```
> restart:with(linalg):with(plots):

Warning, new definition for norm

Warning, new definition for trace
```

1 Võredega seotud põhimõisted

Võre definitsioon

Võreks L(b_1, b_2,.., b_n) nimeteatakse, m-komponendiliste reaalarvuliste lineaarselt sõltumatute vektorite b_1, b_2,.., b_n

kõikvõimalike täisarvulisite lineaarkombinatsioonide hulka.

Edaspidi vaatame ainult täisarvuliste komponentidega võresid. On lihtne aru saada, et nii oleme vaadanud ka ratsionaalarvuliste komponentidega võred. Me võime viia kõik vektorite komponendid ühisele nimetajale N ja selle nö.

kõikide vektorite ette tuua st. veidi formaalsemalt rääkides korraldame isomorfismi F hulgast L(b_1, b_2,.., b_n) täisarvuliste komponentidega L'=N*L(b_1, b_2,.., b_n), kus F(b_i)= $N\,b_i$. **Kuna peamiselt huvitavad meid struktuuri lineaaromadused ja meetrika**, mille F samuti säilitab, siis pole mõtet ratsionaalseid võresid vaadata. Irratsionaalsete võrede korral pole asjad esiteks nii lihtsad ning teiseks ei anna irratsionaalsuste lubamine meile midagi juurde, sest võre on algebralises mõistes projektiivne moodul üle Z (Mati Kilp, "Algebra II", lk 50-51) ja seega esitatav täisarvuliste vektorite kaudu ning meetrika on võimalik vastava normi defineerimisega üle kanda.

Võre baasiks nimetatakse lineaarselt sõltumatute vektorite hulka võres, mille täisarvuliste lineaarkombinatsioonidena avaldub iga võre vektor.

Võre baasiga seotakse **baasimaatriks** B, mille veergudeks on baasi vektorid (m*n maatriks). Ühelt baasilt teisele üleminekumaatriksitel T on järgmine oluline omadus det T=+-1. See tuleneb sellest, et

T* $T^{(-1)}$=E. Samuti ilmneb, et sellise omadusega maatriksid moodustavad rühma.

1.1 Näide

Vaatame võret Z^2, mille üheks baasiks on $b_1 = (1, 0)$ ja $b_2 = (0, 1)$ ning teiseks baasiks on $c_1 = (1, 0)$ ja $c_2 = (1, 1)$, kusjuures

$b_1 = c_1$,

$b_2 = c_2 - c_1$.

Siis üleminekumaatriks T baasilt b_1, b_2 baasile c_1, c_2 ning vastavad baasimaatriksid on

```
> T:=matrix(2,2,[1,1,0,1]); B:=matrix(2,2,[1,0,0,1]); C:=evalm(B&*T);C=B*T;
```

$$T := \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

$$B := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$C := \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

$$C = B\,T$$

Nüüd baasiteisenduse T pöördmaatiks on

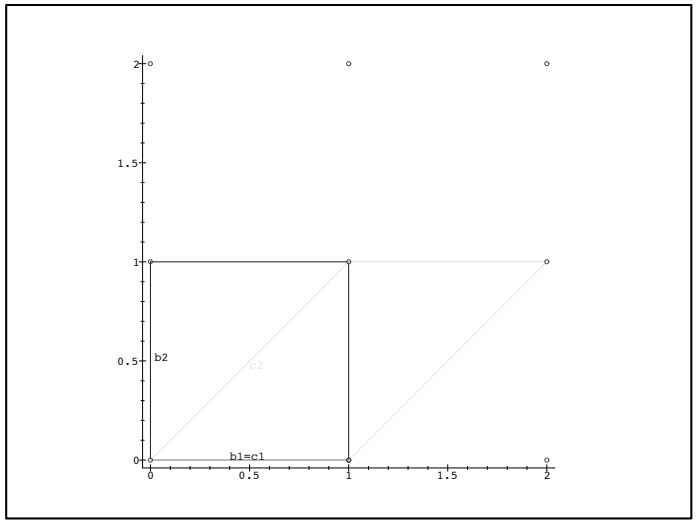
```
> T1:=inverse(T);B=C*T1;B=evalm(C&*T1);
```

$$T1 := \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$

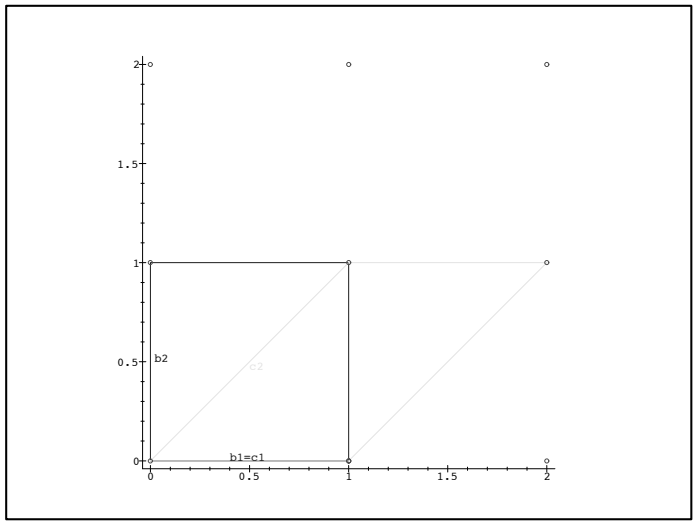
$$B = C\ T1$$

$$B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Seega on baasiteisendused praktiliselt samasugused nagu vektorruumide korralgi, erinevuseks on vaid maatriksi elementide täisarvulisus ja det T=+-1. Graafiliselt näeb teisendus välja järgnevalt



2 Võre determinant



Nagu jooniselt näha võib, katavad mõlemad baasivektoritele ehitatud rööpkülikud ühesuuruse pinna. See ei ole juhuslik. Üldistame antud fakti kõikidele võredele.

Definitsioon

Võre L(B) **determinandiks** loomuliku skalaarkorrutise (. , .) suhtes nimetatakse suurust

det L = $\sqrt{\det(B^T B)}$ st. ruutjuurt võre baasi Grami maatriksi determinandist.

Teoreem

Võre determinant ei sõltu baasi valikust.

Tuleneb sellest, et det T=+-1

Kui võre mõõde n=m, siis räägitakse **täismõõtmelise**t võrest ja siis on võre determinant baasivektoritest moodustatud

rööptahuka ruumala. Ülejäänud juhtudel tuleneb baasivektorite lineaarsest sõltumatusest, et $n < m$ ning edaspidi nimetame seda tüüpi võresid **alamõõtmelisteks** võredeks.

3 Võre diskreetsus

Definitsioon

Olgu meil vektorruum R^m varustatud suvalise normiga ——.——, siis hulga X kuhjumispunktiks **x** nimetatakse punkti, mille igas ümbruses leidub hulga X punkte st. iga $0 < \varepsilon$ korral leidub element **y** hulgast X, nii et $||x - y|| < \varepsilon$.

Märkus

Kuna lõplikumõõtmelises vektorruumis R^m on kõik normid ekvivalentsed, siis järgmise teoreemi seisukohalt pole oluline, mis normi suhtes on teoreem sõnastatud.

Teoreem 1

Olgu G vektorruumi R^m kui aditiivse rühma alamrühm, siis järgmised väited on samaväärsed:

1) alamrühm G on diskreetne hulk st. hulgal G pole ühtegi kuhjumispunkti;

2) punk 0 ei ole G kuhjumispunkt;

3) hulk {x— x on G element ja ——x——<r} on lõplik iga reaalarvulise r>0;

4) iga x korral hulgast G on suurus inf{——x-y——: $x \neq y$, x,y on G elemendid}>0;

5) iga x korral hulgast G leidub δ>0 nii, et kui ——x-y——< δ, siis x=y.

Teoreem 2

Vektoruumi R^m alamhulk L on võre parajasti siis, kui L on R^m diskreetne alamrühm liitmise suhtes.

Võre diskreetsus on peamine meetrikaline omadus, mis lubab kasutada teoreemi 1 väiteid 3) ja 5), mida kasutatakse nii lühima vektori leidumise kui ka teatud algoritmide korrektsuse näitamiseks.

4 Hermite'i normaalkuju(HNF)

Ilmneb, et võre baasil on olemas eriline kuju, mida nimetatakse **Hermite'i normaalkujuks.**

4.1 Täismõõtmelise võre HNF

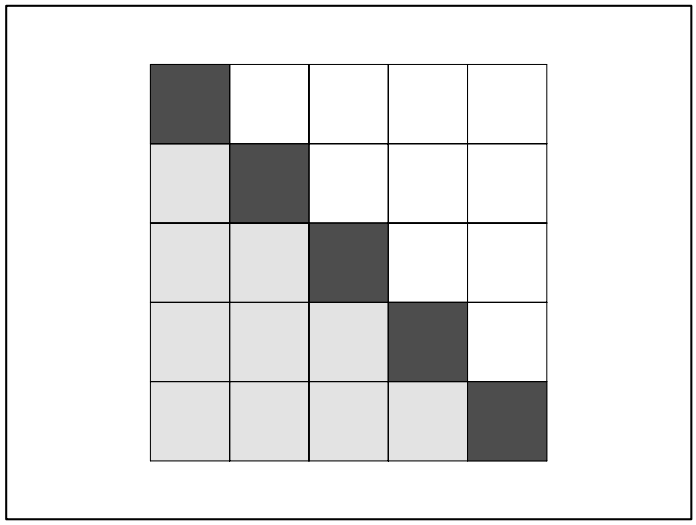
Definitsioon

Me ütleme, et võre baas B on Hermite'i normaalkujul, kui on täidetud 2 tingimust:

1) B on alumine kolmnurkmaatriks;

2) iga 0<j<i kehtib $0 \leq b_{i,j}$ ja $b_{i,j} < b_{i,i}$.

Maatriksi peadiogonaali elemendid domineerivad samas reas olevaid vasakpoolseid elemente



Idee 1

Baasimaatriksiga võib teha järgmisi elementaarteisendusi:

1) liita ühele veerule täisarv korda teine veerg;

2) korrutada veergu läbi -1;

3)vahetada kahe veeru asukohad;

ilma, et võre muutuks

Idee 2

Need kolm elementaaroperatsiooni lubavad teha nö. Eukleidese algoritmi veergude peal fikseerides mingi rea.

Nüüd ja edaspidi tähistavad rasvaselt trükitud tähed maatriksi veeruvektoreid

ColumnGCD(b1,b2,row)

if $b2_{row} = 0$ then **b1:=:b2**

while $b2_{row}$ <>0do

c:=floor($\frac{b1_{row}}{b2_{row}}$);

b1:=b1-c*b2 //siin kohas on selge $0 \leq b1_{row}$;

b1:=:b2

od;

return **b1,b2**

Et tegu on Eukleidese algoritmiga, on tulemuseks veerud **b1'** ja **b2'**, kus $b1_{row}$=GCD($b1_{row}$, $b2_{row}$) ja $b2_{row} = 0$

Algoritmi keerukushinnang on O($n\,m$) vektoroperatsiooni, kus m,n on $b1_{row}$ ja $b2_{row}$ pikkused bittides

Idee 3

Idee 2 annab võimaluse reas olevate elementide nullimiseks, seega HNF 1 tingimus pole probleem

Teise tingimuse saame täita, siis kui jagame eelnevaid rea elemente $b1_{i+1}$, $b2_{i+1}$,.. bi_{i+1} läbi elemendiga
`bi+1[i+1]`ja jätame alles jäägi.

HNF Algoritm

row=1;
col=1;
while(col<=n) do
for i=col,..,n do if bi[row]<0 then bi:=-bi od; //kõik elemendid reas row on positiivsed
for i=col+1,..,n do [bcol,bi]:=ColumnGCD(bcol,bi,row) od;// kõik elemendid reas row on nullid kui i>col
//kuna rank B=n, siis peab $bcol_{row} \neq 0$
for i=1,..,col-1 do
c=floor(bi[row]/bcol[row]);
bi:=bi-c*bcol; // Et sellel hetkel on maatriks kuni veeruni col alumisel kolmnurksel kujul, siis see op. ei muuda //eelmistel ridadel olevat ja selle elemendi bi[row] jaoks on HNF 2. tingimus täidetud.
od;
row++;col++;
od;

Keerukushinnang.

Esimene tsükkel nõuab n(n-1)/2 vektori läbi korrutamist, teine tsükkel nõuab n(n-1)/2 ColumGCD tegemist ja viimane tsükkel nõuab n(n-1)/2 jagamist ja vektoriga korrutamist ning liitmist. Seega, kui meil oleks vektori komponentide väärtused tõkestatud, siis oleks selgelt tegu polünomiaalse ülesandega.

4.2 Praktiline algoritm

```
> ColumnGCD:=proc(B::matrix,col,k,row,n)
> local abix,i,c;
> #veegude vahetamine
> if B[row,k]=0 then B:=swapcol(B,k,col) fi:
> while B[row,k]<>0 do
> #Maple käsk iquo ei sobi, sest negatiivsete arvude
> #annab vale tulemuse
> c:=floor(B[row,col]/B[row,k]);
> #liidame veerule col -c korda veeru k
> B:=addcol(B,k,col,-c);
> B:=swapcol(B,k,col):
> od:
> #RETURN(B):
> end:
> HNF1:=proc(B::matrix)
> local row,k,i,c,n;
> if coldim(B)<>rowdim(B) then ERROR(‘Pole ruutmaatriks‘): fi:
```

```
> if det(B)=0 then ERROR(‘Pole täismõõtmeline võre‘): fi:
> n:=coldim(B):
> for row from 1 to n do
> for k from row to n do
> #korrutame vajaduse korral veeru -1 läbi
> if B[row,k]<0 then B:=mulcol(B,k,-1): fi:
> od:
> for k from row+1 to n do ColumnGCD(B,row,k,row,n): od:
> for k from 1 to row-1 do
> c:=floor(B[row,k]/B[row,row]):
> B:=addcol(B,row,k,-c):
> od:
> od:
> print(B);
> end:
```

4.3 Näide maatriksi elementide kiirest kasvamisest

Eelpool näidatud algoritm töötab polünomiaalses ajas vaid siis, kui maatriksi elementide absoluutväärtused on kogu algoritmi käigus tõkestatud. Siin on näide maatriksistest, mille korral alustame üsna lihtsate elementidega maatriksiga ja lõpetame sammuti üsna lihtsate elementidega maatriksiga, aga algoritmi käigus paisuvad elemendid üsna suureks. Need maatrisid on saadud täiesti juhuslikult kordajaid valides mitte spetsiaalselt elemente konstrueerides, seega on elementide paisumine üsna tõsine probleem.

$$A1 := \begin{bmatrix} -4 & -2 & 3 & -4 & 3 & 0 & 5 & -4 & 3 & -3 \\ -4 & -1 & -4 & 2 & 1 & -2 & 2 & -5 & 0 & -5 \\ 2 & 1 & -1 & 4 & -2 & -5 & -3 & 2 & 2 & -5 \\ 2 & 5 & 4 & -2 & -1 & 4 & 0 & 2 & 2 & 0 \\ -1 & 5 & -3 & 3 & -3 & 2 & 1 & 3 & 0 & 0 \\ 4 & -2 & -3 & -5 & 3 & 2 & 5 & 0 & 4 & -4 \\ 3 & -1 & 3 & 2 & 0 & 2 & 3 & -1 & -1 & 5 \\ 0 & 3 & -5 & -4 & -1 & -5 & -4 & 5 & -1 & 4 \\ 3 & 2 & 4 & 5 & -4 & 0 & -5 & 2 & -2 & 5 \\ -1 & 4 & -5 & 4 & -5 & -2 & -5 & -2 & -1 & 1 \end{bmatrix}$$

$$A2 := \begin{bmatrix} 5 & -5 & -2 & -1 & 4 & 1 & -5 & -4 & -1 & -4 \\ 5 & -5 & 0 & -5 & 2 & 2 & -1 & 0 & 1 & -2 \\ 3 & 1 & -5 & -5 & 2 & 0 & 2 & -2 & -3 & 5 \\ -1 & 5 & 4 & -1 & 5 & 5 & -5 & -4 & -1 & -5 \\ -4 & -4 & 2 & 5 & 0 & -5 & -5 & 5 & -2 & 5 \\ -1 & -5 & -1 & -1 & -4 & -3 & 4 & -5 & -4 & -2 \\ -3 & 4 & 0 & -1 & -3 & -4 & 5 & -2 & 4 & -1 \\ -1 & 4 & 2 & -5 & -4 & 2 & 4 & 1 & -5 & 5 \\ 3 & 1 & 3 & -2 & 2 & -5 & 0 & -5 & -2 & -5 \\ -2 & 2 & 4 & -1 & 2 & -2 & -4 & -2 & 2 & -3 \end{bmatrix}$$

$$A3 := \begin{bmatrix} 4 & 0 & -1 & 1 & -2 & 2 & 4 & 4 & 4 & 4 \\ 1 & 4 & -5 & -5 & -3 & -5 & -2 & -1 & -5 & -4 \\ 0 & -2 & 0 & 2 & -4 & -3 & 0 & -1 & 0 & 3 \\ -2 & -4 & 5 & 0 & 3 & -5 & -3 & -5 & 1 & -1 \\ -3 & -2 & 3 & 5 & -3 & 2 & 2 & 0 & 0 & 5 \\ -2 & 2 & -3 & 5 & 3 & -3 & 0 & -2 & -4 & 1 \\ -4 & -4 & 4 & -4 & 3 & -5 & 2 & -4 & 3 & 5 \\ 5 & -4 & -4 & -2 & 3 & -2 & -2 & 2 & 1 & -1 \\ 5 & 2 & -1 & -3 & -4 & -2 & 5 & 5 & -1 & 5 \\ 1 & -3 & 4 & -3 & -4 & 2 & -1 & 1 & 1 & -5 \end{bmatrix}$$

$$A4 := \begin{bmatrix} 3 & -4 & -4 & 1 & -1 & 2 & 3 & -1 & 5 & 5 \\ 3 & -3 & 3 & 5 & -5 & 4 & 2 & 1 & 4 & 3 \\ 4 & 4 & 3 & -3 & -5 & 2 & -4 & 1 & 0 & 0 \\ -5 & -1 & -3 & -2 & 3 & 3 & -4 & 0 & 3 & -2 \\ -1 & -2 & 5 & 3 & -4 & -5 & -5 & -2 & -3 & 2 \\ 2 & -3 & -5 & -4 & 5 & -2 & -4 & -3 & 5 & 4 \\ 5 & -2 & -3 & -3 & -3 & 3 & 3 & 2 & 5 & -3 \\ -5 & 2 & -5 & -2 & 3 & 0 & -1 & -1 & -4 & -5 \\ 2 & 1 & 4 & -2 & 4 & -3 & 2 & -3 & -3 & 2 \\ 0 & 0 & -1 & 3 & 3 & -5 & -5 & 5 & 3 & -3 \end{bmatrix}$$

Need maatriksid on üsna väikeste komponentidega. Kasutades nüüd HNF1 algoritmi, mis väljastab ka vahetulemused

on näha maatriksielementide paisumine

```
> illustrative_HNF1:=proc(B::matrix)
> local row,k,i,c,n;
> if coldim(B)<>rowdim(B) then ERROR(‘Pole ruutmaatriks‘): fi:
> if det(B)=0 then ERROR(‘Pole täismõõtmeline võre‘): fi:
> n:=coldim(B):
> for row from 1 to n do
> for k from row to n do
> #korrutame vajaduse korral veeru -1 läbi
> if B[row,k]<0 then B:=mulcol(B,k,-1): fi:
> od:
> for k from row+1 to n do ColumnGCD(B,row,k,row,n): od:
> print(B);
> for k from 1 to row-1 do
> c:=floor(B[row,k]/B[row,row]):
> B:=addcol(B,row,k,-c):
> od:
> print(B):
> od:
> #print(B);
> end:
> #illustrative_HNF1(B1);illustrative_HNF1(B2); illustrative_HNF1(B3);illustrative_HNF1
```

4.4 Täismõõtmelise võre HNF saamine polünomiaalses ajas

Et täismõõtmelise võre HNF leidmise algoritm oleks ajas polünomiaalne, on tarvis kuidagi piirata maatriksi elementide

suurust. Seda võimaldab meile järgmine teoreem

Teoreem

Iga täismõõtmeline L võre sisaldab endas alamvõrena võret det L Z^n

Tõestus tuleb välja LVS Crameri lahenditest.

Idee 4

HNF algoritm töötab õieti ka juhul, kui me lisame baasile veel mõned veerud. Ainult siis saame algoritmi väljundisse nende lisatud veergude asemele nullveerud. Lisades nüüd veerud (det L,0,...),(0,det L,..),...,(..,0,det L), oleme, saanud teoreetilise põhjenduse, miks võib igal sammul teha teisendusi mod det L, sest tagumist maatriksi osa võib kasutada modulaarseks taandamiseks. Peadiagonaalile peab kirjutama süt(detL,B[i,i]).

Seega on võimalik leida HNF polünomiaalses ajas.

4.5 Täiendatud HNF algoritm

```
> mod_ColumnGCD:=proc(B::matrix,col,k,row,modul us,n)
> local abix,i,c;
> if B[row,k]=0 then B:=swapcol(B,k,col): fi:
> while B[row,k]<>0 do
> c:=floor(B[row,col]/B[row,k]);
> for i from 1 to n do
> B[i,col]:=irem(B[i,col]-c*B[i,k], modulus);
> od;
> B:=swapcol(B,k,col)
> od:
> #print(B);
> end:
> mod_HNF1:=proc(B::matrix)
> local row,k,i,c,detB,n;
> #täismõõtmelise võre determinant on võrdne baasimaatriksi determinandi
> #absoluutväärtusega
> if coldim(B)<>rowdim(B) then ERROR(‘Pole ruutmaatriks‘): fi:
> n:=coldim(B):
> detB:=abs(det(B)):
> if detB=0 then ERROR(‘Pole täismõõtmeline võre‘): fi:
> #maatriksi elementide normaliseerimine mod detB
> for i from 1 to n do
> for k from 1 to n do
> B[i,k]:=irem(B[i,k],detB):
> od:
> od:
> for row from 1 to n do
> for k from row to n do
> if B[row,k]<0 then B:=mulcol(B,k,-1): fi:
> od:
> for k from row+1 to n do mod_ColumnGCD(B,row,k,row,detB,n): od:
> #element peadiagonaalil tuleb asendada gcd(B[row,row],detB)
> B[row,row]:=gcd(B[row,row],detB):
> for k from 1 to row-1 do
> c:=floor(B[row,k]/B[row,row]):
> for i from 1 to n do
> B[i,k]:=irem(B[i,k]-c*B[i,row],detB):
> od:
> od:
> print(B)
> od:
> print(B);
> end:
```

Võrdle nüüd hariliku HNF algoritmiga

```
> #mod_HNF1(A1);mod_HNF1(A2);mod_HNF1(A3);mod_H NF1(A4);
```

4.6 Alamõõtmelise võre HNF

Definitsioon

Alamõõtmeliseks võreks vektoruumis R^m loetakse võret, mille baasis on vähem vektoreid kui m st. kui B on m x n baasimaatriks, siis $n < m$.

Alamõõtmeliste võrede korral eksisteerib kaks HNF definitsiooni.

Definitsioon 1

Me ütleme, et võre baas B on **Hermite'i normaalkujul**, kui on täidetud järgmised tingimused

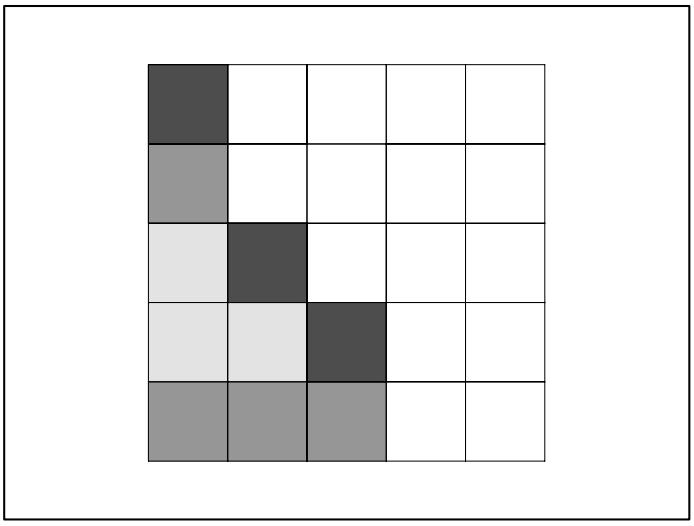
1) leidub selline rangelt kasvav jada i_1, i_2,.. i_n nii, et kui j> i_i siis $B_{i,j} = 0$ st. et maatriks on kolmnurkses kujus;

2) iga $k < j$ korral $0 \leq B_{i_j,k}$ ja $B_{i_j,k} < B_{i_j,j}$ st. kõige ülemine nullist erinev element veerus on

suurim oma reas.

Näiteks võib võre parameetritega m=5 ja n=3 omada järgmist HNF-i, kus punased elemendid on suuremad kui samas

reas olevad kollased elemendid; kollased elemendid on mittenegatiivsed ning roheliste elementide kohta pole mingisugust infot. Viimaseid kahte veegu ei ole tegelikult baasimaatriksis, sest see on 5*3 maatriks, need on lisatud selleks, et demostreerida, mida tähendab võre alamõõtmelisus.



Definitsioon 2

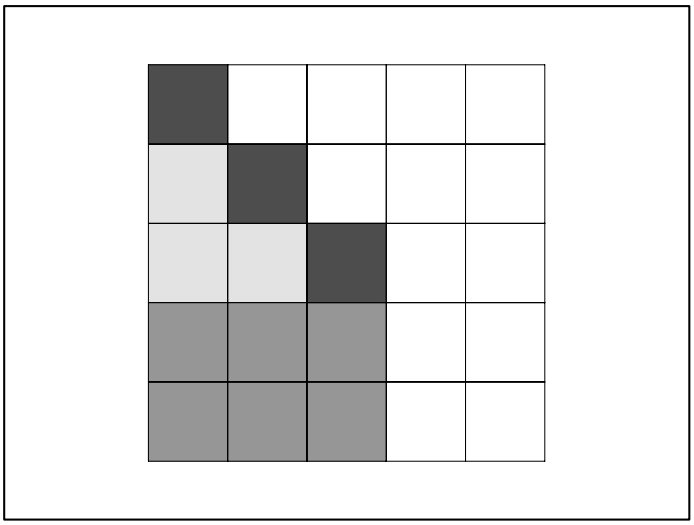
Me ütleme, et võre baas B on **Hermite'i normaalkujul**, kui on täidetud järgmised tingimused

1) kui $i < j$, siis $B_{i,j} = 0$ st. et maatriks on kolmnurkses kujus;

2) iga $k < i$ korral $0 \leq B_{i,k}$ ja $B_{i,k} < B_{i,i}$ st. kõige ülemine nullist erinev element veerus on

suurim oma reas.

Näiteks võib võre parameetritega m=5 ja n=3 omada järgmist HNF-i, kus värvide tähendused on samad, mis ennemgi.



Definitsioon 2 on definitsioon 1 erijuht ja seda kasutatakse tihti teoreetilistes arutlustes. Definitsioonile 1 vastavat võret

pole reeglina võimalik viia kujule 2. Kui vaadata näitemaatriksit 1, siis on selge, et mitte ühegi lubatava veeruteisendusega pole võimalik tekitada teise veergu teise ritta punane element. Ainus võimalus sinna üldse mingit nullist erinevat elementi saada on liita teisele veerule esimene veerg, aga siis kaob kolmnurkkuju. Kui me lubame esialgse vektorruumi baasis vektoreid ümber järjestada st. meie maatriksi ridu ümber tõsta, saame lihtsalt kujust 1 kuju 2.

Täiendame esialgset HNF arvutamise algoritmi nii, et seda saaks kasutada ka alamõõtmelise võre viimiseks

definitsiooni 1 järgsesse HNF-i.

Tähelepanek

Kui me rakendame ColumGCD, siis võib juhtuda, et ka alumine rida saab täielikult nullitud. Siis tuleb minna edasi ja vaadata seda rida, kus on tagapool mõni nullist erinev element ja teha sellega täpselt nagu esialgses algoritmis.

HNF Algoritm alamõõdulise võre korral

row=1;

col=1;

while(col<=n and row<=m) do

for i=col,..,n do if bi[row]<0 then bi:=-bi od; //Kõik elemendid reas row on positiivsed

for i=col,..,n do [bcol,bi]:=ColumnGCD(bcol,bi,row) od;// Kõik elemendid reas row on nullid kui i>col

if B[row,col]=0 then row++

else

for i=1,..col-1 do

c=floor(bi[row]/bcol[row]);

bi:=bi-c*bcol; // Et sellel hetkel on maatriks kuni veeruni col alumisel kolmnurksel kujul, siis see operatsioon ei //muuda eelmistel ridadel olevat ja selle elemendi bi[row] on HNF teine tingimus täidetud.

row++;col++;

od;

od;

Ka selle algoritmi keerukus võib tulla seoses vektori komponentide kasvamisega mittepolünomiaalne.

4.7 Praktiline algoritm

```
> HNF2:=proc(B::matrix)
> local row,col,k,i,c,m,n;
> m:=rowdim(B):
> n:=coldim(B):
> row:=1:col:=1:
> while row<=m and col<=n do
> for k from col to n do
> if B[row,k]<0 then B:=mulcol(B,k,-1):  fi:
> od:
> for k from col+1 to n do ColumnGCD(B,col,k,row,m):  od:
> if B[row,col]=0 then row:=row+1:
> else
> for k from 1 to row-1 do
> c:=floor(B[row,k]/B[row,col]):
> B:=addcol(B,row,k,-c):
> od:
> row:=row+1:col:=col+1:
> fi:
> print(B);
> od:
> print(B);
> end:
> A:=matrix(3,2,[5,4,3,2,1,0]):HNF2(A,3,2);
```

$$\begin{bmatrix} 1 & 0 \\ 1 & -2 \\ 1 & -4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 1 & 2 \\ 1 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 1 & 2 \\ 1 & 4 \end{bmatrix}$$

Kontrollime võre determinandi säilimist

```
> A:=matrix(3,2,[5,4,3,2,1,0]);det(transpose(A)&*A);
> A:=matrix([[1, 0],[1,2], [1,4]]):det(transpose(A)&*A);
```

$$A := \begin{bmatrix} 5 & 4 \\ 3 & 2 \\ 1 & 0 \end{bmatrix}$$

$$24$$

$$24$$

4.8 Alamõõtmelise võre HNF saamine polünomiaalses ajas

Idee seisneb alamõõtmelise võre HNF leidmise taandamises täismõõtmelise võre HNF leidmisele Olgu maatriksi B astak k, siis leiduvad read r_1, r_2,..., r_k, mis on lineaarselt sõltumatud. Need read saab välja eraldada näiteks Gramm-Schmidti ortogonaliseerimisprotsessi abil st. kui Gramm-Schmidti algoritm teisendab rea nulliks, siis ei kuulu rea indeks hulka { r_1, r_2,..., r_k}.**Oluline on veel see, et kui rida pole hulgas** { r_1, r_2,..., r_k}**, siis avaldub ta eelnevate lineaarkombinatsioonina.(*)** Nüüd defineerime projektori π : R^m -> R^n nii, et igale vektorile x seab projektor vastavusse vektori x komponentidest { r_1, r_2,..., r_k}moodustatud vektori π(x)=(x_{r_1}, x_{r_2},..., x_{r_k}). Antud projeksioon on üksühene võrel L(B) ja pöördkujutis on arvutav on polünomiaalses ajas. Nüüd rakendades B_1= $\pi(B)$ polünomiaalset HNF täismõõtmelise võre jaoks, saame HNF-il maatriksi

B_2. Rakendades pöördteisendust $\pi^{(-1)}$(B_2), saame maatriksi B_3, mis ilmselt on võre L baasimaatriks. Väide (*) kindlustab selle, et

B_3 on definitsiooni 1 mõttes HNF.

4.9 Praktiline algoritm

Realiseerime esmalt Gram-Schmidt'i ortogonaliseerimisprotsessi(Maple's on küll olemas vastav käsk GramSchmidt, kuid see ei anna meile vajaminevat üleminekumaatriksit) st. arvutame vektorid c_i järgmise reegli alusel

$c_1 = b_1$

$c_i = b_i - \left(\sum_{j=1}^{i-1} \mu_{i,j}\, c_j\right)$,

kus $c_i \neq 0$ parajasti siis, kui b_i ei ole avaldatav eelnevate vektorite lineaarkombinatsioonina. Realiseerime Gramm-Schmidti algoritmi nii, et igal sammul oleks avaldatud vektorid c_i esialgsete baasivektorite kaudu. Siis on lihtne hiljem leida pöördkujutust $\pi^{(-1)}$

```
> GS_orto:=proc(B::matrix)
> local i,j,k,T,C,M,m,n;
> #maatriks C sisaldab lõpuks B ridade ortogonaliseeritud baasi
> #maatriks T sisaldab ülemineku kordajaid st.  C=T&*B
> #vektor M sisaldab ortogonaliseerimissammu kordajaid
> #väljundiks on maatriksid C ja T
> m:=rowdim(B):
> n:=coldim(B):
> C:=matrix(m,n,[]);
> #Maple kipub otsese võrduse C:=B korral muutma mõlema maatriksi
sisu
> for i from 1 to m do for j from 1 to n do C[i,j]:=B[i,j] od:od:
> T:=matrix(m,m,[]);
> M:=vector(m,[seq(0,i=1..m)]):
> for i from 1 to m do
> for j from 1 to i-1 do
> #arvutame ort.  kordajad
> k:='k':
> M[j]:=sum(B[i,k]*C[j,k],k=1..n):
> k:='k':
> #kui ortogonaliseeritud j-reavektor pole null, siis jagame
> #kordaja läbi vektori normi ruuduga
> if M[j]<>0 then M[j]:=M[j]/sum(C[j,k]^2,k=1..n):  fi:
> od:
> #arvutame rea C[i]
> for k from 1 to n do
> j:='j':
> C[i,k]:=C[i,k]-sum(M[j]*C[j,k],j=1..i-1):
> od;
> #arvutame maatriksi T i-nda rea
> for k from 1 to i-1 do
> j:='j':
> T[i,k]:=-sum(M[j]*T[j,k],j=1..i-1):
> od:
> T[i,i]:=1:
> for k from i+1 to m do T[i,k]:=0:  od:
> #asume järgmise sammu kallale
> od:
> RETURN([C,T]);
> end:
```

Nüüd leiame vajaliku teisendusmaatriksi, mis realiseerib pöördteisenduse $\pi^{(-1)}$. Kui rida $c_i = 0$, siis $0 = \left(\sum_{j=1}^{i-1} T_{i,j}\, b_j\right) + b_i$ ning seega $b_i := -\left(\sum_{j=1}^{i-1} T_{i,j}\, b_j\right)$(**). Olgu projekteeritavad ridade indeksid vektoris(hulgas) K, siis kui $c_i \neq 0$ peab i kuuluma hulka K ning seega $b_i = 1\, b_i$(***). On lihtne tõestada, et $T_{i,k} = 0$, kui k ei kuulu hulka K, seega võib avaldises (**) summerida vaid üle hulka K kuuluvate indeksite. Nüüd järgnev protseduur eraldabki indeksite hulga K ja kasutades seoseid (**) ja (***) konstrueerib pöördteisenduse maatriksi P.

```
> inverse_Pi:=proc(C,T,m)
> local P,i,j,K:
> P:=matrix(m,rank(C),0):
> K:=[]:
> for i from 1 to m do
> if norm(row(C,i))=0 then
> for j from 1 to nops(K) do P[i,j]:=-T[i,K[j]]:  od:
> else
> #lisame K-sse uue indeksi
> K:=[op(K),i]:
> P[i,nops(K)]:=1:
> fi:
> od:
> #tagastame K, Pi pöördkujutuse P
> RETURN([K,P]):
> end:
```

Nüüd tuleb realiseerida vaid kujutus π

```
> project:=proc(B,K,n)
> local L,i;
> L:=[seq(i,i=1..n)]:
> RETURN(submatrix(B,K,L)):
> end:
```

Ning algoritm ise kujul

```
> mod_HNF2:=proc(B::matrix)
> local L,P,B1,B2,m,n;
> m:=rowdim(B):
> n:=coldim(B):
> #ortogonaliseerime, et saaksime lihtsalt arvutada pöördteisenduse
> L:=GS_orto(B);
> L:=inverse_Pi(L[1],L[2],m):
> P:=L[2]:
> #muudame projektsiooniga võre täismõõtmeliseks
> B1:=project(B,L[1],n);
> #leiame võre HNF-i modulaarse algoritmiga
> mod_HNF1(B1);
> #tuleme pöördteisendusega tagasi
> RETURN(evalm(P&*B1));
```

```
> end:
```

5 Hermite'i normaalkuju omadused ja rakendused

Teoreem

Igal võrel on vaid üks HNF.

Seega on võimalik kahe võre võrdsust näidata nii, et leiame mõlema võre HNF

Lihtne on näidata vektori kuulumist võresse või mitte.

Algoritmi sisend vektor **x** ja baas B=(**b1**,**b2**,..**bn**)=B[i,j] HNF-s

Väljund kui x on võres, siis x kordinaadid baasil c[1],c[2],..c[n]

vastasel korral ERROR

row=1;col=1;

while row<=m and col<=n do

//Invariant on leitud c[1],c[2],..,c[col-1] ja x peab kuuluma võresse L(**bcol**,..,**bn**)

while B[row,col]=0 and x[row]=0 do row++;od;

//et alles jäänud vektor oleks võres peavad vastavad komponendid olema nullid

kui B[row,col]=0 siis x pole võres return ERROR;

c[col]:=x[row]/B[row,col];

kui c ei ole täisarv, siis x pole võres return ERROR;

x=**x**-c***bcol**; //lahutame vastava vektori

col++;row++;

od;

if x<>0 return ERROR;

//Seega x on võres

return c[1],c[2],c[3],..c[n];

Praktiline algoritm

```
> is_in_lattice:=proc(B::matrix,x0::vector)
> local m,n,col,row,c,x,i;
> m:=rowdim(B):n:=coldim(B);
> if vectdim(x0)<>m then RETURN(‘Vektor pole võres‘); fi:
> c:=[]:#olgu c tyhi list
> x:=vector(m,[seq(x0[i],i=1..m)]):
> row:=1:col:=1:
> while row<=m and col<=n do
> while B[col,row]=0 and x[row]=0 and row<=m do row:=row+1:  od:
> if B[col,row]=0 then RETURN(‘Vektor pole võres‘); fi:
> if irem(x[row],B[row,col])<>0 then RETURN(‘Vektor pole võres‘);
fi:
> c:=[op(c),iquo(x[row],B[row,col])]:#lisame vektorile uue komponendi
> for i from row to m do
> x[i]:=x[i]-c[col]*B[i,col]:
> od:
> col:=col+1:row:=row+1:
```

```
> od:
> if dotprod(x,x)>0 then RETURN(‘Vektor pole võres‘); fi:
> RETURN(c);
> end:
```

Näide

```
> B:=matrix(3,2,[1,0,0,1,0,1]);x:=vector(3,[1,4 ,5]);
```

$$B := \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

$$x := [1,\ 4,\ 5]$$

```
> is_in_lattice(B,x);
```

Vektor pole vres

Baasi üleskirjutus on mahult hästi väike ja sobib seega võre säilitamiseks. Õieti on võrel olemas veel väiksema mahuga esitus,

milleks on kõige lühematest vektoritest koosnev Mikowski baas. Esiteks pole Mikowski baasi võimalik polünomiaalses ajas leida. Teiseks oleks sellise baasi esimene vektor lühim võres. Harilikult tahetakse lühimat vektorit saladuses hoida, seega nende kahe põhjuse pärast Mikowski baasi praktikas ei kasutata.

6 Lühima vektori probleemi lahendus kasutades HNF-i

6.1 Lühima vektori probleem(SVP)

Definitsioon

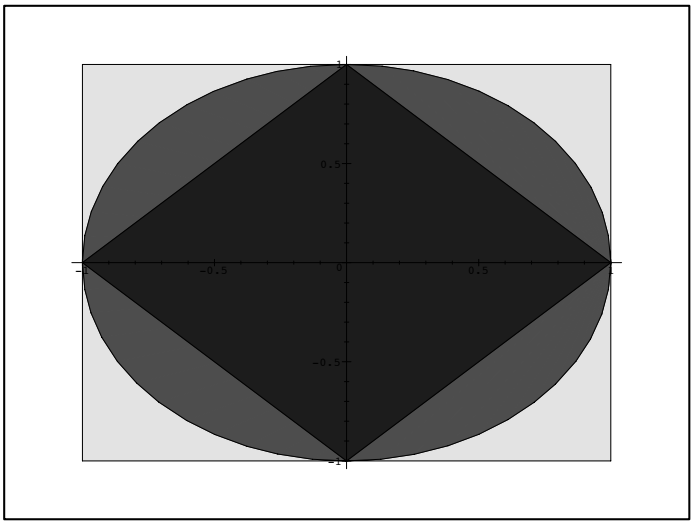
Võre nullist erinevat vektorit, mille pikkus on minimaalne, nimetatakse lühimaks vektoriks.

Võre diskreetsus kindlustab lühima nullist erineva vektori olemasolu võres. Kasutame nüüd võre HNF, et hinnata ja leida lühim vektor. Kuna vektori pikkus sõltub vektorruumis kasutatavast normist, siis võib iga normi korral saada eri vastuse. Peamiselt kasutatakse lõpmatusnormi, Eukleidist normi e. 2-normi ning p-normi. p-norm defineeritakse järgnevalt

$||x||_p = (\sum_{i=1}^{m} x_i{}^p)^{(\frac{1}{p})}$, kus $1 \leq p$.

Neist kõige nõrgem on lõpmatusnorm ning kõige tugevam on 1-norm. Kõik teised p-normid jäävad nende kahe normi vahele.

Seda illustreerib järgmine joonis ühikkeradest, kus kollane on lõpmatusnormi ühikkera, punane on 2-normi ühikkera ning sinine on 1-normi ühikkera. Kui lõpmatusnorm ja p-normid sõltuvad oluliselt baasi valikust, siis 2-norm on invariantne ortonormeeritud baasi suhtes. Selleks veendumiseks võib mõelda, mis juhtub kui kordinaattelgi keerata näiteks 30 kraadi.



Vaatleme siin esmalt lõpmatusnormi norm(b)= max($|b_i|$) ning otsime lühimat vektorit vaadates läbi kõik vektorid, mis on mingis keras B(0,r)={x — norm(x) $\leq$ r}. Võre diskreetsusest on seal loomulikult lõplik arv elemente ning me saame leida vähima nullist erineva pikkusega vektori. Olgu meil baas HNF-s, siis baasivektorite pikkused annavad meile esimese hinnangu $d = \min(\text{norm}(b_i))$ lühima vektori pikkusele. Nüüd vaatame läbi kõik vektorid, mis on keras B(0,d) ja lühima vektori baasis võtame SVP lahendi kandidaadiks **y**. Kui meil oleks suvaline baas, siis oleks kõigi keras B(0,d) olevate vektorite leidmine raske, aga me võime kasutada HNF peamist omadust, et viimased vektorid ei mõjuta eelmise vektorite teatavaid komponente. Alustame esimese baasivektori **b1** lineaarkombinatsioonidega. Meil on lootust mingi kombinatsioon k*b1 suruda kerasse B(0,d) vaid siis, kui vektori **b2** pool mittemõjustatavad komponendid on absoluutväärtuselt väiksemad või võrdsed d-st. Tuues sisse formalisatsiooni:

Def.

HNF-l oleva baasi **b1**,**b2**,...**bn** vektori **bi** sõltumatuteks komponentideks nimetatakse komponente bi_j, mis ei muutu, kui vektorile **bi** liita vektor c***bj** (j>i). Tähistame Ind(**bi**) nende indeksite j hulka ja ind_i(x)=max(— x_j—, kus j kuulub hulka Ind(**bi**)).

ja tehes kolm olulist tähelepanekut:

Idee 5

Et mingit lineaarkombinatsiooni c_1***b1**+ c_2***b2**+...+ c_{k-1}***bk-1** oleks võimalik suruda kerasse B(0,d) peavad **bk-**st sõltumatud komponendid olema absoluutväärtuselt väiksemad või võrdsed kui d ehk

ind_{k-1}(c_1***b1**+ c_2***b2**+...+ c_{k-1}***bk-1**)<=d.

Idee 6

Kui baasivektori **b1** üks sõltumatu komponent on suurem kui d, siis peab selle vektori kordaja lineaarkombinatsioonis

olema 0 ja selle vektori võib baasist välja visata, sest seda ei lähe lühima vektori otsimisel tarvis. Seda ideed saab rekursiivselt rakendada.

Idee 7

Kui me oleme leidnud mõne vektori, mille pikkus d_1 on väiksem kui d, siis võib vaadelda vaid vektoreid keras B(0, d_1), sest meil on seal lühima vektori kandidaat olemas. See tähendab tegelikult selleks ajaks leitud lineaarkombinatsioonide uut ülevaatamist kasutades ideed 5, sest B(0, d_1) sisaldub keras B(0,d).

Idee 8

Me võime vaadelda vektoreid, mille esimene komponent on positiivne, sest normi sümmeetria tõttu on **-u** sama pikk.

See idee vähendab vektorite arvu 2 korda. Sõltumatute komponentide tõttu on on seda ka lihtne kontrollida.

Need ideed kokku annavad algoritmi

SVP-lõpmatus

Sisend baas **b1,b2,...bn**, mis on HNF-l

Väljund SVP-lõpmatus lahend

//idee 6 rekursiivne rakendamine

$d = \min(\text{norm}(b_i))$

y=bi

Jätame vaatluse alt välja baasivektorid b_1, b_2,.. b_i kui $d_1 \leq \text{ind}(b_1)$, ind(b_2),...ind(b_i) ja võtame lühima vektori kandidaadiks s baasivektori, mille norm on d. Olgu tulemuseks ümberjärjestatud vektorite süsteem **b1',b2',..bk'** mis on HNF-s

//olgu U kõigi lineaarkominatsioonide hulk, mis võivad mahtuda kerasse B(0,d) ja koosnevad vektorite **b1,b2,..bi-1** täisarvulistest kombinatsioonidest

U:=[];

for i=1..k do

for **u** on U element do

V:=[];

Vaatleme kõiki täisarvulisi kombinatsioone **u**+c***bi**, mille korral ind_i(**u**+c***bi**)<=d ja $c \neq 0$

//et bi on olemas vähemalt üks sõltumatu komponent, mille väärtus on vähemalt 1, siis neid vektoreid

//saab olla ülimalt 2*d+1

Lisame vektori **u**+c***bi** hulka V

Kui norm(**u**+c***bi**)<=d, siis y=**u**+c***bi** d=norm(**u**+c***bi**) ning viskame hulgast U välja kõik vektorid **w**,

mille korral ind_{i-1}(**w**)>d ning hulgast V välja vektorid **w**, mille korral ind_i (**w**)>d

//selle jaoks ei pea eriti tööd tegema kui vektorid on järjestatud ind_i järgi kasvavalt

od

//siiamaani võib hulgas V olla —U—*(2*d+1) vektorit

Vaatleme kõiki täisarvulisi kombinatsioone c***bi**, mille korral ind_i(**u**+c***bi**)<=d ja $c \neq 0$

Lisame vektori c*$\mathbf{bi}$ hulka V

Kui norm(c*$\mathbf{bi}$)<=d, siis y=c*$\mathbf{bi}$ d=norm(c*$\mathbf{bi}$) ning viskame hulgast V välja kõik vektorid $\mathbf{w}$,

mille korral ind_i ($\mathbf{w}$)>d

//nüüd võis lisanduda veel 2*d+1 vektorit

//seega halvimal juhul on hulgas V rohkem kui $(2\,d+1)^{(i-1)}$ ja vähem $(2\,d+2)^{(i-1)}$ vektorit

od

//nüüd on kõikvõimalikud kombinatsioonid läbi vaadatud

//selleks kulub ülimalt rohkem $2\,d+1+\ (2\,d+1)^2+..+\ (2\,d+1)^n$ vektori läbivaatust

return $\mathbf{y}$,d

See viitab vähemalt NP-täielikule ülesandele, mis on lõpmatusnormi korral ka tõestatud. Teiste normide korral on tegu NP-raske ülesandega, sest kui meil on olemas lahendus, siis selle korrektsuses veendumiseks tuleb meil see algoritm läbi teha(võibolla on kuskil veel parem algoritm aga keerukus on garanteeritult eksponentsiaalne), milleks võib kuluda eksponentsiaalne aeg. Muidugi mida täpsem on d hinnang, seda rohkem lineaarkombinatsioone saab välja jätta ning üldiselt laheneb ülesanne paremini kui võiks loota. Kui lõpmatusnormi asemel on näiteks 2-norm, siis tuleb ideed 5 veidi modifitseerida, kui mõtte üdi jääb samaks: tuleb kasutada sõltumatuid komponente lineaarkombinatsiooni normi alt hindamiseks.