pyfan Documentation

Release 0.1.48

Fan Wang

Dec 20, 2020

CONTENTS

`pyfan` is a work-in-progress package consisting of various python support functions for various research related-tasks. This is a part of the pyfan repository. Materials are gathered from various projects in which python code is used for research and paper-administrative tasks. This package is to make it easier to reuse functions created for one project in another project.

There is an associated repository that contains a variety of python examples Py4Econ.

CHAPTER

ONE

GET STARTED

For a list of functions and classes available in `pyecon`, please have a look at our *API Reference*.

1.1 Installation

1.1.1 Using github

```
pip uninstall pyfan -y
pip install git+https://github.com/fanwangecon/pyfan.git#egg=pyfan
pip show pyfan
```

1.1.2 Using PyPI

```
pip uninstall pyfan -y
pip install pyfan
pip show pyfan
```

1.1.3 Other requirements

`pyfan` builds on (and hence depends on) numpy`` and `scipy` libraries other packages shown in the toml file

This page provides the API reference for package. Modules and functions are listed below in different sections.

CHAPTER

TWO

1 DATA STRUCTURES

Various data structures.

2.1 1.1 Array

Functions to manipulate numpy arrays and other structures.

pyfan.amto.array.geomspace	Created on May 24, 2018
pyfan.amto.array.gridminmax	Created on Nov 27, 2017
pyfan.amto.array.mesh	Created on Nov 26, 2017
pyfan.amto.array.scalararray	Created on Dec 2, 2017

2.1.1 pyfan.amto.array.geomspace

Created on May 24, 2018

@author: fan

To have a better grid denser at the beginning

Functions

gen_geom_grid(start, stop, num, geom_ratio, a)	Specify geom_ratio, the z below:
grid_to_geom(choice_grid, choice_grid_max, ...)	the code now is under the assumption that initial start and end were 0 and 1
grid_to_geom_short(choice_grid, ...)	
grid_to_geom_short_core(choice_grid, a, ...)	
tester([a, b, max_power])	1. 1 to 51, geomspace
tester_plus1([a, b, max_power, adjust])	to accomndate zero,

pyfan.amto.array.geomspace.gen_geom_grid

pyfan.amto.array.geomspace.**gen_geom_grid**(*start*, *stop*, *num*, *geom_ratio*, *a*)

Specify geom_ratio, the z below: a*z^0=a a*z^1 a*z^2 ... ... a*z^49=b

Then generate the grid points that is consistent with the geom_ratio

Parameters

start: float same as in linspace

stop: float same as in linspace

num: int same as in linspace

geom_ratio: float z value below kind of except for rescaling

pyfan.amto.array.geomspace.grid_to_geom

pyfan.amto.array.geomspace.**grid_to_geom**(*choice_grid*, *choice_grid_max*, *choice_grid_min*, *start*, *stop*, *num*, *geom_ratio*, *a*)

the code now is under the assumption that initial start and end were 0 and 1

Given geom_grid results, how do we go back to actual data grid. So for interpolation. interpolate not on actual K and B scales, but on any even grid, as long as the grid count is right.

interp_K_grid = np.linspace(0,1,n)

but then there is a vector of actual choices kn_vec, how to map kn_vec to interp_K_grid?

Parameters

choice_grid: this is the choice grid, on the actual choice scale

start: float from gen_geom_grid

stop: float from gen_geom_grid

num: int from gen_geom_grid

geom_ratio: float from gen_geom_grid

pyfan.amto.array.geomspace.grid_to_geom_short

pyfan.amto.array.geomspace.**grid_to_geom_short**(*choice_grid*, *choice_grid_max*, *choice_grid_min*, *start*, *stop*, *num*, *geom_ratio*, *a*)

pyfan.amto.array.geomspace.grid_to_geom_short_core

pyfan.amto.array.geomspace.**grid_to_geom_short_core**(*choice_grid*, *a*, *scaler*, *displacement*, *multiplier*, *geom_ratio*)

pyfan.amto.array.geomspace.tester

pyfan.amto.array.geomspace.**tester**(*a=1*, *b=51*, *max_power=49*)

1. 1 to 51, geomspace

pyfan.amto.array.geomspace.tester_plus1

pyfan.amto.array.geomspace.**tester_plus1**(*a=0*, *b=50*, *max_power=49*, *adjust=1*)

to accomndate zero,

2.1.2 pyfan.amto.array.gridminmax

Created on Nov 27, 2017

@author: fan

Functions

random_vector_mean_sd(mean, sd, grid_count)	
random_vector_min_max(minval, maxval, grid_count)	
three_vec_grids(vara_min, vara_max, vara_grid)	Grid for VFI Temporary code, so that I can deal with minimal school hour.

pyfan.amto.array.gridminmax.random_vector_mean_sd

pyfan.amto.array.gridminmax.**random_vector_mean_sd**(*mean*, *sd*, *grid_count*, *gridtype='grid'*, *seed=382*)

pyfan.amto.array.gridminmax.random_vector_min_max

pyfan.amto.array.gridminmax.**random_vector_min_max**(*minval*, *maxval*, *grid_count*)

pyfan.amto.array.gridminmax.three_vec_grids

pyfan.amto.array.gridminmax.**three_vec_grids**(*vara_min*, *vara_max*, *vara_grid*, *vara_grid_add=None*, *varb_min=None*, *varb_max=None*, *varb_grid=None*, *varb_grid_add=None*, *varc_min=None*, *varc_max=None*, *varc_grid=None*, *varc_grid_add=None*, *gridtype='grid'*, *tomesh=False*, *return_joint=False*, *return_single_col=False*, *seed=999*)

Grid for VFI Temporary code, so that I can deal with minimal school hour. should be deleted in the future. and combined with the method above

2.1.3 pyfan.amto.array.mesh

Created on Nov 26, 2017

@author: fan

Most type of state grid generation: Given N Vectors,

Functions

check_length(mat)	
multipl_mat_mesh(mat_one, mat_two[, ...])	
three_mat_mesh(mat_one, mat_two, mat_three)	**Parameters**
two_mat_mesh(mat_one, mat_two[, ...])	**Parameters**

pyfan.amto.array.mesh.check_length

pyfan.amto.array.mesh.**check_length**(*mat*)

pyfan.amto.array.mesh.multipl_mat_mesh

pyfan.amto.array.mesh.**multipl_mat_mesh**(*mat_one*, *mat_two*, *mat_three=None*, *mat_four=None*, *mat_five=None*, *mat_six=None*)

pyfan.amto.array.mesh.three_mat_mesh

pyfan.amto.array.mesh.**three_mat_mesh**(*mat_one*, *mat_two*, *mat_three*, *return_joint=False*, *return_single_col=False*)

Parameters

return_single_col: boolean mat_one and mat_two are single vector, shape them into 2d array with 1 column, rather than 1d. If not, could cause multiplication problems when we have both 1 column 2d array and single column 1d array in the same formula. But this can not always to set to True, hence default is actually false, because this function could take as input a matrix for mat_one, in that case, already 2d array.

pyfan.amto.array.mesh.two_mat_mesh

pyfan.amto.array.mesh.**two_mat_mesh**(*mat_one*, *mat_two*, *return_joint=False*, *return_single_col=False*)

Parameters

return_single_col: boolean mat_one and mat_two are single vector, shape them into 2d array with 1 column, rather than 1d. If not, could cause multiplication problems when we have both 1 column 2d array and single column 1d array in the same formula. But this can not always to set to True, hence default is actually false, because this function could take as input a matrix for mat_one, in that case, already 2d array.

2.1.4 pyfan.amto.array.scalararray

Created on Dec 2, 2017

@author: fan

Functions

scalar_to_2darray(x[, check_first])	
scalar_to_array(x[, check_first])	
zero_ndims(ndims_var)	
	Parameters

pyfan.amto.array.scalararray.scalar_to_2darray

pyfan.amto.array.scalararray.**scalar_to_2darray**(*x*, *check_first=True*)

pyfan.amto.array.scalararray.scalar_to_array

pyfan.amto.array.scalararray.**scalar_to_array**(*x*, *check_first=True*)

pyfan.amto.array.scalararray.zero_ndims

pyfan.amto.array.scalararray.**zero_ndims**(*ndims_var*)

Parameters

ndims_var: array the dimension of this array to be duplicated

2.2 1.2 JSON

Function to manipulate JSON Structures

pyfan.amto.json.json	Created on Jun 4, 2018

2.2.1 pyfan.amto.json.json

Created on Jun 4, 2018

@author: fan

Functions

jdump(aws_return_dict[, desc, logger, ...])	
json_serial(obj)	JSON serializer for objects not serializable by default json code

pyfan.amto.json.json.jdump

pyfan.amto.json.json.**jdump**(*aws_return_dict*, *desc=''*, *logger=None*, *print_here=False*)

pyfan.amto.json.json.json_serial

pyfan.amto.json.json.**json_serial**(*obj*)

JSON serializer for objects not serializable by default json code

2.3 1.3 List and Dict

List and dictionary

pyfan.amto.lsdc.lsdcconvert	The *pyfan.amto.lsdc.lsdcconvert* module provides list and dict converters.

2.3.1 pyfan.amto.lsdc.lsdcconvert

The *pyfan.amto.lsdc.lsdcconvert* module provides list and dict converters.

Created on Dec 18, 2020

import pyfan.amto.lsdc.lsdcconvert as pyfan_amto_lsdcconvert

Includes method `ff_decimal_rounder_uncommon()` and `ff_decimal_rounder()`.

Functions

ff_ls2dc(ls_list[, st_counter_str, ...])	Convert list to dict with list name and index and dict keys

pyfan.amto.lsdc.lsdcconvert.ff_ls2dc

pyfan.amto.lsdc.lsdcconvert.**ff_ls2dc**(*ls_list*, *st_counter_str='i'*, *st_all_str='o'*, *st_ls_name=None*, *verbose=False*)

Convert list to dict with list name and index and dict keys

Parameters

ls_list [list] A list of values.

st_counter_str [str] String prefix for list counter in dictionary name.

st_all_str [str] String prefix in front of total ele length in dict key name.

Returns

dict A dictionary of equal length to *ls_list* input, list converted to dict.

Examples using `pyfan.amto.lsdc.lsdcconvert.ff_ls2dc`

- *List and Dictionary Convertions*

2.4 1.4 Numeric

Numeric manipulations

pyfan.amto.numeric.round	The *pyfan.amto.numeric.round* provides decimal rounding for float arrays.

2.4.1 pyfan.amto.numeric.round

The *pyfan.amto.numeric.round* provides decimal rounding for float arrays.

Given an array of numbers, provide conditional decimal formatting rounding via fstring. This is used by table function to generate table specific rounding rules.

For example, a table with birthweight in grams, and ratios, might have 2 decimals for numbers less than 1, but no decimals for numbers larger than 1000 (which are the grams).

import pyfan.amto.numeric.round as pyfan_amto_round

Created on Dec 14, 2020

Includes method *ff_decimal_rounder_uncommon()* and *ff_decimal_rounder()*.

Functions

ff_decimal_rounder(ls_fl_num2format, ...[, ...])	Decimal rounding function with common decimal formatting
ff_decimal_rounder_uncommon([...])	Decimal rounding function with conditional formatting by number size

pyfan.amto.numeric.round.ff_decimal_rounder

pyfan.amto.numeric.round.**ff_decimal_rounder**(*ls_fl_num2format*, *it_or_dc_round_decimal*, *verbose=False*)

Decimal rounding function with common decimal formatting

Parameters

ls_fl_num2format [list of float] see *ff_decimal_rounder()*

it_or_dc_round_decimal [int or dict] the number of decimal points to keep. If dict, same as dc_round_decimal for *ff_decimal_rounder()*. If decimal, generate dict that provides

common formating

Returns

list of str Decimal formatted string outputs

pyfan.amto.numeric.round.ff_decimal_rounder_uncommon

pyfan.amto.numeric.round.**ff_decimal_rounder_uncommon**(*ls_fl_num2format=[0.0012345, 0.12345, 12.345, 123.45, 1234.5, 123456.789]*, *dc_round_decimal={0.1: 4, 1: 3, 100: 2, inf: 0}*, *verbose=False*)

Decimal rounding function with conditional formatting by number size

Given an array of numbers, format and return as a list of string, with different decimal formatting given different number sizes.

Parameters

ls_fl_num2format [list of float] list of numbers of approximate to decimals

dc_round_decimal [dict] dict incremental formatter. For example, for the default, if below 0.1 keep 4 decimals, If below 1 keep 3, if below 100 keep 2, if otherwise above, then keep 0 decimals Loop over formatter.

Returns

list of str Decimal formatted string outputs

CHAPTER

THREE

2 AMAZON WEB SERVICES

Functions to support AWS service usages.

3.1 2.1 General

AWS general functions.

pyfan.aws.general.credentials	
pyfan.aws.general.path	The *pyfan.aws.general.path* file paths etc

3.1.1 pyfan.aws.general.credentials

Functions

boto3_start_service([st_aws_service])	

pyfan.aws.general.credentials.boto3_start_service

pyfan.aws.general.credentials.**boto3_start_service**(*st_aws_service='s3'*)

3.1.2 pyfan.aws.general.path

The *pyfan.aws.general.path* file paths etc

Includes method *detect_store_path()*, *save_img()*.

Functions

detect_store_path([bl_check_path_exist, ...])	Detects checks if program is running on an AWS Linux Instance
save_img(plt, sna_image_name[, ...])	Saves Graph Locally, and also upload to S3 if requested

pyfan.aws.general.path.detect_store_path

pyfan.aws.general.path.**detect_store_path**(*bl_check_path_exist=True*, *srt_sub_path=None*, *st_local_path=None*)

Detects checks if program is running on an AWS Linux Instance

In our case, all code that run on AWS linux are running inside conda containers. If running on container, save to data folder. If running on some local machine save results to the user's home path's download folder, data subfolder.

Parameters

bl_check_path_exist [*bool*] checking saving path if it does not exist

srt_sub_path: `string`, optional this is the subpath to be used, in the data folder in EC2 container, or inside the downloads data folder under user directory.

st_local_path: `string`, optional local overriding string save path, if not, use download/data folder. This will replace the local path

Returns

tuple[bool, string] returns boolean if on amzn splatform, then the directory where to store save files

pyfan.aws.general.path.save_img

pyfan.aws.general.path.**save_img**(*plt*, *sna_image_name*, *spt_image_path=None*, *dpi=300*, *papertype='a4'*, *orientation='horizontal'*, *bl_upload_s3=False*, *st_s3_bucket=None*, *srt_s3_bucket_folder=None*)

Saves Graph Locally, and also upload to S3 if requested

Given figure object,

Parameters

plt: `matplotlib.pyplot` a matplotlib pyplot object from a graph that was just generated

sna_image_name: `string` image name, without the suffix of png

spt_image_path: `string`, optional path to image, if None, then use default local path in *detect_store_path()*

dpi: `integer`, optional image dpi

papertype: `string`, optional One of 'letter', 'legal', 'executive', 'ledger', 'a0' through 'a10', 'b0' through 'b10'.

orientation: `string`, optional 'horizontal' or 'portrait'

bl_upload_s3: `bool`, optional if file will be uploaded to s3

st_s3_bucket: `string`, optional Assuming that AWS credentials are already stored in the container on EC2 or locally in a .aws credential file. So *st_s3_bucket* bucket name refers to bucket in the credentialed user's s3 account.

srt_s3_bucket_folder: `string`, optional folder in s3 bucket to store image

Returns

tuple[bool, string] returns boolean if on amzn splatform, then the directory where to store save files

3.2 2.2 S3

Functions for S3 storage.

pyfan.aws.s3.pushsync	The *pyfan.aws.s3.pushsync* savse files to s3 and syncs

3.2.1 pyfan.aws.s3.pushsync

The *pyfan.aws.s3.pushsync* savse files to s3 and syncs

Includes method `ar_draw_random_normal()`.

Functions

s3_upload(spn_img_pwdfn[, st_s3_bucket, ...])	Upload an existing file to s3

pyfan.aws.s3.pushsync.s3_upload

pyfan.aws.s3.pushsync.**s3_upload**(*spn_img_pwdfn*, *st_s3_bucket='fans3testbucket'*, *srt_s3_bucket_folder='pyfan_scatterline3'*)

Upload an existing file to s3

Upload to a particular bucket and subfolder, file in some local directory

Parameters

spn_img_pwdfn: `string` full path to image, including the image name

st_s3_bucket: `string`, optional Assuming that AWS credentials are already stored in the container on EC2 or locally in a .aws credential file. So *st_s3_bucket* bucket name refers to bucket in the credentialed user's s3 account.

srt_s3_bucket_folder: `string`, optional folder in s3 bucket to store image

Returns

none

Examples

```
>>> spn_img_pwdfn = 'C:/Users/fan/Downloads/data/test/test_image.png'
>>> st_s3_bucket = 'fans3testbucket'
>>> srt_s3_bucket_folder = 'pyfan_scatterline3/folder1/'
>>> s3_upload(spn_img_pwdfn, st_s3_bucket, srt_s3_bucket_folder)
```

CHAPTER

FOUR

3 DEVELOPMENT

Package and function development support functions.

4.1 3.1 Log Support

Log support functions.

—

4.2 3.2 Object

Object support functions.

pyfan.devel.obj.classobjsupport	Created on Mar 16, 2017

4.2.1 pyfan.devel.obj.classobjsupport

Created on Mar 16, 2017

@author: fan

Functions

dynamic_obj_attr([attribute_array, ...])	

pyfan.devel.obj.classobjsupport.dynamic_obj_attr

pyfan.devel.obj.classobjsupport.**dynamic_obj_attr**(*attribute_array=['r_save', 'r_borr', 'delta']*, *attribute_values_array=['0.02', '0.05', '0.10']*, *print_values=False*)

CHAPTER

FIVE

4 GENERATE

Generate specific data-structures.

5.1 4.1 Random

Data structures based on random seed draws.

pyfan.gen.rand.randgrid	The *pyfan.gen.rand.randgrid* generate a grid with randomly spaced grid points.

5.1.1 pyfan.gen.rand.randgrid

The *pyfan.gen.rand.randgrid* generate a grid with randomly spaced grid points.

$$x \sim N(\mu, \sigma)$$

Includes method *ar_draw_random_normal()*.

Functions

ar_draw_random_normal(fl_mu, fl_sd, it_draws)	Draw a Vector of Possibly Sorted and Bounded Normal Shocks

pyfan.gen.rand.randgrid.ar_draw_random_normal

pyfan.gen.rand.randgrid.**ar_draw_random_normal**(*fl_mu*, *fl_sd*, *it_draws*, *it_seed=None*, *it_draw_type=0*, *fl_lower_sd=-3*, *fl_higher_sd=None*)

Draw a Vector of Possibly Sorted and Bounded Normal Shocks

Parameters

fl_mu, fl_sd [*float*] The mean and standard deviation of the normal process

it_draws: `int` Number of Draws

it_seed: `int`, optional External random seed externally. Default is 123.

it_draw_type: `int`, optional Indicates which type of normal draws to make. 0 sorted normal draws cut off at bounds. 1 equi-quantile unequal distance points; 2 normal draws unsorted.

fl_lower_sd, fl_higher_sd [*float*] Impose lower and upper bounds (in sd units) on shock draws. The normal distribution does not have lower or upper bounds.

Returns

numpy.array of shape (1, it_draws) A vector of sorted or unsorted random grid points, or equi-quantile points.

Notes

This method requires a dataset of equal-sized time series

Examples

```
>>> fl_mu = 0
>>> fl_sd = 1
>>> it_draws = 5
>>> it_seed = 123
>>> fl_lower_sd = -1
>>> fl_higher_sd = 0.8
>>> it_draw_type = 0
>>> ar_draw_random_normal(fl_mu, fl_sd, it_draws,
...                       it_seed, it_draw_type,
...                       fl_lower_sd, fl_higher_sd)
[-1.          0.8         0.2829785 - 1. - 0.57860025]
```

```
>>> it_draw_type = 1
>>> ar_draw_random_normal(fl_mu, fl_sd, it_draws,
...                       it_seed, it_draw_type,
...                       fl_lower_sd, fl_higher_sd)
[-1. - 0.47883617 - 0.06672597  0.3338994   0.8]
```

```
>>> it_draw_type = 2
>>> ar_draw_random_normal(fl_mu, fl_sd, it_draws,
...                       it_seed, it_draw_type,
...                       fl_lower_sd, fl_higher_sd)
[-1. - 1. - 0.57860025  0.2829785   0.8]
```

CHAPTER

SIX

5 GRAPH

Graphing support tools.

6.1 5.1 Example

Graphing example functions.

—

6.2 5.2 Generic

All purpose graph support functions

pyfan.graph.generic.allpurpose	Created on Sep 24, 2013

6.2.1 pyfan.graph.generic.allpurpose

Created on Sep 24, 2013

@author: fan

Functions

OLSEmaxGraphs(saveFileSuffix, yVal, allDataX)	
OLSEmaxValAndChoicesGraphs(allDataY, allDataX)	
contourAnd3D(xData, yData, zData, xLabStr, ...)	
graph_emaxKCash_Value(soluSupObj, resources, ...)	
grid(x, y, z[, resX, resY])	Convert 3 column data to matplotlib grid
guassian_kde_graph(data_fordensity[, ...])	
sampleDataGraphs()	
subplot_square_counter([totalimages])	
tripleAngle3dSave(ax, graphTitleDisp, ...[, ...])	

pyfan.graph.generic.allpurpose.OLSEmaxGraphs

pyfan.graph.generic.allpurpose.**OLSEmaxGraphs**(*saveFileSuffix*, *yVal*, *allDataX*, *saveDirectory='default'*, *saveFileName='default'*, *yLabelName='yLabelName'*, *xLabelNames=['Height', 'Weight', 'Income']*)

pyfan.graph.generic.allpurpose.OLSEmaxValAndChoicesGraphs

pyfan.graph.generic.allpurpose.**OLSEmaxValAndChoicesGraphs**(*allDataY*, *allDataX*, *saveFileSuffix=''*, *yLabelNames=['Emax', 'Choice']*, *xLabelNames=['Height', 'Weight', 'Income']*, *saveDirectory='default'*, *saveFileName='default'*)

pyfan.graph.generic.allpurpose.contourAnd3D

pyfan.graph.generic.allpurpose.**contourAnd3D**(*xData*, *yData*, *zData*, *xLabStr*, *yLabStr*, *zLabStr*, *graphTitleDisp*, *graphTitleSave*, *savedpi=125*, *angleType=[1, [1, 2, 3]]*, *drawContour=False*, *draw3D=True*, *draw3DSurf=False*, *contourXres=100*, *contourYres=100*, *s=20*, *alpha=0.6*, *subplot=None*, *fig=None*)

pyfan.graph.generic.allpurpose.graph_emaxKCash_Value

pyfan.graph.generic.allpurpose.**graph_emaxKCash_Value**(*soluSupObj*, *resources*, *k_vec*, *emaxValsCur*, *emaxChoicesCur*, *emaxChoiceOfMaxCollCur*, *predictUtil*)

pyfan.graph.generic.allpurpose.grid

pyfan.graph.generic.allpurpose.**grid**(*x*, *y*, *z*, *resX=100*, *resY=100*)

Convert 3 column data to matplotlib grid

pyfan.graph.generic.allpurpose.guassian_kde_graph

pyfan.graph.generic.allpurpose.**guassian_kde_graph**(*data_fordensity*, *graph_xgrid=False*, *xgridpoints=1000*, *color='b'*, *label=False*, *showOnScreen=False*)

pyfan.graph.generic.allpurpose.sampleDataGraphs

pyfan.graph.generic.allpurpose.**sampleDataGraphs**()

pyfan.graph.generic.allpurpose.subplot_square_counter

pyfan.graph.generic.allpurpose.**subplot_square_counter**(*totalimages=15*)

pyfan.graph.generic.allpurpose.tripleAngle3dSave

pyfan.graph.generic.allpurpose.**tripleAngle3dSave**(*ax*, *graphTitleDisp*, *xLabStr*, *yLabStr*, *zLabStr*, *graphTitleSave*, *savedpi=125*, *angleType=[1, [1, 2, 3]]*)

Classes

graphFunc([showOrNot, saveDirectory, saveDPI])	

Methods

pyfan.graph.generic.allpurpose.graphFunc

class pyfan.graph.generic.allpurpose.**graphFunc**(*showOrNot=False*, *saveDirectory='C:/Users/fan/Pictures'*, *saveDPI=125*)

Methods

graphingEachType(graphType, ... [, colorVar, ...])	If do not use basic pylab, but have external axis
xyPlotMultiYOneX([xData, yDataMat, ...])	Graph general

sampleGraphs	
savingFig	

graphingEachType(*graphType*, *xSingleArrayData*, *ySingleArrayata*, *keywords*, *colorVar=None*, *label=False*, *color='b'*, *pylabUse=None*)

If do not use basic pylab, but have external axis

xyPlotMultiYOneX(*xData=array([4.86471187, 5.64833321, 6.05121941, 6.1249189, 6.1934786, 6.60990612, 6.84237463, 6.85701589, 6.98730587, 7.00795485, 7.09075929, 7.1206578, 7.13921199, 7.15180429, 7.15602334, 7.1811552, 7.26076649, 7.26744348, 7.27342273, 7.30397367, 7.31894912, 7.40775932, 7.60926549, 7.76953977, 7.86564097, 8.10881129, 8.11871991, 8.15467481, 8.20264858, 8.2186071, 8.22094054, 8.23323072, 8.28016582, 8.32284236, 8.3256191, 8.45004457, 8.4700339, 8.48298421, 8.49155303, 8.5030347, 8.58638633, 8.67435893, 8.68368368, 8.7272959, 8.78174355, 8.81669893, 8.86041522, 8.8610798, 8.96614101, 9.06408182, 9.06762354, 9.08999328, 9.11493678, 9.11639981, 9.14512116, 9.14782115, 9.18763291, 9.19088391, 9.21928007, 9.23486544, 9.24275827, 9.29151088, 9.29628784, 9.35186871, 9.36392877, 9.38073905, 9.41889945, 9.4722915, 9.50541893, 9.54178978, 9.55199131, 9.57945794, 9.5889156, 9.62500985, 9.64880251, 9.66312659, 9.70172844, 9.71454613, 9.76100199, 9.78801369, 9.79190217, 9.84749769, 9.87456769, 9.96772857, 9.97993323, 10.04877935, 10.05715421, 10.08434413, 10.09000457, 10.09062525, 10.09076919, 10.0973051, 10.11592262, 10.13025182, 10.14743154, 10.1511082, 10.15614128, 10.1582611, 10.20432813, 10.21085435, 10.22380816, 10.23635907, 10.23784285, 10.2808753, 10.2815864, 10.28544398, 10.36783924, 10.37363085, 10.38895095, 10.39121575, 10.45390955, 10.45774046, 10.46386799, 10.47344846, 10.48198575, 10.50492863, 10.53065441, 10.55443431, 10.5672081, 10.56893849, 10.56940588, 10.58129952, 10.61058708, 10.62506172, 10.68673621, 10.69861688, 10.71672169, 10.71808441, 10.7322955, 10.73811565, 10.77077802, 10.78061921, 10.80570846, 10.8085109, 10.89862154, 10.90147879, 10.94546186, 10.9613325, 10.97610054, 11.03398717, 11.04816552, 11.07767961, 11.09272265, 11.16297833, 11.16718859, 11.18480857, 11.20999984, 11.22121849, 11.22408063, 11.29259247, 11.30330059, 11.35107889, 11.37153371, 11.37176227, 11.38997023, 11.5055071, 11.52500095, 11.53524831, 11.56563468, 11.57191949, 11.61933093, 11.62130875, 11.67177541, 11.70323905, 11.70981581, 11.80063977, 11.8333497, 11.90166503, 11.91959236, 11.92589249, 11.93682633, 11.96614061, 11.97174901, 12.0515141, 12.07303774, 12.13109676, 12.15501903, 12.33691446, 12.40667318, 12.4211352, 12.49964024, 12.52034292, 12.52450031, 12.52833649, 12.60216293, 12.78802262, 12.93531052, 12.96576691, 13.03937324, 13.12951383, 13.24265605, 13.43277003, 13.58124123, 13.66049983, 13.69789859, 13.78345007, 13.87531035, 14.31262601, 14.61953676, 15.76443763]), yDataMat=array([[0.21147133, 9.9493942], [- 0.28209892, 9.6697611], [0.26698715, 10.90732676], [0.63264969, 9.51828939], [1.56130036, 8.92512708], [- 1.05243111, 10.79907541], [- 0.85033604, 11.43259478], [0.16187254, 8.83820968], [0.04406206, 9.37290445], [1.01880142, 10.59262814], [- 0.16255127, 8.18875284], [- 0.41731215, 10.41978739], [0.28206779, 9.73795128], [0.64048697, 10.22742965], [1.19433617, 10.93290896], [0.87751523, 10.32365428], [- 1.1193347, 8.95530774], [0.28268712, 8.83024881], [0.2944292, 10.42454536], [0.39989993, 10.79403587], [0.71046353, 10.94198748], [- 0.51270223, 11.63945511], [0.28563999, 10.39768566], [- 2.00422032, 10.63537064], [2.18263667, 10.74773528], [- 0.07432916, 9.46865524], [0.22285787, 9.47331728], [0.73709499, 9.42325366], [0.25289561, 10.5624536], [- 1.02641087, 10.50922961], [- 1.38819153, 9.15657443], [- 0.25860746, 9.54606349], [- 1.82463688, 10.22535588], [0.72875842, 9.64696274], [- 1.28900336, 8.75365466], [0.26527143, 9.55974539], [0.87360393, 9.19824574], [0.41649347, 9.87211807], [- 0.7645099, 9.23991941], [1.41775072, 10.98530703], [- 1.70744423, 9.18470159], [- 0.6734346, 10.46424145], [0.56301635, 11.11423069], [0.95860021, 10.00236895], [0.23163272, 9.40011941], [0.37735236, 10.30179212], [- 1.05078859, 9.93860265], [0.68125554, 10.19813029], [- 0.12584076, 11.54888061], [- 0.87531559, 9.90177315], [0.10700528, 9.99246446], [0.96983081, 9.5937271], [0.45452417, 9.19687273], [- 1.3920557, 11.08140392], [0.52611735, 10.65510555], [0.66100261, 10.97225791],*

Graph general

yDataMat each column corresponds to x

Examples using `pyfan.graph.generic.allpurpose.graphFunc`

- *Generate Graphs using the Generic Graphing Tool*

6.3 5.3 Tools

Some graphing tools.

pyfan.graph.tools.subplot	Created on Aug 6, 2018

6.3.1 pyfan.graph.tools.subplot

Created on Aug 6, 2018

@author: fan

Design page subplot

Functions

subplot_design([plot_count, base_multiple, ...])	subplot grid and size given total plot count

pyfan.graph.tools.subplot.subplot_design

pyfan.graph.tools.subplot.**subplot_design**(*plot_count=10*, *base_multiple=4*, *base_multiple_high_frac=0.6*)

subplot grid and size given total plot count

figsize = (width height)

Examples

import Support.graph.subplot as sup_graph_subplot figsize, rows, cols = sup_graph_subplot.subplot_design(plot_count=10, base_multiple=4, base_multiple_high_frac = 0.60)

CHAPTER

SEVEN

6 PANDAS

Pandas related functions.

7.1 6.1 Categorical

Functions to handle categorical variables.

pyfan.panda.categorical.catevars	Created on Aug 10, 2018
pyfan.panda.categorical.strsvarskeys	Created on Aug 9, 2018

7.1.1 pyfan.panda.categorical.catevars

Created on Aug 10, 2018

@author: fan

Functions

show_cates(df, varname)	

pyfan.panda.categorical.catevars.show_cates

pyfan.panda.categorical.catevars.**show_cates**(*df*, *varname*)

7.1.2 pyfan.panda.categorical.strsvarskeys

Created on Aug 9, 2018

@author: fan

Store key variable names, file names, etc.

Individual analysis files should refer to this file.

Functions

conditions(df[, condi_dict, return_subset])	Commonly used conditionings
file_names()	
main_data_directory([dataset_name])	
var_names([vartype, file_name, out_type])	

pyfan.panda.categorical.strsvarskeys.conditions

pyfan.panda.categorical.strsvarskeys.**conditions**(*df*, *condi_dict=None*, *return_subset=False*)

Commonly used conditionings

Can combine conditioning statements together in list

Parameters

condi_dict: dictionary condi_dict = {'region':'central', 'years':2002-2005}

file_name: string different files could have different variable names for the same variables, although these should be unified if possible

pyfan.panda.categorical.strsvarskeys.file_names

pyfan.panda.categorical.strsvarskeys.**file_names**()

pyfan.panda.categorical.strsvarskeys.main_data_directory

pyfan.panda.categorical.strsvarskeys.**main_data_directory**(*dataset_name='thai_data_z'*)

pyfan.panda.categorical.strsvarskeys.var_names

pyfan.panda.categorical.strsvarskeys.**var_names**(*vartype=1*, *file_name='thaiMthly_Annulized_DataZ'*, *out_type='coln'*)

7.2 6.2 In and Out

Functions for combine, export, etc dataframes.

—

7.3 6.3 Stats

Stats operations on dataframes.

pyfan.panda.stats.cutting	Created on Aug 13, 2018
pyfan.panda.stats.mean_varcov	Created on Aug 9, 2018
pyfan.panda.stats.polynomial_regression	Created on Sep 23, 2018

7.3.1 pyfan.panda.stats.cutting

Created on Aug 13, 2018

@author: fan

import panda.cutting as pd_cut

Functions

pd_winsorize_columnwise(df, . . . [, . . .])	Winsorizing column by column, no dependence across coluns.
sample_run()	

pyfan.panda.stats.cutting.pd_winsorize_columnwise

pyfan.panda.stats.cutting.**pd_winsorize_columnwise**(*df*, *winsor_coln_list*, *coln_perc_cutoffs*, *return_type*, *print_array=False*, *json_debug=False*)

Winsorizing column by column, no dependence across coluns. Winsorize column by column

cols = 5 rows = 20 np.random.seed(123) data = (np.random.rand(rows,cols)-0.5)*100

df = pd.DataFrame(data, columns=['col' + str(ctr) for ctr in range(cols)]) winsor_coln_list = ['col0', 'col1','col3','col4']

Parameters

df: dataFrame initial dataset

winsor_coln_list: list list of column names to winsorize ['col0', 'col1','col3','col4']

coln_perc_cutoffs: dictionary a nested dictionary where keys are elements of winsor_coln_list, and values are a dictionary with min and max percentiles of winsorizing values. if min is 0, do not create cutcolss

{'col0':{'q_ge':0,'q_le':0.9, 'v_ll':10}, 'col1':{'q_ge':0.30,'v_le':50}, 'col3':{'q_ge':0.01,'q_le':0.60, 'v_ll':40}, 'col4':{'q_ge':0.01,'q_le':1, 'v_ll':33, 'v_gg':-5}}

return_type: string 'winsorize' or 'cutsubset'

pyfan.panda.stats.cutting.sample_run

pyfan.panda.stats.cutting.**sample_run**()

7.3.2 pyfan.panda.stats.mean_varcov

Created on Aug 9, 2018

@author: fan

Generate mean, and variance covariance of key state variables from data

Functions

gen_mean(df, mean_var_list[, ...])	
gen_varcov(df, varcov_var_list[, ...])	

pyfan.panda.stats.mean_varcov.gen_mean

pyfan.panda.stats.mean_varcov.**gen_mean**(*df*, *mean_var_list*, *short_mean_var_list=None*, *group_by_var_list=None*, *conditioning=None*)

pyfan.panda.stats.mean_varcov.gen_varcov

pyfan.panda.stats.mean_varcov.**gen_varcov**(*df*, *varcov_var_list*, *short_varcov_var_list=None*, *group_by_var_list=None*, *conditioning=None*)

7.3.3 pyfan.panda.stats.polynomial_regression

Created on Sep 23, 2018

@author: fan

Functions

ols_formula(df, dependent_var, *excluded_cols)	Generates the R style formula for statsmodels (patsy) given the dataframe, dependent variable and optional excluded columns as strings
tester()	

pyfan.panda.stats.polynomial_regression.ols_formula

pyfan.panda.stats.polynomial_regression.**ols_formula**(*df*, *dependent_var*, **excluded_cols*)

Generates the R style formula for statsmodels (patsy) given the dataframe, dependent variable and optional excluded columns as strings

pyfan.panda.stats.polynomial_regression.tester

pyfan.panda.stats.polynomial_regression.**tester**()

CHAPTER

EIGHT

7 STATISTICS

Statistical functions.

8.1 7.1 Interpolate

Interpolation functions.

pyfan.stats.interpolate.interpolate2d	Created on Mar 7, 2017

8.1.1 pyfan.stats.interpolate.interpolate2d

Created on Mar 7, 2017

@author: fan

Functions

exp_value_interpolate_bp(prod_inst, ...)	interpolate value function and expected value function.
exp_value_interpolate_bpkp(hhp_inst, ...)	interpolate value function and expected value function.
exp_value_interpolate_main(u1, x1, y1, x2, ...)	A. Get Interpolant
inter_states_bp(prod_inst, util_opti, ...)	interpolate value function and expected value function.
interp2d(prod, cash[, z, interpolant, kind])	Centralize the invokation of 2D interpolation tool
interpRbf2D(prod, cash[, z, interpolant, kind])	
interpRbf3D(prod, cash, A[, z, interpolant, ...])	
interp_griddata(cur_u, cur_x1, cur_x2, ...)	Centralize the invokation of 2D interpolation tool
k_alpha_cash(hhp_inst, b_vec, k_vec)	
regress(dependent_var, rhs_var)	
regress_mat(k_alpha, cash)	

pyfan.stats.interpolate.interpolate2d.exp_value_interpolate_bp

pyfan.stats.interpolate.interpolate2d.**exp_value_interpolate_bp**(*prod_inst*, *util_opti*, *b_ssv_sd*, *k_ssv_sd*, *epsilon_ssv_sd*, *b_ssv*, *k_ssv*, *epsilon_ssv*, *b_ssv_zr*, *k_ssv_zr*, *epsilon_ssv_zr*, *states_vfi_dim*, *shocks_vfi_dim*)

interpolate value function and expected value function.

Need three matrix here: 1. state matrix x shock matrix where optimal choices were solved at

- previously, shock for this = 0, but now shock vector might not be zero

2. **state matrix x shock matrix where shocks are drawn monte carlo way to allow** for averaging, integrating over shocks for each x row
3. state matrix alone, shock = 0, each of the x row in matrix x

pyfan.stats.interpolate.interpolate2d.exp_value_interpolate_bpkp

pyfan.stats.interpolate.interpolate2d.**exp_value_interpolate_bpkp**(*hhp_inst*, *util_opti*, *b*, *k*, *b_shk*, *k_shk*)

interpolate value function and expected value function.

cash and k_alpha calculation below does not repeat what happened already inside lifetimeutility. Inside lifetimeutility, we have next period cash and k_alpha here is this period

pyfan.stats.interpolate.interpolate2d.exp_value_interpolate_main

pyfan.stats.interpolate.interpolate2d.**exp_value_interpolate_main**(*u1*, *x1*, *y1*, *x2*, *y2*, *x2_noshk*, *y2_noshk*, *states_dim*, *shocks_dim*, *return_uxy=False*)

A. Get Interpolant

pyfan.stats.interpolate.interpolate2d.inter_states_bp

pyfan.stats.interpolate.interpolate2d.**inter_states_bp**(*prod_inst*, *util_opti*, *b_ssv_sd*, *k_ssv_sd*, *epsilon_ssv_sd*, *b_ssv*, *k_ssv*, *epsilon_ssv*, *b_ssv_zr*, *k_ssv_zr*, *epsilon_ssv_zr*, *states_vfi_dim*, *shocks_vfi_dim*)

interpolate value function and expected value function.

Need three matrix here: 1. state matrix x shock matrix where optimal choices were solved at

- previously, shock for this = 0, but now shock vector might not be zero

2. **state matrix x shock matrix where shocks are drawn monte carlo way to allow** for averaging, integrating over shocks for each x row
3. state matrix alone, shock = 0, each of the x row in matrix x

pyfan.stats.interpolate.interpolate2d.interp2d

pyfan.stats.interpolate.interpolate2d.**interp2d**(*prod*, *cash*, *z=None*, *interpolant=None*, *kind='linear'*)

Centralize the invokation of 2D interpolation tool

Potentially chagne this to something else if I don't like it.

pyfan.stats.interpolate.interpolate2d.interpRbf2D

pyfan.stats.interpolate.interpolate2d.**interpRbf2D**(*prod*, *cash*, *z=None*, *interpolant=None*, *kind='linear'*)

pyfan.stats.interpolate.interpolate2d.interpRbf3D

pyfan.stats.interpolate.interpolate2d.**interpRbf3D**(*prod*, *cash*, *A*, *z=None*, *interpolant=None*, *kind='cubic'*)

pyfan.stats.interpolate.interpolate2d.interp_griddata

pyfan.stats.interpolate.interpolate2d.**interp_griddata**(*cur_u*, *cur_x1*, *cur_x2*, *new_x1*, *new_x2*)

Centralize the invokation of 2D interpolation tool

Potentially chagne this to something else if I don't like it.

pyfan.stats.interpolate.interpolate2d.k_alpha_cash

pyfan.stats.interpolate.interpolate2d.**k_alpha_cash**(*hhp_inst*, *b_vec*, *k_vec*)

pyfan.stats.interpolate.interpolate2d.regress

pyfan.stats.interpolate.interpolate2d.**regress**(*dependent_var*, *rhs_var*)

pyfan.stats.interpolate.interpolate2d.regress_mat

pyfan.stats.interpolate.interpolate2d.**regress_mat**(*k_alpha*, *cash*)

8.2 7.2 Markov

Markov related functions.

pyfan.stats.markov.transprobcheck	The *pyfan.stats.markov.transprobcheck* checks markov transition row sums.

8.2.1 pyfan.stats.markov.transprobcheck

The *pyfan.stats.markov.transprobcheck* checks markov transition row sums.

A markov transition matrix where each row does not sum up to 1 due to simulation errors. Check if the gap between 1 and the row values are too big, and then normalize.

import pyfan.stats.markov.transprobcheck as pyfan_stats_transprobcheck

Includes method *markov_trans_prob_check()* and *markov_condi_prob2one()*.

Functions

markov_condi_prob2one(mt_trans)	Rescale markov transitions rows to sum to 1
markov_trans_prob_check(mt_trans[, ...])	Markov conditional transition probability check

pyfan.stats.markov.transprobcheck.markov_condi_prob2one

pyfan.stats.markov.transprobcheck.**markov_condi_prob2one**(*mt_trans*)

Rescale markov transitions rows to sum to 1

Suppose each transition matrix row sums up to slighly less than one, rescale so it sums to one.

Parameters

mt_trans [numpy.array of shape (N, N)] The AR1 transition matrix, each row is a state, each value in each row is the conditional probability of moving from state i (row) to state j (column)

Returns

ndarray The rescaled numpy array

pyfan.stats.markov.transprobcheck.markov_trans_prob_check

pyfan.stats.markov.transprobcheck.**markov_trans_prob_check**(*mt_trans*, *fl_atol_per_row=1e-05*, *fl_atol_avg_row=1e-08*, *fl_sum_to_match=1*)

Markov conditional transition probability check

Parameters

mt_trans [numpy.array of shape (N, N)] The AR1 transition matrix, each row is a state, each value in each row is the conditional probability of moving from state i (row) to state j (column)

fl_atol_per_row [*float*, optional] Tolerance for the difference between 1 and each row sum

fl_atol_avg_row [*float*, optional] Tolerance for the difference between 1 and average of row sums

fl_sum_to_match [*float*, optional] This should be 1, unless the function is not used to handle transition matrixes

Returns

———-

tuple A tuple of booleans, the first element is if satisfies the overall criteria. Second is if satisifes the per_row condition. Third if satisfies the average criteria.

Examples

```
>>> mt_ar1_trans = np.array([[0.4334, 0.5183, 0.0454],
>>>                          [0.2624, 0.5967, 0.1245],
>>>                          [0.1673, 0.5918, 0.2005]])
>>> bl_ar1_sum_pass, bl_per_row_pass, bl_avg_row_pass = markov_trans_prob_
↪check(mt_ar1_trans)
>>> print(f'{bl_ar1_sum_pass=}')
bl_ar1_sum_pass=False
>>> print(f'{bl_per_row_pass=}')
bl_per_row_pass=False
>>> print(f'{bl_avg_row_pass=}')
bl_avg_row_pass=False
```

8.3 7.3 Multinomial

Discrete choice multinomial functions.

pyfan.stats.multinomial.multilogit	Created on Dec 4, 2017

8.3.1 pyfan.stats.multinomial.multilogit

Created on Dec 4, 2017

@author: fan

Classes

UtilityMultiNomial([scale_coef])	each_j_indirect_utility:

pyfan.stats.multinomial.multilogit.UtilityMultiNomial

class pyfan.stats.multinomial.multilogit.**UtilityMultiNomial**(*scale_coef=1*)

each_j_indirect_utility: N by J matrix

N is the number of individuals (unique states) J is the number of choices

N might be 0

Methods

expected_u_integrate_allj(prob_denominator)	'see Train discussion on consumer surplus and logit' 'Need to check the reference that Train cites to make sure this integration applies in my case, should derive it myself' If one option has much higher utility, and if variance is low, integrated utility is linear in this option, in fact they are equal
prob_denominator(all_J_indirect_utility)	if:
prob_j(all_J_indirect_utility[, ...])	

get_outputs	

expected_u_integrate_allj(*prob_denominator*)

'see Train discussion on consumer surplus and logit' 'Need to check the reference that Train cites to make sure this integration applies in my case, should derive it myself' If one option has much higher utility, and if variance is low, integrated utility is linear in this option, in fact they are equal

prob_denominator(*all_J_indirect_utility*)

if:

all_J_indirect_utility/self.scale_coef = -598.66/0.75

then: prob_denominator = exp(-598.66/0.75) = 0.0

then: sum(prob_denominator) = 0

then: np.exp(all_J_indirect_utility/self.scale_coef)/prob_denominator_tile = INVALID

so there must be some minimal level for the division here. in terms of scaling

CHAPTER

NINE

8 UTILITIES

General support functions.

9.1 8.1 In and Out

Export, import etc.

—

9.2 8.2 Path

Path and location related functions.

pyfan.util.path.movefiles	

9.2.1 pyfan.util.path.movefiles

Functions

fp_agg_move_subfiles([spt_root_src, . . .])	Aggregate and Move a Collection of Non-empty Folders

pyfan.util.path.movefiles.fp_agg_move_subfiles

pyfan.util.path.movefiles.**fp_agg_move_subfiles**(*spt_root_src='C:/Users/fan/pyfan/vig/support/inout/_folder/fd/fa*
st_srt_srh='_images',
st_fle_srh=''*, *srt_agg='img'*,
ls_srt_dest=['C:/Users/fan/pyfan/vig/support/inout/_folder/fd/fa
'C:/Users/fan/pyfan/vig/support/inout/_folder/'],
bl_delete_src=True, *bl_test=True*, *verbose=False*)

Aggregate and Move a Collection of Non-empty Folders

A program (forexample mlx to tex conversion) creates in a folder a number of subfolder that stores images. Aggregate all the various image folders into a common image folder. And then move this common image folder to other destinations in order to flexibly generate aggregation files with common path that rely on images from various subfolders.

Parameters

spt_root_src: string root folder where subfolders are contained

st_srt_srh: string gather subfolder names that contain this string

st_fle_srh: string search in subfolders for files whose name contain string

srt_agg: string name of subfolder where found folders are aggregated at

ls_srt_dest: :obj:`list` of :obj:`str` list of folder paths to move aggregate subfolders over to

bl_delete_src: bool delete folders at existing locations

bl_test: bool test by searching for paths dest and src, do not move

verbose: bool print details

Returns

None nothing is returned

Examples

```
>>> fp_agg_move_subfiles(spt_root_src="C:/Users/fan/Math4Econ/matrix_application/
↪",
>>                                                 st_srt_srh="_images",
>>                                                 st_fle_srh="*.png",
>>                                                 srt_agg='img',
>>                                                 ls_srt_dest=["C:/Users/fan/Math4Econ/
↪"],
>>                                                 bl_delete_src=False,
>>                                                 bl_test=False,
>>                                                 verbose=False)
```

9.3 8.3 PDF

PDF generation support functions

pyfan.util.pdf.pdfgen	pyfan generate and clean pdf files from folder The *pyfan.util.pdf.pdfgen* generates pdf files from tex files.

9.3.1 pyfan.util.pdf.pdfgen

pyfan generate and clean pdf files from folder The *pyfan.util.pdf.pdfgen* generates pdf files from tex files.

Gather all tex files from a folder, allow for exclusion strings. Generate PDFs from the tex files. And then clean up extraneous PDF outputs.

Includes method *ff_pdf_gen_clean()*.

Functions

ff_pdf_gen_clean([ls_spt_srh, spt_out, . . .])	Generate pdf files from latex files in various folders.

pyfan.util.pdf.pdfgen.ff_pdf_gen_clean

pyfan.util.pdf.pdfgen.**ff_pdf_gen_clean**(*ls_spt_srh=None*, *spt_out='C:/Users/fan/Documents/Dropbox (UH-ECON)/Project Emily Minority Survey/EthLang/reg_lang_abi_cls_mino/'*, *spn_pdf_exe='C:/texlive/2019/bin/win32/xelatex.exe'*, *ls_st_contain=None*, *ls_st_ignore=None*, *bl_recursive=False*, *bl_clean=True*, *ls_suf_clean=None*)

Generate pdf files from latex files in various folders.

This file serves important paper generation function. It compiles multiple files satisfying string search requirements or exclusion conditions in multiple folders, and saves resulting pdf outputs in one folder. This allows for easy testing and management of mutiple pdf/latex files for the same project. Suppose there is a longer version of a paper, a shorter version, and an appendix file. We want to regularly test the compilations of all files, otherwise, as we work on one of the files, perhaps we some something in the some shared files that lead to other files breaking without knowing.

This should be run for all outward facing pdf/tex files for a project regularly in order to check if all files still compile.

By brining resulting outputs to a single folder, this makes it easier to see all paper and project relevant outputs. Additionally, this cleans up all pdf generated extraneous files once we have pdf itself, saving pdf compile folder clutter.

Parameters

ls_spt_srh [list of str] A list of strings of the path in which to search for tex files. They should be all on the same path. If *bl_recursive* is true, then this searchs in all subfolders.

spt_out [str] The Path to store outputs. All PDFs stored under single directory. This path must be directly on the same path as `ls_spt_srh', can be higher up on the same tree, but not on a different branch.

spn_pdf_exe: str The path to the pdflatex or alternative exe file

ls_st_contain: :obj:`list` of :obj:`str` a list of strings the found names must contain one of these search words, not all, just one of.

ls_st_ignore [list of str] a list of string file names to ignore

bl_recursive [bool] Whether to search for all tex files within subfolders

bl_clean [bool] To clean up after file generation

ls_suf_clean: :obj:`list` of :obj:`str` list of

Returns

dict A list of string pdf file names outputed,

Examples using `pyfan.util.pdf.pdfgen.ff_pdf_gen_clean`

- *Generate PDFs and Clean*

9.4 8.4 RMD

RMD and bookdown related functions.

9.5 8.5 Timer

Timer functions.

pyfan.util.timer.timer	The *pyfan.util.timer.timer* generates various timer related strings.

9.5.1 pyfan.util.timer.timer

The *pyfan.util.timer.timer* generates various timer related strings.

Includes method *getDateTime()*.

Functions

curTimeDiff([startTime])	
getDateTime([timeType])	

pyfan.util.timer.timer.curTimeDiff

pyfan.util.timer.timer.**curTimeDiff**(*startTime=None*)

pyfan.util.timer.timer.getDateTime

pyfan.util.timer.timer.**getDateTime**(*timeType=6*)

Classes

Timer([name])	

pyfan.util.timer.timer.Timer

class pyfan.util.timer.timer.**Timer**(*name=None*)

CHAPTER

TEN

GALLERY OF EXAMPLES

10.1 Data Type Examples

10.1.1 Numeric Rounding Function

Given an array of numbers round it with conditioning formattings.

```
# Author: Fan Wang (fanwangecon.github.io)
import pyfan.amto.numeric.round as pyfan_amto_round
import numpy as np
import matplotlib.pyplot as plt
```

Common rounding

```
# construct data inputs
ar_fl_exa = np.array([0.4334, 0.5183, 0.0454, 0.0027, 0.0002])
ls_st_numformated_common = pyfan_amto_round.ff_decimal_rounder(ls_fl_num2format=ar_fl_
↪exa, it_or_dc_round_decimal=2)
print(f'{ls_st_numformated_common=}')
```

Out:

```
ls_st_numformated_common=['0.43', '0.52', '0.05', '0.00', '0.00']
```

Uncommon rounding by number size with fractions

```
dc_round_decimal = {0.001:4, 0.01:3, 0.1:2, float("inf"):2}
ls_st_numformated_uncommon = pyfan_amto_round.ff_decimal_rounder_uncommon(ls_fl_
↪num2format=ar_fl_exa,
                                                                          dc_round_
↪decimal=dc_round_decimal)
print(f'{ls_st_numformated_uncommon=}')
```

Out:

```
ls_st_numformated_uncommon=['0.43', '0.52', '0.05', '0.003', '0.0002']
```

Uncommon rounding by number size test 2 with large numbers

```
ls_fl_num2format = [0.0012345, 0.12345, 12.345, 123.45, 1234.5, 123456.789]
dc_round_decimal = {0.1:4, 1:3, 100:2, float("inf"):0}
ls_st_numformated_large_uncommon = pyfan_amto_round.ff_decimal_rounder_uncommon(ls_fl_
↪num2format=ls_fl_num2format,
                                                                                dc_
↪round_decimal=dc_round_decimal)
print(f'{ls_st_numformated_large_uncommon=}')


# Start Plot
fig, ax = plt.subplots()

# Text Plot
ax.text(0.5, 0.5, f'{ar_fl_exa}\n{ls_st_numformated_common=}\n{dc_round_decimal=}\n
↪{ls_st_numformated_uncommon=}'
                  f'\n\n{ls_fl_num2format=}\n{dc_round_decimal=}\n{ls_st_numformated_
↪large_uncommon=}',
        horizontalalignment='center',
        verticalalignment='center',
        fontsize=10, color='black',
        transform=ax.transAxes)

# Labeling
ax.set_axis_off()
plt.show()
```

Out:

```
ls_st_numformated_large_uncommon=['0.0012', '0.123', '12.35', '123', '1234', '123457']
```

Total running time of the script: (0 minutes 0.083 seconds)

10.1.2 List and Dictionary Convertions

Convert between list and dictionary

```
# Author: Fan Wang (fanwangecon.github.io)
import pyfan.amto.lsdc.lsdcconvert as pyfan_amto_lsdcconvert
import pprint
import matplotlib.pyplot as plt
import textwrap
```

Convert list to dictionary

```
# list
ls_combo_type = ["e", "20201025x_esr_list_tKap_mlt_ce1a2", ["esti_param.kappa_ce9901",
↪ "esti_param.kappa_ce0209"],
                 1, "C1E31M3S3=1"]

# convert calling function without parameters:
dc_ls_combo_type_a = pyfan_amto_lsdcconvert.ff_ls2dc(ls_combo_type)
print(f'{dc_ls_combo_type_a=}')

# convert calling function without parameters:
dc_ls_combo_type_b = pyfan_amto_lsdcconvert.ff_ls2dc(ls_combo_type, 'i', 'o',␣
↪verbose=True)
print(f'{dc_ls_combo_type_a=}')

# convert calling function with later parameter names:
dc_ls_combo_type_c = pyfan_amto_lsdcconvert.ff_ls2dc(ls_combo_type, st_counter_str='i
↪', st_all_str='o')
print(f'{dc_ls_combo_type_b=}')

# convert calling function with all named parameters:
dc_ls_combo_type_d = pyfan_amto_lsdcconvert.ff_ls2dc(ls_list=ls_combo_type, st_
↪counter_str='i', st_all_str='o')
print(f'{dc_ls_combo_type_c=}')

# provide name for list
dc_ls_combo_type_e= pyfan_amto_lsdcconvert.ff_ls2dc(ls_list=ls_combo_type, st_counter_
↪str='CTR', st_all_str='OF', st_ls_name='ls_other_name')
print(f'{dc_ls_combo_type_d=}')

# check three calling methods all work
print(f'{dc_ls_combo_type_a==dc_ls_combo_type_b=}')
print(f'{dc_ls_combo_type_a==dc_ls_combo_type_c=}')
print(f'{dc_ls_combo_type_a==dc_ls_combo_type_d=}')
print(f'{dc_ls_combo_type_a==dc_ls_combo_type_e=}')

# Start Plot
fig, ax = plt.subplots()

# Text Plot
ax.text(0.5, 0.5,
        f'ls_combo_type is:\n{textwrap.fill(str(ls_combo_type), width=80)}'
        f'\n\n'
        f'dc_ls_combo_type_c is:\n{textwrap.fill(str(dc_ls_combo_type_c), width=80)}',
        horizontalalignment='center',
        verticalalignment='center',
        fontsize=10, color='black',
        transform=ax.transAxes)

# Labeling
ax.set_axis_off()
plt.show()
```

Out:

```
dc_ls_combo_type_a={'ls_combo_type_i0o5': 'e', 'ls_combo_type_i1o5': '20201025x_esr_
↪list_tKap_mlt_ce1a2', 'ls_combo_type_i2o5': ['esti_param.kappa_ce9901', 'esti_param.
↪kappa_ce0209'], 'ls_combo_type_i3o5': 1, 'ls_combo_type_i4o5': 'C1E31M3S3=1'}
('dc_ls_combo_type_b '
 '= '
 'pyfan_amto_lsdcconvert.ff_ls2dc(ls_combo_type, '
 "'i', "
 "'o', "
 'verbose=True)')
{'ls_combo_type_i0o5': 'e',
 'ls_combo_type_i1o5': '20201025x_esr_list_tKap_mlt_ce1a2',
 'ls_combo_type_i2o5': ['esti_param.kappa_ce9901',
                        'esti_param.kappa_ce0209'],
 'ls_combo_type_i3o5': 1,
 'ls_combo_type_i4o5': 'C1E31M3S3=1'}
dc_ls_combo_type_a={'ls_combo_type_i0o5': 'e', 'ls_combo_type_i1o5': '20201025x_esr_
↪list_tKap_mlt_ce1a2', 'ls_combo_type_i2o5': ['esti_param.kappa_ce9901', 'esti_param.
↪kappa_ce0209'], 'ls_combo_type_i3o5': 1, 'ls_combo_type_i4o5': 'C1E31M3S3=1'}
dc_ls_combo_type_b={'ls_combo_type_i0o5': 'e', 'ls_combo_type_i1o5': '20201025x_esr_
↪list_tKap_mlt_ce1a2', 'ls_combo_type_i2o5': ['esti_param.kappa_ce9901', 'esti_param.
↪kappa_ce0209'], 'ls_combo_type_i3o5': 1, 'ls_combo_type_i4o5': 'C1E31M3S3=1'}
dc_ls_combo_type_c={'ls_combo_type_i0o5': 'e', 'ls_combo_type_i1o5': '20201025x_esr_
↪list_tKap_mlt_ce1a2', 'ls_combo_type_i2o5': ['esti_param.kappa_ce9901', 'esti_param.
↪kappa_ce0209'], 'ls_combo_type_i3o5': 1, 'ls_combo_type_i4o5': 'C1E31M3S3=1'}
dc_ls_combo_type_d={'ls_combo_type_i0o5': 'e', 'ls_combo_type_i1o5': '20201025x_esr_
↪list_tKap_mlt_ce1a2', 'ls_combo_type_i2o5': ['esti_param.kappa_ce9901', 'esti_param.
↪kappa_ce0209'], 'ls_combo_type_i3o5': 1, 'ls_combo_type_i4o5': 'C1E31M3S3=1'}
dc_ls_combo_type_a==dc_ls_combo_type_b=True
dc_ls_combo_type_a==dc_ls_combo_type_c=True
dc_ls_combo_type_a==dc_ls_combo_type_d=True
dc_ls_combo_type_a==dc_ls_combo_type_e=False
```

Total running time of the script: (0 minutes 0.062 seconds)

10.2 Data Structures Examples

10.2.1 Draw Shock Grid

In this example, we draw shock grids.

```
# Author: Fan Wang (fanwangecon.github.io)
import numpy as np
import matplotlib.pyplot as plt
import pyfan.gen.rand.randgrid as pyfan_gen_rand
```

Shared parameters

```
fl_mu = 0
fl_sd = 1
it_draws = 25
it_seed = 123
fl_lower_sd = -2
fl_higher_sd = 2
```

Type 0 Shock draw

```
it_draw_type = 0
ar_shock_t0 = \
    pyfan_gen_rand.ar_draw_random_normal(fl_mu, fl_sd, it_draws,
                                         it_seed, it_draw_type,
                                         fl_lower_sd, fl_higher_sd)
print('it_draw_type=0')
print(ar_shock_t0)
```

Out:

```
it_draw_type=0
[-1.0856306   0.99734545  0.2829785  -1.50629471 -0.57860025  1.65143654
 -2.         -0.42891263  1.26593626 -0.8667404  -0.67888615 -0.09470897
  1.49138963 -0.638902   -0.44398196 -0.43435128  2.          2.
  1.0040539   0.3861864   0.73736858  1.49073203 -0.93583387  1.17582904
 -1.25388067]
```

Type 1 Shock draw

```
it_draw_type = 1
ar_shock_t1 = \
    pyfan_gen_rand.ar_draw_random_normal(fl_mu, fl_sd, it_draws,
                                         it_seed, it_draw_type,
                                         fl_lower_sd, fl_higher_sd)
print('it_draw_type=1')
print(ar_shock_t1)
```

Out:

```
it_draw_type=1
[-2.         -1.53395018 -1.26860059 -1.07109838 -0.90840016 -0.76678646
 -0.63911191 -0.52110766 -0.40996255 -0.30367558 -0.20072104 -0.09985637
  0.          0.09985637  0.20072104  0.30367558  0.40996255  0.52110766
  0.63911191  0.76678646  0.90840016  1.07109838  1.26860059  1.53395018
  2.        ]
```

Type 2 Shock draw

```
it_draw_type = 2
ar_shock_t2 = \
    pyfan_gen_rand.ar_draw_random_normal(fl_mu, fl_sd, it_draws,
                                         it_seed, it_draw_type,
                                         fl_lower_sd, fl_higher_sd)
print('it_draw_type=2')
print(ar_shock_t2)
```

Out:

```
it_draw_type=2
[-2.         -1.50629471 -1.25388067 -1.0856306  -0.93583387 -0.8667404
 -0.67888615 -0.638902   -0.57860025 -0.44398196 -0.43435128 -0.42891263
 -0.09470897  0.2829785   0.3861864   0.73736858  0.99734545  1.0040539
  1.17582904  1.26593626  1.49073203  1.49138963  1.65143654  2.
  2.        ]
```

Draw Shocks Jointly

```
fig, ax = plt.subplots()
# Graph
ar_it_x_grid = np.arange(1, it_draws + 1)
ax.plot(ar_it_x_grid, ar_shock_t0,
                      color='blue', linestyle='dashed', marker='x',
                      label='Type 0: Bounded Shock Draws')
ax.scatter(ar_it_x_grid, ar_shock_t1,
                      color='red',
                      label='Type 1: Quantile Points')
ax.plot(ar_it_x_grid, ar_shock_t2,
                      color='black', marker='d',
                      label='Type 3: Sorted Bounded Shock Draws')
# Labeling
ax.legend(loc='upper left')
plt.ylabel('Shock Values')
plt.xlabel('Shock Draw Points')
plt.title('Shock, Sorted and Bounded Shocks, Quantile Points')
plt.grid()
plt.show()
```

Total running time of the script: (0 minutes 0.115 seconds)

10.3 Graph Examples

10.3.1 Generate Graphs using the Generic Graphing Tool

In this example, we generate a line plot, a density plot and a scatter plot.

```
# Author: Fan Wang (fanwangecon.github.io)
import pyfan.graph.generic.allpurpose as pyfan_graph_allpurpose
import numpy as np
```

Plot Time Series Lines of Temperatures in Two Cities

```
# construct data inputs
np.random.seed(0)
it_days = 365
ar_x = np.linspace(1, 365, it_days)
ar_y1 = np.random.normal(25, 3, it_days)
ar_y2 = np.random.normal(15, 5, it_days)
mt_y = np.column_stack((ar_y1, ar_y2))

# graphing class object instance
co_grapher = pyfan_graph_allpurpose.graphFunc()
co_grapher.xyPlotMultiYOneX(xData=ar_x, yDataMat=mt_y,
                            basicTitle="Temperature Flucations Two Cities",
                            basicXLabel="days of the year",
                            basicYLabel="daily temperatures",
                            labelArray=["city 1, mean=25, sd=3",
                                        "city 2, mean=15, sd=5"], noLabel=False,
                            graphType='plot',
                            saveOrNot=False, showOrNot=False)
```

Out:

```
<module 'pylab' from 'G:\\ProgramData\\Anaconda3\\envs\\wk_main\\lib\\site-packages\\
↪pylab.py'>
```

Plot Three Densities of Test Score Distributions

```
# construct data inputs
np.random.seed(0)
it_students_perclass = 100
ar_student_id = np.arange(it_students_perclass)
ar_class_a_tests = np.random.normal(80, 3, it_students_perclass)
ar_class_b_tests = np.random.normal(75, 10, it_students_perclass)
ar_class_c_tests = np.random.normal(50, 20, it_students_perclass)
mt_y = np.column_stack((ar_class_a_tests, ar_class_b_tests, ar_class_c_tests))

# graphing class object instance
co_grapher = pyfan_graph_allpurpose.graphFunc()
co_grapher.xyPlotMultiYOneX(xData=ar_x, yDataMat=mt_y,
                            basicTitle="Test Score Densities (100 students per class)
↪",
```

(continues on next page)

(continued from previous page)

```
                              basicXLabel="Test Scores",
                              basicYLabel="Densities",
                              labelArray=["Class 1", "Class 2", "Class 3"],␣
↪noLabel=False,
                              graphType='density',
                              saveOrNot=False, showOrNot=False)
```

Out:

```
<module 'pylab' from 'G:\\ProgramData\\Anaconda3\\envs\\wk_main\\lib\\site-packages\\
↪pylab.py'>
```

Plot a Scatter Plot of the Relationship Between Wage and Education

```
# construct data inputs
np.random.seed(0)
it_worker_obs = 100
ar_worker_edu = np.random.choice(18, it_worker_obs);
ar_log_wage_shock = np.random.normal(0, 0.2, it_worker_obs)
ar_worker_wage = np.exp(2 + ar_worker_edu*0.05 + ar_log_wage_shock)

# graphing class object instance
co_grapher = pyfan_graph_allpurpose.graphFunc()
co_grapher.xyPlotMultiYOneX(xData=ar_worker_edu, yDataMat=ar_worker_wage,
                              basicTitle="Hourly Wage and Years of Education",
                              basicXLabel="Years of Schooling",
                              basicYLabel="Hourly Wage",
                              graphType='scatter', scattersize=10,
                              saveOrNot=False, showOrNot=False)
```

Out:

```
<module 'pylab' from 'G:\\ProgramData\\Anaconda3\\envs\\wk_main\\lib\\site-packages\\
↪pylab.py'>
```

Total running time of the script: (0 minutes 0.344 seconds)

10.4 Stats Examples

10.4.1 Markov Transition Probability Check and Transform

In this example, use several markov transition matrixes where each row does not sum up to 1 due to simulation errors. Check if the gap between 1 and the row values are too big, and then normalize.

```
# Author: Fan Wang (fanwangecon.github.io)
import pyfan.stats.markov.transprobcheck as pyfan_stats_transprobcheck
import numpy as np
import matplotlib.pyplot as plt
```

Check Row Sum of a Five by Five Transition matrix

```
# construct data inputs
mt_ar1_trans = np.array([[0.4334, 0.5183, 0.0454, 0.0027, 0.0002],
                         [0.2624, 0.5967, 0.1245, 0.0145, 0.0016],
                         [0.1673, 0.5918, 0.2005, 0.0343, 0.0052],
                         [0., 0.0312, 0.6497, 0.2774, 0.0371],
                         [0., 0.0681, 0.6569, 0.2379, 0.0327],
                         [0., 0.2201, 0.581, 0.168, 0.0264]])
ar_row_sums_ar1 = np.sum(mt_ar1_trans, axis=1)
print(f'{ar_row_sums_ar1=}')

# Check with default conditions, does not pass
bl_ar1_sum_pass, bl_per_row_pass, bl_avg_row_pass = \
    pyfan_stats_transprobcheck.markov_trans_prob_check(mt_ar1_trans)
print(f'{bl_ar1_sum_pass=}')
print(f'{bl_per_row_pass=}')
print(f'{bl_avg_row_pass=}')

# Check with relaxed conditions, pass per row does not pass average
fl_atol_per_row = 1e-02
fl_atol_avg_row = 1e-03
bl_ar1_sum_pass, bl_per_row_pass, bl_avg_row_pass = \
    pyfan_stats_transprobcheck.markov_trans_prob_check(mt_ar1_trans, fl_atol_per_row,␣
↪fl_atol_avg_row)
print(f'{bl_ar1_sum_pass=}')
print(f'{bl_per_row_pass=}')
print(f'{bl_avg_row_pass=}')

# Relax condition further, passes
fl_atol_per_row = 1e-02
fl_atol_avg_row = 5e-03
bl_ar1_sum_pass, bl_per_row_pass, bl_avg_row_pass = \
    pyfan_stats_transprobcheck.markov_trans_prob_check(mt_ar1_trans, fl_atol_per_row,␣
↪fl_atol_avg_row)
print(f'{bl_ar1_sum_pass=}')
print(f'{bl_per_row_pass=}')
print(f'{bl_avg_row_pass=}')

# Start Plot
fig, ax = plt.subplots()

# Text Plot
ax.text(0.5, 0.5, f'{mt_ar1_trans} '
                  f'\n\n {fl_atol_per_row=} and {fl_atol_avg_row=} '
                  f'\n\n {bl_ar1_sum_pass=} \n {bl_per_row_pass=} \n {bl_avg_row_
↪pass=}',
        horizontalalignment='center',
        verticalalignment='center',
        fontsize=10, color='black',
        transform=ax.transAxes)

# Labeling
ax.set_axis_off()
plt.show()
```

Out:

```
ar_row_sums_ar1=array([1.    , 0.9997, 0.9991, 0.9954, 0.9956, 0.9955])
bl_ar1_sum_pass=False
bl_per_row_pass=False
bl_avg_row_pass=False
bl_ar1_sum_pass=False
bl_per_row_pass=True
bl_avg_row_pass=False
bl_ar1_sum_pass=True
bl_per_row_pass=True
bl_avg_row_pass=True
```

Rescale a Three by Three Transition so Each Row Sums to One

```
mt_ar1_trans = np.array([[0.4334, 0.5183, 0.0454], [0.2624, 0.5967, 0.1245], [0.1673,␣
↪0.5918, 0.2005]])
bl_ar1_sum_pass, bl_per_row_pass, bl_avg_row_pass = pyfan_stats_transprobcheck.markov_
↪trans_prob_check(mt_ar1_trans)
mt_ar1_trans_rescaled = pyfan_stats_transprobcheck.markov_condi_prob2one(mt_ar1_trans)
bl_ar1_sum_pass_rescaled, bl_per_row_pass_rescaled, bl_avg_row_pass_rescaled = \
    pyfan_stats_transprobcheck.markov_trans_prob_check(mt_ar1_trans_rescaled)

# Start Plot
fig, ax = plt.subplots()

# Text Plot
ax.text(0.5, 0.5, f'{mt_ar1_trans} '
                  f'\n\n {bl_ar1_sum_pass=}'
                  f'\n\n {mt_ar1_trans_rescaled}'
                  f'\n\n {bl_ar1_sum_pass_rescaled=}',
        horizontalalignment='center',
        verticalalignment='center',
        fontsize=10, color='black',
        transform=ax.transAxes)

# Labeling
ax.set_axis_off()
plt.show()
```

Total running time of the script: (0 minutes 0.091 seconds)

10.5 Support Utilities Examples

10.5.1 Generate PDFs and Clean

In this example, we generate PDFs in one location from tex files in possibly various other locations, and clean.

```
# Author: Fan Wang (fanwangecon.github.io)
import pyfan.util.pdf.pdfgen as pyfan_pdfgen
import pprint
import matplotlib.pyplot as plt
import textwrap
import json
```

Generate PDF for one specific file and clean afterwards

```
# spt_loc = 'C:/Users/fan/Documents/Dropbox (UH-ECON)/repos/Tex4Econ/_other/equation/'
spt_loc = 'G:/Dropbox (UH-ECON)/repos/Tex4Econ/_other/equation/'
spt_loc_output = 'C:/Users/fan/Documents/'
spn_file = 'cases.tex'
spn_pdf_exe = 'C:/texlive/2020/bin/win32/pdflatex.exe'
dc_tex_pdf_a = pyfan_pdfgen.ff_pdf_gen_clean(ls_spt_srh=[spt_loc], spt_out=spt_loc_
↪output,
                                             spn_pdf_exe=spn_pdf_exe, ls_st_
↪contain=[spn_file],
                                             bl_clean=True)
print(dc_tex_pdf_a)
```

Out:

```
{}
```

Generate PDF from all tex files in all subfolders of a main folder, output PDF store in one location

1. spt_loc_search_root: Tex Search folder
2. spt_loc_output: only consider files with this in name
3. st_search_string: include in one of the element in list
4. ls_st_ignore: ignore files with this in name
5. PDF Destination Folder: same root path earlier folder to store possibly

```
# spt_loc_search_root = 'C:/Users/fan/Documents/Dropbox (UH-ECON)/repos/Tex4Econ/_
↪other/'
spt_loc_search_root = 'G:/Dropbox (UH-ECON)/repos/Tex4Econ/_other/'
spt_loc_output = 'C:/Users/fan/Documents/'
st_search_string = ['fs_', 'cases']
ls_st_ignore = ['tikz', 'pandoc']
spn_pdf_exe = 'C:/texlive/2020/bin/win32/pdflatex.exe'
dc_tex_pdf_b = pyfan_pdfgen.ff_pdf_gen_clean(ls_spt_srh=[spt_loc_search_root], spt_
↪out=spt_loc_output,
                                             spn_pdf_exe=spn_pdf_exe,
                                             ls_st_contain=st_search_string, ls_st_
↪ignore=ls_st_ignore,
                                             bl_recursive=True, bl_clean=True)
print(dc_tex_pdf_b)
```

Out:

```
{}
```

perl latexpand example

use latexpand conda activate wk_perl cd "C:/Users/fan/Documents/Dropbox (UH-ECON)/repos/HgtOptiAlloDraft/zmain/" perl "C:/Users/fan/.conda/envs/wk_perl/latexpand/latexpand" draft_main_s1.tex > draft_main_s1_flat.tex perl "C:/ProgramData/Anaconda3/envs/wk_perl/latexpand/latexpand" draft_main_s1.tex > draft_main_s1_flat.tex pandoc –bibliography=C:/Users/fan/HgtOptiAlloDraft/_bib/zoteroref.bib -o draft_main_s1_flat.docx draft_main_s1_flat.tex

cd "C:/Users/fan/Documents/Dropbox (UH-ECON)/repos/HgtOptiAlloDraft/beamer/" perl "C:/Users/fan/.conda/envs/wk_perl/latexpand/latexpand" present.tex > present_flat.tex perl "C:/ProgramData/Anaconda3/envs/wk_perl/latexpand/latexpand" present.tex > present_flat.tex

pandoc –bibliography=C:/Users/fan/HgtOptiAlloDraft/_bib/zoteroref.bib -o present_flat.docx present_flat.tex

Plot String as Figure

```
# Dict of String to String
str_dc_records = 'One Tex to Root PDF:'.upper() + '\n' + \
                 textwrap.fill(json.dumps(dc_tex_pdf_a), width=70) + '\n' + \
                 'Recursive Search Tex to PDF Folder:'.upper() + '\n' + \
                 textwrap.fill(json.dumps(dc_tex_pdf_b), width=70)
# Start Plot
fig, ax = plt.subplots()

# Text Plot
ax.text(0.5, 0.5, str_dc_records,
        horizontalalignment='center',
        verticalalignment='center',
        fontsize=14, color='black',
        transform=ax.transAxes)

# Labeling
ax.set_axis_off()
plt.show()
```

Total running time of the script: (0 minutes 0.044 seconds)

CHAPTER

ELEVEN

API REFERENCE

This page contains auto-generated API reference documentation[1].

11.1 pyfan

11.1.1 Subpackages

pyfan.amto

Subpackages

pyfan.amto.array

Submodules

pyfan.amto.array.geomspace

Created on May 24, 2018

@author: fan

To have a better grid denser at the beginning

Module Contents

Functions

grid_to_geom_short(choice_grid, choice_grid_max, choice_grid_min, start, stop, num, geom_ratio, a)	
grid_to_geom_short_core(choice_grid, a, scaler, displacement, multiplier, geom_ratio)	
grid_to_geom(choice_grid, choice_grid_max, choice_grid_min, start, stop, num, geom_ratio, a)	the code now is under the assumption that initial start and end were 0 and 1
gen_geom_grid(start, stop, num, geom_ratio, a)	Specify geom_ratio, the z below:

continues on next page

[1] Created with sphinx-autoapi

Table 1 – continued from previous page

tester(a=1, b=51, max_power=49)	1. 1 to 51, geomspace
tester_plus1(a=0, b=50, max_power=49, adjust=1)	to accomndate zero,

pyfan.amto.array.geomspace.**logger**

pyfan.amto.array.geomspace.**grid_to_geom_short**(*choice_grid*, *choice_grid_max*, *choice_grid_min*, *start*, *stop*, *num*, *geom_ratio*, *a*)

pyfan.amto.array.geomspace.**grid_to_geom_short_core**(*choice_grid*, *a*, *scaler*, *displacement*, *multiplier*, *geom_ratio*)

pyfan.amto.array.geomspace.**grid_to_geom**(*choice_grid*, *choice_grid_max*, *choice_grid_min*, *start*, *stop*, *num*, *geom_ratio*, *a*)

the code now is under the assumption that initial start and end were 0 and 1

Given geom_grid results, how do we go back to actual data grid. So for interpolation. interpolate not on actual K and B scales, but on any even grid, as long as the grid count is right.

interp_K_grid = np.linspace(0,1,n)

but then there is a vector of actual choices kn_vec, how to map kn_vec to interp_K_grid?

Parameters

choice_grid: this is the choice grid, on the actual choice scale

start: float from gen_geom_grid

stop: float from gen_geom_grid

num: int from gen_geom_grid

geom_ratio: float from gen_geom_grid

pyfan.amto.array.geomspace.**gen_geom_grid**(*start*, *stop*, *num*, *geom_ratio*, *a*)

Specify geom_ratio, the z below: a*z^0=a a*z^1 a*z^2 ... ... a*z^49=b

Then generate the grid points that is consistent with the geom_ratio

Parameters

start: float same as in linspace

stop: float same as in linspace

num: int same as in linspace

geom_ratio: float z value below kind of except for rescaling

pyfan.amto.array.geomspace.**tester**(*a=1*, *b=51*, *max_power=49*)

1. 1 to 51, geomspace

pyfan.amto.array.geomspace.**tester_plus1**(*a=0*, *b=50*, *max_power=49*, *adjust=1*)

to accomndate zero,

pyfan.amto.array.geomspace.**FORMAT = %(filename)s - %(funcName)s - %(lineno)d - %(asctime)s**

pyfan.amto.array.gridminmax

Created on Nov 27, 2017

@author: fan

Module Contents

Functions

three_vec_grids(vara_min, vara_max, vara_grid, vara_grid_add=None, varb_min=None, varb_max=None, varb_grid=None, varb_grid_add=None, varc_min=None, varc_max=None, varc_grid=None, varc_grid_add=None, gridtype='grid', tomesh=False, return_joint=False, return_single_col=False, seed=999)	Grid for VFI
random_vector_mean_sd(mean, sd, grid_count, gridtype='grid', seed=382)	
random_vector_min_max(minval, maxval, grid_count)	

pyfan.amto.array.gridminmax.**three_vec_grids**(*vara_min*, *vara_max*, *vara_grid*, *vara_grid_add=None*, *varb_min=None*, *varb_max=None*, *varb_grid=None*, *varb_grid_add=None*, *varc_min=None*, *varc_max=None*, *varc_grid=None*, *varc_grid_add=None*, *gridtype='grid'*, *tomesh=False*, *return_joint=False*, *return_single_col=False*, *seed=999*)

Grid for VFI Temporary code, so that I can deal with minimal school hour. should be deleted in the future. and combined with the method above

pyfan.amto.array.gridminmax.**random_vector_mean_sd**(*mean*, *sd*, *grid_count*, *gridtype='grid'*, *seed=382*)

pyfan.amto.array.gridminmax.**random_vector_min_max**(*minval*, *maxval*, *grid_count*)

pyfan.amto.array.gridminmax.**vara_min = 1**

pyfan.amto.array.mesh

Created on Nov 26, 2017

@author: fan

Most type of state grid generation: Given N Vectors,

Module Contents

Functions

two_mat_mesh(mat_one, mat_two, return_joint=False, return_single_col=False)	**Parameters**
three_mat_mesh(mat_one, mat_two, mat_three, return_joint=False, return_single_col=False)	**Parameters**
multipl_mat_mesh(mat_one, mat_two, mat_three=None, mat_four=None, mat_five=None, mat_six=None)	
check_length(mat)	

pyfan.amto.array.mesh.**logger**
Created on Mar 17, 2017

@author: fan

pyfan.amto.array.mesh.**two_mat_mesh**(*mat_one*, *mat_two*, *return_joint=False*, *return_single_col=False*)

Parameters

return_single_col: boolean mat_one and mat_two are single vector, shape them into 2d array with 1 column, rather than 1d. If not, could cause multiplication problems when we have both 1 column 2d array and single column 1d array in the same formula. But this can not always to set to True, hence default is actually false, because this function could take as input a matrix for mat_one, in that case, already 2d array.

pyfan.amto.array.mesh.**three_mat_mesh**(*mat_one*, *mat_two*, *mat_three*, *return_joint=False*, *return_single_col=False*)

Parameters

return_single_col: boolean mat_one and mat_two are single vector, shape them into 2d array with 1 column, rather than 1d. If not, could cause multiplication problems when we have both 1 column 2d array and single column 1d array in the same formula. But this can not always to set to True, hence default is actually false, because this function could take as input a matrix for mat_one, in that case, already 2d array.

pyfan.amto.array.mesh.**multipl_mat_mesh**(*mat_one*, *mat_two*, *mat_three=None*, *mat_four=None*, *mat_five=None*, *mat_six=None*)

pyfan.amto.array.mesh.**check_length**(*mat*)

pyfan.amto.array.mesh.**mat_one**

pyfan.amto.array.scalararray

Created on Dec 2, 2017

@author: fan

Module Contents

Functions

scalar_to_2darray(x, check_first=True)	
scalar_to_array(x, check_first=True)	
zero_ndims(ndims_var)	

Parameters

pyfan.amto.array.scalararray.**scalar_to_2darray**(*x*, *check_first=True*)

pyfan.amto.array.scalararray.**scalar_to_array**(*x*, *check_first=True*)

pyfan.amto.array.scalararray.**zero_ndims**(*ndims_var*)

Parameters

ndims_var: array the dimension of this array to be duplicated

pyfan.amto.json

Submodules

pyfan.amto.json.json

Created on Jun 4, 2018

@author: fan

Module Contents

Functions

json_serial(obj)	JSON serializer for objects not serializable by default json code
jdump(aws_return_dict, desc='', logger=None, print_here=False)	

pyfan.amto.json.json.**logger**

pyfan.amto.json.json.**json_serial**(*obj*)
JSON serializer for objects not serializable by default json code

pyfan.amto.json.json.**jdump**(*aws_return_dict*, *desc=''*, *logger=None*, *print_here=False*)

pyfan.amto.lsdc

Submodules

pyfan.amto.lsdc.lsdcconvert

The *pyfan.amto.lsdc.lsdcconvert* module provides list and dict converters.

Created on Dec 18, 2020

import pyfan.amto.lsdc.lsdcconvert as pyfan_amto_lsdcconvert

Includes method `ff_decimal_rounder_uncommon()` and `ff_decimal_rounder()`.

Module Contents

Functions

ff_ls2dc(ls_list, st_counter_str='i', st_all_str='o', st_ls_name=None, verbose=False)	Convert list to dict with list name and index and dict keys

pyfan.amto.lsdc.lsdcconvert.**ff_ls2dc**(*ls_list*, *st_counter_str='i'*, *st_all_str='o'*, *st_ls_name=None*, *verbose=False*)

Convert list to dict with list name and index and dict keys

Parameters

ls_list [list] A list of values.

st_counter_str [str] String prefix for list counter in dictionary name.

st_all_str [str] String prefix in front of total ele length in dict key name.

Returns

dict A dictionary of equal length to *ls_list* input, list converted to dict.

pyfan.amto.numeric

Submodules

pyfan.amto.numeric.round

The *pyfan.amto.numeric.round* provides decimal rounding for float arrays.

Given an array of numbers, provide conditional decimal formatting rounding via fstring. This is used by table function to generate table specific rounding rules.

For example, a table with birthweight in grams, and ratios, might have 2 decimals for numbers less than 1, but no decimals for numbers larger than 1000 (which are the grams).

import pyfan.amto.numeric.round as pyfan_amto_round

Created on Dec 14, 2020

Includes method *ff_decimal_rounder_uncommon()* and *ff_decimal_rounder()*.

Module Contents

Functions

ff_decimal_rounder(ls_fl_num2format, it_or_dc_round_decimal, verbose=False)	Decimal rounding function with common decimal formatting
ff_decimal_rounder_uncommon(ls_fl_num2format=[0.0012345, 0.12345, 12.345, 123.45, 1234.5, 123456.789], dc_round_decimal={0.1: 4, 1: 3, 100: 2, float('inf'): 0}, verbose=False)	Decimal rounding function with conditional formatting by number size

pyfan.amto.numeric.round.**ff_decimal_rounder**(*ls_fl_num2format*, *it_or_dc_round_decimal*, *verbose=False*)

Decimal rounding function with common decimal formatting

Parameters

ls_fl_num2format [list of float] see *ff_decimal_rounder()*

it_or_dc_round_decimal [int or dict] the number of decimal points to keep. If dict, same as dc_round_decimal for *ff_decimal_rounder()*. If decimal, generate dict that provides common formating

Returns

list of str Decimal formatted string outputs

pyfan.amto.numeric.round.**ff_decimal_rounder_uncommon**(*ls_fl_num2format=[0.0012345, 0.12345, 12.345, 123.45, 1234.5, 123456.789]*, *dc_round_decimal={0.1: 4, 1: 3, 100: 2, float('inf'): 0}*, *verbose=False*)

Decimal rounding function with conditional formatting by number size

Given an array of numbers, format and return as a list of string, with different decimal formatting given different number sizes.

Parameters

ls_fl_num2format [list of float] list of numbers of approximate to decimals

dc_round_decimal [dict] dict incremental formatter. For example, for the default, if below 0.1 keep 4 decimals, If below 1 keep 3, if below 100 keep 2, if otherwise above, then keep 0 decimals Loop over formatter.

Returns

list of str Decimal formatted string outputs

pyfan.aws

Subpackages

pyfan.aws.general

Submodules

pyfan.aws.general.credentials

Module Contents

Functions

boto3_start_service(st_aws_service='s3')	

pyfan.aws.general.credentials.**boto3_start_service**(*st_aws_service='s3'*)

pyfan.aws.general.path

The *pyfan.aws.general.path* file paths etc

Includes method *detect_store_path()*, *save_img()*.

Module Contents

Functions

detect_store_path(bl_check_path_exist=True, srt_sub_path=None, st_local_path=None)	Detects checks if program is running on an AWS Linux Instance
save_img(plt, sna_image_name, spt_image_path=None, dpi=300, papertype='a4', orientation='horizontal', bl_upload_s3=False, st_s3_bucket=None, srt_s3_bucket_folder=None)	Saves Graph Locally, and also upload to S3 if requested

pyfan.aws.general.path.**detect_store_path**(*bl_check_path_exist=True*, *srt_sub_path=None*, *st_local_path=None*)

Detects checks if program is running on an AWS Linux Instance

In our case, all code that run on AWS linux are running inside conda containers. If running on container, save to data folder. If running on some local machine save results to the user's home path's download folder, data subfolder.

Parameters

bl_check_path_exist [*bool*] checking saving path if it does not exist

srt_sub_path: `string`, optional this is the subpath to be used, in the data folder in EC2 container, or inside the downloads data folder under user directory.

st_local_path: `string`, optional local overriding string save path, if not, use download/data folder. This will replace the local path

Returns

tuple[bool, string] returns boolean if on amzn splatform, then the directory where to store save files

pyfan.aws.general.path.**save_img**(*plt*, *sna_image_name*, *spt_image_path=None*, *dpi=300*, *papertype='a4'*, *orientation='horizontal'*, *bl_upload_s3=False*, *st_s3_bucket=None*, *srt_s3_bucket_folder=None*)

Saves Graph Locally, and also upload to S3 if requested

Given figure object,

Parameters

plt: `matplotlib.pyplot` a matplotlib pyplot object from a graph that was just generated

sna_image_name: `string` image name, without the suffix of png

spt_image_path: `string`, optional path to image, if None, then use default local path in *detect_store_path()*

dpi: `integer`, optional image dpi

papertype: `string`, optional One of 'letter', 'legal', 'executive', 'ledger', 'a0' through 'a10', 'b0' through 'b10'.

orientation: `string`, optional 'horizontal' or 'portrait'

bl_upload_s3: `bool`, optional if file will be uploaded to s3

st_s3_bucket: `string`, optional Assuming that AWS credentials are already stored in the container on EC2 or locally in a .aws credential file. So *st_s3_bucket* bucket name refers to bucket in the credentialed user's s3 account.

srt_s3_bucket_folder: `string`, optional folder in s3 bucket to store image

Returns

tuple[bool, string] returns boolean if on amzn splatform, then the directory where to store save files

pyfan.aws.s3

Submodules

pyfan.aws.s3.pushsync

The *pyfan.aws.s3.pushsync* savse files to s3 and syncs

Includes method `ar_draw_random_normal()`.

Module Contents

Functions

s3_upload(spn_img_pwdfn, st_s3_bucket='fans3testbucket', srt_s3_bucket_folder='pyfan_scatterline3')	Upload an existing file to s3

pyfan.aws.s3.pushsync.**s3_upload**(*spn_img_pwdfn*, *st_s3_bucket='fans3testbucket'*, *srt_s3_bucket_folder='pyfan_scatterline3'*)

Upload an existing file to s3

Upload to a particular bucket and subfolder, file in some local directory

Parameters

spn_img_pwdfn: `string` full path to image, including the image name

st_s3_bucket: `string`, optional Assuming that AWS credentials are already stored in the container on EC2 or locally in a .aws credential file. So *st_s3_bucket* bucket name refers to bucket in the credentialed user's s3 account.

srt_s3_bucket_folder: `string`, optional folder in s3 bucket to store image

Returns

none

Examples

```
>>> spn_img_pwdfn = 'C:/Users/fan/Downloads/data/test/test_image.png'
>>> st_s3_bucket = 'fans3testbucket'
>>> srt_s3_bucket_folder = 'pyfan_scatterline3/folder1/'
>>> s3_upload(spn_img_pwdfn, st_s3_bucket, srt_s3_bucket_folder)
```

pyfan.devel

Subpackages

pyfan.devel.flog

Submodules

pyfan.devel.flog.logsupport

The *pyfan.devel.flog.logsupport* initiates logging and set logging options, output log path points.

This is imported into other programs as *import pyfan.devel.flog.logsupport as pyfan_logsup*

Includes method *log_vig_start()*, *log_format()*

Module Contents

Functions

log_vig_start(spt_root, main_folder_name, file_name='fs_gen_combo_type', sub_folder_name=None, subsub_folder_name=None, it_time_format=8, log_level=logging.WARNING, **kwargs)	Start logging to log file
log_format(bl_set_print_opt=True, it_print_opt=1)	Logging formats

pyfan.devel.flog.logsupport.**log_vig_start**(*spt_root*, *main_folder_name*, *file_name='fs_gen_combo_type'*, *sub_folder_name=None*, *subsub_folder_name=None*, *it_time_format=8*, *log_level=logging.WARNING*, ***kwargs*)

Start logging to log file

Generate path to log file and initiate log file. Return this full path to log file. Configure the log file with formating

Parameters

spt_root [str] folder root to log file.

main_folder_name [str] main folder, appended to *spt_root*.

file_name [str] file name for the log file, without suffix.

sub_folder_name [str, optional] possible subfolder name. This is double pound vig level.

subsub_folder_name [str, optional] possible subsub folder name. try not to have lower than this level. This is triple pound vig level.

it_time_format [int] different types of time formatting, if *it_time_format* is zero, no time suffix

log_level [int] logging level integers to report, including CRITICAL 50 ERROR 40 WARNING 30 INFO 20 DEBUG 10 NOTSET 0.

****kwargs** Arguments for functions that is called, including *log_format()*

Returns

str return the path to the log file

Examples

```
>>> log_vig_start(spt_root = 'C:/Users/fan/',
...                main_folder_name='logvig', sub_folder_name='parameters',
...                subsub_folder_name='combo_type',
...                file_name='fs_gen_combo_type',
...                it_time_format=8, log_level=logging.INFO)
C:\Users\fan\logvig\parameters\combo_type\fs_gen_combo_type_20201030.log.py
```

pyfan.devel.flog.logsupport.**log_format**(*bl_set_print_opt=True*, *it_print_opt=1*)

Logging formats

This is called by *log_vig_start()*, with parameters fed in with *kwargs*

Parameters

bl_set_print_opt [bool, optional] If to set numpy table printing options, how many columns and decimal controls

it_print_opt [int, optional] Different possible options to set

Returns

str formatting string options for logging config

Examples

```
>>> log_format(bl_set_print_opt = True, it_print_opt = 1)
'%(filename)s - %(funcName)s - %(lineno)d -  %(asctime)s - %(levelname)s
↪%(message)s'
```

pyfan.devel.obj

Submodules

pyfan.devel.obj.classobjsupport

Created on Mar 16, 2017

@author: fan

Module Contents

Functions

dynamic_obj_attr(attribute_array=['r_save', 'r_borr', 'delta'], attribute_values_array=['0.02', '0.05', '0.10'], print_values=False)

pyfan.devel.obj.classobjsupport.**dynamic_obj_attr**(*attribute_array=['r_save', 'r_borr', 'delta']*, *attribute_values_array=['0.02', '0.05', '0.10']*, *print_values=False*)

pyfan.gen

Subpackages

pyfan.gen.rand

Submodules

pyfan.gen.rand.randgrid

The *pyfan.gen.rand.randgrid* generate a grid with randomly spaced grid points.

$$x \sim N(\mu, \sigma)$$

Includes method *ar_draw_random_normal()*.

Module Contents

Functions

ar_draw_random_normal(fl_mu, fl_sd, it_draws, it_seed=None, it_draw_type=0, fl_lower_sd=-3, fl_higher_sd=None)	Draw a Vector of Possibly Sorted and Bounded Normal Shocks

pyfan.gen.rand.randgrid.**ar_draw_random_normal**(*fl_mu*, *fl_sd*, *it_draws*, *it_seed=None*, *it_draw_type=0*, *fl_lower_sd=-3*, *fl_higher_sd=None*)

Draw a Vector of Possibly Sorted and Bounded Normal Shocks

Parameters

fl_mu, fl_sd [*float*] The mean and standard deviation of the normal process

it_draws: `int` Number of Draws

it_seed: `int`, optional External random seed externally. Default is 123.

it_draw_type: `int`, optional Indicates which type of normal draws to make. 0 sorted normal draws cut off at bounds. 1 equi-quantile unequal distance points; 2 normal draws unsorted.

fl_lower_sd, fl_higher_sd [*float*] Impose lower and upper bounds (in sd units) on shock draws. The normal distribution does not have lower or upper bounds.

Returns

numpy.array of shape (1, it_draws) A vector of sorted or unsorted random grid points, or equi-quantile points.

Notes

This method requires a dataset of equal-sized time series

Examples

```
>>> fl_mu = 0
>>> fl_sd = 1
>>> it_draws = 5
>>> it_seed = 123
>>> fl_lower_sd = -1
>>> fl_higher_sd = 0.8
>>> it_draw_type = 0
>>> ar_draw_random_normal(fl_mu, fl_sd, it_draws,
```

(continues on next page)

(continued from previous page)

```
...                         it_seed, it_draw_type,
...                         fl_lower_sd, fl_higher_sd)
[-1.          0.8         0.2829785 - 1. - 0.57860025]
```

```
>>> it_draw_type = 1
>>> ar_draw_random_normal(fl_mu, fl_sd, it_draws,
...                         it_seed, it_draw_type,
...                         fl_lower_sd, fl_higher_sd)
[-1. - 0.47883617 - 0.06672597  0.3338994   0.8]
```

```
>>> it_draw_type = 2
>>> ar_draw_random_normal(fl_mu, fl_sd, it_draws,
...                         it_seed, it_draw_type,
...                         fl_lower_sd, fl_higher_sd)
[-1. - 1. - 0.57860025  0.2829785   0.8]
```

pyfan.graph

Subpackages

pyfan.graph.exa

Submodules

pyfan.graph.exa.scatterline3

The pyfan.graph.example.scatterline3 generates a graprh with three lines. This is the functionalized vesrion of plot_randgrid Example.

Includes method *gph_scatter_line_rand()*.

Module Contents

Functions

gph_scatter_line_rand(fl_mu=0, fl_sd=1, it_draws=25, it_seed=123, fl_lower_sd=-2, fl_higher_sd=2, bl_show_fig=True, bl_save_fig=False, st_s3_bucket='fans3testbucket')	A randomly generated graph with scatter plot and lines.

pyfan.graph.exa.scatterline3.**parser**

pyfan.graph.exa.scatterline3.**args**

pyfan.graph.exa.scatterline3.**gph_scatter_line_rand**(*fl_mu=0*, *fl_sd=1*, *it_draws=25*, *it_seed=123*, *fl_lower_sd=-2*, *fl_higher_sd=2*, *bl_show_fig=True*, *bl_save_fig=False*, *st_s3_bucket='fans3testbucket'*)

A randomly generated graph with scatter plot and lines.

Parameters

fl_mu, fl_sd [*float*, optional] The mean and standard deviation of the normal process for lines

it_draws: `integer`, optional Number of Draws lines

it_seed: `integer`, optional External random seed externally. Default is 123. for lines

fl_lower_sd, fl_higher_sd [*float*, optional] Impose lower and upper bounds (in sd units) on shock draws. The normal distribution does not have lower or upper bounds.

bl_show_fig: `bool`, optional Show graph in documentation if needed. When storing graph to disc and uploading to s3, do not need to show.

Returns

pandas.DataFrame of shape (*it_draws*, 4) A pandas dataframe with *it_draws* number of rows and four columns. First for x values, the next three for three types of randomly generated variables that are been plotted out.

Examples

```
>>> fl_mu = 0
>>> fl_sd = 1
>>> it_draws = 20
>>> it_seed = 456
>>> fl_lower_sd = -1
>>> fl_higher_sd = 0.8
>>> scatter_line_rand_graph(fl_mu, fl_sd,
...                                                    it_draws, it_seed,
...                                          fl_lower_sd, fl_higher_sd)
             x    shk_t0    shk_t1    shk_t2
1    1.0 -0.668129 -2.000000 -2.000000
2    2.0 -0.498210 -1.533950 -1.130231
3    3.0  0.618576 -1.268601 -1.111846
4    4.0  0.568692 -1.071098 -0.971485
5    5.0  1.350509 -0.908400 -0.668129
6    6.0  1.629589 -0.766786 -0.498210
7    7.0  0.301966 -0.639112 -0.384060
8    8.0  0.449483 -0.521108 -0.345811
9    9.0 -0.345811 -0.409963 -0.325130
10  10.0 -0.315231 -0.303676 -0.315231
11  11.0 -2.000000 -0.200721 -0.106208
12  12.0 -1.130231 -0.099856 -0.088752
13  13.0 -1.111846  0.000000  0.237851
14  14.0  0.237851  0.099856  0.301966
15  15.0 -0.325130  0.200721  0.449483
16  16.0  1.944702  0.303676  0.568692
17  17.0  1.915676  0.409963  0.618576
18  18.0  0.920348  0.521108  0.920348
19  19.0  0.936398  0.639112  0.936398
20  20.0  1.157552  0.766786  1.139873
21  21.0 -0.106208  0.908400  1.157552
22  22.0 -0.088752  1.071098  1.350509
23  23.0 -0.971485  1.268601  1.629589
24  24.0 -0.384060  1.533950  1.915676
25  25.0  1.139873  2.000000  1.944702
```

pyfan.graph.exa.scatterline3.**it_seed_arg**

pyfan.graph.generic

Submodules

pyfan.graph.generic.allpurpose

Created on Sep 24, 2013

@author: fan

Module Contents

Classes

graphFunc	

Functions

contourAnd3D(xData, yData, zData, xLabStr, yLabStr, zLabStr, graphTitleDisp, graphTitleSave, savedpi=125, angleType=[1, [1, 2, 3]], drawContour=False, draw3D=True, draw3DSurf=False, contourXres=100, contourYres=100, s=20, alpha=0.6, subplot=None, fig=None)	
tripleAngle3dSave(ax, graphTitleDisp, xLabStr, yLabStr, zLabStr, graphTitleSave, savedpi=125, angleType=[1, [1, 2, 3]])	
grid(x, y, z, resX=100, resY=100)	Convert 3 column data to matplotlib grid
graph_emaxKCash_Value(soluSupObj, resources, k_vec, emaxValsCur, emaxChoicesCur, emaxChoiceOfMaxCollCur, predictUtil)	
OLSEmaxValAndChoicesGraphs(allDataY, allDataX, saveFileSuffix='', yLabelNames=['Emax', 'Choice'], xLabelNames=['Height', 'Weight', 'Income'], saveDirectory='default', saveFileName='default')	
OLSEmaxGraphs(saveFileSuffix, yVal, allDataX, saveDirectory='default', saveFileName='default', yLabelName='yLabelName', xLabelNames=['Height', 'Weight', 'Income'])	
guassian_kde_graph(data_fordensity, graph_xgrid=False, xgridpoints=1000, color='b', label=False, showOnScreen=False)	
subplot_square_counter(totalimages=15)	
sampleDataGraphs()	

pyfan.graph.generic.allpurpose.**logger**

pyfan.graph.generic.allpurpose.**contourAnd3D**(*xData*, *yData*, *zData*, *xLabStr*, *yLabStr*, *zLabStr*, *graphTitleDisp*, *graphTitleSave*, *savedpi=125*, *angleType=[1, [1, 2, 3]]*, *drawContour=False*, *draw3D=True*, *draw3DSurf=False*, *contourXres=100*, *contourYres=100*, *s=20*, *alpha=0.6*, *subplot=None*, *fig=None*)

pyfan.graph.generic.allpurpose.**tripleAngle3dSave**(*ax*, *graphTitleDisp*, *xLabStr*, *yLabStr*, *zLabStr*, *graphTitleSave*, *savedpi=125*, *angleType=[1, [1, 2, 3]]*)

pyfan.graph.generic.allpurpose.**grid**(*x*, *y*, *z*, *resX=100*, *resY=100*)

Convert 3 column data to matplotlib grid

pyfan.graph.generic.allpurpose.**toGraphHere = False**

pyfan.graph.generic.allpurpose.**graph_emaxKCash_Value**(*soluSupObj*, *resources*, *k_vec*, *emaxValsCur*, *emaxChoicesCur*, *emaxChoiceOfMaxCollCur*, *predictUtil*)

pyfan.graph.generic.allpurpose.**OLSEmaxValAndChoicesGraphs**(*allDataY*, *allDataX*, *saveFileSuffix=''*, *yLabelNames=['Emax', 'Choice']*, *xLabelNames=['Height', 'Weight', 'Income']*, *saveDirectory='default'*, *saveFileName='default'*)

pyfan.graph.generic.allpurpose.**OLSEmaxGraphs**(*saveFileSuffix*, *yVal*, *allDataX*, *saveDirectory='default'*, *saveFileName='default'*, *yLabelName='yLabelName'*, *xLabelNames=['Height', 'Weight', 'Income']*)

class pyfan.graph.generic.allpurpose.**graphFunc**(*showOrNot=False*, *saveDirectory=saveDirectory*, *saveDPI=saveDPI*)

points = 200

xData

xData

xData

yData1

yData2

yDataMat

labelLoc1t0 = best

labelColCount = 1

labelArray = ['line y1', 'line y2']

basicTitle = Image Name

basicXLabel = X Title Name

basicYLabel = Y Title Name

showOrNot = False

saveDirectory = C:/Users/fan/Pictures

saveFileName = temp.png

saveDPI = 125

colorCounter = 0

xyPlotMultiYOneX(*self*, *xData=xData*, *yDataMat=yDataMat*, *colorVar=None*, *labelArray=labelArray*, *noLabel=True*, *basicTitle=basicTitle*, *basicXLabel=basicXLabel*, *basicYLabel=basicYLabel*, *labelLoc1t0=labelLoc1t0*, *labelColCount=labelColCount*, *line45Deg=False*, *showOrNot=False*, *saveOrNot=True*, *graphType='plot'*, *saveDirectory=saveDirectory*, *saveFileName=saveFileName*, *saveDPI=1000*, *toScale=True*, *pylabUse=None*, *ylim=None*, *xlim=None*, *sequential_color=False*, *subplot=None*, *clear_first=False*, ***keywords*)

Graph general

yDataMat each column corresponds to x

graphingEachType(*self*, *graphType*, *xSingleArrayData*, *ySingleArrayata*, *keywords*, *colorVar=None*, *label=False*, *color='b'*, *pylabUse=None*)

If do not use basic pylab, but have external axis

savingFig(*self*, *saveDirectory=saveDirectory*, *saveFileName=saveFileName*, *saveDPI=saveDPI*, *saveOrNot=True*, *showOrNot=False*, *pylabUse=None*, *toScale=True*, *subplots_adjust=True*)

sampleGraphs(*self*, *graphSampleType*, *graphType='plot'*)

pyfan.graph.generic.allpurpose.**guassian_kde_graph**(*data_fordensity*, *graph_xgrid=False*, *xgridpoints=1000*, *color='b'*, *label=False*, *showOnScreen=False*)

pyfan.graph.generic.allpurpose.**subplot_square_counter**(*totalimages=15*)

pyfan.graph.generic.allpurpose.**sampleDataGraphs**()

pyfan.graph.generic.allpurpose.**grapher**

pyfan.graph.tools

Submodules

pyfan.graph.tools.subplot

Created on Aug 6, 2018

@author: fan

Design page subplot

Module Contents

Functions

subplot_design(plot_count=10, base_multiple=4, base_multiple_high_frac=0.6)	subplot grid and size given total plot count

pyfan.graph.tools.subplot.**subplot_design**(*plot_count=10*, *base_multiple=4*, *base_multiple_high_frac=0.6*)

subplot grid and size given total plot count

figsize = (width height)

Examples

import Support.graph.subplot as sup_graph_subplot figsize, rows, cols = sup_graph_subplot.subplot_design(plot_count=10, base_multiple=4, base_multiple_high_frac = 0.60)

pyfan.panda

Subpackages

pyfan.panda.categorical

Submodules

pyfan.panda.categorical.catevars

Created on Aug 10, 2018

@author: fan

Module Contents

Functions

show_cates(df, varname)	

pyfan.panda.categorical.catevars.**logger**

pyfan.panda.categorical.catevars.**show_cates**(*df*, *varname*)

pyfan.panda.categorical.strsvarskeys

Created on Aug 9, 2018

@author: fan

Store key variable names, file names, etc.

Individual analysis files should refer to this file.

Module Contents

Functions

main_data_directory(dataset_name='thai_data_z')	
file_names()	
var_names(vartype=1, file_name='thaiMthly_Annulized_DataZ', out_type='coln')	
conditions(df, condi_dict=None, return_subset=False)	Commonly used conditionings

pyfan.panda.categorical.strsvarskeys.**logger**

pyfan.panda.categorical.strsvarskeys.**main_data_directory**(*dataset_name='thai_data_z'*)

pyfan.panda.categorical.strsvarskeys.**file_names**()

pyfan.panda.categorical.strsvarskeys.**var_names**(*vartype=1*, *file_name='thaiMthly_Annulized_DataZ'*, *out_type='coln'*)

pyfan.panda.categorical.strsvarskeys.**conditions**(*df*, *condi_dict=None*, *return_subset=False*)

Commonly used conditionings

Can combine conditioning statements together in list

Parameters

condi_dict: dictionary condi_dict = {'region':'central', 'years':2002-2005}

file_name: string different files could have different variable names for the same variables, although these should be unified if possible

pyfan.panda.inout

Submodules

pyfan.panda.inout.combine

Created on Aug 14, 2018

@author: fan

Combine Files Together

Module Contents

Functions

search_combine(search_directory=None, file_search_str=None, save_file_name=None, save_panda=True, return_current=False)	Estimation saves csv files in different folders. Each folder has different starting value.

pyfan.panda.inout.combine.**logger**

pyfan.panda.inout.combine.**search_combine**(*search_directory=None*, *file_search_str=None*, *save_file_name=None*, *save_panda=True*, *return_current=False*)

Estimation saves csv files in different folders. Each folder has different starting value. Different estimation method.

Gather results together, create single csv file. To find which parameters lead to smallest objective.

Parameters

return_current: boolean return_current if true, do not save file, return current file

Examples

import panda.io.combine as pd_combine all_esti_df = pd_combine.search_combine(search_directory = None,

file_search_str = None, save_file_name = None)

pyfan.panda.inout.readexport

Created on Aug 9, 2018

@author: fan

Module Contents

Functions

unflatten_denormalize(dictionary=None, dump_print=False)	https://stackoverflow.com/questions/6037503/python-unflatten-dict
read_stata(file_directory_name)	
read_file(file_key='thai_data_z')	
read_csv(csv_file_folder)	directory = 'C:/Users/fan/Documents/Dropbox (UH-ECON)/Project Dissertation/model_test/test_solu/'
read_file_main(file_name, format_suffix, directory, data_format='stata', show_columns=True)	
dict_to_panda(list_of_dict, save_file_directory=None)	

pyfan.panda.inout.readexport.**logger**

pyfan.panda.inout.readexport.**unflatten_denormalize**(*dictionary=None*, *dump_print=False*)
https://stackoverflow.com/questions/6037503/python-unflatten-dict json_normalize results in dict that has dots for nests revert back

Examples

import panda.io.readexport as readexport dictionary = {} nested_dict = readexport.unflatten_denormalize(dictionary)

pyfan.panda.inout.readexport.**read_stata**(*file_directory_name*)

pyfan.panda.inout.readexport.**read_file**(*file_key='thai_data_z'*)

pyfan.panda.inout.readexport.**read_csv**(*csv_file_folder*)
directory = 'C:/Users/fan/Documents/Dropbox (UH-ECON)/Project Dissertation/model_test/test_solu/' param_combo = '20180701_basicJ7_basic' file_name = '**solu_**'+param_combo+'.csv' csv_file_folder = directory + '/' + file_name solu_opti_pd = proj_sys_sup.read_csv(csv_file_folder)

pyfan.panda.inout.readexport.**read_file_main**(*file_name*, *format_suffix*, *directory*, *data_format='stata'*, *show_columns=True*)

pyfan.panda.inout.readexport.**dict_to_panda**(*list_of_dict*, *save_file_directory=None*)

pyfan.panda.stats

Submodules

pyfan.panda.stats.cutting

Created on Aug 13, 2018

@author: fan

import panda.cutting as pd_cut

Module Contents

Functions

pd_winsorize_columnwise(df, winsor_coln_list, coln_perc_cutoffs, return_type, print_array=False, json_debug=False)	Winsorizing column by column, no dependence across coluns.
sample_run()	

pyfan.panda.stats.cutting.**logger**

pyfan.panda.stats.cutting.**pd_winsorize_columnwise**(*df*, *winsor_coln_list*, *coln_perc_cutoffs*, *return_type*, *print_array=False*, *json_debug=False*)
Winsorizing column by column, no dependence across coluns. Winsorize column by column

cols = 5 rows = 20 np.random.seed(123) data = (np.random.rand(rows,cols)-0.5)*100

df = pd.DataFrame(data, columns=['col' + str(ctr) for ctr in range(cols)]) winsor_coln_list = ['col0', 'col1','col3','col4']

Parameters

df: dataFrame initial dataset

winsor_coln_list: list list of column names to winsorize ['col0', 'col1','col3','col4']

coln_perc_cutoffs: dictionary a nested dictionary where keys are elements of winsor_coln_list, and values are a dictionary with min and max percentiles of winsorizing values. if min is 0, do not create cutcolss

{'col0':{'q_ge':0,'q_le':0.9, 'v_ll':10}, 'col1':{'q_ge':0.30,'v_le':50}, 'col3':{'q_ge':0.01,'q_le':0.60, 'v_ll':40}, 'col4':{'q_ge':0.01,'q_le':1, 'v_ll':33, 'v_gg':-5}}

return_type: string 'winsorize' or 'cutsubset'

pyfan.panda.stats.cutting.**sample_run**()

pyfan.panda.stats.mean_varcov

Created on Aug 9, 2018

@author: fan

Generate mean, and variance covariance of key state variables from data

Module Contents

Functions

gen_mean(df, mean_var_list, short_mean_var_list=None, group_by_var_list=None, conditioning=None)	
gen_varcov(df, varcov_var_list, short_varcov_var_list=None, group_by_var_list=None, conditioning=None)	

pyfan.panda.stats.mean_varcov.**logger**

pyfan.panda.stats.mean_varcov.**gen_mean**(*df*, *mean_var_list*, *short_mean_var_list=None*, *group_by_var_list=None*, *conditioning=None*)

pyfan.panda.stats.mean_varcov.**gen_varcov**(*df*, *varcov_var_list*, *short_varcov_var_list=None*, *group_by_var_list=None*, *conditioning=None*)

pyfan.panda.stats.polynomial_regression

Created on Sep 23, 2018

@author: fan

Module Contents

Functions

ols_formula(df, dependent_var, *excluded_cols)	Generates the R style formula for statsmodels (patsy) given
tester()	

pyfan.panda.stats.polynomial_regression.**logger**

pyfan.panda.stats.polynomial_regression.**ols_formula**(*df*, *dependent_var*, **excluded_cols*)

Generates the R style formula for statsmodels (patsy) given the dataframe, dependent variable and optional excluded columns as strings

pyfan.panda.stats.polynomial_regression.**tester**()

pyfan.panda.stats.polynomial_regression.**FORMAT**
:annotation: = %(filename)s - %(funcName)s - %(asctime)s - %(levelname)s - %(message)s

pyfan.stats

Subpackages

pyfan.stats.interpolate

Submodules

pyfan.stats.interpolate.interpolate2d

Created on Mar 7, 2017

@author: fan

Module Contents

Functions

exp_value_interpolate_bp(prod_inst, util_opti, b_ssv_sd, k_ssv_sd, epsilon_ssv_sd, b_ssv, k_ssv, epsilon_ssv, b_ssv_zr, k_ssv_zr, epsilon_ssv_zr, states_vfi_dim, shocks_vfi_dim)	interpolate value function and expected value function.
inter_states_bp(prod_inst, util_opti, b_ssv_sd, k_ssv_sd, epsilon_ssv_sd, b_ssv, k_ssv, epsilon_ssv, b_ssv_zr, k_ssv_zr, epsilon_ssv_zr, states_vfi_dim, shocks_vfi_dim)	interpolate value function and expected value function.
exp_value_interpolate_main(u1, x1, y1, x2, y2, x2_noshk, y2_noshk, states_dim, shocks_dim, return_uxy=False)	A. Get Interpolant
exp_value_interpolate_bpkp(hhp_inst, util_opti, b, k, b_shk, k_shk)	interpolate value function and expected value function.
k_alpha_cash(hhp_inst, b_vec, k_vec)	
interp_griddata(cur_u, cur_x1, cur_x2, new_x1, new_x2)	Centralize the invokation of 2D interpolation tool
interp2d(prod, cash, z=None, interpolant=None, kind='linear')	Centralize the invokation of 2D interpolation tool
interpRbf2D(prod, cash, z=None, interpolant=None, kind='linear')	
interpRbf3D(prod, cash, A, z=None, interpolant=None, kind='cubic')	
regress_mat(k_alpha, cash)	
regress(dependent_var, rhs_var)	

pyfan.stats.interpolate.interpolate2d.**exp_value_interpolate_bp**(*prod_inst*, *util_opti*, *b_ssv_sd*, *k_ssv_sd*, *epsilon_ssv_sd*, *b_ssv*, *k_ssv*, *epsilon_ssv*, *b_ssv_zr*, *k_ssv_zr*, *epsilon_ssv_zr*, *states_vfi_dim*, *shocks_vfi_dim*)

interpolate value function and expected value function.

Need three matrix here: 1. state matrix x shock matrix where optimal choices were solved at

- previously, shock for this = 0, but now shock vector might not be zero

2. **state matrix x shock matrix where shocks are drawn monte carlo way to allow** for averaging, integrating over shocks for each x row
3. state matrix alone, shock = 0, each of the x row in matrix x

pyfan.stats.interpolate.interpolate2d.**inter_states_bp**(*prod_inst*, *util_opti*, *b_ssv_sd*, *k_ssv_sd*, *epsilon_ssv_sd*, *b_ssv*, *k_ssv*, *epsilon_ssv*, *b_ssv_zr*, *k_ssv_zr*, *epsilon_ssv_zr*, *states_vfi_dim*, *shocks_vfi_dim*)

interpolate value function and expected value function.

Need three matrix here: 1. state matrix x shock matrix where optimal choices were solved at

- previously, shock for this = 0, but now shock vector might not be zero

2. **state matrix x shock matrix where shocks are drawn monte carlo way to allow** for averaging, integrating over shocks for each x row
3. state matrix alone, shock = 0, each of the x row in matrix x

pyfan.stats.interpolate.interpolate2d.**exp_value_interpolate_main**(*u1*, *x1*, *y1*, *x2*, *y2*, *x2_noshk*, *y2_noshk*, *states_dim*, *shocks_dim*, *return_uxy=False*)

A. Get Interpolant

pyfan.stats.interpolate.interpolate2d.**exp_value_interpolate_bpkp**(*hhp_inst*, *util_opti*, *b*, *k*, *b_shk*, *k_shk*)

interpolate value function and expected value function.

cash and k_alpha calculation below does not repeat what happened already inside lifetimeutility. Inside lifetimeutility, we have next period cash and k_alpha here is this period

pyfan.stats.interpolate.interpolate2d.**k_alpha_cash**(*hhp_inst*, *b_vec*, *k_vec*)

pyfan.stats.interpolate.interpolate2d.**interp_griddata**(*cur_u*, *cur_x1*, *cur_x2*, *new_x1*, *new_x2*)

Centralize the invokation of 2D interpolation tool

Potentially chagne this to something else if I don't like it.

pyfan.stats.interpolate.interpolate2d.**interp2d**(*prod*, *cash*, *z=None*, *interpolant=None*, *kind='linear'*)

Centralize the invokation of 2D interpolation tool

Potentially chagne this to something else if I don't like it.

pyfan.stats.interpolate.interpolate2d.**interpRbf2D**(*prod*, *cash*, *z=None*, *interpolant=None*, *kind='linear'*)

pyfan.stats.interpolate.interpolate2d.**interpRbf3D**(*prod*, *cash*, *A*, *z=None*, *interpolant=None*, *kind='cubic'*)

pyfan.stats.interpolate.interpolate2d.**regress_mat**(*k_alpha*, *cash*)

pyfan.stats.interpolate.interpolate2d.**regress**(*dependent_var*, *rhs_var*)

pyfan.stats.markov

Submodules

pyfan.stats.markov.transprobcheck

The *pyfan.stats.markov.transprobcheck* checks markov transition row sums.

A markov transition matrix where each row does not sum up to 1 due to simulation errors. Check if the gap between 1 and the row values are too big, and then normalize.

import pyfan.stats.markov.transprobcheck as pyfan_stats_transprobcheck

Includes method *markov_trans_prob_check()* and *markov_condi_prob2one()*.

Module Contents

Functions

markov_trans_prob_check(mt_trans, fl_atol_per_row=1e-05, fl_atol_avg_row=1e-08, fl_sum_to_match=1)	Markov conditional transition probability check
markov_condi_prob2one(mt_trans)	Rescale markov transitions rows to sum to 1

pyfan.stats.markov.transprobcheck.**markov_trans_prob_check**(*mt_trans*, *fl_atol_per_row=1e-05*, *fl_atol_avg_row=1e-08*, *fl_sum_to_match=1*)

Markov conditional transition probability check

Parameters

mt_trans [numpy.array of shape (N, N)] The AR1 transition matrix, each row is a state, each value in each row is the conditional probability of moving from state i (row) to state j (column)

fl_atol_per_row [*float*, optional] Tolerance for the difference between 1 and each row sum

fl_atol_avg_row [*float*, optional] Tolerance for the difference between 1 and average of row sums

fl_sum_to_match [*float*, optional] This should be 1, unless the function is not used to handle transition matrixes

Returns

———-

tuple A tuple of booleans, the first element is if satisfies the overall criteria. Second is if satisifes the per_row condition. Third if satisfies the average criteria.

Examples

```
>>> mt_ar1_trans = np.array([[0.4334, 0.5183, 0.0454],
>>>                          [0.2624, 0.5967, 0.1245],
>>>                          [0.1673, 0.5918, 0.2005]])
>>> bl_ar1_sum_pass, bl_per_row_pass, bl_avg_row_pass = markov_trans_prob_
↪check(mt_ar1_trans)
>>> print(f'{bl_ar1_sum_pass=}')
bl_ar1_sum_pass=False
>>> print(f'{bl_per_row_pass=}')
bl_per_row_pass=False
>>> print(f'{bl_avg_row_pass=}')
bl_avg_row_pass=False
```

pyfan.stats.markov.transprobcheck.**markov_condi_prob2one**(*mt_trans*)

Rescale markov transitions rows to sum to 1

Suppose each transition matrix row sums up to slighly less than one, rescale so it sums to one.

Parameters

mt_trans [numpy.array of shape (N, N)] The AR1 transition matrix, each row is a state, each value in each row is the conditional probability of moving from state i (row) to state j (column)

Returns

ndarray The rescaled numpy array

pyfan.stats.multinomial

Submodules

pyfan.stats.multinomial.multilogit

Created on Dec 4, 2017

@author: fan

Module Contents

Classes

UtilityMultiNomial	each_j_indirect_utility:

pyfan.stats.multinomial.multilogit.**logger**

class pyfan.stats.multinomial.multilogit.**UtilityMultiNomial**(*scale_coef=1*)

each_j_indirect_utility: N by J matrix

N is the number of individuals (unique states) J is the number of choices

N might be 0

prob_denominator(*self*, *all_J_indirect_utility*)

if:

all_J_indirect_utility/self.scale_coef = -598.66/0.75

then: prob_denominator = exp(-598.66/0.75) = 0.0

then: sum(prob_denominator) = 0

then: np.exp(all_J_indirect_utility/self.scale_coef)/prob_denominator_tile = INVALID

so there must be some minimal level for the division here. in terms of scaling

prob_j(*self*, *all_J_indirect_utility*, *prob_denominator=None*)

expected_u_integrate_allj(*self*, *prob_denominator*)

'see Train discussion on consumer surplus and logit' 'Need to check the reference that Train cites to make sure this integration applies in my case, should derive it myself' If one option has much higher utility, and if variance is low, integrated utility is linear in this option, in fact they are equal

get_outputs(*self*, *all_J_indirect_utility*)

pyfan.table

Subpackages

pyfan.table.reg

Submodules

pyfan.table.reg.txt2textab

The *pyfan.stats.markov.transprobcheck* checks markov transition row sums.

A markov transition matrix where each row does not sum up to 1 due to simulation errors. Check if the gap between 1 and the row values are too big, and then normalize.

import pyfan.stats.markov.transprobcheck as pyfan_stats_transprobcheck

Includes method `markov_trans_prob_check()` and `markov_condi_prob2one()`.

Module Contents

Functions

tab_txt2tex_f2f(spt_root='', st_rglob='tab_*_fmd.md', **kwargs)	
tab_txt2tex(ls_st_txt_regs, it_col_count=6, fl_adj_box_maxwidth=1, it_or_dc_round_decimal=2, fl_col_label_width_cm=5, fl_col_coef_width_cm=2, fl_indent_pound1_mm=0, fl_indent_pound2_mm=0, fl_indent_pound3_mm=6)	Markov conditional transition probability check

pyfan.table.reg.txt2textab.**tab_txt2tex_f2f**(*spt_root=''*, *st_rglob='tab_*_fmd.md'*, ***kwargs*)

pyfan.table.reg.txt2textab.**tab_txt2tex**(*ls_st_txt_regs*, *it_col_count=6*, *fl_adj_box_maxwidth=1*, *it_or_dc_round_decimal=2*, *fl_col_label_width_cm=5*, *fl_col_coef_width_cm=2*, *fl_indent_pound1_mm=0*, *fl_indent_pound2_mm=0*, *fl_indent_pound3_mm=6*)

Markov conditional transition probability check

Parameters

it_col_count [int] Number of latex table columns

fl_atol_per_row [*float*, optional] Tolerance for the difference between 1 and each row sum

fl_atol_avg_row [*float*, optional] Tolerance for the difference between 1 and average of row sums

fl_sum_to_match [*float*, optional] This should be 1, unless the function is not used to handle transition matrixes

Returns

——-

list string formated to tex to return A tuple of booleans, the fiit element is if satisfies the overall criteria. Second is if satisifes the per_row condition. Third if satisfies the average criteria.

Examples

>>> mt_ar1_trans = np.array([[0.4334, 0.5183, 0.0454], # >>> [0.2624, 0.5967, 0.1245], # >>> [0.1673, 0.5918, 0.2005]]) # >>> bl_ar1_sum_pass, bl_per_row_pass, bl_avg_row_pass = markov_trans_prob_check(mt_ar1_trans) # >>> print(f'{bl_ar1_sum_pass=}') # bl_ar1_sum_pass=False # >>> print(f'{bl_per_row_pass=}') # bl_per_row_pass=False # >>> print(f'{bl_avg_row_pass=}') bl_avg_row_pass=False

pyfan.table.reg.txt2textab.**spt_root = C:/Users/fan/Box/Pollution and inequality/drafts/pape**

pyfan.util

Subpackages

pyfan.util.path

Submodules

pyfan.util.path.getfiles

The *pyfan.util.path.getfiles* generate and get various file paths.

import pyfan.util.path.getfiles as getfiles

Includes method *gen_path_file()*, *gen_path()*

Module Contents

Functions

gen_path_file(spt_folder='', snm_file='', st_file_type='')	Return full path to file given folder path and file name with suffix
gen_path(spt_root='', main_folder_name='', sub_folder_name=None, subsub_folder_name=None)	Generate a file path and return it
fp_search_rglob(spt_root='../', st_rglob='*.py', ls_srt_subfolders=None, verbose=False)	Searches for files with search string in a list of folders

pyfan.util.path.getfiles.**gen_path_file**(*spt_folder=''*, *snm_file=''*, *st_file_type=''*)

Return full path to file given folder path and file name with suffix

Parameters

spt_folder [str] full path to folder

snm_file [str] file name without suffix (or with)

st_file_type [str] type of file, see options below, with pre-determined suffix, by default, there is no suffix. so if specify nothing, will save the *snm_file* with what is externally fed in

Returns

str Full path to file with path and file name.

Examples

```
>>> gen_path_file(spt_folder = 'C:/Users/fan/logvig/parameters/combo_type/',
...               snm_file = 'fs_gen_combo_type_20201030', st_file_type='log')
C:\Users\fan\logvig\parameters\combo_type\fs_gen_combo_type_20201030.log.py
```

pyfan.util.path.getfiles.**gen_path**(*spt_root=''*, *main_folder_name=''*, *sub_folder_name=None*, *subsub_folder_name=None*)

Generate a file path and return it

Parameters

spt_root [str] root name

main_folder_name [str] folder name

sub_folder_name [str, optional] subfolder name

subsub_folder_name [str] subsub folder name

Returns

str the full path to the folder

Examples

```
>>> gen_path(spt_root = 'C:/Users/fan/',
...          main_folder_name='logvig', sub_folder_name='parameters',
...          subsub_folder_name='combo_type')
C:\Users\fan\logvig\parameters\combo_type\
```

pyfan.util.path.getfiles.**fp_search_rglob**(*spt_root='../'*, *st_rglob='*.py'*, *ls_srt_subfolders=None*, *verbose=False*)

Searches for files with search string in a list of folders

Parameters

spt_root [string] root folder to search in

st_rglob [string] search for files with this rglob string

ls_srt_subfolders [list of str] a list of subfolders of the spt_root to search in

verbose: bool print details

Returns

list of str A list of file names

Examples

```
>>> ls_spn = fp_search_rglob(spt_root="../",
>>>                          ls_srt_subfolders=['rmd', 'pdf'],
>>>                          st_rglob = '*d*.py')
[WindowsPath('../rmd/bookdownparse.py'), WindowsPath('../rmd/mattexmd.py'),␣
↪WindowsPath('../rmd/rmdparse.py'), WindowsPath('../pdf/pdfgen.py')]
```

pyfan.util.path.movefiles

Module Contents

Functions

fp_agg_move_subfiles(spt_root_src='C:/Users/fan/pyfan/vig/support/inout/_folder/fd/faa/', st_srt_srh='_images', st_fle_srh='*', srt_agg='img', ls_srt_dest=['C:/Users/fan/pyfan/vig/support/inout/_folder/fd/faa/', 'C:/Users/fan/pyfan/vig/support/inout/_folder/'], bl_delete_src=True, bl_test=True, verbose=False)	Aggregate and Move a Collection of Non-empty Folders

pyfan.util.path.movefiles.**fp_agg_move_subfiles**(*spt_root_src='C:/Users/fan/pyfan/vig/support/inout/_folder/fd/fa*, *st_srt_srh='_images'*, *st_fle_srh='*'*, *srt_agg='img'*, *ls_srt_dest=['C:/Users/fan/pyfan/vig/support/inout/_folder/fd/fa*, *'C:/Users/fan/pyfan/vig/support/inout/_folder/']*, *bl_delete_src=True*, *bl_test=True*, *verbose=False*)

Aggregate and Move a Collection of Non-empty Folders

A program (forexample mlx to tex conversion) creates in a folder a number of subfolder that stores images. Aggregate all the various image folders into a common image folder. And then move this common image folder to other destinations in order to flexibly generate aggregation files with common path that rely on images from various subfolders.

Parameters

spt_root_src: string root folder where subfolders are contained

st_srt_srh: string gather subfolder names that contain this string

st_fle_srh: string search in subfolders for files whose name contain string

srt_agg: string name of subfolder where found folders are aggregated at

ls_srt_dest: :obj:`list` of :obj:`str` list of folder paths to move aggregate subfolders over to

bl_delete_src: bool delete folders at existing locations

bl_test: bool test by searching for paths dest and src, do not move

verbose: bool print details

Returns

None nothing is returned

Examples

```
>>> fp_agg_move_subfiles(spt_root_src="C:/Users/fan/Math4Econ/matrix_application/
↪",
>>>                                                 st_srt_srh="_images",
>>>                                                 st_fle_srh="*.png",
>>>                                                 srt_agg='img',
>>>                                                 ls_srt_dest=["C:/Users/fan/Math4Econ/
↪"],
>>>                                                 bl_delete_src=False,
>>>                                                 bl_test=False,
>>>                                                 verbose=False)
```

pyfan.util.path.movefiles.**spt_root_src_u = C:/Users/fan/Math4Econ/matrix_application/**

pyfan.util.pdf

Submodules

pyfan.util.pdf.pdfgen

pyfan generate and clean pdf files from folder The *pyfan.util.pdf.pdfgen* generates pdf files from tex files.

Gather all tex files from a folder, allow for exclusion strings. Generate PDFs from the tex files. And then clean up extraneous PDF outputs.

Includes method *ff_pdf_gen_clean()*.

Module Contents

Functions

ff_pdf_gen_clean(ls_spt_srh=None, spt_out='C:/Users/fan/Documents/Dropbox (UH-ECON)/' + 'Project Emily Minority Survey/' + 'EthLang/reg_lang_abi_cls_mino/', spn_pdf_exe='C:/texlive/2019/bin/win32/xelatex.exe', ls_st_contain=None, ls_st_ignore=None, bl_recursive=False, bl_clean=True, ls_suf_clean=None)	Generate pdf files from latex files in various folders.

pyfan.util.pdf.pdfgen.**ff_pdf_gen_clean**(*ls_spt_srh=None*, *spt_out='C:/Users/fan/Documents/Dropbox (UH-ECON)/' + 'Project Emily Minority Survey/' + 'EthLang/reg_lang_abi_cls_mino/'*, *spn_pdf_exe='C:/texlive/2019/bin/win32/xelatex.exe'*, *ls_st_contain=None*, *ls_st_ignore=None*, *bl_recursive=False*, *bl_clean=True*, *ls_suf_clean=None*)

Generate pdf files from latex files in various folders.

This file serves important paper generation function. It compiles multiple files satisfying string search requirements or exclusion conditions in multiple folders, and saves resulting pdf outputs in one folder. This allows for easy testing and management of mutiple pdf/latex files for the same project. Suppose there is a longer version of a paper, a shorter version, and an appendix file. We want to regularly test the compilations of all files, otherwise, as we work on one of the files, perhaps we some something in the some shared files that lead to other files breaking without knowing.

This should be run for all outward facing pdf/tex files for a project regularly in order to check if all files still compile.

By brining resulting outputs to a single folder, this makes it easier to see all paper and project relevant outputs. Additionally, this cleans up all pdf generated extraneous files once we have pdf itself, saving pdf compile folder clutter.

Parameters

ls_spt_srh [list of str] A list of strings of the path in which to search for tex files. They should be all on the same path. If *bl_recursive* is true, then this searchs in all subfolders.

spt_out [str] The Path to store outputs. All PDFs stored under single directory. This path must be directly on the same path as `ls_spt_srh', can be higher up on the same tree, but not on a different branch.

spn_pdf_exe: str The path to the pdflatex or alternative exe file

ls_st_contain: :obj:`list` of :obj:`str` a list of strings the found names must contain one of these search words, not all, just one of.

ls_st_ignore [list of str] a list of string file names to ignore

bl_recursive [bool] Whether to search for all tex files within subfolders

bl_clean [bool] To clean up after file generation

ls_suf_clean: :obj:`list` of :obj:`str` list of

Returns

dict A list of string pdf file names outputed,

pyfan.util.rmd

Submodules

pyfan.util.rmd.bookdownparse

Generate README from Fan Rmd Frontmatter The *pyfan.util.rmd.bookdownparse* generates rmd frontmatter.

Generates README.md TOC contents based on parsed YAML frontmatter from files listed in a bookdown yaml file that contains a list of RMD files to process through.

Includes method *fs_yml2readme()*

Module Contents

Functions

fs_yml2readme(sfc_prj='R4Econ', sph_prj='C:/Users/fan/R4Econ/', spn_prj_bookdown_yml='_bookdown.yml', spn_prj_readme_toc='README_toc.md', ls_st_ignore=['index.Rmd', 'README_appendix.md', 'title.Rmd', 'main.Rmd'], sph_pdf='htmlpdfr', sph_html='htmlpdfr', sph_r='htmlpdfr', st_file_type='r', verbose=False)	Write to file README detailed TOC for files in bookdown yaml list

pyfan.util.rmd.bookdownparse.**fs_yml2readme**(*sfc_prj='R4Econ'*, *sph_prj='C:/Users/fan/R4Econ/'*, *spn_prj_bookdown_yml='_bookdown.yml'*, *spn_prj_readme_toc='README_toc.md'*, *ls_st_ignore=['index.Rmd', 'README_appendix.md', 'title.Rmd', 'main.Rmd']*, *sph_pdf='htmlpdfr'*, *sph_html='htmlpdfr'*, *sph_r='htmlpdfr'*, *st_file_type='r'*, *verbose=False*)

Write to file README detailed TOC for files in bookdown yaml list

Parameters

sfc_prj [string] The git project name

sph_prj [string] the local path to the git project

spn_prj_bookdown_yml [string] yml file name under project root contains rmd names under 'rmd_files' key

spn_prj_readme_toc [string] md generated file name under project root

ls_st_ignore: list list of string names to ignore

sph_pdf [string] subfolder to store pdf files in the rmd folder

sph_html [string] subfolder to store html files in the rmd folder

sph_r [string] subfolder to store r files in the rmd folder does not have to be r, any other raw file type, m of py for example

st_file_type: string the RMD file is for which underlying language: r for R, m for matlab, py for python

verbose: bool print details

Returns

None nothing is returned, the spn_prj_readme_toc toc file is generated

Examples

```
>>> fs_yml2readme(sfc_prj='pyfan', sph_prj='../../../', verbose=False)
```

pyfan.util.rmd.mattexmd

Convert matlab MLX files to MD and RMD files The *pyfan.util.rmd.mattexmd* generates md and rmd file from tex file converted from matlab mlx.

Use matlab's own functions to export a MLX file to tex. Clean some elements of this text file, use pandoc to convert to MD. Then resave the md file as a RMD file by combining yml info from a preamble.yml file that is in the folder.

Includes method *fp_md2rmd()*, *fp_mlxtex2md()*

Module Contents

Functions

fp_md2rmd(spt_root='C:/Users/fan/Math4Econ/', ls_srt_subfolders=['calconevar/'], snm_folder_yml='preamble.yml', st_yml_file_key='file', st_rglob_md='*.md', verbose=False)	Generate RMD from MD with YAML Header
fp_mlxtex2md(spt_root='C:/Users/fan/Math4Econ/', ls_srt_subfolders=['calconevar/'], st_rglob_tex='*.tex', verbose=False)	Edit and replace MLX based tex and convert to markdown

pyfan.util.rmd.mattexmd.**fp_md2rmd**(*spt_root='C:/Users/fan/Math4Econ/'*, *ls_srt_subfolders=['calconevar/']*, *snm_folder_yml='preamble.yml'*, *st_yml_file_key='file'*, *st_rglob_md='*.md'*, *verbose=False*)

Generate RMD from MD with YAML Header

In each folder, there is a prior preamble.yaml for all files in folder. There are md files generated by the fp_mlxtex2md function, this file stacks the yaml header on top of the md after searching for the yml header with the no suffix file name of the md file, which is the search key in the yaml.

Parameters

spt_root [string] The root folder

ls_srt_subfolders [list of str] List of subfolders to search in

snm_folder_yml: string name of the file containing all rmd yaml header under spt_root

st_yml_file_key: string the key in the yaml header snm_folder_yml with file names no suffix

st_rglob_md [string] md files search string rglob

verbose [bool] print extra

Returns

None nothing is returned, mlx generated tex files updates, and pandoc md generated

Examples

```
>>> fp_md2rmd(spt_root="C:/Users/fan/M4Econ/amto/",
...           ls_srt_subfolders=['array/'],
...           snm_folder_yml='preamble.yml',
...           st_yml_file_key='file',
...           st_rglob_md='fs_slicing.md', verbose=False)
```

pyfan.util.rmd.mattexmd.**fp_mlxtex2md**(*spt_root='C:/Users/fan/Math4Econ/'*, *ls_srt_subfolders=['calconevar/']*, *st_rglob_tex='*.tex'*, *verbose=False*)

Edit and replace MLX based tex and convert to markdown

Several mlx auto converted tex elements need to be constructed, otherwise the md file would not work. Importantly, convert how images are stored to aggregate subfolder. Which relies on movefiles.py to generate to put in the image folder structure as specified here which is under an aggregate img folder.

Parameters

spt_root [string] The root folder

ls_srt_subfolders [list of str] List of subfolders to search in

st_rglob_tex [string] tex files search string rglob

verbose [bool] print extra

Returns

None nothing is returned, mlx generated tex files updates, and pandoc md generated

Examples

```
>>> fp_mlxtex2md(spt_root='C:/Users/fan/Math4Econ/matrix_application/',
...              ls_srt_subfolders=None, st_rglob_tex='twogoods.tex',␣
↪verbose=True)
>>> fp_mlxtex2md(spt_root='C:/Users/fan/M4Econ/amto/array/',
...              ls_srt_subfolders=None, st_rglob_tex='fs_slicing.tex',␣
↪verbose=True)
```

pyfan.util.rmd.rmdparse

Fan Rmd Fronter Matter Core Parser

In mine Rmd files front matter, there are some special parameters that provide summaries for the key contents in the code, and also provide key programs and dependencies that were used. The information can be parsed to string lists. The special strings are meant to be used with a special README.md github pages file. The goal is to automatically gather meta-data across RMD files within folders in a package to prepare data needed to provide a detailed table of content type contents needed for a README.md file.

Module Contents

Functions

fs_rmd_yml_parse(sfc_prj='R4Econ', sph_prj='C:/Users/fan/R4Econ', spn_prj_rmd='/summarize/aggregate/fs_group_unique_agg.Rmd', sph_gitpages_root='https://fanwangecon.github.io/', sph_github_root='https://github.com/FanWangEcon/', sph_branch='/master', sph_pdf='htmlpdfr', sph_html='htmlpdfr', sph_r='htmlpdfr', st_file_type='r')	Parse Yaml Frontmatter to get list of paths, summaries, dependencies

pyfan.util.rmd.rmdparse.**fs_rmd_yml_parse**(*sfc_prj='R4Econ'*, *sph_prj='C:/Users/fan/R4Econ'*, *spn_prj_rmd='/summarize/aggregate/fs_group_unique_agg.Rmd'*, *sph_gitpages_root='https://fanwangecon.github.io/'*, *sph_github_root='https://github.com/FanWangEcon/'*, *sph_branch='/master'*, *sph_pdf='htmlpdfr'*, *sph_html='htmlpdfr'*, *sph_r='htmlpdfr'*, *st_file_type='r'*)

Parse Yaml Frontmatter to get list of paths, summaries, dependencies

Parameters

sfc_prj [string] The git project name

sph_prj [string] the local path to the git project

spn_prj_rmd [string] the path (within project) to the Rmd file

sph_gitpages_root [string] github pages site url root

sph_github_root [string] github project page url

sph_branch [string] which branch

sph_pdf [string] subfolder to store pdf files in the rmd folder

sph_html [string] subfolder to store html files in the rmd folder

sph_r [string] subfolder to store r files in the rmd folder does not have to be r, any other raw file type, m of py for example

st_file_type: string the RMD file is for which underlying language: r for R, m for matlab, py for python

pyfan.util.timer

Submodules

pyfan.util.timer.timer

The *pyfan.util.timer.timer* generates various timer related strings.

Includes method *getDateTime()*.

Module Contents

Classes

Timer

Functions

curTimeDiff(startTime=None)
getDateTime(timeType=6)

pyfan.util.timer.timer.**curTimeDiff**(*startTime=None*)

pyfan.util.timer.timer.**getDateTime**(*timeType=6*)

class pyfan.util.timer.timer.**Timer**(*name=None*)

Bases: object

__enter__(*self*)

__exit__(*self*, *type*, *value*, *traceback*)

11.1.2 Package Contents

pyfan.**__version__ = 0.1.48**

pyfan python support package for various tasks to enable mostly generic python operations.

11.2 sandbox

Created on Aug 13, 2018

@author: fan

11.2.1 Module Contents

sandbox.**logger**

sandbox.**cols = 5**

sandbox.**rows = 20**

sandbox.**data**

sandbox.**df**

sandbox.**winsor_coln_list = ['col0', 'col1', 'col3', 'col4']**

sandbox.**coln_perc_cutoffs**

sandbox.**return_type = winsorize**
Deep copy original dataframe

sandbox.**condi_ctr = 0**

sandbox.**col_perc_cutoff_dict**
Initialize cut min max

sandbox.**df_return**

11.3 iosupport

Created on Oct 5, 2013

@author: fan

11.3.1 Module Contents

Classes

csvIO

class iosupport.**csvIO**

numerical_parmlist_stringify(*self*, *trueParameters*)

createDirectory(*self*, *save_file_directory*, *addName=''*)

saveCSV(*self*, *saveFileDirectory*, *dataToSave*, *header=''*)

loadCSV(*self*, *saveFileDirectory*, *unpack=True*)

loadText(*self*, *saveFileDirectory*)

print_data_summary(*self*, *numpy_matrix*, *numpy_matrix_name*, *saveFileDirectory=None*, *printSummary=True*, *printFullMatrix=False*, *saveText=True*, *printConsole=True*, *printFullArray=False*, *precision=None*, *linewidth=200*)

saveText(*self*, *saveFileDirectory*, *dataToSave*, *type='w'*, *addTimer=False*, *lineBreak=True*, *doubleBreak=False*, *printConsole=True*, *precision=None*, *linewidth=200*)

11.4 exportpanda

Created on Mar 28, 2018

@author: fan

11.4.1 Module Contents

Functions

export_history_csv(store, store_map, filepath='', filename='', export=True)	
saveJSON(saveFileDirectory, data, replace)	
saveCSV(saveFileDirectory, dataToSave, header='', rowindex=False, is_panda=True, replace=True, export=True)	if get permissiond denied error, because possible saveFileDirectory did not have .csv end

exportpanda.**logger**

exportpanda.**export_history_csv**(*store*, *store_map*, *filepath=''*, *filename=''*, *export=True*)

exportpanda.**saveJSON**(*saveFileDirectory*, *data*, *replace*)

exportpanda.**saveCSV**(*saveFileDirectory*, *dataToSave*, *header=''*, *rowindex=False*, *is_panda=True*, *replace=True*, *export=True*)
if get permissiond denied error, because possible saveFileDirectory did not have .csv end

11.5 sandbox_20200623

11.5.1 Module Contents

sandbox_20200623.**spt_root = C:/Users/fan/Downloads/_data/**

sandbox_20200623.**spn_dl_test_grib**

sandbox_20200623.**c**

sandbox_20200623.**res**

11.6 sandbox_20201105

11.6.1 Module Contents

sandbox_20201105.**dc_invoke_main_args_default**

sandbox_20201105.**dc_invoke_main_args**

sandbox_20201105.**ls_st_spec_key_dict = ['NG_S_D', 'NG_S_D=KAP_M0_NLD_M_SIMU=2=3']**

sandbox_20201105.**st_connector = =**

sandbox_20201105.**ls_st_esti_simu = ['esti', 'simu']**

sandbox_20201105.**it_len_spec_key_split**

PYTHON MODULE INDEX

e

i

p

s

INDEX

Symbols

A

B

C

D

E

F

G

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z