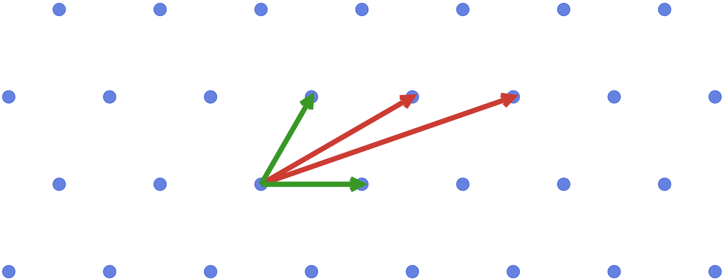
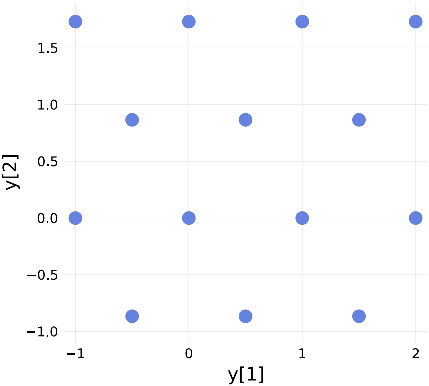
Lattice Reduction with LLLplus.jl

Chris Peel

JuliaCon 2021



What's a Lattice?



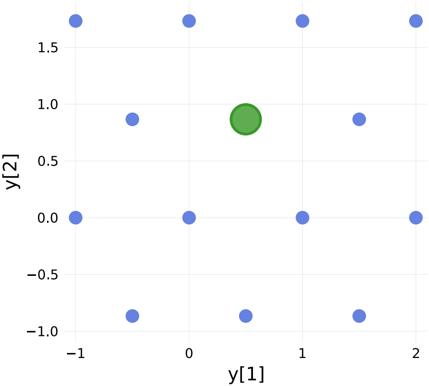
- A full-rank discrete additive subgroup of (say) $\mathbb{R}^n$ or $\mathbb{C}^n$
- $\mathbb{Z}^n$ is a lattice in $\mathbb{R}^n$

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:

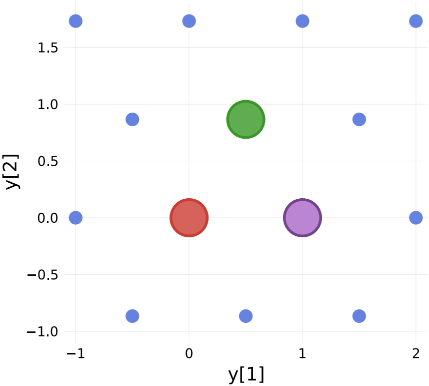
$$\mathcal{L}(B) = \{B\mathbf{z} : \mathbf{z} \in \mathbb{Z}^n\}$$

How did you generate the green lattice point?




```
julia> B=[1.0  0.5
          0.0  0.866025];
julia> zgreen=[0; 1];
julia> ygreen=B*zgreen
2-element Vector{Float64}:
 0.5
 0.866025
```

How did you generate the purple and red lattice points?




```
julia> B=[1.0  0.5
          0.0  0.866025];
julia> zgreen=[0; 1];
julia> ygreen=B*zgreen
2-element Vector{Float64}:
 0.5
 0.866025
julia> zpurple=[1; 0];
julia> ypurple=B*zpurple
2-element Vector{Float64}:
 1.0
 0.0
julia> zred=[0; 0];
julia> yred=B*zred
2-element Vector{Float64}:
 0.0
 0.0
```

Why should I care about lattices?

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

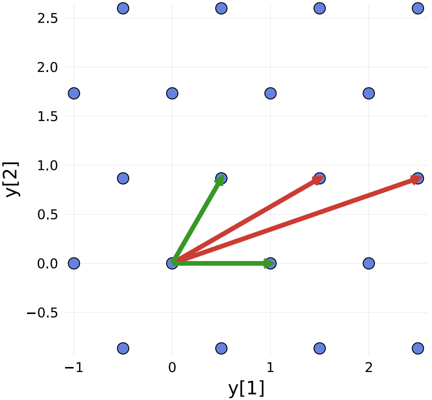
There are practical uses:

- Post-quantum cryptography (LWE)
- Integer programming
- 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 eqns (more)
- Geometry of flat tori
- Finding Spigot formulas
- Factoring Polynomials
- Computing the Riemann theta function
- Physics (Feynman integrals)

For a lattice, how many bases are possible?



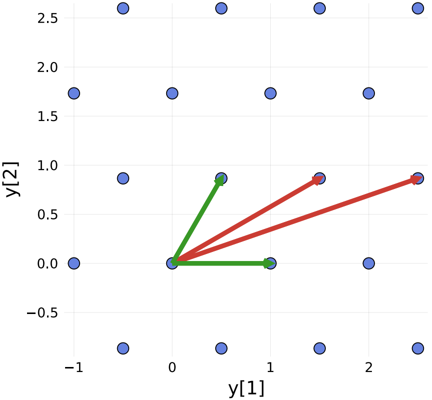
There are an infinite number of bases for a lattice; which one should we use?

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

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

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

How 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

'Lattice reduction is like QR for integer problems.'

Jack Poulson

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 δ.

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

```
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

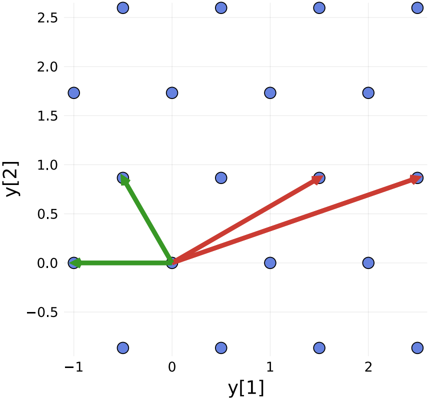
Outline

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

Toy (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> Pkg.add("LLLplus");
julia> using LLLplus;
julia> Bred=[2.5     1.5;
            0.866025 0.866025];
julia> Bgreen,_ = lll(Bred);
julia> Bgreen
2×2 Matrix{Float64}:
 -1.0  -0.5
  0.0   0.866025
```

Our examples have been with 2-dimensional lattices; many interesting problems have larger lattices, say 100 dimensions or more

What types does LLLplus.lll work on?

`LLLplus.lll` works on bases over all `Signed` integers, `AbstractFloats`, `Complex`, and user-defined subtypes like `BitIntegers`. I've tried around 34 types.

```
julia> using BitIntegers, LLLplus
julia> B = rand(0:Int512(2)^33,2,2)+
         im*rand(0:Int512(2)^33,2,2)
2×2 Matrix{Complex{Int64}}:
 5941420354+5486248041im  5574890144+3732896516im
 2538719538+1638107804im  3830374646+2133953247im
julia> Blll,Tlll = lll(B); Tlll
2×2 Matrix{Complex{Int64}}:
 -1+0im  -1+2im
  1+0im   2-2im
```

To have LLLplus.lll work with a new type, check that `LinearAlgebra.qr` works, then add a method to `LLLplus.getIntType` for new float types

Outline

Subset-Sum

(Cryptographer's) Subset-Sum

Given a vector **a** of integers, and a sum s, if there is a binary vector **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.

```
julia> setprecision(BigFloat,300); N=50; Bitdepth=190;
julia> a=rand(0:2^BigInt(Bitdepth)-1,N);
julia> xtrue=rand(Bool,N); s=a'*xtrue;
julia> @elapsed x,_=LLLplus.subsetsum(a,s)
2.535546165
julia> s-x'*a
0.0
```

Challenge for you: find or write a tool to beat the speed of `LLLplus.subsetsum` in the scenario above or scenarios using 64-bit math ($N = 20$, $Bitdepth = 25$)

Thank You!