НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ "ВЫСШАЯ ШКОЛА ЭКОНОМИКИ"

Факультет экономических наук

Бакалаврская курсовая работа

Астахова Ирина

Практическое исследование результативности применения различных моделей прогнозирования временных рядов на примере реальных данных.

Кафедра математической статистики и теории вероятностей

Научный руководитель: Демешев Б.Б.
Образовательная программа: Экономика

Москва
2018

Введение
Цели, задачи, актуальность работы

Прогнозирование временных рядов является крайне важной задачей для дальнейшего развития экономики в наше время по нескольким причинам.

Причина первая – избыток информации вынуждает применять новые алгоритмы ее анализа и отражения. Из-за наличия огромного количества информации в открытом доступе, быстрого ее распространения и множества способов к ее получению, конкуренция в бизнесе стала более жесткой, а период мониторинга основных изменений сократился, компании постоянно занимаются внедрением чего-то нового, чтобы не отстать от своих конкурентов, не потерять долю принадлежащешго им рынка, компании оказались слишком «близко» друг к другу в том плане, что все изменения, которые просходят в бизнесе, сразу становятся видны. И так же быстро за ним следует реакция потребительской аудитории, которая тоже за последние 100 лет стала гораздо более разношерстной, стратифицированной, в ней возмжно выделять кластеры с различными интересами, стилями жизни и тд, взаимодействие со всеми ними различно, и чтобы правильно понеять реакцию толпы, уже недостаточно так называемого «общественного мнения», понадобился анализ данных, который в последние 10 лет является абсолютным трендом во всех направлениях, так или иначе связанных с экономикой и хозяйственной деятельностью разных уровней, управлением. Но еще 30 лет назад не могло быть и речи о том, чтобы анализировать такие объемы данных. Теперь это стало осознанной необходимостью, но данные в статике это не более чем реализация ряда связанных или несвязанных друг с другом случайных величин. А данные в динамике уже в большинстве случаев являются разного вида временными рядами.

Причина вторая-большие компании вынуждены планировать свое развитие и инновационное направление, чтобы рассчитать и предугадать все возможные сценарии развития существующей ситуации, и связанные с тем затраты и выгоды. Сценарный подход успел зарекомендовать себя с точки зрения учета реализации институциональных и другого вида экзогенных рисков, компании предпочитают выявлять свое взвешенное с учетом рисков положение и прогнозировать его максимально точно.

Но при этом данная высокая цель не всегда может быть успешно достигнута на практике. Очень часто это происходит не из-за недостатков математических моделей, заложенных в пакеты прогнозирования, а по причине недостаточно длинного ряда данных. В подобной ситуации масштабные модели, которые способны учесть линейные и нелинейные цеплочки зависимости, использовать много гиперпараметров, очень часто проигрывают по метрикам качества более простым алгоритмам.

Цель данной работы – попытаться посмотреть на реализацию разных методов прогнозирования временных рядов и выявить особенности, достоинства и недостатки каждой, постараться сравнить их прогнозную силу.

В качестве примера можно привести модели из макроэконометрики(Benedictow et al., 2013), которые специфицируются с помощью большого количества уравнений. Если мы используем более гибкий подход, который строится на основе векторных авторегрессий, то вынуждены брать малое количество параметров, с наиболее вероятно существующей зависимостью, образующих уже некое соотношение, или используя знаковые ограничения на функции импульсивных откликов или воспринимая существующие переменные как наиболее важные детерминанты процесса.

Обзор Литературы

В самом начале истории прогнозирования данных основополагающим предположением всех специалистов являлась детерминированность. Детерминированность в том смысле, что по поведению временного ряда на некотором даже очень малом промежутке можно судить и о его поведения вне этого интервала. Но в XX веке произошел переход к стохастическому подходу, то есть к убеждению, что величина временного ряда является случайной на самом деле случайной. И одной из статей, в которой временной ряд рассматривался как случайный процесс, стала работа Yule (1927). После этого момента началось бурное развитие одномерных методов моделирования временных рядов.

1.1. Одномерные модели временных рядов

Экспоненциальное сглаживание и модели авторегрессионного интегрированного скользящего среднего являются самыми простыми по спецификации одномерными моделями прогнозирования временных рядов, но при этом самыми популярными, при этом они не повторяют и не противоречат друг другу, а дополняют одна другую, экспоненциальное сглаживание прежде всего отражает тренд и сезонность в имеющихся данных, а интегрированное скользящее среднее специализируется в большей степени на автокорреляционной связи между наблюдениями.

Модели экспоненциального сглаживания активно описываются в литературе в период 1950-х годов. Среди наиболее заметных статей можно перечислить Holt (1957), Winters (1960), именно там впервые был использован прогноз, который базировался на убывании весов более ранних, отдаленных от момента предскахзани янаблюдений.

Модели экспоненциального сглаживания начинают упоминаться в литературе 1950-х годов. В статьях Holt (1957), Brown (1959) и Winters (1960) впервые использовался прогноз, представляющий собой среднее взвешанное всех доступных наблюдений с экспоненциально убывающими весами. В статье Muth (1960) впервые обсуждаются статистические свойства времен- ных рядов, которые описываются моделью простого экспоненциального сглаживания. Автор показал, что такая модель является оптимальной для прогнозирования случайного блуждания с шумом.

При этом большое количество авторов сделали попытку выйти за рамки прогнозирования на 1 шаг вперед и таким образом сначала появился метод Хольта, где тренд аддитивен, а затем родилась модель Хольта-Уинтерса: где помимо аддитивного трегда присутствует еще и сезонность(причем она может быть и аддитивной и мультипликативной).

В статье Hyndman, Koehler, Snyder, and Grose (2002) подробно рассматриваются все возможные спецификации моделей экспоненциального сглаживания, их формулировки в виде моделей пространства состояний с мультипликативными и аддитивными ошибками, а также сравнивается качество прогнозов, полученных в результате применения разных спецификаций к различным временным рядам.

Теперь стоит сказать несколько слов о модели ARIMA, которая на момент написания этой курсовой является самой популярной из применяемых в прогнозировании временных рядов одномерных моделей.

ARIMA является комбинацией авторегрессии (AR) и модели скользящего среднего (MA), допускающей дифференцирование исходного ряда. К чему дифференцирование? Дело в том, что даже если исходные данные имеют ярко выраженный тренд, скорее всего разности межлу наблюдениями или же темпы прироста наблюдений являются стационарным рядом, то есть ошибка не имеет систематического характера, распределена нормально с нулевым матожиданием и ненулевой, но постоянной дисперсией, а также сами разности являются стационарным в широком смысле рядом. Стационарность в широком смысле(предполагающая постоянство матожидания, дисперсии и автокореляции n-ого порядка) –менее строгое понятие чем стационарность в узком смысле, предполагающая одинаковое распределение находящихся рядом n наблюдений, и потому встречается гораздо чаще на практике и эмпирические наблюдения показывают, что большинство временных рядов, если взять разности первого или второго порядков, превращаются, наконец, в стационарные. Поэтому можно сказать, что если первые разности временного ряда стационарны и описываются моделью ARMA, то сам ряд может описываться моделью ARIMA. Внушительное число работ было посвящено выбору спецификаций модели, оценке коэффициентов и прогнозированию с использованием моделей из семейства ARIMA. В результате, все основные, доступные на тот момент, научные достижения в этой области были

объединены в работе Box & Jenkins (1970). В своей книге авторы подробно описывают разработанный ими подход к анализу временных рядов, известный до сих пор как методология Бокса-Дженкинса, который состоит из выбора подхо- дящей модели, оценки параметров и проверки соблюдения всех необходи мых предпосылок.

Когда моделей прогнозирования было выявлено уже достаточно много, и к проблеме построения прогноза уже успели подойти и со стороны ошибок, и со стороны прежних значений временного ряда, вопрос встал об оъединениии прогнозов различных компаний

Одной из первых работ на эту тему считается статья Bates and Granger (1969), где авторы описали несколько способов комбинации точечных прогнозов и показали, что такая техника способна улучшить качество прогноза по сравнению с результатами каждого метода по отдельности. На данный момент существует огромное количество способов комбинации прогнозов. Например, в статье Granger and Ramanathan (1984) было предложено оценивать веса прогнозов, используя метод наименьших квадратов на прошлых данных, а Deutsch, Granger, and Terasvirta (1994) предлагали использовать веса, изменяющиеся во времени.

1.2. Векторная авторегрессия

Одним из наиболее популярных в ряду многомерных моделей прогнозироания является векторная автоегрессия (VAR). На самом деле совместная динамика временных рядов не терпит никаких изменений в рамках данной модели.

У VAR моделей существует большое количество спецификаций, благодаря чему их можно использовать для прогнозирования различных видов данных. Так, в статье Funke (1990) сравниваются результаты прогноза одномерной модели ARIMA и пяти разных спецификаций VAR модели (в частности структурной и байесовской VAR с различными априорными распределениями). Но есть один важный минус, на который стоит обратить внимание-VAR модели часто называют излишне параметризированными, причем при увеличении размерности модели или с ростом числа включае- мых лагов, параметризация рискует привести к невозможности вычисления оценок коэффициентов.

Для решения описанной выше проблемы Litterman (1986) предложил ввести ограничение в виде априорного распределения параметров. Модели, в которых используется байесовский подход к оценке коэффициентов, были названы BVAR (Bayesian Vector Autoregression). Главный недостаток байесовских моделей, необходимость использования сопряженных распределений, перестал быть актуальным в современной науке благодаря высокой скорости, с которой современные компьютеры проводят симуляции и, как следствие, возможности применения методов MCMC (Markov Chain

Monte Carlo). К плюсам данного подхода несомненно можно отнести тот факт, что теперь, в первую очередь, стало возможным включение огромного числа переменных и лагов.

Также, делая выбор между различными априорными распределениями, исследователи могут снизить неопределенность относительно значений параметров, опираясь на опыт коллег или общепринятые представления о динамике тех или иных переменных. Выгодным преимуществом модели BVAR является ее экзотический подход к определению априорных распределений, то есть веры экспериментатора в то, что детерминанта у может быть распределена не нормально, к примеру, а по закону Хи-квадрат и так далее. Безусловно наши первоначальные представления могут оказаться совершенно ошибочными, потому что не всегда нам дано знать природу поступивших данных, но эта ошибка ничего не будет стоить, потому что наши апрриорные представления нужны только с целью нахождения апостериореной функции распределения наблюдаемых величин, и чем больше наблюдений находится в нашем распоряжении, тем меньше влияние на конечный результат наших первоначальных предположений. Но это достоинство может обернуться недостатком в случае недостатка широты выборки, в этом случае качество прогнозирования будет напрямую зависеть от того, насколько точна оказалась наша оценка априорного распределения величин.

Во многих исследованиях BVAR используют для прогнозирования всевозможных временных рядов. Так, можно привести примеры прогнозирования макроэкономических данных (Artis and Zhang (1990); Beauchemin, Kenneth and Zaman (2011); Berg,Oliver and Henzel (2013)), рыночных долей компаний (Ramos (2003)), финансовых рядов (Carriero, Kapetanios and Marcellino (2012))

1.3. Модели с переменными параметрами

Модели, в которых параметры изменяются со временем, достаточно широко используются в макроэкономике и финансах. Такая тенденция вполне объяснима, так как закономерностям в данных отраслях свойственно изменяться по мере развития технологического прогресса, изменения конъюнктуры и перемен в обществе. Одной из первых работ, в которой байесовская векторная авторегрессия применялась для прогнозирования временных рядов была Doan, Litterman and Sims (1984). Авторы применили фильтр Калмана для обновления вектора параметров, представляющих со- бой процесс AR(1) и оценили значения гиперпараметров, при которых их модель давала наилучший прогноз среди других VAR моделей.

По мере роста популярности байесовских методов стал расти и интерес к векторным авторегрессиям с переменными коэффициентами (TVP- VAR). Наиболее знаменитые работы в этой области — Cogley and Sargent (2002, 2005),

Primiceri (2005), Koop and Korobilis (2009). В первых трех статьях прогнозированию не уделялось особого значения, однако, в них были предложены основные принципы применения байесовского подхода к TVP-VAR моделям.

В статье Primiceri (2005) была подробно рассмотрена байесовская векторная авторегрессия с переменными параметрами (далее — TVP-BVAR). У данной модели существует две основные особенности: во-первых, пред- полагается, что все коэффициенты модели могут изменяться со временем, а во-вторых, ковариационная матрица остатков также зависит от времени. Такие предположения позволяют строить модель для длительных временных промежутков, подстраиваясь под изменения конъюнктуры и меняю- щиеся зависимости между рассматриваемыми переменными. Как уже было сказано выше, автор статьи описывает применение байесовского подхода к подобной модели, а также предлагает алгоритм, позволяющий получить выборку из апостериорного распределения параметров. Однако, через несколько лет после публикации данной статьи в алгоритме была обнаружена ошибка, которую авторы исправили относительно недавно, опубли- ковав статью Primiceri, Del Negro (2015). Вскоре после этого была опубликована статья Kruger (2015), где был представлен пакет bvarsv для языка программирования R, позволяющий оценивать модель TVP-BVAR с ис- пользованием актуального алгоритма.

Практика

Эконометристы должны уметь решать четыре логически отличающиеся задачи:
- описание данных;
- экономический прогноз;
- структурный вывод;
- анализ политики.

Логика данной работы состоит в том, чтоб пройти путь в прогнозировании данных с самого начала, с наиболее легко специфицированных моделей и одномерных методов и потом подключитьь уже многомерные методы с большим количеством параметров с той целью, чтобы пронаблюдать, насколько благотворно это скажется на метриках качества, графическом представлении данных и тд.
Начинаем с того, что таргетируем прогноз на среднее из тех значений, что мы уже наблюдали, если бы мы имели дело с выборкой, где все наблюдения являются независимыими и одинаково распределенными, это была бы одна из возможных BLUE оценок для целевой переменной по теореме Гаусса-Маркова.
Модификацией простой скользящей средней является взвешенная средняя, внутри которой наблюдениям придаются различные веса, в сумме дающие единицу, при этом обычно последним наблюдениям присваивается больший вес.

$$\hat{y}_t = \sum_{n=1}^{k} \omega_n y_{t+1-n}$$

А теперь посмотрим, что произойдёт, если вместо взвешивания последних
N значений ряда мы начнем взвешивать все доступные наблюдения, при этом экспоненциально уменьшая веса по мере углубления в исторические данные.
Вот так выглядит формула простого экспоненциального сглаживания:

$$\hat{y}_t = \alpha \cdot y_t + (1 - \alpha) \cdot \hat{y}_{t-1}$$

Двойное экспоненциальное сглаживание

До сих пор мы могли получить от наших методов в лучшем случае прогноз лишь на одну точку вперёд (и ещё красиво сгладить ряд), это здорово, но недостаточно, поэтому переходим к расширению экспоненциального сглаживания, которое позволит строить прогноз сразу на две точки вперед (и тоже красиво сглаживать ряд).

В этом нам поможет разбиение ряда на две составляющие — уровень (level, intercept)
ℓ и тренд b (trend, slope). Уровень, или ожидаемое значение ряда, мы предсказывали при помощи предыдущих методов, а теперь такое же экспоненциальное сглаживание применим к тренду, наивно или не очень полагая, что будущее направление изменения ряда зависит от взвешенных предыдущих изменений.

$$\ell_x = \alpha y_x + (1-\alpha)(\ell_{x-1} + b_{x-1})$$
$$b_x = \beta(\ell_x - \ell_{x-1}) + (1-\beta)b_{x-1}$$
$$\hat{y}_{x+1} = \ell_x + b_x$$

В результате получаем набор функций. Первая описывает уровень — он, как и прежде, зависит от текущего значения ряда, а второе слагаемое теперь разбивается на предыдущее значение уровня и тренда. Вторая отвечает за тренд — он зависит от изменения уровня на текущем шаге, и от предыдущего значения тренда. Здесь в роли веса в экспоненциальном сглаживании выступает коэффициент β. Наконец, итоговое предсказание представляет собой сумму модельных значений уровня и тренда.

Теперь настраивать придется уже два параметра —
α и β. Первый отвечает за сглаживание ряда вокруг тренда, второй — за сглаживание самого тренда. Чем выше значения, тем больший вес будет отдаваться последним наблюдениям и тем менее сглаженным окажется модельный ряд. Комбинации параметров могут выдавать достаточно причудливые результаты, особенно если задавать их руками. А о не ручном подборе параметров расскажу чуть ниже, сразу после тройного экспоненциального сглаживания.

Тройное экспоненциальное сглаживание Holt-Winters

Итак, успешно добрались до следующего варианта экспоненциального сглаживания, на сей раз тройного.

Идея этого метода заключается в добавлении еще одной, третьей, компоненты — сезонности. Соответственно, метод применим только в случае, если ряд этой сезонностью не обделён, что в нашем случае верно. Сезонная компонента в модели будет объяснять повторяющиеся колебания вокруг уровня и тренда, а характеризоваться она будет длиной сезона — периодом, после которого начинаются повторения колебаний. Для каждого наблюдения в сезоне формируется своя компонента, например, если длина сезона составляет 7 (например, недельная сезонность), то получим 7 сезонных компонент, по штуке на каждый из дней недели.

AR

Оценим векторную авторегрессионную модель с рлагами (Xt – вектор, содержащий n переменных из таблицы 1):

$$Xt = A0 + A1Xt\text{-}1 + A2Xt\text{-}2... + ApXt\text{-}p + \varepsilon t$$

Оценка авторегрессии повторяет алгоритм поиска сумму наименбших квадратов ошибок.

ARIMA

ARIMA использует три основных параметра (p, d, q), которые выражаются целыми числами. Потому модель также записывается как ARIMA(p, d, q). Вместе эти три параметра учитывают сезонность, тенденцию и шум в наборах данных:

- p – порядок авторегрессии (AR), который позволяет добавить предыдущие значения временного ряда. Этот параметр можно проиллюстрировать утверждением «завтра, вероятно, будет тепло, если в последние три дня было тепло».
- d – порядок интегрирования (I; т. е. порядок разностей исходного временного ряда). Он добавляет в модель понятия разности временных рядов (определяет количество прошлых временных точек, которые нужно вычесть из текущего значения). Этот параметр иллюстрирует такое утверждение: «завтра, вероятно, будет такая же температура, если разница в температуре за последние три дня была очень мала».

q – порядок скользящего среднего (MA), который позволяет установить погрешность модели как линейную комбинацию наблюдавшихся ранее значений ошибок.

Для отслеживания сезонности используется сезонная модель ARIMA – ARIMA(p,d,q)(P,D,Q)s. Здесь (p, d, q) – несезонные параметры, описанные выше, а (P, D, Q) следуют тем же определениям, но применяются к сезонной составляющей временного ряда. Параметр s определяет периодичность временного ряда (4 — квартальные периоды, 12 – годовые периоды и т.д.).

VAR

На первый взгляд, VAR - не более, чем обобщение одномерной авторегрессии на многомерный случай, и каждое уравнение в VAR - не более, чем обычная регрессия по методу наименьших квадратов одной переменной на запаздывающие значения себя и других переменных в VAR. Но этот вроде бы простой инструмент дал возможность систематически и внутренне согласованно уловить богатую динамику многомерных временных рядов, а статистический инструментарий, который сопутствует VAR, оказался удобным и, что очень важно, его было легко интерпретировать.

Выделяют три различных VAR-модели:
- приведенная форма VAR;
- рекурсивная VAR;

- структурная VAR.

Все три являются динамическими линейными моделями, которые связывают текущие и прошлые значения вектора Y_t n-мерного временного ряда. Приведенная форма и рекурсивные VAR - это статистические модели, которые не используют никакие экономические соображения за исключением выбора переменных. Эти VAR используются для описания данных и прогноза. Структурная VAR включает ограничения, полученные из макроэкономической теории, и эта VAR используется для структурного вывода и анализа политики.

Приведенная форма VAR выражает Y_t в виде распределенного лага прошлых значений плюс серийно некоррелированный член ошибки, то есть обобщает одномерную авторегрессию на случай векторов. Математически приведенная форма модели VAR - это система n уравнений, которые можно записать в матричной форме следующим образом:

$$Y_t = \alpha + A_1 Y_{t-1} + \ldots + A_p Y_{t-p} + \varepsilon_t,$$

Оценить параметры приведенной формы VAR легко. Каждое из уравнений содержит одни и те же регрессоры ($Y_{t-1},\ldots,Y_{t-p}$), и нет взаимных ограничений между уравнениями. Таким образом, эффективная оценка (метод максимального правдоподобия с полной информацией) упрощается до обычного МНК, примененного к каждому из уравнений. Матрицу ковариаций ошибок можно состоятельно оценить выборочной ковариационной матрицей полученных из МНК остатков.

Единственная тонкость - определить длину лага p, но это можно сделать, используя информационный критерий, такой как AIC или BIC.

На уровне матричных уравнений рекурсивная и структурная VAR выглядят одинаково. Эти две модели VAR учитывают в явном виде одновременные взаимодействия между элементами Y_t, что сводится к добавлению одновременного члена к правой части уравнения. Соответственно, рекурсивная и структурная VAR обе представляются в следующем общем виде:

$$Y_t = \beta + B_0 Y_t + B_1 Y_{t-1} + \ldots + B_p Y_{t-p} + \eta_t,$$

Наличие в уравнении матрицы B_0 означает возможность одновременного взаимодействия между n переменными; то есть B_0 позволяет сделать так, чтобы эти переменные, относящиеся к одному моменту времени, определялись совместно.

Рекурсивную VAR можно оценить двумя способами. Рекурсивная структура дает набор рекурсивных уравнений, которые можно оценить с помощью МНК. Эквивалентный способ оценивания заключается в том, что уравнения приведенной формы, рассматриваемые как система, умножаются слева на нижнюю треугольную матрицу.

Метод оценивания структурной VAR зависит от того, как именно идентифицирована B_0. Подход с частичной информацией влечет

использование методов оценивания для отдельного уравнения, таких как двух шаговый метод наименьших квадратов. Подход с полной информацией влечет использование методов оценивания для нескольких уравнений, таких как трех шаговый метод наименьших квадратов.
Необходимо помнить о множественности различных типов VAR. Приведенная форма VAR единственна. Данному порядку переменных в Y_t соответствует единственная рекурсивная VAR, но всего имеется n! таких порядков, т.е. n! различных рекурсивных VAR. Количество структурных VAR - то есть наборов предположений, которые идентифицируют одновременные взаимосвязи между переменными, - ограничено только изобретательностью исследователя.
Поскольку матрицы оцененных коэффициентов VAR затруднительно интерпретировать непосредственно, результаты оценивания VAR обычно представляют некоторыми функциями этих матриц. К таким статистикам разложения ошибки прогноза.
Разложения дисперсии ошибки прогноза вычисляются в основном для рекурсивных или структурных систем. Такое разложение дисперсии показывает, насколько ошибка в j-м уравнении важна для объяснения неожиданных изменений i-й переменной. Когда ошибки VAR некоррелированы по уравнениям, дисперсию ошибки прогноза на h периодов вперед можно записать как сумму компонентов, являющихся результатом каждой из этих ошибок.

BVAR

Для оценки параметров данной модели использовался байесовский подход, позволяющий осуществить сжатие избыточного количества неизвестных параметров при помощи введения априорной информации Степень информативности выбранных априорных распределений определяется при помощи максимизации функции маргинального правдоподобия (marginal likelihood).

Рассмотрим данную задачу подробнее. Для практического применения модель переписывается в векторной форме следующим образом:

$yt = xt\,\beta + \varepsilon t,$

где $yt = Xt$, $xt = In\,[1\ X't\text{-}1 \ldots X't\text{-}p]$, $\beta \equiv vec([A0,A1,\ldots,Ap]')$, $\varepsilon t \sim N(0,\Sigma)$.

Основным предположением об априорном распределении параметров модели является, то что поведение переменных схоже со случайным блужданием (приор Minnesota) то есть матожидание принимает значение ед иницы, если наблюдения между собой равны, и 0 иначе.

Следующие два приора вводятся при помощи конструирования дополнительных фиктивных переменных следующего вида.

1. Приор sum-of-coefficients используется для реализации предположения о наличии единичного корня в уравнениях модели:
2. Приор dummy-initial observation используется для реализации предположения о наличии коинтеграции:

Возникает вопрос, а как подобрать вид априорного поведения переменных, если имею дело с подобными данными в первый раз, не зная природы процесса, которые породил наблюдаемые данные и обладая достаточно маленьким количеством наблюдений ?

Для поиска параметров априорного распределения был применен эмпирический байесовский метод (empirical Bayesian method), позволяющий оценивать искомые параметры, в значительной степени опираясь на данные. Для оценки апостериорного распределения параметров модели использовался стандартный алгоритм Метрополиса - Гастингса (Metropolis-Hastings algorithm). Рассмотренная процедура позволяет автоматически выбирать степень информативности априорных распределений, увеличивая значимость априорной информации в случае недостатка данных и уменьшая ее значимость по мере возрастания объема накопленной выборки.

К сожалению, знать априорные распределения параметров и функцию правдоподобия недостаточно для того, чтобы получиться апостериорное

распределение. Как видно из формулы ниже, это позволит нам получить только функцию, пропорциональную апостериорному распределению. В теории не составляет труда найти значение недостающей константы по формуле полной вероятности $P(Y_t) = \int P(Y_T|\Theta)\, P(\Theta)d\Theta$, но на практи- ке очень часто посчитать такой многомерный интеграл не представляется возможным. Существует два способа решения этой проблемы: использо- вание сопряженных распределений и применение алгоритмов MCMC для получения выборки из апостериорного распределения.

Сопряжённое априорное распределение подбирается таким образом, чтобы при вычислении его произведения с функцией правдоподобия, получалось апостериорное распределение того же семейства. В таком случае апостериорное распределение выписывается в явном виде.

Второй способ получить апостериорное распределение — использование методов MCMC (Markov Chain Monte Carlo). Данные методы помогают получить выборку из апостериорного распределения, которая в дальнейшем может быть использована для получения интересующих нас характеристик

оцениваемых параметров. Существует три основных вида алгоритмов, позволяющих получить выборку из апостериорного распределения — сэмплирование по Гиббсу, алгоритм Метрополиса-Гастингса и Гибридный метод Монте-Карло. Данные методы основываются на предположении, что все возможные значения апостериорного распределения представляют собой вершины Марковской цепи — направленного графа с некоторыми фиксированными вероятностями перехода из одной вершины в другую. Стационарное состояние такой цепи и является плотностью искомого апостериорного распределения. Алгоритм инициируется в какой-то точке распределения и дальше, используя определенные критерии, начинает двигаться из одной точки в другую. При достаточно большом количестве шагов полученная выборка будет достаточно близка к выборке из апостериорного распределения.

Анализ функции импульсных откликов

Чтобы проверить адекватность взаимосвязей между переменными в модели, были проанализировны функции импульсных откликов. Модель не предназначена для структурного анализа, поэтому использовались обобщенные отклики (Pesaran and Shin, (1998) вместо какой-либо схемы идентификации структурных шоков. Соответственно, полученные отклики не имеют экономической интерпретации, однако позволяют сделать вывод об адекватности параметризации модели (в частности, можно удостовериться в отсутствии осцилляции и статистической значимости взаимосвязей).

Заключение

На реальных экономических данных были протестированыметоды скользящего среднего, авторегрессии, авторегрессии с интегрированным скользящим средним, векторная авторегрессия и реализация байесовской векторной авторегрессии с априорными распределениями экзогенных переменных, при этом результаты можно охарактеризовать как неоднозначные, потому что байесовская реализация с тремя разными приорами проиграла обычной векторной авторегрессии по причине относительно малого количества наблюдений, хорошо показали себя одномерные методы прогнозирования, я делаю вывод, что учет взавимосвязи переменных в многомерном временном ряде важен, но необходима одновременная работа с несколькими моделями, потому что при разном характере наблюдений многомерные методы ведут себя неоднозначно, а более действенными зачастую оказываются более простые методы.

Список литературы

- Banbura M., Giannone D., Reichlin L. Large Bayesian VARs // J. of Applied Econometrics. 2010. Vol. 25 (1). P. 71–92.
- Box, G. E. P., & Jenkins, G. M. (1970). Time series analysis: forecasting and control, 1976. ISBN: 0-8162-1104-3.y.
- De Mol C., Giannone D., Reichlin L. Forecasting using a large number of predictors: Is Bayesian shrinkage a valid alternative to principal components? // J. of Econometrics. 2008. No 146 (2). P. 318–328.
- Del Negro, M., & Primiceri, G. E. (2015). Time varying structural vector autoregressions and monetary policy: a corrigendum. The review of economic studies, 82(4), 1342-1345.
- Demeshev, B., & Malakhovskaya, O. (2015). Forecasting Russian macroeconomic indicators with BVAR.
- Deutsch, M., Granger, C. W., & Teräsvirta, T. (1994). The combination of forecasts using changing weights. International Journшдal of Forecasting, 10(1), 47-57.
- Doan, T., Litterman, R., & Sims, C. (1984). Forecasting and conditional projection using realistic prior distributions. Econometric reviews, 3(1), 1-100.
- Giannone D., Lenza M., Primiceri G.E. Prior Selection for Vector Autoregressions // ECB Working Paper Series. 2012. No 1494.
- Koop, G., & Korobilis, D. (2010). Bayesian multivariate time series methods for empirical macroeconomics. Foundations and Trends® in Econometrics, 3(4), 267-358.
- Litterman, R. (1986). Forecasting with Bayesian vector autoregressions—Five years of experience: Robert B. Litterman, Journal of Business and Economic Statistics 4 (1986) 25–38. International Journal of Forecasting, 2(4), 497-498.
- Pesaran H., Shin Y. Generalized impulse response analysis in linear multivariate models // Economic Letters. 1998. No 58(1). P. 17–29.

Приложение №1: основной код с комментариями, выполнено в Питоне

Прежде чем начать прогнозирование, загрузим все необходимые пакеты, главные из них: math и numpy для вычисления метрик качества и работы с формулами, pd для работы с массивами датасетов, plt для построения графиков, statsmodels для использования основных статистических моделей.

In [2]:

```
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy
import scipy.stats
from __future__ import division, print_function
import statsmodels as sm

from collections import defaultdict
from math import log
from numpy import hstack, vstack
from numpy.linalg import inv, svd

from statsmodels.compat.python import range, string_types, iteritems
from statsmodels.iolib.summary import Summary
from statsmodels.iolib.table import SimpleTable
from statsmodels.tools.decorators import cache_readonly
from statsmodels.tools.sm_exceptions import HypothesisTestWarning
from statsmodels.tools.tools import chain_dot
from statsmodels.tsa.tsatools import duplication_matrix, vec

import statsmodels.tsa.base.tsa_model as tsbase
import statsmodels.tsa.vector_ar.irf as irf
import statsmodels.tsa.vector_ar.plotting as plot
from statsmodels.tsa.vector_ar.var_model import forecast, VAR, ma_rep, _compute_a

mdata = sm.datasets.macrodata.load().data
mdata = mdata[['realgdp','realcons','realinv']]
names = mdata.dtype.names
data = mdata.view((float,3))
data = np.diff(np.log(data), axis=0)
```

In [3]:

```
newdata = pd.DataFrame(mdata)
newdata[:8]
```

Out[3]:

	realgdp	realcons	realinv
0	2710.349	1707.4	286.898
1	2778.801	1733.7	310.859
2	2775.488	1751.8	289.226
3	2785.204	1753.7	299.356
4	2847.699	1770.5	331.722
5	2834.390	1792.9	298.152
6	2839.022	1785.8	296.375
7	2802.616	1788.2	259.764

Мы загрузили датасет, который зашит в статистическом пакете python-statsmodels.

Как мы видим, наш временной ряд включает в себя динамику трех переменных - реального ВВП, реального потребления и реальных инвестиций, данные собраны годовые и более чем за 200 лет.

In [4]:

```
len(newdata)
```

Out[4]:

```
203
```

In [5]:

```
date = pd.date_range('1814-01-01', end = '2016-01-01', freq = 'AS')
newdata.index = date
```

In [6]:

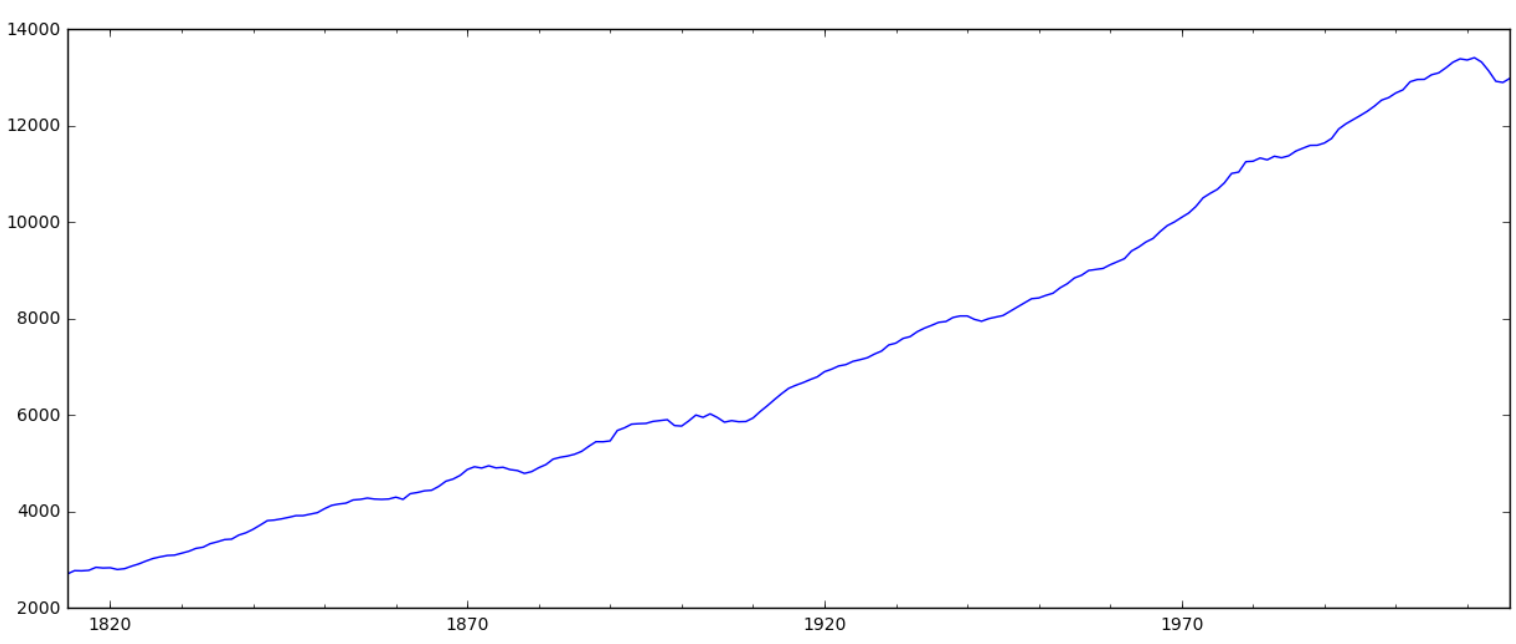
```
newdata[:8]
```

Out[6]:

	realgdp	realcons	realinv
1814-01-01	2710.349	1707.4	286.898
1815-01-01	2778.801	1733.7	310.859
1816-01-01	2775.488	1751.8	289.226
1817-01-01	2785.204	1753.7	299.356
1818-01-01	2847.699	1770.5	331.722
1819-01-01	2834.390	1792.9	298.152
1820-01-01	2839.022	1785.8	296.375
1821-01-01	2802.616	1788.2	259.764

In [7]:

```
real_GDP = newdata['realgdp']
real_GDP.plot(figsize=(15, 6))
plt.show()
```



In [8]:

```
import sys
import warnings
warnings.filterwarnings('ignore')

from sklearn.metrics import mean_absolute_error, mean_squared_error

import statsmodels.formula.api as smf
import statsmodels.tsa.api as smt
import statsmodels.api as sm
import scipy.stats as scs
from scipy.optimize import minimize

import matplotlib.pyplot as plt
from tqdm import tqdm
```

Начнем моделирование с предположения — "завтра будет, как вчера", но вместо модели вида y(t)=y(t-1) будем считать, что будущее значение переменной зависит от среднего n её предыдущих значений, а значит, воспользуемся скользящей средней.

In [9]:

```
def moving_average(series, n):
    return np.average(series[-n:])

moving_average(real_GDP, 24)
# прогноз по последним 24 наблюдениям
```

Out[9]:

```
12859.946249999999
```

К сожалению, такой прогноз долгосрочным сделать не удастся — для получения предсказания на шаг вперед предыдущее значение должно быть фактически наблюдаемой величиной. Зато у скользящей средней есть другое применение — сглаживание исходного ряда для выявления трендов. В пакете pandas есть готовая реализация — DataFrame.rolling(window).mean(). Чем больше зададим ширину интервала — тем более сглаженным окажется тренд. В случае, если данные сильно зашумлены, что особенно часто встречается, например, в финансовых показателях, такая процедура может помочь с определением общих взавимосвязей.

In [10]:

```
def plotMovingAverage(series, n):

    rolling_mean = series.rolling(window=n).mean()


    plt.figure(figsize=(15,5))
    plt.title("Moving average\n window size = {}".format(n))
    plt.plot(rolling_mean, "g", label="Rolling mean trend")


    plt.plot(real_GDP[n:], label="Actual values")
    plt.legend(loc="upper left")
    plt.grid(True)
```

Для нашего ряда тренды и так вполне очевидны, причем в агрегированном смысле тяжело задать длину интервала, который бы представлял собой цикл активности, или экономический цикл, если бы данные были по какой-то определенной стране в определенный период, мы могли бы попробовать поставить активность в зависимость от срока нахождения на посту президента или, к примеру, пятилетних планов по развитию определенной отрасли или даже всех отраслей сразу.

Модификацией простой скользящей средней является взвешенная средняя, внутри которой наблюдениям придаются различные веса, в сумме дающие единицу, при этом обычно последним наблюдениям присваивается больший вес.

In [12]:

```
def weighted_average(series, weights):
    result = 0.0
    weights.reverse()
    for n in range(len(weights)):
        result += series[-n-1] * weights[n]
    return result

weighted_average(real_GDP, [0.6, 0.2, 0.1, 0.07, 0.03])
```

Out[12]:

```
13208.500510000002
```

Экспоненциальное сглаживание, модель Хольта-Винтерса

Простое экспоненциальное сглаживание

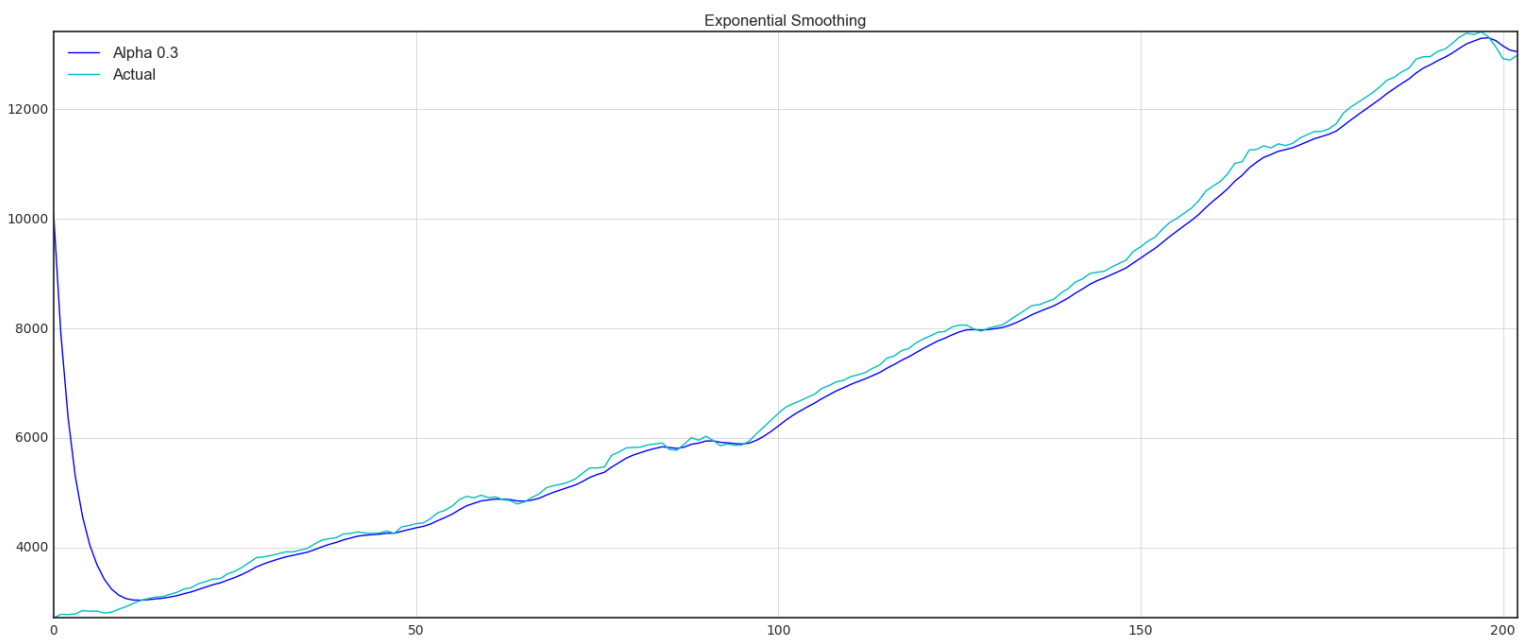
А теперь посмотрим, что произойдёт, если вместо взвешивания последних n значений ряда мы начнем взвешивать все доступные наблюдения, при этом экспоненциально уменьшая веса по мере углубления в исторические данные. В этом нам поможет формула простого экспоненциального сглаживания Экспоненциальность скрывается в рекурсивности функции — каждый раз мы умножаем 1- α на предыдущее модельное значение, которое, в свою очередь, также содержало в себе 1- α , и так до самого начала.

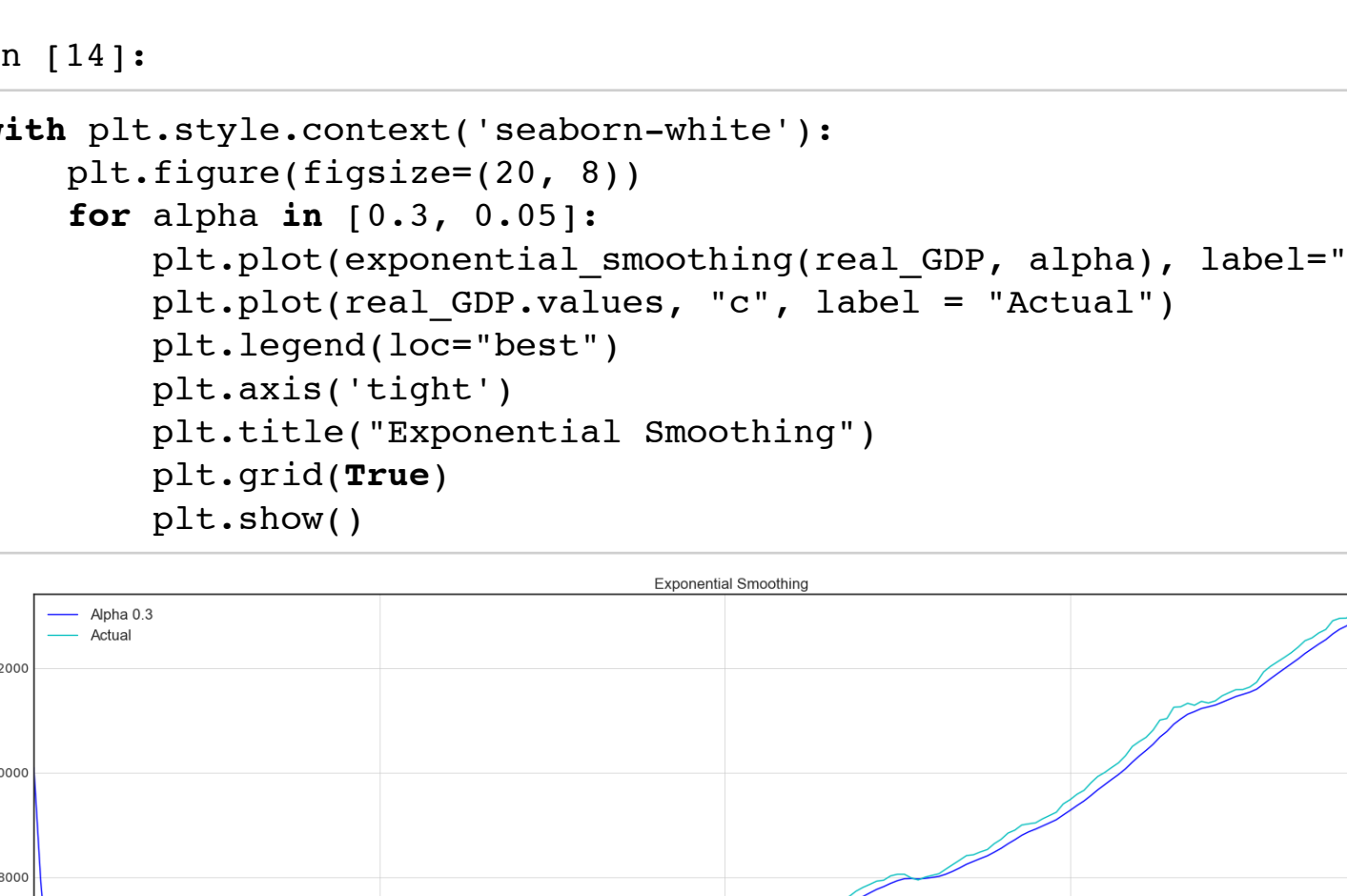
In [13]:

```
def exponential_smoothing(series, alpha):
    result = [series[0]] # first value is same as series
    for n in range(1, len(series)):
        result.append(alpha * series[n] + (1 - alpha) * result[n-1])
    return result
```

In [14]:

```
with plt.style.context('seaborn-white'):
    plt.figure(figsize=(20, 8))
    for alpha in [0.3, 0.05]:
        plt.plot(exponential_smoothing(real_GDP, alpha), label="Alpha {}".format(
        plt.plot(real_GDP.values, "c", label = "Actual")
        plt.legend(loc="best")
        plt.axis('tight')
        plt.title("Exponential Smoothing")
        plt.grid(True)
        plt.show()
```





Двойное экспоненциальное сглаживание

До сих пор мы могли получить от наших методов в лучшем случае прогноз лишь на одну точку вперёд, поэтому переходим к расширению экспоненциального сглаживания, которое позволит строить прогноз на значения.

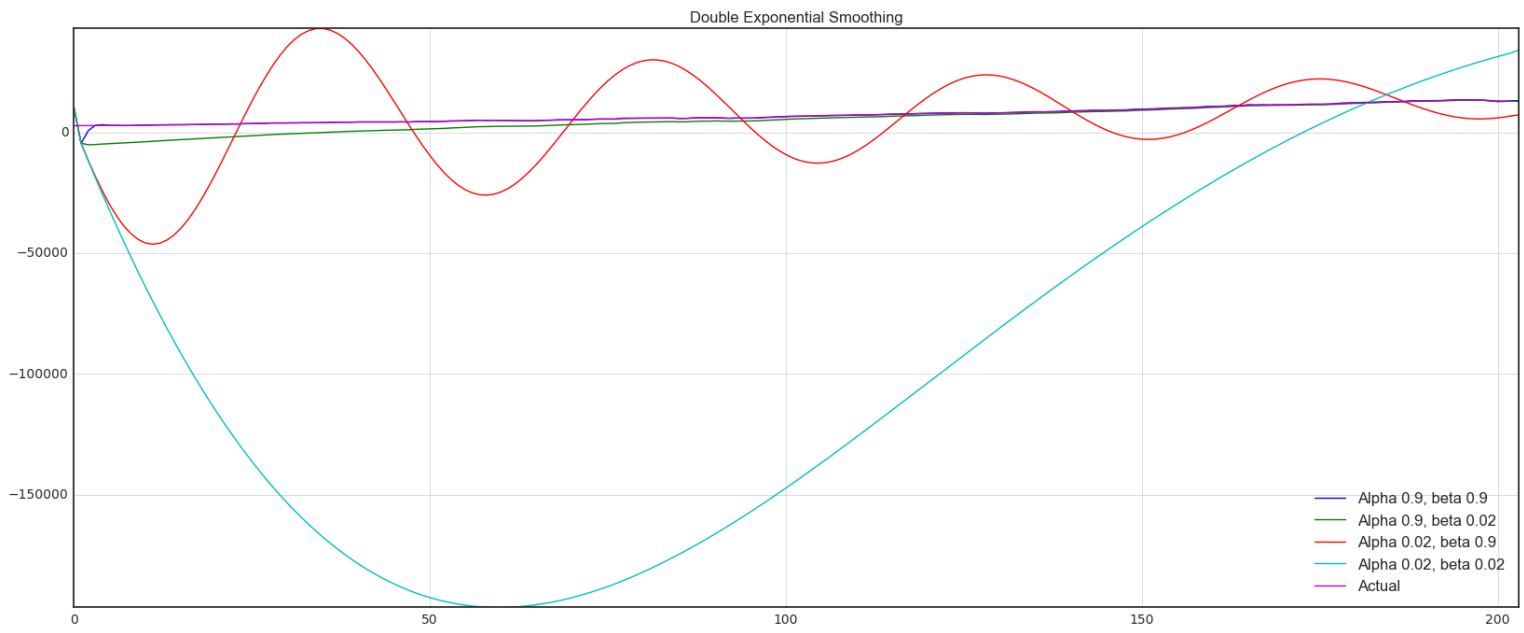
В этом нам поможет разбиение ряда на две составляющие — уровень (level, intercept) ℓ и тренд b (trend, slope). Уровень, или ожидаемое значение ряда, мы предсказывали при помощи предыдущих методов, а теперь такое же экспоненциальное сглаживание применим к тренду, полагая, что будущее направление изменения ряда зависит от взвешенных предыдущих изменений.

In [15]:

```
def double_exponential_smoothing(series, alpha, beta):
    result = [series[0]]
    for n in range(1, len(series)+1):
        if n == 1:
            level, trend = series[0], series[1] - series[0]
        if n >= len(series): # прогнозируем
            value = result[-1]
        else:
            value = series[n]
        last_level, level = level, alpha*value + (1-alpha)*(level+trend)
        trend = beta*(level-last_level) + (1-beta)*trend
        result.append(level+trend)
    return result
```

In [16]:

```
with plt.style.context('seaborn-white'):
    plt.figure(figsize=(20, 8))
    for alpha in [0.9, 0.02]:
        for beta in [0.9, 0.02]:
            plt.plot(double_exponential_smoothing(real_GDP, alpha, beta), label="
    plt.plot(real_GDP.values, label = "Actual")
    plt.legend(loc="best")
    plt.axis('tight')
    plt.title("Double Exponential Smoothing")
    plt.grid(True)
    plt.show()
```



Теперь настраивать необходимо уже два параметра — α и β. Первый отвечает за сглаживание ряда вокруг тренда, второй — за сглаживание самого тренда. Чем выше значения, тем больший вес

вокруг тренда, второй — за сглаживание самого тренда. Чем выше значения, тем больший вес будет отдаваться последним наблюдениям и тем менее сглаженным окажется модельный ряд. Комбинации параметров могут выдавать достаточно причудливые результаты, особенно если задавать их руками. А о не ручном подборе параметров расскажу чуть ниже, сразу после тройного экспоненциального сглаживания.

Тройное экспоненциальное сглаживание Holt-Winters

Идея этого метода заключается в добавлении еще одной, третьей, компоненты — сезонности. Соответственно, метод применим только в случае, если ряд этой сезонностью не обделён, что в нашем случае верно. Сезонная компонента в модели будет объяснять повторяющиеся колебания вокруг уровня и тренда, а характеризоваться она будет длиной сезона — периодом, после которого начинаются повторения колебаний. Для каждого наблюдения в сезоне формируется своя компонента, например, если длина сезона составляет 7 (например, недельная сезонность), то получим 7 сезонных компонент, по штуке на каждый из дней недели.

Получаем новую систему:

Уровень теперь зависит от текущего значения ряда за вычетом соответствующей сезонной компоненты, тренд остаётся без изменений, а сезонная компонента зависит от текущего значения ряда за вычетом уровня и от предыдущего значения компоненты. При этом компоненты сглаживаются через все доступные сезоны, например, если это компонента, отвечающая за понедельник, то и усредняться она будет только с другими понедельниками. Подробнее про работу усреднении и оценку начальных значении тренда и сезонных компонент можно почитать здесь. Теперь, имея сезонную компоненту, мы можем предсказывать уже не на один, и даже не на два, а на произвольные m шагов вперёд, что не может не радовать.

In [17]:

```
class HoltWinters:

    def __init__(self, series, slen, alpha, beta, gamma, n_preds, scaling_factor=
        self.series = series
        self.slen = slen
        self.alpha = alpha
        self.beta = beta
        self.gamma = gamma
        self.n_preds = n_preds
        self.scaling_factor = scaling_factor

    def initial_trend(self):
        sum = 0.0
        for i in range(self.slen):
            sum += float(self.series[i+self.slen] - self.series[i]) / self.slen
        return sum / self.slen

    def initial_seasonal_components(self):
        seasonals = {}
        season_averages = []
        n_seasons = int(len(self.series)/self.slen)
        # вычисляем сезонные средние
```

```
        for j in range(n_seasons):
            season_averages.append(sum(self.series[self.slen*j:self.slen*j+self.s
        # вычисляем начальные значения
        for i in range(self.slen):
            sum_of_vals_over_avg = 0.0
            for j in range(n_seasons):
                sum_of_vals_over_avg += self.series[self.slen*j+i]-season_average
            seasonals[i] = sum_of_vals_over_avg/n_seasons
        return seasonals

    def triple_exponential_smoothing(self):
        self.result = []
        self.Smooth = []
        self.Season = []
        self.Trend = []
        self.PredictedDeviation = []
        self.UpperBond = []
        self.LowerBond = []

        seasonals = self.initial_seasonal_components()

        for i in range(len(self.series)+self.n_preds):
            if i == 0: # инициализируем значения компонент
                smooth = self.series[0]
                trend = self.initial_trend()
                self.result.append(self.series[0])
                self.Smooth.append(smooth)
                self.Trend.append(trend)
                self.Season.append(seasonals[i%self.slen])
                self.PredictedDeviation.append(0)

                self.UpperBond.append(self.result[0] +
                                      self.scaling_factor *
                                      self.PredictedDeviation[0])

                self.LowerBond.append(self.result[0] -
                                      self.scaling_factor *
                                      self.PredictedDeviation[0])

                continue
            if i >= len(self.series): # прогнозируем
                m = i - len(self.series) + 1
                self.result.append((smooth + m*trend) + seasonals[i%self.slen])

                # во время прогноза с каждым шагом увеличиваем неопределенность
                self.PredictedDeviation.append(self.PredictedDeviation[-1]*1.01)

            else:
                val = self.series[i]
                last_smooth, smooth = smooth, self.alpha*(val-seasonals[i%self.sl
                trend = self.beta * (smooth-last_smooth) + (1-self.beta)*trend
                seasonals[i%self.slen] = self.gamma*(val-smooth) + (1-self.gamma)
                self.result.append(smooth+trend+seasonals[i%self.slen])

                # Отклонение рассчитывается в соответствии с алгоритмом Брутлага
                self.PredictedDeviation.append(self.gamma * np.abs(self.series[i]
                                               + (1-self.gamma)*self.PredictedDev
```

```
            self.UpperBond.append(self.result[-1] +
                                  self.scaling_factor *
                                  self.PredictedDeviation[-1])

            self.LowerBond.append(self.result[-1] -
                                  self.scaling_factor *
                                  self.PredictedDeviation[-1])

            self.Smooth.append(smooth)
            self.Trend.append(trend)
            self.Season.append(seasonals[i % self.slen])
```

Кросс-валидация на временных рядах, подбор параметров

Перед тем, как построить модель, поговорим, наконец, о не ручной оценке параметров для моделей.

Ничего необычного здесь нет, по-прежнему сначала необходимо выбрать подходящую для данной задачи функцию потерь: RMSE, MAE, MAPE и др., которая будет следить за качеством подгонки модели под исходные данные. Затем будем оценивать на кросс-валидации значение функции потерь при данных параметрах модели, искать градиент, менять в соответствии с ним параметры и бодро опускаться в сторону глобального минимума ошибки.

Небольшая загвоздка возникает только в кросс-валидации. Проблема состоит в том, что временной ряд имеет, как ни парадоксально, временную структуру, и случайно перемешивать в фолдах значения всего ряда без сохранения этой структуры нельзя, иначе в процессе потеряются все взаимосвязи наблюдений друг с другом. Поэтому придется использовать чуть более хитрый способ для оптимизации параметров, официального названия которому я так и не нашел, но на сайте CrossValidated, где можно найти ответы на всё, кроме главного вопроса Жизни, Вселенной и Всего Остального, предлагают название "cross-validation on a rolling basis", что не дословно можно перевести как кросс-валидация на скользящем окне.

Суть достаточно проста — начинаем обучать модель на небольшом отрезке временного ряда, от начала до некоторого t , делаем прогноз на $t+ n$ шагов вперед и считаем ошибку. Далее расширяем обучающую выборку до $t+ n$ значения и прогнозируем с $t+ n$ до $t+ 2 n$, так продолжаем двигать тестовый отрезок ряда до тех пор, пока не упрёмся в последнее доступное наблюдение. В итоге получим столько фолдов, сколько n уместится в промежуток между изначальным обучающим отрезком и всей длиной ряда.

In [18]:

```
from sklearn.model_selection import TimeSeriesSplit

def timeseriesCVscore(x):
    # вектор ошибок
    errors = []

    values = data.values
    alpha, beta, gamma = x

    # задаём число фолдов для кросс-валидации
    tscv = TimeSeriesSplit(n_splits=3)

    # идем по фолдам, на каждом обучаем модель, строим прогноз на отложенной выбо
    for train, test in tscv.split(values):

        #model = HoltWinters(series=values[train], slen = 24*7, alpha=alpha, beta
        model = HoltWinters(series=values[train], slen = 1, alpha=alpha, beta=bet
        model.triple_exponential_smoothing()

        predictions = model.result[-len(test):]
        actual = values[test]
        error = mean_squared_error(predictions, actual)
        errors.append(error)

    # Возвращаем средний квадрат ошибки по вектору ошибок
    return np.mean(np.array(errors))
```

В модели Хольта-Винтерса, как и в остальных моделях экспоненциального сглаживания, есть ограничение на величину сглаживающих параметров — каждый из них может принимать значения от 0 до 1, поэтому для минимизации функции потерь нужно выбирать алгоритм, поддерживающий ограничения на параметры, в данном случае — Truncated Newton conjugate gradient.

In [19]:

```
%%time
data = real_GDP[:-50] # отложим часть данных для тестирования
from scipy.optimize import minimize
# инициализируем значения параметров

x = [0, 0, 0]

# Минимизируем функцию потерь с ограничениями на параметры
opt = minimize(timeseriesCVscore, x0=x, method="TNC", bounds = ((0, 1), (0, 1), (

# Из оптимизатора берем оптимальное значение параметров
alpha_final, beta_final, gamma_final = opt.x
print(alpha_final, beta_final, gamma_final)
```

```
0.330847303068 0.110770471072 0.148634920095
CPU times: user 955 ms, sys: 14.7 ms, total: 970 ms
Wall time: 963 ms
```

Передадим полученные оптимальные значения коэффициентов α , β и γ и построим прогноз на 5

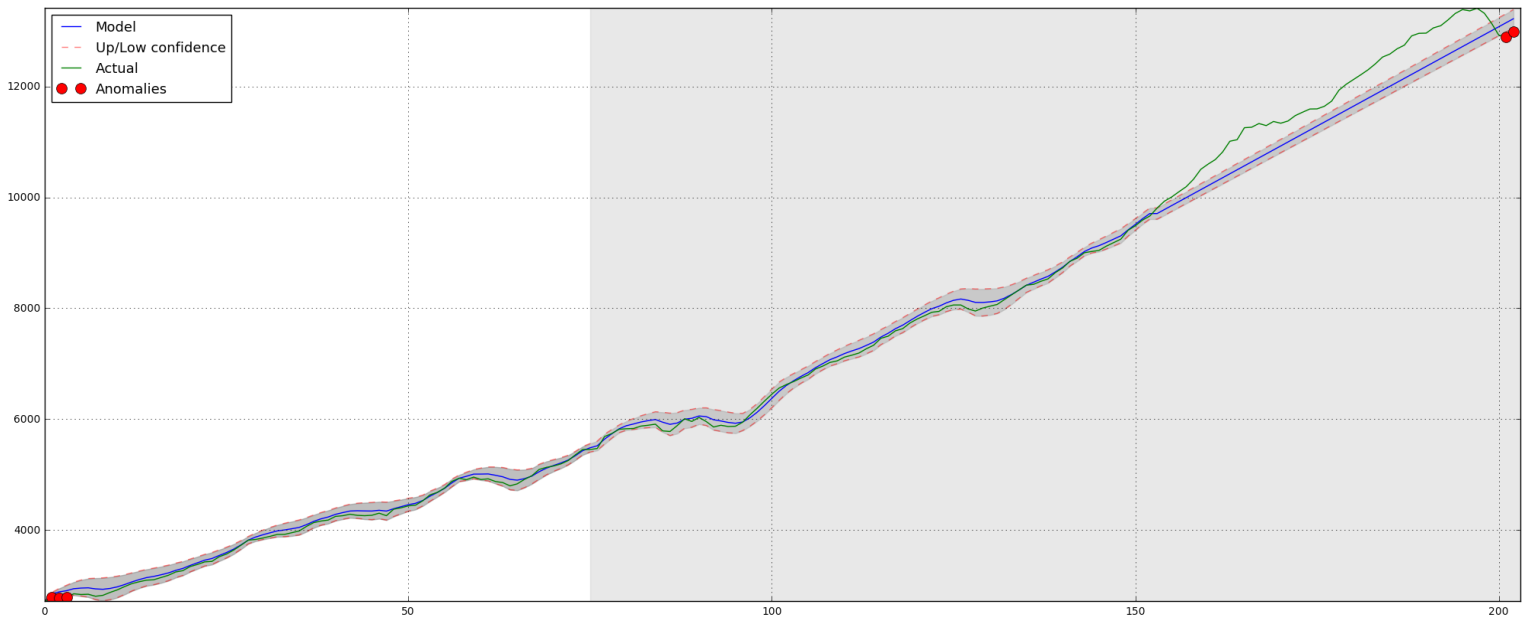
дней вперёд (128 часов)

In [20]:

```
# Передаем оптимальные значения модели
data = real_GDP
model = HoltWinters(data[:-50], slen = 1, alpha = alpha_final, beta = beta_final,
model.triple_exponential_smoothing()
```

In [21]:

```
def plotHoltWinters():
    Anomalies = np.array([np.NaN]*len(data))
    Anomalies[data.values<model.LowerBond] = data.values[data.values<model.LowerB
    plt.figure(figsize=(25, 10))
    plt.plot(model.result, label = "Model")
    plt.plot(model.UpperBond, "r--", alpha=0.5, label = "Up/Low confidence")
    plt.plot(model.LowerBond, "r--", alpha=0.5)
    plt.fill_between(x=range(0,len(model.result)), y1=model.UpperBond, y2=model.L
    plt.plot(data.values, label = "Actual")
    plt.plot(Anomalies, "o", markersize=10, label = "Anomalies")
    plt.axvspan(len(data)-128, len(data), alpha=0.5, color='lightgrey')
    plt.grid(True)
    plt.axis('tight')
    plt.legend(loc="best", fontsize=13);

plotHoltWinters()
plt.show()
```



Судя по графику, модель неплохо описала исходный временной ряд. Если посмотреть на смоделированное отклонение, хорошо видно, что модель достаточно резко регирует на значительные изменения в структуре ряда, но при этом быстро возвращает дисперсию к обычным значениям, "забывая" прошлое. Такая особенность позволяет неплохо и без значительных затрат на подготовку-обучение модели настроить систему по детектированию аномалий даже в достаточно шумных рядах. Видно, что, научившись на прошлых наблюдениях, модель скопировала тренд, и прогнозы легли в рамках данного тренда, хотя и произошло упрощение зависимости и это отразилось в простой прямой на отрезке прогнозирования.

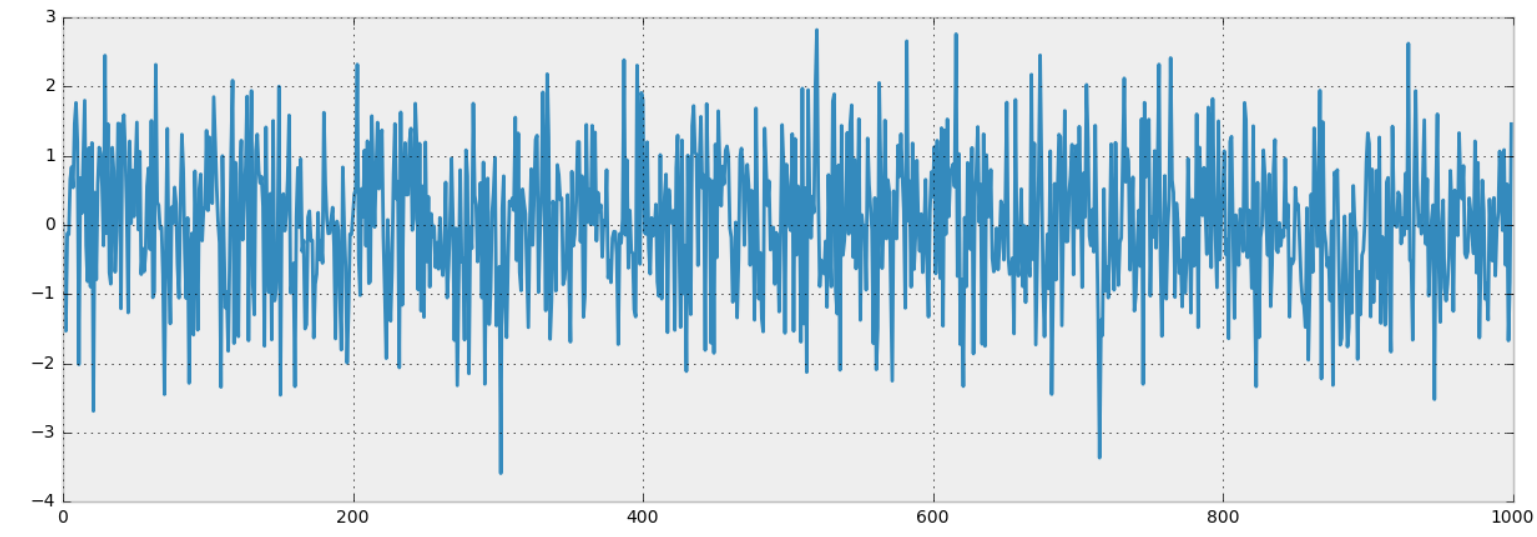
Эконометрический подход

Стационарность, единичные корни

Перед тем, как перейти к моделированию, стоит сказать о таком важном свойстве временного ряда, как стационарность. Под стационарностью понимают свойство процесса не менять своих статистических характеристик с течением времени, а именно постоянство матожидания, постоянство дисперсии (она же гомоскедастичность) и независимость ковариационной функции от времени (должна зависеть только от расстояния между наблюдениями).

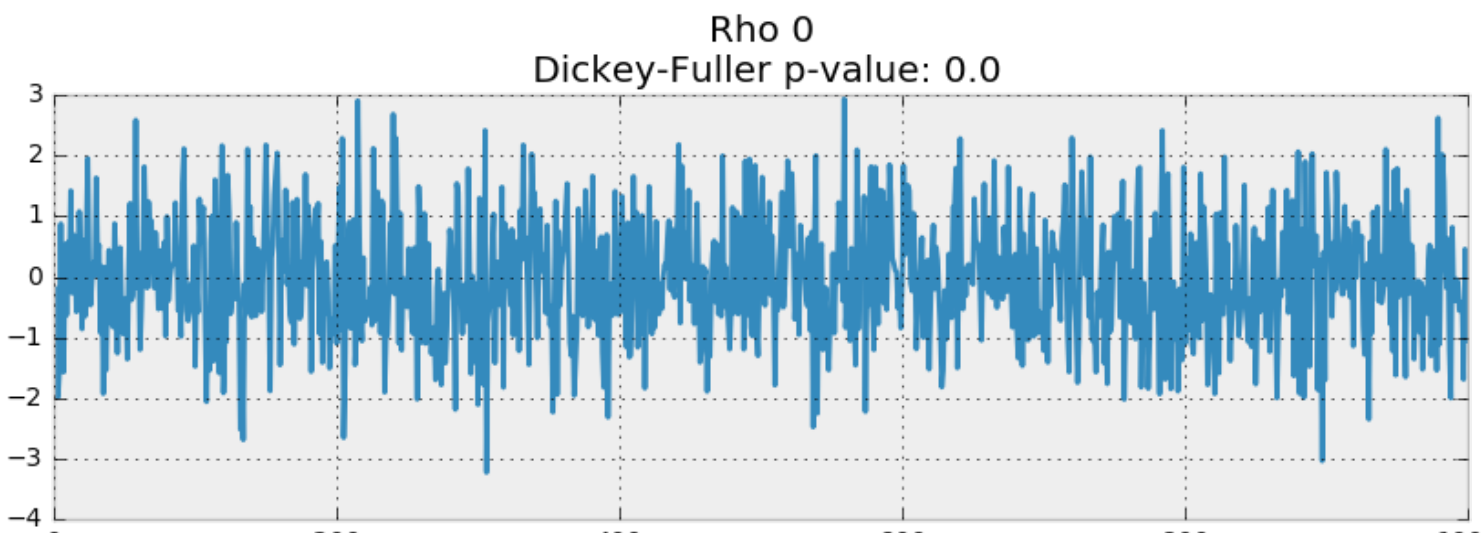
In [22]:

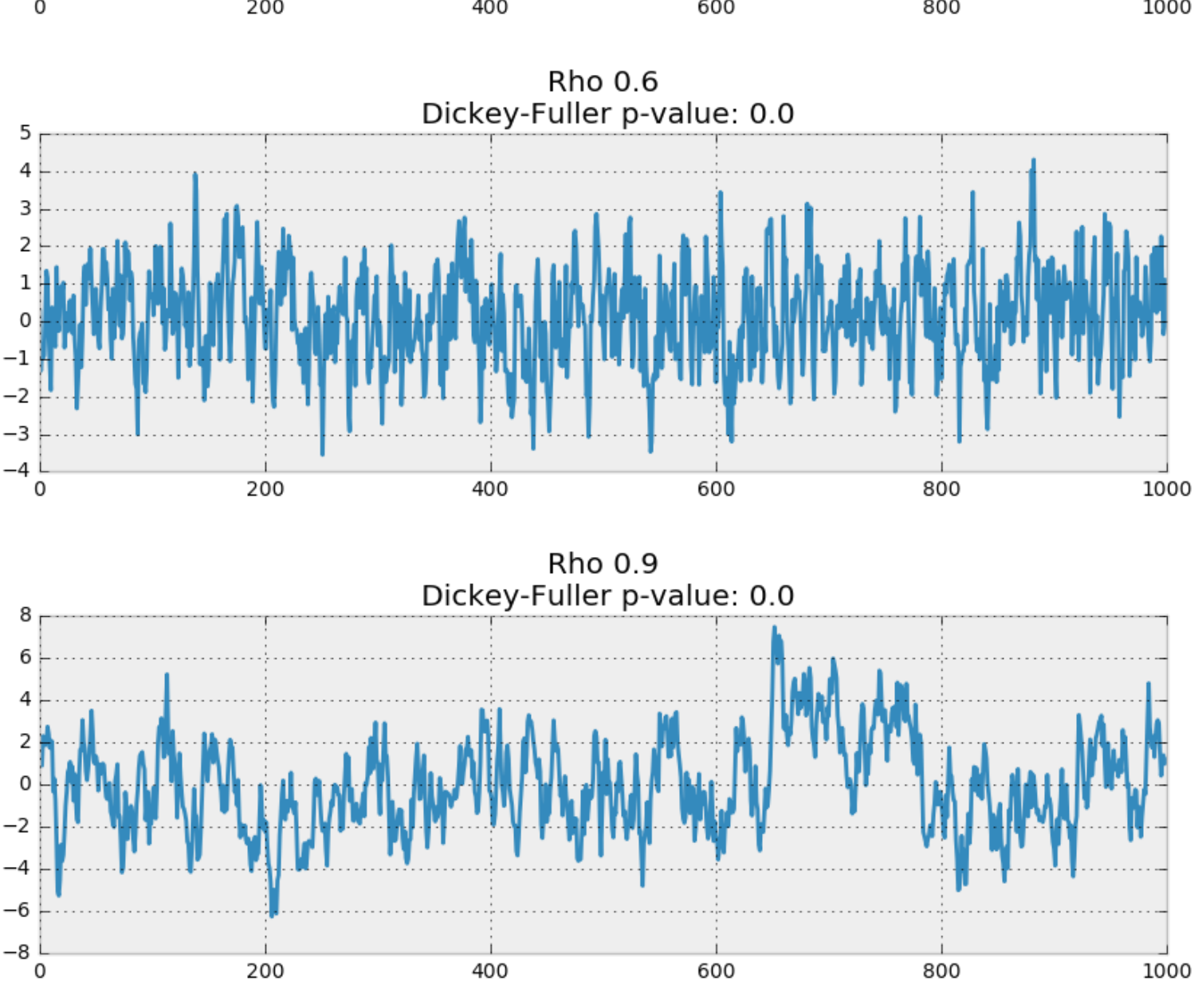
```
white_noise = np.random.normal(size=1000)
with plt.style.context('bmh'):
    plt.figure(figsize=(15, 5))
    plt.plot(white_noise)
    plt.show()
```

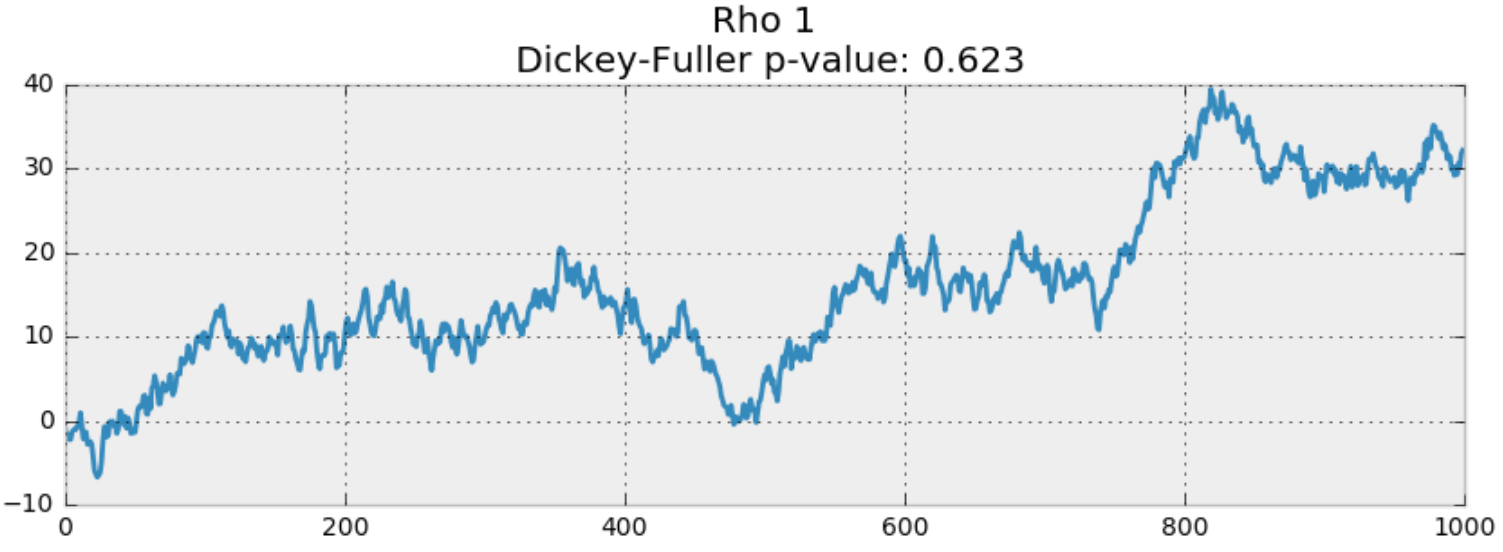


In [23]:

```
def plotProcess(n_samples=1000, rho=0):
    x = w = np.random.normal(size=n_samples)
    for t in range(n_samples):
        x[t] = rho * x[t-1] + w[t]

    with plt.style.context('bmh'):
        plt.figure(figsize=(10, 3))
        plt.plot(x)
        plt.title("Rho {}\n Dickey-Fuller p-value: {}".format(rho, round(sm.tsa.s

for rho in [0, 0.6, 0.9, 1]:
    plotProcess(rho=rho)
    plt.show()
```







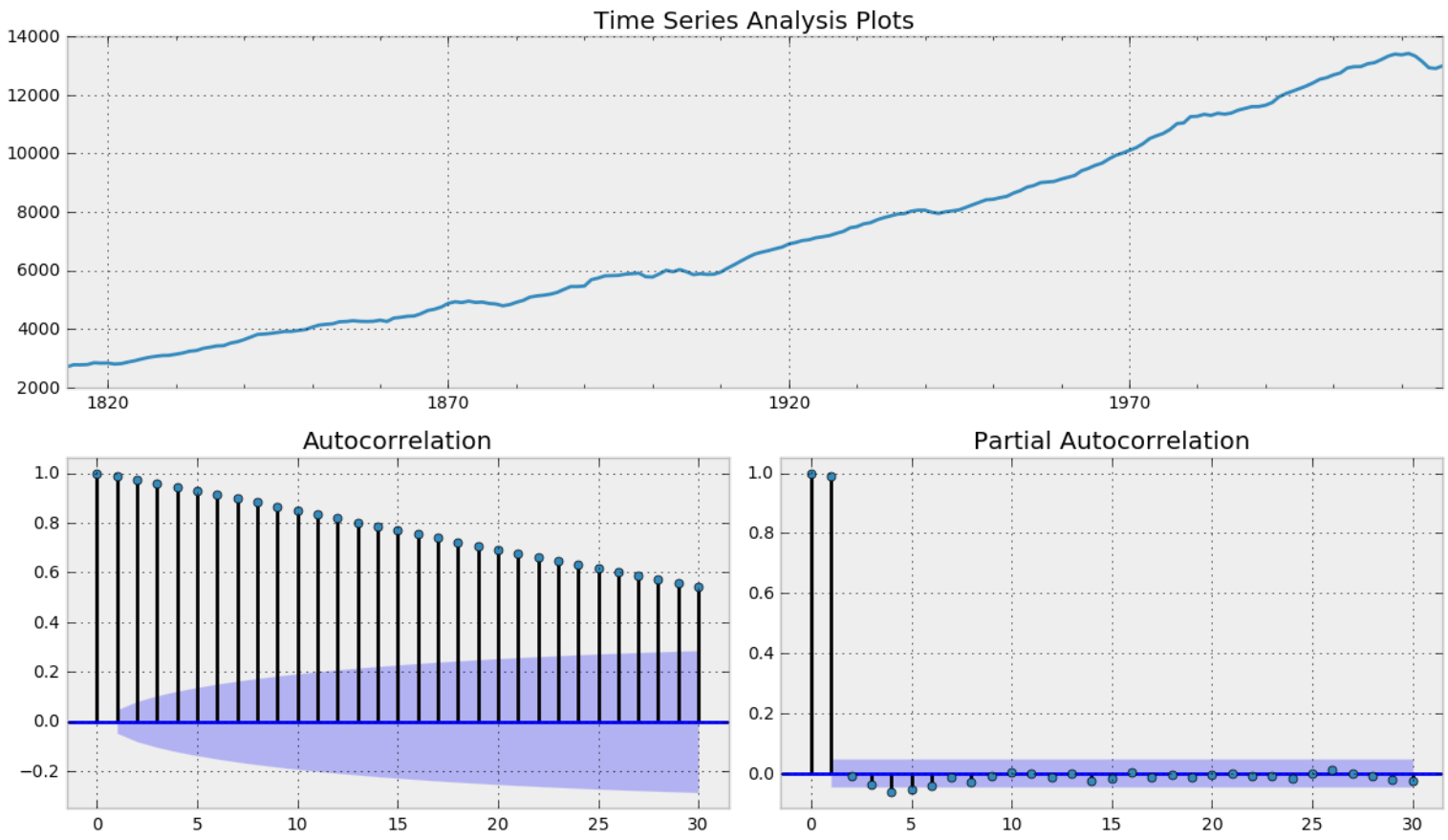
SARIMA

SARIMA является разновидностью все той же ARIMA, только в ней снова реализуется попытка поймать сезонную компоненту, однако в нашем случае это работает не так эффективно по той причине, что данные по временном ряду выбраны ежегодные, и есть смысл думать об отсутствии сезонности.

In [24]:

```
def tsplot(y, lags=None, figsize=(12, 7), style='bmh'):
    if not isinstance(y, pd.Series):
        y = pd.Series(y)
    with plt.style.context(style):
        fig = plt.figure(figsize=figsize)
        layout = (2, 2)
        ts_ax = plt.subplot2grid(layout, (0, 0), colspan=2)
        acf_ax = plt.subplot2grid(layout, (1, 0))
        pacf_ax = plt.subplot2grid(layout, (1, 1))

        y.plot(ax=ts_ax)
        ts_ax.set_title('Time Series Analysis Plots')
        smt.graphics.plot_acf(y, lags=lags, ax=acf_ax, alpha=0.5)
        smt.graphics.plot_pacf(y, lags=lags, ax=pacf_ax, alpha=0.5)

        print("Критерий Дики-Фуллера: p=%f" % sm.tsa.stattools.adfuller(y)[1])

        plt.tight_layout()
    return

tsplot(real_GDP, lags=30)
plt.show()
```

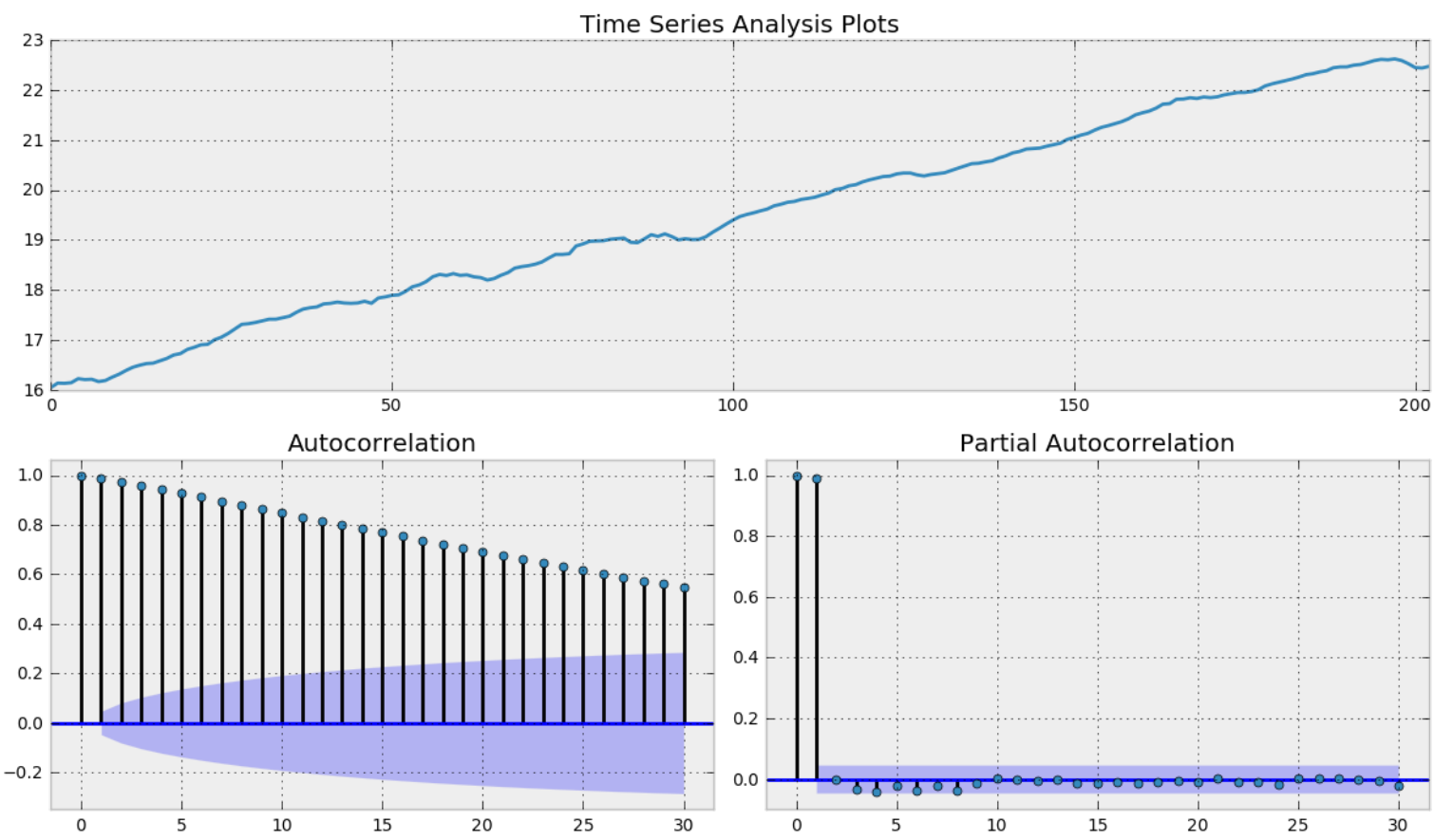
```
Критерий Дики-Фуллера: p=0.998246
```



In [25]:

```
# Попробуем стабилизировать дисперсию преоразованием Бокса-Кокса.
def invboxcox(y,lmbda):
    # обрабтное преобразование Бокса-Кокса
    if lmbda == 0:
        return(np.exp(y))
    else:
        return(np.exp(np.log(lmbda*y+1)/lmbda))

GDP = real_GDP.copy()
GDP['GDP_box'], lmbda = scs.boxcox(real_GDP + 1) # прибавляем единицу, так как в
tsplot(GDP.GDP_box, lags=30)
plt.show()
print("Оптимальный параметр преобразования Бокса-Кокса: %f" % lmbda)
```

Критерий Дики-Фуллера: p=0.648612



Оптимальный параметр преобразования Бокса-Кокса: 0.162072

А теперь попробуем взять не стабилизирование дисперсии, так как по природе наш ряд не зашумленный и неамплитудный, но при этом имее постоянную тенденцию к росту

In [33]:

```
returns = (real_GDP.apply(pd.to_numeric, errors='coerce').pct_change()).dropna()
tsplot(returns, lags=30)
```

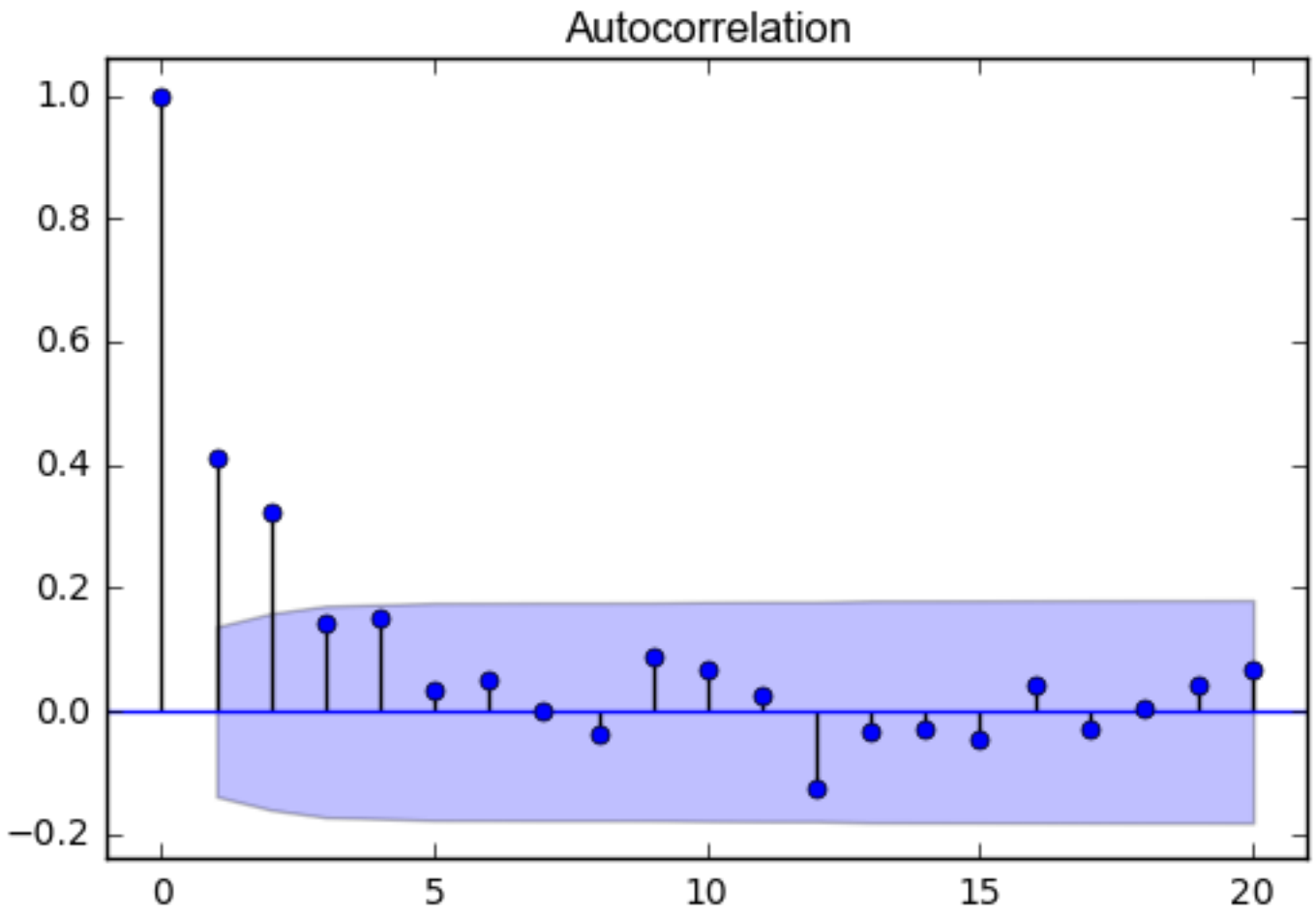
Критерий Дики-Фуллера: p=0.000000

Теперь гипотеза о нестационарности ряда отвергается, значит именно это преобразование было необходимо, и мы можем двигаться дальше, можно даже взять не темпы прироста, а абсолютные разности.

AR; MA; ARMA

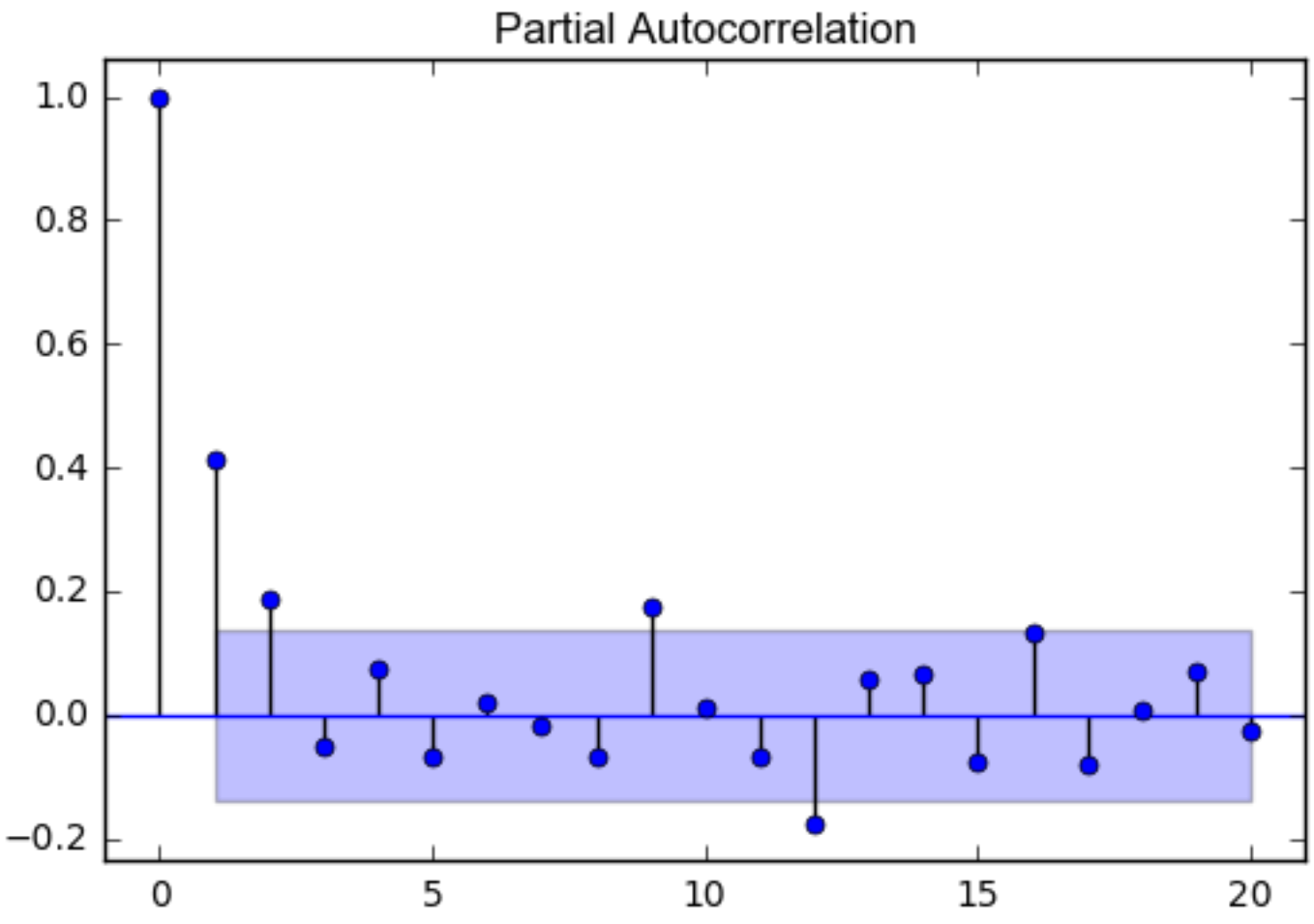
In [29]:

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf


chg_temp = (real_GDP.diff()).dropna()

plot_acf(chg_temp, lags=20)
plt.show()
```



In [30]:

```
plot_pacf(chg_temp, lags=20)
plt.show()
```



In [31]:

```
from statsmodels.tsa.arima_model import ARMA


mod = ARMA(chg_temp, order=(1,0))
res = mod.fit()
print("The AIC for an AR(1) is: ", res.aic)


mod = ARMA(chg_temp, order=(2,0))
res = mod.fit()
print("The AIC for an AR(2) is: ", res.aic)


mod = ARMA(chg_temp, order=(0,1))
res = mod.fit()
print("The AIC for an MA(1) is: ", res.aic)


mod = ARMA(chg_temp, order=(1,1))
res = mod.fit()
print("The AIC for an ARMA(1,1) is: ", res.aic)

#AR(2) и ARMA(1,1) обладают наименьщими метриками AIC,
#то есть проявляют себя лучше чем альтернативные им варианты
```

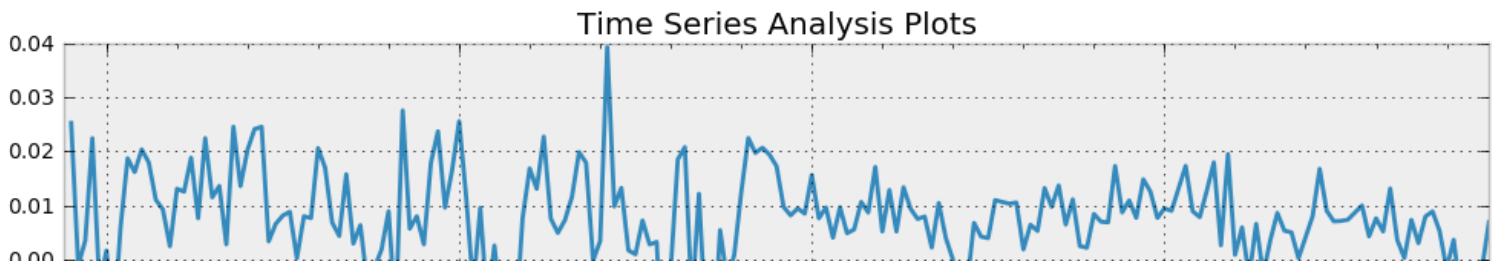
```
The AIC for an AR(1) is:  2195.2085071557376
The AIC for an AR(2) is:  2190.195344961745
The AIC for an MA(1) is:  2207.9569949823863
The AIC for an ARMA(1,1) is:  2191.7911407906513
```

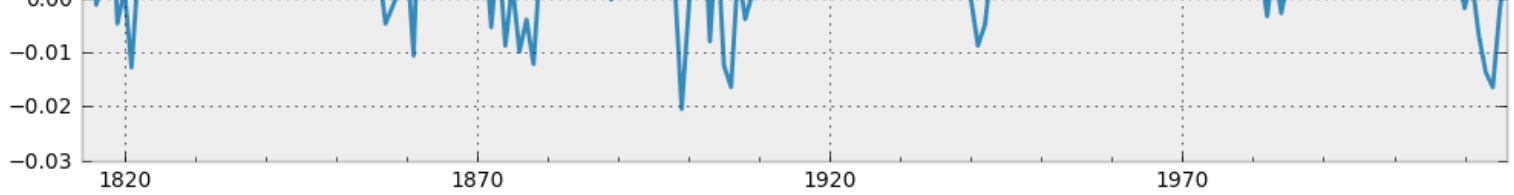
лучше всего себя проявила авторегрессия 2 порядка.

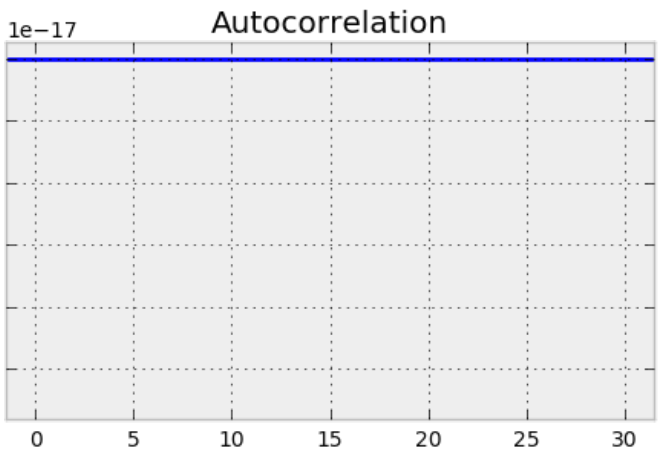
ARIMA

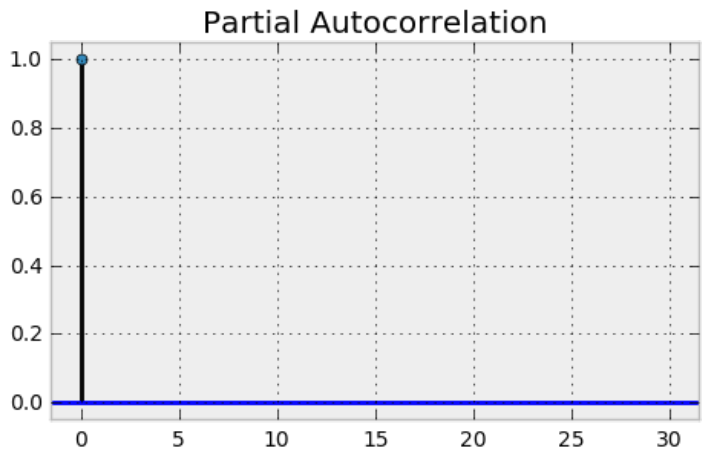
In [36]:

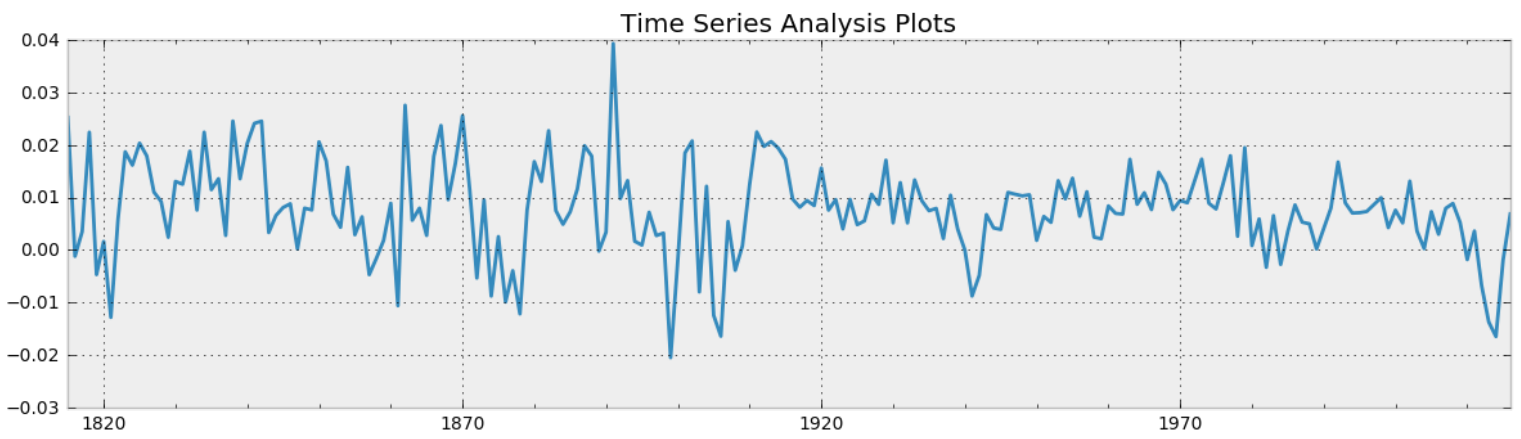
```
from statsmodels.tsa.arima_model import ARIMA


mod = ARIMA(real_GDP, order=(1,1,1))
res = mod.fit()

res.plot_predict(start='1872-01-01', end='2072-01-01')
plt.show()
```

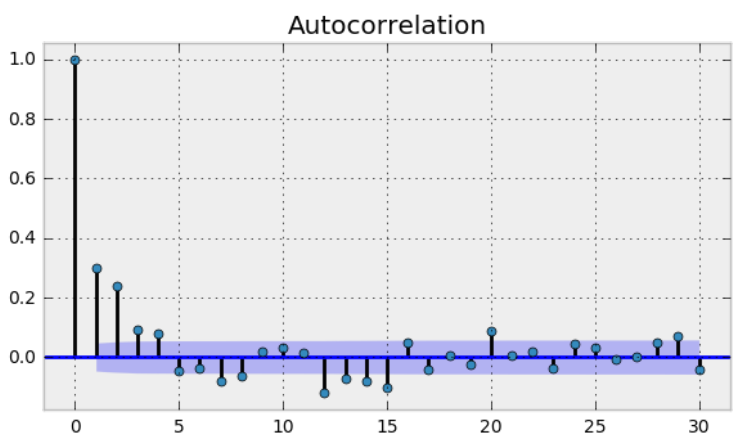


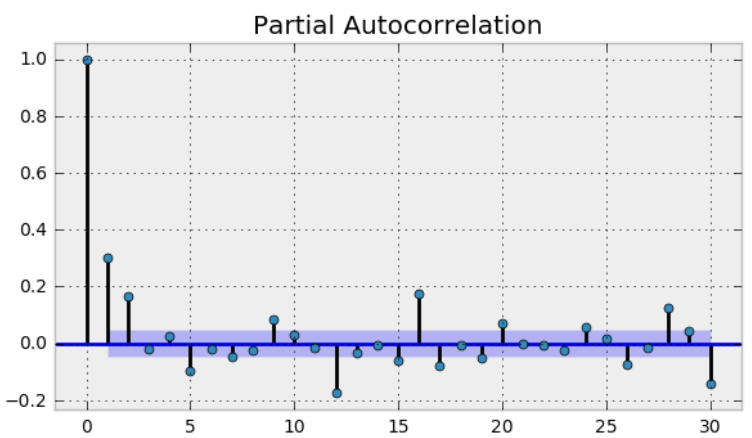


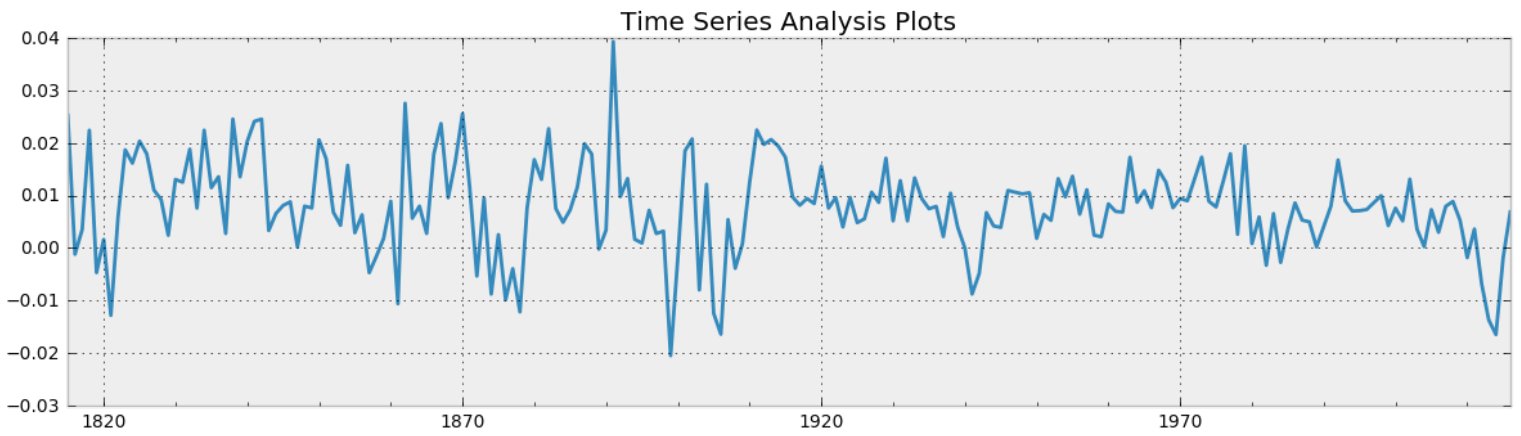

Autocorrelation
1e−17

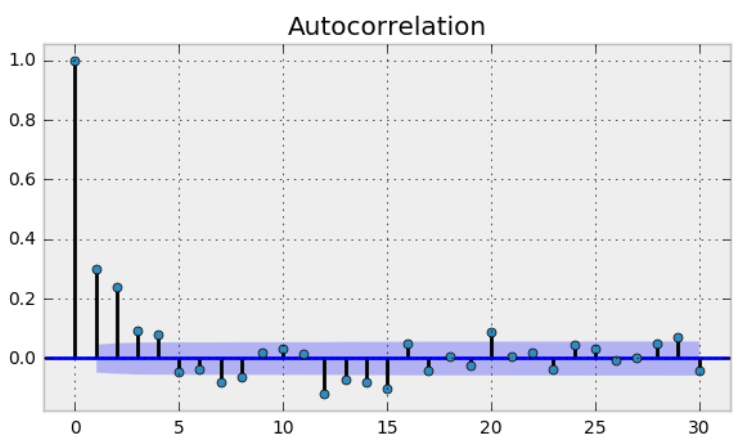

Partial Autocorrelation

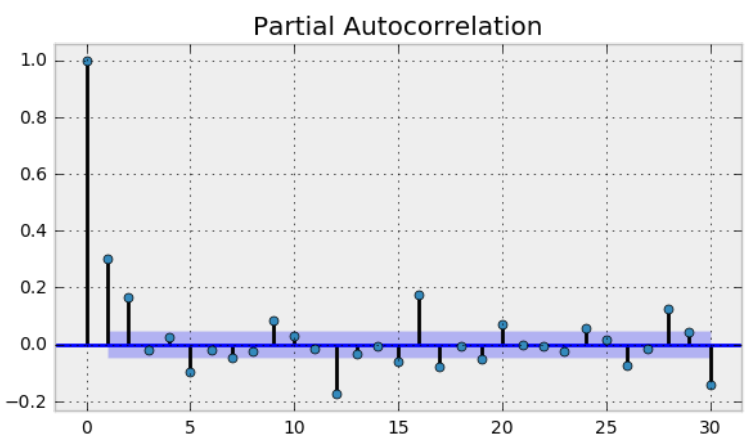

Time Series Analysis Plots


Autocorrelation

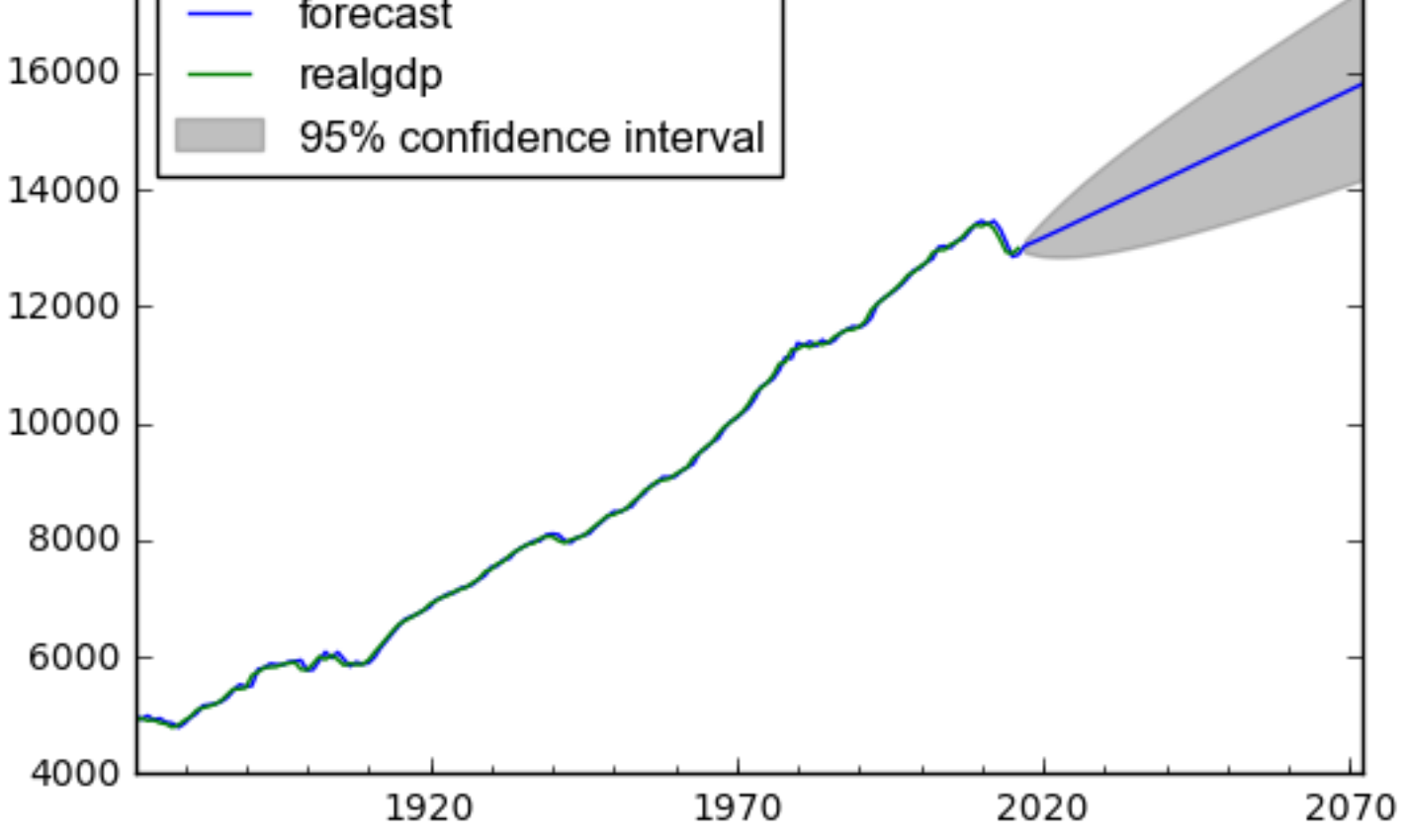

Partial Autocorrelation


Time Series Analysis Plots


Autocorrelation


Partial Autocorrelation

18000



Наконец, получили стационарный ряд, по автокорреляционной и частной автокорреляционной функции прикинем параметры для SARIMA модели, на забыв, что предварительно уже сделали первые и сезонные разности.

Начальные приближения Q = 1, P = 4, q = 3, p = 4

In [37]:

```
ps = range(0, 5)
d=1
qs = range(0, 4)
Ps = range(0, 5)
D=1
Qs = range(0, 1)

from itertools import product

parameters = product(ps, qs, Ps, Qs)
parameters_list = list(parameters)
len(parameters_list)
```

Out[37]:

```
100
```

In [39]:

```
%%time
best_model = sm.tsa.statespace.SARIMAX(GDP.GDP_box, order=(4, 1, 3),
                                       seasonal_order=(4, 1, 1, 24)).fit(disp=-1
print(best_model.summary())



#model=sm.tsa.statespace.SARIMAX(GDP.GDP_box, order=(param[0], d, param[1]),
                                       #seasonal_order=(param[2], D, param[3], 2
```

```
/Users/Tatyana/anaconda/lib/python3.5/site-packages/statsmodels/base
/model.py:508: ConvergenceWarning: Maximum Likelihood optimization f
ailed to converge. Check mle_retvals
  "Check mle_retvals", ConvergenceWarning)
```

Statespace Model Results

Dep. Variable:	y	No. Observations:	203
Model:	SARIMAX(4, 1, 3)x(4, 1, 1, 24)	Log Likelihood	329.557
Date:	Thu, 28 Jun 2018	AIC	-633.113
Time:	09:51:17	BIC	-591.750
Sample:	0	HQIC	-616.339
	- 203		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.1465	0.529	-0.277	0.782	-1.183	0.890
ar.L2	1.0512	0.477	2.203	0.028	0.116	1.987
ar.L3	-0.0258	0.305	-0.085	0.933	-0.623	0.572
ar.L4	-0.2637	0.174	-1.513	0.130	-0.605	0.078
ma.L1	0.4183	0.536	0.780	0.435	-0.632	1.469
ma.L2	-0.7063	0.533	-1.324	0.185	-1.752	0.339
ma.L3	-0.1840	0.345	-0.533	0.594	-0.861	0.493
ar.S.L24	-0.7661	0.550	-1.392	0.164	-1.845	0.313
ar.S.L48	-0.5664	0.364	-1.554	0.120	-1.281	0.148
ar.S.L72	-0.3640	0.249	-1.464	0.143	-0.851	0.123
ar.S.L96	-0.2875	0.139	-2.069	0.039	-0.560	-0.015
ma.S.L24	0.0371	0.598	0.062	0.951	-1.136	1.210
sigma2	0.0013	0.000	9.353	0.000	0.001	0.002

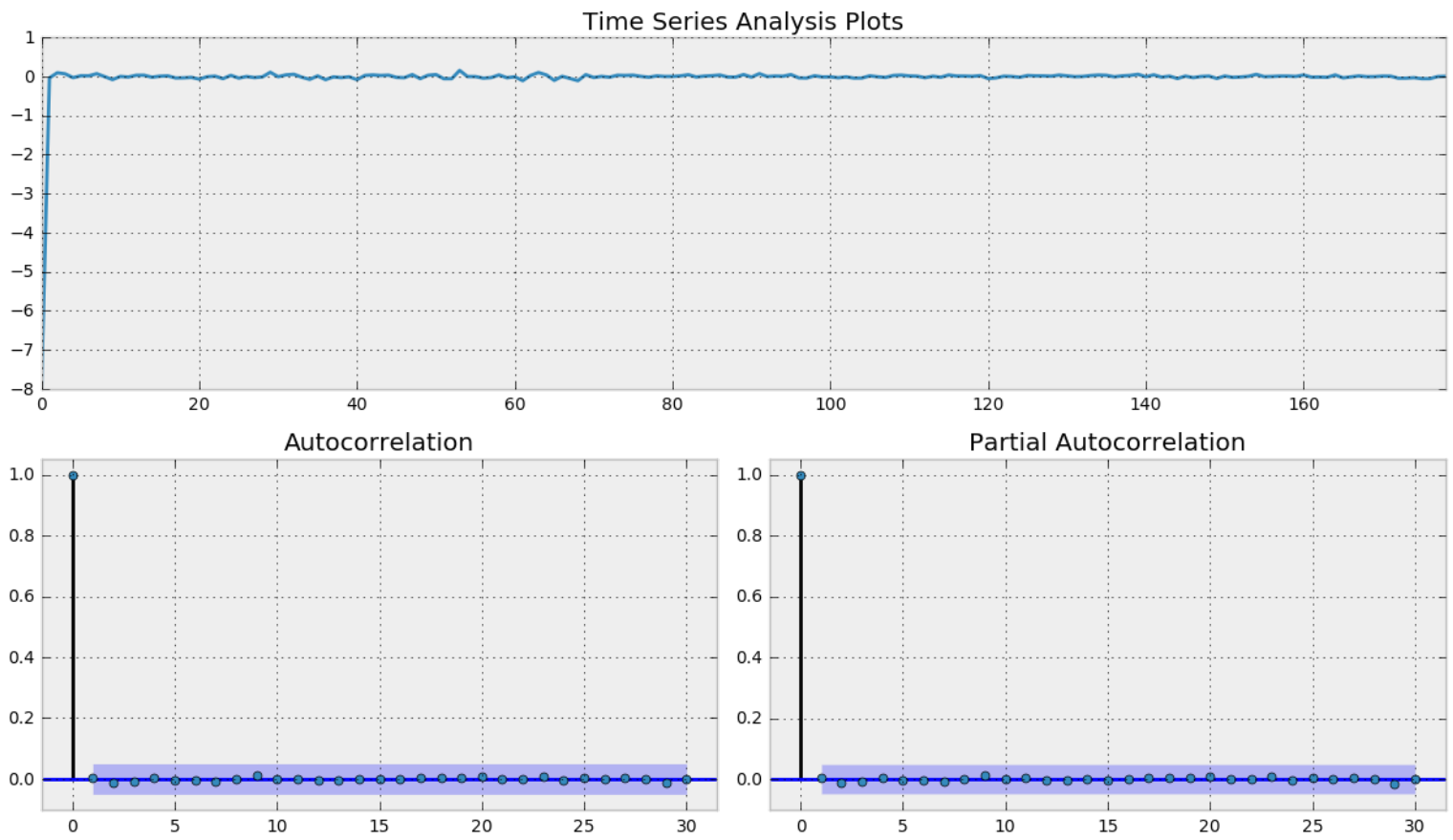
Ljung-Box (Q):	38.98	Jarque-Bera (JB):	18.81
Prob(Q):	0.52	Prob(JB):	

```
0.00
Heteroskedasticity (H):                    0.44   Skew:
0.18
Prob(H) (two-sided):                       0.00   Kurtosis:
4.55
================================================================================
===============

Warnings:
[1] Covariance matrix calculated using the outer product of gradient
s (complex-step).
CPU times: user 3min 42s, sys: 797 ms, total: 3min 43s
Wall time: 1min 52s
```

In [40]:

```
tsplot(best_model.resid[24:], lags=30)
plt.show()
```

Критерий Дики-Фуллера: p=0.000000



In [41]:

```
pd.DataFrame(GDP).columns
pd.DataFrame(GDP)
```

Out[41]:

	realgdp
1814-01-01 00:00:00	2710.35
1815-01-01 00:00:00	2778.8
1816-01-01 00:00:00	2775.49
1817-01-01 00:00:00	2785.2

1818-01-01 00:00:00	2847.7
1819-01-01 00:00:00	2834.39
1820-01-01 00:00:00	2839.02
1821-01-01 00:00:00	2802.62
1822-01-01 00:00:00	2819.26
1823-01-01 00:00:00	2872.01
1824-01-01 00:00:00	2918.42
1825-01-01 00:00:00	2977.83
1826-01-01 00:00:00	3031.24
1827-01-01 00:00:00	3064.71
1828-01-01 00:00:00	3093.05
1829-01-01 00:00:00	3100.56
1830-01-01 00:00:00	3141.09
1831-01-01 00:00:00	3180.45
1832-01-01 00:00:00	3240.33
1833-01-01 00:00:00	3264.97
1834-01-01 00:00:00	3338.25
1835-01-01 00:00:00	3376.59
1836-01-01 00:00:00	3422.47
1837-01-01 00:00:00	3431.96
1838-01-01 00:00:00	3516.25
1839-01-01 00:00:00	3563.96
1840-01-01 00:00:00	3636.28
1841-01-01 00:00:00	3724.01
1842-01-01 00:00:00	3815.42
1843-01-01 00:00:00	3828.12
...	...
1988-01-01 00:00:00	11596.4
1989-01-01 00:00:00	11598.8
1990-01-01 00:00:00	11645.8
1991-01-01 00:00:00	11738.7
1992-01-01 00:00:00	11935.5
1993-01-01 00:00:00	12042.8
1994-01-01 00:00:00	12127.6

1995-01-01 00:00:00	12213.8
1996-01-01 00:00:00	12303.5
1997-01-01 00:00:00	12410.3
1998-01-01 00:00:00	12534.1
1999-01-01 00:00:00	12587.5
2000-01-01 00:00:00	12683.2
2001-01-01 00:00:00	12748.7
2002-01-01 00:00:00	12915.9
2003-01-01 00:00:00	12962.5
2004-01-01 00:00:00	12965.9
2005-01-01 00:00:00	13060.7
2006-01-01 00:00:00	13099.9
2007-01-01 00:00:00	13204
2008-01-01 00:00:00	13321.1
2009-01-01 00:00:00	13391.2
2010-01-01 00:00:00	13366.9
2011-01-01 00:00:00	13415.3
2012-01-01 00:00:00	13324.6
2013-01-01 00:00:00	13141.9
2014-01-01 00:00:00	12925.4
2015-01-01 00:00:00	12901.5
2016-01-01 00:00:00	12990.3
GDP_box	[16.048442334, 16.1384080329, 16.13409678, 16....

204 rows × 1 columns

Постараемся не просто установить параметры модели, а подобрать их.

In [46]:

```
# подбор параметров перебором
%%time
results = []
best_aic = float("inf")

for param in tqdm(parameters_list):
    #try except нужен, потому что на некоторых наборах параметров модель не обуча
    try:
        model=sm.tsa.statespace.SARIMAX(GDP.GDP_box, order=(param[0], d, param[1]
                                        seasonal_order=(param[2], D, param[3], 24
    #выводим параметры, на которых модель не обучается и переходим к следующему н
    except ValueError:
        print('wrong parameters:', param)
        continue
    aic = model.aic
    #сохраняем лучшую модель, aic, параметры
    if aic < best_aic:
        best_model = model
        best_aic = aic
        best_param = param
    results.append([param, model.aic])

warnings.filterwarnings('default')

result_table = pd.DataFrame(results)
result_table.columns = ['parameters', 'aic']
print(result_table.sort_values(by = 'aic', ascending=True).head())
```

```
  File "<ipython-input-46-ae75e0e76667>", line 2
    %%time
    ^
SyntaxError: invalid syntax
```

VAR

Выразить векторную авторегрессионную модель мы можем довольно простой для понимания формулой

[y*{t}* = *c* + *A*{1}y*{t-1}* + ... + *A*{p}y*{t-p}* + *e*{t}]

Как мы видим, предсказываемое нами значение временного ряда с некоторым коэффициентом определяется предыдущими значениями и случайной ошибкой.

In [47]:

```
mdata = sm.datasets.macrodata.load().data
mdata = mdata[['realgdp','realcons','realinv']]
names = mdata.dtype.names
data = mdata.view((float,3))
data = np.diff(np.log(data), axis=0)
newdata = pd.DataFrame(mdata)
```

In [48]:

```
model = VAR(data)
res = model.fit(2)
```

In [52]:

```
res.summary()
```

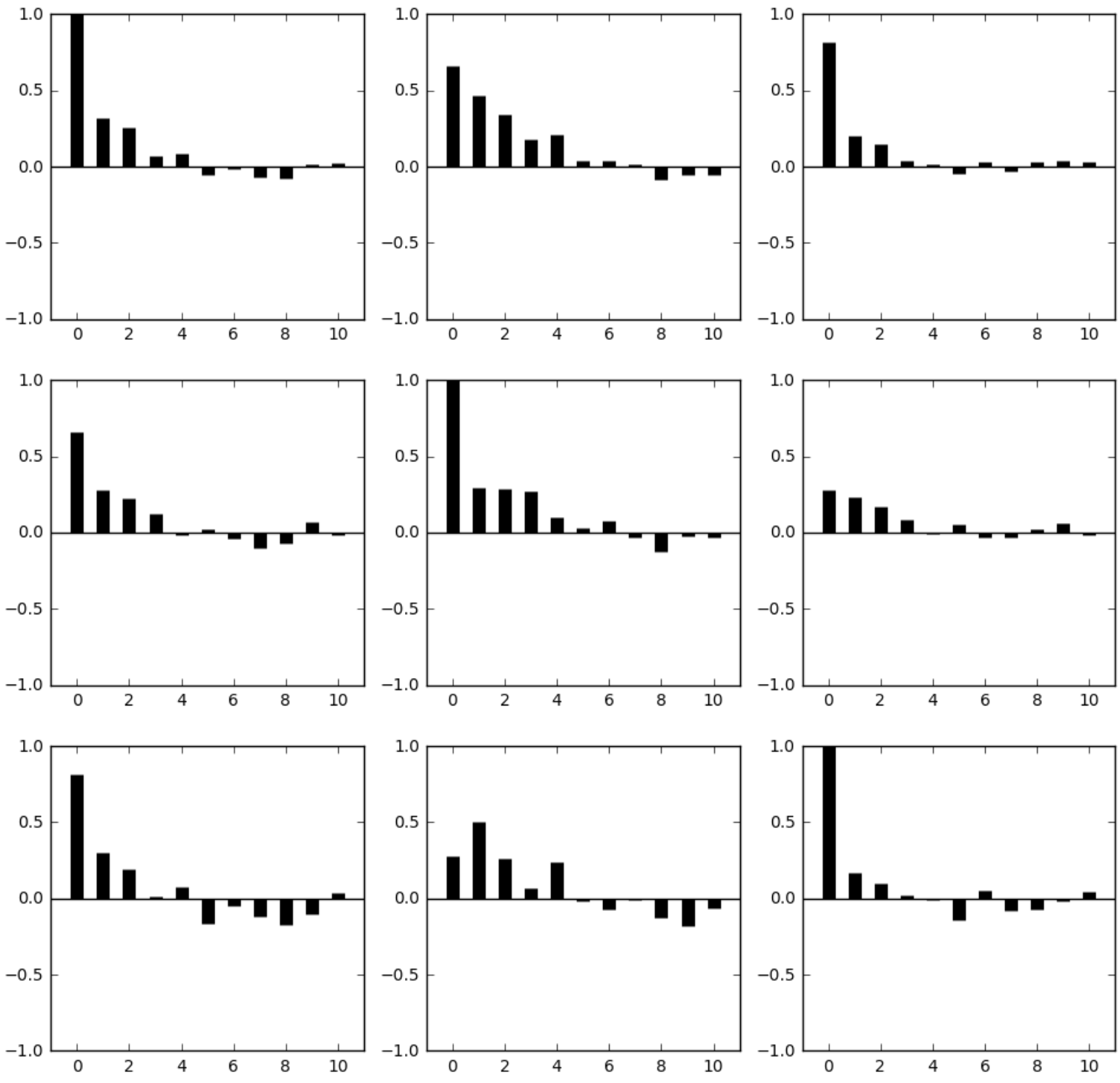
Out[52]:

```
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Thu, 28, Jun, 2018
Time:                     10:01:32
--------------------------------------------------------------------
No. of Equations:         3.00000    BIC:                    -27.5830
Nobs:                     200.000    HQIC:                   -27.7892
Log likelihood:           1962.57    FPE:                 7.42129e-13
AIC:                     -27.9293    Det(Omega_mle):      6.69358e-13
--------------------------------------------------------------------
Results for equation y1
====================================================================
====
          coefficient       std. error           t-stat
prob
--------------------------------------------------------------------
----
const         0.001527         0.001119            1.365           0
.172
L1.y1        -0.279435         0.169663           -1.647           0
.100
L1.y2         0.675016         0.131285            5.142           0
.000
L1.y3         0.033219         0.026194            1.268           0
.205
L2.y1         0.008221         0.173522            0.047           0
.962
L2.y2         0.290458         0.145904            1.991           0
.047
L2.y3        -0.007321         0.025786           -0.284           0
.776
====================================================================
====

Results for equation y2
```

```
==========================================================================
====
          coefficient       std. error           t-stat
prob
--------------------------------------------------------------------------
----
const         0.005460         0.000969            5.634            0
.000
L1.y1        -0.100468         0.146924           -0.684            0
.494
L1.y2         0.268640         0.113690            2.363            0
.018
L1.y3         0.025739         0.022683            1.135            0
.257
L2.y1        -0.123174         0.150267           -0.820            0
.412
L2.y2         0.232499         0.126350            1.840            0
.066
L2.y3         0.023504         0.022330            1.053            0
.293
==========================================================================
====

Results for equation y3
==========================================================================
====
          coefficient       std. error           t-stat
prob
--------------------------------------------------------------------------
----
const        -0.023903         0.005863           -4.077            0
.000
L1.y1        -1.970974         0.888892           -2.217            0
.027
L1.y2         4.414162         0.687825            6.418            0
.000
L1.y3         0.225479         0.137234            1.643            0
.100
L2.y1         0.380786         0.909114            0.419            0
.675
L2.y2         0.800281         0.764416            1.047            0
.295
L2.y3        -0.124079         0.135098           -0.918            0
.358
==========================================================================
====

Correlation matrix of residuals
            y1        y2        y3
y1    1.000000  0.603316  0.750722
y2    0.603316  1.000000  0.131951
y3    0.750722  0.131951  1.000000
```

In [44]:

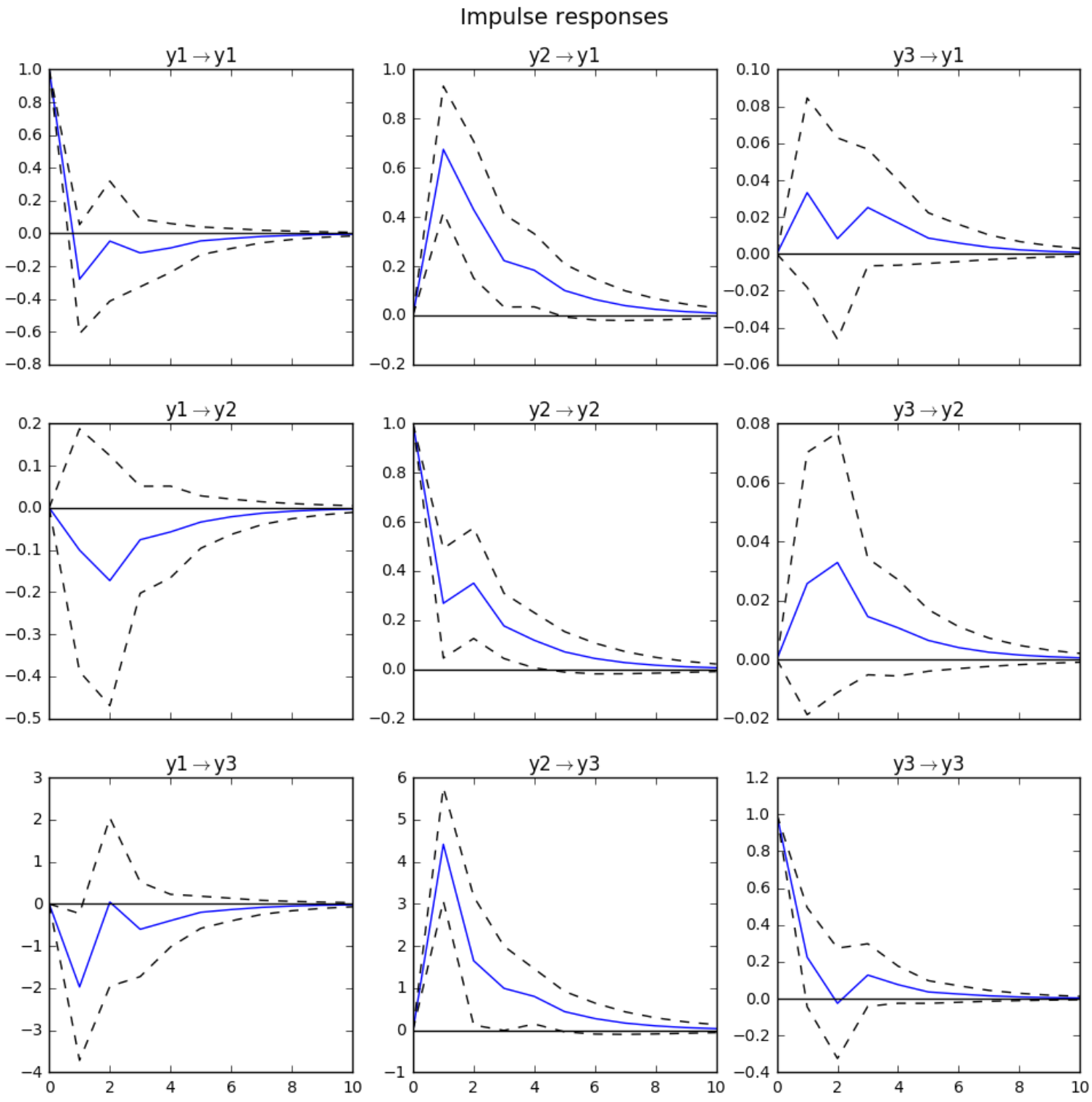
```
res.plot_sample_acorr()
plt.show()
```



На этих графиках очень хорошо видно, что автокореляция между наблюдениями затухает на третьем по величине лаге, а затем становится либо отрицательной, либо почти нулевой. А теперь мы выведем на график функции импульсных откликов, чтобы проанализировать, как связаны между собой наши переменные.

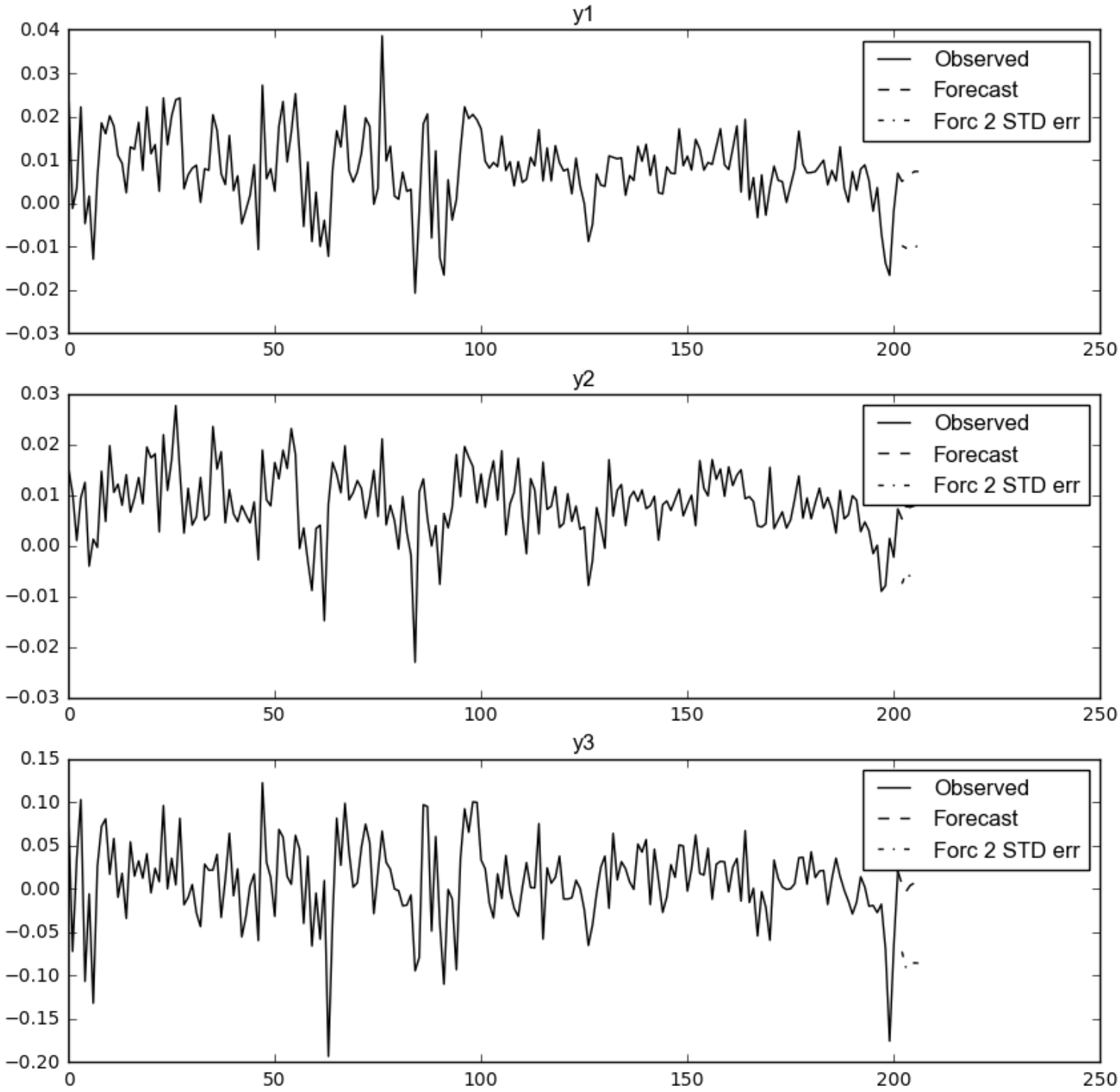
In [45]:

```
irf = res.irf(10) # 10 periods
irf.plot()
plt.show()
```



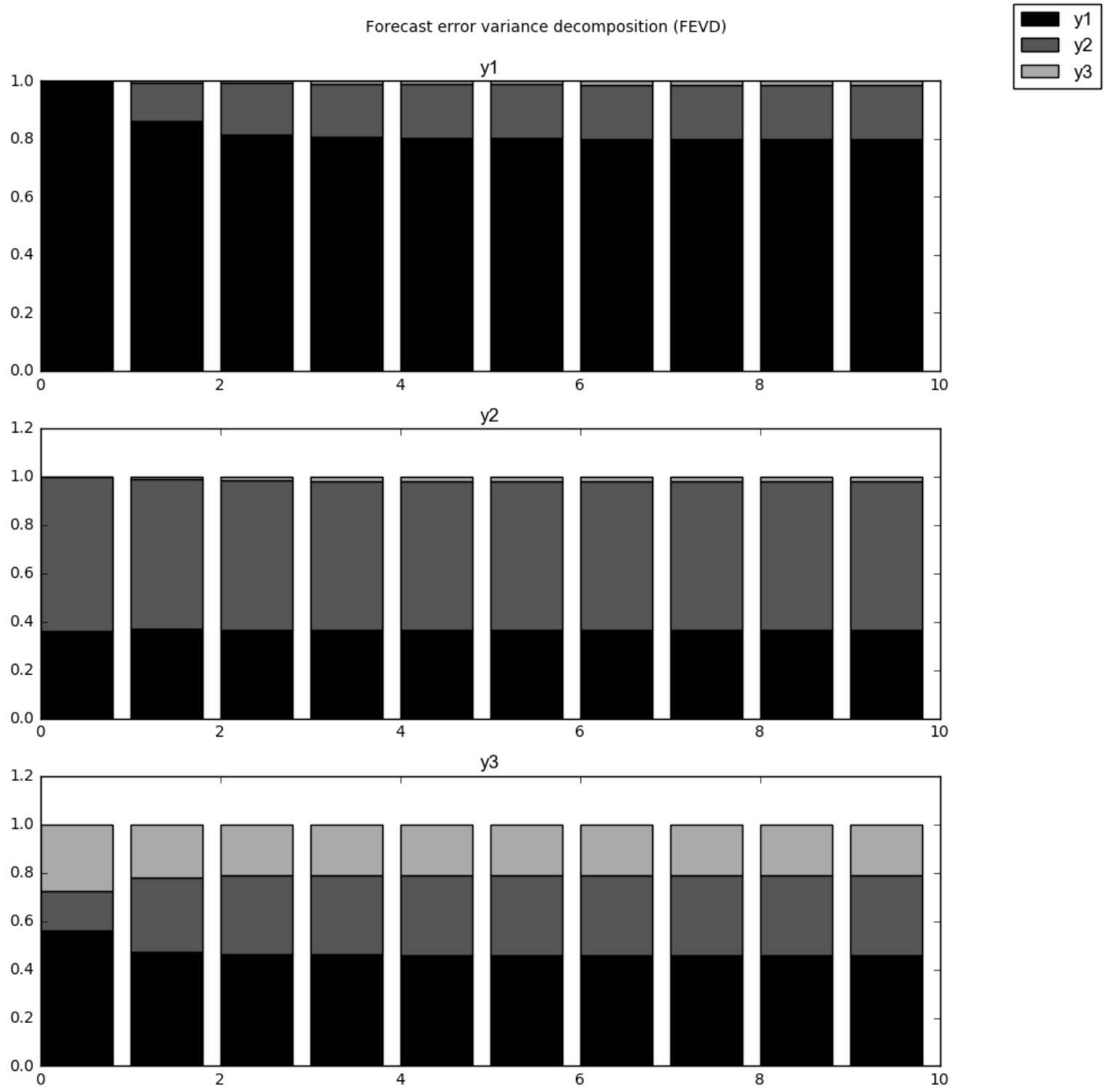
In [46]:

```
res.plot_forecast(5)
plt.show()
```



In [47]:

```
res.fevd().plot()
plt.show()
```



Трудно сказать, что какая-либо из составляющих перетягивает ошибку на себя, однако видно, что сильнее всего выделяется в декомпозиции ошибки первый столбец.

По остаткам модель VAR повела себя лучше чем другие и оказалась не менее эффективной.

Чтобы реализовать в качестве априорных предположений для переменных алгоритм черной коробки, Лапласа и Метрополиса-Гастингса, используем библиотеку pyflux и выведем таблицы summary для каждой из вновь получившихся моделей.

import pyflux as pf model = pf.VAR(data=data, lags=2, integ=1)

print(model.z)

model.list_priors()

```
model.adjust_prior(2, pf.Normal(0,1))

a = model.fit(method='BBVI', iterations='10000', optimizer='ADAM')

b = model.fit(method='Laplace')

c = model.fit(method='M-H')

Black Box = a.summary()

Laplace = b.summary()

Metropolis_Hastings = c.summary()
```

Если вести сравнение по метрике квадрата ошибки, то BVAR показала себя даже хуже чем обычная векторная авторегрессионная модель

In [ ]: