PRIMA: What, Why, How, and So What?

Zaikun Zhang

The Hong Kong Polytechnic University

Optimization 2023, Aveiro, Portugal

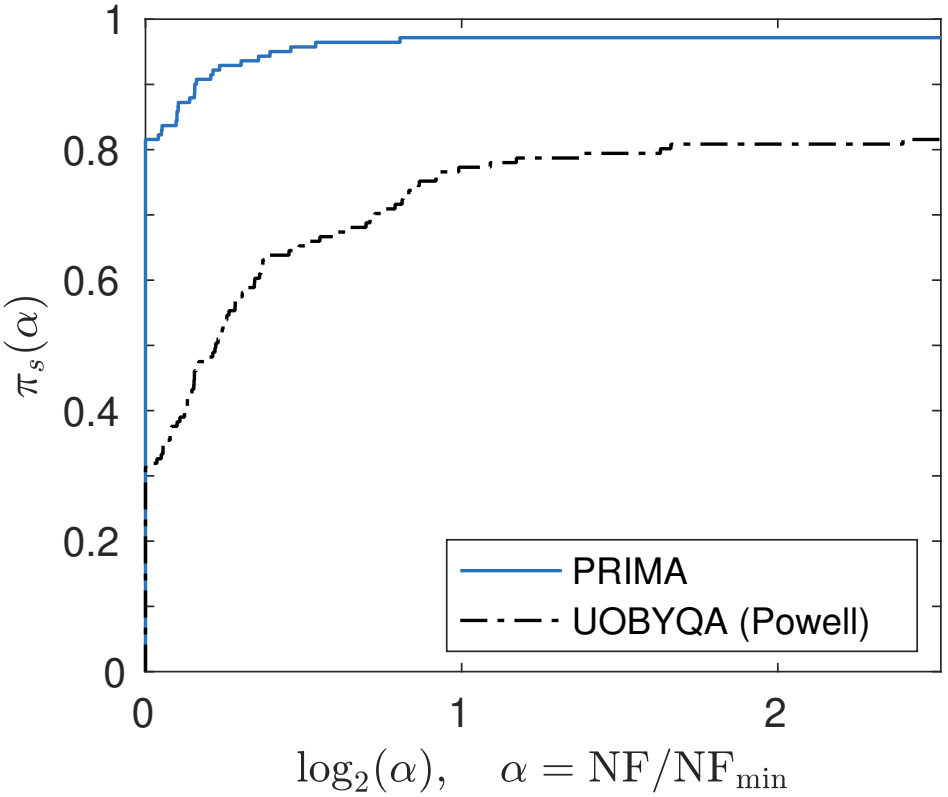
What is this talk about?

Topics of this talk:

1. A new DFO solver named PRIMA
2. The mathematics behind it

Let us start with some tests of PRIMA based on the CUTEst problems.

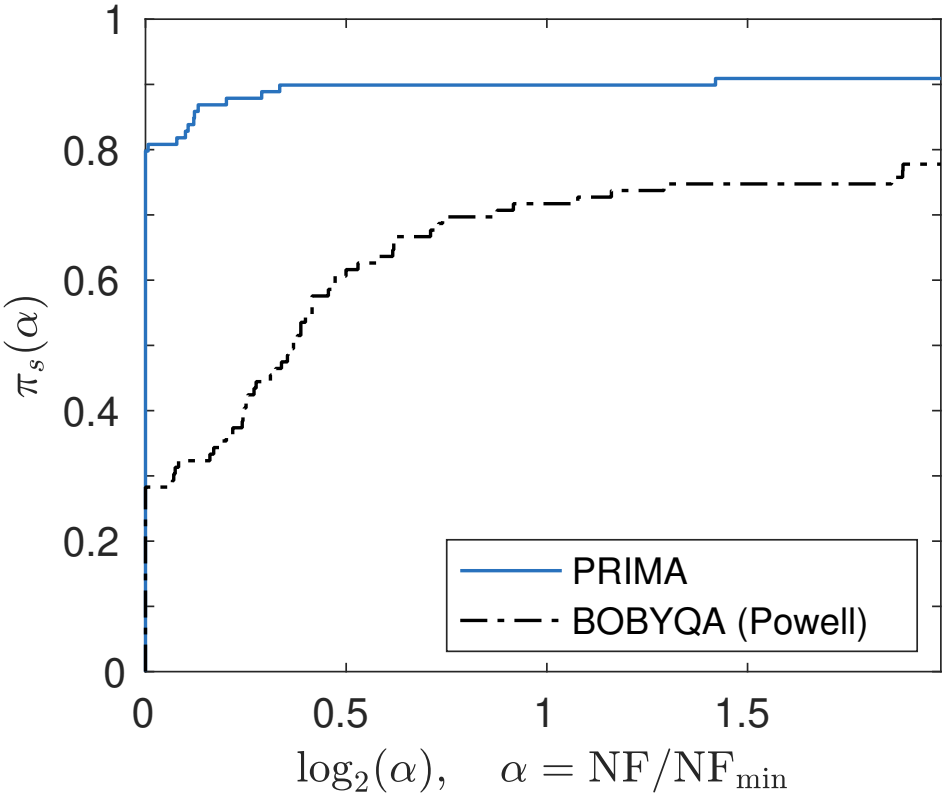
Performance of PRIMA compared with Powell's solvers



PRIMA v.s. UOBYQA

(unconstrained problems, at most 100 variables)

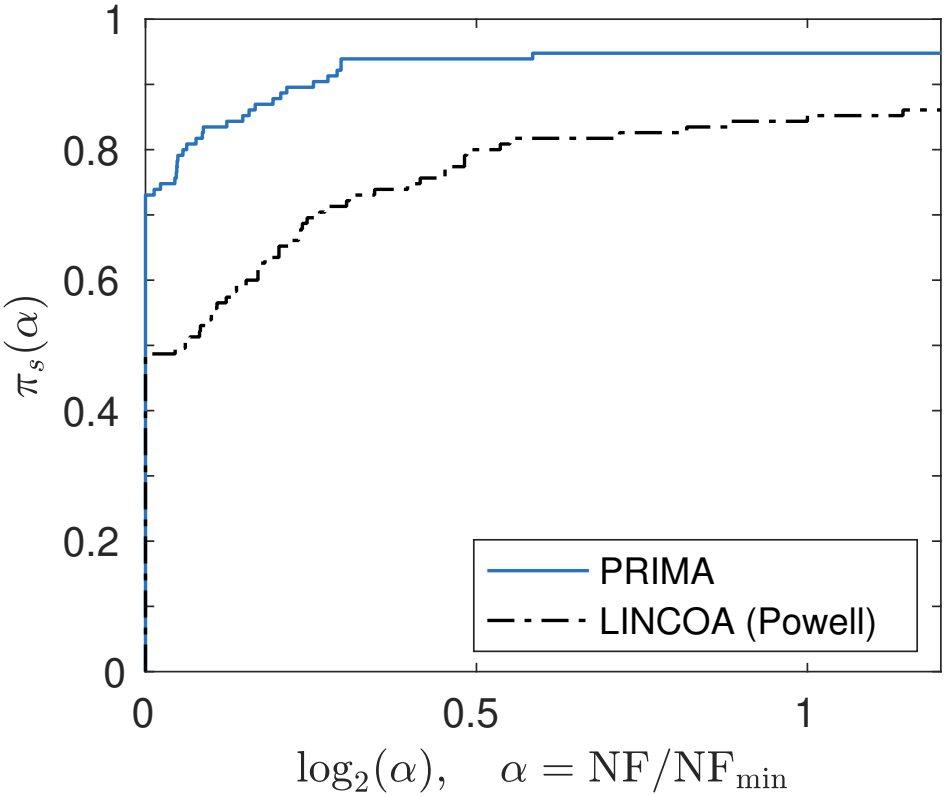
Performance of PRIMA compared with Powell's solvers



PRIMA v.s. BOBYQA

(bound-constrained problems, at most 200 variables)

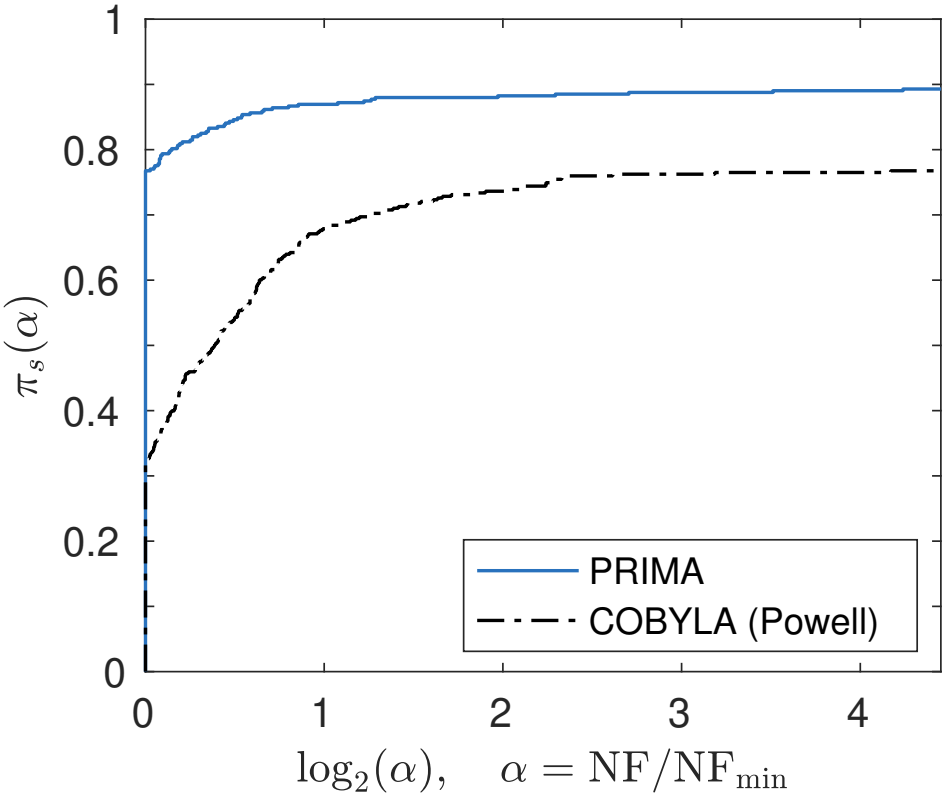
Performance of PRIMA compared with Powell's solvers



PRIMA v.s. LINCOA

(linearly constrained problems, at most 200 variables, 20,000 constraints)

Performance of PRIMA compared with Powell's solvers



PRIMA v.s. COBYLA

(nonlinearly constrained problems, at most 100 variables, 10,000 constraints)

Sorry, I lied ...

Topics of this talk:

1. ~~A new DFO solver named PRIMA~~

   PRIMA is not a new solver but a re-implementation of Powell's solvers.

2. ~~The mathematics behind it~~

   There is no mathematics in this talk.

PRIMA: Reference Implementation for Powell's Methods with Modernization and Amelioration

Zaikun Zhang

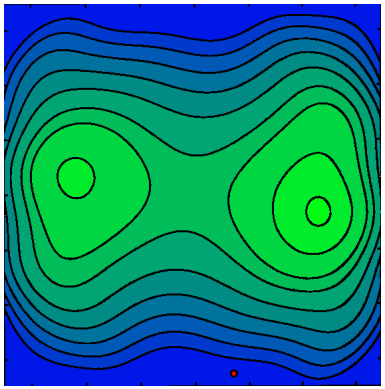
The Hong Kong Polytechnic University

Optimization 2023, Aveiro

Dedicated to the late Professor **M. J. D. Powell** FRS (1936–2015)

Funding: Hong Kong RGC grants 253012/17P, 153054/20P, and 153066/21P.

Trust-region DFO methods based on interpolation models

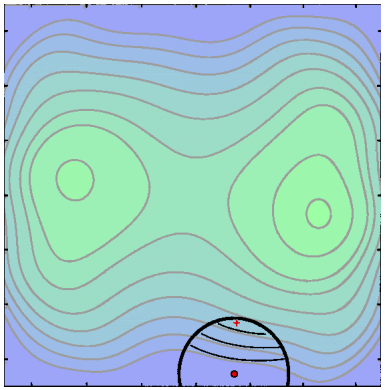


$$x_{k+1} \approx x_k + \underset{\|d\| \leq \Delta_k}{\arg\min}\, M_k(x_k + d)$$

- M_k is the trust-region model (surrogate)
  - $M_k(x) \approx f(x)$ around x_k
  - M_k interpolates f on a set $\mathcal{X}_k$ consisting of previous iterates
- $\|d\| \leq \Delta_k$ is the trust-region constraint
  - If "things work well", increase Δ_k
  - Otherwise, decrease Δ_k

Images by Dr. F. V. Berghen from `http://www.applied-mathematics.net`.

Trust-region DFO methods based on interpolation models

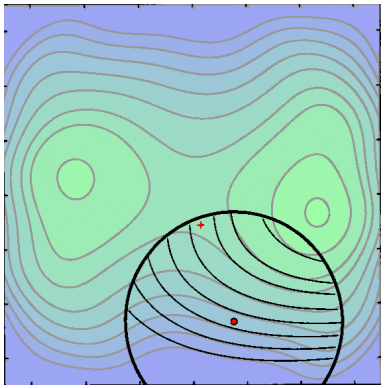


$$x_{k+1} \approx x_k + \underset{\|d\| \leq \Delta_k}{\arg\min}\, M_k(x_k + d)$$

- M_k is the trust-region model (surrogate)
  - $M_k(x) \approx f(x)$ around x_k
  - M_k interpolates f on a set $\mathcal{X}_k$ consisting of previous iterates
- $\|d\| \leq \Delta_k$ is the trust-region constraint
  - If "things work well", increase Δ_k
  - Otherwise, decrease Δ_k

Images by Dr. F. V. Berghen from `http://www.applied-mathematics.net`.

Trust-region DFO methods based on interpolation models

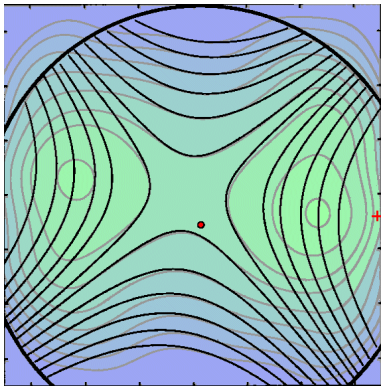


$$x_{k+1} \approx x_k + \underset{\|d\| \leq \Delta_k}{\arg\min}\, M_k(x_k + d)$$

- M_k is the trust-region model (surrogate)
  - $M_k(x) \approx f(x)$ around x_k
  - M_k interpolates f on a set $\mathcal{X}_k$ consisting of previous iterates
- $\|d\| \leq \Delta_k$ is the trust-region constraint
  - If "things work well", increase Δ_k
  - Otherwise, decrease Δ_k

Images by Dr. F. V. Berghen from `http://www.applied-mathematics.net`.

Trust-region DFO methods based on interpolation models

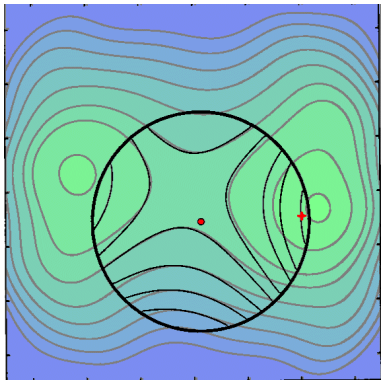


$$x_{k+1} \approx x_k + \underset{\|d\| \leq \Delta_k}{\arg\min}\, M_k(x_k + d)$$

- M_k is the trust-region model (surrogate)
  - $M_k(x) \approx f(x)$ around x_k
  - M_k interpolates f on a set $\mathcal{X}_k$ consisting of previous iterates
- $\|d\| \leq \Delta_k$ is the trust-region constraint
  - If "things work well", increase Δ_k
  - Otherwise, decrease Δ_k

Images by Dr. F. V. Berghen from `http://www.applied-mathematics.net`.

Trust-region DFO methods based on interpolation models



$$x_{k+1} \approx x_k + \underset{\|d\| \leq \Delta_k}{\arg\min}\, M_k(x_k + d)$$

- M_k is the trust-region model (surrogate)
  - $M_k(x) \approx f(x)$ around x_k
  - M_k interpolates f on a set $\mathcal{X}_k$ consisting of previous iterates
- $\|d\| \leq \Delta_k$ is the trust-region constraint
  - If "things work well", increase Δ_k
  - Otherwise, decrease Δ_k

Images by Dr. F. V. Berghen from `http://www.applied-mathematics.net`.

Maintenance of the interpolation set

The interpolation set $\mathcal{X}_k$ must be updated with care.

- $\mathcal{X}_k$ must reuse previous iterates as much as possible.
- The geometry of $\mathcal{X}_k$ must ensure the well-conditioning of the problem

$$M_k(x) = f(x), \quad x \in \mathcal{X}_k.$$

- Normally, $\mathcal{X}_{k+1} = (\mathcal{X}_k \cup \{x_{k+1}\}) \setminus \{\text{a "bad" point}\}$.
- Take geometry-improving steps if the geometry of $\mathcal{X}_k$ deteriorates.

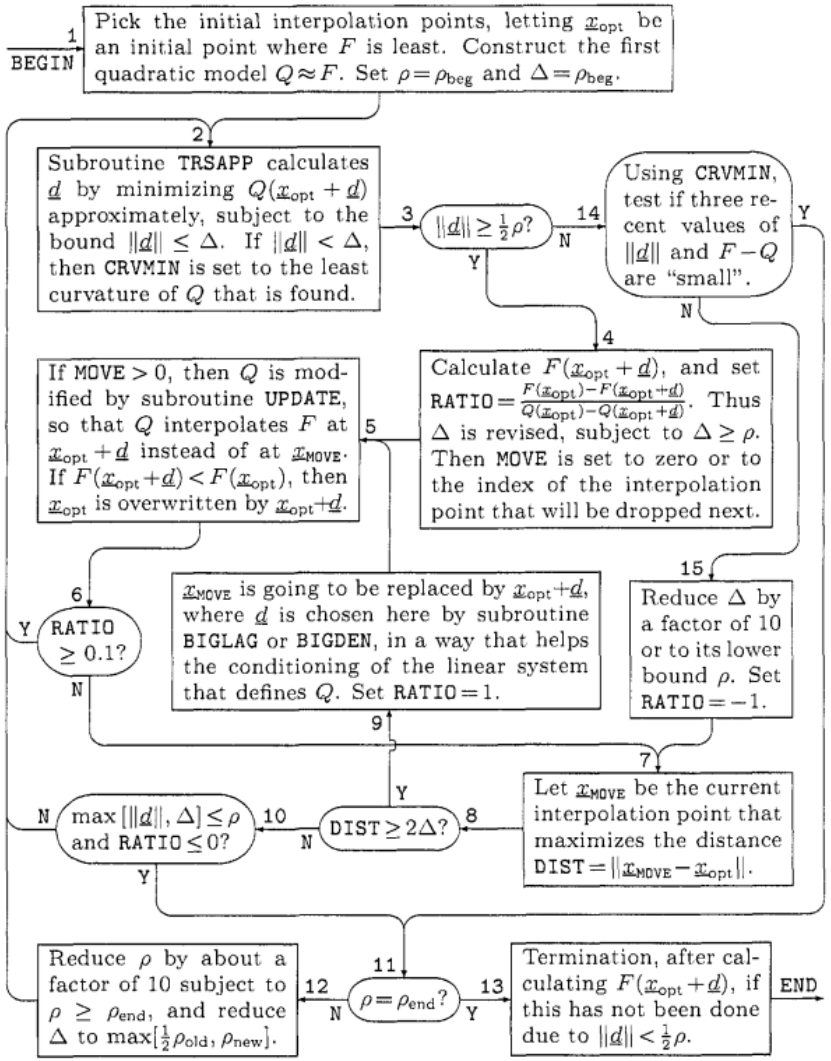
Powell's trust-region DFO algorithms and software

- COBYLA: solving general nonlinearly constrained problems using linear models; code released in 1992; paper published in 1994
- UOBYQA: solving unconstrained problems using quadratic models; code released in 2000; paper published in 2002
- NEWUOA: solving unconstrained problems using quadratic models; code released in 2004; paper published in 2006
- BOBYQA: solving bound-constrained problems using quadratic models; code released and paper written in 2009
- LINCOA: solving linearly constrained problems using quadratic models; code released in 2013 but no paper written

☞ COBYQA is the successor of COBYLA using quadratic models. Visit:

cobyqa.com

What do these algorithms look like?



NEWUOA

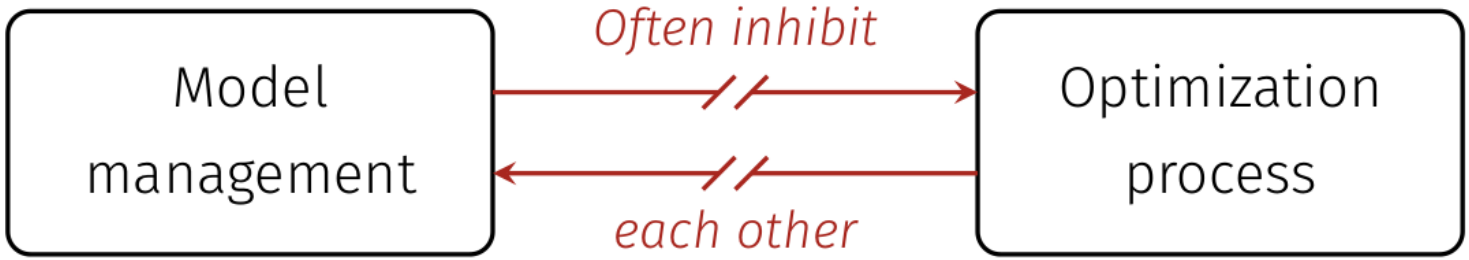
Implementation of these methods is HARD

> The development of NEWUOA has taken nearly three years. The work was very frustrating ...
>
> — M. J. D. Powell
>
> The NEWUOA software for unconstrained optimization without derivatives, 2006

N.B.

- NEWUOA was Powell's third trust-region DFO solver, COBYLA and UOBYQA being the first two.
- Mathematically speaking, NEWUOA and UOBYQA are essentially the same except for the ways they construct the model.
- Given the experience with UOBYQA (and COBYLA), Powell still spent three frustrating years on the development of NEWUOA.

The central difficulty



Powell's implementation

- Powell implemented these five methods into publicly available solvers.
- The solvers are widely used by scientists and engineers.
- They are often used as benchmarks when designing new algorithms.
- However, the implementation was in Fortran 77, with plenty of GOTOs: in total, 7939 lines of code with 249 GOTOs!

A modernized implementation is greatly needed.

Why should I work on a modernized implementation

- Professor Powell, April 2015: "It would be a relief to me if you would kindly continue to look after my optimisation software (NEWUOA, BOBYQA and LINCOA). Also I would like you to add COBYLA and TOLMIN if you do not have them already."
- Stefan Wild, ICCOPT 2016, Tokyo: People do not want interfaces. They want implementations that they can understand and play with.
- Jeff Larson, ISMP 2018, Bordeaux: Numerical linear algebra people have standard implementations for standard algorithms, e.g., LAPACK, whereas we all work on our own implementation of interpolation, model improvement, ...

Isn't it a perfect project for an engineer or a student?

- Given that Powell spent three frustrating years on the development of his own algorithm NEWUOA despite his abundant experience, where could I find this genius engineer who can learn all the five algorithms from scratch and implement them in a reasonable amount of time?
- Assume that I am lucky to find the abovementioned genius engineer and he/she happens to be my Ph.D. student. How should I persuade him/her to be fully devoted to a project for three years (as I did) without producing a single publication? Am I even allowed to do so?

PRIMA

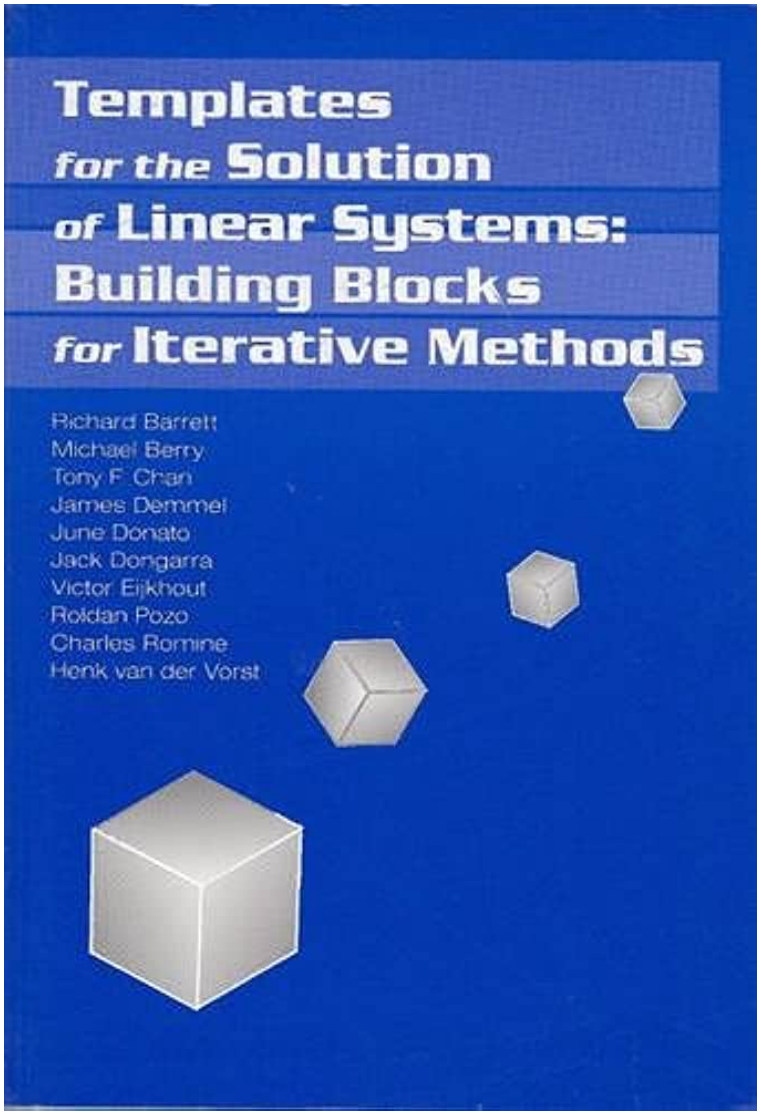
libprima.net

PRIMA is an acronym for

"Reference Implementation for Powell's Methods
with Modernization and Amelioration",

"P" for Powell.

An inspiration



An overview of PRIMA

- The solvers are implemented in a structured and modularized way so that they are understandable, maintainable, extendable, fault-tolerant, and future-proof.
- The code has no GOTO and uses matrix-vector procedures instead of loops whenever possible.
- The implementation is mathematically equivalent to Powell's except for the bug fixes and improvements we introduce intentionally.
- The implementation of PRIMA in modern Fortran (F2008 or above) has been finished.
- Versions in MATLAB, Python, Julia, R, ... will be implemented using the modern Fortran as a reference.
- A MATLAB interface is provided to use the modern Fortran version.
- The inclusion of PRIMA into SciPy is under discussion, and the major SciPy maintainers are positive about it.

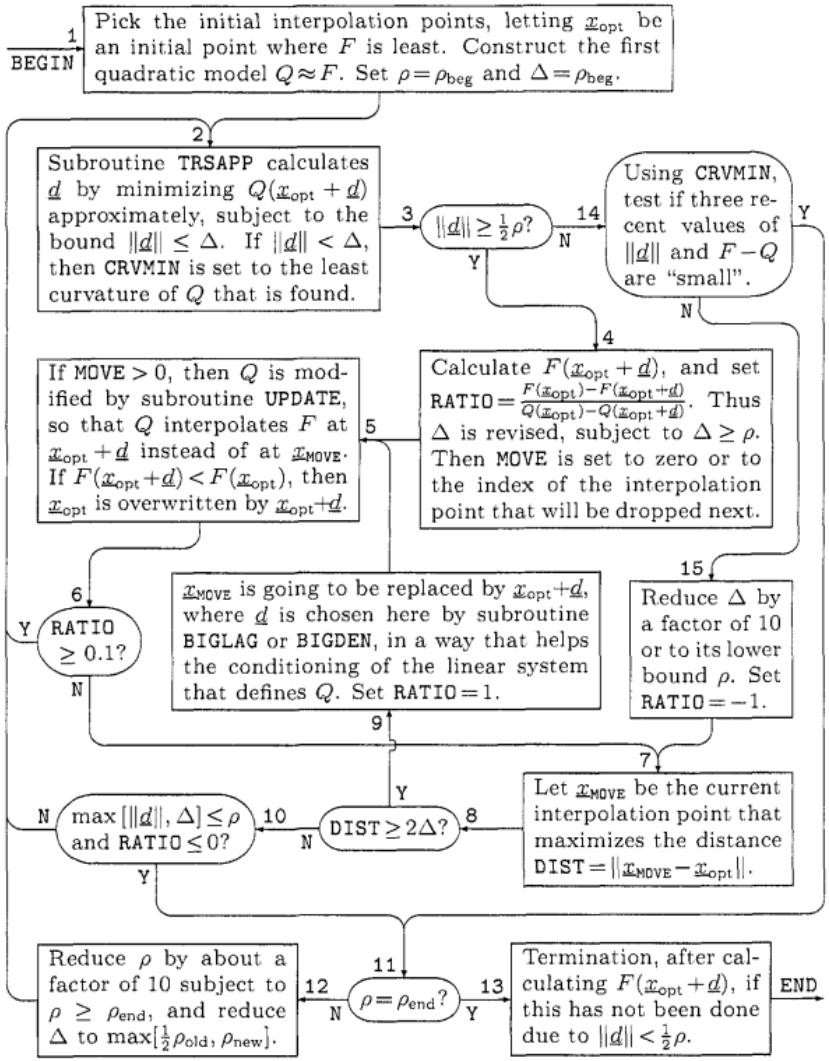
Why do I start with modern Fortran?

Fortran? Are you a caveman?

- The syntax and style of modern Fortran are very similar to MATLAB.
- I start with modern Fortran, so that I can systematically verify the bit-to-bit faithfulness of PRIMA, as the original code is Fortran.
- With other languages, the verification is hard, if not impossible.

Ultimate goal of PRIMA:
Make Powell's methods available to everyone in her/his favorite languages.

Powell's description of NEWUOA (recapped)



NEWUOA

The original implementation of NEWUOA: a snippet

```
117
118   100 KNEW=0
119       CALL TRSAPP (N,NPT,XOPT,XPT,GQ,HQ,PQ,DELTA,D,W,W(NP),
120      1  W(NP+N),W(NP+2*N),CRVMIN)
121       DSQ=ZERO
122       DO 110 I=1,N
123   110 DSQ=DSQ+D(I)**2
124       DNORM=DMIN1(DELTA,DSQRT(DSQ))
125       IF (DNORM .LT. HALF*RHO) THEN
126           KNEW=-1
127           DELTA=TENTH*DELTA
128           RATIO=-1.0D0
129           IF (DELTA .LE. 1.5D0*RHO) DELTA=RHO
130           IF (NF .LE. NFSAV+2) GOTO 460
131           TEMP=0.125D0*CRVMIN*RHO*RHO
132           IF (TEMP .LE. DMAX1(DIFFA,DIFFB,DIFFC)) GOTO 460
133           GOTO 490
134       END IF
135
136   120 IF (DSQ .LE. 1.0D-3*XOPTSQ) THEN
137           TEMPQ=0.25D0*XOPTSQ
138           DO 140 K=1,NPT
139           SUM=ZERO
140           DO 130 I=1,N
141   130     SUM=SUM+XPT(K,I)*XOPT(I)
142           TEMP=PQ(K)*SUM
143           SUM=SUM-HALF*XOPTSQ
144           W(NPT+K)=SUM
145           DO 140 I=1,N
146           GQ(I)=GQ(I)+TEMP*XPT(K,I)
147           XPT(K,I)=XPT(K,I)-HALF*XOPT(I)
148           VLAG(I)=BMAT(K,I)
149           W(I)=SUM*XPT(K,I)+TEMPQ*XOPT(I)
150           IP=NPT+I
151           DO 140 J=1,I
152   140     BMAT(IP,J)=BMAT(IP,J)+VLAG(I)*W(J)+W(I)*VLAG(J)
153
```

Faithful pseudocode of NEWUOA in PRIMA

Pick $\mathcal{X} \subset \mathbb{R}^n$ and $\rho > 0$. Let M interpolate f on $\mathcal{X}$. $x_{\mathsf{o}} := \arg\min_{x \in \mathcal{X}} f(x)$. $\Delta := \rho$.

while not converged **do**
  Calculate a trust-region trial point $x_{\mathsf{tr}} \approx \arg\min\{M(x) : \|x - x_{\mathsf{o}}\| \le \Delta\}$
  if $M(x_{\mathsf{o}}) - M(x_{\mathsf{tr}})$ is too small or $\|x_{\mathsf{tr}} - x_{\mathsf{o}}\|$ is too short **then**
    Reduce Δ subject to $\Delta \ge \rho$
  else
    Evaluate the reduction ratio and update Δ accordingly subject to $\Delta \ge \rho$
    if it is proper to replace a point $x_{\mathsf{drop}} \in \mathcal{X}$ with x_{tr} **then**
      Set $\mathcal{X} = \big(\mathcal{X} \cup \{x_{\mathsf{tr}}\}\big) \setminus \{x_{\mathsf{drop}}\}$, and then update M and x_{o}
    end if
  end if
  `improve_geo` := x_{tr} is bad & the geometry of $\mathcal{X}$ is inadequate
  `reduce_rho` := x_{tr} is bad & the geometry of $\mathcal{X}$ is adequate & Δ is small
  if `improve_geo` **then**
    Decide a point $x_{\mathsf{drop}} \in \mathcal{X}$ to drop and a geometry-improving point x_{geo}
    Set $\mathcal{X} = \big(\mathcal{X} \setminus \{x_{\mathsf{drop}}\}\big) \cup \{x_{\mathsf{geo}}\}$, and then update M and x_{o}
  end if
  if `reduce_rho` **then** reduce ρ and reduce Δ subject to $\Delta \ge \rho$
end while

N.B.: The updates keep M interpolating f on $\mathcal{X}$ and $x_{\mathsf{o}} = \arg\min_{x \in \mathcal{X}} f(x)$.

PRIMA NEWUOA: trust-region phase (ln. 2–10)

```
161
162 do tr = 1, maxtr
163     call trsapp(delta, gopt, hq, pq, trtol, xpt, crvmin, d)
164     dnorm = min(delta, norm(d))
165     shortd = (dnorm < HALF * rho)
166     qred = -quadinc(d, xpt, gopt, pq, hq)
167
168     if (shortd .or. .not. qred > 0) then
169         delta = TENTH * delta
170         if (delta <= gamma3 * rho) then
171             delta = rho  ! Set DELTA to RHO when it is close to or below.
172         end if
173     else
174         x = xbase + (xpt(:, kopt) + d)
175         call evaluate(calfun, x, f)
176         nf = nf + 1
177
178         dnorm_rec = [dnorm_rec(2:size(dnorm_rec)), dnorm]
179         moderr = f - fval(kopt) + qred
180         moderr_rec = [moderr_rec(2:size(moderr_rec)), moderr]
181
182         ratio = redrat(fval(kopt) - f, qred, eta1)
183         delta = trrad(delta, dnorm, eta1, eta2, gamma1, gamma2, ratio)
184         if (delta <= gamma3 * rho) then
185             delta = rho  ! Set DELTA to RHO when it is close to or below.
186         end if
187
188         ximproved = (f < fval(kopt))
189         knew_tr = setdrop_tr(idz, kopt, ximproved, bmat, d, delta, rho, xpt, zmat)
190         if (knew_tr > 0) then
191             xdrop = xpt(:, knew_tr)
192             xosav = xpt(:, kopt)
193             call updateh(knew_tr, kopt, d, xpt, idz, bmat, zmat)
194             call updatexf(knew_tr, ximproved, f, xosav + d, kopt, fval, xpt)
195             call updateq(idz, knew_tr, ximproved, bmat, d, moderr, xdrop, xosav, xpt, zmat, gopt, hq, pq)
196             call tryqalt(idz, bmat, fval - fval(kopt), ratio, xpt(:, kopt), xpt, zmat, itest, gopt, hq, pq)
197         end if
198     end if  ! End of IF (SHORTD .OR. .NOT. QRED > 0). The normal trust-region calculation ends here.
199
```

PRIMA NEWUOA: improve_geo, reduce_rho (ln. 11, 12)

```
199
200     accurate_mod = all(abs(moderr_rec) <= 0.125 * crvmin * rho**2) .and. all(dnorm_rec <= rho)
201     distsq = sum((xpt - spread(xpt(:, kopt), dim=2, ncopies=npt))**2, dim=1)
202     close_itpset = all(distsq <= 4.0 * delta**2)
203     adequate_geo = (shortd .and. accurate_mod) .or. close_itpset
204     small_trrad = (max(delta, dnorm) <= rho)
205     bad_trstep = (shortd .or. (.not. qred > 0) .or. ratio <= eta1 .or. knew_tr == 0)
206     improve_geo = bad_trstep .and. .not. adequate_geo
207     bad_trstep = (shortd .or. (.not. qred > 0) .or. ratio <= 0 .or. knew_tr == 0)
208     reduce_rho = bad_trstep .and. adequate_geo .and. small_trrad
209
```

PRIMA NEWUOA: post-processing phase (ln. 13–17)

```
209
210     if (improve_geo) then
211         knew_geo = int(maxloc(distsq, dim=1), kind(knew_geo))
212         delbar = max(min(TENTH * sqrt(maxval(distsq)), HALF * delta), rho)
213         d = geostep(idz, knew_geo, kopt, bmat, delbar, xpt, zmat)
214         x = xbase + (xpt(:, kopt) + d)
215         call evaluate(calfun, x, f)
216         nf = nf + 1
217
218         dnorm = min(delbar, norm(d))
219         dnorm_rec = [dnorm_rec(2:size(dnorm_rec)), dnorm]
220         moderr = f - fval(kopt) - quadinc(d, xpt, gopt, pq, hq)
221         moderr_rec = [moderr_rec(2:size(moderr_rec)), moderr]
222
223         ximproved = (f < fval(kopt))
224         xdrop = xpt(:, knew_geo)
225         xosav = xpt(:, kopt)
226         call updateh(knew_geo, kopt, d, xpt, idz, bmat, zmat)
227         call updatexf(knew_geo, ximproved, f, xosav + d, kopt, fval, xpt)
228         call updateq(idz, knew_geo, ximproved, bmat, d, moderr, xdrop, xosav, xpt, zmat, gopt, hq, pq)
229     end if  ! End of IF (IMPROVE_GEO). The procedure of improving geometry ends.
230
231     if (reduce_rho) then
232         if (rho <= rhoend) then
233             info = SMALL_TR_RADIUS
234             exit
235         end if
236         delta = HALF * rho
237         rho = redrho(rho, rhoend)
238         delta = max(delta, rho)
239         dnorm_rec = REALMAX
240         moderr_rec = REALMAX
241     end if  ! End of IF (REDUCE_RHO). The procedure of reducing RHO ends.
242
243     if (sum(xpt(:, kopt)**2) >= 1.0E2 * delta**2) then  ! 1.0E2 works better than 1.0E3 on 20230227.
244         call shiftbase(kopt, xbase, xpt, zmat, bmat, pq, hq, idz)
245     end if
246 end do  ! End of DO TR = 1, MAXTR. The iterative procedure ends.
247
```

Issues in the Fortran 77 implementation: an example

```
C
C     If KNEW is zero initially, then pick the index of the interpolation
C       point to be deleted, by maximizing the absolute value of the
C       denominator of the updating formula times a weighting factor.
C
C
      IF (KNEW .EQ. 0) THEN
          DENMAX=ZERO
          DO 100 K=1,NPT
          HDIAG=ZERO
          DO 80 J=1,NPTM
          TEMP=ONE
          IF (J .LT. IDZ) TEMP=-ONE
   80     HDIAG=HDIAG+TEMP*ZMAT(K,J)**2
          DENABS=DABS(BETA*HDIAG+VLAG(K)**2)
          DISTSQ=ZERO
          DO 90 J=1,N
   90     DISTSQ=DISTSQ+(XPT(K,J)-XPT(KOPT,J))**2
          TEMP=DENABS*DISTSQ*DISTSQ
          IF (TEMP .GT. DENMAX) THEN
              DENMAX=TEMP
              KNEW=K
          END IF
  100     CONTINUE
      END IF
C
C     Apply the rotations that put zeros in the KNEW-th row of ZMAT.
C
      JL=1
      IF (NPTM .GE. 2) THEN
          DO 120 J=2,NPTM
          IF (J .EQ. IDZ) THEN
              JL=IDZ
          ELSE IF (ZMAT(KNEW,J) .NE. ZERO) THEN
```

The above code may crash, as `KNEW` may be used uninitialized.

Issues in the Fortran 77 implementation

- The Fortran 77 solvers may crash with memory violations (segfaults).
  Reason: Some indices are only initialized under conditions that can never be met because of NaN resulted from floating point exceptions.
- The Fortran 77 solvers may get stuck in infinite loops.
  Reason: Some loops are only terminated under conditions that can never be met because of NaN resulted from floating point exceptions.

N.B.:

- The problems are due to floating point exceptions in the Fortran 77 code rather than flaws in the algorithms.
- The problems affect all implementations or wrappers of these solvers based on the Fortran 77 code, including SciPy (COBYLA), NLopt, ...

How to ensure PRIMA does not have similar issues?

Strategy 1. Programming by contract

```
120
121 ! Preconditions
122 if (DEBUGGING) then
123     call assert(n >= 1 .and. npt >= n + 2, 'N >= 1, NPT >= N + 2', srname)
124     call assert(delta > 0, 'DELTA > 0', srname)
125     call assert(size(gopt_in) == n, 'SIZE(GOPT) = N', srname)
126     call assert(size(hq_in, 1) == n .and. issymmetric(hq_in), 'HQ is an NxN symmetric matrix', srname)
127     call assert(size(pq_in) == npt, 'SIZE(PQ) = NPT', srname)
128     call assert(all(is_finite(xpt)), 'XPT is finite', srname)
129     call assert(size(s) == n, 'SIZE(S) == N', srname)
130 end if
131
```

```
431
432 ! Postconditions
433 if (DEBUGGING) then
434     call assert(size(s) == n .and. all(is_finite(s)), 'SIZE(S) == N, S is finite', srname)
435     ! Due to rounding, it may happen that ||S|| > DELTA, but ||S|| > 2*DELTA is highly improbable.
436     call assert(norm(s) <= TWO * delta, '||S|| <= 2*DELTA', srname)
437     call assert(crvmin >= 0, 'CRVMIN >= 0', srname)
438 end if
439
```

- The preconditions and postconditions are checked only in the debug mode. In the code that users receive, they are disabled by default.
- In the debug mode, if some subroutine receives strange inputs or produces strange outputs, the program will raise an error so that the developer (i.e., Zaikun Zhang) can check the issue and fix it.

How to ensure PRIMA does not have similar issues?

Strategy 2. TOUGH (Tolerance Of Untamed and Genuine Hazards) tests

```
650
651 function f = noisy(f, x, noise_level)
652 if nargin < 3
653     noise_level = 2e-1;
654 end
655 r = cos(1.0D6 * sin(1.0D6 * (abs(f) + 1.0D0) * cos(1.0D6 * sum(abs(x)))));
656 f = f*(1+noise_level*r);
657 if (r > 0.9)
658     error('Function evaluation fails!');
659 elseif (r > 0.75)
660     f = inf;
661 elseif (r > 0.5)
662     f = NaN;
663 elseif (r < - 0.999)
664     f = -1e30;
665 end
666 return
667
```

- In TOUGH tests, objective functions are corrupted as above and then fed to the solvers.
- We make sure that PRIMA solvers work properly even if the objective functions are corrupted in this severe way. (What about your solvers?)

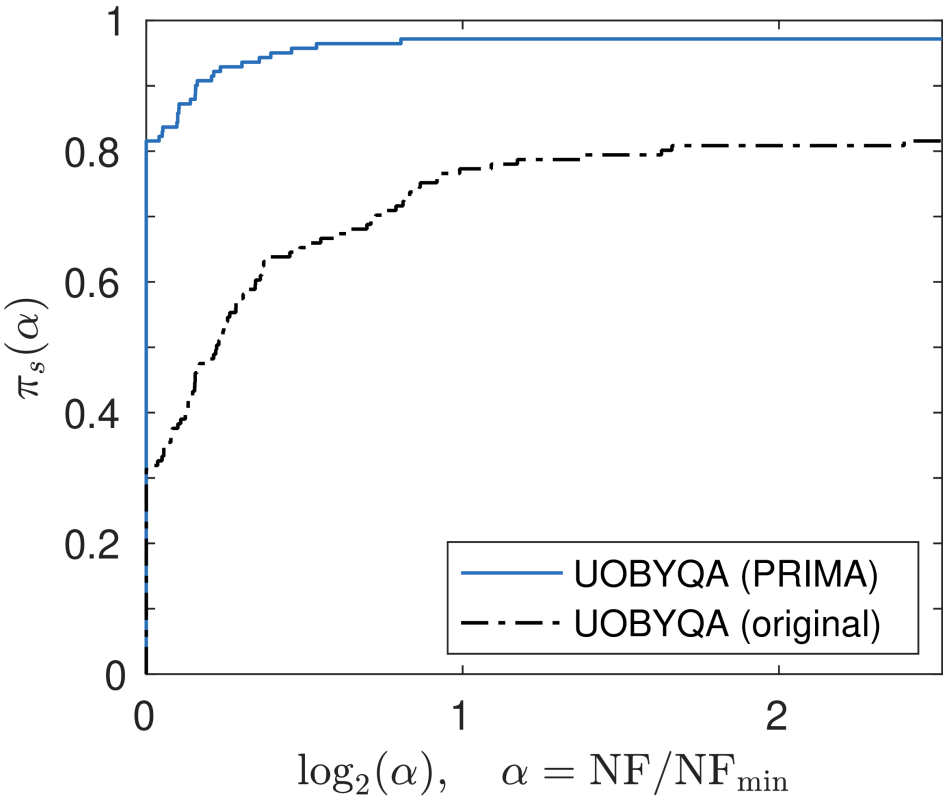
How to ensure PRIMA does not have similar issues?

Strategy 3. Automated and randomized tests using GitHub Actions

- Every day, extensive TOUGH tests and other tests are conducted automatically on randomized variants of CUTEst problems.
- The longer time passes, the more reliable PRIMA is, automatically.
- As of June 2023, $>$ 42,000 workflows have been successfully run.
- Each workflow consists of $\sim$ 5 (sometimes more than 150) randomized tests, each test taking from tens of minutes to several hours.
- In other words, PRIMA has been verified by more than 200,000 hours (or more than 20 years) of randomized tests.

Code must be battle-tested before it becomes software.

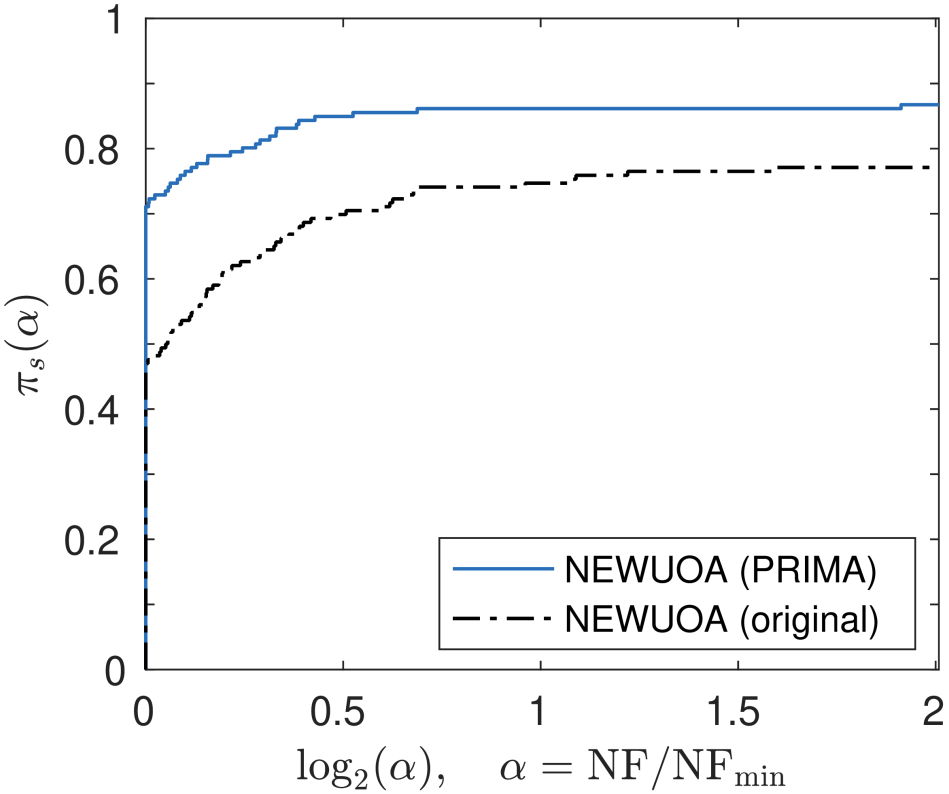
The performance of PRIMA



UOBYQA

(unconstrained problems, at most 100 variables)

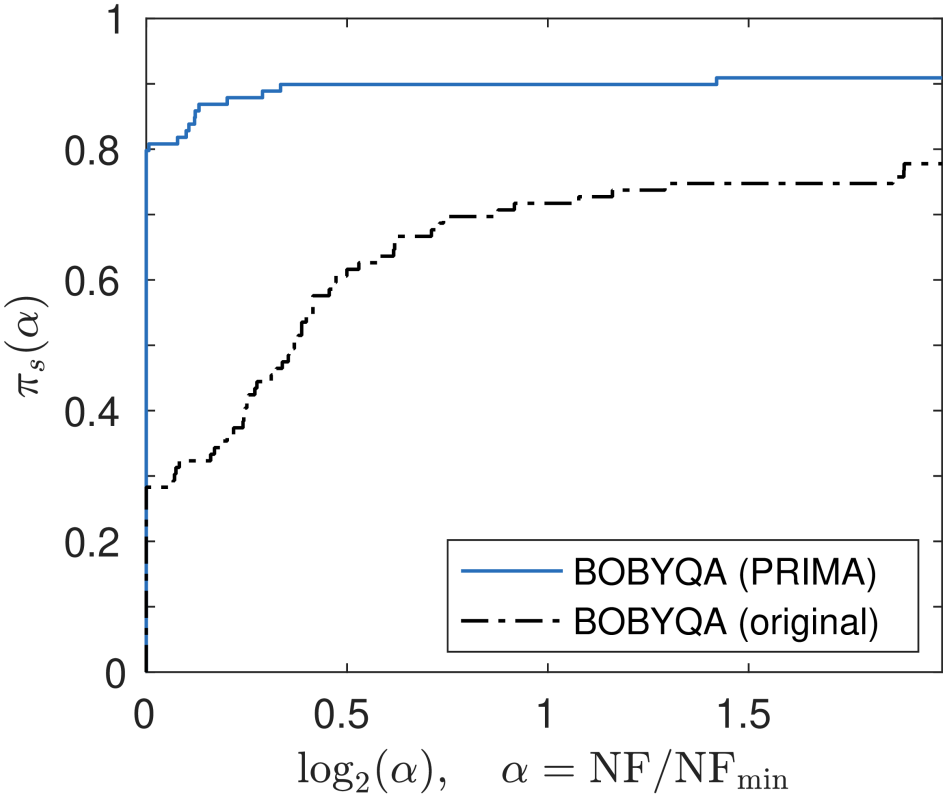
The performance of PRIMA



NEWUOA

(unconstrained problems, at most 200 variables)

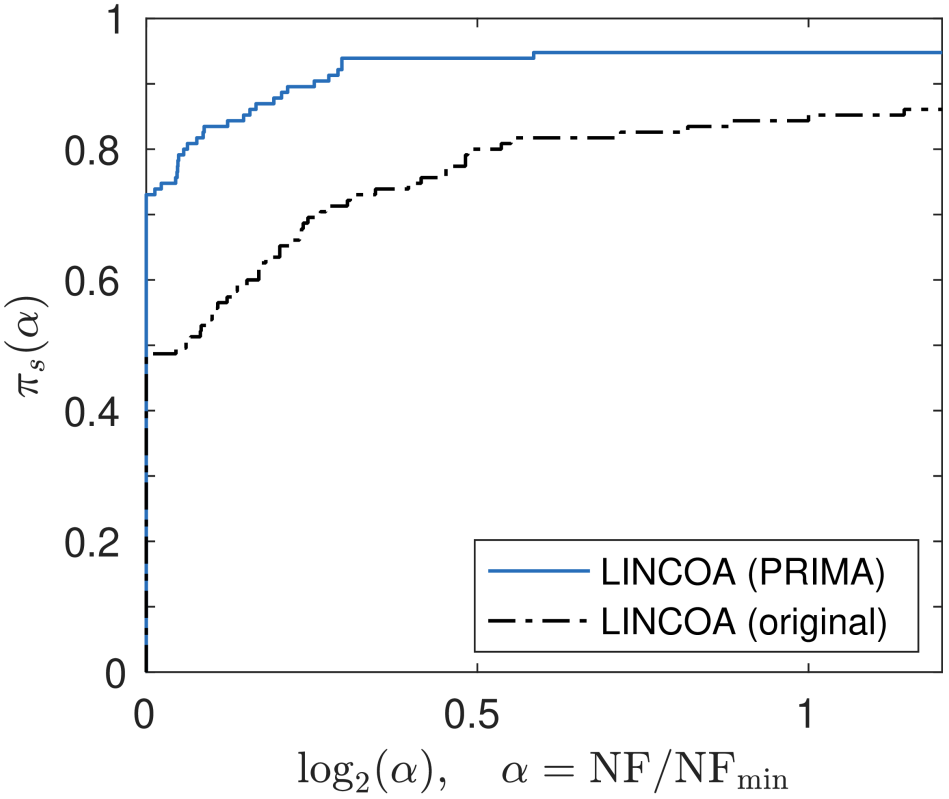
The performance of PRIMA



BOBYQA

(bound-constrained problems, at most 200 variables)

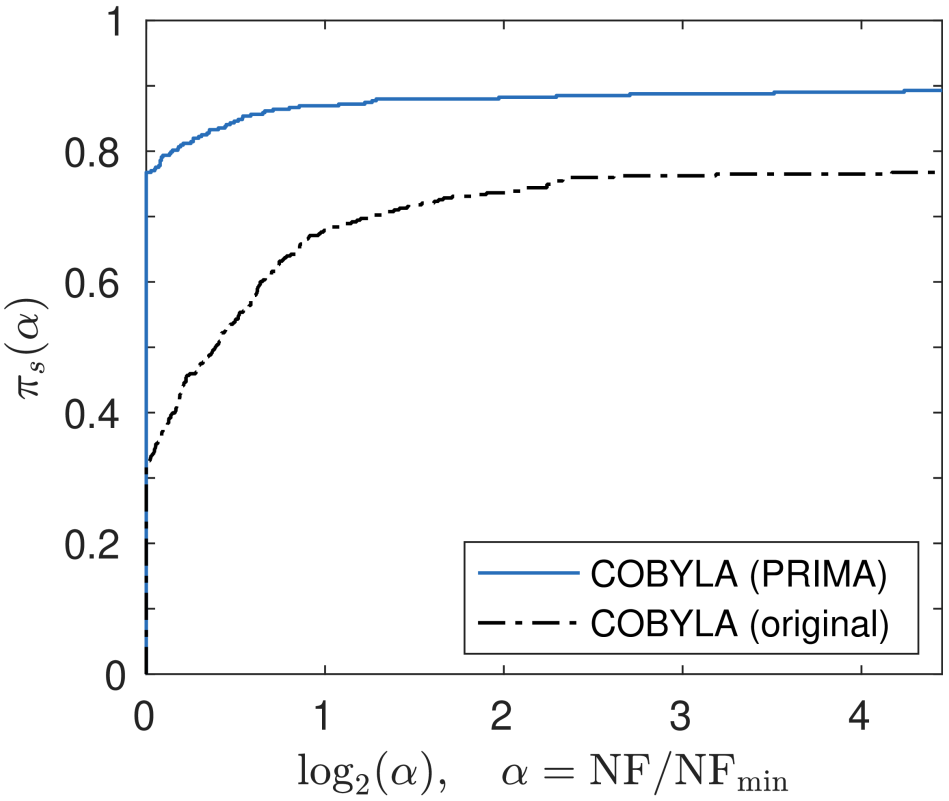
The performance of PRIMA



LINCOA

(linearly constrained problems, at most 200 variables, 20,000 constraints)

The performance of PRIMA



COBYLA

(nonlinearly constrained problems, at most 100 variables, 10,000 constraints)

How to ensure the improvements are not by luck?

Take COBYLA as an example.

The PRIMA implementation of COBYLA is tested on 359 nonlinearly constrained CUTEst problems with at most 100 variables and 10,000 constraints. Seven tests are made.

1. A plain test
2. A test that permutes the variables randomly
3. A test that perturbs the starting point randomly
4. A test based on single-precision objective & constraint values
5. A test using only 5 significant digits of the objective & constraints
6. A test contaminating the objective & constraints by deterministic noise
7. A test contaminating the objective & constraints by random noise

How to ensure the improvements are not by luck?

168 performance profiles of the new and old implementations of COBYLA

A “fun” fact ...

- Working on PRIMA, I have spotted a dozen of bugs in reputable Fortran compilers and two bugs in MATLAB.
- Each of them represents days of bitter debugging.
- From an unusual angle, they reflect how intensive the coding is.
- The bitterness behind this fun fact is exactly why I work on PRIMA:
  - I hope all the frustrations that I have experienced will not happen to any user of Powell's methods anymore.
  - I hope I am the last one in the world to decode a maze of 244 GOTOs in 7939 lines of Fortran 77 code — I did this for three years and I do not want anyone else to do it again.

But it is not quite rewarding in terms of career and life ...

- You may write 3 good papers in 1 year, but not 1 good package in 3 years, especially if you start with a nontrivial Fortran 77 codebase.
- Internet: "Writing software is a low-status academic activity."
- Internet: "A general problem is that ... professors are usually rewarded for publications, not their software."
- Comments on my grant proposal: The PI's expertise seems in software development, but he may not be a good mathematician.
- As a "not-so-good mathematician", I much prefer spending my time on proofs, which are a lot easier and much more enjoyable for me.
- Sometimes we do things that are not enjoyable but have to be done.
- Teaser: Who translated Euclid's *Elements* to modern languages? You probably do not know (and do not care).

Concluding remarks

- Implementation of model-based DFO solvers is intrinsically hard
- PRIMA provides the reference implementation of Powell's DFO solvers
- The modern Fortran version and a MATLAB interface is finished
- PRIMA will also be implemented in MATLAB, Python, Julia, R, C++
- PRIMA fixes issues in the original Fortran 77 code
- PRIMA is tested extensively to ensure its correctness & robustness
- PRIMA outperforms the original implementation of the solvers

`libprima.net`

Muito obrigado!