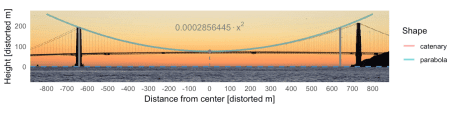
**Abstract:**

In this post we use R’s capabilities to solve nonlinear equation systems in order to answer an extension of the hanging cable problem to suspension bridges. We then use R and ggplot to overlay the solution to an image of the Golden Gate Bridge in order to bring together theory and practice.



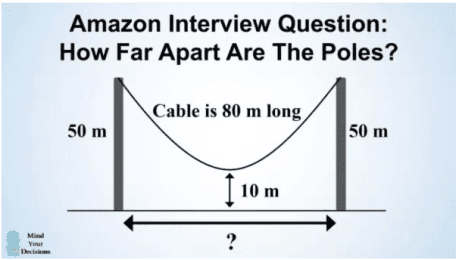
**Introduction**

The so called Amazon’s hanging cable problem explained as follows:

*A cable of 80 meters (m) is hanging from the top of two poles that are both 50 m from the ground. What is the distance between the two poles, to one decimal place, if the center of the cable is*:

1. *20 m above the ground?*
2. *10 m above the ground?*

Allegedly, (b) has been used as an Amazon interview question, however, the problem is much older and has otherwise nothing to do with Amazon. Can you solve (b)? Or even (a)? The problem can be illustrated as follows:



Hint: The solution to (a) is concisely described and for (b) you need to do little more than just think. So instead of applying at Amazon, let’s take the question to the next level: Apply for the orange belt in R: How you *wouldn’t* solve the hanging cable problem by instead solving the hanging cable problem **suspension bridge style**!

As explained the [catenary curve](https://en.wikipedia.org/wiki/Catenary) is the geometric shape, a cable assumes under its own weight when supported only at its ends. If instead the cable supports a uniformly distributed vertical load, the cable has the shape of a [parabolic curve](https://en.wikipedia.org/wiki/Parabola). This would for example be the case for a [**suspension bridge**](https://en.wikipedia.org/wiki/Suspension_bridge) with a horizontal suspended deck, if the cable itself is not too heavy compared to the road sections. A prominent example of a suspension bridges is the [Golden Gate Bridge](https://en.wikipedia.org/wiki/Golden_Gate_Bridge), which we will use as motivating example for this post.

**Solving the Cable Problem**

**Parabola Shape**

Rephrasing the cable problem as the ‘*suspension bridge problem*‘ we need to solve a two-component non-linear equation system:

1. the first component ensures that the parabolic curve with vertex at \((0,0)\) goes through the poles at the x-values \(-x\) and \(x\). In other words: the distance between the two poles is \(2x\). Note that the coordinate system is aligned such that the lowest point of the cable is at the origo.
2. the second component ensures that the arc-length of the parabola is as given by the problem. Since the parabola is symmetric it is sufficient to study the positive x-axis

The two criteria are converted into an equation system as follows: \[  
\begin{align\*} a x^2 &= 50 – \text{height above ground} \\  
\int\_0^x \sqrt{1 + \left(\frac{d}{du} a u^2\right)^2} du &= 40.  
\end{align\*}  
\]

Here, the general equation for [arc-length](https://en.wikipedia.org/wiki/Arc_length) of a function \(y=f(u)\) has been used. Solving the arc-length integral for a parabola can either be done by numerical integration or by solving the integral analytically or just look up the resulting analytic expression as eqn 4.25 in Spiegel (1968). Subtracting the RHS from each LHS gives us a non-linear equation system with unknowns \(a\) and \(x\) of the form

\[  
\left[  
\begin{array}{c}  
y\_1(a,x) \\  
y\_2(a,x)  
\end{array}  
\right]  
=  
\left[  
\begin{array}{c}  
0 \\  
0  
\end{array}  
\right].  
\]

The Arc Length of a Parabola Let us calculate the length of the parabolic arc y = x 2 , 0 ≤ x ≤ a. According to the arc length formula, L(a) = Z a 0 p 1 + y 0 (x) 2 dx = Z a 0 p 1 + (2x) 2 dx. Replacing 2x by x, we may write L(a) = 1 2 Z 2a 0 √ 1 + x 2 dx. Thus the task is to find the antiderivative of √ 1 + x 2 . This is often done by setting x = sinh t or x = tan t. We will obtain the answer by manipulating square roots. Two observations are needed: x √ x 2 + 10 = √ x 2 + 1 + x 2 √ x 2 + 1 = √ x 2 + 1 + √ x 2 + 1 − 1 √ x 2 + 1 = 2√ x 2 + 1 − 1 √ x 2 + 1 . ln x + √ 1 + x 2 0 = 1 + x/√ 1 + x 2 x + √ 1 + x 2 = 1 √ 1 + x 2 . Hence 2√ 1 + x 2 = x √ x 2 + 10 + ln x + √ 1 + x 2 0 . Hence Z √ 1 + x 2 dx = 1 2 x √ 1 + x 2 + 1 2 ln x + √ 1 + x 2 + C. It follows that L(a) = 1 2 a √ 1 + 4a 2 + 1 4 ln 2a + √ 1 + 4a 2 . In particular, L(1) = 1 2 √ 5 + 1 4 ln(2 + √ 5) ≈ 1.48.

Writing this in R code:

## Height of function at the location x from center is (pole\_height - height\_above\_ground)

y1\_parabola <- function(a, x, pole\_height=50, above\_ground=20) {

a\*x^2 - (pole\_height - above\_ground)

}

## Arc-length of the parabola between [-x,x] is given as cable\_length

y2\_parabola <- function(a,x, cable\_length=80, arc\_method=c("analytic","numeric")) {

##Arc-length of a parabola a\*u^2 within interval [0,x]

if(arc\_method == "numeric") {

f <- function(u) return( sqrt(1 + (2\*a\*u)^2))

half\_arclength <- integrate(f, lower=0, upper=x)$value

} else if (arc\_method=="analytic") {

half\_arclength <- 1/(4\*a)\*(2\*a\*x\*sqrt(4\*a^2\*x^2+1) + asinh(2\*a\*x))

}

##The equation: s = cable\_length/2

half\_arclength - cable\_length/2

}

## The non-linear equation system \bm{y}(\theta) = \bm{0}, where the LHS

## is given by a list with two components containing y\_1(\theta) and y\_2(\theta)

f\_sys <- function(theta, y, pole\_height=50, above\_ground=20, cable\_length=80, ...) {

##Parameters

a <- theta[1]

x <- exp(theta[2]) ##ensure x is positive

c(y[[1]](a,x, pole\_height=pole\_height, above\_ground=above\_ground),

y[[2]](a,x, cable\_length=cable\_length, ...))

}

##Helper function to transform theta parameter vector to (a,x)'

theta2ax <- function(theta) {

c(a=theta[1], x=exp(theta[2]))

}

To ensure \(x>0\) we re-parametrized the equations with \(\theta\_2 = \log(x)\) and provide the function theta2ax to backtransform the result. We can now use the nleqslv package to solve the non-linear equation system using a one-liner:

Library(nlegslv)

y\_parabola <- list(y1\_parabola, y2\_parabola)

sol\_parabola <- nleqslv(x=c(0.1,0.1),f\_sys, y=y\_parabola, arc\_method="analytic")

theta2ax(sol\_parabola$x)

## a x

## 0.05355207 23.66859605

In other words, for a cable of length 80m the pole of a suspension bridge will be located 23.7m from the origo, which means the two poles of the bridge will be 47.3m apart, which is also the span of the bridge.

Using arc\_method="numeric" instead of the analytic solution gives

## a x

## 0.05355207 23.66859605

It is re-assuring to see that the numerical integration method yields the same result as the analytic method. The analytic method has mathematical beauty, the numerical method allows the data scientist to solve the problem without diving into formula compendiums or geometry.

**Catenary Shape**

Using the same code, but with the y-functions formulated for the catenary case we obtain

## Value of y=f(u) evaluated at u=x

y1\_catenary <- function(a,x, pole\_height=50, above\_ground=20) {

a \* cosh(x/a) - a - (pole\_height- above\_ground)

}

## Arc-length condition

y2\_catenary <- function(a,x, cable\_length=80) {

a \* sinh(x/a) - cable\_length/2

}

## Solve equation system

y\_catenary <- list(y1\_catenary, y2\_catenary)

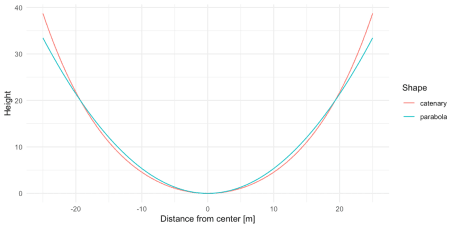
sol\_catenary <- nleqslv(x=c(0.1,0.1),f\_sys, y=y\_catenary, method="Newton")

theta2ax(sol\_catenary$x)

## a x

## 11.66667 22.70229

In other words the solution to the original cable problem is \(x=22.7 m\) whereas the answer to the suspension bridge version is \(x=23.7m\). The difference to the parabolic form can be seen from the following graph:



**Testing the theory**

We test our theory by studying the cable of the Golden Gate suspension bridge. Shown below is a photograph. For presentation in this post the image was tilted by -0.75 degrees (around the camera’s view axis) with the imager package to make sea level approximately horizontal. Parabolic and catenary overlays (no real difference between the two) were done using the theory described above.

Library(Imager)

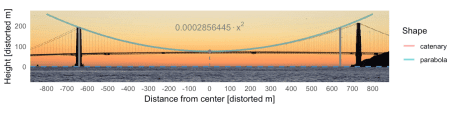
##Preprocess image

img <- imager::load.image(file.path(fullFigPath, "Golden\_Gate\_Bridge.png"))

img <- imager::imrotate(img, angle=-0.75, interpolation=1)

img <- imager::resize(img,-50,-50, interpolation\_type=5)

We manually identify center, sea level and poles from the image and use annotation\_raster to overlay the image on the ggplot of the corresponding parabola and catenary.



The fit is not perfect, which is due to the camera’s direction not being orthogonal to the plane spanned by the bridge – for example the right pole appears to be closer to the camera than the left pole[2](http://staff.math.su.se/hoehle/blog/2018/07/23/cable.html#fn2). We scaled and ‘offsetted’ the image so the left pole is at distance 640m from origo, but did not correct for the tilting around the \(y\)-axis. Furthermore, distances are being distorted by the lens, which might explain the poles being too small. [Rectification](https://en.wikipedia.org/wiki/Image_rectification) and [perspective control](https://en.wikipedia.org/wiki/Perspective_control) of such images is a [**photogrammetric**](https://en.wikipedia.org/wiki/Photogrammetry) method beyond the scope of this post!

**Discussion**

This post may not to impress a Matlab coding engineer, but it shows how R has developed into a versatile tool going way beyond statistics: We used its optimization and image analysis capabilities. Furthermore, given an analytic form of \(y(\theta)\), R can symbolically determine the Jacobian and, hence, implement the required Newton-Raphson solving of the non-linear equation system directly – see the Appendix. In other words: R is also a full stack mathematical problem solving tool!

As a **challenge** to the interested reader: Can you write R code, for example using imager, which automatically identifies poles and cable in the image and based on the known specification of these parameters of the Golden Gate Bridge (pole height: 230m, span 1280m, clearance above sea level: 67.1m), and perform a rectification of the image?

**Appendix – Newton-Raphson Algorithm**

Because the y\_1(a,x) and y\_2(a,x) are both available in closed analytic form, one can form the Jacobian of non-linear equations system by combining the two gradients. This can be achieved symbolically using the deriv or Deriv::Deriv functions in R.

Library(Deriv)

Given starting value \(\theta\) the iterative procedure to find the root of the non-linear equation system \(y(\theta) = 0\) is given by (Nocedal and Wright 2006, Sect. 11.1)

\[  
\theta^{(k+1)} = \theta^k – J(\theta^k)^{-1} y(\theta),  
\]

where \(J\) is the Jacobian of the system, which in this case is a 2×2 matrix.

gradient\_y1 <- Deriv::Deriv(y1\_parabola, x=c("a","x"))

y2\_parabola\_analytic <- function(a,x, cable\_length=80) {

1/(4\*a)\*(2\*a\*x\*sqrt(4\*a^2\*x^2+1) + asinh(2\*a\*x)) - cable\_length/2

}

gradient\_y2 <- Deriv::Deriv(y2\_parabola\_analytic, x=c("a","x"))

##Jacobian

J <- function(theta, pole\_height=50, above\_ground=20, cable\_length=80, ...) {

a <- theta[1]

x <- exp(theta[2]) # x <- exp(theta[2])

##Since we use x = exp(theta[2])=g(theta[2]) we need the chain rule to find the gradient in theta

##this is g'(theta[2]) = exp(theta[2]) = x

rbind(gradient\_y1(a,x, pole\_height=pole\_height, above\_ground=above\_ground)\* c(1, x),

gradient\_y2(a,x, cable\_length=cable\_length) \* c(1, x))

}

By iterating Newton-Raphson steps we can find the solution of the equation system manually:

##Start values

theta <- c(0.1,log(10))

thetanew <- c(0.1,log(20))

##Log with the values

log <- t(theta2ax(theta))

##Iterate Newton-Raphson steps until convergence

while ( (sum(thetanew - theta)^2 / sum(theta^2)) > 1e-15) {

theta <- thetanew

##Update step

thetanew <- theta - solve(J(theta=theta)) %\*% f\_sys(theta, y=y\_parabola, arc\_method="analytic")

##Add to log

log <- rbind(log, theta2ax(thetanew))

}

##Look at the steps taken

log

## a x

## [1,] 0.10000000 10.00000

## [2,] 0.02667392 25.46647

## [3,] 0.04632177 25.43589

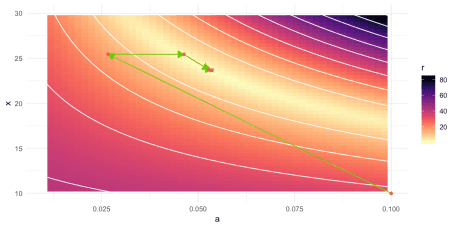
## [4,] 0.05270610 23.75416

## [5,] 0.05354318 23.66953

## [6,] 0.05355207 23.66860

## [7,] 0.05355207 23.66860

We show the moves of the algorithm in a 2D contour plot for \(r(a,x) = \sqrt{y\_1(a,x)^2 + y\_2(a,x)^2}\). The solution to the system has \(r(a,x)=0\).



The R Code to the Amazon Hanging Cable Problem

## Solving the Cable Problem

### Parabola Shape

Rephrasing the cable problem as the '\*suspension bridge problem\*' we

need to solve a two-component non-linear equation system:

1. the first component ensures that the parabolic curve with vertex at

$(0,0)$ goes through the poles at the x-values $-x$ and $x$. In other

words: the distance between the two poles is $2x$. Note that the

coordinate system is aligned such that the lowest point of the cable

is at the origo.

2. the second component ensures that the arc-length of the parabola is

as given by the problem. Since the parabola is symmetric it is

sufficient to study the positive x-axis

The two criteria are converted into an equation system as follows:

$$

\begin{align\*} a x^2 &= 50 - \text{height above ground} \\

\int\_0^x \sqrt{1 + \left(\frac{d}{du} a u^2\right)^2} du &= 40.

\end{align\*}

$$

Here, the general equation for

[arc-length](https://en.wikipedia.org/wiki/Arc\_length) of a function

$y=f(u)$ has been used. Solving the arc-length integral for a

parabola can either be done by numerical integration or by

[solving the integral analytically](http://www.math.drexel.edu/~tolya/arc\_length\_x%5e2.pdf)

or just look up the resulting analytic expression as eqn 4.25 in

@spiegel1968. Subtracting the RHS from each LHS gives us a non-linear equation

system with unknowns $a$ and $x$ of the form

$$

\left[

\begin{array}{c}

y\_1(a,x) \\

y\_2(a,x)

\end{array}

\right]

=

\left[

\begin{array}{c}

0 \\

0

\end{array}

\right].

$$

Writing this in R code:

```{r PARABOLA, echo=TRUE}

## Height of function at the location x from center is (pole\_height - height\_above\_ground)

y1\_parabola <- function(a, x, pole\_height=50, above\_ground=20) {

a\*x^2 - (pole\_height - above\_ground)

}

## Arc-length of the parabola between [-x,x] is given as cable\_length

y2\_parabola <- function(a,x, cable\_length=80, arc\_method=c("analytic","numeric")) {

##Arc-length of a parabola a\*u^2 within interval [0,x]

if(arc\_method == "numeric") {

f <- function(u) return( sqrt(1 + (2\*a\*u)^2))

half\_arclength <- integrate(f, lower=0, upper=x)$value

} else if (arc\_method=="analytic") {

half\_arclength <- 1/(4\*a)\*(2\*a\*x\*sqrt(4\*a^2\*x^2+1) + asinh(2\*a\*x))

}

##The equation: s = cable\_length/2

half\_arclength - cable\_length/2

}

## The non-linear equation system \bm{y}(\theta) = \bm{0}, where the LHS

## is given by a list with two components containing y\_1(\theta) and y\_2(\theta)

f\_sys <- function(theta, y, pole\_height=50, above\_ground=20, cable\_length=80, ...) {

##Parameters

a <- theta[1]

x <- exp(theta[2]) ##ensure x is positive

c(y[[1]](a,x, pole\_height=pole\_height, above\_ground=above\_ground),

y[[2]](a,x, cable\_length=cable\_length, ...))

}

##Helper function to transform theta parameter vector to (a,x)'

theta2ax <- function(theta) {

c(a=theta[1], x=exp(theta[2]))

}

```

To ensure $x>0$ we re-parametrized the equations with $\theta\_2 =

\log(x)$ and provide the function `theta2ax` to backtransform the result. We can now use the

[`nleqslv`](https://cran.r-project.org/web/packages/nleqslv/index.html)

package to solve the non-linear equation system using a one-liner:

```{r, echo=TRUE}

y\_parabola <- list(y1\_parabola, y2\_parabola)

sol\_parabola <- nleqslv(x=c(0.1,0.1),f\_sys, y=y\_parabola, arc\_method="analytic")

theta2ax(sol\_parabola$x)

```

In other words, for a cable of length 80m the pole of a suspension bridge will be located `r sprintf("%.1fm",theta2ax(sol\_parabola$x)[2])` from the origo, which means the two poles of the bridge will be `r sprintf("%.1fm",2\*theta2ax(sol\_parabola$x)[2])` apart, which is also the span of the bridge.

Using `arc\_method="numeric"` instead of the analytic solution gives

```{r NUMERICALPARABOLA}

sol\_parabola2 <- nleqslv(x=c(0.1,0.1),f\_sys, y=y\_parabola, arc\_method="numeric")

theta2ax(sol\_parabola2$x)

```

It is re-assuring to see that the numerical integration method yields

the same result as the analytic method. The analytic method has

mathematical beauty, the numerical method allows the data scientist to

solve the problem without diving into formula compendiums or geometry.

### Catenary Shape

Using the same code, but with the y-functions formulated for the

catenary case we obtain

```{r, echo=TRUE}

## Value of y=f(u) evaluated at u=x

y1\_catenary <- function(a,x, pole\_height=50, above\_ground=20) {

a \* cosh(x/a) - a - (pole\_height- above\_ground)

}

## Arc-length condition

y2\_catenary <- function(a,x, cable\_length=80) {

a \* sinh(x/a) - cable\_length/2

}

## Solve equation system

y\_catenary <- list(y1\_catenary, y2\_catenary)

sol\_catenary <- nleqslv(x=c(0.1,0.1),f\_sys, y=y\_catenary, method="Newton")

theta2ax(sol\_catenary$x)

```

In other words the solution to the original cable problem is

$x=`r sprintf("%.1f m",theta2ax(sol\_catenary$x)[2])`$ whereas the answer to

the suspension bridge version is $x=`r sprintf("%.1fm",theta2ax(sol\_parabola$x)[2])`$.

The difference to the parabolic form can be seen from the following

graph:

```{r PARABOLACATENARYPLOT}

data.frame(x=seq(-25,25,length=1000)) %>%

mutate(catenary=y1\_catenary(x, a=sol\_catenary$x[1], above\_ground=50),

parabola=y1\_parabola(x, a=sol\_parabola$x[1], above\_ground=50)) %>%

tidyr::gather("Shape","y", -x) %>%

ggplot(aes(x=x,y=y, color=Shape)) + geom\_line() + xlab("Distance from center [m]") + ylab("Height")

```

## Testing the theory

```{r}

##true height is 230m, see https://en.wikipedia.org/wiki/Golden\_Gate\_Bridge, but picture is distorted

gg\_pole\_height <- gg\_pole\_height <- 192

##true height above sealvl is 67.1m, but that's the bridge not the cable

gg\_above\_ground <- 75

##half the span of the bridge, total is 1280m,

gg\_pole\_position <- 640

```

We test our theory by studying the cable of the Golden Gate suspension

bridge. Shown below is a photograph by

[D Ramey Logan](https://commons.wikimedia.org/wiki/File:Golden\_Gate\_Bridge\_Dec\_15\_2015\_by\_D\_Ramey\_Logan.jpg)

available under a

[CC BY 4.0](https://creativecommons.org/licenses/by/4.0/deed.en)

license. For presentation in this post the image was tilted by -0.75

degrees (around the camera's view axis) with the

[`imager`](https://dahtah.github.io/imager/imager.html) package to

make the sea level approximately horizontal. Parabolic and catenary

overlays (no real difference between the two) were done using the

theory described above.

```{r PREPROCESS, echo=TRUE}

##Preprocess image

img <- imager::load.image(file.path(fullFigPath, "Golden\_Gate\_Bridge.png"))

img <- imager::imrotate(img, angle=-0.75, interpolation=1)

img <- imager::resize(img,-50,-50, interpolation\_type=5)

```

```{r, eval=FALSE}

##Fix negative channel values, if one uses interpolation=2 in imrotate

img[img < 0] <- 0

img[img > 1] <- 1

```

We manually identify center, sea level and poles from the image and

use [`annotation\_raster`](https://ggplot2.tidyverse.org/reference/annotation\_raster.html)

to overlay the image on the `ggplot` of the

corresponding parabola and catenary. See the [code](`r paste0("https://raw.githubusercontent.com/hoehleatsu/hoehleatsu.github.io/master/\_source/",current\_input())`)

on github for details.

```{r GOLDENGATE, fig.width=8, fig.height=2}

##Convert to ggplot compatible format (alternative: as.data.frame)

ggate <- aperm(img[,,1,], perm=c(2,1,3))

a\_parabola <- uniroot(function(a) y1\_parabola(a, x=gg\_pole\_position, pole\_height=gg\_pole\_height, above\_ground=gg\_above\_ground), interval=c(0, 30))$root

a\_catenary <- uniroot(function(a) y1\_catenary(a, x=gg\_pole\_position, pole\_height=gg\_pole\_height, above\_ground=gg\_above\_ground), interval=c(1e-12, 5000))$root

p <- data.frame(x=seq(-800,800,length=1000)) %>%

mutate(catenary=y1\_catenary(x, a=a\_catenary, pole\_height=gg\_pole\_height, above\_ground=gg\_above\_ground) + gg\_pole\_height,

parabola=y1\_parabola(x, a=a\_parabola, pole\_height=gg\_pole\_height, above\_ground=gg\_above\_ground) + gg\_pole\_height) %>%

tidyr::gather("Shape","y", -x) %>%

ggplot(aes(x=x,y=y, color=Shape)) +

# geom\_line() +

ylim(c(-60, NA)) +

scale\_x\_continuous(name="Distance from center [distorted m]", breaks=seq(-800,800,by=100)) +

ylab("Height [distorted m]")

##This is a rather messy piece of code scaling the image to fit acceptably

##to known parameters. But since the photo is not really orthogonal to the bridge

##this only works very very approximately...

xy\_ratio <- dim(ggate)[1] / dim(ggate)[2]

x\_scale <- 1.7 ##ymin=-110

p + annotation\_raster(ggate, ymin = -124,ymax= 2\*gg\_pole\_position\*xy\_ratio,xmin = -x\_scale\*gg\_pole\_position, xmax = x\_scale\*gg\_pole\_position, interpolate=FALSE) +

coord\_equal() +

geom\_line(lwd=1, alpha=0.6) +

geom\_text(data=data.frame(x=0, y=200,text=paste0(a\_parabola, " %.% x^2"),Shape=NA), aes(x=x, y=y, label=text),parse=TRUE, show.legend=FALSE) +

geom\_hline(yintercept=0,color="steelblue",lty=2, size=0.8) +

geom\_line(data=data.frame(x=rep(-gg\_pole\_position, 2), y=c(0,gg\_pole\_height), Shape=NA),show.legend=FALSE,lwd=1.2) +

geom\_line(data=data.frame(x=rep(gg\_pole\_position, 2), y=c(0,gg\_pole\_height), Shape=NA),show.legend=FALSE,lwd=1.2) +

geom\_point(data=data.frame(x=0, y=gg\_above\_ground, Shape=NA), show.legend=FALSE, size=1.5) +

geom\_line(data=data.frame(x=rep(0,2), y=c(0,gg\_above\_ground), Shape=NA), show.legend=FALSE, lty=2)

```

The fit is not perfect, which is due to the camera's direction not

being orthogonal to the plane spanned by the bridge -

for example the right pole appears to be closer to the camera than the

left pole^[A manual investigation using the "Map | Map Object" Filter in Gimp showed that the angle of tilting around the y-axis is about 20 degrees.]. We

scaled and 'offsetted' the image so the left pole is at distance `r gg\_pole\_position`m from origo, but did not correct for the tilting around the $y$-axis. Furthermore, distances are being distorted

by the lens, which might explain the poles being too

small. [Rectification](https://en.wikipedia.org/wiki/Image\_rectification)

and

[perspective control](https://en.wikipedia.org/wiki/Perspective\_control)

of such images is a

\*\*[photogrammetric](https://en.wikipedia.org/wiki/Photogrammetry)\*\*

method beyond the scope of this post!

## Discussion

This post may not to impress a Matlab coding engineer, but it shows

how R has developed into a versatile tool going way beyond statistics:

We used its optimization and image analysis capabilities. Furthermore,

given an analytic form of $y(\theta)$, R can symbolically determine

the Jacobian and, hence, implement the required Newton-Raphson solving

of the non-linear equation system directly - see the Appendix. In

other words: R is also a full stack mathematical problem solving tool!

As a \*\*challenge\*\* to the interested reader: Can you write R code,

for example using `imager`, which automatically identifies poles and cable in

the image and based on the known specification of these parameters of

the Golden Gate Bridge (pole height: 230m, span 1280m, clearance above

sea level: 67.1m), and perform a rectification of the image? If yes,

Stockholm University's Math Department

[hires](https://www.math.su.se/english/education/phd-studies/admission-and-vacant-positions)

for Ph.D. positions every April! The challenge could work well as pre-interview project. `r emo::ji("wink")`

## Appendix - Newton-Raphson Algorithm

Because the `y\_1(a,x)` and `y\_2(a,x)` are both available in closed

analytic form, one can form the Jacobian of non-linear equations

system by combining the two gradients. This can be achieved

symbolically using the `deriv` or

[`Deriv::Deriv`](https://cran.r-project.org/web/packages/Deriv/index.html)

functions in R.

Given starting value $\theta$ the iterative procedure to find the root

of the non-linear equation system $y(\theta) = 0$ is given by

[@nocedal\_wright2006, Sect. 11.1]

$$

\theta^{(k+1)} = \theta^k - J(\theta^k)^{-1} y(\theta),

$$

where $J$ is the

[Jacobian](https://en.wikipedia.org/wiki/Jacobian\_matrix\_and\_determinant)

of the system, which in this case is a 2x2 matrix.

```{r NEWTONRAPHSON, echo=TRUE}

gradient\_y1 <- Deriv::Deriv(y1\_parabola, x=c("a","x"))

y2\_parabola\_analytic <- function(a,x, cable\_length=80) {

1/(4\*a)\*(2\*a\*x\*sqrt(4\*a^2\*x^2+1) + asinh(2\*a\*x)) - cable\_length/2

}

gradient\_y2 <- Deriv::Deriv(y2\_parabola\_analytic, x=c("a","x"))

##Jacobian

J <- function(theta, pole\_height=50, above\_ground=20, cable\_length=80, ...) {

a <- theta[1]

x <- exp(theta[2]) # x <- exp(theta[2])

##Since we use x = exp(theta[2])=g(theta[2]) we need the chain rule to find the gradient in theta

##this is g'(theta[2]) = exp(theta[2]) = x

rbind(gradient\_y1(a,x, pole\_height=pole\_height, above\_ground=above\_ground)\* c(1, x),

gradient\_y2(a,x, cable\_length=cable\_length) \* c(1, x))

}

```

```{r JMANUAL, echo=FALSE, results='hide'}

J\_manual <- function(theta) {

a <- theta[1]

x <- exp(theta[2])

##Derivatives dy\_1/da and dy\_1/dx found with maxima

y1da <- x^2

y2da <- (2\*x\*sqrt(4\*a^2\*x^2+1)+(8\*a^2\*x^3)/sqrt(4\*a^2\*x^2+1)+(2\*x)/sqrt(4\*a^2\*x^2+1))/(4\*a)-(asinh(2\*a\*x)+2\*a\*x\*sqrt(4\*a^2\*x^2+1))/(4\*a^2)

##Derivates dy\_2/da and dy\_2/dx found with maxima + chain rule

y1dx <- 2\*a\*x \* x

y2dx <- (2\*a\*sqrt(4\*a^2\*x^2+1)+(8\*a^3\*x^2)/sqrt(4\*a^2\*x^2+1)+(2\*a)/sqrt(4\*a^2\*x^2+1))/(4\*a) \* x

##Return result

matrix(c(y1da,y1dx, y2da, y2dx), 2,2, byrow=TRUE)

}

##Start value

theta0 <- c(0.1,log(10))

##Compare Jacobian at theta0

J(theta=theta0)

J\_manual(theta=theta0)

```

By iterating Newton-Raphson steps we can find the solution of the

equation system manually:

```{r NRITERATE, echo=TRUE}

##Start values

theta <- c(0.1,log(10))

thetanew <- c(0.1,log(20))

##Log with the values

log <- t(theta2ax(theta))

##Iterate Newton-Raphson steps until convergence

while ( (sum(thetanew - theta)^2 / sum(theta^2)) > 1e-15) {

theta <- thetanew

##Update step

thetanew <- theta - solve(J(theta=theta)) %\*% f\_sys(theta, y=y\_parabola, arc\_method="analytic")

##Add to log

log <- rbind(log, theta2ax(thetanew))

}

##Look at the steps taken

log

```

We show the moves of the algorithm in a 2D contour plot for $r(a,x) =

\sqrt{y\_1(a,x)^2 + y\_2(a,x)^2}$. The solution to the system has

$r(a,x)=0$. See the [code](`r

paste0("https://raw.githubusercontent.com/hoehleatsu/hoehleatsu.github.io/master/\_source/",current\_input())`)

on github for details.

```{r NRSTEPSPLOT, warning=FALSE}

grid <- expand.grid(a=seq(0.01,0.1, length=50), x=seq(10,30, length=50)) %>% as.data.frame

##apply(grid, 2, fun) would be an alternative

grid %<>% rowwise %>%

mutate(r = sqrt(sum( f\_sys( c(a,log(x)), y=y\_parabola, arc\_method="analytic")^2))) %>%

ungroup

ggplot(grid, aes(x=a, y=x, z=r)) + geom\_raster(aes(fill = r)) +

viridis::scale\_fill\_viridis(option="magma",direction=-1) +

geom\_contour(colour = "white") +

geom\_segment(data=as.data.frame(log) %>% mutate(r=NA), aes(x=a, xend=lag(a), y=x, yend=lag(x)),arrow = arrow(length=unit(0.30,"cm"), ends="first", type = "closed"),color="chartreuse3") +

geom\_point(data=as.data.frame(log) %>% mutate(r=NA), color="salmon3") +

geom\_point(data=data.frame(a=theta2ax(sol\_parabola$x)[1], x=theta2ax(sol\_parabola$x)[2],r=NA), aes(x=a, y=x), pch=4, color="salmon3",size=2) +

xlim(c(0.01,0.1)) +

ylim(c(10,30))

```

## Literature