Tutorial Sage

Release 9.7

The Sage Group

20 set 2022

Indice

Sage è un software di calcolo numerico e simbolico libero e open-source di supporto nell'insegnamento e della ricerca in algebra, geometria, teoria dei numeri, crittografia, calcolo numerico e aree correlate. Sia il modello di sviluppo di Sage, sia la tecnologia utilizzata in Sage stesso, si distinguono per un'enfasi particolarmente forte su apertura, community, cooperazione e collaborazione: vogliamo creare l'automobile, non reinventare la ruota. L'obiettivo complessivo di Sage è creare un'alternativa adeguata, libera e open-source a Maple, Mathematica, Magma e MATLAB.

Questo tutorial è la via migliore per familiarizzare con Sage in poche ore e può essere letto in HTML o PDF.

CAPITOLO 1

Introduzione

Questo tutorial dovrebbe richiedere circa 3/4 ore per una lettura completa. Lo si può leggere in versione HTML o PDF.

Nonostante molto in Sage sia implementato usando Python, la conoscenza di Python non è un prerequisito per la lettura di questo tutorial. Per chi volesse imparare il Python (un linguaggio molto divertente!) allo stesso tempo, ci sono molte risorse eccellenti e libere per farlo tra le quali [PyT] e [PyB]. Se si vuole solo provare velocemente Sage, questo tutorial è il punto di partenza adatto. Per esempio:

```
sage: 2 + 2
4
sage: factor(-2007)
-1 * 3^2 * 223

sage: A = matrix(4,4, range(16)); A
[ 0  1  2  3]
[ 4  5  6  7]
[ 8  9 10 11]
[12 13 14 15]

sage: factor(A.charpoly())
x^2 * (x^2 - 30*x - 80)

sage: m = matrix(ZZ,2, range(4))
sage: m[0,0] = m[0,0] - 3
sage: m
[-3  1]
[ 2  3]

sage: E = EllipticCurve([1,2,3,4,5]);
sage: E
Elliptic Curve defined by y^2 + x*y + 3*y = x^3 + 2*x^2 + 4*x + 5
over Rational Field
sage: E.anlist(10)
```

(continues on next page)

(continua dalla pagina precedente)

```
[0, 1, 1, 0, -1, -3, 0, -1, -3, -3, -3]
sage: E.rank()
1

sage: k = 1/(sqrt(3)*I + 3/4 + sqrt(73)*5/9); k
36/(20*sqrt(73) + 36*I*sqrt(3) + 27)

sage: N(k)
0.165495678130644 - 0.0521492082074256*I
sage: N(k,30)      # 30 "bits"
0.16549568 - 0.052149208*I
sage: latex(k)
\frac{36}{20 \, \sqrt{73} + 36 i \, \sqrt{3} + 27}
```

1.1 Installazione

Se non si ha Sage installato su un computer e si vogliono solamente provare alcuni comandi, si può usare online all'indirizzo https://sagecell.sagemath.org/.

Si veda la Sage Installation Guide nella sezione documentazione della homepage di Sage [Sage] per istruzioni sull'installazione di Sage sul proprio computer. Qui vengono fatti solamente due commenti.

1. Il file di download di Sage arrive con le «batterie incluse». In altre parole, nonostante Sage usi Python, IPython, PARI, GAP, Singular, Maxima, NTL, GMP e così via, non è necessario installarli separatemente siccome sono incluse con la distribuzione di Sage. Comunque, per usare certe feature di sage, ad es. Macaulay o KASH, bisogna installare il pacchetto opzionale Sage che interessa o almeno avere i programmi in questioni gia installati sul proprio computer. Macaulay e KASH sono pacchetti di Sage (per una lista dei pacchetti opzionali disponibili, digitare «sage -optional», o sfogliare la pagina «Download» sul sito web di Sage.

2. Le versioni binarie precompilate di Sage (che si trovano sul sito web di Sage) possono essere più facili e più veloci da installare invece che la versione da codice sorgente. Basta solo spachettare il file e eseguire «sage».

1.2 Modi di usare Sage

Sage si può usare in molti modi.

- **Interfaccia grafica del notebook:** vedere la sezioni sul Notebook nel manuale di riferimento e la :ref:»sezione-notebook» sotto,
- **Linea di comando interattiva:** vedere :ref:"capitolo-shell_interattiva",
- **Programmi:** scrivendo programmi interpretati e compilati in Sage (vedere :ref:"sezione-loadattach" e :ref:"sezione-compilazione"), e
- **Scripts:** scrivendo degli script autosufficienti che usino la libreria Sage (vedere :ref:"sezione-autosufficienti").

1.3 Obiettivi di lungo periodo per Sage

- **Utile**: il pubblico per Sage il quale sage è stato pensato sono gli studentu di matematica (dalla scuola superiore all'università), gli insegnanti e i ricercatori in matematica. Lo scopo è di fornire software che possa essere usato per esplorare e sperimentare le costruzioni matematiche in algebra, geometria, teoria dei numeri, calcolo, calcolo numerico, ecc. Sage aiuta a rendere più facile la sperimentazione interattiva con gli oggetti matematici.
- **Efficiente:** essere veloce. Sage usa del software maturo e altamente ottimizzato come GMP, PARI, GAP e NTL e così è molto veloce con certe operazioni.
- **Libero e open source:** il codice sorgente deve essere liberamente disponibile e leggibile, così che gli utenti posssano capire cosa stia facendo veramente il sistema e possano estenderlo più facilmente. Così come i matematici acquisiscono una comprensione più profonda di un teorema leggendo attentamete o almeno scorrendo velocemente la dimostrazione, le persone che fanno calcoli dovrebbero essere capaci di capire come funzionano i calcoli leggengo il codice sorgente documentato. Se si usa Sage per fare calcoli in un articolo che si pubblica, si può essere rassicurati dal fatto che i lettori avranno sempre libero accesso a Sage e a tutto il suo codice sorgente ed è persino concesso di archiviare la versione di Sage che si è utilizzata.
- **Facile da compilare:** Sage dovrebbe essere facile da compilare dal sorgente per gli utenti Linux, macOS e Windows. Questo garantisce maggiore flessibilità agli utenti di modificare il sistema.
- **Cooperazione:** Fornire un interfaccia robusta alla maggior parte degli altri sistemi di algebra computazionale, compresi: PARI, GAP, Singular, Maxima, KASH, Magma, Maple e Mathematica. Sage è pensato per unificare e estendere il software matematico esistente.
- **Ben documentato:** tutorial, guida alla programmazione, manuale di riferimento e how to con numerosi esempi e discussioni della matematica sottostante.
- **Amichevole verso l'utente:** dovrebbe essere facile capire quale funzionalità è fornita per un dato oggetto e guardare la documentazione e il codice sorgente. Bisogna anche raggiungere un alto livello di supporto agli utenti.

CAPITOLO 2

Algebra di base e Analisi

Sage sa svolgere diversi calcoli legati all'algebra di base ed all'analisi: per esempio, risoluzione di equazioni, calcolo differenziale ed integrale e trasformate di Laplace. Si veda la documentazione per le «Costruzioni di Sage» per ulteriori esempi.

2.1 Risoluzione di equazioni

La funzione `solve` risolve le equazioni. Per usarla, bisogna anzitutto specificare alcune variabili; pertanto gli argomenti di `solve` sono un'equazione (od un sistema di equazioni), insieme con le variabili rispetto alle quali risolvere:

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

Si possono risolvere le equazioni rispetto ad una variabile in funzione delle altre:

```
sage: x, b, c = var('x b c')
sage: solve([x^2 + b*x + c == 0],x)
[x == -1/2*b - 1/2*sqrt(b^2 - 4*c), x == -1/2*b + 1/2*sqrt(b^2 - 4*c)]
```

Si può anche risolvere rispetto a diverse variabili:

```
sage: x, y = var('x, y')
sage: solve([x+y==6, x-y==4], x, y)
[[x == 5, y == 1]]
```

Il seguente esempio dell'uso di Sage per risolvere un sistema di equazioni non lineari è stato fornito da Jason Grout: per prima cosa, si risolve il sistema simbolicamente:

```
sage: var('x y p q')
(x, y, p, q)
sage: eq1 = p+q==9
```

(continues on next page)

(continua dalla pagina precedente)

```
sage: eq2 = q*y+p*x==-6
sage: eq3 = q*y^2+p*x^2==24
sage: solve([eq1,eq2,eq3,p==1],p,q,x,y)
[[p == 1, q == 8, x == -4/3*sqrt(10) - 2/3, y == 1/6*sqrt(10) - 2/3],
[p == 1, q == 8, x == 4/3*sqrt(10) - 2/3, y == -1/6*sqrt(10) - 2/3]]
```

Per una soluzione numerica, si può invece usare:

```
sage: solns = solve([eq1,eq2,eq3,p==1],p,q,x,y, solution_dict=True)
sage: [[s[p].n(30), s[q].n(30), s[x].n(30), s[y].n(30)] for s in solns]
[[1.0000000, 8.0000000, -4.8830369, -0.13962039],
 [1.0000000, 8.0000000, 3.5497035, -1.1937129]]
```

(La funzione `n` scrive un'approssimazione numerica, e l'argomento è il numero di bit di precisione.)

2.2 Differenziazione, Integrazione, etc.

Sage è in grado di differenziae ed integrare molte funzioni. Per esempio, per differenziare $\sin(u)$ rispetto a u, si procede come nelle righe seguenti:

```
sage: u = var('u')
sage: diff(sin(u), u)
cos(u)
```

Per calcolare la derivata quarta di $\sin(x^2)$:

```
sage: diff(sin(x^2), x, 4)
16*x^4*sin(x^2) - 48*x^2*cos(x^2) - 12*sin(x^2)
```

Per calcolare le derivate parziali di $x^2 + 17y^2$ rispetto a *x* e *y*, rispettivamente:

```
sage: x, y = var('x,y')
sage: f = x^2 + 17*y^2
sage: f.diff(x)
2*x
sage: f.diff(y)
34*y
```

Passiamo agli integrali, sia indefiniti che definiti. Per calcolare $\int x\sin(x^2)\,dx$ e $\int_0^1 \frac{x}{x^2+1}\,dx$

```
sage: integral(x*sin(x^2), x)
-1/2*cos(x^2)
sage: integral(x/(x^2+1), x, 0, 1)
1/2*log(2)
```

Per calcolare la decomposizione in frazioni parziali di $\frac{1}{x^2-1}$:

```
sage: f = 1/((1+x)*(x-1))
sage: f.partial_fraction(x)
-1/2/(x + 1) + 1/2/(x - 1)
```

2.3 Risoluzione di Equazioni Differenziali

Si può usare Sage per studiare le equazioni differenziali ordinarie. Per risolvere l'equazione $x' + x - 1 = 0$:

```
sage: t = var('t')    # definisce una variabile t
sage: x = function('x')(t)   # definisce x come funzione di quella variabile
sage: DE = diff(x,t) + x - 1
sage: desolve(DE, [x,t])
(_C + e^t)*e^(-t)
```

Questo metodo utilizza l'interfaccia di Sage per Maxima [Max], e così il suo output può essere leggermente diverso dagli altri output di Sage. In questo caso, risulta che la soluzione generale dell'equazione differenziale è $x(t) = e^{-t}(e^t + c)$.

Si può anche calcolare la trasformata di Laplace; la trasformata di Laplace di $t^2 e^t - \sin(t)$ è calcolata come segue:

```
sage: s = var("s")
sage: t = var("t")
sage: f = t^2*exp(t) - sin(t)
sage: f.laplace(t,s)
-1/(s^2 + 1) + 2/(s - 1)^3
```

Il successivo è un esempio più articolato. Lo scostamento dall'equilibrio (rispettivamente) per due molle accoppiate fissate ad un muro a sinistra

```
|------\/\/\/\/\---|massa1|----\/\/\/\/\/----|massa2|
         molla1                   molla2
```

è modellizzato dal sistema di equazioni differenziali del secondo ordine

$$m_1 x_1'' + (k_1 + k_2)x_1 - k_2 x_2 = 0 m_2 x_2'' + k_2(x_2 - x_1) = 0,$$

dove m_i è la massa dell'oggetto i, x_i è lo scostamento dall'equilibrio della massa i, e k_i è la costante elastica della molla i.

Esempio: Usare Sage per risolvere il problema precedente con $m_1 = 2$, $m_2 = 1$, $k_1 = 4$, $k_2 = 2$, $x_1(0) = 3$, $x_1'(0) = 0$, $x_2(0) = 3$, $x_2'(0) = 0$.

Soluzione: Calcolare la trasformata di Laplace della prima equazione (con la notazione $x = x_1$, $y = x_2$:

```
sage: de1 = maxima("2*diff(x(t),t, 2) + 6*x(t) - 2*y(t)")
sage: lde1 = de1.laplace("t","s"); lde1
2*((-%at('diff(x(t),t,1),t = 0))+s^2*'laplace(x(t),t,s)-x(0)*s) -2*'laplace(y(t),t,s)+6*
↪'laplace(x(t),t,s)
```

Questo è di difficile lettura, ma dice che

$$-2x'(0) + 2s^2 * X(s) - 2sx(0) - 2Y(s) + 6X(s) = 0$$

(dove la trasformata di Laplace di una funzione in minuscolo come $x(t)$ è la funzione in maiuscolo $X(s)$). Calcolare la trasformata di Laplace della seconda equazione:

```
sage: de2 = maxima("diff(y(t),t, 2) + 2*y(t) - 2*x(t)")
sage: lde2 = de2.laplace("t","s"); lde2
(-%at('diff(y(t),t,1),t = 0))+s^2*'laplace(y(t),t,s) +2*'laplace(y(t),t,s)-2*
↪'laplace(x(t),t,s) -y(0)*s
```

che significa

$$-Y'(0) + s^2Y(s) + 2Y(s) - 2X(s) - sy(0) = 0.$$

Imporre le condizioni iniziali per $x(0)$, $x'(0)$, $y(0)$, e $y'(0)$, e risolvere le due equazioni risultanti:

```
sage: var('s X Y')
(s, X, Y)
sage: eqns = [(2*s^2+6)*X-2*Y == 6*s, -2*X +(s^2+2)*Y == 3*s]
sage: solve(eqns, X,Y)
[[X == 3*(s^3 + 3*s)/(s^4 + 5*s^2 + 4),
Y == 3*(s^3 + 5*s)/(s^4 + 5*s^2 + 4)]]
```

Ora si calcola la trasformata inversa di Laplace per ottenere la risposta:

```
sage: var('s t')
(s, t)
sage: inverse_laplace((3*s^3 + 9*s)/(s^4 + 5*s^2 + 4),s,t)
cos(2*t) + 2*cos(t)
sage: inverse_laplace((3*s^3 + 15*s)/(s^4 + 5*s^2 + 4),s,t)
-cos(2*t) + 4*cos(t)
```

Pertanto, la soluzione è

$$x_1(t) = \cos(2t) + 2\cos(t), \quad x_2(t) = 4\cos(t) - \cos(2t).$$

Essa può essere disegnata in forma parametrica usando

```
sage: t = var('t')
sage: P = parametric_plot((cos(2*t) + 2*cos(t), 4*cos(t) - cos(2*t) ),
....: (0, 2*pi), rgbcolor=hue(0.9))
sage: show(P)
```

Le singole componenti possono essere tracciate usando:

```
sage: t = var('t')
sage: p1 = plot(cos(2*t) + 2*cos(t), 0, 2*pi, rgbcolor=hue(0.3))
sage: p2 = plot(4*cos(t) - cos(2*t), 0, 2*pi, rgbcolor=hue(0.6))
sage: show(p1 + p2)
```

BIBLIOGRAFIA: Nagle, Saff, Snider, Fundamentals of Differential Equations, 6th ed, Addison-Wesley, 2004. (si veda § 5.5).

2.4 Metodo di Eulero per i sistemi di equazioni differenziali

Nel prossimo esempio, si illustrerà il metodo di Eulero per le ODE di primo e secondo ordine. Per prima cosa ricordiamo l'idea di base per le equazioni di primo ordine. Dato un problema di Cauchy della forma

$$y' = f(x, y)y(a) = c$$

si vuole trovare il valore approssimato della soluzione a $x = b$ con $b > a$.

Ricordando dalla definizione di derivata che

$$y'(x) \approx \frac{y(x+h) - y(x)}{h},$$

dove $h > 0$ è dato e piccolo. Questo e la DE insieme danno give $f(x, y(x)) \approx \frac{y(x+h)-y(x)}{h}$. Ora si risolve per $y(x+h)$:

$$y(x+h) \approx y(x) + h * f(x, y(x)).$$

Se chiamiamo $hf(x, y(x))$ il «termine di correzione» (per mancanza di un termine migliore), $y(x)$ il «vecchio valore di *y*», e $y(x+h)$ il «nuovo valore di *y*», allora questa approssimazione può essere espressa come

$$y_{new} \approx y_{old} + h * f(x, y_{old}).$$

Se si spezza l'intervallo da *a* a *b* in *n* intervalli, dimodoché $h = \frac{b-a}{n}$, allora si possono registrare le informazioni per questo metodo in una tabella.

x	y	$hf(x,y)$
a	c	$hf(a,c)$
$a+h$	$c+hf(a,c)$	$\ldots$
$a+2h$	$\ldots$	
$\ldots$		
$b=a+nh$	???	$\ldots$

L'obiettivo è riempire tutti gli spazi vuoti della tavella, una riga alla volta, finché si arriva al valore ???, che è il metodo di approssimazione di Eulero per $y(b)$.

L'idea per sistemi di ODE è simile.

Esempio: Si approssimi numericamente $z(t)$ a $t = 1$ usando 4 passi del metodo di Eulero, dove $z'' + tz' + z = 0$, $z(0) = 1$, $z'(0) = 0$.

Si deve ridurre l'ODE di secondo ordine ad un sistema di due equazioni del primo ordine (usando $x = z$, $y = z'$) ed applicare il metodo di Eulero:

```
sage: t,x,y = PolynomialRing(RealField(10),3,"txy").gens()
sage: f = y; g = -x - y * t
sage: eulers_method_2x2(f,g, 0, 1, 0, 1/4, 1)
      t                x            h*f(t,x,y)                y       h*g(t,x,y)
      0                1                  0.00                0            -0.25
    1/4              1.0                -0.062            -0.25            -0.23
    1/2             0.94                 -0.12            -0.48            -0.17
    3/4             0.82                 -0.16            -0.66           -0.081
      1             0.65                 -0.18            -0.74            0.022
```

Pertanto, $z(1) \approx 0.75$.

Si possono anche tracciare i punti (x, y) per ottenere un grafico approssimato della curva. La funzione `eulers_method_2x2_plot` svolge questa funzione; per usarla, bisogna definire le funzioni *f* e *g* che prendono on argomento con tre coordinate: (*t*, *x*, *y*).

```
sage: f = lambda z: z[2]        # f(t,x,y) = y
sage: g = lambda z: -sin(z[1])  # g(t,x,y) = -sin(x)
sage: P = eulers_method_2x2_plot(f,g, 0.0, 0.75, 0.0, 0.1, 1.0)
```

A questo punto, P ha in memoria due grafici: `P[0]`, il grafico di *x* vs. *t*, e `P[1]`, il grafico di *y* vs. *t*. Si possono tracciare entrambi come mostrato qui in seguito:

```
sage: show(P[0] + P[1])
```

2.5 Funzioni speciali

Sono implementati diversi polinomi ortogonali e funzioni speciali, usando sia PARI [GAP] che Maxima [Max]. Essi sono documentati nelle sezioni apposite («Polinomi ortogonali» e «Funzioni speciali», rispettivamente) del manuale di Sage.

```
sage: x = polygen(QQ, 'x')
sage: chebyshev_U(2,x)
4*x^2 - 1
sage: bessel_I(1,1).n(250)
0.56515910399248502720769602760986330732889962162109200948029448947925564096
sage: bessel_I(1,1).n()
0.565159103992485
sage: bessel_I(2,1.1).n()
0.167089499251049
```

A questo punto, Sage ha soltanto incorporato queste funzioni per l'uso numerico. Per l'uso simbolico, si usi direttamente l'intefaccia di Maxima, come nell'esempio seguente:

```
sage: maxima.eval("f:bessel_y(v, w)")
'bessel_y(v,w)'
sage: maxima.eval("diff(f,w)")
'(bessel_y(v-1,w)-bessel_y(v+1,w))/2'
```

CAPITOLO 3

Indici e tabelle

- genindex
- modindex
- search

Bibliografia

[PyB] (en) The Python Beginner's Guide, https://wiki.python.org/moin/BeginnersGuide

[PyT] (en) The Python Tutorial, https://docs.python.org/3/tutorial/

[Sage] (en) Sage, https://www.sagemath.org

[GAP] (en) The GAP Group, GAP - Groups, Algorithms, and Programming, Version 4.11; 2021, https://www.gap-system.org

[Max] (en) Maxima, Version 5.45; 2021, http://maxima.sf.net/