Assignment

- Chapter 3: 37 pages
  - My lectures are going much faster than the reading assignments
  - You will need Ch 3 for Assignment 2
- Start Chapter 4
  - Shorter, but probably more confusing

Clickers

- If any of these are yours, write down your name and the clicker ID on a sheet of paper:
- #01C18B4B
- #0CB07CC0
- #18C2B16B
- #19CB6FBD
- #25D2F700

Homework

How is the homework going?

A) I’ve turned it in

B) I’m nearly done

C) I’ve barely started

D) I haven’t started

E) None of the above (you should probably tell me what you mean)

Homework

Was the homework...

A) Too short

B) Pretty much OK

C) Too long

D) I haven’t done enough to know for sure

E) None of the above

Class so far

Anonymous

Are the tutorials useful?

A) Yes, they’re the best part of class

B) Yes, but lectures are more useful

C) No, but lectures are useful

D) No, and the lectures aren’t very useful either

E) None of the above

A little more about data types

- You can “cast” a variable or number to a different type

```
IDL> help,1S,fix(1)
<Expression>    INT       =        1
<Expression>    INT       =        1
IDL> help,1L,long(1)
<Expression>    LONG      =                1
<Expression>    LONG      =                1
IDL> help,1S,fix(1)
<Expression>    INT       =        1
<Expression>    INT       =        1
IDL> help,1B,byte(1)
<Expression>    BYTE      =     1
<Expression>    BYTE      =     1
```

This is a silly name. It exists for historical reasons, or at least I assume it does.

Rounding #’s

```
IDL> help,round(5.5),floor(5.5),ceil(5.5)
<Expression>    LONG      =                6
<Expression>    LONG      =                5
<Expression>    LONG      =                6
```

Details of `size`

- Returns: `[# of dimensions, size of each dimension, type code, number of elements]`

```
IDL> print,size(1)
           0           2           1
```

0 dimensions: scalar (no size of dimension)

Type Code 2: integer

Number of elements: 1

Details of `size`

- Returns: `[# of dimensions, size of each dimension, type code, number of elements]`

```
IDL> print,size([1])
           1           1           2           1
```

1 dimension | Size of dimension 1: 1 | Type Code 2: integer | Number of elements: 1

Size

What will `size([1,2,3,4])` return?

A)`[0, 2, 1]`

B)`[1, 4, 2, 1]`

C)`[1, 4, 2, 4]`

D)`[4, 4, 2, 4]`

E) None of the above / I don’t know

Size

What will `size([1.,2.,3.])` return?

A)`[0, 2, 3]`

B)`[1, 3, 2, 1]`

C)`[1, 3, 2, 3]`

D)`[3, 3, 2, 3]`

E) None of the above / I don’t know

shifting

```
IDL> print,indgen(5)
       0       1       2       3       4
IDL> print,shift(indgen(5),2)
       3       4       0       1       2
IDL> print,shift(indgen(5),3)
       2       3       4       0       1
```

Clipping

- If you want, e.g., all positive numbers in an array, do "`arr > 0`" (sets negative numbers = 0)

```
IDL> x = randomn(seed,5)
IDL> print,x
     -2.09936    -0.983745     0.437967    -0.211478     0.360088
IDL> print,x>0
      0.00000      0.00000     0.437967      0.00000     0.360088
```

Array Scaling

- What do you do if you want an array going from -2 to 2 with 10 elements?
  - first, `findgen(10)` gets you 10 elements from 0-9
  - second, divide by 9 to get your array going from 0 to 1
  - third, multiply by the range `end-start = 2-(-2) = 4`
  - finally, subtract 2

Array Scaling

- What do you do if you want an array going from -2 to 2 with 10 elements?

```
IDL> print,findgen(10)/9*4-2
     -2.00000     -1.55556     -1.11111    -0.666667    -0.222222
     0.222222     0.666667      1.11111      1.55556      2.00000
IDL> print,-2 + (4./9.) * findgen(10)
     -2.00000     -1.55556     -1.11111    -0.666667    -0.222222
     0.222222     0.666667      1.11111      1.55556      2.00000
IDL> xmin=-2
IDL> xmax=2
IDL> xrange=xmax-xmin
IDL> npoints=10
IDL> print,xmin+(float(xrange)/(npoints-1))*findgen(npoints)
     -2.00000     -1.55556     -1.11111    -0.666667    -0.222222
     0.222222     0.666667      1.11111      1.55556      2.00000
```

Array Scaling

- What do you do if you want an array going from -2 to 2 with 10 elements?

Probably the best way:

```
IDL> zero_to_one = findgen(npoints)/(npoints-1)
IDL> xmin=-2
IDL> xmax=2
IDL> xrange=xmax-xmin
IDL> print,xmin+xrange*zero_to_one
      -2.00000      -1.55556      -1.11111     -0.666667     -0.222222
      0.222222      0.666667       1.11111       1.55556       2.00000
```

Don’t do this:

```
 IDL> print,-2+0.4444444444*findgen(10)
      -2.00000      -1.55556      -1.11111     -0.666667     -0.222222
      0.222222      0.666667       1.11111       1.55556       2.00000
```

Array Scaling

- What do you do if you want an array going from -2 to 2 with 10 elements?

python and matlab have the convenience function

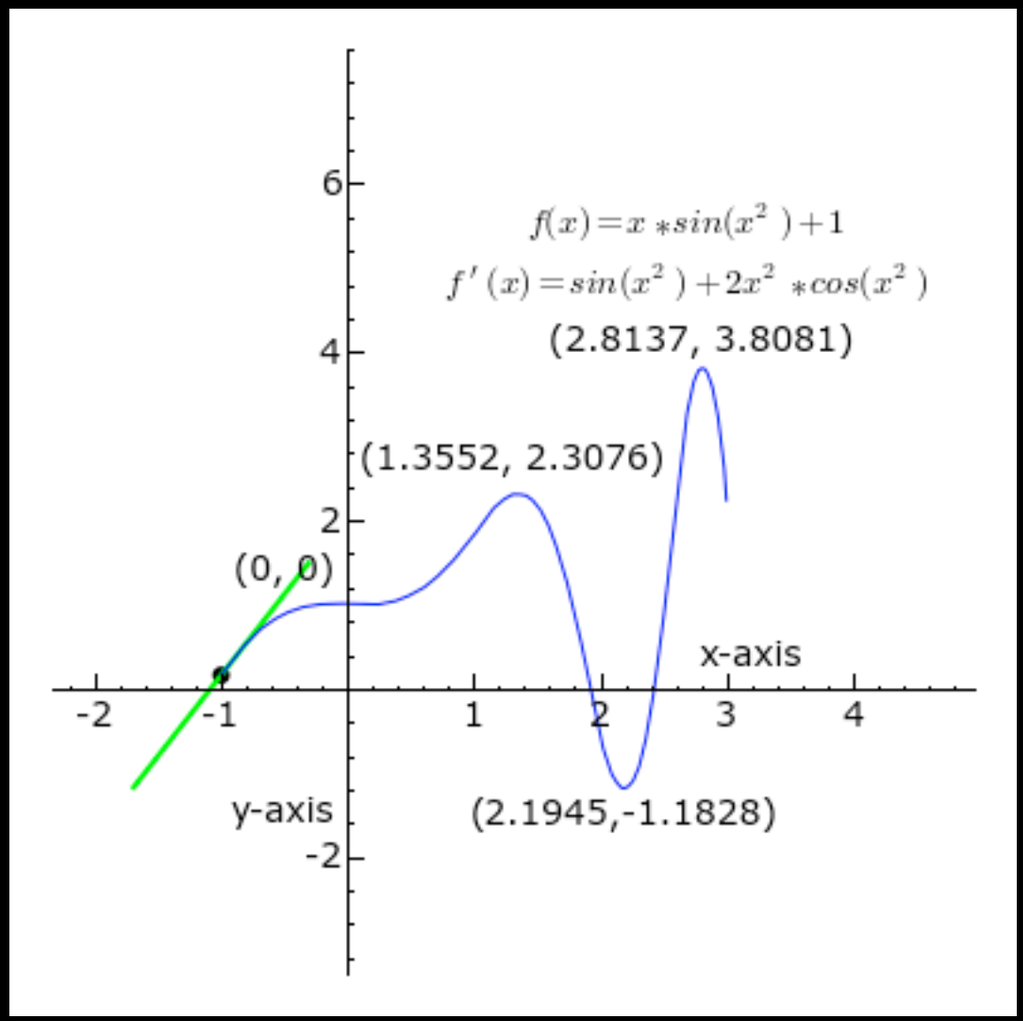
“`linspace`”:

```
linspace(-2,2,10)
```

We’ll make one in IDL soon

Mathematical Applications

- Finite-Difference Method for derivatives
- Derivatives represent the local slope



Finite Difference

- For this discussion, we have an `x` array and a `y` array, where `y=f(x)`, but we don't need to know anything else about `y`.
- But we'll use `y=sin(x)`

Calc Review

What is $\frac{d}{dx}\sin(x)$?

A)arcsin(x)

B)cos(x)

C)-cos(x)

D)cosecant(x)

E) None of the above / I don't know

Setup...

```
IDL> xmin=0
IDL> xmax=!pi/2.
IDL> xrange=xmax-xmin
IDL> npoints=5
IDL> zero_to_one = findgen(npoints)/(npoints-1)
IDL> x = xmin+xrange*zero_to_one
IDL> y = sin(x)
```

Finite Difference

- We want to know what the derivative, `dy/dx`, is
- We can use the `shift` function to determine `dx` and `dy`

```
IDL> dx = x - shift(x,1)
IDL> dy = y - shift(y,1)
IDL> print,dx
      -1.57080     0.392699     0.392699     0.392699     0.392699
IDL> print,dy
      -1.00000     0.786520     0.130484    0.0633972    0.0195988
```

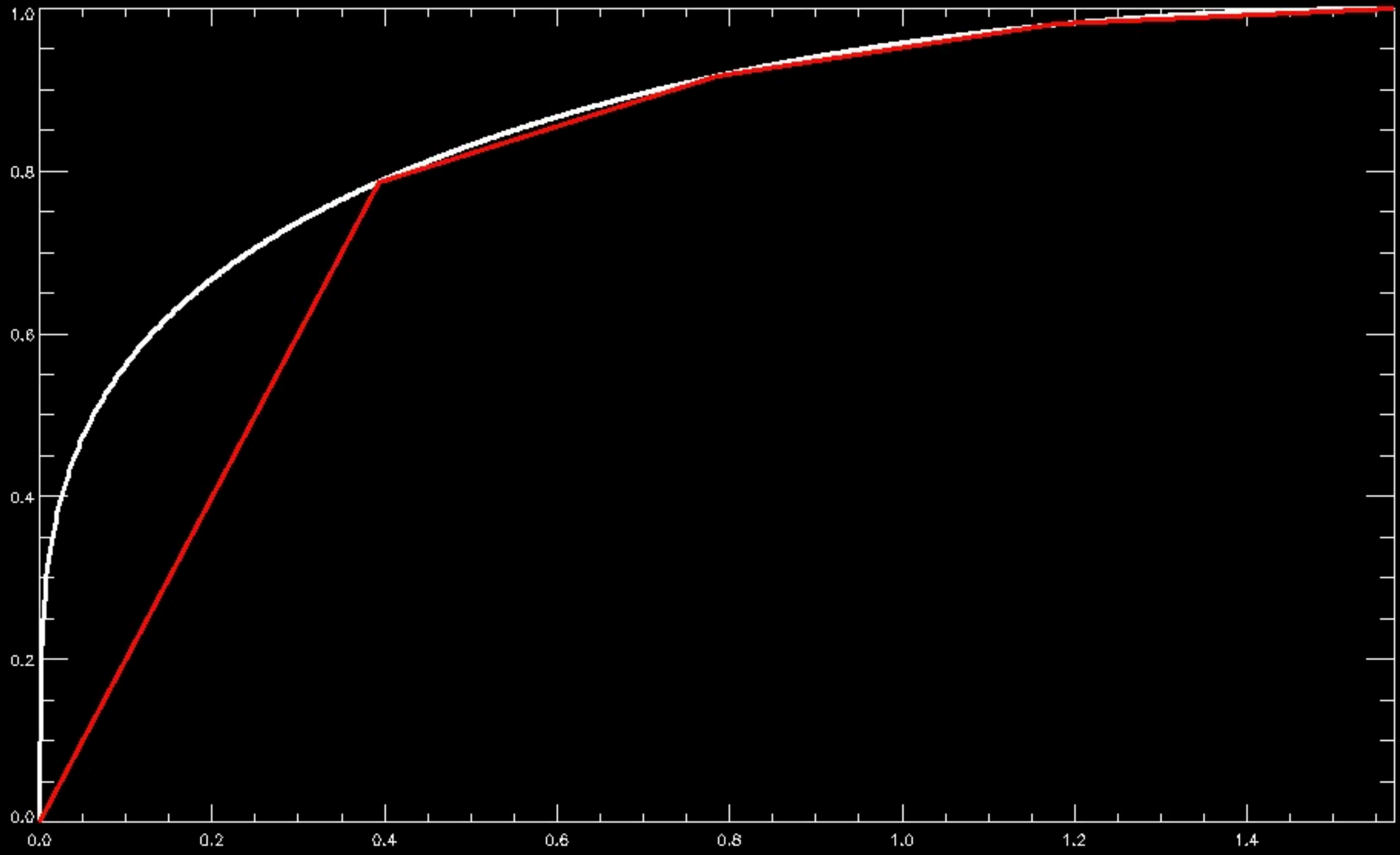
Finite Difference

- We want to know what the derivative, `dy/dx`, is
- We can use the `shift` function to determine `dx` and `dy`

But this is wrong! We can only have 4 delta-x and delta-y measurements for 5 points

```
IDL> dx = x - shift(x,1)
IDL> dy = y - shift(y,1)
IDL> print,dx
      -1.57080      0.392699      0.392699      0.392699      0.392699
IDL> print,dy
      -1.00000      0.786520      0.130484     0.0633972     0.0195988
```

Why’s it wrong?




```
IDL> xhi = xmin+xrange*findgen(1000)/999
IDL> yhi = sin(xhi)^0.25
IDL> plot,xhi,yhi,/xs,thick=3
IDL> oplot,x,y,color='0000FF'x,thick=3
```

Finite Difference

- Can also use the “backwards difference” method
  - same problem

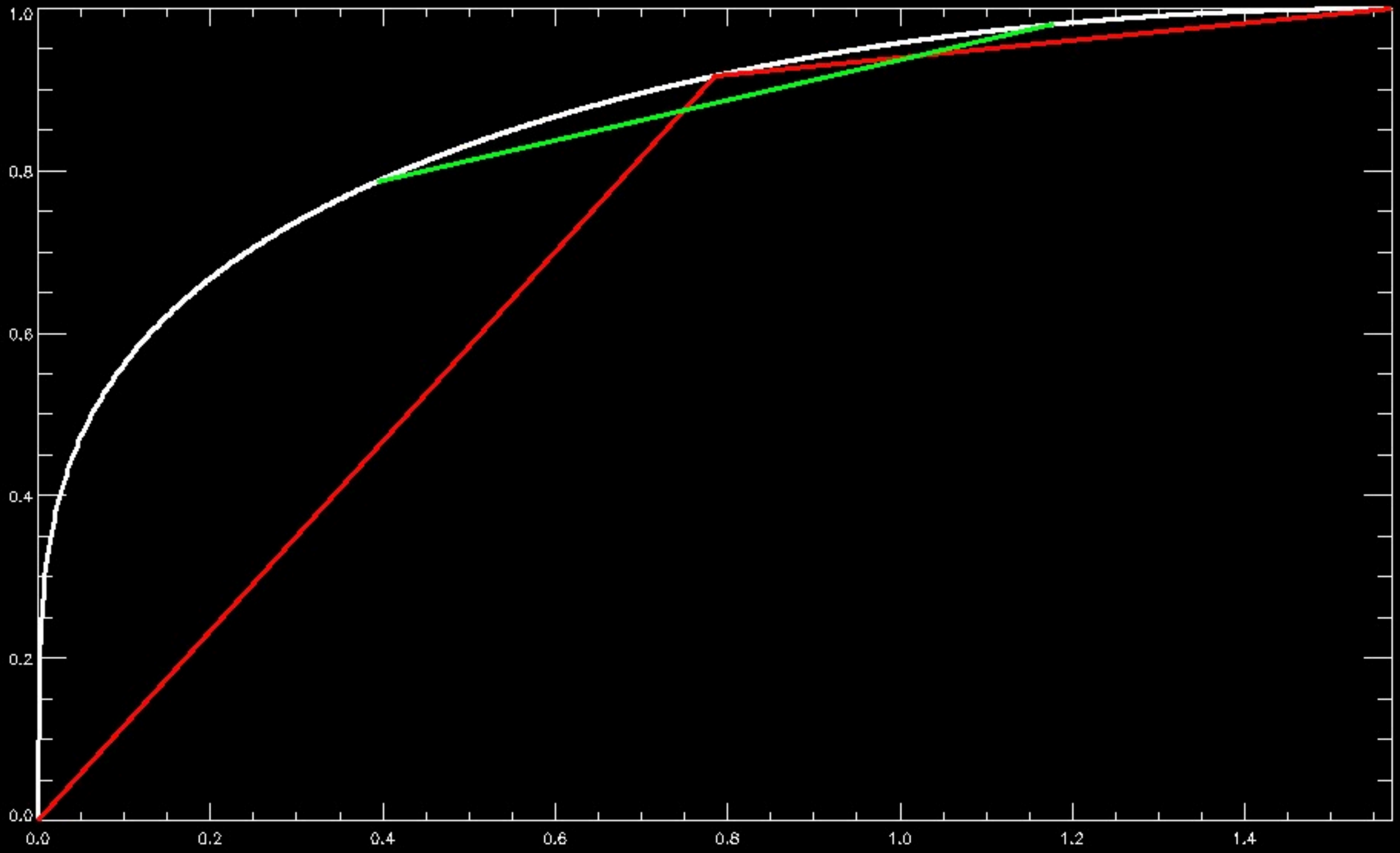
```
IDL> dx = x - shift(x,-1)
IDL> dy = y - shift(y,-1)
IDL> print,dx
    -0.392699    -0.392699    -0.392699    -0.392699      1.57080
IDL> print,dy
    -0.786520    -0.130484   -0.0633972   -0.0195988      1.00000
```

Finite Difference

- Instead, the central difference is better
- but you lose the end points

```
IDL> d2x = shift(x,1) - shift(x,-1)
IDL> d2y = shift(y,1) - shift(y,-1)
IDL> print,d2x
      1.17810    -0.785398    -0.785398    -0.785398      1.17810
IDL> print,d2y
     0.213480    -0.917004    -0.193881   -0.0829960     0.980401
```

Central Difference



Central Difference

- Use the central difference to get all points where it’s valid
- Then use forward-difference for the last and backward-difference for the first
  - this actually doesn’t do a great job unless you have lots of points

The code (for reference)

```
IDL> dxf = x - shift(x,1)
IDL> dyf = y - shift(y,1)
IDL>
IDL> dxb = x - shift(x,-1)
IDL> dyb = y - shift(y,-1)
IDL>
IDL> d2x = shift(x,1) - shift(x,-1)
IDL> d2y = shift(y,1) - shift(y,-1)
IDL> print,d2x
      1.17810    -0.785398    -0.785398    -0.785398     1.17810
IDL> print,d2y
     0.213480    -0.917004    -0.193881   -0.0829960     0.980401
IDL>
IDL> dx = d2x
IDL> dx[0] = dxb[0]
IDL> dx[-1] = dxf[-1]
IDL> dy = d2y
IDL> dy[0] = dyb[0]
IDL> dy[-1] = dyf[-1]
IDL> print,dy/dx
      2.00286      1.16757     0.246857     0.105674    0.0499079
```

Second Derivative

- A good approximation for the second derivative is:

$(d^2y/d^2x)_i = (y[i-1] - 2*y[i] + y[i+1])/dx^2$

```
d2x/d2y = (shift(y,1)-2*y+shift(y,-1))/dx^2
```

The text has some more detail

Derivatives

What is the derivative of x^2 evaluated at x=3?

A) 6

B) 9

C) 2

D) 0

E) None of the above / I don’t know

Derivatives

What is the *second* derivative of x^2 evaluated at x=3?

A) 6

B) 9

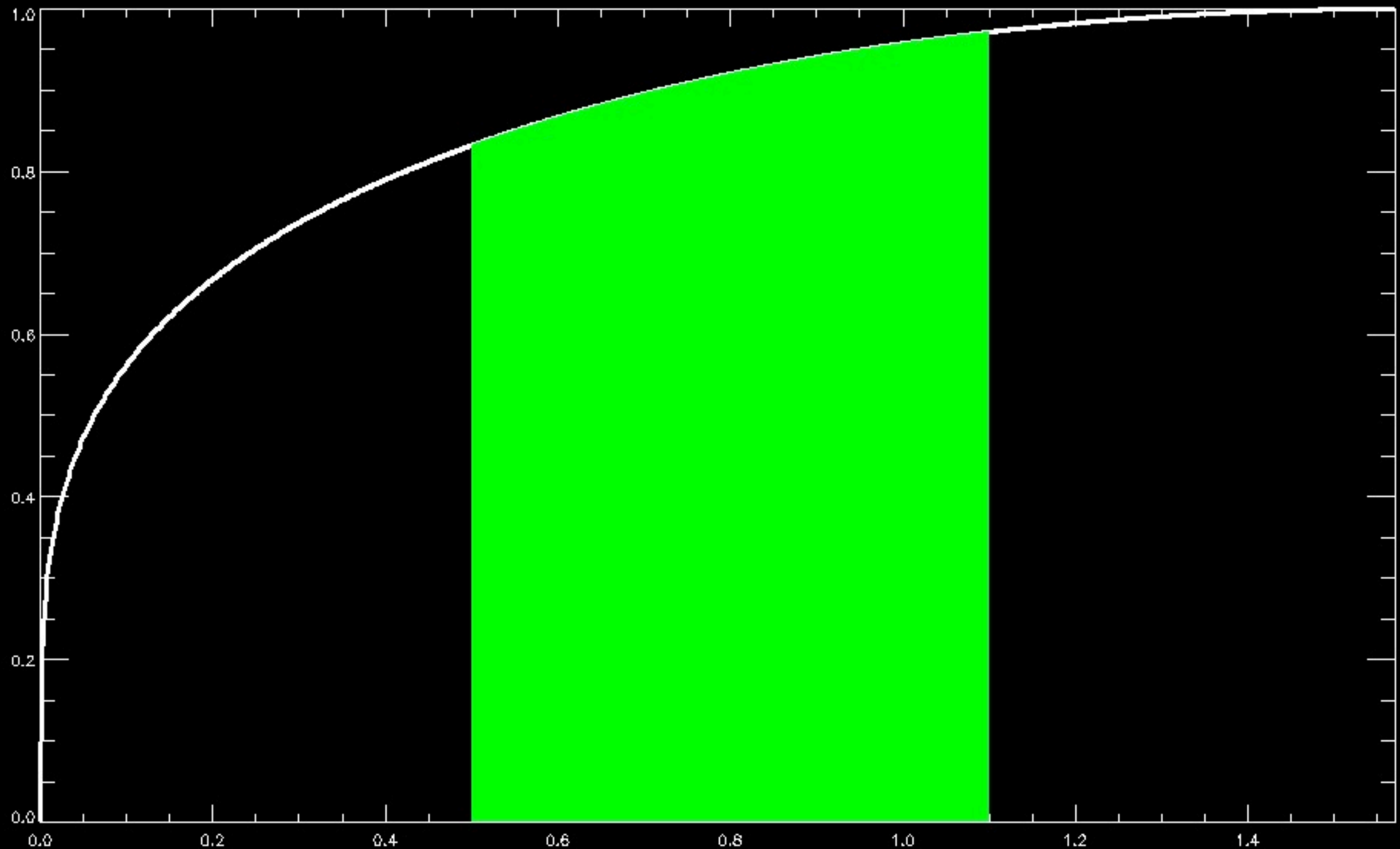
C) 2

D) 0

E) None of the above / I don’t know

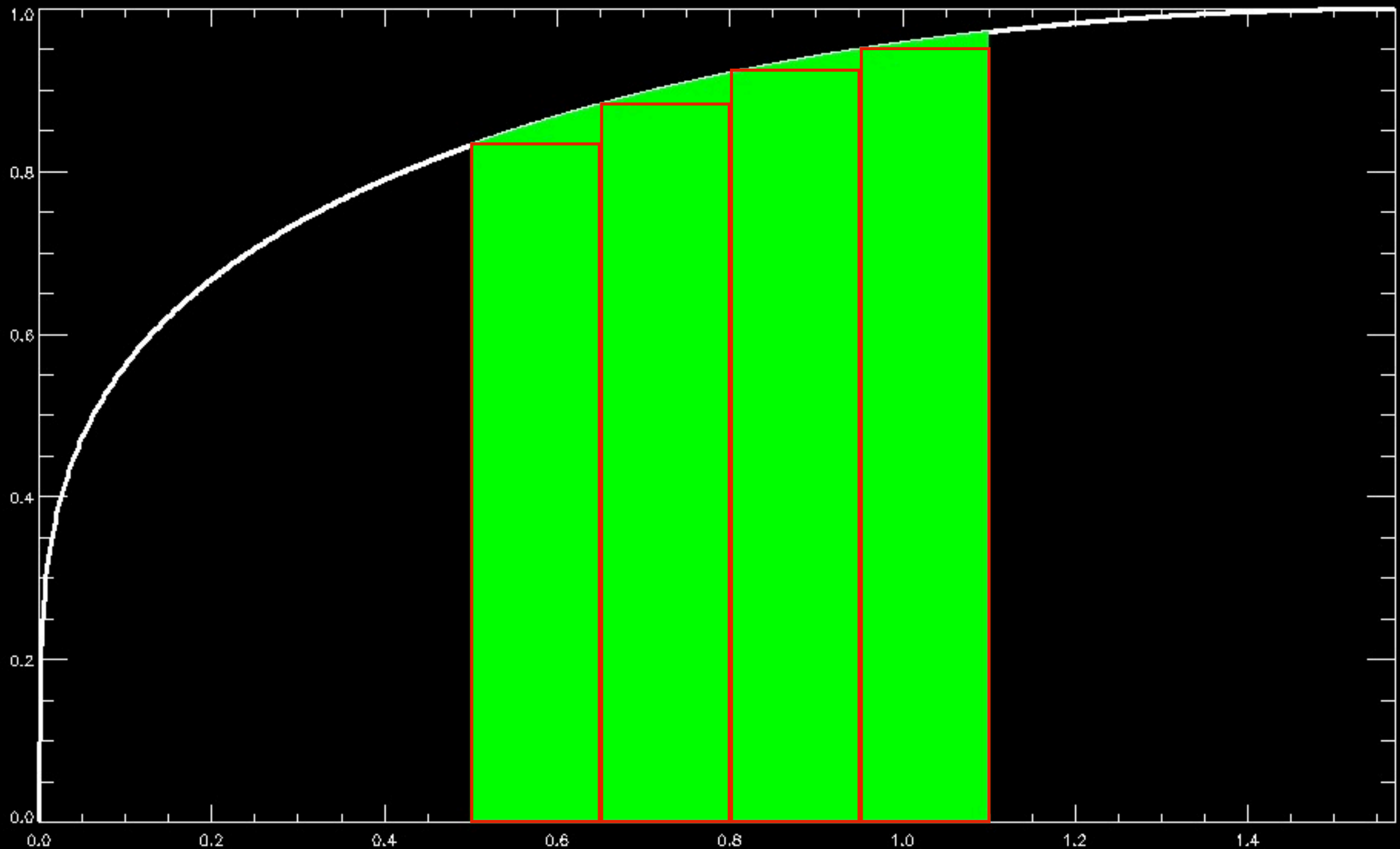
Integration

- Area under a curve



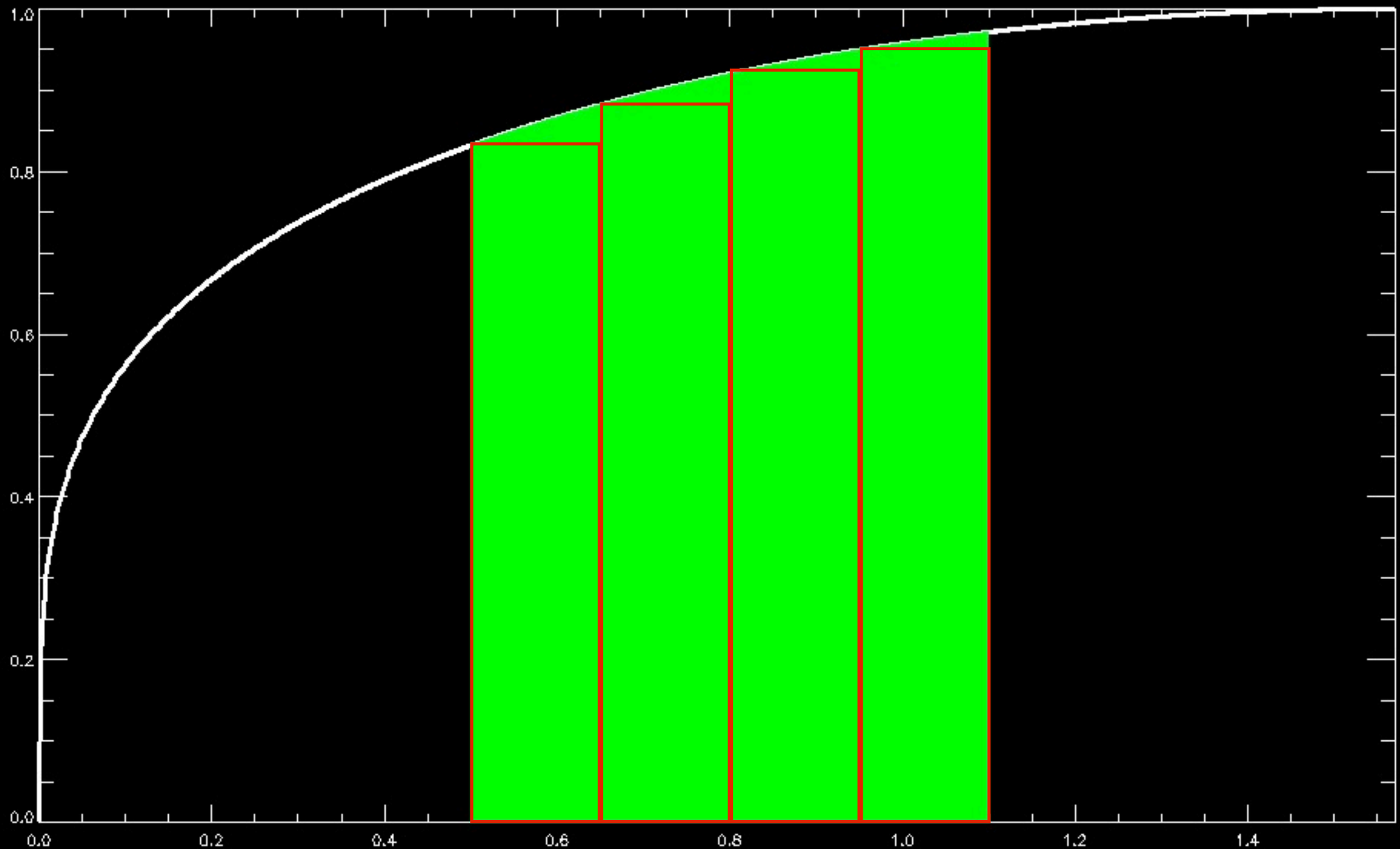
Integration

- Can approximate with rectangles



Integration

- Each rectangle has area dx * y



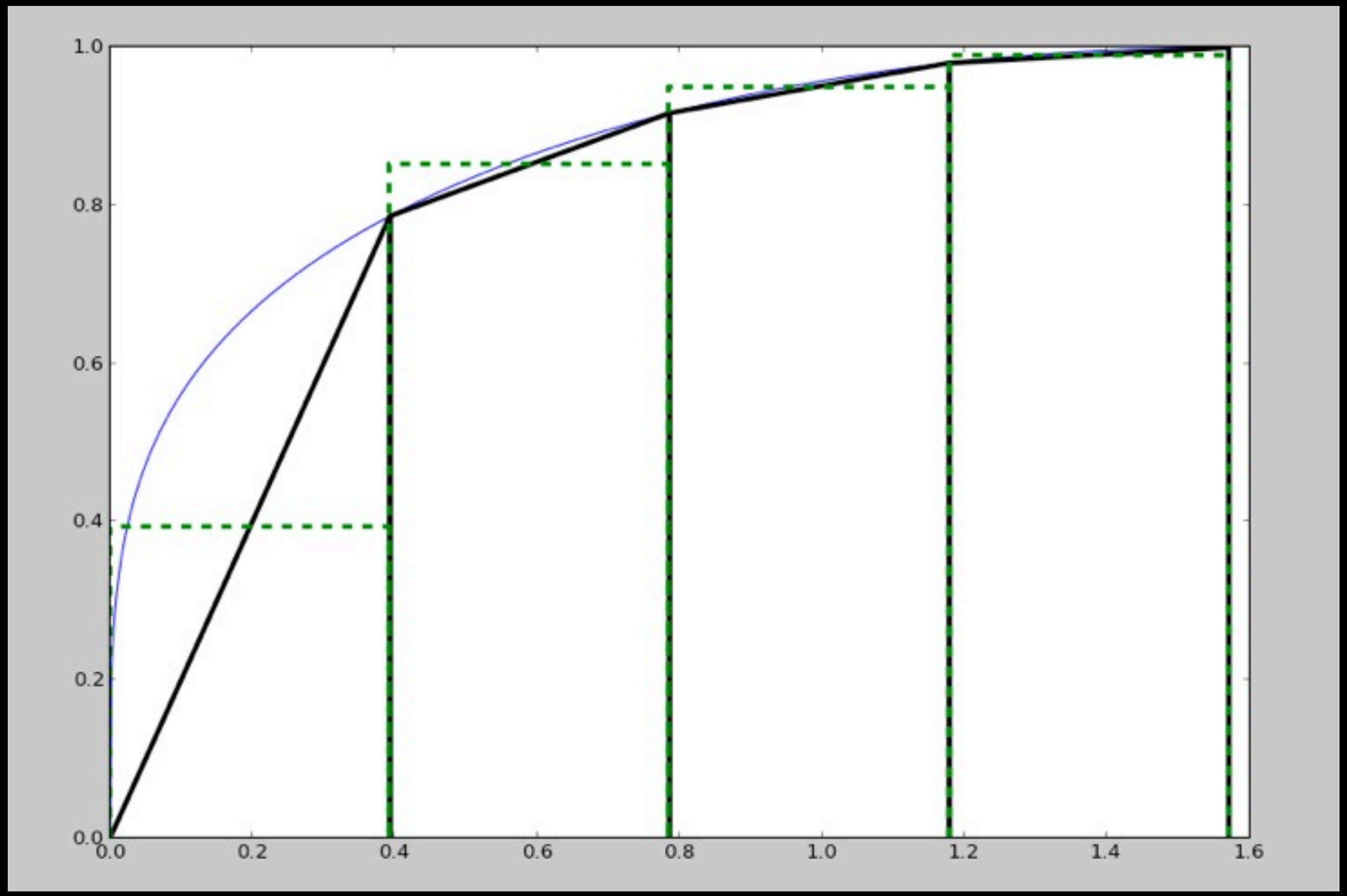
IDL integration

```
IDL> integral_y = total( y * dx )
```

- but this isn’t very accurate for a small number of points

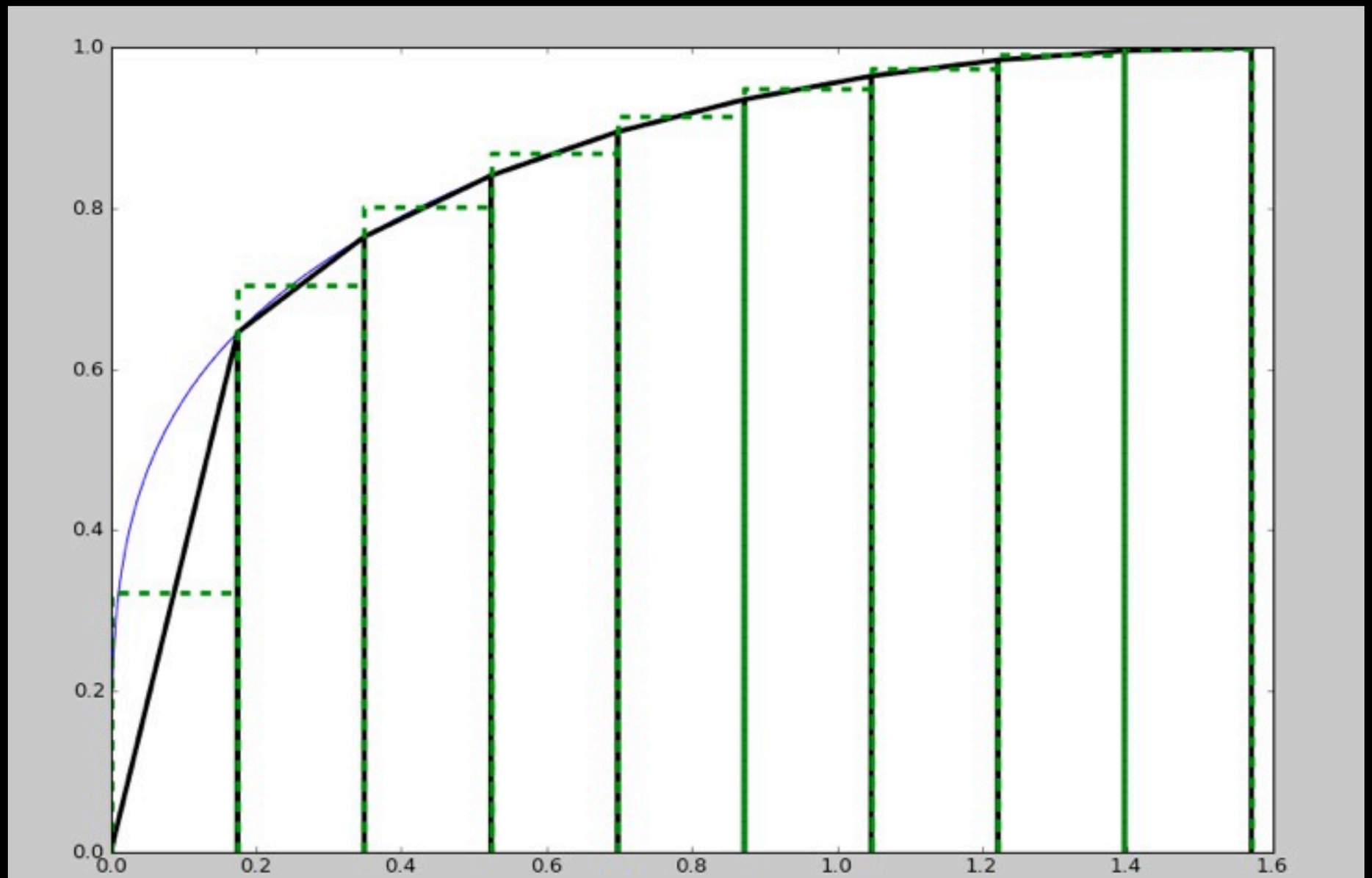
Integration: Trapezoid Rule

- Mathematical trick to drastically improve accuracy



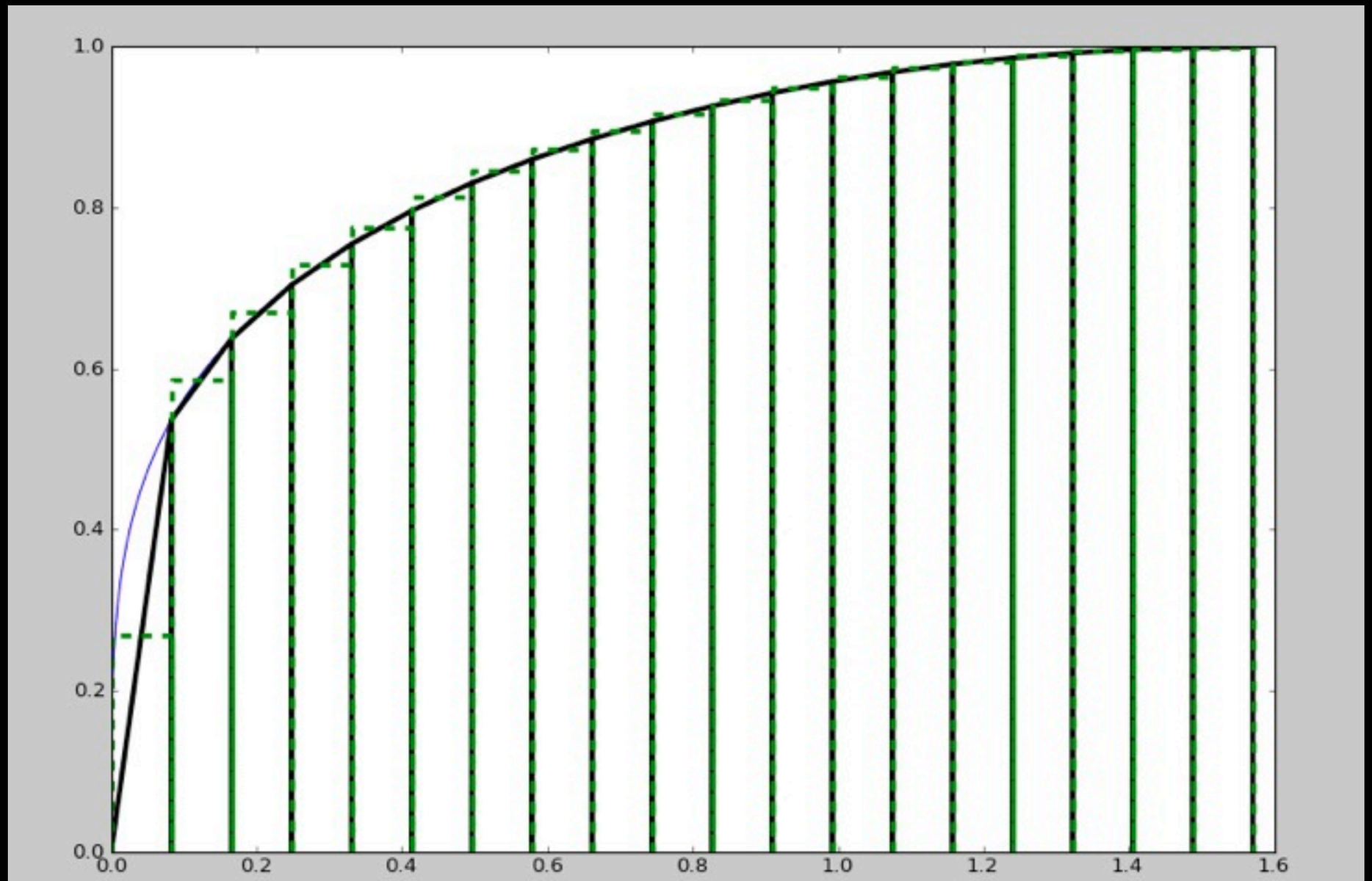
Integration: Trapezoid Rule

- Mathematical trick to drastically improve accuracy



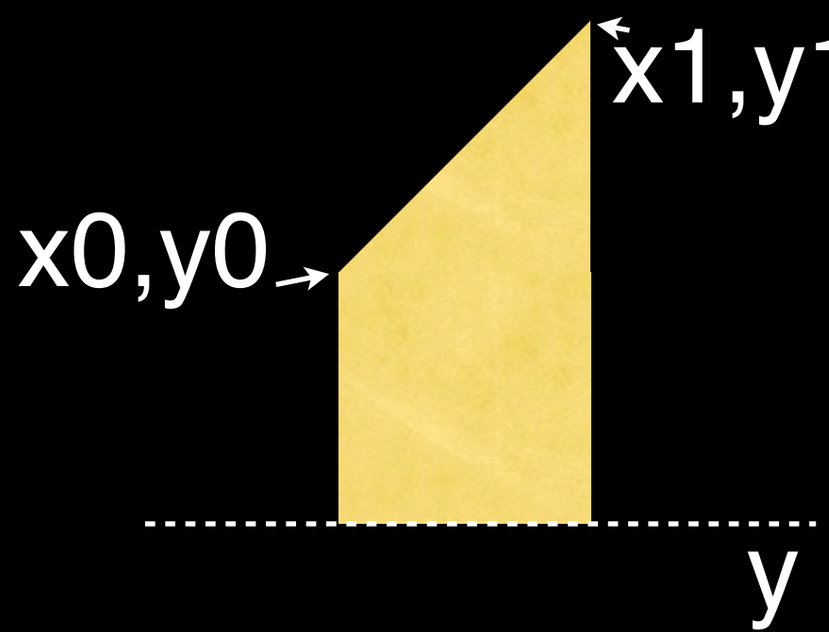
Integration: Trapezoid Rule

- Mathematical trick to drastically improve accuracy



Trapezoid Rule

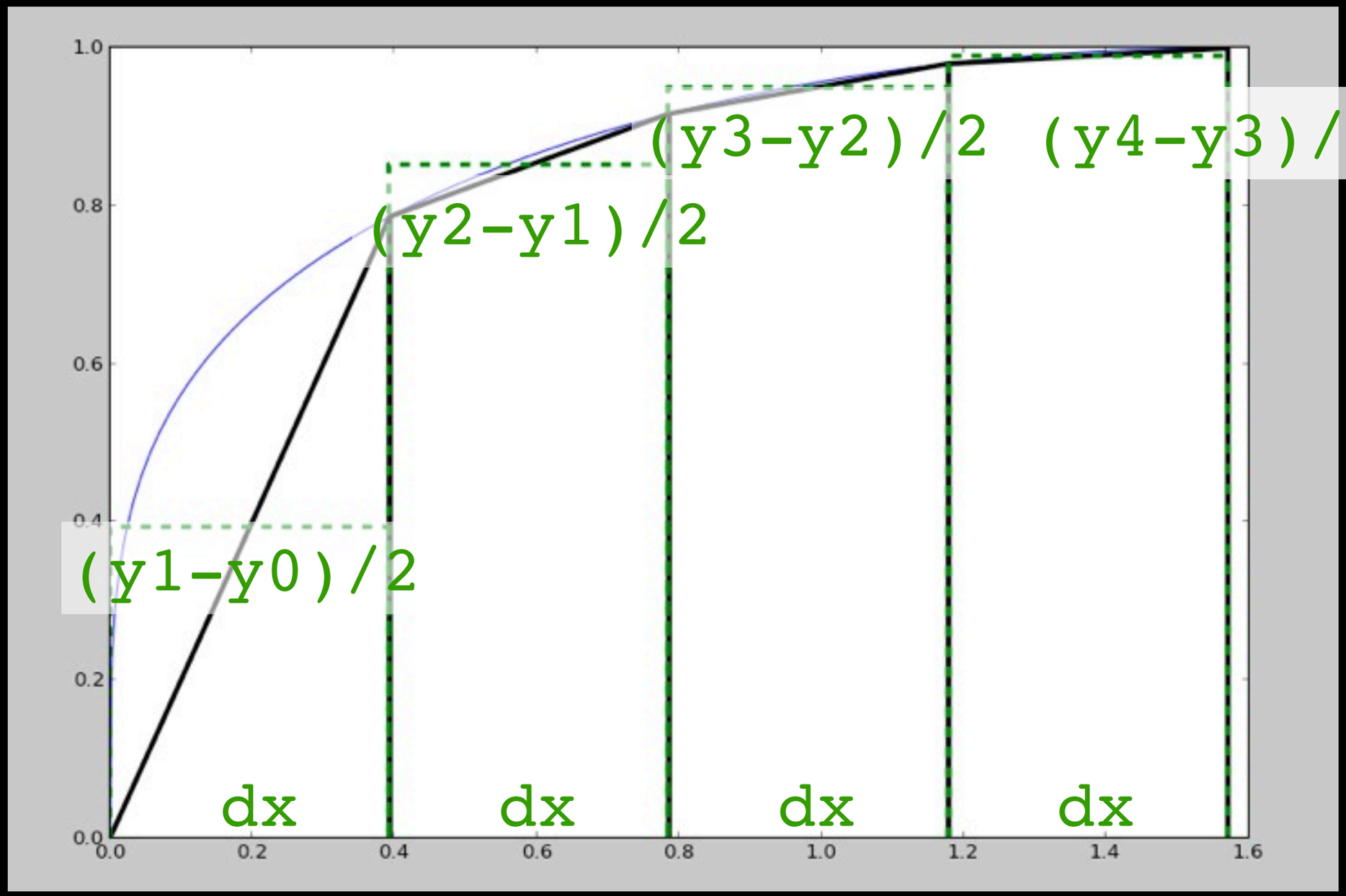
- The area of a trapezoid is just the area of a triangle plus the area of a rectangle (because it's a "right trapezoid")
- The upper edge of the triangle is defined by two points on the curve




```
A = (x1-x0)*(y1-y0)/2
   +(x1-x0)*y0
```

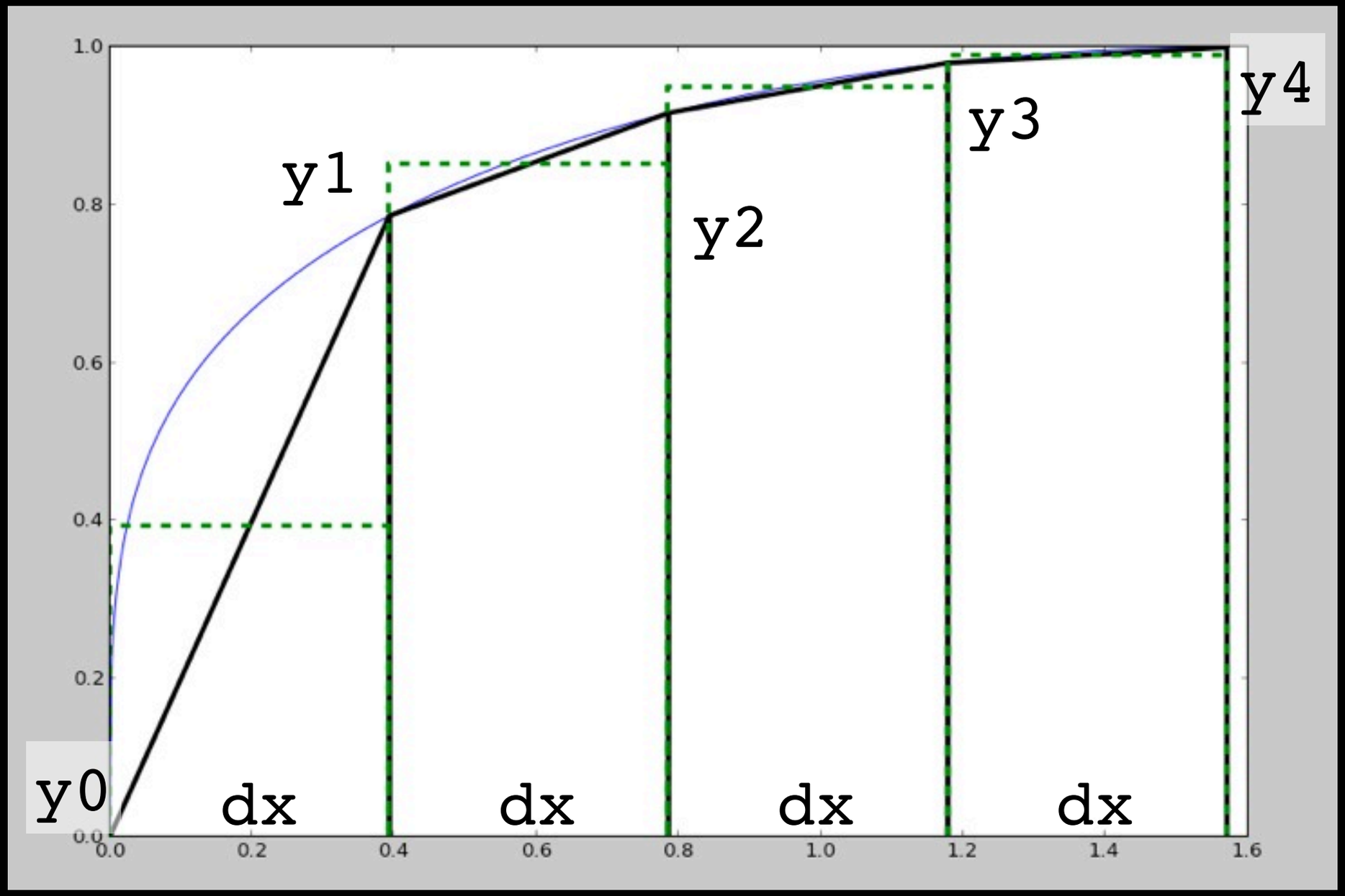
Integration

- Numbers you use for the “box” method



Integration: Trapezoid Rule

- The Trapezoid method



Two methods

- Box:
  `(y[1:*]+y[0:-2])/2. * (x[1:*]-x[0:-2])`
- Trapezoid:
  `(y[0:-2]+(y[1:*]-y[0:-2])) * (x[1:*]-x[0:-2]) / 2`
- There's an even better method called "Simpson's rule": `IDL` function `qsimp`
- You could also do this using `shift`
  - I like the indexing approach because you know which elements you've lost