# 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.run.**execute**(*programs*, *input_file*, *dirs*, *chain_iters={}*, *product_iters={}*, *co_iters={}*, *just_replace={}*, *filter_func='f90nml'*, *use_stdin=False*, *calc_all=<function calc_all>*, ***kwargs*)

Run sequence of programs with different parameters defined by iters.

**Parameters**

- **programs** (*list*) – List of strings with names of programs. Should contain absolute paths. Could alternately contain functions
- **input_file** (*str*) – Input file
- **dirs** (*str or runana.run.Dirs*) – Base directory in which programs will be run
- **chain_iters** (*dict*) – Entries of the form {'Name of parameter':[*values to replace with*]}
- **product_iters** (*dict*) – Like *chain_iters*, but runs all combinations
- **co_iters** (*dict*) – Runs with several parameters changing simultanously
- **just_replace** (*dict*) – Entries of the form {'Name of parameter':*value to replace with*}
- **filter_func** (*str*) – Which filter function to use. Options are listed as keys in the INPUT_FILE_FILTERS dictionary
- **use_stdin** (*bool*) – send in the content of the filtered input file through stdin rather passing the name of the input file as the first command line argument
- **calc_all** (*func*) – Hook for the parallel decorator, please ignore this argument

runana.run.**execute_lock_par**(*lock*, *parallel*, **args*, ***kwargs*)

Convenience function for running execute with a lock and/or in parallel

runana.run.**generate_seq**(*start*, *incr*, *nvalues=0*, *incr_func=<built-in function add>*)
Iterator that returns a sequence of numbers

**Parameters** **incr_func** (*func or str*) – function used to increment the return value. Can be one of the strings 'add', 'mul' or 'div'

runana.run.**generate_list**(**args*, ***kwargs*)
Wrap of generate_seq that returns a list instead of an iterator

**class** runana.run.**Dirs**(*scratch_base, local_scratch_base=None, copy_2_scratch=['*.txt', '*.nml', '*.stdout', '*.dat']*)
Container class for names of directories

**Parameters**

- **scratch_base** (*str*) – Directory prefix
- **local_scratch_base** (*str*) – Prefix for directory in which programs are run. If *None* then *scratch_base* is used
- **copy_2_scratch** (*list*) – List of strings that are globbed and copied from the local scratch directory to scratch directory

runana.run.**print_time**(*file_=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
Contextmanager that prints how much time was spent in it

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>**
Available input file filter functions

### 4.1.1 Extra functions

runana.run.**common_start**(*chain_iters*, *just_replace*)
Returns modified *chain_iters* and *just_replace* such that the calculations will start at the first value of each variable in chain_iter

runana.run.**run_core**(*programs*, *inp_file_relative*, *use_stdin=False*, *use_inp_file=True*, *add_temp_ignore_file=True*)

### 4.1.2 Automated convergence testing

**class** runana.run.**ConvCrit**(*data_read*, *eps=0.001*, *conv_func=<function rel_err_rel_var>*, *itermax=10*, *common_start=False*)
Contains information on how to check for convergence.

**Parameters**

- **data_read** (*func*) – Function which will be executed in the directory in which the programs was run. It should return the observable in terms of which convergence is sought
- **eps** (*float*) – Desired precision
- **conv_funv** (*func*) – Function that calculates convergence criterion. It should take 4 arguments, *f(O1,O2,x1,x2)*, where *x1* and *x2* are the values of the numerical parameter at the current and previous calculation and *O1* and *O2* are the corresponding observable values

runana.run.**rel_err_rel_var**(*O1*, *O2*, *x1*, *x2*)
Estimate of relative error *abs(x2/(x2-x1)*(O1-O2)/O2)*

runana.run.**auto_conv**(*programs*, *inp_file*, *dirs*, *conv_crit*, *chain_iters*, *product_iters={}*, *co_iters={}*, *just_replace={}*, *auto_converge_var=<function auto_converge_var>*, *auto_conv_sub=<function auto_conv_sub>*)
Run programs until converged or chain_iters is exhausted.

**Parameters**

- **programs** (*list*) – List of strings with names of programs. Should contain absolute paths. Could alternately contain functions
- **input_file** (*str*) – Input file
- **dirs** (*str or runana.run.Dirs*) – Base directory in which programs will be run
- **conv_crit** (*runana.run.ConvCrit*) – Object specifying type of convergence
- **chain_iters** (*dict*) – Entries of the form {'Name of parameter':[*values to replace with*]}
- **product_iters** (*dict*) – Like *chain_iters*, but runs all combinations
- **co_iters** (*dict*) – Runs with several parameters changing simultanously
- **use_stdin** (*bool*) – send in the content of the filtered input file through stdin rather passing the name of the input file as the first command line argument

runana.run.**auto_conv_rerun**(*fun*)
Decorator for rerunning *auto_conv()*, until convergence is achieved in the first two calculations for each parameter. This is useful for cases where parameters are strongly correlated

## 4.2 Analyse API

### 4.2.1 Get data

#### Get parameters

runana.analyse.**read_input_files**(*workdir*, *indices=[]*, *read_func=<function read_input_files_f90nml>*)
Read information from input files.

Recursively searches through all subdirectories of *workdir*. *read_func* is run in any directory containing a file named 'hostname.txt', and the result is stored in a *ParamDict*, with the path in tuple-form as key. This *ParamDict* is returned.

If *indices* is given as a non-empty list, the indices in this argument will be used instead of recursive search

Subdirectories of a directory with a 'hostname.txt' file are not searched.

**class** runana.analyse.**ParamDict**
Dictionary that holds dictionaries of parameters

ParamDict.**diff**()
Call *dictdiff()* on *ParamDict* object

ParamDict.**unpack_list_values**()
Takes any numerical parameter value in *ParamDict* object that is a list and packs it into individual slots with name numparam_name + idx

Works in-place

**class** runana.analyse.**ChangedParams**(*param_dicts*, **args*, ***kwargs*)
Parameters that changed. Derived from dict

**Parameters** **param_dicts** (*dict*) – Dictionary containing dictionaries of parameters, in the form returned from e.g. collect_from_all()

ChangedParams.**groupby_varname**()
Groups elements according to the name of the variable.

Returns a dictionary containing the values of the variables, and another with all the pairs of runs

**class** runana.analyse.**Seqs**(*param_dicts*, **args*, ***kwargs*)
Seqeunces of related runs

**Parameters** **param_dicts** (*dict*) – Dictionary containing dictionaries of parameters, in the form returned from e.g. collect_from_all()

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.

runana.analyse.**groupby_n_var_params**(*dict_w_params*, *n_var_params=1*)
Convenience function for finding sets of data for which *n_var_params* parameters are changing

#### Get other stuff

runana.analyse.**make_collector_function**(*workdir*, *read_func*, **args*, ***kwargs*)
Returns a function that runs *read_func(*args,**kwargs)* in the directory that is the join of *workdir* and the argument to the function

runana.analyse.**prepend_dir**(*workdir*)
Returns a function that takes a tuple of directories and returns the combination of those into a path, with *workdir* prepended

runana.analyse.**read_from_dir**(*read_func*, *workdir*)
Composes *read_func* with *prepend_dir(workdir)*

runana.analyse.**dictdiff**(*alldicts*)
In-place removes all key:value pairs that are shared across all dicts

**Parameters** **alldicts** (*dict*) – a dictionary containing dictionaries

### 4.2.2 Analyse with pandas API

This module requires the pandas package

**class** runana.analyse_pandas.**SeqsDataFrame**(*data=None*, *index: Axes | None = None*, *columns: Axes | None = None*, *dtype: Dtype | None = None*, *copy: bool | None = None*)

SeqsDataFrame.**import_from_seq**(*seqsnew*, *varvals*, *inplace=False*)
Converts the seqs object into a SeqsDataFrame

SeqsDataFrame.**calc_convergence**()
Calculate an estimate for relative convergence error.

Calculates *(O2-O1)/O2*x2/(x2-x1)* where *O* are values and *x* are numerical parameters, which is an estimate of the difference between the value calculated at the given numerical parameter and the "true" value of the fully converged limit.

All numerical parameter values have to be scalar and numeric.

Returns a new *SeqsDataFrame*.

SeqsDataFrame.**plot_**(*outfile*, *logx=False*, *logy=False*, *grid=False*, *param_panda=None*)
Requires numpy and matplotlib

runana.analyse_pandas.**return_dict_element**(*dict_*, *error=<class 'KeyError'>*)
Returns a function that returns *dict_[arg]*, while ignoring *error*

runana.analyse_pandas.**make_a_seq_panda**(*dict_w_params*)
Convenience function for finding sequences of data, and putting them in a Pandas structure

### 4.2.3 Read numbers API

runana.read_numbers.**read_number_from_file**(*fname*, *inumber*, *pattern=''*)

runana.read_numbers.**read_last_number_from_file**(*fname*, *pattern=''*)

runana.read_numbers.**read_column_from_file**(*fname*, *icolumn*, *pattern=''*)

runana.read_numbers.**read_file_one_block**(*fname*)

runana.read_numbers.**read_file_sev_blocks**(*fname*)

### 4.2.4 Matplotlib managers

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

**class** runana.matplotlib_managers.**plot_manager**(*outfile*, *print_outfile=True*, **args*, ***kwargs*)

Creates pdf file using `matplotlib.backends.backend_pdf.PdfPages()` and returns the handle to this pdf.

**args* and **kwargs* are passed to *PdfPages*

**class** runana.matplotlib_managers.**single_fig_manager**(*pp*, **args*, ***kwargs*)

Create a `matplotlib.figure.Figure`

*args* and *kwargs* are passed to the `Figure` constructor

**Parameters** **pp** (*matplotlib.backends.backend_pdf.PdfPages*) – handle that has a savefig method

**class** runana.matplotlib_managers.**single_ax_manager**(*pp*, **args*, ***kwargs*)

Create an axis with a single subplot in a *Figure* object.

Subclassed from *single_fig_manager*, and all the arguments are the same

runana.matplotlib_managers.**plot_ax_manager**(*outfile*, **args*, ***kwargs*)

Create a pdf with name *outfile* containing one plot

CHAPTER 5

# Indices and tables

- genindex
- modindex
- search

# Index

## S



## U