附录A SageMath 常用函数

一、算术函数

1.1 基本运算

```
In:  1+2,2-3,3*4,5/4,RDF(5/3),5//4,5%4,-3%2,2^3,2**3,floor(4/3),ceil(4/3)
     ##基本的加法、减法、乘法、保留精度除法、近似除法、整除、模运算、幂运算、下底、上底等运算，RDF == RealDoubleField 通常用来在近似计算中将表达式变为实数，损失一定精度，但可提高计算效率。
Out: (3, -1, 12, 5/4, 1.6666666666666667, 1, 1, 1, 8, 8, 1, 2)
```

1.2 最大公因数

```
In:  gcd(123,36)
Out: 3
In:  gcd([25,10,-5])  ##3 个以上整数求最大公因数，输入要用 list
Out: 5
In:  gcd(8/15,20/27)  ##分子的最大公因数/分母的最小公倍数
Out: 4/135
```

1.3 扩展欧几里得算法，计算 sa+tb=gcd(a,b)

```
In:  g,s,t=xgcd(56, 44);g,s,t  ##4=4*56+(-5)*44
     ##计算 gcd(a,b)以及 a,b,使得 sa+tb=gcd(a,b)。
Out: (4, 4, -5)
```

1.4 最小公倍数

```
In:  lcm(-10,25)
Out: 50
In:  lcm([2,10,30])   ##3 个以上整数求最小公倍数，输入要用 list
Out: 30
In:  lcm(8/15,20/27)  ##分子的最小公倍数/分母的最大公因数
Out: 40/3
```

1.5 模幂运算

```
In:  power_mod(2,390,391)  ##2^390(mod 391), 计算a^e(mod N)。
Out: 285
In:  power_mod(2,-1,7)     ##还可以求逆 2^-1(mod 7)
Out: 4
```

1.6 模逆运算

In： inverse_mod(-5,14) ##求逆$(-5)^{-1}$(mod 14)
Out：11
In： timeit('inverse_mod(498923849032809432,333333333333333333333331)')
##timeit 计算函数运算时间，求逆时，inverse_mod 效率比 power_mod 略高。
Out：625 loops, best of 3: 1.23 µs per loop
In： timeit('power_mod(498923849032809432,-1,333333333333333333333331)')
Out：625 loops, best of 3: 4.2 µs per loop

1.7 中国剩余定理

In： crt(2, 1, 3, 5) ##解同余式组 x=2(mod3) and x=1(mod5)
Out：11
In： crt([2, 1, 0], [3, 5, 22]) ##3 个以上同余式，输入要用 list
Out：176

1.8 素数相关函数

In： is_prime(19),is_prime(22343) ##判断 n 是否素数
Out：(True, True)
In： nth_prime(1),nth_prime(2),nth_prime(1000) ##第 n 个素数
Out：(2, 3, 7919)
In： next_prime(2),next_prime(10) ##大于 n 的最小素数
Out：(3, 11)
In： previous_prime(10) ##小于 n 的最大素数
Out：7
In： primes_first_n(10) ##前 n 个素数
Out：[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
In： list(primes(2,17)) ##区间内的素数
Out：[2, 3, 5, 7, 11, 13]
In： euler_phi(20) ##欧拉函数
Out：8

1.9 整数分解

In： factor(-100) ##分解整数
Out：-1 * 2^2 * 5^2
In： ecm(1234567) ##椭圆曲线分解寻找素因子
Out：'GMP-ECM 7.0.4 [configured with MPIR 3.0.0, --enable-asm-redc]
[ECM]Input number is 1234567 (7 digits)
Using B1=10, B2=84, polynomial x^1, sigma=1:4277771191
Step 1 took 0ms
Step 2 took 0ms
********** Factor found in step 2: 1234567

```
In:   qsieve(12345678901234567890123456789012345678902)
      ##二次筛法寻找因子
Out:  ([2,
        22,
        6418,
        70598,
        349745854025172608009389976741900498,
        3847204394276898688103289744160905478,
        1122334445566778899102132435364758698082],
       '')
In:   f=factor(-100);list(f),f.value()    ##获取整数素因子和相应次数
Out:  ([(2, 2), (5, 2)], -100)
In:   prime_divisors(-100)                ##仅列出整数的素因子
Out:  [2, 5]
```

1.10 二次剩余函数

```
In:   legendre_symbol(2,37)    ##Legendre 符号
Out:  -1
In:   jacobi_symbol(2,37)      ##Jacobi 符号
Out:  -1
In:   primitive_root(11),primitive_root(22)    ##计算整数的原根
Out:  (2, 13)
```

二、代数系统

2.1 p元有限域

```
In:   k1=GF(7)    ##定义 k1 为元素个数为 7 的有限域
In:   a=k1(5);print k1.characteristic(),a^-1,a^10+1,a.log(3)
      ##将 a 设为有限域 k1 中的 5，以后与 a 相关的运算均在 k1 中进行，其中 a.log(3)是求以 3 为底，5 的离散对数
Out:  7 3 3 5
In:   k1(2).nth_root(5),k1(2).sqrt()
      ##2 的 5 次方根 x^5=2(mod7) ，2 的平方根，x^2=2(mod7)
Out:  (4, 3)
In:   k1.modulus(),k1.gen()
      ##模多项式及其根，缺省为 x-1,此时 gen 返回 x-1 的根 1
Out:  (x + 6, 1)
In:   k2=GF(7,modulus='primitive');k2.modulus(),k2.gen()
      ##一个本元多项式及其根,此时 gen 函数返回一个本原元，即本原多项式的一个根
Out:  (x + 4, 3)
In:   s=a.lift();print s,type(s),type(a)
      ##将域中的元素提升回整数，以后 s 的运算就按整数进行，而不是 k1 中进行
```

```
Out: 5
     <type 'sage.rings.integer.Integer'>
     <type 'sage.rings.finite_rings.integer_mod.IntegerMod_int'>
```

2.2 p^n元有限域

```
In:  K1.<x>=GF(7^3,modulus='primitive')
     ##定义 K1 为元素个数为 7^3 的有限域,模为本原多项式,x 为模多项式的一个根
In:  K1.modulus(),K1.gen(),K1.order(),x.multiplicative_order()
     ##分别输出 K1 的本原多项式，多项式的根，K1 的元素个数，元素 x 的乘法阶
     ##x 是本原元，乘法阶等于 7^3-1=342
Out: (x^3 + 6*x^2 + 4, x, 343, 342)
In:  x^100   ##在有限域 K1 中计算 x^100
Out: 2*x^2 + x + 4
In:  K2.<a>=GF(7^3,modulus=[1,0,2,1])
     ##定义 K2 为元素个数为 7^3 的有限域,模为指定不可约多项式 x^3 + 2*x^2 + 1,
     a 为模多项式的一个根
In:  K2.modulus(),K2.gen(),K2.order(),a.multiplicative_order()
     ##分别输出 K2 的模多项式，多项式的根，K2 的元素个数，元素 a 的乘法阶
     ##多项式用 list 表示时，系数从低到高，a 不是本原元，阶不等于 342
Out: (x^3 + 2*x^2 + 1, a, 343, 38)
In:  f=a^100;print (f,f.minimal_polynomial())
     ##计算 a^100 与其极小多项式
Out: (2*a^2 + 5*a + 4, x^3 + 4*x^2 + 4*x + 6)
In:  f.minimal_polynomial()[0]
     ##多项式的系数可以通过 list 的方式取得
Out: 6
In:  f.polynomial(),f.polynomial()[0]
     ##如果要获取有限域中元素 f 的系数，可以将其先转化成多项式
Out: (2*a^2 + 5*a + 4, 4)
In:  K3=GF(7^3,'c')   ##上述有限域的另外一种写法
In:  K3.modulus(),K3.gen(),K3.order()
Out: (x^3 + 6*x^2 + 4, c, 343)
In:  c^100
     ##这种表示方法中，c 不能直接用作域中的元素
Out: NameError
     Traceback (most recent call last)
     <ipython-input-47-e83006da3f2e> in <module>()----> 1 c**Integer(100)
     NameError: name 'c' is not defined
In:  t=K3.gen();t^100
     ##可以先赋值给变量 t，再进行运算
Out: 2*c^2 + c + 4
```

2.3 整数环

In: r=Zmod(26) ##定义 r 为环$\mathbb{Z}_{26}$
In: r=Integers(26) ##以上 Zmod(26)的等价写法
In: a=r(9);b=r(8);a*b ##变量会自动按照 mod(26)进行运算
Out: 20
In: a^-1 ,a.log(7),a.is_square(),a.sqrt()
    ##求逆,求离散对数（如果存在）,判断是否平方元，求平方根
Out: (3, 4, True, 3)
In: a.multiplicative_order(),a.additive_order()
    ##求 a 的乘法阶和加法阶
Out: (3, 26)
In: a.minimal_polynomial() ##求 a 的极小多项式
Out: x + 17
In: mod(9,26) in r,mod(9,26).sqrt()
    ##Zmod 中元素的简写，这样可以不用事先定义 r=Zmod(26)
Out: (True, 3)
In: type(a),type(mod(9,26)),type(mod(9,26).lift())
    ##Zmod 中的元素不是 Integer，需要 lift 才能当作正常的 Integer 对待
Out: (<type 'sage.rings.finite_rings.integer_mod.IntegerMod_int'>,
     <type 'sage.rings.finite_rings.integer_mod.IntegerMod_int'>,
     <type 'sage.rings.integer.Integer'>)

2.4 多项式环

一元多项式环

```
In:  R.<x>=ZZ[]              ##定义 R 为整系数一元多项式环,文字为 x
In:  f=4*x^2+4*x+1           ##定义 f 为整系数一元多项式
In:  print(f.is_irreducible(),f.factor())  ##判断一元多项式是否可约
Out: (False, (2*x + 1)^2)
In:  print(f.factor_mod(2))    ##模素数 2 分解多项式
Out: 1
In:  print f.roots()           ##查找多项式的整数解
Out: []
In:  g=3*x^2+2*x+1; s=R([1,2,3]); print (g,s,g==s)
     ##可以用 list 定义多项式，其优点是可导入大量系数
Out: (3*x^2 + 2*x + 1, 3*x^2 + 2*x + 1, True)
```

多元多项式环

```
In:  R.<x,y>=ZZ[]             ##定义 R 为整系数二元多项式环,文字为 x,y
In:  f=x*y*(x^2+2*y^2+21)    ##定义 f 为 R 中的一个多项式
In:  print f+x^2+x+y          ##多项式运算
Out: x^3*y + 2*x*y^3 + x^2 + 21*x*y + x + y
In:  print f(1,1/2)           ##多项式的值
Out: 45/4
```

In：print f.factor()　　　##多项式的因式分解
Out：y * x * (x^2 + 2*y^2 + 21)
In：print f[1,3]　　　　　##多项式 f 中 xy^3 的系数
Out：2

环上的多项式

In：R.<x>=Zmod(26)[]　　　　##定义 R 为系数在环$\mathbb{Z}_{26}$上的一元多项式类型,文字为 x
In：f=(x+1)*(x^3+x+1)　　　##定义环 R 上的一个多项式 f
In：print (f,f.degree(),f(7),f^2,f.gcd(x+1),f.xgcd(x+1),f.quo_rem(3*x - 1),f%(3*x - 1),f//(3*x - 1))
##多项式常用运算，多项式的次数，多项式的值，幂运算，最高公因式，扩展欧几里得除法，带余除法，模运算，整除
Out：(x^4 + x^3 + x^2 + 2*x + 1, 4, 0, x^8 + 2*x^7 + 3*x^6 + 6*x^5 + 7*x^4 + 6*x^3 + 6*x^2 + 4*x + 1, x + 1, (x + 1, 0, 1), (9*x^3 + 12*x^2 + 13*x + 5, 6), 6, 9*x^3 + 12*x^2 + 13*x + 5)
In：print f.roots(multiplicities=False)　##多项式求根,不考虑重数
Out：[7, 25]
In：print (power_mod(f,2,x^2+1),inverse_mod(f,x^2+x+1),gcd(f,x+1),lcm(f,x+1))
##多项式的模幂，模逆,最高公因式和最低公倍式
Out：(2*x, 25*x, x + 1, x^4 + x^3 + x^2 + 2*x + 1)

域上的多项式

In：R.<x>=Zmod(13)[]　　　　##定义 R 为系数在环$\mathbb{Z}_{13}$上的一元多项式类型,文字为 x
In：f=(x+1)*(x^3+x+1)　　　##定义环 R 上的一个多项式 f
In：print (f,f.degree(),f(7),f^2,f.gcd(x+1),f.xgcd(x+1),f.quo_rem(3*x - 1),f%(3*x - 1),f//(3*x - 1))
##多项式常用运算，多项式的次数，多项式的值，幂运算，最高公因式，扩展欧几里得除法，带余除法，模运算，整除
Out：(x^4 + x^3 + x^2 + 2*x + 1, 4, 0, x^8 + 2*x^7 + 3*x^6 + 6*x^5 + 7*x^4 + 6*x^3 + 6*x^2 + 4*x + 1, x + 1, (x + 1, 0, 1), (9*x^3 + 12*x^2 + 5, 6), 6, 9*x^3 + 12*x^2 + 5)
In：print (f.is_irreducible(),f.factor())
##判断多项式是否可约，多项式因式分解，当前仅支持模为素数的情况
Out：(False, (x + 1) * (x + 6) * (x^2 + 7*x + 11))
In：print f.roots()　##多项式求根，考虑重数
Out：[(12, 1), (7, 1)]
In：R.<x>=GF(13)[]　　　　##和 Zmod(13)的定义等价
In：f=(x+1)*(x^3+x+1)
In：print (f,f.degree(),f(7),f^2,f.gcd(x+1),f.xgcd(x+1),f.quo_rem(3*x - 1),f%(3*x - 1),f//(3*x - 1))
Out：(x^4 + x^3 + x^2 + 2*x + 1, 4, 0, x^8 + 2*x^7 + 3*x^6 + 6*x^5 + 7*x^4 + 6*x^3 + 6*x^2 + 4*x + 1, x + 1, (x + 1, 0, 1), (9*x^3 + 12*x^2 + 5, 6), 6, 9*x^3 + 12*x^2 + 5)

In: print (f.is_irreducible(),f.factor(),list(f.factor()))
Out: (False, (x + 1) * (x + 6) * (x^2 + 7*x + 11), [(x + 1, 1), (x + 6, 1), (x^2 + 7*x + 11, 1)])
In: print f.roots(multiplicities=False)   ##多项式求根，不考虑重数
Out: [12, 7]

多项式更换系数环

In: R.<x> = QQ[] ##定义 R 为有理数域上的一元多项式环，文字为 x
In: f=x^2+1
In: print f.is_irreducible() ##f 在有理数域上不可约
Out: True
In: print f.change_ring(CC).roots() ##f 在复数域上可约
Out: [(-1.00000000000000*I, 1), (1.00000000000000*I, 1)]
In: print f.is_irreducible() ##由于 f 未重新赋值，所以还是有理数域上的多项式
Out: True
In: g=f.change_ring(GF(2))   ##将 f 的系数环更换为 GF(2)，并赋值给 g
In: print (g,g.factor(),g(5)) #g 是 GF(2)上的环
Out: (x^2 + 1, (x + 1)^2, 0)
In: g=g.change_ring(QQ)   ##将 g 的系数环更换为有理数域
In: print g(5)
Out: 26

三、矩阵操作

3.1 矩阵定义

In: mt=matrix(ZZ,3,3)   ##定义整数环上的 3 行 3 列的矩阵，初始值全部为 0
In: mt=matrix(ZZ,0,3)   ##定义整数环上的 0 行 3 列的矩阵，以后动态添加行
In: mt=mt.stack(vector([1,2,3])) ##矩阵后面添加一行
In: mt=mt.stack(vector([4,5,6]))
In: mt=mt.insert_row(1,vector([7,8,9])) ##在指定行号插入一行
In: mt[2,2]=10   ##可以直接修改矩阵中的元素
In: print (mt,mt.rank(),mt.is_invertible(),mt.nrows(),
     mt.ncols(),mt.determinant())
     ##is_invertible()是指 ZZ 上的逆矩阵
Out: ([1 2 3]
     [7 8 9]
     [4 5 10], 3, False, 3, 3, -24)
In: mt=matrix(QQ,3,[1,2,3,4,5,10,7,8,9])   #定义 QQ 上的 3 列的矩阵,并赋值
In: print (mt,mt.rank(),mt.is_invertible(),mt.nrows(),
     mt.ncols(),mt.determinant())
     ##is_invertible()是指 QQ 上的逆矩阵
Out: ([1 2 3]

```
     [ 4  5 10]
     [ 7  8  9], 3, True, 3, 3, 24)
In:  mt=mt.change_ring(ZZ)          #矩阵可以 change_ring
In:  print (mt,mt.rank(),mt.is_invertible(),mt.nrows(),
     mt.ncols(),mt.determinant())
     ##is_invertible()是指 ZZ 上的逆矩阵
Out: ([ 1  2  3]
     [ 4  5 10]
     [ 7  8  9], 3, False, 3, 3, 24)
```

3.2 求解线性方程组和线性相关组

```
In:  A = matrix(QQ,4,2, [0, -1, 1, 0, -2, 2, 1, 0])  ##4 行，2 列
In:  B = matrix(QQ,2,2, [1, 0, 1, -1])                ##2 行，2 列
In:  X = A.solve_left(B)          ##求矩阵 X 满足 XA=B
In:  print(X,X*A == B)            ##X，2 行，4 列
Out: ([0 1 0 0]
     [1 1 0 0], True)
In:  A = matrix(QQ, 3, [1,2,3,-1,2,5,2,3,1])
In:  b = vector(QQ,[1,2,3])
In:  x = A \ b                    ##x 满足 Ax=b
In:  y = A.solve_right(b)         ##等价于上面的写法
In:  print x,y
Out: (-13/12, 23/12, -7/12) (-13/12, 23/12, -7/12)
In:  A = matrix(QQ,4,2, [0, -1, 1, 0, -2, 2, 1, 0]);print A    ##4 行，2 列
Out: [ 0 -1]
     [ 1  0]
     [-2  2]
     [ 1  0]
In:  X = A.left_kernel()
     ##X 满足 XA=0 ,用于找线性相关的行向量
In:  print X
Out: Vector space of degree 4 and dimension 2 over Rational Field
     Basis matrix:
     [  1   0 1/2   1]
     [  0   1   0  -1]
     ##即[ 0 -1]+1/2*[-2  2]+ [ 1  0]= [ 0  0], [ 1  0]- [ 1  0]= [ 0  0]
In:  print A.linear_combination_of_rows(X.gen(0))
     ##X.gen(0)即[  1   0 1/2   1]
     ##计算 X.gen(0)*A
Out: (0, 0)
In:  print A.linear_combination_of_rows(X.gen(1))
     ##X.gen(1)即[  0   1   0  -1]
     ##计算 X.gen(1)*A
```

```
Out: (0,  0)
In:  A = matrix(QQ,3,[1,2,3,4,5,6,7,8,9])  ##3 行，3 列
In:  X = A.right_kernel()                  ##X 满足 AX=0，用于找线性相关列向量
In:  print(X)
Out: Vector space of degree 3 and dimension 1 over Rational Field
     Basis matrix:
     [ 1 -2  1]
```

3.3 LLL 格基约简算法

```
In:  A=matrix(4, 8, [1, 0, 0, 0, 1, 1, 1,7,0, 1, 0, 0, 0, 0, 1, 1,0, 0, 1, 0, 0,
     1, 1, 1,0, 0, 0, 1, 1, 0, 0, 0]);A        ##4 行，8 列
Out: [1 0 0 0 1 1 1 7]
     [0 1 0 0 0 0 1 1]
     [0 0 1 0 0 1 1 1]
     [0 0 0 1 1 0 0 0]
In:  A.LLL()
     ##执行 LLL 算法，寻找最短正交基，输入按行向量计算
Out: [ 0  0  0  1  1  0  0  0]
     [ 0  1  0  0  0  0  1  1]
     [ 0 -1  1  0  0  1  0  0]
     [ 1 -2 -1  0  1  0 -2  4]
```