shiny intro

Outline

- short intro to shiny programming by example
- description of how the current skelesim parameter-specification shiny frontend is constructed

shiny

- Interfaces R with javascript and html, but hides the details from the programmer
- Builds interactive web applications on the fly
- somewhat strange way of thinking for a traditional R programmer

shiny apps have a user interface and a server

Each is stored in a different file. Naming conventions are

- ui.R for the user interface file
- server.R for the "backend" portion

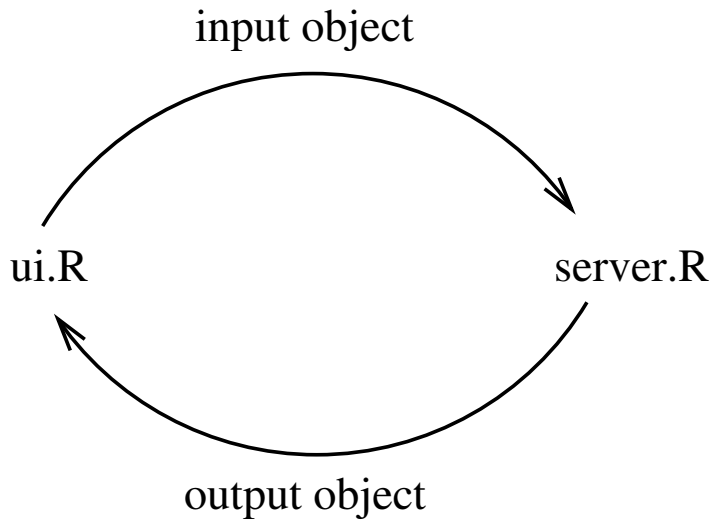
the user interface and server can exchange information

Two objects are created and used in the ui.R and server.R files.

input One of the two functions of ui.R is to build an R object called 'input' This object acts like a list so that input$a refers to an object 'a' added to this list. Though ui.R builds the input object, it cannot read the object, this only happens in server.R

output this object is created in server.R but is accessible in ui.R It is also a list-like structure that in this case contains information to output on the web-page

the user interface and server can exchange information



ui.R

The code in ui.R determines how input is solicited from the user in a web-page. It also determines how the output from the server is arranged on the page.
The function that actually creates these interfaces is called 'shinyUI()' and comprises the main part of a ui.R file.
shinyUI() is responsible for creating an input object and an output object
Useful links for setting up a UI:

- examples directory in the shiny installation on your computer. Many/Most of the examples here come from these files
- http://shiny.rstudio.com/articles/layout-guide.html

server.R

server.R sets up the environment and runs the function 'shinyServer()' Shiny server takes inputs constructed by shinyUI and stored in the 'input' object and uses them to create elements in the 'output' list. These elements are usually objects that can be rendered on a webpage.
Two important things to keep in mind:

- there are numerous functions to be used in 'shinyServer()'. Most are rendering functions
- in a rendering function, if a 'input$x' variable is referenced, it is 'reactive'

reactives

Reactive objects are the heart of shiny apps. If an object changes in shinyUI() due to user intervention, shinyServer() automatically updates the relevant outputs (ones that are constructed based on the changed input)

walkthrough simple example

use hello_01 distributed with shiny. Show how to add an element to the ui and the server

overview of shiny parameter selector

I confronted a couple of problems when setting up this (start at a) parameter selector

- lots of conditional aspects to the problem
- complex code that will be un-maintainable if in really big function calls to shinyUI() and shinyServer()
- interactive matrix inputs (surprisingly hard)

overview of shiny parameter selector

I confronted a few problems when setting up this (start at a) parameter selector

- lots of conditional aspects to the problem
  - in many cases used a function called 'renderUI()' in server. This actually build an input selector on the server side and then passes it to ui.R
  - this allows use of R boolean for conditioning
  - in a couple of places still using 'conditionalPanel()' in the shinyUI(), but this could typically be replaced.

overview of shiny parameter selector

I confronted a few problems when setting up this (start at a) parameter selector

- complex code that will be un-maintainable if in really big function calls to shinyUI() and shinyServer()
  - used 'source()' and separate code modules into different files. sort of like the C-preprocessor does
  - mostly include modules for the server-side of the equation

overview of shiny parameter selector

I confronted a couple of problems when setting up this (start at a) parameter selector

- interactive matrix inputs (surprisingly hard)
  - ended up using (someone elses) approach that writes a little javascript to take in user input in a mtrix format, but each cell is a separate variable
  - then have a function on the server that converts these multiple variables into a matrix
  - not the best solution, but no dependencies. If anybody wants to dive into javascript, this is a place where we could improve things

Design decisions

- organization of interface
- deal with multiple scenarios
- build a skelesim object and do what with it?
  - write to disk
  - deliver to the users workspace (may be kludgy and this could be a general problem with shiny)
  - run simulation and report results? If so how to include flexibility?
- the much larger list of things I haven't thought about

Bibliography