Datasets and Demos

Prof. Jacob M. Montgomery

Statistical Computing

March 22, 2016

```
squaresPack
├── DESCRIPTION
├── NAMESPACE
├── R
│   ├── addSquares.R
│   ├── subtractSquares.R
│   ├── exampleDataset.R
│   └── squaresPack-package.R
├── man
│   ├── squaresPack.Rd
│   ├── addSquares.Rd
│   ├── subtractSquares.Rd
│   └── exampleDataset.Rd
├── data
│   └── exampleDataset.rda
└── demo
    ├── 00Index
    ├── addSquares.R
    └── subtractSquares.R
```

Datasets

- Make a dataset of some kind
- `save(obj, file=''DataFrame.rda'')`
- Save it/move it into the data subdirectory

Datasets

- Make a dataset of some kind
- `save(obj, file=''DataFrame.rda'')`
- Save it/move it into the data subdirectory

Demos: 00Index file

```
addSquares          Demo file for addSquares
subtractSquares     Demo file for subtractSquares
```

Demos: 00Index file

```
addSquares          Demo file for addSquares
subtractSquares     Demo file for subtractSquares
```

Demos: R file

The following R script will be run when the user types `demo(addSquares)`. This script is stored in the `demo` directory as `addSquares.R`

```
dx <- 2
dy <- 3:4
addSquares(dx, dy)
```

A new checklist

1. Edit R code and/or data files
2. Run `as.package()`, `load_all()`, and `document()`
3. Check the code: `check(current.code)`
4. Make a Windows build: `build_win(current.code)`
5. Double-check the `DESCRIPTION` file
6. Submit the package to CRAN:
   `release(current.code, check=FALSE)`

In class activity

- Last time you added new functionality to the S4 package
- Now, add in a new dataset.
- Add a demo for your function that:
  - calls in your new dataset
  - uses your new function

Midterm

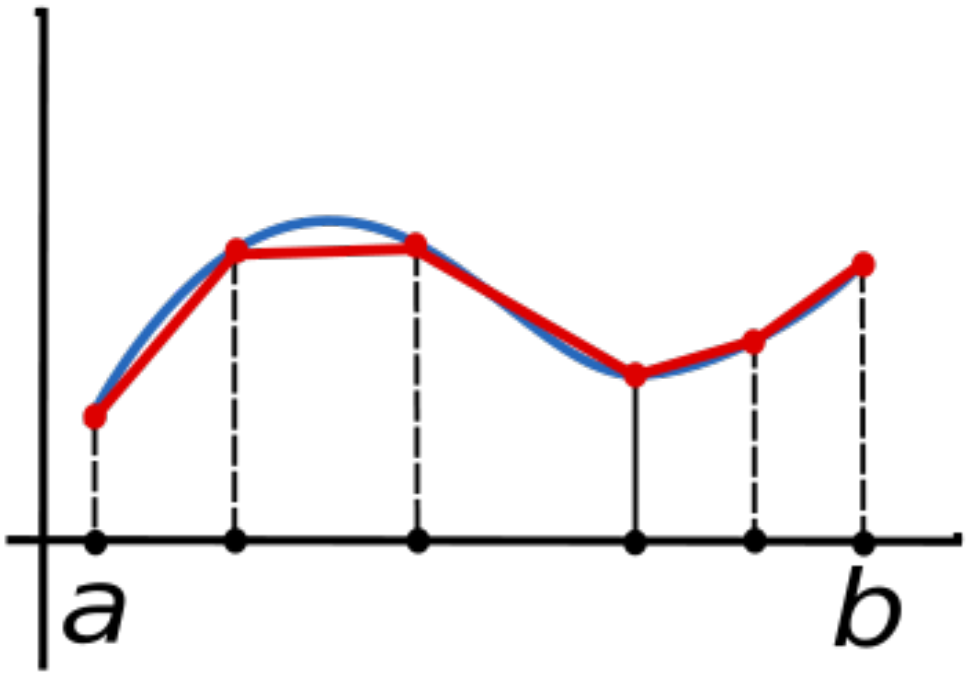
- S4 Package named: `integrateIt`
- Two classes: `Trapezoid`, `Simpson`
- One generic: `integrateIt`
- Three methods: `integrateIt`, `plot`, `print`
- Graduate students will make an extra generic/method: `tolTest`
- A development file for putting it together using `devtools`

Background: Trapezoidal rule

$$\int_a^b f(x)dx \approx T$$

$$T = \frac{h}{2}\left(f(x_0) + 2f(x_1) + \ldots + 2(f_{n-1}) + f(x_n)\right)$$

where $h = \frac{b-a}{n}$.



Background: Simpsons rule

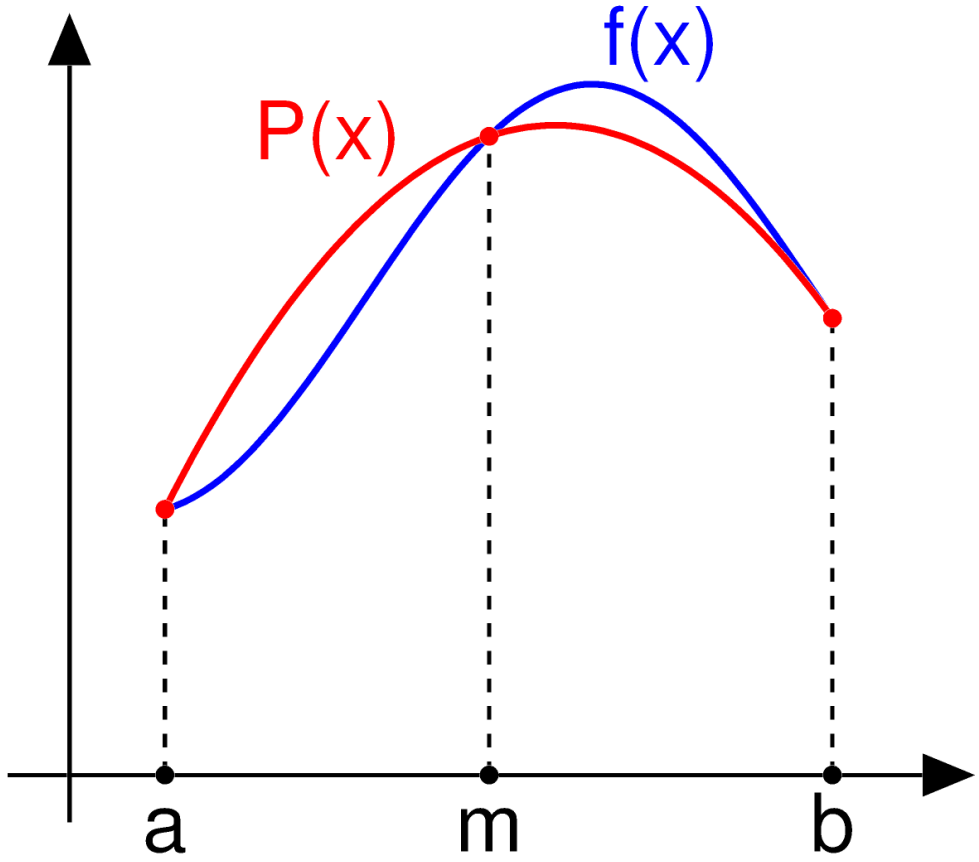
$$\int_a^b f(x)dx \approx S$$

$$S = \frac{h}{3}\left(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) \ldots + +4f(x_{n-1}) + f(x_n)\right)$$

where $h = \frac{b-a}{n}$.

Background: Simpsons rule

We are gong to draw a parabola from the points $(a, f(a))$ to $(b, f(b))$ that also goes through the $(m, f(m))$. We approximate the area under that parabola as being equal to $\int_a^b f(x)dx$.



Background: Simpsons rule

The parabola between any two points, $(u, f(u))$ and $(w, f(w))$ is drawn according to the formula (let $v = \frac{w-u}{2}$):

$$p(x) = f(u)\frac{(x-v)(x-w)}{(u-v)(u-w)} + f(v)\frac{(x-u)(x-w)}{(v-u)(v-w)} + f(w)\frac{(x-u)(x-v)}{(w-u)(w-v)}$$

The integral under that parabola is:

$$\int_u^w p(x)dx = \frac{h}{3}\big(f(u) + 4(f(v)) + f(w)\big)$$

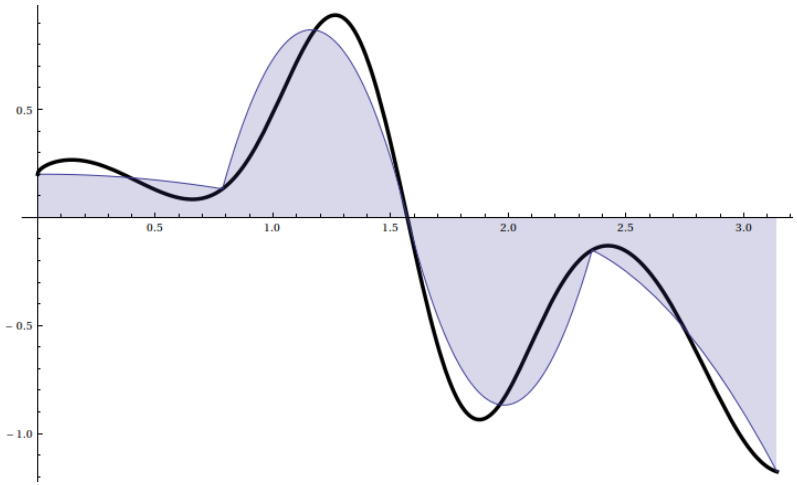
Background: Simpsons rule

If we imagine carrying on that calculation many times between different points along the curve, we get:

$$\int_a^b f(x)dx \approx S$$

$$S = \frac{h}{3}\left(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) \ldots + 4f(x_{n-1}) + f(x_n)\right)$$

where $h = \frac{b-a}{n}$.



NOTE: This requires that there be an even number of points being evaluated (HINT, HINT)

`integrateIt` method

Input:

- A vector of values (x)
- A vector of evaluated values ($f(x) = y$)
- Starting/ending values (a,b)
- A "Rule" argument that can be either "Trap" or "Simpson"

Output:

- An object of class `Trapezoid` or class `Simpson`
- Should contain the values of x and y.
- Should also include the result.
- Both classes should have validation methods that include a few appropriate tests.
- Note: you will need to create an `initialize` function for each class, which will be used internally by `integrateIt`

`print` method

- A very simple print method for each class, which prints out just the integrated value (rather than all of the results)
- Extra credit if you accomplish this using a subclass scheme rather than writing separate functions for each class.

`plot` methods

- A plot method for the `Trapezoid` class that will draw the trapezoids used to approximate the curve. (HINT: `segments()`, and/or `polygon()`)
- A plot method for the Simpson class that will draw the parabolas used to approximate the curve. (HINT: `segments()` and/or `lines()`. `curve()` could work, but will take some thought.)
- In both cases, the plots should have good defaults for things like plot limits, labels, title, etc. Feel free to add in additional arguments if that is helpful.

`tolTest` method

For graduate students only.

The `tolTest` method should take in

- A `fun` (function)
- A `tolerance` argument.
- A `rule` argument that indicates whether the Trapezoidal or Simpson's
- A `start` argument for the number of intervals it should start with
- A `correct` argument that provides the correct answer for the integral

The `tolTest` operations

- It should take in a function and increase the number of intervals n until the answer it provides using the specified approximation is within `tolerance` of the correct answer. (HINT: `integrate()`)

The `tolTest` output

- All of the inputs.
- The final n
- The absolute error of the estimate

General rules

- You have until the end of class today until the beginning of the next class. I expect everyone to be done within 4 hours. (Exceptions)

General rules

- You have until the end of class today until the beginning of the next class. I expect everyone to be done within 4 hours. (Exceptions)
- You should spend *no more* than six hours total on the project, but those six hours can come in any time window in that periods.

General rules

- You have until the end of class today until the beginning of the next class. I expect everyone to be done within 4 hours. (Exceptions)
- You should spend *no more* than six hours total on the project, but those six hours can come in any time window in that periods.
- We will begin the time from the moment you start your GitHub repository.

General rules

- You have until the end of class today until the beginning of the next class. I expect everyone to be done within 4 hours. (Exceptions)
- You should spend *no more* than six hours total on the project, but those six hours can come in any time window in that periods.
- We will begin the time from the moment you start your GitHub repository.
- You will be graded based on:

General rules

- You have until the end of class today until the beginning of the next class. I expect everyone to be done within 4 hours. (Exceptions)
- You should spend *no more* than six hours total on the project, but those six hours can come in any time window in that periods.
- We will begin the time from the moment you start your GitHub repository.
- You will be graded based on:
  - Comments (!)

General rules

- You have until the end of class today until the beginning of the next class. I expect everyone to be done within 4 hours. (Exceptions)
- You should spend *no more* than six hours total on the project, but those six hours can come in any time window in that periods.
- We will begin the time from the moment you start your GitHub repository.
- You will be graded based on:
  - Comments (!)
  - Correct/frequent use of GitHub (commits are good. Syncing I care nothinga about).

General rules

- You have until the end of class today until the beginning of the next class. I expect everyone to be done within 4 hours. (Exceptions)
- You should spend *no more* than six hours total on the project, but those six hours can come in any time window in that periods.
- We will begin the time from the moment you start your GitHub repository.
- You will be graded based on:
  - Comments (!)
  - Correct/frequent use of GitHub (commits are good. Syncing I care nothinga about).
  - Elegance of code (apply rather than for loops, etc.)

General rules

- You have until the end of class today until the beginning of the next class. I expect everyone to be done within 4 hours. (Exceptions)
- You should spend *no more* than six hours total on the project, but those six hours can come in any time window in that periods.
- We will begin the time from the moment you start your GitHub repository.
- You will be graded based on:
  - Comments (!)
  - Correct/frequent use of GitHub (commits are good. Syncing I care nothinga about).
  - Elegance of code (apply rather than for loops, etc.)
  - Readability of code/stability of naming conventions

General rules

- You have until the end of class today until the beginning of the next class. I expect everyone to be done within 4 hours. (Exceptions)
- You should spend *no more* than six hours total on the project, but those six hours can come in any time window in that periods.
- We will begin the time from the moment you start your GitHub repository.
- You will be graded based on:
  - Comments (!)
  - Correct/frequent use of GitHub (commits are good. Syncing I care nothinga about).
  - Elegance of code (apply rather than for loops, etc.)
  - Readability of code/stability of naming conventions
  - Documentation/full completion of the package structure.

General rules

- You have until the end of class today until the beginning of the next class. I expect everyone to be done within 4 hours. (Exceptions)
- You should spend *no more* than six hours total on the project, but those six hours can come in any time window in that periods.
- We will begin the time from the moment you start your GitHub repository.
- You will be graded based on:
  - Comments (!)
  - Correct/frequent use of GitHub (commits are good. Syncing I care nothinga about).
  - Elegance of code (apply rather than for loops, etc.)
  - Readability of code/stability of naming conventions
  - Documentation/full completion of the package structure.
- You do not need a demo *per se*, but a development file should show how to use your functions with maybe an example or two.

General rules

- You have until the end of class today until the beginning of the next class. I expect everyone to be done within 4 hours. (Exceptions)
- You should spend *no more* than six hours total on the project, but those six hours can come in any time window in that periods.
- We will begin the time from the moment you start your GitHub repository.
- You will be graded based on:
  - Comments (!)
  - Correct/frequent use of GitHub (commits are good. Syncing I care nothinga about).
  - Elegance of code (apply rather than for loops, etc.)
  - Readability of code/stability of naming conventions
  - Documentation/full completion of the package structure.
- You do not need a demo *per se*, but a development file should show how to use your functions with maybe an example or two.
- Email the Homola with your gitHub repository. Include a start/end time in that email.