shellLL v1.0

The world's first (and hopefully last) pure-bash implementation of the LLL algorithm!

---

https://github.com/asuhl-ucsd/shellLL

Why a shell script?

- Usability
- Readability
- Maintainability
- or maybe you accidentally deleted everything in /usr/bin except bash

```
1  #!/bin/bash
2  set -e
3
4  gcd() {
5   # Usage: gcd a b
6   # Prints gcd(a,b) to stdout
7   if (("$1" < 0)) ; then
8    gcd $((-1 * "$1")) "$2"
9   elif (("$2" > "$1")) ; then
10   gcd "$2" "$1"
11  elif (("$2" == 0)) ; then
12   printf '%d\n' "$1"
13  else
14   gcd "$2" $(( $1 % $2 ))
15  fi
16 }
```

```
18 ### Integer vector (ivec) functions ###
19 #
20 # ivecs are integers separated by commas
21 # e.g. "1,2,3,4,5" represents the vector
       [1,2,3,4,5].
```

```
23 ivec_scale() {
24  local A i
25  IFS=',' read -ra A <<-EOF
26   $2
27  EOF
28  printf "%d" "$(( A[0] * $1 ))"
29  i=0
30  while (( ++i < "${#A[*]}" )); do
31   printf ",%d" "$(( A[i] * $1 ))"
32  done
33  echo
34 }
```

```
36 ivec_add() {
37   local A B i=0
38   IFS=',' read -ra A <<-EOF
39     $1
40   EOF
41   IFS=',' read -ra B <<-EOF
42     $2
43   EOF
44   if [ "${#A[*]}" -ne "${#B[*]}" ]; then
45     echo "ivec_add: vectors must have same
           dimension" 1>&2
46     return 1
47   fi
48   printf "%d" "$(( A[0] + B[0] ))"
49   while (( ++i < "${#A[*]}" )) ; do # start
         from i=1
50     printf ",%d" "$(( A[i] + B[i] ))"
51   done
52 }
```

```
54  ivec_sub() {
55   local A B i=0
56   IFS=',' read -ra A <<-EOF
57    $1
58   EOF
59   IFS=',' read -ra B <<-EOF
60    $2
61   EOF
62   if [ "${#A[*]}" -ne "${#B[*]}" ]; then
63    echo "ivec_sub: vectors must have same
          dimension" 1>&2
64    return 1
65   fi
66   printf "%d" "$(( A[0] - B[0] ))"
67   while (( ++i < "${#A[*]}" )) ; do # start
         from i=1
68    printf ",%d" "$(( A[i] - B[i] ))"
69   done
70  }
```

```
72  ivec_dot() {
73   local accum=0 i=0 A B
74   IFS=',' read -ra A <<-EOF
75    $1
76   EOF
77   IFS=',' read -ra B <<-EOF
78    $2
79   EOF
80   (( "${#A[*]}" == "${#B[*]}" )) || exit 1
81   i=0
82   while (( i < "${#A[*]}" )); do
83    accum=$(( accum + (A[i] * B[i]) ))
84    (( ++i ))
85   done
86   printf '%d\n' "$accum"
87  }
```

```
89  ##### Functions for working with rationals
90
91  rfrac() {
92    # Usage: rfrac denom num
93    # Reduces the fraction
94    # Prints output to stdout as "denom num"
95    local common denom num
96    common="$(gcd "$1" "$2")"
97    denom="$(( $1 / common ))"
98    num="$(( $2 / common ))"
99    printf "%d %d\n" "$denom" "$num"
100 }
```

```
##### BigRat(ionals)
#
# "Rationals of unusual size?
#    I don't think they exis--"
#
#      n   r+    _,-===-,_
#    __L\_/@|-+"          "+,
#  /     o  /               \
# 0                          k_,-==-,_
#  \vv--_                     --'""--,"\
#        \                   /         ||
#        / ___     _,=,_    /  _,,,___//
#      <<,/   <<,/   <,,,"   "=""==--"
#
```

```
##### BigRat(ionals)
#
# "Rationals of unusual size?
#    I don't think they exis--"
#
#      n   r+    _,-===-,_
#    __L\_/@|-+"          "+,
#  /    o  /                \
# 0                          k_,-==-,_
#  \vv--_                     --'""--,"\
#        \                   /         ||
#        / ___    _,=,_    /  _,,,___//
#      <<,/   <<,/   <,,,"  "=""==--"
#
## TODO: implement
```

```
118 roundnearest() {
119   # round to nearest, rounding .5 toward +Inf
120   # (bash's division rounds toward zero)
121   # overflows if abs(num) > 2^62 or so
122   local num="$2" denom="$1"
123   if (( (num < 0) && (-num != num) )); then
124     # ensure num is nonnegative
125     roundnearest "$((-denom))" "$((-num))"
126     return
127   fi
128   if (( (denom > 0) )); then
129     printf "%d\n" "$(( (2 * num + denom) / (2
          * denom) ))"
130   else
131     printf "%d\n" "$(( (2 * (num % denom) > -
          denom) ? (num / denom) - 1 : (num /
          denom) ))"
132   fi
133 }
```

```
135 ##### Rational vector (rvec) functions #####
136 #
137 #   rvecs are integers separated by colons
138 #   The first integer is the denominator for
        the entire vector.
139 #   The rest are the numerators.
140 #   e.g. "10:9:8:7" represents the vector
        [0.9, 0.8, 0.7].
141 #   We do this instead of having separate
        denominators for each component
142 #   because "it seemed like a good idea at
        the time."
```

```
144  ivec_to_rvec() {
145   # Usage: ivec_to_rvec ivec
146   # Convert an integer vector to a rational
          vector
147   # Prints rvec result to stdout
148   # Example: ivec_to_ratvec 1,2,3
149   # Example output: 1:1:2:3
150   local A
151   IFS=',' read -ra A <<-EOF
152    $1
153   EOF
154   (IFS=":" ; printf "1:%s\n" "${A[*]}")
155  }
```

```
157 rvec_reduce() {
158   local accum A i=0 j=0
159   IFS=':' read -ra A <<-EOF
160     $1
161   EOF
162   # Step 1: take gcd of everything
163   accum="${A[0]}"
164   while (( ++i < "${#A[*]}" )); do
165     accum="$( gcd "${A[$i]}" "$accum" )"
166   done
167   # Step 2: divide everything by gcd
168   printf "%d" "$(( A[j] / accum ))"
169   while (( ++j < "${#A[*]}" )); do
170     printf ":%d" "$(( A[j] / accum ))"
171   done; echo
172 }
```

```
174 rvec_scale() {
175   # Usage: rvec_scale denom num rvec
176   # Computes (num/denom) * rvec
177   # rvecs should be integers separated by
        colons, where the first is the
        denominator
178   # Prints rvec result to stdout
179   local A i
180   IFS=':' read -ra A <<-EOF
181    $3
182   EOF
183   printf "%d" "$(( A[0] * $1 ))"
184   i=0
185   while (( ++i < "${#A[*]}" )); do
186    printf ":%d" "$(( A[i] * $2 ))"
187   done; echo
188 }
```

```
190 rvec_add() {
191   # Usage: rvec_add rvec1 rvec2
192   # Performs vector+vector addition of two
          rational vectors
193   # Prints (reduced) rvec result to stdout
194   local A B g denom scaleA scaleB AplusB
195   IFS=':' read -ra A <<-EOF
196    $1
197   EOF
198   IFS=':' read -ra B <<-EOF
199    $2
200   EOF
201   if [ "${#A[*]}" -ne "${#B[*]}" ]; then
202    echo "rvec_add: vectors must have same
          dimension" 1>&2
203    return 1
204   fi
```

```
205   g="$(gcd "${A[0]}" "${B[0]}" )"
206   denom="$(( (A[0] * B[0]) / g ))" # denom =
          lcm(denom1, denom2)
207   scaleA="$(( B[0] / g ))"
208   scaleB="$(( A[0] / g ))"
209   AplusB="$( {
210    printf "%d" "$denom"
211    i=0
212    while (( ++i < "${#A[*]}" )) ; do # start
          from 1
213     printf ":%d" "$(( A[i] * scaleA + B[i] *
            scaleB ))"
214    done
215    unset i
216   } )"
217   rvec_reduce "$AplusB"
218  }
```

```
220 rvec_sub() {
221   # Usage: rvec_sub rvec1 rvec2
222   # Performs vector-vector subtraction of two
          rational vectors
223   # Prints (reduced) rvec result to stdout
224   local A B g denom scaleA scaleB AminusB
225   IFS=':' read -ra A <<-EOF
226    $1
227   EOF
228   IFS=':' read -ra B <<-EOF
229    $2
230   EOF
231   if [ "${#A[*]}" -ne "${#B[*]}" ]; then
232    echo "rvec_sub: vectors must have same
          dimension" 1>&2
233    return 1
234   fi
```

```
235   g="$(gcd "${A[0]}" "${B[0]}" )"
236   denom="$(( (A[0] * B[0]) / g ))" # denom =
          lcm(denom1, denom2)
237   scaleA="$(( B[0] / g ))"
238   scaleB="$(( A[0] / g ))"
239   AminusB="$( {
240    printf "%d" "$denom"
241    i=0
242    while (( ++i < "${#A[*]}" )) ; do # start
          from 1
243     printf ":%d" "$(( A[i] * scaleA - B[i] *
           scaleB ))"
244    done
245    unset i
246   } )"
247   rvec_reduce "$AminusB"
248  }
```

```
254   local A B accum i=0 num=0 denom
255   IFS=':' read -ra A <<-EOF
256    $1
257   EOF
258   IFS=':' read -ra B <<-EOF
259    $2
260   EOF
261   (( "${#A[*]}" == "${#B[*]}" )) || return 1
262   while (( ++i < "${#A[*]}" )); do
263    num=$(( num + (A[i] * B[i]) ))
264   done
265   denom="$(( A[0] * B[0] ))"
266   rfrac "$denom" "$num"
267 }
```

```
250 rvec_dot() {
251   # Usage: rvec_dot rvec1 rvec2
252   # Computes dot product of two rational vecs
253   # Prints result to stdout as "denom num"
254   local A B accum i=0 num=0 denom
255   IFS=':' read -ra A <<-EOF
256     $1
257   EOF
258   IFS=':' read -ra B <<-EOF
259     $2
260   EOF
261   (( "${#A[*]}" == "${#B[*]}" )) || return 1
262   while (( ++i < "${#A[*]}" )); do
263     num=$(( num + (A[i] * B[i]) ))
264   done
265   denom="$(( A[0] * B[0] ))"
266   rfrac "$denom" "$num"
267 }
```

```
269 rvec_mu() {
270   # Usage: rvec_mu a b
271   # Computes <a,b> / <b,b> (where a and b are
          rvecs)
272   # Prints scalar result to stdout as (
          reduced) "denom num"
273   local denom1 num1 denom2 num2
274   read -r denom1 num1 <<-EOF
275    $( rvec_dot "$1" "$2" )
276   EOF
277   read -r denom2 num2 <<-EOF
278    $( rvec_dot "$2" "$2" )
279   EOF
280   rfrac "$(( denom1 * num2 ))" "$(( denom2 *
          num1 ))"
281 }
```

```
283 rvec_proj() {
284   # Usage: rvec_proj a b
285   # where a and b are both rvecs
286   # Computes the projection of a onto b, i.e
        ., b * mu(a,b)
287   # Prints (reduced) rvec result to stdout
288   local denom num
289   read -r denom num <<-EOF
290    $( rvec_mu "$1" "$2" )
291   EOF
292   rvec_reduce "$( rvec_scale "$denom" "$num"
        "$2" )"
293 }
```

```
295  rvec_projout() {
296    # Usage: rvec_projout a b
297    # where a and b are both rvecs
298    # Projects out the b component of a,
         leaving something orthogonal to b
299    # i.e., rvec_projout(a,b) + rvec_proj(a,b)
         = a
300    # Prints (reduced) rvec result to stdout
301    rvec_reduce "$( rvec_sub "$1" "$( rvec_proj
         "$1" "$2" )" )"
302  }
```

```
304  rvec_iszero() {
305   # Returns 0 if vector is zero, nonzero if
          vector is nonzero
306   local i _vec
307   IFS=':' read -ra _vec <<-EOF
308    $1
309   EOF
310   i=1
311   while (( i < ${#_vec[*]} ))
312   do
313    if [ "${_vec[$i]}" -ne 0 ]; then
314     return 1
315    fi
316    i="$(( i + 1 ))"
317   done
318   return 0
319  }
```

```
321 ##### Gram-Schmidt Orthogonalization, Babai's Nearest Plane, etc. #####
322 #
323 #   These functions read/write to global array variable Bstar (which consists of rvecs)
324 #   and in some cases global array variable Basis (which consists of ivecs)
```

```
326 rvec_gs() {
327  local tmp
328  declare -ag Bstar
329  while read -r basisvec
330  do
331   tmp="$basisvec"
332   for bstarvec in "${Bstar[@]}" ; do
333    tmp="$( rvec_projout "$tmp" "$bstarvec" )
         "
334   done
335   if ! rvec_iszero "$tmp" ; then
336    Bstar[${#Bstar[*]}]="$tmp"
337    printf "%s\n" "$tmp"
338   fi
339  done
340 }
```

```
342 rvec_projoutall() {
343   local i=0 x="$1" l="$2" c
344   l=$(( l < "${#Bstar[*]}" ? l : "${#Bstar
          [*]}" ))
345   while (( i < l )); do
346     c="$( rvec_mu "$x" "${Bstar[$i]}" )" # c
          of the form "denom num"
347     x="$( rvec_sub "$x" "$( rvec_scale $c "${
          Bstar[$i]}" )" )"
348     i="$((i + 1))"
349   done
350   printf "%s\n" "$x"
351 }
```

```
nearestplane() {
 local target="$1" l="$2" i="$2" out
 while (( i > 0 )); do
  i="$(( i - 1 ))"
  # c = round(mu(target, Bstar[i]))
  local c="$( roundnearest $(rvec_mu "
      $target" "${Bstar[$i]}" ) )"
  if [ -z "$out" ]; then
   out="$( ivec_scale "$c" "${Basis[$i]}" )"
  else
   out="$( ivec_add "$out" "$(ivec_scale "$c
      " "${Basis[$i]}" )" )"
  fi
  target="$( rvec_add "$target" "$(
      ivec_to_rvec "$(ivec_scale "$(( -c ))"
      "${Basis[$i]}" )" )" )"
 done
 printf "%s\n" "$out"
}
```

```
369  sizereduce() {
370   # Modifies Basis in-place; Bstar does not
          change
371   local i=1 tmp
372   while (( i < "${#Basis[*]}" )) ; do
373    tmp="$(ivec_to_rvec "${Basis[$i]}")"
374    tmp="$(rvec_sub "$tmp" "${Bstar[$i]}")"
375    tmp="$(nearestplane "$tmp" "$i")" # tmp is
          an ivec now
376    Basis[$i]="$(ivec_sub "${Basis[$i]}" "$tmp
         ")"
377    i=$(( i + 1 ))
378   done
379  }
```

```
383 readmatrix() {
384   # Read a matrix as one vector per line,
          space-separated integers
385   # Compute GSO at the same time
386   # Stores output in globals Basis and Bstar
387   # Matrix must be square and full-rank
388   local j=0 tmp
389   unset Basis Bstar
390   declare -ag Basis
391   declare -ag Bstar
392   while read -r -a basisvec
393   do
394     # basisvec is space separated; we want
            comma separated
395     Basis[$j]="$( (IFS="," ; printf "%s" "${
            basisvec[*]}") )"
396     tmp="1:$( (IFS=":" ; printf "%s" "${
            basisvec[*]}") )" # rvec
397     for bstarvec in "${Bstar[@]}" ; do
```

```
397     for bstarvec in "${Bstar[@]}" ; do
398      tmp="$( rvec_projout "$tmp" "$bstarvec" )
             "
399     done
400     if ! rvec_iszero "$tmp" ; then
401      Bstar[${#Bstar[*]}]="$tmp"
402     else
403      printf "Error: Matrix is not full-rank\n"
             1>&2
404      return 1
405     fi
406     j=$((j+1))
407   done
408   unset tmp
409 }
```

```
412 lll() {
413  local tmp i bhat bprimehat LHS_num
         LHS_denom RHS_num RHS_denom
414  # If stdin and stderr are terminals, show a
       help message.
415  if [ -t 0 ] && [ -t 2 ]; then
416   echo 'Enter a full-rank integer matrix,
       one vector per line, as numbers
       separated by spaces. Press Ctrl-D after
        the last line.' 1>&2
417  fi
418  readmatrix # does GSO, puts result in Bstar
419  echo "Running LLL..." 1>&2
420  while true; do
421   sizereduce
422   for (( i=0 ; i < ${#Basis[*]} ; i++ )); do
423    if (( i == ${#Basis[*]} - 1 )); then
424     break 2
425    fi
```

```
426     bhat="$( rvec_projoutall "$(ivec_to_rvec
            "${Basis[i]}")" "$i" )" # project out
            Bstar[0...i-1] from B[i]
427     bprimehat="$( rvec_projoutall "$(
            ivec_to_rvec "${Basis[i+1]}")" "$i" )"
             # project out Bstar[0], ..., Bstar[i
            -1] from B[i+1]
428     # lovasz: delta*||bhat||^2 >? ||b'hat||^2
429     read -r LHS_denom LHS_num <<-EOF
430      $(rvec_dot "$bhat" "$bhat")
431     EOF
432     LHS_denom="$((LHS_denom * 100))"
433     LHS_num="$((LHS_num * 99))"
434     read -r LHS_denom LHS_num <<-EOF
435      $(rfrac "$LHS_denom" "$LHS_num")
436     EOF
437     read -r RHS_denom RHS_num <<-EOF
438      $(rvec_dot "$bprimehat" "$bprimehat")
439     EOF
```

```
440       if (( LHS_num * RHS_denom > RHS_num *
              LHS_denom )); then
441         tmp="${Basis[i]}"
442         Basis[i]="${Basis[i+1]}"
443         Basis[i+1]="$tmp"
444         Bstar[i]="$( rvec_projoutall "$(
                ivec_to_rvec "${Basis[i]}")" "$i" )"
445         Bstar[i+1]="$( rvec_projoutall "$(
                ivec_to_rvec "${Basis[i+1]}")" "$((i
                +1))" )"
446         break 1
447       fi
448     done
449   done
450   for tmp in "${Basis[@]}"; do
451     printf "%s\n" "$tmp"
452   done
453 }
```

code is at

`github.com/asuhl-ucsd/shellLL`

in case you blinked and missed something

Now that you've read all ≈ 500 lines of code...

Let's do a "live" demo!

```
adam@cryptomania$ cat matrix
1 5 21 3 14
17 0 12 12 13
12 21 15 6 6
4 13 24 7 16
20 9 22 27 8
adam@cryptomania$
```

```
adam@cryptomania$ cat matrix
1 5 21 3 14
17 0 12 12 13
12 21 15 6 6
4 13 24 7 16
20 9 22 27 8
adam@cryptomania$ time ./lll.sh <matrix
```

```
adam@cryptomania$ cat matrix
1 5 21 3 14
17 0 12 12 13
12 21 15 6 6
4 13 24 7 16
20 9 22 27 8
adam@cryptomania$ time ./lll.sh <matrix
Running LLL...
```

```
adam@cryptomania$ cat matrix
1 5 21 3 14
17 0 12 12 13
12 21 15 6 6
4 13 24 7 16
20 9 22 27 8
adam@cryptomania$ time ./lll.sh <matrix
Running LLL...
3,-3,6,-6,0
3,8,3,4,2
-3,-7,4,7,-9
11,5,-3,-7,-10
8,-13,0,10,9

real    0m1.413s
user    0m1.109s
sys     0m0.385s
adam@cryptomania$
```

Open Questions

Open Questions

- why did I do this

Thanks!

https://github.com/asuhl-ucsd/shellLL