TRUST REGION OPTIMIZATION AT MIXED PRECISION

A PREPRINT

Anonymous Alpaca
Argonne National Laboratory
alpaca@anl.gov

May 6, 2021

ABSTRACT

We present a method to perform trust region based optimization for nonlinear unconstrained problems. The method selectively uses function and gradient evaluations at different floating point precisions to reduce the overall power consumption and time to solution.

***K*eywords** Optimization · Trust Region Methods · Mixed Precision

1 Introduction

Optimization methods are used in many applications, including engineering, science, and machine learning. The memory footprint and run time of optimization methods has been studied extensively and determines the problem sizes that can be run on existing hardware. Similarly, the power consumption of optimization methods determines their cost and carbon footprint, which is recently becoming a growing concern [1]. Previous work has found significant differences in the overall power consumption for double and single precision computations [2]. TODO: Plenty of work on power consumption overall, few specific to scientific computing (e.g. [3]), but there is not much about concrete factors between single/double.

Also TODO: Should we have an acronym, and if so, which one? Let's collect ideas here. Filler words like "for", "at", "with" can be added as needed.

- ~~TROMP - Trust Region Optimization Mixed Precision~~
- ~~TROUT - Trust Region Optimization UnconsTrained Problems~~
- TRUMP - Trust Region method Using Mixed Precision
- ~~UMMPA - Unconstrained Minimization Mixed Precision Arithmetic~~
- ~~TRUMPET - TRUst region Mixed Precision arithmETic~~

MM: Come on guys. Was this even a question?

2 Background

Related work [4].

Probably the most relevant related paper, need to be clear how we improve over their work [5].

3 Method

We can integrate text into earlier sections as required. I'm just trying to create consistent notation and a self-contained algorithm description for now.

We consider the unconstrained minimization of a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$,

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}). \tag{1}$$

We suppose that we have access to a hierarchy of arithmetic precisions for the evaluation of both $f(\mathbf{x})$ and $\nabla f(\mathbf{x})$, but the direct evaluation of $f(\mathbf{x}), \nabla f(\mathbf{x})$ is unavailable. We formalize this slightly by supposing we are given oracles that compute $f^p(\mathbf{x}), \nabla f^p(\mathbf{x})$ for $p \in \{1, \ldots, P\}$. At a majority of input $\mathbf{x}$, the relation

$$|f^p(\mathbf{x}) - f(\mathbf{x})| > |f^{p+1}(\mathbf{x}) - f(\mathbf{x})|, \quad \|\nabla f^p(\mathbf{x}) - \nabla f(\mathbf{x})\| > \|\nabla f^{p+1}(\mathbf{x}) - \nabla f(\mathbf{x})\|$$

is expected to hold for $p = 1, \ldots, P-1$. For a tangible example, if intermediate computations involved in the computation of $f(\mathbf{x})$ can be done in half, single, or double precision, then we can denote $f^1(\mathbf{x})$ as the oracle using only half precision computations, $f^2(\mathbf{x})$ as the oracle using only single precision, and $f^3(\mathbf{x})$ as the oracle using only double precision.

We now propose a trust-region method for the solution of (1). Trust-region methods are iterative methods for optimization, and we will denote the iteration counter by k. Given an *incumbent point* $\mathbf{x}_k$ in the kth iteration, a standard trust-region method builds some smooth model $m_k : \mathbb{R}^n \to \mathbb{R}$ of the function $\mathbf{s} \mapsto f(\mathbf{x}_k + \mathbf{s})$. The model m_k is intended to be "good" in a sense to be formalized later on the *trust-region* $\{\mathbf{s} \in \mathbb{R}^n : \|\mathbf{s}\| \leq \delta_k\}$ for $\delta_k > 0$, the *trust-region radius*. A *trial step* $\mathbf{s}_k$ is then computed via a(n approximate) solution to the *trust-region subproblem*

$$\min_{\mathbf{s}\in\mathbb{R}^n} m_k(\mathbf{s}) : \|\mathbf{s}\| \leq \delta_k. \tag{2}$$

We formalize in the appendix MM: TODO the condition imposed on the quality of the approximate solution $\mathbf{s}_k$ to (2), but remark now to the reader familiar with trust-region methods that it just a standard Cauchy decrease condition. Having computed $\mathbf{s}_k$, the standard trust-region method then compares the true decrease in the function obtained at the trial point $\mathbf{x}_k + \mathbf{s}_k$ compared to the incumbent point $\mathbf{x}_k$, $f(\mathbf{x}_k) - f(\mathbf{x} + \mathbf{s}_k)$ with the decrease predicted by the model, $m_k(\mathbf{0}) - m_k(\mathbf{s}_k)$. In particular, one computes the quantity

$$\rho_k = \frac{f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{s}_k)}{m_k(\mathbf{0}) - m_k(\mathbf{s}_k)}. \tag{3}$$

If ρ_k is sufficiently positive ($\rho_k > \eta_1$ for fixed $\eta_1 > 0$), then the algorithm accepts $\mathbf{x}_k + \mathbf{s}_k$ as the incumbent point $\mathbf{x}_{k+1}$ and may possibly increase the trust-region radius $\delta_k < \delta_{k+1}$ (if $\rho_k > \eta_2$ for fixed $\eta_2 \geq \eta_1$). This scenario is called a *successful iteration*. On the other hand, if ρ_k is not sufficiently positive (or is negative), then the incumbent point $\mathbf{x}_{k+1} = \mathbf{x}_k$ and $\delta_k > \delta_{k+1}$.

From this basic algorithm description, we can identify two additional difficulties presented in the multiple precision setting. First, it is presently unclear how to compute ρ_k in (3), since we assumed that we have no access to an oracle that directly computes $f(\cdot)$. Second, because models m_k are typically constructed from function and gradient information provided by $f(\cdot), \nabla f(\cdot)$, we must identify appropriate conditions on m_k.

For the first of these issues, we make a practical assumption that **the highest level of precision available to us should be treated as if it were real precision.** Although this is a theoretically poor assumption, virtually all computational optimization makes this assumption implicitly; optimization methods are typically analyzed as if computation can be done in real precision, but are only ever implemented at some level of arithmetic precision (typically double). Thus, in the notation we have developed, the optimization problem we actually aim to solve is not (1), but

$$\min_{\mathbf{x}\in\mathbb{R}^n} f^P(\mathbf{x}) \tag{4}$$

so that the ρ-test in (3) is replaced with

$$\rho_k = \frac{f^P(\mathbf{x}_k) - f^P(\mathbf{x}_k + \mathbf{s}_k)}{m_k(\mathbf{0}) - m_k(\mathbf{s}_k)} = \frac{\text{ared}_k}{\text{pred}_k}. \tag{5}$$

We introduced ared to denote "actual reduction" and pred to denote "predicted reduction". However, computing (5) still entails two evaluations of the highest-precision oracle $f^P(\cdot)$, and the intention of using mixed precision was to avoid exactly this behavior. Our algorithm avoids the cost of full-precision evaluations by dynamically adjusting the precision level $p_k \in \{1, \ldots, P\}$ between iterations so that in the kth iteration, ρ_k is approximated by

$$\tilde{\rho}_k = \frac{f^{p_k}(\mathbf{x}_k) - f^{p_k}(\mathbf{x}_k + \mathbf{s}_k)}{m_k(\mathbf{0}) - m_k(\mathbf{s}_k)} = \frac{\text{ered}_k}{\text{pred}_k}, \tag{6}$$

introducing ered to denote "estimated reduction". To update p_k, we are motivated by a strategy similar to one employed in [6, 7]. We initialize $p_1 = 1$. We introduce a variable θ_k, which is not properly initialized until the end of the first unsuccessful iteration. When the first unsuccessful iteration is encountered, we set

$$\theta_k \leftarrow |\text{ared}_k - \text{ered}_k|. \tag{7}$$

```
Initialize $0 < \eta_1 \leq \eta_2 < 1$, $\omega \in (0,1)$, $\gamma_{\text{inc}} > 1$, $\gamma_{\text{dec}} \in (0,1)$, forcing sequence $\{r_k\}$.
Choose initial $\delta_1 > 0$, $\mathbf{x}_1 \in \mathbb{R}^n$.
$\theta_1 \leftarrow 0, p_k \leftarrow 1, k \leftarrow 1, \text{failed} \leftarrow \text{FALSE}$
while some stopping criterion not satisfied do
    Construct model $m_k$.
    (Approximately solve) (2) to obtain $\mathbf{s}_k$.
    Compute $\tilde{\rho}_k$ as in (6).
    if $\tilde{\rho}_k > \eta_1$ (successful iteration) then
        $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \mathbf{s}_k$.
        if $\tilde{\rho}_k > \eta_2$ (very successful iteration) then
            $\delta_{k+1} \leftarrow \gamma_{\text{inc}}\delta_k$.
        end
    else
        if not failed then
            Compute $\theta_k$ as in (7).
            failed $\leftarrow$ TRUE.
        end
        if (8) holds then
            $\delta_{k+1} \leftarrow \gamma_{\text{dec}}\delta_k$.
        else
            $p_{k+1} \leftarrow p_k + 1$.
            Compute $\theta_k$ as in (7).
            $\delta_{k+1} \leftarrow \delta_k$.
        end
        $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k$.
    end
    $k \leftarrow k + 1$.
end
```

Algorithm 1: Trust-Region method Using Mixed Precision (TRUMP)

Notice that this means that we must incur the cost of two evaluations of $f^P(\cdot)$ following the first unsuccessful iteration in order to compute ared_k. From that point on in the algorithm, θ_k is involved in a test to determine if the precision level p_k should be increased. Introducing a predetermined *forcing sequence* $\{r_k\}$ satisfying $r_k \in [0,\infty)$ for all k and $\lim_{k\to\infty} r_k = 0$, and fixing a parameter $\omega \in (0,1)$, we check on any unsuccessful iteration whether

$$\theta_k^\omega \leq \eta \min\{\text{pred}_k, r_k\} \tag{8}$$

where $\eta = \min\{\eta_1, 1-\eta_2\}$. If (8) does not hold, then we increase $p_{k+1} = p_k + 1$ and again update the value of θ_k according to (7) (thus incurring two more evaluations of $f^P(\cdot)$).

It remains to describe how we deal with our second identified difficulty, the construction of m_k in the absence of evaluations of $f(\cdot), \nabla f(\cdot)$. As is frequently done in trust-region methods, we will employ quadratic models of the form

$$m_k(\mathbf{s}) = f_k + \mathbf{g}_k^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top \mathbf{H}_k \mathbf{s}. \tag{9}$$

Having already defined rules for the update of p_k through the test (8), we take in the kth iteration $f_k = f^{p_k}(\mathbf{x})$ and $\mathbf{g}_k = \nabla f^{p_k}(\mathbf{x})$. In theory, we require $\mathbf{H}_k$ to be any Hessian approximation with a spectrum bounded above and below uniformly for all k. In practice, we update $\mathbf{H}_k$ via limited memory SR1 updates, with the caveat that on all iterations where $p_{k+1} = p_k + 1$, all point/gradient displacement pairs are deleted from memory, effectively restarting the approximation of $\mathbf{H}_k$.

Pseudocode for our algorithm is provided in Algorithm 1.

4 Implementation

JH: TODO: describe briefly the way CUTEst was made to work for us.

MM: We can release code if you think it's appropriate. However, reproducing our experiments requires the user to have their CUTEst (and less obnoxiously, preducer) environment set up correctly... Algorithm 1 was implemented in Julia. Observe that if we ignore the if conditional on the variable failed, and if we set $\eta = \infty$, then

Algorithm 1 reduces to a standard trust-region method. This observation results in a set of meaningful comparators for the performance of Algorithm 1; we can run the standard trust-region method with all the remaining parameters set to the same values as those used in Algorithm 1 and perform every function/gradient evaluation at a fixed level of precision, $P = 1$. We will do this in our numerical experiments.

5 Test Problems

JH: TODO: Describe the CUTEst problems

6 Experimental Results

MM: Will we have other functions besides the CUTEst functions that allow for more variable precision? Even if it's just one or two simple hand-coded functions that can use the variable precision datatypes, that would make a stronger case for this paper. CUTEst is just nice because every optimizer worth their salt knows what CUTEst is

Due to the challenges presented by CUTEst, resulting in only being able to evaluate CUTEst test functions in single or double precision, Algorithm 1 is limited to $P = 2$. Thus, we compare Algorithm 1 (TRUMP) to a trust-region method with access only to single-precision function evaluations (TR-single) and a trust-region method with access only to double-precision function evaluations (TR-double). We selected 100 test problems from CUTEst that had between 2 and 10 variables. MM: We can do bigger problems later, no worries. We display results across this problem set using data and performance profiles [8, 9], but we use a particular notion of the cost of function evaluations. Motivated by the computational models in MM: citations needed please!, we assume that if implemented properly, a function evaluation performed in single precision costs 25% of the total energy of the same function evaluation performed in double precision. This is because heat generated and energy spent is proportional to the total surface area of the chip ... MM: Please correct language/understanding. Thus, when profiling, we normalize the cost of a double precision evaluation as one function evaluation, and suppose that a single precision evaluation is 0.25 evaluations. Results are shown in Figure **??**, both in terms of the value of $f(x)$ returned and the size of the gradient norm $\|\nabla f(x)\|$.

7 Conclusion, Future Work

References

[1] Karen Hao. Training a single ai model can emit as much carbon as five cars in their lifetimes. *MIT Technology Review*, 2019.

[2] Daniel Molka, Daniel Hackenberg, Robert Schöne, and Matthias S Müller. Characterizing the energy consumption of data transfers and arithmetic operations on x86- 64 processors. In *International conference on green computing*, pages 123–133. IEEE, 2010.

[3] Gokcen Kestor, Roberto Gioiosa, Darren J Kerbyson, and Adolfy Hoisie. Quantifying the energy cost of data movement in scientific applications. In *2013 IEEE international symposium on workload characterization (IISWC)*, pages 56–65. IEEE, 2013.

[4] Robert Strzodka and Dominik Göddeke. Mixed precision methods for convergent iterative schemes. *EDGE*, 6:23–24, 2006.

[5] Serge Gratton and Ph L Toint. A note on solving nonlinear optimization problems in variable precision. *arXiv preprint arXiv:1812.03467*, 2018.

[6] Matthias Heinkenschloss and Luis Vicente. Analysis of inexact trust-region sqp algorithms. *SIAM Journal on Optimization*, 12:283–302, 02 2002.

[7] Drew P. Kouri, Matthias Heinkenschloss, Denis Ridzal, and Bart G. van Bloemen Waanders. Inexact objective function evaluations in a trust-region algorithm for pde-constrained optimization under uncertainty. *SIAM J. Scientific Computing*, 36, 2014.

[8] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.

[9] Jorge J. Moré and Stefan M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*, 20(1):172–191, 2009.

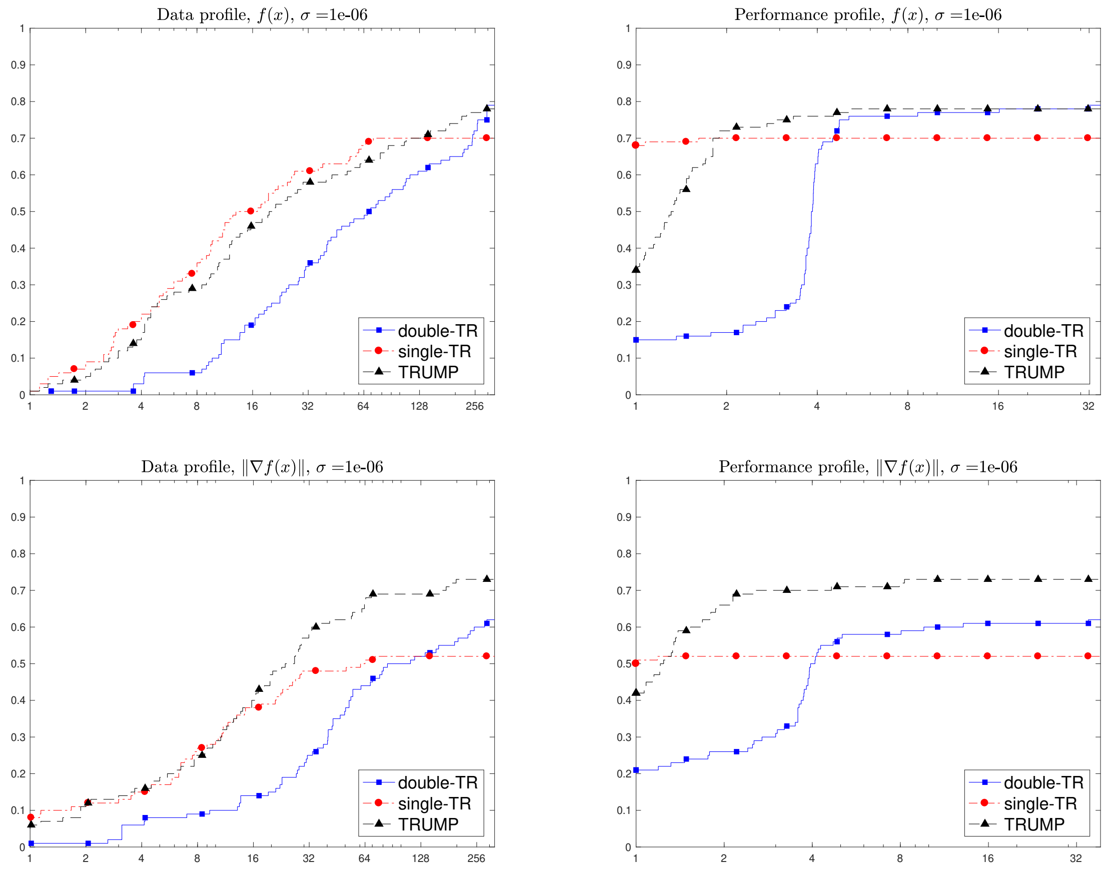

Figure 1: Data profiles (left) and performance profiles (right) in terms of both the value of $f(x)$ (top) and gradient norm $\|\nabla f(x)\|$ (bottom). Threshold was chosen as $\sigma = 10^{-6}$.

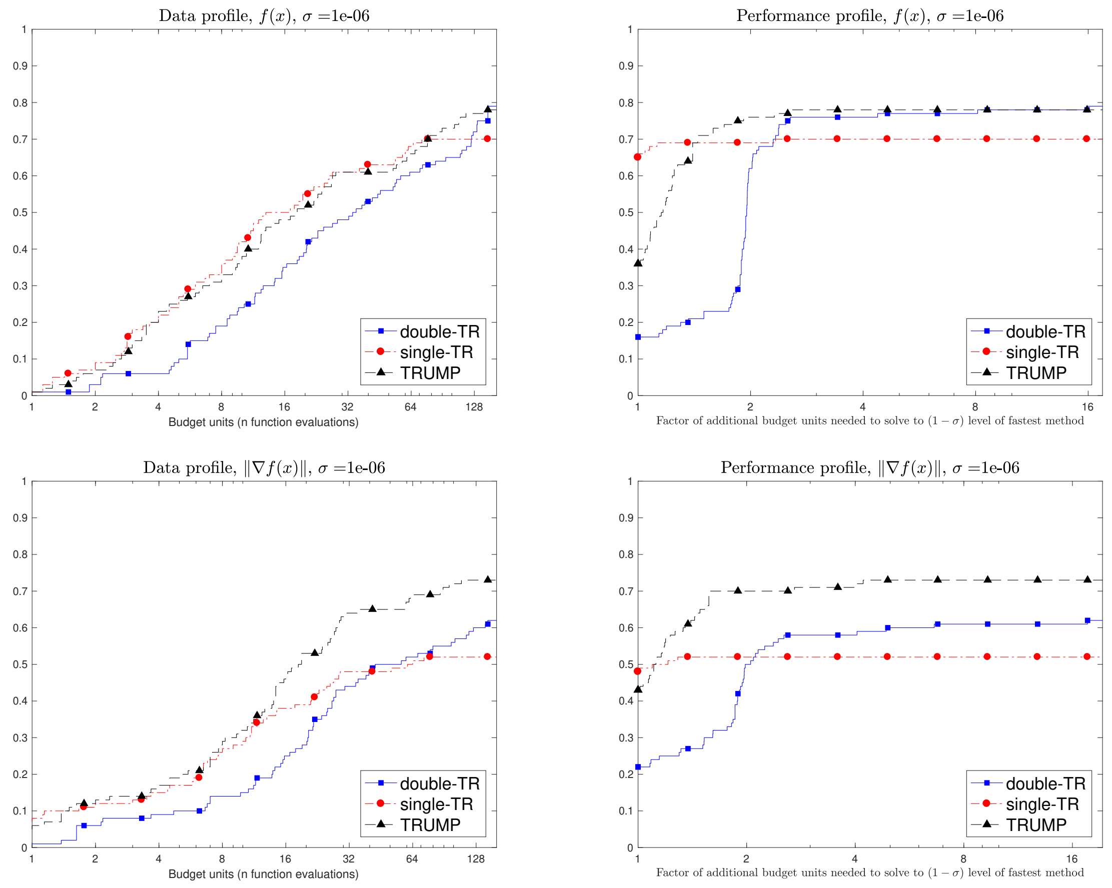

Figure 2: Same as Figure **??**, but with the factor of relative costs between single and double set to 2, instead of 4.