Are you here on time?

A) Yes

B) No

C) Maybe

D) None of the above

E) All of the above

Random Numbers

- We've already made extensive use of `randomu` and `randomn`
- But we ran into some oddities when doing the randomwalk code
  - Turns out, the book covers this exact problem in Ch 17, pgs 6-11

Computers and Repeatability

- A great advantage of computers is that all actions are repeatable
  - 3+5=8, always
  - 0.1+0.2d = 0.3000000014901161
    - might be weird, but at least it’s always the same kind of weird

Repeatably Random

- Sometimes you need random numbers
- But it's a good thing for "reproducibility" that you can get the same random numbers
  - reproducibility is an important part of the scientific method!

Repeatably Random

- How do you get a random number on a computer, which is otherwise deterministic?
  - deterministic = you get out something that is a function of what you put in
- Pseudo-random algorithms

Pseudorandom Example

- Start with some number between 0 and 1 (a "seed")
- Multiply by pi
- Take the 5th power
- Subtract off the integer component

Pseudorandom Example

```
IDL> x = 0.7
IDL> print,x*!pi
      2.19911
IDL> print,(x*!pi)^5
      51.4327
IDL> print,(x*!pi)^5-51
     0.432720
```

- This gets you numbers that are “kind of random”, but not really because they’re *predictable* and truly random numbers shouldn’t be

Pseudorandom Problems

- Pseudorandom number generators that are formulaic (as in the example) can run into repeating patterns

```
IDL> print,1/7d, format='(F20)'
  0.1428571428571428
```

- This makes your distribution non-random: there are some numbers you'll never see

IDL `random`

- Recall that both random commands take a seed and a shape
- The seed can be a variable (if it's blank, it will be set), an integer, or an old seed
- The return value of `seed` is a 36-element long integer "state array"

Seed State Gets Saved

- The very first time you call `randomu` or `randomn` in a session, the `seed` variable gets set using the system clock time
- After that, it is recorded and the next seed depends entirely on the previous one

Example Code

```
pro print_random
    for ii=0,4 do begin
        x = randomu(seed)
        print,x
    endfor
end
```

Using the next seed

```
IDL> print_random
     0.0979873
      0.267236
      0.690355
      0.298657
    0.00524174
```

```
IDL> print_random
      0.267236
      0.690355
      0.298657
    0.00524174
      0.296611
```

```
IDL> print_random
      0.690355
      0.298657
    0.00524174
      0.296611
      0.436195
```

Scaling Random Numbers

- `randomu` always returns a number between 0 and 1 (but never 0 or 1)

How would you scale `randomu` to get numbers between 0 and 100?

A) `x = randomu(0,100)`

B) `x = randomu(seed,0,100)`

C) `x = randomu(seed) * 100`

D) `x = randomu(seed) * 100 + 1`

E) None of the above

How would you scale `randomu` to get numbers between 5 and 10?

A) `x = randomu(5,10)`

B) `x = randomu(seed) * 5 + 10`

C) `x = randomu(seed) + 5 * 5`

D) `x = randomu(seed) * 5 + 5`

E) None of the above

Simulating Die Rolls

- A die can only have 1 of 6 outcomes, there are no decimals. How do you simulate this?
- First, get numbers between 1 and 7, where 1.0-1.99999 is just as likely as 6.0-6.99999 (and the same for 2,3,4,5)
- Then, *truncate* the float (cut off the decimal part)

Simulating Die Rolls

```
die_random_dec = randomu(seed)*6 + 1
die_random = fix(die_random_dec)
help,die_random_dec,die_random
```

```
IDL> print,fix(randomu(seed,28)*6 + 1)
       5       5       3       1       6       1       1
       1       4       5       4       6       6       5
       3       2       6       2       1       4       5
       3       4       2       4       4       4       1
```

SURVEY: How much time have you spent on Assignment 8 so far?

A) <~ 1 hour

B) ~ 2 hours

C) ~ 3 hours

D) ~ 4 hours

E) >~ 5 hours

SURVEY: How much work is this class compared to other astro classes? (if you’re not taking any, just don’t answer)

A) A lot more

B) A little more

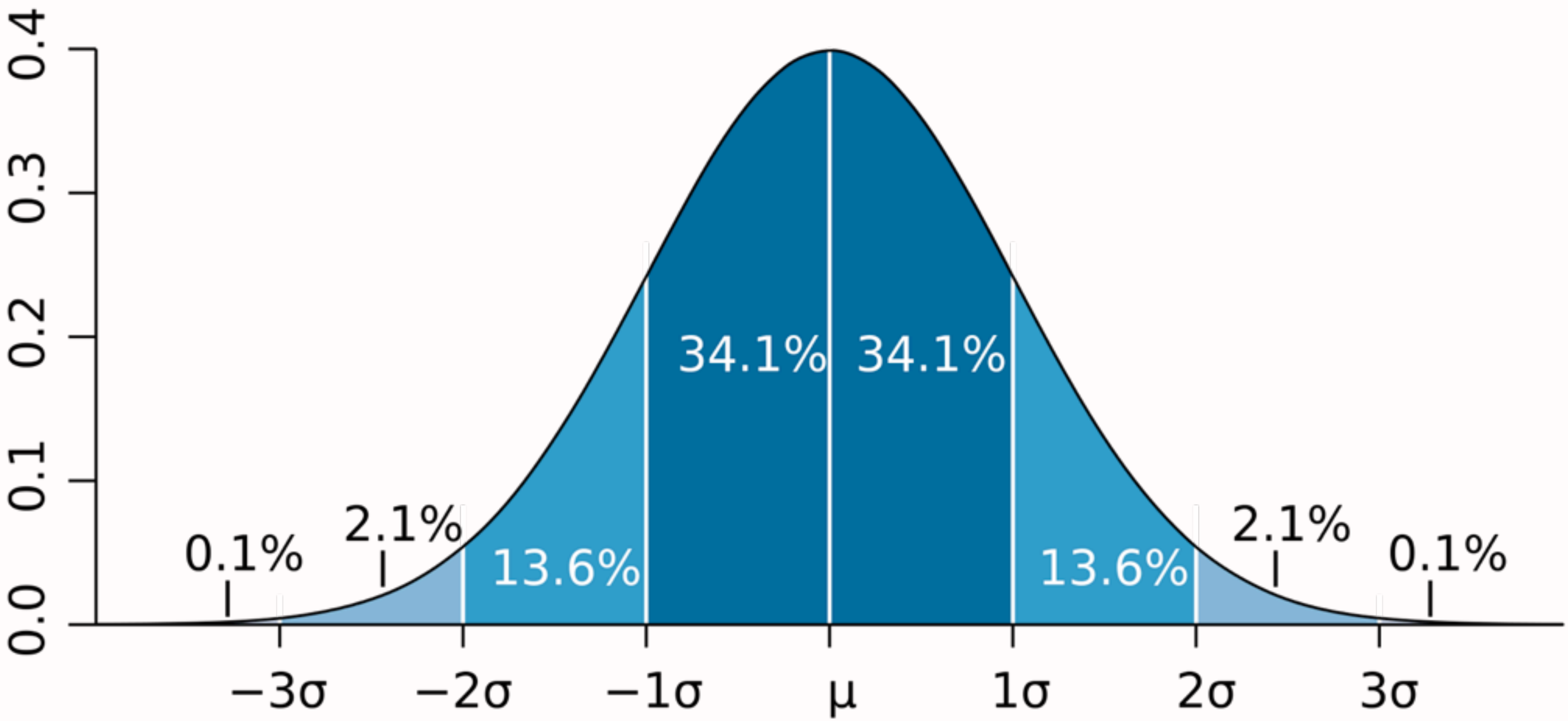
C) About the same

D) A little less

E) A lot less

The Normal distribution

- AKA a Gaussian distribution
- Mean of zero, standard deviation of 1
- looks like a “bell curve”


0.4
0.3
0.2
0.1
0.0
34.1%
34.1%
13.6%
13.6%
2.1%
2.1%
0.1%
0.1%
−3σ
−2σ
−1σ
μ
1σ
2σ
3σ

The Gaussian Distribution

- Probably the most important distribution in all of probability
- Represents nearly everything (in the limit of large numbers)
- Is the result of the “central limit theorem”

How would you scale the normal to a distribution with mean π and standard deviation of 2?

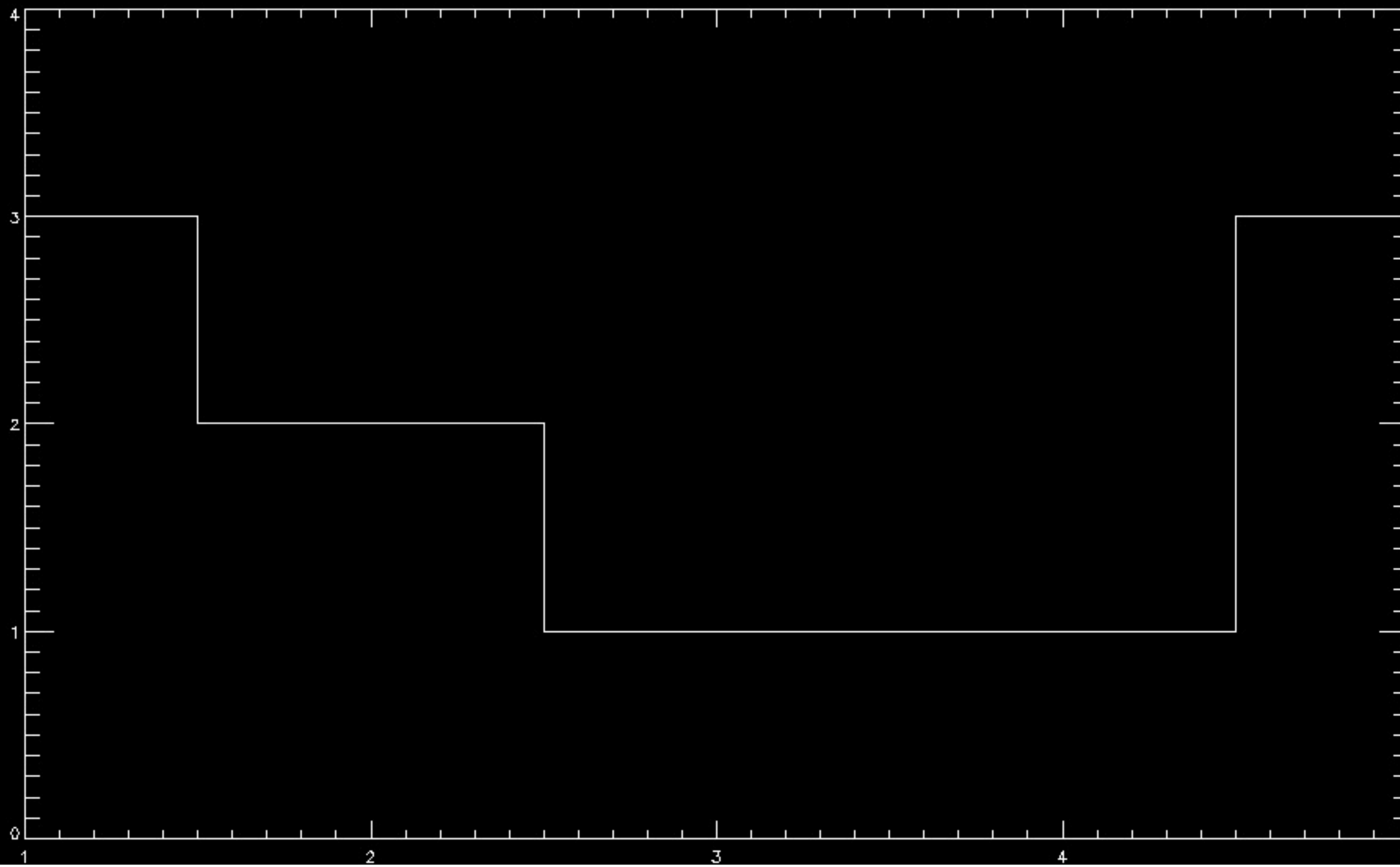
```
A)randomn(seed) * 2 + !pi

B)randomn(seed) * !pi + 2

C)randomn(seed) * (!pi + 2)

D)randomn(seed) - 2 + !pi
```

E) None of the above

Histograms

- To visualize a distribution, you usually want to work with histograms
- Histograms tell you how many values are in each bin
- Example: If you have the array [1,1,1,2,2,3,4,5,5,5], there are three 1’s, two 2’s, one 3, one 4, three 5’s

A Histogram



IDL histogram

- IDL’s histogram is kind of ugly
  - the `histogram` command is very powerful, but doesn’t produce pretty plots easily
- We’ll still use it

Histograms

- Are great for examining distributions
- The uniform distribution should be uniform from 0 to 1 (i.e., same number in each bin)

Histogram Example

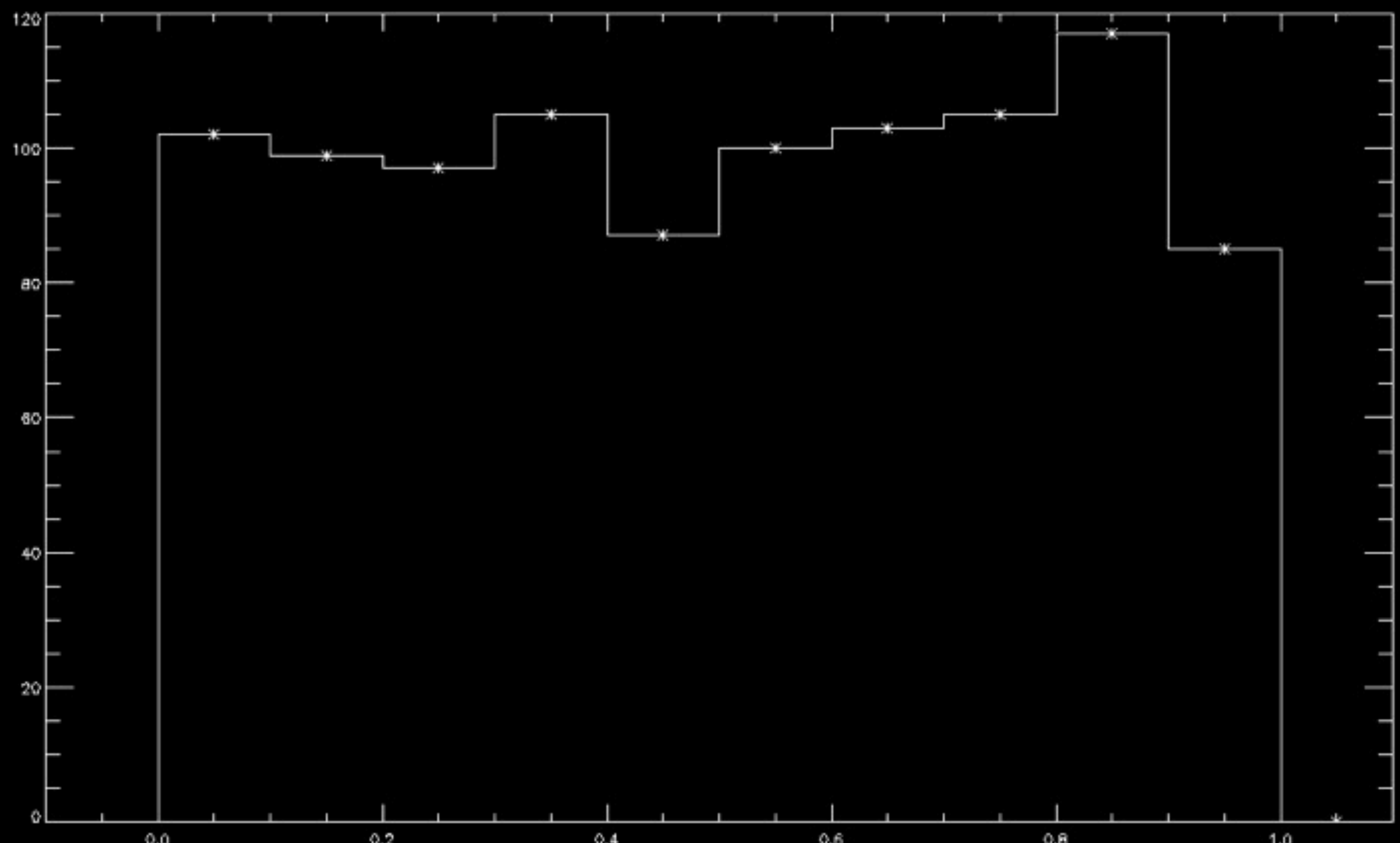
```
IDL> x=randomu(seed,1000)
IDL> h=histogram(x)
IDL> help,h
H               LONG      = Array[1]
```

- Unfortunately, `histogram` isn’t quite clever enough to determine the right bin size on its own
- We need keywords...

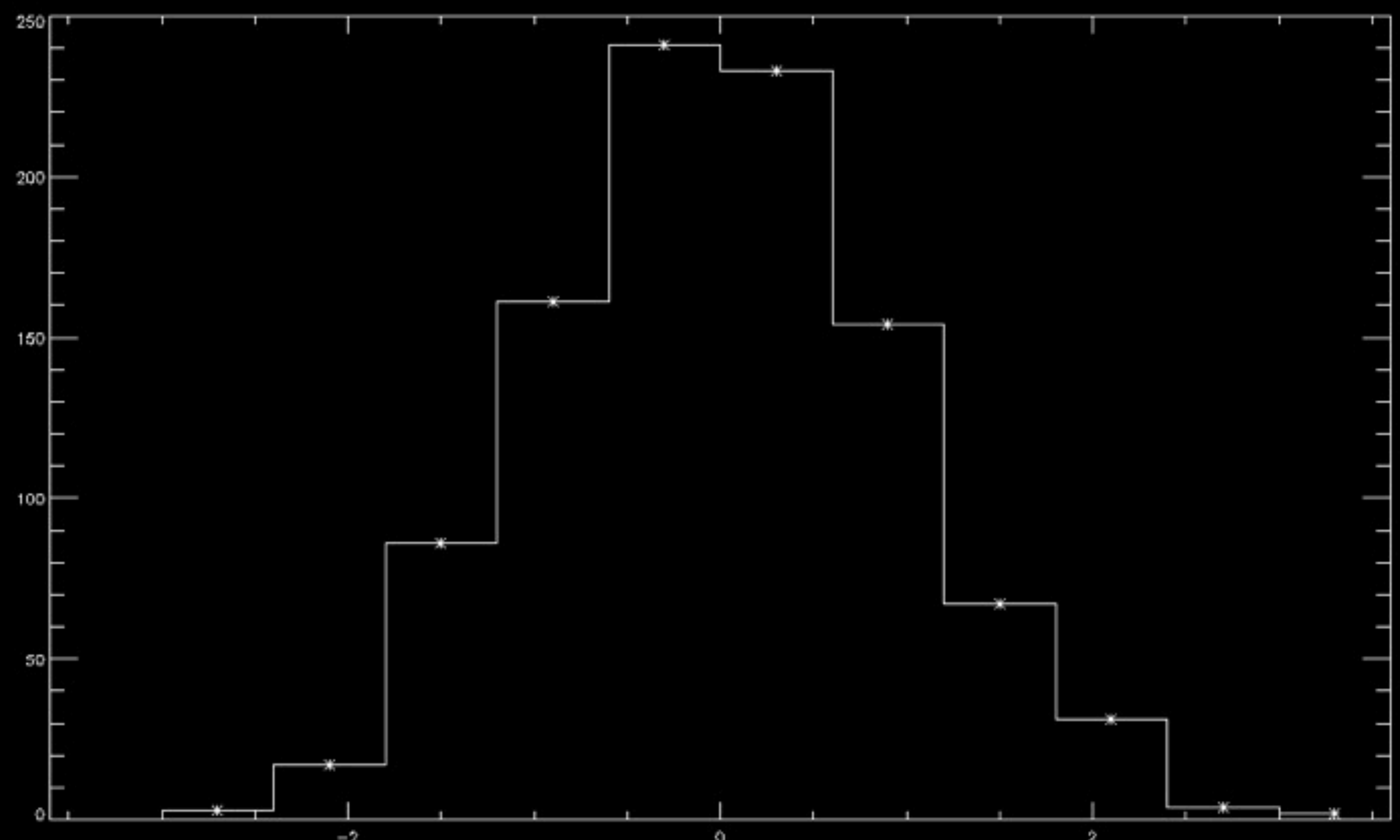
Histogram

- The endpoints are weird
  - To draw a histogram properly, you actually need 1 more point than you have bins! [draw on board]
- Creating a real histogram is therefore problematic. IDL will give you a “trailing zero”, but you have to add your own “leading zero”

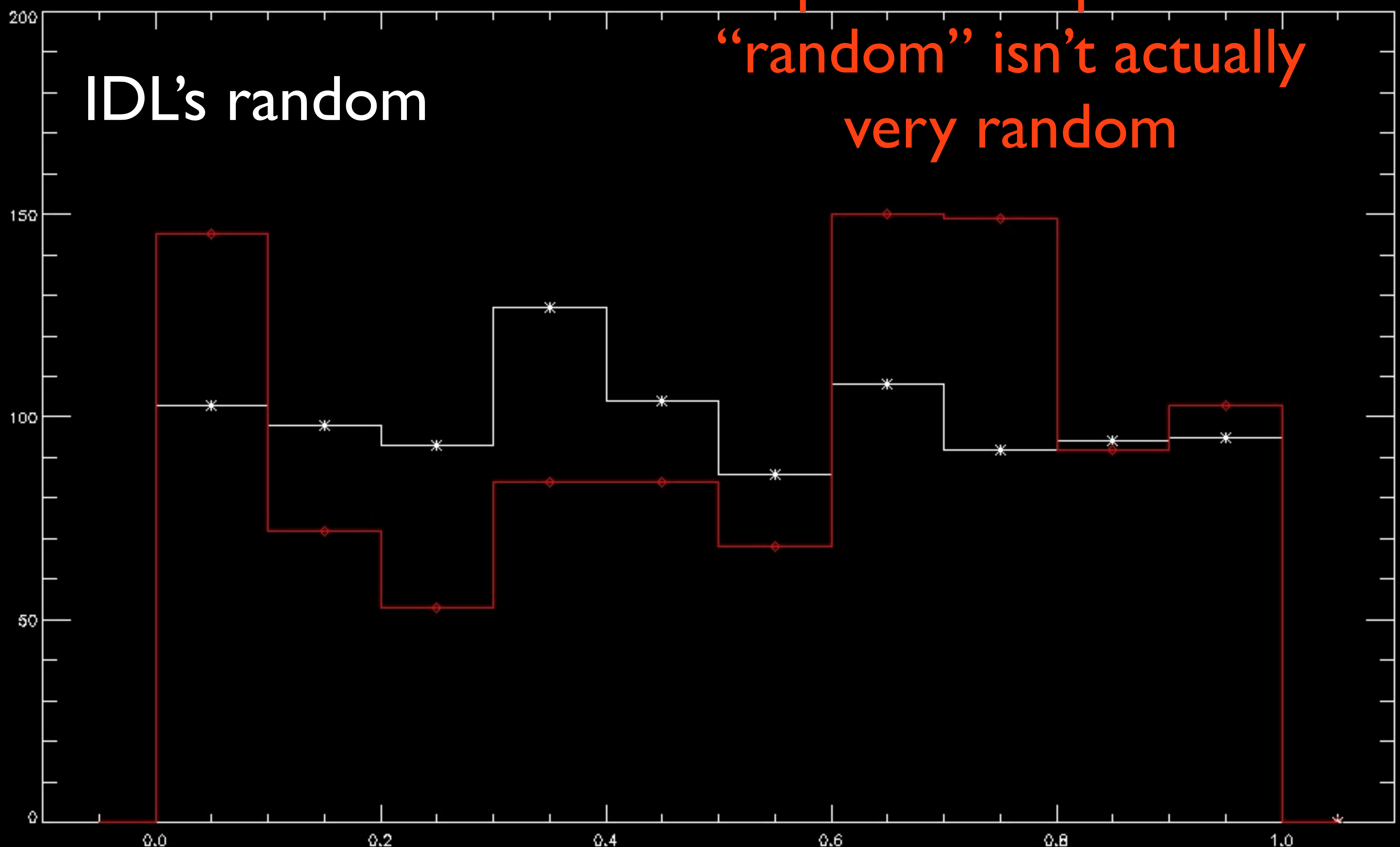
```
x = randomu(seed,1000)
nbins=11
min=0
max=1
h = histogram(x,nbins=nbins,min=min,max=max)
binsize = (max - min) / float(nbins-1)
plot,findgen(nbins)*binsize+min+binsize/2.,h,psym=2,$
    xrange=[min-binsize,max+binsize],xs=1
oplot,findgen(nbins+1)*binsize+min-binsize/2.,[0,h],psy
```



```
x = randomn(seed,1000)
nbins=11
min=-3
max=3
h = histogram(x,nbins=nbins,min=min,max=max)
binsize = (max - min) / float(nbins-1)
plot,findgen(nbins)*binsize+min+binsize/2.,h,psym=2,$
    xrange=[min-binsize,max+binsize],xs=1
oplot,findgen(nbins+1)*binsize+min-binsize/2.,[0,h],psym=
```



Comparing Histograms



Last note on histograms

- I had to play with the histogram parameters for quite a while to figure out what was "right"
- You will use histograms on the next set of exercises - use this lecture and pgs 14-17 of Ch 17 as a reference

(ch 18) Pointers

- Pointers are sort of a new data type
- We will use them to make “linked lists”
  - Linked lists are the solution to the problem, “What if I don’t know how many elements I need in my array when I make it?”, especially for structs where you can’t change your setup

Pointers

- In most programming languages, pointers are variables that contain a *location in memory* rather than a value
- In principle, that location in memory can contain anything
- IDL does something a little different: pointers point to “unnamed heap variables”, which are *global* but only exist as long as something refers to them

Why Pointers?

- Pointers are also a more efficient way to deal with data
  - If you have a single array (say, a movie) that is 2 GB, you don't want to create multiple copies of it!
    - when you run out of RAM, your computer uses swap, which is slow
  - Instead, you "refer to" the data in memory

How pointers?

```
IDL> x = ptr_new(5)
IDL> print,x
<PtrHeapVar1>
IDL> print,*x
       5
```

- The “heap” is the global storage location, in principle accessible to any program

```
x = ptr_new(5)
y = x
print,y
```

What do you think will print out?

A) 5

B) ‘x’

C) x

D) <PtrHeapVar1>

E) None of the above

```
x = ptr_new(5)
y = x
print,*y
```

What do you think will print out?

A) 5

B) ‘x’

C) x

D) <PtrHeapVar1>

E) None of the above

Pointers & Assignment

- When you assign one pointer to another, it creates a copy of the pointer
  - This is just like any other variable
- It does *not* create a copy of the data the pointer points to

```
x = ptr_new(5)
y = x
*x = 2
print,*x,*y
```

What will print?

A) 5,2

B) 2,5

C) 2,2

D) 5,5

E) None of the above

Garbage Collection

- A general term in programming meaning “free up the memory when you’re done using it”
- With heap variables and pointers, you have to clean up after yourself
- `ptr_free,x` empties the data space that `x` pointed to

Free Pointers

```
x = ptr_new(5)
y = x
*x = 2
print,*x,*y

ptr_free,x
print,*y
% Invalid pointer: Y.
% Execution halted at: $MAIN$
```

- `y` points to the same data as `x`, but now it's gone

Memory Leak

- Always free your pointers when you’re done with them
- If not, you get a memory leak
  - This will first slow down your computer, then crash it
  - Can have pretty nasty consequences, sometimes crashing the whole system

`NULL` pointer

- You can make an empty pointer:

```
nulp = ptr_new()
help,nulp
NULP                  POINTER    = <NullPointer>
```

- You can then assign it a value later

Structures with Arrays

- Revisiting the old car structure:

```
car1 = {CarCatalogEntry,$  ; Structure Name
    license:"AAAA1111",$   ; String Variable
    make: 'Saturn',$       ; String Variable
    miles: 12000L,         ; Long Variable
    serviced_at_miles: [3000,6000,20000]}
        ; Array variable
```

- Problem: What if you have two cars with different numbers of service visits?

Structures with Arrays

- We can replace the array with a *pointer* to an array:

```
car1 = {CarCatalogEntry,$  ; Structure Name
    license:"AAAA1111",$   ; String Variable
    make: 'Saturn',$       ; String Variable
    miles: 12000L,         ; Long Variable
    serviced_at_miles: ptr_new([3000,6000,20000])}
        ; Array variable
```

- Now any time we make a new car, `serviced_at_miles` is a pointer, and the array it points to can be any length

Referencing

```
IDL> print,(car1.serviced_at_miles)
<PtrHeapVar2>
IDL> print,*(car1.serviced_at_miles)
     3000     6000    20000
```

- Parentheses are necessary!
- They define the “order of operations” between “.” and “*”
  - Otherwise, you wouldn’t know if `car1` or `serviced_at_miles` was the pointer

New Structs & Null Pointers

```
IDL> car2 = {CarCatalogEntry}
IDL> help,car2.serviced_at_miles
<Expression>    POINTER   = <NullPointer>
```

- New instances of the structure default to null pointers
- You must use `ptr_new` any time you assign something to a null pointer

Assignment & ptr_new

```
IDL> car2 = {CarCatalogEntry}
IDL> help,car2.serviced_at_miles
<Expression>    POINTER   = <NullPointer>
IDL> *(car2.serviced_at_miles) = [3000,6000,10000,15000]
% Unable to dereference NULL pointer: <POINTER
   (<NullPointer>)>.
% Execution halted at: $MAIN$
IDL> car2.serviced_at_miles = ptr_new([3000,6000,10000,1
5000])
```

Lab Today

- Tutorial 18 - Global Variables & Unit Tests
  - (plus, change `carstruct__define.pro`)
- First, brief review of `execute`

```
varname =  "x"
value = '15'
cmd = string(varname,value,format="(A,'=',A)")
print,"Command: ",cmd
did_it_work = execute(cmd)
print,"Did it work?  1 for yes, 0 for no: ",did_it_work
print,"What is X now?",x
end
```

```
IDL> .r generated_code
% Compiled module: $MAIN$.
Command: x=15
Did it work?  1 for yes, 0 for no:        1
What is X now?      15
```

Small but complex example

```
pro test_mks_units

    ; make a hash storing the appropriate values
    tagvals = hash('au',1.496e11,$
                   'kmpersec',1e3,$
                   'year',365L*24*3600,$
                   'parsec',3.08567758e16)

    ; make a nicely formatted table
    print,"Tag Name","Command","Value",format="(3A20)"
    foreach val,tagvals,tag do begin
        cmd = 'OK = ((mks_units()).'+ tag +') eq '+string(val)
        test_status = execute(cmd)
        msg1 = test_status ? "Passed" : "X Failed X"
        msg2 = OK ? "Passed" : "X Failed X"
        print,tag,msg1,msg2,format="(3A20)"
    endforeach

end ; test_mks_units
```

(this code is available, with better comments, on github)