DEBUGGING

Prof. Chris Jermaine
cmj4@cs.rice.edu

Prof. Scott Rixner
rixner@cs.rice.edu
Everyone Writes Code With Bugs

• Being able to quickly squash can radically increase productivity
• But how to do it?
Everyone Writes Code With Bugs

• Being able to quickly squash can radically increase productivity
• But how to squash bugs?
  — A poor or beginning programmer will often:
    — Stare at the code for a long time
    — Make a random change (“voodoo programming”)
    — Try the test again, watch it fail
    — Repeat until exhausted
    — Then, s/he’ll blame the compiler/faulty memory/FP error, etc.
Experienced and Skilled Programmers

• Tend not to change any code until they know what the problem is
  — That is, they actively avoid voodoo programming, knowing it’s a waste of time
  — (Though even the best programmers lose their cool sometimes!)

• So how do they debug?
  — They imagine the space of possible things that can go wrong:
Experienced and Skilled Programmers

• Tend not to change any code until they know what the problem is
  — That is, they actively avoid voodoo programming, knowing it’s a waste of time
  — (Though even the best programmers lose their cool sometimes!)

• So how do they debug?
  — They come up with a sanity check that can slice off as much of the space of bad stuff as possible
Experienced and Skilled Programmers

• Tend not to change any code until they know what the problem is
  — That is, they actively avoid voodoo programming, knowing it’s a waste of time
  — (Though even the best programmers lose their cool sometimes!)

• So how do they debug?
  — Pass the check? Then rule out that problem and all its causes. Then, devise another check...

devise a check to rule this out
Experienced and Skilled Programmers

• Tend not to change any code until they know what the problem is
  — That is, they actively avoid voodoo programming, knowing it’s a waste of time
  — (Though even the best programmers lose their cool sometimes!)

• So how do they debug?
  — Fail the check? You know there is a problem in here...
Experienced and Skilled Programmers

• Tend not to change any code until they know what the problem is
  — That is, they actively avoid voodoo programming, knowing it’s a waste of time
  — (Though even the best programmers lose their cool sometimes!)

• So how do they debug?
  — So keep going...

Maybe a problem in here

devise a check to rule this out
Key: Be Systematic and Thoughtful

• Your sanity check should always remove as much of the space of possible problems as is possible

• In voodoo programming, you are removing almost nothing
  — When (in despair) you change:
    \[ x\.thisIsMyCode \ (a, \ b, \ c) ; \]
  — To:
    \[ y\.thisIsMyNewCode \ (c) ; \]
  — You are examining only the possibility that the real problem was that the first one needed to be replaced with the second. How is that useful?
When You Are Debugging

• Heed the advice of Sherlock Holmes:
  “How often have I said that when you have eliminated the impossible, whatever remains, however improbable, must be the truth?”
What Do These Checks Look Like?

• Most often they are bits of code that print some program state
• Sometimes, they are very complicated, and involved writing many (hundreds of?) SLOC to gather and distill that state
• Debuggers can sometimes be useful
  — Because you can interactively check program state
  — But if the check involves more than just checking the state of some objects...
  — It’s often better to just write some code and eschew the debugger
  — Debuggers are especially useful for languages with pointers and w/o good exception handling (for example, C)
  — But don’t use ‘em much myself when writing Java code
Let’s Look At an Example From A4

• Here’s the output from one of the test cases:

testMultinomial1:

"Got 33791.99999, expected 0.0 when I was checking the total distance..."
Carefully Look At the Context of the Error

"Got 33791.99999, expected 0.0 when I was checking the total distance..."

testMultinomial1:

```java
int len = 100;
SparseDoubleVector probs = new SparseDoubleVector (len, 0.0);
for (int i = 0; i < len; i += 2) {
    probs.setItem (i, i);
}
probs.normalize ();

// now, set up a distribution
IRandomGenerationAlgorithm<IDoubleVector> myRNG =
    new Multinomial (27, new MultinomialParam (1024, probs));

// and check the mean...
DenseDoubleVector expectedMean = new DenseDoubleVector (len, 0.0);
DenseDoubleVector expectedStdDev = new DenseDoubleVector (len, 0.0);
for (int i = 0; i < len; i++) {
    expectedMean.setItem (i, probs.getItem (i) * 1024);
    expectedStdDev.setItem (i, Math.sqrt (probs.getItem (i) * 1024 * (1.0 - probs.getItem (i))));
}
checkMeanAndVar (myRNG, expectedMean, expectedStdDev, 5.0, 5.0, 5000, "multinomial number one");
```
"Got 33791.99999, expected 0.0 when I was checking the total distance..."

Key question:
How could I possibly be off by 30K with 1024 trials?
Unless the # of trials is incorrect...

```java
int len = 100;
SparseDoubleVector probs = new SparseDoubleVector (len, 0.0);
for (int i = 0; i < len; i += 2) {
    probs.setItem (i, i);
}
probs.normalize ();

// now, set up a distribution
IRandomGenerationAlgorithm<IDoubleVector> myRNG =
    new Multinomial (27, new MultinomialParam (1024, probs));

// and check the mean...
DenseDoubleVector expectedMean = new DenseDoubleVector (len, 0.0);
DenseDoubleVector expectedStdDev = new DenseDoubleVector (len, 0.0);
for (int i = 0; i < len; i++) {
    expectedMean.setItem (i, probs.getItem (i) * 1024);
    expectedStdDev.setItem (i, Math.sqrt (probs.getItem (i) * 1024 * (1.0 - probs.getItem (i))));
}

checkMeanAndVar (myRNG, expectedMean, expectedStdDev,
                 5.0, 5.0, 5000, "multinomial number one");
Start By Looking At checkMeanAndVar

```java
IDoubleVector firstOne = myRNG.getNext();
DenseDoubleVector meanObs = new DenseDoubleVector (firstOne.getLength (), 0.0);
DenseDoubleVector stdDevObs = new DenseDoubleVector (firstOne.getLength (), 0.0);

// add in a bunch more
for (int i = 0; i < numTrials; i++) {
    IDoubleVector next = myRNG.getNext();
    next.addMyselfToHim (meanObs);
    for (int j = 0; j < next.getLength (); j++) {
        stdDevObs.setItem (j, stdDevObs.getItem (j) + next.getItem (j) *
            next.getItem (j));
    }
}

// divide by the number of trials to get the mean
for (int i = 0; i < meanObs.getLength (); i++) {
    meanObs.setItem (i, meanObs.getItem (i) / numTrials);
    stdDevObs.setItem (i, Math.sqrt (stdDevObs.getItem (i) / numTrials -
        meanObs.getItem (i) * meanObs.getItem (i)));
}

// see if the mean and var are acceptable
checkTotalDiff (meanObs, expectedMean, errorMean,  
    "total distance from true mean", dist);
checkTotalDiff (stdDevObs, expectedStdDev, errorStdDev, 
    "total distance from true standard deviation", dist)
```
Add In a Check on the Number of Trials

```
IDoubleVector firstOne = myRNG.getNext ();
DenseDoubleVector meanObs = new DenseDoubleVector (firstOne.getLength (), 0.0);
DenseDoubleVector stdDevObs = new DenseDoubleVector (firstOne.getLength (), 0.0);

// add in a bunch more
double len = 0;
for (int i = 0; i < numTrials; i++) {
    IDoubleVector next = myRNG.getNext ();
    len += next.l1Norm ();
    next.addMyselfToHim (meanObs);
    for (int j = 0; j < next.getLength (); j++) {
        stdDevObs.setItem (j, stdDevObs.getItem (j) +
                          next.getItem (j) * next.getItem (j));
    }
}
System.out.println ("avg number of balls found was " + len / numTrials);

// divide by the number of trials to get the mean
for (int i = 0; i < meanObs.getLength (); i++) {
    meanObs.setItem (i, meanObs.getItem (i) / numTrials);
    stdDevObs.setItem (i, Math.sqrt (stdDevObs.getItem (i) / numTrials -
                                  meanObs.getItem (i) * meanObs.getItem (i)));
}

// see if the mean and var are acceptable
checkTotalDiff (meanObs, expectedMean, errorMean,
               "total distance from true mean", dist);
checkTotalDiff (stdDevObs, expectedStdDev, errorStdDev,
               "total distance from true standard deviation", dist)
```
Add In a Check on the Number of Trials

```java
IDoubleVector firstOne = myRNG.getNext ();
DenseDoubleVector meanObs = new DenseDoubleVector (firstOne.getLength (), 0.0);
DenseDoubleVector stdDevObs = new DenseDoubleVector (firstOne.getLength (), 0.0);

// add in a bunch more
double len = 0;
for (int i = 0; i < numTrials; i++) {
    IDoubleVector next = myRNG.getNext ();
    len += next.l1Norm ();
    next.addMyselfToHim (meanObs);
    for (int j = 0; j < next.getLength (); j++) {
        stdDevObs.setItem (j, stdDevObs.getItem (j) +
                             next.getItem (j) * next.getItem (j));
    }
}
System.out.println ("avg number of balls found was " + len / numTrials);

// divide by the number of trials to get the mean
for (int i = 0; i < meanObs.getLength (); i++) {
    meanObs.setItem (i, meanObs.getItem (i) / numTrials);
    stdDevObs.setItem (i, Math.sqrt (stdDevObs.getItem (i) / numTrials -
                                meanObs.getItem (i) * meanObs.getItem (i)));
}

// see if the mean and var are acceptable
checkTotalDiff (meanObs, expectedMean, errorMean,
                "total distance from true mean", dist);
checkTotalDiff (stdDevObs, expectedStdDev, errorStdDev,
                "total distance from true standard deviation", dist)
```

Output is:

“avg number of balls found was 1024.0”
Hmmm... how is this possible? We gotta look at the vecs
So Add In Some Code To Print Them Out

```java
IDoubleVector firstOne = myRNG.getNext();
DenseDoubleVector meanObs = new DenseDoubleVector(firstOne.getLength(), 0.0);
DenseDoubleVector stdDevObs = new DenseDoubleVector(firstOne.getLength(), 0.0);

// add in a bunch more
for (int i = 0; i < numTrials; i++) {
    IDoubleVector next = myRNG.getNext();
    next.addMyselfToHim(meanObs);
    for (int j = 0; j < next.getLength(); j++) {
        stdDevObs.setItem(j, stdDevObs.getItem(j) + next.getItem(j) * next.getItem(j));
    }
}

// divide by the number of trials to get the mean
for (int i = 0; i < meanObs.getLength(); i++) {
    meanObs.setItem(i, meanObs.getItem(i) / numTrials);
    stdDevObs.setItem(i, Math.sqrt(stdDevObs.getItem(i) / numTrials - meanObs.getItem(i) * meanObs.getItem(i)));
}

// see if the mean and var are acceptable
checkTotalDiff(meanObs, expectedMean, errorMean, "total distance from true mean", dist);
checkTotalDiff(stdDevObs, expectedStdDev, errorStdDev, "total distance from true standard deviation", dist)
```
So Add In Some Code To Print Them Out

IDoubleVector firstOne = myRNG.getNext ();
DenseDoubleVector meanObs = new DenseDoubleVector (firstOne.getLength (), 0.0);
DenseDoubleVector stdDevObs = new DenseDoubleVector (firstOne.getLength (), 0.0);

// add in a bunch more
for (int i = 0; i < numTrials; i++) {
    IDoubleVector next = myRNG.getNext ();
    next.addMyselfToHim (meanObs);
    for (int j = 0; j < next.getLength (); j++) {
        stdDevObs.setItem (j, stdDevObs.getItem (j) + next.getItem (j) *
            next.getItem (j));
    }
}

System.out.format ("\nfound: ");
// divide by the number of trials to get the mean
for (int i = 0; i < meanObs.getLength (); i++) {
    meanObs.setItem (i, meanObs.getItem (i) / numTrials);
    if (i < 10) {
        System.out.format (meanObs.getItem (i) + " ");
    }
    stdDevObs.setItem (i, Math.sqrt (stdDevObs.getItem (i) / numTrials -
        meanObs.getItem (i) * meanObs.getItem (i)));}
System.out.format ("...\nexpected: ");
for (int i = 0; i < 10; i++) {
    System.out.format (expectedMean.getItem (i) + " ");
}
System.out.format ("...
");
So Add In Some Code To Print Them Out

```java
IDoubleVector firstOne = myRNG.getNext ();
DenseDoubleVector meanObs = new DenseDoubleVector (firstOne.getLength (), 0.0);
DenseDoubleVector stdDevObs = new DenseDoubleVector (firstOne.getLength (), 0.0);

// add in a bunch more
for (int i = 0; i < numTrials; i++) {
    IDoubleVector next = myRNG.getNext ();
    next.addMyselfToHim (meanObs);
    for (int j = 0; j < next.getLength (); j++) {
        stdDevObs.setItem (j, stdDevObs.getItem (j) + next.getItem (j) *
            next.getItem (j));
    }
}

System.out.format ("\nfound: ");
// divide by the number of trials to get the mean
for (int i = 0; i < meanObs.getLength (); i++) {
    meanObs.setItem (i, meanObs.getItem (i) / numTrials);
    if (i < 10) {
        System.out.format (meanObs.getItem (i) + " ");
    }
    stdDevObs.setItem (i, Math.sqrt (stdDevObs.getItem (i) / numTrials -
        meanObs.getItem (i) * meanObs.getItem (i)));
}

System.out.format ("\nexpected: ");
for (int i = 0; i < 10; i++) {
    System.out.format (expectedMean.getItem (i) + " ");
}
System.out.format ("...
expected can’t be right...Is the test code wrong?
```

Output is:

```
found: 0.0 0.0 0.8298 0.0 1.6868 0.0 2.5124 0.0 3.3072 0.0 ...
expected: 0.0 0.0 0.8359 0.8359 2.507 2.507 5.015 5.015 8.359 8.359 ...
```
Back to testMultinomialOne

int len = 100;
SparseDoubleVector probs = new SparseDoubleVector (len, 0.0);
for (int i = 0; i < len; i += 2) {
    probs.setItem (i, i);
}   
probs.normalize ();

// now, set up a distribution
IRandomGenerationAlgorithm<IDoubleVector> myRNG = new
    Multinomial (27, new MultinomialParam (1024, probs));

// and check the mean... we repeatedly double the prob vector to multiply it by 1024
DenseDoubleVector expectedMean = new DenseDoubleVector (len, 0.0);
DenseDoubleVector expectedStdDev = new DenseDoubleVector (len, 0.0);
for (int i = 0; i < len; i++) {
    expectedMean.setItem (i, probs.getItem (i) * 1024);
    expectedStdDev.setItem (i, Math.sqrt (probs.getItem (i) * 1024 *
               (1.0 - probs.getItem (i))));
}

checkMeanAndVar (myRNG, expectedMean, expectedStdDev, 5.0, 5.0,
    5000, "multinomial number one");

Let’s print out the mean...
int len = 100;
SparseDoubleVector probs = new SparseDoubleVector (len, 0.0);
for (int i = 0; i < len; i += 2) {
    probs.setItem (i, i);
}
probs.normalize ();
System.out.format ("expected right here: ");
for (int i = 0; i < 10; i++) {
    System.out.format (probs.getItem (i) * 1024 + " ");
}
System.out.format ("...");

// now, set up a distribution
IRandomGenerationStrategy<IDoubleVector> myRNG = new Multinomial (27, new MultinomialParam (1024, probs));

// and check the mean... we repeatedly double the prob vector to multiply it by 1024
DenseDoubleVector expectedMean = new DenseDoubleVector (len, 0.0);
DenseDoubleVector expectedStdDev = new DenseDoubleVector (len, 0.0);
for (int i = 0; i < len; i++) {
    expectedMean.setItem (i, probs.getItem (i) * 1024);
    expectedStdDev.setItem (i, Math.sqrt (probs.getItem (i) * 1024 * (1.0 - probs.getItem (i))));
}
checkMeanAndVar (myRNG, expectedMean, expectedStdDev, 5.0, 5.0, 5000, "multinomial number one");
```java
int len = 100;
SparseDoubleVector probs = new SparseDoubleVector(len, 0.0);
for (int i = 0; i < len; i += 2) {
    probs.setItem(i, i);
}
probs.normalize();
System.out.format("expected right here: ");
for (int i = 0; i < 10; i++) {
    System.out.format(probs.getItem(i) * 1024 + " ");
}
System.out.format("...");

// now, set up a distribution
IRandomGenerationAlgorithm<IDoubleVector> myRNG = new Multinomial(27, new MultinomialParam(1024, probs));

// and check the mean... we repeatedly double the prob vector to multiply it by 1024
DenseDoubleVector expectedMean = new DenseDoubleVector(len, 0.0);
DenseDoubleVector expectedStdDev = new DenseDoubleVector(len, 0.0);
for (int i = 0; i < len; i++) {
    expectedMean.setItem(i, probs.getItem(i) * 1024);
    expectedStdDev.setItem(i, Math.sqrt(probs.getItem(i) * 1024 * (1.0 - probs.getItem(i))));
}
checkMeanAndVar(myRNG, expectedMean, expectedStdDev, 5.0, 5.0, 5000, "multinomial number one");
```

Output is:

```
expected right here: 0.0 0.0 0.8359 0.0 1.6718 0.0 2.507 0.0 3.343 0.0 ... 
found: 0.0 0.0 0.8298 0.0 1.6868 0.0 2.5124 0.0 3.3072 0.0 ... 
expected: 0.0 0.0 0.8359 0.8359 2.507 2.507 5.015 5.015 8.359 8.359 ... 
```
int len = 100;
SparseDoubleVector probs = new SparseDoubleVector (len, 0.0);
for (int i = 0; i < len; i += 2) {
    probs.setItem (i, i);
}
probs.normalize ();
System.out.format ("expected right here: ");
for (int i = 0; i < 10; i++) {
    System.out.format (probs.getItem (i) * 1024 + " ");
}
System.out.format ("...");

// now, set up a distribution
IRandomGenerationAlgorithm<IDoubleVector> myRNG = new Multinomial (27, new MultinomialParam (1024, probs));

// and check the mean... we repeatedly double the prob vector to multiply it by 1024
DenseDoubleVector expectedMean = new DenseDoubleVector (len, 0.0);
DenseDoubleVector expectedStdDev = new DenseDoubleVector (len, 0.0);
for (int i = 0; i < len; i++) {
    expectedMean.setItem (i, probs.getItem (i) * 1024);
    expectedStdDev.setItem (i, Math.sqrt (probs.getItem (i) * 1024 * (1.0 - probs.getItem (i))));
}
checkMeanAndVar (myRNG, expectedMean, expectedStdDev, 5.0, 5.0, 5000, "multinomial number one");
Let’s See If It Does In Fact Change

```java
int len = 100;
SparseDoubleVector probs = new SparseDoubleVector (len, 0.0);
for (int i = 0; i < len; i += 2) {
    probs.setItem (i, i);
}
probs.normalize ();
System.out.format ("expected right here: ");
for (int i = 0; i < 10; i++) {
    System.out.format (probs.getItem (i) * 1024 + " ");
}
System.out.format ("...");

// now, set up a distribution
IRandomGenerationAlgorithm<IDoubleVector> myRNG = new
      Multinomial (27, new MultinomialParam (1024, probs));

// and check the mean... we repeatedly double the prob vector to multiply it by 1024
DenseDoubleVector expectedMean = new DenseDoubleVector (len, 0.0);
DenseDoubleVector expectedStdDev = new DenseDoubleVector (len, 0.0);
for (int i = 0; i < len; i++) {
    expectedMean.setItem (i, probs.getItem (i) * 1024);
    expectedStdDev.setItem (i, Math.sqrt (probs.getItem (i) * 1024 *
        (1.0 - probs.getItem (i))));
}
System.out.format ("expected mean: ");
for (int i = 0; i < 10; i++) {
    System.out.format (expectedMean.getItem (i) + " ");
}
System.out.format ("...");
```
Let’s See If It Does In Fact Change

```java
int len = 100;
SparseDoubleVector probs = new SparseDoubleVector (len, 0.0);
for (int i = 0; i < len; i += 2) {
   probs.setItem (i, i);
}
probs.normalize ();
System.out.format ("expected right here: ");
for (int i = 0; i < 10; i++) {
   System.out.format (probs.getItem (i) * 1024 + " ");
}
System.out.format ("...");

// now, set up a distribution
IRandomGenerationAlgorithm<IDoubleVector> myRNG = new Multinomial (27, new MultinomialParam (1024, probs));

// and check the mean... we repeatedly double the prob vector to multiply it by 1024
DenseDoubleVector expectedMean = new DenseDoubleVector (len, 0.0);
DenseDoubleVector expectedStdDev = new DenseDoubleVector (len, 0.0);
for (int i = 0; i < len; i++) {
   expectedMean.setItem (i, probs.getItem (i) * 1024);
   expectedStdDev.setItem (i, Math.sqrt (probs.getItem (i) * 1024 * (1.0 - probs.getItem (i))));
}
System.out.format ("expected mean: ");
for (int i = 0; i < 10; i++) {
   System.out.format (expectedMean.getItem (i) + " ");
}
System.out.format ("...");
```

Output is:

```
expected right here: 0.0 0.0 0.8359 0.0 1.6718 0.0 2.507 0.0 3.343 0.0 ...
expected mean: 0.0 0.0 0.8359 0.8359 2.507 2.507 5.015 5.015 8.359 8.359 ...
found: 0.0 0.0 0.8298 0.0 1.6868 0.0 2.5124 0.0 3.3072 0.0 ...
expected: 0.0 0.0 0.8359 0.8359 2.507 2.507 5.015 5.015 8.359 8.359 ...
```
Let’s See If It Does In Fact Change

```java
int len = 100;
SparseDoubleVector probs = new SparseDoubleVector(len, 0.0);
for (int i = 0; i < len; i += 2) {
    probs.setItem(i, i);
}
probs.normalize();
System.out.format("expected right here: ");
for (int i = 0; i < 10; i++) {
    System.out.format(probs.getItem(i) * 1024 + " ");
}
System.out.format("...");

// now, set up a distribution
IRandomGenerationAlgorithm<IDoubleVector> myRNG = new Multinomial(27, new MultinomialParam(1024, probs));

// and check the mean... we repeatedly double the prob vector to multiply it by 1024
DenseDoubleVector expectedMean = new DenseDoubleVector(len, 0.0);
DenseDoubleVector expectedStdDev = new DenseDoubleVector(len, 0.0);
for (int i = 0; i < len; i++) {
    expectedMean.setItem(i, probs.getItem(i) * 1024);
    expectedStdDev.setItem(i, Math.sqrt(probs.getItem(i) * 1024 * (1.0 - probs.getItem(i))));
}
System.out.format("\nexpected mean: ");
for (int i = 0; i < 10; i++) {
    System.out.format(expectedMean.getItem(i) + " ");
}
System.out.format("...");
```

So these two do not match up. How is this possible?
The Only Reasonable Explanation...

```java
int len = 100;
SparseDoubleVector probs = new SparseDoubleVector (len, 0.0);
for (int i = 0; i < len; i += 2) {
    probs.setItem (i, i);
}
probs.normalize ();
System.out.format ("expected right here: ");
for (int i = 0; i < 10; i++) {
    System.out.format (probs.getItem (i) * 1024 + " ");
}
System.out.format ("...");

// now, set up a distribution
IRandomGenerationAlgorithm<IDoubleVector> myRNG = new
    Multinomial (27, new MultinomialParam (1024, probs));

// and check the mean... we repeatedly double the prob vector to multiply it by 1024
DenseDoubleVector expectedMean = new DenseDoubleVector (len, 0.0);
DenseDoubleVector expectedStdDev = new DenseDoubleVector (len, 0.0);
for (int i = 0; i < len; i++) {
    expectedMean.setItem (i, probs.getItem (i) * 1024);
    expectedStdDev.setItem (i, Math.sqrt (probs.getItem (i) * 1024 *
        (1.0 - probs.getItem (i))));
}
System.out.format ("expected mean: ");
for (int i = 0; i < 10; i++) {
    System.out.format (expectedMean.getItem (i) + " ");
}
System.out.format ("...");
```

Constructor is screwing with probs
class Multinomial ...

    private IDoubleVector sums;

    public Multinomial (long seed, MultinomialParam myParams)
    {
        super(seed);
        try {
            sums = myParams.getProbs ();
            double tot = 0.0;
            for (int i = 0; i < sums.getLength (); i++) {
                tot += sums.getItem (i);
                sums.setItem (i, tot);
            }
        } catch (OutOfBoundsException e) {...}
Got It!

```java
public Multinomial (long seed, MultinomialParam myParams) {
    super(seed);
    try {
        sums = myParams.getProbs();
        double tot = 0.0;
        for (int i = 0; i < sums.getLength(); i++) {
            tot += sums.getItem(i);
            sums.setItem(i, tot);
        }
    } catch (OutOfBoundsException e) {...}
}

• Test code assumes you don’t mess with params
  — Should this have been explicitly stated?
  — Perhaps, but generally assumed you don’t change param state uness explicitly
    stated somewhere in doc that you can or will
  — Regardless, we found the bug!
  — To fix? Just allocate a new vector...
```
Questions?