

SageMath简明教程

Author: [tl2cents](#)

目录

- 1 序言
- 2 格理论基础
 - 2.1 基本数论
 - 2.2 格理论
 - 2.2.1 格的定义
 - 2.2.2 最短向量问题 (Shortest Vector Problem: SVP)
 - 2.2.3 最近向量问题 (Closest Vector Problem: CVP)
- 3 SageMath简明教程
 - 3.1 sage安装与环境配置
 - 3.2 sage基本函数使用
 - 3.2.1 sage与python的不同
 - 3.2.2 基本的环、域
 - 3.2.3 基本数论函数
 - 3.2.4 线性代数相关函数
 - 3.2.5 多项式环及其函数
 - 3.2.6 格相关函数*

1 序言

笔者在编写USTC信息安全实践课程的二级密码学讲义中，简要编写了SageMath的入门教程，主要针对CTF的CRYPTO类赛题的脚本的撰写。时间仓促，难免有所纰漏。后续会不定期更新该教程。

已同步更新至Github仓库：<https://github.com/tl2cents/SageMath-Concise-Tutorial>。

2 格理论基础

SageMath (System for Algebra and Geometry Experimentation)，是一个覆盖许多数学功能的应用软件，包括代数、组合数学、图论、计算数学、数论、微积分和统计。尤其是数论上的许多算法，sagemath都集成了其实现接口。sagemath本质上是以python编程语言为基础的数学应用软件，因此需要有一定的python基础和数论理论基础。

在介绍sagemath的使用之前，我们先简要回顾一下基本数论以及介绍格理论的相关内容。

2.1 基本数论

SageMath需要的基本数论理论和线性代数如下，为前置内容，此处不赘述：

- 有限域 $\mathbb{Z}_{mod}(N)$
- 有限域的阶、循环群
- 模多项式环
- 离散对数
- 矩阵的秩
- 行列式
- 线性子空间（线性相关）

2.2 格理论

2.2.1 格的定义

算法数论中常常遇到这样一个问题：它有连续与离散两种成分，一个典型的例子就是满足某些不等式的整数系之寻求，具有这个性质的问题通常可以用格的算法理论来成功求解。标准地来说：一个格是欧式向量空间的一个离散子群。

回忆线性代数的基本内容：欧式向量空间是实数域 \mathbb{R} 上的一个有限维向量空间 E ，并且定义了内积。实数的 n 元组的全体构成了 \mathbb{R} 上一个 n 维空间，实数坐标空间 \mathbb{R}^n 就能认为是一个 n 维欧式空间。简单地说，二维欧式空间就是：

$$E^2 = \{(x, y) | x \in \mathbb{R}, y \in \mathbb{R}\}$$

在线性代数里实数域上对 n 个 d 维线性无关向量 $\{b_i \in \mathbb{R}^d, i = 1, 2, \dots, n\}$ 的线性组合（线性子空间）可以表示为：

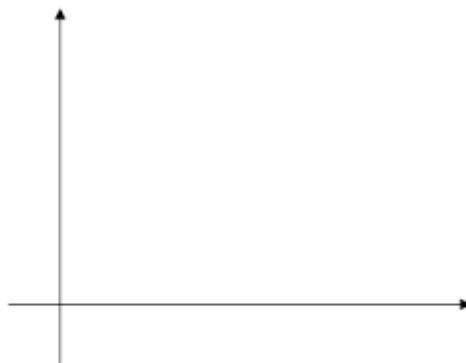
$$L_R = \sum_{i=1}^n \mathbb{R}b_i = \left\{ \sum_{i=1}^n c_i b_i : c_i \in \mathbb{R}, i = 1, 2, \dots, n \right\}$$

而格的定义则将系数 c_i 是转到了整数域 \mathbb{Z} 上:

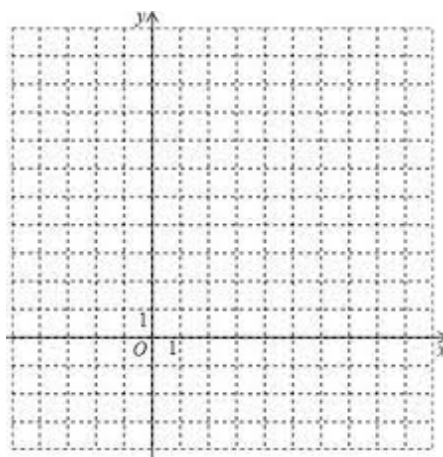
$$L_R = \sum_{i=1}^n \mathbb{Z}b_i = \left\{ \sum_{i=1}^n c_i b_i : c_i \in \mathbb{Z}, i = 1, 2, \dots, n \right\}$$

在此基础上我们给出 $(1, 0), (0, 1)$ 张成的线性子空间和二维格的形象化表示:

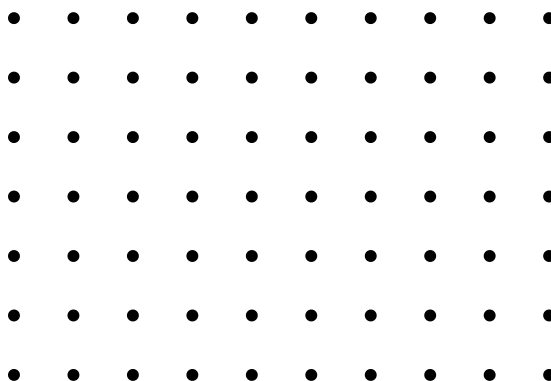
- 二维子空间，二维全平面:



- 二维格:



也就是上图方格中所有的格点，这也是为什么我们称这个空间为格空间，如下图



本教程中不涉及复杂的格理论，对于格，在CTF中最常用的就是形式是把它表示为矩阵形式，考虑一个 \mathbb{Z} 上的 $n * n$ 矩阵：

$$M = \begin{bmatrix} m_{11} & m_{12} & \dots & m_{1n} \\ m_{21} & m_{22} & \dots & m_{2n} \\ \dots & \dots & \dots & \dots \\ m_{n1} & m_{n2} & \dots & m_{nn} \end{bmatrix} = [m_1, m_2, \dots, m_n]^T$$

where $m_{ij} \in \mathbb{Z}$ and m_i is M 's row vector

$$L = \mathbb{Z}_{\text{span}}\{m_1, m_2, \dots, m_n\}$$

CTF中许多模方程的问题都可以转换到M的行向量 $[m_1, m_2, \dots, m_n]$ 所张成的n维格空间L中进行求解。

2.2.2 最短向量问题 (Shortest Vector Problem: SVP)

格中的最短向量问题，亦即齐次逼近问题，规范式的描述是：

- SVP问题

给定一个正秩格 L ，模长映射为 $q: L \rightarrow \mathbb{R}$ ，寻求一个非零元素 $x \in L$ 使得模长 $q(x)$ 最小（尽可能地小）。

利用上面矩阵表示的形式来阐述就是，求 $x \in L$ ，模长尽可能地小，即：

$$\min \|x\|^2 \quad s.t. \quad x = \sum_{i=1}^n c_i * \vec{m}_i \quad \text{where } c_i \in \mathbb{Z}$$

关于格的最小向量问题的最主要的理论成果是Minkowski定理

- Minkowski定理

每一个正秩n维格L皆包含一个非零元素x满足：

$$q(x) \leq \frac{4}{\pi} * \frac{n}{2}!^{\frac{2}{n}} * d(L)^{\frac{2}{n}} \leq n * d(L)^{\frac{2}{n}}$$

其中 $d(L)$ 是格L对应矩阵的行列式。

然而，我们需要注意Minkowski定理是非构造性的，也就是我们知道其存在性，但是并没有有效的算法可以得到上述x。为了解决格中的最短向量问题，我们介绍一个经典算法：LLL算法。

- LLL算法

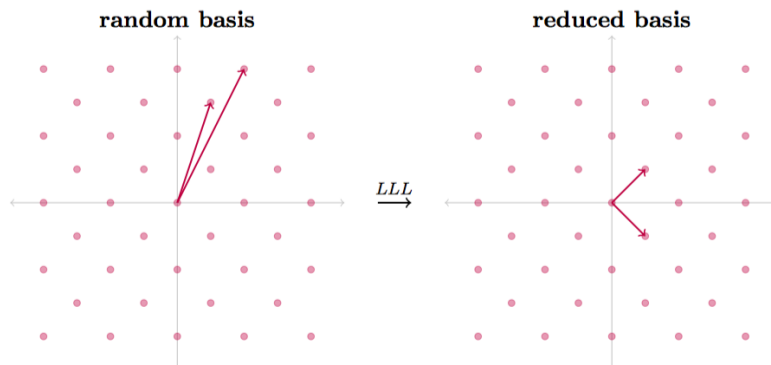
在格上找到一组基，满足如下效果

Definition 1. Let L be a lattice with a basis B . The δ -LLL algorithm applied on L 's basis B produces a new basis of $L : B' = \{b_1, \dots, b_n\}$ satisfying :

$$\forall 1 \leq j < i \leq n \text{ we have } |\mu_{i,j}| \leq \frac{1}{2} \quad (1)$$

$$\forall 1 \leq i < n \text{ we have } \delta \cdot \|\tilde{b}_i\|^2 \leq \|\mu_{i+1,i} \cdot \tilde{b}_i + \tilde{b}_{i+1}\|^2 \quad (2)$$

with $\mu_{i,j} = \frac{b_i \cdot \tilde{b}_j}{\tilde{b}_j \cdot \tilde{b}_j}$ and $\tilde{b}_1 = b_1$ (Gram-Schmidt)



并且这组基具有如下性质:

Property 1. Let L be a lattice of dimension n . In polynomial time, the LLL algorithm outputs reduced basis vectors v_i , for $1 \leq i \leq n$, satisfying :

$$\|v_1\| \leq \|v_2\| \leq \dots \leq \|v_i\| \leq 2^{\frac{n(n-1)}{4(n+1-i)}} \cdot \det(L)^{\frac{1}{n+1-i}}$$

LLL算法很好的一个性质是可以在多项式时间内找到符合上述条件的一组基，并且在sagemath上有实现接口。另外一个效果可能更好、参数设置过大时间复杂度达到指数级别的SVP求解算法是BKZ算法，多数情况下利用LLL算法已经足够，这里不再详细介绍。

2.2.3 最近向量问题 (Closest Vector Problem: CVP)

格的最近向量问题或称非齐次逼近问题规范化定义如下:

- CVP问题

在一个欧式向量空间 E 中给定一个格 L ，以及一个元素 $x \in E$ ，寻求 $y \in L$ 具有最小的可能距离 $d(x, y)$ 。

利用上面矩阵表示的形式来阐述就是，求 $y \in L$ ，模长尽可能地小，即：

$$\min \|y - x\|^2 \quad s. t. \quad y = \sum_{i=1}^n c_i * \vec{m}_i \quad \text{where } c_i \in \mathbb{Z}$$

当然我们还可以在CVP上增加约束：

给定一个Lattice L ，与一个随机点 t ，还有搜索距离 d ，并且假设 $\mu(t, L) \leq d$ ，CVP问题是让我们找到一个合理的格点 $B * x \in L$ 并且这个点到 t 的距离小于等于 d 。

简单来说就寻找格 L 中距离非格点 x 最近的一个格点 y ，我们不过多介绍理论部分，这里给出一个CVP问题求解的一个常用算法Babai's nearest plane algorithm:

- Babai算法

输入秩为 n 的格 L 的一组基 B 和一个目标向量 t ，输出CVP问题的近似解

INPUT: Basis $B \in \mathbb{Z}^{m \times n}$, $t \in \mathbb{Z}^m$

OUTPUT: A vector $x \in \mathcal{L}(B)$ such that $\|x - t\| \leq 2^{\frac{n}{2}} \text{dist}(t, \mathcal{L}(B))$

1. Run δ -LLL on B with $\delta = \frac{3}{4}$

2. $b \leftarrow t$

for $j = n$ to 1 **do**

$b \leftarrow b - c_j \tilde{b}_j$ where $c_j = \lceil \langle b, \tilde{b}_j \rangle / \langle \tilde{b}_j, \tilde{b}_j \rangle \rceil$

 Output $t - b$

从算法可以看出，Babai是以LLL算法为基础的，因为Baiba算法需要在性质较好的规约基上运行。

3 SageMath简明教程

SageMath 是一个基于GPL协议的开源数学软件。它使用Python作为通用接口，将现有的许多开源软件包整合在一起，构建一个统一的计算平台。

对于sagemath，将它理解为python在数学应用上的一个扩展应用程序即可，在你掌握了python的基本语法后，sagemath相当于多了许多自带扩展库，熟悉这些库的基本应用，足以应对大多数CTF的密码学赛题。




sagemath学习比较大的一个问题在于缺失比较好的中文文档，许多时候我们难以找到sage中许多现成的框架，主要原因在于本地安装好sage查看源码的功能也比较麻烦，甚至不如直接去sage的Github仓库找函数。特别是目前sage的两个函数 `search_doc` 和 `search_src` 使用起来尤为不便，你可以在sage的console或者notebook输入 `search_doc("matrix")`，返回元素冗长而难懂。本人尝试写该节教程，希望有助于sagemath的入门学习，囿于时间和精力限制，暂时只就sagemath的部分常用函数都进行简单介绍。

3.1 sage安装与环境配置

- sagemath的官网下载地址：<https://www.sagemath.org/>，支持windows，linux，MacOS多种系统。
- 由于sage的本地环境较大（接近10G），如果不想在本地安装，sage也提供了在线的环境<https://sagecell.sagemath.org/>，以及cocalc：<https://cocalc.com/>，建议使用cocalc。你可以使用Github，Twitter，Google等账号注册cocalc。当然在线环境会比较慢，如果想深入学习sage建议使用本地环境。

以windows系统下本地的sage 9.2 环境为例：

三个常用的sage交互接口如下：

 SageMath 9.2 Notebook	2021/4/5 17:45	快捷方式	3 KB
 SageMath 9.2 Shell	2021/4/5 17:45	快捷方式	3 KB
 SageMath 9.2	2021/4/5 17:45	快捷方式	3 KB

三个接口常见的使用方法：

- Notebook

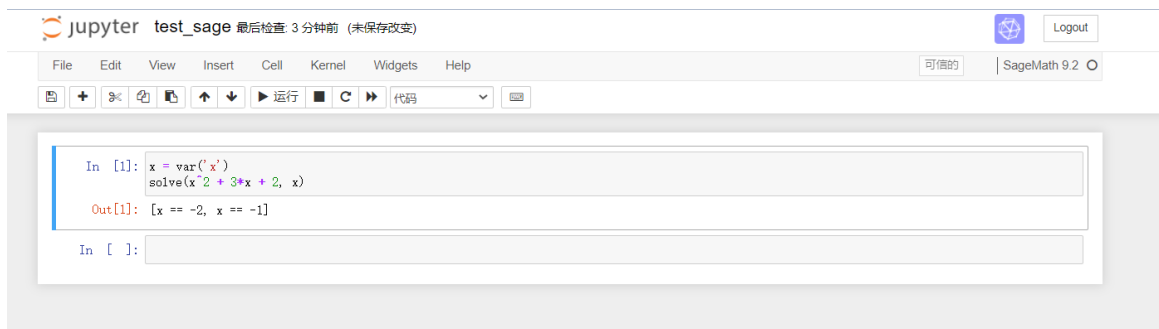
打开后会自动在本地搭建一个Jupyter Notebook，会自动弹出浏览器访问，本地访问对应的url，Notebook的界面如下：



我们可以直接打开对应sage或者python脚本进行编辑，但是不能运行。可以new一个python3或者sage的notebook来编写、调试、运行代码：

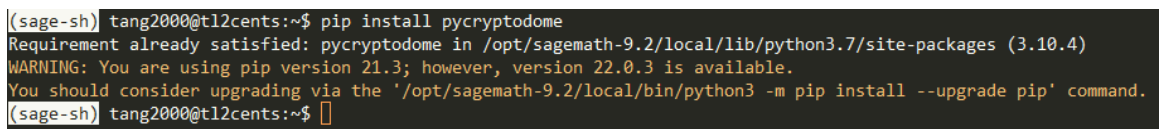


解一个一元二次方程的代码实例，之后sage内`ctrl+c`可以退出环境：



- Shell

一个命令行工具，基本的linux的命令行指令都可以运行，可以在sage安装许多必要的python包，如安装 pycryptodome:



sage中可用命令行命令 `cd` 任意更改当前路径，进入目标路径后通过命令 `sage xxx.sage` 运行脚本

另外可以在shell里面的目录下运行 `jupyter notebook` 在对应目录下打开notebook。

更多使用帮助参考: `sage -h`

- sage console

第三个sage虚拟环境应该是sage最好用的交互方式了。

如图是sage的复数域、整数环、有理数域


```
SageMath version 9.2, Release Date: 2020-10-24
Using Python 3.7.7. Type "help()" for help.
```

```
sage: CC
Complex Field with 53 bits of precision
sage: ZZ
Integer Ring
sage: RR
Real Field with 53 bits of precision
sage: QQ
Rational Field
sage: □
```

可以每写完一行执行一次，也可以批量执行代码（可以直接复制粘贴到sage），如下：

```
1 sage: from hashlib import md5
2 ....: username = b'Android'
3 ....: password = b'Greatly'
4 ....: mid_hash = "1a9852e856816224"
5 ....:
6 ....: for hour in range(24):
7 ....:     for minute in range(60):
8 ....:         for second in range(60):
9 ....:             s1 = str(hour) + str(minute) + str(second)
10 ....:             s2 = md5(s1.encode()).hexdigest()[8:24]
11 ....:             s = b"flag{" + s2.encode() + b"}" + username + password
12 ....:             res=md5(s).hexdigest()
13 ....:             if mid_hash in res:
14 ....:                 print(s)
15 ....:                 break
16 ....:
17 b'flag{80d0169d22da3c35}AndroidGreatly'
18 sage:
```

console是入门sage很重要的环境，调试和写代码都相对简便。在console内也可以切换目录，文件打开的相对路径都是以当前路径为准的，因此如果你需要导入sage脚本或者打开数据文件都需要确认一下文件是否存在在当前目录下：

```
sage: cd Desktop/
/home/sage/Desktop
sage: □
```

最重要的一个调试方式是，tab键可以自动补全，如下图，我们想要知道多项式的根方法，tab键后即可找到roots：

```
SageMath version 9.2, Release Date: 2020-10-24
Using Python 3.7.7. Type "help()" for help.

sage: P.<x>=PolynomialRing(ZZ)
sage: a=getrandbits(100)
sage: b=getrandbits(100)
sage: f=x^2-(a+b)*x+a*b
sage: f.r
radical()      real_roots()      reset_name()      revert_series()
rational_reconstruct() reciprocal_transform() resultant()      root_field()
real_root_intervals() rename()      reverse()          roots()
```

另外再介绍一个sage比较好用的源码查看的方法，比如我们想查看sage的离散对数函数的实现代码，console里或则notebook内输入 `discrete_log??`，得到函数的具体用法和源码：

```
1 Signature:
2 discrete_log(
3     a,
4     base,
5     ord=None,
6     bounds=None,
7     operation='*',
8     identity=None,
9     inverse=None,
10    op=None,
11 )
12 ...
```

下翻可查看源码，按 q 退出查看：

```
Source:
def discrete_log(a, base, ord=None, bounds=None, operation='*', identity=None, inverse=None, op=None):
    """
    Totally generic discrete log function.

    INPUT:

    - ``a`` - group element
    - ``base`` - group element (the base)
    - ``ord`` - integer (multiple of order of base, or ``None``)
    - ``bounds`` - a priori bounds on the log
    - ``operation`` - string: '*', '+', 'other'
    - ``identity`` - the group's identity
    - ``inverse()`` - function of 1 argument ``x`` returning inverse of ``x``
    - ``op()`` - function of 2 arguments ``x``, ``y`` returning ``x*y`` in group

    ``a`` and ``base`` must be elements of some group with identity
    given by identity, inverse of ``x`` by ``inverse(x)`` , and group
    operation on ``x``, ``y`` by ``op(x,y)``.

    If operation is '*' or '+' then the other
    arguments are provided automatically; otherwise they must be
    provided by the caller.
```

如果你不需要查看源码，只需要：`discrete_log?` 即可。同理对于类的方法实现代码同样可以按照类似方法查看，如：`f.small_roots??`。

3.2 sage基本函数使用

基本的python方法这里就不再赘述，这里主要讲sage自带的一些函数与特殊运算，由于目前sagemath的中文相关文档较少，这里详细介绍一下CTF中经常用到的函数。

3.2.1 sage与python的不同

```
1 5          # 5的类型是整数环上的元素，而不是int
2 sage: type(5)
3 <class 'sage.rings.integer.Integer'>
4
5 # 注意int(5)和5的方法是不一样的，如5的比特长方法
6 sage: int(5).bit_length()
7 3
8 sage: 5.nbits()
9 3
10
11 5**3      # **代表幂
12 5^3       # ^也是幂，python中是异或
13 5^^3      # ^^表示sage中的异或
14 10/3      # 注意10/3是一个有理数，而在python中它会变成小数
15 sage: type(10/3)
16 <class 'sage.rings.rational.Rational'>
```

另外在编程中特别需要注意的一点是 $\text{pow}(a, x, N)$ 运算，它的返回结果是 $\text{Zmod}(N)$ 上的元素，你之后不在这个有限环上运算，最好转成正常整数 $\text{ZZ}(\text{pow}(a, x, N))$ 再运算：

```
1 sage: type(pow(2,4,7))
2 <class 'sage.rings.finite_rings.integer_mod.IntegerMod_int'>
3
4 sage: type(ZZ(pow(2,4,7)))
5 <class 'sage.rings.integer.Integer'>
```

3.2.2 基本的环、域

- 整数环：ZZ，也等价于Integers
- 有理数环：QQ
- 实数域：RR

- 复数域: `CC`
- 多项式环: `PolynomialRing()`

```
1 | # 定义在有理数环上的多项式环
2 | sage: PQ.<x> = PolynomialRing(QQ)
```

PQ: 多项式环的名字, 自定义

x: 变量的名字, 自定义

- 其他有限域、环:

伽罗瓦域(N必须是某个素数或者某个素数的k次方): `GF(N)`、`GF(2^8,modulus=[1,0,0,1,1,1,0,0,1])`

一般有限环: `Zmod(N)`

一个元素在各种域上的转换示例如下:

```
1 | sage: e=12
2 | sage: CC(e)
3 | 12.000000000000000
4 | sage: ZZ(e)
5 | 12
6 | sage: G=GF(7)
7 | sage: G(e)
8 | 5
9 | sage: Z7=Zmod(7)
10 | sage: Z7(e)
11 | 5
```

3.2.3 基本数论函数

- 求逆: e 模 n 的逆

定义在`Zmod(N)`上的元素 e , 直接 `e^-1` 或者 `1/e`

否则直接使用 `inverse_mod(e,n)` 函数获得 e 模 n 的逆

- 最大公因数: (a,b) 的最大公因数

`gcd(a,b)`

- 最大公倍数: (a,b) 的最大公倍数

`lcm(a,b)`

- 模幂: $e^x \bmod n$

定义在 $\mathbb{Z}_{\text{mod}}(N)$ 上的元素 e ，直接 `e^x`

否则直接使用 `pow(e,x,n)`

- 素数判断

```
1 | sage: x=123721984710347123123
2 | sage: x.is_prime()
3 | False
```

- 阶乘:

```
factorial(x)
```

- 欧拉函数:

```
1 | euler_phi(n)
2 | # 求n的欧拉函数 phi = n*prod([1 - 1/p for p in prime_divisors(n)])
```

- 中国剩余定理求解

$$\begin{aligned} x &\equiv m_1 \pmod{n_1} \\ x &\equiv m_2 \pmod{n_2} \end{aligned}$$

解为:

```
1 | crt([m1,m2],[n1,n2])
2 | sage: crt([1,2],[3,5])
3 | 7
```

- 扩展欧几里得算法:

$$d = \gcd(a, b) = u * a + v * b$$

```
1 | d,u,v=xgcd(a,b)
```

- 素数分解

```
1 | factor(1024) -> 2^10
2 | prime_divisors(1024) -> [2]
3 | divisors() #所有因子
```

- 开根(有限域开根)

整数域开根

$$x^m = y$$

已知 m, y 求 x

```
1 sage: y=87^8
2 sage: y.nth_root(8)
3 87
```

有限域开根:

$$x^m \equiv y \pmod{N}$$

已知 m, y, N 求 x

```
1 sage: y=pow(78,888,65537)
2 # 此时已经定义在有限域Zmod(65537)上
3 sage: y.nth_root(888)
4 65459
5 sage: x=y.nth_root(888)
6 sage: x+78
7 0
```

注意: 开根有多解, `nth_root`只会返回一个解, 如果需要得到所有解, 可以加一个参数:

```
1 sage: x=y.nth_root(888,all=True)
```

当然, sage的有限域开根函数实现并不好, 可以手动实现 `AMM` 算法或者使用更为强大的工具 `mma` (`mma` 解方程的性能绝对是世界顶尖水平的)。

- 离散对数

sage上实现了多种离散对数的求解方式, 包括gp接口的离散对数求解。

参数说明: 求解以 `base` 为底, `a` 的对数; `ord` 为 `base` 的阶, 可以缺省, `operation` 可以是 `+` 与 `*`, 默认为 `*`; `bounds` 是一个区间 `(ld,ud)`, 需要保证所计算的对数在此区间内。

- `discrete_log(a,base,ord,operation)`

通用的求离散对数的方法。

- `discrete_log_rho(a,base,ord,operation)`

求离散对数的Pollard-Rho算法。

- `discrete_log_lambda(a,base,bounds,operation)`

求离散对数的Pollard-kangaroo算法 (也称为lambda算法)。

- `bsgs(base,a,bounds,operation)`

小步大步法。

其中`operation`为`+`时一般是应用在椭圆曲线 (ECC) 的离散对数, ECC部分本教程暂未涉及。

代码示例:

```
1  #生成32位的素数p作为模数, int为32位, 超过int要在数字后加L
2  p=random_prime(2L**32)
3
4  #定义有限域GF(p)
5  G=GF(p)
6
7  #找一个模p的原根
8  gp ('znprimroot('+str(p)+')')
9  #输出Mod(rt,p), 则x是模p的原根
10 g=G.primitive_element()
11
12 #生成私钥
13 x=G(ZZ.random_element(p-1))
14
15 #公钥y=g^x mod p, 由于已经定义在GF(p)上, 因此g**x就是g^x mod p
16 y=g**x
17
18 #计算离散对数的通用方法
19 discrete_log(y,g)==x
20 #计算离散对数的lambda方法
21 discrete_log_lambda(y,g,(floor(ZZ(x)/2),2*ZZ(x)))==x
22 #小步大步法计算离散对数
23 bsgs(g,y,(floor(ZZ(x)/2),2*ZZ(x)))==x
24
25 #n为质数或质数幂 (线性筛Index Calculus)
26 R = Integers(99)
27 a = R(4)
28 b = a^9
29 b.log(a)
30 n=99
31 #或
32 x = int(pari(f"znlog({int(b)},Mod({int(a)},{int(n)}))"))
33 x = gp.znlog(b, gp.Mod(a, n))
34 # 代码修改自 Lazzaro @ https://lazzaro.github.io
```

输出:

```
1  Mod(2, 3223324843)
2  True
3  True
4  True
5  9
```

3.2.4 线性代数相关函数

- 矩阵定义、运算

一般矩阵定义:

```
1 # 直接定义ZZ矩阵并且初始化
2 A = Matrix(ZZ, [[1,2,3],[3,2,1],[1,1,1]])
3 A = matrix(Zmod(2), [[1,2,3],[3,2,1],[1,1,1]])
4 # 模2有限域上的矩阵
5 B=matrix(GF(2),A)
6 # 向量定义
7 Y = vector(ZZ,[0,-4,-1])
8 Y = vector(GF(2),[0,-4,-1])
9
10 # 定义矩阵但不初始化: GF(2) 上的10*10矩阵, 默认所有元素为0
11 m = matrix(GF(2), 10, 10)
```

分块矩阵定义:

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

```
1 A = matrix(GF(2), 10, 10)
2 B = matrix(GF(2), 10, 30)
3 C = matrix(GF(2), 5, 10)
4 D = matrix(GF(2), 5, 30)
5 M=block_matrix([[A,B],[C,D]])
6 #等价于
7 M=block_matrix([[A,B],[C,D]],nrows=2)
```

矩阵运算:

```
1 m1=matrix([[1,2],[1,3]])
2 m2=matrix([[2,1],[2,3]])
3 m1+m2
4 m1*m2
5 输出
6 [3 3]
7 [3 6]
8 [ 6 7]
9 [ 8 10]
```

转置: `M.transpose()`

求逆: `M.inverse()` 或者 M^{-1}

特征值: `M.eigenvalues()`

特征向量（右）：`M.eigenvectors_right()`

行列式：`M.det()`

求秩：`M.rank()`

- 解线性方程组：

对于任意线性（模）方程组，sage都可以很方便地求解：

$$\text{线性方程组: } \begin{cases} a_{11} * x_1 + a_{12} * x_2 + \dots + a_{1m} * x_m = b_1 \\ a_{21} * x_1 + a_{22} * x_2 + \dots + a_{2m} * x_m = b_2 \\ \dots \\ a_{n1} * x_1 + a_{n2} * x_2 + \dots + a_{nm} * x_m = b_n \end{cases}$$

矩阵表示： $A * X = B$

其中 $A \in \mathbb{R}^{n \times m}, X \in \mathbb{R}^m, B \in \mathbb{R}^n$

其中 \mathbb{R} 可以替换成任意域： $Zmod(N)$ 、 ZZ 、 QQ

给定有限域或者整数域上的矩阵 A, B ，求解 $A * X = B$ ：

```
1 | X = A.solve_right(B)
2 | #等价于
3 | X = A \ B
```

给定有限域或者整数域上的矩阵 A, B ，求解 $X * A = B$ ：

```
1 | X = A.solve_left(B)
```

特别地，sagemath还集成右（左）核空间的求解，也就是 $A * X = 0$ 和 $X * A = 0$ 的所有解空间

```
1 | X = A.left_kernel() # X*A=0
2 | X = A.right_kernel() # A*X=0
3 | # 矩阵形式
4 | X=A.right_kernel_matrix()
```

对于一般（模）方程的求解：

```
1 | # 一般方程
2 | x, y = var('x, y')
3 | solve([x+y==10, x-y==0], x, y)
4 |
5 | # 解模方程
6 | sage: solve_mod([x+y==66, x-y==23], 97)
7 | [(93, 70)]
```

3.2.5 多项式环及其函数

- 多项式环定义

前面已经稍微涉及了多项式环的定义：

```
1  # 定义在有理数环上的一元多项式环
2  sage: PQ.<x> = PolynomialRing(QQ)
3  # PQ: 多项式环的名字, 自定义
4  # x: 变量的名字, 自定义
5
6  # 或者等价定义
7  sage: PQ = PolynomialRing(QQ, 'x')
8  sage: x=PQ.gen()
9  # 或者
10 sage: PQ = QQ['t']
```

二元多项式环可以如下定义，其他类似：

```
1  PQ2.<x,y> = PolynomialRing(QQ)
```

- 多项式定义

在定义好多项式环后：`PQ.<x> = PolynomialRing(QQ)`

```
1  PQ.<x> = PolynomialRing(QQ)
2  f=4*x^3+2*x+1
```

一个等价的定义方式是用列表：

```
1  sage: PQ.<x> = PolynomialRing(QQ)
2  ....: PQ([1,2,0,4])
3  4*x^3 + 2*x + 1
```

生成随机多项式

```
1  sage: R.<y> = PolynomialRing(GF(65537))
2  ....: degree=7
3  ....: S = R.random_element(degree)
4  sage: S
5  60064*y^7 + 31544*y^6 + 19047*y^5 + 19873*y^4 + 53675*y^3 + 39961*y^2 + 40289*y
   + 30493
```

特别地，对于伽罗瓦域 $GF(2^n)$ 上元素的声明，如下（注意模多项式需要是本元多项式）：

```

1 F.<x> = GF(2^8,modulus=[1,0,0,1,1,1,0,0,1])
2 F.fetch_int(21) == F(21.bits())
3 F(21.bits()).integer_representation()#逆过程

```

- 多项式相关运算、函数

多项式分解:

```

1 sage: R.<y> = PolynomialRing(GF(65537))
2 sage: f=R.random_element(degree)*R.random_element(degree)
3 sage: f.factor()
4 (64343) * (y + 10474) * (y + 49729) * (y^2 + 7588*y + 41809) * (y^3 + 20885*y^2
+ 60459*y + 29275) * (y^3 + 27396*y^2 + 16874*y + 56765) * (y^4 + 25258*y^3 +
12887*y^2 + 59574*y + 64190)

```

多项式最大公因式:

```

1 sage: g=R.random_element(degree)
2 sage: f1=R.random_element(degree)*g
3 sage: f2=R.random_element(degree)*g
4 sage: gcd(f1,f2)
5 y^7 + 10115*y^6 + 11715*y^5 + 59953*y^4 + 53514*y^3 + 2238*y^2 + 55526*y +
46381
6 sage: f1.gcd(f2)
7 y^7 + 10115*y^6 + 11715*y^5 + 59953*y^4 + 53514*y^3 + 2238*y^2 + 55526*y +
46381

```

首一多项式:

```

1 sage: f.monic()
2 y^14 + 10256*y^13 + 12184*y^12 + 38005*y^11 + 59412*y^10 + 48187*y^9 +
45070*y^8 + 34871*y^7 + 50384*y^6 + 61263*y^5 + 52827*y^4 + 24231*y^3 +
4268*y^2 + 31669*y + 46586

```

多项式的根:

```

1 sage: f.roots()
2 [(55063, 1), (15808, 1)]
3
4 # small_roots,使用前必须保证f是首一的多项式
5 # 关于small_roots的详细参数可以参考coppersmith节
6 sage: f.monic().small_roots()
7 []

```

多项式的结式：

在两个多项式有公共根时,可以求结式，之后再求解根

```

1 sage: f.resultant(g)
2 51209

```

Groebner basis:

在已知模方程的一些关系后，我们可以通过Groebner basis得到一些方程的根，如下图得到的就是 a, b, c 三个解

```

1 sage: G=GF(next_prime(getrandbits(512)))
2 sage: a=G(getrandbits(512))
3 sage: b=G(getrandbits(512))
4 sage: c=a+b+233
5 sage: a3=a^3
6 sage: b3=b^3
7 sage: c3=c^3
8 sage: x,y,z =G['x,y,z'].gens()
9 ....: I = ideal(z-x-y-233, x^3-a3, y^3-b3, z^3 - c3)
10 ....: B = I.groebner_basis()
11 verbose 0 (3837: multi_polynomial_ideal.py, groebner_basis) Warning: falling
back to very slow toy implementation.
12 sage: B
13 [x +
411536540793027167214591795906419798687587992333726001948039912952254695741419
6891761965008866716311717547095730704169414503027605502719768955802977966739,
y +
143819664047863398839470865609778212405062931393142640124479882190403460146369
5109291179031271158354211457757606425866100114378067643728009244830918500697,
z +
555356204840890566054062661516198011092650923726868642072519795142658155887789
2001053144040137874665929004853337130035514617405673146447778200633896467203]

```

3.2.6 格相关函数*

sagemath集成了格上许多经典算法，在这里介绍crypto中常用的几个：

- LLL算法

```
1 sage: M=matrix.random(GF(next_prime(getrandbits(10))),5,5)
2 ....: sage: M=M.change_ring(ZZ)
3 ....: sage: M.LLL()
4 [-62  55 -57  -9 -48]
5 [-59  58  74  22 -39]
6 [ 11   7 -39   8 125]
7 [144  60 -14 -41  24]
8 [ 41  25  17 176 -11]
```

- coppersmith算法

也就是前文提到的small_roots函数，其中对多项式有如下约束：必须是首一多项式（monicpolynomial），可以使用 $f = f.monice()$ 解决；f必须是一元多项式，暂不支持多元多项式。

```
1 # 这些参数的调整可以参加coppersmith节
2 x=f.small_roots(X=upperbound,beta=0.5,epsilon=1/20)
```

- Babai算法

Babai算法sagemath上并没有集成的接口，这里给出一个CTF中crypto常用的实现：

```
1 # 输入矩阵 basis, 目标向量, v
2 def approximate_closest_vector(basis, v):
3     """Returns an approximate CVP solution using Babai's nearest plane
4     algorithm.
5     """
6     BL = basis.LLL()
7     # 施密特正交化基
8     G, _ = BL.gram_schmidt()
9     _, n = BL.dimensions()
10    small = vector(ZZ, v)
11    for i in reversed(range(n)):
12        c = QQ(small * G[i]) / QQ(G[i] * G[i])
13        c = c.round()
14        small -= BL[i] * c
15    return (v - small).coefficients()
```