The **runcode** package

Haim Bar and HaiYing Wang
haim.bar@uconn.edu, haiying.wang@uconn.edu

June 23, 2023

Abstract

`runcode` is a LaTeX package that executes programming source codes (including all command line tools) from LaTeX, and embeds the results in the resulting pdf file. Many programming languages can be easily used and any command-line executable can be invoked when preparing the pdf file from a tex file. `runcode` is also available on CTAN.

It is recommended to use this package in the server mode together with the Python talk2stat package. Currently, the server mode supports Julia, MatLab, Python, and R. More languages will be added.

For more details and usage examples and troubleshooting, refer to the package's github repository, at `https://github.com/Ossifragus/runcode`.

1 Installation

You can simply put the runcode.sty file in the LaTeX project folder.

The server mode requires the talk2stat package. To install it from the command line, use:

```
pip3 install talk2stat
```

Note: `runcode` requires to enable the `shell-escape` option when compiling a LaTeX document.

2 Usage

2.1 Load the package:

```
\usepackage[options]{runcode}
```

Available options are:

- `cache`: use cached results.
- `fvextra`: use the fvextra package to show code.
- `julia`: start server for Julia (requires talk2stat).
- `listings`: use the listings package to show code.
- `matlab`: start server for MatLab (requires talk2stat).

- `minted`: use the minted package to show code (requires the pygments package). This is the default option.
- `nominted`: use the fvextra package instead of the minted package to show code (this does not require the pygments package, but it does not provide syntax highlights).
- `nohup`: use the `nohup` command when starting a server. When using the server-mode, some editors terminate all child processes after LaTeX compiling such as Emacs with Auctex. This option set the variable notnohup to be false, and the server will not be terminated by the parent process. **This option has to be declared before declaring any language**, e.g., `[nohup, R]` works but `[R, nohup]` does not work.
- `python`: start server for Python (requires talk2stat).
- `run`: run source code.
- `R`: start server for R (requires talk2stat).
- `stopserver`: stop the server(s) when the pdf compilation is done.

Note: If minted is used, the style of the code block is controlled through the minted package, e.g.:

```
\setminted[julia]{linenos, frame=single, bgcolor=bg, breaklines=true}
\setminted[R]{linenos, frame=single, bgcolor=lightgray, breaklines=true}
```

Similarly, fvextra and listings packages can be customized through the `\fvset` and `\lstset` commands, respectively, e.g.:

```
\fvset{fontsize=\small, linenos=true, frame=single}
\lstset{basicstyle=\large, frame=single}
```

The outputs from executing codes are displayed in tcolorbox, so the style can be customized with `\tcbset`, e.g.:

```
\tcbset{breakable,colback=red!5!white,colframe=red!75!black}
```

2.2 Basic commands:

- `\runExtCode{Arg1}{Arg2}{Arg3}[Arg4]` runs an external code.
  - `Arg1` is the executable program.
  - `Arg2` is the source file name.
  - `Arg3` is the output file name (with an empty value, the counter `codeOutput` is used).
  - `Arg4` controls whether to run the code. `Arg4` is optional with three possible values.
    - empty value or skipped: the value of the global Boolean variable `runcode` is used;
    - `run`: the code will be executed;

    - `cache` (or anything else): use cached results; if the output file does not exist, the cache option will be overridden and the code will run.

- `\showCode{Arg1}{Arg2}[Arg3][Arg4]` shows the source code, using minted (requires pygments), fvextra, or listings.
  - `Arg1` is the programming language.
  - `Arg2` is the source file name.
  - `Arg3` is the first line to show (optional with a default value 1).
  - `Arg4` is the last line to show (optional with a default value of the last line).

- `\includeOutput{Arg1}[Arg2]` is used to embed the output from executed code.
  - `Arg1` is the output file name, and it needs to have the same value as that of `Arg3` in `\runExtCode`. If an empty value is given to `Arg1`, the counter `codeOutput` is used.
  - `Arg2` is optional and it controls the type of output with a default value `vbox`.
    - `vbox` (or skipped) = verbatim in a box;
    - `tex` = pure latex;
    - `inline` = embed result in text.

- `\inln{Arg1}{Arg2}[Arg3][Arg4]` is designed for simple calculations; it runs one command (or a short batch) and displays the output within the text.
  - `Arg1` is the executable program or programming language.
  - `Arg2` is the source code.
  - `Arg3` is the output file name (optional);
  - `Arg4` is the output type and controls whether to run the code.
    - `inline` (or skipped or with empty value): embed result in text;
    - `vbox`: verbatim in a box;
    - `cache` or appending `.cache` to the argument (e.g., `vbox.cache`) AND the output file in `Arg3` exists: the cached result will be used and the code will not run.

2.3 Extended commands:

- `\runCodeIncOut{Arg1}{Arg2}[Arg3][Arg4][Arg5]` runs an external code and embeds the output. This is a combination of `\runExtCode` and `\includeOutput`.
  - `Arg1` is the executable program.
  - `Arg2` is the source file name.
  - `Arg3` (optional) controls whether to run the code. Its functionality is the same as that of `Arg4` of `\runExtCode`.

- `Arg4` (optional) is the output file name. Its functionality is the same as that of `Arg3` of `\runExtCode`.
- `Arg5` (optional) controls the type of output with a default value `vbox`. Its functionality is the same as that of `Arg3` of `\includeOutput`.

2.4 Language specific shortcuts:

Julia

- `\runJulia[Arg1]{Arg2}{Arg3}[Arg4]` runs an external Julia code file.
  - `Arg1` is optional and uses talk2stat's Julia server by default.
  - `Arg2`, `Arg3`, and `Arg4` have the same effects as those of the basic command `\runExtCode`.
- `\runJuliaIncOut[Arg1]{Arg2}[Arg3][Arg4][Arg5]` runs an external Julia code file and embeds the output.
  - `Arg1` is optional and uses talk2stat's Julia server by default.
  - `Arg2`, `Arg3`, `Arg4`, and `Arg5` have the same effects as those of the command `\runCodeIncOut`.
- `\inlnJulia[Arg1]{Arg2}[Arg3][Arg4]` runs Julia source code (`Arg2`) and displays the output in line.
  - `Arg1` is optional and uses the Julia server by default.
  - `Arg2` is the Julia source code to run. If the Julia source code is wrapped between "‘‘‘" on both sides (as in the markdown grammar), then it will be implemented directly; otherwise the code will be written to a file on the disk and then be called.
  - `Arg3` and `Arg4` have the same effects as those of the basic command `\inln`.

MatLab

- `\runMatLab[Arg1]{Arg2}{Arg3}[Arg4]` runs an external MatLab code file.
  - `Arg1` is optional and uses talk2stat's MatLab server by default.
  - `Arg2`, `Arg3`, and `Arg4` have the same effects as those of the basic command `\runExtCode`.
- `\runMatLabIncOut[Arg1]{Arg2}[Arg3][Arg4][Arg5]` runs an external MatLab code file and embeds the output.
  - `Arg1` is optional and uses talk2stat's MatLab server by default.
  - `Arg2`, `Arg3`, `Arg4`, and `Arg5` have the same effects as those of the command `\runCodeIncOut`.
- `\inlnMatLab[Arg1]{Arg2}[Arg3][Arg4]` runs MatLab source code (`Arg2`) and displays the output in line.
  - `Arg1` is optional and uses the MatLab server by default.

  - `Arg2` is the MatLab source code to run. If the MatLab source code is wrapped between "“‘" on both sides (as in the markdown grammar), then it will be implemented directly; otherwise the code will be written to a file on the disk and then be called.
  - `Arg3` and `Arg4` have the same effects as those of the basic command `\inln`.

Python

- `\runPython[Arg1]{Arg2}{Arg3}[Arg4]` runs an external Python code file.
  - `Arg1` is optional and uses talk2stat's Python server by default.
  - `Arg2`, `Arg3`, and `Arg4` have the same effects as those of the basic command `\runExtCode`.
- `\runPythonIncOut[Arg1]{Arg2}[Arg3][Arg4][Arg5]` runs an external Python code file and embeds the output.
  - `Arg1` is optional and uses talk2stat's Python server by default.
  - `Arg2`, `Arg3`, `Arg4`, and `Arg5` have the same effects as those of the command `\runCodeIncOut`.
- `\inlnPython[Arg1]{Arg2}[Arg3][Arg4]` runs Python source code (`Arg2`) and displays the output in line.
  - `Arg1` is optional and uses the Python server by default.
  - `Arg2` is the Julia source code to run. If the Python source code is wrapped between "“‘" on both sides (as in the markdown grammar), then it will be implemented directly; otherwise the code will be written to a file on the disk and then be called.
  - `Arg3` and `Arg4` have the same effects as those of the basic command `\inln`.
- `\runPythonBatch[Arg1][Arg2]` runs an external Python code file in batch mode (without a server running). Python (at least currently), unlike the other languages we use, does not have an option to save and restore a session, which means that once a Python session ends, the working environement (variable, functions) is deleted. In order to allow a batch-mode in Python, we implemented such capability. It requires the dill module, which has to be installed via `pip3 install dill`.
  - `Arg1` is the Python source file name,
  - `Arg2` is the output file name.

R

- `\runR[Arg1]{Arg2}{Arg3}[Arg4]` runs an external R code file.
  - `Arg1` is optional and uses talk2stat's R server by default.
  - `Arg2`, `Arg3`, and `Arg4` have the same effects as those of the basic command `\runExtCode`.

- `\runRIncOut[Arg1]{Arg2}[Arg3][Arg4][Arg5]` runs an external R code file and embeds the output.
  - `Arg1` is optional and uses talk2stat's R server by default.
  - `Arg2`, `Arg3`, `Arg4`, and `Arg5` have the same effects as those of the command `\runCodeIncOut`.
- `\inlnR[Arg1]{Arg2}[Arg3][Arg4]` runs R source code (`Arg2`) and displays the output in line.
  - `Arg1` is optional and uses the R server by default.
  - `Arg2` is the R source code to run. If the R source code is wrapped between "“‘" on both sides (as in the markdown grammar), then it will be implemented directly; otherwise the code will be written to a file on the disk and then be called.
  - `Arg3` and `Arg4` have the same effects as those of the basic command `\inln`.

3 Revisions

- v2.0, June 23, 2023: add `\runCodeIncOut` command that runs an external code and embeds the output and andd some language specific shortcuts; update `\runExtCode` so that if the output file does not exist, the `cache` option will be overridden and the code will run; update `\inln` so that its `Arg4` accept `cache` or appending `.cache` to use the cached result. We thank kiryph for suggesting these features.
- v1.9, June 13, 2023: update `\inln` command; the optional `Arg3` is the output file name and the optional `Arg4` is the output type.
- v1.8, January 18, 2023, add support to listings.
- v1.7, August 20, 2022: changed the tmp/ folder to generated/ in order to conform with CTAN suggestions; renamed the troubleshooting file.
- v1.6, August 10, 2022: stop only configured/running servers; a new reducedspace option - some document classes put more space after the code box; changed the default timeout of servers to 60 seconds; expanded the troubleshooting document. New examples are now available on GitHub, including how to collaborate with people who use Overleaf.
- v1.5, July 23, 2022: Removed the utf8x option when loading inputenc due to a conflict with hyperref.
- v1.4, July 18, 2022: Fixed a bug in the cache mode.
- v1.3, May 14, 2022: Removed the hard-coded minted options.
- v1.2, May 3, 2022: Added python options (server and batch).
- v1.1, April 17, 2021: Added a nohup option; improved error handling (missing code files, zero bytes in output files.)

4 Contributing

We welcome your contributions to this package by opening issues on GitHub and/or making a pull request. We also appreciate more example documents written using `runcode`.

Citing `runcode`: Haim Bar and HaiYing Wang (2021). Reproducible Science with LaTeX, *Journal of Data Science* 2021; 19, no. 1, 111-125, DOI 10.6339/21-JDS998