A Few Remarks About Debugging in R

Norm Matloff
Dept. of Computer Science
University of California, Davis

October 13, 2009
Where I’m Coming From

My sources of bias:

▶ Early career was as a statistics professor at UCD.
Where I’m Coming From

My sources of bias:

- Early career was as a statistics professor at UCD.
- UCD CS Dept. since 1983. Main research areas: probabilistic/statistical modeling of computer systems; parallel processing; computer architecture and networks; Chinese-language computing.

(Will release my Rdsm shared-memory parallel R package to CRAN in the next couple of weeks, much improved over the α version.)

- I'm a longtime Linux fan/promoter. I use it for everything, both professional and personal.
- Have been programming since I was 17. (FORTRAN, punched cards.)
- GUIs and IDEs are nice, but:
  - I didn't grow up with it. Not a "necessity" to me like it is for most new grads today.
  - I like to use the same text editor for everything I do.
Where I’m Coming From

My sources of bias:

▶ Early career was as a statistics professor at UCD.
▶ UCD CS Dept. since 1983. Main research areas: probabilistic/statistical modeling of computer systems; parallel processing; computer architecture and networks; Chinese-language computing.
(Will release my Rdsm shared-memory parallel R package to CRAN in the next couple of weeks, much improved over the $\alpha$ version.)
Where I’m Coming From

My sources of bias:

► Early career was as a statistics professor at UCD.
► UCD CS Dept. since 1983. Main research areas: probabilistic/statistical modeling of computer systems; parallel processing; computer architecture and networks; Chinese-language computing. 
(Will release my Rdsm shared-memory parallel R package to CRAN in the next couple of weeks, much improved over the \( \alpha \) version.)
► I’m a longtime Linux fan/promoter. I use it for everything, both professional and personal.
Where I’m Coming From

My sources of bias:

► Early career was as a statistics professor at UCD.
► UCD CS Dept. since 1983. Main research areas: probabilistic/statistical modeling of computer systems; parallel processing; computer architecture and networks; Chinese-language computing. (Will release my Rdsm shared-memory parallel R package to CRAN in the next couple of weeks, much improved over the $\alpha$ version.)
► I’m a longtime Linux fan/promoter. I use it for everything, both professional and personal.
► Have been programming since I was 17.
Where I’m Coming From

My sources of bias:

▶ Early career was as a statistics professor at UCD.
▶ UCD CS Dept. since 1983. Main research areas: probabilistic/statistical modeling of computer systems; parallel processing; computer architecture and networks; Chinese-language computing. (Will release my Rdsm shared-memory parallel R package to CRAN in the next couple of weeks, much improved over the α version.)
▶ I’m a longtime Linux fan/promoter. I use it for everything, both professional and personal.
▶ Have been programming since I was 17. (FORTRAN, punched cards.)
Where I’m Coming From

My sources of bias:

▶ Early career was as a statistics professor at UCD.
▶ UCD CS Dept. since 1983. Main research areas: probabilistic/statistical modeling of computer systems; parallel processing; computer architecture and networks; Chinese-language computing. (Will release my Rdsm shared-memory parallel R package to CRAN in the next couple of weeks, much improved over the $\alpha$ version.)
▶ I’m a longtime Linux fan/promoter. I use it for everything, both professional and personal.
▶ Have been programming since I was 17. (FORTRAN, punched cards.)
▶ GUIs and IDEs are nice, but:
  ▶ I didn’t grow up with it. Not a “necessity” to me like it is for most new grads today.
Where I’m Coming From

My sources of bias:

▶ Early career was as a statistics professor at UCD.
▶ UCD CS Dept. since 1983. Main research areas: probabilistic/statistical modeling of computer systems; parallel processing; computer architecture and networks; Chinese-language computing. (Will release my Rdsm shared-memory parallel R package to CRAN in the next couple of weeks, much improved over the $\alpha$ version.)
▶ I’m a longtime Linux fan/promoter. I use it for everything, both professional and personal.
▶ Have been programming since I was 17. (FORTRAN, punched cards.)
▶ GUIs and IDEs are nice, but:
  ▶ I didn’t grow up with it. Not a “necessity” to me like it is for most new grads today.
  ▶ I like to use the same text editor for everything I do.
My Biases (cont’d.)

- Debugging is one of the most underrated software skills.
Debugging is one of the most underrated software skills. I may not think IDEs are important, but debugging tools are crucial.
My Biases (cont’d.)

- Debugging is one of the most underrated software skills. I may not think IDEs are important, but debugging tools are crucial. That’s why Pete Salzmann and I wrote our book on debugging.
Debugging is one of the most underrated software skills. I may not think IDEs are important, but debugging tools are crucial. That's why Pete Salzman and I wrote our book on debugging.

R’s debugging tools have been its weakest link.
Debugging is one of the most underrated software skills. I may not think IDEs are important, but debugging tools are crucial. That’s why Pete Salzman and I wrote our book on debugging.

R’s debugging tools have been its weakest link.

I thus am delighted that REvolution Computing is stepping into the void. I hope they or others go to open source/cross platform.
What Has Been Available?

- R’s built-in `debug()` function: non-GUI, limited capabilities but serviceable. Somewhat improved in R 2.10.0.

- `debug()`, `trace()`, `browser()`, `recovery()` etc. functions: These add finer control, e.g. conditional breakpoints, crash post mortems, etc.

- Mark Bravington’s `debug` package: GUI view of one’s source code as one traverses it. But can’t use mouse to click-and-set breakpoints, query values of variables, etc. “If it doesn’t click, you must acquit.” (Mark just informed me today he’s preparing an update to the package.)
What Has Been Available?

- R’s built-in `debug()` function: non-GUI, limited capabilities but serviceable. Somewhat improved in R 2.10.0.
- R’s built-in `trace()`, `browser()`, `recovery()` etc. functions: These add finer control, e.g. conditional breakpoints, crash post mortems, etc.
What Has Been Available?

- R’s built-in `debug()` function: non-GUI, limited capabilities but serviceable. Somewhat improved in R 2.10.0.
- R’s built-in `trace()`, `browser()`, `recovery()` etc. functions: These add finer control, e.g. conditional breakpoints, crash post mortems, etc.
- Mark Bravington’s `debug` package: GUI view of one’s source code as one traverses it. But can’t use mouse to click-and-set breakpoints, query values of variables, etc.

“If it doesn’t click, you must acquit.” (Mark just informed me today he’s preparing an update to the package.)
What Has Been Available?

- R’s built-in `debug()` function: non-GUI, limited capabilities but serviceable. Somewhat improved in R 2.10.0.
- R’s built-in `trace()`, `browser()`, `recovery()` etc. functions: These add finer control, e.g. conditional breakpoints, crash post mortems, etc.
- Mark Bravington’s `debug` package: GUI view of one’s source code as one traverses it. But can’t use mouse to click-and-set breakpoints, query values of variables, etc. “If it doesn’t click, you must acquit.”
What Has Been Available?

- R’s built-in `debug()` function: non-GUI, limited capabilities but serviceable. Somewhat improved in R 2.10.0.
- R’s built-in `trace()`, `browser()`, `recovery()` etc. functions: These add finer control, e.g. conditional breakpoints, crash post mortems, etc.
- Mark Bravington’s `debug` package: GUI view of one’s source code as one traverses it. But can’t use mouse to click-and-set breakpoints, query values of variables, etc. “If it doesn’t click, you must acquit.” (Mark just informed me today he’s preparing an update to the package.)
Desiderata for Debugging Tools

▶ Viewable breakpoints.

“It's 10 p.m. Do you where your breakpoints are?” Not in the above tools. (An example in my forthcoming R book has a kludge fix for this.)

▶ Saveable debugging sessions. So can come back the next day for more torture without setup time. :-)

Not in the above tools.

▶ User-customizable: Ability to program macros, etc. Doable in above tools via `trace()` and `browser()`.

▶ GUI. Most people other than me really want it (me too, to a large degree).

▶ Watchpoints. Run until specified variable changes value. Fairly easy to implement in an interpreted language like R.

▶ Ability to debug parallel code (next slide).
Desiderata for Debugging Tools

▶ Viewable breakpoints. “It’s 10 p.m. Do you where your breakpoints are?”

▶ Saveable debugging sessions. So can come back the next day for more torture without setup time. :-)

▶ User-customizable: Ability to program macros, etc. Doable in above tools via `trace()` and `browser()`.

▶ GUI. Most people other than me really want it (me too, to a large degree).

▶ Watchpoints. Run until specified variable changes value. Fairly easy to implement in an interpreted language like R.

▶ Ability to debug parallel code (next slide).
Desiderata for Debugging Tools

▶ Viewable breakpoints. “It’s 10 p.m. Do you where your breakpoints are?”
Not in the above tools. (An example in my forthcoming R book has a kludge fix for this.)
Desiderata for Debugging Tools

- Viewable breakpoints. “It’s 10 p.m. Do you where your breakpoints are?”
  Not in the above tools. (An example in my forthcoming R book has a kludge fix for this.)

- Saveable debugging sessions. So can come back the next day for more torture without setup time. :-)

- User-customizable: Ability to program macros, etc.
  Doable in above tools via `trace()` and `browser()`.

- GUI. Most people other than me really want it (me too, to a large degree).

- Watchpoints. Run until specified variable changes value.
  Fairly easy to implement in an interpreted language like R.

- Ability to debug parallel code (next slide).
Desiderata for Debugging Tools

- Viewable breakpoints. “It’s 10 p.m. Do you where your breakpoints are?”
  Not in the above tools. (An example in my forthcoming R book has a kludge fix for this.)
- Saveable debugging sessions. So can come back the next day for more torture without setup time. :-)
  Not in the above tools.
Desiderata for Debugging Tools

- Viewable breakpoints. “It’s 10 p.m. Do you where your breakpoints are?”
  Not in the above tools. (An example in my forthcoming R book has a kludge fix for this.)

- Saveable debugging sessions. So can come back the next day for more torture without setup time. :-)
  Not in the above tools.

- User-customizable: Ability to program macros, etc.
Desiderata for Debugging Tools

▶ Viewable breakpoints. “It’s 10 p.m. Do you where your breakpoints are?”
Not in the above tools. (An example in my forthcoming R book has a kludge fix for this.)

▶ Saveable debugging sessions. So can come back the next day for more torture without setup time. :-)  
Not in the above tools.

▶ User-customizable: Ability to program macros, etc.  
Doable in above tools via `trace()` and `browser()`.
Desiderata for Debugging Tools

▶ Viewable breakpoints. “It’s 10 p.m. Do you where your breakpoints are?”
Not in the above tools. (An example in my forthcoming R book has a kludge fix for this.)

▶ Saveable debugging sessions. So can come back the next day for more torture without setup time. :-)
Not in the above tools.

▶ User-customizable: Ability to program macros, etc.
Doable in above tools via \texttt{trace()} and \texttt{browser()}.

▶ GUI. Most people other than me really want it (me too, to a large degree).
Desiderata for Debugging Tools

- Viewable breakpoints. “It’s 10 p.m. Do you where your breakpoints are?”
  Not in the above tools. (An example in my forthcoming R book has a kludge fix for this.)
- Saveable debugging sessions. So can come back the next day for more torture without setup time. :-)
  Not in the above tools.
- User-customizable: Ability to program macros, etc.
  Doable in above tools via `trace()` and `browser()`.
- GUI. Most people other than me really want it (me too, to a large degree).
- Watchpoints. Run until specified variable changes value.
  Fairly easy to implement in an interpreted language like R.
Desiderata for Debugging Tools

- Viewable breakpoints. “It’s 10 p.m. Do you where your breakpoints are?”
  Not in the above tools. (An example in my forthcoming R book has a kludge fix for this.)

- Saveable debugging sessions. So can come back the next day for more torture without setup time. :-)
  Not in the above tools.

- User-customizable: Ability to program macros, etc.
  Doable in above tools via `trace()` and `browser()`.

- GUI. Most people other than me really want it (me too, to a large degree).

- Watchpoints. Run until specified variable changes value.
  Fairly easy to implement in an interpreted language like R.

- Ability to debug parallel code (next slide).
Debugging Parallel Code

- Debugging of parallel processes much harder than for single process.
- Not many tools out there even for C/C++. TotalView (commercial), XMPI (open source, for MPI).
- Screen footprint problem: If have n processes, that means n windows. Problem is compounded if use GUI.
- Many existing parallel R platforms make parallel debugging very difficult, due to lack of terminals for the processes. (One of my Rdsm modes allows it.)
Debugging Parallel Code

- Debugging of parallel processes much harder than for single process.
Debugging Parallel Code

- Debugging of parallel processes much harder than for single process.
- Not many tools out there even for C/C++. TotalView (commercial), XMPI (open source, for MPI).
Debugging Parallel Code

- Debugging of parallel processes much harder than for single process.
- Not many tools out there even for C/C++. TotalView (commercial), XMPI (open source, for MPI).
- Screen footprint problem: If have n processes, that means n windows. Problem is compounded if use GUI.
Debugging Parallel Code

- Debugging of parallel processes much harder than for single process.
- Not many tools out there even for C/C++. TotalView (commercial), XMPI (open source, for MPI).
- Screen footprint problem: If have n processes, that means n windows. Problem is compounded if use GUI.
- Many existing parallel R platforms make parallel debugging very difficult, due to lack of terminals for the processes. (One of my Rdsm modes allows it.)
Tips on Debugging R

- **Fundamental principles of debugging:**
  - **Principle of Confirmation:** Even though you are "sure" a certain variable has a certain value, or that a certain if-condition holds, confirm it.
  - **Start small:** Try the program on a small vector, maybe in with a scaled-down version of the program itself.
  - **Top-down approach:** When debugging f(), don't follow calls to g() at first. Check first whether the return value of g() is correct.
  - **Use binary search:** Say you have a syntax error that's baffling you. Comment-out half the code of the function, to see if the error disappears. Then comment-out half of the half that triggers the error, etc.
Tips on Debugging R

Sorry, no magic bullets.

Fundamental principles of debugging:

- Principle of Confirmation: Even though you are “sure” a certain variable has a certain value, or that a certain if-condition holds, confirm it.
- Start small: Try the program on a small vector, maybe in a scaled-down version of the program itself.
- Top-down approach: When debugging `f()` which calls `g()`, don’t follow calls to `g()` at first. Check first whether the return value of `g()` is correct.
- Use binary search: Say you have a syntax error that’s baffling you. Comment-out half the code of the function, to see if the error disappears. Then comment-out half of the half that triggers the error, etc.
Tips on Debugging R

Sorry, no magic bullets.

► Fundamental principles of debugging:

- Principle of Confirmation: Even though you are “sure” a certain variable has a certain value, or that a certain if-condition holds, confirm it.
- Start small: Try the program on a small vector, maybe in with a scaled-down version of the program itself.
- Top-down approach: When debugging f() which calls g(), don’t follow calls to g() at first. Check first whether the return value of g() is correct.
- Use binary search: Say you have a syntax error that’s baffling you. Comment-out half the code of the function, to see if the error disappears. Then comment-out half of the half that triggers the error, etc.
Tips on Debugging R

Sorry, no magic bullets.

▶ Fundamental principles of debugging:
  ▶ Principle of Confirmation:
    ▶ Even though you are "sure" a certain variable has a certain value, or that a certain if-condition holds, confirm it.
    ▶ Start small: Try the program on a small vector, maybe in with a scaled-down version of the program itself.
    ▶ Top-down approach: When debugging f() which calls g(), don't follow calls to g() at first. Check first whether the return value of g() is correct.
    ▶ Use binary search: Say you have a syntax error that's baffling you. Comment-out half the code of the function, to see if the error disappears. Then comment-out half of the half that triggers the error, etc.
Tips on Debugging R

Sorry, no magic bullets.

- Fundamental principles of debugging:
  - Principle of Confirmation: Even though you are “sure” a certain variable has a certain value, or that a certain if-condition holds, **confirm** it.
Tips on Debugging R

Sorry, no magic bullets.

- Fundamental principles of debugging:
  - Principle of Confirmation: Even though you are “sure” a certain variable has a certain value, or that a certain if-condition holds, confirm it.
  - Start small: Try the program on a small vector, maybe in with a scaled-down version of the program itself.
Tips on Debugging R

Sorry, no magic bullets.

- **Fundamental principles of debugging:**
  - Principle of Confirmation: Even though you are “sure” a certain variable has a certain value, or that a certain if-condition holds, confirm it.
  - Start small: Try the program on a small vector, maybe in with a scaled-down version of the program itself.
  - Top-down approach: When debugging \( f() \) which calls \( g() \), don’t follow calls to \( g() \) at first. Check first whether the return value of \( g() \) is correct.
Tips on Debugging R

Sorry, no magic bullets.

- Fundamental principles of debugging:
  - Principle of Confirmation: Even though you are “sure” a certain variable has a certain value, or that a certain if-condition holds, confirm it.
  - Start small: Try the program on a small vector, maybe in with a scaled-down version of the program itself.
  - Top-down approach: When debugging f() which calls g(), don’t follow calls to g() at first. Check first whether the return value of g() is correct.
  - Use binary search: Say you have a syntax error that’s baffling you. Comment-out half the code of the function, to see if the error disappears. Then comment-out half of the half that triggers the error, etc.
Tips (cont’d.)

▶ Have a boolean global variable, say `dbg`, that turns debugging on and off, and then insert breakpoints with something like

```r
if (dbg) browser()
```

▶ My aforementioned kludge may help you organize better, e.g. keep track of your breakpoints. Download from http://heather.cs.ucdavis.edu/DebugKludge.r.

▶ If you are using a terminal-less parallel R package and are forced to use print statements in lieu of a debugging tool, use `message()` instead of `print()`. (The latter may not actually print.)