$$
\begin{aligned}
H^\top H = HH &= (I - 2\frac{vv^\top}{v^\top v})(I - 2\frac{vv^\top}{v^\top v}) \\
&= I^2 - 2I\frac{vv^\top}{v^\top v} - 2I\frac{vv^\top}{v^\top v} + 4\frac{vv^\top}{v^\top v}\frac{vv^\top}{v^\top v} \\
&= I - 4\frac{vv^\top}{v^\top v} + 4\frac{vv^\top vv^\top}{v^\top vv^\top v} \\
&= I - 4\frac{vv^\top}{v^\top v} + 4\frac{(v^\top v)vv^\top}{(v^\top v)^2} \\
&= I
\end{aligned}
$$

Remark 4.20. *Let $H_1, H_2, ..., H_n$ be Householder matrices as stated in example 4.2. Then by using the properties proofed in the previous remark the QR decomposition is given by:*

$$
\begin{aligned}
& H_n H_{n-1} \cdot \ldots \cdot H_1 A = R \\
\Leftrightarrow \quad & Q^\top A = R \\
\Leftrightarrow \quad & QQ^\top A = QR \\
\Leftrightarrow \quad & A = QR
\end{aligned}
$$

After demonstrating how a QR decomposition can be performed using Householder transformations, the next section is dedicated to the question of performance.

4.2.2. Performance

Even if runtime analyses of algorithm 4 have shown that a large part of the computing time had to be dedicated to the calculation of the gradients, the importance of finding the solution for the least squares problems must not be underestimated and will therefore be analysed as well. Since not only computing time but also the numerical stability of the grouping process is of great interest in practice, the runtimes and accuracies of different approaches for solving the least squares problem are compared in this section. Obviously, an implementation of a grouping algorithm which is numerically stable on the one hand and fast on the other hand with regard to the computing time is the preferred solution. Therefore seven different implementations will be compared using only functions that are available in R by default. Since these are standard functions in the area of linear algebra, it can be assumed that the functions

used are already highly optimized with respect to performance. In particular, some functions access implementations of the software packages LINPACK and LAPACK directly and therefore act only as wrappers. Examples of such R-functions are `solve` and `qr` whose default methods are interfaces to the LAPACK routines `DGESV` and `ZGESV` as well as `DQRDC` from the LINPACK package.

- `DGESV` computes the solution to a real system of linear equations $A * X = B$, where A is an N-by-N matrix and X and B are N-by-NRHS matrices [**lapack**].
- `DQRDC` uses householder transformations to compute the qr factorization of an n by p matrix x. column pivoting based on the 2-norms of the reduced columns may be performed at the users option [**linpack**].

The two software libraries LINPACK and LAPACK are written in Fortran and enjoy great popularity in many areas of application due to their efficient implementations with respect to memory usage and computational speed. Of course, there is also a big number of packages available on CRAN (Comprehensive R Archive Network) that offer different implementations for solving least squares problems. Since a comprehensive analysis of these implementations is not possible due to the daily growing number of packages available, the focus is on standard functions available in R. The basic R-functions used to perform the comparison are therefore `solve`, `backsolve`, `qr.solve`, `qr` and `t`. The notation used in algorithm 4 steps e) and iv. is simplified for the following comparison to the degree that the restriction to set P is not explicitly specified, i.e. $A = A^P$. The first three approaches pursue a solution without the calculation of a QR decomposition whereas the last four approaches are based on a QR decomposition.

Method 1: This method uses the calculation procedure for solving a least squares problem given in algorithm 4. By using the function `solve` with only one matrix as parameter the inverse of that matrix will be returned, i.e. `solve(A)` $\widehat{=} Ax = I \Leftrightarrow x = A^{-1}$.

$$s = (A^\top A)^{-1} A^\top b$$

```
solve((t(A) %*% A)) %*% t(A) %*% b
```

Method 2: For this method, the initial approach from the previous method is transformed in such a way that no inverse has to be calculated any more. The least squares problem is then given by:

$$(A^\top A)s = A^\top b$$

```
solve(t(A) %*% A, t(A) %*% b)
```

Method 3: The last approach using no QR decomposition solves the least squares problem using the `crossprod` function. According to the documentation `crossprod` should be slightly faster than a direct calculation via the transposed matrix. The approach is therefore identical to the one from method 2 except that a different implementation is used:

$$(A^\top A)s = A^\top b$$

```
solve(crossprod(A), crossprod(A,b)))
```

Method 4: This approach uses a QR decomposition and directly implements formula (4.12) derived in remark **??** for the solution of the least squares problem:

$$s = R^{-1}Q^\top b.$$

```
deco <- qr(A, LAPACK = FALSE)
solve(qr.R(deco)) %*% t(qr.Q(deco)) %*% b
```

Method 5: This approach again uses a QR decomposition and applies formula (4.13), which is a transformation of formula (4.12). Therefore no inverse has to be calculated to solve the least squares problem

$$Rs = Q^\top b.$$

```
deco <- qr(A, LAPACK = FALSE)
solve(qr.R(deco), t(qr.Q(deco)) %*% b)
```

Method 6: This method again uses a QR decomposition, but also makes use of the special form of the decomposition where R is an upper triangular matrix. The function `backsolve` solves a system of linear equations where the coefficient matrix is upper triangular [**R**]:

$$Rs = Q^\top b$$

```
deco <- qr(A, LAPACK = FALSE)
backsolve(qr.R(deco), t(qr.Q(deco)) %*% b)
```

Method 7: The last method uses a QR decomposition as well. The result of the QR decomposition is not saved in a temporary variable as before, but is passed directly to the `qr.solve` method.

$$QRs = b$$

```
qr.solve(qr(A, LAPACK = FALSE), b)
```

In order to determine how the number of columns of matrix A affects the run time, the following setup was chosen. Matrix A was defined as a square matrix with $n = 5000$, where the entries of A are uniformly distributed between 0 and 100000, i.e. $A \in R^{n \times n}$. The values of vector s are uniformly distributed between 0 and 1000. By defining matrix A and vector b, the result vector s is determined as well. For each of the seven methods, the number of columns used from matrix A was successively increased and then the least squares problem was solved. The results obtained in the test runs were produced with the following configuration:

- Processor: Intel® Core™ i7-6700
- RAM: 64GB
- R-version: 3.5.0

In order to obtain reliable results, all calculations were repeated ten times and the individual results averaged. Figure (4.4) shows how the time needed to solve the least squares problem increases with the number of columns used. It is important to note that due to illustration purposes the methods are presented separately based on their underlying approach, but both panels use the same

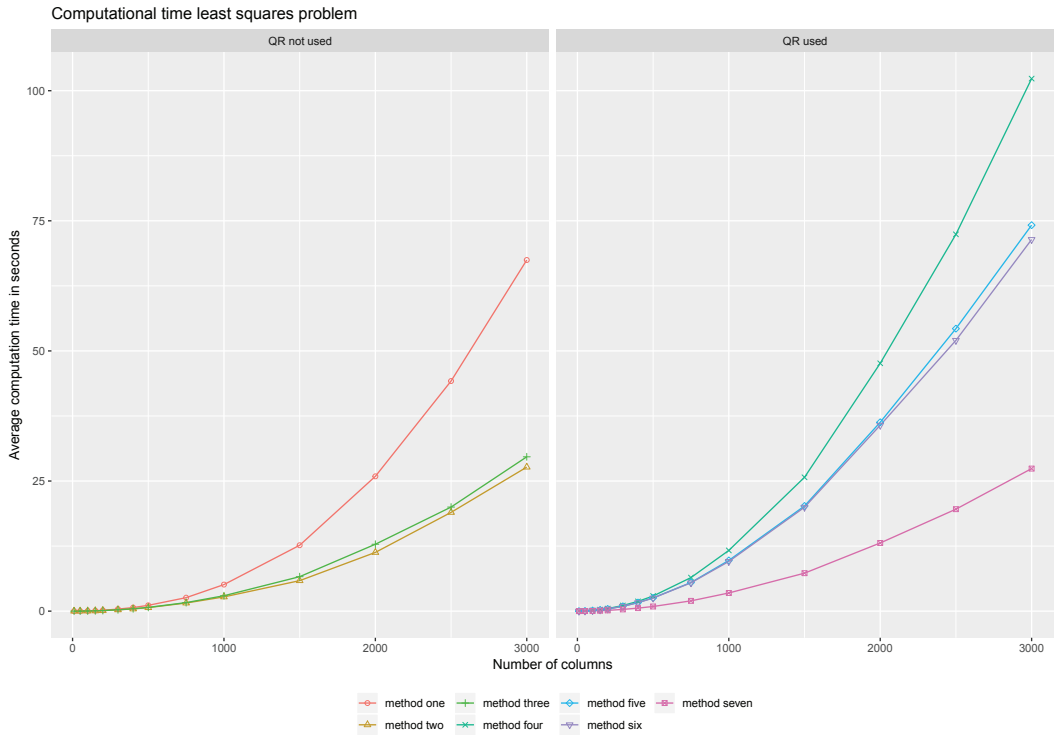

Figure 4.4.: Computation time for solving the least squares problem $As = b$ in seconds.

scaling. The left part of the graph shows the results of methods one to three, i.e. all those methods that do not use a QR decomposition to solve the least squares problem. Even if it is difficult to recognize in the left panel of figure (4.4) due to the scaling, the data shows that method one generally has significantly longer runtimes than method two and three even for small values of n. Starting from a column number of about 1000, it is clearly visible that the runtime for method one increases significantly faster than for the other two methods. This can be attributed to the fact that method one requires the explicit calculation of an inverse matrix which is not advisable. When an inverse is needed rather an LU composition should be performed [**lund1978hct**]. Moreover, there are no significant differences between method two and method three, although method two generally has slightly shorter runtimes. The usually existing speed advantage given in the help of the `crossprod` function could not be verified. In the right part of figure (4.4), the results of methods four to seven are shown, i.e. all those methods that use a QR decomposition. In general, it can be seen that, with the exception of method seven, all methods are considerably slower than those without QR decomposition. Method four shows significantly longer running times than all other methods, which is due to the fact that both, a QR decomposition and a matrix inversion must be performed. Methods five and six behave very similarly for the most part, with a small runtime advantage for method six being observed as the number of columns increases. Since the two methods are basically identical and method six uses only the special structure of the upper triangular matrix, it can be assumed that this advantage is only effective for larger systems of equations. The method that distinguishes significantly from the others in terms of runtime is method seven. Over the entire range, its computational time is significantly shorter than that of the other methods. Since method seven is the only method that does not temporarily save the result of the QR decomposition but directly processes it, it seems reasonable to conclude that this already results in a significant performance advantage. Considering the results regardless of whether a QR decomposition was used or not, the following summary results:

- Methods two, three and seven are to be classified as equivalent in terms of runtime and are also the fastest altogether.
- Methods one, five and six have similar runtimes, with method one always performing best.
- The slowest method by far is method four.

After analysing how the runtimes of the different approaches behave in relation

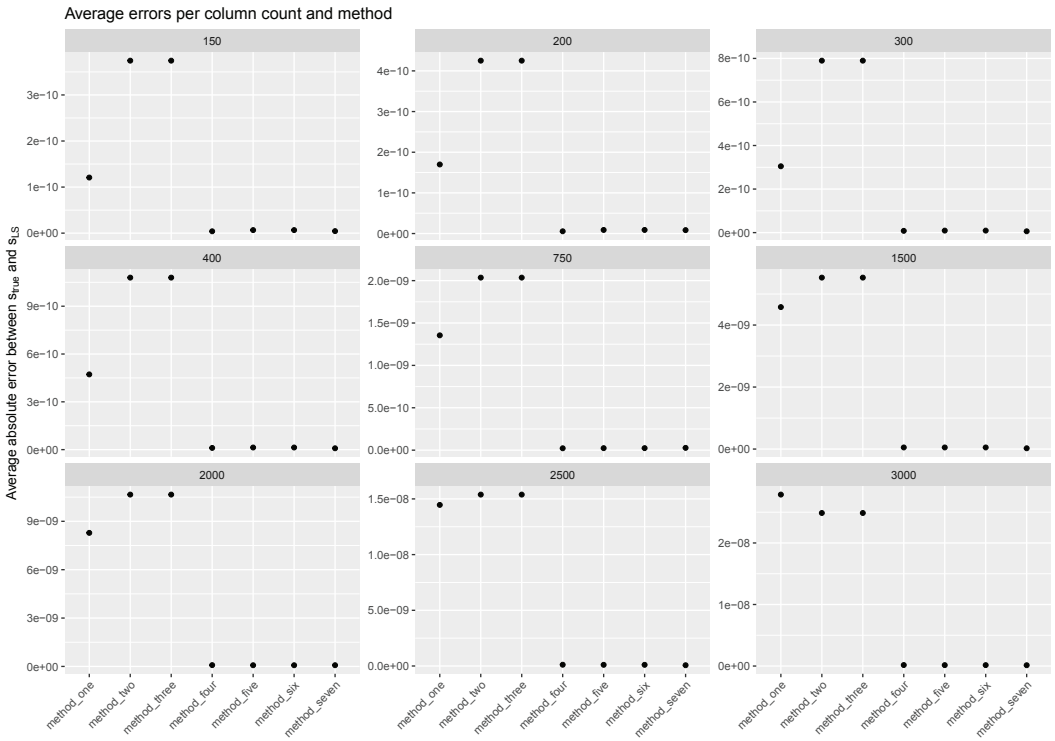

Figure 4.5.: Comparison of average absolute errors based on method and number of columns used from matrix A

to the number of columns, there is still a need to analyse their accuracy. As mentioned above, methods two, three and seven scale best with an increasing number of columns of matrix A. Since method seven uses a QR decomposition, while methods two and three do not use it, an important aspect is to compare their accuracy. Figure (4.5) shows how much the calculated solutions s_{LS} deviate on average in absolute values from the actual values s_{true}, i.e. $err_{avg} = \sum_{i=1}^{n} \frac{1}{n} |s_{LS}^i - s_{true}^i|$. Each box represents the deviations of all seven methods for a fixed number of columns. For example, the box in the upper right corner of figure (4.5) shows the results for a matrix A with 300 columns. In contrast to figure (4.4), the data points for the column numbers 10, 50, 100, 500 and 1000 were not shown. This is because by omitting the plots a better readability and arrangement was possible without suffering a loss of information. It should therefore be noted once again that all missing boxes basically provide the same results as those shown here and therefore do not provide any information gain. Considering the nine panels, a clear pattern can be identified. In each individual case methods one to three show on average

higher deviations than methods four to seven. Taking the definitions of the methods into account it can be revealed that methods four to seven are based on a QR decomposition. Within those four methods no significant difference can be seen with respect to the deviations. The results of the simulation therefore suggest that all methods based on a QR decomposition are equivalent in terms of accuracy. Methods one to three, which don't use a QR decomposition, show a differentiated picture. Methods two and three show similar deviations, which is not surprising because of their definition, as they differ only in the functions `t` and `crossprod`. For method one, there are strong indications that the average deviation increases with the number of columns. In the case that matrix A has 150 columns, the average differences between s_{LS} and s_{true} are significantly smaller for method one than for method two or method three. If the number of columns is increased step by step, this difference becomes smaller and smaller. Looking at the case where matrix A has 3000 columns, it is evident that method one has the highest deviation of all tested methods. Since method one is the only method which uses the calculation of an inverse, the results leads to the conclusion that calculating an inverse becomes more unstable as the number of columns increases. This shows, as already mentioned in previous sections, that an algorithm for finding the solution of the least squares problem should be implemented, which doesn't rely on inverse matrix calculations. After both the execution time and the accuracy of the seven approaches have been examined, the following picture emerges:

- In all test cases considered, it could be proven that those methods that use a QR decomposition show the highest accuracy.
- Among those methods that use a QR decomposition, the fastest is the one that does not temporarily store the QR results but directly processes them.
- Looking at the fastest methods of each group (QR decomposition used or not) no significant difference can be found.
- If an inverse matrix is calculated, the more columns the matrix has, the worse the overall accuracy gets for the test sample.
- The calculation time required to solve the least squares problem scales exponentially with the number of columns.

Taking all aspects into account, method seven is the preferred method in terms of both runtime and accuracy.

Betrachtet man Fortran sieht man dass die QR-Zerlegung nicht immer neu berechnet wird sondern in jedem Schritt die QR-Zerlegung nur adaptiert wird. Soll das aufgenommen werden?