Statistical Computing: Numerical computations

- Numbers on the real line $\mathbb{R}$ cannot all be represented exactly in a computer.
- Usually, we use *floating point storage*, which uses a fixed finite amount of bit $(0/1)$ storage to represent a subset of the real numbers.

Double precision floating point numbers (Standard: IEEE 754)

Double precision means that we have 64 bits (0/1) at our disposal, split into

Bits	Interpretation
1	$\text{sign}(x) \in \{-1, +1\}$
11	$\text{exponent}(x) \in [-1022, +1023]$
52	$\text{fraction}(x) \in [0, 1)$

These components combine to define the *computed* version of a number x as

$$\text{comp}(x) = \text{sign} \cdot (1 + \text{fraction}) \cdot 2^{\text{exponent}},$$

and $|x| < 2^{-1022}$ is treated specially.

Relative storage error

The closest floating point number $\mathrm{comp}(x)$ to a real value $x \not\approx 0$ has bounded *relative error*:

$$\frac{|\,\mathrm{comp}(x) - x|}{|x|} \lesssim \epsilon_0,$$

where ϵ_0 is called the *machine precision*.

Machine precision

The machine precision is the smallest number such that $\mathrm{comp}(1 + \epsilon_0) > 1$.
In R the value can be accessed as `.Machine$double.eps`, and is $\epsilon_0 \approx 2.220446 \cdot 10^{-16}$.

This allows a simple model for handling numerical approximation errors:

$$\mathrm{comp}(x) = (1 + \epsilon)x,$$

for some ϵ such that $|\epsilon| \lesssim \epsilon_0$.

For simple storage of numbers, the error is quite small, but we need to beware of amplifying the error when we perform computations!

Approximation errors for numerical derivatives

Finite differences

The derivative of a function $f(\theta)$ can be approximated by the asymmetric finite difference

$$\mathrm{comp}\left\{\frac{\mathrm{comp}[f(\mathrm{comp}[\theta+h])]-\mathrm{comp}[f(\theta)]}{h}\right\}\approx\frac{f(\theta+h)-f(\theta)}{h}\approx f'(\theta).$$

How small should we choose h in order to minimise the approximation error?

Local polynomial approximations will help in analysing the errors:

Taylor's theorem

For a function $f:\mathbb{R}\mapsto\mathbb{R}$ with two continuous derivatives near a $\theta\in\mathbb{R}$,

$$f(\theta+h)=f(\theta)+hf'(\theta+t_1h), \text{ and}$$

$$f(\theta+h)=f(\theta)+hf'(\theta)+\frac{h^2}{2}f''(\theta+t_2h),$$

for some $t_1\in[0,1]$ and $t_2\in[0,1]$.

First order difference approximation error

Assume that θ is stored exactly.
First, consider the error in computing $f(\theta + h)$:

$$\begin{aligned}\mathrm{comp}\{f(\mathrm{comp}[\theta + h])\} &= (1+\epsilon_f) f([1+\epsilon_\theta][\theta + h]) \\ &\approx (1+\epsilon_f)\left[f(\theta+h) + \epsilon_\theta(\theta+h) f'(\theta + t_1 h)\right] \\ &\approx f(\theta+h) + \epsilon_f f(\theta+h) + \epsilon_\theta \theta f'(\theta + t_1 h) + \text{h.o.t.},\end{aligned}$$

for some $|\epsilon_f| \lesssim \epsilon_0$ and $|\epsilon_\theta| \lesssim \epsilon_0$.
Assume that $f(\cdot)$ is bounded near θ, with $|f(\cdot)| \leq L_0$, $|f'(\cdot)| \leq L_1$, and $|f''(\cdot)| \leq L_2$. Then

$$\left| \frac{\mathrm{comp}\{f(\mathrm{comp}[\theta + h])\} - \mathrm{comp}\{f(\theta)\}}{h} - \frac{f(\theta+h) - f(\theta)}{h} \right| \lesssim \frac{\epsilon_0(2L_0 + |\theta| L_1)}{h}.$$

First order difference approximation error (cont.)

Applying Taylor's theorem to the exact finite difference leads to

$$\left|\frac{f(\theta+h)-f(\theta)}{h}-f'(\theta)\right|=\frac{h}{2}\left|f''(\theta+t_2h)\right|\leq\frac{hL_2}{2}.$$

Using the triangle inequality ($|E_1+E_2|\leq|E_1|+|E_2|$), we get the total error bound,

$$\left|\frac{\mathrm{comp}\{f(\mathrm{comp}[\theta+h])\}-\mathrm{comp}\{f(\theta)\}}{h}-f'(\theta)\right|\lesssim\frac{\epsilon_0(2L_0+|\theta|L_1)}{h}+\frac{hL_2}{2},$$

which is minimised for $h=\sqrt{\epsilon_0\frac{4L_0+2|\theta|L_1}{L_2}}\propto\epsilon_0^{1/2}$.

Note:
If θ wasn't stored exactly, the additional error term would be

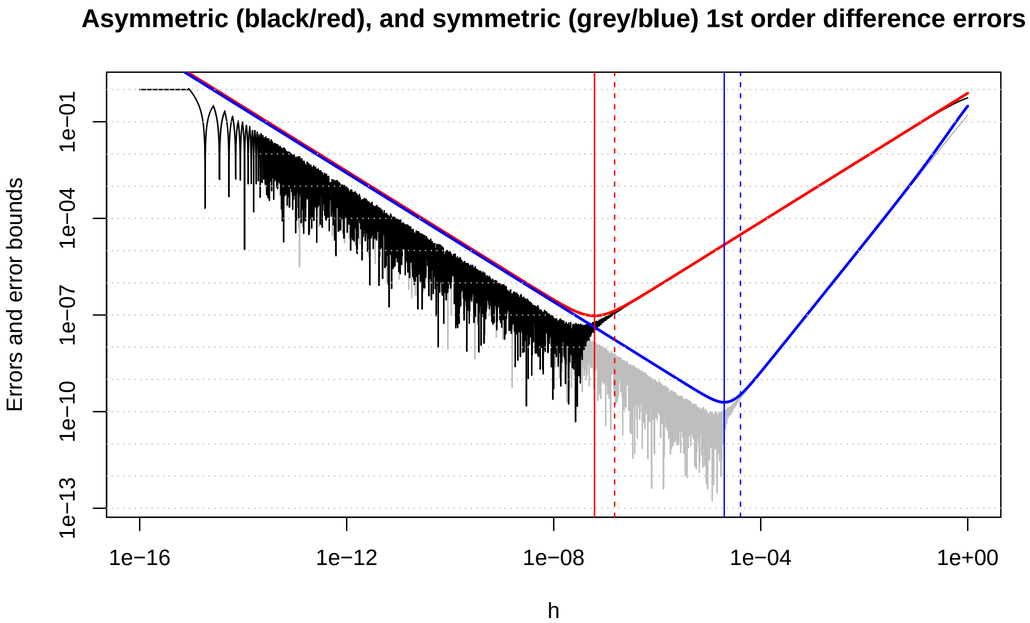
$$|f'(\mathrm{comp}[\theta])-f'(\theta)|\approx|f'(\theta)+\epsilon_3\theta f''(\theta+t_3h)-f'(\theta)|\lesssim\epsilon_0|\theta|L_2,$$

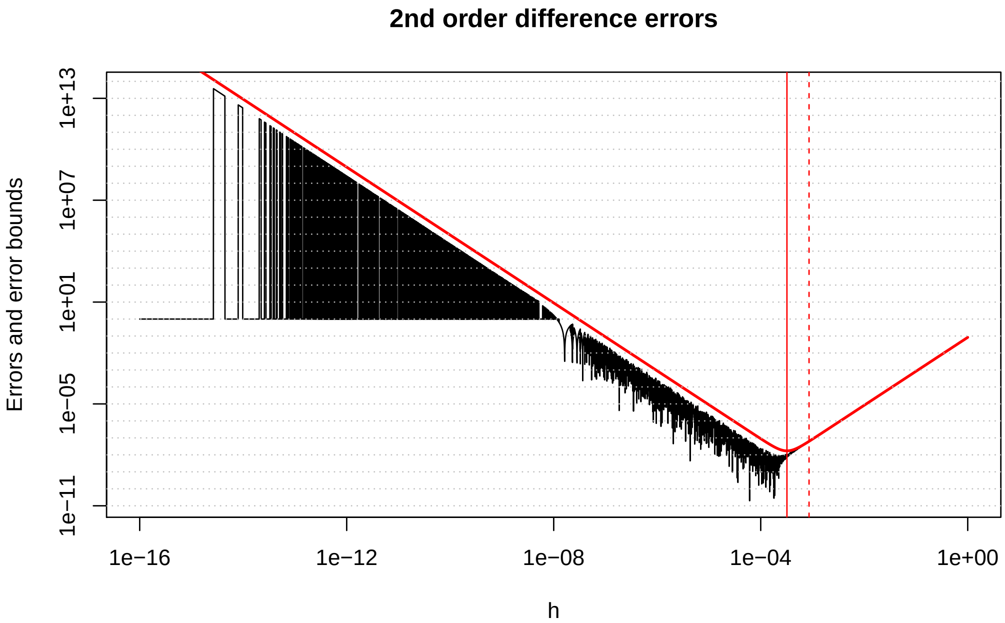
which doesn't depend on h.

Optimal stepsize for finite differences

Let L_k be bounds for the k:th derivatives around θ. The errors from *floating point cancellation* and *Taylor series truncation* can be bounded and minimised by choosing the *step size* h:

- Asymmetric first order differences for $f'(\theta)$, using $f(\theta)$ and $f(\theta+h)$, gives the bound
$$\lesssim \frac{\epsilon_0(2L_0+|\theta|L_1)}{h}+\frac{hL_2}{2}, \quad \text{which is minimised for } h=\sqrt{2\epsilon_0\frac{2L_0+|\theta|L_1}{L_2}}\sim\epsilon_0^{1/2}.$$
- Symmetric first order differences for $f'(\theta)$, using $f(\theta-h)$ and $f(\theta+h)$, gives the bound
$$\lesssim \frac{\epsilon_0(L_0+|\theta|L_1)}{h}+\frac{h^2L_3}{6}, \quad \text{which is minimised for } h=\left(3\epsilon_0\frac{L_0+|\theta|L_1}{L_3}\right)^{1/3}\sim\epsilon_0^{1/3}.$$
- 2nd order differences for $f''(\theta)$, using $f(\theta-h)$, $f(\theta+h)$, and $f(\theta)$, gives the bound
$$\lesssim \frac{\epsilon_0(4L_0+2|\theta|L_1)}{h^2}+\frac{h^2L_4}{12}, \quad \text{which is minimised for } h=\left(24\epsilon_0\frac{2L_0+|\theta|L_1}{L_4}\right)^{1/4}\sim\epsilon_0^{1/4}.$$
- Approximate rule of thumb: Plugin $L_k\equiv 1$ and a representative $|\theta|$ value.


Asymmetric (black/red), and symmetric (grey/blue) 1st order difference errors
Errors and error bounds
1e−01
1e−04
1e−07
1e−10
1e−13
1e−16
1e−12
1e−08
1e−04
1e+00
h


2nd order difference errors
Errors and error bounds
1e+13
1e+07
1e+01
1e−05
1e−11
1e−16
1e−12
1e−08
1e−04
1e+00
h

Numerics for Least Squares estimation of linear models

- In matrix/vector form, we write linear models for
    - observations $\boldsymbol{y} = \left[y_1, \dots, y_n\right]^\top : n \times 1$,
    - explanatory/predictor covariates $\boldsymbol{X} = \left[\boldsymbol{x}_1, \dots, \boldsymbol{x}_p\right] : n \times p$, $n > p$,
    - parameters $\boldsymbol{\beta} = \left[\beta_1, \dots, \beta_p\right]^\top : p \times 1$, and
    - observation noise $\boldsymbol{e} = \left[e_1, \dots, e_n\right]^\top : n \times 1$,

  as

$$\boldsymbol{y} = \sum_{k=1}^{p} \boldsymbol{x}_k \beta_k + \boldsymbol{e} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{e}.$$

- The least squares estimate of $\boldsymbol{\beta}$ is *in theory* given by $\widehat{\boldsymbol{\beta}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$.
- We will first analyse the floating point errors involved in $\boldsymbol{X}^\top \operatorname{comp}(\boldsymbol{y})$ and $(\boldsymbol{X}^\top \boldsymbol{X})^{-1} \operatorname{comp}(\boldsymbol{X}^\top \boldsymbol{y})$.
- Then we will develop a method for computing $\widehat{\boldsymbol{\beta}}$ that does *not* involve inverting any matrices, and avoids unneccesarily amplifying the floating point errors.
- We need some analytical tools!

Singular Value Decomposition (SVD, `svd(A)` in R)

For any rectangular matrix $\boldsymbol{A} \neq \boldsymbol{0}$, $n \times m$, there exist matrices $\boldsymbol{U} : n \times p$, $\boldsymbol{V} : m \times p$, and $\boldsymbol{D} = \operatorname{diag}\{d_1, \dots, d_p\}$ such that

$$\begin{aligned} &\boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}_p, \quad \text{i.e. the column vectors in } \boldsymbol{U} = [\boldsymbol{u}_1, \dots, \boldsymbol{u}_p] \text{ are orthonormal,} \\ &\boldsymbol{V}^\top \boldsymbol{V} = \boldsymbol{I}_p, \quad \text{i.e. the column vectors in } \boldsymbol{V} = [\boldsymbol{v}_1, \dots, \boldsymbol{v}_p] \text{ are orthonormal, and} \\ &\quad d_1 \geq d_2 \geq \cdots \geq d_r > 0 = d_{r+1} = \cdots = d_p, \end{aligned}$$

for some $r \leq p = \min(n, m)$, and

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top = \sum_{k=1}^{r} d_k \boldsymbol{u}_k \boldsymbol{v}_k^\top .$$

The value r is the *rank* of A. A matrix with $\operatorname{rank}(A) = p$ is said to have *full rank*.

The column vectors of $\boldsymbol{U}$ and $\boldsymbol{V}$ are called the left and right *singular vectors* of $\boldsymbol{A}$, and the corresponding $d_1, \dots, d_p$ are the *singular values* of $\boldsymbol{A}$.

The SVD algorithm is also used for *dimension reduction* in *Principal Component Analysis*.

The *complete* SVD expands $\boldsymbol{U}$ and $\boldsymbol{V}$ to square and completely orthogonal matrices $\begin{bmatrix}\boldsymbol{U} & \boldsymbol{U}_0\end{bmatrix}$ and $\begin{bmatrix}\boldsymbol{V} & \boldsymbol{V}_0\end{bmatrix}$ as needed, and expands $\boldsymbol{D}$ with zeros.
For a *tall* matrix with full rank, $n > m = p = r$. The complete decomposition is

$$\boldsymbol{A} = \begin{bmatrix}\boldsymbol{U} & \boldsymbol{U}_0\end{bmatrix} \begin{bmatrix}\boldsymbol{D} \\ \boldsymbol{0}\end{bmatrix} \boldsymbol{V}^\top$$

and

$$\begin{aligned} \boldsymbol{U}^\top\boldsymbol{U} &= \boldsymbol{I}_p, \qquad \boldsymbol{U}_0^\top\boldsymbol{U}_0 = \boldsymbol{I}_{n-p}, \qquad \boldsymbol{U}^\top\boldsymbol{U}_0 = \boldsymbol{0}, \\ \boldsymbol{U}\boldsymbol{U}^\top + \boldsymbol{U}_0\boldsymbol{U}_0^\top &= \boldsymbol{I}_n, \\ \boldsymbol{V}^\top\boldsymbol{V} = \boldsymbol{V}\boldsymbol{V}^\top &= \boldsymbol{I}_m \end{aligned}$$

These identities allow us to split a vector $\boldsymbol{w} \in \mathbb{R}^n$ into a component $\boldsymbol{w}_A$ that lies within the vector space spanned by the columns of $\boldsymbol{A}$, and a component $\boldsymbol{w}_0$ orthogonal to $\boldsymbol{A}$:

$$\boldsymbol{w} = \boldsymbol{U}\boldsymbol{U}^\top\boldsymbol{w} + \boldsymbol{U}_0\boldsymbol{U}_0^\top\boldsymbol{w} = \boldsymbol{w}_A + \boldsymbol{w}_0, \qquad \boldsymbol{A}^\top\boldsymbol{w}_A = \boldsymbol{A}^\top\boldsymbol{w}, \qquad \boldsymbol{A}^\top\boldsymbol{w}_0 = \boldsymbol{0}.$$

Vector length amplification

Multiplying with $\boldsymbol{A}^\top$ leads to different vector length amplification depending on in which direction $\boldsymbol{w}$ is pointing:

$$\begin{aligned}\|\boldsymbol{A}^\top\boldsymbol{w}\| &= \sqrt{\boldsymbol{w}^\top\boldsymbol{A}\boldsymbol{A}^\top\boldsymbol{w}} = \sqrt{\boldsymbol{w}^\top\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top\boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^\top\boldsymbol{w}} = \sqrt{\boldsymbol{w}^\top\boldsymbol{U}\boldsymbol{D}^2\boldsymbol{U}^\top\boldsymbol{w}}\\ &= \|\boldsymbol{D}\boldsymbol{U}^\top\boldsymbol{w}\| = \|\boldsymbol{D}\boldsymbol{U}^\top\boldsymbol{w}_A\|\end{aligned}$$

The norm is minimised when $\boldsymbol{w}_A = \boldsymbol{u}_p\|\boldsymbol{w}_A\|$ and maximised when $\boldsymbol{w}_A = \boldsymbol{u}_1\|\boldsymbol{w}_A\|$, so that

$$d_p\|\boldsymbol{w}_A\| \leq \|\boldsymbol{A}^\top\boldsymbol{w}\| \leq d_1\|\boldsymbol{w}_A\|.$$

Each component of $\mathrm{comp}(\boldsymbol{w})$ has a relative error ϵ_i, $|\epsilon_i| \lesssim \epsilon_0$.

$$\|\,\mathrm{comp}(\boldsymbol{w}) - \boldsymbol{w}\| = \sqrt{\sum_{i=1}^n (\epsilon_i w_i)^2} \leq \max_{i\in\{1,\ldots,n\}}(|\epsilon_i|)\sqrt{\sum_{i=1}^n w_i^2} \lesssim \epsilon_0\|\boldsymbol{w}\|.$$

For $\|[\mathrm{comp}(\boldsymbol{w}) - \boldsymbol{w}]_A\|$, we'll ignore the additional $\sqrt{n}$ term from $\lesssim \epsilon_0(\|\boldsymbol{w}_A\| + \sqrt{n})$.

Numerical matrix multiplication error

Numerical error propagation: condition numbers

Let $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top$ be the SVD of $\boldsymbol{X}$, and we will assume $\boldsymbol{X}$ has full rank.
Given that $\boldsymbol{X}^\top \boldsymbol{y} \neq \boldsymbol{0}$, the relative error of $\boldsymbol{X}^\top \boldsymbol{y}$ is

$$\frac{\|\boldsymbol{X}^\top \mathrm{comp}(\boldsymbol{y}) - \boldsymbol{X}^\top \boldsymbol{y}\|}{\|\boldsymbol{X}^\top \boldsymbol{y}\|} = \frac{\|\boldsymbol{X}^\top [\mathrm{comp}(\boldsymbol{y}) - \boldsymbol{y}]\|}{\|\boldsymbol{X}^\top \boldsymbol{y}\|} \leq \frac{d_1 \|[\mathrm{comp}(\boldsymbol{y}) - \boldsymbol{y}]_U\|}{d_p \|\boldsymbol{y}_U\|} \lesssim \epsilon_0 \kappa(\boldsymbol{X}),$$

where $\kappa(\boldsymbol{X}) = d_1/d_p$ is the *condition number* of $\boldsymbol{X}$.

For the second step of $\widehat{\boldsymbol{\beta}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \mathrm{comp}(\boldsymbol{X}^\top \boldsymbol{y})$,

$$(\boldsymbol{X}^\top \boldsymbol{X})^{-1} = (\boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^\top \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top)^{-1} = (\boldsymbol{V}\boldsymbol{D}^2\boldsymbol{V}^\top)^{-1} = \boldsymbol{V}\boldsymbol{D}^{-2}\boldsymbol{V}^\top$$

This looks like another SVD, except for a reverse order of the singular values, $1/d_p^2, \ldots, 1/d_1^2$.
The same bounds as before apply, so that the condition number is
$\kappa[(\boldsymbol{X}^\top \boldsymbol{X})^{-1}] = d_1^2/d_p^2 = \kappa(\boldsymbol{X})^2$. The resulting relative computational error is

$$\frac{\|(\boldsymbol{X}^\top \boldsymbol{X})^{-1} \mathrm{comp}(\boldsymbol{X}^\top \boldsymbol{y}) - (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}\|}{\|(\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}\|} \lesssim \epsilon_0 \kappa(\boldsymbol{X})^2.$$

Numerical example

```
n <- 100000
X <- cbind(1, (1:n), (1:n)^2) # 3 columns, (badly!) defining a quadratic regression curve model

## Compute the condition number
s <- svd(X) # Gives a list(u, d, v) for U, D=diag(d), and V
print(condition_number <- max(s$d) / min(s$d))

## [1] 13416843918

## Relative error bound for X'y
.Machine$double.eps * condition_number

## [1] 2.979138e-06

## Relative error bound for (X'X)^(-1)
.Machine$double.eps * condition_number^2

## [1] 39970.63
```

Numerical solves

Let's investigate a case without measurement noise, so that the solution in a perfect would would be exactly equal to the true parameters:

```
beta_true <- c(1, 1, 1)
y <- X %*% beta_true # '%*%' is the matrix multiplication operator in R
```

A direct solve of the linear system $\boldsymbol{X}^\top \boldsymbol{X}\widehat{\boldsymbol{\beta}} = \boldsymbol{X}^\top \boldsymbol{y}$ (the so called *normal equations*) fails:

```
beta1 <- solve(t(X) %*% X, t(X) %*% y)

## Error in solve.default(t(X) %*% X, t(X) %*% y):  system is computationally singular:
reciprocal condition number = 5.5549e-21
```

Using the computed SVD, $\widehat{\boldsymbol{\beta}} = \boldsymbol{V}\boldsymbol{D}^{-1}\boldsymbol{U}^\top \boldsymbol{y}$:

```
beta2 <- s$v %*% ((t(s$u) %*% y) / s$d)
vec_norm(beta2 - beta_true) / vec_norm(beta_true) # You'll define vec_norm() in Lab 5!

## [1] 6.450106e-05
```

The error is a factor ~ 20 larger than our estimated bound
$\epsilon_0 \kappa(\boldsymbol{X}) = \epsilon_0 \kappa(\boldsymbol{V}\boldsymbol{D}^{-1}\boldsymbol{U}^\top) = 2.9791378 \times 10^{-6}$.
Note: This is reasonable. What issues did our analysis ignore?

QR decomposition

We introduced the SVD mainly to help with our theoretical analysis, but it is expensive to compute.
For least squares problems, the main alternative is the following method:

QR decomposition (here only for tall matrices)

For any square or tall matrix $\boldsymbol{A} : n \times m$, $n \geq m$, there exist matrices $\boldsymbol{Q} : n \times m$ and $\boldsymbol{R} : m \times m$, such that $\boldsymbol{Q}^\top \boldsymbol{Q} = \boldsymbol{I}_m$ and $\boldsymbol{R}$ is upper triangular, and $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}$.

The least squares solution based on $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{R}$ becomes

$$\widehat{\boldsymbol{\beta}} = (\boldsymbol{R}^\top \boldsymbol{Q}^\top \boldsymbol{Q}\boldsymbol{R})^{-1} \boldsymbol{R}^\top \boldsymbol{Q}^\top \boldsymbol{y} = \boldsymbol{R}^{-\top} \boldsymbol{Q}^\top \boldsymbol{y} \qquad \text{(if } \boldsymbol{X} \text{ has full rank, } \boldsymbol{R} \text{ is invertible)}$$

i.e. one matrix multiplication with $\kappa(\boldsymbol{Q}) = 1$ and one triangular linear system solve, $\kappa(\boldsymbol{R}) = \kappa(\boldsymbol{X})$.

```r
beta3 <- qr.solve(X, y)
vec_norm(beta3 - beta_true) / vec_norm(beta_true)

## [1] 3.277137e-05
```

About half the error of the SVD method. What about the speed?

Computational cost

- For large models, computational speed and memory usage are vital issues.
- Choosing the right algorithm can mean the difference of waiting for a few seconds and waiting for several weeks!

```
n <- 100000
m <- 50
X <- matrix(rnorm(n * m), n, m)
rbind(
  system.time({ svd(X) }),
  system.time({ qr(X) })
)

##      user.self sys.self elapsed user.child sys.child
## [1,]     0.853    0.040   0.893          0         0
## [2,]     0.284    0.032   0.316          0         0
```

Both methods take $\propto n^3$ operations to compute for a wide range of least squares problems, but QR is consistently around a factor 3 faster than SVD.
In R, the `lm()` function uses QR decomposition internally.

Summary

- If we're not careful, the finite computer representation error may be amplified by computations.
- Theoretically correct formulas are not necessarily appropriate to compute directly as written.
- Numerical matrix decomposition methods and method parameters chosen by minimising error bounds can help minimise numerical errors.
- Large condition numbers may need manual intervention; can we formulate a linear statistical model in more than one way? (See Lab 5 for examples of simple condition number reduction methods.)