Final project for the Course High Performance Programming, Philipp Noel von Bachmann

March 20, 2022

1 Introduction

Machine Learning and Data analysis are some of the most influencial fields in our current society. One of the main task in Machine Learning is to give a prediction based on some input variables. A common algorithm to use is linear-least-square, which finds the best fit for a linear model. Here, we need an efficient way to calculate this fit. In Data analysis, a common tool is the Principal Component analysis, which tries to reduce the dimensionality of the data in a way, that the reconstruction error gets minimized. PCA relies on an eigenvalue decomposition. For both methods, we can use the QR decomposition too speed up the computation.

2 Problem description

Suppose $A \in \mathbb{R}^{m \times n}$ is a real, square matrix. Then it can be shown that A can be decomposed into

$$A = QR \tag{1}$$

where Q is orthogonal and R is upper triangular. Finding $Q \in \mathbb{R}^{m \times m}$ and $R \in \mathbb{R}^{m \times n}$ is the task of the algorithm.

3 Solution

We will implement an algorithm known as **Givens Rotation**. This algorithm relies on construction a sequences of matrices G_i, such that when multiplying A with G_i, we get a new matrix with a zero at a predefined place. Choosing G_i such that we eliminate the lower diagonal of A, we end up with a R and by multiplying all G_i with Q. We will first show how to eliminate one value at the time by constructing G_i and then how to combine these.

3.1 Givens rotation

First, we define a Givens rotation matrix as

$$G(i,j,\theta) = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & c_{ii} & \cdots & -s_{ij} & & \\ & & \vdots & \ddots & \vdots & & \\ & & s_{ji} & \cdots & c_{jj} & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix} \tag{2}$$

where $c = \cos(\theta)$, $s = \sin(\theta)$ and any not filled out values are 0. Note: Therefore we can equally represent G by $G(i, j, c, s)$

3.2 Eliminating one value

Now the goal is to find a Givens rotation matrix to eliminate/set to 0 a specific A_{ij}. This is equivalent to finding θ such that

$$G(i,j,\theta)^T \begin{bmatrix} \times \\ \vdots \\ a \\ \vdots \\ b \\ \vdots \\ \times \end{bmatrix} = \begin{bmatrix} \times \\ \vdots \\ r \\ \vdots \\ 0 \\ \vdots \\ \times \end{bmatrix} \tag{3}$$

where $\times$ are arbitrary numbers $a, b \in \mathbb{R}$, $r = \sqrt{a^2 + b^2}$. A trivial solution would be

$$c = \frac{a}{r} \tag{4}$$

$$s = \frac{b}{r} \tag{5}$$

However, r is prone to overflow, so we can instead store it in a different way. If

- $|b| \geq |a|$:
  $t = \frac{a}{b}$, $s = \frac{sgn(b)}{\sqrt{1+t^2}}, c = st$
- else:
  $t = \frac{b}{a}$, $c = \frac{sgn(a)}{\sqrt{1+t^2}}, s = ct$

This leads to the following algorithm

Algorithm 1 computing Givens rotation to eliminate $A_{i,j}$

```
1: procedure GIVENS(A, i, j)
2:     choose k ∈ 0, ... i − 1
3:     compute G(i, k, c, s) with a = A_{k,j}, b = A_{i,j}
4:     return G
```

We note that we can choose $k \in 0, \ldots i-1$ arbitrary, the next section will explain this choice in more detail.

3.3 Order of elimination

The next question is which order of elimination to choose. Here, we note that if we compute the Givens rotation to eliminate $A_{i,j}$, we then need to multiply GA to eliminate the value. This multiplication affects the whole row i and k of A, where we choose k freely, see algorithm 1. In turn, if we don't want to discard other 0s (say $A_{i,l}$ for example) in those rows again, we need to make sure that $A_{k,l}$ is also 0.

As an more illustrative example, let us assume that we have $R = \begin{bmatrix} x & x & x \\ a & x & x \\ b = 0 & c & x \end{bmatrix}$. Now we want to set $c = 0$. This in turn modifies row 1,2 in R afterwards, which means a needs to be 0 in order to keep $b = 0$.

One commonly used scheme is to eliminate all columns after each other from 0 to n-1 and per column go from bottom to top, which looks like this for a 5×5 matrix:

$$\begin{bmatrix} x & x & x & x & x \\ 3 & x & x & x & x \\ 2 & 6 & x & x & x \\ 1 & 5 & 8 & x & x \\ 0 & 4 & 7 & 9 & x \end{bmatrix} \tag{6}$$

where the numbers indicate the order of elimination. It is easy to see that now all previously eliminated columns will stay 0.

3.4 Final algorithm

The elimination scheme 6 leads to the following algorithm.

Algorithm 2 QR factorization with Givens rotation

```
1: procedure FACTORIZE(A ∈ ℝ^{m×n})
2:     set R = A, Q = I
3:     for j in 1 to n do
4:         for i in m down to j + 1 do
5:             G = GIVENS(A, i, j) with k = i − 1
6:             set R = G(i, j, c, s)R, Q = QG(i, j, c, s)
```

4 Code

TODO: add

4.1 Check for correctness

5 Experiments and Results

5.1 Hardware description

5.2 Format

In the following section, we will show timing results for different matrix sizes. We denote the sizes by i, j, which we defined as decomposing the matrix $A \in \mathbb{R}^{i \times j}$.

5.3 Baseline Results

The following table shows the results for the base algorithm, given in file **qr_base.c**.

m \ n	10	25	50	100
10	0.0005	0.0008	0.001	0.0018
25	0.018	0.034	0.049	0.082
50	0.235	0.611	1.076	1.602
100	3.623	9.512	19.525	34.626

By varying n, the runtime increases slighlty below linear, because increasing n doesn't result in more values that need to get eliminated, but a more costly matrix multiplication in the update GR, see algorithm 2, line 6.

By varying m, the runtime increases superlinear, because we need to eliminate more values and the matrix multiplications become more expensive.

5.4 Optimizing the matmul

In general, we could optimize the matrix multiplication with common techniques like cache-blocking etc. However, if we take a closer look at the Givens rotation matrices, we see that they are very sparse. Additionally, we know in advance which entries of the givens rotation wil be nonzero. Thus we can optimize our matmul. First of all, we can only recalculate columns that will get modified by the multiplication. If we multiply GR, where $G = G(i, j, c, s)$, then we only need to modify rows i, j in R. Secondly, we know that for calculating those rows we only need to muliply by the c, s in the corresponding row of G, since all other values are 0.

We need further distinguish between left- (GR) and right-hand (QG) multiplication. We show the pseudo-code for left-hand, but right-hand is very similar.

Algorithm 3 optimized left-hand Matmul with Givens

1: **procedure** MATMUL($G = G(i, j, c, s)$, R)
2: copy the two rows i,j of R into temporary array $T = \begin{bmatrix} R_{i,:} \\ R_{j,:} \end{bmatrix}$
3: **for** k in columns of R **do**
4: $R_{i,k} = c \cdot T_{1,k} - s \cdot T_{2,k}$
5: $R_{j,k} = s \cdot T_{1,k} + c \cdot T_{2,k}$
return R

This leads to the following results:

m \ n	10	25	50
50	0.0003	0.0007	0.0011
500	0.019	0.040	0.079

We see that for $m = 50, n = 50$ we get a runtime of roughly $\frac{1}{1000}$ of the original runtime. However these new runtimes vary quite a bit and we would need to scale them up to get more stable, which is not feasible for the baseline.

5.5 Compiler flags

For i,j=500	Baseline	-O1	-O2	-O3	-03 -march=native -ffast-math
	0.747	0.167	0.137	0.134	0.119

5.6 Further serial optimizations

Tried those further optimizations

- Write pow calls out (compiler flag probably does it already)
- Exchanged div in computation of c, s by multiplication with $\frac{1}{r}$
- Declare the array sizes as constant

Those ones helped:

- Inline functions, took time down from 1.09 to 0.99 for i,j=1000
- maybe profile code
- DONE use compiler flags, best gcc -O3 -march=native -ffast-math
- DONE look if we can reduce arithmetic, eg no div in loop
- DONE inline functions
- loop unrolling
- no if statements
- change pow calls
- cache blocking
- make constants, especially i and j
- pure functions
- uautovectorization (included in 03)

5.7 Memory optimization

The improved matrix multiplication algorithm 3 shows that we only need to acces two rows per multiplication to perform the update of R, for Q we need to acces two columns however. Ultimately, we want to acces matrix elements as close to each other as possible, since by reading one element, we read the whole cache-line and therefore have faster acces to nearby elements in the following steps. Therefore, it makes sense to store R row-wise, and Q column-wise. In that way, we get the most optimal data locality.

both Q, R row-wise. For j=1000	i	100	500	1000
	baseline	0.0061	0.147	0.999
	cache optimized	0.0047	0.115	0.590

For i=1000	j	100	500	1000
	baseline	0.135	0.612	0.991
	cache optimized	0.067	0.339	0.609

5.8 Parallelization

5.8.1 Theory and Implementation

In order to parallelize our code, we first need to check which parts can be done in parallel. First, note that the computation of the Givens rotation in line 5 of algorithm 2 is a pure function, eg just depends on the inputs and doesn't have any side effects. Thus, we only need to focus on updating Q, R in line 6. Here, it is necessary to distinguish in two cases:

- For left-hand Givens rotation GR, if $G = G(i, j, c, s)$, only rows i, j of R get modified
-
- For righ-hand Givens rotation QG, if $G = G(i, j, c, s)$, only columns i, j of Q get modified

Consequently, two update steps rotating around $G(i_1, j_1, c_1, s_1)$ and $G(i_2, j_2, c_2, s_2)$ are independent, if i_1, i_2, j_1, j_2 are pairwise distinct. This means we need to make sure that if we set $R_{i,j}$ to 0 in one iteration, we can set any other elements in Row i, j to 0 in the same iteration, in order to not interfere between the computations.

In combination with the constrain that if we want to set $R_{i,j}$ to 0, $R_{i,:j}$ and $R_{i-1,:j}$ already need to be 0, (see Section 3.3) we arrive at the following scheme, shown for a 5×5 matrix:

$$\begin{bmatrix} x & x & x & x & x \\ 3 & x & x & x & x \\ 2 & 4 & x & x & x \\ 1 & 3 & 5 & x & x \\ 0 & 2 & 4 & 6 & x \end{bmatrix} \tag{7}$$

where the values denote in which iteration we can set the element to 0 without cross-interference. This shows that we can set $R_{2,0}$ and $R_{4,1}$ to 0 in parallel for example.

We arrive at the parallel algorithm:

Algorithm 4 parallel QR factorization

```
1: procedure FACTORIZE_PARALLEL(A ∈ ℝ^{m×n})
2:     set R = A, Q = I
3:     for it in 1 to (n − 1) ∗ 2 + m − n do
4:         In parallel:
5:         for i, j given by the iteration scheme 7 do
6:             compute the Givens rotation G(i, i − 1, c, s) eliminating R_ij
7:             set R = G(i, j, c, s)R, Q = QG(i, j, c, s)
```

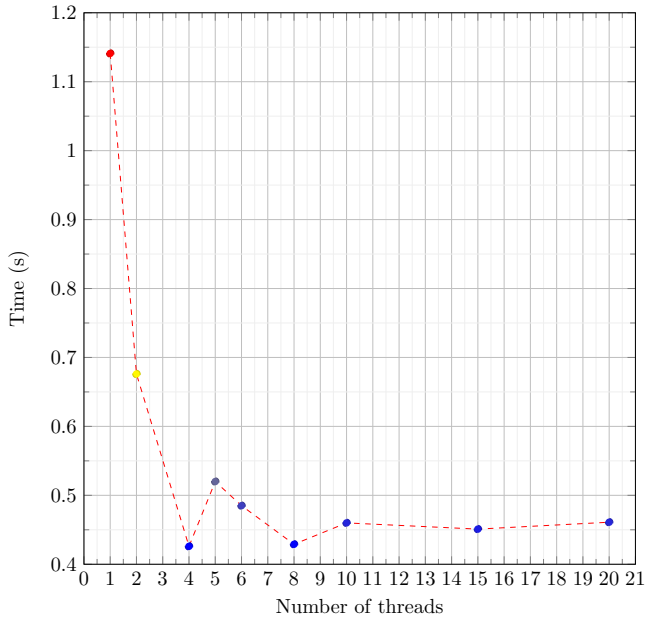
The code was getting parallelized with **OpenMP**. Since we have equal load in line 6+7 of algorithm 4, we don't need to use load balancing.

Looking at 7, we cannot expect a linear scaling with the number of threads. For example in iteration 0 and 1, we only have 1 element so there is no speedup by using more threads. More formally if $A \in \mathbb{R}^{m \times n}$, then we can have a maximum of $\min(m/2, n)$ computations running in parallel, achieved in the itertation where we set $R_{1,0}$ to 0.

5.8.2 Results

Because the computation of i, j is a bit more involved now, we get a slightly higher baseline with 1.137s compared to 0.986s for our serial code and $i, j = 1000$. However when

increasing the number of threads:



	10	100	1000	2000	4000
$j = 1000, i =$	0.031	0.023	0.425	4.125	58.866
$i = 1000, j =$	0.031	0.087	0.430	1.084	2.247

6 Memory optimization

Maybe try to store Q in column format and R in row format??

7 Random

- we can order them by affecting row and above
- best is probably to create tasks because upper rows require less work
- maybe need memory optimizations, but most of the operations seem to be near anyway

TODO:

- implement checks
- implement good storage of G
- implement efficient matmul of G
- implement one givens rotation
- implement outer loop
- serial optimizations
- parallelizations, probably with openmp

8 Experiments

9 Conclusion

10 References

$https://www.math.usm.edu/lambers/mat610/sum10/lecture9.pdf https://www.math.usm.edu/lamber$
$//en.wikipedia.org/wiki/Givens_rotation$