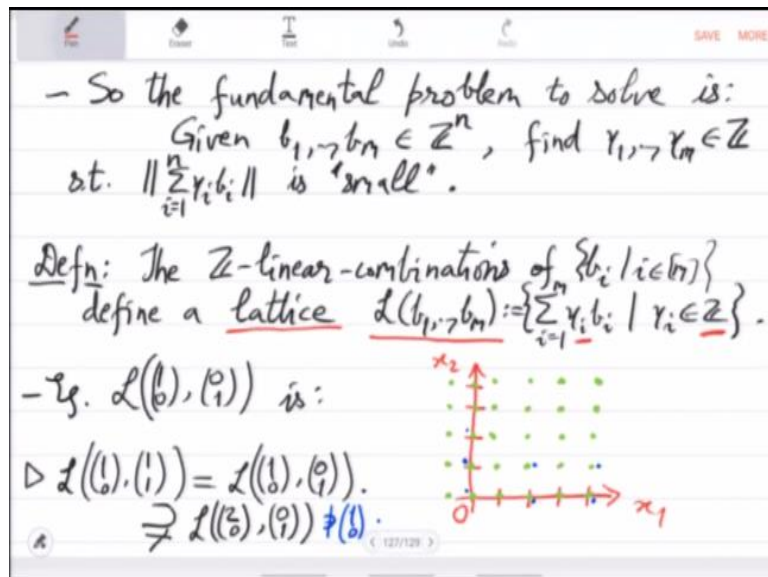


Computational Number Theory and Algebra
Prof. Nitin Saxena
Department of Computer Science and Engineering
Indian Institute of Technology-Kanpur

Lecture – 21

Screen recorder. Okay start. So in the last class we had started studying this new mathematical object that we are calling lattices, right.

(Refer Slide Time: 00:38)



– So the fundamental problem to solve is:
 Given $b_1, \dots, b_m \in \mathbb{Z}^n$, find $\gamma_1, \dots, \gamma_m \in \mathbb{Z}$
 s.t. $\|\sum_{i=1}^m \gamma_i b_i\|$ is 'small'.

Defn: The \mathbb{Z} -linear-combinations of $\{b_i : i \in [m]\}$
 define a lattice $\mathcal{L}(b_1, \dots, b_m) := \{\sum_{i=1}^m \gamma_i b_i \mid \gamma_i \in \mathbb{Z}\}$.

– eg. $\mathcal{L}(\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix})$ is:

$\triangleright \mathcal{L}(\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}) = \mathcal{L}(\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix})$.
 $\nRightarrow \mathcal{L}(\begin{pmatrix} 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}) \neq \mathcal{L}(\begin{pmatrix} 1 \\ 0 \end{pmatrix})$.

The diagram shows a 2D coordinate system with axes x_1 and x_2 . A grid of green dots represents the lattice points. The origin is marked with a red '0'. The axes are labeled with red arrows and x_1 , x_2 .

So pictorially they are just points arranged in a grid multi-dimensional or higher dimensional grid. So in the input you are given basis elements, you are given vectors b_1 to b_m and then the lattice generated by these vectors is integral combinations, right. So the important thing here is that γ_i 's that you are using they are all integers. Why integers?

(Refer Slide Time: 01:12)

How do we compute \tilde{g} (with "small" coeffs)?

Let g_k be of $\deg = n' < n$. Unknown polynomials are:
 $\tilde{g} =: \sum_{i=0}^{n-1} \underline{c}_i \cdot x^i$ & $\underline{l}_k =: \sum_{i=0}^{n-1-n'} \underline{\alpha}_i \cdot x^i$ s.t.

$$\underline{\tilde{g}} \equiv g_k \cdot \underline{l}_k \pmod{p^{2^k}}$$

$$\Rightarrow \sum_{i=0}^{n-1} \underline{c}_i \cdot x^i = \sum_{i=0}^{n-1-n'} \underline{\alpha}_i \cdot (x^i g_k) + \sum_{i=0}^{n-1} \underline{\beta}_i \cdot (p^{2^k} x^i) \quad (ii)$$

Find integral $\underline{c}, \underline{\alpha}, \underline{\beta}$'s in eqn. (ii) s.t. $\|\underline{c}\| = \sqrt{\sum \underline{c}_i^2}$ is "small" $< 2^{(l+ln)n}$.

Well because in the problem of factoring polynomials this alpha i and beta i that you used they were integers, right. And moreover this x to the i g k and p raised to 2 raised to k, these are polynomials or these which will be seen as vectors with integral coordinates, right. So everything here is happening over integers. And there you want a short vector.

(Refer Slide Time: 01:39)

– Shortest vector problem (SVP): Find a vector $\underline{v} \in \mathcal{L}(b_1, \dots, b_n)$ s.t. $\|\underline{v}\| = \min_{0 \neq \underline{u} \in \mathcal{L}} \|\underline{u}\|$.

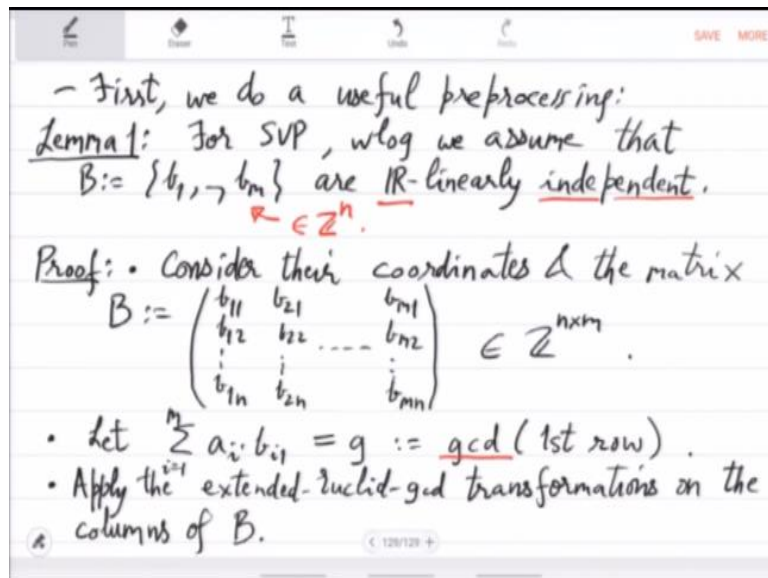
▷ [Ajtai'98] SVP is NP-hard.
 [Micciancio'98] constant-approx. of SVP is NP-hard.

– But, we need merely a 2^n -approximation for Step 4!

→ We'll develop this approximation algorithm ($L^3 =$ Lenstra-Lenstra-Lovász)

So that is the SVP problem. SVP problem would be a stronger problem, stronger question. You want the shortest, you want a shortest vector in the lattice spanned by the vectors b_1 to b_m . But that is hard. So instead what we will now do is we will find an approximate shortest vector, okay. And this algorithm, approximation algorithm is due to L^3 , Lenstra – Lenstra – Lovasz from the 80's.

(Refer Slide Time: 02:08)



So this was a breakthrough result and it is applied in many places as we will see later. It started with the integral polynomial factoring. Okay, so the first preprocessing step which is a good property to study for SVP is that we can assume the generating vectors b_1 to b_m to be linearly independent over reals, okay. So that will be kind of a simplification. So why is that possible? So let us consider coordinates of b_1 to b_m .

So consider their coordinates and the matrix B that they defined. I used the same b . So b_1 has coordinates b_{11}, b_{12} to b_{1n} , n is the ambient space. Then the same thing you do with b_2 . And finally with b_m , okay. So this is the n cross m matrix of integers. Remember that m is right now m in this question may be less than n , equal to n or greater than n , okay. These were just given vectors.

So m can be anything but we will simplify it. So the way we will do is you can think of it as a modification of column operations, okay. So we will actually take combinations, linear combinations of the columns. And we will try to make this matrix triangular matrix, okay. But remember that you are only allowed integral combinations. You cannot use non-integers.

So the first thing that we will do is we will take the GCD of b_{11} the first row. So b_{11}, b_{21}, b_{m1} , okay. So let consider a combination of these b_i ones. So b_{11} to b_{m1} . Consider a combination which is equal to GCD of the first row, okay. So compute the GCD and a_i 's can be computed by for example, you can use Euclid's extended, extended Euclid GCD algorithm. So do that piece by piece, okay?

So simulate every step of the Euclid GCD algorithm on the columns. So let us write that down. Apply the extended Euclid GCD algorithm or let us say transformations on the columns, on the columns of the matrix B, okay fine. So what do you get? You get a transform matrix.

(Refer Slide Time: 06:56)

- Say the new columns are b'_1, \dots, b'_m .
 - Ensure: $(1,1)$ -th entry becomes g .
 Remaining entries in 1st-row become zero!
 $\Rightarrow B \mapsto B' := \begin{pmatrix} g & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots \end{pmatrix}_{n \times m}$
 - The transformation is $B' = B \cdot U$, where U follows each step of Euclid-gcd-algorithm.
 $\triangleright U$ is unimodular, i.e. $|U| = \pm 1 \Rightarrow U^{-1}$ integral.
 $\Rightarrow \mathcal{L}(B') = \mathcal{L}(B)$.
 Pf: $\begin{pmatrix} a \\ b \end{pmatrix} \rightarrow \begin{pmatrix} a \\ b - qa \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -q & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \end{pmatrix}; \left| \begin{pmatrix} 1 & 0 \\ -q & 1 \end{pmatrix} \right| = 1. \square$

So say the new columns are b_1 prime to b_m prime. So for example, if b_{11} is smaller than b_{21} then what you will do is you will actually compute the quotient of b_{21} divided by b_{11} , right. And that quotient you will multiply with b_{11} and then you will subtract it from b_{21} . So do that operation. And then you get a remainder and you get b_{11} . Then you divide b_{11} by the remainder and multiply by the quotient.

Take the difference. So keep on doing this. Ultimately what you will have is GCD g in the top left corner. So in this matrix b prime $(1, 1)$ -th entry becomes g , okay. Let us ensure this. Ensure that the top left corner becomes g after all these operations and the remaining entries will become zero right. So let us look at this. So the remaining entries in first row becomes zero, right.

So which means that now the matrix B looks like, so B will be transformed to B prime which looks like this. There is g and the remaining entries in the first row have vanished. In the first column you have some entries. They may be zero, they may not be zero, you do not know. And then you have this lower sub matrix, right. So this is n cross m . So you get this n cross m matrix B prime.

So what kind of this combined first step has achieved is it has made the first row in this normal form. So it is all, it is almost the zero row except the first entry being GCD. So what is this transformation? The transformation is B prime is basically B or you did column operations, right? So it is some multiplication by a matrix on the right. B times u where u is what?

So u follows Euclid's GCD algorithm. U follows each step of Euclid GCD algorithm? So what can you say about the determinant of u ? Right, that is the so that question is important. Because you want to claim that the lattice generated by B and the lattice generated by B prime they are the same, okay. If they were not the same then it will not be a reduction, right.

Then looking at lattice B prime, we will not be able to get SVP of lattice of B . So now we want to show something special about u so that the lattices are the same. So the claim is that u is unimodular. What is unimodular? That the determinant is plus minus 1. That is the determinant of u is plus or minus 1, which means that its inverse, so u inverse is integral.

So once we show this you can see that the lattices of B prime and B are the same because the matrix you are applying its inverse is also an integral matrix, right. So any combination that you take off the integral combination of vectors or columns of B , that corresponds to something in B prime and vice versa, right. Because you can multiply B prime with u inverse.

So this will imply that the lattice generated by B prime is the same as the lattice generated by B . So what is the proof idea? Well, the proof is just that the division step, one step of Euclid GCD is division step. So $a \ b$ is going to $a \ b - qa$ which is equal to $a \ b$ times $\begin{pmatrix} 1 & 0 \\ 0 & 1 - q \end{pmatrix}$, right. And if you consider the determinant, so determinant of $\begin{pmatrix} 1 & 0 \\ 0 & 1 - q \end{pmatrix}$ this determinant is 1, okay.

So this is the only thing which is needed to prove that u is unimodular. Because in one step of Euclid GCD you are doing division. You are transforming two numbers $a \ b$. So a vector $a \ b$ to this by multiplying with a matrix, which is which has determinant 1.

And when you repeat this you are doing you are multiplying such matrices so the determinant will remain 1 or -1 okay.

It may become -1 because you may be permuting the columns also, okay. So when you permute column then you may get a sign. So it will always remain plus minus 1 in the kind of transformations that we did, okay. So that is it, right. That finishes the proof of, well that finishes the first step. Now we have to focus on this blue part the sub matrix.

(Refer Slide Time: 15:50)

• Repeatedly apply this Gauss-Euclid trick, to get a matrix

$$\tilde{B} = \left(\begin{array}{c|c} A_{m' \times m'} & 0 \end{array} \right)_{n \times m}$$

where A is lower-triangular & invertible.

▷ $\mathcal{L}(\tilde{B}) = \mathcal{L}(B)$.

▷ First m' columns of \tilde{B} form an \mathbb{R} -basis of size $m' \leq \min(h, m)$. □

— So, we work with \mathbb{R} -l.i. $b_1, \dots, b_{m'} \in \mathbb{Z}^n$.

So repeatedly apply. So this trick is by Gauss and Euclid. So apply this trick to get a matrix B tilde. So what is happening? Row is becoming 0, right. Except in the first entry the row is 0. So you are getting a lower triangular matrix. Ultimately, you will stop at this. So some A . So in the end, you will have only 0 entries and the first columns, first few columns will be nonzero. Let us say m prime many, okay.

So we have 0 in the end and the matrix A is a square matrix with which is actually lower triangular, where A is lower triangular and most importantly, the lattice remains unchanged, okay. So we have a very nice structural property here. What we have shown is that given any input vectors with integral entries, we can reduce, without changing the lattice we can reduce them to linearly independent.

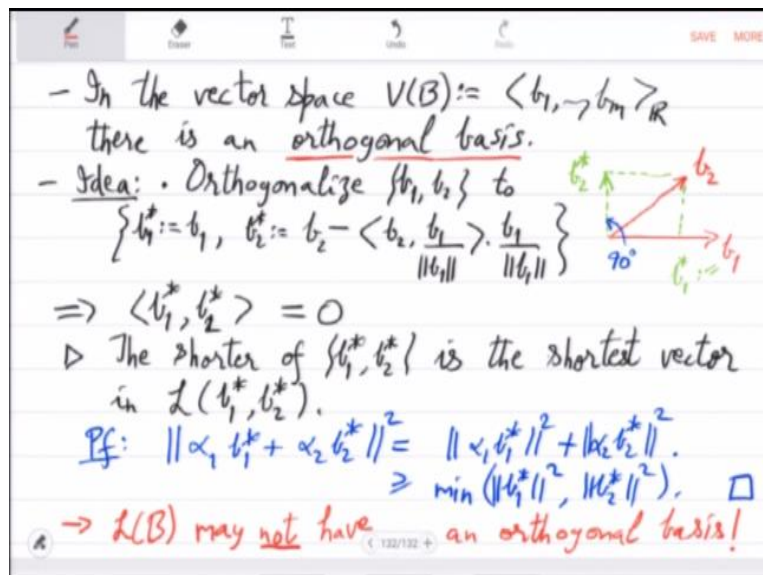
So A is lower triangular and invertible, okay. So the rank of this remaining part is m prime. So whatever was the rank of the input vectors that rank many vectors you will

get. So first m prime columns form a basis over reals. Note that m prime is at most m and also since we are talking about rank now it cannot exceed n , okay and it spans lattice that we already said. So this finishes the proof of the preprocessing step.

So from now on what we will do is so we work with R linearly independent vectors b_1 to b_m in the ambient space \mathbb{Z} to the n and we are interested in the lattice generated by them and we want a short vector, okay. So now we will do some we will actually revise some linear algebra. So given a basis of a vector space of a real vector space what you know is you can find another basis by linear transformations.

You can find another basis which will be orthogonal which means that the vectors they will be perpendicular to each other, okay. So the angle between the basis vectors in the orthogonal basis is 90 degrees. So how is that done? So let us revise the idea, basic idea of that.

(Refer Slide Time: 21:28)



- In the vector space $V(B) := \langle b_1, \dots, b_m \rangle_{\mathbb{R}}$ there is an orthogonal basis.
 - Idea: • Orthogonalize $\{b_1, b_2\}$ to $\{b_1^*, b_2^*\}$ where $b_1^* := b_1$, $b_2^* := b_2 - \langle b_2, \frac{b_1}{\|b_1\|} \rangle \cdot \frac{b_1}{\|b_1\|}$.
 $\Rightarrow \langle b_1^*, b_2^* \rangle = 0$
 \triangleright The shorter of $\{b_1^*, b_2^*\}$ is the shortest vector in $\mathcal{L}(b_1^*, b_2^*)$.
 Pf: $\| \alpha_1 b_1^* + \alpha_2 b_2^* \|^2 = \| \alpha_1 b_1^* \|^2 + \| \alpha_2 b_2^* \|^2 \geq \min(\|b_1^*\|^2, \|b_2^*\|^2)$. \square
 $\rightarrow \mathcal{L}(B)$ may not have an orthogonal basis!

So in the vector space generated by this B . So b_1 to b_m okay using coefficients from reals there is an orthogonal basis. This is a critical, this is a very important property in real vector spaces and the idea can be seen pictorially. So what you do is you have this vector b_1 and you have some vector b_2 . So you project b_2 on b_1 . Okay you project this. Basically look at the component of b_2 parallel to b_1 and subtract that component.

So when you subtract that component you will get this vector which is the component of b_2 orthogonal to b_1 . Okay, so pick this. Call this b_2^* and let us call b_1 b_1^* , right. So this is the starting point of Gram - Schmidt orthogonalization algorithm, right. So you are given vectors, you start with the first two. You are given a basis in fact, start with the first two. Remove, project b_2 and b_1 and remove that component from b_2 .

So what you are left with is called b_2^* and that is orthogonal to b_1^* , right. So the important thing here is this 90 degrees, right. This orthogonality is what we want. So this two dimensional idea is the most basic idea and then we will build on this. We will make it higher dimensional. So let us make it symbolic first. So orthogonalize the vectors b_1, b_2 to b_1^* is just b_1 , b_2^* is b_2 .

From that you remove project b_2 on b_1 in that direction right. So you have to take the unit vector. So it is this. So basically look at the direction b_1 and project b_2 on that. That is the component b_2, b_1 over norm of b_1 . And then in that direction you do the subtraction. So you can check that $b_1^* b_2^*$ inner product is zero, okay. This follows easily. And second thing is that the shorter of b_1^*, b_2^* is the shortest vector in the lattice generated by the two, okay.

So the, why is that? So this is an important point. This is an this is basically a special case of lattices where the generating set or the basis is orthogonal. So in that case SVP problem is very simple. You just look at the shortest basis, shortest basis vector. That will be the that will be the SVP. That will be the shortest vector in the lattice in fact. The proof is just by looking at combination.

So α_1, b_1^* plus α_2, b_2^* if you look at the norm, again Euclidean norm. So this is just sum of squares, okay. So now since so any vector in the lattice α_1, α_2 are integers. So α_1, α_2 are at least I mean either one of them is zero or at least 1 right. So if α_1 for example is 1 or more then this will contribute the length of b_1^* square. So any combinations length will be at least the smallest vector, okay.

This is an important thing to remember, right. So this is the sum of square property which is happening because of orthogonality and this shows that lattices of orthogonal vectors shortest vector is very easy to find. When you are given an arbitrary basis of a lattice it may be impossible to so there may not be an orthogonal basis, right. That is the issue. Lattice B may not have an orthogonal basis.

So that is the hard case. So what do you do? So since we are only interested in approximation of the shortest vector, so we will compute the Gram - Schmidt orthogonalization anyways and then try to use it to approximate, estimate the shortest vector okay. So let us look at Gram - Schmidt in general, higher dimension.

(Refer Slide Time: 29:39)

Gram-Schmidt Orthogonalization (GSO):

- 1) Let $b_1^* := b_1$.
- 2) For $2 \leq i \leq m$, do

$$b_i^* := b_i - \sum_{j=1}^{i-1} \left(\frac{\langle b_i, b_j^* \rangle}{\|b_j^*\|^2} \right) \cdot b_j^*.$$

▷ GSO gives an orthogonal basis.

Lemma: For $0 \neq b \in \mathcal{L}(b_1, \dots, b_m)$, $\|b\| \geq \min_i \|b_i^*\|$.

Pf: Let $b = \sum_{i=1}^m \lambda_i b_i$ for $\lambda_i \in \mathbb{Z}$ & $\lambda_m \neq 0$.

$$\Rightarrow b = \lambda_1 b_1^* + \lambda_2 (b_2^* + \mu_{2,1} b_1^*) + \dots + \lambda_m (b_m^* + \mu_{m,m-1} b_{m-1}^* + \dots + \mu_{m,1} b_1^*)$$

$$\Rightarrow \|b\|^2 = (\dots)^2 \|b_1^*\|^2 + \dots + \lambda_m^2 \|b_m^*\|^2 \geq \lambda_m^2 \|b_m^*\|^2$$

So we will call it GSO. So this is just an extension of the picture we saw. So the first vector we will call b_1 , b_1^* . And then the other vectors we will subtract the projections. So this is just a for loop. So b_i^* is from b_i you subtract the components of the previous orthogonal vectors. So b_1 to b_{i-1}^* , okay. So b_1 star to b_{i-1}^* star is the space for which we have already computed an orthogonal basis.

So you project b_i on that and just remove that projection. So obviously what remains will be orthogonal to the first $i-1$ vector space. So that gives you b_i^* . This quantity will be very important and so we will call this quantity, we will give it a name. This is μ_{ij} okay. So what combination are you taking of b_j^* ? So that is μ_{ij} . Okay. So let us prove a, well so it is an easy observation, you can prove it as a small exercise that GSO gives an orthogonal basis, okay.

This you must have seen in basic linear algebra courses over reals. Is there any relationship of our shortest vector to b_1^* . Okay, so for every nonzero vector in the lattice let us call it b ; b is at least b_1^* . This is a claim. So this relates kind of the length of the vectors in the lattice to what you get after GSO, okay. So if you just pick the minimum vector in the orthogonal basis you at least know that every nonzero vector in the lattice is lower bounded by this.

So the shortest vector cannot be smaller than this. But we do not know how good an estimate this is. Okay, but let us start with this property. It is a simple proof. So consider this combination of b_i 's. Since b_i 's we are assuming linearly independent λ_i 's will be unique and they are integers for integral λ_i 's. For λ_m integral and let us say λ_m is nonzero.

Okay, something has to be nonzero because b is nonzero. So let us say λ_m is nonzero. Let us now use the GSO basis. So let us write b as $\lambda_1 b_1^*$. Well, because b_1^* is b_1 . What is b_2^* . So b_2^* is or well what is b_2 . So b_2 is b_2^* plus $\mu_{21} b_1^*$. What is b_3^* ? So this pattern will continue, right. b_3 will be b_3^* plus this $\mu_{31} b_1^*$ plus $\mu_{32} b_2^*$ and so on.

So in the end you will just get $\lambda_m b_m^*$ plus m projected on $m-1$, m projected on 1 fine, okay. And then you can look at the norm square. So now since you have written b as a combination of orthogonal vectors you can use the sum of squares, right. So you will get the coefficient of b_1^* squared the last one which is b_m^* . So what is the coefficient of b_m^* square? b_m^* square is λ_m^2 square, right.

So you get these sum of squares b_1^* square, b_2^* square, b_m^* square with appropriate coefficients depending on the λ 's and so this is clearly because of the real squaring, this is at least $\lambda_m^2 b_m^*$ square.

(Refer Slide Time: 38:42)

$\Rightarrow \|b\| \geq |\lambda_m| \cdot \|b_m^*\| \geq \|b_m^*\| \geq \min_i \|b_i^*\|$

- So, L^3 -algo. tries to make the angles, in a basis of $L(B)$, close to 60° . *← pseudo-orthogonal!*
 \rightarrow In that basis it'll pick the first!

- Defn: L^3 finds a reduced basis of $L(B)$. These are lattice elements $\{c_1, \dots, c_m\} \subset L(B)$ s.t.

c_{i+1} not much smaller than c_i
angles $\approx 60^\circ$

(i) $\forall i, \|c_i^*\|^2 \leq \frac{4}{3} \cdot \|c_{i+1}^* + \mu_{i+1,i} c_i^*\|^2$
 (ii) $\forall i \geq j, |\mu_{ij}| \leq \frac{1}{2}$, where $\mu_{ij} := \frac{\langle c_i, c_j^* \rangle}{\|c_j^*\|^2}$.

Okay, so which means that we have shown the norm of b is at least λ_m times b_m^* which is at least b_m^* . So we have shown that the length of b is at least as much as this b_m^* which contributes as a component to b , okay which means that the min property is satisfied. So this is at least the min of b_i^* right, which is what we wanted to show. So this finishes the proof of the lemma.

That min of b_i^* is a lower bound on the shortest vector of the lattice. So what will L^3 do? So the L^3 algorithm tries to make the angles in a basis of lattice close to 60 degrees, okay. So instead of trying to achieve 90 degrees which is what orthogonalization is. L^3 algorithm actually tries to get to 60 degrees. So you can call this pseudo-orthogonalization, okay.

So it tries to pseudo-orthogonalize the basis of the lattice because orthogonal basis may not exist, right. So you can only work with some approximation. And then in that basis in that basis it will pick the first one okay. So just like we saw in the previous lemma that the smallest b_i^* may be a good vector to start with or a good length to start with this is what L^3 algorithm will do.

It will actually try to orthogonalize as much as possible around 60 degrees angle and then in that basis once it has found the basis in that basis it will just pick the smallest one and output. Okay, this is roughly the idea which we will now implement, okay. So let us define this basis now properly. So L^3 finds a reduced basis of the lattice b_1 to b_m . So these are these are lattice elements say c_1 to c_m .

There will be m many because you started with m many, number cannot change and they will be integral. So remember that you want the same lattice. So they should all be in the lattice such that so the condition, first condition is a bit mysterious. But you will see why it is important when we analyze. Okay, so this is the first condition. This says that essentially so you first look at the GSO of c_1 to c_m .

So let us say it is c_1 star to c_m star. Okay, let us just look at c_1 star and c_2 star. So what is this condition saying for that? So this condition is saying that this c_2 star should be pretty close to what it would have been if you were able to implement GSO in the lattice. Okay, so if you were able to implement GSO, then this RHS this c_i plus 1 star plus $\mu_{i+1} c_i$ star this would have been equal to c_{i+1} , right.

So this is actually comparing the kind of the length of c_{i+1} with c_2 with c_1 . And it basically is it is a way to in a way it is trying to ensure that c_2 is not much smaller than c_1 , okay. The ratio is kind of three fourth. c_2 should be at least well square root of 3 by 2 times c_1 . Okay, this is so it is a bit hard to motivate this. You will understand this only in the analysis.

And the second condition it wants is that this after GSO and c_1 to c_m this so called this thing which we are calling reduced basis the μ_{ij} should be at most half where μ_{ij} is defined as before in GSO, okay. So the first condition is kind of saying that c_{i+1} is not much smaller than c_i . Let us say that and the second condition is saying so μ_{ij} if you think of it as $\cos \theta$ right, this $\cos \theta$ less than equal to half is or $\cos \theta$ around half would mean θ is around 60 degrees.

So angles around 60 degrees, okay. So this is what we define as a reduced basis. So in a lattice, can you find a basis where the vectors in this order their GSO is the c_{i+1} vector is not much smaller than c_i . So kind of increasing in length and second is that the angle of c_{i+1} on the previous subspace is around 60 degrees. So kind of orthogonal also, okay. So kind of sorted and kind of orthogonal.

These are the two ideas that we are trying to approximate here. Well, we of course do not know whether reduced basis exists, but our algorithm will construct it. So in

particular, it will show that this exists as well and efficiently computable. Let us see the implications of this.

(Refer Slide Time: 49:08)

Handwritten mathematical derivation on a digital whiteboard:

- (i), $\Rightarrow \|c_i^*\|^2 \leq \frac{4}{3} \|c_{i+1}^*\|^2 + \frac{1}{3} \|c_i^*\|^2$
- (ii) $\Rightarrow \|c_i^*\| \leq \sqrt{2} \cdot \|c_{i+1}^*\|$
- $\Rightarrow \|c_i^*\| \leq \min_i \{ \sqrt{2}^{i-1} \cdot \|c_1^*\| \} \leq \sqrt{2}^{m-1} \cdot \|c_1^*\|$ (by (i))
- Shortest-length in lattice $:= \lambda(L(c_1, \dots, c_m)) \geq \|c_1^*\|$ ($\because c_1^* = c_1 \in L$)
- $\& \|c_1^*\| \leq 2^{\frac{m-1}{2}} \cdot \lambda(L(c_1, \dots, c_m))$ [by (i)]
- $\triangleright c_1$ estimates $\lambda(L)$ by a factor of $2^{(m-1)/2}$.

So this in particular implies the condition 1 for example. Condition 1 implies that c_i^* star square is less than equal to 4 by 3 c_{i+1}^* star plus this 4 by 3 μ square, right. And μ by second condition is less than equal to half in magnitude. So that is one-third thing. Okay, which implies that c_i^* star is at most c_{i+1}^* star right, which is again saying the thing that I was saying before that c_{i+1}^* star is not much smaller than c_i^* star.

And equivalently c_i^* star is not much bigger than say c_{i+1}^* star, right. So kind of order is being maintained by this and the impact of this is when you repeatedly do this what you get is that c_1^* star, what is c_1^* star? c_1^* star is less than equal to square root two times c_2^* star. Then square root 2 square right and so on. So it will as you increase i you will get this ultimately, okay.

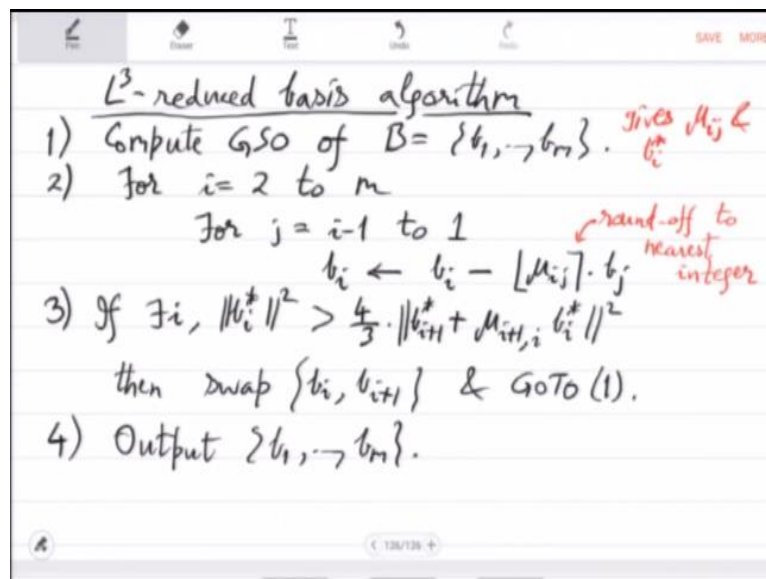
So c_1^* star is smaller than square root of 2 raised to $m - 1$ times c_i^* star for all i . So now if you use this orthogonality of c_1^* star to or you use the lemma that we had shown before right that in the lattice generated by c_1 to c_m . So by the lemma before we get that the shortest vector in the lattice generated by c_1 to c_m will be at least obviously, c_1^* star. c_1 is equal to c_1^* star right. So this is a lattice element.

So the shortest vector is at least c_1 star. Since c_1 star is equal to c_1 is in the lattice. So the shortest vector is at least c_1 star and c_1 star is less than equal to this thing 2^{m-1} raised to $m-1$ by 2 times the shortest vector, okay. This follows from the above inequality because, you know the orthogonal basis comparison of c_1 star with other c_i stars.

So using that you can deduce that c_1 star cannot exceed 2^{m-1} by 2 times the shortest vector because it is unable to exceed that for any c_i star, right. So this implies together, all in all it implies that c_1 is a good c_1 is in the lattice and it estimates the shortest vector well by this factor. Oh, maybe I did not define λ_1 . Let me define that also. So λ_1 is shortest length in the lattice.

So this means that c_1 star estimates estimate shortest length by a factor of 2^{m-1} by 2 which is actually c_1 , okay. So the first vector in the reduced basis is what you will output. Once you have a reduced basis you will just output this. So that is the key takeaway out of the definition, okay. So now we move to the final question, which is how will you, why does reduced basis exist and if it exist how will you find it? So we will answer both these questions together and very constructively.

(Refer Slide Time: 55:51)



So this is L^3 's reduced basis algorithm, okay. So the first step is compute GSO of the vectors given to you. So you will get b_1 star to b_m star. This can be done efficiently. Next is you, okay let me first give the step. Then I will explain it. So say i

is equal to 2. So what you will do is you will read calculate b_2 . You will change b_2 , transform it by, see the point is that okay what is the output of step 1?

So this gives μ_{ij} and b_i^* s. But μ_{ij} 's are real numbers, right. They may not be integers. So as a first attempt what we will try is we will round off μ_{ij} to an integer and just remove that component, okay. So let us do that. So μ_{ij} let us round it off. So round off to the nearest integer, okay. So this is a transformed b_i . So we are trying to get to b_i^* but we may not be able to do that.

But we will just subtract whatever nearest integer multiple of b_j we can which is this. So this will take care of the μ_{ij} 's being reduced to half or less, right? Remember condition 2 we wanted this projection let us say constants μ to be less than equal to half. So this rounding off will make sure that it is less than equal to half. But the first condition may still not be satisfied.

So let us see whether it is satisfied or not. So suppose if there exists an i says that it is violated, okay. So LHS should have been less than equal to this for condition 1, but suppose it is not, suppose it is violated. So, if it is violated what should you do? So, the simple hack is you swap b_i and b_{i+1} . So you will swap them. But remember that GSO is very sensitive to order of b_1 to b_m in what order you are doing the computation.

Now since you have swapped b_i with b_{i+1} , this will change the GSO calculation, so you have to go back to step 1. Okay, so go back to step 1. Again, you will try to approximate the b_i^* in step 2 and after you have done, after you have covered all the b_i 's, then you will move to step 3 where you will try to ensure condition 1. If it is violated you will again swap, again go to step 1, okay.

So you will keep doing this and then you will output the answer. Okay, so this is the reduced basis algorithm. At this point it is not clear why it will work or whether it will ever stop. That is the first thing. If it stops, why is b_1 to b_m a reduced basis? That is not clear. So we will do this next time.