```

$$\left[\left[x = \frac{101}{28}\,r_3 + \frac{1725}{7}, y = \frac{5}{28}\,r_3 - \frac{275}{7}, z = r_3\right]\right]$$

Çözüm r_3 diye bir parametre içeriyor. Bu parametreye bağlı sonsuz çözüm bulunabilir.

Örnek 7.14.3

$$x - y + 5z = 1$$
$$-x + y + z = 4$$
$$x - y + z = 1$$

```
var("y z")
cozum=solve( [ -x+y+z==0, x-y+z==1, x-y+5*z==1 ], (x,y,z) )
show(cozum)
```

```
[]
```

Görüldüğü üzere elemanı olmayan bir liste elde ettik. Buradan sistemin çözüm kümesinin boş küme olduğunu anlıyoruz.

Alıştırma 7.14.1 *Aşağıdaki denklem sistemini sağlayan 20 tane* (x, y, z) *üçlüsü bulun-/listeleyin:*

$$x + y - 4z = 17$$
$$2x - 3y + z = 11$$

7.15 Adi Diferensiyel Denklemler

Bu alt bölümde `desolve` komutunu kullanarak diferensiyel denklem çözeceğiz. Komut, *differential equation* (yani diferensiyel denklem) teriminin kısaltılmış hali olan `de` ile `solve` (yani *çöz*) ifadelerinin birleşmesinden oluşuyor.

Aşağıdaki birinci mertebeden lineer diferensiyel denklemi SageMath kullanarak çözelim:

$$y' - 3y = 2x$$

```
x = var("x")
y = function("y")(x)
desolve(diff(y,x) -3*y == 2*x , y)
```

```
-1/9*(2*(3*x + 1)*e^(-3*x) - 9*_C)*e^(3*x)
```

Yukarıda ilk satırda x'in bir bağımsız değişken, ikinci satırda da y'nin x'e bağlı bir fonksiyon olduğunu belirtiyoruz. Sonra da diferensiyel denklemi `desolve` komutuyla yazıp denklemi y fonksiyonuna göre çözmek istediğimizi belirtiyoruz. y'nin x'e göre türevini `diff(y,x)` komutu ile ifade ettiğimizi hatırlayın.

Yanıtı daha anlaşılır bir şekilde görmek için denklemin çözümünü `cozum` diye bir değişkene atayıp `show` komutunu kullanalım:

```
cozum = desolve( diff(y,x) -3*y == 2*x , y)
show(cozum)
```

$$-\frac{1}{9}\left(2\,(3\,x+1)e^{(-3\,x)} - 9\,C\right)e^{(3\,x)}$$

Elde ettiğimiz bu ifadeyi de `expand` komutu ile açıp daha sade bir sonuç bulalım:

```
show(cozum.expand())
```

$$Ce^{(3\,x)} - \frac{2}{3}\,x - \frac{2}{9}$$

Not 7.15.1 Bilindiği üzere pek çok diferensiyel denklem analitik olarak çözülememektedir. Ancak bilinen denklemlerin birçoğu için `desolve` komutunun etkili olduğunu söyleyebiliriz.

İkinci bir örnek olarak yine birinci mertebeden olan aşağıdaki Bernoulli denklemini çözelim. Bu defa hangi metodun kullanıldığını da `show_method=True` komutuyla sorgulayalım.

$$y' - 3y = 2xy^2$$

```
x = var('x')
y = function('y')(x)
cozum = desolve( diff(y,x) -3*y == 2*x*y^2 , y, show_method=True )
show(cozum)
```

$$\left[-\frac{9\,e^{(3\,x)}}{2\,(3\,x-1)e^{(3\,x)}-9\,C}, \texttt{bernoulli}\right]$$

7.16 Başlangıç Değer Problemi

Yukarıdaki Bernoulli denklemine bir başlangıç koşul ekleyerek problemi aşağıdaki başlangıç değer problemine çevirelim, sonra da çözelim:

$$y' - 3y = 2xy^2, \quad y(0) = 5$$

```
x = var('x')
y = function('y')(x)
cozum = desolve( diff(y,x) -3*y == 2*x*y^2 , y, ics=[0,5] )
show(cozum)
```

$$-\frac{45\,e^{(3\,x)}}{10\,(3\,x-1)e^{(3\,x)}+1}$$

Farklı olarak sadece `ics` (*initial conditions*, yani *başlangıç koşulları* ifadesinin kısaltılmış hali) komutunu eklediğimize, $y(0) = 5$ başlangıç koşulunu $[0, 5]$ şeklinde gösterdiğimize dikkat edin.

Yüksek mertebeden denklemler de benzer şekilde çözülür. Aşağıda sabit katsayılı homojen olmayan ikinci mertebeden lineer bir başlangıç değer problemi var. Daha önceden bildiğimiz üzere ikinci mertebeden türevi `diff(y,x,2)` şeklinde ifade ediyoruz. Başlangıç koşulu yine bir liste olarak giriliyor.

$$y'' - 3y' + 2y = 1 + x^2, \quad y(0) = 3, \quad y'(0) = 2$$

```
x = var('x')
y = function('y')(x)
cozum = desolve( diff(y,x,2)-3*diff(y,x)+2*y == 1+x^2 ,y, ics=[0,3,2] )
show(cozum)
```

$$\frac{1}{2}x^2 + \frac{3}{2}x - \frac{1}{4}e^{(2x)} + e^x + \frac{9}{4}$$

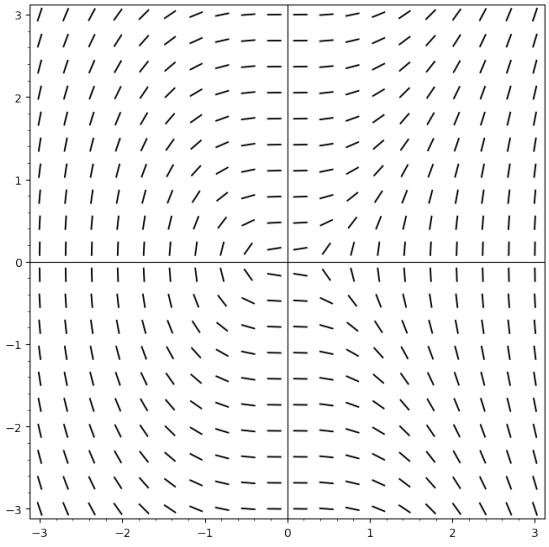
7.17 Eğim Alanı

$\frac{dy}{dx} = f(x, y)$ formundaki bir diferensiyel denklem için eğim alanı `plot_slope_field` komutu ile çizdirilebilir. Örneğin

$$\frac{dy}{dx} = \frac{x^2}{y}$$

denklemi için eğim alanını görelim. İşin içinde eğim varsa farklı boy/en oranları yanıltıcı olabileceğinden `aspect_ratio=1` seçeneğini eklemek uygun olacaktır:

```
var("y")
ea=plot_slope_field( x^2/y , (x,-3,3 ), (y,-3,3), aspect_ratio=1)
show(ea)
```



Alıştırma 7.17.1 *a. Aşağıdaki differensiyel denklem için eğim alanını çizdirin:*

$$\frac{dy}{dx} = -\frac{2x}{y}$$

b. Aşağıdaki başlangıç değer problemini `desolve` *komutunu kullanarak çözün. Elde ettiğiniz çözümün grafiğini çizdirin.*

$$\frac{dy}{dx} = -\frac{2x}{y}, \quad y(1) = -1.$$

c. (a) ve (b) kısımlarında elde ettiğiniz grafikleri aynı koordinat sisteminde çizdirip yorumlayın.

7.18 Diferensiyel Denklem Sistemleri

Aşağıda bir diferensiyel denklem sistemi veriliyor:

$$\frac{dx}{dt} = x - 4y$$
$$\frac{dy}{dt} = -2x + 3y$$

Bu sistemin çözümünü $x(0) = 1, y(0) = 2$ başlangıç koşulu ile SageMath kullanarak bulalım. Burada t bağımsız değişken; x ve y de t'ye bağlı fonksiyonlar. Diferensiyel denklem sistemi çözmek için `desolve_system` komutunu kullanıyoruz. Denklemleri ve başlangıç koşulunu nasıl girdiğimizi inceleyin:

```
t = var("t")
x = function("x")(t)
y = function("y")(t)
de1= diff(x,t)==x-4*y
de2= diff(y,t)==-2*x+3*y
cozum = desolve_system( [ de1 ,de2 ] ,(x,y), ics=[0,1,2] )
show(cozum)
```

$$\left[x\left(t\right) = -e^{(5\,t)} + 2\,e^{(-t)}, y\left(t\right) = e^{(5\,t)} + e^{(-t)}\right]$$

Not 7.18.1 (Vektör Alanı) Bir diferensiyel denklem sisteminin çözümünü vektör alanıyla birlikte görmek isteyebiliriz. Aşağıdaki kodu kullanarak vektör alanını (*vector field*) SageMath ile çizdirebilirsiniz. Daha fazla detay için uygulamalardan 8.13 numaralı alt bölümü inceleyin.

```
var("x y")
plot_vector_field( (x-4*y, -2*x+3*y ), (x,-2,2), (y,-2,2),
    aspect_ratio=1 )
```

7.19 Laplace ve Ters Laplace Dönüşümleri

Son olarak Lapace ve ters Laplace dönüşümlerini SageMath ile nasıl yapacağımızı aşağıdaki birkaç örnekle görelim.

Laplace dönüşümü için `laplace` komutunu kullanıyoruz:

$$\mathscr{L}\left\{e^{-3t}\right\}$$

```
var("t s")
f=exp(-3*t)
f.laplace(t,s)
```

```
1/(s + 3)
```

$$\mathscr{L}\left\{t\sin t\right\}$$

```
var("t s")
f=t*sin(t)
f.laplace(t,s).show()
```

$$\frac{2\,s}{\left(s^2+1\right)^2}$$

Ters Laplace dönüşümü için kullanılan komut ise `inverse_laplace`:

$$\mathscr{L}^{-1}\left\{\frac{2s}{(s^2+1)^2}\right\}$$

```
var("t s")
F=2*s/(s^2 + 1)^2
inverse_laplace(F, s, t)
```

```
t*sin(t)
```

$$\mathscr{L}^{-1}\left\{\frac{1+s+s^2}{1+s^2+s^4}\right\}$$

```
var("t s")
F=(1+s+s^2)/(1+s^2 + s^4)
inverse_laplace(F, s, t).show()
```

$$\frac{2}{3}\sqrt{3}e^{\left(\frac{1}{2}t\right)}\sin\left(\frac{1}{2}\sqrt{3}t\right)$$

Bölüm 8

SageMath ile Uygulamalar

Bu bölümde farklı alanlarda SageMath uygulamaları verilmiştir. Alt bölümler birbirinden bağımsızdır. İlginizi çekmeyen ya da ön bilgi gerektirdiğini düşündüğünüz uygulamaları atlayabilirsiniz. Bu bölümün amacı zor problemler çözmek değil, çeşitli problemleri çözmemize yardımcı olacak araçları basit problemlerle tanıtmaktır.

8.1 Fermat Asalları

n negatif olmayan bir tam sayı olmak üzere,

$$F_n = 2^{2^n} + 1$$

şeklindeki sayıları düşünelim. Pierre de Fermat 1650'de yukarıdaki F_n formülünün sadece asal sayı veren bir formül olduğu sanısını ortaya atmış. Seksen yıldan fazla zaman aksini iddia eden olmamış; ta ki İsviçreli matematikçi Leonhard Euler 1732'de bu formülün $n = 5$ için asal olmadığını ve aşağıdaki gibi çarpanlarına ayrılabileceğini gösterene kadar:

$$2^{2^5} + 1 = 641 \times 6700417$$

Fermat bu sanısını bugün ortaya atsaydı SageMath bir saniyeden daha kısa zamanda Euler'in bulduğunu bulurdu:

```
is_prime(2^2^5+1)
```

```
False
```

```
factor(2^2^5+1)
```

```
641 * 6700417
```

F_n formundaki bir sayıya *Fermat sayısı*, eğer sayı asal ise *Fermat asalı* denir. Günümüzde bilinen sadece 5 tane Fermat asalı var; bunlar da $n = 0, 1, 2, 3, 4$ için elde edilenler. SageMath yardımıyla $n = 0, 1, 2, \ldots, 14, 15$ durumlarına bakalım:

```
[ (i,is_prime(2^2^i+1)) for i in [0..15] ]
```

```
[(0, True),
 (1, True),
 (2, True),
 (3, True),
 (4, True),
 (5, False),
 (6, False),
 (7, False),
 (8, False),
 (9, False),
 (10, False),
 (11, False),
 (12, False),
 (13, False),
 (14, False),
 (15, False)]
```

Alıştırma 8.1.1 *p asal sayı olmak üzere $M_p = 2^p - 1$ formundaki asal sayılara Mersenne asalı denir.*

a. İlk 100 tane p asal sayısı için M_p sayısının Mersenne asalı olup olmadığını bulun.

b. En küçük 15 Mersenne asalını listeleyin.

Not 8.1.1 Bu satırlar yazılıyorken (Eylül 2022), son bulunanı Aralık 2018'de olmak üzere sadece 51 tane Mersenne asalı bilinmekteydi (`https://www.mersenne.org/primes/`).

8.2 Bir Denklem

Aşağıdaki denklemin 1'den büyük bir kökünün olmadığını iddia ediyoruz:

$$x^7 - 2x^5 + 10x^2 - 1 = 0 \tag{8.1}$$

Bunu göstermek için yukarıdaki denklemde $x = y + 1$ dönüşümü yapalım. Yani x gördüğümüz yere $y + 1$ yazalım. Sonra da gerekli genişletmeleri yapıp ifadeyi sadeleştirelim. Tabii ki SageMath ile:

```
(x^7-2*x^5+10*x^2-1).subs(x=y+1).expand()
```

```
y^7 + 7*y^6 + 19*y^5 + 25*y^4 + 15*y^3 + 11*y^2 + 17*y + 8
```

İşlemimiz tamam. Aşağıdaki denklemi elde ettik:

$$y^7 + 7y^6 + 19y^5 + 25y^4 + 15y^3 + 11y^2 + 17y + 8 = 0 \tag{8.2}$$

x ve y arasındaki ilişki $y = x-1$ şeklinde olduğundan dolayı, 8.1 numaralı denklemin 1'den büyük kökünün olmadığını söylemek, 8.2 numaralı denklemin 0'dan büyük (yani pozitif) kökünün olmadığını söylemekle aynı şey. 8.2 denkleminin bütün katsayıları pozitif olduğundan pozitif bir y değeri, 8.2 denkleminin sol tarafını pozitif yapacaktır; yani hiçbir pozitif y değeri denklemi sağlamayacaktır. Böylece 1'den büyük hiçbir x değeri de 8.1 denklemini sağlamayacaktır.

Not 8.2.1 Bu küçük problem Howard Eves'in *Fundamentals of Modern Elementary Geometry* [5] adlı kitabında, dönüşümlerin işleri kolaylaştırdığı örneklerden biri olarak veriliyor.

Alıştırma 8.2.1 *$f : \mathbb{R} \to \mathbb{R}$ bir fonksiyon olmak üzere, $f(x) = 0$ denkleminin kökleri $y = f(x)$ fonksiyonunun grafiğinin x-ekseniyle kesiştiği noktalardır. 8.1 numaralı denklemin 1'den büyük kökünün olmadığı iddiasını grafikle destekleyin. Ayrıca, söz konusu dönüşümün grafiksel karşılığını yorumlayın.*

8.3 Fahrenhayttan Santigrata Çeviri

Sıcaklık birimleri olan *fahrenhayt* (°F) ve *santigrat* (°C) arasında aşağıdaki gibi bir ilişki var:

$$C = \frac{5}{9}(F - 32)$$

Şimdi bizim için bu işlemi yapacak küçük bir SageMath programı yazıp 80 °F'ın kaç derece olduğunu hesaplayalım:

```
F=80
C=5/9*(F-32)
round(C,2)
```

```
26.67
```

Buradaki `round(C,2)` komutunun noktadan sonra 2 basamak kullanarak yuvarlama yaptığını 2.6 numaralı alt bölümden biliyoruz. Farklı bir fahrenhayt değeri için benzer şekilde sonucu bulma şansımız olsa da, bir fonksiyon kullanmak daha kullanışlı olacaktır. Aşağıda `FdenCye` adında bir fonksiyon oluşturuyoruz:

```
def FdenCye(F):
    C=5/9*(F-32)
    print(round(C,2))
```

Fonksiyon tamam; hemen kullanalım:

```
FdenCye(80)
```

```
26.67
```

Şimdi biraz daha ileri gidip etkileşimli bir araç yaratalım. Böylece girdi kutusuna girilen bir fahrenhayt ifadesini hemen santigrat ifadesine çevirmiş olacağız.

```
@interact
def _( F=input_box(label="Fahrenhayt:",default=80,width=15) ):
    C=5/9*(F-32)
    print(round(C,2))
```

Alıştırma 8.3.1 *Santigrattan fahrenhayta çeviri yapan,* `CdenFye` *adlı bir fonksiyon oluşturun ve bu fonksiyonu interaktif bir araç oluşturmak için kullanın.*

Alıştırma 8.3.2 *Kullanıcının buton kullanarak C'de F'ye ve F'den C'ye seçeneklerinden birini seçerek çeviri yapacağı bir interaktif araç oluşturun. 6.6 numaralı alt bölüm buton oluşturmak için yardımcı olabilir.*

Not 8.3.1 Bu alt bölümdekine benzer bir şekilde uzunluk, alan, hacim ve ağırlık gibi birimler için çeviri aracı oluşturulabilirsiniz.

Alıştırma 8.3.3 (Sayısal Loto) *6/49 biçimindeki loto, 49 sayıdan rastgele seçilecek 6 sayıyı tutturmaya dayalı bir oyun. Daha genel olarak, r/n biçimindeki loto da n tane sayıdan seçilecek r tane sayıyı tutturmaya dayalı.*

r tane sayının tamamını tutturma olasılığı $1/C(n,r)$. Sadece k tanesini tutturma olasılığı da

$$\frac{C(r,k)\,C(n-r,r-k)}{C(n,r)}.$$

Girilen n, r ve k doğal sayıları için n sayıdan $r \leq n$ tanesini seçerek sadece $k \leq r$ tanesini tutturma olasılığını nümerik olarak veren bir fonksiyon oluşturun.

8.4 Bir Nümerik Çözüm Örneği: $x^2 = 2^x$

Bu alt bölümde matematik yazılımlarının çok yaygın kullanıldığı ve gerçekten de iyi iş çıkardıkları bir uygulama olan denkem çözümünü basit bir örnek ile inceleyeceğiz. Bilindiği üzere bazı denklemlerin bazı çözümlerini ancak nümerik olarak bulabiliriz. Örneğin

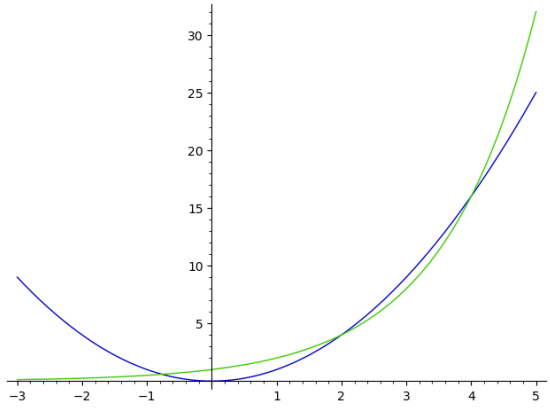
$$x^2 = 2^x$$

böyle bir denkem. Bu denklemin bütün reel sayı çözümlerini bulmaya çalışalım. 2 ve 4 değerleri, verilen denklemi sağlar ve bu değerlerin denklemin birer çözümü oldukları açıktır. Acaba başka bir çözüm var mı? Bunu görmek için denklemin iki tarafındaki

fonksiyonların grafiklerini aynı koordinat sisteminde SageMath'e çizdirip ortak bir x değerinde aynı değeri alıp almadıklarına bakalım. Aslında bunun anlamı grafiklerin kesişip kesişmedikleridir. Her bir kesişim noktasının apsisi, verilen denklem için bir çözümdür.

Aşağıdaki komut ile grafikleri çizdiriyoruz:

```
plot( [x^2 , 2^x] , (x,-3,5) )
```



Yukarıdaki komut SageMath'e $y = x^2$ ve $y = 2^x$ fonksiyonlarının grafiklerini $x = -3$'ten $x = 5$'e kadar olan kısmını çizdiriyor. Üç kesişim noktası görüyoruz. Bunlardan iki tanesi, tahmin ettiğimiz gibi $x = 2$ ve $x = 4$ değerleridir. Üçüncü kesişim noktasından anladığımıza göre bu çözüm negatif bir sayıdır.

4.5 numaralı alt bölümden bildiğimiz üzere kökü bulmak için `find_root` komutunu kullanabiliriz. Bir de kökün, içinde bulunduğu tahmini bir aralık girmemiz gerekiyor. Bu konuda grafikten destek alırsak $(-1, 0)$ veya $(-2, 0)$ aralıklarından biri uygun olacaktır. $(-2, 0)$ aralığını kullanalım:

```
find_root( 2^x == x^2, -2, 0 )
```

```
-0.7666646959621225
```

Böylece $x^2 = 2^x$ denkleminin, ikisi tam sayı olmak üzere üç tane reel sayı çözümü olduğunu görüyoruz. Tam sayı olmayan çözümün de yaklaşık değeri hemen yukarıda.

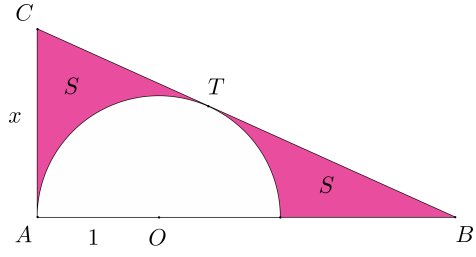
Not 8.4.1 Grafiklerle ilgili renk, çizgi stili, çizgi kalınlığı vb. gibi pek çok seçeneği 3 numaralı bölümde bulabilrsiniz.

Alıştırma 8.4.1 $3^x = x^3$ *denkleminin bütün reel sayı çözümlerinin mümkünse tam sonuçlarını, değilse yaklaşık değerlerini bulun.*

Alıştırma 8.4.2 $x > 0$ *olmak üzere* $e^x = x^e$ *denkleminin* $x = e$ *çözümünden başka bir çözümü var mı? Bu denklemi SageMath ile inceleyin.*

Alıştırma 8.4.3 *$f(x) = \sin x$ fonksiyonunun grafiğini 1 birim sağa kaydırıp $g(x)$ fonksiyonunun grafiğini elde ediyoruz. $y = f(x)$ ve $y = g(x)$ fonksiyonlarının grafiklerinin tüm kesim noktalarını bulun.*

Alıştırma 8.4.4 *Şekilde ABC dik üçgeni ve üçgenin iki kenarına A ve T noktalarında teğet olan O merkezli 1 birim yarıçaplı yarım çember görülmektedir. Taralı bölgelerin alanları eşit olduğuna göre $|AC| = x$ kaçtır?*



Alıştırma 8.4.5 *$x \geq 1$ olmak üzere aşağıdaki ifadenin alabileceği en büyük değeri bulun:*

$$(x-1)^{1/x}$$

Alıştırma 8.4.6 (En Büyük Eğim)

$$y = \frac{\sin x}{x}$$

fonksiyonunun, eğimi en büyük olan teğet doğrusunun denklemini bulun. Verilen fonksiyonun ve söz konusu teğetin grafiğini aynı koordinat düzleminde çizdirin.

8.5 Viviani Eğrisi

Viviani eğrisi, r pozitif bir sayı olmak üzere

$$x^2 + y^2 + z^2 = r^2$$

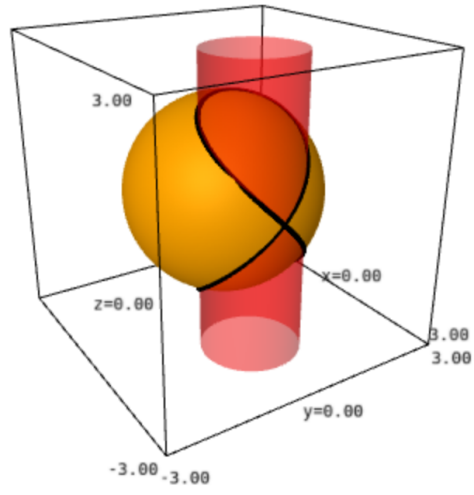
denklemiyle verilen küre yüzeyiyle

$$x^2 + y^2 = rx$$

silindir yüzeyinin kesişimiyle oluşan bir eğridir. Bu alt bölümde söz konusu küre ve silindirle birlikte, onların kesişimleri olan Viviani eğrisini aynı koordinat sisteminde çizdireceğiz. Özel olarak $r = 2$ alalım. Siz farklı bir pozitif değer seçebilirsiniz.

Komutlar aşağıda. Grafiği fare kullanarak çevirip inceleyiniz. Üç boyutlu grafikler için detaylar 5.13 numaralı alt bölümde.

```
var("x y z t")
r=2
kure=implicit_plot3d( x^2+y^2+z^2==r^2, (x,-3,3), (y,-3,3) , (z,-3,3),
    color="orange", opacity=1 , plot_points=150 )
silindir=implicit_plot3d( x^2+y^2==r*x, (x,-3,3), (y,-3,3) , (z,-3,3) ,
    color="red" , opacity=.5 , plot_points=150 )
viviani= parametric_plot3d(( r*cos(t)*cos(t), r*cos(t)*sin(t), r*sin(t)
    ), (t,-pi,pi), color="black", thickness=4, plot_points=1000)
show(kure+silindir+viviani)
```



Yukarıda önce sırasıyla küre ve silindiri `implicit_plot3d` komutuyla çizdiriyoruz. Sonra da Viviani eğrisini bu yüzeylerden bağımsız bir şeklide parametrik olarak `parametric_plot3d` ile oluşturuyoruz. Son olarak da bu üç grafiği aynı koordinat sisteminde gösteriyoruz. Beklediğimiz üzere eğri tam da yüzeylerin kesişiminden geçiyor.

Alıştırma 8.5.1 (Simit Kesitleri) *$R = 2$ ve $r = 1$ olmak üzere, üç boyutlu uzayda*

$$(x^2 + y^2 + z^2 + R^2 - r^2)^2 = 4R^2(x^2 + y^2)$$

simitini (torus) ve $y = 2$ düzlemini ele alalım.

a. *Verilen simit ve düzlemin grafiklerini aynı koordinat sisteminde çizdirin.*

b. *Grafiklerin kesim noktaları bir Bernoulli lemniskatı verecektir. Bu arakesiti de aynı koordinat sisteminde gösterin.*

c. *Farklı **R** ve **r** değerleri için bu durumu grafiklerle inceleyin.*

8.6 Vücut Kitle İndeksi

Quetelet İndeksi ya da daha yaygın deyimiyle *Vücut Kitle İndeksi* (*VKİ*) yetişkin bir kişinin ağırlığının boyuna göre normal olup olmadığını belirlemekte kullanılan bir parametredir. Bunun için doku kütlesi (kas, yağ ve kemik) ve boy temel alınır ve bu parametre vücut ağırlığının (kilogram), boy uzunluğunun (metre) karesine bölünmesi ile hesaplanır. Yani m ağırlık, h boy olmak üzere *VKİ* aşağıdaki gibi tanımlanır:

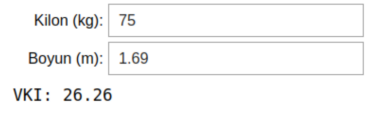
$$VKİ = \frac{m}{h^2}$$

VKİ, insanları kabaca zayıf, normal kilolu, fazla kilolu ve obez kategorilerine ayırmak için evrensel olarak kullanılmaktadır. Yaygın olarak kullanılan aralıklar aşağıdaki tabloda verilmiştir:

Kategori	Aralık
Zayıf	$VKİ < 18.50$
Normal Kilolu	$18.50 \leq VKİ < 25$
Fazla Kilolu	$25 \leq VKİ < 30$
Obez	$VKİ \geq 30$

Kullanıcının, kilosunu ve boyunu girip *VKİ* değerini elde edeceği interaktif bir araç yaratalım.

```
@interact
def _(
    m=input_box(label="Kilon (kg):", default=75),
    h=input_box(label="Boyun (m):", default=round(1.69,2) )
     ):
    print( "VKI:", round(m/h^2,2) )
```



Görüldüğü üzere kilo ve boy bilgileri, `input_box` (girdi kutucuğu) komutu kullanılarak girdi olarak isteniyor ve bu girdiler sırasıyla m ve h değişkenlerine atanıyor. Daha sonra *VKİ* formülününden elde edilen değer `print` komutuyla ekrana yazdırılıyor. Bunu yaparken metin ve değişkenleri virgül ile ayırdığımıza dikkat edin. Daha önce de yaptığımız gibi `round` komutuyla noktadan sonra iki basamak görmek istediğimizi belirttik. Varsayılan değerleri de kilo ve boy için sırasıyla 75 ve 1.69 olarak aldık.

Koşullu Önermeler ile Kategori Bilgisi

Oluşturduğumuz interaktif araç bize *VKİ* için sadece bir sayı değeri veriyor, biz de tablodan bakarak kategoriyi buluyoruz. İstersek bu kodları biraz geliştirip programın ilgili kategoriyi vermesini sağlayabiliriz. Aşağıdaki programı inceleyin:

```
@interact
def _(
    m=input_box(label="Kilon (kg):", default=75),
    h=input_box(label="Boyun (m):", default=round(1.69,2) )
     ):
    vki = m/h^2
    if vki < 18.5:
        kategori="Zayıf"
    if 18.5 <= vki < 25:
        kategori="Normal"
    if 25 <= vki < 30:
        kategori="Fazla Kilolu"
    if vki >= 30:
        kategori="Obez"
    print( "VKI:", round(vki,2) )
    print( "Kategori:", kategori )
```

Yukarıdaki programda bu problem için `if-elif-else` yapısının daha uygun olacağını tahmin etmiş olabilirsiniz. Aşağıdaki program tam da o yapıyı kullanıyor ve tamamıyla aynı işlevi görüyor. Programı çalıştırıp yorumlamayı size bırakıyorum. Koşullu önermeler için ilgili bölüm: 6.3.

```
@interact
def _(
    m=input_box(label="Kilon (kg):", default=75),
    h=input_box(label="Boyun (m):", default=round(1.69,2) )
     ):
    vki = m/h^2
    if vki < 18.5:
        kategori="Zayıf"
    elif vki < 25:
        kategori="Normal"
    elif vki < 30:
        kategori="Fazla Kilolu"
    else:
        kategori="Obez"
    print( "VKI:", round(vki,2) )
    print( "Kategori:", kategori )
```

Kategori Sınırları ve Seviye Eğrileri

Şimdi de *VKİ* foksiyonuna biraz geometrik açıdan bakalım. Örneğin Ağırlık-Boy düzleminde hangi noktalar *VKİ* değerini 18.5 yapar? Bu noktaların kümesi, "zayıf" ile "normal kilo" kategorilerinin sınırını oluşturur. Bu bir eğri, hatta tam olarak $\frac{m}{h^2} = 18.5$ ya da daha açık bir ifadeyle $m = 18.5h^2$ parabolü olacaktır. Benzer şekilde 25 ve 30 değerleri için

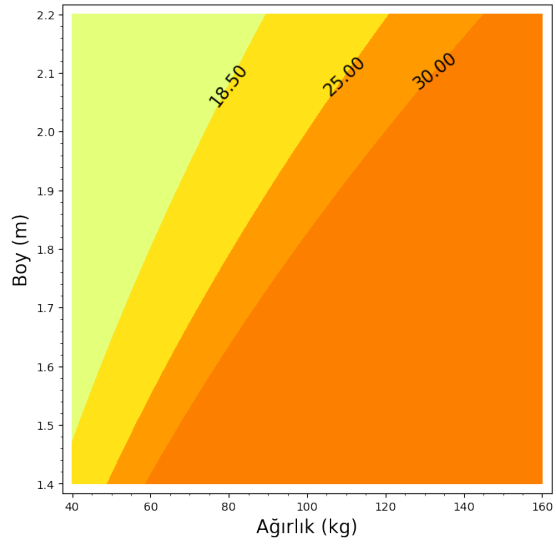
sınırları belirleyen eğriler de bulunabilir ve hepsi aynı düzlemde çizilebilir. Bu şekilde düzlemde üç tane parabol yayı ve dört farklı kategori için birer bölge oluşturabiliriz.

Ancak şu yöntem daha pratik olabilir: $VK\dot{I} = \frac{m}{h^2}$ iki değişkenli bir fonksiyon ve üç boyutlu uzayda bir yüzey ifade eder. Bu yüzeyin de C bir sabit olmak üzere,

$$\frac{m}{h^2} = C$$

biçiminde seviye eğrileri vardır. Örneğin $\frac{m}{h^2} = 18.5$ seviye eğrilerinden biridir. Yani daha önce incelediğimiz `contour_plot` komutuyla çok daha pratik bir şekilde tüm sınırları elde edebiliriz. Aşağıda `contour_plot` komutunu girdikten sonra pek çok seçenek giriyoruz. Bu sebeple kalabalık görünüyor. 5.7 numaralı bölümdeki seçeneklere bakılabilir.

```
var("m h")
contour_plot( m/h^2 , (m,40,160), (h,1.4,2.2), contours=[ 18.5, 25 ,
    30], cmap="Wistia", label_fontsize=15, linestyles="-", linewidths=0,
    figsize=10, label_inline=True, fill=True, aspect_ratio=150,
    axes_labels=[ "Ağırlık (kg)","Boy (m)" ], labels=True,
    label_colors="black" )
```



Yukarıdaki kodlar sınırları veren üç seviye eğrisi ve oluşturdukları bölgeleri çizdiriyor. Kilo ve boy için uygun aralıklar, bu aralıkların anlaşılır sonuç vermesi için oranlamalar (`aspect_ratio` komutu) ve biraz da makyaj ile istediğimiz sonucu elde ediyoruz. Deneyip siz de görün.

Hangi değerler için kontur çizdireceğimizi `contours` komutu ile giriyoruz. Seçeneklerde `True`'ları `False`'a çevirerek keşifler yapın. `cmaps` komutu ile renk grubu belirliyoruz; bu eğlenceli bir komut. Eksen etiketleme `axes_labels` ile yapılıyor. Detaylar 5.7 numaralı alt bölümde.

Son olarak kontur grafiği, *VKİ* ve kategoriyi aynı çıktıda veren bir araç oluşturalım. Yani önceki kodları birleştirelim. Ayrıca girilen *m* ve *h* bilgilerini aynı grafikte bir nokta olarak gösterelim. Aşağıdaki programı çalıştırıp görün.

```
@interact
def _(
    m=input_box(label="Kilon (kg):", default=75),
    h=input_box(label="Boyun (m):", default=round(1.69,2) )
     ):
    vki = m/h^2
    if vki < 18.5:
        kategori="Zayif"
    elif vki < 25:
        kategori="Normal"
    elif vki < 30:
        kategori="Fazla Kilolu"
    else:
        kategori="Obez"
    var("mm hh")
    cp = contour_plot( mm/hh^2 , (mm,40,160),(hh,1.4,2.2), contours=[
        18.5, 25 , 30],
        label_fontsize=15,linestyles="-",linewidths=0,figsize=10,
        cmap="Wistia",
        label_inline=True,fill=True,aspect_ratio=150,colorbar=False,
        axes_labels=[ "Ağırlık (kg)","Boy (m)" ], labels=True,
        label_colors="black" )
    pp=point( (m,h),pointsize=50 , zorder=5, color="black" )
    show(cp+pp)
    print( "VKI:", round(vki,2) )
    print( "Kategori:", kategori )
```

Alıştırma 8.6.1 *Boy ile kıyaslayınca, kilo yetişkin biri için daha müdahale edilebilir bir değişken. Yani uygun aralıkta olmak için boyumuzu değil de kilomuzu değiştirmeyi mantıklı buluruz. Şöyle bir araç oluşturun: Kullanıcı sadece boyunu girsin; boyuna bağlı olarak kilo sınır değerlerini veya aralıklarını çıktı olarak elde etsin.*

Alıştırma 8.6.2 *Yukarıda çizmiş olduğumuz kontur grafik dört farklı bölgeye ayrılmış durumda. Ancak hangi bölgenin hangi gruba ait olduğu grafikte görünmüyor. Bu bölgelere ilgili metin etiketlerini yerleştirin. Detaylar 5.10 numaralı alt bölümde.*

Alıştırma 8.6.3 (Cobb-Douglas Fonksiyonu) *Cobb-Douglas fonksiyonu ekonomide sıkça kullanılan bir üretim fonksiyonudur ve aşağıdaki gibi tanımlanır:*

$$Q = AK^{\alpha}L^{\beta}$$

Burada K ve L girdi faktörleri (genellikle, sırasıyla sermaye ve işgücü), Q ise bu girdilere bağlı olarak elde edilen üretim seviyesidir. α ve β parametreleri üretimin sırasıyla sermaye ve iş gücüne göre elastikliği olarak tanımlanır. A ise sabittir.

Kullanıcının α, β ve A değerlerini girip, K-L koordinat sisteminde bazı seviye eğrilerini görüp üretim seviyesi hakkında bilgi edineceği interaktif bir araç yaratın.

8.7 Brahmagupta Formülü

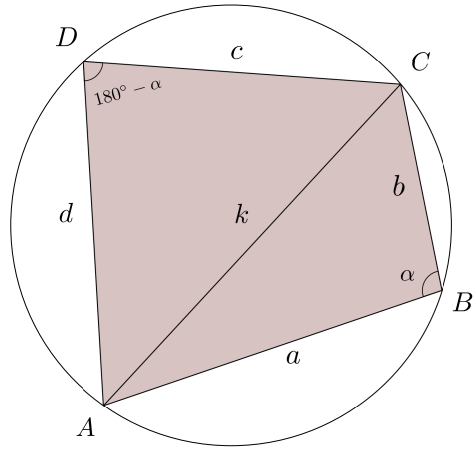
Brahmagupta Formülü kenar uzunlukları verilen bir kirişler dörtgeninin alanını veren formüldür. Diyelim ki kenar uzunlukları a, b, c, d olsun. s sayısı kirişler dörtgeninin çevre uzunluğunun yarısı, yani

$$s = \frac{a+b+c+d}{2}$$

olmak üzere, dörtgenin alanını aşağıdaki formülle bulabiliriz:

$$A = \sqrt{(s-a)(s-b)(s-c)(s-d)}$$

Brahmagupta Formülü'nün çeşitli ispatları var. En popüler olanı trigonometrik ispattır. Bu ispat uzun sadeleştirmeler ve işlemler içerdiğinden, sembolik hesaplamalar için SageMath'in kullanılabileceği güzel bir örnek.



Şekilde kenarları a, b, c, d olan bir kirişler dörtgeni görüyoruz. Amacımız, bu dörtgenin alanını a, b, c, d cinsinden ifade etmek ve bu ifadenin yukarıda verilen A'ya özdeş olduğunu göstermek. Dörtgenin alanına S diyelim. $[AC]$ köşegeninin uzunluğu k, ABC açısı α derece olsun; böylece ADC açısı $(180° - \alpha)$ olacaktır. Dörtgenin alanı ABC ve ADC üçgenlerinin alanlarının toplamıdır. Üçgenlerin alanları için Sinüs Alan Teoremi'ni kullanırsak:

$$\begin{aligned} S &= A(ABC) + A(ADC) \\ &= \frac{1}{2}ab\sin\alpha + \frac{1}{2}cd\sin\left(180^\circ - \alpha\right) \end{aligned}$$

Buradan, aşağıdaki sonuç elde edilir:

$$S = \frac{1}{2}(ab + cd)\sin\alpha \tag{8.3}$$

S'yi bulduk, ama henüz tam olarak kenar uzunluklarıyla ifade edilmiş değil. α açısından kurtulmamız lazım. Bunu Kosinüs Teoremi yardımıyla yapıyoruz. ABC ve ADC üçgenlerinde, B ve D açılarıyla Kosinüs Teoremi'ni kullanıp k'yı elimine edersek:

$$\begin{aligned} a^2 + b^2 - 2ab\cos\alpha &= c^2 + d^2 - 2cd\cos\left(180^\circ - \alpha\right) \\ a^2 + b^2 - c^2 - d^2 &= 2(ab + cd)\cos\alpha \end{aligned}$$

Her iki tarafı 4'e bölelim:

$$\frac{1}{4}\left(a^2 + b^2 - c^2 - d^2\right) = \frac{1}{2}(ab + cd)\cos\alpha \tag{8.4}$$

Şimdi de (8.3) ve (8.4) denklemlerinin her iki tarafının karelerini alarak taraf tarafa toplayıp S^2'yi çekersek aşağıdaki ifadeyi elde ederiz:

$$S^2 = -\frac{1}{16}\left(a^2 + b^2 - c^2 - d^2\right)^2 + \frac{1}{4}(ab + cd)^2 \tag{8.5}$$

Amacımız S'nın A'ya özdeş olduğunu göstermek. İkisi de pozitif sayılar olduğundan S^2'nin A^2'ye eşit olduğunu göstermek yeterli olacaktır.

Formüldeki s ifadesini kenarlar cinsinden yazarsak A^2 aşağıdaki gibi olur:

$$A^2 = -\frac{1}{16}(a + b + c - d)(a + b - c + d)(a - b + c + d)(a - b - c - d) \tag{8.6}$$

Bu iki karmaşık ifadenin, yani (8.5) ve (8.6) denklemlerinin sağ taraflarının birbirine özdeş olduğunu göstermek istiyoruz. Bunu SageMath yapacak. Aşağıdaki kodlarda ifadelerin birine `alan1` diğerine de `alan2` diyoruz. `bool` komutu ile bu iki ifadenin özdeş olup olmadıklarını sorguluyoruz; ve `True` yanıtını alıyoruz:

```
var(" s a b c d ")
alan1=-1/16*(a^2+b^2-c^2-d^2)^2 + 1/4*(a*b+c*d)^2
alan2=((s-a)*(s-b)*(s-c)*(s-d)).subs(s=(a+b+c+d)/2)
bool( alan1 == alan2 )
```

```
True
```

Not 8.7.1 Bu sadeleştirmeyi `bool` komutuna alternatif olarak aşağıdaki şekilde de yapabiliriz:

```
(alan1-alan2).simplify_full()
```

```
0
```

Alıştırma 8.7.1 (Heron Formülü) *Brahmagupta formülünün $d = 0$ için özel durumu üçgenin alanını bulmak için kullanılabilir. Kenar uzunlukları a, b, c birim ve*

$$s = \frac{a+b+c}{2}$$

olmak üzere, üçgenin alanını veren aşağıdaki formüle Heron formülü denir:

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

Üçgenin kenar uzunlukları girildiğinde alanını hesaplayan bir araç oluşturun. Ayrıca, girilen kenar uzunlukları bir üçgen oluşturmuyorsa bu aracın, kullanıcıyı uyarmasını istiyoruz. `if-else` *yapısı ve üçgen eşitsizliği konusunda yardımcı olabilecek detaylar 6.3 numaralı alt bölümde.*

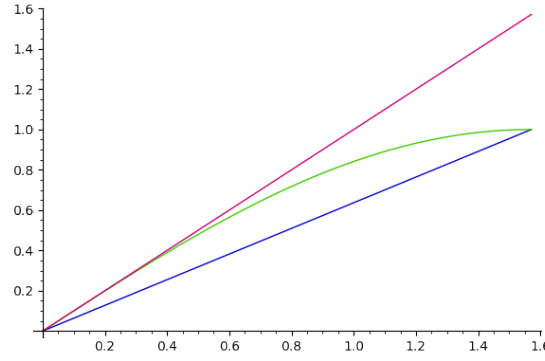
8.8 Jordan Eşitsizliği

$0 \leq x \leq \frac{\pi}{2}$ olmak üzere aşağıdaki eşitsizlik Jordan eşitsizliği olarak bilinir:

$$\frac{2}{\pi}x \leq \sin x \leq x$$

Geometrik bir bakış açısıyla bu eşitsizlik şunu söylüyor: Verilen aralıkta $y = \sin x$ fonksiyonunun grafiği, $y = \frac{2}{\pi}x$ ile $y = x$ doğrularının grafikleri arasında kalır. Bu üç fonksiyonun grafiğini söz konusu aralıkta çizdirelim:

```
plot( [2/pi*x , sin(x) , x ] , (x,0,pi/2) )
```



Gerçekten de $y = \sin x$ fonksiyonun grafiği eşitsizlikte verilen iki doğru arasına sıkışmış durumda. Ayrıca, grafikteki kesişim noktalarından anladığımıza göre x değişkeninin tanımlandığı aralığın uç değerlerinde eşitlik durumları oluşuyor: $x = 0$ olduğunda üç fonksiyon birbirlerine eşit ve sıfır oluyorlar; $x = \frac{\pi}{2}$ durumunda da $y = \sin x$ ve $y = \frac{2}{\pi}x$ fonksiyonları aynı anda 1 değerine, böylece birbirlerine eşit oluyorlar.

Bu resim ikna edici olsa da matematiksel bir ispat olmaktan çok uzak diye düşünebilirsiniz. Bunun felsefesini burada yapmayalım ama bu konuda biraz kafa yormak isteyenlere James Robert Brown'ın *Proofs and Pictures* (İspatlar ve Resimler) [3] adlı makalesini önerebilirim.

Alıştırma 8.8.1 (Kober Eşitsizliği) $0 \leq x \leq \frac{\pi}{2}$ *olmak üzere*

$$\cos x \geq 1 - \frac{2}{\pi}x$$

eşitsizliğini görselleştirin.

Not 8.8.1 Kober eşitsizliği Jordan eşitsizliğinin özel bir hali. Bunu $x \longmapsto \frac{\pi}{2} - x$ dönüşümüyle görebilirsiniz.

Alıştırma 8.8.2 (Bernoulli Eşitsizliği) $r > 1$ *ve* $x > -1$ *olmak üzere, aşağıda Bernoulli eşitsizliğinin bir varyantı var:*

$$(1 + x)^r \geq 1 + rx$$

Bu eşitsizlik için etkileşimli bir araç yaratın ve böylece pek çok r *reel sayısı için ikna edici grafikler çizdirin.*

Alıştırma 8.8.3 *Her* $x > 0$ *reel sayısı için* $x^x \geq x$. *Bu eşitsizliği görselleştirin.*

8.9 Türevin Geometrik Anlamı: Teğetin Eğimi

Bu alt bölümde, verilen bir f fonksiyonu ve x_0 değeri için teğet doğrusunun denklemini ve grafiğini veren etkileşimli bir araç yaratacağız. Bunun için türevin geometrik anlamını hatırlayalım: x_0 noktasında türevlenebilir bir $y = f(x)$ fonksiyonu için $f'(x_0) = m$ demek, $y = f(x)$ fonksiyonunun grafiğine $x = x_0$ değerinde çizilen teğet doğrusunun eğimi m demektir.

Aşağıda bir program ve onun çıktısını göreceksiniz. Sonrasında da bazı komutlara dair yorumlar var. Etkileşimli araç için detayları 6.6 numaralı alt bölümde bulabilirsiniz.

```
x, y = var("x y")
@interact
def _(f=input_box(label="$f(x)$",default= 1/(1+x^2) - x^3 ,width=50 ),
     x0=input_box(label="$x_0$",default= 1/2,width=30 ),
     x_aralik=input_box(label="$x$ aralığı",default=(-1,2) , width=30),
     y_aralik=input_box(label="$y$ aralığı",default=(-1/2,1) ,width=30),
     fs=slider( [1..10] ,label="Grafik Boyutu",default= 5 ),
     axes_check=("Eksenleri Göster", True)
```

```
 ):
a=x_aralik[0]
b=x_aralik[1]
c=y_aralik[0]
d=y_aralik[1]
D=diff(f,x)
egim=D.subs(x=x0)
y0=f(x=x0)
teget=plot(egim*(x-x0)+y0,(x,a,b),ymin=c,ymax=d,color="grey")
plo=plot(f,(x,a,b), ymin=c,ymax=d,color="brown" ,
    aspect_ratio=1,figsize=fs )
show(plo+teget, frame=not axes_check,
    axes=axes_check,axes_labels=["$x$","$y$"] )
show( LatexExpr(r"\text{Fonksiyon: } y="),f)
show( LatexExpr(r"(x_0,y_0)= "), (x0,y0))
show( LatexExpr(r"\text{Teğet Doğru: } y="),(egim*x-egim*x0+y0) )
```

$f(x)$ -x^3 + 1/(x^2 + 1)

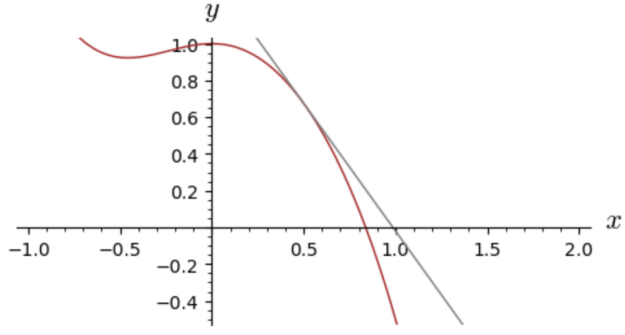
x_0 1/2

x aralığı (-1, 2)

y aralığı (-1/2, 1)

Grafik Boyutu 5

Eksenleri Göster



Fonksiyon: $y = -x^3 + \frac{1}{x^2+1}$

$(x_0, y_0) = \left(\frac{1}{2}, \frac{27}{40}\right)$

Teğet Doğru: $y = -\frac{139}{100}x + \frac{137}{100}$

Yukarıdaki programa dair bazı açıklamalar:

1. f fonksiyonu, x_0 noktası ve çerçeveyi oluşturacak x ve y aralıkları `input_box` komutu ile kullanıcı tarafından girilecek şekilde ayarlanıyor.

2. Kullanıcının, grafik boyutunu kaydırgaç (*slider*) ile ayarlamasını istiyoruz.

3. Onay kutusu (*checkbox*), eksenler ve çerçeveyi gösterip gizlemek için kullanılacak.

4. Girilen x aralığı (a, b), y aralığı da (c, d) olacak şekilde değer atamalarını yapıyoruz.

5. f fonksiyonunun x değişkenine göre türevine D diyoruz ve hemen sonra da D ifadesinde $x = x_0$ uyguluyoruz. Bu şekilde $f'(x_0)$ değerini, yani eğimi buluyoruz ve bunu `egim` adlı değişkene atıyoruz.

6. Sonra da `egim`, `x0` ve `y0` değerlerini kullanarak adı `teget` olan teğet doğru denklemini oluşturuyoruz.

7. Grafikleri, girilen `a,b,c,d` sayılarıyla sınırlanmış şekilde çizdiriyoruz.

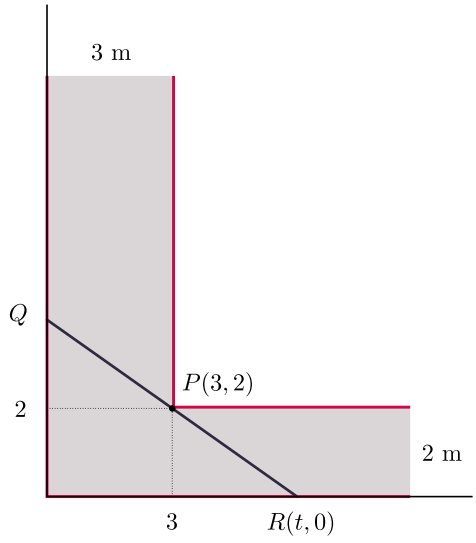
8. LaTeX ifadesi kullanma imkanı veren `LatexExpr` komutuyla gerekli bilgileri ekranda gösteriyoruz.

Diğer komut ve seçeneklerin yorumu okuyucuya bırakılmıştır.

Alıştırma 8.9.1 *Kullanıcının $y = f(x)$ fonksiyonunu girince kaydırgaç kullanarak en fazla 2 birim sağa ve 2 birim sola kaydıracağı bir grafik elde etmesini istiyoruz. Böyle bir interaktif araç oluşturun.*

Alıştırma 8.9.2 *Girilen bir f fonksiyonu için f^{-1} fonksiyonunu (f fonksiyonunun tersi) aynı koordinat sisteminde çizdiren bir interaktif araç oluşturun. Ters fonksiyonu gösterip gizlemek için onay kutusu (checkbox) kullanın. Eksen aralıkları ve grafik boyutu da kullanıcı tarafından değiştirilebilir olsun.*

8.10 Dik Koridor Problemi



Şekilde yukarıdan görüntüsü verilen dik köşeli koridordan, çelik düz bir boru (borunun kalınlığını ihmal ediyoruz) yere paralel tutulacak şekilde geçirilecektir. Koridor genişlikleri 3 metre ve 2 metre olduğuna göre, borunun boyu en fazla hangi uzunlukta olabilir?

Bu bölümde yukarıdaki probleme SageMath yardımıyla cevap vereceğiz. Sezgisel olarak, aranan en uzun borunun duvarlara P, Q ve R noktalarında değeceği açıktır. Aslında $P(3,2)$ noktasından geçip x eksenini $t > 3$ noktasında kesen doğrulardan QR doğru parçasını en kısa yapan durumla ilgileniyoruz. Benzer üçgenlerden yararlanarak veya doğru denklemini kullanarak $Q = \left(0, \frac{2t}{t-3}\right)$ olduğunu bulabilirsiniz. Böylece $|QR|$ uzunluğu t değişkenine bağlı aşağıdaki $L(t)$ fonksiyonu olacaktır:

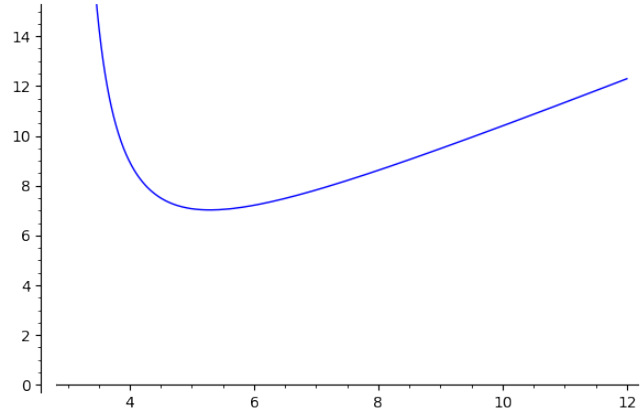
$$|QR| = L(t) = \sqrt{t^2 + \left(\frac{2t}{t-3}\right)^2}$$

Yukarıdaki resimden ya da denklemden görüleceği gibi, büyük t değerleri ve 3'e yakın t değerleri $L(t)$ uzunluğunu büyük yapacaktır. Biz $L(t)$ uzunluğunu en kısa yapan t değeri ile ilgileniyoruz.

Not 8.10.1 Karekök fonksiyonu artan bir fonksiyon olduğundan karekökün içini minimum yapan t değeri, kendisini de yani $|QR|$ değerini de minimum yapar. Böylece verilen L fonksiyonunda karekökün içini en küçük yapan t değerini bulup daha sonra $L(t)$ değerini bulabilirsiniz. Ama biz kareköklü şekliyle işlemlerimizi yapacağız.

Şimdi $L(t)$ fonksiyonunun grafiğini çizip uzunluğun t değişkenine göre nasıl değiştiğini görelim:

```
var("t")
rr=t
qq=2*t/(t-3)
L(t)=sqrt(rr^2+qq^2)
plot( L , (t,3,12) ,ymax=15, ymin=0 )
```



Grafikten gözlemlediğimize göre $L(t)$ fonksiyonunun ilgili aralıkta bir tane minimum değeri var. Şimdi de bu değeri bulmak için fonksiyonun türevinin sıfır olduğu t değerini bulmaya çalışalım:

```
cozum=solve(diff(L(t),t)==0,t)
show(cozum)
```

$$\left[t = -\frac{1}{2}\cdot 12^{\frac{1}{3}}\left(i\sqrt{3}+1\right)+3, t = -\frac{1}{2}\cdot 12^{\frac{1}{3}}\left(-i\sqrt{3}+1\right)+3, t = 12^{\frac{1}{3}}+3, t = 0\right]$$

Çözüm kümesinden anladığımıza göre iki tane reel kök var ve bunlardan $t = 0$ durumu $t > 3$ koşulunu sağlamadığından uygun değil. Bu durumda istediğimiz çözüm sondan ikinci çözüm, yani:

```
show(cozum[-2])
```

$$t = 12^{\frac{1}{3}}+3$$

Bu değeri L fonksiyonuna uygulayalım:

```
L(t).subs(cozum[-2]).show()
```

$$\sqrt{\frac{1}{3}\cdot 12^{\frac{1}{3}}\left(12^{\frac{1}{3}}+3\right)^2+\left(12^{\frac{1}{3}}+3\right)^2}$$

Son olarak da nümerik yaklaşık sonucu `n()` komutu ile bulalım:

```
L(t).subs(cozum[-2]).n()
```

```
7.02348237921997
```

Not 8.10.2 Eğer P noktasını (a, b) gibi genel bir nokta olarak alacak olursak problemi sembolik hesaplamalarla çözebiliriz. Bu durumda en küçük değerini istediğimiz fonksiyon

$$L(t) = \sqrt{t^2 + \left(\frac{tb}{t-a}\right)^2}$$

olacaktır.

Alıştırma 8.10.1 (Genel Durum) *Verilen problemi genel $P(a, b)$ durumu için çözün ve sonucun*

$$\left(a^{\frac{2}{3}} + b^{\frac{2}{3}}\right)^{\frac{3}{2}}$$

ifadesine eşit olduğunu sembolik hesaplamalarla gösterin.

Alıştırma 8.10.2 (İki Gemi) *Koordinat düzleminde biri $A(200, 0)$ noktasından kuzeye 50 km/saat, diğeri $B(0, 100)$ noktasından kuzeydoğuya 70 km/saat hızla aynı anda harekete başlayan iki gemi düşünün. Gemilerin birbirlerine en yakın olduğu an ne zaman olur? O anda aralarındaki mesafe kaç km'dir? (Bu soruya cevap vermek için mesafenin zamana bağlı fonksiyonunu ve bu fonksiyonun alacağı en küçük değer bulunmalıdır. Grafik ile sonuçları destekleyin.)*

8.11 Sophie Germain Asalları

p bir asal sayı olmak üzere, eğer $2p + 1$ sayısı da asal ise p asalına *Sophie Germain asalı* denir (kısaca SGA diyelim). Örneğin 5 sayısı SGA'dır, çünkü 5'in kendisi asal olduğu gibi $2 \times 5 + 1 = 11$ sayısı da asaldır.

Bu alt bölümde SGA ile ilgili aşağıdaki üç uygulamayı yapacağız:

1. Bir sayının SGA olup olmadığını veren bir program.
2. İlk 100 asal sayıdan SGA olanları listeleyen bir program.
3. İlk 100 tane SGA'yı listeleyen program.

Alıştırma yapmak isterseniz bu üç uygulamayı önce kendiniz yapmayı deneyebilirsiniz. Bu uygulamalarda kullanacağımız temel komutlar sırasıyla `if`, `for` ve `while` komutlarıdır.

SGA mı, Değil mi?

Bu uygulama için `is_sga` diye bir fonksiyon tanımlayalım:

```
def is_sga(p):
       if is_prime(p)==True and is_prime(2*p+1)==True:
           print(True)
       else:
           print(False)
is_sga(5)
```

```
True
```

Yukarıdaki basit algoritma şunu yapıyor: Eğer p ve $2p+1$ sayıları asalsa ekrana `True` yaz, değilse `False` yaz.

İlk n Tane Asaldan SGA Olanlar

Aşağıdaki program da ilk n tane p asalı için $2p+1$ de asalsa ekrana p asalını yazdırıyor. Yani p asalının SGA olduğu durumda onu ekrana yazdırıyor.

```
def sga(a):
    for i in range(a):
        p=Primes()[i]
        if is_prime(2*p+1)==True:
            print(p, end=" ")
sga(100)
```

```
2 3 5 11 23 29 41 53 83 89 113 131 173 179 191 233 239 251 281 293 359
    419 431 443 491 509
```

İlk n Tane SGA

Şimdi de ilk n tane SGA'yı veren bir program yazalım. Tanımladığımız foksiyonun adını öncekilerden farklı olsun diye `SGA` diye adlandıralım.

```
def SGA(m):
    sga_liste=[]
    k=0
    while len(sga_liste)<m:
        asal=Primes()[k]
        if is_prime(2*asal+1)==True:
            sga_liste.append(asal)
        k=k+1
    print(sga_liste)
```

Yukarıdaki programı şöyle yorumlayabiliriz: `sga_liste` adında boş bir listeyle işe başlıyoruz. Bu listeye SGA testinden geçen sayıları ekleyeceğiz. Aslında $2p+1$ testinden

geçen asalları demek daha doğru. Bu `sga_liste` listesinin eleman sayısı m'den küçük olduğu sürece bir sonraki asal sayıyı testten geçireceğiz. Bir sonrakine geçiş için k sayacını 1 arttırıyoruz. `while` komutu ile ilgili detayları 6.5 numaralı alt bölümde bulabilirsiniz. Programı daha detaylı incelemek okuyucuya kalsın.

```
SGA(100)
```

```
[2, 3, 5, 11, 23, 29, 41, 53, 83, 89, 113, 131, 173, 179, 191, 233, 239,
    251, 281, 293, 359, 419, 431, 443, 491, 509, 593, 641, 653, 659,
    683, 719, 743, 761, 809, 911, 953, 1013, 1019, 1031, 1049, 1103,
    1223, 1229, 1289, 1409, 1439, 1451, 1481, 1499, 1511, 1559, 1583,
    1601, 1733, 1811, 1889, 1901, 1931, 1973, 2003, 2039, 2063, 2069,
    2129, 2141, 2273, 2339, 2351, 2393, 2399, 2459, 2543, 2549, 2693,
    2699, 2741, 2753, 2819, 2903, 2939, 2963, 2969, 3023, 3299, 3329,
    3359, 3389, 3413, 3449, 3491, 3539, 3593, 3623, 3761, 3779, 3803,
    3821, 3851, 3863]
```

Alıştırma 8.11.1 (Goldbach Sanısı) *Sayılar teorisindeki en popüler çözülmemiş problemlerden biri olan Goldbach sanısı diyor ki: 2'den büyük her çift sayı iki tane asal sayının toplamı şeklinde yazılabilir. Örneğin 28 çift sayısı 5 ile 23 asallarının toplamı olduğundan bu iddiayı sağlar.*

Girilen bir çift sayı için toplamları bu çift sayı olan iki asal sayı bulmaya çalışan bir program yazın.

Alıştırma 8.11.2 (Bertrand Postulatı) *Bertrand postulatının ifadesi şöyle: 1'den büyük her n doğal sayısı için n ile* $2n$ *arasında bir asal sayı vardır. 1845'te Joseph Bertrand tarafından ortaya atılan bu postulat, Chebyshev tarafından 1852 yılında ispatlandı. Böylece bu önermeye bir teorem diyebiliriz.*

Kullanıcının girdiği bir n değeri için n ile $2n$ *arasında bir asal sayı veren bir fonksiyon ve bu foksiyonu kullanarak bir interaktif araç oluşturun.*

8.12 Cassini Eğrileri

Elipsin geometrik yer ile ifade edilen tanımını hatırlayalım: Düzlemde verilen iki noktaya uzaklıkları toplamı sabit olan noktaların geometrik yerine elips denir. Bu tanımda "toplamı" yerine "çarpımı" yazarsak ne olur? Bu problem 1680 yılında Giovanni Domenico Cassini adında İtalyan bir astronom matematikçi tarafından incelenmiş ve oluşan eğriler de ismini bu matematikçiden almış.

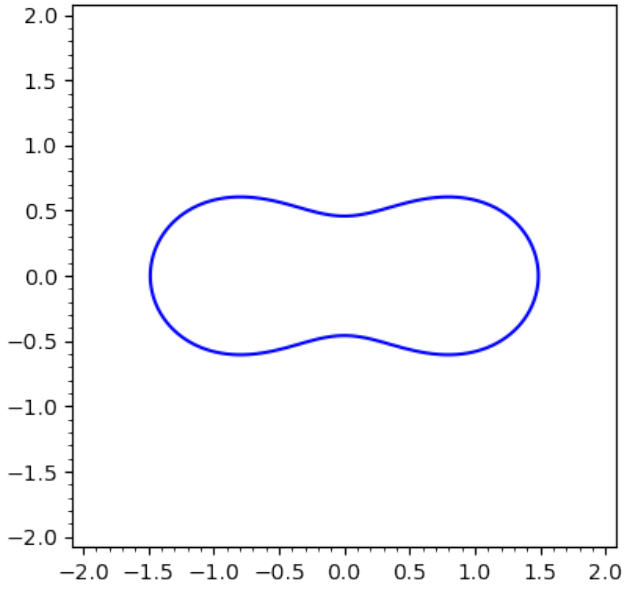
Söz konusu noktalar P_1 ve P_2, $b > 0$ bir sabit sayı olmak üzere $|PP_1| \times |PP_2| = b^2$ koşulunu sağlayan P noktalarının kümesi bir *Cassini eğrisi* olacaktır. P_1 ve P_2 noktalarına eğrinin *odak*ları denir.

Odak noktalarını koordinat düzleminde $P_1(-a, 0)$ ve $P_2(a, 0)$ olarak alıp uzaklık formülünü kullanacak olursak, eğrinin denklemi şöyle olur:

$$\left((x-a)^2 + y^2\right)\left((x+a)^2 + y^2\right) = b^4$$

Burada $e = \frac{b}{a}$ oranı sabit kalacak şekilde a ve b sayılarını değiştirirsek, benzer (yani, biri diğerinin düzgün büyütülmüş ya da küçültülmüş hali) Cassini eğrileri buluruz. Bu e oranı eğrinin karakterini belirler. $e < 1$, $e = 1$ ve $e > 1$ durumları farklı karakterde grafikler verir. Aşağıdaki kodla $e = 1.1$ durumunu çizdiriyoruz:

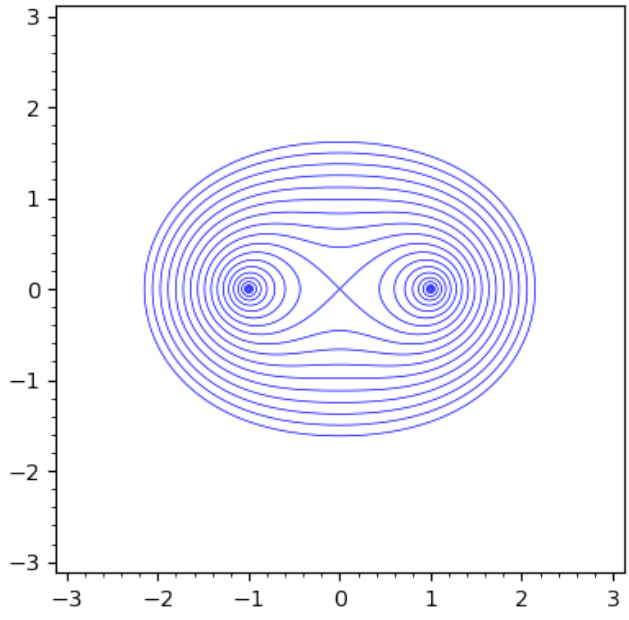
```
var("y b")
a=1
b=1.1
implicit_plot( ((x-a)^2+y^2)*((x+a)^2+y^2)==b^4 , (x,-2,2), (y,-2,2) )
```



Şimdi de sadece b değerini değiştirip $e = 0.1$'den $e = 2$'ye kadar 0.1'lik adımlarla pek çok Cassini eğrisi çizdirelim. Aşağıda bunun için yazılmış, aynı işlevi gören farklı iki kod var. İlk programda farklı b değerlerine karşılık gelen grafikler += komutuyla biriktiriliyorken, ikinci programda bu işlem sum komutuyla yapılıyor:

```
resim=plot([])
for b in srange(.1,2,.1):
    resim += implicit_plot( ((x-1)^2+y^2)*((x+1)^2+y^2)==b^4, (x,-3,3),
        (y,-3,3) , linewidth=.5 , plot_points=400 )
show(resim)
```

```
var("y b")
sum( implicit_plot( ((x-1)^2+y^2)*((x+1)^2+y^2)==b^4 , (x,-3,3),
    (y,-3,3), linewidth=.5 , plot_points=400 ) for b in srange(0.1,2,.1)
    )
```



Not 8.12.1 $e < 1$ durumu iki tane kapalı simetrik eğri veren durumdur. $e > 1$ olduğu durum ise bir tane kapalı eğri verir ve bu kapalı eğri $e \geq \sqrt{2}$ durumunda dışbükeydir.

Not 8.12.2 (Bernoulli'nin Lemniskatı) $e = 1$ olduğu durum yatay 8 şeklindeki durumdur ve bu eğriye *Bernoulli'nin Lemniskatı* denir.

Alıştırma 8.12.1 (İnteraktif Araç) *Cassini eğrilerini* $0.1 \leq e \leq 2$ *aralığında 0.05'lik adımlarla görmemizi sağlayacak kaydırgaçlı interaktif bir araç oluşturun.*

Alıştırma 8.12.2 (Animasyon) *Cassini eğrilerini* $0.1 \leq e \leq 2$ *aralığında 0.05'lik adımlarla görmemizi sağlayacak bir animasyon oluşturun.*

8.13 Bir Aşk Hikayesi

Bu bölümde Steven Strogatz'ın 1988'de Harvard Üniversitesi tarafından yayımlanan *Mathematics Magazine* adlı dergideki makalesini [9] biraz yerelleştirip Leyla ile Mecnun'un aşk hikayesini modelleyen basit bir diferensiyel denklem sistemi oluşturacağız. Şöyle sağlıksız bir ilişkileri olduğunu varsayalım: Mecnun Leyla'nın sevgi ya da nefretine benzer tepkiyi veriyorken, Leyla Mecnun'un sevgi ya da nefretine karşıt bir tepki veriyor. Örneğin Leyla Mecnun'u sevdikçe Mecnun da onu seviyor, Leyla ilgisizken o da ilgisizleşiyor. Diğer taraftan, Mecnun Leyla'yı sevdikçe Leyla'nın ilgisi azalmaya başlıyor, Mecnun ilgisini kaybedince de Leyla'nın ilgisi artıyor.

L fonksiyonu Leyla'nın Mecnun'a olan sevgi (pozitif değer) ya da nefret (negatif değer) duygusunu, M fonksiyonu da Mecnun'un Leyla'ya olan sevgi ya da nefret duygusunu ifade etsin. Aşağıdaki diferensiyel denklem sistemi Leyla ile Mecnun ilişkisini modelleyebilir:

$$\begin{aligned} \frac{dL}{dt} &= -M, \\ \frac{dM}{dt} &= L \end{aligned} \tag{8.7}$$

Örneğin Mecnun Leyla'yı seviyorken, yani M pozitif iken, ilk denklemde L'nin türevinin negatif olacağını, buradan da Leyla'nın sevgisinin azalacağını görebiliriz. Benzer gözlemleri size bırakalım ve birbirlerine olan sevgileri 1'er birim iken nasıl bir hayat onları bekliyor görelim. Yani sistemi $L(0) = 1$, $M(0) = 1$ başlangıç koşuluyla çözelim. Önce diferensiyel denklem sistemini SageMath'e tanıtalım. İlgili bazı detaylar için 7.15 ve 7.18 numaralı alt bölümleri inceleyebilirsiniz.

```
var("t")
L = function("L")(t)
M = function("M")(t)
difdenk1= diff(L,t)==-M
difdenk2= diff(M,t)==L
```

Denklem sistemini tanıttık. Şimdi de `desolve_system` komutuyla sistemi çözelim. $L(0) = 1$, $M(0) = 1$ başlangıç koşulunu $[0, 1, 1]$ olarak yazdığımıza dikkat edin.

```
difdenkcozum = desolve_system( [difdenk1,difdenk2], [L,M], [0,1,1] )
show(difdenkcozum)
```

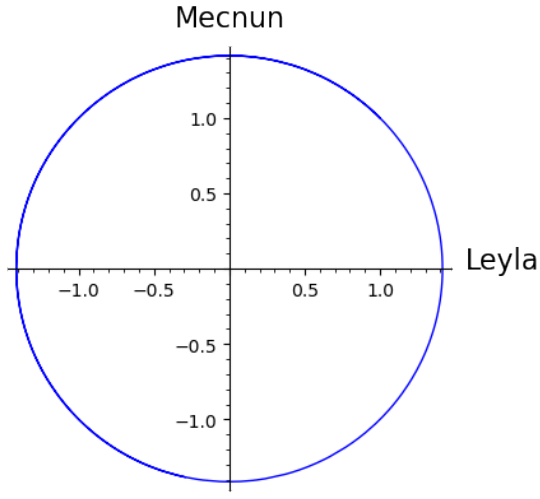
$[L(t) = \cos(t) - \sin(t), M(t) = \cos(t) + \sin(t)]$

Böylece (8.7) diferensiyel denklem sisteminin çözümünü parametrik olarak elde etmiş olduk.

Çözümün Grafiği ve Yorumu

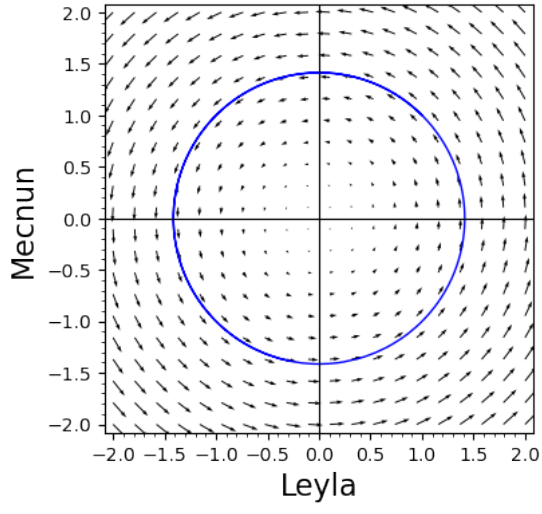
Yukarıda elde ettiğimiz çözümün grafiği aşağıdaki parametrik eğri (bu problem için çember) olacaktır:

```
var("t")
parametric_plot( [difdenkcozum[0].rhs() , difdenkcozum[1].rhs() ] ,
    (t,0,10), axes_labels=["Leyla","Mecnun"], aspect_ratio=1, ).show()
```



Bu başlangıç değer problemini $L(0) = 1, M(0) = 1$ için çözdük. Yani Leyla-Mecnun koordinat sisteminde, $(1, 1)$ noktası ile başlarsak neler olacağının grafiğini görüyoruz. Daha açıklayıcı olması açısından vektör alanını da grafiğe ekleyelim ve durumu yorumlayalım:

```
var("t L M")
p1=parametric_plot( [difdenkcozum[0].rhs() , difdenkcozum[1].rhs() ] ,
    (t,0,10), axes_labels=["Leyla","Mecnun"], aspect_ratio=1, )
p2=plot_vector_field( (-M,L), (L,-2,2), (M,-2,2), aspect_ratio=1 )
show(p1+p2)
```



Birbirlerine olan sevgileri 1'er birim olarak başlarsa, bu pozitif değere karşılık Mecnun'un sevgisi artacakken, Leyla'nınki azalacaktır. Yani grafikte (1, 1) noktasından başlayıp saat yönünün tersine bir hareket olacaktır. L negatif olduğunda Mecnun'un sevgisi de azalmaya başlayacak ve bir süre sonra (M negatif olduğunda) Leyla'nın sevgisi artacaktır. Buna karşılık Mecnun'un sevgisi tekrar artacak, Leyla'nın tekrar azalacak ve bu döngü bir ömür sürüp gidecektir.

Alıştırma 8.13.1 *(8.7) diferensiyel denklem sisteminin çözümü olarak elde edilen, zamana bağlı $L(t)$ ve $M(t)$ fonksiyonlarını aynı koordinat sisteminde çizdirip yorumlayın. Bu konuda 3.11 numaralı alt bölüm yardımcı olabilir.*

Alıştırma 8.13.2 *Aşağıdaki diferensiyel denklem sistemi veriliyor:*

$$\frac{dx}{dt} = -\frac{x}{2} + \frac{y}{3},$$
$$\frac{dy}{dt} = \frac{x}{3} - \frac{y}{3}$$

$x(0) = 1$, $y(0) = \frac{1}{2}$ başlangıç koşuluyla başlangıç değer problemini çözün ve çözümün grafiğini sistemin vektör alanıyla birlikte çizdirip çözümü yorumlayın.

8.14 Lojistik Fonksiyon

Aşağıdaki özyineli (recursive/rekürsif) fonksiyonu düşünelim:

$$x_{n+1} = f(x_n)$$

Bir x_0 değeri verilmiş olsun. Bu durumda:

$$\begin{aligned} x_1 &= f(x_0), \\ x_2 &= f(x_1) = f(f(x_0)) = f^2(x_0), \\ x_3 &= f(x_2) = f(f^2(x_0)) = f^3(x_0) \\ &\vdots \\ x_k &= f^k(x_0) \end{aligned}$$

olur. Bu bileşke işlemleri sonucunda *yörünge* dediğimiz

$$\{x_0, x_1, x_2, x_3, \dots, x_k, \dots\}$$

dizisini elde ederiz.

Bu alt bölümün amacı Lojistik fonksiyon denilen meşhur

$$x_{n+1} = r x_n(1 - x_n)$$

özyineli fonksiyonunun farklı r parametre değerleri için davranışını, yani ne tip yörüngeler verdiğini hem nümerik olarak incelemek hem de daha anlaşılır olması açısından zaman serisi ile görselleştirmektir.

$$f(x) = rx(1 - x)$$

alıp örneğin $r = 3.1$ ve $x_0 = 0.5$ değerleri için x_1, x_2, x_3,...,x_{10} değerlerini bulalım. Tabii ki bu sıkıcı işlemi bir `for` döngüsü ile yapacağız:

```
r=3.1
x0=.5
f=r*x*(1-x)
L=[x0]
for i in range(10):
    L.append( f(x=L[-1]) )
show(L)
```

```
[0.500000000000000, 0.775000000000000, 0.540562500000000,
    0.769899519140625, 0.549178173659744, 0.767502672430025,
    0.553171192752663, 0.766235755209903, 0.555267420208218,
    0.765531088016937, 0.556429048019278]
```

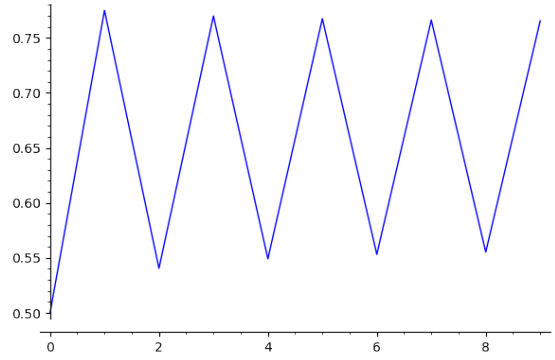
Elde ettiğimiz x_k değerlerini k değerleriyle birlikte aynı tabloda gösterelim:

```
T=[ (i,L[i] ) for i in range(10) ]
table(T ,header_row = ["$k$", "$x_k$"] )
```

k	x_k
0	0.500000000000000
1	0.775000000000000
2	0.540562500000000
3	0.769899519140625
4	0.549178173659744
5	0.767502672430025
6	0.553171192752663
7	0.766235755209903
8	0.555267420208218
9	0.765531088016937

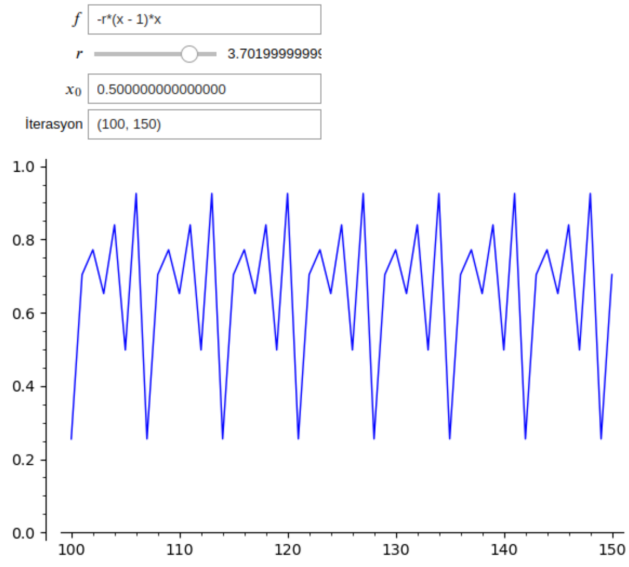
Tabloda verilen bilgileri biraz daha somutlaştıralım ve liste grafiği olarak gösterelim:

```
list_plot(L, plotjoined=True)
```



Elde ettiğimiz grafik $r = 3.1$ değeri için. Şimdi de $0 \leq r \leq 4$ olmak üzere, pek çok r değeri için denklemin davranışını gözlemleyebileceğimiz bir interaktif araç yaratalım:

```
var("r f x0 L")
@interact
def _( f=input_box( label="$f$", default=r*x*(1-x) ),
     param=slider( srange(2.7,4,.001) ,label="$r$",default= .5 ),
     x0=input_box( label="$x_0$",default=.5 ),
     iterasyon=input_box( label="iterasyon", default=(0,10) )
):
    f=f(r=param)
    L=[x0]
    for i in [1..iterasyon[1] ]:
        L.append( f(x=L[-1]) )
    gosterL=[ (k,L[k]) for k in [iterasyon[0]..iterasyon[1] ] ]
    graf=list_plot(gosterL, plotjoined=True ,ymax=1,ymin=0 )
    show(graf)
```



Yukarıdaki programda pek yeni bir şey yok aslında. Kulanıcının *r* parametresi olarak girdiği değeri `param` değişkenine atıyoruz ve bu değeri de `f=f(r=param)` satırıyla *f* fonksiyonuna uyguluyoruz.

Bu interaktif aracı kullanarak farklı *r* değerleri için Lojistik denklemin ne kadar farklı davranışlar sergilediğini gözlemleyebilirsiniz. Buna kaotik davranış da dahildir.

Alıştırma 8.14.1 (Collatz Sanısı) *1'den büyük pozitif bir tam sayı seçin. Sayı çift ise 2'ye bölün, tek ise 3 ile çarpıp 1 ekleyin. Elde ettiğiniz yeni sayıya aynı şeyi uygulayın. Çift ise 2'ye bölün, tek ise 3 ile çarpıp 1 ekleyin. Mesela 34 ile başlayalım ve bu kuralı uygulayalım:*

$$34 \to 17 \to 52 \to 26 \to 13 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$$

Collatz sanısı diyor ki, bu işlemi yeteri kadar sayıda uygularsanız bütün başlangıç değerleri için 1 sayısına ulaşırsınız. Collatz sanısının ifadesi oldukça basit ve anlaşılır olsa da günümüze kadar ispatlanamamıştır ve bu sanı matematikteki çözülememiş en ünlü problemlerden biridir.

a. *Kullanıcının girdiği sayıya tekrar tekrar yukarıdaki kuralı uygulayıp elde ettiği sayıları listeleyen bir program yazın. Ayrıca, zaman serisi kullanarak bu listeyi görselleştirin.*

b. *İlk 1000 tane doğal sayının bu sanıyı sağladığını SageMath yardımıyla gösterin.*

Alıştırma 8.14.2 (Göç Problemi) *İzmir ve İstanbul'un nüfusları sırasıyla 4.4 ve 15.5 milyondur. Varsayalım ki her sene İzmir'in nüfusunun yüzde 2'si İstanbul'a, İstanbul'un nüfusunun da yüzde 1'i İzmir'e göç ediyor. Yüz yıl sonra bu şehirlerin nüfusları nasıl olur? İki şehir için senelik değişimleri tablo ve zaman serisi olarak gösterin.*

8.15 Doğum Günü Problemi

Bir odada 23 kişi var. Bu odadaki birilerinin aynı doğum gününe sahip olma olasılığı % 50 gibi bir şey. Kişi sayısı 75 olsaydı bu olasılık % 100'e yakın olacaktı. Şaşırtıcı ama gerçek.

Bu bölümde n kişilik bir grupta en az iki kişinin aynı doğum gününe sahip olma olasılığını SageMath yardımıyla hesaplayacağız. Birkaç varsayımla başlayalım: Grupta ikiz, üçüz vb. olmadığını, yılın 365 gün ve doğum günlerinin eş olası olduğunu varsayıyoruz.

Daha somut olması açısından, ilk olarak grubun 23 kişi olduğu durumu inceleyelim; sonra genelleriz. Bu grupta en az iki kişinin ortak doğum gününe sahip olması olayına A diyelim. Bu durumda hiç kimsenin ortak doğum gününe sahip olmaması olayı A'nın tümleyeni yani A^c olacaktır. Bizim amacımız A'nın olasılığını, yani $P(A)$ değerini bulmak. Bunun için önce hesaplaması daha kolay olan $P(A^c)$ olasılığını bulacağız sonra da

$$P(A) = 1 - P(A^c) \tag{8.8}$$

eşitliğinden istediğimiz olasılığı bulabiliriz. A^c olayını, yani hiç kimsenin aynı doğum gününe sahip olmaması durumunu şöyle ifade edelim: 2'nci kişinin 1'inci kişiden farklı, 3'üncü kişinin 2 ve 1'inci kişilerden farklı, 4'üncü kişinin 3, 2 ve 1'inci kişilerden farklı ve benzer şekilde 23'üncü kişinin de 22'nci, 21'inci, ..., 2'nci ve 1'inci kişilerden farklı doğum gününe sahip olması durumu. Matematiksel olarak 2'inci kişinin 1'inci kişiden farklı olma olasılığı $1 - \frac{1}{365} = \frac{364}{365}$ olacaktır. 1'incinin 2'nciden farklı olması koşuluyla 3'üncünün diğer ikisinden farklı olma olasılığı $1 - \frac{2}{365} = \frac{363}{365}$ olacaktır. Benzer şekilde bir hesaplamayla 23 kişinin farklı doğum günlerine sahip olma olasılığını aşağıdaki gibi elde ederiz:

$$P(A^c) = \frac{364}{365} \times \frac{363}{365} \times \frac{362}{365} \times \cdots \times \frac{343}{365}$$

Buradan, (8.8) eşitliğini kullanarak $P(A)$ olasılığını elde ederiz.

Genel olarak, k kişi için bu çarpımı $1 - \frac{k-1}{365}$ terimine kadar sürdürürüz. SageMath ile bu hesaplamayı yaparken bir `for` döngüsü işleri kolaylaştıracaktır. Aşağıda gruptaki kişi sayısına bağlı olarak istediğimiz olasılığı veren, `dg` adında bir doğum günü fonksiyonu tanımlıyoruz:

```
def dg(k):
    olas=1
    for j in [1..(k-1)]:
        olas=olas*(1-j/365)
    return( (1-olas).n() )
```

```
dg(23)
```

```
0.507297234323986
```

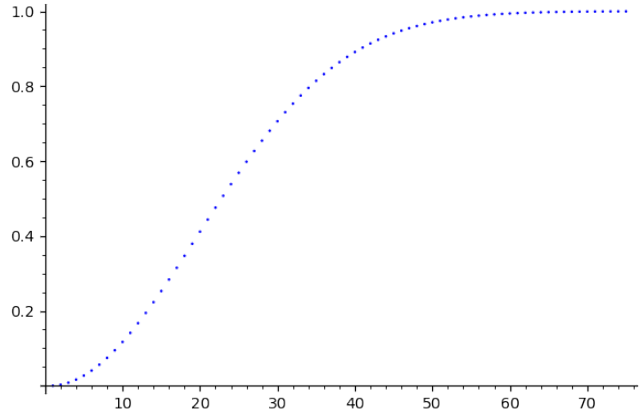
Yukarıda, 23 kişilik durum için olasılık sonucunu elde ettik. 75 kişi için:

```
dg(75)
```

```
0.999719878173811
```

Bu durumu daha somut ve genel bir şekilde görmek için grafiği inceleyelim. Kişi sayısı ve ona karşılık gelen olasılık bilgisini veren grafiği ilk 75 değer için `list_plot` ile çizelim:

```
list_plot( [ (j, dg(j)) for j in [1..75] ] , size=3 )
```



Görüldüğü üzere oldukça hızlı büyüyen bir fonksiyon var karşımızda.

Not 8.15.1 Bu hızlı büyüme kişi sayısı biraz arttığında hesaplamada bazı sorunlara neden olur. Eğer `dg(150)` değerini hesaplarsanız cevabın saçma bir şekilde 1 olduğunu göreceksiniz. Bunun sebebi SageMath'in kullandığı hassassiyetin (*precision*) varsayılan değerinin 53 bit olması. Bunu arttırmak için `prec` komutunu kullanırız. Aşağıda bu hassasiyeti 600 bit alıyoruz ve artık 365'e kadar hassas hesaplamalar yapacak durumdayız.

```
def dg(k):
    olas=1
    for j in [1..(k-1)]:
        olas=olas*(1-j/365)
    return( (1-olas).n(prec=600) )
```

```
dg(365)
```

```
0.99999999999999999999999999999999999999999999999999999999999999999999999999\
99999999999999999999999999999999999999999999999999999999999999999999999999\
99999999999999999999854504478438129659662860
```

Not 8.15.2 Görüldüğü üzere kişi sayısının 365 olduğu durumda bile olasılık hala 1 değil, 1'den biraz küçük. Çünkü olasılık `1-dg(365)` gibi çok küçük bir sayı olsa da 365

kişinin doğum gününün 365 farklı günde olma olasılığı var tabii ki. Ama 366 (ya da daha fazla) kişi olduğu durumda doğum günü aynı gün olan birilerinin kesin olarak bulunacağı açıktır. Bu durum matematikte *Güvercin Yuvası İlkesi* olarak bilinir.

Alıştırma 8.15.1 $x_0 = 2$ *olmak üzere*

$$x_{n+1} = \frac{1}{2}\left(\frac{n}{x_n} - x_n\right)$$

özyineli fonksiyonu veriliyor. İlk 50 iterasyonu zaman serisi olarak gösterin.

8.16 Sudoku

SageMath sizin için sudoku da çözüyor. Çözmek istediğiniz sudoku bulmacasını matris olarak yazıyorsunuz ve `sudoku` komutu size problemin bir çözümünü veriyor. Bilmemiz gereken tek şey bulmacadaki boşluklar yerine 0 yazıyor olmamız. Aşağıda basit bir 4×4 sudoku bulmacası çözdürüyoruz.

```
S = matrix( [ [3,0,4,0],[0,1,0,2],[0,4,0,3],[2,0,1,0] ] )
S
```

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

```
sudoku(S)
```

```
[3 2 4 1]
[4 1 3 2]
[1 4 2 3]
[2 3 1 4]
```

8.17 Algoritmik Sanat

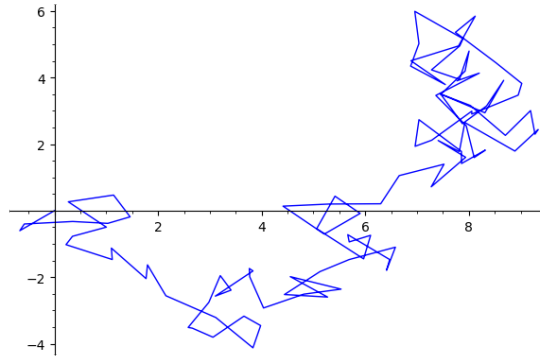
Programlama konusunda gelişmek için en uygun alanlardan biri hiç şüphesiz Algoritmik Sanat. Çünkü yaratıcılığınız sınırsız takılırken kodlama beceriniz de onu takip etmek zorunda kalacak ve böylece eğlenerek gelişme şansınız olacaktır. Bu bölümde SageMath kullanarak birkaç adımda basit bir algoritmik sanat örneği oluşturacağız.

Rastgele İlerleyen Bir Doğru Parçası Zinciri

Diyelim ki koordinat düzleminde $(0,0)$ noktasını alıp bu noktanın x ve y bileşenlerine -1 ile 1 arasından rastgele seçilen birer sayı ekledik. Böylece yeni bir nokta elde ettik. Yeni noktaya aynı işlemi uyguladık. Yani rastgele sayılar ekleyip bir nokta daha elde

ettik. Bu işlemi 100 defa uygulayıp noktaları aynı sırayla, doğru parçaları kullanarak birleştirdik. Aşağıda bu işlemi yapan bir program yazıyoruz. Detaylar ise şöyle: Sadece ilk noktayı, yani $(0,0)$ noktasını içeren bir L listesi oluşturarak işe başlıyoruz. Her adımda bu listenin son elemanının bileşenlerine rastgele sayıları, yani `rx` ve `ry` değerlerini (rastgele seçim için 4.9 numaralı bölüme bakınız) ekliyoruz. Daha sonra `append` komutu ile listeyi genişletip `line` komutu ile noktaları birleştirip son olarak da grafiği çizdiriyoruz.

```
x0=0; y0=0
L=[(x0,y0)]
for i in range(100):
    rx=-1+2*random()
    ry=-1+2*random()
    L.append( (L[-1][0]+rx , L[-1][1]+ry ) )
line(L)
```

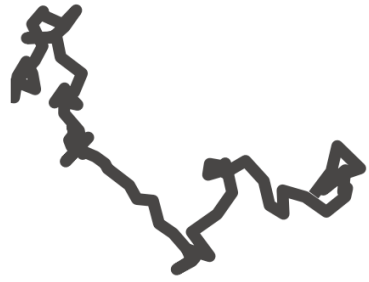


Şimdi de bu işlemin biraz daha detaylı ve bol seçenekli halini fonksiyon kullanarak yapmaya çalışalım. Aşağıda `rastgele1` diye bir fonksiyon tanımlıyoruz. Girdi olarak sırasıyla; başlangıç noktası P_0, renk, çizgi kalınlığı, saydamlık, boy/en oranı, x aralığı, y aralığı ve figür boyutu var. Bu arada `axes=False` komutu ile eksenleri gizleyelim.

```
def rastgele1(P0,iterasyon,renk,kalinlik,saydamlik,oran, xaralik,
    yaralik, figboyut ):
    x0=P0[0]
    y0=P0[1]
    L=[(x0,y0)]
    for i in [1..iterasyon]:
        rx=-1+2*random()
        ry=-1+2*random()
        L.append( (L[-1][0]+rx , L[-1][1]+ry ) )
    return line(L, axes=False, hue=renk, thickness=kalinlik,
        alpha=saydamlik, aspect_ratio=oran, xmin=xaralik[0],
        xmax=xaralik[1], ymin=yaralik[0], ymax=yaralik[1],
        figsize=figboyut )
```

Fonksiyonu tanımladık. Şimdi de bunu çalıştırıp bir doğru parçası zinciri elde ediyoruz.

```
rastgele1( (1.4,0) , 100, (.1, .1, .1 ), 10 , .8 , 1 , (-10,10),
    (-10,10) , 10 )
```

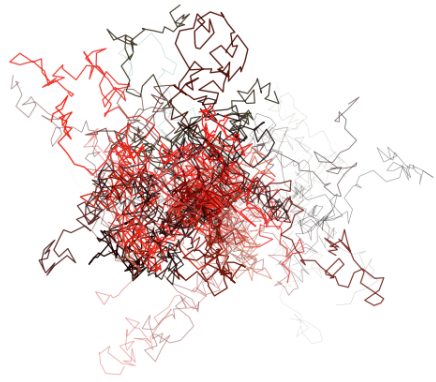


Burada renk kodunu RGB üçlüsü olarak almadık. Bunun yerine bu tip uygulamlarda yönetimi daha kolay olan HSV üçlüsü, yani Hue-Saturation-Value (Ton-Doygunluk-Değer) olarak aldık. Bunun SageMath ifadesi 3.8.1 numaralı notta verildiği üzere `hue` komutu.

Rastgele İlerleyen Pek Çok Doğru Parçası Zinciri

Şimdi `rastgele1` fonksiyonunu $(0,0)$ noktasına defalarca uygulayıp elde edilen grafik çıktılarını birleştirelim. Bunu yaparken grafiğin seçeneklerini `i` parametresine bağlı olarak değiştiridiğimize dikkat edin:

```
sum( rastgele1( (0,0), 100, ( 1/(i^2+1) , abs(cos((i^2+1) )) , random()
    ) , abs(cos(i^2+1)) , abs(cos(i^2+1)) , 1 , (-30,30), (-30,30) ,15 )
    for i in [-20..20] )
```



Kullanmış olduğumuz görsel seçenekler, komutlar ve fonksiyonlar için biraz açıklama, detay ve öneri faydalı olabilir:

1. Normalde grafik birleştirmeyi `p1+p2` şeklinde toplama işlemi ile yapıyoruz. Ancak burada pek çok grafik birleştireceğimiz için bu işlemi `sum` komutunu kullanarak yaptık.

2. Bu örnekte başlangıç noktası olarak orijini aldık ve `rastgele1` fonksiyonunu defalarca orijine uyguladık.

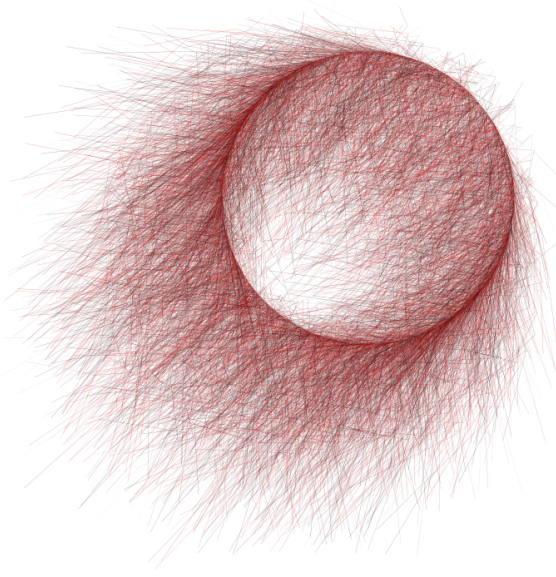
3. Renk için kullanılan `hue` komutu (HSV üçlüsü) önemli bir rol üstleniyor burada. Kırmızı yoğunluklu bir sonuç elde etmek istedik ve bunu `hue` komutunun ilk bileşenini birkaç adım sonra 0'a yakın değerler alacak olan $\frac{1}{i^2+1}$ fonksiyonu ile yaptık.

4. Kosinüs fonksiyonu 1'den büyük olmayan değerler alacağından, çizgiler hiçbir zaman çok kalın olmayacaktır. Ayrıca saydamlık da kalın çizgilerde daha az, yani çizgiler daha belirgin olacaktır.

Çembersel Başlangıç Noktaları

Şimdi de `rastgele1` fonksiyonunda küçük bir değişiklik yapalım: Rastgele sıçramaları `rx=-8+10*random()` ve `ry=-8+10*random()` olarak alalım. Yeni fonksiyonu uygulamak için de başlangıç noktası olarak $(0,0)$ yerine hareketli noktalar seçelim. Örneğin $(10\cos i, 10\sin i)$ şeklinde `i` parametresine bağlı bir fonksiyon alalım. Bu fonksiyon, başlangıç noktalarını merkezi orijin olan 10 birim yarıçaplı çember üzerinde gezdirecektir. Bununla birlikte çizgi kalınlığını 0.1 gibi çok daha ince seçip iterasyon sayısını da 2 gibi çok daha küçük bir sayı alalım. Çember üzerinde hareketli noktalara 10000 defa bu fonksiyonu uygulayalım. Sonuç aşağıda:

```
def rastgele1(P0,iterasyon,renk,kalinlik,saydamlik,oran, xaralik,
    yaralik, figboyut ):
    x0=P0[0]
    y0=P0[1]
    L=[(x0,y0)]
    for i in [1..iterasyon]:
        rx=-8+10*random()
        ry=-8+10*random()
        L.append( (L[-1][0]+rx , L[-1][1]+ry ) )
    return line(L, axes=False, hue=renk, thickness=kalinlik,
        alpha=saydamlik, aspect_ratio=oran, xmin=xaralik[0],
        xmax=xaralik[1], ymin=yaralik[0], ymax=yaralik[1],
        figsize=figboyut )
sum( rastgele1( (10*cos(i),10*sin(i)), 2, ( 1/(i^2+1) , abs(cos((i^2+1)
    )) , random() ) , .1 , abs(cos(i^2+1)) , 1 , (-30,30), (-30,30) ,15
    ) for i in [0..10000] )
```



Not 8.17.1 SageMath kullanılarak elde edilen çeşitli algoritmik sanat çalışmaları için:

`www.k-interact.net/ddsart`

Alıştırma 8.17.1 *Yukarıdaki kodları kullanarak, kullanıcının iterasyon sayısını, renk fonksiyonunu ve başlangıç noktasını (iterasyona bağlı bir fonksiyon) gireceği bir interaktif araç oluşturun.*

Alıştırma 8.17.2 *Yaratıcılığınızı kullanın ve kendinize ait bir algoritmik sanat eseri yaratın.*

Ek A

SageMath'in Kurulumu

Aşağıda Windows, Linux, ve MacOS işletim sistemlerinde SageMath programını bilgisayarınıza nasıl kuracağınız kısaca anlatılıyor. Kurulum önerileri, oldukça kullanışlı olan Jupyter Not Defteri'yle çalışmak için verilmiştir:

A.1 Windows

1. `https://github.com/sagemath/sage-windows/releases` adresine gidin.

2. `SageMath-x.x-Installer-vx.x.x.exe` dosyasını indirin. Dosya ismindeki `x`'ler yerine versiyona dair sayılar göreceksiniz.

3. İndirdiğiniz dosyayı çalıştırın ve adımları takip edin.

4. Kurulum tamamlandıktan sonra `SageMath x.x Notebook` adındaki programı çalıştırın.

A.2 Linux

1. `https://www.sagemath.org/download-source.html` adresine gidin ve bulunduğunuz yere yakın bir indirme sunucusu seçin.

2. Uygun dosyayı indirin. İndireceğiniz dosyanın uzantısı `.tar.gz` veya `.tar.bz2` olmalıdır.

3. İndirdiğiniz dosya sıkıştırılmış bir `tar` dosyasıdır. Bunu istediğiniz bir klasöre açın.

4. Komut satırında `sage -n jupyter` yazarak programı çalıştırın.

A.3 MacOS

1. `https://github.com/3-manifolds/Sage_macOS/releases` adresine gidin ve size uygun `dmg` dosyasını indirin.
2. İndirdiğiniz dosyaya çift tıklayın.
3. `SageMath` klasörünü `/Applications/`'a sürükleyin.
4. `Applications` klasöründe, `SageMath-x-x` dosyasına çift tıklayın. Arayüz seçiminde *Jupyter notebook from folder* opsiyonunu seçip dilediğiniz bir klasör seçimi yapın.

Not A.3.1 Jupyter ile kıyaslandığında komut satırı arayüzü çok tercih edilmeyen, kısıtlı bir arayüzdür. Yine de programı bu şekilde çalıştırmak isterseniz:

1. **Windows.** `SageMath x.x` isimli dosyayı çalıştırın.
2. **Linux.** Komut satırı arayüzünde `sage` komutunu çalıştırın.
3. **MacOS.** SageMath klasöründeki `sage` dosyasını çalıştırın.

Kaynakça

[1] Anastassiou, G. A., & Mezei, R. A. (2015). *Numerical analysis using sage.* Springer.

[2] Bard, G.V. (2022). *Sage for Undergraduates: Compatible with Python 3.* American Mathematical Society.

[3] Brown, J. R. (1997). Proofs and pictures. *The British Journal for the Philosophy of Science, 48(2)*, 161-180.

[4] Eröcal, B., and Stein, W. (2010). *The Sage Project: Unifying free mathematical software to create a viable alternative to Magma, Maple, Mathematica and MATLAB.* International Congress on Mathematical Software. Springer, Berlin, Heidelberg.

[5] Eves, H. W. (1992). *Fundamentals of modern elementary geometry.* Royal Society of Chemistry.

[6] Mezei, R. A. (2015). *An introduction to SAGE programming: With applications to SAGE interacts for numerical methods.* John Wiley & Sons.

[7] SageMath, the Sage Mathematics Software System (Version 9.7), The Sage Developers, 2022, https://www.sagemath.org.

[8] Stein, W. (2011). *Sage: Creating a Viable Free Open Source Alternative to Magma, Maple, Mathematica, and MATLAB.* In F. Cucker, T. Krick, A. Pinkus, & A. Szanto (Eds.), Foundations of Computational Mathematics, Budapest (London Mathematical Society Lecture Note Series, pp. 230-238). Cambridge: Cambridge University Press. doi:10.1017/CBO9781139095402.011

[9] Strogatz, S. H. (1988). *Love affairs and differential equations.* Mathematics Magazine, 61(1), 35-35.

[10] Zimmermann, P., Casamayou, A., Cohen, N., Connan, G., Dumont, T., Fousse, L., ... & Thomas, H. (2018). *Computational mathematics with SageMath.* Society for Industrial and Applied Mathematics.

Dizin