

Working in the R Ecosystem

Building Applications & Content for Your Gateway

Derrick Kearney
dsk@rstudio.com

Slides and Examples
<https://github.com/dskard/sgci-201805>

Background



Background



\neq



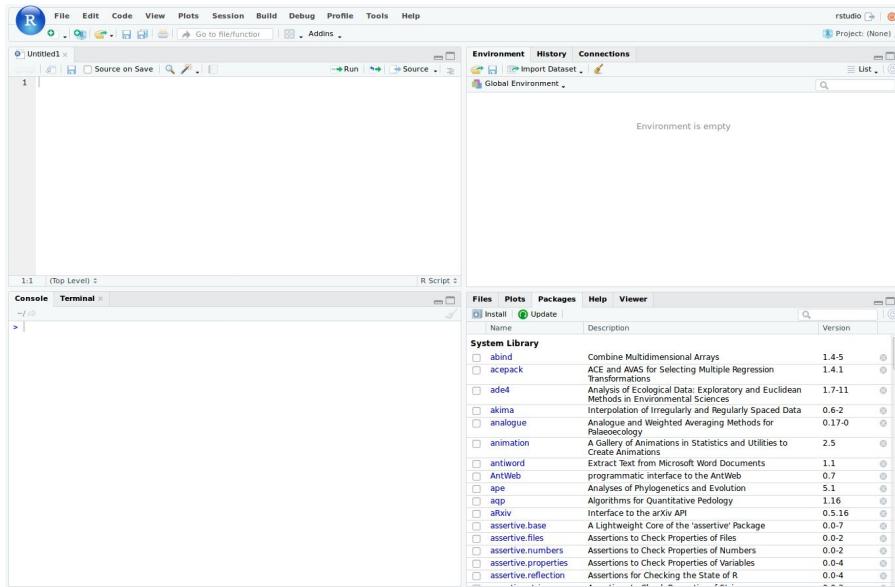
Background



Background



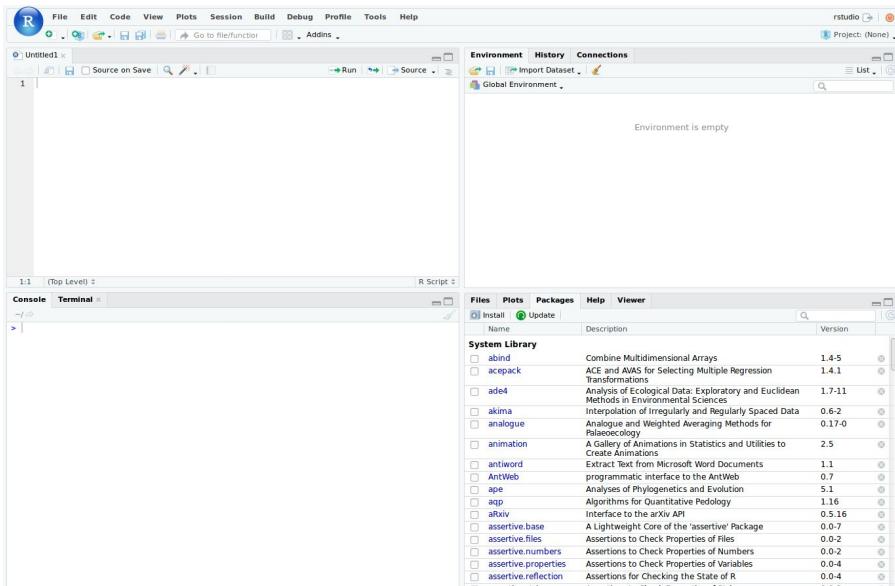
RStudio IDE



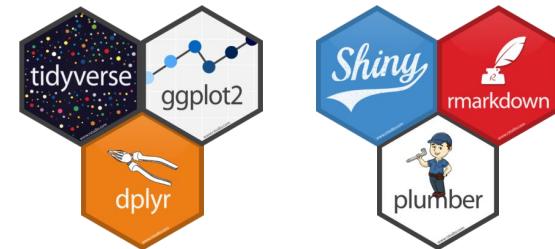
Background



RStudio IDE



Open Source

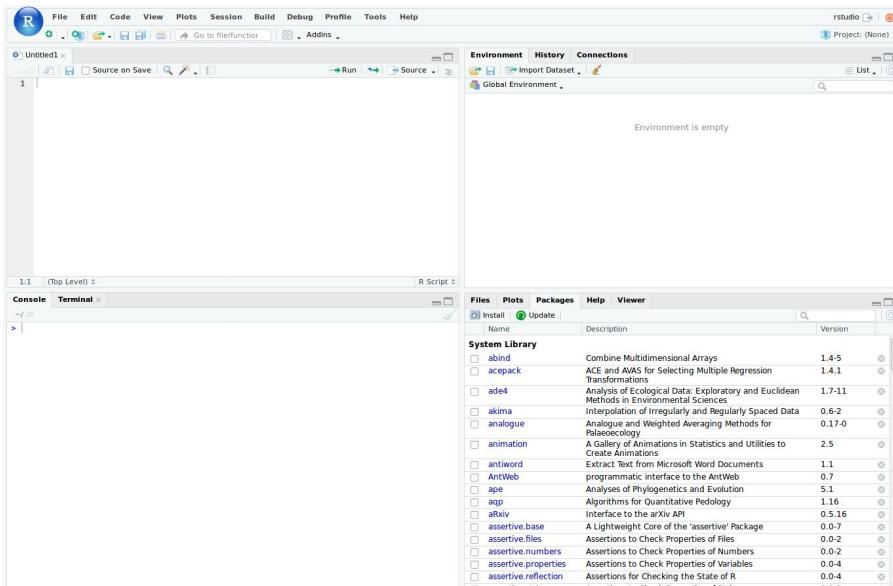


RStudio, Shiny, and R Markdown
are trademarks of RStudio, Inc.
Images from rstudio.com

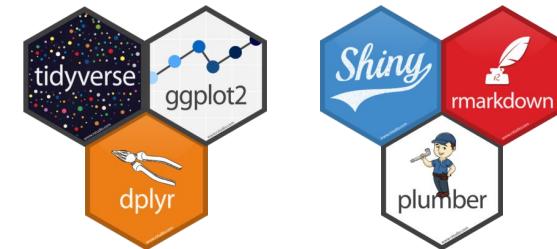
Background



RStudio IDE



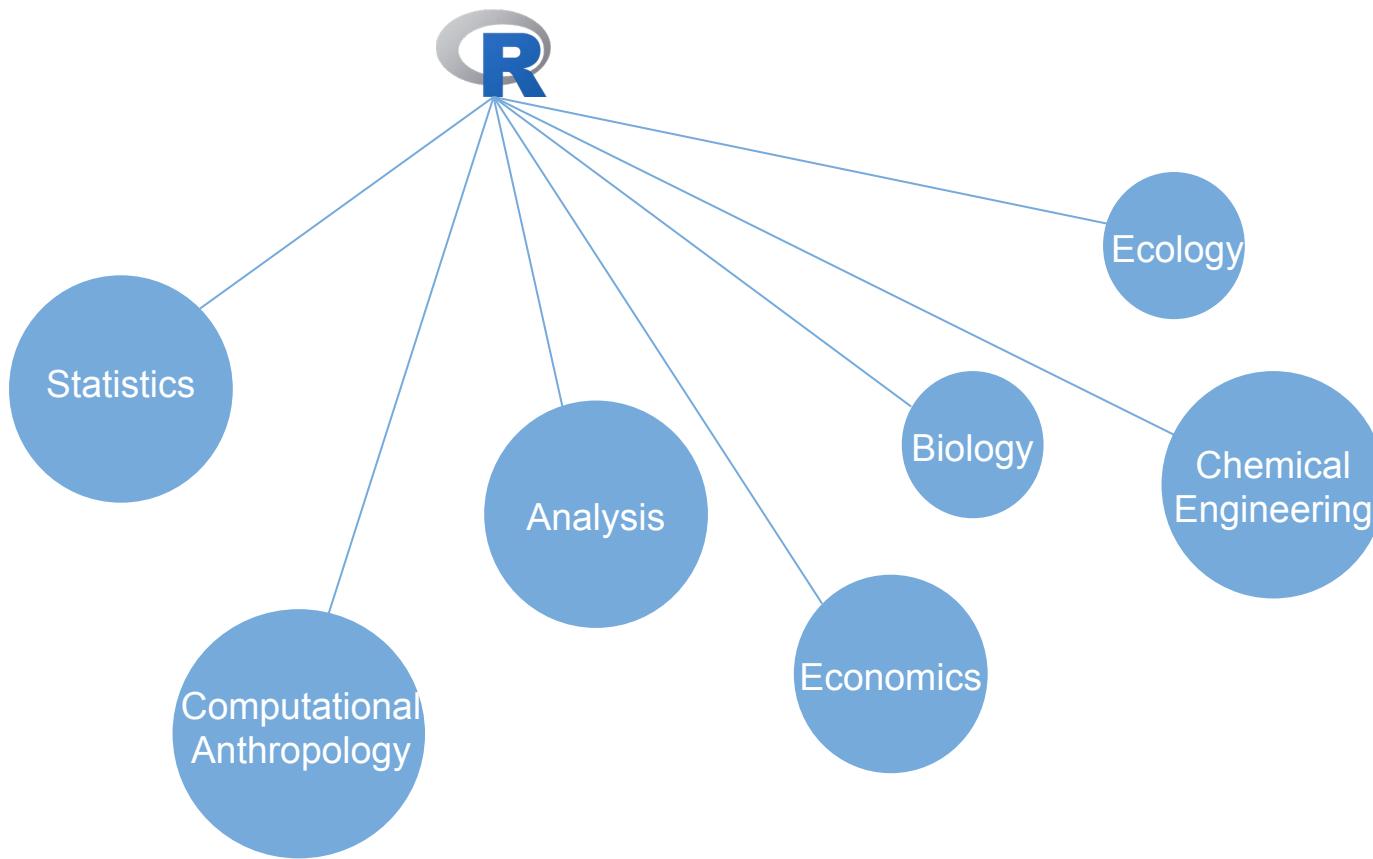
Open Source



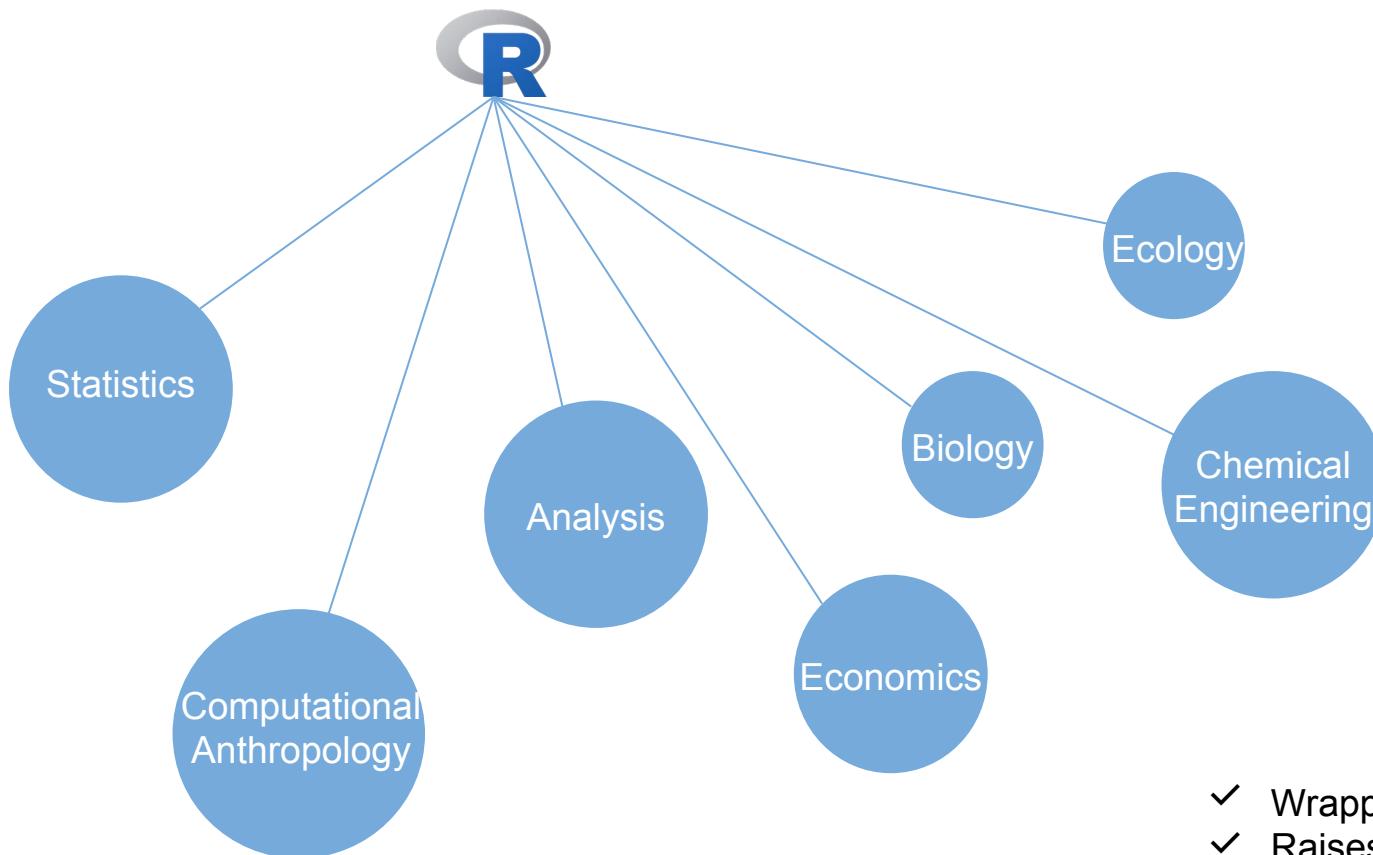
Enterprise-Ready Professional Products

- RStudio Server Pro and Commercial Desktop
- RStudio Connect
- RStudio Shiny Server Pro
- Shinyapps.io

Who is Using the R Language



Who is Using the R Language



- ✓ Wrapper around math libraries
- ✓ Raises the level of thinking

The “AGE of the WEB”

The “AGE of the WEB”

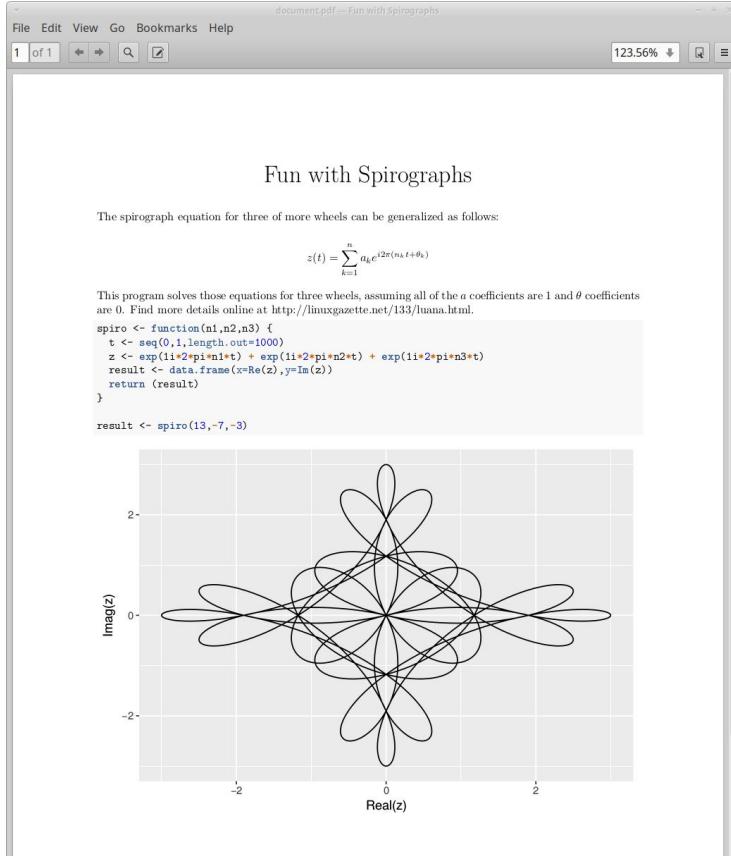


Copyright Columbia Pictures
Images from IMDB

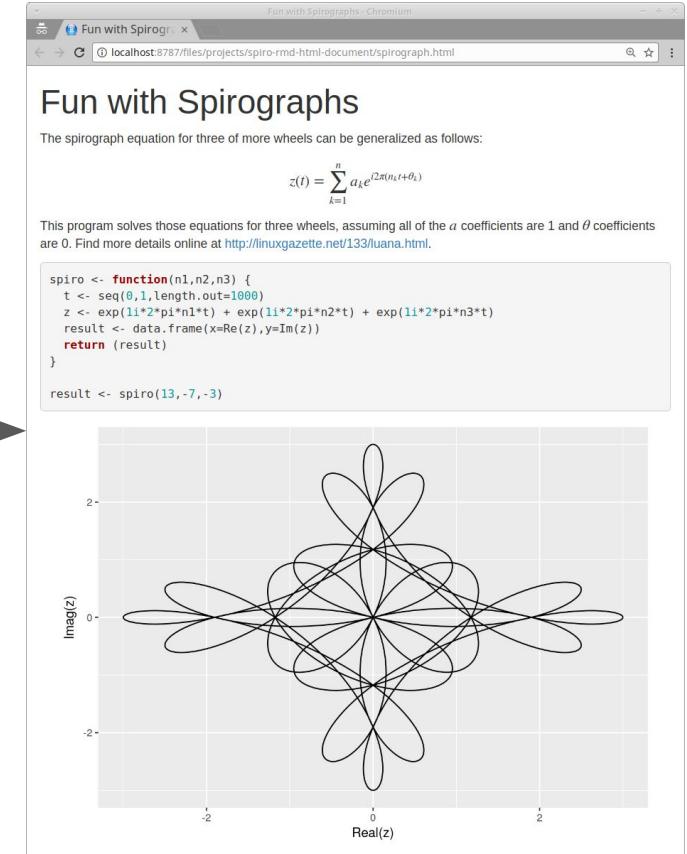
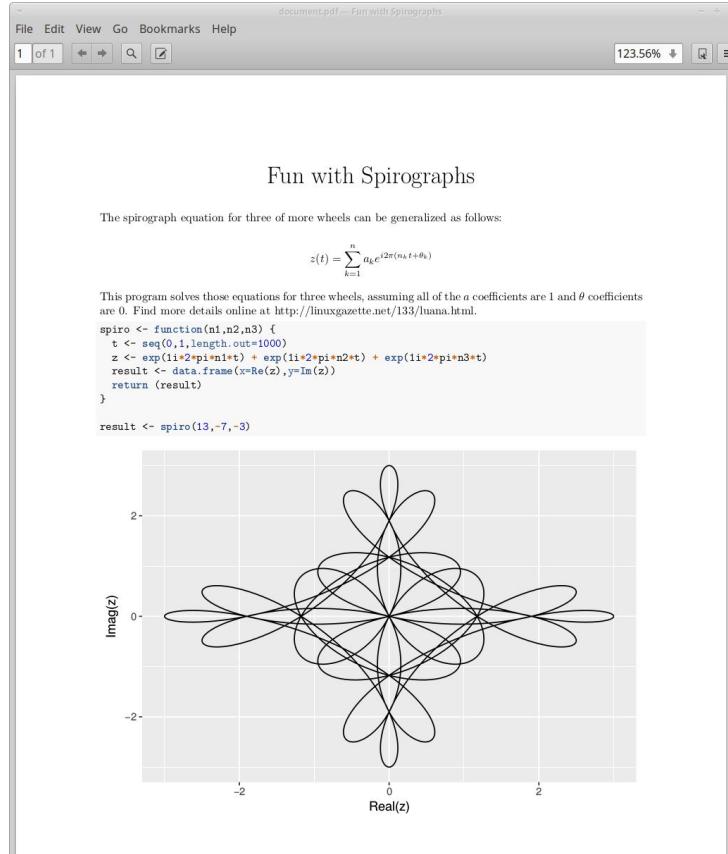
The “AGE of the WEB” 2.0

The “AGE of the WEB” **2.0** (or 1.x)

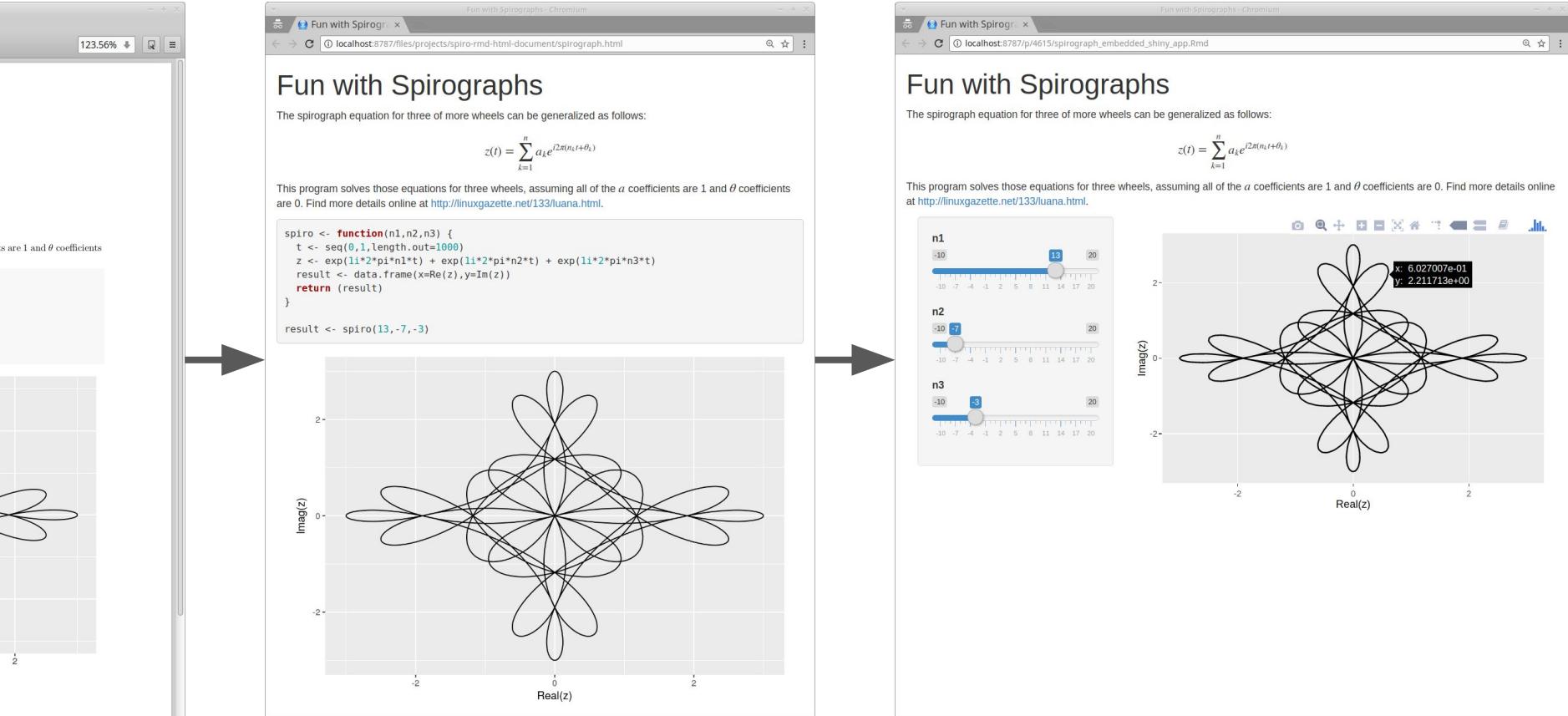
The “AGE of the WEB” 2.0 (or 1.x)



The “AGE of the WEB” 2.0 (or 1.x)



The “AGE of the WEB” 2.0 (or 1.x)



The “AGE of the WEB” 2.0 (or 1.x)

Fun with Spirographs

The spirograph equation for three or more wheels can be generalized as follows:

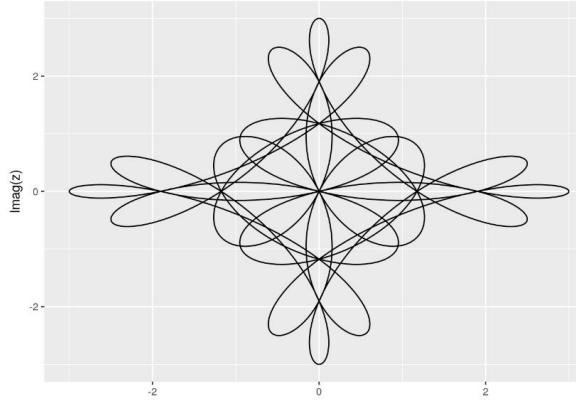
$$z(t) = \sum_{k=1}^n a_k e^{j2\pi(n_k t + \theta_k)}$$

This program solves those equations for three wheels, assuming all of the a coefficients are 1 and θ coefficients are 0. Find more details online at <http://linuxgazette.net/133/luana.html>.

```
spiro <- function(n1,n2,n3) {
  t <- seq(0,1,length.out=1000)
  z <- exp(j1*2*pi*n1*t) + exp(j1*2*pi*n2*t) + exp(j1*2*pi*n3*t)
  result <- data.frame(x=Re(z),y=Im(z))
  return (result)
}

result <- spiro(13,-7,-3)
```

2 are 1 and θ coefficients



Fun with Spirographs

The spirograph equation for three or more wheels can be generalized as follows:

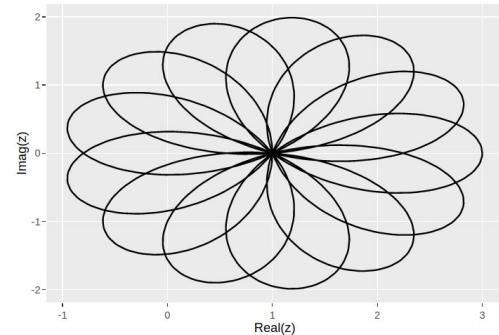
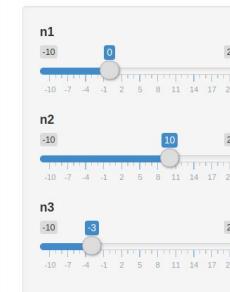
$$z(t) = \sum_{k=1}^n a_k e^{j2\pi(n_k t + \theta_k)}$$

This program solves those equations for three wheels, assuming all of the a coefficients are 1 and θ coefficients are 0. Find more details online at <http://linuxgazette.net/133/luana.html>.

n1

n2

n3



Today's Goals

Three ways to build reproducible, R based, content
that can be published on a science gateway.

Applications



<https://shiny.rstudio.com>

APIs



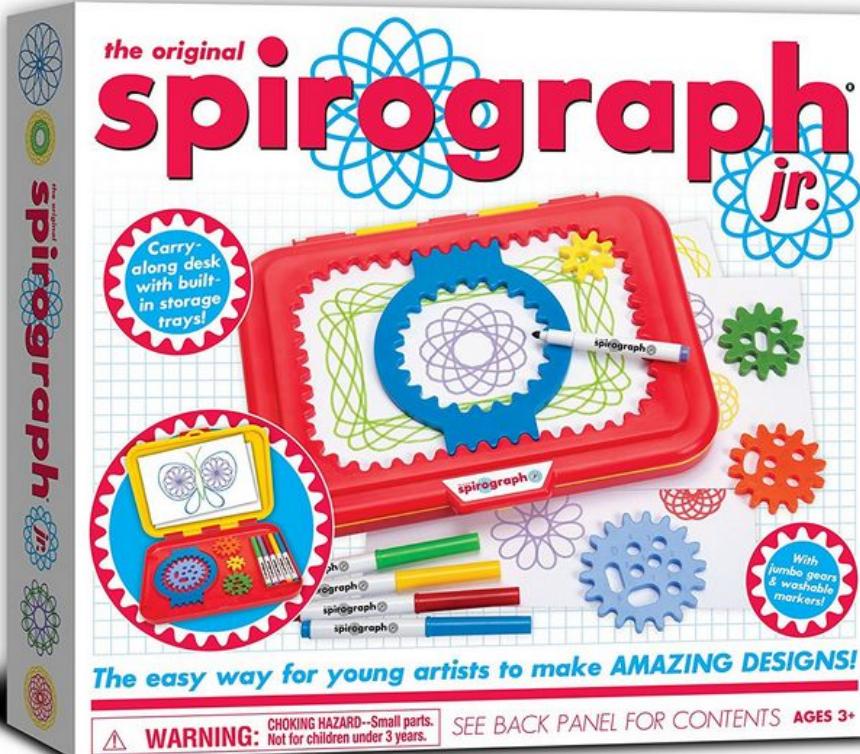
<https://www.rplumber.io>

Articles



<https://rmarkdown.rstudio.com>

Spirographs... For Ages 3+



Spirographs... For Ages 3+

[← prev](#) | [next →](#)

Plotting the spirograph equations with 'gnuplot'

By Victor Luana

Universidad de Oviedo, Departamento de Química Física y Analítica, E-33006-Oviedo, Spain.

[The author had specifically requested that we keep the large font used in his article in order to match the font size of the equation images: I agreed, since the two would look disproportionate otherwise. My apologies to anyone whose eyeballs exploded due to the rapid decompression.]

- Ben]

GNUPLOT's internal programming capabilities are used to plot the continuous and segmented versions of the spirograph equations. The segmented version, in particular, stretches the program model and requires the emulation of internal loops and conditional sentences. As a final exercise, we will develop an extensible mini-language, mixing GAWK and GNUPLOT programming, that lets the user combine any number of generalized spirographic patterns in a design.

A PDF version of this article is available for archiving and printing.

I. Introduction

Imagine the movement of a small circle that rolls, without slipping, on the inside of a rigid circle. Imagine now that the small circle has an arm, rigidly attached, with a plotting pen fixed at some point. That is a recipe for drawing the **hypotrochoid**, a member of a large family of curves including epitrochoids (the moving circle rolls on the outside of the fixed one), cycloids (the pen is on the edge of the rolling circle), and roulettes (several forms rolling on many different types of curves) in general.

The concept of wheels rolling on wheels can, in fact, be generalized to any number of embedded circles. Complex lathe engines, known as **Gulloché** machines, have been used since the 17th or 18th century for engraving beautiful designs onto watches, jewels, and other items of fine craftsmanship. Many sources attribute the first use of **Gulloché** engravings on a watch to Abraham-Louis Breguet in 1786, but the technique was already in use on jewelry. Ancient machines are still being used, and can be seen at the [RGM Watch Company Web pages](#). Intricate **Gulloché** patterns are usually incorporated on bank notes and official documents to prevent forgery. The name "Spirograph" comes, actually, from the trade name of a toy invented in 1962 by Denys Fisher, a British electronic engineer, and licensed to several toy companies over the years.

Our purpose, however, is not to explore the history or even the mathematical aspects of the Spirograph decorations: our interest is centered on the techniques needed to use GNUPLOT as the drawing engine of the cycloid-related curves.

Section II presents a simple derivation for the hypotrochoid equations and discusses a generalization to any number of rolling wheels described by [E. Fermi](#). Section III describes the techniques required to draw the cycloid-related curves with gnuplot. From the use of complex arithmetic to the simulation of an implicit do loop and the recursive definition of user functions, gnuplot offers a large capability for the creation of algorithmic designs. The techniques discussed in [Section III](#) are embedded within a simple GAWK filter that reads a formal description of a cycloid pattern and uses gnuplot to produce the final plot. The design of this filter is the subject of [Section IV](#).

II. The hypotrochoid and some related curves

Figure 1 shows the formation of a hypotrochoid and will help us in determining the parametric equations for the curve. Three lengths determine the shape of the curve: R, the radius of the fixed circle; r, the radius of the moving circle; and p, the distance from the pen to the moving circle center. The center of the fixed circle, point O, will serve as the origin of the coordinate system. Points O' and P designate the current position of the rolling circle center and of the pen, respectively.

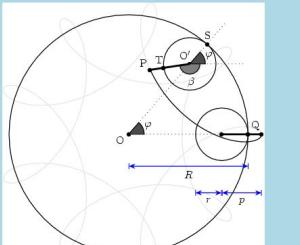


Figure 1 Geometry for the hypotrochoid equations. The grayed figure corresponds to $R=9$, $r=2$, and $p=3$.

Spirographs... For Ages 3+

← prev | next →

Plotting the spirograph equations with 'gnuplot'

By Victor Luana

Universidad de Oviedo, Departamento de Química Física y Analítica, E-33006-Oviedo, Spain.

[The author had specifically requested that we keep the large font used in his article in order to match the font size of the equation images; I agree, since the two would look disproportionate otherwise. My apologies to anyone whose eyeballs exploded due to the rapid decompression. - Ben]

GNUPLOT's internal programming capabilities are used to plot the continuous and segmented versions of the spirograph equations. The segmented version, in particular, stretches the program model and requires the emulation of internal loops and conditional sentences. As a final exercise, we will develop an extensible mini-language, mixing GAWK and GNUPLOT programming, that lets the user combine any number of generalized spirographic patterns in a design.

A PDF version of this article is available for archiving and printing.

I. Introduction

Imagine the movement of a small circle that rolls, without slipping, on the inside of a rigid circle. Imagine now that the small circle has an arm, rigidly attached, with a plotting pen fixed at some point. That is a recipe for drawing the **hypotrochoid**, a member of a large family of curves including epitrochoids (the moving circle rolls on the outside of the fixed one), cycloids (the pen is on the edge of the rolling circle), and roulettes (several forms rolling on many different types of curves) in general.

The concept of wheels rolling on wheels can, in fact, be generalized to any number of embedded circles. Complex lathe engines, known as **Gulloché** machines, have been used since the 17th or 18th century for engraving beautiful designs onto watches, jewels, and other items of fine craftsmanship. Many sources attribute the first use of **Gulloché** engravings on a watch to Abraham-Louis Breguet in 1786, but the technique was already in use on jewelry. Ancient machines are still being used, and can be seen at the [RGM Watch Company Web pages](#). Intricate **Gulloché** patterns are usually incorporated on bank notes and official documents to prevent forgery. The name "Spirograph" comes, actually, from the trade name of a toy invented in 1962 by Denys Fisher, a British electronic engineer, and licensed to several toy companies over the years.

Our purpose, however, is not to explore the history or even the mathematical aspects of the Spirograph decorations: our interest is centered on the techniques needed to use GNUPLOT as the drawing engine of the cycloid-related curves.

Section II presents a simple derivation for the hypotrochoid equations and discusses a generalization to any number of rolling wheels described by [Equation I](#). Section III describes the techniques required to draw the cycloid-related curves with gnuplot. From the use of complex arithmetic to the simulation of an implicit do loop and the recursive definition of user functions, gnuplot offers a large capability for the creation of algorithmic designs. The techniques discussed in [Section III](#) are embedded within a simple GAWK filter that reads a formal description of a cycloid pattern and uses gnuplot to produce the final plot. The design of this filter is the subject of [Section IV](#).

II. The hypotrochoid and some related curves

Figure 1 shows the formation of a hypotrochoid and will help us in determining the parametric equations for the curve. Three lengths determine the shape of the curve: R, the radius of the fixed circle; r, the radius of the moving circle; and p, the distance from the pen to the moving circle center. The center of the fixed circle, point O, will serve as the origin of the coordinate system. Points O' and P designate the current position of the rolling circle center and of the pen, respectively.

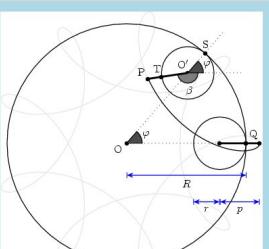


Figure 1 Geometry for the hypotrochoid equations. The grayed figure corresponds to R=9, r=2, and p=3.

TL;DR:

1. We can model spirographs as a sum of exponentials:

$$z(t) = \sum_{k=1}^n a_k e^{i2\pi(n_k t + \theta_k)}, t \in [0, 1]$$

Spirographs... For Ages 3+

← prev | next →

Plotting the spirograph equations with 'gnuplot'

By Victor Luana

Universidad de Oviedo, Departamento de Química Física y Analítica, E-33006-Oviedo, Spain.

[The author had specifically requested that we keep the large font used in his article in order to match the font size of the equation images: I agreed, since the two would look disproportionate otherwise. My apologies to anyone whose eyeballs exploded due to the rapid decompression. - Ben]

GNUPLOT's internal programming capabilities are used to plot the continuous and segmented versions of the spirograph equations. The segmented version, in particular, stretches the program model and requires the emulation of internal loops and conditional sentences. As a final exercise, we will develop an extensible mini-language, mixing GAWK and GNUPLOT programming, that lets the user combine any number of generalized spirographic patterns in a design.

A PDF version of this article is available for archiving and printing.

I. Introduction

Imagine the movement of a small circle that rolls, without slipping, on the inside of a rigid circle. Imagine now that the small circle has an arm, rigidly attached, with a plotting pen fixed at some point. That is a recipe for drawing the **hypotrochoid**, a member of a large family of curves including epitrochoids (the moving circle rolls on the outside of the fixed one), cycloids (the pen is on the edge of the rolling circle), and roulettes (several forms rolling on many different types of curves) in general.

The concept of wheels rolling on wheels can, in fact, be generalized to any number of embedded elements. Complex lathe engines, known as **Gulloché** machines, have been used since the 17th or 18th century for engraving beautiful designs onto watches, jewels, and other items of fine craftsmanship. Many sources attribute the first use of **Gulloché** engravings on a watch to Abraham-Louis Breguet in 1786, but the technique was already in use on jewelry. Ancient machines are still being used, and can be seen at the [RGM Watch Company Web pages](#). Intricate **Gulloché** patterns are usually incorporated on bank notes and official documents to prevent forgery. The name "Spirograph" comes, actually, from the trade name of a toy invented in 1962 by Denys Fisher, a British electronic engineer, and licensed to several toy companies over the years.

Our purpose, however, is not to explore the history or even the mathematical aspects of the Spirograph decorations: our interest is centered on the techniques needed to use GNUPLOT as the drawing engine of the cycloid-related curves.

Section II presents a simple derivation for the hypotrochoid equations and discusses a generalization to any number of rolling wheels described by [Equation 1](#). Section III describes the techniques required to draw the cycloid-related curves with gnuplot. From the use of complex arithmetic to the simulation of an implicit do loop and the recursive definition of user functions, gnuplot offers a large capability for the creation of algorithmic designs. The techniques discussed in [Section III](#) are embedded within a simple gawk filter that reads a formal description of a cycloid pattern and uses gnuplot to produce the final plot. The design of this filter is the subject of [Section IV](#).

II. The hypotrochoid and some related curves

Figure 1 shows the formation of a hypotrochoid and will help us in determining the parametric equations for the curve. Three lengths determine the shape of the curve: R, the radius of the fixed circle; r, the radius of the moving circle; and p, the distance from the pen to the moving circle center. The center of the fixed circle, point O, will serve as the origin of the coordinate system. Points O' and P designate the current position of the rolling circle center and of the pen, respectively.

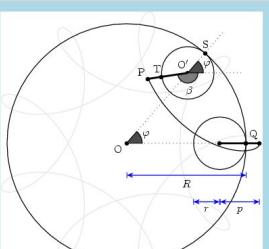


Figure 1 Geometry for the hypotrochoid equations. The grayed figure corresponds to R=9, r=2, and p=3.

TL;DR:

1. We can model spirographs as a sum of exponentials:

$$z(t) = \sum_{k=1}^n a_k e^{i2\pi(n_k t + \theta_k)}, t \in [0, 1]$$

2. We can model a 3-wheeled system, where $\alpha_k = 1$ and $\theta_k = 0$, with this equation:

$$z(t) = e^{i2\pi(n_1 t)} + e^{i2\pi(n_2 t)} + e^{i2\pi(n_3 t)}, t \in [0, 1]$$

Spirographs... For Ages 3+

← prev | next →

Plotting the spirograph equations with 'gnuplot'

By Victor Luana

Universidad de Oviedo, Departamento de Química Física y Analítica, E-33006-Oviedo, Spain.

[The author had specifically requested that we keep the large font used in his article in order to match the font size of the equation images: I agree, since the two would look disproportionate otherwise. My apologies to anyone whose eyeballs exploded due to the rapid decompression. - Ben]

GNUPLOT's internal programming capabilities are used to plot the continuous and segmented versions of the spirograph equations. The segmented version, in particular, stretches the program model and requires the emulation of internal loops and conditional sentences. As a final exercise, we will develop an extensible mini-language, mixing GAWK and GNUPLOT programming, that lets the user combine any number of generalized spirographic patterns in a design.

A PDF version of this article is available for archiving and printing.

I. Introduction

Imagine the movement of a small circle that rolls, without slipping, on the inside of a rigid circle. Imagine now that the small circle has an arm, rigidly attached, with a plotting pen fixed at some point. That is a recipe for drawing the **hypotrochoid**, a member of a large family of curves including epitrochoids (the moving circle rolls on the outside of the fixed one), cycloids (the pen is on the edge of the rolling circle), and roulettes (several forms rolling on many different types of curves) in general.

The concept of wheels rolling on wheels can, in fact, be generalized to any number of embedded elements. Complex lathe engines, known as **Gulloché** machines, have been used since the 17th or 10th century for engraving beautiful designs onto watches, jewels, and other items of fine craftsmanship. Many sources attribute the first use of **Gulloché** engravings on a watch to Abraham-Louis Breguet in 1786, but the technique was already in use on jewelry. Ancient machines are still being used, and can be seen at the [RGM Watch Company Web pages](#). Intricate **Gulloché** patterns are usually incorporated on bank notes and official documents to prevent forgery. The name "Spirograph" comes, actually, from the trade name of a toy invented in 1962 by Denys Fisher, a British electronic engineer, and licensed to several toy companies over the years.

Our purpose, however, is not to explore the history or even the mathematical aspects of the Spirograph decorations: our interest is centered on the techniques needed to use GNUPLOT as the drawing engine of the cycloid-related curves.

Section II presents a simple derivation for the hypotrochoid equations and discusses a generalization to any number of rolling wheels described by [Equation](#). Section III describes the techniques required to draw the cycloid-related curves with gnuplot. From the use of complex arithmetic to the simulation of an implicit do loop and the recursive definition of user functions, gnuplot offers a large capability for the creation of algorithmic designs. The techniques discussed in [Section III](#) are embedded within a simple gawk filter that reads a formal description of a cycloid pattern and uses gnuplot to produce the final plot. The design of this filter is the subject of [Section IV](#).

II. The hypotrochoid and some related curves

Figure 1 shows the formation of a hypotrochoid and will help us in determining the parametric equations for the curve. Three lengths determine the shape of the curve: R, the radius of the fixed circle; r, the radius of the moving circle; and p, the distance from the pen to the moving circle center. The center of the fixed circle, point O, will serve as the origin of the coordinate system. Points O' and P designate the current position of the rolling circle center and of the pen, respectively.

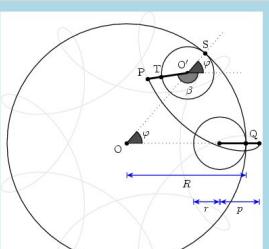


Figure 1 Geometry for the hypotrochoid equations. The grayed figure corresponds to R=9, r=2, and p=3.

TL;DR:

1. We can model spirographs as a sum of exponentials:

$$z(t) = \sum_{k=1}^n a_k e^{i2\pi(n_k t + \theta_k)}, t \in [0, 1]$$

2. We can model a 3-wheeled system, where $\alpha_k = 1$ and $\theta_k = 0$, with this equation:

$$z(t) = e^{i2\pi(n_1 t)} + e^{i2\pi(n_2 t)} + e^{i2\pi(n_3 t)}, t \in [0, 1]$$

3. Need three inputs from the user: n_1 , n_2 , and n_3

Spirographs... For Ages 3+

TL;DR:

```
spiro <- function(n1,n2,n3) {  
  t <- seq(0,1,length.out=1000)  
  z <- exp(1i*2*pi*n1*t) +  
    exp(1i*2*pi*n2*t) +  
    exp(1i*2*pi*n3*t)  
  result <- tibble(x=Re(z),y=Im(z))  
  return (result)  
}
```

1. We can model spirographs as a sum of exponentials:

$$z(t) = \sum_{k=1}^n a_k e^{i2\pi(n_k t + \theta_k)}, t \in [0, 1]$$

2. We can model a 3-wheeled system, where $\alpha_k = 1$ and $\theta_k = 0$, with this equation:

$$z(t) = e^{i2\pi(n_1 t)} + e^{i2\pi(n_2 t)} + e^{i2\pi(n_3 t)}, t \in [0, 1]$$

3. Need three inputs from the user: n_1 , n_2 , and n_3

Spirographs... For Ages 3+



```
spiro <- function(n1,n2,n3) {  
  t <- seq(0,1,length.out=1000)  
  z <- exp(1i*2*pi*n1*t) +  
    exp(1i*2*pi*n2*t) +  
    exp(1i*2*pi*n3*t)  
  result <- tibble(x=Re(z),y=Im(z))  
  return (result)  
}
```

TL;DR:

1. We can model spirographs as a sum of exponentials:

$$z(t) = \sum_{k=1}^n a_k e^{i2\pi(n_k t + \theta_k)}, t \in [0, 1]$$

2. We can model a 3-wheeled system, where $\alpha_k = 1$ and $\theta_k = 0$, with this equation:

$$z(t) = e^{i2\pi(n_1 t)} + e^{i2\pi(n_2 t)} + e^{i2\pi(n_3 t)}, t \in [0, 1]$$

3. Need three inputs from the user:

n₁, n₂, and n₃

Spirographs... For Ages 3+

TL;DR:

```
spiro <- function(n1,n2,n3) {  
  t <- seq(0,1,length.out=1000)  
  z <- exp(1i*2*pi*n1*t) +  
    exp(1i*2*pi*n2*t) +  
    exp(1i*2*pi*n3*t)  
  result <- tibble(x=Re(z),y=Im(z))  
  return (result)  
}
```

1. We can model spirographs as a sum of exponentials:

$$z(t) = \sum_{k=1}^n a_k e^{i2\pi(n_k t + \theta_k)}, \quad t \in [0, 1]$$

2. We can model a 3-wheeled system, where $\alpha_k = 1$ and $\theta_k = 0$, with this equation:

$$z(t) = e^{i2\pi(n_1 t)} + e^{i2\pi(n_2 t)} + e^{i2\pi(n_3 t)}, \quad t \in [0, 1]$$

3. Need three inputs from the user: n_1 , n_2 , and n_3

Spirographs... For Ages 3+

```
spiro <- function(n1,n2,n3) {  
  t <- seq(0,1,length.out=1000)  
  z <- exp(1i*2*pi*n1*t) +  
    exp(1i*2*pi*n2*t) +  
    exp(1i*2*pi*n3*t)  
  result <- tibble(x=Re(z),y=Im(z))  
  return (result)  
}
```



TL;DR:

1. We can model spirographs as a sum of exponentials:

$$z(t) = \sum_{k=1}^n a_k e^{i2\pi(n_k t + \theta_k)}, t \in [0, 1]$$

2. We can model a 3-wheeled system, where $\alpha_k = 1$ and $\theta_k = 0$, with this equation:

$$z(t) = e^{i2\pi(n_1 t)} + e^{i2\pi(n_2 t)} + e^{i2\pi(n_3 t)}, t \in [0, 1]$$

3. Need three inputs from the user: n_1 , n_2 , and n_3

Spirographs... For Ages 3+

```
spiro <- function(n1,n2,n3) {  
  t <- seq(0,1,length.out=1000)  
  z <- exp(1i*2*pi*n1*t) +  
    exp(1i*2*pi*n2*t) +  
    exp(1i*2*pi*n3*t)  
  result <- tibble(x=Re(z),y=Im(z))  
  return (result)  
}
```



TL;DR:

1. We can model spirographs as a sum of exponentials:

$$z(t) = \sum_{k=1}^n a_k e^{i2\pi(n_k t + \theta_k)}, t \in [0, 1]$$

2. We can model a 3-wheeled system, where $\alpha_k = 1$ and $\theta_k = 0$, with this equation:

$$z(t) = e^{i2\pi(n_1 t)} + e^{i2\pi(n_2 t)} + e^{i2\pi(n_3 t)}, t \in [0, 1]$$

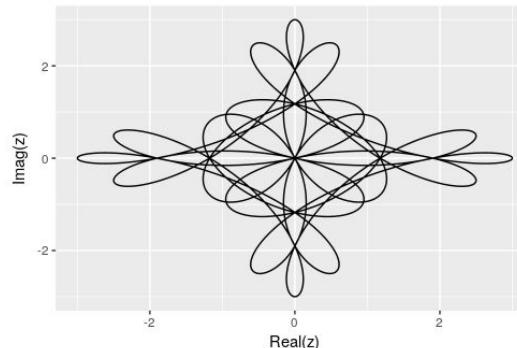
3. Need three inputs from the user: n_1 , n_2 , and n_3

Spirographs... For Ages 3+

```
spiro <- function(n1,n2,n3) {  
  t <- seq(0,1,length.out=1000)  
  z <- exp(1i*2*pi*n1*t) +  
    exp(1i*2*pi*n2*t) +  
    exp(1i*2*pi*n3*t)  
  result <- tibble(x=Re(z),y=Im(z))  
  return (result)  
}
```

→

```
> z <- spiro(13,-7,-3)  
> z  
# A tibble: 1,000 x 2  
      x     y  
   <dbl> <dbl>  
1     3     0  
2     3.00  0.0188  
3     2.98  0.0371  
4     2.96  0.0546  
5     2.93  0.0707  
6     2.89  0.0850  
7     2.84  0.0971  
8     2.78  0.107  
9     2.72  0.113  
10    2.65  0.116  
# ... with 990 more rows  
> ggplot(...)
```



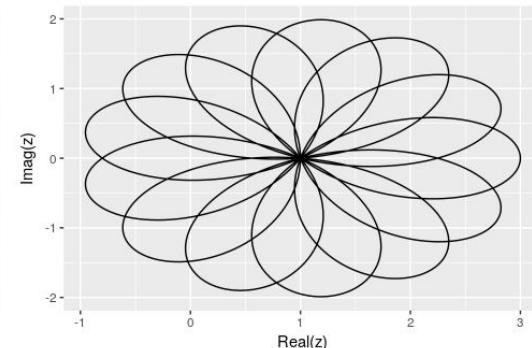
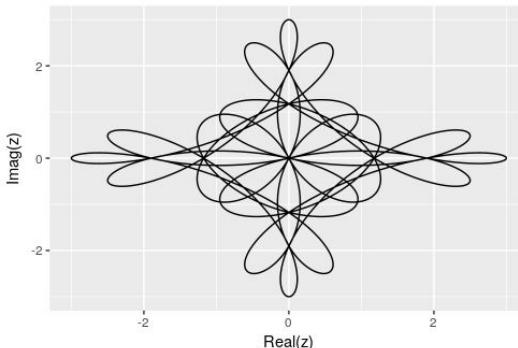
Spirographs... For Ages 3+

```
spiro <- function(n1,n2,n3) {  
  t <- seq(0,1,length.out=1000)  
  z <- exp(1i*2*pi*n1*t) +  
    exp(1i*2*pi*n2*t) +  
    exp(1i*2*pi*n3*t)  
  result <- tibble(x=Re(z),y=Im(z))  
  return (result)  
}
```

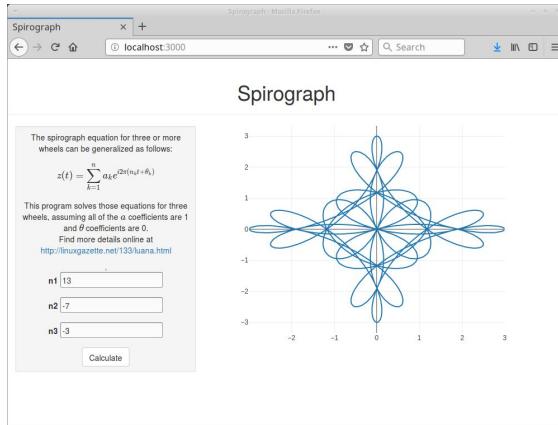
```
> z <- spiro(13,-7,-3)  
> z  
# A tibble: 1,000 × 2  
      x     y  
   <dbl> <dbl>  
1  3.00  0.0188  
2  2.98  0.0371  
3  2.96  0.0546  
4  2.93  0.0707  
5  2.89  0.0850  
6  2.84  0.0971  
7  2.78  0.107  
8  2.72  0.113  
9  2.65  0.116  
# ... with 990 more rows  
> ggplot(...)
```

→

```
> z <- spiro(0,10,-3)  
> z  
# A tibble: 1,000 × 2  
      x     y  
   <dbl> <dbl>  
1  3.00  0.0000  
2  2.99  0.0440  
3  2.98  0.0877  
4  2.98  0.131  
5  2.97  0.174  
6  2.95  0.215  
7  2.92  0.256  
8  2.90  0.294  
9  2.86  0.332  
10 2.83  0.367  
# ... with 990 more rows  
> ggplot(...)
```



Web Applications



POST “/”
localhost:3000

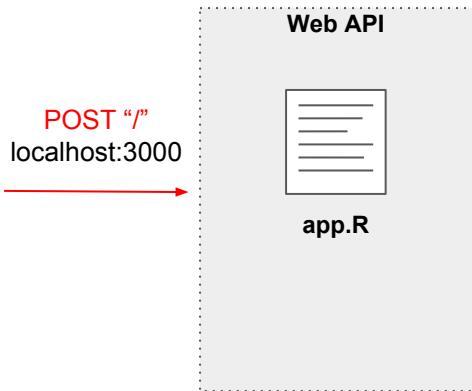
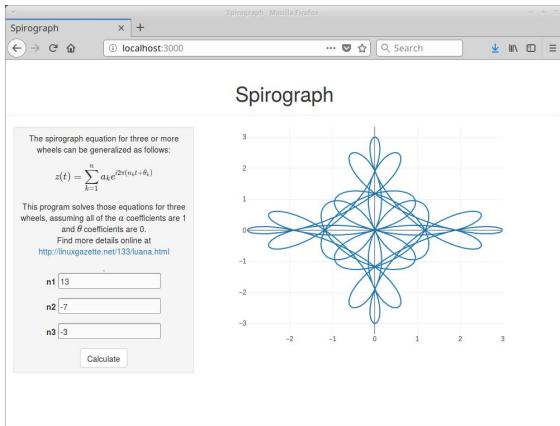
Web API



app.R



Building Web Applications with Shiny



```
spiro <- function(n1,n2,n3) {  
  t <- seq(0, 1, length.out=1000)  
  z <- exp(1i*2*pi*n1*t) +  
    exp(1i*2*pi*n2*t) +  
    exp(1i*2*pi*n3*t)  
  result <- tibble(x=Re(z),y=Im(z))  
  return (result)  
}  
  
# Define UI for spirograph application  
ui <- fluidPage(  
  
  # Input and Output  
  # User Interface Widgets  
  
)  
  
# Define server logic required to draw  
# a spirograph  
server <- function(input, output) {  
  
  # Call spiro()  
  # Return x,y points  
  
}  
  
# Run the application  
shinyApp(ui,server)
```



Building Web Applications with Shiny

Inputs

Action

Choice A

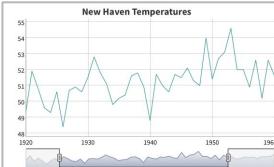
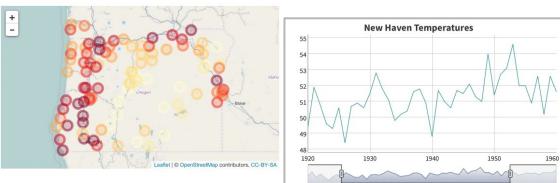
2014-01-01

Choice 1

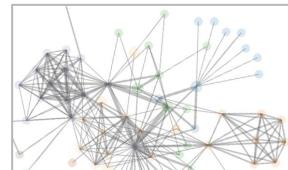
2018-05-08 to 2018-05-08

0 25 75 100

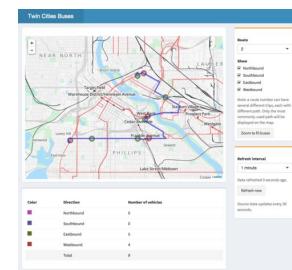
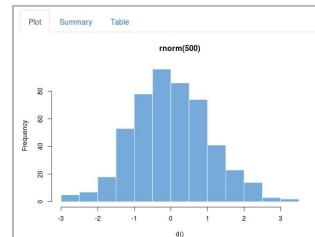
Outputs



- Choice 1
- Choice 2
- Choice 3



Layouts



```
spiro <- function(n1,n2,n3) {  
  t <- seq(0,1,length.out=1000)  
  z <- exp(1i*2*pi*n1*t) +  
    exp(1i*2*pi*n2*t) +  
    exp(1i*2*pi*n3*t)  
  result <- tibble(x=Re(z),y=Im(z))  
  return (result)  
}
```

```
# Define UI for spirograph application  
ui <- fluidPage(
```

```
  # Input and Output  
  # User Interface Widgets  
)
```

```
# Define server logic required to draw  
# a spirograph
```

```
server <- function(input, output) {
```

```
  # Call spiro()  
  # Return x,y points
```

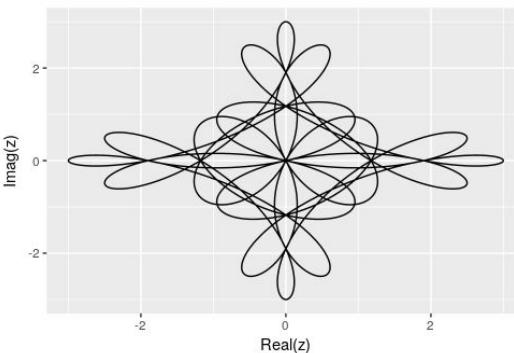
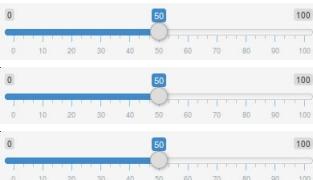
```
}
```

```
# Run the application  
shinyApp(ui,server)
```



Building Web Applications with Shiny

Server glues widgets to science code

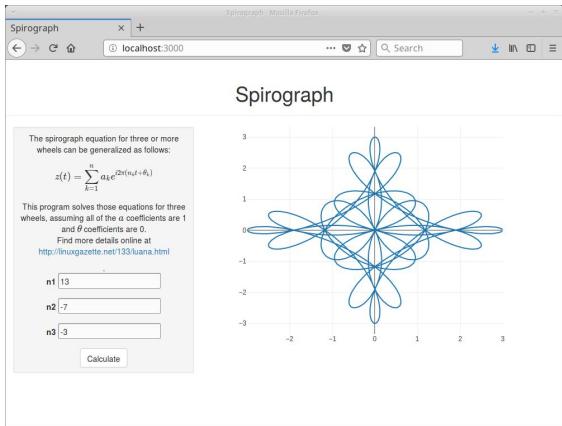


- ✓ Reactivity - responds to user changes
- ✓ Brushing - linking widgets together

```
spiro <- function(n1,n2,n3) {  
  t <- seq(0, 1, length.out=1000)  
  z <- exp(1i*2*pi*n1*t) +  
    exp(1i*2*pi*n2*t) +  
    exp(1i*2*pi*n3*t)  
  result <- tibble(x=Re(z),y=Im(z))  
  return (result)  
}  
  
# Define UI for spirograph application  
ui <- fluidPage(  
  
  # Input and Output  
  # User Interface Widgets  
  
)  
  
# Define server logic required to draw  
# a spirograph  
server <- function(input, output) {  
  
  # Call spiro()  
  # Return x,y points  
  
}  
  
# Run the application  
shinyApp(ui,server)
```



Building Web Applications with Shiny



POST “/”
localhost:3000



```
spiro <- function(n1,n2,n3) {  
  t <- seq(0, 1, length.out=1000)  
  z <- exp(1i*2*pi*n1*t) +  
    exp(1i*2*pi*n2*t) +  
    exp(1i*2*pi*n3*t)  
  result <- tibble(x=Re(z),y=Im(z))  
  return (result)  
}  
  
# Define UI for spirograph application  
ui <- fluidPage(  
  
  # Input and Output  
  # User Interface Widgets  
  
)  
  
# Define server logic required to draw  
# a spirograph  
server <- function(input, output) {  
  
  # Call spiro()  
  # Return x,y points  
  
}  
  
# Run the application  
shinyApp(ui,server)
```



Getting Started with Shiny

Interact. Analyze. Communicate.

Take a fresh, interactive approach to telling your data story with Shiny. Let users interact with your data and your analysis. And do it all with R.

Shiny is an R package that makes it easy to build interactive web apps straight from R. You can host stand-alone apps on a webpage or embed them in other sites.

<https://shiny.rstudio.com>

The basic parts of a Shiny app

LAST UPDATED: 28 JUN 2017

The Shiny package comes with eleven built-in examples that demonstrate how Shiny works. This article reviews the first three examples, which demonstrate the basic structure of a Shiny app.

Example 1: Hello Shiny

The Hello Shiny example is a simple application that plots R's built-in `iris` dataset with a configurable number of bins. To run the example, type:

```
library(shiny)
```

<https://shiny.rstudio.com/articles/basics.html>

shinydashboard makes it easy to use Shiny to create dashboards like these:

Dashboard

Time Zone Map

<https://htmlwidgets.org>

htmlwidgets for R

Bring the best of JavaScript data visualization to R

Use JavaScript visualization libraries at the R console, just like plots

Embed widgets in R Markdown documents and Shiny web applications

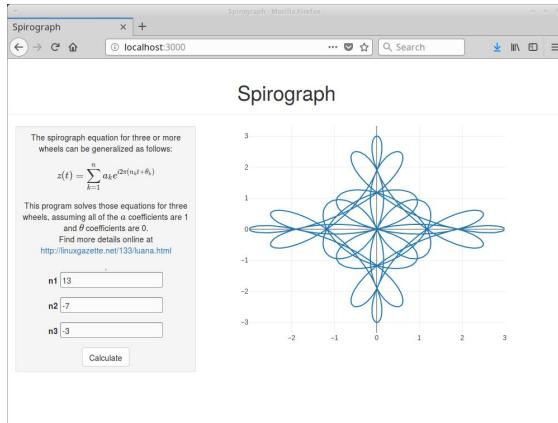
Develop new widgets using a framework that seamlessly integrates R and JavaScript

Widgets in action

Interactive dashboards

<https://rstudio.github.io/shinydashboard>

Web Application Basics



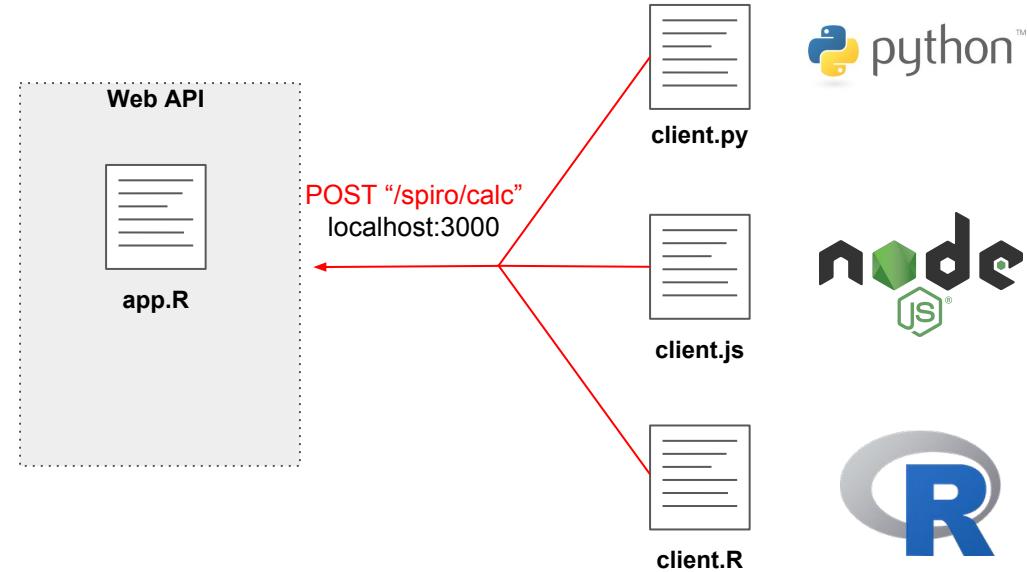
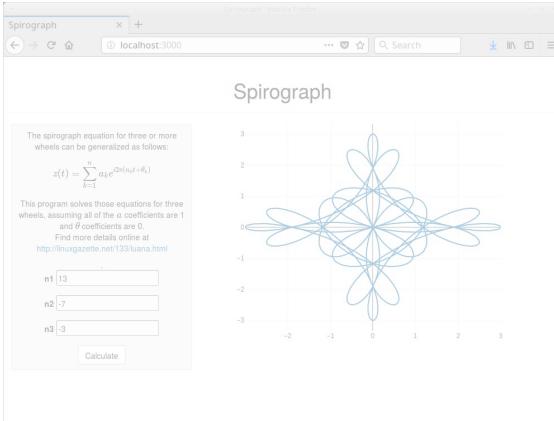
POST “/”
localhost:3000

Web Application



app.R

Web API Basics

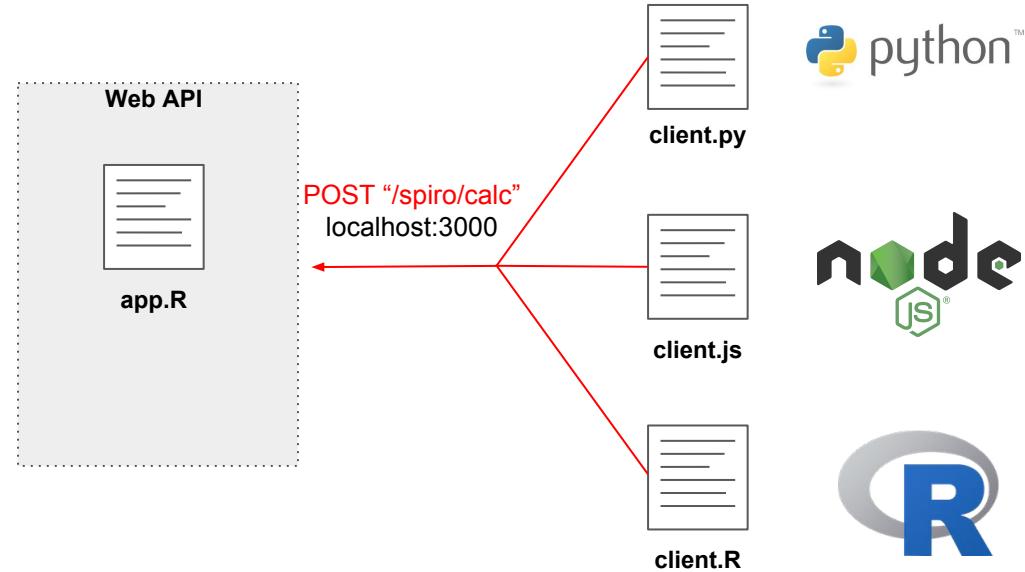
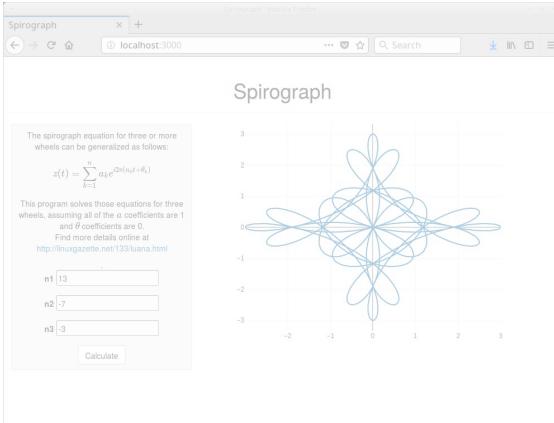


 python™

 node.js®

 R

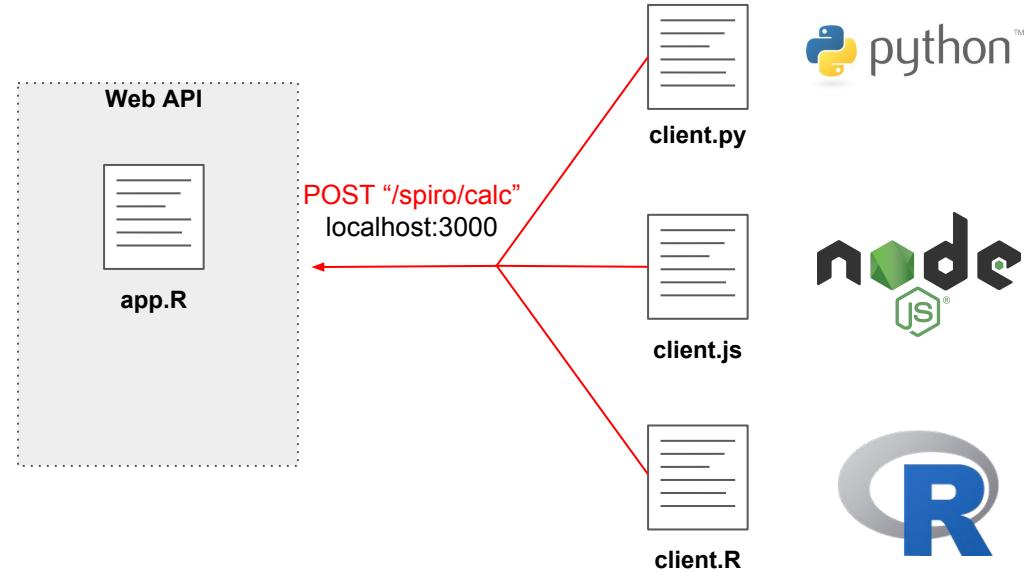
Web API Basics



- ✓ Still sending messages
- ✓ Focused on sharing information
- ✓ Communication with other programs

Web API Basics

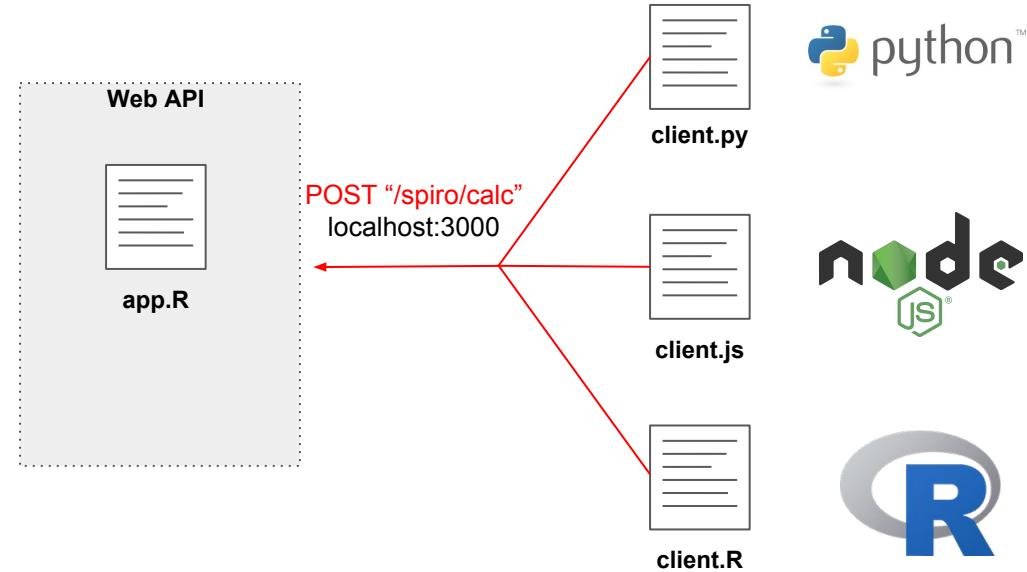
Who's Using Web APIs?



- ✓ Still sending messages
- ✓ Focused on sharing information
- ✓ Communication with other programs

Web API Basics

Who ^ Isn't Using Web APIs?



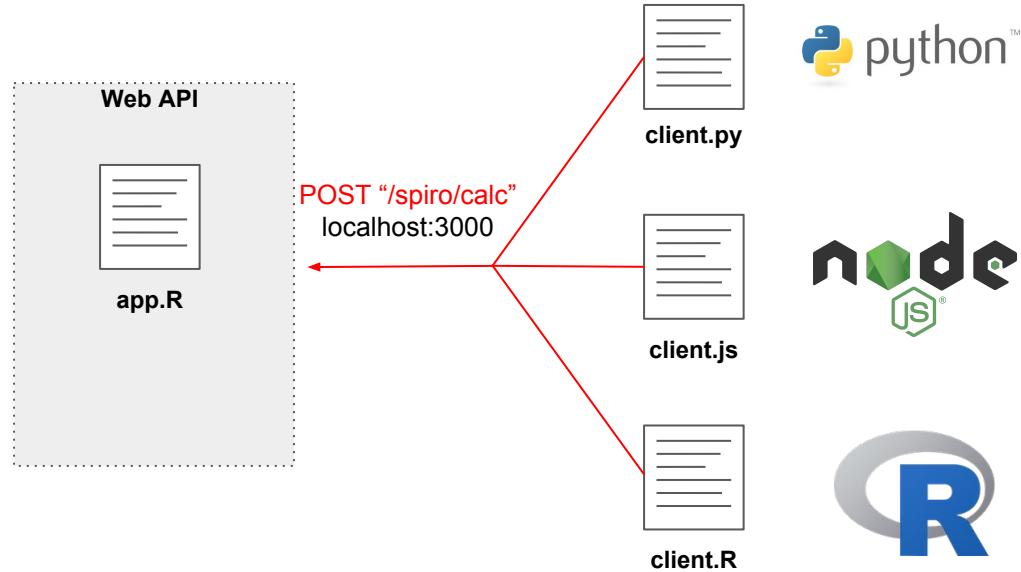
- ✓ Still sending messages
- ✓ Focused on sharing information
- ✓ Communication with other programs

Web API Basics

Who ^{Isn't} Using Web APIs?

The screenshot shows the Chronicling America homepage with a red watermark "Who Isn't Using Web APIs?". Below the watermark, the page content includes:

- Library of Congress logo and navigation links: ASK A LIBRARIAN, DIGITAL COLLECTIONS, LIBRARY CATALOG.
- Search bar: Search Search Loc.gov
- Main content area: CHRONICLING AMERICA Historic American Newspapers. Subtext: Search America's historic newspaper pages from 1709-1963 in our U.S. Newspaper Directory to find information about American newspapers published between 1680-present. Chronicling America is sponsored jointly by the National Endowment for the Humanities and the Library of Congress. Learn More.
- Section: About the Site and API. Subtext: Chronicling America provides access to information about historic newspapers and select digitized newspaper pages. To encourage a wide range of potential uses, we designed several different views of the data we provide, of which are publicly visible. Each uses common Web protocols, and access is not restricted in any way. You do not need to apply for a special key to use them. Together they make up an extensive application programming interface (API).
- Section: Details about these interfaces are below. In case you want to dive right in, though, we set up some conventions to advertise the availability of these views. If you are a software developer or researcher or anyone else who might be interested in programmatic access to the data in Chronicling America, we encourage you to look around the site, "view source" often, and follow where the different links take you to get started.
- Section: If you're interested in other data and machine-useable interfaces available from the Library of Congress, you might find the LC for Robots page helpful at http://lcweb.loc.gov/for_robots/.
- Section: The API. Subtext: Jump to:
 - Search the newspaper directory and digitized page contents using OpenSearch.
 - Auto Suggest API for looking up newspaper titles.
 - API for finding newspaper records.
 - JSON views of Chronicling America resources.
 - Linked Data views of Chronicling America resources.
 - Bulk Data for research and external services.
 - CORS and JSON support for your JavaScript applications.
- Section: Searching the directory and newspaper pages using OpenSearch. Subtext: The directory of newspaper files contains nearly 140,000 records of newspapers and libraries that hold copies of these newspapers. The title records are based on MARC data gathered and enhanced as part of the NNDPL program.
- Section: Searching the title records is possible using the [OpenSearch protocol](#). This is advertised in a LINK header element of the site's HTML template as "NNDPL Title Search", using the [OpenSearch Description documents](#).

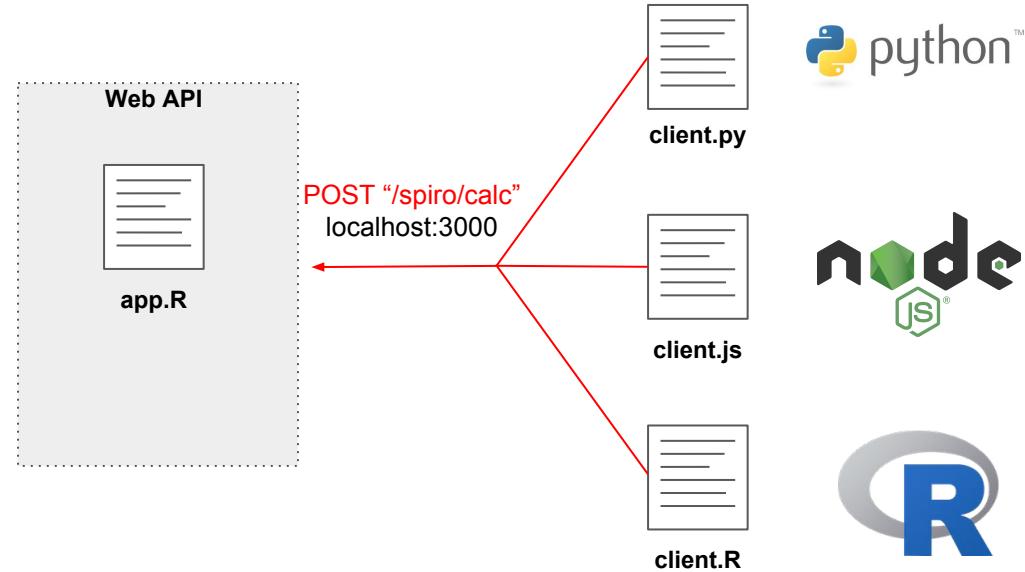
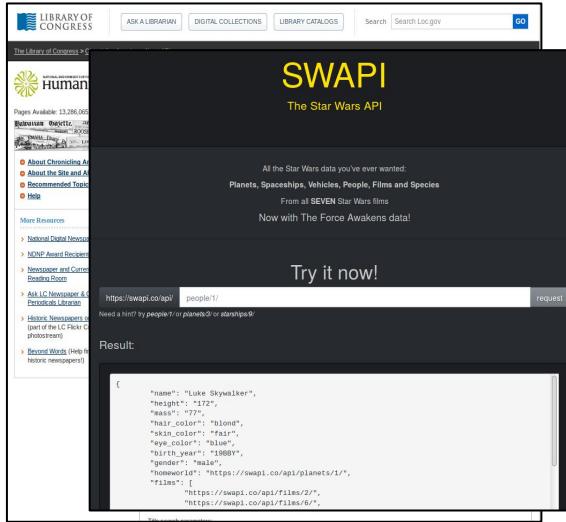


- ✓ Still sending messages
- ✓ Focused on sharing information
- ✓ Communication with other programs



Web API Basics

Who ^{Isn't} Using Web APIs?



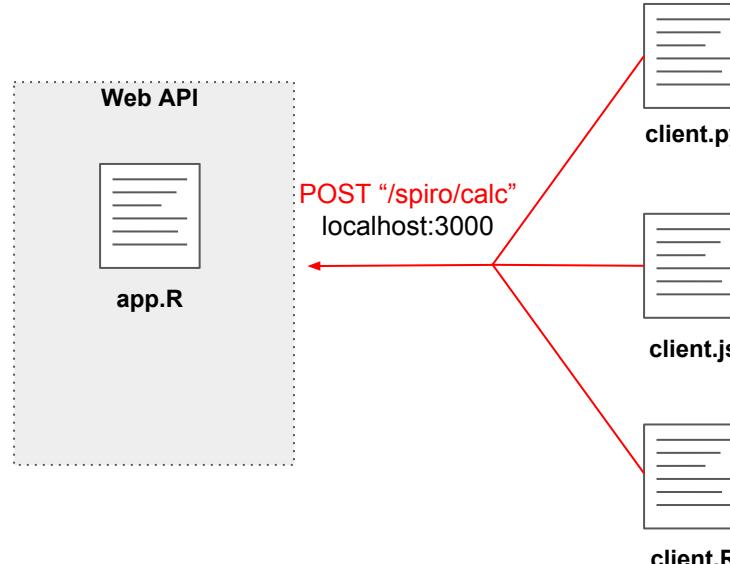
- ✓ Still sending messages
- ✓ Focused on sharing information
- ✓ Communication with other programs



Web API Basics

~~Who Isn't~~
Who ^ Using Web APIs?

The top screenshot shows the SWAPI (Star Wars API) website. It features a dark background with the title "SWAPI" and "The Star Wars API". Below the title, it says "All the Star Wars data you've ever wanted:". The bottom screenshot shows the SEPTA API website. It has a header with the SEPTA logo and navigation links like "Getting Around", "About", "Customer Service", "Media", "Contact Us", and "Business". A sidebar on the left titled "Choose Your Service" lists various transportation options: Regional Rail, Market-Frankford Line, Broad Street Line, Trolley Lines, Norristown High Speed Line, Buses, and CCT Connect. The main content area shows a "Trip Planner" section with fields for "From", "To", date "5/08/18", and time "7:36 PM". It also includes "Quick Links" for Schedules, Maps, Fares, and System Status.

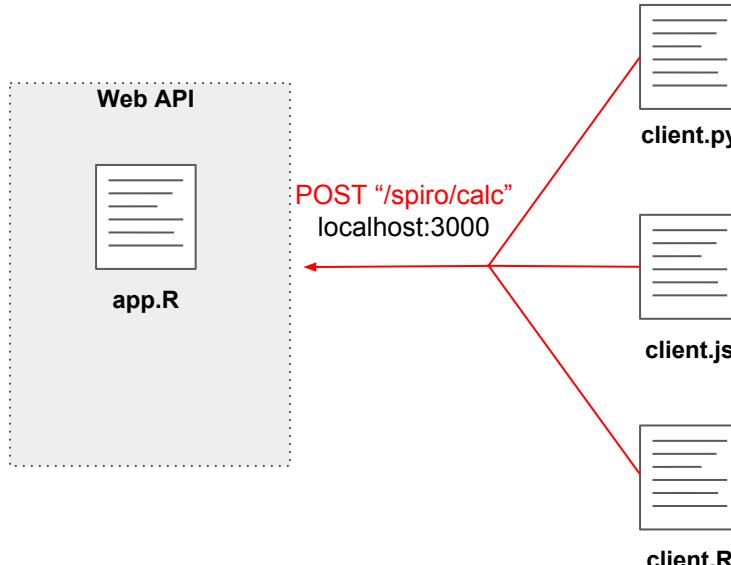


- ✓ Still sending messages
- ✓ Focused on sharing information
- ✓ Communication with other programs



Web API Basics

Who Isn't Using Web APIs?



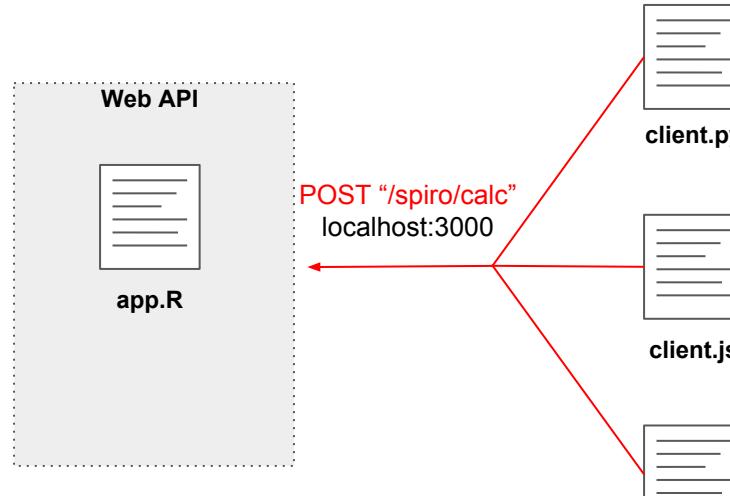
- ✓ Still sending messages
 - ✓ Focused on sharing information
 - ✓ Communication with other programs

Web API Basics

Who ^{Isn't} Using Web APIs?

The image contains three side-by-side screenshots of different web APIs:

- SWAPI**: A screenshot of the Star Wars API (SWAPI) documentation. It shows a search bar at the top, followed by sections for "OpenFEC API Documentation" and "NASA APIs". The "NASA APIs" section features a large image of a galaxy and text about the NASA API portal.
- OpenFEC API Documentation**: A screenshot of the OpenFEC API documentation. It includes a logo for FEC, a search bar, and sections for "Getting Started", "Live Example", "Get Your API Key", "Authentication", "NASA API Listing", "Contributing", and "About".
- NASA APIs**: A screenshot of the NASA API portal. It has a sidebar with links like "Getting Started", "Live Example", "Get Your API Key", "Authentication", "NASA API Listing", "Contributing", and "About". The main content area discusses the NASA API portal and provides links to "NASA DATA PORTAL", "NASA ON GITHUB", and "NASA OPEN SOURCE".



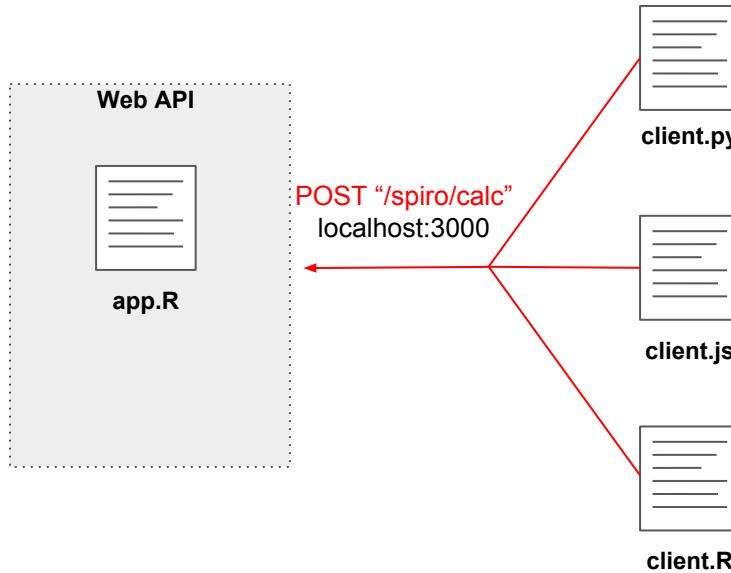
- ✓ Still sending messages
- ✓ Focused on sharing information
- ✓ Communication with other programs

```

spiro <- function(n1=13,n2=-7,n3=-3) {
  t <- seq(0,1,length.out=1000)
  z <- exp(1i*2*pi*n1*t) +
    exp(1i*2*pi*n2*t) +
    exp(1i*2*pi*n3*t)
  result <- tibble(x=Re(z),y=Im(z))
  return(result)
}

```

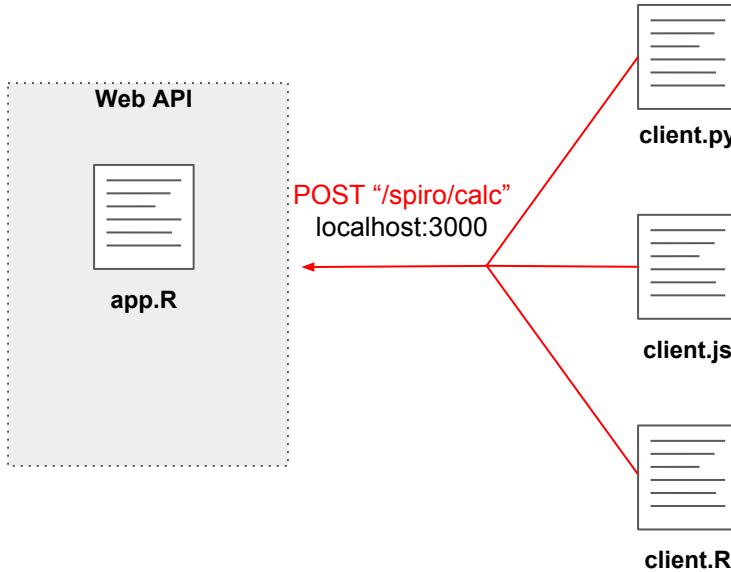
Building Web APIs with Plumber





Building Web APIs with Plumber

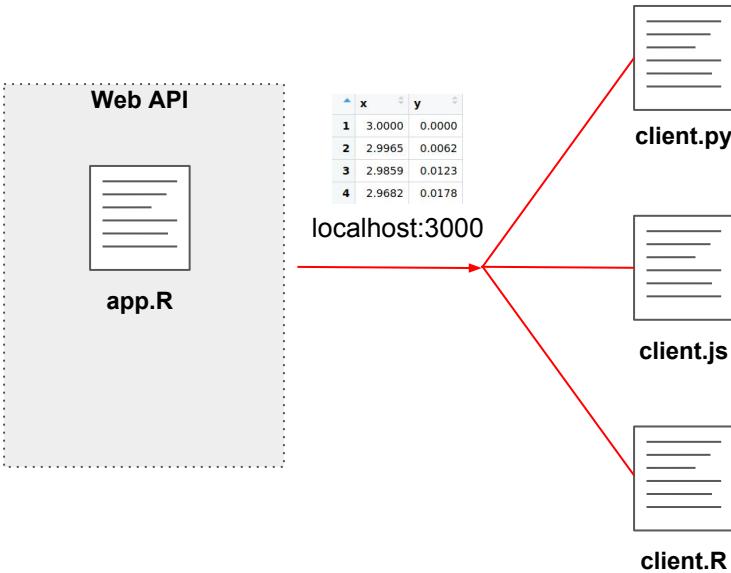
```
spiro <- function(n1=13, n2=-7, n3=-3) {  
  t <- seq(0, 1, length.out=1000)  
  z <- exp(1i*2*pi*n1*t) +  
    exp(1i*2*pi*n2*t) +  
    exp(1i*2*pi*n3*t)  
  result <- tibble(x=Re(z), y=Im(z))  
  return(result)  
}  
  
## Spirograph with custom n1, n2, n3  
## @post /spiro/calc  
## @param n3 Characteristics for the third wheel  
## @param n2 Characteristics for the second wheel  
## @param n1 Characteristics for the first wheel  
## @json  
function(n1,n2,n3){  
  return(spiro(as.numeric(n1),  
               as.numeric(n2),  
               as.numeric(n3)))  
}
```





Building Web APIs with Plumber

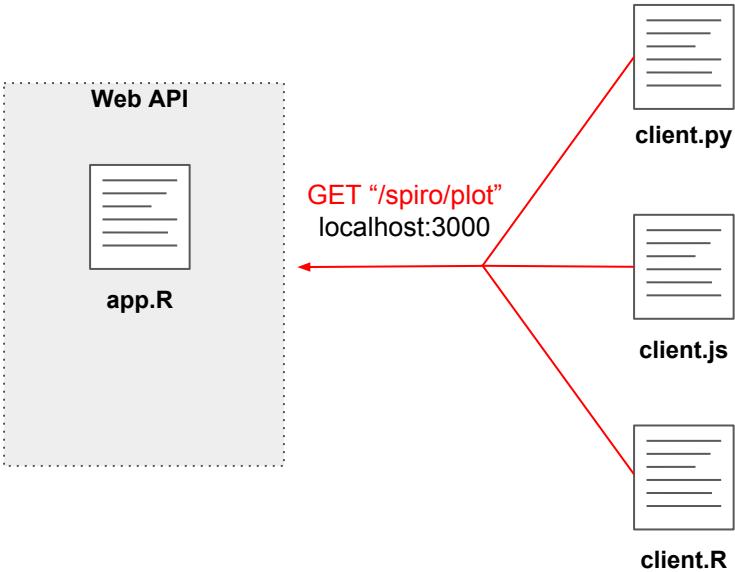
```
spiro <- function(n1=13, n2=-7, n3=-3) {  
  t <- seq(0, 1, length.out=1000)  
  z <- exp(1i*2*pi*n1*t) +  
    exp(1i*2*pi*n2*t) +  
    exp(1i*2*pi*n3*t)  
  result <- tibble(x=Re(z), y=Im(z))  
  return(result)  
}  
  
## Spirograph with custom n1, n2, n3  
## @post /spiro/calc  
## @param n3 Characteristics for the third wheel  
## @param n2 Characteristics for the second wheel  
## @param n1 Characteristics for the first wheel  
## @json  
function(n1,n2,n3){  
  return(spiro(as.numeric(n1),  
               as.numeric(n2),  
               as.numeric(n3)))  
}
```





Building Web APIs with Plumber

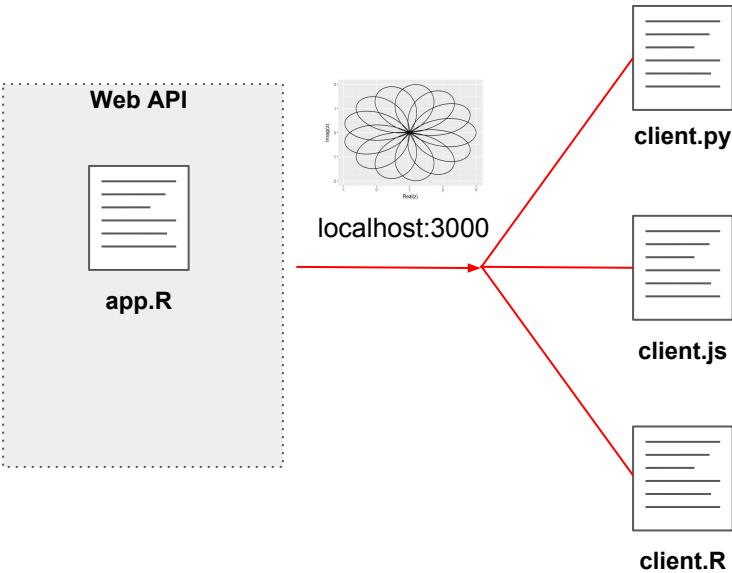
```
spiro <- function(n1=13, n2=-7, n3=-3) {  
  t <- seq(0, 1, length.out=1000)  
  z <- exp(1i*2*pi*n1*t) +  
    exp(1i*2*pi*n2*t) +  
    exp(1i*2*pi*n3*t)  
  result <- tibble(x=Re(z), y=Im(z))  
  return(result)  
}  
  
## Spirograph with custom n1, n2, n3  
## @post /spiro/calc  
## @param n3 Characteristics for the third wheel  
## @param n2 Characteristics for the second wheel  
## @param n1 Characteristics for the first wheel  
## @json  
function(n1,n2,n3){  
  return(spiro(as.numeric(n1),  
               as.numeric(n2),  
               as.numeric(n3)))  
}  
  
## Spirograph plot  
## @get /spiro/plot  
## @param n3 Characteristics for the third wheel  
## @param n2 Characteristics for the second wheel  
## @param n1 Characteristics for the first wheel  
## @png  
function(n1,n2,n3){  
  result <- spiro(as.numeric(n1),  
                  as.numeric(n2),  
                  as.numeric(n3))  
  plot(result$x, result$y,  
        xlab="Real(z)",  
        ylab="Imag(z)")  
}
```





Building Web APIs with Plumber

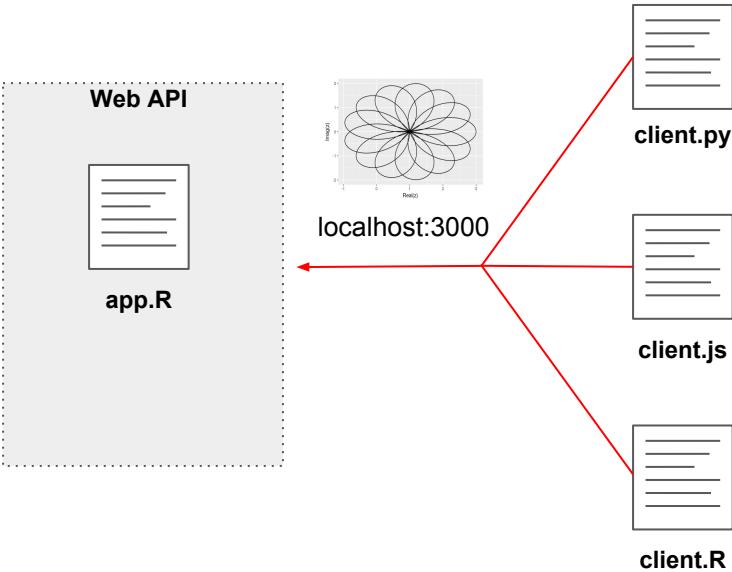
```
spiro <- function(n1=13,n2=-7,n3=-3) {  
  t <- seq(0,1,length.out=1000)  
  z <- exp(1i*2*pi*n1*t) +  
    exp(1i*2*pi*n2*t) +  
    exp(1i*2*pi*n3*t)  
  result <- tibble(x=Re(z),y=Im(z))  
  return(result)  
}  
  
## Spirograph with custom n1, n2, n3  
## @post /spiro/calc  
## @param n3 Characteristics for the third wheel  
## @param n2 Characteristics for the second wheel  
## @param n1 Characteristics for the first wheel  
## @json  
function(n1,n2,n3){  
  return(spiro(as.numeric(n1),  
               as.numeric(n2),  
               as.numeric(n3)))  
}  
  
## Spirograph plot  
## @get /spiro/plot  
## @param n3 Characteristics for the third wheel  
## @param n2 Characteristics for the second wheel  
## @param n1 Characteristics for the first wheel  
## @png  
function(n1,n2,n3){  
  result <- spiro(as.numeric(n1),  
                  as.numeric(n2),  
                  as.numeric(n3))  
  plot(result$x, result$y,  
        xlab="Real(z)",  
        ylab="Imag(z)")  
}
```





Building Web APIs with Plumber

```
spiro <- function(n1=13, n2=-7, n3=-3) {  
  t <- seq(0, 1, length.out=1000)  
  z <- exp(1i*2*pi*n1*t) +  
    exp(1i*2*pi*n2*t) +  
    exp(1i*2*pi*n3*t)  
  result <- tibble(x=Re(z), y=Im(z))  
  return(result)  
}  
  
## Spirograph with custom n1, n2, n3  
## @post /spiro/calc  
## @param n3 Characteristics for the third wheel  
## @param n2 Characteristics for the second wheel  
## @param n1 Characteristics for the first wheel  
## @json  
function(n1,n2,n3){  
  return(spiro(as.numeric(n1),  
               as.numeric(n2),  
               as.numeric(n3)))  
}  
  
## Spirograph plot  
## @get /spiro/plot  
## @param n3 Characteristics for the third wheel  
## @param n2 Characteristics for the second wheel  
## @param n1 Characteristics for the first wheel  
## @png  
function(n1,n2,n3){  
  result <- spiro(as.numeric(n1),  
                  as.numeric(n2),  
                  as.numeric(n3))  
  plot(result$x, result$y,  
        xlab="Real(z)",  
        ylab="Imag(z)")  
}
```



Try it: <https://beta.rstudioconnect.com/connect/#/apps/3533>





Getting Started with Plumber

The screenshot shows the plumber package documentation. At the top, there's a navigation bar with links for 'plumber', 'Demos', 'Examples', and 'See code on GitHub'. Below the navigation is a header with the word 'plumber' and a small cartoon character of a plumber holding a wrench. The main content area has a section titled 'Introduction' with a brief description: 'An R package that converts your existing R code to a web API using a handful of special one-line comments.' Below this is a code snippet from 'plumber.R':

```
# plumber.R
# This file contains the plumbing logic
# for the 'hello' endpoint
function() {
  message <- paste0("The message is: ", msg, "!")
}
# This function adds two numbers
# and returns the sum
function(x, y) {
  sum <- x + y
}
# Return the sum of two numbers
# and return the result in JSON
return(sum)
}
```

<https://www.rplumber.io>

Keep in mind...

1. APIs are good for sharing access to datasets you put together.
2. Watch out for long running processes.
3. Need an R process to run the server and background processes.



Publishing with R Markdown

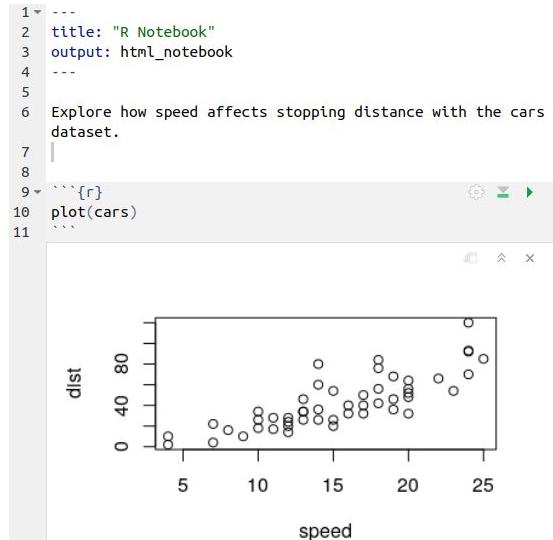
R + Markdown	# H1	H1
✓ Write reproducible content using the Markdown syntax.	## H2	H2
	### H3	H3
	Emphasis	<i>Emphasis</i>
	Bold	Bold
	1. Ordered	1. Ordered
	2. Lists	2. Lists
	3. Items	3. Items
	[inline links](https://sciencegateways.org)	inline links



Publishing with R Markdown

R + Markdown

- ✓ Write reproducible content using the Markdown syntax.
- ✓ Embed chunks of code between lines of narrative in literate programming style.

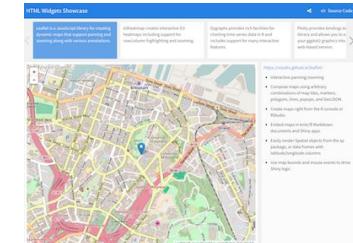
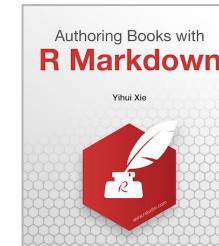
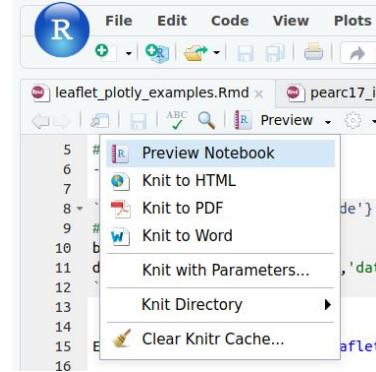




Publishing with R Markdown

R + Markdown

- ✓ Write reproducible content using the Markdown syntax.
- ✓ Embed chunks of code between lines of narrative in literate programming style.
- ✓ Output to multiple formats.





Getting Started with R Markdown

R Markdown documents are fully reproducible. Use a production notebook interface to weave code to produce multiple outputs.

<https://rmarkdown.rstudio.com>

flexdashboard: Easy interactive dashboards for R

Install the `flexdashboard` package from CRAN as follows:

```
install.packages("flexdashboard")
```

To author a `flexdashboard` you create an `.Rmd` document with the `flexdashboard:::flex_dashboard` output format. You can do this now with `flexDashboard` using the `new.RMarkdown` dialog:

<https://rmarkdown.rstudio.com/flexdashboard>

<https://bookdown.org>

htmlwidgets for R

Bring the best of JavaScript data visualization to R

Widgets in action

Interactive dashboards

https://rmarkdown.rstudio.com/developer_parameterized_reports.html

Learn More

Three ways to build reproducible, R based, content
that can be published on a science gateway.

Applications



<https://shiny.rstudio.com>

APIs



<https://www.rplumber.io>

Articles



<https://rmarkdown.rstudio.com>

Slides and Examples

<https://github.com/dskard/sgci-201805>