Proposta da palestra

- Visão geral da apresentação: foco em R, pandemia como motivação;
- Perfil do apresentador;
- Perfil esperado do público.

O que é o R ?

- Biblioteca completa de estatística – com acesso à Internet: Twitter;
- Software livre casa bem com academia;
- Ambiente expansível com bibliotecas de alto desempenho em C/C++;
- Linguagem interpretada de alto nível, especializada em estatística;
- Mundo à parte da estatística.

Características de uma pandemia

- Processo exponencial;
- 1 pessoa contagia 10;
- 10 pessoas contagiam outras 10 cada uma;
- No passo 0, $10^0 = 1$ pessoa;
- No passo 1, $10^1 = 10$ pessoas;
- No passo 2, $10^2 = 100$ pessoas etc.

Onde obtemos os dados

- Municípios coletam dados, encaminham para secretaria da saúde estadual;
- Secretarias de saúde enviam para Ministério da Saúde;
- Ministério da Saúde encaminha tudo para OMS;
- OMS divulga pela Internet.
- Em nosso caso, pegamos dados da OWID – *Our World in Data*;
- OWID disponibiliza dados no `github`, portal de compartilhamento de dados e código;
- Dados sempre atualizados.

Sobre o R Studio

- IDE – *Integrated Development environment*;
- Ambiente de trabalho preferido para *R* – muito amigável;
- Evolução do Eclipse e do Microsoft Visual Studio – programas poderosos, mas caóticos na organização de comandos.

Obtendo dados através do R

scripts/get–full_data.R

```
url <- "https://github.com/owid/covid-19-data/raw/master/public/data/owid-covid-data.csv"
full_data <- read.csv(url, header=TRUE)
```

- uso de *snippets* para reduzir erros de digitação;
- endereçamento dos dados através de URL;
- normalmente (mesmo em Python) os dados seriam baixados em arquivo, depois processados.

Apresentação dos dados

scripts/show–full_data.R

```
View(full_data)
ncol(full_data)
nrow(full_data)
str(full_data)
```

- A função `View(full_data)` apresenta os dados em uma planilha;
- A função `ncol(full_data)` informa quantas colunas há nesses dados;
- A função `nrow(full_data)` informa quantas linhas há nesses dados;
- A função `str(full_data)` informa a estrutura desses dados, além de permitir uma "xereteda" neles;

Filtragem dos dados para Brasil

scripts/gen–brazil.R

```
brazil <- subset(full_data, iso_code == "BRA")
```

- Para filtrar os dados apenas do Brasil, usa-se a função `subset()`;
- A coluna `iso_code` dessas linhas deve ser `"BRA"`, código de Brasi;
- Observe que o operador de igualdade usa dois sinais de igual.

Apresentação dos dados do Brasil

scripts/show–brazil.R

```
View(brazil)
ncol(brazil)
nrow(brazil)
str(brazil)
```

- O *script* `show-brazil.R` é praticamente igual a `show-full_datal.R`; a não ser sobre a variável a que se aplica.
- Importante que as variáveis de fator são iguais !
- Isto difere do uso normal em TI: foram extirpadas as linhas dos outros países – é o mundo da estatística.
- A utilidade é permitir que se possam gerar relatórios de subtotais como antes;
- A maior diferença, é claro, é que não haverá valores para os outros países ou continentes.

Filtragem dos dados da pandemia

scripts/gen–work.R

```
work <- subset(brazil, total_cases > 0)
```

- Mas os dados do pacote de dados `brazil` ainda não estão como necessário;
- O registro da pandemia no mundo começou no último dia de 2019;
- Mas no Brasil, ele começou apenas em fevereiro, início do primeiro caso.
- Filtra-se então o pacote `brazil` para se ter o pacote de trabalho `work`;
- Usa-se a variável `total_cases` porque `new_cases` foi nula em alguns dias do início da pandemia.

Apresentação dos dados da pandemia (`show_work.R`)

scripts/show–work.R

```
View(work)
ncol(work)
nrow(work)
str(work)
```

- O *script* `show-work.R` é totalmente aos análogos `show-full_data.R` e `show-brazil.R` anteriores;
- A não ser, é claro, que se aplica ao pacote `work` recém criado;
- A verificação é importante para garantir que temos finalmente os dados adequados para trabalho.

Escolha de dados: `new_cases` e `total_cases`

- Usaremos os casos de infecção, não de falecimentos, ou *deaths*;
- As duas variáveis são relacionadas, mas os casos são em maior quantidade e, portanto, menos sujeitos a erros de medida.
- Contudo, resta decidir se devemos usar a ocorrência diária de novos casos (`new_cases`) ou seu valor acumulado `total_cases`.
- Uma forma simples de avaliar variáveis é através de sua apresentação em gráfico – sua "plotagem", ou *plot*.

Criação de variável de apoio day

scripts/creat–day.R

```
day <- 1:length(work$total_cases)
```

- Infelizmente, como vimos nas apresentações dos dados, as variáveis de fator são preservadas nos subpacotes de dados;
- Assim, a variável `date` será sempre iniciada a partir de 2019-12-31, não do início da pandemia.
- Por isso criamos a variável `day` que terá valor 1 no primeiro dia da pandemia, e terá tantos elementos quanto os dias em que a pandemia foi registrada.
- No *script* `length(work$total_cases)` define o comprimento total (ou número de linhas) da variável `total_cases` do pacote de dados `work` que estamos trabalhando;
- o cifrão `$` informa a subordinação entre `work` e `total_cases`; ou seja: `total_cases` *pertence* ao pacote `work`.

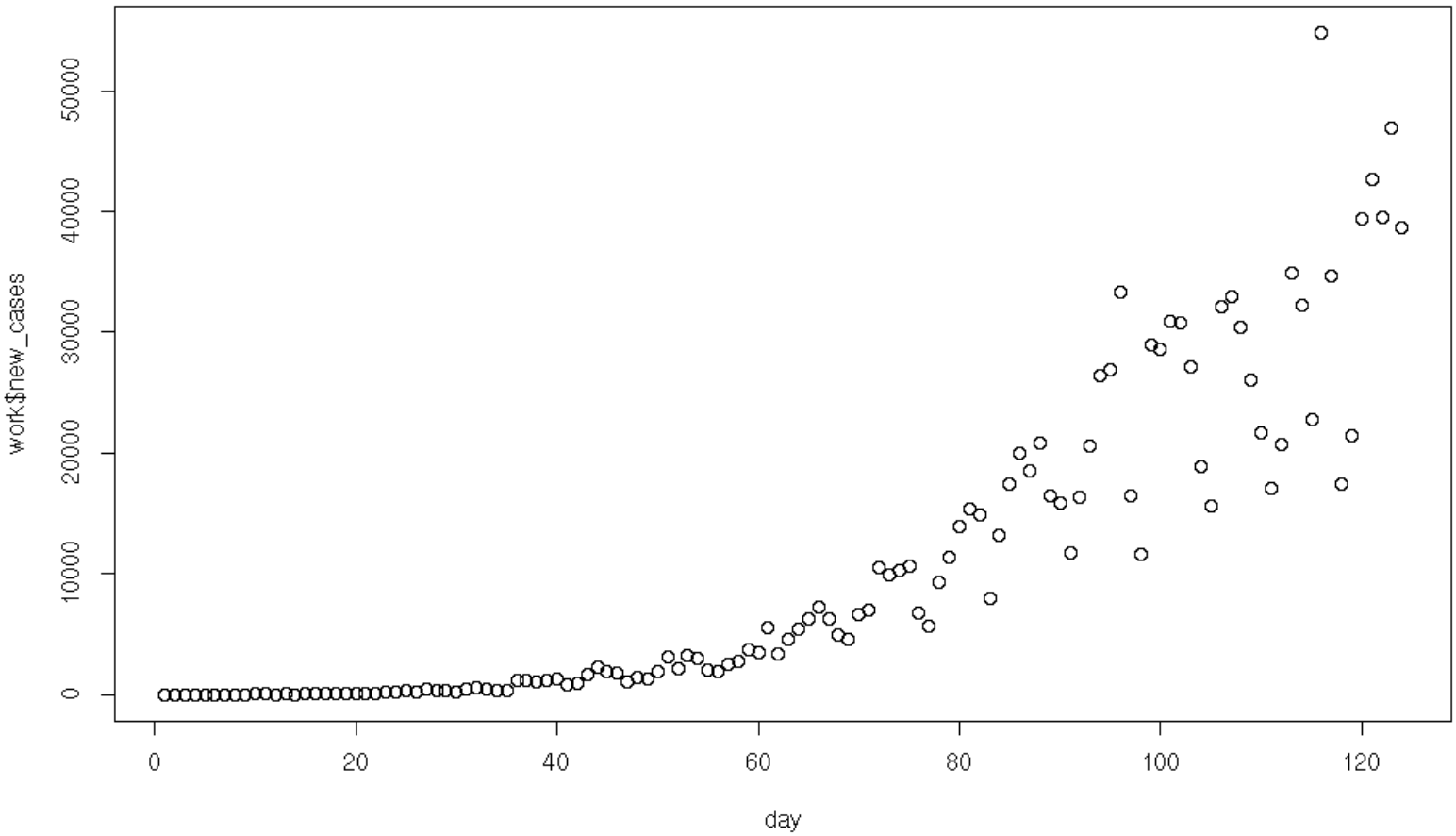
Plotagem de `day` versus `new_cases`

scripts/plot–new_cases.R

```
plot(day, work$new_cases)
```

- O programa é fácil de ser entendido ele desenha (*to plot*) duas variáveis: uma na abscissa (ou x) e outra na ordenada (ou y) – respectivamente `day` e `work$new_cases`.
- Mas o desenho resultante é difícil de ser entendido: a partir dos primeiros 70 dias não há mais um padrão claro.

Plotagem de day versus new_cases (*cont*)



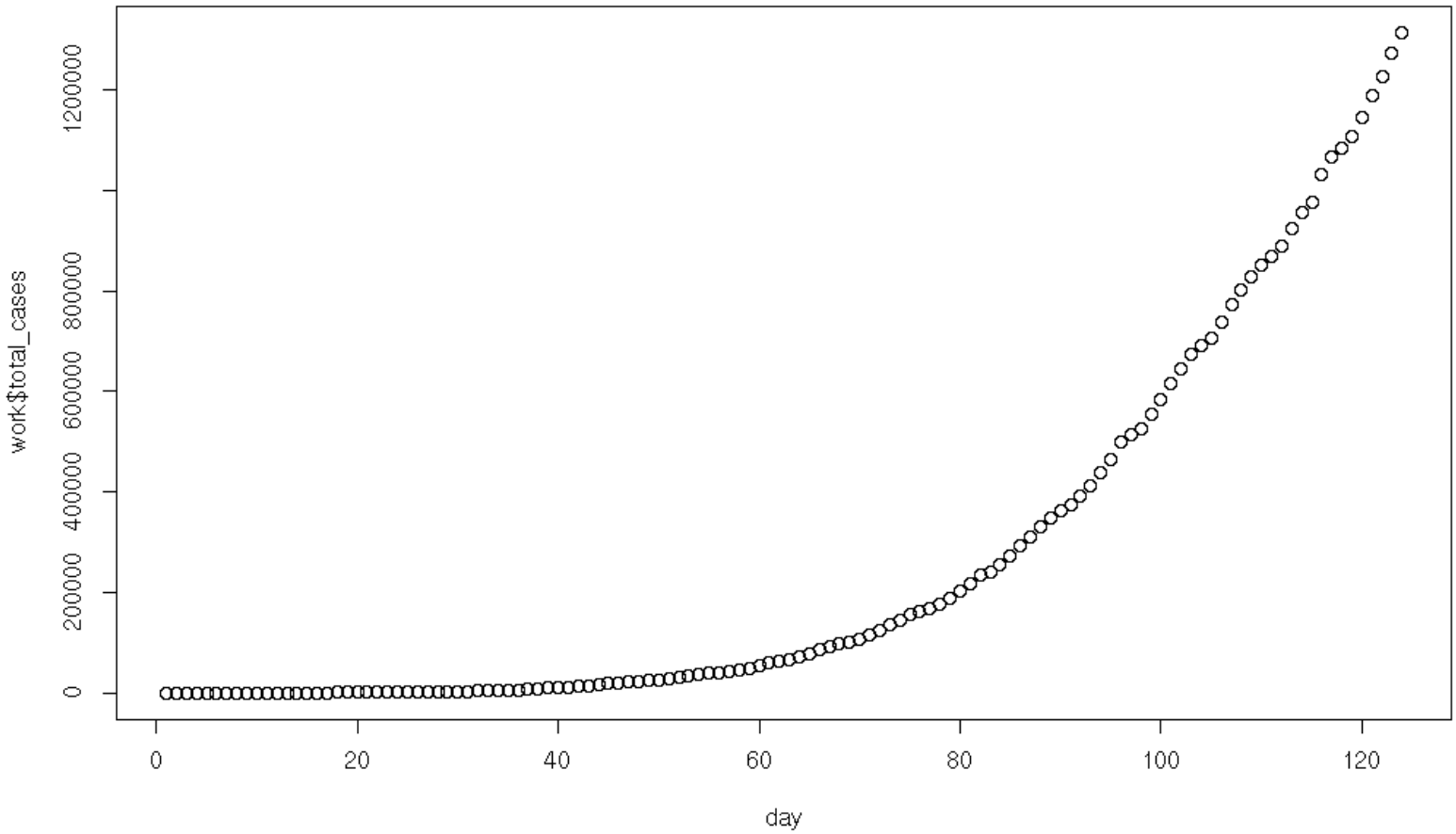
Plotagem de day versus total_cases

scripts/plot–total_cases.R

```
plot(day, work$total_cases)
```

- A compreensão do programa é a mesma do caso anterior; naturalmente substituindo `new_cases` por `total_cases`.
- Mas é muito mais fácil ver um padrão na variável `total_cases` do que na variável `new_cases`.
- Por isso é com ela que seguiremos os estudos.

Plotagem de day versus `total_cases` (*cont*)



Regressão linear de day versus total_cases

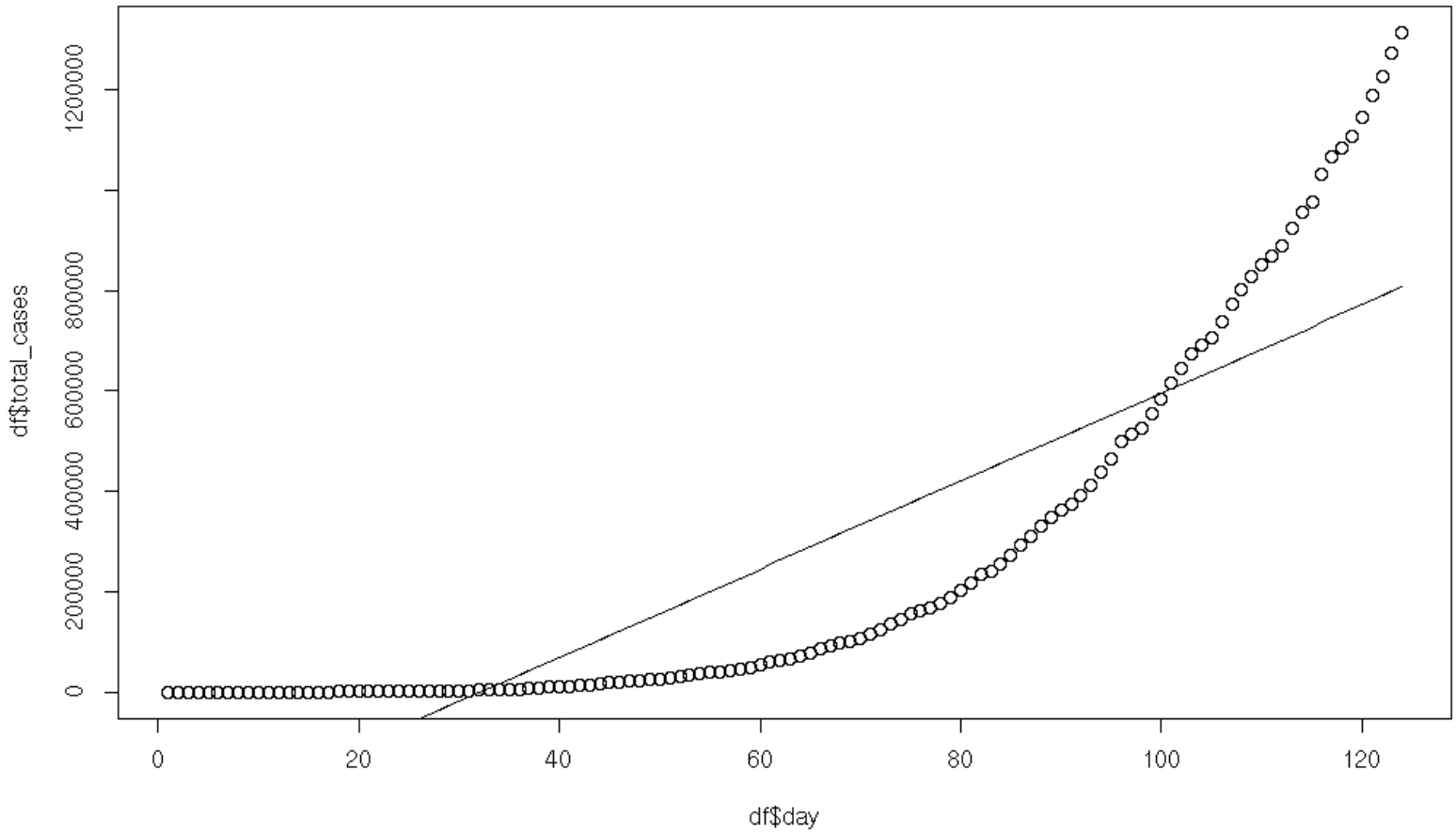
scripts/plot–lin.R

```
df_lin <- data.frame(day, total_cases = work$total_cases)
fm_lin <- lm(total_cases ~ day, data = df_lin)

dev.off()
plot(df_lin$day, df_lin$total_cases)
lines(df_lin$day, fm_lin$fitted.values)
```

- A regressão linear permite escolher a melhor reta da variável **independente** day para expressar a variável **dependente** total_cases;
- Isto é feito em *R* agregando-se ambas em uma *data frame*;
- Ajusta-se então um modelo linear, entre elas;
- A seguir, apaga-se a tela;
- Plota-se day e total_cases;
- E a seguir a reta aproximada.

Regressão linear de day versus total_cases (*cont.*)



O que é uma função exponencial

- A função exponencial clássica é $y = e^x$, onde e é a chamada *constante de Euler*.
- Ela é muito importante para a matemática, mas neste caso estamos interessados em sua propriedade de crescer muito rapidamente.
- Essa equação simples será reescrita como $T = e^{\alpha d + \beta}$.
- Seguindo o hábito dos matemáticos, de usar uma única letra para variáveis, T significa `total_cases` e d signfica `day`.
- α e β são constantes que dão mais flexibilidade ao ajuste da função.

Uso de logaritmo para facilitar a regressão linear

- Infelizmente, a técnica de regressão linear trabalha apenas com retas, planos e similares.
- Para fazer surgir uma reta neste caso, aplica-se a função logaritmo nos dois lados da igualdade:

$$log(T) = log(e^{\alpha d+\beta}) = \alpha d + \beta.$$

- a transformação de $log(e^{\alpha d+\beta})$ em $\alpha d + \beta$ é possível porque $log(x)$ e e^x são funções inversas.
- Sua *composição* (aplicação sucessiva) é a *função identidade*.
- Ou seja: $log(e^x) = x$. Além disso, $e^{log(x)} = x$

Regressão de day versus *log*(*total_cases*)

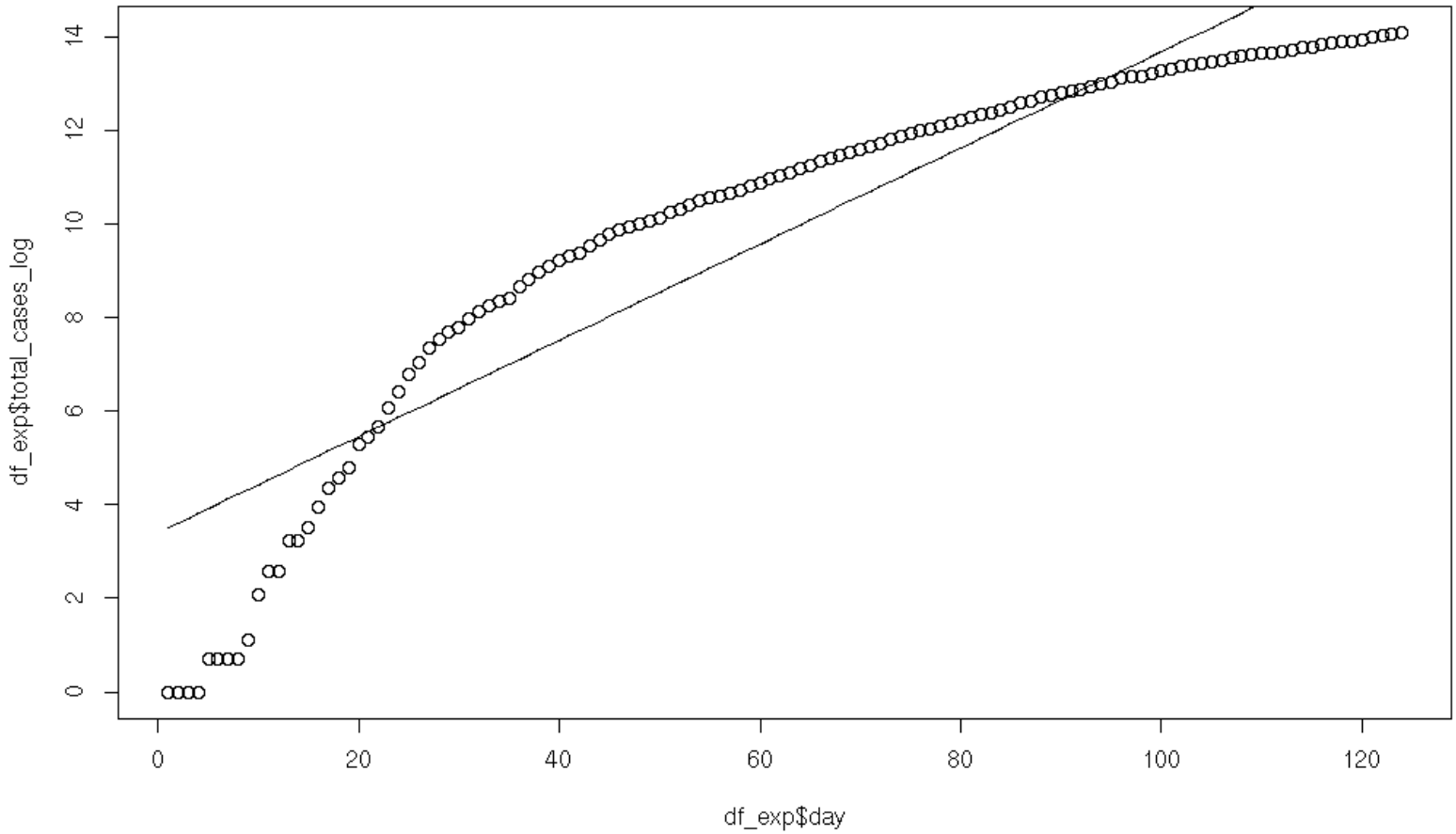
scripts/plot–exp.R

```
total_cases_log <- log(work$total_cases)
df_exp <- data.frame(day, total_cases_log)
fm_exp <- lm(total_cases_log ~ day, data = df_exp)

dev.off()
plot(df_exp$day, df_exp$total_cases_log)
lines(df_exp$day, fm_exp$fitted.values)
```

- Primeiro criamos a variável `total_cases_log` transformando os valores de `total_cases` através da função *log*();
- Depois criamos o *data frame* `df_exp` para a regressão.
- A seguir, plotamos as variáveis usadas e a reta aproximada.

Regressão de day versus *log*(*total_cases*) (*cont.*)



Plotagem sem transformação logarítmica

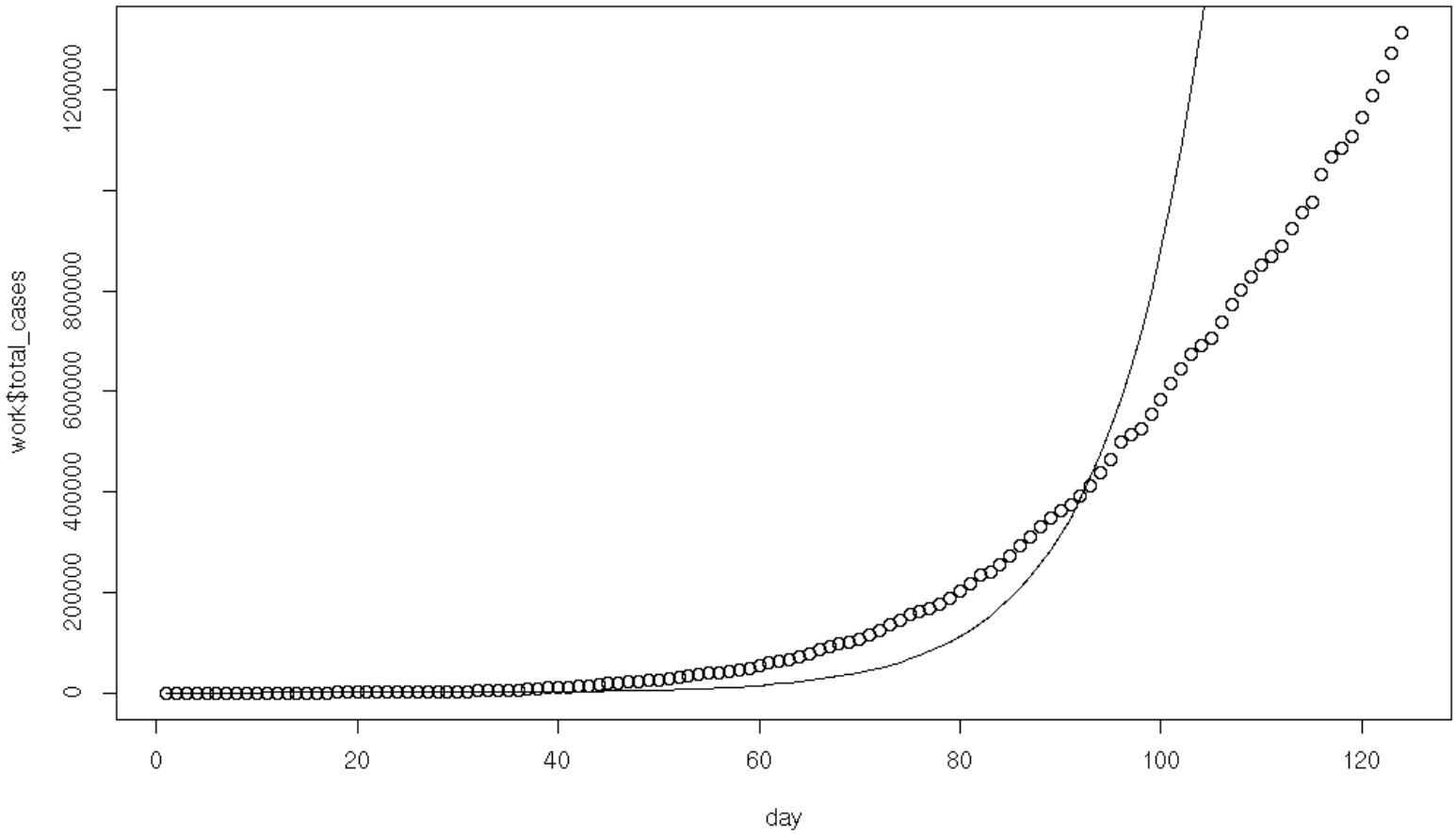
scripts/plot–unexp.R

```
fitted_values_exp <- exp(fm_exp$fitted.values)

dev.off()
plot(day, work$total_cases)
lines(day, fitted_values_exp)
```

- Para avaliar a aproximação nos dados reais, converteremos os dados da regressão em dados reais, desfazendo a transformação logarítica.
- Como a função inversa de $log(x)$ é e^x, que se escreve `exp(x)` em *R*, isto é feito na primeira linha do *snippet*.

Plotagem sem transformação logarítmica (*cont.*)



Melhor aproximação $T = e^{\gamma d^2 + \delta d + \epsilon}$

- Para melhorar o ajuste de uma função de `day` à variável dependente `total_cases`, incluiremos um termo quadrático.
- Ou seja: tentaremos expressar `total_cases` através de uma parábola de `day`, em vez de simples reta.
- Uma parábola tem a vantagem de que seu crescimento é variável e podemos ajustá-la a trechos dos dados.

Adição de day^2 à função exponencial

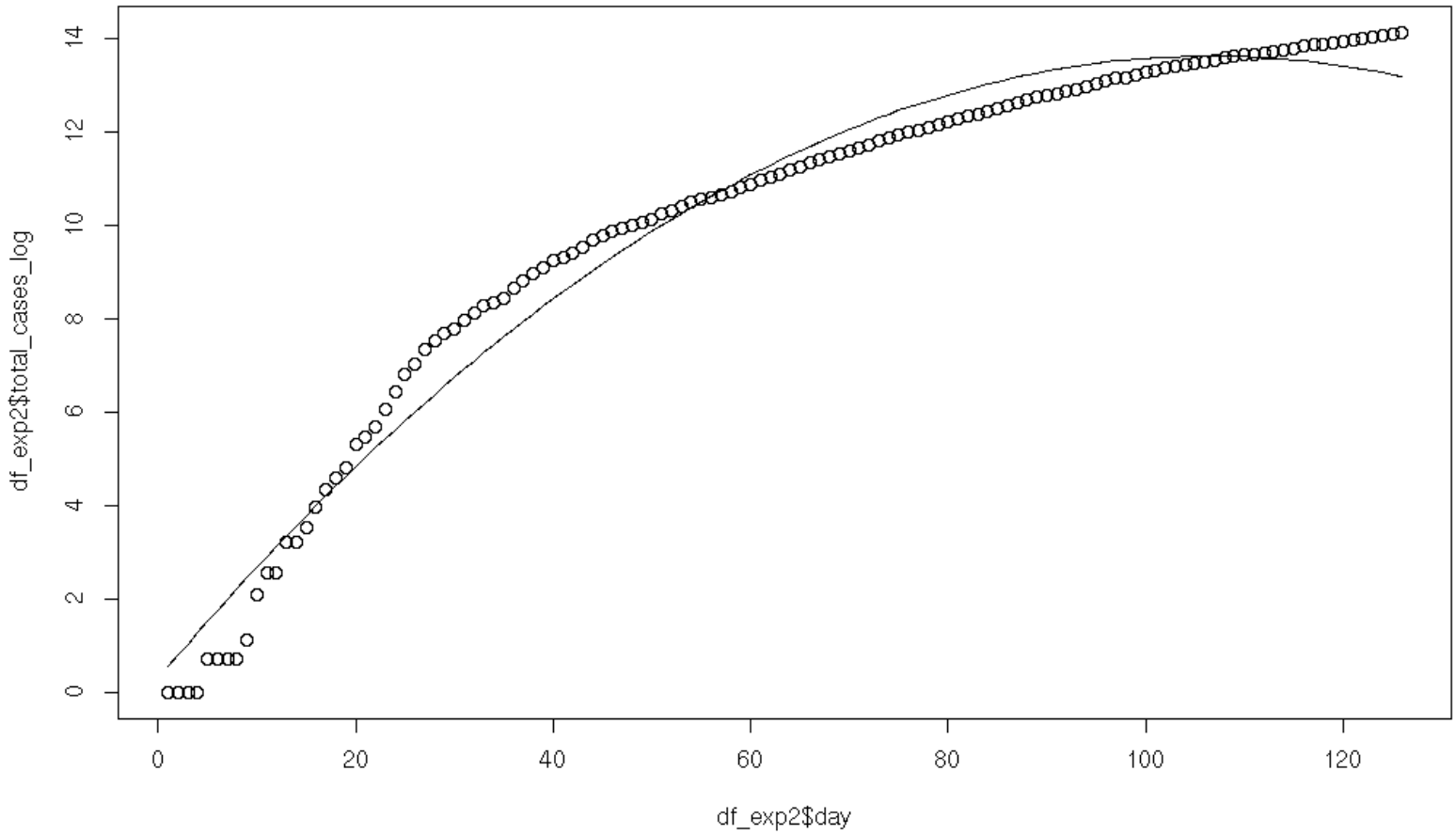
scripts/plot–exp2.R

```
day_sq <- day * day
df_exp2 <- data.frame(day, day_sq, total_cases_log)
fm_exp2 <- lm(total_cases_log ~ day + day_sq, data = df_exp2)

dev.off()
plot(df_exp2$day, df_exp2$total_cases_log)
lines(df_exp2$day, fm_exp2$fitted.values)
```

- Primeiro se cria a variável `day_sq` como o quadrado da variável `day`, ou o produto de `day` por ela mesma;
- A seguir, cria-se a *data frame* `df_exp2` para conter todas variáveis da nova regressão;
- O novo modelo de regressão linear agora inclui `day_sq`.
- A seguir, plota-se `day`, `total_cases` e a função aproximada, como feito antes.

Adição de day^2 à função exponencial (*cont.*)



Plotagem da exponencial quadrática sem transformação logarítmica

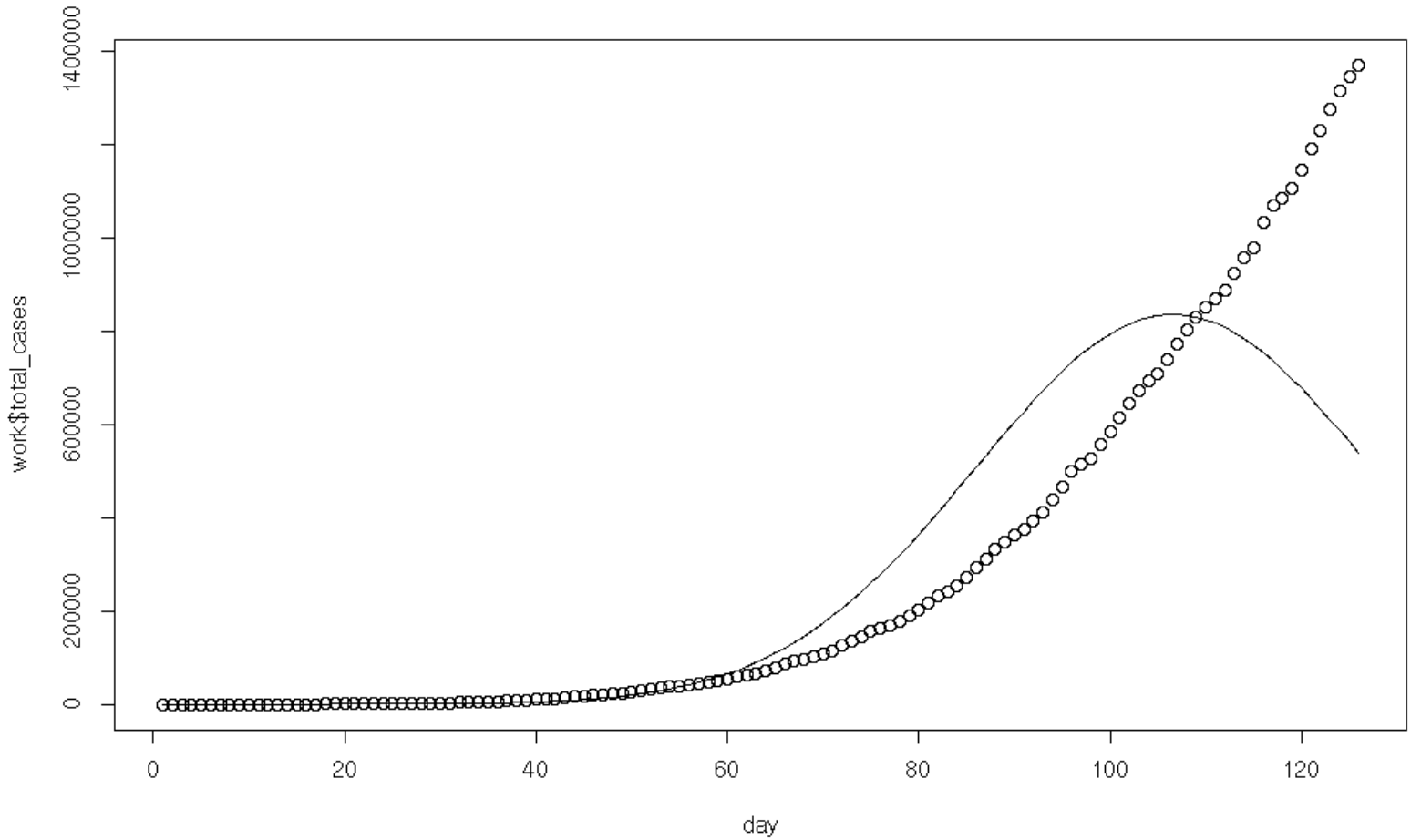
scripts/plot–unexp2.R

```
fitted_values_exp2 <- exp(fm_exp2$fitted.values)

dev.off()
plot(day, work$total_cases)
lines(day, fitted_values_exp2)
```

- Neste caso, como feito antes, usa-se a função exponencial para converter os dados aproximados em termos de `total_case`, não mais de `log(total_case)`.

Plotagem da exponencial quadrática sem transformação logarítmica (*cont.*)



Forma da parábola

scripts/summary–exp2.R

```
summary(fm_exp2)
```

- Para avaliar a aproximação em termos estatísticos, é preciso considerar os parâmetros da regressão.
- É isto que faz o comando `summary()`.

Forma da parábola (*cont.*)

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.925e-01  1.671e-01    1.75   0.0826 .
day          2.503e-01  6.074e-03   41.21   <2e-16 ***
day_sq      -1.174e-03  4.633e-05  -25.34   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6153 on 123 degrees of freedom
Multiple R-squared:  0.9767,    Adjusted R-squared:  0.9763
F-statistic:  2576 on 2 and 123 DF,  p-value: < 2.2e-16
```

- Os coeficientes confirmam o gráfico: esta é uma parábola comando a concavidade para baixo: seu coeficiente para o termo quadrátic é negativo: `-1.174e-03`, notação científica para o número 0.001174.
- É isto que faz o comando `summary()`.

Estimativa do pico da epidemia

- Isto permite calcular o pico da epidemia, que terá a forma

$$\frac{-b}{2a} = \frac{-0,2503}{2 \times (-0.001174)} \approx 107.$$

Criação de tabela com date e day

scripts/show–day_date.R

```
day_date <- data.frame(day, work$date)
View(day_date)
```

- Primeiro se cria a *data frame* `day_date`, que associa `day` com `work$date`;
- Depois, o comando `View(day_date)` mostra a *data frame* como uma tabela.
- Navegando nela, pode-se verificar que `day` de número 107 corresponde à `date` com valor `2020-06-11`;
- o mesmo resultado poderia ser obtido com a notação usual de vetores `work$date[107]`;
- Assim, de acordo com o modelo, a data do pico teria sido em 11 de junho de 2020.
- Portanto, de acordo com esse modelo, o pior já passou.