Model Analysis ToolKit (MATK) Documentation

Release 0

Dylan R. Harp

May 17, 2015

1 Obtaining MATK **1**
1.1 Downloading MATK . . . 1
1.2 Installing MATK . . . 1
1.3 Testing installation . . . 1

2 Examples **3**
2.1 Latin Hypercube Sampling . . . 3
2.2 Calibration . . . 14

3 Class Documentation **17**
3.1 MATK . . . 17
3.2 Parameter . . . 22
3.3 Observation . . . 22
3.4 SampleSet . . . 23

4 Indices and tables **27**

Python Module Index **29**

Index **31**

CHAPTER

ONE

OBTAINING MATK

1.1 Downloading MATK

MATK can be obtained by:

1. Using git:

```
git clone https://github.com/dharp/matk
```

2. Clicking here
3. Going to https://github.com/dharp/matk and clicking on the **Download ZIP** button on the right

1.2 Installing MATK

To install MATK, enter the main directory in a terminal and

```
python setup.py install
```

Depending on your system setup and privileges, you may want to do this as root (*nix and mac systems):

```
sudo python setup.py install
```

or as user:

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

If all these fail, you can set your PYTHONPATH to point to the MATK *src* directory

1. bash:

```
export PYTHONPATH=/path/to/matk/src
```

2. tcsh:

```
setenv PYTHONPATH /path/to/matk/src
```

3. Windows, I have no clue!

1.3 Testing installation

To test that the MATK module is accessible, open a python/ipython terminal and

```
import matk
```

If the MATK module is accessible, this will load without an error.

For more in depth analysis of MATK functionality on your system, the test suite can be run by entering the MATK *tests* directory in a terminal and:

```
python -W ignore matk_unittests.py -v
```

Test results will be printed to the terminal.

CHAPTER

TWO

EXAMPLES

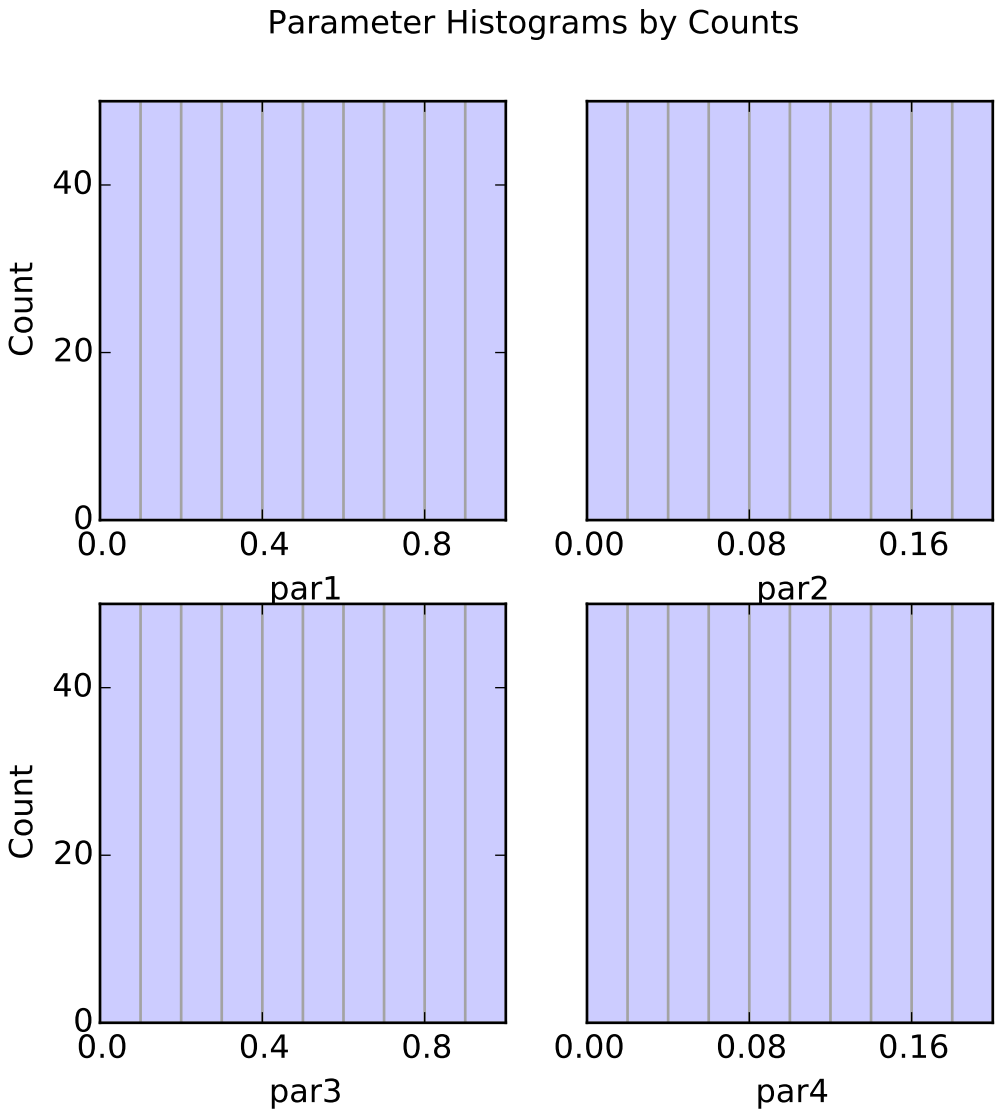
2.1 Latin Hypercube Sampling

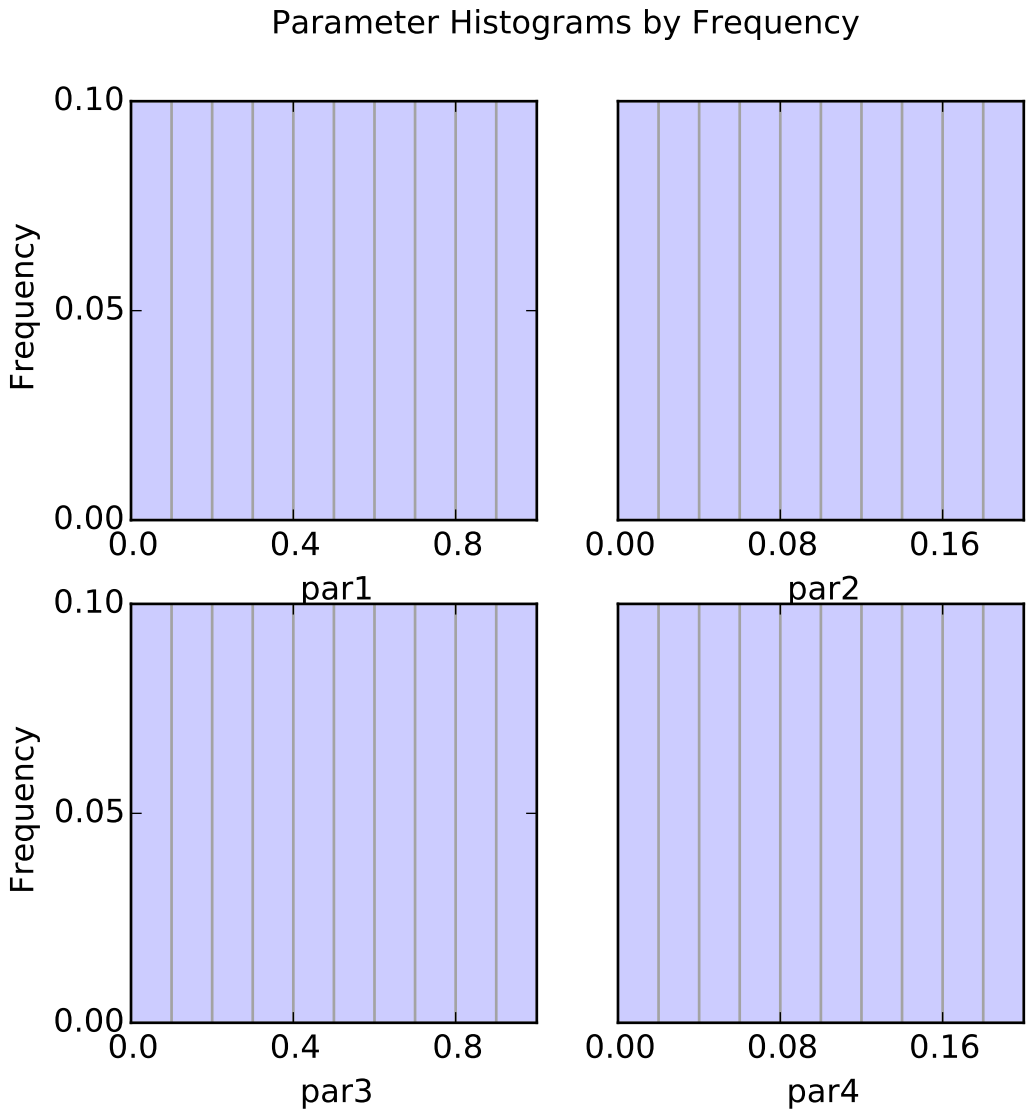
This example demonstrates a Latin Hypercube Sampling of a 4 parameter 4 response model using the `lhs` function. The generation of diagnostic plots is demonstrated using *hist*, *panels*, and *corr*.

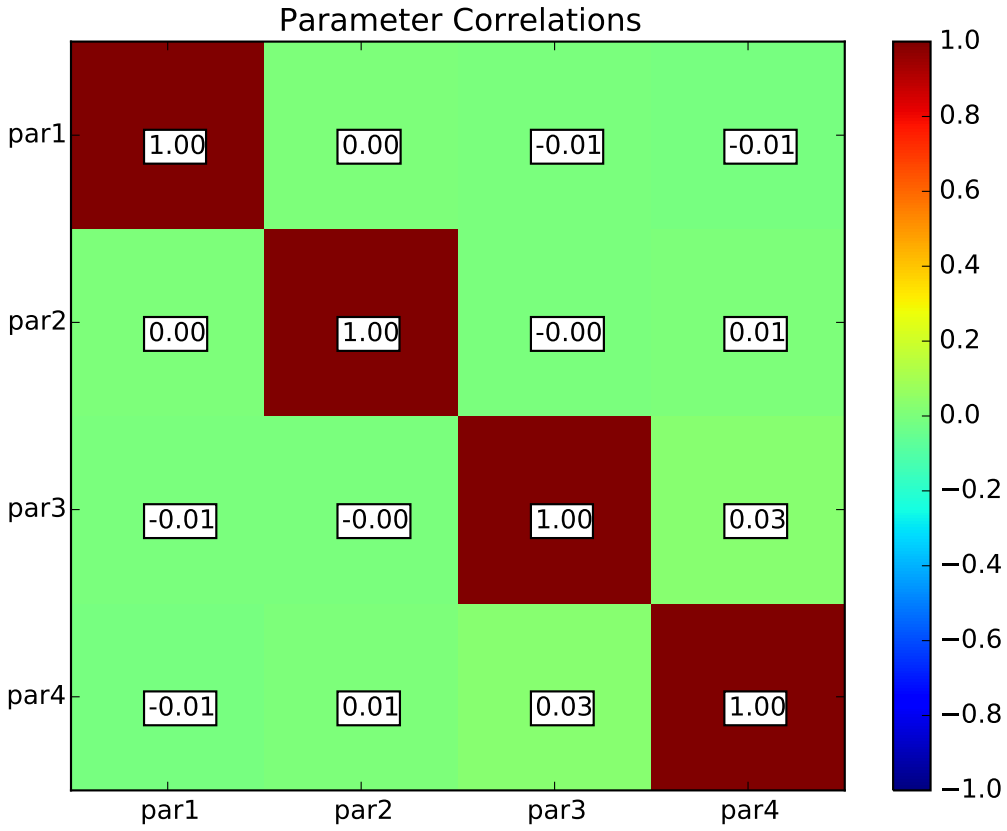
The plots generated by the script are displayed below the code block. The script *sampling.py* can be downloaded using the **Source code** link below, or the *examples* folder of the repository.

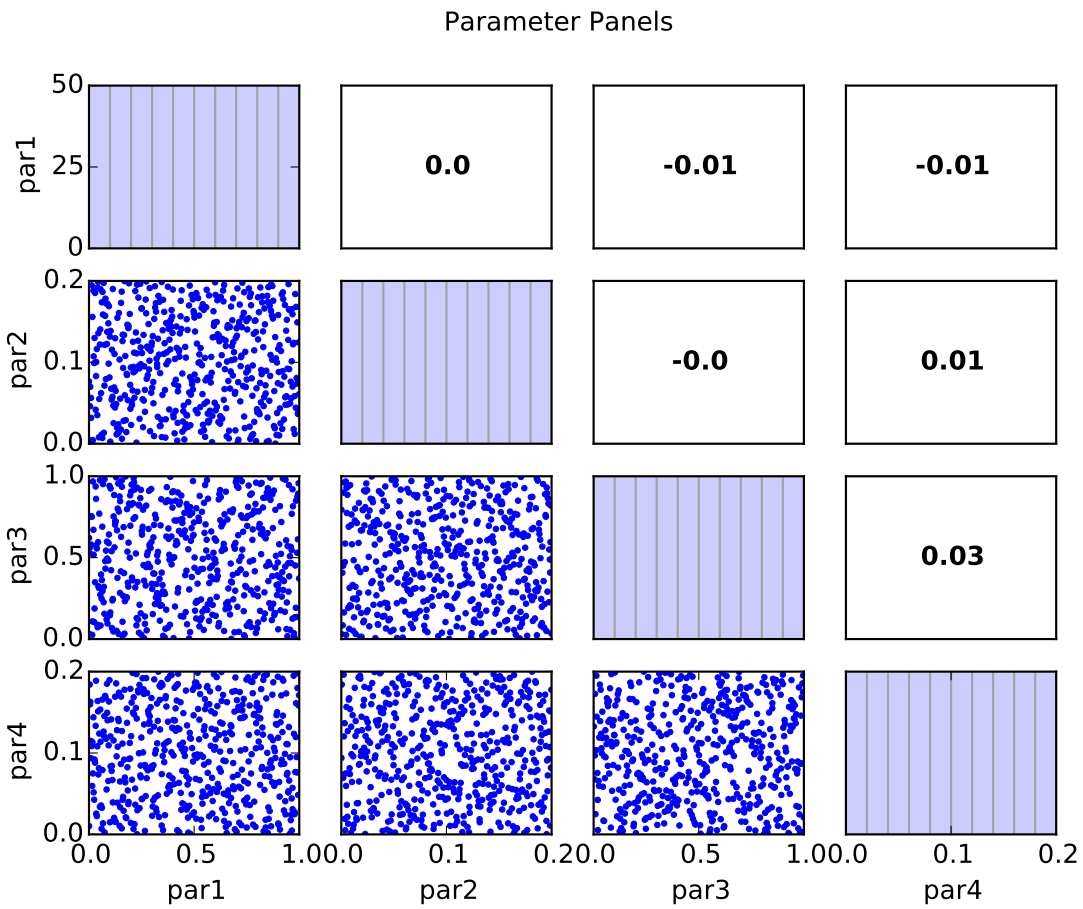
```
import sys,os
try:
    import matk
except:
    try:
        sys.path.append(os.path.join('..','src','matk'))
        import matk
    except ImportError as err:
        print 'Unable to load MATK module: '+str(err)
import numpy
from scipy import arange, randn, exp
from multiprocessing import freeze_support

# Model function
def dbexpl(p):
    t=arange(0,100,20.)
    y =  (p['par1']*exp(-p['par2']*t) + p['par3']*exp(-p['par4']*t))
    #nm =  ['o1','o2','o3','o4','o5']
    #return dict(zip(nm,y))
    return y

def run():
        # Setup MATK model with parameters
        p = matk.matk(model=dbexpl)
        p.add_par('par1',min=0,max=1)
        p.add_par('par2',min=0,max=0.2)
        p.add_par('par3',min=0,max=1)
        p.add_par('par4',min=0,max=0.2)

        # Create LHS sample
        s = p.lhs('lhs', siz=500, seed=1000)

        # Look at sample parameter histograms, correlations, and panels
        s.samples.hist(ncols=2,title='Parameter Histograms by Counts')
        s.samples.hist(ncols=2,title='Parameter Histograms by Frequency',frequency=True)
        parcor = s.samples.corr(plot=True, title='Parameter Correlations')
```

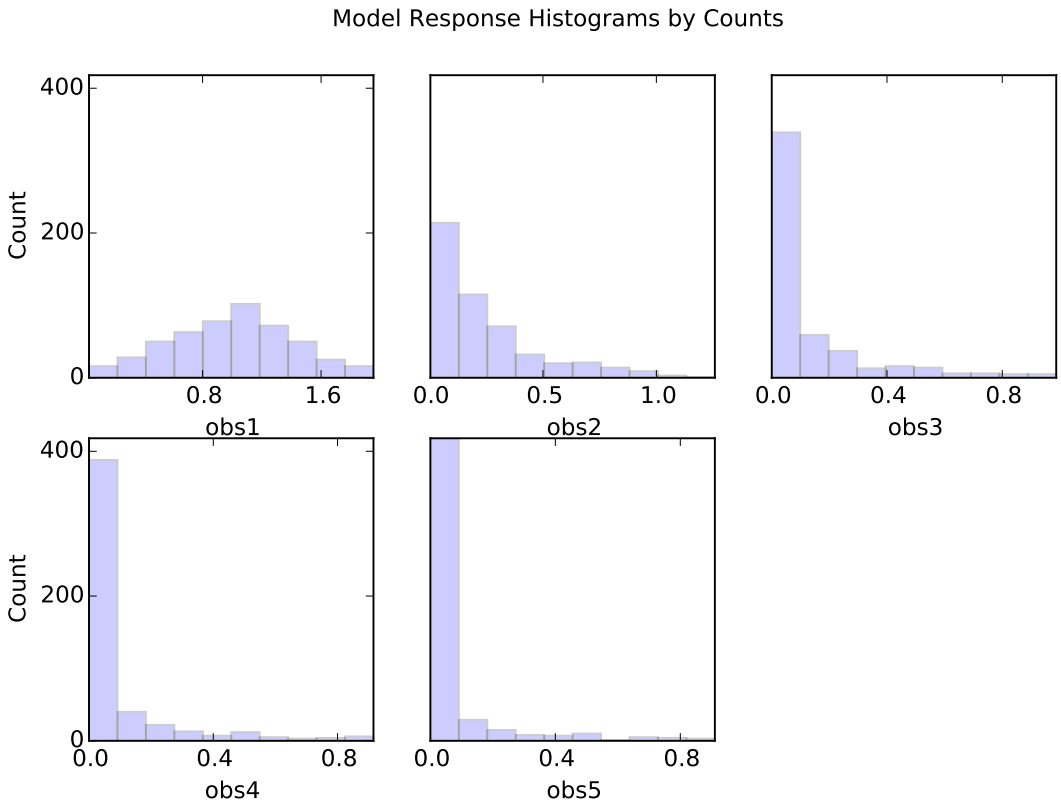
```
        s.samples.panels(title='Parameter Panels')

        # Run model with parameter samples
        s.run( cpus=2, outfile='results.dat', logfile='log.dat',verbose=False)

        # Look at response histograms, correlations, and panels
        s.responses.hist(ncols=3,title='Model Response Histograms by Counts')
        s.responses.hist(ncols=3,title='Model Response Histograms by Frequency',frequency=True)
        rescor = s.responses.corr(plot=True, title='Model Response Correlations')
        s.responses.panels(title='Response Panels')

        # Print and plot parameter/response correlations
        print "\nPearson Correlation Coefficients:"
        pcorr = s.corr(plot=True,title='Pearson Correlation Coefficients')
        print "\nSpearman Correlation Coefficients:"
        scorr = s.corr(plot=True,type='spearman',title='Spearman Rank Correlation Coefficients')
        s.panels(figsize=(10,8))

# Freeze support is necessary for multiprocessing on windows
if __name__== "__main__":
        freeze_support()
        run()
```

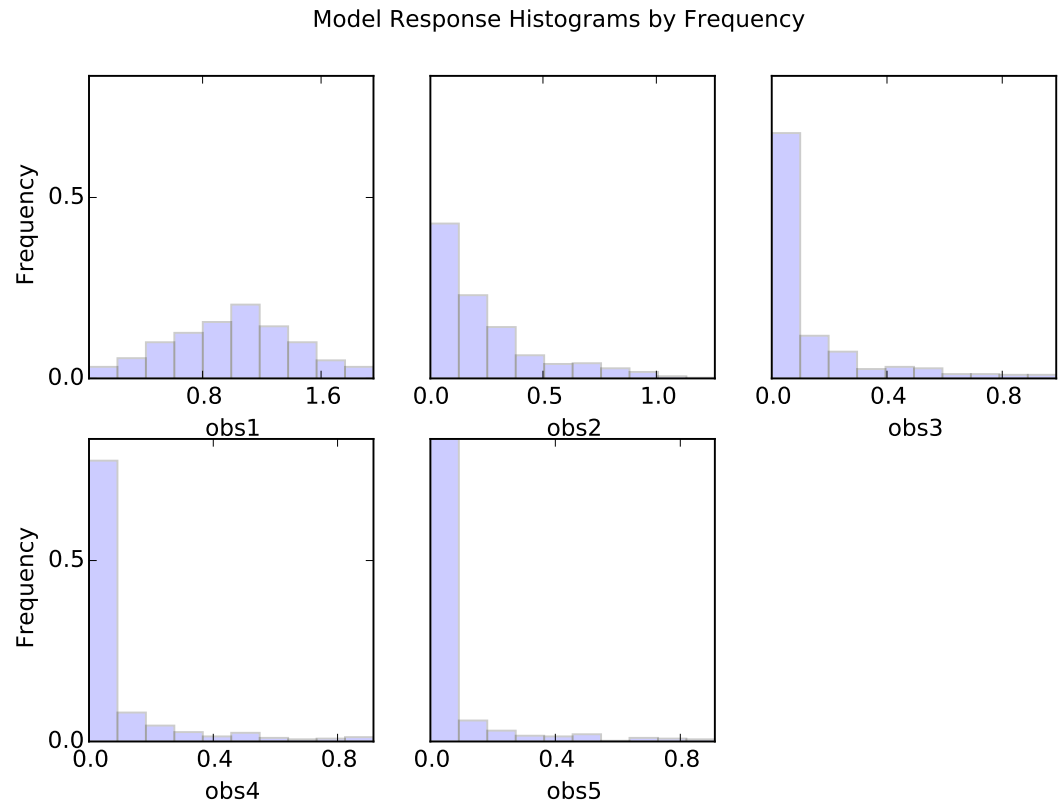

Parameter Histograms by Counts
Count
40
20
0
0.0
0.4
0.8
par1
0.00
0.08
0.16
par2
Count
40
20
0
0.0
0.4
0.8
par3
0.00
0.08
0.16
par4

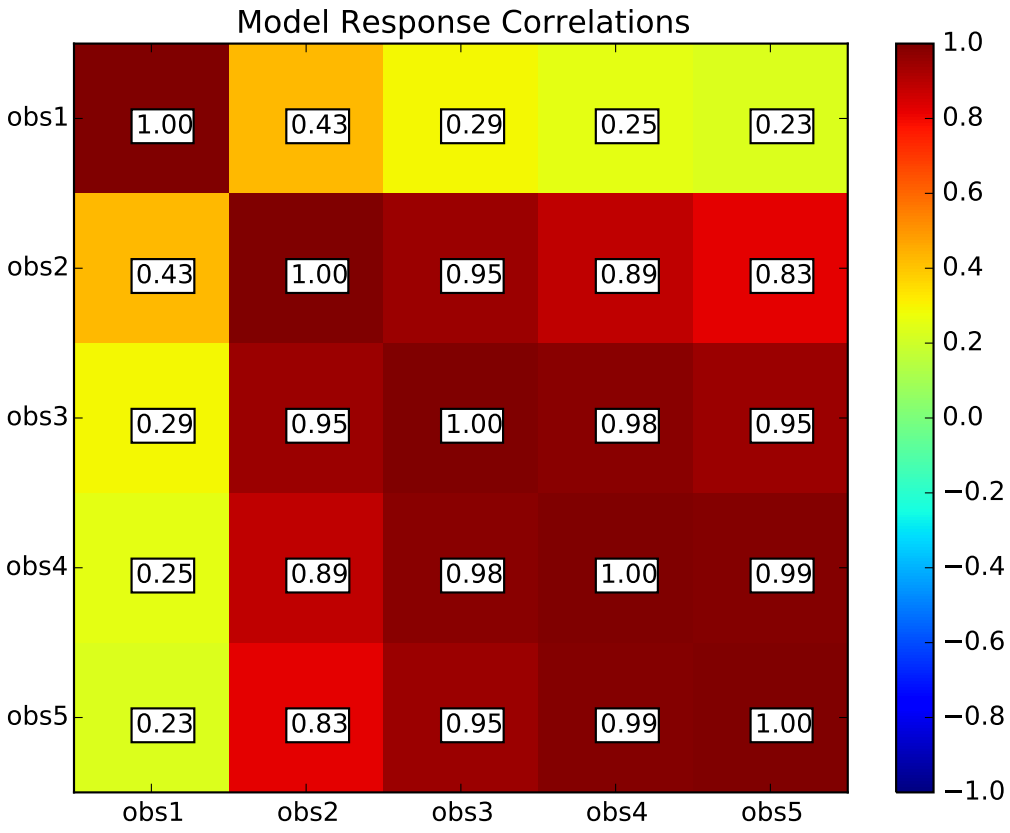

Parameter Histograms by Frequency
Frequency
0.10
0.05
0.00
0.0
0.4
0.8
par1
0.00
0.08
0.16
par2
Frequency
0.10
0.05
0.00
0.0
0.4
0.8
par3
0.00
0.08
0.16
par4

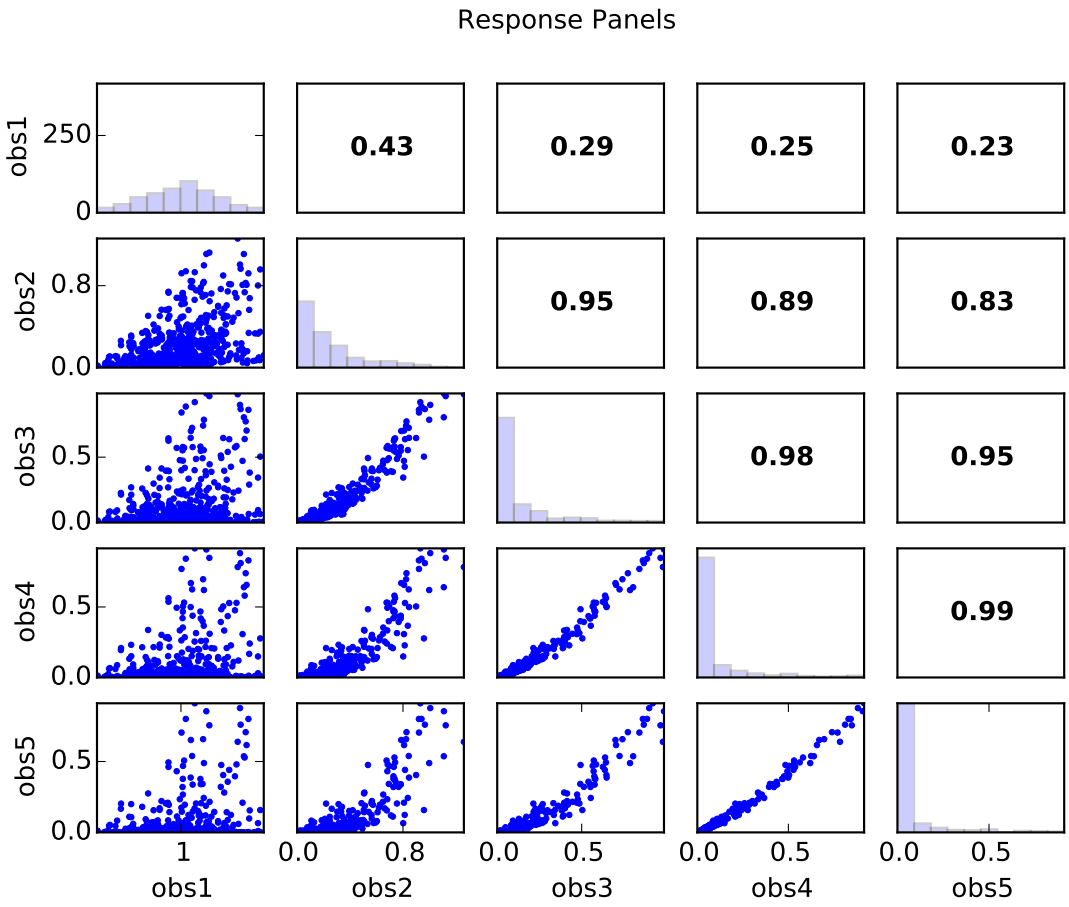

Parameter Correlations
par1
par2
par3
par4
1.00
0.00
-0.01
-0.01
0.00
1.00
-0.00
0.01
-0.01
-0.00
1.00
0.03
-0.01
0.01
0.03
1.00
par1
par2
par3
par4
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0

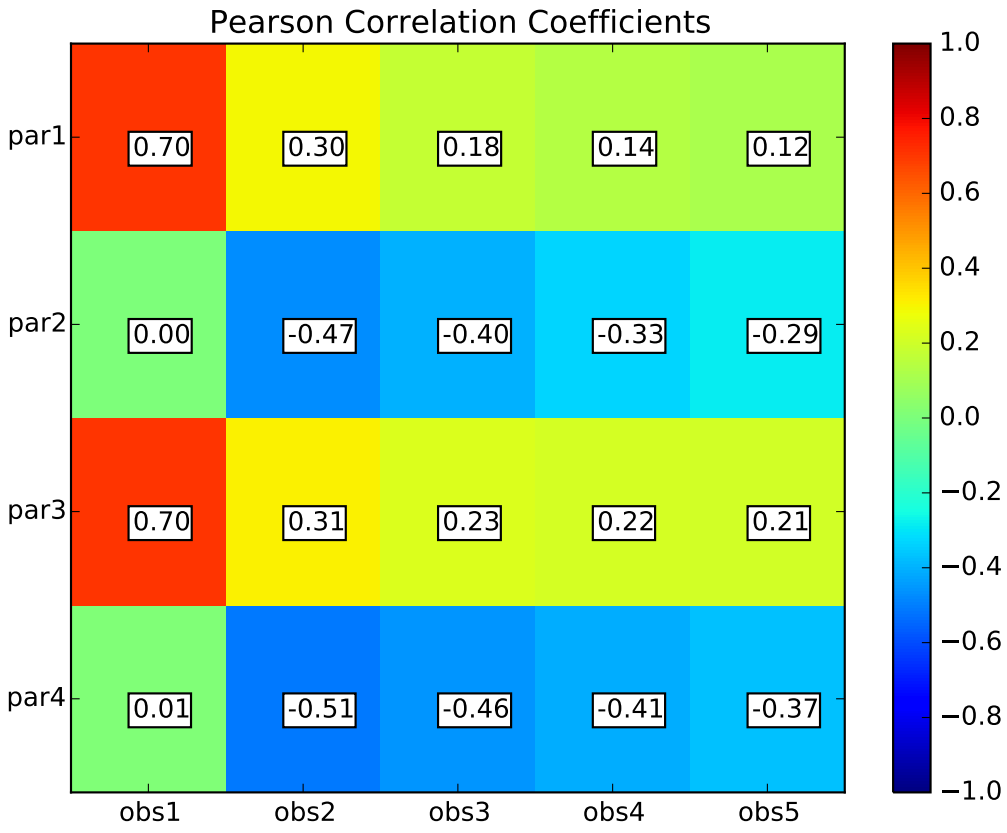

Parameter Panels
par1
par2
par3
par4
0.0
-0.01
-0.01
-0.0
0.01
0.03
0.0
0.5
1.0
0.1
0.2
par1
par2
par3
par4

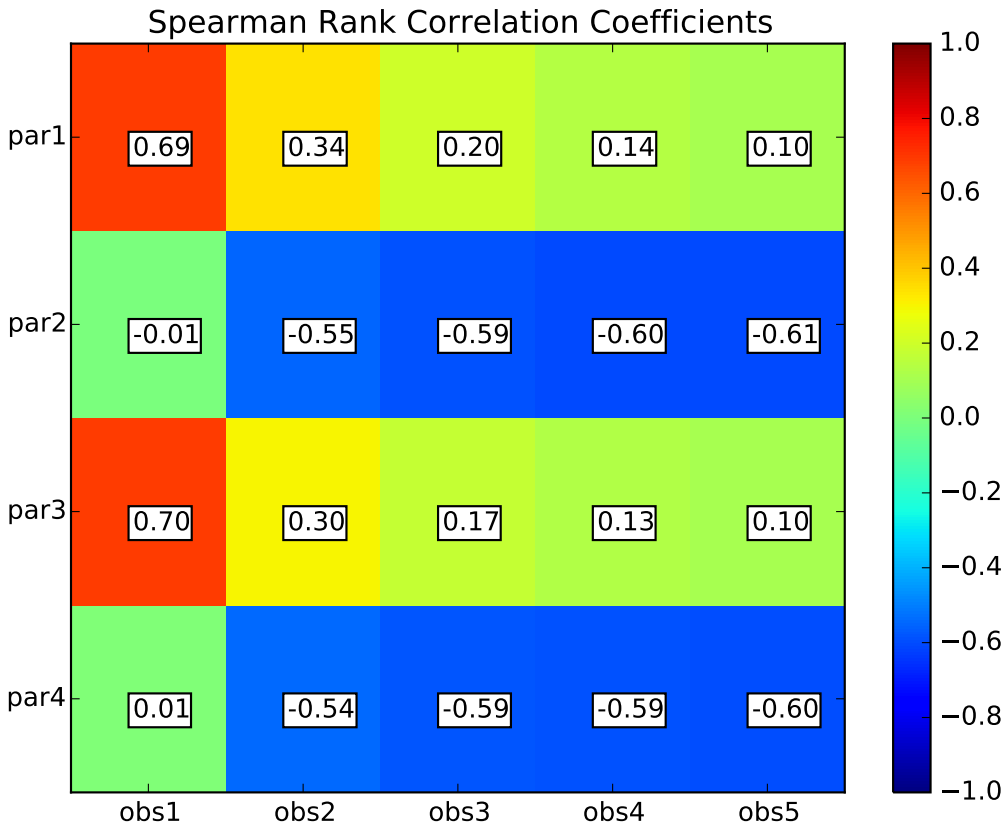

Model Response Histograms by Counts
Count
400
200
0
0.8
1.6
obs1
0.0
0.5
1.0
obs2
0.0
0.4
0.8
obs3
obs4
obs5

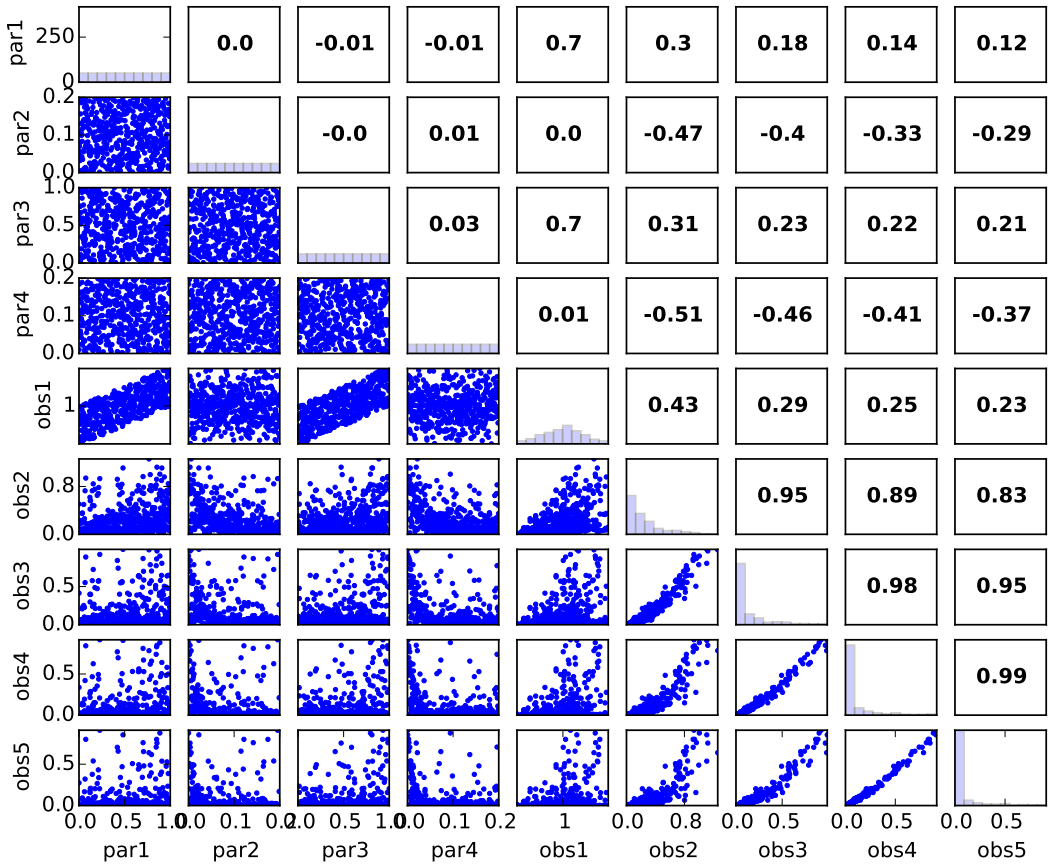

Model Response Histograms by Frequency
Frequency
0.5
0.0
0.8
1.6
obs1
0.0
0.5
1.0
obs2
0.0
0.4
0.8
obs3
obs4
obs5


Model Response Correlations
obs1
obs2
obs3
obs4
obs5
1.00
0.43
0.29
0.25
0.23
0.43
1.00
0.95
0.89
0.83
0.29
0.95
1.00
0.98
0.95
0.25
0.89
0.98
1.00
0.99
0.23
0.83
0.95
0.99
1.00
obs1
obs2
obs3
obs4
obs5
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0


Response Panels
obs1
obs2
obs3
obs4
obs5
0.43
0.29
0.25
0.23
0.95
0.89
0.83
0.98
0.95
0.99
250
0
0.8
0.0
0.5
1
0.8
0.5


Pearson Correlation Coefficients
par1
par2
par3
par4
0.70
0.30
0.18
0.14
0.12
0.00
-0.47
-0.40
-0.33
-0.29
0.70
0.31
0.23
0.22
0.21
0.01
-0.51
-0.46
-0.41
-0.37
obs1
obs2
obs3
obs4
obs5
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0


Spearman Rank Correlation Coefficients
par1
par2
par3
par4
0.69
0.34
0.20
0.14
0.10
-0.01
-0.55
-0.59
-0.60
-0.61
0.70
0.30
0.17
0.13
0.10
0.01
-0.54
-0.59
-0.59
-0.60
obs1
obs2
obs3
obs4
obs5
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0



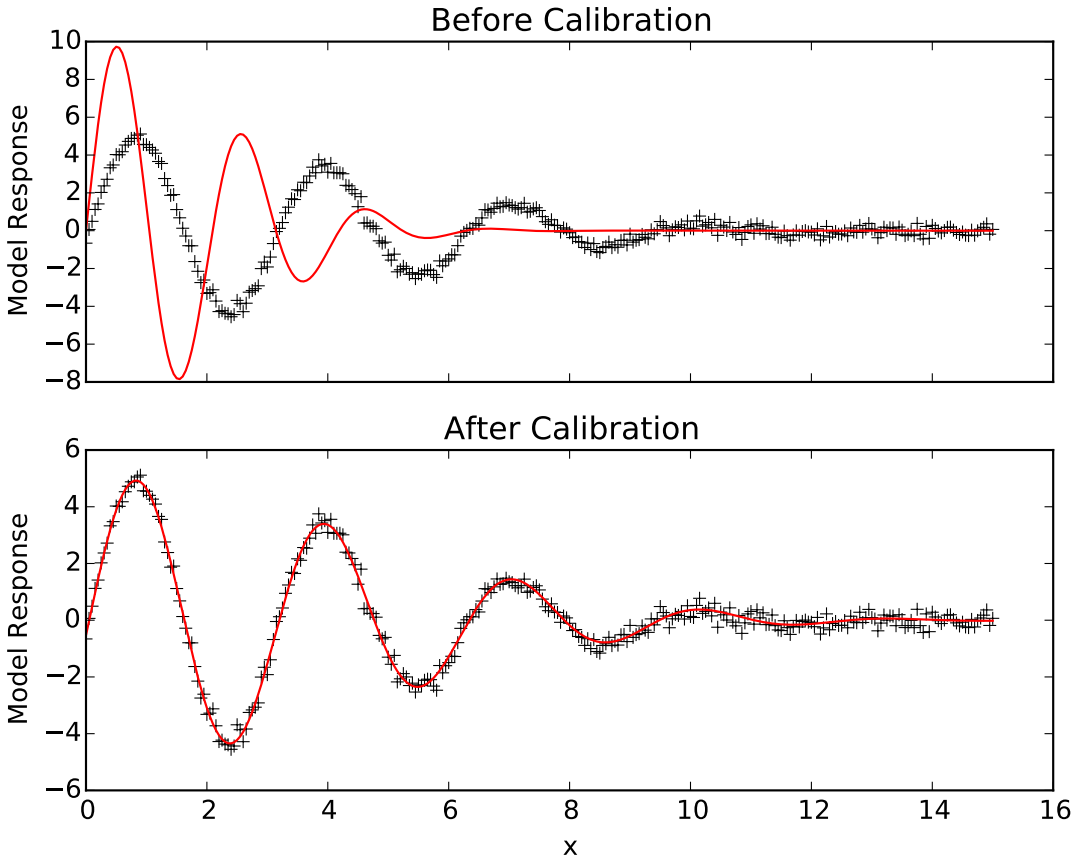
2.2 Calibration

This example demonstrates the calibration of a simple sinusoidal decay model using the `lmfit` function. `lmfit` uses the MINPACK Levenberg-Marquardt algorithm via the lmfit python module.

The plots generated by the script are displayed below the code block. The script *calibrate_sine_lmfit.py* can be downloaded using the **Source code** link below, or the *examples* folder of the repository.

```
# Calibration example modified from lmfit webpage
# (http://cars9.uchicago.edu/software/python/lmfit/parameters.html)
import sys,os
try:
    import matk
except:
    try:
        sys.path.append(os.path.join('..','src','matk'))
        import matk
    except ImportError as err:
        print 'Unable to load MATK module: '+str(err)
import numpy as np
from matplotlib import pyplot as plt
from multiprocessing import freeze_support
```

```
# define objective function: returns the array to be minimized
def sine_decay(params, x, data):
    """ model decaying sine wave, subtract data"""
    amp = params['amp']
    shift = params['shift']
    omega = params['omega']
    decay = params['decay']

    model = amp * np.sin(x * omega + shift) * np.exp(-x*x*decay)

    obsnames = ['obs'+str(i) for i in range(1,len(data)+1)]
    return dict(zip(obsnames,model))


def run():
    # create data to be fitted
    x = np.linspace(0, 15, 301)
    np.random.seed(1000)
    data = (5. * np.sin(2 * x - 0.1) * np.exp(-x*x*0.025) +
            np.random.normal(size=len(x), scale=0.2) )

    # Create MATK object
    p = matk.matk(model=sine_decay, model_args=(x,data,))

    # Create parameters
    p.add_par('amp', value=10, min=0.)
    p.add_par('decay', value=0.1)
    p.add_par('shift', value=0.0, min=-np.pi/2., max=np.pi/2.)
    p.add_par('omega', value=3.0)

    # Create observation names and set observation values
    for i in range(len(data)):
        p.add_obs('obs'+str(i+1), value=data[i])

    # Look at initial fit
    p.forward()
    f, (ax1,ax2) = plt.subplots(2,sharex=True)
    ax1.plot(x,data, 'k+')
    ax1.plot(x,p.simvalues, 'r')
    ax1.set_ylabel("Model Response")
    ax1.set_title("Before Calibration")

    # Calibrate parameters to data, results are printed to screen
    p.lmfit(cpus=2)

    # Look at calibrated fit
    ax2.plot(x,data, 'k+')
    ax2.plot(x,p.simvalues, 'r')
    ax2.set_ylabel("Model Response")
    ax2.set_xlabel("x")
    ax2.set_title("After Calibration")
    f.show()

# Freeze support is necessary for multiprocessing on windows
if __name__== "__main__":
    freeze_support()
    run()
```


Before Calibration
Model Response
10
8
6
4
2
0
−2
−4
−6
−8
After Calibration
Model Response
6
4
2
0
−2
−4
−6
0
2
4
6
8
10
12
14
16
x

CHAPTER

THREE

CLASS DOCUMENTATION

3.1 MATK

class matk.**matk**(*model=''*, *model_args=None*, *model_kwargs=None*, *cpus=1*, *workdir_base=None*, *workdir=None*, *results_file=None*, *seed=None*, *sample_size=10*, *hosts={}*)

Class for Model Analysis ToolKit (MATK) module

Jac(*h=0.001*, *cpus=1*, *workdir_base=None*, *save=True*, *reuse_dirs=False*)

Numerical Jacobian calculation

Parameters **h** (*fl64 or ndarray(fl64)*) – Parameter increment, single value or array with npar values

Returns ndarray(fl64) – Jacobian matrix

MCMC(*nruns=10000*, *burn=1000*, *init_error_std=1.0*, *max_error_std=100.0*, *verbose=1*)

Perform Markov Chain Monte Carlo sampling using pymc package

Parameters

- **nruns** (*int*) – Number of MCMC iterations (samples)
- **burn** (*int*) – Number of initial samples to burn (discard)
- **verbose** (*int*) – verbosity of output
- **init_error_std** (*fl64*) – Initial standard deviation of residuals
- **max_error_std** (*fl64*) – Maximum standard deviation of residuals that will be considered

Returns pymc MCMC object

add_obs(*name*, *sim=None*, *weight=1.0*, *value=None*)

Add observation to problem

Parameters

- **name** (*str*) – Observation name
- **sim** (*fl64*) – Simulated value
- **weight** (*fl64*) – Observation weight
- **value** (*fl64*) – Value of observation

Returns Observation object

add_par(*name*, ***kwargs*)

Add parameter to problem

Parameters

- **name** (*str*) – Name of parameter
- **kwargs** – keyword arguments passed to parameter class

calibrate(*cpus=1*, *maxiter=100*, *lambdax=0.001*, *minchange=1e-16*, *minlambdax=1e-06*, *verbose=False*, *workdir=None*, *reuse_dirs=False*, *h=1e-06*)

Calibrate MATK model using Levenberg-Marquardt algorithm based on original code written by Ernesto P. Adorio PhD. (UPDEPP at Clarkfield, Pampanga)

Parameters

- **cpus** (*int*) – Number of cpus to use
- **maxiter** (*int*) – Maximum number of iterations
- **lambdax** (*fl64*) – Initial Marquardt lambda
- **minchange** (*fl64*) – Minimum change between successive ChiSquares
- **minlambdax** (*fl4*) – Minimum lambda value
- **verbose** (*bool*) – If True, additional information written to screen during calibration

Returns best fit parameters found by routine

Returns best Sum of squares.

Returns covariance matrix

copy_sampleset(*oldname*, *newname=None*)

Copy sampleset

Parameters

- **oldname** (*str*) – Name of sampleset to copy
- **newname** (*str*) – Name of new sampleset

cpus

Set number of cpus to use for concurrent model evaluations

create_sampleset(*samples*, *name=None*, *responses=None*, *indices=None*, *index_start=1*)

Add sample set to problem

Parameters

- **name** (*str*) – Name of sample set
- **samples** (*list(fl64),ndarray(fl64)*) – Matrix of parameter samples with npar columns in order of matk.pars.keys()
- **responses** (*list(fl64),ndarray(fl64)*) – Matrix of associated responses with nobs columns in order matk.obs.keys() if observation exists (existence of observations is not required)
- **indices** (*list(int),ndarray(int)*) – Sample indices to use when creating working directories and output files

emcee(*lnprob=None*, *nwalkers=100*, *nsamples=500*, *burnin=50*, *pos0=None*)

Perform Markov Chain Monte Carlo sampling using emcee package

Parameters

- **lnprob** (*function*) – Function specifying the natural logarithm of the likelihood function
- **nwalkers** (*int*) – Number of random walkers

- **nsamples** (*int*) – Number of samples per walker
- **burnin** (*int*) – Number of "burn-in" samples per walker to be discarded
- **pos0** (*list*) – list of initial positions for the walkers

Returns numpy array containing samples

forward(*pardict=None*, *workdir=None*, *reuse_dirs=False*, *job_number=None*, *hostname=None*, *processor=None*)
Run MATK model using current values

Parameters

- **pardict** (*dict*) – Dictionary of parameter values keyed by parameter names
- **workdir** (*str*) – Name of directory where model will be run. It will be created if it does not exist
- **reuse_dirs** (*bool*) – If True and workdir exists, the model will reuse the directory
- **job_number** (*int*) – Sample id
- **hostname** (*str*) – Name of host to run job on, will be passed to MATK model as kwarg 'hostname'
- **processor** (*str or int*) – Processor id to run job on, will be passed to MATK model as kwarg 'processor'

Returns int – 0: Successful run, 1: workdir exists

levmar(*workdir=None*, *reuse_dirs=False*, *max_iter=1000*, *full_output=True*)
Calibrate MATK model using levmar package

Parameters

- **workdir** (*str*) – Name of directory where model will be run. It will be created if it does not exist
- **reuse_dirs** (*bool*) – If True and workdir exists, the model will reuse the directory
- **max_iter** (*int*) – Maximum number of iterations
- **full_output** – If True, additional output displayed during calibration

Returns levmar output

lhs(*name=None*, *siz=None*, *noCorrRestr=False*, *corrmat=None*, *seed=None*, *index_start=1*)
Draw lhs samples of parameter values from scipy.stats module distribution

Parameters

- **name** (*str*) – Name of sample set to be created
- **siz** (*int*) – Number of samples to generate, ignored if samples are provided
- **noCorrRestr** (*bool*) – If True, correlation structure is not enforced on sample, use if siz is less than number of parameters
- **corrmat** (*matrix*) – Correlation matrix
- **seed** (*int*) – Random seed to allow replication of samples
- **index_start** – Starting value for sample indices

Type int

Returns matrix – Parameter samples

lmfit (*maxfev=0*, *report_fit=True*, *cpus=1*, *epsfcn=None*, *xtol=1e-07*, *ftol=1e-07*, *workdir=None*, *verbose=False*, ***kwargs*)
Calibrate MATK model using lmfit package

Parameters

- **maxfev** (*int*) – Max number of function evaluations, if 0, 100*(npars+1) will be used
- **report_fit** (*bool*) – If True, parameter statistics and correlations are printed to the screen
- **cpus** (*int*) – Number of cpus to use for concurrent simulations during jacobian approximation
- **epsfcn** (*float*) – jacobian finite difference approximation increment
- **xtol** (*float*) – Relative error in approximate solution
- **ftol** (*float*) – Relative error in the desired sum of squares
- **workdir** (*str*) – Name of directory to use for model runs, calibrated parameters will be run there after calibration
- **verbose** (*bool*) – If true, print diagnostic information to the screen

Returns lmfit minimizer object

Additional keyword argments will be passed to scipy leastsq function: http://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.optimize.leastsq.html

make_workdir (*workdir=None*, *reuse_dirs=False*)
Create a working directory

Parameters

- **workdir** (*str*) – Name of directory where model will be run. It will be created if it does not exist
- **reuse_dirs** (*bool*) – If True and workdir exists, the model will reuse the directory

Returns int – 0: Successful run, 1: workdir exists

model
Python function that runs model

model_args
Tuple of extra arguments to MATK model expected to come after parameter dictionary

model_kwargs
Dictionary of extra keyword arguments to MATK model expected to come after parameter dictionary and model_args

obsnames
Get observation names

obsvalues
Observation values

obsweights
Get observation names

pardist_pars
Get parameters needed by parameter distributions

pardists
Get parameter probabilistic distributions

parmaxs
Get parameter lower bounds

parmins
Get parameter lower bounds

parnames
Get parameter names

parstudy (*name=None*, *nvals=2*)
Generate parameter study samples

Parameters

- **name** (*str*) – Name of sample set to be created
- **outfile** (*str*) – Name of file where samples will be written. If outfile=None, no file is written.
- **nvals** (*int or list(int)*) – number of values for each parameter

Returns ndarray(fl64) – Array of samples

parvalues
Parameter values

read_sampleset (*file*, *name=None*)
Read MATK output file and assemble corresponding sampleset with responses.

Parameters

- **name** (*str*) – Name of sample set
- **file** (*str*) – Path to MATK output file

residuals
Get least squares values

results_file
Set the name of the results_file for parallel runs

seed
Set the seed for random sampling

simvalues
Simulated values :returns: lst(fl64) – simulated values in order of matk.obs.keys()

ssr
Sum of squared residuals

workdir
Set the base name for parallel working directories

workdir_base
Set the base name for parallel working directories

workdir_index
Set the working directory index for parallel runs

3.2 Parameter

class matk.parameter.**Parameter**(*name*, *value=None*, *vary=True*, *min=None*, *max=None*, *expr=None*, *discrete_vals=[]*, *discrete_counts=[]*, ***kwargs*)

MATK parameter class

dist
Probabilistic distribution of parameter belonging to scipy.stats module

dist_pars
Distribution parameters required by self.dist e.g. if dist == uniform, dist_pars = (min,max-min)

if dist == norm, dist_pars = (mean,stdev))

expr
Mathematical expression to use to evaluate value

setup_bounds()
set up Minuit-style internal/external parameter transformation of min/max bounds.

returns internal value for parameter from self.value (which holds the external, user-expected value). This internal values should actually be used in a fit....

As a side-effect, this also defines the self.from_internal method used to re-calculate self.value from the internal value, applying the inverse Minuit-style transformation. This method should be called prior to passing a Parameter to the user-defined objective function.

This code borrows heavily from JJ Helmus' leastsqbound.py

value
Parameter value

vary
Boolean indicating whether or not to vary parameter

3.3 Observation

class matk.observation.**Observation**(*name*, *sim=None*, *weight=1.0*, *value=None*)

MATK observation class

name
Observation name

residual
Observation value minus simulated value

sim
Simulated value generated by MATK model

value
Observation value

weight
Weight to apply to simulated values

3.4 SampleSet

class matk.sampleset.**SampleSet**(*name*, *samples*, *parent*, *index_start=1*, ***kwargs*)

MATK SampleSet class - Stores information related to a sample including parameter samples, associated responses, and sample indices

calc_sse()

Calculate sum of squared errors (sse) for all samples

Returns lst(fl64)

corr(*type='pearson'*, *plot=False*, *printout=True*, *plotvals=True*, *figsize=None*, *title=None*)

Calculate correlation coefficients of parameters and responses

Parameters

- **type** (*str*) – Type of correlation coefficient (pearson by default, spearman also avaialable)
- **plot** (*bool*) – If True, plot correlation matrix
- **printout** (*bool*) – If True, print correlation matrix with row and column headings
- **plotvals** (*bool*) – If True, print correlation coefficients on plot matrix
- **figsize** (*tuple(fl64,fl64)*) – Width and height of figure in inches
- **title** (*str*) – Title of plot

Returns ndarray(fl64) – Correlation coefficients

index_start

Starting integer value for sample indices

indices

Array of sample indices

main_effects()

For each parameter, compile array of main effects.

name

Sample set name

obsnames

Array of observation names

panels(*type='pearson'*, *alpha=0.2*, *figsize=None*, *title=None*, *tight=False*, *symbol='.'*, *fontsize=None*, *corrfontsize=None*, *ms=5*, *mins=None*, *maxs=None*, *frequency=False*, *bins=10*, *ylim=None*, *labels=[]*, *filename=None*, *xticks=2*, *yticks=2*)

Plot histograms, scatterplots, and correlation coefficients in paired matrix

Parameters

- **type** (*str*) – Type of correlation coefficient (pearson by default, spearman also avaialable)
- **alpha** (*float*) – Histogram color shading
- **figsize** (*tuple(fl64,fl64)*) – Width and height of figure in inches
- **title** (*str*) – Title of plot
- **tight** (*bool*) – Use matplotlib tight layout
- **symbol** (*str*) – matplotlib symbol for scatterplots
- **fontsize** (*fl64*) – Size of font for axis labels

- **corrfontsize** (*fl64*) – Size of font for correlation coefficients
- **ms** (*fl64*) – Scatterplot marker size
- **frequency** (*bool*) – If True, the first element of the return tuple will be the counts normalized by the length of data, i.e., n/len(x)
- **bins** (*int*) – Number of bins in histograms
- **ylim** (*tuples - 2 element tuples with y limits for histograms*) – y-axis limits for histograms.
- **labels** (*lst(str)*) – Names to use instead of parameter names in plot
- **filename** (*int*) – Name of file to save plot. File ending determines plot type (pdf, png, ps, eps, etc.). Plot types available depends on the matplotlib backend in use on the system. Plot will not be displayed.
- **xticks** – Number of ticks along x axes
- **yticks** – Number of ticks along y axes

pardict(*index*)
Get parameter dictionary for sample with specified index

Parameters **index** (*int*) – Sample index

Returns dict(fl64)

parnames
Array of observation names

rank_parameter_frequencies()
Yields a printout of parameter value frequencies in the sample set

returns An array of tuples, each containing the parameter name tagged as min or max and a second tuple containing the parameter value and the frequency of its appearance in the sample set.

recarray
Structured (record) array of samples

run(*cpus=1*, *workdir_base=None*, *save=True*, *reuse_dirs=False*, *outfile=None*, *logfile=None*, *verbose=True*, *hosts={}*)
Run model using values in samples for parameter values If samples are not specified, LHS samples are produced

Parameters

- **cpus** (*int,dict(lst)*) – number of cpus; alternatively, dictionary of lists of processor ids keyed by hostnames to run models on (i.e. on a cluster); hostname provided as kwarg to model (hostname=<hostname>); processor id provided as kwarg to model (processor=<processor id>)
- **workdir_base** (*str*) – Base name for model run folders, run index is appended to workdir_base
- **save** (*bool*) – If True, model files and folders will not be deleted during parallel model execution
- **reuse_dirs** (*bool*) – Will use existing directories if True, will return an error if False and directory exists
- **outfile** (*str*) – File to write results to
- **logfile** (*str*) – File to write details of run to during execution
- **hosts** (*lst(str)*) – Option deprecated, use cpus instead

Returns tuple(ndarray(fl64),ndarray(fl64)) - (Matrix of responses from sampled model runs siz rows by npar columns, Parameter samples, same as input samples if provided)

savetxt(*outfile*)
Save sampleset to file

Parameters **outfile** (*str*) – Name of file where sampleset will be written

subset(*boolfcn*, *obs*, **args*, ***kwargs*)
Collect samples based on response values, remove all others

Parameters

- **boofcn** – Function that returns true for samples to keep and false for samples to remove
- **obs** (*str*) – Name of response to apply boolfcn to
- **args** – Additional arguments to add to boolfcn
- **kwargs** – Keyword arguments to add to boolfcn

matk.sampleset.**hist**(*rc*, *ncols=4*, *figsize=None*, *alpha=0.2*, *title=None*, *tight=False*, *mins=None*, *maxs=None*, *frequency=False*, *bins=10*, *ylim=None*, *printout=True*, *labels=[]*, *filename=None*, *fontsize=None*, *xticks=3*)
Plot histograms of dataset

Parameters

- **ncols** (*int*) – Number of columns in plot matrix
- **figsize** (*tuple(fl64,fl64)*) – Width and height of figure in inches
- **alpha** (*float*) – Histogram color shading
- **title** (*str*) – Title of plot
- **tight** (*bool*) – Use matplotlib tight layout
- **mins** (*lst(fl64)*) – Minimum values of recarray fields
- **maxs** (*lst(fl64)*) – Maximum values of recarray fields
- **frequency** (*bool*) – If True, the first element of the return tuple will be the counts normalized by the length of data, i.e., n/len(x)
- **bins** (*int or lst(lst(int))*) – If an integer is given, bins + 1 bin edges are returned. Unequally spaced bins are supported if bins is a list of sequences for each histogram.
- **ylim** (*tuples - 2 element tuple with y limits for histograms*) – y-axis limits for histograms.
- **labels** (*lst(str)*) – Names to use instead of parameter names in plot
- **filename** (*str*) – Name of file to save plot. File ending determines plot type (pdf, png, ps, eps, etc.). Plot types available depends on the matplotlib backend in use on the system. Plot will not be displayed.
- **fontsize** (*fl64*) – Size of font
- **xticks** (*int*) – Number of ticks on xaxes

Returns dict(lst(int),lst(fl64)) - dictionary of histogram data (counts,bins) keyed by name

matk.sampleset.**corr**(*rc1*, *rc2*, *type='pearson'*, *plot=False*, *printout=True*, *plotvals=True*, *figsize=None*, *title=None*)
Calculate correlation coefficients of parameters and responses

Parameters

- **rc1** – Data
- **rc2** – Data
- **type** (*str*) – Type of correlation coefficient (pearson by default, spearman also avaialable)
- **plot** (*bool*) – If True, plot correlation matrix
- **printout** (*bool*) – If True, print correlation matrix with row and column headings
- **plotvals** (*bool*) – If True, print correlation coefficients on plot matrix
- **figsize** (*tuple(fl64,fl64)*) – Width and height of figure in inches
- **title** (*str*) – Title of plot

Returns ndarray(fl64) – Correlation coefficients

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

m

matk.observation, 22
matk.parameter, 22
matk.sampleset, 2

A

add_obs() (matk.matk method), 17
add_par() (matk.matk method), 17

C

calc_sse() (matk.sampleset.SampleSet method), 23
calibrate() (matk.matk method), 18
copy_sampleset() (matk.matk method), 18
corr() (in module matk.sampleset), 25
corr() (matk.sampleset.SampleSet method), 23
cpus (matk.matk attribute), 18
create_sampleset() (matk.matk method), 18

D

dist (matk.parameter.Parameter attribute), 22
dist_pars (matk.parameter.Parameter attribute), 22

E

emcee() (matk.matk method), 18
expr (matk.parameter.Parameter attribute), 22

F

forward() (matk.matk method), 19

H

hist() (in module matk.sampleset), 25

I

index_start (matk.sampleset.SampleSet attribute), 23
indices (matk.sampleset.SampleSet attribute), 23

J

Jac() (matk.matk method), 17

L

levmar() (matk.matk method), 19
lhs() (matk.matk method), 19
lmfit() (matk.matk method), 19

M

main_effects() (matk.sampleset.SampleSet method), 23
make_workdir() (matk.matk method), 20
matk (class in matk), 17
matk (module), 17
matk.observation (module), 22
matk.parameter (module), 22
matk.sampleset (module), 2, 16, 23
MCMC() (matk.matk method), 17
model (matk.matk attribute), 20
model_args (matk.matk attribute), 20
model_kwargs (matk.matk attribute), 20

N

name (matk.observation.Observation attribute), 22
name (matk.sampleset.SampleSet attribute), 23

O

Observation (class in matk.observation), 22
obsnames (matk.matk attribute), 20
obsnames (matk.sampleset.SampleSet attribute), 23
obsvalues (matk.matk attribute), 20
obsweights (matk.matk attribute), 20

P

panels() (matk.sampleset.SampleSet method), 23
Parameter (class in matk.parameter), 22
pardict() (matk.sampleset.SampleSet method), 24
pardist_pars (matk.matk attribute), 20
pardists (matk.matk attribute), 20
parmaxs (matk.matk attribute), 20
parmins (matk.matk attribute), 21
parnames (matk.matk attribute), 21
parnames (matk.sampleset.SampleSet attribute), 24
parstudy() (matk.matk method), 21
parvalues (matk.matk attribute), 21

R

rank_parameter_frequencies() (matk.sampleset.SampleSet method), 24
read_sampleset() (matk.matk method), 21
recarray (matk.sampleset.SampleSet attribute), 24
residual (matk.observation.Observation attribute), 22
residuals (matk.matk attribute), 21

results_file (matk.matk attribute), 21
run() (matk.sampleset.SampleSet method), 24

S

SampleSet (class in matk.sampleset), 23
savetxt() (matk.sampleset.SampleSet method), 25
seed (matk.matk attribute), 21
setup_bounds() (matk.parameter.Parameter method), 22
sim (matk.observation.Observation attribute), 22
simvalues (matk.matk attribute), 21
ssr (matk.matk attribute), 21
subset() (matk.sampleset.SampleSet method), 25

V

value (matk.observation.Observation attribute), 22
value (matk.parameter.Parameter attribute), 22
vary (matk.parameter.Parameter attribute), 22

W

weight (matk.observation.Observation attribute), 22
workdir (matk.matk attribute), 21
workdir_base (matk.matk attribute), 21
workdir_index (matk.matk attribute), 21