Experiments with quad precision for iterative solvers

Ding Ma and Michael Saunders

Management Science and Engineering (MS&E)
Systems Optimization Laboratory (SOL)
Institute for Computational Mathematics and Engineering (ICME)
Stanford University

SIAM Conference on Optimization

San Diego, CA

Abstract

Implementing conjugate-gradient-type methods in Quad precision is a plausible alternative to reorthogonalization in Double precision. Since the data for $Ax = b$ is likely to be Double, products $y = Av$ (with Quad y and v) need to take advantage of A being Double; for example, by splitting $v = v_1 + v_2$ (with v_1 and v_2 Double) and forming $y = Av_1 + Av_2$. This needs dot-products with double-precision data and quad-precision accumulation, but there is no such intrinsic function in Fortran 90. We explore ways of using double-precision floating-point to achieve quad-precision products.

Partially supported by the
National Institute of General Medical Sciences
of the National Institutes of Health (NIH)
Award U01GM102098

Color coding

green	$\Rightarrow$	**Double**	A, b
magenta	$\Rightarrow$	**Quad**	y, v

Motivation

real(16), float128
Quad LP
Quad LSMR

Why Quad CG, MINRES, LSQR, LSMR, . . . ?

- Because we can — real(16), float128 — gfortran, GCC libquadmath
- Pity poor President Clinton
- Does anyone do CG in Single? — rapid loss of orthogonality
- Quad LP has worked well — Quad SQOPT, MINOS
- Quad LSMR works well on the LSQR test problems

LSQR and LSMR

$$Ax = b \qquad \min \|Ax - b\| \qquad \min \left\| \begin{pmatrix} A \\ \delta I \end{pmatrix} - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|$$

- LSQR (Paige & S 1982) $\equiv$ CG on $A^T A x = A^T b$
  LSMR (Fong & S 2011) $\equiv$ MINRES on $A^T A x = A^T b$

- Stopping rules:

$$Ax = b: \qquad \|r_k\| \leq \texttt{atol}\, \|A\|\, \|x_k\| + \texttt{btol}\, \|b\|$$

$$\min \|Ax - b\|: \qquad \|A^T r_k\| \leq \texttt{atol}\, \|A\|\, \|r_k\|$$

Implementation in f90

One f90 approach

```
module lsmrDataModule
  integer, parameter ::  ip = 4
  integer, parameter ::  dp = 8
end module lsmrDataModule
```

A 1-line change converts solver from Double to Quad

```
module lsmrDataModule
  integer, parameter ::  ip = 4
  integer, parameter ::  dp = 16
end module lsmrDataModule
```

One f90 approach

```
module lsmrModule
  use  lsmrDataModule,    only : ip, dp        ! dp = 8 or 16

contains
  subroutine LSMR  ( m, n, Aprod1, Aprod2, b, damp,                    &
                     atol, btol, conlim, itnlim, localSize, nout,      &
                     x, istop, itn, normA, condA, normr, normAr, normx )
    integer(ip), intent(in)  :: m, n, itnlim, localSize, nout
    integer(ip), intent(out) :: istop, itn
    real(dp),    intent(in)  :: b(m)
    real(dp),    intent(out) :: x(n)
    real(dp),    intent(in)  :: atol, btol, conlim, damp
    real(dp),    intent(out) :: normA, condA, normr, normAr, normx

    interface
       subroutine Aprod1(m,n,x,y)                    ! y := y + A*x
         use lsmrDataModule, only : ip, dp
         integer(ip), intent(in)    :: m, n
         real(dp),    intent(in)    :: x(n)
         real(dp),    intent(inout) :: y(m)
       end subroutine Aprod1
```

Alternative f90 approach

```
module qlsmrModule
  use  qlsmrDataModule,   only : ip, dp, qp    ! qp = 16

contains
  subroutine qLSMR ( m, n, Aprod1, Aprod2, b, damp,                    &
                     atol, btol, conlim, itnlim, localSize, nout,      &
                     x, istop, itn, normA, condA, normr, normAr, normx )
    integer(ip), intent(in)  :: m, n, itnlim, localSize, nout
    integer(ip), intent(out) :: istop, itn
    real(qp),    intent(in)  :: b(m)
    real(qp),    intent(out) :: x(n)
    real(qp),    intent(in)  :: atol, btol, conlim, damp
    real(qp),    intent(out) :: normA, condA, normr, normAr, normx

    interface
       subroutine Aprod1(m,n,x,y)                    ! y := y + A*x
         use lsmrDataModule, only : ip, qp
         integer(ip), intent(in)    :: m, n
         real(qp),    intent(in)    :: x(n)
         real(qp),    intent(inout) :: y(m)
       end subroutine Aprod1
```

Quad LP

Quad LP

"Obvious" approach:

- Cold start Double simplex or barrier
- Warm start Quad simplex
- A few additional iterations give astounding accuracy at moderate cost

- Analogous strategy for Quad CG, LSMR, etc?
- Cold start Double CG + Warm start Double CG? Normally not good
- Cold start Double CG + Warm start Quad CG? Good for high accuracy

- First task: compare iterations for Double LSMR and Quad LSMR

The LSQR test problems (ACM TOMS 1982)

$P(m, n, d, p, \delta)$ generates a least-squares test problem

$$\min \|Ax - b\| \quad \text{or} \quad \min \left\| \begin{pmatrix} A \\ \delta I \end{pmatrix} x - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|$$

with $m \times n$ matrix $A = (I - 2yy^T)D(I - 2zz^T)$ and known solution x

- $D = diag(\sigma_i^p)$
- $d = 40$ duplicates of each singular value σ_i^p
- $p =$ 3 4 5 6 7 8
  $\delta =$ 10^{-3} 10^{-4} 10^{-5} 10^{-6} 10^{-7} 0

  increasingly ill-conditioned
- $(m, n) = (2000, 1000), \quad (1000, 1000), \quad (1000, 2000)$

Iterations for LSMR on three groups of increasingly ill-conditioned problems with 40 clusters of singular values in A

$$\min \left\| \begin{pmatrix} A \\ \delta I \end{pmatrix} x - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|$$

$$A = (I - 2yy^T) D (I - 2zz^T)$$

$$"D" = \begin{pmatrix} D \\ 0 \end{pmatrix}, \quad D, \quad \begin{pmatrix} D & 0 \end{pmatrix}$$

2000×1000
1000×1000
1000×2000

	δ	Double LSMR	Quad LSMR
$m > n$	10^{-3}	62	38
	10^{-4}	93	48
	10^{-5}	133	57
	10^{-6}	173	48
	10^{-7}	241	47
	0	265	46
$m = n$	10^{-3}	102	39
	10^{-4}	172	50
	10^{-5}	218	61
	10^{-6}	324	73
	10^{-7}	465	87
	0	735	91
$m < n$	10^{-3}	101	40
	10^{-4}	174	48
	10^{-5}	220	61
	10^{-6}	328	74
	10^{-7}	496	84
	0	763	88
		0.11 secs	0.59 secs

LPnetlib test problems

How many iterations in Double and Quad?

LPnetlib test problems

- Tim Davis's Sparse Matrix collection contains 86 LPnetlib problems

  $$\min\ c^T x\ \ st\ \ Ax = b,\ \ l \le x \le u$$

  Downloaded in Matrix Market format

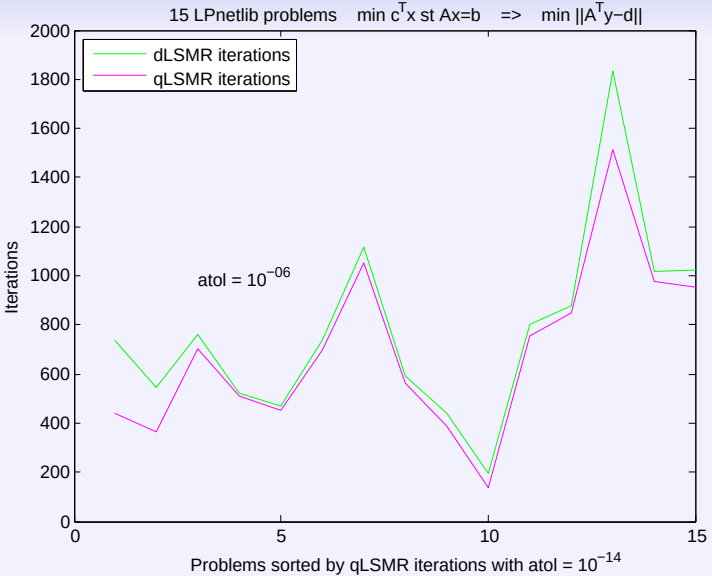
- Apply f90 LSMR to least squares problems $\min \|A^T y - d\|$, $d = \begin{bmatrix} +1 \\ -1 \\ +1 \\ -1 \\ \vdots \end{bmatrix}$

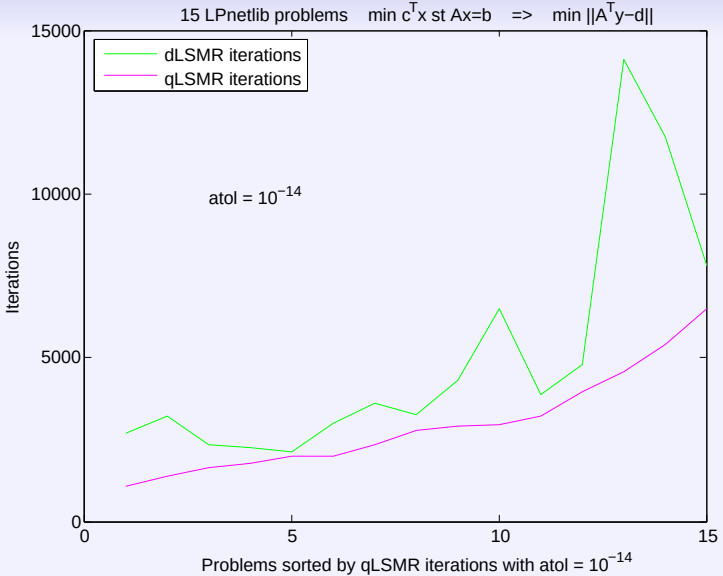
  Columns of A^T normalized first

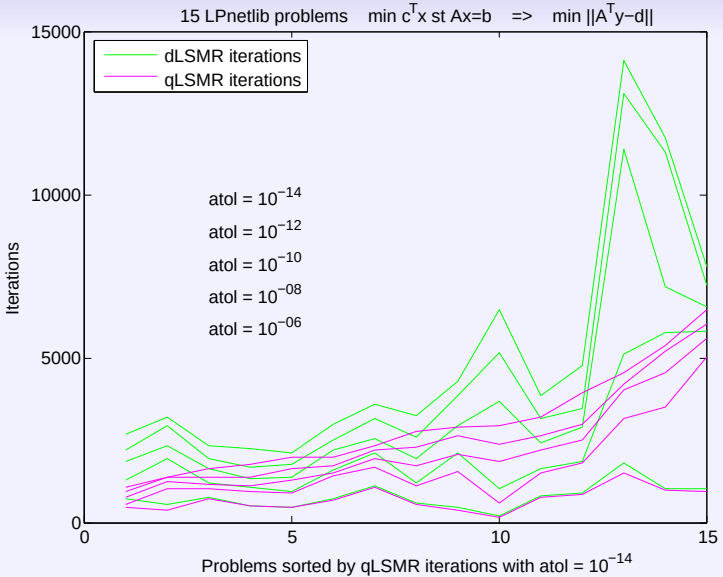
  On 15 problems, Double LSMR needed > 2000 iterations

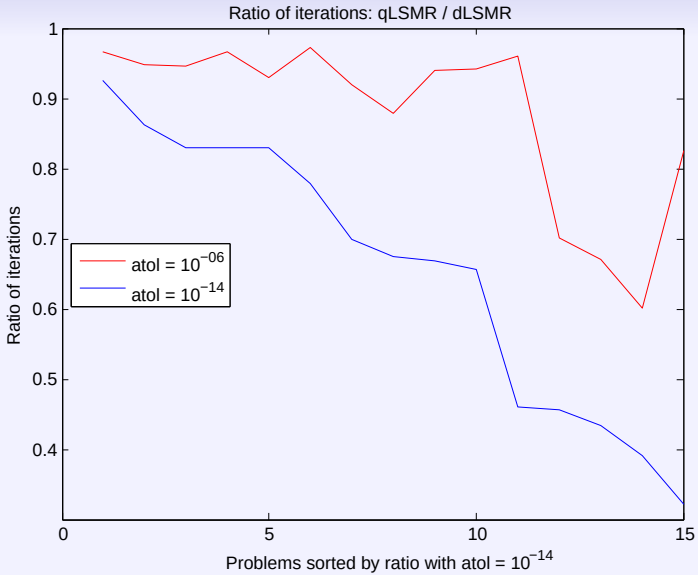
- Apply Double LSMR and Quad LSMR with various stopping tolerances

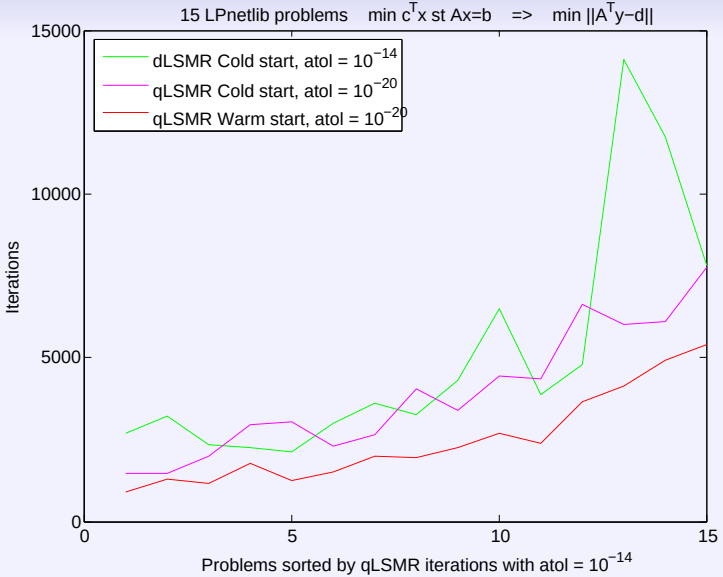
  `atol = btol` $= 10^{-6},\ 10^{-8},\ 10^{-10},\ 10^{-12},\ 10^{-14}$


15 LPnetlib problems min $c^T x$ st $Ax=b$ => min $||A^T y - d||$
dLSMR iterations
qLSMR iterations
atol = 10^{-06}
Iterations
Problems sorted by qLSMR iterations with atol = 10^{-14}
0
200
400
600
800
1000
1200
1400
1600
1800
2000
0
5
10
15


15 LPnetlib problems min $c^T x$ st Ax=b => min $||A^T y - d||$
dLSMR iterations
qLSMR iterations
atol = 10^{-14}
Iterations
15000
10000
5000
0
0
5
10
15
Problems sorted by qLSMR iterations with atol = 10^{-14}


15 LPnetlib problems min $c^T x$ st Ax=b => min $||A^T y - d||$
dLSMR iterations
qLSMR iterations
atol = 10^{-14}
atol = 10^{-12}
atol = 10^{-10}
atol = 10^{-08}
atol = 10^{-06}
Iterations
15000
10000
5000
0
0
5
10
15
Problems sorted by qLSMR iterations with atol = 10^{-14}


Ratio of iterations: qLSMR / dLSMR
Ratio of iterations
Problems sorted by ratio with atol = 10^{-14}
atol = 10^{-06}
atol = 10^{-14}
1
0.9
0.8
0.7
0.6
0.5
0.4
0
5
10
15


15 LPnetlib problems min c^T x st Ax=b => min ||A^T y−d||
dLSMR Cold start, atol = 10^−14
qLSMR Cold start, atol = 10^−20
qLSMR Warm start, atol = 10^−20
Iterations
15000
10000
5000
0
0
5
10
15
Problems sorted by qLSMR iterations with atol = 10^−14

The challenge

- ONE DAY machines will have Quad hardware
- Right now we have GCC libquadmath in software
- Quad is 10–50 times slower than Double
- LP or CG methods 10–50 times fewer iterations? Not likely

- Second task: speed up Av, $A^T u$

Matrix-vector products

$$y = Av$$

Matrix-vector products $y = Av$

- Likely to have Double data A, b (not Quad)
- We need Double*Quad products $y = Av$ (each row a dot product $y_i = a_i^T v$)
- f90 has `y = dot_product(a,v)` but vectors must have the same type
- We can write our own loop, but still not good:

```
do j = 1, n                          do j = 1, n
   y = y + a(i,j)*v(j)       ≡          y = y + quad(a(i,j))*v(j)
end do                               end do
```

The need for qdot(a,v) dot-products with accumulation

- Iterative refinement of $Ax = b$
  Needs high-precision residuals $r = \text{quad}(b - Ax)$
  For each row of A we need $r_i = b_i - \text{qdot}(a_i, x)$

- Products $y = Av$
  We can split $v = v_1 + v_2$ (the sum of two double vectors):

$$\begin{aligned} \text{v1} &= \text{double(v)} \\ \text{q1} &= \text{quad(v1)} \\ \text{v2} &= \text{double(v-q1)} \end{aligned}$$

- Products $y = Av_1 + Av_2$
  seem to need $y = \text{qdot}(A, v_1) + \text{qdot}(A, v_2)$
  Luckily we can do $y = \text{qdot}(A, v_1) + \text{ddot}(A, v_2)$

Implementing qdot(d_1, d_2) using only Double

- T. J. Dekker (1971)
  A floating-point technique for extending the available precision
  Numer. Math. 18, 224–242
- Stef Graillat and Valérie Ménissier-Morain (2012)
  Accurate summation, dot product and polynomial evaluation in complex floating point arithmetic
  Information and Computation 216, 57–71

Some short utilities (all quantities Double):

```
twosum( a,b,x,y )          a+b    = x+y
split ( a,x,y )            a      = x+y
twoproduct( a,b,x,y )      a*b    = x+y
sum2  ( v,n,x,y )          sum(v) = x+y
dot2  ( v,w,n,x,y )        v'*w   = x+y
```

```
q = quad(x) + quad(y)  is accurate to Quad precision
```

Implementing qdot(d_1, d_2) using only Double

```
subroutine dot2( v,w,n,x,y )
  integer(ip), intent(in)  ::  n
  real(dp),    intent(in)  ::  v(n), w(n)
  real(dp),    intent(out) ::  x, y

  !   dot2 computes the dotproduct v(1:n)'*w(1:n).
  !  qdot2 = real(x,qp) + real(y,qp) is accurate to Quad precision.

  integer(ip) ::  i
  real(dp)    ::  x1, x2, y1, y2

  call twoproduct( v(1),w(1),x,y )

  do i = 2, n
     call twoproduct( v(i),w(i),x1,y1 )
     call twosum( x,x1,x2,y2 )
     x = x2
     y = y + (y1+y2)
  end do
end subroutine dot2
```

Timing qdot(d_1, d_2) using only Double

```
call dot2i( v,w,n,x,y )
q = quad(x) + quad(y)
```

vs

```
q = dot_product(qv,qw)
```

n	speedup
10	5.5
20	9.2
40	14.6
100	16.8
200	17.4
400	18.0
800	17.2
900	3.8
1000	1.8
2000	1.6

`v,w` = vectors of length n

`dot2i` = inline version of `dot2`

Timing Av, A^Tu using only Double

	m	n	itns
lp_israel	316	174	1567
lp_vtp_base	346	198	1518
lp_scfxm3	1800	990	2007
lp_stocfor2	3045	2157	2963
lp_cre_d	73948	8926	3026
lp_pilotnov	2446	975	2324
lp_pilot_ja	2267	940	2693
lp_cre_a	7248	3516	4070
lp_d2q06c	5831	2171	4372
lp_gfrd_pnc	1160	616	3429
lp_fit2p	13525	3000	4690
lp_stocfor3	23541	16675	6595
lp_fffff800	1028	524	6317
lp_maros	1966	846	6588
lp_cycle	3371	1903	7813

Quad LSMR `atol` $= 10^{-20}$

	Product time	Total time
Quad Av, A^Tu	580 secs	1220 secs
Av, A^Tu via Double	104 secs	740 secs
Speedup	6x	2x

Inexact Krylov methods

Inexact Krylov methods

- Approximate $y = Av$ by $y = Av_1$? $v_1 = \text{double}(v)$, $y \approx \text{quad}(y)$
- 15 digits is not that "inexact"!
- Unfortunately damages convergence of qLSMR (total time about the same)

atol $= 10^{-20}$	m	n	itns	$\rightarrow$ itns
lp_israel	316	174	1567	3796
lp_vtp_base	346	198	1518	4058
lp_scfxm3	1800	990	2007	2907
lp_stocfor2	3045	2157	2963	3818
lp_cre_d	73948	8926	3026	4034
lp_pilotnov	2446	975	2324	3770
lp_pilot_ja	2267	940	2693	4382
lp_cre_a	7248	3516	4070	5630
lp_d2q06c	5831	2171	4372	5339
lp_gfrd_pnc	1160	616	3429	4881
lp_fit2p	13525	3000	4690	9169
lp_stocfor3	23541	16675	6595	7825
lp_fffff800	1028	524	6317	16020
lp_maros	1966	846	6588	13008
lp_cycle	3371	1903	7813	13461

Inexact Krylov methods

Valeria Simoncini and Daniel Szyld:

Theory of inexact Krylov subspace methods and applications to scientific computing
SISC 25:2 454–477 (2003)

On the occurrence of superlinear convergence of exact and inexact Krylov subspace methods
SIAM Review 47:2 247–272 (2005)

- $p_k = (A + E_k)v_k$
- ... an additional possible explanation of the phenomenon that $\|E_k\|$ needs to be small in the initial iterations while it can be allowed to grow in later steps
- $\Rightarrow$ Quad for *early* iterations??
  Double later?

Conclusions

Conclusions

- Quad LSMR always takes fewer iterations
- Reduction factor 1–4 on LPnetlib data (not 2–10 like LSQR test problems)
- qdot(a,v) is much faster than quadmath dot_product
- Does anyone want to do iterative refinement on $Ax = b$?
- Does anyone have challenging LS problems?
- quadLP is already more successful!
  (see results from April 2012)

Flux Balance Analysis (FBA) on Thermotoga maritima

$$\min c^T v \text{ subject to } Sv = 0, \quad \ell \le v \le u$$

S rows and cols	18000×21000
Nonzero S_{ij}	34000
max and min $\lvert S_{ij} \rvert$	10^4 and 10^{-6}

SQOPT in double precision (15 digits)

```
Feasibility tol          1e-6
Optimality  tol          1e-6
```

SQOPT in quad precision (33 digits)

```
Feasibility tol          1e-15
Optimality  tol          1e-15
```

Flux Balance Analysis (FBA) on Thermotoga maritima

$$\min c^T v \text{ subject to } Sv = 0, \quad \ell \le v \le u$$

S rows and cols	18000×21000
Nonzero S_{ij}	34000
max and min $\lvert S_{ij} \rvert$	10^4 and 10^{-6}

SQOPT in double precision (42 secs)

```
SQOPT  EXIT  10 -- the problem appears to be infeasible

Problem name                     ThMa
No. of iterations               18500   Objective value      8.2286249495E-07
No. of infeasibilities              9   Sum of infeas        1.9606461069E-03
No. of degenerate steps         11611   Percentage                    62.76
Max x       (scaled)     3482 8.2E+00   Max pi      (scaled)    18210 9.8E-01
Max x                    5134 5.9E+00   Max pi                  18210 1.0E+00
Max Prim inf(scaled)    32832 1.3E-03   Max Dual inf(scaled)    16417 1.0E+00
Max Primal infeas       32832 5.6E-06   Max Dual infeas         32669 2.3E+02
```

Flux Balance Analysis (FBA) on Thermotoga maritima

$$\min c^T v \text{ subject to } Sv = 0, \quad \ell \le v \le u$$

S rows and cols	18000×21000
Nonzero S_{ij}	34000
max and min $\lvert S_{ij} \rvert$	10^4 and 10^{-6}

Restart SQOPT in quad precision (36 secs)

```
SQOPT  EXIT   0 -- finished successfully

Problem name                          ThMa
No. of iterations                      498   Objective value      8.7036461686E-07
No. of infeasibilities                   0   Sum of infeas        0.0000000000E+00
No. of degenerate steps                220   Percentage                     44.18
Max x       (scaled)     3482 8.2E+00   Max pi      (scaled)     2907 1.3E+00
Max x                    5134 5.9E+00   Max pi                  15517 1.1E+00
Max Prim inf(scaled)    16475 5.2E-28   Max Dual inf(scaled)    13244 1.9E-32
Max Primal infeas       16475 5.2E-29   Max Dual infeas         13244 4.8E-33
```