runana Documentation

Jens Svensmark

Sep 09, 2021

Contents:

Utility library for running programs and analysing their results.

Useful for convergence testing. Integrates well with Fortran programs.

Documentation: http://runana.readthedocs.org/en/latest/

CHAPTER 1

Installation

`runana` can be installed from pypi:

```
$ pip install runana
```

The latest version of `runana` can be installed from source:

```
$ git clone https://github.com/jenssss/runana.git
$ cd runana
$ python setup.py install
```

Users without install privileges can append the `--user` flag to `setup.py`:

```
$ python setup.py install --user
```

CHAPTER 2

Example usage

A number of examples are included in the `examples` directory of the source code. The subfolder `f90nml` uses configuration files in the fortran namelist format, while `upname` uses a configuration name format in which the names and values of variables are given on consequtive lines with entries seperated by white space.

Here follows one of the examples from the `f90nml` folder. A simple program for performing a numerical integration is given in `examples/f90nml/integrate_test.py`. The main content of this file is:

```
#!/usr/bin/env python
from sys import argv
import numpy as np
import f90nml

config = f90nml.read(argv[1])

npoints = config['nlIntegrate']['npoints']

x = np.linspace(0, 2, npoints)
y = 10*x**2

I = np.trapz(y, x)

print('Integral of 10*x**2 from 0 to 2: ', I)
```

The program can be configured through a namelist configuration, which should be given as the first argument when calling the program `./intergrate_test.py config.nml`. An example of such a configuration is located at `examples/f90nml/config.nml` and contains entries of the form:

```
&nlGroup
  var = 1
&end

&nlIntegrate
  npoints = 10
&end
```

We want to run this program for a number of different values of the `npoints` parameter, and compare the results. For this we can use `runana`. The file `examples/f90nml/run_integrate.py` contains a script showing how this can be run:

```
from os import path, getcwd
from runana.run import execute, print_time, generate_list

def setup_programs():
    programs = ['integrate_test.py',]
    programs = [path.join(getcwd(), program) for program in programs]
    return programs

def setup_replacers():
    nvar_values = 10
    chain_iters = {('nlIntegrate', 'npoints'): generate_list(
        start=10, incr=10, incr_func='add', nvalues=nvar_values),
    }
    return chain_iters

input_file = 'config.nml'

chain_iters = setup_replacers()

scratch_base = path.expanduser('~/test_run/runana/integrate_test')

programs = setup_programs()

print('Running in ', scratch_base)

with print_time():
    execute(programs, input_file, scratch_base,
            chain_iters=chain_iters)
```

Running this script will run the integration program with 10 values of the `npoints` parameter in increments of 10 starting from 10. The results of the calculations will be stored in `~/test_run/runana/integrate_test`, specified in the `scratch_base` variable. For each parameter, a seperate run of the program will be performed, and the results stored in separate subdirectories of `~/test_run/runana/integrate_test`. This script can be run by running `python run_integrate.py` in the `examples/f90nml/` directory.

Finally, the results can be analyzed using the script in `examples/f90nml/analyse_integrate.py`, which contains:

```
from os import path
from runana import analyse
from runana import analyse_pandas
from runana import read_numbers

workdir = path.expanduser('~/test_run/runana/integrate_test')

params_to_dirs = analyse.read_input_files(workdir)

params_to_dirs.diff()

panda_data = analyse_pandas.make_a_seq_panda(params_to_dirs)

read_var = analyse.make_collector_function(
      workdir,
      read_numbers.read_last_number_from_file,
```

(continues on next page)

(continued from previous page)

```
      fname="integrate_test.py.stdout",
      pattern="Integral",
  )
panda_var = panda_data.applymap(read_var)
print("Values of integral")
print(panda_var)

panda_conv = panda_var.calc_convergence()
print("Estimated difference between current and fully converged value")
print(panda_conv)
param_panda = panda_data.applymap(
      analyse_pandas.return_dict_element(params_to_dirs)
      )
panda_var.plot_("plot_test_integral_var.pdf", param_panda=param_panda)
panda_conv.plot_("plot_test_integral_conv.pdf", logy=True, param_panda=param_panda)
```

Running this script should print out:

```
Values of integral:
                                0
NumParam NumParamValue
npoints  10.0           26.831276
         20.0           26.703601
         30.0           26.682521
         40.0           26.675433
         50.0           26.672220
         60.0           26.670497
         70.0           26.669467
         80.0           26.668803
         90.0           26.668350
         100.0          26.668027

Estimated difference between current and fully converged value:
                          0_conv
NumParam NumParamValue
npoints  10.0                NaN
         20.0           0.009562
         30.0           0.002370
         40.0           0.001063
         50.0           0.000602
         60.0           0.000388
         70.0           0.000270
         80.0           0.000199
         90.0           0.000153
         100.0          0.000121
```

The script collects the values calculated by the integration program and puts them into a pandas `DataFrame`, indexed by the value of the varying numerical parameter. It also calculates an estimate for how well converged the calculation is. Finally the script plots these values to the files `plot_test_integral_var.pdf` and `plot_test_integral_conv.pdf`.

CHAPTER 3

Similar software

https://github.com/ioam/lancet

CHAPTER 4

Welcome to runana's documentation!

4.1 Run API

runana.input_file_handling.**filter_inp_file_upname**(*inp_file_in*, *inp_file_out*, *replace_with_these*)

Replaces elements in *inp_file_in* and places the result in *inp_file_out*

replace_with_these is a dict with entries of the form {'Name of parameter':*value to replace with*}

This version replaces entries that is one line below a string matching *Name of parameter*, in the same position as the string

runana.input_file_handling.**filter_inp_file_f90nml**(*inp_file_in*, *inp_file_out*, *replace_with_these*)

Replaces elements in *inp_file_in* and places the result in *inp_file_out*

replace_with_these is a dict with entries of the form {'Name of parameter':*value to replace with*}

This version works on namelist files using the f90nml package

runana.input_file_handling.**INP_FILE_FILTERS = {'f90nml': <function filter_inp_file_f90nml>**

dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs** in the keyword argument list. For example: dict(one=1, two=2)

4.1.1 Extra functions

4.1.2 Automated convergence testing

4.2 Analyse API

4.2.1 Get data

Get parameters

runana.input_file_handling.**read_input_files_f90nml**(*patterns=['*.nml'], read_one_file=<function read_and_flatten_namelist>*)

Read the all files matching *patterns* with `f90nml.read()`

The namelists are flattened and supersetted, the resulting dict is returned

runana.input_file_handling.**read_input_files_upname**(*patterns=['*.inp']*)

Read the all files matching *patterns* with *read_upname_file()*

The data from all the files is supersetted and the resulting dict is returned

runana.input_file_handling.**read_input_files_positional**(*patterns=['*.inp']*)

Read the all files matching *patterns* with *read_positional_file()*

The data from all the files is supersetted and the resulting dict is returned

runana.input_file_handling.**read_upname_file**(*filename*)

Reads file in upname format

In this format the names of variables are given on a line, with the values on the following line. Each variable/name should be seperated by at least two whitespaces or a tab. Each set of names/values lines should be seperated by at least one blank line

runana.input_file_handling.**read_positional_file**(*filename*)

Reads input file

A dictionary is returned, the keys of which are the tuples *(iline,iword)* with the line and word index of the value. Each variable/name should be seperated by at least two whitespaces, a comma or a tab.

Get other stuff

4.2.2 Analyse with pandas API

This module requires the `pandas` package

4.2.3 Read numbers API

4.2.4 Matplotlib managers

Context managers for using `matplotlib`. Use these together with pythons *with* statement

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index