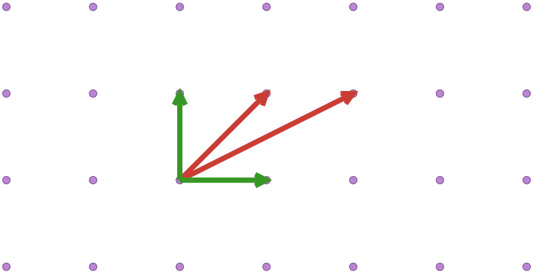
ELI5: Lattice Reduction

Chris Peel

TIBCO Open Source Community Day

20 May, 2021



Why should I care about lattices?

You may not need to care! Even so, hopefully you'll get a few ideas (and buzzwords!) that you can wield on your nerdy friends

- Cryptography will be a hot field for the next 100+ years
- One of the hottest areas is the use of lattice tools to build post-quantum cryptography, as well as fancy tools like fully-homomorphic encryption

This talk will give a brief introduction to lattices, specifically lattice reduction. A background in linear algebra will be helpful

These slides are available in the file ossLLL.pdf at github.com/christianpeel/pub/

When could I use lattice reduction?

Lattice tools are often used in places where one would normally use linear algebra, but an integer-valued solution is desired

There are practical uses:

- Integer programming
- Post-quantum cryptography (LWE)
- Cryptanalysis (cracking SSH, HTTPS)
- Digital communication
- Encrypted ML (FHE in Julia)
- Coding theory
- Finding anagrams :-)

And there are theoretical uses

- Disproving Merten's Conjecture
- Sphere packing (with Julia!)
- Diophantine equations
  - Solving $x^3 + y^3 + z^3 = d$
- Geometry of flat tori
- Finding Spigot formulas
- Factoring Polynomials
- Computing the Riemann theta function
- Physics (Feynman integrals)

Outline

Background

- Basics, Definitions
- Lattice Reduction

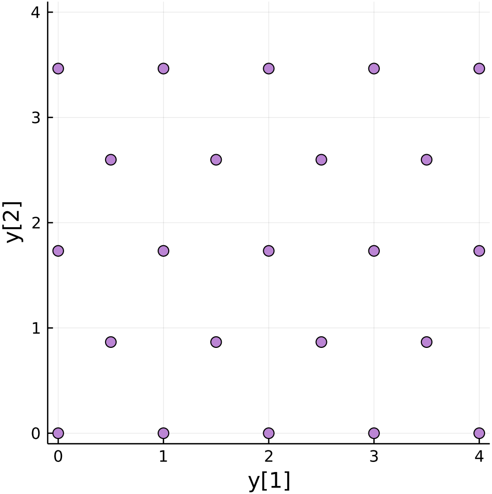
`LLLplus.jl`

- What's in it?

Demo

- Cracking Merkle–Hellman knapsack cryptosystem

What's a Lattice?



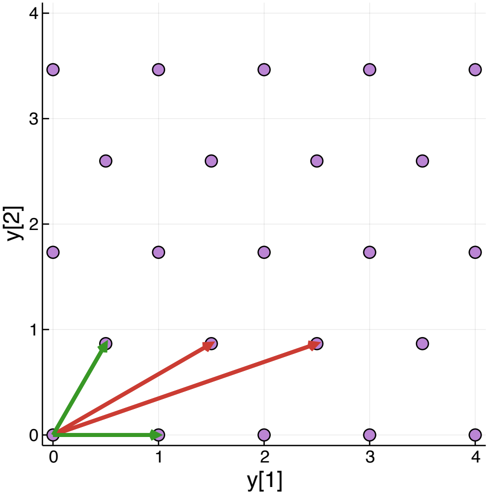
- A full-rank discrete additive subgroup of (say) $\mathbb{R}^n$ or $\mathbb{C}^n$
- $\mathbb{Z}^n$ is a lattice in $\mathbb{R}^n$

A practical definition

For a basis matrix B, and a vector of integers $\mathbf{z}$, the set of points $\mathbf{y}$ reachable by $\mathbf{y} = B\mathbf{z}$ is a lattice:

$$\mathcal{L}(B) = \{B\mathbf{z} : \mathbf{z} \in \mathbb{Z}^n\}$$

For a lattice, how many bases are possible?



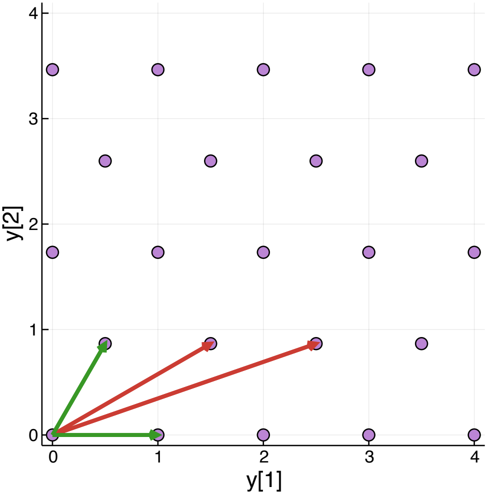
There are an infinite number of bases for a lattice; which one should we use?

$$B_{red} = \begin{bmatrix} 2.5 & 1.5 \\ .86602 & .86602 \end{bmatrix}$$

$$B_{green} = \begin{bmatrix} 1.0 & .5 \\ 0.0 & .86602 \end{bmatrix}$$

In many problems, we want a short, close-to-orthogonal basis, like the green basis

How are different lattice bases related?



A **unimodular matrix** is square, integer-valued, and has determinant ±1. Its inverse is also unimodular

Relation between Bases

Every pair of lattice bases B_1 and B_2 are related by a unimodular matrix T: $B_1 = B_2 T$

For the bases in the figure, $B_{red} = B_{green} T$, where

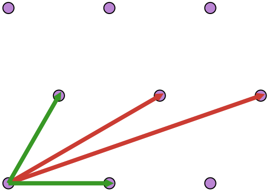
$$T = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$$

How can we find a good basis?

Use lattice reduction

Given lattice with basis B_1, the goal of lattice reduction is to find another basis B_2 for the same lattice which has short, closer-to-orthogonal basis vectors

Often, “short” and “orthogonal” are defined according to the Euclidian norm. So $B_2^T B_2$ is closer to diagonal than $B_1^T B_1$, and the diagonal elements of $B_2^T B_2$ are smaller than those of $B_1^T B_1$



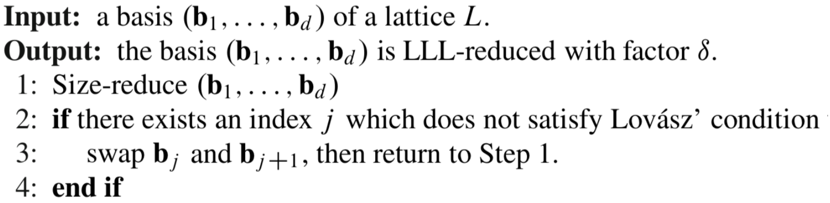
‘Lattice reduction is like QR for integer problems.’
Jack Poulson

Instead of an orthonormal Q, we have a close-to-orthogonal reduced basis, and instead of a triangular R we have a unimodular matrix: $B_1 = B_2 T$

How does one do lattice reduction?

The most important lattice reduction technique is from Lenstra, Lenstra, and Lovász[1], known as the LLL algorithm

LLL in pseudocode



```
Input: a basis $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ of a lattice $L$.
Output: the basis $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ is LLL-reduced with factor $\delta$.
1: Size-reduce $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$
2: if there exists an index $j$ which does not satisfy Lovász' condition
3:     swap $\mathbf{b}_j$ and $\mathbf{b}_{j+1}$, then return to Step 1.
4: end if
```

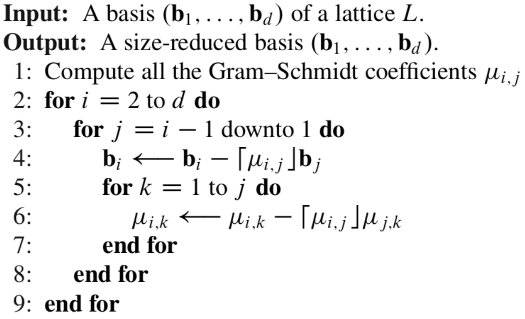
Lovász' condition is $||\mathbf{b}_{j+1}||^2 \geq (\delta - \mu_{j+1,j}^2)||\mathbf{b}_j||^2$ where the coeficients μ are Gram-Schmidt coefficients from size reduction

[1] A. K. Lenstra; H. W. Lenstra Jr.; L. Lovász; "Factoring polynomials with rational coefficients". Mathematische Annalen 261, 1982.

Size Reduction? Gram-Schmidt? Do I need to know this?

No, most LLL users can skip previous, current, next slides :-)

Size Reduction pseudocode[2]



```
Input: A basis $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ of a lattice $L$.
Output: A size-reduced basis $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$.
Compute all the Gram–Schmidt coefficients $\mu_{i,j}$
for $i = 2$ to $d$ do
    for $j = i - 1$ downto 1 do
        $\mathbf{b}_i \longleftarrow \mathbf{b}_i - \lceil \mu_{i,j} \rfloor \mathbf{b}_j$
        for $k = 1$ to $j$ do
            $\mu_{i,k} \longleftarrow \mu_{i,k} - \lceil \mu_{i,j} \rfloor \mu_{j,k}$
        end for
    end for
end for
```

Size reduction is GS with rounding

There are LLL variants which use

- Gram-Schmidt (shown)
- Givens rotations
- Householder rotations
- A Cholesky decomposition (fastest)

[2] The LLL and size reduction pseudocode are from P. Q. Nguyen "Hermite's constant and lattice algorithms," a chapter of The LLL Algorithm, Springer, Berlin, Heidelberg, 2009, pp 19-69

Givens-based LLL in Julia

```julia
function lll(H::Matrix{Td},δ::Float64=3/4) where {Td<:Number}
    B = copy(H);   N,L = size(B);  _,R = qr(B)
    lx  = 2
    while lx <= L
        for k=lx-1:-1:1
            rk = R[k,lx]/R[k,k]
            mu = round(rk)
            if abs(mu)>0
                B[:,lx]   -= mu * B[:,k]
                R[1:k,lx] -= mu * R[1:k,k]
            end
        end
        nrm = norm(R[lx-1:lx,lx])
        if δ*abs(R[lx-1,lx-1])^2 > nrm^2
            B[:,[lx-1,lx]]    = B[:,[lx,lx-1]]
            R[1:lx,[lx-1,lx]] = R[1:lx,[lx,lx-1]]
            cc = R[lx-1,lx-1] / nrm
            ss = R[lx,lx-1]   / nrm
            Θ = [cc' ss; -ss cc]  # Givens rotation
            R[lx-1:lx,lx-1:end] .= Θ * R[lx-1:lx,lx-1:end]
            lx = max(lx-1,2)
        else; lx = lx+1; end
    end
    return B
end
```

What should I remember about the LLL?

Remember two things:

- LLL runs fast; $O(d^5)$ for bases of size d
- LLL reduces the basis: $||\mathbf{b}_1|| \leq (\frac{2}{\sqrt{4\delta-1}})^{d-1}\lambda_1(\mathcal{L})$

The LLL is a baseline lattice tool. Its polynomial speed and acceptable reduction quality is what brought interest to lattice tools

Outline

Background

- Basics, Definitions
- Lattice Reduction

`LLLplus.jl`

- What's in it?

Demo

- Cracking Merkle–Hellman knapsack cryptosystem

Lattice Tools in LLLplus.jl

Lattice Tool	Function	Use case
LLL lattice reduction	`lll`	most lattice problems
Seysen lattice reduction	`seysen`	math, WiFi
Brun lattice reduction	`brun`	math, WiFi
CVP solver	`cvp`	WiFi, GGH
SVP solver	`svp`	NTRU, RLWE

Demo function	Application
`subsetsum`	cryptanalysis, integer relations
`integerfeasibility`	integer programming feasibility
`rationalapprox`	find rational approx for vector
`spigotBBP`	spigot formulas for irrationals

How about an LLL demo?

```
julia> Br=[2.5    1.5; 0.866025 0.866025];
julia> Pkg.add("LLLplus"); using LLLplus

julia> B,T,_ = lll(Br); B
2×2 Array{Float64,2}:
 -1.0  -0.5
  0.0   0.866025

julia> T
 2×2 Array{Int64,2}:
 -1  -2
  1   3

julia> [det(T) det(inv(T))]
1×2 Array{Float64,2}:
 -1.0  -1.0

julia> islllreduced(B)
true
```

Outline

Background

- Basics, Definitions
- Lattice Reduction

`LLLplus.jl`

- What's in it?

Demo

- Cracking Merkle–Hellman knapsack cryptosystem

Subset-Sum and Integer Relations

Subset-Sum

Given a vector $\mathbf{a}$ of integers, and a sum s, if there is a binary vector $\mathbf{x}$ such that $\mathbf{x}^T\mathbf{a} = s$, find it.

The LLL-based technique from Lagarias and Oldyzko was designed to solve low-density subset-sum problems. It breaks the Merkle–Hellman knapsack cryptosystem and is widely useful, for example to solve related problems like:

Integer Relations

Given a vector $\mathbf{a}$ of real numbers, if there is an integer vector $\mathbf{x}$ such that $\mathbf{x}^T\mathbf{a} = 0$, find it.

Integer relations solvers can be used to make spigot algorithms, say giving the nth digit of π without computing any of the other digits

Demo: A subset-sum problem

```
julia> setprecision(BigFloat,300); N=50; Bitdepth=190;
julia> # Bitdepth can be 256+, just doesn't fit on screen
       a=rand(0:2^BigInt(Bitdepth)-1,N);
julia> a[1:3]
3-element Array{BigInt,1}:
 911200129391658686469201173324473216271570073348033300075
 666563007748951582781404496296235427608875017772431026416
 832622399672004543019820919656212490862510181960061392073
julia> xtrue=rand(Bool,N); s=a'*xtrue;
julia> @elapsed x,_=LLLplus.subsetsum(a,s)
2.535546165
julia> s-x'*a
0.0
```

Are there any...

Questions?