Approximated PCA

Iteration 2

Rodrigo Arias

March 3, 2017

Computing eigenvalues and eigenvectors

1. Take the $n \times n$ target matrix $A = S$.
2. Compute a tridiagonal matrix T: $A = PTP^T$.
3. From T compute a diagonal matrix D: $T = QDQ^T$.
4. The eigenvalues of A are in the diagonal of D.
5. Compute the eigenvectors from D.

Computing a tridiagonal matrix is called **tridiagonalization**. For the diagonal, **diagonalization**. Several algorithms exists for both steps.

Tridiagonalization

The Householder algorithm transform a **symmetric** matrix A into a new pair of matrices P and T such that P is orthogonal, T is tridiagonal, and $A = PTP^T$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} = P \begin{pmatrix} t_{11} & t_{12} & & \\ t_{21} & t_{22} & t_{23} & \\ & t_{32} & t_{33} & t_{34} \\ & & t_{43} & t_{44} \end{pmatrix} P^T$$

Householder algorithm

The existing implementation is written in C, and uses only the following 6 operations in floating point arithmetic:

- Addition
- Substration
- Multiplication
- Division
- Square root
- Absolute value

Householder algorithm

The existing implementation is written in C, and uses only the following 6 operations in floating point arithmetic:

- Addition
- Substration
- Multiplication
- Division
- Square root
- Absolute value

To change the bit-width, those operations needed to be implemented for a specific mantissa length.

MPFR library

The MPFR library provides support to perform computations with custom **bit-width** floating point arithmetic.

`c = a + b` $\rightarrow$ `mpfr_add(c, a, b, RND);`

MPFR library

The MPFR library provides support to perform computations with custom **bit-width** floating point arithmetic.

`c = a + b` $\rightarrow$ `mpfr_add(c, a, b, RND);`

Problem: Algorithms had to be rewritten for each operation.

Design of the experiment

- Compute a golden result using very large bit-width.
- Starting from 2 bits up to 100 do:
- Run Householder with the same input data
- Compute the error with the golden result
- Repeat until 100 bits

Design of the experiment

First choose a random symmetric matrix, and compute Householder with great precision.
Then compare the results with lower precisions. Repeat the experiment many times to see the error distribution.

for $r = 1$ to 10000 **do**
  $A \leftarrow$ Random symmetric matrix.
  $G \leftarrow$ Householder of A with 500 bits.
  for $b = 2$ to 100 **do**
    $T \leftarrow$ Householder of A with b bits.
    $\epsilon_b \leftarrow$ Measure error between T and G
  end for
end for

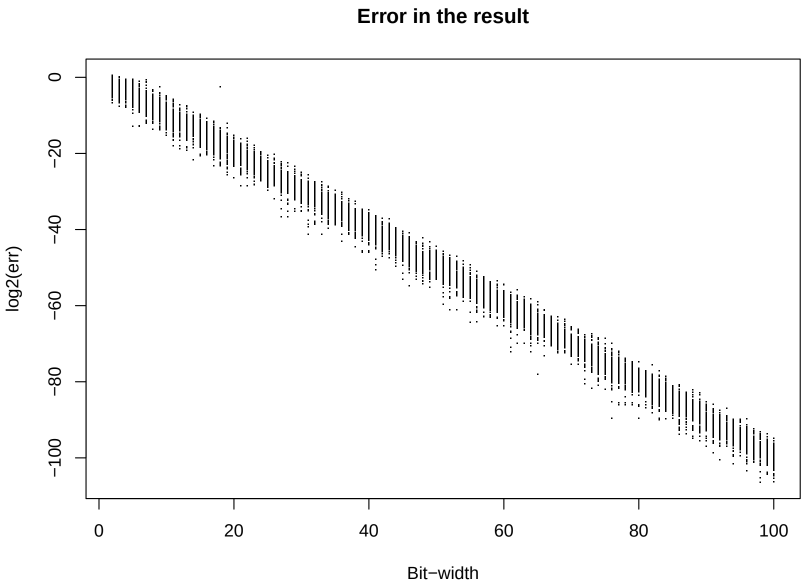
Measuring the error

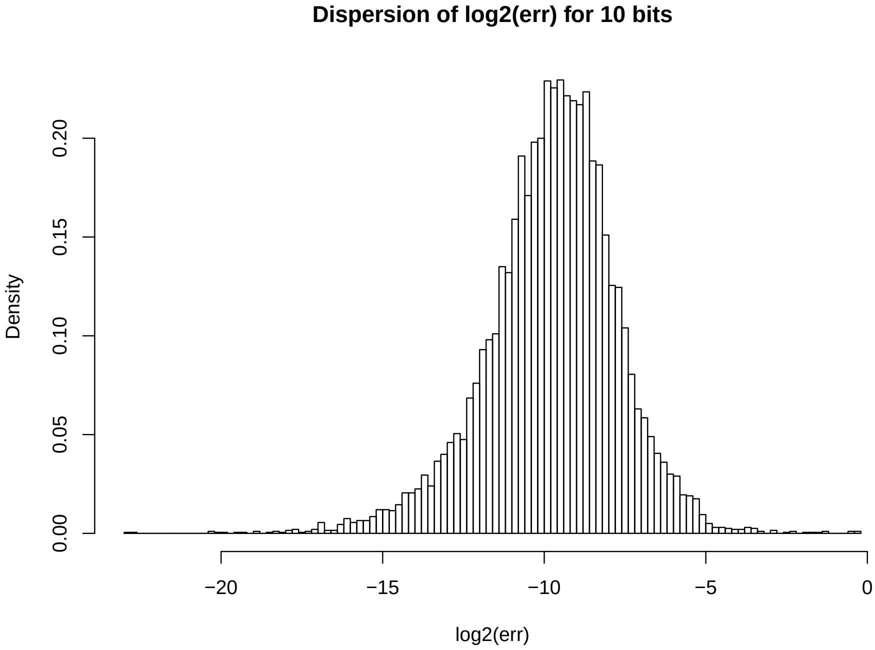
To measure the difference with the golden result, two methods have been used:

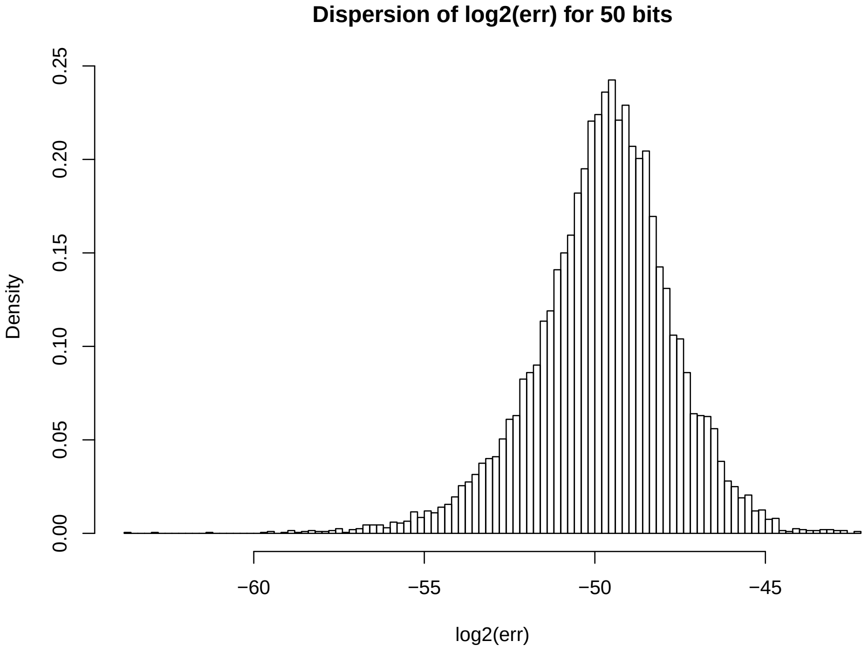
- Compute the 2-norm of the difference in the diagonal.
- Check the error in only one element.

In the plotted results, only the last method is shown. The 2-norm shows smaller errors.

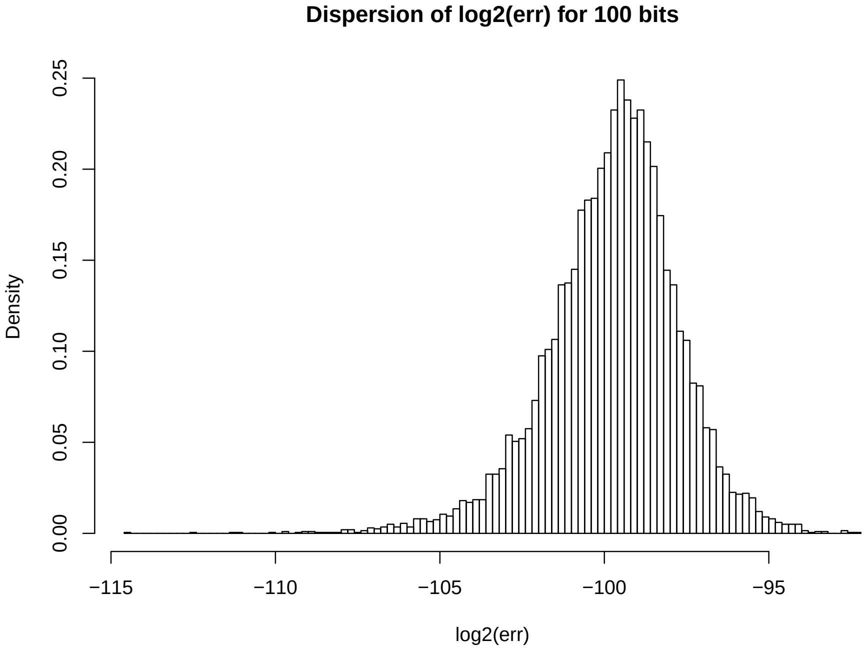
Results of the experiment




Dispersion of log2(err) for 10 bits
Density
0.00
0.05
0.10
0.15
0.20
−20
−15
−10
−5
0
log2(err)



Dispersion of log2(err) for 50 bits
Density
0.00
0.05
0.10
0.15
0.20
0.25
−60
−55
−50
−45
log2(err)


Dispersion of log2(err) for 100 bits
Density
0.00
0.05
0.10
0.15
0.20
0.25
−115
−110
−105
−100
−95
log2(err)

Observations

Let ϵ_b be the error in the result with width b, then the following hypothesis can be proposed:

- The mean of $\log_2(\epsilon_b) \approx -b$.
- The standard deviation ≈ 2.

Skip modifications

- The modification of the Householder algorithm to add a custom bit-width can be almost avoided using C++.
- The MPFR C++ library, uses a wrapper to replace automatically all the operations with the custom bit-width arithmetic.
- However, the documentation is almost inexistent, and not all capabilities are available.

Future steps

- Test the hyphothesis for the current results.
- Perform experiments in Householder with different precision in the variables.
- Determine how to use the MPFR wrapper to avoid modifications.

Possible application

PCA can be applied to satellite images:

- Satellites have different bands, so different layers of the same region are obtained.
- Those layers are correlated, and PCA can extract the main information in only a few uncorrelated layers.
- **Example:** from 6 bands, PCA could extract 99.3% of variance in only 3 layers.

Component	1	2	3	4	5	6
Percentage	88.82	17.62	2.94	0.38	0.18	0.05