41　代数的計算

41.1　文字式の展開・因数分解・単純化

関数の展開 (expand)

文字式 f の展開は次で行う：

```
f.expand()
```

または

```
expand(f)
```

例えば $(x+1)^3$ を展開するには次のようにする：

```
f = (x+1)^3
print f.expand()
```

実行結果の例

```
x^3 + 3*x^2 + 3*x + 1
```

関数 f の因数分解 (factor)

関数 f の因数分解は次で行う：

```
 f.factor()
```

または

```
 factor(f)
```

試しに $x^3+3x-2x-6$ の因数分解をしてみましょう。

```
f = x^3 + 3*x^2 - 2*x - 6
factor(f)
```

実行結果の例

```
(x + 3)*(x^2 - 2)
```

となります。factor では整数係数の範囲でしか因数分解されないので，より一般的に多項式の根を求めるには後述の solve を使います。

方程式を単純化するには simplify があります。

関数 f の単純化 (simplify と simplify_full)

複雑な関数 f を単純化して整理するには

```
f.simplify()     または     simplify(f)
```

とします。時間はかかるけれどもできる限り単純にするには次のようにします：

```
f.simplify_full()    または    simplify_full(f)
```

例として $f=x^2+2x+5-3x$ を単純化してみましょう：

```
f = x^2 + 2*x + 5 - 3*x
simplify(f)
```

実行結果の例

```
x^2 - x + 5
```

simplify は加法・減法でできる程度の単純化しかできませんが，simplify_full は三角法による単純化もできます：

```
sage: f = sin(x)^2 + cos(x)^2
sage: f.simplify()       # simplify は sin^2+cos^2=1 を知らない
sin(x)^2 + cos(x)^2      # 変化なし
sage: f.simplify_full() # こちらなら
1                        # 1になる
```

41.2 $x^{105}-1$ の因数分解

計算機により手計算では分からない意外な発見をすることがあるかもしれません。$x^{20}-1$ を整数係数の範囲で因数分解してみましょう。

```
factor(x^20 - 1)
```

答えは

$$\left(x^8-x^6+x^4-x^2+1\right)\left(x^4+x^3+x^2+x+1\right)\left(x^4-x^3+x^2-x+1\right)\left(x^2+1\right)(x+1)(x-1)$$

となります。このように，x^n-1 を整数係数で因数分解したとき，係数は ±1 しか出てこないでしょうか？答えは否定的です。次の例を見てみましょう。

```
factor(x^105 - 1)
```

$$\begin{aligned}&\left(x^{48}+x^{47}+x^{46}-x^{43}-x^{42}-2\,x^{41}-x^{40}-x^{39}+x^{36}+x^{35}+x^{34}+x^{33}+x^{32}+x^{31}-x^{28}-x^{26}\right.\\&\quad\left.-x^{24}-x^{22}-x^{20}+x^{17}+x^{16}+x^{15}+x^{14}+x^{13}+x^{12}-x^9-x^8-2\,x^7-x^6-x^5+x^2+x+1\right)\\&\times\left(x^{24}-x^{23}+x^{19}-x^{18}+x^{17}-x^{16}+x^{14}-x^{13}+x^{12}-x^{11}+x^{10}-x^8+x^7-x^6+x^5-x+1\right)\\&\times\left(x^{12}-x^{11}+x^9-x^8+x^6-x^4+x^3-x+1\right)\left(x^8-x^7+x^5-x^4+x^3-x+1\right)\\&\times\left(x^6+x^5+x^4+x^3+x^2+x+1\right)\left(x^4+x^3+x^2+x+1\right)\left(x^2+x+1\right)\left(x-1\right)\end{aligned}$$

となります。赤色の文字で示したように，因数分解したとき ±1 以外の係数 2 が現れます。

41.3 有理式の部分分数展開

有理式を

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

のように分解することを部分分数分解といいます。

式 f の部分分数分解

```
f.partial_fraction()
```

例：

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

元に戻すには factor を使います。

```
sage: g = -2/(x + 1) + 1/(x + 1)^2 +1
sage: g.factor()
x^2/(x + 1)^2
```

41.4 連分数

実数 x に対して x を越えない最大の整数を $a_0 = \lfloor x \rfloor$ とし，$x_1 = 1/(x - a_0)$ とおきます。このとき

$$x = a_0 + \frac{1}{x_1}$$

です。同様に $a_1 = \lfloor x_1 \rfloor$，$x_2 = 1/(x_1 - a_1)$ とすると

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{x_2}}$$

となります。これを繰り返すと

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \cdots}}}}$$

という表示が得られます。x の連分数表示といいます。この連分数を簡略化して次の記号で書きます：

$$x = [a_0 : a_1, a_2, a_3, a_4, \cdots].$$

x が有理数ならば連分数表示は有限の長さになることが知られています。例えば

$$\frac{345}{29} = 11 + \frac{26}{29} = 11 + \cfrac{1}{1 + \cfrac{3}{26}} = 11 + \cfrac{1}{1 + \cfrac{1}{8 + \cfrac{2}{3}}}$$

$$= 11 + \cfrac{1}{1 + \cfrac{1}{8 + \cfrac{1}{1 + \cfrac{1}{2}}}}$$

したがって $345/29 = [11; 1, 8, 1, 2]$。連分数表示の重要な特徴は，無理数であっても非常にきれいなパターンで表せることです。

数の連分数表示

実数 a の連分数表示は

```
continued_fraction(a)     # 連分数を返す
continued_fraction_list(a, オプション)    # 連分数のリストを返す
```

オプションは連分数の精度を決める bits=40 などを指定します。

例：

```
sage: continued_fraction(345/29)
[11, 1, 8 , 1, 2]
sage: continued_fraction(sqrt(2))
[1; 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, ...]
```

上の計算では有限個しか表示されていないが，実際は無限の長さの連分数であり $\sqrt{2} = [1; 2, 2, 2, 2, \cdots]$ です。また continued_fraction_list を使うと精度を指定して連分数展開のリストを得ることができます。

```
sage: a = continued_fraction_list(sqrt(3), bits=40)  #40ビットの精度
sage: a
[1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 3]
```

超越数であっても美しい連分数展開ができる場合があります。自然対数の底の連分数表示は

```
sage: continued_fraction(RealField(400)(e))  # ネピア数の連分数表示
[2; 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1,
12, 1, 1, 14, 1, 1, 16, 1, 1, 18, 1, 1, 20, 1, 1, 22,
 1, 1, 24, 1, 1, 26, 1, 1, 28, 1, 1, 30, 1, 1, 32, 1,
1, 34, 1, 1, 36, 1, 1, 38, 1, 1, 40, 1, 1, 42, 1, 1,
44, 1, 1, 46, 1, 1, 48, 1, 1, 50, 1, 1, 52, 1, 1, 54,
1, 1, 56, 1, 1, 58, 1, 1, 60, 1, 1, 62, 1, 1, 64, 1, 1,
66, 1, 1, 68, 2]
```

となります。上で RealField(400)(e) としたのは e の近似値を使うためです。

41.5 極限と条件

Sage では無限大 ∞ を oo のように小文字のオーを二つ並べて表します。関数の極限 $\lim_{x\to a} f(x)$ を求めるには limit(....) を用います。

式 f の極限

関数 f の極限

$$\lim_{x\to a} f(x) \qquad \lim_{x\to a+0} f(x) \qquad \lim_{x\to a-0} f(x)$$

はそれぞれ次のように計算します。

```
f.limit(x = a)     または     limit(f, x=a)
f.limit(x=a, dir='+')    # 右極限
f.limit(x=a, dir='-')    # 左極限
```

試しに次式の右辺の極限を計算してみましょう：

$$e = \lim_{x\to\infty} \left(1 + \frac{1}{x}\right)^x$$

```
sage: f = (1 + 1/x)^x
sage: f.limit(x=oo)
e    # ネピア数
```

```
sage: n(e)
2.71828182845905
```

上の命令で f.limit(x=oo) 部分は limit(f, x = oo) と書いても同じ事です。

右極限と左極限の指定は次の様にします：

```
sage: limit(1/x, x=0, dir='+')    # 右極限
+Infinity                         # 正の無限大
sage: limit(1/x, x=0, dir='-')    # 左極限
-Infinity                         # 負の無限大
```

そして左極限と右極限が一致しないのですが

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

となるので，Infinity は必ずしも $+\infty$ を意味していないことがわかります。

さて次に極限

$$\lim_{x\to 0} x^a \tag{8}$$

を求めることを考えてみます。これが定まるかどうかは a の値に依存します。このようなとき，assume によって a の性質を指定すると解決する場合があります。

以下の命令を，一回実行するごとに Restart worksheet を繰り返しながら実行してみよう：

```
var('a')
limit(x^a, x=0)
```

実行結果の例

```
Traceback (click to the left of this block for traceback)
...
Is  a  positive, negative, or zero?          #a の値を聞いてくる
```

そこで $a>0$ と仮定します

```
var('a')
assume(a>0)      # a>0と仮定する
limit(x^a, x=0)
```

実行結果の例

```
Traceback (click to the left of this block for traceback)
...
Is a an integer?                             #まだ値は定まらない。
```

そして右極限を指定すると求まります：

```
var('a');  assume(a>0)
limit(x^a, x=0, dir='+')      # 右極限
```

実行結果の例

```
0
```

また a が偶数であると指定しても極限は求まります：

```
var('a');
assume(a,'even')
limit(x^a, x=0)
```

実行結果の例

```
0
```

41.6 代数的な和の計算

Sage で和を計算するには sum を使います。有限和だけでなく無限級数も厳密に計算することもできます。

$\sum_{k=a}^{b} f(k)$ を計算する

```
1 var('k')              # kを変数とする
2 sum(f(k),k,a,b)
```

1 行目のように和をとるときのダミー変数 k を定義することに注意。

まずは $1+2+\cdots+50$ を計算してみましょう。

```
1 var('k');     sum(k,k,1,50)
```

実行結果の例

```
1275
```

sum は数値としての和の算出だけでなく

$$\sum_{k=1}^{m} k = \frac{m(m+1)}{2}$$

のように厳密に成り立つ式も導出してくれます：

```
1 sage: var('k m')              # kとmを変数とする
2 sage: ans = sum(k,k,1,m)      # 和を計算してansに代入
3 sage: print ans               # ansをプリントする
4 1/2*m^2 + 1/2*m          # これが答え
5 sage: factor(ans)        # ansを因数分解
6 1/2*(m + 1)*m            # 上の答えを因数分解したもの
```

次のような公式を得ることも出来ます：

$$\sum_{k=0}^{\infty} \frac{1}{k!} = e, \qquad \sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}, \qquad \sum_{k=1}^{\infty} \frac{2^{-k}}{k(k+1)} = -\log 2 + 1$$

```
1 var('k')
2 print sum(1/factorial(k),k,0,oo)
3 print sum(1/k^2,k,1,oo)
4 print sum(2^(-k)/(k*(k+1)),k,1,oo)
```

実行結果の例

```
e
1/6*pi^2
-log(2) + 1
```

42 方程式の解

42.1 方程式を意味する等号 “==”

Python では『=』は右辺を左辺に代入することを意味しました。例えば

```
>>> x = 3
>>> print x
3
```

と書けば変数 x が新たに作られて，x には 3 が代入されます。また，『==』は方程式の真偽値を返し，if 文や while 文などの条件を記述するときに用いました。たとえば

```
>>> 2**3 == 8
True
>>> 8 > 9
False
```

のようになります。Sage ではこれらとは別に『==』には形式的な記号としての等号の意味があります。例えば

```
var('y')
x^2 + y^2 == x*y + 1     # これは単なる方程式
```

と書いても何も返されませんが，上の方程式を一つの記号式として変数に代入することができます。

```
var('y')                    # y も不定元(=形式的文字)とする。
f = x^2 + y^2 == x*y + 1    # 変数 f に方程式 x^2+y^2==x*y+1 を代入
```

このとき，次のように f の右辺や左辺を取り出す事ができます：

```
print f             #f をプリント
print f.rhs()       #f の右辺をプリント
print f.lhs()       #f の左辺をプリント
```

実行結果の例

```
x^2 + y^2 == x*y + 1
x*y + 1
x^2 + y^2
```

ここで rhs，lhs はそれぞれ右辺（right hand side），左辺（left hand side）の略です。

42.2　方程式の厳密解を求める（solve）

$f(x) = g(x)$ を満たす x を求めるには solve を使います：

方程式 $f(x) = g(x)$ を解く

```
solve(f(x)==g(x),x)     # f(x)=g(x) を x について解く
```

方程式の等号は『==』であることに注意

試しに $2x + 3 = 7$ を解いてみましょう：

```
solve(2*x+3==7,x)
```

実行結果の例

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

変数 x, y についての連立方程式を解くには次のようにします：

連立方程式の解を求める (solve)

```
var('y')
solve([方程式1,方程式2],x,y)
```

例えば連立方程式 $x + y = 6$, $x - y = 4$ を解くには次のようにします：

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

実行結果の例

```
[[x == 5, y == 1]]
```

さて 3 次方程式 $x^3 + 2 * x + 1 == 0$ を解いてみましょう：

```
solve(x^3+2*x+1==0)
```

3 次方程式の解は 3 つあり，次のようになります：

実行結果の例

```
[x == -1/2*(1/18*sqrt(59)*sqrt(3) - 1/2)^(1/3)*(I*sqrt(3) + 1) - 1/3*(I*sqrt(3) - 1)/(1/18
*sqrt(59)*sqrt(3) - 1/2)^(1/3), x == -1/2*(1/18*sqrt(59)*sqrt(3) - 1/2)^(1/3)*(-I*sqrt(3)
+ 1) - 1/3*(-I*sqrt(3) - 1)/(1/18*sqrt(59)*sqrt(3) - 1/2)^(1/3), x == (1/18*sqrt(59)
*sqrt(3) - 1/2)^(1/3) - 2/3/(1/18*sqrt(59)*sqrt(3) - 1/2)^(1/3)]
```

結果は複雑で少し見づらいですが，解はリストの形をしているので好きな部分を取り出すことができます。たとえば，3 つの解を改行してプリントするには：

```
a = solve(x^3+2*x+1==0,x)
print a[0]; print ""
print a[1]; print ""
print a[2]
```

実行結果の例

```
x == -1/2*(1/18*sqrt(59)*sqrt(3) - 1/2)^(1/3)*(I*sqrt(3) + 1) - 1/3*(I*sqrt(3) - 1)/(1/18
*sqrt(59)*sqrt(3) - 1/2)^(1/3)

x == -1/2*(1/18*sqrt(59)*sqrt(3) - 1/2)^(1/3)*(-I*sqrt(3) + 1) - 1/3*(-I*sqrt(3) - 1)/(1/
18*sqrt(59)*sqrt(3) - 1/2)^(1/3)

x == (1/18*sqrt(59)*sqrt(3) - 1/2)^(1/3) - 2/3/(1/18*sqrt(59)*sqrt(3) - 1/2)^(1/3)
```

1 番目の解の値だけが必要な場合には，方程式の右辺を取り出せばよいので `rhs()` を使って

```
a = solve(x^3+2*x+1==0,x)
a[0].rhs()
```

実行結果の例

```
-1/2*(1/18*sqrt(59)*sqrt(3) - 1/2)^(1/3)*(I*sqrt(3) + 1)
- 1/3*(I*sqrt(3) - 1)/(1/18*sqrt(59)*sqrt(3) - 1/2)^(1/3)
```

とします。`solve` コマンドでは上のような 3 次方程式や 4 次方程式は代数的に解くことが可能ですが，よく知られているように 5 次以上の方程式は代数的に解くことがきません。そのような場合は入力した方程式がそのまま出力されます：

```
solve(x^5+3*x+1==0,x)
```

実行結果の例

```
[0=x^5+3x+1]
```

でも特別な係数をもつ方程式については代数的に解くことができるかもしれません：

```
solve(x^5+x+1==0,x)
```

実行結果の例

```
[x == -1/2*(I*sqrt(3) + 1)*(1/18*sqrt(3)*sqrt(23) - 25/54)^(1/3) -
1/18*(-I*sqrt(3) + 1)/(1/18*sqrt(3)*sqrt(23) - 25/54)^(1/3) + 1/3, x ==
-1/2*(-I*sqrt(3) + 1)*(1/18*sqrt(3)*sqrt(23) - 25/54)^(1/3) -
1/18*(I*sqrt(3) + 1)/(1/18*sqrt(3)*sqrt(23) - 25/54)^(1/3) + 1/3, x ==
(1/18*sqrt(3)*sqrt(23) - 25/54)^(1/3) + 1/9/(1/18*sqrt(3)*sqrt(23) -
25/54)^(1/3) + 1/3, x == -1/2*I*sqrt(3) - 1/2, x == 1/2*I*sqrt(3) - 1/2]
```

また三角関数を含む方程式に対しても解を求めることもできるようです：

```
solve(sin(x)==1/2,x)
```

実行結果の例

```
[x == 1/6*pi]
```

しかし，もう一つの解 $x = \frac{1}{6}\pi - \pi$ は求められていないようです。さらに三角関数の加法定理などを必要とする方程式は解けないようです：

```
solve(sin(x)*cos(x)==1/2,x)
```

実行結果の例

```
[sin(x) == 1/2/cos(x)]
```

以上のように solve で解ける方程式もありますが解けない方程式も多くあります。

42.3 方程式の数値解（find_root）

方程式の数値解を求めるには find_root を使います：

方程式の数値解を区間 $[a, b]$ の中で探す

```
find_root(方程式,a,b)
```

find_root は区間の中に解が複数個存在しても 1 つの解しか返しません。

例：$x^2 + 2x - 9 = 0$ の解を $[-5, 5]$ の中で探すには次のようにします：

```
sage: find_root(x^2+2*x-9==0,-5,5)
2.1622776601683791
```

上と同じ方程式で解を探す範囲を狭くすると，解が無くなってしまいますが，このときはエラーとなります：

```
find_root(x^2+2*x-9==0,-2,2)
```

実行結果の例

```
Traceback (click to the left of this block for traceback)
...
RuntimeError: f appears to have no zero on the interval
```

エラーメッセージでは方程式 f はその区間では 0 にならなかったといっています。find_root の利点は，答えは厳密ではないけれども，複雑な関数であっても数値解を求める事ができる点にあります：

```
sage: find_root(tan(x)==8-x^3,-10,10)
2.1261689044831993
```

ただ，解付近の振る舞いが非常に悪い関数関数では，それほど正確な答えが出ないこともあります：

```
sage: find_root(x^400,-5,5)
0.13155619723565876
```

もちろん上の方程式の解は $x = 0$ のみです。これは数値解を求めるときに Newton 法という方法を用いているためで，この方法では微分係数が非常に小さくなる点の付近では近似の精度が悪くなります。

43 微分と積分

関数 $f(x)$ の微分は次で行います：

微分の計算

関数 $f(x)$ の微分は次のいずれかで行います。

```
diff(f,x)
derivative(f,x)
f.derivative(x)
```

例：

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

Sage による不定積分は次の命令で行います：

不定積分の計算

関数 $f(x)$ の不定積分 $\int f(x)dx$ は次で計算します。

```
integral(f(x),x)
integrate(f(x),x)
```

代数的に計算され結果は厳密です。ただし出力結果に積分定数はありません

例：

```
sage: integral(sin(x),x)
-cos(x)
sage: integral(x^4)
1/5*x^5
sage: integral(1/(1-x^3))
1/3*sqrt(3)*arctan(1/3*(2*x + 1)*sqrt(3)) - 1/3*log(x - 1)
+ 1/6*log(x^2 + x + 1)
```

もちろん，いつでも原始関数が求まるわけではありません：

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

上の計算では，integrate が残っており，積分が求まったとはいえません。

被積分関数に他の変数が混じっている場合は次のように，変数の宣言をしてから積分の命令を実行します：

```
a = var('a')
integral(cos(a*x),x)
```

実行結果の例

```
sin(a*x)/a
```

定積分の計算

関数 $f(x)$ の定積分 $\int_a^b f(x)dx$ は次で求めます。

```
integral(f,x,a,b)
integral(f,(x,a,b))
f.integral(x,a,b)
```

また integral のかわりに integrate を使うこともできます。

例：

```
sage: integral(x^2,x,0,2)
8/3
```

43.1 原始関数は求まらないが定積分が求まる場合

原始関数は求まらないが，不定積分の厳密値なら得られる場合があります。たとえば $(1-x^4)^{-1/2}$ の原始関数は integral では求まりませんが，定積分 $\int_0^1 (1-x^4)^{-1/2}dx$ は求まります。次を実行してみましょう：

```
ans = integrate(1/sqrt((1-x^4)),0,1)      # 積分値をansとする
print ans          #ansをプリント
print n(ans)       #ansの数値をプリント
```

実行結果の例

```
1/4*beta(1/4, 1/2)
1.31102877714606
```

ここで beta はベータ関数

$$\beta(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1}dt$$

です。Sage 4x ではベータ関数はデフォルトでサポートされていなかったので数値を得るためには Maxima を呼び出す必要がありましたが，Sage 5x からはサポートされて手軽に扱えるようになりました。

次の積分を計算してみます。

$$\int_0^\infty \frac{x^3}{e^x-1}dx = \frac{\pi^4}{15} \tag{9}$$

これはプランクの輻射公式から空洞のエネルギー密度を計算するときに現れます。次のように積分を実行してみます。

```
f = x^3/(e^x-1)          # 関数fを定義
integrate(f,x,0,oo)      # f(x)を0から∞まで積分
```

Sage 8.2 では実行結果は次のようになりました。

実行結果の例

```
-1/15*pi^4 + limit(-1/4*x^4 + x^3*log(-e^x + 1) + 3*x^2*dilog(e^x) - 6*x*polylog(3, e^x)
+ 6*polylog(4, e^x), x, +Infinity, minus)
```

これでは積分値が求まったとは言えませんが，積分を実行する際に，オプションをつける事で，正しい積分値が求まります。

```
f = x^3/(e^x-1)          # 関数fを定義
integrate(f,x,0,oo,'giac')    # オプション'giac'を付す。
```

実行結果の例

```
1/15*pi^4
```

Sage が積分を計算する際に利用するアルゴリズムをオプションとして指定することができます。

- 'maxima'：デフォルトなので，何も指定しないとこれになる
- 'sympy'：sympy を使う。Sage に含まれている。
- 'mathematica_free'：Wolfram alpha を使う。
- 'fricas'：FriCAS を使う。別途インストールしてある場合に利用可能。
- 'giac'：Giac を使う。

43.2 数値積分

integrate で不定積分も定積分も求まらないような関数はたくさんあります。そのような場合でも数値積分の命令 numerical_integral を使うことで，定積分の近似値を得ることができます。

Sage による数値積分は次の命令で行います：

関数 $f(x)$ の数値積分 $\int_a^b f(x)dx$

```
numerical_integral(f(x),a,b)
```

結果として (値，誤差) が返されます。オプション max_points により積分のサンプル点の最大個数を指定することができる。

例：

```
sage: numerical_integral(sin(x),0,pi)
(1.9999999999999998, 2.220446049250313e-14)
```

上の積分は厳密には 2 となるはずですが，微少の誤差が出ています。実行結果の二つ目は誤差を表していて，ここでは積分値はおよそ $2 * 10^{-14}$ の誤差が出ていることを表しています。ただし，これは厳密な誤差ではなくアルゴリズムから予測される真の値との誤差です。数値だけほしい場合には，実行結果の第 0 成分だけ取り出せばよいので，次のようにします：

```
sage: val = numerical_integral(sin(x),0,pi)
sage: val[0]
1.9999999999999998
```

ただし，数値積分の結果が信頼できるものであるかどうかは被積分関数の特性によります。振動が激しい関数や，積分が緩やかに発散する関数の場合などは，真の値と大幅に違った値になってしまうこともあります。

数値積分 numerical_integral は積分区間を分割して計算していますが，そのとき区間の分割が十分でないと誤差が大きくなります。サンプル点の最大個数はデフォルトでは 87 個ですが，オプションで変更可能です。例えば，次の数値積分は誤差が 0.0001 ほどありますが，サンプル点の数を多くして誤差を小さくすることができます：

```
sage: numerical_integral(sin(1/x),0,1)
(0.5040702199680797, 0.00012692441400448108)
sage: numerical_integral(sin(1/x),0,1,max_points=1000)
(0.5040670267347616, 1.4633387576762725e-05)
```

numerical_integral は GNU Scientific Library(GSL) という数値計算用の C のアルゴリズムを用いていますが，maxima という数式処理ソフトを用いて数値積分を行う命令があります。こちらでは，要求する精度を指定して積分を計算することができます。

$\int_a^b f(x)dx$ の数値積分 (maxima を使用)

```
f.nintegral(x,a,b,精度)
```

実行結果は（値，誤差，近似する区間の数，エラーコード）となります。

エラーコードは 0 から 6 まであり

- 0：エラーなし
- 1：分割する区間が多すぎる
- 2：丸め誤差が大きすぎる
- 3：被積分関数の振る舞いが最悪
- 4：収束しない
- 5：積分はおそらく発散するもしくはゆっくり収束する
- 6：入力が正しくない

を意味します。次の例を実行してみましょう：

```
sage: f=sin(1/x);    f.nintegral(x,0,1)
(0.50411061321101469, 3.4994456304171528e-05, 8379, 1)
```

積分値は以前の計算とは 5 桁めが異なります。被積分関数の振動が激しいのでエラーメッセージ 1 がでています。つぎに精度を 1/100 として計算してみます：

```
sage: f=sin(1/x);      f.nintegral(x,0,1,1/100)    # 精度は1/100と指定
(0.50279836327622895, 0.0046840581358000843, 567, 0)
```

最後の出力のエラーコードは 0 なので 0.01 の精度で確からしい値が得られたことがわかります。

44　Taylor 展開

44.1　Taylor 展開

f を適当な回数だけ微分可能な関数とする。f の Taylor 展開 (または Taylor 級数) とは

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-1) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \cdots$$

のことです。特に $a = 0$ の場合を Maclaurin 展開といいます。Taylor 展開は次のように求めます：

Taylor 級数： $f(a) + \frac{f'(a)}{1!}(x-1) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n$

関数 f(x) の x=a の周りの x についての n 次までの Taylor 級数を返す：

```
taylor(f(x),x,a,n)
```

例えば，e^x の $x = 0$ のまわりの 3 次までの Taylor 級数は

```
taylor(e^x,x,0,3)
```

実行結果の例

```
1/6*x^3 + 1/2*x^2 + x + 1
```

で求めます。2 変数関数 $f(x,y)$ の Taylor 展開は

2 変数関数 $f(x,y)$ の点 (a,b) のまわりの n 次までの Taylor 級数

```
taylor(f(x,y),(x,a),(y,b),n)
```

です。例えば,

```
var('y')
taylor(sin(x+y),(x,0),(y,0),4)
```

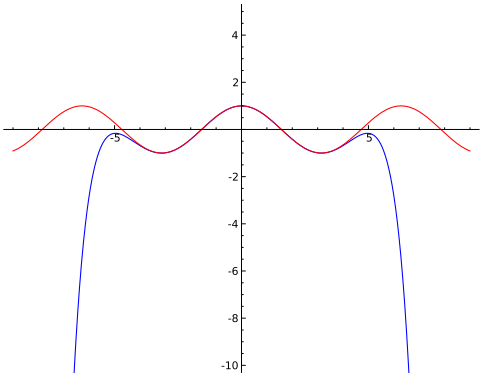
の実行結果は次のようになります：

$$-\frac{1}{6}x^3-\frac{1}{2}x^2y-\frac{1}{2}xy^2-\frac{1}{6}y^3+x+y$$

44.2　応用編：Taylor 展開で近似される様子を描画

応用編として，$\cos(x)$ とその Taylor 展開の x^{10} 次までの多項式のグラフを重ねて描いてみましょう：

```
f = taylor(cos(x),x,0,10)                    # fはcos(x)の10次までの展開
p1 = plot(f,(x,-9,9))                        # fのグラフ
p2 = plot(cos(x),(x,-9,9), color='red')      # cos(x)のグラフ
p = p1 + p2;                                 # 二つのグラフを重ねたものをpとする
p.show(ymin=-10,ymax=5)                      # pを描画
```



45　練習問題

以下の問題の答えを Sage を使って求めてみましょう。

次の極限を求めよ。

$$(1)\quad \lim_{h\to 0}\frac{(x+h)^3-x^3}{h}\qquad (2)\quad \lim_{x\to\infty}\left(\frac{x^x}{x!}\right)^{1/x}\qquad (3)\quad \lim_{x\to\infty}\arctan(x)$$

次の式を部分分数展開せよ。ただし $x!$ は `factorial(x)`。

$$(4)\quad \frac{3x-37}{(x+1)(x-4)}\qquad (5)\quad \frac{9-9x}{2x^2+7x-4}\qquad (6)\quad \frac{4x^2}{(x-1)(x-2)^2}\qquad (7)\quad \frac{8x^2-12}{x(x^2+2x-6)}$$

次の数の 11 次までの連分数展開を求めよ。

(8)　黄金比 = golden_ratio　　(9)　円周率 = pi

次の関数を x で微分せよ。

(10) $\sin(cx)$（c は定数）
(11) $|x|/(1+x^2)$(絶対値 $|x|$ は abs(x) で表します。(absolute value の略))

次の x についての関数の原始関数を求めよ。

(12) $\dfrac{1}{a+x}$　　(13) a^x　　(14) $\dfrac{1}{\sqrt{a^2+x^2}}$

次の関数の右に書かれている区間についての定積分を求めよ。

(15) x^2，区間 $[0,1]$
(16) $\sin(x)$, 区間 $[0,\pi]$
(17) $x\cos(x)$，区間 $[0,\pi]$

次の関数について，$x=0$ のまわりで 5 次までのテイラー級数をもとめよ。

(18) x^2e^x
(19) $\sin(x)/(1+x^2)$

上の問題の答えをすべて出力する Sage プログラムを作成しましょう。ファイル名は answer.sage とすること。提出するプログラムはコマンドラインから sage answer.sage と実行したときに問題番号とその解答を表示 (print) するようにせよ。例えば，最初の問題の答えは次のようになる：

- ファイル名：answer.sage

```
1 # -*- coding: utf-8 -*-
2 print '(1)',
3 var('h')
4 f1 = ((x+h)^3-x^3)/h
5 print limit(f1,h=0)
```