559

# Differentiation and integration in R

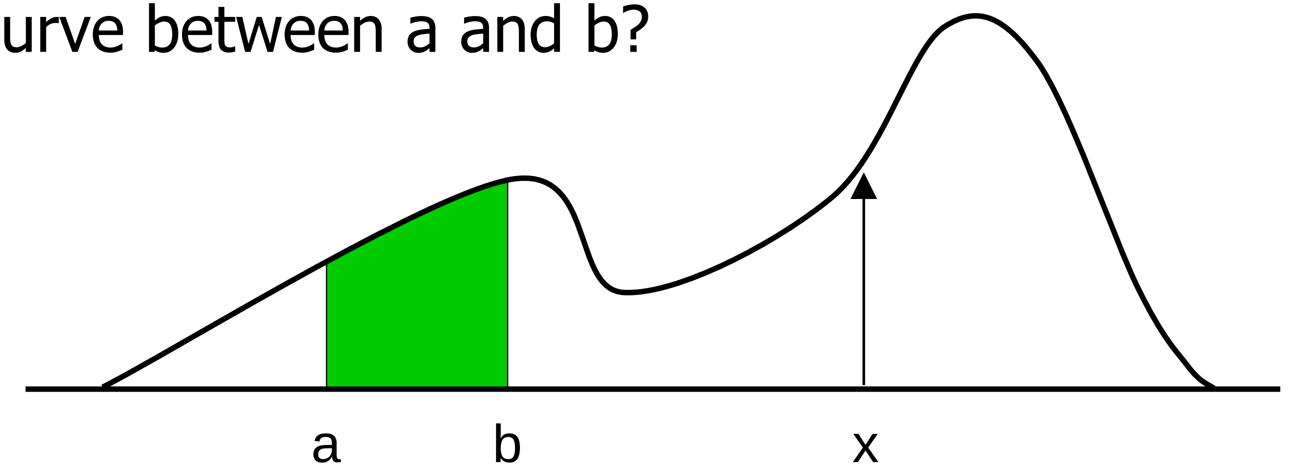
Fish 559; Lecture 10

# Outline

- Calculus
  - Symbolic
  - Algorithmic
  - Numeric
- Differentiation
- Integration

# Calculus

- Differentiation
  - How steep is the curve at point x?
- Integration
  - How much area is under the curve between a and b?



# Kinds of answers

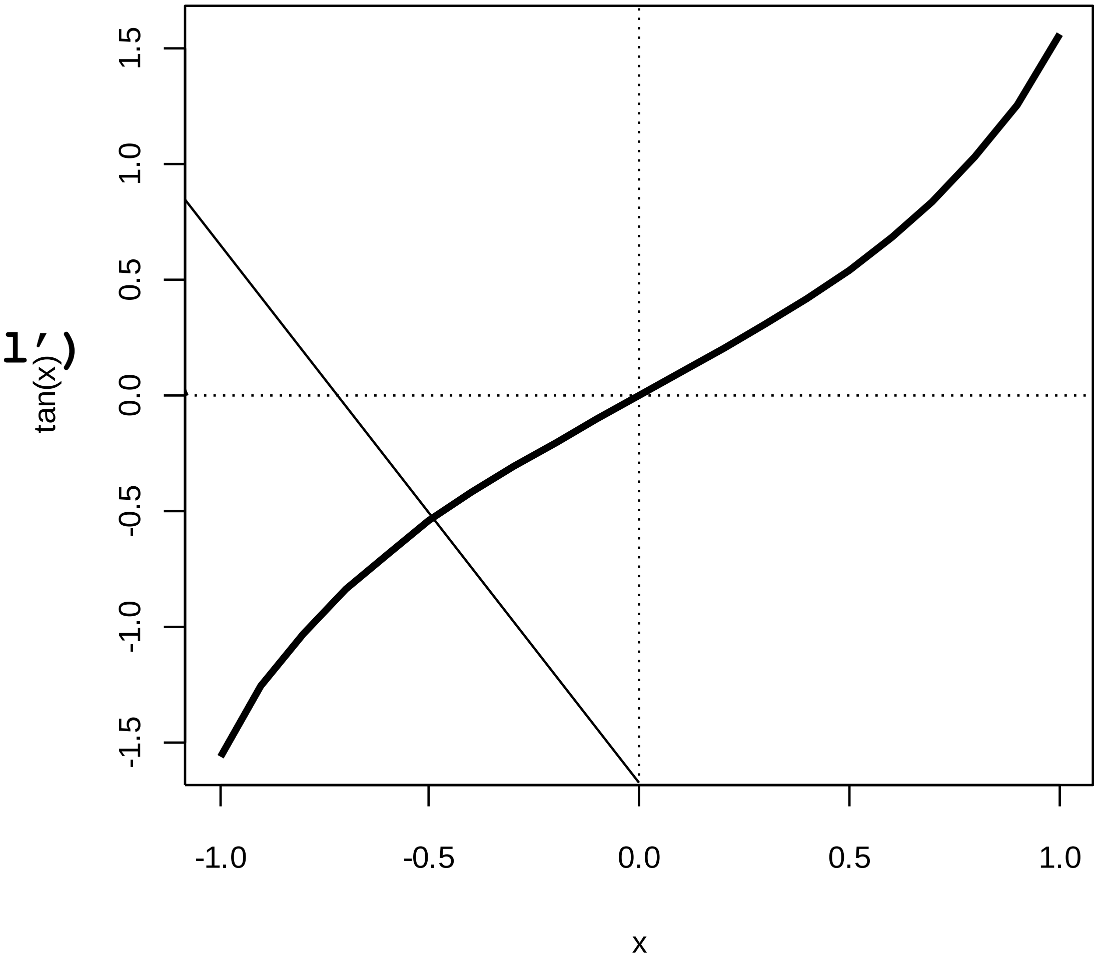
- Symbolic e.g.
  - deriv() function from Lec9
  - Formula/equation, like sqrt(x+3x^2)
- Algorithmic/analytic
  - Value, like 12.41
  - Exact answer from formula
- Numeric
  - Value, like 12.37
  - Approximate answer from numerical methods

# Differentiation

```
> x <- seq(-1, 1, 0.1)

> plot(x, tan(x),type='l')

> cbind(x, tan(x))
```



# Symbolic

- We pass the formula as an expression
- Function that returns another function
  - First derivative, tan'(x)

```
> D(expression(tan(x)), "x")
1/cos(x)^2
```

  - Second derivative, tan''(x)

```
> D(D(expression(tan(x)),"x"), "x")
2 * (sin(x) * cos(x))/(cos(x)^2)^2
NOT ELEGANT, CAN BE SIMPLIFIED
```

# Algorithmic/Analytic: Define our own

- One approach is to define our own function, knowing that tan'(x) = 1/cos(x)^2
- Returns value, not another function

```
> dtanx.dx <- function(x)
  {
    gradient <- 1/cos(x)^2
    return(gradient)
  }
> cbind(x, dtanx.dx(x))
```

# Algorithmic: Single variable

- Another approach is to define the derivative

```
> dtanx.dx <- D(expression(tan(x)), "x")
```

  - And evaluate it

```
> eval(dtanx.dx)
```

  - The deriv function gives more information

```
> dtanx.dx <- deriv(expression(tan(x)),"x")
> eval(dtanx.dx)
> attr(eval(dtanx.dx),"gradient")
```

# Algorithmic: Multivariable

- Partial differentiation, with optional Hessian

```
> dtanxy.dxy <- deriv(expression(tan(x*y)),
  c("x","y"), hessian=TRUE)
```

- Evaluate with a simple y vector

```
> y <- seq(-1,1,0.1)
> eval(dtanxy.dxy)
```

- Read the help and any book on calculus

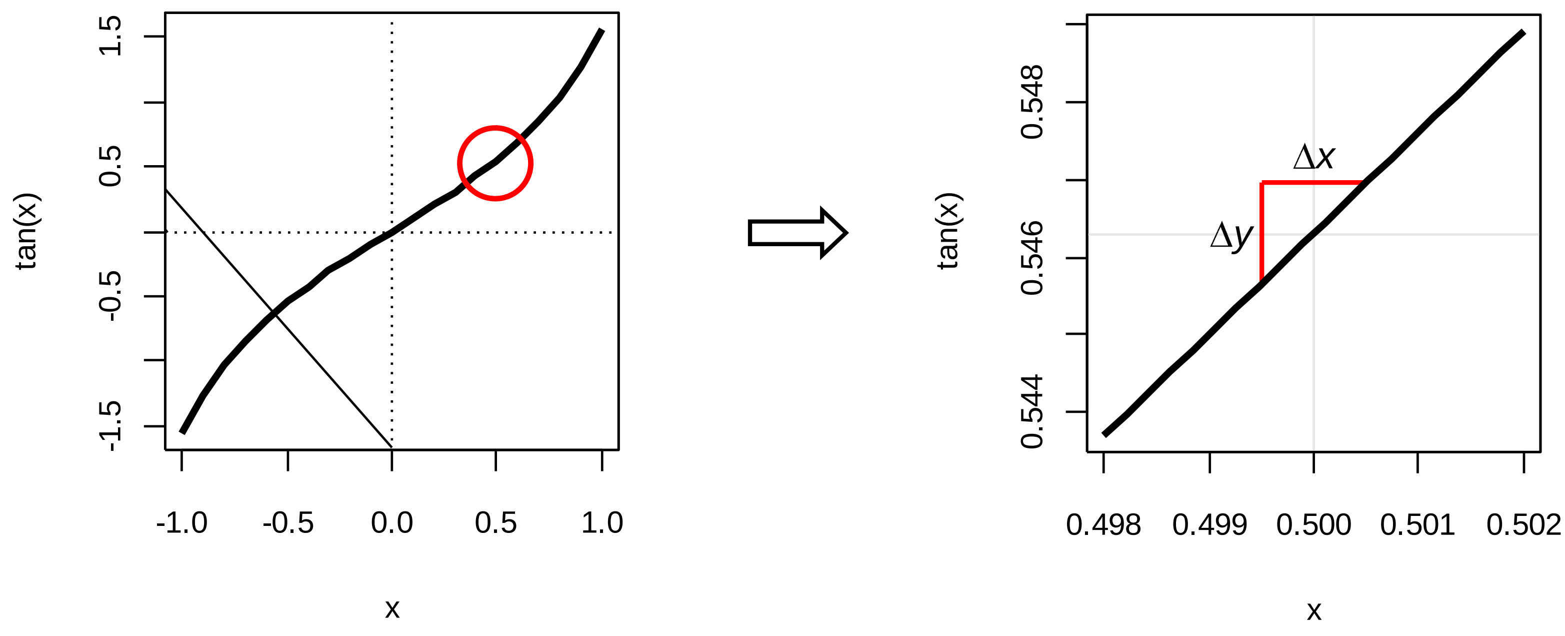
```
> ?deriv
```

# Differentiation

507

Numeric – exact if linear fxn, but if there is error if fxn is  nonlinear:
-if delta is too big, error do to approximation
- if your delta is too small, error do to rounding

$$f'(x) \approx \frac{\Delta y}{\Delta x} \quad \text{when } \Delta x \text{ is small}$$



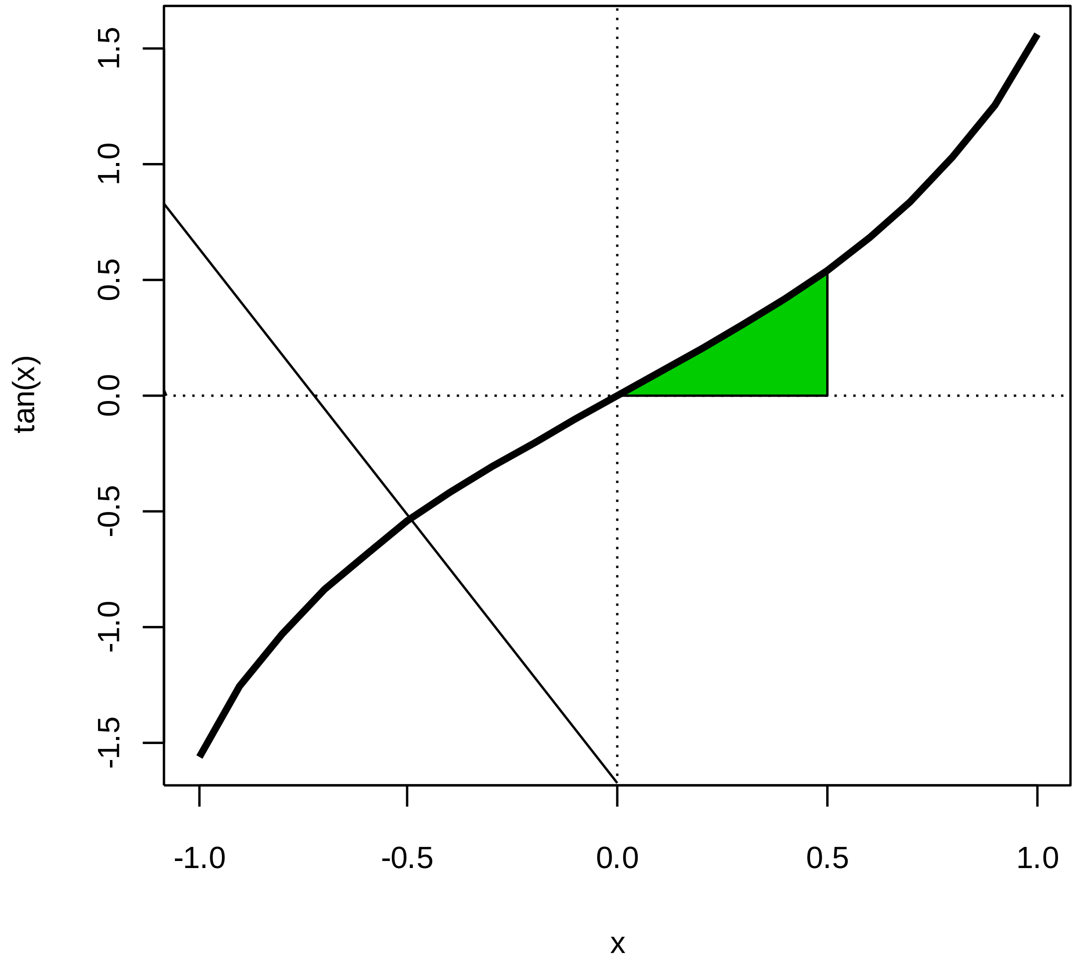
# Numeric – central difference approach

- Approximate differentiation may be needed for complicated problems

```
> dfx.dx <- function(f, x, delta)
  {
    y1 <- f(x - delta/2)
    y2 <- f(x + delta/2)
    approx.gradient <- (y2-y1) / delta
    return(approx.gradient)
  }
> dfx.dx(tan, x, 0.001)
```

# Integration

- Symbolic
  - Anti-derivative
    = indefinite integral
- Algorithmic/Numeric
  - Area between a and b
    = definite integral



# Algorithmic

```
> indef.tanx.dx <- function(x)
  {
    answer <- -log(cos(x))
    return(answer)
  }
```

# R integrators

- **`integrate()`**
  - Single variable adaptive algorithm by R. Piessens et al.
  - Package: base
- **`area()`**
  - Single variable trapezium bisection with Simpson's rule
  - Package: MASS
- **`adaptIntegrate()`**
  - Multivariable
  - Package: subature

# Single variable integration

```
> integrate(tan, 0, 0.5)
0.1305842 with absolute error < 1.4e-15
```

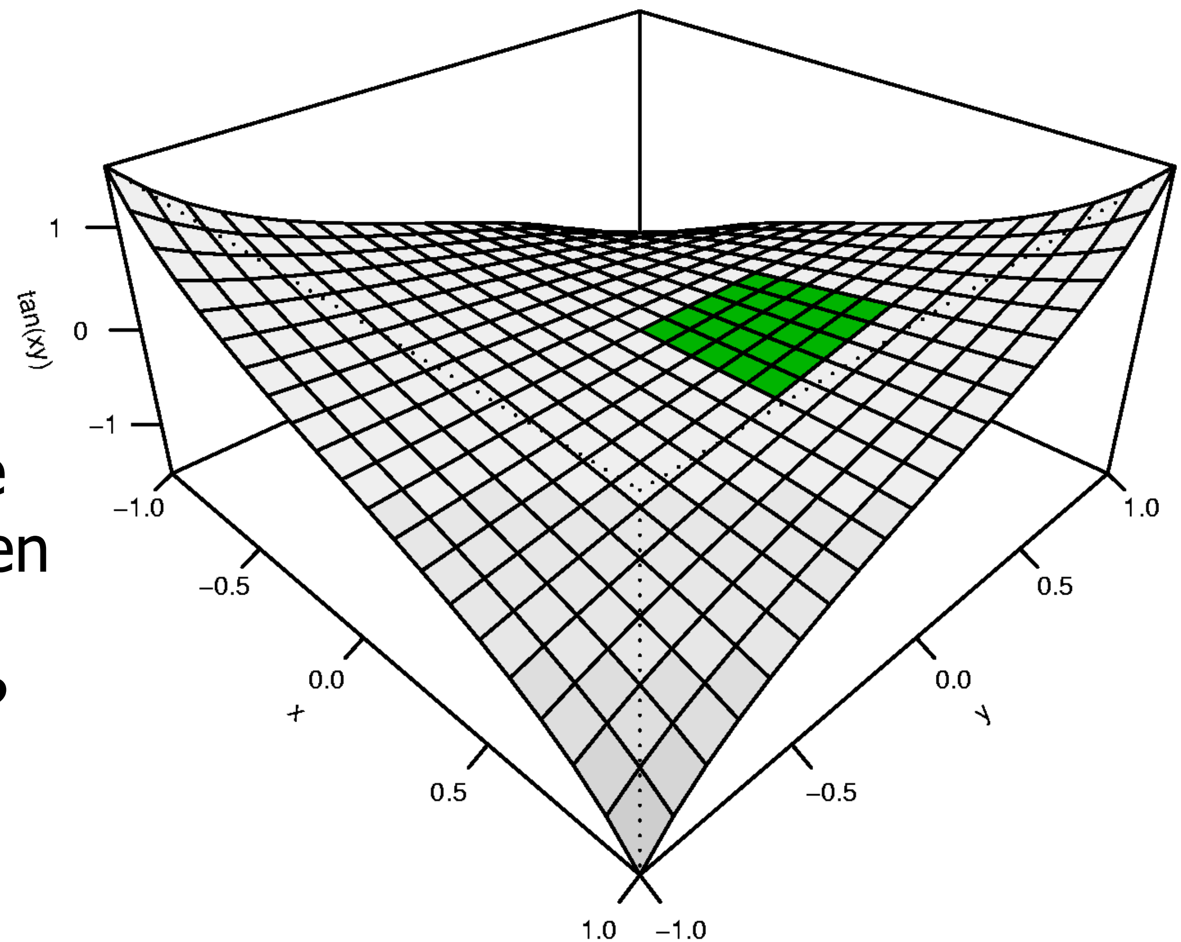
```
> library(MASS)
> area(tan, 0, 0.5)
0.1305842
```

507

# Multivariable integration

$$\int_0^{\frac{1}{2}}\int_0^{\frac{1}{2}} \tan(xy)\,dx\,dy$$

- What is the volume sandwiched between the green surface and the z=0 plane?



# Multivariable integration

```
> tan.xy <- function(xy)
  {
    answer <- tan(xy[1] * xy[2])
    return(answer)
  }

> library(cubature)
> adaptIntegrate(tan.xy, lowerLimit=c(0,0), u
  pperLimit=c(0.5,0.5))

$integral: 0.0157073
```

**Have to integrate over a grid. If this is a bagel we have to look at it in a different dimensions so that it's in x/y coordinates(ie 0 to radius, 0 to 2pi, etc)**

# Summary

| | Differentiation | Integration |
|---|---|---|
| Symbolic | **`D`** | **(2)** |
| Algorithmic | **`D, deriv`** | **(3)** |
| Numeric | **(1)** | **`integrate`**<br>**`area, adapt`** |

**(1)** we wrote **`dfx.dx(f,x,delta)`** to do this
**(2)** look up in tables or integrals.wolfram.com
**(3)** we wrote **`def.tanx.dx(a,b)`** to do this for **`tan(x)`**