Time Series analysis `tsa`

`statsmodels.tsa` contains model classes and functions that are useful for time series analysis. Basic models include univariate autoregressive models (AR), vector autoregressive models (VAR) and univariate autoregressive moving average models (ARMA). Non-linear models include Markov switching dynamic regression and autoregression. It also includes descriptive statistics for time series, for example autocorrelation, partial autocorrelation function and periodogram, as well as the corresponding theoretical properties of ARMA or related processes. It also includes methods to work with autoregressive and moving average lag-polynomials. Additionally, related statistical tests and some useful helper functions are available.

Estimation is either done by exact or conditional Maximum Likelihood or conditional least-squares, either using Kalman Filter or direct filters.

Currently, functions and classes have to be imported from the corresponding module, but the main classes will be made available in the statsmodels.tsa namespace. The module structure is within statsmodels.tsa is

- stattools : empirical properties and tests, acf, pacf, granger-causality, adf unit root test, kpss test, bds test, ljung-box test and others.
- ar_model : univariate autoregressive process, estimation with conditional and exact maximum likelihood and conditional least-squares
- arima_model : univariate ARMA process, estimation with conditional and exact maximum likelihood and conditional least-squares
- statespace : Comprehensive statespace model specification and estimation. See the statespace documentation.
- vector_ar, var : vector autoregressive process (VAR) and vector error correction models, estimation, impulse response analysis, forecast error variance decompositions, and data visualization tools. See the vector_ar documentation.
- kalmanf : estimation classes for ARMA and other models with exact MLE using Kalman Filter
- arma_process : properties of arma processes with given parameters, this includes tools to convert between ARMA, MA and AR representation as well as acf, pacf, spectral density, impulse response function and similar
- sandbox.tsa.fftarma : similar to arma_process but working in frequency domain
- tsatools : additional helper functions, to create arrays of lagged variables, construct regressors for trend, detrend and similar.
- filters : helper function for filtering time series
- regime_switching : Markov switching dynamic regression and autoregression models

Some additional functions that are also useful for time series analysis are in other parts of statsmodels, for example additional statistical tests.

Some related functions are also available in matplotlib, nitime, and scikits.talkbox. Those functions are designed more for the use in signal processing where longer time series are available and work more often in the frequency domain.

Descriptive Statistics and Tests

`stattools.acovf`(x[, unbiased, demean, fft, …])	Autocovariance for 1D
`stattools.acf`(x[, unbiased, nlags, qstat, …])	Autocorrelation function for 1d arrays.
`stattools.pacf`(x[, nlags, method, alpha])	Partial autocorrelation estimated
`stattools.pacf_yw`(x[, nlags, method])	Partial autocorrelation estimated with non-recursive yule_walker
`stattools.pacf_ols`(x[, nlags, efficient, …])	Calculate partial autocorrelations via OLS
`stattools.pacf_burg`(x[, nlags, demean])	Burg’s partial autocorrelation estimator
`stattools.ccovf`(x, y[, unbiased, demean])	crosscovariance for 1D
`stattools.ccf`(x, y[, unbiased])	cross-correlation function for 1d
`stattools.periodogram`(X)	Returns the periodogram for the natural frequency of X

stattools.adfuller(x[, maxlag, regression, …])	Augmented Dickey-Fuller unit root test
stattools.kpss(x[, regression, lags, store])	Kwiatkowski-Phillips-Schmidt-Shin test for stationarity.
stattools.coint(y0, y1[, trend, method, …])	Test for no-cointegration of a univariate equation
stattools.bds(x[, max_dim, epsilon, distance])	Calculate the BDS test statistic for independence of a time series
stattools.q_stat(x, nobs[, type])	Return’s Ljung-Box Q Statistic
stattools.grangercausalitytests(x, maxlag[, …])	four tests for granger non causality of 2 timeseries
stattools.levinson_durbin(s[, nlags, isacov])	Levinson-Durbin recursion for autoregressive processes
stattools.innovations_algo(acov[, nobs, rtol])	Innovations algorithm to convert autocovariances to MA parameters
stattools.innovations_filter(endog, theta)	Filter observations using the innovations algorithm
stattools.levinson_durbin_pacf(pacf[, nlags])	Levinson-Durbin algorithm that returns the acf and ar coefficients
stattools.arma_order_select_ic(y[, max_ar, …])	Returns information criteria for many ARMA models
x13.x13_arima_select_order(endog[, …])	Perform automatic seaonal ARIMA order identification using x12/x13 ARIMA.
x13.x13_arima_analysis(endog[, maxorder, …])	Perform x13-arima analysis for monthly or quarterly data.

Estimation

The following are the main estimation classes, which can be accessed through statsmodels.tsa.api and their result classes

Univariate Autogressive Processes (AR)

ar_model.AR(endog[, dates, freq, missing])	Autoregressive AR(p) model
ar_model.ARResults(model, params[, …])	Class to hold results from fitting an AR model.

Autogressive Moving-Average Processes (ARMA) and Kalman Filter

The basic ARIMA model and results classes that should be the starting point for for most users are:

arima_model.ARMA(endog, order[, exog, …])	Autoregressive Moving Average ARMA(p,q) Model
arima_model.ARMAResults(model, params[, …])	Class to hold results from fitting an ARMA model.
arima_model.ARIMA(endog, order[, exog, …])	Autoregressive Integrated Moving Average ARIMA(p,d,q) Model
arima_model.ARIMAResults(model, params[, …])	**Methods**

Some advanced underlying low-level classes and functions that can be used to compute the log-likelihood function for ARMA-type models include (note that these are rarely needed by end-users):

kalmanf.kalmanfilter.KalmanFilter	Kalman Filter code intended for use with the ARMA model.
innovations.arma_innovations.arma_innovations(endog)	Compute innovations using a given ARMA process
innovations.arma_innovations.arma_loglike(endog)	Compute loglikelihood of the given data assuming an ARMA process
innovations.arma_innovations.arma_loglikeobs(endog)	Compute loglikelihood for each observation assuming an ARMA process

innovations.arma_innovations.arma_score(endog)	Compute the score (gradient of the loglikelihood function)
innovations.arma_innovations.arma_scoreobs(endog)	Compute the score per observation (gradient of the loglikelihood function)

Exponential Smoothing

holtwinters.ExponentialSmoothing(endog[, …])	Holt Winter's Exponential Smoothing
holtwinters.SimpleExpSmoothing(endog)	Simple Exponential Smoothing
holtwinters.Holt(endog[, exponential, damped])	Holt's Exponential Smoothing
holtwinters.HoltWintersResults(model, …)	Holt Winter's Exponential Smoothing Results

ARMA Process

The following are tools to work with the theoretical properties of an ARMA process for given lag-polynomials.

arima_process.ArmaProcess([ar, ma, nobs])	Theoretical properties of an ARMA process for specified lag-polynomials
arima_process.ar2arma(ar_des, p, q[, n, …])	Find arma approximation to ar process
arima_process.arma2ar(ar, ma[, lags])	Get the AR representation of an ARMA process
arima_process.arma2ma(ar, ma[, lags])	Get the MA representation of an ARMA process
arima_process.arma_acf(ar, ma[, lags])	Theoretical autocorrelation function of an ARMA process
arima_process.arma_acovf(ar, ma[, nobs, …])	Theoretical autocovariance function of ARMA process
arima_process.arma_generate_sample(ar, ma, …)	Generate a random sample of an ARMA process
arima_process.arma_impulse_response(ar, ma)	Get the impulse response function (MA representation) for ARMA process
arima_process.arma_pacf(ar, ma[, lags])	Partial autocorrelation function of an ARMA process
arima_process.arma_periodogram(ar, ma[, …])	Periodogram for ARMA process given by lag-polynomials ar and ma
arima_process.deconvolve(num, den[, n])	Deconvolves divisor out of signal, division of polynomials for n terms
arima_process.index2lpol(coeffs, index)	Expand coefficients to lag poly
arima_process.lpol2index(ar)	Remove zeros from lag polynomial
arima_process.lpol_fiar(d[, n])	AR representation of fractional integration
arima_process.lpol_fima(d[, n])	MA representation of fractional integration
arima_process.lpol_sdiff(s)	return coefficients for seasonal difference (1-L^s)
sandbox.tsa.fftarma.ArmaFft(ar, ma, n)	fft tools for arma processes

Statespace Models

See the statespace documentation..

Vector ARs and Vector Error Correction Models

See the vector_ar documentation..

Regime switching models

regime_switching.markov_regression.MarkovRegression(…)	First-order k-regime Mar-

	kov switching regression model
`regime_switching.markov_autoregression.MarkovAutoregression`(…)	Markov switching regression model

Time Series Filters

`filters.bk_filter.bkfilter`(X[, low, high, K])	Baxter-King bandpass filter
`filters.hp_filter.hpfilter`(X[, lamb])	Hodrick-Prescott filter
`filters.cf_filter.cffilter`(X[, low, high, drift])	Christiano Fitzgerald asymmetric, random walk filter
`filters.filtertools.convolution_filter`(x, filt)	Linear filtering via convolution.
`filters.filtertools.recursive_filter`(x, ar_coeff)	Autoregressive, or recursive, filtering.
`filters.filtertools.miso_lfilter`(ar, ma, x)	use nd convolution to merge inputs, then use lfilter to produce output
`filters.filtertools.fftconvolve3`(in1[, in2, …])	Convolve two N-dimensional arrays using FFT.
`filters.filtertools.fftconvolveinv`(in1, in2)	Convolve two N-dimensional arrays using FFT.
`seasonal.seasonal_decompose`(x[, model, …])	Seasonal decomposition using moving averages

TSA Tools

`tsatools.add_trend`(x[, trend, prepend, …])	Adds a trend and/or constant to an array.
`tsatools.detrend`(x[, order, axis])	Detrend an array with a trend of given order along axis 0 or 1
`tsatools.lagmat`(x, maxlag[, trim, original, …])	Create 2d array of lags
`tsatools.lagmat2ds`(x, maxlag0[, maxlagex, …])	Generate lagmatrix for 2d array, columns arranged by variables

VARMA Process

`varma_process.VarmaPoly`(ar[, ma])	class to keep track of Varma polynomial format

Interpolation

`interp.denton.dentonm`(indicator, benchmark)	Modified Denton’s method to convert low-frequency to high-frequency data.