Ein Rundgang durch Sage

Release 10.5.rc0

The Sage Development Team

16.11.2024

Inhaltsverzeichnis

Das ist ein Rundgang durch Sage, der sich stark an der „Tour of Mathematica“ am Beginn des Mathematica-Buchs orientiert.

KAPITEL 1

Das Sage-„Notebook“

Abweichend vom sonstigen Sprachgebrauch bezeichnet der Begriff „Notebook“ hier keinen kleinen mobilen Computer, sondern die grafische Benutzeroberfläche von Sage. Das Besondere daran ist, dass Sage dazu den vorhandenen Webbrowser verwendet: Im Notebook-Modus startet Sage einen Webserver, der zwischen dem Browser und Sage die Ein- und Ausgabe vermittelt.

Dies kann einerseits lokal geschehen, das heißt, auf den Webserver kann nur vom gleichen Rechner aus zugegriffen werden. Es ist damit aber auch möglich, mit Sage über das Internet zu arbeiten (sogar mit einem internetfähigen Handy) oder einen Server für die Benutzer eines Computerlabors (etwa im Rahmen einer Computeralgebravorlesung) einzurichten. Natürlich unterstützt Sage einige Sicherheitsmaßnahmen, damit nur berechtigte Personen Zugang zum Sage-Notebook und damit Zugang zum Rechner bekommen.

Man kann mit Sage aber auch direkt auf der Kommandozeile arbeiten. Ob man das Notebook oder die Kommandozeile verwendet, ist letztlich Geschmackssache.

KAPITEL 2

Sage als „Taschenrechner“

Die Eingabezeile von Sage hat eine Eingabeaufforderung `sage:`. Sie müssen also `sage:` nicht selbst eingeben. Wenn Sie Sage in der Notebook-Version benutzen, dann geben Sie alle Eingaben in eine Eingabezelle ein. Die Berechnung und Ausgabe des Wertes erfolgt nach gleichzeitigem Drücken der Tasten Shift + Return (in deutsch: Umschalt- oder Hochstelltaste + Eingabetaste).

```
sage: 3 + 5
8
```

Der Zirkumflex (oft als „Dach“ bezeichnet) bedeutet „Potenzieren“.

```
sage: 57.1 ^ 100
4.60904368661396e175
```

Wir invertieren eine 2×2-Matrix in Sage:

```
sage: matrix([[1,2], [3,4]])^(-1)
[  -2    1]
[ 3/2 -1/2]
```

Hier integrieren wir eine einfache Funktion.

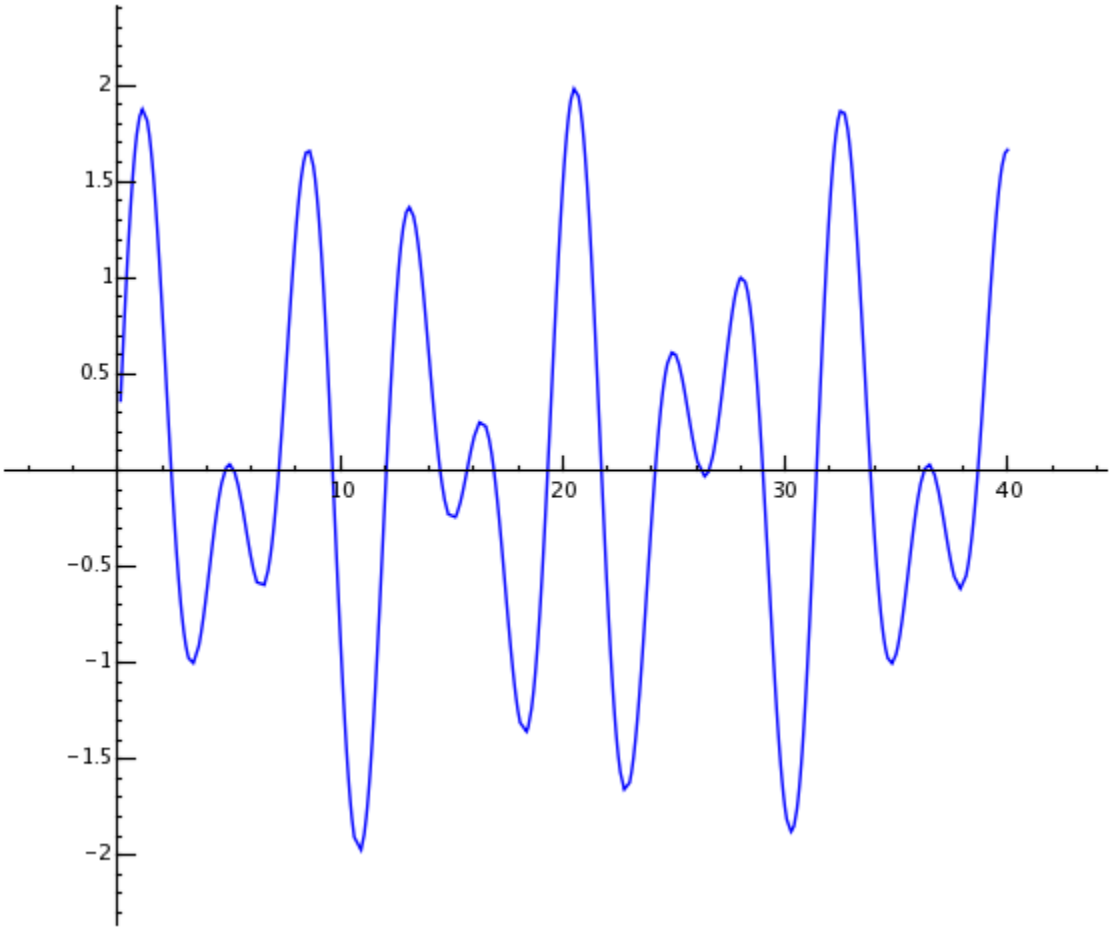
```
sage: x = var('x')   # erzeuge eine symbolische Variable
sage: integrate(sqrt(x)*sqrt(1+x), x)
1/4*((x + 1)^(3/2)/x^(3/2) + sqrt(x + 1)/sqrt(x))/((x + 1)^2/x^2 - 2*(x + 1)/x + 1) -␣
↪1/8*log(sqrt(x + 1)/sqrt(x) + 1) + 1/8*log(sqrt(x + 1)/sqrt(x) - 1)
```

Als nächstes lassen wir Sage eine quadratische Gleichung lösen. Das doppelte Gleichheitszeichen == ist in Sage das mathematische Gleichheitszeichen. (Das Zeichen = bedeutet eine Wertzuweisung.)

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

Das Ergebnis ist eine Liste von Lösungsgleichungen – hier zwei.

```
sage: S[0].rhs()
-1/2*sqrt(4*a + 1) - 1/2
sage: show(plot(sin(x) + sin(1.6*x), 0, 40))
```



KAPITEL 3

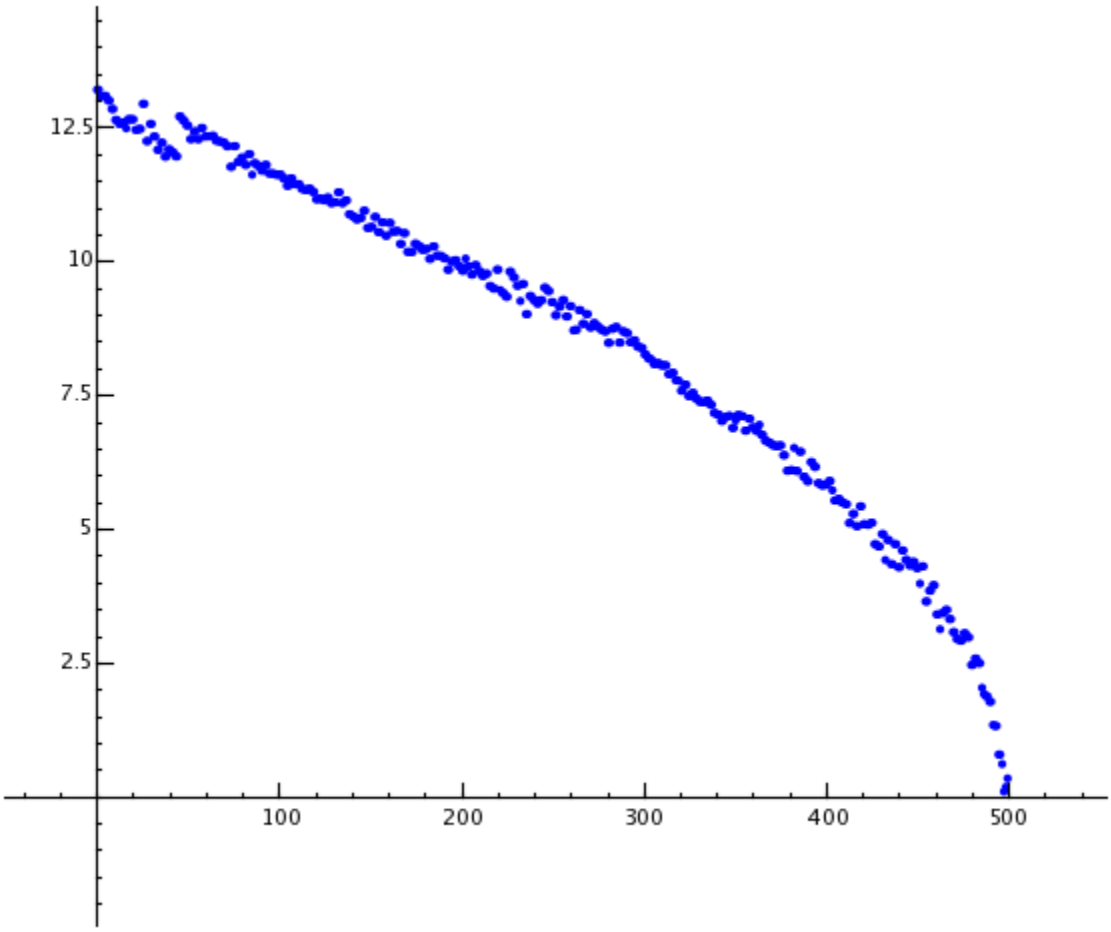
Hochleistungsrechnen mit Sage

Zuerst erstellen wir eine 500×500-Matrix mit zufälligen Einträgen.

```
sage: m = random_matrix(RDF,500)
```

Sage benötigt nur wenige Sekunden, um die Eigenwerte der Matrix zu berechnen und zu plotten.

```
sage: e = m.eigenvalues()  # ungefähr 2 Sekunden
sage: w = [(i, abs(e[i])) for i in range(len(e))]
sage: show(points(w))
```



Dank der GNU Multiprecision Library (GMP) kann Sage sehr große Zahlen mit Millionen oder Milliarden von Stellen berechnen.

```
sage: factorial(100)
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862
sage: n = factorial(1000000)  # ungefähr 2,5 Sekunden
```

Nachfolgend werden 100 Stellen von π berechnet.

```
sage: N(pi, digits=100)
3.
↪14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706
```

Sage kann ein Polynom mit zwei Variablen faktorisieren.

```
sage: R.<x,y> = QQ[]
sage: F = factor(x^99 + y^99)
sage: F
(x + y) * (x^2 - x*y + y^2) * (x^6 - x^3*y^3 + y^6) *
(x^10 - x^9*y + x^8*y^2 - x^7*y^3 + x^6*y^4 - x^5*y^5 +
 x^4*y^6 - x^3*y^7 + x^2*y^8 - x*y^9 + y^10) *
(x^20 + x^19*y - x^17*y^3 - x^16*y^4 + x^14*y^6 + x^13*y^7 -
 x^11*y^9 - x^10*y^10 - x^9*y^11 + x^7*y^13 + x^6*y^14 -
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
 x^4*y^16 - x^3*y^17 + x*y^19 + y^20) * (x^60 + x^57*y^3 -
 x^51*y^9 - x^48*y^12 + x^42*y^18 + x^39*y^21 - x^33*y^27 -
 x^30*y^30 - x^27*y^33 + x^21*y^39 + x^18*y^42 - x^12*y^48 -
 x^9*y^51 + x^3*y^57 + y^60)
sage: F.expand()
x^99 + y^99
```

Sage benötigt weniger als 5 Sekunden, um alle Partitionen (d.h. alle möglichen Zerlegungen als Summe positiver ganzer Zahlen) von 10^8, also 100 Millionen, zu bestimmen. Die Anzahl der Möglichkeiten ist gigantisch, wir geben hier nur die ersten 40 Ziffern an.

```
sage: z = Partitions(10^8).cardinality() # ungefähr 4,5 Sekunden
sage: str(z)[:40]
'1760517045946249141360373894679135204009'
```

KAPITEL 4

Sage-Algorithmen benutzen

Immer wenn Sie Sage benutzen, nutzen Sie die weltgrößte Sammlung von Open Source Computeralgorithmen. Open Source ist frei verfügbare Software, deren Quelltext öffentlich zugänglich ist, beliebig kopiert, verändert, verbreitet und genutzt werden darf, sofern der weitergegeben Quelltext öffentlich verfügbar bleibt.