Iterative Methods at Lower Precision

Yizhou Chen*, Xiaoyun Gong†, Xiang Ji‡

Project Advisor: James G. Nagy§

Abstract

Since numbers in the computer are represented with a fixed number of bits, loss of accuracy during calculation is unavoidable. At high precision where more bits (e.g. 64) are allocated to each number, round-off errors are typically small. On the other hand, calculating at lower precision, such as half (16 bits), has the advantage of being much faster. This research focuses on experimenting with arithmetic at different precision levels for large-scale inverse problems, which are represented by linear systems with ill-conditioned matrices. We modified the Conjugate Gradient Method for Least Squares (CGLS) and the Chebyshev Semi-Iterative Method (CS) with Tikhonov regularization to do arithmetic at lower precision using the MATLAB **chop** function, and we ran experiments on applications from image processing and compared their performance at different precision levels. We concluded that CGLS is a more stable algorithm, but overflows easily due to the computation of inner products, while CS is less likely to overflow but it has more erratic convergence behavior. When the noise level is high, CS outperforms CGLS by being able to run more iterations before overflow occurs; when the noise level is close to zero, CS appears to be more susceptible to accumulation of round-off errors.

1 Introduction

Most computer processors today use double-precision binary floating point arithmetic, which represents floating point numbers with 64 bits. The convention follows from the IEEE-754 standard established in 1985, specifying the number of bits for the sign, exponent, and mantissa (fraction) for various floating-point formats including binary16 (half precision), binary32 (single precision), and binary64 (double precision). The difference between these formats is illustrated in Figure 1.

Taking a quarter of memory of their traditional 64-bit counterparts, computing with half precision numbers has attracted interest from researchers and manufacturers because of the potential for significantly reduced computational time. In both [13] and [6], the authors demonstrated with numerical experiments that the number of floating point operations per second of half and single precision is about 4× and 2× that of double precision. The format accelerates deep learning training, allows larger models, and improves gaming latency, providing players with better experiences

*Department of Mathematics, Emory University, Atlanta, GA 30322, USA. E-mail: rileychen111@gmail.com

†Department of Mathematics, Emory University, Atlanta, GA 30322, USA. E-mail: kristinagxy51@gmail.com

‡Department of Mathematics, Emory University, Atlanta, GA 30322, USA. Corresponding author. E-mail: zoejix@outlook.com

§Department of Mathematics, Emory University, Atlanta, GA 30322, USA. E-mail: jnagy@emory.edu

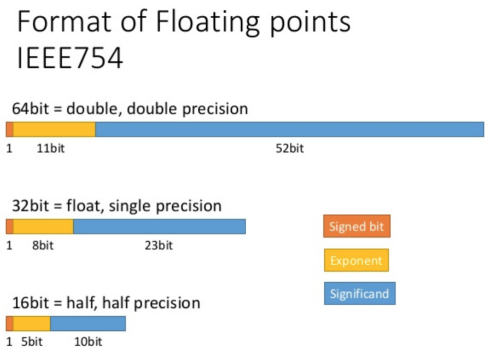

Figure 1: The IEEE standard for floating point arithmetic (IEEE 754). This diagram was obtained from [4].

[15]. However, aside from these benefits, half precision also brings one obvious disadvantage: decreased accuracy in floating point number representation. One goal of this research project is to investigate the impact of transferring from double precision operations to low precision arithmetic when solving large-scale inverse problems.

Due to their wide variety of applications, we specifically utilized and modified codes for iterative methods in solving inverse problems to evaluate the performance of low precision algorithms. Inverse problems arise when using outside measurements to acquire information about internal or hidden data [7]. We operate on known outputs with some errors to compute the true input value. For example, X-ray computed tomography is a contactless imaging method for reconstructing target objects, most commonly, pathologies in the human body. Another example is image deblurring problems, which occur when the true picture is to be reconstructed from its blurry (and sometimes noisy) version [2]. These cases can be abstracted as linear systems $\boldsymbol{b} = A\boldsymbol{x} + \boldsymbol{e}$, where A is a large-scale, typically ill-conditioned matrix and $\boldsymbol{b}$ is a vector output blended with noise, $\boldsymbol{e}$.

A naive solution to this problem can be obtained by directly calculating a solution to $A\boldsymbol{x} = \boldsymbol{b}$. However, this naive solution is often corrupted by noise due to the ill-conditioning of matrix A. Regularization methods are often needed to balance signal and noise. One approach for regularization uses the singular value decomposition of A. However, when the matrix A is large, as is often the case for many real-life applications (including our test cases), such direct methods may be difficult to implement, since decomposing the matrix can be very computationally costly. When the matrix is sparse, meaning it contains a lot of zero entries, computing matrix-vector products can be done very efficiently. Therefore, for such problems, implementing iterative methods, which only require matrix-vector products and vector operations, is a more efficient way to solve inverse problems than direct methods, with the former requiring $O(n^3)$ work for an $n \times n$ matrix A, while the latter involves significantly less than $O(n^2)$ work per iteration [16].

2 Simulating Low Precision

2.1 Chop

The MATLAB function **chop** written by Higham and Pranesh provides users a way to simulate low precision numbers and arithmetic. The function can simulate precisions, such as fp16, bfloat16, as well as custom formats where the user is able to choose the number of bits in the significand and the maximum value of the exponent. The function doesn't create the new data type; instead, it keeps numbers in double or single precision and makes sure that they have the right scale as in lower precision. Therefore, once the rounding meets the bit limit of the corresponding target precision, the remaining bits are set to zero. Computer operations are also carried out in double or single precision, and after that, the result is rounded to low precision [12].

Fully simulating computations in low precision requires users to call **chop** after each arithmetic operation. For example, for the operation

$$a = x + y \times z,$$

the appropriate use of **chop** is:

$$a = chop(x + chop(y \times z)).$$

This is burdensome yet unavoidable, so our modified version of iterative methods using **chop**, which only simulates low precision arithmetic, usually takes a long time to run, despite the fact that low precision arithmetic would run significantly faster with the corresponding hardware. However, for vector or matrix operations, there are ways to reduce the number of calls to **chop**, improving the efficiency of our codes. For example, for the vector inner product between $\boldsymbol{x}$ and $\boldsymbol{y}$, instead of using the line:

```
sum = 0;
for i = 1:vector_length
    sum = chop(sum + chop(x(i) * y(i)));
end
```

we can use an element-wise operation at the beginning:

```
sum = 0;
z = chop(x.*y)
for i = 1:vector_length
    sum = chop(sum + z(i));
end
```

such that we successfully reduce $2n$ calls of **chop** to only $n + 1$ calls, accelerating our running process.

2.2 Blocking

Since we are using floating-point arithmetic, inaccuracy is inevitable in computations. However, blocking can be used to reduce the error bound. The method breaks a large number of operations

into several smaller pieces, computes them independently, and sums them up. Consider the inner product between two vectors:

$$\boldsymbol{x} = [x_1, x_2, x_3, x_4, x_5, x_6], \quad \boldsymbol{y} = [y_1, y_2, y_3, y_4, y_5, y_6]\,.$$

Instead of calculating it directly, we could break it into:

$$\boldsymbol{x}_1 = [x_1, x_2, x_3], \quad \boldsymbol{x}_2 = [x_4, x_5, x_6]$$

$$\boldsymbol{y}_1 = [y_1, y_2, y_3], \quad \boldsymbol{y}_2 = [y_4, y_5, y_6]\,.$$

Then we calculate $\boldsymbol{x}_1^T\boldsymbol{y}_1$ and $\boldsymbol{x}_2^T\boldsymbol{y}_2$, and add the sum. The result can have less errors than the direct calculation. The error bound for inner products $\boldsymbol{x}^T\boldsymbol{y}$ is [10]:

$$|\boldsymbol{x}^T\boldsymbol{y} - fl(\boldsymbol{x}^T\boldsymbol{y})| \leq \gamma_n |\boldsymbol{x}|^T|\boldsymbol{y}|$$

where $\gamma_n = \dfrac{nu}{1-nu}$ and u is the unit round-off of floating point arithmetic (e.g. $9.77 * 10^{-4}$ for half precision, $1.19 * 10^{-7}$ for single precision, and $2.22 * 10^{-16}$ for double precision). Nonetheless, with blocking, the error bound is reduced to [10]:

$$|s_n - \hat{s}_n| \leq \gamma_{(\log_2 n)+1} |\boldsymbol{x}|^T|\boldsymbol{y}|.$$

s_n is the summation of the results for each block using the true value of $\boldsymbol{x}$ and $\boldsymbol{y}$, whereas $\hat{s}_n$ is the summation of each block's result using the floating number representation of $\boldsymbol{x}$ and $\boldsymbol{y}$. The detailed proof is in [10, Chapter 3]. We plotted the error graph for double, single, and half precision against block size when computing the inner product of two random vectors in Figure 2. We did inner products for each precision 20 times and calculated the average error, which is computed as the difference between the result from our chopped version of inner products and MATLAB's default double precision computation.

Figure 2 includes all three precision levels. Not surprisingly, we can see that the error for half precision is the largest, since it has the least bits. Focusing on half precision only, Figure 3 shows that errors decrease sharply as blocking is introduced. However, the errors increase as the block size increases, indicating that an appropriate block size needs to be carefully chosen. The reason behind this increase is that as the block size grows larger, it has the similar effect as doing no blocking; the whole vector or matrix is put into the first block, not divided into smaller sections. In our modified codes, we chose the block size to be 256, appropriate for the 4096×4096 matrices we used in the test cases. We note that there are many blocking techniques that one can use, or other approaches, such as Kahan's compensated summation method [9]. Another approach, which reduces the number of times to call the **chop** function, is to do pairwise summations and exploit the vectorization capability of the **chop** function.

3 Conjugate Gradient Method for Least Squares

3.1 Method Overview

The Conjugate Gradient (CG) Method was introduced by Hestenes and Stiefel [8]. It is an iterative method for solving the linear system $A\boldsymbol{x} = \boldsymbol{b}$, where A is a symmetric and positive definite matrix.

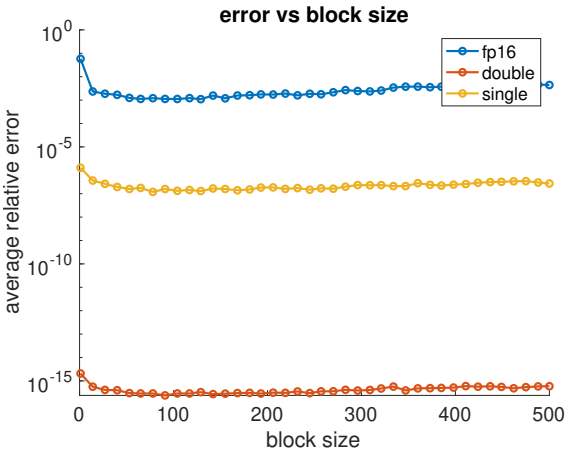

Figure 2: Relationship between block sizes and average relative errors for different precision levels

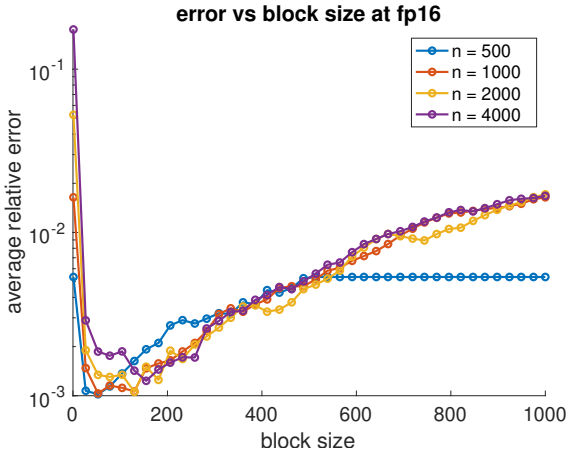

Figure 3: Relationship between block sizes and average relative errors for different vector lengths

The CG method can be viewed as an optimization problem of minimizing a convex quadratic function:

$$\phi(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{b}.$$

The gradient is zero at minimum point, meaning $\nabla\phi(\boldsymbol{x}) = A\boldsymbol{x} - \boldsymbol{b} = 0$, which is exactly the linear system we are trying to solve.

The CG method is a Krylov subspace method, which means its approximated solutions lie in Krylov subspaces. In each iteration, $\boldsymbol{x}$ is allowed to explore subspaces of increasing dimensions. An interesting property of CG is that during each iteration, $\boldsymbol{x}$ is updated to the point within the subspaces where the A-norm of the error is minimized [16]. If we assume zero round-off errors, CG is guaranteed to converge within a limited number of steps. Specifically, if A is an $n \times n$ matrix, the method will find the solution in no greater than n steps. Besides, the updated $\boldsymbol{x}$ is closer to the true solution than the previous $\boldsymbol{x}$ in each iteration.

The CGLS algorithm is the least squares version of the CG method, applied to the normal equations $A^T A\boldsymbol{x} = A^T\boldsymbol{b}$. Algorithm 1 describes CGLS [1].

One potential problem with this method for low precision is that the calculation of inner products can easily result in overflow, as illustrated in the experiment section below.

3.2 Experiments

With the **chop** function, we modified the CGLS method implemented in the IRtools package, which offers various iterative methods for large-scale, ill-posed inverse problems and a set of test problems for these iterative methods [3]. Then we used two large-scale, ill-posed inverse problems from the same package: image deblurring and tomography reconstruction. The first is to reconstruct an approximation of the true image from the observed blurry version, whereas the second one is to

Algorithm 1 Conjugate Gradient Method Least Squares

Let $\boldsymbol{x}^{(0)}$ be an initial approximation, set $\mathbf{r}^{(0)} = \boldsymbol{b} - A\boldsymbol{x}^{(0)}, \mathbf{p}^{(0)} = \mathbf{s}^{(0)} = A^T\mathbf{r}^{(0)}, \psi_0 = ||\mathbf{s}^{(0)}||_2^2$
while $\psi_k > tol$ **do**
  $\mathbf{q}^{(k)} = A\mathbf{p}^{(k)}$,
  $\alpha_k = \psi_k / ||\mathbf{q}^{(k)}||_2^2$,
  $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha_k \mathbf{p}^{(k)}$,
  $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} + \alpha_k \mathbf{q}^{(k)}$,
  $\mathbf{s}^{(k+1)} = A^T\mathbf{r}^{(k+1)}$,
  $\psi_{k+1} = ||\mathbf{s}^{(k+1)}||_2^2$,
  $\beta_k = \psi_{k+1}/\psi_k$,
  $\mathbf{p}^{(k+1)} = \mathbf{s}^{(k+1)} + \beta_k \mathbf{p}^{(k)}$.
end while

reconstruct an image from measured projections, which can be obtained, for example, from X-ray beams. We investigated these two test problems in different sizes, floating-point precision levels, and noise levels. In each of these examples, the problem is modeled as $\boldsymbol{b} = A\boldsymbol{x} + \boldsymbol{e}$, where A and $\boldsymbol{b}$ are given.

3.2.1 Image Deblurring

Here we generate an image deblurring test problem using the **PRblur** function in the IRtools package. The true image is a picture of the Hubble space telescope, and the observed data is corrupted by a Gaussian blurring function; for details, see [3]. We first added no noise to $\boldsymbol{b}$ (i.e. $\boldsymbol{e} = 0$) to see how our modified CGLS method works. In Figures 4, 5, and 6, we displayed the computed approximation of $\boldsymbol{x}$ at the best iteration, i.e. where CGLS achieves its minimum relative error. Note that we could consider displaying all of the images with the same color scale, but this would be a false color map, and we feel that there is also merit in displaying the images with the actual computed values. We continue this convention for all of our numerical experiments.

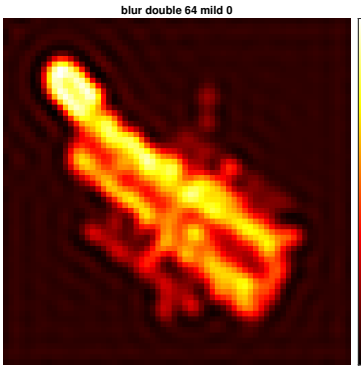

Figure 4: Double precision, size 64, zero noise.

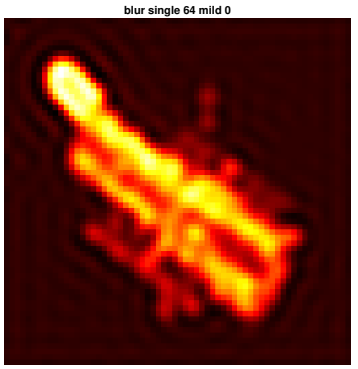

Figure 5: Single precision, size 64, zero noise.

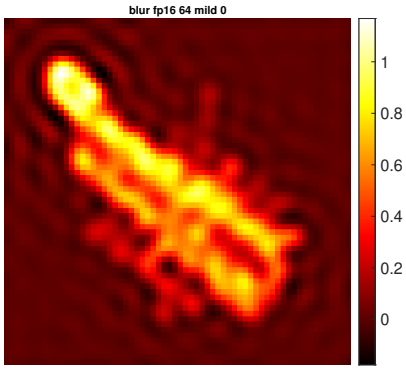

Figure 6: Half precision, size 64, zero noise.

The single-precision result is very similar to that computed in double-precision, but the half-precision result is more blurry, and the background contains more artifacts. We also plotted the

error graphs for all three formats in Figure 7 and the singular values of matrix $A \in \mathbb{R}^{4096\times 4096}$ to see how ill-conditioned the matrix is in Figure 8. As we can see in Figure 8, the singular values decay quickly, which is typical in large-scale inverse problems.

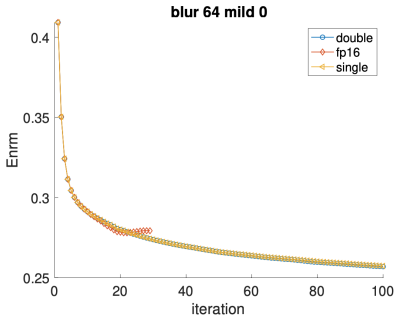

Figure 7: The error norm of a size 64 problem with mild blurring of different precisions.

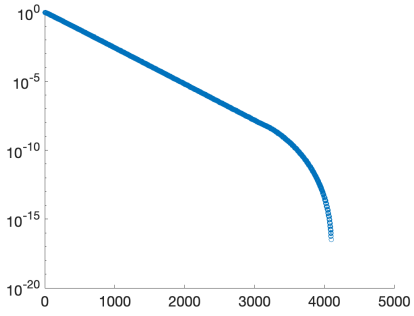

Figure 8: Singular value of matrix for the deblurring problem of size 64 with mild blur.

All three error norms overlap from the beginning until around 20^{th} iteration, where the half-precision errors begin to deviate from those in single and double precision. The difference is due to the round-off errors of half precision, which add up and take over. Moreover, the error norms of half precision terminates at 28^{th} iteration, because overflow of inner products causes NaNs (Not a Number) to be computed during the iteration.

In real life, the noise-free image, $\boldsymbol{b}$, is not often obtainable but most likely it is blended with noise. Therefore, it is imperative to make sure that the chopped CGLS method works effectively with noisy data as well. We added noise at different levels to $\boldsymbol{b}$ and displayed the computed approximations of $\boldsymbol{x}$ in the last iteration in half precision Figures 9, 10, and 11.

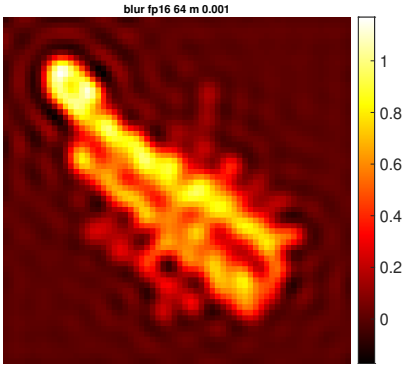

Figure 9: Half precision, size 64, 0.1% noise.

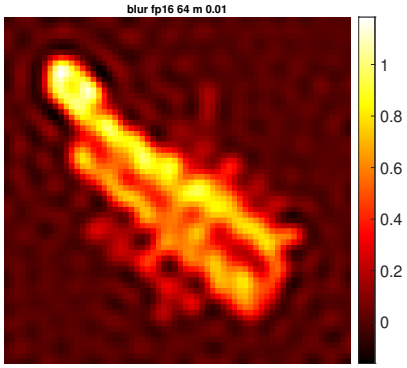

Figure 10: Half precision, size 64, 1% noise.

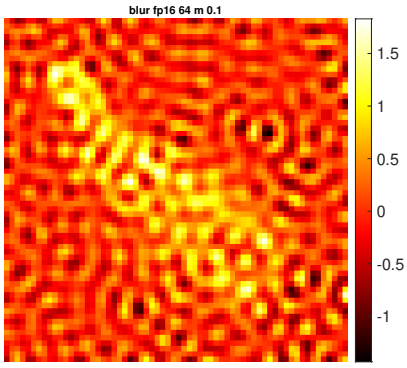

Figure 11: Half precision, size 64, 10% noise.

However, the last iteration doesn't necessarily mean it's the iteration with the best performance, so the results generated using $\boldsymbol{x}$ from the best iteration (i.e. the iteration with smallest error norms) are also shown in Figures 12, 13, and 14. For results in the last iteration, while images obtained

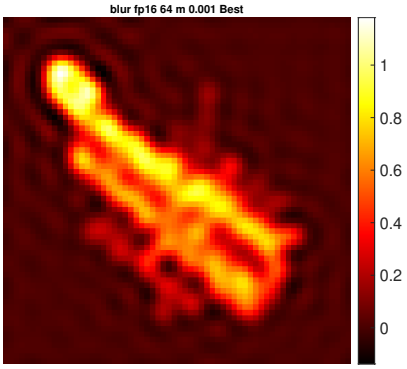

Figure 12: Half precision, problem size 64 with mild blurring and 0.1% noise at best iteration.

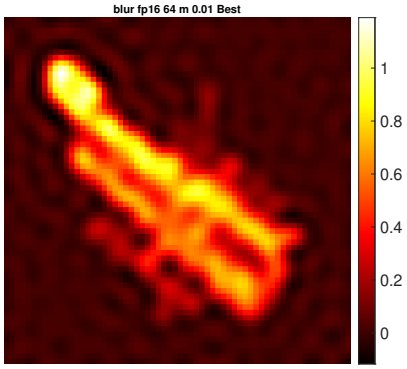

Figure 13: Half precision, problem size 64 with mild blurring and 1% noise at best iteration.

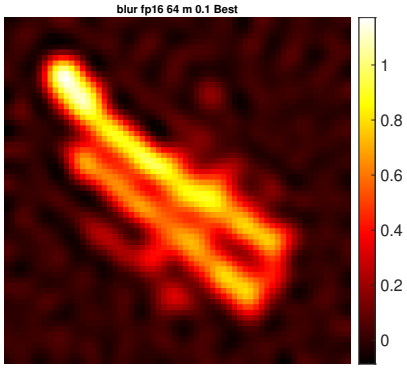

Figure 14: Half precision, problem size 64 with mild blurring and 10% noise at best iteration.

when using half precision are more distorted than their single counterparts, they follow the same trend in terms of the impact of noise. 0.1% noise behaves very similarly to the cases with no noise; however, 1% noise begins to dominate, and the background has substantially more artifacts, while the object is still identifiable. Eventually, the computed result is horribly corrupted by artifacts in the 10% noise case; the picture contains very little meaningful information. For all the following images we show in the paper, we display the reconstructed image from the the best iteration if not specified.

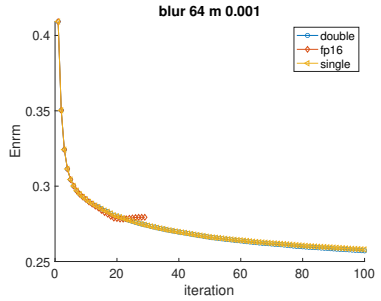

Figure 15: Error norm for problem size 64 with mild blurring and 0.1% noise.

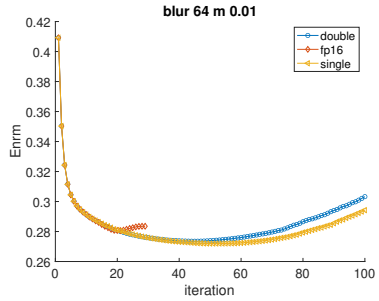

Figure 16: Error norm for problem size 64 with mild blurring and 1% noise.

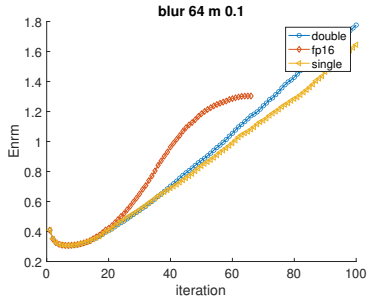

Figure 17: Error norm for problem size 64 with mild blurring and 10% noise.

An interesting phenomenon emerges if we examine the error norm for each iteration in Figures 15, 16, and 17. While the norm decreases as the iteration continues for the 0.1% noise cases, those with 1% and 10% noise present a different t rend: t he e rror n orm s tarts t o i ncrease a gain after certain iterations. The reason for this reversal is that the early iterations reconstruct components of the solution corresponding to large singular values, and later iterations begin to reconstruct components corresponding to small singular values. Thus, the early iterations begin to converge to something that resembles a truncated SVD solution, and the later iterations converge to the inverse solution, $\hat{\boldsymbol{x}} = A^{-1}\boldsymbol{b} = A^{-1}(A\boldsymbol{x} + \boldsymbol{e}) = \boldsymbol{x} + A^{-1}\boldsymbol{e}$. When A is severely ill-conditioned, the $A^{-1}\boldsymbol{e}$ term dominates, so the computed solution $\hat{\boldsymbol{x}}$ is not a good approximation to the true solution. This

phenomenon is known as semi-convergence in the inverse problems literature [7]. It is interesting to observe that the error norms during the early iterations for all precision levels are similar. This is an important observation that might be worth of exploiting in future work; for example, using low precision methods for preconditioning.

3.2.2 Tomography Reconstruction

Tomography reconstruction is another type of inverse problem that produces images from X-ray projection data of various angles. We used it as a test problem since the IRtools software package includes a simulation in the **PRtomo** function. In this test problem, matrix $A \in \mathbb{R}^{16380\times 4096}$. It is not as ill-conditioned as that in the previous problem, as shown by the plot of singular values in Figure 18. The modified CGLS method works well for double and single precision and generates nice reconstructions as shown in Figures 19 and 20, but issues arise for half precision because entries of the solution $\boldsymbol{x}$ are all NaNs. Since half precision formats allocate only 5 bits for the exponent, overflow easily occurs when the inner product is calculated. NaNs are the results from dividing infinity by infinity in the CGLS method.

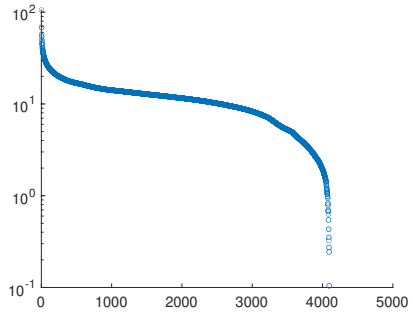

Figure 18: Singular value of the matrix for the tomography problem.

One of the solutions to the overflow problem is to rescale the matrices. After scaling both A and $\boldsymbol{b}$ by 0.01, we obtain Figure 21. Although the image is still not as clear as those from double and single precision, the inner products did not overflow and a meaningful picture was obtained. The images shown in Figures 19, 20, and 21 are those corresponding to the iteration where the relative error is minimized.

The noise level affects the quality of the reconstruction as well. As Figures 22, 23, 24 illustrate, while random noise does not completely destroy the reconstruction and take over the original information as is the case for image deblurring, the artifacts caused by the noise make it difficult to see small objects in the image.

As was the case with the deblurring problem, the error norm begins to increase at some point of the iteration due to the accumulation of the inverted noise, as shown in Figures 25, 26, and 27. At half precision, the norm grows after the 11^{th} iteration even without noise in Figure 25. The reason

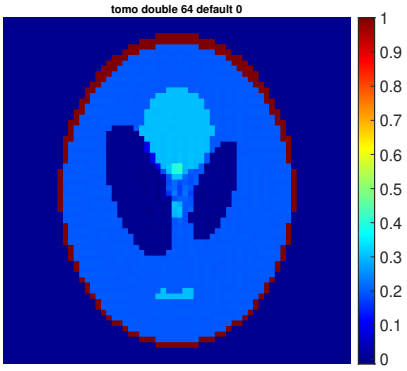

Figure 19: Double precision, size 64, zero noise.

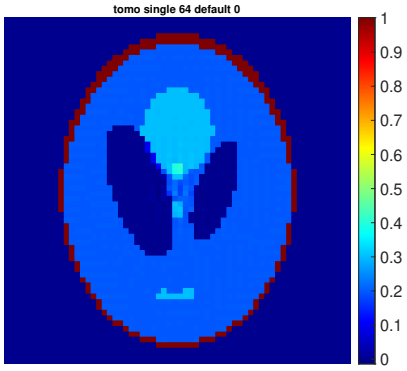

Figure 20: Single precision, size 64, zero noise.

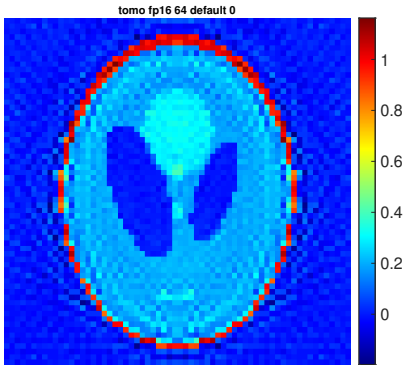

Figure 21: Half precision, size 64, zero noise (after rescaling).

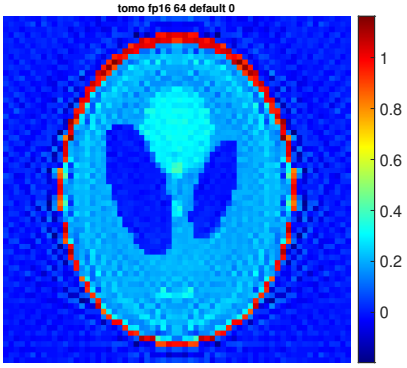

Figure 22: Half precision, size 64, zero noise.

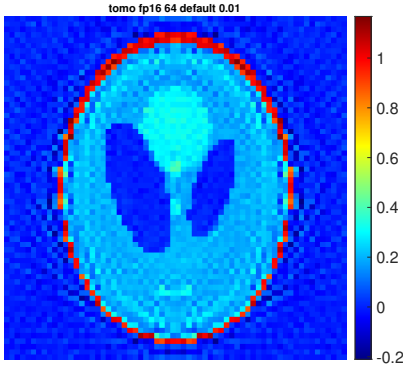

Figure 23: Half precision, size 64, 1% noise.

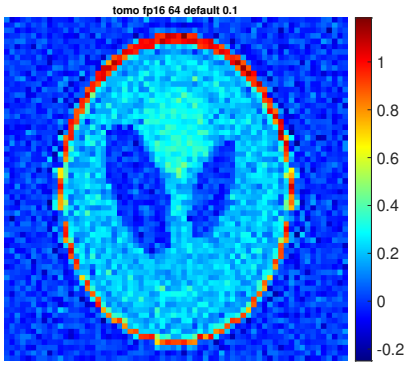

Figure 24: Half precision, size 64, 10% noise (after rescaling).

this time is not the attached noise but the truncation errors.

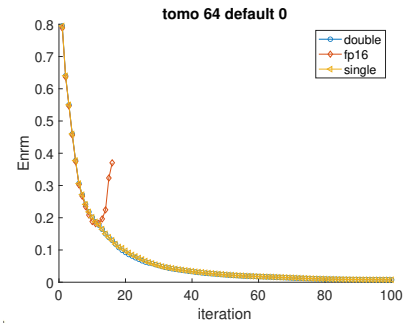

Figure 25: Error norm for size 64 problem with zero noise.

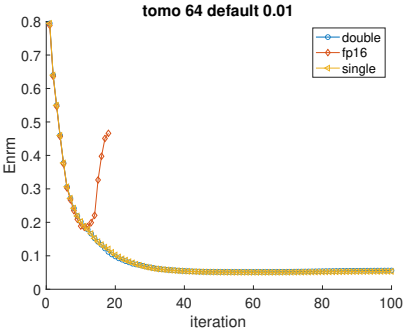

Figure 26: Error norm for size 64 problem with 1% noise.

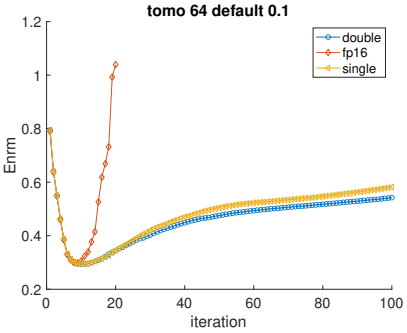

Figure 27: Error norm for size 64 problem with 10% noise.

4 Chebyshev Semi-Iterative Method

4.1 Method Overview

The Chebyshev Semi-Iterative (CS) Method requires no inner product computation, so it is safer in terms of overflow issues when at low precision. The trade-off here is that the CS method needs prior knowledge of the range of singular values of the matrix A [17].

As with CG, the CS method is generally derived for symmetric positive definite linear systems, but there are variations for least squares problems, which are applied to the normal equation $A^TA\boldsymbol{x} = A^T\boldsymbol{b}$. The following algorithm[1] describes the implementation of the CS method for least squares problems that we used in our work [14] [5]:

Algorithm 2 Chebyshev semi-iterative method

Given $A \in \mathbb{R}^{m\times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, and a tolerance $\epsilon > 0$, choose $0 < \sigma_L \leq \sigma_U$ such that all nonzero singular values of A in $[\sigma_L, \sigma_U]$, and let $d = \frac{{\sigma_U}^2+{\sigma_L}^2}{2}$ and $c = \frac{{\sigma_U}^2-{\sigma_L}^2}{2}$
Let $\boldsymbol{x} = 0$, $\boldsymbol{v} = 0$, and $\mathbf{r} = \boldsymbol{b}$.
for $k = 0, 1, \ldots, \lceil \log\epsilon - \log 2/\log\frac{\sigma_U-\sigma_L}{\sigma_U+\sigma_L} \rceil$ **do**

$$\beta \leftarrow \begin{cases} 0, & \text{if } k = 0 \\ \frac{1}{2}(c/d)^2, & \text{if } k = 1 \\ (\alpha c/2)^2, & \text{otherwise,} \end{cases} \qquad \alpha \leftarrow \begin{cases} 1/d, & \text{if } k = 0 \\ 1/(d - c^2/(2d)), & \text{if } k = 1 \\ 1/(d - \alpha c^2/4), & \text{otherwise} \end{cases}$$

$\boldsymbol{v} \leftarrow \beta\boldsymbol{v} + A^T\mathbf{r}$.
$\boldsymbol{x} \leftarrow \boldsymbol{x} + \alpha\boldsymbol{v}$.
$\mathbf{r} \leftarrow \mathbf{r} - \alpha A\boldsymbol{v}$.
end for

4.2 Experiments

Again, we used image deblurring and tomography reconstruction as two test problems for the CS method in low precision. At first we ran the image deblurring test problems of size 32 with default blur and 1% noise in double precision. We got estimations for the bound of singular values of matrix A with the built-in MATLAB function **SVDS**. We set the maximum number of iterations to be 500, and we used ϵ to be 0.01, which we chose based on some experiments (we do not know the optimal way to choose ϵ). We plotted the graph at the first, 5^{th} (where the error norm is smallest), and last iteration, as shown in Figures 28, 29, and 30.

The reconstruction is doing poorly even at the best iteration (Figure 29). At the last iteration, it is even worse due to over-fitting. After a closer look, we noticed that the number of iterations recommended by the algorithm is at order of magnitude of 10^{12}. If we increase the size of the problem to 64×64, the number overflows to -Inf, which results from the tiny estimation of the lower bound of matrix A's singular values. The estimation is so close to zero that we suspect this is actually a result of round-off errors on a zero entry. Therefore, we implemented Tikhonov

[1] We are using Algorithm 3 from [14], but we remark that there is a typographical error for the α parameter for the case $k = 1$. We show the correct formula in our paper.

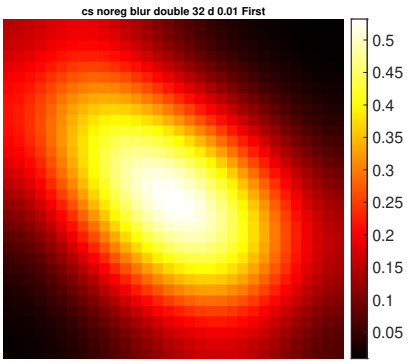

Figure 28: Reconstruction at first iteration.

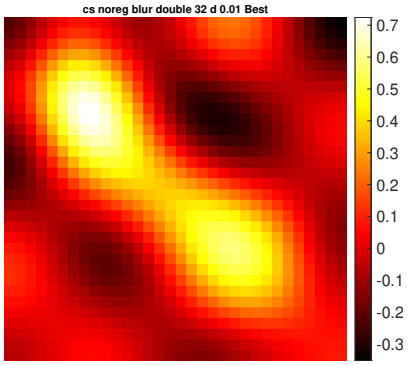

Figure 29: Reconstruction at the best iteration.

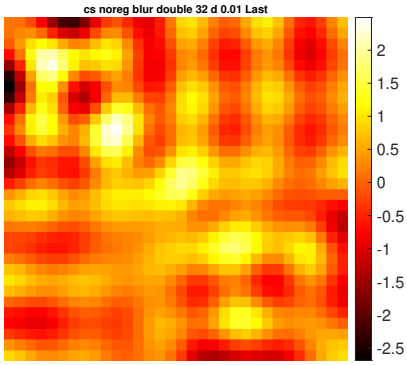

Figure 30: Reconstruction at the last iteration.

regularization for two reasons: a) to avoid over-fitting; and b) to obtain a more valid lower bound of singular values by increasing them to a larger value. Specifically, the singular values for the matrix A after Tikhonov regularization with regularization parameter λ would be $\sqrt{\sigma_i^2 + \lambda^2}$. The technique is discussed in more details in the next section 4.2.1.

4.2.1 Tikhonov Regularization

Tikhonov Regularization includes a regularization term to the original least squares problem:

$$\min_x \{||A\boldsymbol{x} - \boldsymbol{b}||_2^2 + \lambda^2 ||\boldsymbol{x}||_2^2\},$$

where the regularization parameter λ balances the residual term $||A\boldsymbol{x} - \boldsymbol{b}||_2^2$ and the regularization term $||\boldsymbol{x}||_2^2$. We can rewrite the Tikhonov problem as a least squares problem

$$\min_{\boldsymbol{x}} \left\| \begin{pmatrix} A \\ \lambda I \end{pmatrix} \boldsymbol{x} - \begin{pmatrix} \boldsymbol{b} \\ 0 \end{pmatrix} \right\|_2,$$

and the solution to this least squares problem is

$$\boldsymbol{x}_\lambda = (A^T A + \lambda^2 I)^{-1} A^T \boldsymbol{b}.$$

To see why Tikhonov regularization is effective, observe that if we substitute the singular value decomposition, $A = U\Sigma V^T$, into the expression for $\boldsymbol{x}_\lambda$, and expand the matrix multiplication column-wise, we would get

$$\boldsymbol{x}_\lambda = \sum_{i=1}^{n} \phi_i^{[\lambda]} \frac{\boldsymbol{u}_i^T \boldsymbol{b}}{\sigma_i} \boldsymbol{v}_i,$$

where the filter factors $\phi_i^{[\lambda]}$ are

$$\phi_i^{[\lambda]} = \frac{\sigma_i^2}{\sigma_i^2 + \lambda^2}.$$

Notice that the filter factor $\phi_i^{[\lambda]}$ is approximately equal to 0 for smaller singular values and approximately equal to 1 for larger singular values. It therefore acts like a filter by decreasing the effects

of magnifying noise in $\boldsymbol{b}$ when divided by tiny singular values.

With regularization, we are running the CS method with the matrix $\begin{pmatrix} A \\ \lambda I \end{pmatrix}$ instead of A.

Given $A = U\Sigma V^T$, we have

$$A^T A = V\Sigma^T \Sigma V^T$$

and

$$\begin{pmatrix} A^T & \lambda I \end{pmatrix} \begin{pmatrix} A \\ \lambda I \end{pmatrix} = A^T A + \lambda^2 I = V\Sigma^T \Sigma V^T + V\lambda^2 V^T = V(\Sigma^T \Sigma + \lambda^2)V^T.$$

Therefore, we know the singular values of $\begin{pmatrix} A \\ \lambda I \end{pmatrix}$ are $\sqrt{\sigma_i^2 + \lambda^2}$, where σ_i are singular values of A.

When σ_i is small, $\sqrt{\sigma_i^2 + \lambda^2} \approx \lambda$, and we can directly use λ as the lower bound of singular values for the matrix after regularization. With Tikhonov regularization, the singular values are increased to a more reasonable lower bound.

Choosing a regularization parameter is a challenging topic. To get regularization parameters for our experiments, we first ran a hybrid LSQR method in double precision, as implemented in [3], which uses a generalized cross validation method to obtain an estimate for the Tikhonov regularization parameter.

4.2.2 Image Deblurring

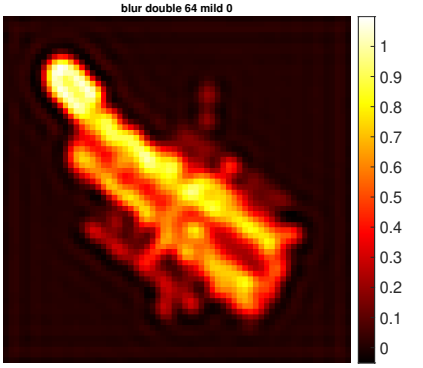

Figure 31: Image reconstruction in double precision of size 64 problem with mild blurring and no noise.

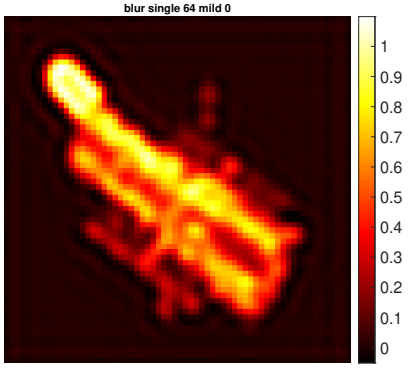

Figure 32: Image reconstruction in single precision of size 64 problem with mild blurring and no noise.

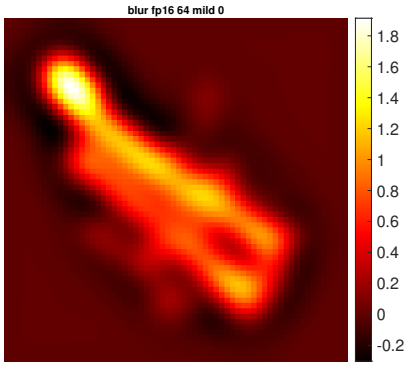

Figure 33: Image reconstruction in half precision of size 64 problem with mild blurring and no noise.

With regularization, we successfully ran 478 iterations in all three precisions, without the occurrence of NaNs. We plotted the results in Figures 31, 32, and 33. For half precision, the result is not clear as expected, and the image seems to be dominated by round-off errors. We plotted the error norm in Figure 37.

The error norms do not converge to a point like those generated by the CGLS algorithm. Instead, they oscillate at the beginning, and then converge for double and single precision. However,

for half precision, the error norm increases rapidly, indicating that the round-off errors accumulate and take over. We suspect this is because the regularization parameter is too small, so we may need to develop better ways to determine a suitable regularization parameter for half precision.

We then added noise to $\boldsymbol{b}$ and displayed the resulting computed reconstructions in Figures 34, 35, and 36. Here we only showed results with 10% noise; results for other noise levels are consistent

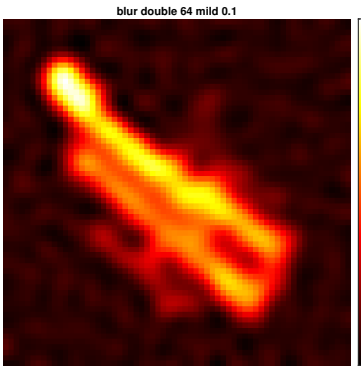

Figure 34: Image reconstruction in double precision of size 64 problem with mild blurring and 10% noise.

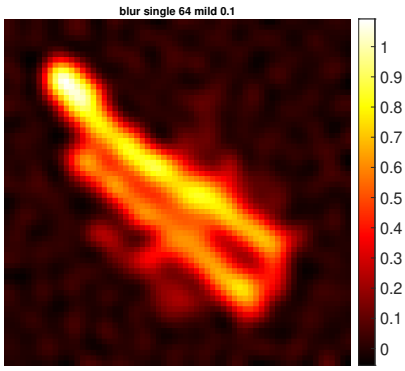

Figure 35: Image reconstruction in single precision of size 64 problem with mild blurring and 10% noise.

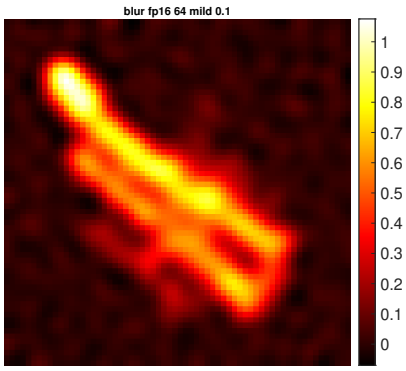

Figure 36: Image reconstruction in half precision of size 64 problem with mild blurring and 10% noise.

with those shown in previous sections. Because of the large amount of noise, we needed to use relatively large regularization parameters, and this provides more stability for all precision levels, including those computed in half precision. It is possible that tuning the regularization parameter for half precision with no noise will produce a better result, but in our computation, we used a standard generalized cross validation approach to choose regularization parameters [3], and we did not attempt to further tune them for the various specific cases.

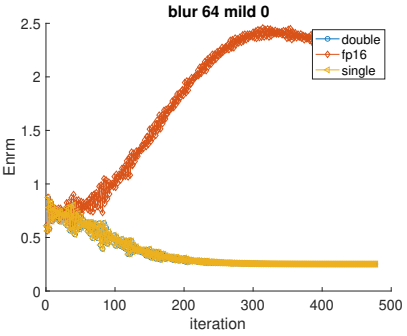

Figure 37: Error norm of a size 64 problem with mild blurring of different precisions with 0 noise.

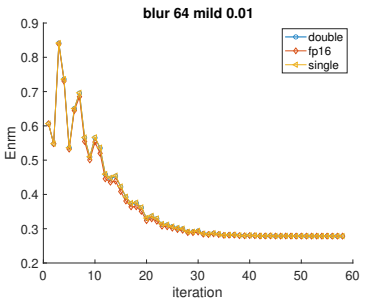

Figure 38: Error norm of a size 64 problem with mild blurring of different precisions with 1% noise.

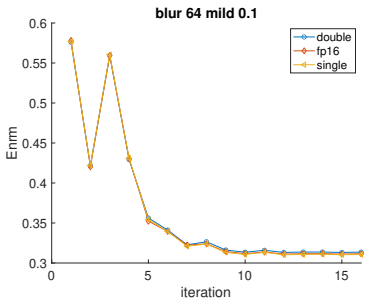

Figure 39: Error norm of a size 64 problem with mild blurring of different precisions with 10% noise.

By observing the error norm plots in Figure 39, we also see that there seems to be very little

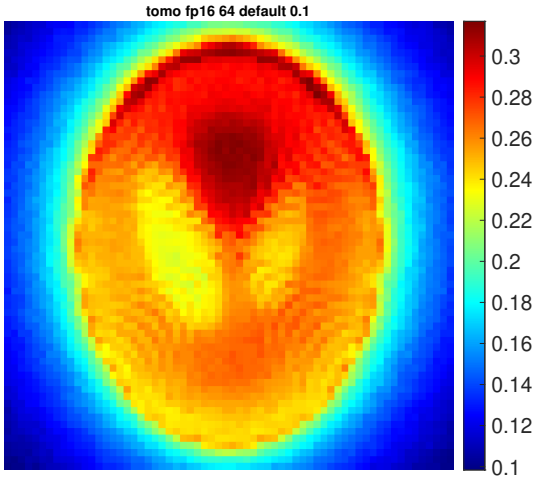

Figure 40: Half precision, size 64, 10% noise, without rescale.

difference in the convergence behavior of CS for the different precision levels when the noise level is high. The more noise we have, the closer are the results between double and half precision.

4.2.3 Tomography Reconstruction

Although CS avoids the computation of inner products, overflow still occurs at half precision during the calculation of matrix-vector multiplications for the tomography reconstruction problem. NaNs start to occur at the second iteration. Figure 50 shows the resulting image at the end of the first iteration at half precision for a problem with 10% noise, which is already reasonably good with visible shapes and boundaries.

Due to overflow, we rescaled the problem as for CGLS so that CS could run more iterations for half precision. The algorithm successfully runs to the end without occurrences of overflow. In the figures below, we show the results at the last iteration for problems with different precision and noise levels.

After rescaling, when there is noise on the right hand side $\boldsymbol{b}$ of the problem, the resulting image at low precision is as valid as images produced at high precision. With 10% noise (Figures 43, 46, and 49), the CS algorithm ran 32 iterations, and produced good reconstructions with clear boundaries and background for all three precision levels.

However, when there is zero noise, the result at half precision is a poor reconstruction. The background and the object both look blurry and blend together, and the object contents are hardly visible. We believe the dissatisfying image is a combined result of accumulation of round-off errors and under-regularization.

After a closer look at the result of each iteration, we noticed that the reconstructions in the first few iterations look smooth and improved as the iteration moves on, but at some point they start to become noisy and blurry. Unlike CGLS where the output image at each iteration refines steadily,

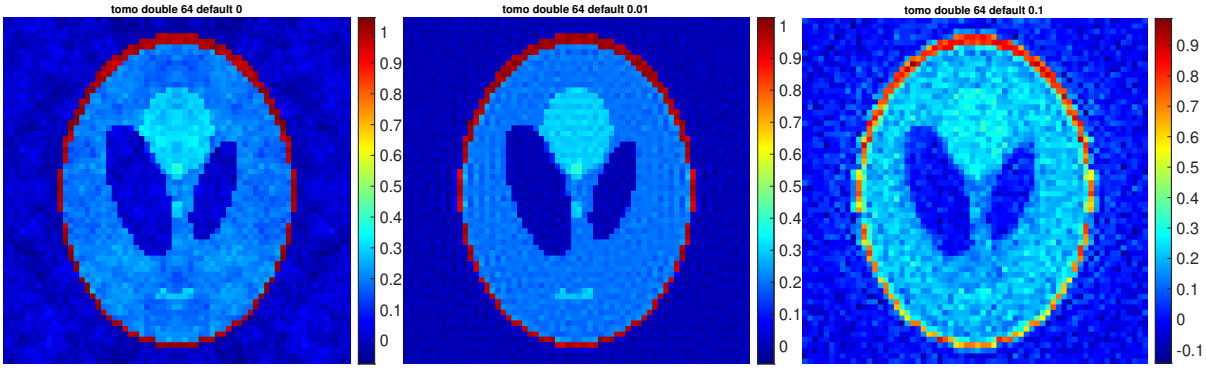

Figure 41: Double precision, size 64, zero noise.

Figure 42: Double precision, size 64, 1% noise.

Figure 43: Double precision, size 64, 10% noise.

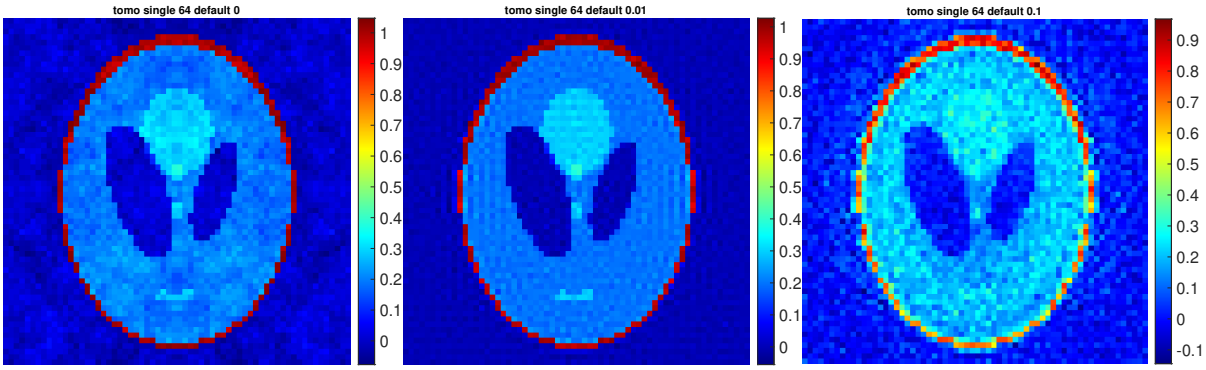

Figure 44: Single precision, size 64, zero noise.

Figure 45: Single precision, size 64, 1% noise.

Figure 46: Single precision, size 64, 10% noise.

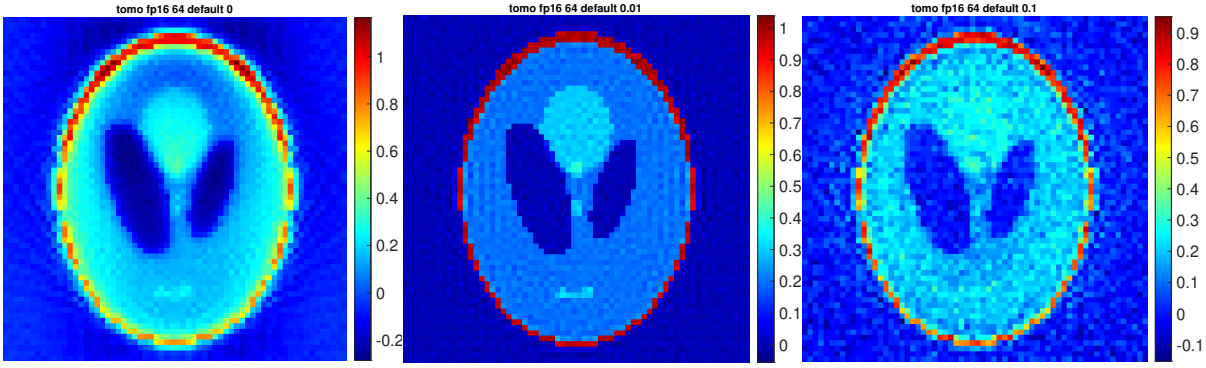

Figure 47: Half precision, size 64, zero noise (after rescaling).

Figure 48: Half precision, size 64, 1% noise (after rescaling).

Figure 49: Half precision, size 64, 10% noise (after rescaling).

the results from CS have more oscillations from iteration to iteration, and it has a general trend of becoming noisy as the iteration goes on. Below we show the resulting images at the first iteration (Figure 50), the 5^{th} iteration (Figure 51) where the error norm is the smallest, and the 250^{th} iteration (Figure 52). At the 250^{th} iteration the image is already very noisy as round-off errors

accumulated along the way.

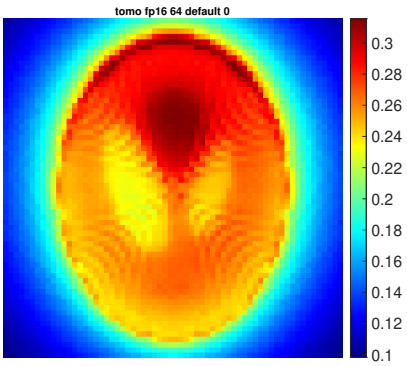

Figure 50: Reconstruction at first iteration, half precision, zero noise (after rescaling).

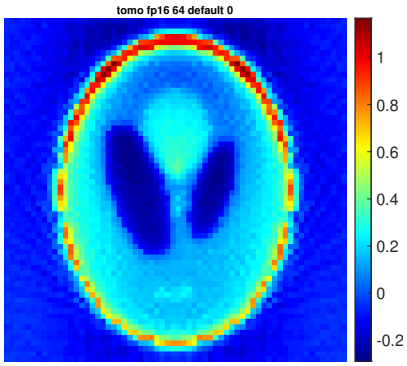

Figure 51: Reconstruction at best iteration, half precision, zero noise (after rescaling).

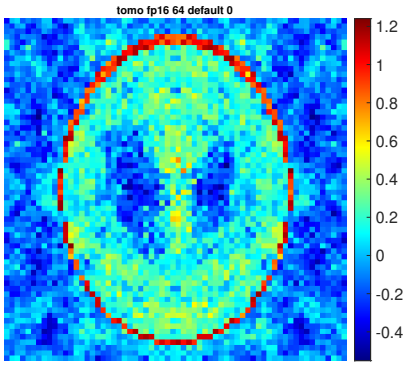

Figure 52: Reconstruction at 250^{th} iteration, half precision, zero noise (after rescaling).

Figures 53, 54 and 55 present the error norms at different precision levels across different noise levels. As expected, the error norms overlap for test problems with noise, implying that the quality of the reconstructions is similar. However, for the noise-free problems, the error norms follow a decreasing trend for double and single precision and a slightly increasing trend for half precision. Besides, the norms oscillate more as noise level decreases, which corresponds with our observation from the resulting images.

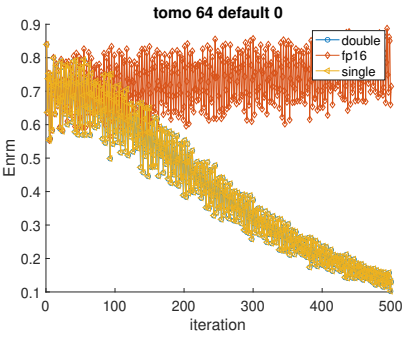

Figure 53: Error norm of a size 64 problem at different precisions with zero noise.

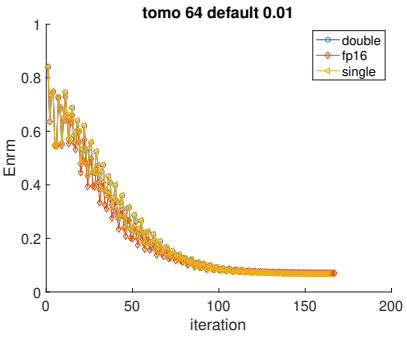

Figure 54: Error norm of a size 64 problem at different precisions with 1% noise.

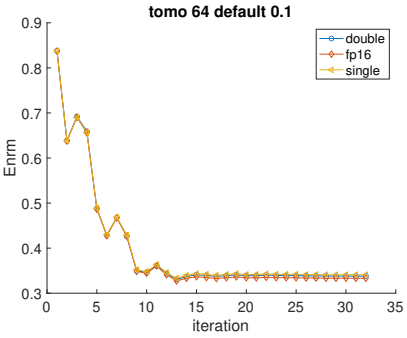

Figure 55: Error norm of a size 64 problem at different precisions with 10% noise.

5 Discussion

We incorporated CGLS with regularization to compare its performance with that of CS more fairly. The effect of regularization is apparent, as the resulting image is less noisy, especially for low precision. As the noise level increases, the difference between high precision and low precision becomes

less significant. Yet still, rescaling is necessary for half precision to avoid overflow for the tomography reconstruction problem. Below we showed the result of CGLS with regularization for the two test problems.

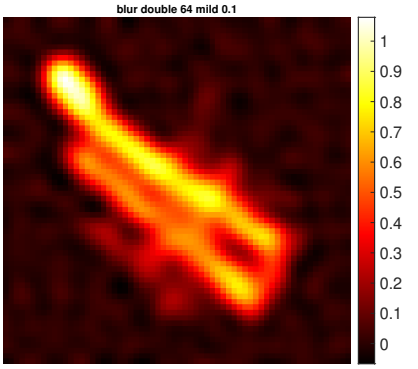

Figure 56: CGLS with regularization, double precision, 10% noise.

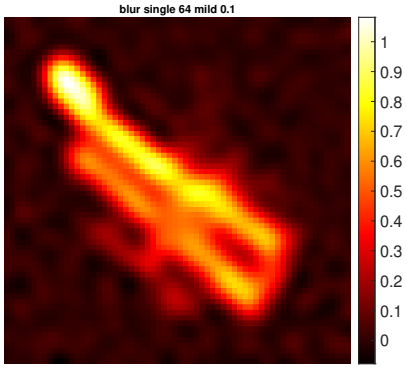

Figure 57: CGLS with regularization, single precision, 10% noise.

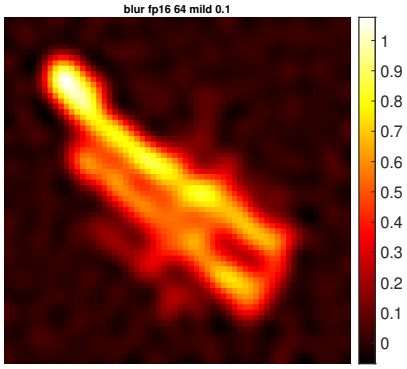

Figure 58: CGLS with regularization, half precision, 10% noise.

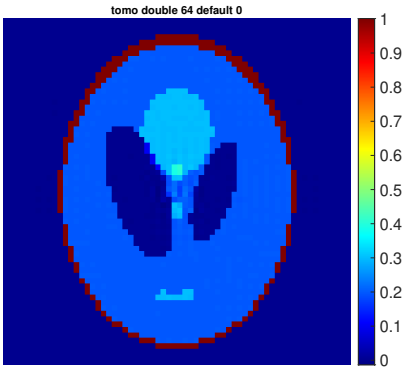

Figure 59: CGLS with regularization, double precision, zero noise.

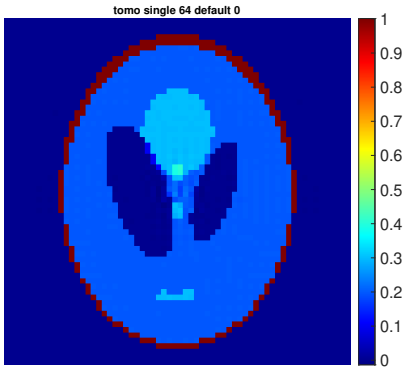

Figure 60: CGLS with regularization, single precision, zero noise.

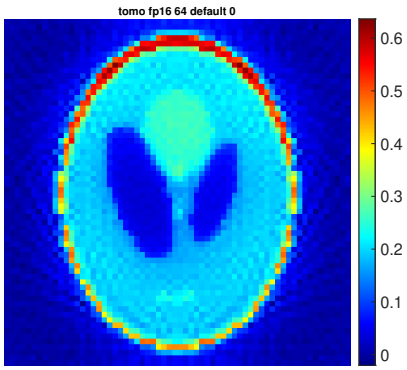

Figure 61: CGLS with regularization, half precision, zero noise (after rescaling).

Overall, CGLS is a more stable algorithm than CS and has less oscillations in the solution. The result given by CGLS is steadily improving in the iterations even with the presence of noise. Switching to lower precision does not have a large impact on the performance of CGLS, as long as no overflow or underflow occurs. And if they do occur, we can rescale the problem to delay the occurrence of overflow and NaNs so that a relatively good result can still be obtained. However, we do expect there to be still significant problems with overflow and NaNs for larger problems.

The main problem with CGLS at low precision is to find the suitable rescaling parameter, which depends on the matrix A and vector $\boldsymbol{b}$ of each problem. We did not have issues of overflow/underflow at all for the image deblurring problem, while NaNs started to appear at the first iteration for the tomography reconstruction problem. We tried several rescaling parameters before we found a

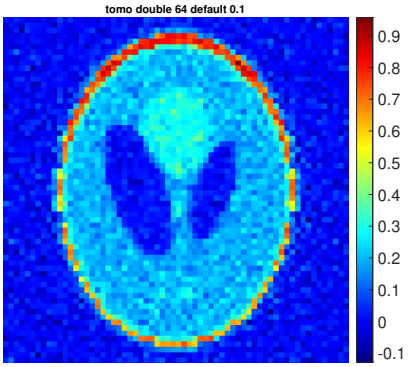

Figure 62: CGLS with regularization, double precision,10% noise.

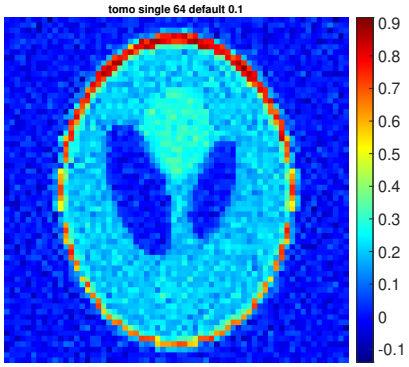

Figure 63: CGLS with regularization, single precision,10% noise.

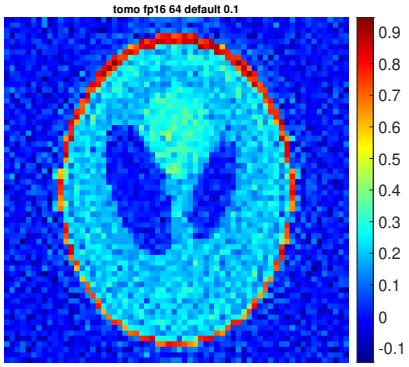

Figure 64: CGLS with regularization, half precision,10% noise (after rescaling).

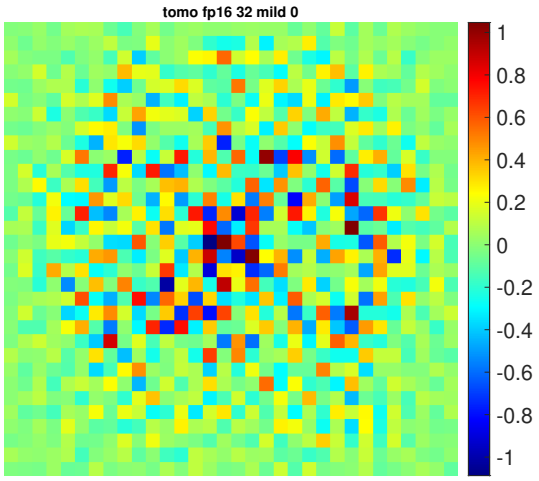

Figure 65: Half precision, size 32, zero noise.

suitable one, and sometimes it is hard to determine whether a parameter is suitable or not if we do not have an idea of what the real image looks like. The purpose of rescaling is to avoid NaNs in the solution, but a solution without NaNs does not imply it has been rescaled properly. For example, Figure 65 is the result of a size-32 tomography reconstruction test problem given by CGLS after 100 iterations. No NaN occurred during the iteration process, but the resulting image is far from the true solution. This is because the overflow leads to an underflow of $\boldsymbol{x}$ to zero immediately in the first iteration. Though it did not result in NaNs, the algorithm did not capture any information about $\boldsymbol{x}$ in that iteration as well. Therefore, the image is not rescaled properly despite the absence of NaNs.

CS, on the other hand, does not necessarily require rescaling at low precision as it avoids the calculation of inner products, which is the main source of overflow. The major advantage of CS is that it is able to run more iterations before the first NaNs occur, so that the resulting images have more chances to get refined and are less likely to be cut off before the optimal point. There are also cases when CGLS overflows at the first iteration and produces anything but NaNs, while CS

is able to run several iterations and produce a relatively meaningful image, as for the tomography reconstruction test problem. In these circumstances, if only a rough estimate of the original image is desired, CS can be a more convenient choice as it saves the trouble of finding a proper rescaling parameter.

However, CS requires the user to have an estimate of the bounds of A's singular values, which may be hard to estimate. Even with Tikhonov regularization, the resulting bound depends on the regularization parameter λ, which in turn depends on the noise level and the problem itself. When λ is small, the computation becomes more risky, especially at low precision. Generally speaking, when the noise is large, we would need a larger λ to filter out the noise. Therefore, it is not surprising that CS has better performance when the noise level is high. However, when the noise is negligible, the more stable CGLS is a better choice.

6 Conclusion

In this project, we explored methods for solving inverse problems at low precision. We first modified several MATLAB built-in operations using **chop** for custom precision levels and applied the blocking method to decrease the error bounds of the simulation. Then we ran the modified CGLS and CS methods on image deblurring and tomography reconstruction tasks to compare their performance across different precision levels. We added Tikhonov regularization to both methods to balance signal and noise.

After comparing results given by the two algorithms, we concluded that CGLS is more stable than CS. Its output image steadily gets closer to the real image. However, CGLS is likely to suffer from overflow at low precision as it involves the calculation of inner products. One solution to this issue is to rescale the problem, but finding a suitable rescaling parameter requires trial and error. Moreover, the rescaling is unlikely to be effective for extremely large problems when using low precision. In the future, we hope to find a more direct way that finds a suitable rescaling parameter given the matrix A, vector $\boldsymbol{b}$, and precision level. Based on intuition gained from experiments, a factor that "rescales" the values to $O(1)$ is often a good choice. We would also look at algorithms for normalizing A as described in the survey by Higham and Mary [11].

The performance of the CS method is less stable and depends more on the noise level of the problem. It has more oscillations than CGLS during the iterating process. However, CS requires no calculation of inner products, and it is therefore less likely to overflow. The method avoids the need of rescaling for some problems; with rescaling, it can refine the result with even more iterations and often ends up with a better resulting image than CGLS. The price paid here is that CS needs some prior knowledge of the range of the matrix's singular values. When the matrix is large and sparse, for low precision levels, we need a regularization parameter large enough to obtain a valid lower bound and avoid the accumulation of round-off errors. For problems with noise on the right hand side $\boldsymbol{b}$, most of the time we would naturally end up with a large enough λ. Therefore, CS has great performance for noisy problems. However, when the problem is noise-free, the estimated λ given by the parameter selection methods are often too small, and CS performs poorly. In the future, we hope to develop better methods for choosing suitable regularization parameters for CS at low precision.

When examining the error norms of the solution at each iteration, we noticed that sometimes the error norm did not match the so-called "eyeball norm." An image with clear shape and background may have a higher error norm than a blurry, noisy image. The latter one, though with a smaller error norm, is obviously less informative than the former one. Therefore, more research is needed on ways to take other aspects of the output image into the account of error measurement so that we could find an image that conveys the most information.

Moreover, we would like to explore more iterative methods other than CGLS and CS and implement them in low precision, as well as mixed precision, which is expected to have both the accuracy of high precision and the gains of speed from low precision.

Acknowledgements

This work was partially supported by the US National Science Foundation under grants: DMS-2051019 and DMS-2208294.

References

[1] Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, 1996.

[2] C. L. Epstein. *Introduction to the Mathematics of Medical Imaging, Second Edition*. SIAM, Philadelphia, PA, 2007.

[3] S. Gazzola, P. C. Hansen, and J. G. Nagy. IR tools: a MATLAB package of iterative regularization methods and large-scale test problems. *Numerical Algorithms*, 81(3):773–811, 2019.

[4] G. Gupta. Using tensor cores for mixed-precision scientific computing. NVIDIA Technical Blog, https://developer.nvidia.com/blog/tensor-cores-mixed-precision-scientific-computing/, Oct-2021.

[5] M. H. Gutknecht and S. Röllin. The Chebyshev iteration revisited. *Parallel Computing*, 28(2):263–283, 2002.

[6] A. Haidar, P. Wu, S. Tomov, and J. Dongarra. Investigating half precision arithmetic to accelerate dense linear system solvers. In *Proceedings of the 8th workshop on latest advances in scalable algorithms for large-scale systems*, pages 1–8, 2017.

[7] P. C. Hansen. *Discrete Inverse Problems: Insight and Algorithms*. SIAM, 2010.

[8] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49(6):409, 1952.

[9] N. J. Higham. The accuracy of floating point summation. *SIAM Journal on Scientific Computing*, 14(4):783–799, 1993.

[10] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2002.

[11] N. J. Higham and T. Mary. Mixed precision algorithms in numerical linear algebra. *Acta Numerica*, 31:347–414, 2022.

[12] N. J. Higham and S. Pranesh. Simulating low precision floating-point arithmetic. *SIAM Journal on Scientific Computing*, 41(5):C585–C602, 2019.

[13] P. Luszczek, J. Kurzak, I. Yamazaki, and J. Dongarra. Towards numerical benchmark for half-precision floating point arithmetic. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–5. IEEE, 2017.

[14] X. Meng, M. A. Saunders, and M. W. Mahoney. LSRN: A parallel iterative solver for strongly over-or underdetermined systems. *SIAM Journal on Scientific Computing*, 36(2):C95–C118, 2014.

[15] P. San Juan, R. Rodríguez-Sánchez, F. D. Igual, P. Alonso-Jordá, and E. S. Quintana-Ortí. Low precision matrix multiplication for efficient deep learning in nvidia carmel processors. *The Journal of Supercomputing*, 77(10):11257–11269, 2021.

[16] L. N. Trefethen and D. Bau III. *Numerical Linear Algebra*, volume 50. SIAM, 1997.

[17] H. Wang. A Chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Transactions on Graphics (TOG)*, 34(6):1–9, 2015.