Documentation of "03-import.py"

A Documentation from Jessica Ahring
j.ahring@fz-juelich.de

Forschungszentrum Jülich
JSC
ESDE
DeepRain
Jülich
February 18, 2020

This is a documentation about the import script for the *meteodatacube*, namely "03-import.py". This needs data what was already preprocessed by "02-split.py". It merges the data into 20 files for faster import and imports it into rasdaman.

name: 03-import.py
location: /mnt/rasdaman/DeepRain/playground
format: python 3
execution: 03-import.py *working_directory*
e.g.: 03-import.py *preproc/<VAR>*

The used functions will be described below.

Checking input

```
if len(sys.argv) > 1:
    if os.path.isdir(sys.argv[1]):
        sourcepath=os.path.abspath(sys.argv[1])
    else:
        print("Path did not exist\n")
        sys.exit()
else:
    print("No path is given!\n")
    sys.exit()

if os.path.isdir(ingest_dir)==False:
    print("No ingest_dir given!")
    sys.exit()

if os.path.isfile(import_script)==False:
    print("Path to import_script does not exist")
    sys.exit()

if os.access(import_script, os.X_OK)==False:
    print("Import_script is not excecutable")
    sys.exit()

missing=sourcepath+"/missing.nc"                                    # path for
    missing timesteps
if os.path.isfile(missing)==False:
    print("No file for missing values given!")
```

In this part we check the given constants ("script_dir", "ingest_dir", "import_script", "missingpath"). For importing the data into the database we need the data, the ingest file and the import script. In the first part of this script all this parts where tested and if something is missing an error message will tell this and the execution is stopped.

Create "constants" out of checked input

After checking the input there are some constants left we need to define, which depend on the input. This are the tempdir, the variable we import (which we extract out of the sourcedirectory-path) and the ingest-file, for importing the data later. The tempdir lays in the sourcedirectory and is needed as working directory for merging the timesteps.

```
tempdir=sourcepath+"/tempdir"                                    #
    workingdirectory - needed to create one datafile out of all timesteps
    given and inserting missing values if needed
variable=sourcepath.split("/")[-1]                               # get the
    variable (sourcepath should be /preproc/<VAR>)
ingest_template_file="ingest-"+variable+".json.template"         # save the
    needed ingest-file (template)
ingest_file="ingest-"+variable+".json"                           # save name
    of used ingest-file
```

As you can see it is very important, that the directory structure that was build by script "01-preproc.py" will not be changed, since we need to extract the variable-name to get the right ingest-file.

Checking again

If the script should stop the execution at any point it should be possible to just start the script again and it will clean up the old data before overwriting it.

```
checkDirectory()
```

This function will be described below.

1. Step – extract start and endtime

```
args1='ls '+sourcepath+'/time:* | head -n 1'          # get first file in
    directory
args2='ls -r '+sourcepath+'/time:* | head -n 1'       # get last file in
    directory

start_time=convertTime(str(subprocess.Popen(args1, shell=True, stdout=
    subprocess.PIPE, stderr=None).communicate()[0]))
end_time=convertTime(str(subprocess.Popen(args2, shell=True, stdout=
    subprocess.PIPE, stderr=None).communicate()[0]))

print("Import files from " + start_time.strftime("%Y-%m-%d:%H") + " to "
    + end_time.strftime("%Y-%m-%d:%H"))

actual_time=start_time
```

The script runs in one big loop over all data it should import. This data is stored in the working-directory *sourcepath*. The big loop needs a start and end-time. This is extracted

in this part.

```
args1='ls '+sourcepath+'/time:* | head -n 1'        # get first file in
    directory
```

This gets only the first file in the directory. Out of this filename we create the start-time with the "convertTime" function described below.

2. Step – Build Data to import

```
while actual_time <= end_time:
    for member in ["01","02","03","04","05","06","07","08","09","10","11"
    ,"12","13","14","15","16","17","18","19","20"]:
        moveFiles(actual_time, member)              # move all founded
    files to tempdir
        existing=[]                                 # check which
    timesteps exist (in tempdir)
        for datafile in glob.glob(tempdir+"/*.nc"):
            existing.append(os.path.basename(datafile))
        buildData(actual_time, member, existing)    # build one datafile
    for actual_time for that member (use existing and missing timesteps)
        removeData(actual_time, member)             # remove all
    datafiles that where used to build the file above
    files = sourcepath + "/" + "processed:"+actual_time.strftime("%Y%m%d%
    H")+".*.nc"    # all created files (20, one per member) is added to
    importlist
```

In this part we iterate over all possible members. For each we move the files to the tempdir we created, check which timesteps (forecasthours) exist and build the data out of this information. Afterwards the old datafiles where deleted and after all members where processed the files that should be imported where defined.

3. Step – Create ingest_file

```
    ingest_file_path=createIngest(files)
```

This function specifies the ingest-file we need in the next step.

4. Step – Import data

```
    args=import_script+" "+ingest_file_path
    termShell(args, "Import failed")
```

This part uses the ingest-file from the previous part and import the data specified in this.

5. Step – Remove ingest_file and datafiles

After the datafiles are imported, they are not needed anymore. The same is true for the ingest-file. (The datafiles where just copied in the preprocessing steps such that deleting this data just means we delete the datafiles we needed for processing. The original datafiles still remain on their old place.)

```
    os.remove(ingest_file_path)
    for datafile in glob.glob(files):
        os.remove(datafile)
```

End

After the datafiles are removed the actual_time (countervariable of the big loop) is adjusted by adding three hours. We need to add three hours since the modelruns start all three hours. After the big loop finished the user gets an output.

```
    actual_time=actual_time+timedelta(0,10800) # add three hours (go to next model run)
print(start_time.strftime("%Y-%m-%d:%H")+" until "+end_time.strftime("%Y-%m-%d:%H")+" imported")
```

Functions

```
def convertTime(path):
  """
    Extracts time information from given path.
    This is needed to get the start and end time from the sourcedirectory
    """
                      # b'/mnt/rasdaman/DeepRain/playground/preproc/t/time:YYYYMMDD-HH.FF.mEE.nc
  time=path.split(":")[-1]   # YYYYMMDD-HH.FF.mEE.nc
  time=time.split(".")[0]    # YYYYMMDD-HH.FF.mEE.nc
  return datetime.strptime(time, "%Y%m%d-%H")
```

This function is used to extract the time information out of the directory content. The extracted time is used as start and end of the big loop.

```
def termShell(args, err_message):
    """
    This function gets arguments "args" what should be executed as a shell command.
    If this does not work "err_message" is printed and the program will exit.
    """
    return_code = subprocess.call(args, shell=True)
    if return_code != 0:
        print(err_message)
        sys.exit()
```

Since this script often uses shell commands this encapsulates the functionality in this function.

```
def moveFiles(a_time, member):
    """
    Moves all files from actual_time (a_time) with member to the tempdir
    """
    os.makedirs(tempdir, exist_ok=True)
    for files in glob.glob(sourcepath+"/time:"+a_time.strftime("%Y%m%d-%H
    ")+".*.m"+member+".nc"):
        outname=tempdir+"/"+files.split(".")[-3]+".nc"
        shutil.move(files, outname)
```

In this function the tempdir will be created and afterwards all files from one specific date and one specific member are moved to this directory. There we will build the data and make sure we change no other files.

```
def buildData(a_time, member, existing):
    """
    Checks for each hour if this exists in the data. If so this is added
    to the list. If not missing data will be inserted.
    After all hours are checked the definied datafiles will be merged
    together
    """
    infiles=" "
    for hour in ["00.nc","01.nc","02.nc","03.nc","04.nc","05.nc","06.nc",
    "07.nc","08.nc","09.nc","10.nc","11.nc","12.nc","13.nc","14.nc","15.nc
    ","16.nc","17.nc","18.nc","19.nc","20.nc","21.nc","22.nc","23.nc","24.
    nc"]:
        if hour not in existing:
            args="ncap2 -s 'time@units=\"hours since "+ a_time.strftime("
    %Y-%m-%d %H:00:00") +"\"' " +missing+" "+ tempdir+"/"+hour
            termShell(args, "Creation of missing values failed")
            print("Added "+hour+" as missing value")
        infiles=infiles+tempdir+"/"+hour+" "
    outfile = sourcepath+"/"+"processed:"+a_time.strftime("%Y%m%d%H")+".m
    " + member +".nc"
    args = "cdo -mergetime " + infiles + " " + outfile
    termShell(args, "Failed merging timesteps")
```

The buildData function works in the tempdir. This directory contains all data that could be found for that given variable, for this specific time and for that specific ensemble member. For each hour (forecasthour) we check if this exists in the data we have. If so this will be used, otherwise we import missing values. This where adjusted with ncap2 so that this fits in the rest data. After all hours where processed the datafiles where merged together such that the resulting file consists of 25 forecasthours (0-24).

```
def removeData(a_time, member):
    """
    All datafiles of the given time (a_time) with the given member will
    be removed
    Afterwards the tempfile will be removed
    This is done after the datafiles where merged together
```

```
    """
    for datafile in glob.glob(sourcepath+"/time:"+a_time.strftime("%Y%m%d
    -%H")+".*.m"+member+".nc"):
        os.remove(datafile)
    shutil.rmtree(tempdir)
    print("Merging        ...ok")
```

After the datafiles containing one forecasthour where merged to one big datafile, the old ones can be removed and the tempdir deleted.

```
def createIngest(files):
    """
    First the ingest-template file will be copied.
    The copy will be out ingest-file. Therefore the placeholder "
    FILE_PATH" must be replaced.
    We use the parameter "files" to replace the placeholder.
    """
    shutil.copyfile(ingest_dir+"/"+ingest_template_file, sourcepath+"/"+
    ingest_file)
    ingest_file_path=glob.glob(sourcepath+"/"+ingest_file)[0]
    args="sed -i -e 's@FILE_PATH@"+files+"@g' '"+ingest_file_path+"'"
    termShell(args, "Replacement failed")
    return ingest_file_path
```

This function replaces the placeholder "FILE_PATH" in the ingest file with the files that should be imported. Afterwards the path to this ingest-file is returned.

```
def checkTime(processed_files):
    a_time = datetime.strptime(processed_files[0].split(":")[-1].split(".
    ")[0], "%Y%m%d%H")
    for datfile in processed_files:
        if a_time != datetime.strptime(datfile.split(":")[-1].split(".")
    [0], "%Y%m%d%H"):
            print("The 'processed:...' files do not belong to the same
    time. Manual check needed")
            sys.exit()
    return a_time
```

This function is used by the checkDirectory function. It checkes the time of all datafiles that start with "processed:...". This means, that the script stopped the execution before and we have to check if we are able to import the data automatically or if the user has to do it by hand.

```
def member(datafile):
    member = datafile.split("m")[-1].split(".")[0]
    return member
```

This function just returns the ensemble member of the given file.

```
def checkFiles(a_time, processed_files):
    members=[]
    for datafile in processed_files:
        members.append(member(datafile))
```

```
    for datafile in glob.glob(sourcepath+"/time:"+a_time.strftime("%Y%m%d
    -%H")+"*.nc"):
        if member(datafile) in members:
            os.remove(datafile)
    return members
```

This function checks if for the "processed:..."- files the original "time:..." files still exist. If so they will be removed.

```
def checkImport(processed_files):
    # TODO
    return
```

This function still needs to be written.

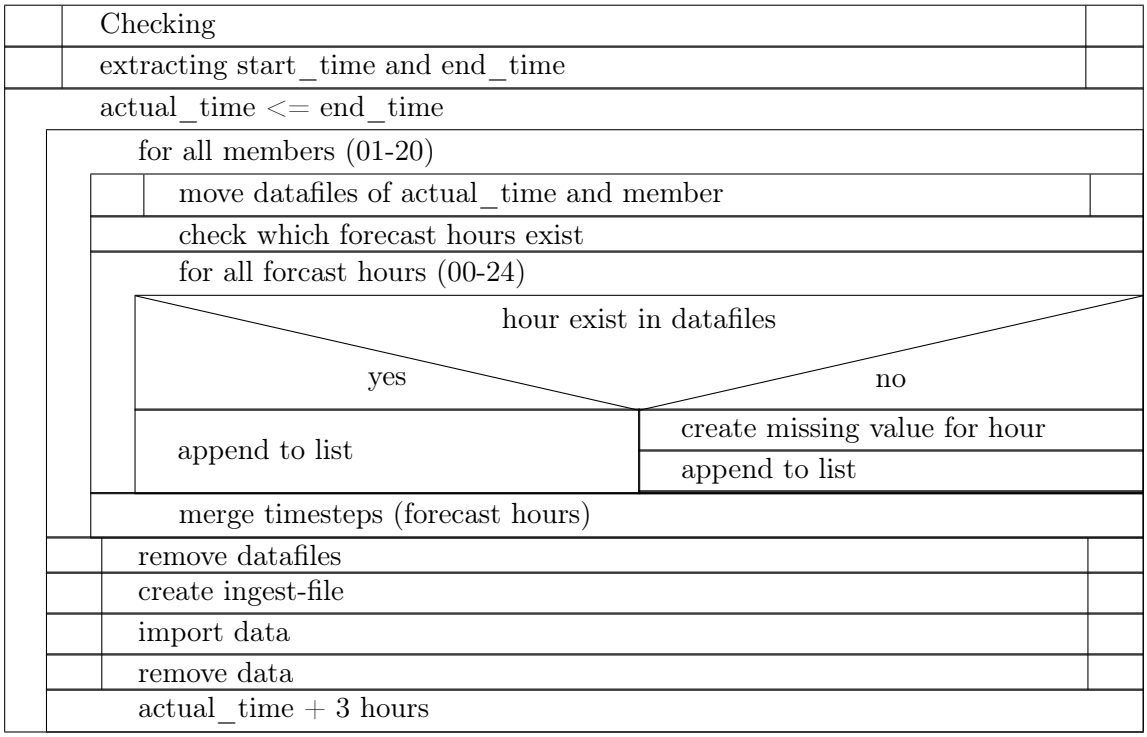
```
def checkDirectory():
    """
    This function checks if this script was executed before but stopped
    its execution.
    This can be seen, if one or more "processed:..." datafiles exist in
    the directory.
    It is possible that this datafiles where already imported but not
    removed (not really likely)
    The more realistic option is that the script broke while merging the
    timesteps (less than 20 files in directoy)
    or that it broke while importing the datafiles (exactly 20 files in
    directory).
    If it are more than 20 files something went wrong, since every time
    20 of this files exist (not more)
    """
    processed_files = glob.glob(sourcepath+"/processed:*")
    if len(processed_files) == 0:
        return
    elif len(processed_files) > 20:
        print("First clean up directory! To many files for automatic
    import.")
        sys.exit()
    # At this point we have either 20 or less files that where already
    processed. But we need to find out if they belong together
    a_time = checkTime(processed_files)
    members = checkFiles(a_time, processed_files)

    if len(processed_files) < 20:
        for member in ["01","02","03","04","05","06","07","08","09","10",
    "11","12","13","14","15","16","17","18","19","20"]:
            if member not in members:
                moveFiles(a_time, member)
                existing=[]
                for datafile in glob.glob(tempdir+"/*.nc"):
                    existing.append(os.path.basename(datafile))
                buildData(a_time, member, existing)
                removeData(a_time, member)

    checkImport(processed_files)
```

```
31      processed_files = glob.glob(sourcepath+"/processed:*")
32      if len(processed_files) >0:
33          files = sourcepath + "/" + "processed:*"
34          ingest_file_path=createIngest(files)
35          args=import_script+" "+ingest_file_path
36          termShell(args, "Import failed")
37          os.remove(ingest_file_path)
38          for datafile in glob.glob(files):
39              os.remove(datafile)
```

If the script stops its execution there is a high probability, that some files "processed:..." still remain in the directory. This could be the case because the script fails in Step 2 where the datafiles where created. In this case it could be possible that less than 20 files where processed and the rest first needs to be processed, before the data could be imported. In that case it is also possible, that the original files still remain in the directory and need to be deleted, since they would "destroy" the loop in the next step. (Less than 20 ensemble members so the script would fill up with missing values and overwrite the right data!).
An other case is, that the script stopped while importing the data. In this case all 20 files exist and need to be imported before the script continues the normal execution.
The last case is, that the data was imported but not deleted. Then the data needs to be deleted and the script can continue the normal execution.

So if no "processed:..." files exist, nothing needs to be done. If it are more than 20 the user needs to check.
In line 17 we check whether the files all have the same timevalue and we delete all corresponding "time:..." files and store all members that have an already processed datafile. If less than 20 files exist we need to process the rest like in the normal execution of the script. Afterwards we import and remove the data.
After this function finishes the normal execution of the script can start.

Summary


Checking
extracting start_time and end_time
actual_time <= end_time
for all members (01-20)
move datafiles of actual_time and member
check which forecast hours exist
for all forcast hours (00-24)
hour exist in datafiles
yes
no
append to list
create missing value for hour
append to list
merge timesteps (forecast hours)
remove datafiles
create ingest-file
import data
remove data
actual_time + 3 hours