Babai's Nearest Plane Algorithm

Eric Ren, Sage Binder, and Katherine Kosaian

March 17, 2025

Abstract

γ-CVP is the problem of finding a vector in L that is within γ times the closest possible to t, where L is a lattice and t is a target vector. If the basis for L is LLL-reduced, Babai's Closest Hyperplane algorithm solves γ-CVP for $\gamma = 2^{n/2}$, where n is the dimension of the lattice L, in time polynomial in n. This session formalizes said algorithm, using the AFP formalization of LLL [2, 1] and adapting a proof of correctness from the lecture notes of Stephens-Davidowitz [4].

Contents

1 Introduction

The (exact) *closest vector problem* (CVP) is the problem of finding the closest vector within a lattice L to a target vector t. This is equivalent to finding the shortest vector in the *lattice coset* $L - t := \{l - t : l \in L\}$. There is a corresponding family of weaker problems, γ-CVP (where γ is some real parameter), where one needs only find a vector in $L - t$ whose length is at most γ times the shortest possible. Through a reduction to the *shortest* *vector problem* [4], solutions to these problems may be used to factor rational polynomials. This problem is therefore of cryptographic interest.

Although exact CVP (or 1-CVP) is NP-Complete [3], Babai's Nearest Plane Algorithm solves $2^{n/2}$-CVP, where n is the dimension of L, in polynomial time, provided that L is presented using an LLL-reduced basis with parameter $\alpha = 4/3$. The proof in this document is mostly a straightforward algebraicization of the proof in Stephens-Davidowitz' lecture notes. It makes use of the coordinate systems defined by the original basis (denoted β) and the Gram-Schmidt orthogonalization of that basis (denoted $\tilde{\beta}$). Let $[u]_\beta$ denote the representation of a vector u under β, with coordinates $[u]_\beta^j$; $j = 1, ..., n$ (likewise for $\tilde{\beta}$). Also, let s_i denote the output of the algorithm after step i and let d be the shortest lattice coset vector, as witnessed by the vector v. The proof works by analysing the coordinates of $[s_n]_{\tilde{\beta}}$, showing that all are at most $1/2$ and that some later coordinates are exactly those of $[v]_{\tilde{\beta}}$.

The algorithm modifies coordinate $n-i$ in both bases for the last time in step i (formalized in lemma `coord_invariance`), during which both coordinates are decreased below $1/2$ (formalized in lemma `small_coord`). Combined, these facts imply that the output s_n has $\left|[s_n]_{\tilde{\beta}}^j\right| \leq 1/2$ for all indices j.

Since $\tilde{\beta}$ is orthogonal, we have

$$\|s_n\|^2 = \sum_{i=1}^{n} \left([s_n]_{\tilde{\beta}}^i \|\tilde{\beta}_i\|\right)^2, \tag{1}$$

so the preceding coordinate bounds $\|s_n\|^2$ by $\frac{1}{4} \sum_{i=1}^{n} \|\tilde{\beta}_i\|^2$. If the $\tilde{\beta}_i$ are all short compared to d, this bound suffices. In fact, if there is any short vector $\tilde{\beta}_I$ in $\tilde{\beta}$ then because β is LLL-reduced, any vector preceding $\tilde{\beta}_I$ in $\tilde{\beta}$ will not be much longer. This bounds the first I terms in Equation 1. By selecting I maximal, we may assume that $\tilde{\beta}$ ends in a series of $n - I$ long vectors. In this case it can be shown $[v]_{\tilde{\beta}}^j$ and $[s_n]_{\tilde{\beta}}^j$ differ by an integral amount for $j = I+1, ..., n$. Therefore, if $[v]_{\tilde{\beta}}^j$ and $[s_n]_{\tilde{\beta}}^j$ differ at all, they differ by at least 1, which would mean $\left|[v]_{\tilde{\beta}}^j\right| \geq 1/2$, since $\left|[s_n]_{\tilde{\beta}}^j\right| \leq 1/2$. This would force v to be longer than d, a contradiction. So $[v]_{\tilde{\beta}}^j = [s_n]_{\tilde{\beta}}^j$ for $j = I + 1, ...n$, which gives a tighter bound on the last $n - I$ terms in equation 1.

Precisely, let I denote $\max\{i : \|\tilde{\beta}_i\| \leq 2d\}$, meaning for all indices $j > I$, $\|\tilde{\beta}_j\| > 2d$. Now, for all $j > I$, $d^2 = \|v\| \geq ([v]_{\tilde{\beta}}^j)^2 \|\tilde{\beta}_j\|^2 > ([v]_{\tilde{\beta}}^j)^2 \cdot 4d^2$, meaning $1/4 > (\tilde{\beta}^j)^2$, or $1/2 > \left|[v]_{\tilde{\beta}}^j\right|$. Since $\left|[s_j]_{\tilde{\beta}}^j\right| \leq 1/2$ from the previous section, $\left|[v]_{\tilde{\beta}}^j - [s_j]_{\tilde{\beta}}^j\right| < 1$. Using properties of the change-of-basis between $\beta, \tilde{\beta}$ formalized in the LLL AFP session, we show that $[v]_{\tilde{\beta}}^j - [s_j]_{\tilde{\beta}}^j =$

$[v]^j_\beta - [s_j]^j_\beta = [v - s_j]^j_\beta$, so that $\left|[v - s_j]^j_\beta\right| < 1$ But since $v - s_j$ lies in the lattice, $[v - s_j]^j_\beta$ is integral, so $\left|[v - s_j]^j_\beta\right| = 0$, meaning $[v]^j_{\tilde{\beta}} = [s_j]^j_{\tilde{\beta}}$. Lemma `coord_invariance` gives that $[v]^j_{\tilde{\beta}} = [s_j]^j_{\tilde{\beta}} = [s_n]^j_{\tilde{\beta}}$. This is formalized by lemma `correct_coord`.

Now $\|s_n\|^2 = \sum_{i=1}^{n}([s_n]^i_{\tilde{\beta}}\|\tilde{\beta}_i\|)^2$, since $\tilde{\beta}$ is orthogonal. Splitting the sum around I equates this to $\sum_{i=1}^{I}([s_n]^i_{\tilde{\beta}})^2 + \sum_{i=I+1}^{n}([s_n]^i_{\tilde{\beta}})^2$. Lemma `small_coord` bounds the terms in the first sum by $\|\tilde{\beta}_i\|^2/4$, while lemma $\texttt{correct}_\texttt{c}\texttt{oord}$ bounds the terms in the second sum by d^2, giving $\|s_n\|^2 \leq (n - I)d^2 + \sum_{i=1}^{I}\|\tilde{\beta}_i\|^2/4$. If β is LLL-reduced with parameter α, $\|\tilde{\beta}_i\|^2 \leq \alpha^I\|\tilde{\beta}_I\|^2$ for all $i \leq I$, which, by the definition of I, is at most $4d^2$. So $\|s_n\|^2 \leq ((n - I) + I\alpha^I)d^2 \leq n\alpha^n d^2$. The standard choice of $\alpha = 4/3$ gives $\|s_n\|^2 \leq 2^n d^2$. All of this is formalized in the final section, which culminates in the main theorem.

To avoid having to prove that a shortest vector exists, we use the definition $\inf\{\|u - t\| : u \in L\}$ for d instead of $\min\{\|u - t\| : u \in L\}$ and rephrase the arguments above to allow $\|v\|$ to exceed d by a small constant factor ϵ. This workaround and its details are contained in the section on the closest distance and negligibly change the rest of the proof.

```
theory Babai-Algorithm

imports LLL-Basis-Reduction.LLL
 HOL.Archimedean-Field
  HOL−Analysis.Inner-Product

begin
fun calculate-c:: rat vec ⇒ rat vec list ⇒ nat => int where
  calculate-c s L1 n  = round ((s · (L1!( (dim-vec s) − n ) ) ) / (sq-norm-vec (L1!(
(dim-vec s) − n ) ) ) )

fun update-s:: rat vec ⇒ rat vec list ⇒ rat vec list ⇒ nat ⇒ rat vec where
  update-s sn M Mt n = ( (rat-of-int (calculate-c sn Mt n)) ·v M!((dim-vec sn)−n))



fun Babai-Help:: rat vec ⇒ rat vec list ⇒ rat vec list ⇒ nat ⇒ rat vec where
 Babai-Help s M Mt 0 = s |
 Babai-Help s M Mt (Suc n) = (let B= (Babai-Help s M Mt n) in B− (update-s B
M Mt (Suc n)) )

definition Babai:: rat vec ⇒ rat vec list ⇒ rat vec where
 Babai s M = Babai-Help s M (gram-schmidt (dim-vec s) M) (dim-vec s)
```

end
theory *Babai*
  imports *Babai-Algorithm*

begin

This theory contains the proof of correctness of the algorithm. The main theorem is "theorem Babai-Correct", under the locale "Babai-with-assms". To use the theorem, one needs to show that lattice, the vectors in the lattice basis, and the target vector all have the same dimension, that the lattice basis vectors are linearly independent and form an invertible matrix, and that the lattice basis is LLL-weakly-reduced.

2 Locale setup for Babai

locale *Babai* =
  fixes *M* :: *int vec list*
  fixes *t* :: *rat vec*
  assumes *length-M*: *length M* = *dim-vec t*
begin

abbreviation *n* **where** *n* ≡ *length M*
definition *α* **where** (*α*::*rat*) = *4/3*
sublocale *LLL n n M α*.

abbreviation *coset*::*rat vec set* **where** *coset*≡{(*map-vec rat-of-int x*)−*t*|*x*. *x*∈*L*}
abbreviation *Mt* **where** *Mt* ≡ *gram-schmidt n* (*RAT M*)

definition *s* :: *nat* ⇒ *rat vec* **where**
  s i = *Babai-Help* (*uminus t*) (*RAT M*) *Mt i*

definition *closest-distance-sq*:: *real* **where**
  closest-distance-sq = *Inf* {*real-of-rat* (*sq-norm x*::*rat*) |*x*. *x* ∈ *coset*}
end

Locale setup with additional assumptions required for main theorem

locale *Babai-with-assms* = *Babai*+
  fixes *mat-M mat-M-inv*:: *rat mat*
  assumes *basis*: *lin-indep M*
  defines *mat-M* ≡ *mat-of-cols n* (*RAT M*)
  defines *mat-M-inv* ≡

(*if* (*invertible-mat mat-M*) *then SOME B*. (*inverts-mat B mat-M*) ∧ (*inverts-mat mat-M B*) *else* (0_m *n n*))
  assumes *inv*:*invertible-mat mat-M*
  assumes *reduced*:*weakly-reduced M n*
  assumes *non-trivial*:*0*<*n*
begin

lemma *dim-vecs-in-M*:
  shows ∀ *v* ∈ *set M*. *dim-vec v* = *length M*
  using *basis* **unfolding** *gs.lin-indpt-list-def* **by** *force*

lemma *inv1*:*mat-M* ∗ *mat-M-inv* = 1_m *n*
proof−
  have *dim-m*:*dim-row mat-M* = *n* **using** *dim-vecs-in-M* **unfolding** *mat-M-def* **by** *fastforce*
  then have *inverts-mat mat-M mat-M-inv* **using** *inv*
  unfolding *mat-M-inv-def*
  by (*smt* (*verit*, *ccfv-SIG*) *invertible-mat-def some-eq-imp*)
  then show *?thesis* **using** *dim-m* **unfolding** *inverts-mat-def* **by** *argo*
qed

lemma *inv2*:*mat-M-inv* ∗ *mat-M* = 1_m *n*
proof−
  have *dim-m*:*dim-col mat-M* = *n* **unfolding** *mat-M-def* **by** *fastforce*
  have *inverts-mat mat-M-inv mat-M* **using** *inv*
  unfolding *mat-M-inv-def*
  by (*smt* (*verit*, *ccfv-SIG*) *invertible-mat-def some-eq-imp*)
  then have *inv*:*mat-M-inv* ∗ *mat-M* = 1_m (*dim-row mat-M-inv*)
    unfolding *inverts-mat-def* **by** *blast*
  then have *dim-n*:*dim-col* (1_m (*dim-row mat-M-inv*)) = *n*
    using *dim-m index-mult-mat*(*3*)[*of mat-M-inv mat-M*] **by** *fastforce*
  have (*dim-row mat-M-inv*)= *n*
  proof(*rule ccontr*)
    assume (*dim-row mat-M-inv*)≠ *n*
    then have *dim-col* (1_m (*dim-row mat-M-inv*)) ≠ *n*
      by *auto*
    then show *False* **using** *dim-n* **by** *blast*
  qed
  then show *?thesis* **using** *inv* **by** *argo*
qed

sublocale *rats*: *vec-module TYPE*(*rat*) *n*.

lemma *M-dim*: *dim-row mat-M* = *n dim-col mat-M* = *n*
  apply (*metis index-mult-mat*(*2*) *index-one-mat*(*2*) *inv1*)
 by (*metis index-mult-mat*(*3*) *index-one-mat*(*3*) *inv2*)

lemma *M-inv-dim*: *dim-row mat-M-inv* = *n dim-col mat-M-inv* = *n*
  apply (*metis M-dim*(*1*) *index-mult-mat*(*2*) *inv1 inv2*)
 by (*metis index-mult-mat*(*3*) *index-one-mat*(*3*) *inv1*)

lemma *Babai-to-Help*:
 shows *s n* = *Babai-Algorithm.Babai* (*uminus t*) (*RAT M*)
 using *Babai.Babai-def Babai.s-def Babai-Algorithm.Babai-def Babai-axioms* **by** *force*

3 Coordinates

This section sets up the use of the lattice basis and its GS orthogonalization as coordinate systems and some properties of that coordinate system. The important lemma here is coord-invariance, which shows that after step i of the algorithm, all coordinates (in both systems) after n-i are invariant.

definition *lattice-coord* :: *rat vec* ⇒ *rat vec*
 where *lattice-coord a* = *mat-M-inv* $*_v$ *a*

lemma *dim-preserve-lattice-coord*:
 fixes *v*::*rat vec*
 assumes *dim-vec v*=*n*
 shows *dim-vec* (*lattice-coord v*) = *n* **unfolding** *lattice-coord-def mat-M-inv-def* **using** *M-inv-dim*
 by (*simp add*: *mat-M-inv-def*)
lemma *vec-to-col*:
 assumes *i* < *n*
 shows (*RAT M*)!*i* = *col mat-M i*
 unfolding *mat-M-def*
 by (*metis Babai-with-assms-axioms Babai-with-assms-axioms-def Babai-with-assms-def M-dim*(*2*)
  assms cols-mat-of-cols cols-nth gs.lin-indpt-list-def mat-M-def)

lemma *unit*:
 assumes *i* < *n*
 shows *lattice-coord* ((*RAT M*)!*i*) = *unit-vec n i*
 using *assms inv2* **unfolding** *lattice-coord-def*
 by (*metis M-dim*(*1*) *M-dim*(*2*) *M-inv-dim*(*2*) *carrier-matI col-mult2 col-one vec-to-col*)

lemma *linear*:
 fixes *i*::*nat*
 fixes *v1*::*rat vec*
 and *v2*:: *rat vec*

and *q*:: *rat*
assumes *dim-vec v1* = *n*
assumes *dim-2*:*dim-vec v2* = *n*
assumes *0*≤*i*
assumes *dim-i*:*i*<*n*
shows (*lattice-coord* (*v1*+(*q*$\cdot_v$*v2*)))$*i* = (*lattice-coord v1*)$*i* + *q*∗((*lattice-coord v2*)$*i*)
using *assms*
proof(−)
have *linear-vec*:(*lattice-coord* (*v1*+(*q*$\cdot_v$*v2*))) = (*lattice-coord v1*) + *q*$\cdot_v$((*lattice-coord v2*))
unfolding *lattice-coord-def*
by (*metis* (*mono-tags*, *opaque-lifting*) *M-inv-dim*(*2*) *assms*(*1*) *assms*(*2*) *carrier-mat-triv*
carrier-vec-dim-vec mult-add-distrib-mat-vec mult-mat-vec smult-carrier-vec)
then have *2*: (*lattice-coord* (*v1*+(*q*$\cdot_v$*v2*)))$*i*= ((*lattice-coord v1*) + *q*$\cdot_v$((*lattice-coord v2*)))$*i* **by** *auto*
also have *dim-v2*: *dim-vec* (*lattice-coord v2*) = *n* **using** *dim-preserve-lattice-coord dim-2* **by** *blast*
then have *i-in-range*: *i*<*dim-vec* (*q*$\cdot_v$(*lattice-coord v2*)) **using** *dim-v2 dim-i* **by** *simp*
also have *3*:((*lattice-coord v1*) + *q*$\cdot_v$((*lattice-coord v2*)))$*i*=(*lattice-coord v1*)$*i*+

(*q*$\cdot_v$(*lattice-coord v2*))$*i* **using** *i-in-range* **by** *simp*
also have *4*: (*q*$\cdot_v$(*lattice-coord v2*))$*i*=*q*∗(*lattice-coord v2*)$*i* **using** *i-in-range* **by** *simp*
thus *?thesis* **unfolding** *vec-def* **using** *linear-vec 2 3 4* **by** *simp*
qed

lemma *sub-s*:
fixes *i*::*nat*
assumes *0*≤*i*
assumes *i*<*n*
shows *s* (*Suc i*) = (*s i*) −
((*rat-of-int* (*calculate-c* (*s i*) *Mt* (*Suc i*))) $\cdot_v$ (*RAT M*)!((*dim-vec* (*s i*)) −(*Suc i*)))
using *assms Babai-Help.simps*[*of* −*t RAT M Mt*] **unfolding** *s-def*
by (*metis update-s.simps*)

lemma *M-locale-1*:
shows *gram-schmidt-fs-Rn n* (*RAT M*)
by (*smt* (*verit*) *M-dim*(*1*) *M-dim*(*2*) *carrier-dim-vec dim-col gram-schmidt-fs-Rn.intro in-set-conv-nth*
mat-M-def mat-of-cols-carrier(*3*) *subset-code*(*1*) *vec-to-col*)

lemma *M-locale-2*:
shows *gram-schmidt-fs-lin-indpt n* (*RAT M*)
using *basis M-locale-1 gram-schmidt-fs-lin-indpt.intro*[*of n* (*RAT M*)] **unfolding** *gs.lin-indpt-list-def*

```
  using gram-schmidt-fs-lin-indpt-axioms.intro by blast


lemma more-dim: length (RAT M) = n
  by simp

lemma Mt-gso-connect:
  fixes j::nat
  assumes j<n
  shows Mt!j = gs.gso j
proof(−)
 have Mt = map gs.gso[0..<n]
    using M-locale-1 gram-schmidt-fs-Rn.main-connect[of n (RAT M)]
    by fastforce
  then show ?thesis
    using assms
    by simp
qed

lemma access-index-M-dim:
  assumes 0 ≤ i
  assumes i < n
  shows dim-vec (map of-int-hom.vec-hom M ! i) =  n
  using assms dim-vecs-in-M
  by auto

lemma s-dim:
  fixes i::nat
  assumes i≤ n
  shows dim-vec (s i) = n ∧ (s i) ∈carrier-vec n
  using assms
  proof(induct i)
  case 0
  have unfold1:s 0 = Babai-Help (uminus t) (RAT M) Mt 0 unfolding s-def by
simp
 also have unfold2:Babai-Help (uminus t) (RAT M) Mt 0 = uminus t unfolding
Babai-Help.simps by simp
  also have unfold3:s 0 = uminus t using unfold1 unfold2 by simp
  also have dim-eq:dim-vec (s 0) = dim-vec (uminus t) using unfold3 by simp
 moreover have dim-minus:dim-vec (uminus t) = n by (metis index-uminus-vec(2)
length-M)
  then have dim-vec (s 0) = n
    using dim-eq dim-minus
    by simp
  then have (s 0) ∈ carrier-vec n
    using carrier-vecI[of (s 0) n]
    by simp
  then show ?case
    by simp
```

```
 next
  case (Suc i)
  then have leq: i≤n by linarith
   have sub:s (Suc i) = (s i) − ( (rat-of-int (calculate-c (s i) Mt (Suc i) ) ) ·v
(RAT M)!( (dim-vec (s i)) −(Suc i)))
    using sub-s Suc
    by auto
  moreover have prev-s-dim:(s i)∈carrier-vec n
    using Suc
    by simp
  moreover have dim-vec (s i)=n
    using Suc
    by simp
  then have 0≤(dim-vec (s i)) −(Suc i)∧ (dim-vec (s i)) −(Suc i)<n
    using Suc
    by linarith
  then have dim-m:(dim-vec ((RAT M)!( (dim-vec (s i)) −(Suc i)))) = n
    using access-index-M-dim[of (dim-vec (s i)) −(Suc i)]
    by simp
  then have dim-qm:dim-vec ( (rat-of-int (calculate-c (s i) Mt (Suc i) ) ) ·v
           (RAT M)!( (dim-vec (s i)) −(Suc i))) = n
    by simp
  then have final-dim:dim-vec ((s i) −
( (rat-of-int (calculate-c (s i) Mt (Suc i) ) ) ·v (RAT M)!( (dim-vec (s i)) −(Suc
i)))) = n
    using index-minus-vec(2) prev-s-dim dim-qm
    by metis
   show ?case
    using final-dim sub carrier-vecI[of s i n]
    by (metis carrier-vec-dim-vec)
 qed

lemma dim-vecs-in-Mt:
  fixes i::nat
  assumes i<n
  shows dim-vec (Mt!i) = n
  using Mt-gso-connect[of i] M-locale-1 assms gram-schmidt-fs-Rn.gso-dim
  by fastforce
lemma upper-tri:
  fixes i::nat
    and j::nat
  assumes j>i
  assumes j<n
  shows ((RAT M)!i)· (Mt!j) =0
proof(−)
   have (gs.gso j)· (RAT M)!i= 0
    using gram-schmidt-fs-lin-indpt.gso-scalar-zero[of n (RAT M) j i]
        Mt-gso-connect[of j]
        assms
```

```
      M-locale-2
      more-dim
    by presburger
  then have (Mt!j)· ((RAT M)!i)= 0
    using Mt-gso-connect[of j] assms
    by simp
  then show ?thesis
    using comm-scalar-prod[of (Mt!j) n ((RAT M)!i)]
          carrier-vecI[of (Mt!j) n]
          carrier-vecI[of ((RAT M)!i) n]
          access-index-M-dim[of i]
          dim-vecs-in-Mt[of j]
          assms
    by auto
qed
lemma one-diag:
  fixes i::nat
  assumes 0≤i
  assumes i<n
  shows ((RAT M)!i)· (Mt!i)=sq-norm (Mt!i)
proof(−)
   have mu:((RAT M)!i)·(Mt!i) = (gs.μ i i)∗sq-norm (Mt!i)
    using gram-schmidt-fs-lin-indpt.fi-scalar-prod-gso[of n (RAT M) i i]
          M-locale-2
          assms
          more-dim
          Mt-gso-connect
    by presburger
  moreover have gs.μ i i=1
    by (meson gs.μ.elims order-less-imp-not-eq2)
  then show ?thesis
    using mu
    by fastforce
qed


lemma coord-invariance:
  fixes j::nat
  fixes k::nat
  fixes i::nat
  assumes k≤j
  assumes j+i≤n
  assumes k>0
  shows (lattice-coord (s (j+i)))$(n−k) = (lattice-coord (s j))$(n−k)
    ∧ (s (j+i)) · Mt!(n−k)=(s j) · Mt!(n−k)
  using assms
proof(induct i)
  case 0
  show ?case by simp
```

```
next
  case (Suc i)
  have j+ (Suc i) = Suc (j+i) by simp
  then have 1:s (Suc (j+i)) =s (j + (Suc i)) by simp
  then have sub:s (Suc (j+i) ) =
    (s (j+i)) −( (rat-of-int (calculate-c (s (j+i)) Mt (Suc (j+i)) ) )
      ·v (RAT M)!( (dim-vec (s (j+i))) −(Suc (j+i)) ) )
    using sub-s[of j+i ] Suc(3) by linarith
  then have dim1: dim-vec (s (j + i)) = n
    using s-dim[of j+i] using Suc(3) by auto
  then have dim2: dim-vec
      (map of-int-hom.vec-hom M !
       (dim-vec (s (j + i)) − Suc (j + i))) = n
    using access-index-M-dim[of n − Suc (j + i)] Suc(3)
    by auto
  have k-in-range:0≤(n−k) ∧(n−k)<n using Suc(2) Suc(3) Suc(4)
    by simp
   have index-in-range:0≤(dim-vec (s (j+i))) −(Suc (j+i))∧(dim-vec (s (j+i)))
−(Suc (j+i))<n
    using Suc(3) s-dim[of j+i]
    by simp
  moreover have carriers: s (j+i) ∈ carrier-vec n∧
                       map of-int-hom.vec-hom M ! (dim-vec (s (j + i)) − Suc (j +
i))∈carrier-vec n
    using dim1 dim2
          carrier-vecI[of s (j + i) n]
          carrier-vecI[of map of-int-hom.vec-hom M ! (dim-vec (s (j + i)) − Suc (j
+ i)) n]
    by fast

  let ?sSuc = (s (Suc (j+i)))
  let ?si = (s (j+i))
  let ?c = (rat-of-int (calculate-c (s (j+i)) Mt (Suc (j+i)) ) )
  let ?ind = (dim-vec (s (j+i))) −(Suc (j+i))

  have ?si − ?c·v (RAT M)!?ind = ?si + (−?c)·v (RAT M)!?ind
    using minus-add-uminus-vec[of ?si n ?c·v (RAT M)!?ind]
          carriers
    by fastforce
  then have (lattice-coord (?si − ?c·v (RAT M)!?ind))$(n−k) =
    (lattice-coord(?si))$(n−k) + (−?c)∗ (lattice-coord((RAT M)!?ind))$(n−k)
    using linear[of ?si (RAT M)!?ind n−k −?c] dim1 dim2 k-in-range
    by metis
  then have lin-lattice-coord:(lattice-coord (?sSuc))$(n−k) =
    (lattice-coord(?si))$(n−k) − ?c∗ (lattice-coord((RAT M)!?ind))$(n−k)
    using sub
    by algebra
  have neq:Suc (j+i)≠k using Suc(3) Suc(2) by auto
  moreover have ((dim-vec (s (j+i))) −(Suc (j+i)))≠ (n−k)
```

using *s-dim*[*of j+i*] *neq Suc*(*3*)
by (*metis Suc*(*2*) ‹*j* + *Suc i* = *Suc* (*j* + *i*)› *diff-0-eq-0 diff-cancel2*
diff-commute diff-diff-cancel diff-diff-eq diff-is-0-eq dim1)
moreover have (*lattice-coord* ((*RAT M*)!((*dim-vec* (*s* (*j+i*))) −(*Suc* (*j+i*)))))$(*n*−*k*)=
(*unit-vec n* ((*dim-vec* (*s* (*j+i*))) −(*Suc* (*j+i*))))$(*n*−*k*)
using *unit*[*of dim-vec* (*s* (*j+i*)) −(*Suc* (*j+i*))] *index-in-range* **by** *presburger*
then have *zero*:(*lattice-coord* ((*RAT M*)!((*dim-vec* (*s* (*j+i*))) −(*Suc* (*j+i*)))))$(*n*−*k*) = *0*
unfolding *unit-vec-def*
using *neq calculation*(*3*) *k-in-range* **by** *fastforce*
then have (*lattice-coord* (*s* (*Suc* (*j+i*))))$(*n*−*k*) = ((*lattice-coord* (*s* (*j+i*)))$(*n*−*k*)) −
(*rat-of-int* (*calculate-c* (*s* (*j+i*)) *Mt* (*Suc* (*j+i*))))
∗*0*
using *zero lin-lattice-coord* **by** *presburger*
then have *conclusion1*:(*lattice-coord* (*s* (*Suc* (*j+i*))))$(*n*−*k*) = ((*lattice-coord* (*s* (*j+i*)))$(*n*−*k*))
by *simp*
have *init-sub*:(*s* (*Suc* (*j+i*)))• *Mt*!(*n*−*k*) = ((*s* (*j+i*)) −
((*rat-of-int* (*calculate-c* (*s* (*j+i*)) *Mt* (*Suc* (*j+i*)))) $\cdot_v$ (*RAT M*)!((*dim-vec* (*s* (*j+i*))) −(*Suc* (*j+i*)))))
• (*Mt*!(*n*−*k*))
using *sub*
by *simp*
moreover have *carrier-prod*:((*rat-of-int* (*calculate-c* (*s* (*j+i*)) *Mt* (*Suc* (*j+i*))))
$\cdot_v$ (*RAT M*)!((*dim-vec* (*s* (*j+i*))) −(*Suc* (*j+i*))))∈*carrier-vec n*
using *smult-carrier-vec*[*of* (*rat-of-int* (*calculate-c* (*s* (*j+i*)) *Mt* (*Suc* (*j+i*))))
(*RAT M*)!((*dim-vec* (*s* (*j+i*))) −(*Suc* (*j+i*))) *n*] *carrier-vecI dim2* **by** *blast*
moreover have *l*:((*s* (*j+i*)) −
((*rat-of-int* (*calculate-c* (*s* (*j+i*)) *Mt* (*Suc* (*j+i*)))) $\cdot_v$ (*RAT M*)!((*dim-vec* (*s* (*j+i*))) −(*Suc* (*j+i*)))))
• (*Mt*!(*n*−*k*)) = (*s* (*j+i*))• (*Mt*!(*n*−*k*)) − ((*rat-of-int* (*calculate-c* (*s* (*j+i*)) *Mt* (*Suc* (*j+i*))))
$\cdot_v$ (*RAT M*)!((*dim-vec* (*s* (*j+i*))) −(*Suc* (*j+i*))))• (*Mt*!(*n*−*k*))
using *s-dim*[*of j+i*]
assms(*2*)
access-index-M-dim
dim-vecs-in-Mt
carrier-vecI[*of Mt*!(*n*−*k*) *n*]
carrier-vecI[*of* (*RAT M*)!((*dim-vec* (*s* (*j+i*))) −(*Suc* (*j+i*))) *n*]
add-scalar-prod-distrib[*of*
(*s* (*j+i*))
n
(*rat-of-int* (*calculate-c* (*s* (*j+i*)) *Mt* (*Suc* (*j+i*)))) $\cdot_v$ (*RAT M*)!((*dim-vec* (*s* (*j+i*))) −(*Suc* (*j+i*)))
(*Mt*!(*n*−*k*))]

using *calculation(5) carriers k-in-range minus-scalar-prod-distrib* **by** *blast*

moreover then have *lin-scalar-prod*:((s (j+i)) −
((rat-of-int (calculate-c (s (j+i)) Mt (Suc (j+i)))) ·ᵥ (RAT M)!((dim-vec (s (j+i))) −(Suc (j+i)))))
· (Mt!(n−k)) = (s (j+i))· (Mt!(n−k)) − (rat-of-int (calculate-c (s (j+i)) Mt (Suc (j+i))))
∗ ((RAT M)!((dim-vec (s (j+i))) −(Suc (j+i))) · (Mt!(n−k)))
by (*metis dim2 dim-vecs-in-Mt k-in-range scalar-prod-smult-left*)
moreover have *step-past-index*:(dim-vec (s (j+i))) −(Suc (j+i))<n−k
using *s-dim*[*of j+i*] *Suc(3) Suc(2)*
by (*simp add*: *calculation(3) diff-le-mono2 dim1 le-SucI nat-less-le trans-le-add1*)
moreover have ((RAT M)!((dim-vec (s (j+i))) −(Suc (j+i))) · (Mt!(n−k))) = 0
using *step-past-index upper-tri*[*of* (dim-vec (s (j+i))) −(Suc (j+i)) n−k] *Suc(4)*
by *simp*
then have (s (Suc (j+i)))· Mt!(n−k) = (s (j+i))· Mt!(n−k) −
((rat-of-int (calculate-c (s (j+i)) Mt (Suc (j+i)))) ∗0)
using *lin-scalar-prod init-sub*
by *algebra*
then have *conclusion2*:(s (Suc (j+i)))· Mt!(n−k) = (s (j+i))· Mt!(n−k) **by** *auto*
show *?case*
by (*metis Suc(2) Suc(3) Suc(4) Suc.hyps Suc-leD* ‹j + Suc i = Suc (j + i)› *conclusion1 conclusion2*)
qed

lemma *small-orth-coord*:
fixes *i*::*nat*
assumes 1≤i
assumes i≤n
shows *abs* ((s i) · Mt!(n−i)) ≤ (sq-norm (Mt!(n−i)))∗(1/2)
proof(−)
have *minus-plus*:Suc (i−1) = i **using** *assms(1)* **by** *auto*
then have *init-sub*:s i = (s (i−1))−((rat-of-int (calculate-c (s (i−1)) Mt i)) ·ᵥ (RAT M)!((dim-vec (s (i−1))) −i))
using *sub-s*[*of i−1*]
by (*metis* (*full-types*) *Suc-le-eq assms(2) less-eq-nat.simps(1)*)
then have *scalar-distrib*:(s i) · Mt!(n−i) = (s (i−1)) · Mt!(n−i)−(((rat-of-int (calculate-c (s (i−1)) Mt i))
·ᵥ (RAT M)!((dim-vec (s (i−1))) −i))·Mt!(n−i))
using *add-scalar-prod-distrib*[*of* (s (i−1)) n ((rat-of-int (calculate-c (s (i−1)) Mt i))
·ᵥ (RAT M)!((dim-vec (s (i−1))) −i)) Mt!(n−i)]
s-dim[*of i−1*]
carrier-vecI[*of* Mt!(n−i)]
carrier-vecI[*of* (RAT M)!((dim-vec (s (i−1))) −i)]
access-index-M-dim[*of* ((dim-vec (s (i−1))) −i)]

```
        dim-vecs-in-Mt[of n−i]
        init-sub
       minus-scalar-prod-distrib[of (s (i−1)) n ( (rat-of-int (calculate-c (s (i−1))
Mt i ) )
                              ·v (RAT M)!( (dim-vec (s (i−1))) −i)) Mt!(n−i)]
  by (metis Suc-leD assms(2) diff-Suc-less gs.smult-closed le0 minus-plus non-trivial)
  also have scalar-commute:(s (i−1)) · Mt!(n−i)−(( (rat-of-int (calculate-c (s
(i−1)) Mt i ) )
                                        ·v (RAT M)!( (dim-vec (s (i−1)))
−i))·Mt!(n−i)) =
          (s (i−1)) · Mt!(n−i)−( (rat-of-int (calculate-c (s (i−1)) Mt i ) )
                           ∗ (((RAT M)!( (dim-vec (s (i−1))) −i)) ·Mt!(n−i) ))
   using scalar-prod-smult-left
        carrier-vecI[of Mt!(n−i)]
        carrier-vecI[of (RAT M)!( (dim-vec (s (i−1))) −i)]
        access-index-M-dim
        dim-vecs-in-Mt

  by (smt (verit) Suc-le-D assms(2) diff-less index-minus-vec(2) index-smult-vec(2)

        init-sub minus-plus s-dim zero-less-Suc)
 moreover have index-in-range: 0≤n−i ∧ n−i<n
   using assms(1) assms(2)
   by simp
 moreover have sq-norm-eq:((RAT M)!( (dim-vec (s (i−1))) −i)) ·Mt!(n−i) =
sq-norm (Mt!(n−i))
   using one-diag[of n−i]
        s-dim[of i−1]
        index-in-range
        assms(1)
        assms(2)
        less-imp-diff-less
   by simp
 then have (s i) · Mt!(n−i) = (s (i−1)) · Mt!(n−i)−
         ( (rat-of-int (calculate-c (s (i−1)) Mt i ) ) ∗ sq-norm (Mt!(n−i)))
   using scalar-distrib scalar-commute sq-norm-eq by argo
  then have final-sub:abs((s i) · Mt!(n−i)) = abs(( (rat-of-int (calculate-c (s
(i−1)) Mt i ) )
                                      ∗ sq-norm (Mt!(n−i))) − (s (i−1)) ·
Mt!(n−i))
   using abs-minus-commute by simp
 then have round-small:abs(rat-of-int (calculate-c (s (i−1)) Mt i )−
                (((s (i−1)) · (Mt!( (dim-vec (s (i−1))) − i ) ) )
                / (sq-norm-vec (Mt!( (dim-vec (s (i−1))) − i ) ) ) ))≤1/2
   by (metis calculate-c.simps of-int-round-abs-le)
 moreover have pos:0≤ sq-norm (Mt!(n−i))
   by (simp add: sq-norm-vec-ge-0)
 then have (sq-norm (Mt!(n−i)))∗abs((rat-of-int (calculate-c (s (i−1)) Mt i )−
          (((s (i−1)) · (Mt!( (dim-vec (s (i−1))) − i ) ) ) /
```

```
          (sq-norm-vec (Mt!( (dim-vec (s (i−1))) − i ) ) ) )))
          ≤(sq-norm (Mt!(n−i)))∗(1/2)
    using pos round-small mult-left-mono by blast
  then have 2:abs((sq-norm (Mt!(n−i)))∗(rat-of-int (calculate-c (s (i−1)) Mt i
)−
           (((s (i−1)) · (Mt!( (dim-vec (s (i−1))) − i ) ) ) /
             (sq-norm-vec (Mt!( (dim-vec (s (i−1))) − i ) ) ) )) )≤(sq-norm
(Mt!(n−i)))∗(1/2)
    using pos by (smt (verit) abs-mult abs-of-nonneg)
  have i≤n
    using assms(2) by simp
  then have abs(
          (sq-norm (Mt!(n−i)))∗(rat-of-int (calculate-c (s (i−1)) Mt i ))−
        (sq-norm (Mt!(n−i)))∗( ((s (i−1)) · (Mt!( (dim-vec (s (i−1))) − i ) ) )/
         (sq-norm (Mt!(n−i))) )
         )≤(sq-norm (Mt!(n−i)))∗(1/2)
    using 2
       s-dim[of i]
    by (smt (verit) Rings.ring-distribs(4) Suc-leD minus-plus s-dim)
  then have 1:abs(
          (sq-norm (Mt!(n−i)))∗(rat-of-int (calculate-c (s (i−1)) Mt i ))−
          ((s (i−1)) · (Mt!( (dim-vec (s (i−1))) − i ) ) )∗
         ( (sq-norm (Mt!(n−i)))/(sq-norm (Mt!(n−i))) )
         )≤(sq-norm (Mt!(n−i)))∗(1/2)
    using assms(2) s-dim
    by (smt (z3) gs.cring-simprules(14) times-divide-eq-right)
  moreover have nonzero:sq-norm (Mt!(n−i))≠0
    using Mt-gso-connect[of n−i] assms
   by (metis M-locale-2 gram-schmidt-fs-lin-indpt.sq-norm-pos index-in-range length-map
rel-simps(70))
  moreover have cancel:(sq-norm (Mt!(n−i)))/(sq-norm (Mt!(n−i)))=1
    using nonzero
    by auto
  moreover have dim-match:dim-vec (s (i−1)) = n
    using s-dim[of i−1] assms(2)
    by linarith
  then have final-ineq:abs(
          (sq-norm (Mt!(n−i)))∗(rat-of-int (calculate-c (s (i−1)) Mt i ))−
          ((s (i−1)) · (Mt!( (dim-vec (s (i−1))) − i ) ) )
         )≤(sq-norm (Mt!(n−i)))∗(1/2)
    using 1 cancel
    by (smt (verit) gs.r-one)
  then have rearrange-final-ineq: abs( (rat-of-int (calculate-c (s (i−1)) Mt i ))
         ∗ (sq-norm (Mt!(n−i))) − ((s (i−1)) · (Mt!( n − i ) ) ))≤(sq-norm
(Mt!(n−i)))∗(1/2)
    using dim-match
    by algebra
  show ?thesis
    using final-sub rearrange-final-ineq
```

```
    by argo
qed
lemma lattice-carrier: L⊆ carrier-vec n
proof−
  have x∈carrier-vec n if x-def:x∈L for x
  proof−
    obtain f where f-def:x = sumlist (map (λi. (f i)·v M!i ) [0..<n])
      using x-def unfolding L-def lattice-of-def by fast
    have (f i)·v M!i∈carrier-vec n if 0≤i∧i<n for i
      using access-index-M-dim[of i]
      by (metis carrier-vec-dim-vec map-carrier-vec nth-map smult-closed that)
    then have set (map (λi. (f i)·v M!i ) [0..<n]) ⊆ carrier-vec n by auto
    then have sumlist (map (λi. (f i)·v M!i ) [0..<n]) ∈ carrier-vec n by simp
    then show x∈carrier-vec n using f-def by fast
  qed
  then show ?thesis by fast
qed
```

4 Lattice Lemmas

```
lemma lattice-sum-close:
  fixes u::int vec and v::int vec
  assumes u∈L v∈L
  shows u+v∈L
proof −
  let ?mM = mat-of-cols n M
  have 1:?mM ∈carrier-mat n n using dim-vecs-in-M by fastforce
  have set-M: set M ⊆ carrier-vec n
    using dim-vecs-in-M carrier-vecI by blast
  have as-mat-mult:lattice-of M = {y∈carrier-vec n. ∃x∈carrier-vec n. ?mM *v x
= y}
     using lattice-of-as-mat-mult[OF set-M] by blast
 then obtain u1 where u1-def:u = ?mM *v u1 ∧ u1∈carrier-vec n using assms
unfolding L-def by auto
  obtain v1 where v1-def:v = ?mM *v v1 ∧ v1∈carrier-vec n
    using assms as-mat-mult unfolding L-def by auto
  have u1+v1∈carrier-vec n using u1-def v1-def by blast
  moreover have ?mM*v (u1+v1) = u+v
    using u1-def v1-def 1 mult-add-distrib-mat-vec[of ?mM n n u1 v1]
    by metis
  moreover have u+v∈carrier-vec n using assms lattice-carrier by blast
  ultimately show u+v∈L
    using as-mat-mult unfolding L-def
    by blast
qed


lemma lattice-smult-close:
  fixes u::int vec and q::int
```

```
  assumes u∈L
  shows q·v u∈L

proof−
  let ?mM = mat-of-cols n M
  have 1:?mM ∈carrier-mat n n using dim-vecs-in-M by fastforce
  have set-M: set M ⊆ carrier-vec n
    using dim-vecs-in-M carrier-vecI by blast
  have as-mat-mult:lattice-of M = {y∈carrier-vec n. ∃ x∈carrier-vec n. ?mM ∗v x
= y}
    using lattice-of-as-mat-mult[OF set-M] by blast
  then obtain v::int vec where v-def:u = ?mM ∗v v∧v∈carrier-vec n
    using assms unfolding L-def by auto
  then have q·v v ∈carrier-vec n by blast
  moreover then have q·v u= ?mM ∗v (q·v v) using 1 v-def by fastforce
  ultimately show q·v u ∈ L
   by (metis (mono-tags, lifting) L-def as-mat-mult assms mem-Collect-eq smult-closed)
qed

lemma smult-vec-zero:
  fixes v ::'a::ring vec
  shows 0 ·v v = 0v (dim-vec v)
  unfolding smult-vec-def vec-eq-iff
  by (auto)

lemma coset-s:
  fixes i::nat
  assumes i≤n
  shows s i ∈coset
  using assms
proof(induct i)
  case 0
  have s 0 = −t unfolding s-def by simp
  moreover have carrier-mt:−t∈carrier-vec n using length-M carrier-vecI[of t n]
by fastforce
  ultimately have pzero:s 0 = of-int-hom.vec-hom (0v n) −t by fastforce
  let ?zero = λ j. 0
  have 0<length M using non-trivial by fast
  then have M!0 ∈ set M by force
  then have M!0∈L using basis-in-latticeI[of M M!0] dim-vecs-in-M carrier-vecI
L-def
    by blast
  then have 0v n ∈L
   using lattice-smult-close[of M!0 0] smult-vec-zero[of M!0] access-index-M-dim[of
0] non-trivial
    unfolding L-def
    by fastforce
  then show ?case  using pzero by blast
next
```

```
  case (Suc i)
  let ?q = (rat-of-int (calculate-c (s i) Mt (Suc i) ) )
  let ?ind = ( (dim-vec (s i)) −(Suc i))
  have sub:s (Suc i) = (s i) −
( ?q ·v (RAT M)!?ind)
    using sub-s[of i] Suc.prems by linarith
  have s i ∈coset using Suc by auto
  then obtain x where x-def:x∈L ∧ (s i) = of-int-hom.vec-hom x−t by blast
  have ( ?q ·v (RAT M)!?ind)∈ of-int-hom.vec-hom‘ L
  proof−
    have dim-vec (s i) = n using s-dim[of i] Suc.prems by fastforce
    then have in-range:?ind<n∧ 0≤?ind using Suc.prems by simp
    then have com-hom:(RAT M)!(?ind) = of-int-hom.vec-hom (M!?ind) by auto
    have M!?ind∈set M using in-range by simp
    then have mil:M!?ind ∈ L using basis-in-latticeI[of M M!?ind] dim-vecs-in-M
carrier-vecI L-def
      by blast
    moreover have ?q·v(of-int-hom.vec-hom (M!?ind)) =
                 of-int-hom.vec-hom ((calculate-c (s i) Mt (Suc i) ) ·v M!?ind)
      by fastforce
    moreover have (calculate-c (s i) Mt (Suc i) ) ·v M!?ind∈L
       using lattice-smult-close[of M!?ind (calculate-c (s i) Mt (Suc i) )] mil by
simp
    ultimately show ( ?q ·v (RAT M)!?ind) ∈ of-int-hom.vec-hom‘ L
      using com-hom
      by force
  qed
  then obtain y where y-def:( ?q ·v (RAT M)!?ind) = of-int-hom.vec-hom y∧
y∈L by blast
  have carrier-x: x∈carrier-vec n using lattice-carrier x-def by blast
  have carrier-y: y∈carrier-vec n using lattice-carrier y-def by blast
  then have carrier-my: −y∈carrier-vec n by simp
  then have 1:−( ?q ·v (RAT M)!?ind) = of-int-hom.vec-hom (−y) using y-def
by fastforce
  then have s (Suc i) = of-int-hom.vec-hom x−t+ of-int-hom.vec-hom (−y)
    using sub x-def y-def 1 by fastforce
  then have s (Suc i) = of-int-hom.vec-hom x + of-int-hom.vec-hom (−y) − t
    using lattice-carrier x-def y-def length-M
    by fastforce
 moreover have of-int-hom.vec-hom x + of-int-hom.vec-hom (−y) = of-int-hom.vec-hom
(x+ −y)
    using carrier-my carrier-x by fastforce
  ultimately have 2:s (Suc i) = of-int-hom.vec-hom (x+ −y) −t
    by metis
  have −y = −1 ·v y by auto
  then have −y∈L using lattice-smult-close y-def by simp
  then have x+−y∈L using lattice-sum-close x-def by simp
  then show ?case using 2 by fast
qed
```

lemma *subtract-coset-into-lattice*:
fixes *v*::*rat vec*
fixes *w*::*rat vec*
assumes *v*∈*coset*
assumes *w*∈*coset*
shows (*v*−*w*)∈*of-int-hom.vec-hom*‘ *L*
proof−
obtain *l1* **where** *l1-def*:*v*=*l1*−*t*∧ *l1*∈*of-int-hom.vec-hom*‘ *L* **using** *assms*(*1*) **by** *blast*
obtain *l2* **where** *l2-def*:*w* = *l2*−*t*∧ *l2*∈*of-int-hom.vec-hom*‘ *L* **using** *assms*(*2*) **by** *blast*
have *carrier-l1*:*l1* ∈ *carrier-vec n* **using** *lattice-carrier l1-def* **by** *force*
have *carrier-l2*:*l2* ∈ *carrier-vec n* **using** *lattice-carrier l2-def* **by** *force*
obtain *l1p* **where** *l1p-def*:*l1* = *of-int-hom.vec-hom l1p* ∧ *l1p*∈*L* **using** *l1-def* **by** *fast*
obtain *l2p* **where** *l2p-def*:*l2* = *of-int-hom.vec-hom l2p* ∧ *l2p*∈*L* **using** *l2-def* **by** *fast*
have −*l2p* = −*1* $\cdot_v$ *l2p* **using** *carrier-l2* **by** *fastforce*
then have *ml2p*:−*l2p*∈ *L* **using** *lattice-smult-close*[*of l2p* −*1*] *l2p-def* **by** *presburger*
then have *of-int-hom.vec-hom* (−*l2p*)∈ *of-int-hom.vec-hom*‘ *L* **by** *simp*
moreover have *of-int-hom.vec-hom* (−*l2p*) = −*l2* **using** *l2p-def* **by** *fastforce*
then have *l1*−*l2* = *of-int-hom.vec-hom* (*l1p* − *l2p*) **using** *l1p-def l2p-def carrier-l1 carrier-l2* **by** *auto*
moreover have *l1p*−*l2p*∈*L* **using** *lattice-sum-close*[*of l1p* −*l2p*]
l1p-def l2p-def ml2p carrier-l1 carrier-l2
by (*simp add*: *minus-add-uminus-vec*)
ultimately have *l1*−*l2*∈ *of-int-hom.vec-hom*‘ *L* **by** *fast*
moreover have *v*−*w* = *l1*−*l2* **using** *l1-def l2-def length-M carrier-vecI carrier-l1 carrier-l2* **by** *force*
ultimately show *?thesis* **by** *simp*
qed
lemma *t-in-coset*:
shows *uminus t* ∈ *coset*
using *coset-s*[*of 0*] *Babai-Help.simps* **unfolding** *s-def* **by** *simp*

5 Lemmas on closest distance

lemma *closest-distance-sq-pos*: *closest-distance-sq*≥*0*
proof−
have ∀ *N*∈ {*real-of-rat* (*sq-norm x*::*rat*) |*x*. *x* ∈ *coset*}. *0*≤*N*
using *sq-norm-vec-ge-0* **by** *auto*
moreover have {*real-of-rat* (*sq-norm x*::*rat*) |*x*. *x* ∈ *coset*}≠{} **using** *t-in-coset* **by** *blast*
ultimately have *0*≤*Inf* {*real-of-rat* (*sq-norm x*::*rat*) |*x*. *x* ∈ *coset*}
by (*meson cInf-greatest*)
then show *?thesis* **unfolding** *closest-distance-sq-def* **by** *blast*
qed

definition *witness*:: *rat vec*⇒*rat* ⇒ *bool*
  where *witness v eps-closest* = (*sq-norm v* ≤ *eps-closest* ∧ *v*∈*coset*∧*dim-vec v* = *n*)

definition *epsilon*::*real* **where** *epsilon* = *11/10*

definition *close-condition*::*rat* ⇒ *bool*
  where *close-condition eps-closest* ≡
(*if closest-distance-sq* = *0 then 0*≤ *real-of-rat eps-closest*
else real-of-rat (*eps-closest*)>*closest-distance-sq*)
∧ (*real-of-rat* (*eps-closest*)≤*epsilon*∗*closest-distance-sq*)

lemma *close-rat*:
  obtains *eps-closest*::*rat*
  where *close-condition eps-closest*
proof(*cases closest-distance-sq* = *0*)
  case *t*:*True*
  then have *epsilon*∗*closest-distance-sq* = *real-of-rat* (*0*::*rat*) **by** *simp*
  then have *real-of-rat* (*0*::*rat*)≤ *epsilon*∗*closest-distance-sq*∧*closest-distance-sq*
          ≤(*real-of-rat* (*0*::*rat*))
    using *t* **by** *force*
  then show *?thesis*
    using *that t* **unfolding** *close-condition-def* **by** *metis*
  next
    case *f*:*False*
    then have *0*<*closest-distance-sq*
      using *closest-distance-sq-pos* **by** *linarith*
    moreover have (*1*::*real*)<*epsilon* **unfolding** *epsilon-def* **by** *simp*
    ultimately have *closest-distance-sq*<*epsilon*∗*closest-distance-sq* **by** *simp*
    then show *?thesis*
     using *Rats-dense-in-real*[*of closest-distance-sq epsilon*∗*closest-distance-sq*] *that*
      unfolding *close-condition-def*
      by (*metis Rats-cases less-eq-real-def*)
qed

definition *eps-closest*::*rat*
  where *eps-closest* = (*if* ∃ *r*. *close-condition r then SOME r*. *close-condition r else 0*)

lemma *eps-closest-lemma*: *close-condition eps-closest*
  using *close-rat* **unfolding** *eps-closest-def* **by** (*metis* (*full-types*))

lemma *rational-tri-ineq*:
  fixes *v*::*rat vec*
  fixes *w*::*rat vec*
  assumes *dim-vec v* = *dim-vec w*
  shows (*sq-norm* (*v*+*w*))≤ *4*∗(*Max* {(*sq-norm v*), (*sq-norm w*)})
proof−

```
  let ?d = dim-vec w
  let ?M = Max {(sq-norm v), (sq-norm w)}
  have carr-v:v∈carrier-vec ?d using assms carrier-vecI[of v ?d] by fastforce
  have carr-w:w∈carrier-vec ?d using carrier-vecI[of w ?d] by fastforce
  have carr-vw:v+w∈carrier-vec ?d using carr-v carr-w add-carrier-vec by blast
  have sq-norm (v+w) = (v+w)·(v+w)
    by (simp add: sq-norm-vec-as-cscalar-prod)
  also have (v+w)·(v+w) = v·(v+w)+w·(v+w)
    using add-scalar-prod-distrib[of v ?d w v+w]
          carr-v carr-w carr-vw by blast
  also have v·(v+w)+w·(v+w) = v·v+v·w+w·v+w·w
    using scalar-prod-add-distrib[of v ?d v w]
          scalar-prod-add-distrib[of w ?d v w]
          carr-v carr-w carr-vw by algebra
  also have v·w=w·v
    using carr-v carr-w comm-scalar-prod by blast
  also have v·v = sq-norm v
    using sq-norm-vec-as-cscalar-prod[of v] by force
  also have w·w = sq-norm w
    using sq-norm-vec-as-cscalar-prod[of w] by force
  finally have sq-norm (v+w) = sq-norm v + sq-norm w + 2*(w·v) by force
  also have b1:sq-norm v≤?M by force
  also have b2:sq-norm w≤?M by force
  also have 2*(w·v)≤2*(Max {(sq-norm v), (sq-norm w)})
  proof−
    have (w·v)^2≤ (sq-norm v) * (sq-norm w)
      using scalar-prod-Cauchy[of w ?d v] carr-w carr-v by algebra
    also have (sq-norm v) * (sq-norm w)≤?M*?M
      using b1 b2 sq-norm-vec-ge-0[of w] sq-norm-vec-ge-0[of v]
            mult-mono[of sq-norm v ?M sq-norm w ?M] by linarith
    also have ?M*?M = ?M^2
      using power2-eq-square[of ?M] by presburger
    finally have (w·v)^2≤?M^2 by blast
    also have (w·v)^2=abs(w·v)^2 by force
    finally have abs(w·v)^2≤?M^2 by presburger
    moreover have 0≤abs(w·v) by fastforce
    moreover have 0≤?M
      using sq-norm-vec-ge-0[of w] sq-norm-vec-ge-0[of v] by fastforce
    ultimately have abs(w·v)≤?M
      using power2-le-imp-le by blast
    also have (w·v)≤abs(w·v) by force
    finally show ?thesis by linarith
  qed
  finally show ?thesis by auto
qed

lemma witness-exists:
  shows ∃ v. witness v eps-closest
proof(cases closest-distance-sq = 0)
```

```
  case t:True
  have eps-closest = 0
    using eps-closest-lemma t
    unfolding witness-def unfolding close-condition-def
    by auto
  then have equiv:?thesis = (∃ v. v∈coset∧ (dim-vec v = n) ∧ (sq-norm v) ≤ 0)
    unfolding witness-def eps-closest-def by auto
  show ?thesis
  proof(rule ccontr)
    assume contra:¬ ?thesis
    have {real-of-rat (sq-norm x::rat) |x. x ∈ coset}≠{} using t-in-coset by fast
   then have limit-point:∃ v::rat vec. real-of-rat (sq-norm v) < (eps::real) ∧ v∈coset
if 0<eps for eps
      using t cInf-lessD[of {real-of-rat (sq-norm x::rat) |x. x ∈ coset} eps] that
      unfolding closest-distance-sq-def by auto
    moreover have 0<real-of-rat ((sq-norm ((RAT M)!0) )/ (4*α^(n−1)))
    proof−
      have 0<1/(4*α^(n−1)) using non-trivial unfolding α-def by force
      moreover have 0< (sq-norm ((RAT M)!0))
        using gram-schmidt-fs-lin-indpt.sq-norm-pos[of n RAT M 0]
              gram-schmidt-fs-lin-indpt.sq-norm-gso-le-f[of n RAT M 0]
              M-locale-2 non-trivial
        by fastforce
      ultimately show ?thesis by auto
    qed
    ultimately obtain v::rat vec where v-def:real-of-rat (sq-norm v)
                              < real-of-rat ((sq-norm ((RAT M)!0) )/ (4*α^(n−1)))∧
v∈coset
      by presburger
    then have dim-vec v = n
      using length-M by force
    then have 0< real-of-rat (sq-norm v)
      using equiv contra v-def by auto
     then obtain w::rat vec where w-def:real-of-rat (sq-norm w) < real-of-rat
(sq-norm v)∧w∈coset
      using limit-point by fast
   then have small-w:real-of-rat (sq-norm w)<real-of-rat ((sq-norm ((RAT M)!0)
)/ (4*α^(n−1)))
      using v-def by argo
    have lat:w−v∈ of-int-hom.vec-hom‘ L using subtract-coset-into-lattice[of w v]
      using v-def w-def by force
    then obtain l where l-def:l∈L∧w−v=of-int-hom.vec-hom l by blast
    then have of-int-hom.vec-hom l ∈ gs.lattice-of (RAT M)
      using lattice-of-of-int[of M n l] dim-vecs-in-M carrier-vecI L-def by blast
    then have lat-hom:w−v ∈ gs.lattice-of (RAT M) using l-def by simp
    have sq-norm v ≠ sq-norm w using w-def by auto
    then have neq:w≠v by meson
     have c1:w∈carrier-vec n using length-M w-def lattice-carrier carrier-dim-vec
by fastforce
```

```
    moreover have c2:v∈carrier-vec n using length-M v-def lattice-carrier car-
rier-dim-vec by fastforce
    ultimately have c3:w−v∈carrier-vec n by simp
    have neqzero:w−v≠0v n
    proof(rule ccontr)
      assume c:¬?thesis
      have w−v=0v n using c by blast
      then have w=v+ 0v n using c1 c2 c3
           by (smt (verit, ccfv-SIG) gs.M.add.r-inv-ex minus-add-minus-vec mi-
nus-cancel-vec minus-zero-vec right-zero-vec)
      then show False using c2 neq by simp
    qed
    then have w−v ∈ gs.lattice-of (RAT M) − {0v n} using lat-hom by blast
    moreover have α^(n−1) ∗ (sq-norm (w−v)) < (sq-norm ((RAT M)!0) )
    proof−
      have w−v = w+ (−v) by fastforce
      then have sq-norm (w−v) = sq-norm (w+(−v)) by simp
      also have sq-norm (w+(−v)) ≤ 4∗Max({sq-norm w, sq-norm (−v)})
        using rational-tri-ineq[of w −v] c1 c2 by fastforce
      also have sq-norm (−v) = sq-norm v
      proof−
        have −v = (−1)·v v by fastforce
      then have sq-norm (−v) = ((−1)·v v)•((−1)·v v) using sq-norm-vec-as-cscalar-prod[of
−v] by force
        then have sq-norm (−v) = (−1)∗(−1)∗(v•v) using c1 c2 by simp
        then show ?thesis using sq-norm-vec-as-cscalar-prod[of v] by simp
      qed
        also have Max({sq-norm w, sq-norm (v)})<((sq-norm ((RAT M)!0) )/
(4∗α^(n−1)))
        using v-def small-w of-rat-less by auto
       finally have sq-norm (w−v)<4∗((sq-norm ((RAT M)!0) )/ (4∗α^(n−1)))
by linarith
     then have sq-norm (w−v)<(sq-norm ((RAT M)!0) )/ (α^(n−1)) by linarith
      moreover have p:0<α^(n−1) unfolding α-def by fastforce
      ultimately show ?thesis using p
        by (metis gs.cring-simprules(14) pos-less-divide-eq)
    qed
    ultimately show False
       using gram-schmidt-fs-lin-indpt.weakly-reduced-imp-short-vector[of n (RAT
M) α w−v]
           M-locale-2 reduced
      unfolding α-def gs.reduced-def L-def by force
  qed
next
  case False
  then have closest-distance-sq < real-of-rat eps-closest
    using eps-closest-lemma unfolding eps-closest-def close-condition-def
    by presburger
  moreover have {real-of-rat (sq-norm x::rat) |x. x ∈ coset}≠{} using t-in-coset
```

by *fast*
ultimately obtain *l* **where** *l*∈{*real-of-rat* (*sq-norm x*::*rat*) |*x*. *x* ∈ *coset*}∧ *l*< *real-of-rat eps-closest*
using *closest-distance-sq-pos*
unfolding *closest-distance-sq-def*
by (*meson cInf-lessD*)
moreover then obtain *v*::*rat vec* **where** *l* = *real-of-rat* (*sq-norm v*) ∧ *v*∈*coset* **by** *blast*
ultimately show *?thesis* **unfolding** *witness-def lattice-carrier*
by (*smt* (*verit*) *length-M index-minus-vec*(*2*) *mem-Collect-eq of-rat-less-eq*)
qed

6 More linear algebra lemmas

lemma *carrier-Ms*:
shows *mat-M* ∈*carrier-mat n n mat-M-inv* ∈*carrier-mat n n*
using *M-dim M-inv-dim*
apply *blast*
by (*simp add*: *M-inv-dim*(*1*) *M-inv-dim*(*2*) *carrier-matI*)

lemma *carrier-L*:
fixes *v*::*rat vec*
assumes *dim-vec v* = *n*
shows *lattice-coord v*∈*carrier-vec n*
unfolding *lattice-coord-def*
using *mult-mat-vec-carrier*[*of mat-M-inv n n v*]
carrier-Ms
carrier-vecI[*of v*]
assms(*1*)
by *fast*

lemma *sumlist-index-commute*:
fixes *Lst*::*rat vec list*
fixes *i*::*nat*
assumes *set Lst*⊆*carrier-vec n*
assumes *i*<*n*
shows (*gs.sumlist Lst*)$*i* = *sum-list* (*map* (λ*j*. (*Lst*!*j*)$*i*) [*0*..<(*length Lst*)])
using *assms*
proof(*induct Lst*)
case *Nil*
have *gs.sumlist Nil* = 0_v *n* **using** *assms* **unfolding** *gs.sumlist-def* **by** *auto*
then have *lhs*:(*gs.sumlist Nil*)$*i* = *0* **using** *assms*(*2*) **by** *auto*
have [*0*..<(*length Nil*)] = *Nil* **by** *simp*
then have (*map* (λ*j*. (*Nil*!*j*)$*i*) [*0*..<(*length Nil*)]) = *Nil* **by** *blast*
then have *sum-list* (*map* (λ*j*. (*Nil*!*j*)$*i*) [*0*..<(*length Nil*)]) = *0* **by** *simp*
then show *?case* **using** *lhs* **by** *simp*
next
case (*Cons a Lst*)
let *?CaLst* = *Cons a Lst*

have *set Lst* ⊆ *carrier-vec n* **using** *Cons.prems* **by** *auto*
then have *carr*:*gs.sumlist Lst* ∈*carrier-vec n* **using** *assms gs.sumlist-carrier*[*of Lst*]
by *blast*
have *gs.sumlist* (*Cons a Lst*) = *a* + *gs.sumlist Lst* **by** *simp*
then have *lhs*:(*gs.sumlist ?CaLst*)$*i* = *a*$*i* + (*gs.sumlist Lst*)$*i* **using** *assms carr* **by** *simp*
have *sum-list* (*map* (λ*j.* (*?CaLst*!*j*)$*i*) [*0* ..<(*length ?CaLst*)]) = *sum-list* (*map* (λ*l. l*$*i*) *?CaLst*)
by (*smt* (*verit*) *length-map map-eq-conv′ map-nth nth-map*)
moreover have *sum-list* (*map* (λ*l. l*$*i*) *?CaLst*) = *a*$*i* + *sum-list* (*map* (λ*l. l*$*i*) *Lst*) **by** *simp*
moreover have *sum-list* (*map* (λ*l. l*$*i*) *Lst*) = *sum-list* (*map* (λ*j.* (*Lst*!*j*)$*i*) [*0* ..<(*length Lst*)])
by (*smt* (*verit*) *length-map map-eq-conv′ map-nth nth-map*)
moreover have *sum-list* (*map* (λ*j.* (*Lst*!*j*)$*i*) [*0* ..<(*length Lst*)]) = (*gs.sumlist Lst*)$*i*
using *Cons.prems Cons.hyps* **by** *simp*
ultimately show *?case* **using** *lhs*
by *argo*
qed

lemma *mat-mul-to-sum-list*:
fixes *A*::*rat mat*
fixes *v*::*rat vec*
assumes *dim-vec v* = *dim-col A*
assumes *dim-row A* = *n*
shows *A*∗_v *v* = *gs.sumlist* (*map* (λ*j. v*$*j* ·_v (*col A j*)) [*0* ..< *dim-col A*])
proof−
have *carrier*:*set* (*map* (λ*j. v* $ *j* ·_v *col A j*) [*0* ..<*dim-col A*]) ⊆ *Rn*
by (*smt* (*verit*) *assms*(*2*) *carrier-dim-vec dim-col ex-map-conv index-smult-vec*(*2*) *subset-code*(*1*))
have (*A*∗_v *v*)$*i* = *gs.sumlist* (*map* (λ*j. v*$*j* ·_v (*col A j*)) [*0* ..< *dim-col A*])$*i* **if** *small*:*i*<*dim-row A* **for** *i*
proof−
let *?rAi* = *row A i*

have *1*:(*A*∗_v *v*)$*i* = *?rAi* · *v* **using** *small* **by** *simp*
have *2*:*?rAi* · *v* = *sum-list* (*map* (λ*j.* (*?rAi*$*j*)∗(*v*$*j*)) [*0* ..<*dim-col A*])
using *assms sum-set-upt-conv-sum-list-nat* **unfolding** *scalar-prod-def* **by** *auto*
have *?rAi*$*j*∗(*v*$*j*) = (*v*$*j* ·_v (*col A j*))$*i* **if** *jsmall*:*j*<*dim-col A* **for** *j*
unfolding *row-def col-def* **using** *small jsmall*
by *force*
then have (*map* (λ*j.* (*?rAi*$*j*)∗(*v*$*j*)) [*0* ..<*dim-col A*]) = (*map* (λ*j.* (*v*$*j* ·_v (*col A j*))$*i*) [*0* ..<*dim-col A*])
by *fastforce*
then have (*A*∗_v *v*)$*i* = *sum-list* (*map* (λ*j.* (*v*$*j* ·_v (*col A j*))$*i*) [*0* ..<*dim-col A*])

using *1 2* **by** *algebra*
then show *?thesis* **using** *sumlist-index-commute*[*of map* (λ*j*. *v*\$*j* $\cdot_v$ (*col A j*)) [*0* ..< *dim-col A*] *i*]
small assms(*2*) *carrier*
by (*smt* (*verit*) *gs.sumlist-vec-index length-map map-equality-iff nth-map subset-code*(*1*))
qed
moreover have *dim-vec* (*A*$*_v$*v*) = *dim-row A* **by** *fastforce*
moreover have *dim-vec* (*gs.sumlist* (*map* (λ*j*. *v*\$*j* $\cdot_v$ (*col A j*)) [*0* ..< *dim-col A*])) = *n*
using *carrier* **by** *auto*
ultimately show *?thesis* **using** *assms*
by *auto*
qed

lemma *recover-from-lattice-coord*:
fixes *v*::*rat vec*
assumes *dim-vec v* = *n*
shows *v* = *gs.sumlist* (*map* (λ*i*. (*lattice-coord v*)\$*i* $\cdot_v$ (*RAT M*)!*i*) [*0* ..< *n*])
proof −
have (*mat-M* * *mat-M-inv*)$*_v$ *v*= *mat-M*$*_v$(*lattice-coord v*)
unfolding *lattice-coord-def*
using *assms*(*1*) *carrier-Ms carrier-vecI*[*of v*]
assoc-mult-mat-vec[*of mat-M n n mat-M-inv n v*]
by *presburger*
then have (1_m *n*)$*_v$*v*=*mat-M*$*_v$(*lattice-coord v*)
using *inv1*
by *simp*
then have *v* = *mat-M*$*_v$(*lattice-coord v*)
by (*metis assms carrier-vec-dim-vec one-mult-mat-vec*)
then have *pre*:*v* = *gs.sumlist* (*map* (λ*i*. (*lattice-coord v*)\$*i* $\cdot_v$ *col mat-M i*) [*0* ..< *dim-col mat-M*])
using *mat-mul-to-sum-list*[*of lattice-coord v mat-M*]
M-dim
assms
dim-preserve-lattice-coord
by *simp*
moreover have *col mat-M i* = (*RAT M*)!*i* **if** *i*<*n* **for** *i*
using *vec-to-col*
by (*simp add*: *that*)
ultimately have (*map* (λ*i*. (*lattice-coord v*)\$*i* $\cdot_v$ *col mat-M i*) [*0* ..< *dim-col mat-M*]) =
(*map* (λ*i*. (*lattice-coord v*)\$*i* $\cdot_v$ (*RAT M*)!*i*) [*0* ..< *n*]) **using** *M-dim*
by *simp*
then show *v* = *gs.sumlist* (*map* (λ*i*. (*lattice-coord v*)\$*i* $\cdot_v$ (*RAT M*)!*i*) [*0* ..< *n*])
using *pre* **by** *presburger*
qed

```
lemma sumlist-linear-coord:
  fixes Lst::int vec list
  assumes ⋀i. i<length Lst ⟹dim-vec (Lst!i) = n
   shows lattice-coord (map-vec rat-of-int (sumlist Lst)) = gs.sumlist (map lat-
tice-coord (RAT Lst))
using assms
proof(induct Lst)
  case Nil
  have rhs:gs.sumlist(map lattice-coord (RAT Nil)) = 0_v n by fastforce
  have map-vec rat-of-int (sumlist Nil) = 0_v n by auto
  then have lattice-coord (map-vec rat-of-int (sumlist Nil)) = 0_v n
    unfolding lattice-coord-def using M-inv-dim
   by (metis carrier-Ms(2) gs.M.add.r-cancel-one' gs.M.zero-closed mult-add-distrib-mat-vec
mult-mat-vec-carrier)
  then show ?case using rhs by simp
next
  case (Cons a Lst)
  let ?CaLst = Cons a Lst
  let ?ra = of-int-hom.vec-hom a
  have dim:i∈set ?CaLst ⟹ dim-vec i = n for i using Cons.prems
    by (metis in-set-conv-nth)
  then have i-lt: (i < length Lst ⟹ dim-vec (Lst ! i) = n) for i
    using Cons.prems carrier-dim-vec by auto
  have carrier:set ?CaLst⊆ carrier-vec n using Cons.prems
    using carrier-vecI dim by fast
  then have carrier-sumCaLst: (sumlist ?CaLst)∈carrier-vec n by force
  have carrier-a: a ∈ carrier-vec n using carrier by force
  have carrier-Lst:set Lst ⊆ carrier-vec n using carrier by simp
  have lhs:lattice-coord (map-vec rat-of-int (sumlist ?CaLst)) = (lattice-coord ?ra)
+ gs.sumlist (map lattice-coord (RAT Lst))
  proof−
    have carrier-sumLst: sumlist Lst∈carrier-vec n using carrier-Lst by force
    have sumlist ?CaLst = a + sumlist Lst by force
     then have (map-vec rat-of-int (sumlist ?CaLst)) = ?ra + (map-vec rat-of-int
(sumlist Lst))
      using carrier-a carrier-sumLst carrier-sumCaLst by auto
    then have lattice-coord (map-vec rat-of-int (sumlist ?CaLst))
            = lattice-coord(?ra) + lattice-coord(map-vec rat-of-int (sumlist Lst))
      unfolding lattice-coord-def
      using carrier-sumCaLst carrier-a carrier-sumLst
      by (metis carrier-Ms(2) map-carrier-vec mult-add-distrib-mat-vec)
    then show ?thesis using i-lt Cons.hyps
      by algebra
  qed
  moreover have rhs:gs.sumlist (map lattice-coord (RAT ?CaLst)) =
                  (lattice-coord ?ra) + gs.sumlist (map lattice-coord (RAT Lst))
    by fastforce
  ultimately show ?case by argo
```

```
qed

lemma integral-sum:
  fixes l::nat
  assumes ⋀j1. j1 < l ⟹
    map f [0..<l] ! j1 ∈ ℤ
  shows sum-list
     (map f [0..<l]) ∈ ℤ
  using assms
proof(induct l)
  case 0
  have (map f [0..<0]) = Nil by auto
  then have sum-list (map f [0..<0]) = 0 by simp
  then show ?case by simp
next
  case (Suc l)
  have nontriv:Suc l>0 by simp
  have break:sum-list (map f [0..<(Suc l)]) = sum-list (map f [0..<l]) + (f l) by
fastforce
  have l<Suc l by simp
  then have [0..<(Suc l)]!l = l
    by (metis nth-upt plus-nat.add-0)
  moreover then have f ([0..<(Suc l)] ! l) = (map f [0..<(Suc l)]) ! l
   by (metis One-nat-def Suc-diff-Suc diff-Suc-1 local.nontriv nat-SN.default-gt-zero

        nth-map-upt nth-upt plus-1-eq-Suc real-add-less-cancel-right-pos)
  ultimately have z:f l ∈ℤ using Suc.prems by fastforce
  have ⋀j1. j1 < l ⟹
    map f [0..<l] ! j1 ∈ ℤ
    by (metis Suc.prems diff-Suc-1' diff-Suc-Suc less-SucI nth-map-upt)
  then have sum-list (map f [0..<l])∈ℤ using Suc by blast
  then show ?case using z break by force
qed


lemma int-coord:
  fixes i::nat
  assumes 0≤i
  assumes i<n
  fixes v::int vec
  assumes v∈L
  assumes dim-vec v = n
  shows (lattice-coord (map-vec rat-of-int v))$i∈ℤ
proof −
 obtain w where w-def:v = sumlist (map (λ i. of-int (w i) ·v M ! i) [0 ..< length
M])
    using L-def assms(3) vec-module.lattice-of-def
    by blast
```

```
  let ?Lst = (map (λ i. of-int (w i) ·v M ! i) [0 ..< length M])
  have dims-j:dim-vec (?Lst!j) = n if j-lt:j<length ?Lst for j
    using access-index-M-dim carrier-vecI j-lt by force
  let ?recover = (map lattice-coord (RAT ?Lst))
  have 1:lattice-coord (map-vec rat-of-int v) = gs.sumlist ?recover
    using sumlist-linear-coord[of ?Lst]
          w-def
          dims-j
    by blast
  have int-recover:⋀j. j<n⟹(?recover!j)$i ∈ℤ∧ (dim-vec (?recover!j)) = n
  proof −
    fix j::nat
    assume small:j<n
    have ?recover!j = lattice-coord ((RAT ?Lst)!j)
      using List.nth-map[of j (RAT ?Lst) lattice-coord]
            small
      by simp
    then have ?recover!j = lattice-coord (of-int-hom.vec-hom (?Lst!j))
      using List.nth-map[of j ?Lst of-int-hom.vec-hom]
            small
      by simp
    then have ?recover!j = lattice-coord (of-int-hom.vec-hom (of-int (w j) ·v M !
j))
      using List.nth-map[of j [0 ..< length M] (λ i. of-int (w i) ·v M ! i)]
            small
      by simp
     then have commuted-maps:?recover!j = mat-M-inv ∗v (of-int-hom.vec-hom
(of-int (w j) ·v M ! j))
      unfolding lattice-coord-def
      by simp
   then have ?recover!j = mat-M-inv ∗v((of-int (of-int (w j))) ·v of-int-hom.vec-hom
(M ! j))
      using of-int-hom.vec-hom-smult[of of-int (w j) M ! j]
      by metis
   then have ?recover!j = (of-int (of-int (w j))) ·v (mat-M-inv ∗v of-int-hom.vec-hom
(M ! j))
        using mult-mat-vec[of mat-M-inv n n of-int-hom.vec-hom (M ! j) (of-int
(of-int (w j)))]
            carrier-Ms
            access-index-M-dim[of j]
            carrier-vecI[of of-int-hom.vec-hom (M ! j) n]
      by (simp add: small)
   then have ?recover!j = (of-int (of-int (w j))) ·v (lattice-coord (of-int-hom.vec-hom
(M ! j)))
      unfolding lattice-coord-def
      by simp
    then have recover-unit:?recover!j = (of-int (of-int (w j))) ·v (unit-vec n j)
      using unit[of j]
            small
```

```
      by simp
    then have (?recover!j)$i=((of-int (of-int (w j))) ·v (unit-vec n j))$i
      by simp
    then have (?recover!j)$i = (of-int (of-int (w j))) * (unit-vec n j)$i
      by (simp add: assms(2))
    then have (?recover!j)$i = (of-int (of-int (w j))) * (if i=j then 1 else 0)
      using small assms(2)
      by simp
    moreover have (if i=j then 1 else 0) ∈ℤ
      by simp
    moreover have (of-int (of-int (w j)))∈ℤ
      by simp
    moreover have dim-vec (?recover!j) = n
    using recover-unit
          smult-closed[of (unit-vec n j) (of-int (of-int (w j)))]
          unit-vec-carrier[of n j]
    by force
    ultimately show (?recover!j)$i ∈ℤ ∧ dim-vec (?recover!j) = n
      by simp
  qed
  then have ∀ v∈set ?recover. dim-vec v = n
    by auto
  then have set ?recover⊆carrier-vec n
    using carrier-vecI
    by blast
  then have (gs.sumlist ?recover)$i = sum-list (map (λj. (?recover!j)$i) [0..<(length
?recover)])
    using sumlist-index-commute[of ?recover i] assms
    by blast
  moreover have length ?recover = n
    by auto
  ultimately have (gs.sumlist ?recover)$i = sum-list (map (λj. (?recover!j)$i)
[0..<n])
    by simp
  moreover have ⋀j. j<n ⟹ (map (λj. (?recover!j)$i) [0..<n])!j ∈ℤ
  proof−
    fix j::nat
    assume jsmall:j<n
    have (map (λj. (?recover!j)$i) [0..<n])!j = (λj. (?recover!j)$i) j
      using List.nth-map[of j [0..<n] (λj. (?recover!j)$i)]
            jsmall
      by simp
    then have (map (λj. (?recover!j)$i) [0..<n])!j =(?recover!j)$i
      by simp
    then show (map (λj. (?recover!j)$i) [0..<n])!j∈ℤ
      using int-recover[of j] jsmall
      by simp
  qed
  ultimately have (gs.sumlist ?recover)$i∈ℤ
```

using *integral-sum*[*of n* (λ*j. map lattice-coord*
(*map of-int-hom.vec-hom* (*map* (λ*i. of-int* (*w i*) $\cdot_v$ *M* ! *i*) [*0*..<*n*])) !
j $
i)]
by *argo*
then show *?thesis*
using *1*
by *simp*
qed

lemma *int-coord-for-rat*:
fixes *i*::*nat*
assumes *0*≤*i*
assumes *i*<*n*
fixes *v*::*rat vec*
assumes *v*∈*of-int-hom.vec-hom*' *L*
assumes *dim-vec v* = *n*
shows (*lattice-coord v*)$*i*∈ℤ
proof−
let *?hom* = *of-int-hom.vec-hom*
obtain *vint* **where** *v* = *?hom vint*∧ *vint*∈*L* **using** *assms*(*3*) **by** *blast*
moreover then have (*lattice-coord* (*?hom vint*))$*i*∈ℤ **using** *int-coord assms* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

7 Coord-Invariance

This section shows that the algorithm output matches true closest (or near-closest) vector in some trailing coordinates.

definition *I* **where**
I = (*if* ({*i*∈{*0*..<*n*}. ((*sq-norm* (*Mt*!*i*)::*rat*))≤*4*∗*eps-closest*}::*nat set*) ≠ {}
then Max ({*i*∈{*0*..<*n*}. ((*sq-norm* (*Mt*!*i*)::*rat*))≤*4*∗*eps-closest*}::*nat set*) *else* −*1*)

lemma *I-geq*:
shows *I*≥−*1*
unfolding *I-def*
by *simp*
lemma *I-leq*:
shows *I*<*n*
unfolding *I-def*
by *force*

lemma *index-geq-I-big*:
fixes *i*::*nat*
assumes *i*>*I*

assumes *i*<*n*
shows ((*sq-norm* (*Mt*!*i*)::*rat*))>*4*∗*eps-closest*
proof(*rule ccontr*)
assume ¬*?thesis*
then have ((*sq-norm* (*Mt*!*i*)::*rat*))≤*4*∗*eps-closest* **by** *linarith*
then have *i-def*:*i*∈({*i*∈{*0*..<*n*}. ((*sq-norm* (*Mt*!*i*)::*rat*))≤*4*∗*eps-closest*}::*nat set*)
using *assms* **by** *fastforce*
then have ({*i*∈{*0*..<*n*}. ((*sq-norm* (*Mt*!*i*)::*rat*))≤*4*∗*eps-closest*}::*nat set*)≠{} **by** *fast*
moreover then have *I*= *Max* ({*i*∈{*0*..<*n*}. ((*sq-norm* (*Mt*!*i*)::*rat*))≤*4*∗*eps-closest*}::*nat set*) **unfolding** *I-def* **by** *presburger*
moreover have *finite* ({*i*∈{*0*..<*n*}. ((*sq-norm* (*Mt*!*i*)::*rat*))≤*4*∗*eps-closest*}::*nat set*)
by *simp*
ultimately show *False* **using** *assms i-def eq-Max-iff* **by** *auto*
qed

lemma *scalar-prod-gs-from-lattice-coord*:
fixes *i*::*nat*
fixes *v*::*rat vec*
assumes *dim-vec v* = *n*
assumes *i*<*n*
shows *v*•*Mt*!*i*=*sum-list* (*map* (λ*k*. (*lattice-coord v*)$*k* ∗ (((*RAT M*)!*k*)•*Mt*!*i*)) [*i*..<*n*])
proof(−)
let *?lc* = *lattice-coord v*
let *?recover* = ((*map* (λ*j*. *?lc*$*j* ·*v* (*RAT M*)!*j*) [*0* ..< *n*]))
let *?gsv* = *Mt*!*i*
have *v* = *gs.sumlist ?recover*
using *recover-from-lattice-coord*[*of v*] *assms*
by *blast*
then have *split-ip*: *v* • *?gsv* = (*gs.sumlist* (*map* (λ*j*. *?lc*$*j* ·*v* (*RAT M*)!*j*) [*0* ..< *n*]))• *?gsv*
by *simp*
have ⋀*u*. *u*∈*set ?recover*⟹*u*∈*carrier-vec n*
proof(−)
fix *u*::*rat vec*
assume *u-init*:*u*∈ *set ?recover*
then have *index-small*:*find-index ?recover u* < *length ?recover*
by (*meson find-index-leq-length*)
then have *carrier-v-ind-M*:(*RAT M*)!(*find-index ?recover u*)∈*carrier-vec n*
using *carrier-vecI*[*of* (*RAT M*)!(*find-index ?recover u*) *n*]
access-index-M-dim
by (*smt* (*z3*) *M-locale-1 gram-schmidt-fs-Rn.f-carrier length-map map-nth*)
then have *u*=*?recover*!(*find-index ?recover u*)
using *u-init*
by (*simp add*: *find-index-in-set*)
then have *u*=(λ*j*. *?lc*$*j* ·*v* (*RAT M*)!*j*) (*find-index ?recover u*)
using *u-init*

```
          List.nth-map[of find-index ?recover u [0..<n] (λj. ?lc$j ·v (RAT M)!j)]
          index-small
      by auto
    then have u = ?lc$(find-index ?recover u)·v (RAT M)!(find-index ?recover u)
      by simp
    then show u∈carrier-vec n
      using carrier-v-ind-M
            smult-carrier-vec[of ?lc$(find-index ?recover u) (RAT M)!(find-index
?recover u) n]
      by presburger
  qed
  then have result-sumlist-L:v · ?gsv =  sum-list (map (λw. w · ?gsv) ?recover)
    using split-ip
        gs.scalar-prod-left-sum-distrib[of ?recover ?gsv]
    by (metis (no-types, lifting) assms(2) carrier-dim-vec dim-vecs-in-Mt)
  let ?L=(map (λw. w · ?gsv) ?recover)
  have 2:⋀k. k<n⟹?L!k = ?lc$k ∗ ((RAT M)!k· ?gsv)
  proof(−)
    fix k::nat
    assume k-bound:k<n
    then have ?L!k= (λw. w · ?gsv) (?recover!k)
      by force
    then have ?L!k = ?recover!k · ?gsv
      by simp
    then have ?L!k = ((λj. (?lc$j ·v (RAT M)!j)) k)  · ?gsv
      using List.nth-map[of k [0..<n] (λj. (?lc$j ·v (RAT M)!j))] k-bound
      by simp
    then have ?L!k = (?lc$k ·v(RAT M)!k)· ?gsv
      by simp
    then show ?L!k =?lc$k ∗ ((RAT M)!k· ?gsv)
      using smult-scalar-prod-distrib[of (RAT M)!k n ?gsv ?L!k]
          access-index-M-dim
          dim-vecs-in-Mt[of i]
          carrier-vecI[of ?gsv n]
          k-bound
          assms
      by force
  qed
  moreover have length ?L = n
    by fastforce
  ultimately have 1:?L = (map (λk. ?lc$k ∗ ((RAT M)!k· ?gsv)) [0..<n])
    by auto
  moreover then have filt:⋀k. k<i⟹ (λk. ?lc$k ∗ ((RAT M)!k· ?gsv)) k =0
  proof(−)
  fix k::nat
  assume tri:k<i
  then have (?gsv ·(RAT M)!k) = 0
    using gram-schmidt-fs-lin-indpt.gso-scalar-zero[of n (RAT M) i k]
        M-locale-2
```

```
        Mt-gso-connect[of i]
        assms(2)
        more-dim
    by presburger
  then have ((RAT M)!k)·?gsv = 0
    using comm-scalar-prod[of ((RAT M)!k) n ?gsv ]
        access-index-M-dim[of k]
        tri
        assms(2)
        dim-vecs-in-Mt[of i]
        carrier-vecI[of ?gsv] carrier-vecI[of ((RAT M)!k)]
    by fastforce
  then have ?lc$k ∗ ((RAT M)!k· ?gsv) = 0
    by simp
  then show (λk. ?lc$k ∗ ((RAT M)!k· ?gsv)) k =0
    by blast
qed
  moreover have k∈ set [0..<n]∧ ¬i≤k⟹k<i
    by linarith
  ultimately have sum-list ?L = sum-list (map (λk. ?lc$k ∗ ((RAT M)!k· ?gsv))
(filter (λk. i≤k) [0..<n] ) )
    using sum-list-map-filter[of [0..<n] (λk. i≤k) (λk. ?lc$k ∗ ((RAT M)!k· ?gsv))
]
    by (metis (no-types, lifting) le-eq-less-or-eq nat-neq-iff)
  moreover have (filter (λk. i≤k) [0..<n] ) = [i..<n]
    using assms(2) bot-nat-0.extremum filter-upt
    by presburger
  ultimately have sum-list ?L = sum-list (map (λk. ?lc$k ∗ ((RAT M)!k· ?gsv))
[i..<n])
    by presburger
  then show ?thesis
    using result-sumlist-L
    by simp
qed

lemma correct-coord-help:
  fixes i::nat
  assumes i<(int n)−1
  assumes witness v (eps-closest)
  assumes 0<i
  shows (lattice-coord (s i))$(n−i)=(lattice-coord v)$(n−i)
        ∧ ( (s i) · Mt!(n−i) = v · Mt!(n−i) )
  using assms
proof(induct i rule: less-induct)
  case (less i)
  let ?lcs = (lattice-coord (s i))
  let ?lcIs = λi. lattice-coord (s i)$(n−i)
  let ?lcv = lattice-coord v
  let ?gsv = Mt!(n−(i))
```

have *leq*:(*int n*)−*I*≤*n*+*1*
  using *I-geq*
  by *simp*
moreover have *nonbase*:*0*<*i*
  using *less* **by** *blast*
then have *1*:*i*≤*n*
  using *leq less*
  by *linarith*
moreover have *nms*:*n*−(*i*)<*n*
  using *1 nonbase* **by** *linarith*
ultimately have *s-ip*:(*s* (*i*)) · *?gsv* = *sum-list* (*map* (λ*j*. *?lcs*$*j* ∗((*RAT M*)!*j*· *?gsv*)) [*n*−(*i*)..<*n*])
  using *scalar-prod-gs-from-lattice-coord*[*of s* (*i*) *n*−(*i*)]
      s-dim[*of i*] **by** *force*
have *dim-v*:*dim-vec v* = *n*
  using *assms*(*2*)
  unfolding *witness-def*
  by *blast*
then have *v-ip*:*v* · *?gsv* = *sum-list* (*map* (λ*j*. *?lcv*$*j* ∗((*RAT M*)!*j*· *?gsv*)) [*n*−(*i*)..<*n*])
  unfolding *witness-def*
  using *scalar-prod-gs-from-lattice-coord*[*of v n*−*i*]
      nms assms(*2*)
      carrier-vecI[*of v n*]
  by *satx*
have [*n*−*i*..<*n*]≠[] **using** *nms* **by** *auto*
then have *split-indices*:[*n*−(*i*)..<*n*] = (*n*−*i*) # [*n*−(*i*)+*1*..<*n*]
  by (*simp add*: *upt-eq-Cons-conv*)
then have *split-s-list*:(*map* (λ*j*. *?lcs*$*j* ∗((*RAT M*)!*j*· *?gsv*)) [*n*−(*i*)..<*n*]) =
    ((λ*j*. *?lcs*$*j* ∗((*RAT M*)!*j*· *?gsv*)) (*n*−(*i*)))#(*map* (λ*j*. *?lcs*$*j* ∗((*RAT M*)!*j*· *?gsv*)) [*n*−(*i*)+*1*..<*n*])
  by *simp*
then have *split-s-ip-pre*:(*s* (*i*)) · *?gsv* = ((λ*j*. *?lcs*$*j* ∗((*RAT M*)!*j*· *?gsv*)) (*n*−(*i*)))
    + *sum-list* (*map* (λ*j*. *?lcs*$*j* ∗((*RAT M*)!*j*· *?gsv*)) [*n*−(*i*)+*1*..<*n*])
  using *s-ip*
  by *force*
then have *split-s-ip*: (*s* (*i*)) · *?gsv* = ((λ*j*. *?lcs*$*j* ∗((*RAT M*)!*j*· *?gsv*)) (*n*−(*i*)))
    + *sum-list* (*map* (λ*j*. *?lcs*$*j* ∗((*RAT M*)!*j*· *?gsv*)) [*n*−*i*+*1*..<*n*])
  by *presburger*
have *split-v-list*:(*map* (λ*j*. *?lcv*$*j* ∗((*RAT M*)!*j*· *?gsv*)) [*n*−(*i*)..<*n*]) =
    ((λ*j*. *?lcv*$*j* ∗((*RAT M*)!*j*· *?gsv*)) (*n*−(*i*)))#(*map* (λ*j*. *?lcv*$*j* ∗((*RAT M*)!*j*· *?gsv*)) [*n*−(*i*)+*1*..<*n*])
  using *split-indices* **by** *simp*
then have *split-v-ip-pre*:*v* · *?gsv* = ((λ*j*. *?lcv*$*j* ∗((*RAT M*)!*j*· *?gsv*)) (*n*−(*i*)))
    + *sum-list* (*map* (λ*j*. *?lcv*$*j* ∗((*RAT M*)!*j*· *?gsv*)) [*n*−(*i*)+*1*..<*n*])
  using *v-ip*

by force
then have split-v-ip:v · ?gsv = ((λj. ?lcv$j ∗((RAT M)!j· ?gsv)) (n−(i)))
+ sum-list (map (λj. ?lcv$j ∗((RAT M)!j· ?gsv)) [n−i+1..<n])
by presburger
have use-coord-inv: (λj. ?lcs$j ∗((RAT M)!j· ?gsv)) k = (λj. ?lcv$j ∗((RAT M)!j· ?gsv)) k **if** k-bound: k<n ∧ k≥n−i+1 **for** k
proof −
have nmssmall:n−k<i
using k-bound **by** linarith
then have arith:(n−k)+(i − (n−k)) = i
using k-bound 1 **by** linarith
have 2:0<n−k
using k-bound **by** linarith
moreover have 3:(n−k)+(i − (n−k))≤n
using 1 arith **by** linarith
moreover have 4:n−k≤n−k **by** auto
ultimately have 5:lattice-coord (s (n−k + (i − (n−k)))) $ (n−(n−k)) = lattice-coord (s (n−k)) $ (n−(n−k))
using coord-invariance[of n−k n−k (i)−(n−k)] **by** blast
also have cancel:n−(n−k) =k
using k-bound 2 **by** auto
then have ?lcs$k = ?lcIs (n−k)
using arith 5 **by** presburger
moreover have int (n−k)<int n −I
using assms nmssmall less **by** linarith
ultimately have ?lcs$k = ?lcv$(n−(n−k))
using less(1)[of n−k] nmssmall assms(2) 2 **by** argo
then have ?lcs$k = ?lcv$k
using cancel **by** presburger
then have ?lcs$k ∗((RAT M)!k· ?gsv) = ?lcv$k ∗((RAT M)!k· ?gsv)
by simp
then show (λj. ?lcs$j ∗((RAT M)!j· ?gsv)) k = (λj. ?lcv$j ∗((RAT M)!j· ?gsv)) k
by simp
qed
then have (map (λj. ?lcs$j ∗((RAT M)!j· ?gsv)) [n−i+1..<n])
= (map (λj. ?lcv$j ∗((RAT M)!j· ?gsv)) [n−i+1..<n])
by simp
then have sum-list (map (λj. ?lcs$j ∗((RAT M)!j· ?gsv)) [n−i+1..<n])
= sum-list (map (λj. ?lcv$j ∗((RAT M)!j· ?gsv)) [n−i+1..<n])
by presburger
then have (s i) · ?gsv =
((λj. ?lcs$j ∗((RAT M)!j· ?gsv)) (n−i)) +
sum-list (map (λj. ?lcv$j ∗((RAT M)!j· ?gsv)) [n−i+1..<n])
using split-s-ip **by** argo
then have (s i) · ?gsv − v · ?gsv =
((λj. ?lcs$j ∗((RAT M)!j· ?gsv)) (n−i))−
((λj. ?lcv$j ∗((RAT M)!j· ?gsv)) (n−i))
using split-v-ip **by** linarith

then have $(s\ i)\ \cdot$ *?gsv* $-\ v\ \cdot$ *?gsv* $=$ $((?lcs\$(n-i) - ?lcv\$(n-i)) * ((RAT\ M)!(n-i)\cdot ?gsv))$
by *algebra*
then have *case-2-from-case-1*:$(s\ i)\ \cdot$ *?gsv* $-\ v\ \cdot$ *?gsv* $=$ $((?lcs\$(n-i) - ?lcv\$(n-i)) * (sq\text{-}norm\ ?gsv))$
using *one-diag*[*of n− i*] *1 nms*
by *fastforce*
then have *abs* $((s\ i)\ \cdot\ ?gsv - v\ \cdot\ ?gsv) = abs(?lcs\$(n-i) - ?lcv\$(n-i)) * abs(sq\text{-}norm\ ?gsv)$
using *abs-mult* **by** *auto*
then have *a*:*abs* $((s\ i)\ \cdot\ ?gsv - v\ \cdot\ ?gsv) = abs(?lcs\$(n-i) - ?lcv\$(n-i)) * (sq\text{-}norm\ ?gsv)$
by (*metis abs-of-nonneg sq-norm-vec-ge-0*)
have *lattice-coord-equal*:$?lcs\$(n-i) - ?lcv\$(n-i)= 0$
proof(*rule ccontr*)
assume $\neg(?lcs\$(n-i) - ?lcv\$(n-i)= 0)$
then have *contra*:$?lcs\$(n-i) - ?lcv\$(n-i) \neq 0$ **by** *simp*
have $?lcs\$(n-i) - ?lcv\$(n-i) = (?lcs-?lcv)\$(n-i)$
using *index-minus-vec*(*1*)[*of n−i ?lcv ?lcs*]
dim-preserve-lattice-coord[*of v*]
assms(*2*) *nms*
unfolding *witness-def* **by** *argo*
moreover have $?lcs-?lcv = lattice\text{-}coord((s\ i)-v)$
using *mult-minus-distrib-mat-vec*
unfolding *lattice-coord-def*
by (*metis 1 carrier-Ms*(*2*) *carrier-vecI dim-v s-dim*)
ultimately have *use-linear*:$?lcs\$(n-i) - ?lcv\$(n-i) = (lattice\text{-}coord((s\ i)-v))\$(n-i)$
by *presburger*
have $(s\ i)-v\in$ *of-int-hom.vec-hom‘ L*
using *subtract-coset-into-lattice*[*of s i v*]
coset-s[*of i*]
1 assms(*2*)
unfolding *witness-def*
by *linarith*
then have *use-int-coord*:$(lattice\text{-}coord(\ ((s\ i)-v))\ )\$(n-i)\in\mathbb{Z}$
using *int-coord-for-rat*[*of n−i* $((s\ i)-v)$] *1 nms*
by (*simp add*: *dim-v*)
then have $abs((lattice\text{-}coord(\ ((s\ i)-v))\ )\$(n-i))>0$
using *contra use-linear*
by *linarith*
then have $abs((lattice\text{-}coord(\ ((s\ i)-v))\ )\$(n-i))\geq 1$
using *use-int-coord*
by (*simp add*: *Ints-nonzero-abs-ge1 contra use-linear*)
then have $abs(?lcs\$(n-i) - ?lcv\$(n-i))\geq 1$
using *use-linear* **by** *presburger*
then have $abs(?lcs\$(n-i) - ?lcv\$(n-i))*(sq\text{-}norm\ ?gsv)\geq sq\text{-}norm\ ?gsv$
using *sq-norm-vec-ge-0*[*of ?gsv*] *mult-left-mono*[*of 1* $abs(?lcs\$(n-i) - ?lcv\$(n-i))$ *sq-norm ?gsv*] **by** *algebra*
then have *big1*:*abs* $((s\ i)\ \cdot\ ?gsv - v\ \cdot\ ?gsv)\geq sq\text{-}norm\ ?gsv$

```
    using a by argo
  then have tri-ineq:abs(v · ?gsv)≥ abs(abs ((s i)  · ?gsv − v · ?gsv) −abs((s i)
· ?gsv))
    using cancel-ab-semigroup-add-class.diff-right-commute
        cancel-comm-monoid-add-class.diff-cancel diff-zero by linarith
  then have smallhalf:abs((s i)  · ?gsv)≤(1/2)∗(sq-norm ?gsv)
    using small-orth-coord[of i] nonbase 1
    by fastforce
  then have abs((s i)  · ?gsv − v · ?gsv) −abs((s i)  · ?gsv)≥ sq-norm ?gsv −
(1/2)∗(sq-norm ?gsv)
    using big1 by linarith
  then have big2:abs((s i)  · ?gsv − v · ?gsv) −abs((s i)  · ?gsv)≥ (1/2)∗(sq-norm
?gsv)
    by linarith
  then have abs((s i)  · ?gsv − v · ?gsv) −abs((s i)  · ?gsv)≥0
    using sq-norm-vec-ge-0[of ?gsv] by linarith
  then have abs(abs ((s i)  · ?gsv − v · ?gsv) −abs((s i)  · ?gsv))
          = abs((s i)  · ?gsv − v · ?gsv) −abs((s i)  · ?gsv)
    by fastforce
  then have abs(v · ?gsv)≥(1/2)∗(sq-norm ?gsv)
    using big2
    by linarith
  moreover have (1/2)∗(sq-norm ?gsv)≥0
    using sq-norm-vec-ge-0[of ?gsv] by simp
  moreover have abs(v · ?gsv)≥0 by simp
  ultimately have abs(v · ?gsv)^2≥((1/2)∗(sq-norm ?gsv))^2
    using nonneg-power-le by blast
  moreover have (sq-norm v) ∗ (sq-norm ?gsv)≥abs(v · ?gsv)^2
    using scalar-prod-Cauchy[of v n ?gsv]
        carrier-vecI[of v n] assms(2)
        carrier-vecI[of ?gsv] dim-vecs-in-Mt[of n−i] nms
    unfolding witness-def
    by fastforce
  ultimately have sq-norm v ∗ sq-norm ?gsv ≥ ((1/2)∗(sq-norm ?gsv))^2
    by order
  then have sq-norm v ∗ sq-norm ?gsv ≥ (1/2)^2 ∗ (sq-norm ?gsv)^2
    by (metis gs.nat-pow-distrib)
  then have sq-norm v ∗ sq-norm ?gsv ≥ 1/4 ∗ (sq-norm ?gsv)^2
  by (smt (z3) numeral-Bit0-eq-double one-power2 power2-eq-square times-divide-times-eq)
  moreover have sq-norm ?gsv > 0
    using gram-schmidt-fs-lin-indpt.sq-norm-pos[of n RAT M n−i]
        M-locale-2 M-locale-1 gram-schmidt-fs-Rn.main-connect[of n (RAT M)]
        nms by force
  ultimately have big:sq-norm v ≥ 1/4 ∗ sq-norm ?gsv
    by (simp add: power2-eq-square)
  have n−i>I
    using less by linarith
  then have big-again:sq-norm ?gsv > 4∗eps-closest
    using index-geq-I-big[of n−i] nms by simp
```

```
    then have sq-norm v> 1/4 *4*eps-closest
      using big by fastforce
    then have sq-norm v > eps-closest by auto
    then show False
      using assms(2)
      unfolding witness-def
      by linarith
  qed
  then have piece1: lattice-coord (s i) $ (n − i) = lattice-coord v $ (n − i)
    using lattice-coord-equal by simp
  have (s i)  · ?gsv − v · ?gsv = 0
    using lattice-coord-equal case-2-from-case-1
    by algebra
  then show ?case using piece1 by simp
qed

lemma correct-coord:
  fixes v::rat vec
  fixes k::nat
  assumes witness v eps-closest
  assumes I<k
  assumes k<n
  shows (s n) · Mt!(k) = v · Mt!(k)
proof −
  have (s n) · Mt!(k) = (s (n−k)) · Mt!(k)
    using coord-invariance[of n−k n−k k] assms
    by force
  moreover have (s (n−k)) · Mt!(k) = v · Mt!(k)
    using correct-coord-help[of n−k v] assms
    by simp
  ultimately show ?thesis by simp
qed
```

8 Main Theorem

This section culminates in the main theorem.

```
lemma sq-norm-from-Mt:
  fixes v::rat vec
  assumes v-carr:v∈carrier-vec n
  shows sq-norm v = sum-list (map (λi. (v·Mt!i)^2/(sq-norm (Mt!i))) [0..<n])
proof−
  let ?Mt-inv-list = map (λi. (1/sq-norm(Mt!i))·v (Mt!i)) [0..<n]
  have nonsing:?Mt-inv-list!i ∈ carrier-vec n if i:0≤i∧i<n for i
  proof−
    have 0< sq-norm(Mt!i)
    using gram-schmidt-fs-lin-indpt.sq-norm-pos[of n RAT M i]
      M-locale-1 gram-schmidt-fs-Rn.main-connect[of n (RAT M)] i
    by (simp add: M-locale-2)
```

```
  then have 0<1/sq-norm(Mt!i) by fastforce
  then have (1/sq-norm(Mt!i))·v (Mt!i)∈carrier-vec n
    using carrier-vecI[of (Mt!i)] dim-vecs-in-Mt[of i] i by blast
  moreover have ?Mt-inv-list!i = (1/sq-norm(Mt!i))·v (Mt!i)
    using i by simp
  ultimately show ?thesis by argo
qed
  let ?Mt-inv-mat = mat-of-rows n ?Mt-inv-list
  have carrier-mat-inv:?Mt-inv-mat∈carrier-mat n n by fastforce
  let ?vMt = ?Mt-inv-mat *v v
  have ?vMt$i = ((1/sq-norm(Mt!i))·v (Mt!i))•v if i:0≤i∧i<n for i
    using i nonsing[of i] by auto
  have dim-vMt:dim-vec ?vMt = n
    using carrier-mat-inv v-carr by auto
  let ?Mt-mat = mat-of-cols n Mt
  have l:length Mt = n
    using gs.gram-schmidt-result[of RAT M Mt] basis dim-vecs-in-M
    unfolding gs.lin-indpt-list-def
    by fastforce
  then have carrier-mat-Mt:?Mt-mat∈carrier-mat n n
    using dim-vecs-in-Mt carrier-vecI by auto
  then have to-sumlist:?Mt-mat*v ?vMt = gs.sumlist (map (λj. ?vMt$j ·v (col
?Mt-mat j)) [0 ..< n])
    using mat-mul-to-sum-list[of ?vMt ?Mt-mat] dim-vMt
    by fastforce
  have ?vMt$i ·v (col ?Mt-mat i) = (1/sq-norm(Mt!i))* ((Mt!i)•v) ·v Mt!i if
i:0≤i∧i<n for i
    using i l dim-vecs-in-Mt v-carr carrier-vecI by fastforce
  then have (map (λj. ?vMt$j ·v (col ?Mt-mat j)) [0 ..< n])
          = (map (λj. (1/sq-norm(Mt!j))* ((Mt!j)•v) ·v Mt!j) [0 ..< n])
    by simp
  then have 1:gs.sumlist (map (λj. ?vMt$j ·v (col ?Mt-mat j)) [0 ..< n])
              =gs.sumlist (map (λj. (1/sq-norm(Mt!j))* ((Mt!j)•v) ·v Mt!j) [0 ..<
n])
    by presburger
  then have 2:?Mt-mat*v ?vMt = gs.sumlist (map (λj. (1/sq-norm(Mt!j))* ((Mt!j)•v)
·v Mt!j) [0 ..< n])
    using to-sumlist by argo
  have ?Mt-mat *v ?vMt = (?Mt-mat * ?Mt-inv-mat)*v v
    using carrier-mat-Mt carrier-mat-inv v-carr by auto
  have (?Mt-inv-mat*?Mt-mat)$$(i,j) = (1m n)$$(i,j)
    if sensible-indices:0≤i ∧ i<n ∧ 0≤j ∧ j<n for i j
  proof−
    have (?Mt-inv-mat*?Mt-mat)$$(i,j) = (row ?Mt-inv-mat i)•(col ?Mt-mat j)
      using sensible-indices carrier-mat-Mt carrier-mat-inv by auto
    then have (?Mt-inv-mat*?Mt-mat)$$(i,j) = ?Mt-inv-list!i•Mt!j
      using sensible-indices carrier-mat-Mt carrier-mat-inv nonsing
      by auto
    then have (?Mt-inv-mat*?Mt-mat)$$(i,j) = ((1/sq-norm(Mt!i))·v (Mt!i))•Mt!j
```

```
      using sensible-indices by simp
    then have (?Mt-inv-mat∗?Mt-mat)$$(i,j) = (1/sq-norm(Mt!i)) ∗((Mt!i)·(Mt!j))
      using dim-vecs-in-Mt[of i] dim-vecs-in-Mt[of j] sensible-indices by auto
    moreover have (1/sq-norm(Mt!i)) ∗((Mt!i)·(Mt!j)) = (if i=j then 1 else 0)
    proof(cases i=j)
      case diag:True
      have nonzero:0< sq-norm(Mt!i)
        using gram-schmidt-fs-lin-indpt.sq-norm-pos[of n RAT M i]
        M-locale-1 gram-schmidt-fs-Rn.main-connect[of n (RAT M)] sensible-indices
          by (simp add: M-locale-2)
     have (1/sq-norm(Mt!i)) ∗((Mt!i)·(Mt!j)) = (1/sq-norm(Mt!i)) ∗ sq-norm(Mt!i)
        using sensible-indices diag sq-norm-vec-as-cscalar-prod[of Mt!i] by auto
      then have (1/sq-norm(Mt!i)) ∗((Mt!i)·(Mt!j)) = 1
        using nonzero by auto
      then show ?thesis using diag by argo
    next
      case off:False
      have nonzero:0< sq-norm(Mt!i)
        using gram-schmidt-fs-lin-indpt.sq-norm-pos[of n RAT M i]
        M-locale-1 gram-schmidt-fs-Rn.main-connect[of n (RAT M)] sensible-indices
        by (simp add: M-locale-2)
      then have 0<1/sq-norm(Mt!i) by simp
      moreover have ((Mt!i)·(Mt!j)) = 0
          using gram-schmidt-fs-lin-indpt.orthogonal[of n (RAT) M i j] off sensi-
ble-indices
              M-locale-1 M-locale-2 gram-schmidt-fs-Rn.main-connect
        by force
      ultimately show ?thesis using off by algebra
    qed
    moreover then have (1/sq-norm(Mt!i)) ∗((Mt!i)·(Mt!j)) = (1_m n)$$(i,j)
      using sensible-indices unfolding one-mat-def by simp
    ultimately show ?thesis by presburger
  qed
  then have inv-Mt:(?Mt-inv-mat∗?Mt-mat) = 1_m n
    using carrier-mat-inv carrier-mat-Mt
    by fastforce
  then have ?Mt-mat ∗ ?Mt-inv-mat = 1_m n
     using mat-mult-left-right-inverse[of ?Mt-inv-mat n ?Mt-mat] carrier-mat-inv
carrier-mat-Mt
    by argo
  then have 3:(?Mt-mat ∗ ?Mt-inv-mat)∗_v v = v
    using v-carr by simp
  then have 4:v = gs.sumlist (map (λj. (1/sq-norm(Mt!j))∗ ((Mt!j)·v)  ·_v Mt!j)
[0 ..< n])
    using v-carr carrier-mat-inv carrier-mat-Mt 1 2 by auto
  have (map (λj. (1/sq-norm(Mt!j))∗ ((Mt!j)·v)  ·_v Mt!j) [0 ..< n])
        = (map (λj. (1/sq-norm(Mt!j))∗ ((Mt!j)·v)  ·_v gs.gso j) [0 ..< n])
    using M-locale-1 gram-schmidt-fs-Rn.main-connect[of n RAT M]
    by auto
```

then have *gs.sumlist* (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*) ·$_v$ *Mt*!*j*) [*0* ..< *n*])
= *gs.sumlist* (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*) ·$_v$ *gs.gso j*) [*0* ..< *n*])
by *argo*
then have *v* = *gs.sumlist* (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*) ·$_v$ *gs.gso j*) [*0* ..< *n*])
using *4* **by** *argo*
then have *v*·*v* = *gs.sumlist* (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*) ·$_v$ *gs.gso j*) [*0* ..< *n*])·
gs.sumlist (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*) ·$_v$ *gs.gso j*) [*0* ..< *n*])
by *simp*
then have *a*:*v*·*v* =
sum-list(*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*gs.gso j* · *gs.gso j*)) [*0*..<*n*])
using *gram-schmidt-fs-lin-indpt.scalar-prod-lincomb-gso*[
of n RAT M n (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)) (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*))]
M-locale-2
M-locale-1 gram-schmidt-fs-Rn.main-connect[*of n RAT M*] **by** *force*
have (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*gs.gso j* · *gs.gso j*)) [*0*..<*n*])
= (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*Mt*!*j* · *Mt*!*j*)) [*0*..<*n*])
using *M-locale-1 gram-schmidt-fs-Rn.main-connect*[*of n RAT M*]
by *auto*
then have *b*:*sum-list* (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*gs.gso j* · *gs.gso j*)) [*0*..<*n*])
=*sum-list* (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*Mt*!*j* · *Mt*!*j*)) [*0*..<*n*])
by *argo*
have (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*Mt*!*j* · *Mt*!*j*) =
(*v*·(*Mt*!*j*))^*2*/(*sq-norm* (*Mt*!*j*)) **if** *sensible-indices*:*0*≤*j*∧*j*<*n* **for** *j*
proof−
have *nonzero*:*0*< *sq-norm*(*Mt*!*j*)
using *gram-schmidt-fs-lin-indpt.sq-norm-pos*[*of n RAT M j*]
M-locale-1 gram-schmidt-fs-Rn.main-connect[*of n* (*RAT M*)] *sensible-indices*
by (*simp add*: *M-locale-2*)
moreover have (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*Mt*!*j* · *Mt*!*j*)
= (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗*sq-norm* (*Mt*!*j*)
using *sq-norm-vec-as-cscalar-prod*[*of Mt*!*j*] **by** *force*
moreover have (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗ *sq-norm* (*Mt*!*j*)
= ((*Mt*!*j*)·*v*)^*2* ∗ (*1*/*sq-norm*(*Mt*!*j*))^*2* ∗*sq-norm* (*Mt*!*j*)

by (*simp add*: *power2-eq-square*)
moreover have $((Mt!j) \cdot v)$^2 $*$ $(1/sq\text{-}norm(Mt!j))$^2 $*sq\text{-}norm$ $(Mt!j)$ = $((Mt!j) \cdot v)$^2/$(sq\text{-}norm(Mt!j))$
using *nonzero*
by (*simp add*: *divide-divide-eq-left′ power2-eq-square*)
moreover have $(Mt!j) \cdot v = v \cdot (Mt!j)$ **using** *v-carr dim-vecs-in-Mt sensible-indices*
by (*metis carrier-vecI comm-scalar-prod*)
ultimately show *?thesis* **by** *argo*
qed
then have (*map* (λ*j*. (*1/sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*1/sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*Mt*!*j* · *Mt*!*j*)) [*0*..<*n*])
= (*map* (λ*j*. (*v*·(*Mt*!*j*))^*2*/(*sq-norm*(*Mt*!*j*))) [*0*..<*n*]) **by** *force*
then have *c*:*sum-list* (*map* (λ*j*. (*1/sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*1/sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*)∗(*Mt*!*j* · *Mt*!*j*)) [*0*..<*n*])
= *sum-list* (*map* (λ*j*. (*v*·(*Mt*!*j*))^*2*/(*sq-norm*(*Mt*!*j*))) [*0*..<*n*]) **by** *argo*
then have *v*·*v* = *sum-list* (*map* (λ*j*. (*v*·(*Mt*!*j*))^*2*/(*sq-norm*(*Mt*!*j*))) [*0*..<*n*]) **using** *a b c* **by** *argo*
moreover have *v*·*v* = *v*·*cv* **by** *force*
ultimately show *?thesis* **using** *sq-norm-vec-as-cscalar-prod*[*of v*] *v-carr* **by** *argo*
qed

lemma *bound-help*:
fixes *N*::*nat*
shows *real-of-rat* ((*rat-of-int N*)∗α^*N*) ∗ *epsilon*≤*2*^*N*
proof(*induct N*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc N*)
let *?SN* = *Suc N*
have *?SN*=*1*∨*?SN*=*2*∨*2*<*?SN* **by** *fastforce*
then show *?case*
proof(*elim disjE*)
{**assume** *1*:*?SN* = *1*
then have *real-of-rat* ((*rat-of-int ?SN*)∗α^*?SN*)∗*epsilon* = *real-of-rat* ((*rat-of-int 1*)∗*4/3*)∗*11/10*
unfolding α*-def epsilon-def* **by** *auto*
also have *real-of-rat* ((*rat-of-int 1*)∗*4/3*)∗*11/10* = *real-of-rat* (*4/3*)∗*11/10* **by** *force*
also have *real-of-rat* (*4/3*)∗*11/10* =*real-of-rat* ((*4/3*)∗ *11/10*)
by (*simp add*: *of-rat-hom.hom-div*)
also have *real-of-rat* ((*4/3*)∗ *11/10*) = *real-of-rat* (*44/30*) **by** *auto*
also have *real-of-rat* (*44/30*)≤(*2*::*real*)
by (*simp add*: *of-rat-hom.hom-div*)
finally show *?thesis* **using** *1* **by** *simp*}
next
{**assume** *2*:*?SN*=*2*
then have *real-of-rat* ((*rat-of-int ?SN*)∗α^*?SN*)∗*epsilon* = *real-of-rat* ((*rat-of-int 2*)∗(*4/3*)^*2*)∗*11/10*

unfolding *α-def epsilon-def*
by (*metis int-ops*(*3*) *times-divide-eq-right*)
also have ((*4*::*rat*)/*3*)^*2* = (*4*∗*4*)/(*3*∗*3*)
using *power2-eq-square*[*of 4/3*] *times-divide-times-eq*[*of 4 3 4 3*] **by** *metis*
also have (*4*∗(*4*::*rat*))/(*3*∗*3*) = *16/9* **by** *auto*
finally have *real-of-rat* ((*rat-of-int ?SN*)∗α^*?SN*)∗*epsilon*= *real-of-rat* ((*rat-of-int 2*)∗(*16/9*))∗*11/10*
by *blast*
also have (*rat-of-int 2*)∗(*16/9*) = *32/9* **by** *force*
finally have *real-of-rat* ((*rat-of-int ?SN*)∗α^*?SN*)∗*epsilon* = *real-of-rat* (*32 / 9*) ∗ *11 / 10*
by *simp*
also have *real-of-rat* (*32 / 9*) ∗ *11 / 10* = *real-of-rat* (*32 / 9* ∗(*11 / 10*))
using *of-rat-hom.hom-mult*[*of 32/9 11/10*]
by (*simp add*: *of-rat-hom.hom-div*)
also have *real-of-rat* (*32 / 9* ∗(*11 / 10*)) = *real-of-rat* (*352/90*)
using *times-divide-times-eq*[*of 32 9 11 10*] **by** *force*
also have *352/90*≤(*4*::*rat*) **by** *linarith*
also have (*4*::*rat*) = *2*^*?SN* **using** *2* **by** *auto*
finally show *?thesis*
by (*simp add*: *2 gs.cring-simprules*(*14*) *int-ops*(*3*) *of-rat-hom.hom-power of-rat-less-eq*)**}**
next
{assume *ind*:*?SN*>*2*
then have *N*>*0* **by** *simp*
then have *?SN* = *N*∗(*?SN/N*) **by** *auto*
moreover have α^*?SN* = α^*N*∗α **by** *auto*
ultimately have *real-of-rat* ((*rat-of-int ?SN*)∗α^*?SN*) = (*N*∗(*?SN/N*)) ∗ (*real-of-rat* (α^*N*∗α))
by (*metis of-int-of-nat-eq of-rat-mult of-rat-of-nat-eq*)
also have (*N*∗(*?SN/N*)) ∗ *real-of-rat* (α^*N*∗α) = *real-of-rat* ((*rat-of-int N*) ∗ α^*N*) ∗ ((*?SN/N*) ∗(*real-of-rat* α))
by (*simp add*: ‹*real* (*Suc N*) = *real N* ∗ (*real* (*Suc N*) / *real N*)› *gs.cring-simprules*(*11*) *mult-of-int-commute of-rat-divide of-rat-mult*)
finally have *real-of-rat* ((*rat-of-int ?SN*)∗α^*?SN*) ∗ *epsilon* = *real-of-rat* ((*rat-of-int N*) ∗ α^*N*) ∗ ((*?SN/N*) ∗(*real-of-rat* α)) ∗ *epsilon*
by *presburger*
then have *real-of-rat* ((*rat-of-int ?SN*)∗α^*?SN*) ∗ *epsilon* = *real-of-rat* ((*rat-of-int N*) ∗ α^*N*) ∗ *epsilon* ∗ ((*?SN/N*) ∗(*real-of-rat* α))
by *argo*
moreover have ((*?SN/N*) ∗(*real-of-rat* α))≤*2*
proof−
have *N-big*:*2*≤*N* **using** *ind*
by *force*
then have *4*≤*2*∗*N* **by** *fastforce*
then have *4*∗*N*+*4*≤*6*∗*N* **by** *fastforce*
then have *4/3*∗(*Suc N*)≤*2*∗*N* **by** *auto*
moreover have *0*<*1/N* **using** *N-big* **by** *simp*
ultimately have (*4/3*∗*?SN*)∗ (*1/N*)≤ *2*∗*N*∗(*1/N*)

using *N-big mult-right-mono*[*of* (*4/3*∗*?SN*) *2*∗*N* (*1/N*)] **by** *linarith*
then have (*4/3*∗*?SN*)/*N*≤ *2*∗*N*/*N* **by** *argo*
then have *4/3*∗(*?SN* / *N*)≤ *2*∗(*N*/*N*) **by** *linarith*
then have *4/3*∗(*?SN*/*N*)≤ *2* **using** *N-big* **by** *auto*
moreover have *4/3* = *real-of-rat* α **using** *of-rat-divide* **unfolding** α*-def*
by (*metis of-rat-numeral-eq*)
ultimately have (*real-of-rat* α)∗(*?SN*/*N*)≤ *2* **by** *algebra*
then show *?thesis* **by** *argo*
qed
moreover have
0≤*real-of-rat* (*rat-of-int* (*int N*) ∗ α ^ *N*) ∗ *epsilon* **unfolding** α*-def*
epsilon-def **by** *force*
moreover have *0*≤(*real-of-rat* α)∗(*?SN*/*N*) **unfolding** α*-def* **by** *simp*
ultimately have *real-of-rat* ((*rat-of-int ?SN*)∗α^*?SN*) ∗ *epsilon*≤*2*^*N* ∗ *2*
using *Suc mult-mono*[*of*
real-of-rat (*rat-of-int* (*int N*) ∗ α ^ *N*) ∗ *epsilon*
2^*N*
((*?SN*/*N*) ∗(*real-of-rat* α))
2] **by** *argo*
then show *?thesis* **by** *simp*}
qed
qed

lemma *present-bound-nicely*:
fixes *N*::*nat*
shows *real-of-rat* ((*rat-of-int N*)∗α^*N*∗ *eps-closest*)≤*2*^*N*∗*closest-distance-sq*
proof−
have *real-of-rat eps-closest*≤ *epsilon*∗*closest-distance-sq*
using *eps-closest-lemma* **unfolding** *close-condition-def* **by** *fastforce*
moreover have *0*≤(*rat-of-int N*)∗α^*N* **unfolding** α*-def* **by** *simp*
ultimately have *real-of-rat* ((*rat-of-int N*)∗α^*N* ∗ *eps-closest*)≤ *real-of-rat*
((*rat-of-int N*)∗α^*N*) ∗ *epsilon*∗*closest-distance-sq*
by (*metis ab-semigroup-mult-class.mult-ac*(*1*) *mult-left-mono of-rat-hom.hom-mult*
zero-le-of-rat-iff)
also have *real-of-rat* ((*rat-of-int N*)∗α^*N*) ∗ *epsilon*∗*closest-distance-sq*≤*2*^*N*∗*closest-distance-sq*
using *bound-help*[*of N*] *closest-distance-sq-pos mult-right-mono* **by** *fast*
finally show *?thesis* **by** *force*
qed

lemma *basis-decay*:
fixes *i*::*nat*
fixes *j*::*nat*
assumes *i*<*n*
assumes *i*+*j*<*n*
shows *sq-norm* (*Mt*!*i*)≤ α^*j*∗*sq-norm*(*Mt*!(*i*+*j*))
using *assms*
proof(*induct j*)
case *0*

```
  have α^0 = 1 by simp
  moreover have sq-norm (Mt!i) = sq-norm(Mt!(i+0)) by simp
  moreover have 0≤ sq-norm(Mt!i)
    using gram-schmidt-fs-lin-indpt.sq-norm-pos[of n RAT M i]
        M-locale-2 M-locale-1 gram-schmidt-fs-Rn.main-connect[of n (RAT M)]
        assms by force
  moreover have (0::rat)≤(1::rat) by force
  ultimately show ?case by simp
next
  case (Suc j)
  have (1::rat) ≤α unfolding α-def by fastforce
  moreover have n≥0 by simp
  ultimately have (1::rat)≤α^j by simp
  moreover have sq-norm (Mt!(i+j))≤α*(sq-norm (Mt!(i+Suc j)))
     using reduced M-locale-1 gram-schmidt-fs-Rn.main-connect[of n (RAT M)]
Suc.prems
    unfolding gs.reduced-def gs.weakly-reduced-def
    by force
  moreover have 0≤ sq-norm (Mt!(i+j))
    using gram-schmidt-fs-lin-indpt.sq-norm-pos[of n RAT M i+j]
     M-locale-2 M-locale-1 gram-schmidt-fs-Rn.main-connect[of n (RAT M)]
     Suc.prems by force
  ultimately have α^j*sq-norm (Mt!(i+j))≤α^j*α*(sq-norm (Mt!(i+Suc j)))
    by simp
  moreover have sq-norm(Mt!i)≤ α^j * sq-norm (Mt!(i+j))
    using Suc by linarith
  ultimately have sq-norm(Mt!i)≤α^j*α*(sq-norm (Mt!(i+Suc j))) by order
  moreover have α^j*α = α^(Suc j) by simp
  ultimately show ?case by argo
qed

lemma basis-decay-cor:
  fixes i::nat
  fixes j::nat
  assumes i<n
  assumes j<n
  assumes i≤j
  shows sq-norm (Mt!i)≤ α^n*sq-norm(Mt!j)
proof−
  have 1:sq-norm (Mt!i)≤ α^(j−i)*sq-norm(Mt!j)
    using basis-decay[of i j−i] assms
    by simp
  have α^(j−i)≤α^n using assms unfolding α-def by force
  then have α^(j−i)*sq-norm(Mt!j)≤α^n*sq-norm(Mt!j)
    using mult-right-mono by blast
  then show ?thesis using 1 by order
qed
```

theorem *Babai-Correct*:
shows *real-of-rat* ((*sq-norm* (*s n*))::*rat*) ≤ *2^n* ∗ *closest-distance-sq*∧ *s n* ∈ *coset*
proof−
let *?s* = *s n*
let *?component* = (λ*i*. (*?s*·*Mt*!*i*)^*2*/(*sq-norm* (*Mt*!*i*)))
obtain *v* **where** *wit-v*:*witness v* (*eps-closest*)
using *witness-exists* **by** *force*
have *split-norm*:*sq-norm ?s* = *sum-list* (*map ?component* [*0*..<*n*])
using *s-dim*[*of n*] *sq-norm-from-Mt*[*of ?s*] **by** *fast*
have *I*+*1*∈ℕ **using** *I-geq*
using *Nats-0 Nats-1 Nats-add R.add.l-inv-ex R.add.r-inv-ex add-diff-cancel-right′*

cring-simprules(*21*) *rangeI range-abs-Nats verit-la-disequality verit-minus-simplify*(*3*)

zabs-def zle-add1-eq-le **by** *auto*
then obtain *Inat* **where** *Inat-def*:*int Inat* = *I*+*1*
using *Nats-cases* **by** *metis*
then have *Inat-small*:*Inat*≤*n* **using** *I-leq* **by** *fastforce*
then have [*0*..<*n*] = [*0*..<*Inat*] @ [*Inat*..<*n*]
by (*metis bot-nat-0.extremum-uniqueI le-Suc-ex nat-le-linear upt-add-eq-append*)
then have *split-norm-sum*:*sq-norm ?s* = *sum-list* (*map ?component* [*0*..<*Inat*])
+ *sum-list* (*map ?component* [*Inat*..<*n*])
using *split-norm* **by** *force*

have *?component i* ≤ *eps-closest* **if** *i*:*Inat*≤*i*∧*i*<*n* **for** *i*
proof−
have *ge0*:*sq-norm* (*Mt*!*i*) > *0*
using *gram-schmidt-fs-lin-indpt.sq-norm-pos*[*of n RAT M i*]
M-locale-2 M-locale-1 gram-schmidt-fs-Rn.main-connect[*of n* (*RAT M*)]
i **by** *force*
then have *?component i* = (*v*· *Mt*!*i*)^*2* / (*sq-norm* (*Mt*!*i*))
using *ge0 correct-coord*[*of v i*] *wit-v Inat-def i*
by *auto*
also have (*v*·*Mt*!*i*)^*2*≤ (*sq-norm v*)∗*sq-norm* (*Mt*!*i*)
using *scalar-prod-Cauchy*[*of v n Mt*!*i*]
dim-vecs-in-Mt[*of i*] *carrier-vecI*[*of v*] *carrier-vecI*[*of Mt*!*i*] *wit-v*
i
unfolding *witness-def*
by *algebra*
also have *sq-norm v* ≤ *eps-closest*
using *wit-v* **unfolding** *witness-def* **by** *fast*
finally show *?thesis* **using** *ge0*
by (*simp add*: *divide-right-mono*)
qed
then have ⋀*x*. *x*∈*set* [*Inat*..<*n*] ⟹ *?component x* ≤ (λ*i*. *eps-closest*) *x* **by** *simp*
then have *sum-list* (*map ?component* [*Inat*..<*n*])≤ *sum-list* (*map* (λ*i*. *eps-closest*)
[*Inat*..<*n*])

```
    using sum-list-mono[of [Inat..<n] ?component (λi. eps-closest)] by argo
  then have right-sum:sum-list (map ?component [Inat..<n])≤(rat-of-nat (n−Inat))∗eps-closest
    using sum-list-triv[of eps-closest [Inat..<n] ] by force
  have (1::rat) ≤α unfolding α-def by fastforce
  moreover have n≥0 by simp
  ultimately have (1::rat)≤α^n by simp
  moreover have (0::rat)≤1 by simp
  moreover have 0≤(rat-of-nat (n−Inat))∗eps-closest
  proof−
    have 0≤(rat-of-nat (n−Inat)) using Inat-small by fast
    moreover have 0≤eps-closest
    proof(cases closest-distance-sq = 0)
      case t:True
     then show ?thesis using eps-closest-lemma closest-distance-sq-pos unfolding
close-condition-def
        by auto
    next
      case f:False
     then show ?thesis using eps-closest-lemma closest-distance-sq-pos unfolding
close-condition-def
        by (smt (verit, del-insts) zero-le-of-rat-iff)
    qed
    ultimately show ?thesis by blast
  qed
  ultimately have (rat-of-nat (n−Inat))∗eps-closest ≤ (rat-of-nat (n−Inat))∗eps-closest
∗ α^n
    using mult-left-mono[of 1 α^n (rat-of-nat (n−Inat))∗eps-closest] by linarith
  then have sum-list (map ?component [Inat..<n])≤(rat-of-nat (n−Inat))∗eps-closest∗α^n
using right-sum by order
   then have right-sum-alpha:sum-list (map ?component [Inat..<n])≤(rat-of-nat
(n−Inat))∗α^n∗eps-closest
    by algebra
  have sum-list (map ?component [0..<Inat]) + sum-list (map ?component [Inat..<n])≤
(rat-of-int n)∗α^n∗eps-closest
  proof(cases Inat = 0)
    case Inat:True
    then have sum-list (map ?component [0..<Inat]) = 0 by auto
   then have sum-list (map ?component [0..<Inat]) + sum-list (map ?component
[Inat..<n])≤(rat-of-int (n−Inat))∗α^n ∗ eps-closest
      using right-sum-alpha by simp
    also have n−Inat = n using Inat by simp
    finally show ?thesis by linarith
  next
    case False
    then have non-zero:Inat>0 by blast
    then have I-not-min:I≥0 using Inat-def by simp
   then have non-empty:I = Max ({i∈{0..<n}. ((sq-norm (Mt!i)::rat))≤4∗eps-closest}::nat
set)
      unfolding I-def by presburger
```

then have *max:Inat−1= Max({i∈{0..<n}. ((sq-norm (Mt!i)::rat))≤4∗eps-closest}::nat set)*
using *Inat-def* **by** *linarith*
then have *Inat −1 ∈ ({i∈{0..<n}. ((sq-norm (Mt!i)::rat))≤4∗eps-closest}::nat set)*
proof−
have *finite ({i∈{0..<n}. ((sq-norm (Mt!i)::rat))≤4∗eps-closest}::nat set)*
by *simp*
moreover have *({i∈{0..<n}. ((sq-norm (Mt!i)::rat))≤4∗eps-closest}::nat set)≠{}*
using *I-not-min* **unfolding** *I-def* **by** *presburger*
ultimately show *Inat −1 ∈ ({i∈{0..<n}. ((sq-norm (Mt!i)::rat))≤4∗eps-closest}::nat set)*
using *max eq-Max-iff* **by** *blast*
qed
then have *2:(sq-norm (Mt!(Inat−1))::rat)≤4∗eps-closest* **by** *blast*
have *(1::rat) ≤α* **unfolding** *α-def* **by** *fastforce*
moreover have *n≥0* **by** *simp*
ultimately have *(1::rat)≤α^n* **by** *simp*
then have *((1/4)::rat)≤1/4 ∗ α^n* **by** *auto*
then have *(0::rat)<1/4∗α^n* **by** *linarith*
moreover have *0<(sq-norm (Mt!(Inat−1))::rat)*
using *gram-schmidt-fs-lin-indpt.sq-norm-pos[of n RAT M Inat−1]*
M-locale-2 M-locale-1 gram-schmidt-fs-Rn.main-connect[of n (RAT M)]
non-zero Inat-small **by** *force*
ultimately have *bound:1/4 ∗α^n∗ (sq-norm (Mt!(Inat−1)))≤ ((1/4 ∗ α^n)∗ 4∗eps-closest)*
using *2* **by** *auto*
have *?component i ≤ α^n ∗eps-closest* **if** *list1:i<Inat* **for** *i*
proof−
have *1:0<n−i* **using** *list1 Inat-small* **by** *simp*
then have *?s•Mt!i = (s (n−i))•Mt!i*
using *coord-invariance[of n−i n−i i]* **by** *fastforce*
then have *abs(?s•Mt!i)≤ (1/2)∗(sq-norm (Mt!i))*
using *small-orth-coord[of n−i] 1* **by** *force*
then have *(?s•Mt!i)^2 ≤ ((1/2)∗(sq-norm (Mt!i)))^2*
by *(meson abs-ge-self abs-le-square-iff ge-trans)*
moreover have *ge0:sq-norm (Mt!i) > 0*
using *gram-schmidt-fs-lin-indpt.sq-norm-pos[of n RAT M i]*
M-locale-2 M-locale-1 gram-schmidt-fs-Rn.main-connect[of n (RAT M)]
list1 Inat-small **by** *force*
ultimately have *?component i ≤((1/2)∗(sq-norm (Mt!i)))^2 / (sq-norm (Mt!i))*
using *divide-right-mono* **by** *auto*
also have *((1/2)∗(sq-norm (Mt!i)))^2/ (sq-norm (Mt!i)) = 1/4 ∗ (sq-norm (Mt!i))^2/ (sq-norm (Mt!i))*
by *(metis (no-types, lifting) gs.cring-simprules(12) numeral-Bit0-eq-double power2-eq-square times-divide-eq-left times-divide-times-eq)*
also have *1/4 ∗ (sq-norm (Mt!i))^2/ (sq-norm (Mt!i)) = 1/4 ∗ (sq-norm*

```
(Mt!i))
        using ge0 by (simp add: power2-eq-square)
      also have 1/4*sq-norm (Mt!i) ≤ 1/4*α^n * (sq-norm (Mt!(Inat−1)))
        using basis-decay-cor[of i Inat−1] list1 Inat-small mult-left-mono[
            of sq-norm (Mt!i) α^n * (sq-norm (Mt!(Inat−1))) 1/4]
          by linarith
      finally have ?component i ≤ 1/4 * α^n * 4 *eps-closest
        using bound by linarith
      also have 1/4 * α^n * 4 * eps-closest=α^n * eps-closest by force
      finally show ?thesis by blast
    qed
    then have sum-list (map ?component [0..<Inat])≤ sum-list (map (λi. α^n *
eps-closest)[0..<Inat])
       using sum-list-mono[of [0..<Inat] ?component (λi. α^n * eps-closest)] by
fastforce
     then have sum-list (map ?component [0..<Inat])≤ (rat-of-int Inat)*α^n *
eps-closest
      using sum-list-triv[of α^n * eps-closest [0..<Inat]] by auto
   then have (sum-list (map ?component [0..<Inat])) + sum-list (map ?component
[Inat..<n])
                ≤ (rat-of-int Inat)*α^n * eps-closest+(rat-of-int (n−Inat))*α^n *
eps-closest
      using right-sum-alpha by linarith
   then have (sum-list (map ?component [0..<Inat])) + sum-list (map ?component
[Inat..<n])
               ≤ ((rat-of-int Inat)+(rat-of-int (n−Inat)))*α^n * eps-closest
      using gs.cring-simprules(13) by auto
    then show ?thesis
    by (metis (no-types, lifting) Inat-small add-diff-inverse-nat diff-is-0-eq' less-nat-zero-code

          of-int-of-nat-eq of-nat-add zero-less-diff)
  qed
  then have sq-norm ?s ≤ (rat-of-int n)*α^n * eps-closest
    using split-norm-sum by argo
 then have real-of-rat (sq-norm ?s) ≤ real-of-rat ((rat-of-int n)*α^n * eps-closest)
    by (simp add: of-rat-less-eq)
  also have real-of-rat ((rat-of-int n)*α^n * eps-closest)≤2^n*closest-distance-sq
    using present-bound-nicely[of n]
    by blast
  finally show ?thesis
    using coset-s[of n]
    by fast
qed


end
end
```

References

[1] R. Bottesch, J. Divasón, and R. Thiemann. Two algorithms based on modular arithmetic: lattice basis reduction and Hermite normal form computation. *Archive of Formal Proofs*, March 2021. https://isa-afp.org/entries/Modular_arithmetic_LLL_and_HNF_algorithms.html, Formal proof development.

[2] J. Divasón, S. J. C. Joosten, R. Thiemann, and A. Yamada. A Verified Factorization Algorithm for Integer Polynomials with Polynomial Complexity. *Archive of Formal Proofs*, February 2018. https://isa-afp.org/entries/LLL_Factorization.html, Formal proof development.

[3] K. Kreuzer. Hardness of Lattice Problems. *Archive of Formal Proofs*, February 2023. https://isa-afp.org/entries/CVP_Hardness.html, Formal proof development.

[4] N. Stephens Davidowitz. Lecture 5: CVP and Babais Algorithm, August 2016.