Computational Assignment 4: Workflows

Ryan Craft

May 30, 2024

1 Info for Running

Main functions were run on an allocation: *salloc -p work –account=courses0100 -t 3:00:00*. The two different version of the main function requested by section 3 are in two different main functions *main.nf* and *main_for.nf*, for the *xargs* and bash for loop approaches respectively.

Doing a full run of them requires that you invoke them using *nextflow -C nextflow.config run <filename> -profile hpc –params.tag=final*. Despite putting the tag *–params.tag=final*,

2 Interpretation

```
echo "seed,ncores,nsrc" > results.csv
```

The > redirects the output of *echo "seed,ncores,nsrc"* to the file *results.csv*, and overwrites the file if it was already present in the directory.

```
echo "${seed},${cores},${nsrc}" >> results.csv
```

The >> appends the value of the output of *echo "${seed},${cores},${nsrc}"* into *results.csv*. Adds the output to the next line. Does not overwrite.

```
cat ${f} | wc -l
```

The | symbol is a piping command. The output of the command on its left is used as the input for the command on the right. This case takes the lines of the file named *f* and sends them into *wc* as input.

```
files=($(ls table*.csv))
```

The difference between *$(<command>)* and *($(<command>))*. Putting *()* around groups the output of the inner command. In this case a command like *$(<command>)* returns a variable, if it returned many variables then *($(<command>))* is grouping all those outputs as one. Passing *files=($(ls table*.csv))*, is passing all of the returns for the *ls* command into the variable *files*.

```
$(ls table*.csv)
```

Returns a string value with the files in the working directory which match with the expression *table*.csv*, such as *table_2_10.csv* and others.

```
tr '_.' '␣'
```

tr is a character translator. Examines a string and replaces all instances of _. with spaces in this case.

```
awk '{print␣$2␣"␣"␣$3}'
```

Selects the second and third values in the string passed to it, separated by a space. For example after doing *echo $f | tr '_.' ' ' |* we get something like the string "1 2 test", and *awk* is pulling out 1 2, so the final string after that line is just "1 2".

```
cat ${f}
```

For a filename stored in variable *$f*, echo the lines in the file. By default it goes to the screen, but can be redirected.

```
wc -l
```

Counts the number of lines that there are in some text which it has been given.

```
echo "$(cat ${f} | wc -l)-1" | bc -l
```

cat ${f}* first echos the lines in the file with name *${f}, the output is piped to *wc -l*. *wc -l* counts the number of lines in the *cat* output that was piped to it. This expression is treated as a variable *$(cat ${f} | wc -l)-1*, with the added expression *-1* on the end. If you were to echo this command by itself you might get something like *"4-1"*. This is passed to *bc -l*, which is the inbuilt basic calculator that evaluates the string *"4-1"* and outputs *"3"*, which echo'd. In the actual code the echo is stored in *nsrc*.

3 Development

Both versions inspect the *results.csv* in the same way. The *xargs* method can be found in *main.nf*, and the shell script in it is seen in listing 3.

```
# make an empty array to hold all the values of core from the results.csv
core_array=()
# for loop reads in all the values, places every core value into the core_array
file=$(ls results.csv)
for line in $(tail -n+2 "$file")
do
        echo $line
        IFS=$','
        read seed cores nsrc <<< $line
        echo $cores
        core_array+=("$cores")
done
echo "${core_array[*]}"

# core values are repeated for every seed so we are going to make an array
# of only unique core values, so we dont have re-runs of the python.
eval uniq=($(printf "%q\n" "${core_array[@]}" | sort -u))

#now we use xargs with tricks about bash that ive learned
printf "%q\n" "${uniq[@]}" | xargs -i -P 0 python3 ../../../plot_completeness.py
                        --cores {} --infile ${file} --outfile "{}_plot_xargs.png"
```

Using a for loop, the shell script inspects the *results.csv* local file and extracts all of the core values, placing them in an array. There are multiple repeated values of core, not all of them are needed to create the plots, so the line

```
eval uniq=($(printf "%q\n" "${core_array[@]}" | sort -u))
```

is used to extract only unique values of the core array before using *xargs* to run the python plotting program in parallel. The string substitution is used to get the correct core number and to appropriately name the plot based on the core number.

The for loop version of the plotting process is found in *main_for.nf*. Similar to *main_.nf* it also uses the same shell script to examine *results.csv*, but it applies a for loop to get the different plots.

Figure 1: e

```
# make an empty array to hold all the values of core from the results.csv
core_array=()
# for loop reads in all the values, places every core value into the core_array
file=$(ls results.csv)
for line in $(tail -n+2 "$file")
do
        echo $line
        IFS=$','
        read seed cores nsrc <<< $line
        echo $cores
        core_array+=("$cores")
done
echo "${core_array[*]}"

# core values are repeated for every seed so we are going to make an array of only unique cor
# so we dont have re-runs of the python. That would be expensive!
eval uniq=($(printf "%q\n" "${core_array[@]}" | sort -u))


# Here we can exploit our for loop to run the python code on the cores we want.
for i in "${uniq[@]}"
do
        echo $i
        python3 ../../../plot_completeness.py --cores "$i" --infile ${file} --outfile "${i}_p
done
```

It uses the same array of unique core values to iterate through the different python execution parameters.

4 Execution

There was no difference seen between the cores. A plot for *core=1* for a full run is given in figure **??**

5 Analysis