Time Series for Python with PyFlux

Ross Taylor

PyData San Francisco

Get these slides at http://www.github.com/RJT1990

I will upload a notebook containing all examples in due course

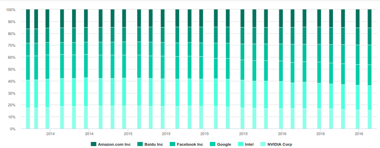
August 13, 2016

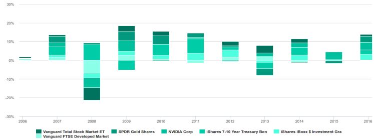
Outline

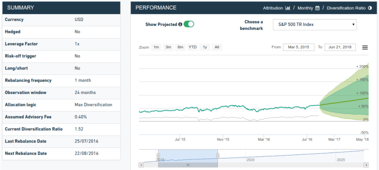
About Me

- Currently: Financial Engineer @ ALPIMA
- Previously: Quantitative Analyst @ Pythia; Economist @ UK Treasury
- Econometrics background (MPhil Economics, Cambridge)
- Work in R and Python; for past 2 years, increasingly Python

About Me







ALPIMA
SMART INVESTING MADE SIMPLE

- London start up: our platform makes quant finance simple and accessible
- We combine traditional financial advice with a strong quantitative foundation in mathematics, statistics and machine learning
- Things our quants like: Bayesian nonparametrics, generative models, reinforcement learning, approximate inference
- Public website: http://www.alpima.net

About PyFlux

- New time series library for Python, with focus on structural models
- New models: score-driven models, non-Gaussian state space models...
- Extremely flexible array of inference options; classical and Bayesian
- See PyFlux.com for examples and documentation

In the long term, these methods probably belong somewhere else in the PyData stack - statsmodels? Haven't decided yet; suggestions welcome.

Outline

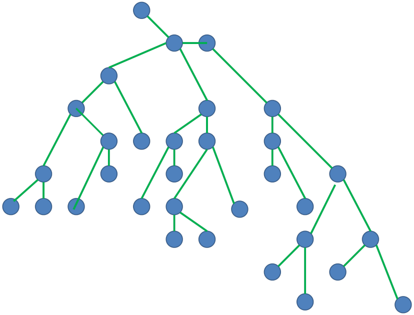
Features of Time Series Data

Time series data have unique characteristics:

- **Sequential observations** - indexed by time, space or location
- **Latent dependence structures** - trends, seasonality, cycles
- **Dynamic behaviour** - abrupt (regime shifts) or gradual (local levels)

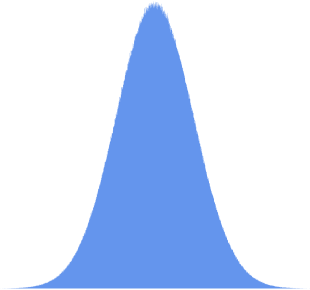
Objective: to forecast future values of a series and their uncertainty

Two Approaches



Algorithmic

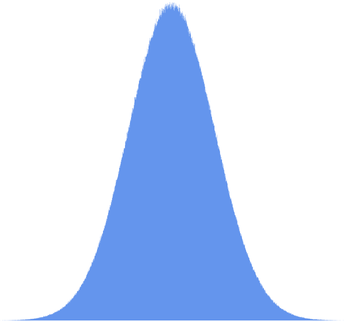
- Black box $f(x_t)$ to predict y_t



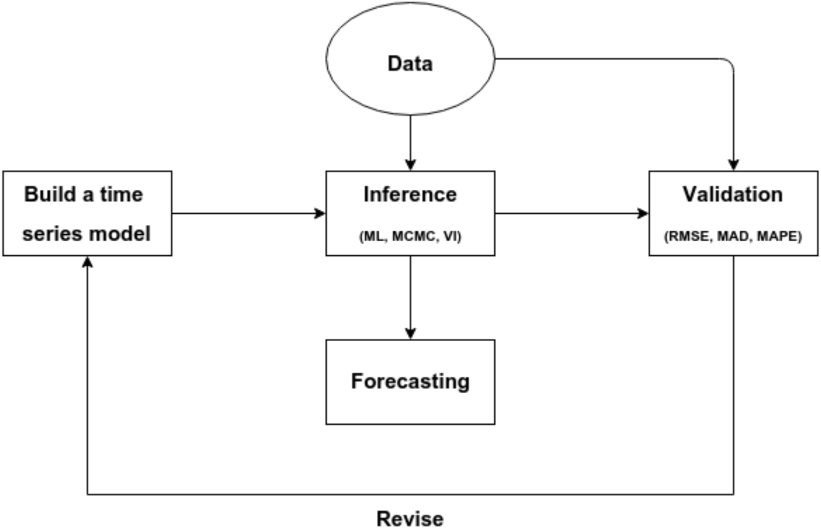
Probabilistic

- Joint probability $p(y_t, z_t)$

Today's Focus



Box's Loop (based on Blei 2014)



Outline

Box-Jenkins Time Series

Currency is the ARIMA model (autoregressive integrated moving average).
Consider a univariate time series y_t:

$$y_t = \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} + \epsilon_t$$

An ARMA(p,q) model

Key principles:

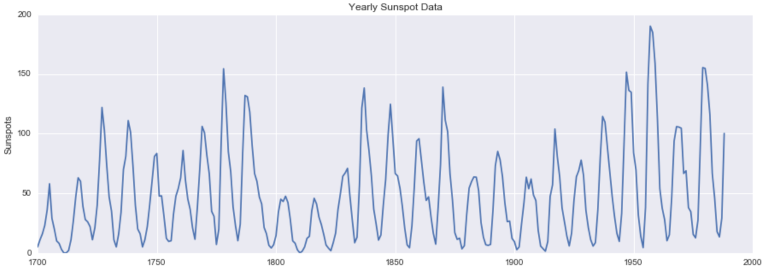
- **Linear** and **Gaussian**
- Captures **autocorrelation** through AR and MA lags
- Series should be differenced to achieve **stationarity**
- Lag order chosen by information criteria or cross-validation

Wolf Sunspot Data

In [1]:
```
import numpy as np
import pandas as pd
import pyflux as pf
import matplotlib.pyplot as plt
%matplotlib inline

data = pd.read_csv('https://vincentarelbundock.github.io/Rdatasets/csv/datasets/sunspot.year.csv')
data.index = data['time'].values

plt.figure(figsize=(15,5))
plt.plot(data.index,data['sunspot.year'])
plt.ylabel('Sunspots')
plt.title('Yearly Sunspot Data');
```



Inference for ARIMAs

- Maximum likelihood - modal approximation - is usually sufficient
- ARIMAs often seen as 'frequentist' as a result; this is misleading.
- For example, consider inference with Metropolis-Hastings:

$$\mu \sim N(0, 100)$$

$$\phi_i \sim N(0, 0.5)$$

In [4]:
```
model = pf.ARIMA(ar=9,ma=0,data=data,target='sunspot.year')
model.adjust_prior([0],pf.Normal(0,100))
print(model.latent_variables)
```

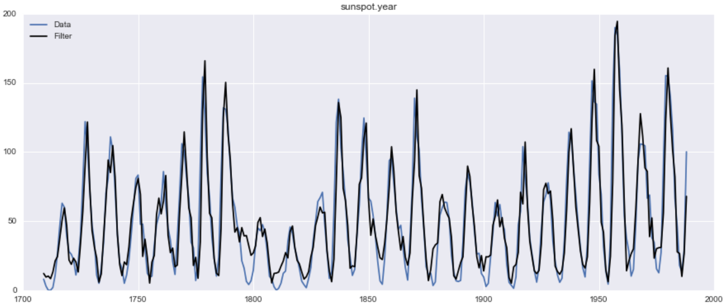
```
Index    Latent Variable           Prior           Prior Latent Vars         V.I. Dist  Transform
======== ========================= =============== ========================= ========== ==========
0        Constant                  Normal          mu0: 0, sigma0: 100       Normal     None
1        AR(1)                     Normal          mu0: 0, sigma0: 0.5       Normal     None
2        AR(2)                     Normal          mu0: 0, sigma0: 0.5       Normal     None
3        AR(3)                     Normal          mu0: 0, sigma0: 0.5       Normal     None
4        AR(4)                     Normal          mu0: 0, sigma0: 0.5       Normal     None
5        AR(5)                     Normal          mu0: 0, sigma0: 0.5       Normal     None
6        AR(6)                     Normal          mu0: 0, sigma0: 0.5       Normal     None
7        AR(7)                     Normal          mu0: 0, sigma0: 0.5       Normal     None
8        AR(8)                     Normal          mu0: 0, sigma0: 0.5       Normal     None
9        AR(9)                     Normal          mu0: 0, sigma0: 0.5       Normal     None
10       Sigma                     Uniform         n/a (non-informative)     Normal     exp
```

ARIMA Model Fit

In [5]:
```
x = model.fit('M-H',nsims=50000)
model.plot_fit(figsize=(15,6))
```

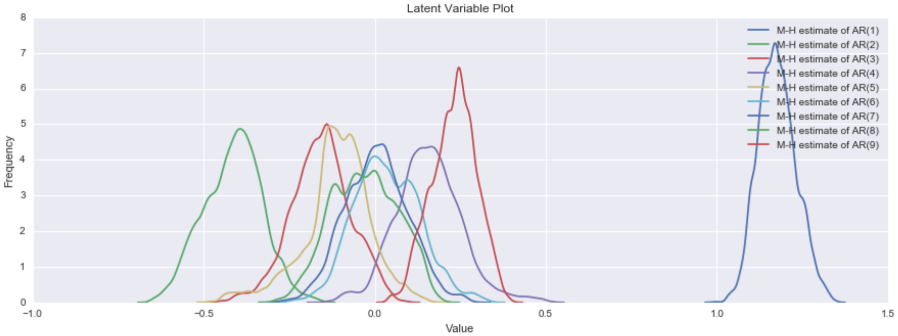
```
Acceptance rate of Metropolis-Hastings is 0.00724
Acceptance rate of Metropolis-Hastings is 0.58036
Acceptance rate of Metropolis-Hastings is 0.48316
Acceptance rate of Metropolis-Hastings is 0.37968

Tuning complete! Now sampling.
Acceptance rate of Metropolis-Hastings is 0.38136
```



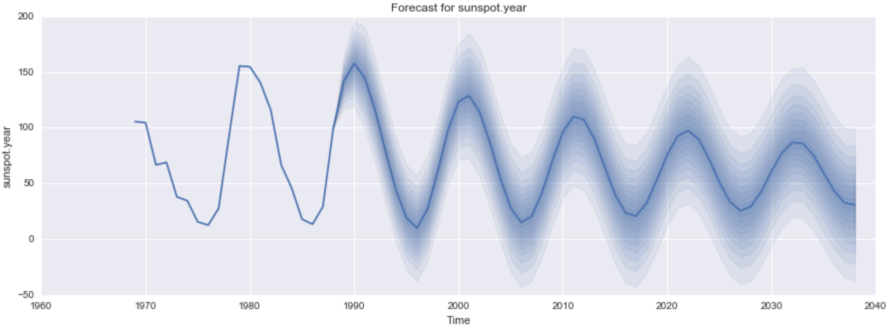
ARIMA Latent Variables

```
In [10]: model.plot_z(list(range(1,10)),figsize=(15,5))
```



Forecasting with ARIMA Models

```
In [4]: model.plot_predict(h=50,figsize=(15,5))
```



Limitations with Box-Jenkins

ARIMAs (and VARs) can take you a long way, but:

- **No decomposition** of the **latent processes** driving the data
- Many problems we care about are **non-Gaussian**
- Many problems we care about are **non-linear**

We need a framework that is more intuitive and theoretically complete

Outline

1. Introduction to Time Series Modelling
2. Box-Jenkins Time Series
3. Structural Time Series
4. Score-Driven Time Series
5. Application: Modelling NFL outcomes

Structural Time Series

Structural models have a **state space form**:

$$y_t = Z_t \alpha_t + \epsilon_t$$

$$\alpha_t = T_t \alpha_{t-1} + \eta_t$$

$$\epsilon_t \sim N(0, \Sigma_\epsilon)$$

$$\eta_t \sim N(0, \Sigma_\eta)$$

Intuition:

- We observe some time series y_t
- The series evolve according to some latent states α_t
- We want to distinguish between signal $Z_t \alpha_t$ and noise ϵ_t

Terminology

State space models solve three problems:

- **Filtering** : the distribution of the current state α_t given current and previous measurements $y_{1:t}$
- **Prediction** : the distribution of a future state α_{t+k} given current and previous measurements $y_{1:t}$
- **Smoothing** : the distribution of the current state α_t given current, previous and future measurements $y_{1:T}$

The Kalman Filter

Closed form solution for linear and Gaussian case is given by the celebrated **Kalman Filter and Smoother**.

$$\alpha_{t+1} = \alpha_t + K_t \left(y_t - Z_t \alpha_t \right)$$
$$P_{t+1} = \left(I - K_t H_t \right) P_t$$

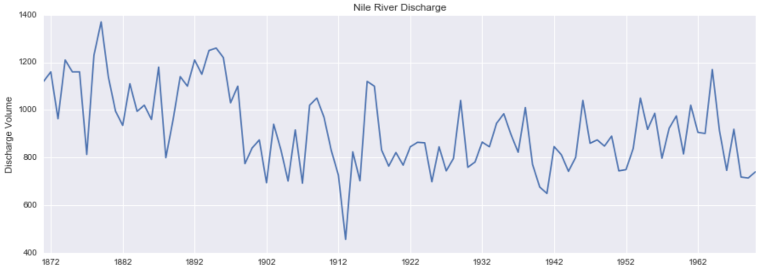
Filtering Equations

High-level intuition:

- $y_t - Z_t \alpha_t$ is a **linear prediction error**; we update in this direction.
- K_t is the **Kalman Gain** determining the signal/noise ratio
- The hyperparameters we estimate affect the strength of updating

Discharge from the River Nile

```
In [20]: nile = pd.read_csv('https://vincentarelbundock.github.io/Rdatasets/csv/datasets/Nile.csv')
         nile.index = pd.to_datetime(nile['time'].values,format='%Y')
         plt.figure(figsize=(15,5))
         plt.plot(nile.index,nile['Nile'])
         plt.ylabel('Discharge Volume')
         plt.title('Nile River Discharge');
         plt.show()
```



Local Level Model

The simplest type of structural model for a time-varying mean:

$$y_t = \mu_t + \epsilon_t$$

$$\mu_t = \mu_{t-1} + \eta_t$$

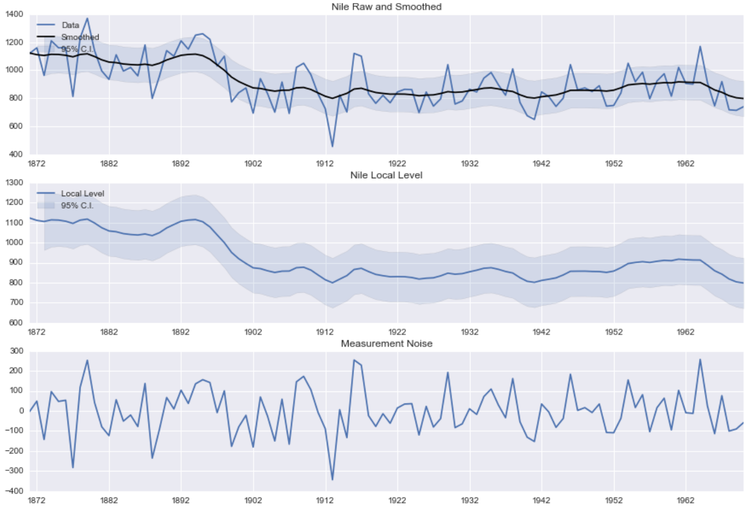
$$\epsilon_t \sim N\left(0, \sigma_\epsilon^2\right)$$

$$\eta_t \sim N\left(0, \sigma_\eta^2\right)$$

- The likelihood for this model is available in closed form.
- We estimate σ_η and σ_ϵ.
- Reduced form is an ARIMA(0,0,1) model.

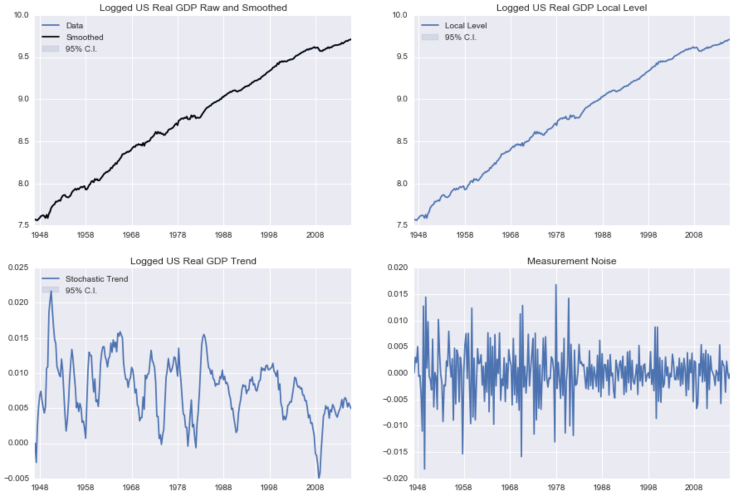
Pub Quiz: Why did the river discharge fall?

```
In [22]: model = pf.LLEV(data=nile,target='Nile')
         model.fit('MLE')
         model.plot_fit(figsize=(15,10))
```



Local Linear Trend Model for US GDP*

In [7]:
```
model = pf.LLT(data=USgrowth)
x = model.fit('MLE')
model.plot_fit(figsize=(15,10))
```



Dynamic Regression Model

Models coefficients β_t as random walk processes:

$$y_t = x_t^{'}\beta_t + \epsilon_t$$

$$\beta_t = \beta_{t-1} + \eta_t$$

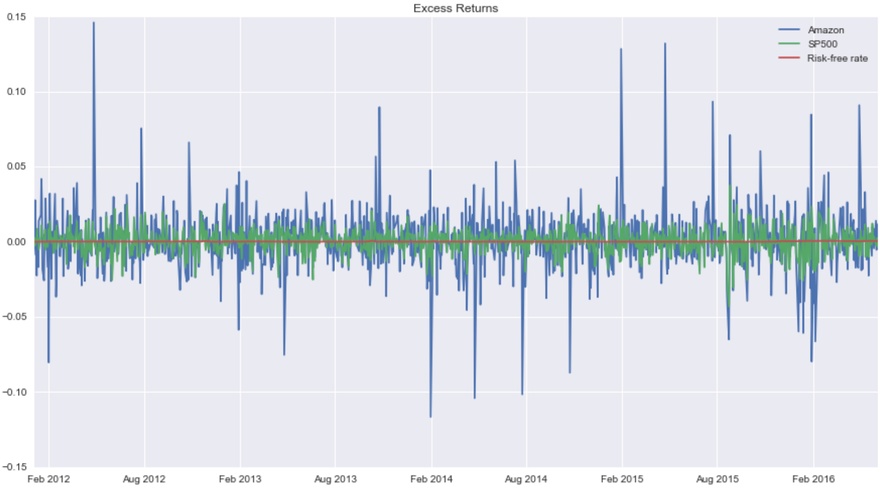
$$\epsilon_t \sim N\left(0, \sigma_\epsilon^2\right)$$

$$\eta_t \sim N\left(0, \Sigma_\eta\right)$$

- We can estimate again through the Kalman filter/smoother
- Simple but powerful model type for **dynamic relationships**

Example: Dynamic Betas for Finance

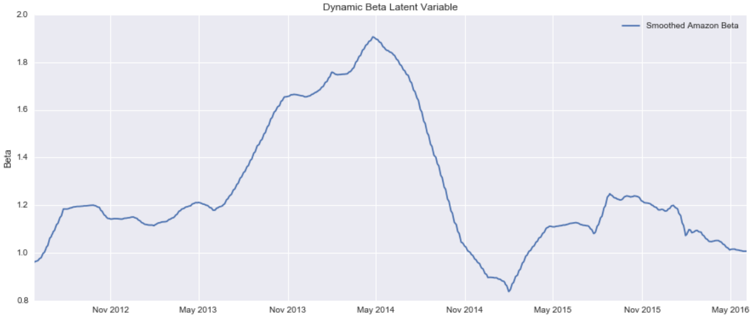
A dynamic regression of a stock on the market allows us to assess its **systematic risk** and how it evolves over time.



Dynamic Beta for Finance

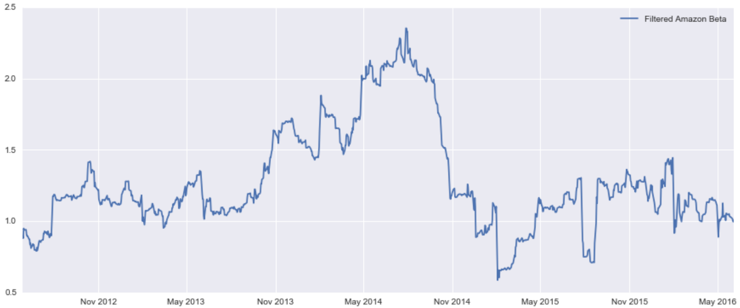
```
In [50]: model3 = pf.DynReg('Amazon ~ SP500',data=final_returns)
         x = model3.fit()

         plt.figure(figsize=(15,6))
         plt.plot(final_returns.index[100:], x.states[1][100:-1], label="Smoothed Amazon Beta");
         plt.ylabel("Beta")
         plt.title("Dynamic Beta Latent Variable");
         plt.legend();
```



The Filtered Estimate is More Problematic...

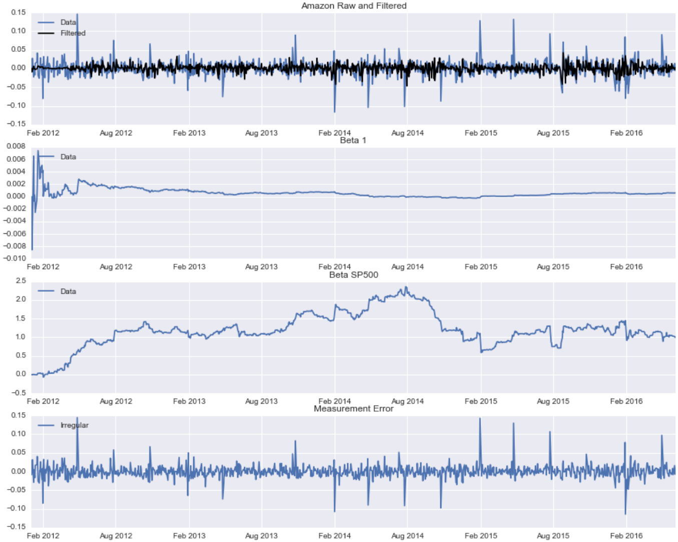
In [54]:
```
states_f, V, _, _, _ = model3._model(model3.data,model3.latent_variables.get_z_values())
plt.figure(figsize=(15,6))
plt.plot(final_returns.index[100:],states_f[1][100:-1],label='Filtered Amazon Beta');
plt.legend();
```



- Kalman filter only optimal for linear and Gaussian assumptions
- Jumps in 2015 caused by tail events for Amazon returns
- Using this β_t after an event would underestimate systematic risk!

Filtered Components

In [59]:
```
model3.plot_fit(series_type='Filtered',figsize=(15,12))
```



Limitations with Gaussian State Space Models

We have just scratched the surface; other state space models include:

- Cyclical and seasonal component models
- Paired comparison/ranking models
- Markov switching models

We have gained more interpretability over ARIMA models, but:

- We want a more general framework that is **non-Gaussian**
- We would like a **non-Gaussian filter** (if possible)
- We might like to **drop linearity** too - but that's for another talk...

Outline

1. Introduction to Time Series Modelling
2. Box-Jenkins Time Series
3. Structural Time Series
4. **Score-Driven Time Series**
5. Application: Modelling NFL outcomes

Score-Driven Time Series

Consider these two equations:

$$\alpha_{t+1} = \alpha_t + K_t \left(y_t - Z_t \alpha_t \right)$$

Kalman state filtering equation

$$y_{t+1} = \phi y_t + \theta \left(y_t - \mu_t \right)$$

ARMA(1,1) process

- Both rely on a **linear prediction error** (not great for heavy tails)
- What if we can achieve **Kalman-like** updates for other distributions?
- One way to achieve this is through the **score** of the distribution

Score of the Normal distribution

$$\log p\left(y_t\right) = -\frac{1}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2}\frac{\left(y_t - \mu_t\right)^2}{\sigma^2}$$

$$\nabla_{\mu_t} \log p\left(y_t\right) = \frac{\left(y_t - \mu_t\right)}{\sigma^2}$$

$$\nabla^2_{\mu_t} \log p\left(y_t\right) = -\frac{1}{\sigma^2}$$

$$\frac{\nabla_{\mu_t} \log p\left(y_t\right)}{-\nabla^2_{\mu_t} \log p\left(y_t\right)} = \left(y_t - \mu_t\right)$$

- The Gaussian scaled score update == the Kalman update
- Score-driven models exploit this principle for other distributions
- Replace Kalman update with score update $\rightarrow$ approximate filter

Score of the Poisson distribution

$$\lambda_t = \exp(\theta_t)$$

$$\log p(y_t) = y_t \theta_t - \exp(\theta_t) - Z$$

$$\nabla_{\theta_t} \log p(y_t) = y_t - \exp(\theta_t)$$

$$\nabla^2_{\theta_t} \log p(y_t) = -\exp(\theta_t)$$

$$\frac{\nabla_{\theta_t} \log p(y_t)}{-\nabla^2_{\theta_t} \log p(y_t)} = \frac{1}{\exp(\theta_t)} (y_t - \exp(\theta_t))$$

- Scale the Kalman filter update by $1/\lambda_t$ for a Poisson approximation

Example: An Olde Rivalry

Results Data for East Midlands Derby

We've been competing for a while...

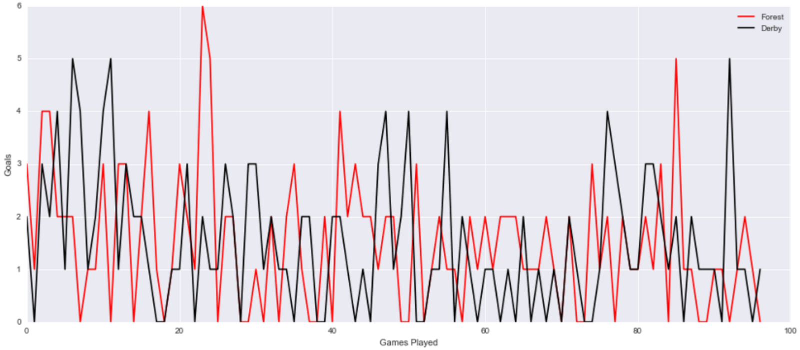
```
In [11]: eastmidlandsderby = pd.read_csv('eastmidlandsderby.csv')
         eastmidlandsderby.head()
```

Out[11]:

	Date	Forest	Derby	ForestHome	DerbyHome
0	01-10-1892	3	2	0	1
1	28-01-1893	1	0	1	0
2	09-12-1893	4	3	0	1
3	30-12-1893	4	2	1	0
4	08-09-1894	2	4	0	1

A Brief History of the East Midlands Derby

In [13]:
```
plt.figure(figsize=(17,7))
plt.plot(eastmidlandsderby['Forest'],label='Forest',color='r')
plt.plot(eastmidlandsderby['Derby'],label='Derby',color='k')
plt.ylabel('Goals')
plt.xlabel('Games Played')
plt.legend();
```



Poisson Local Level Model

- We will model goals as **Poisson local level models**.
- We'll also include a **time-varying home advantage effect**.
- Our model specification for each team's goals y_t is as follows

$$Pois\left(y_t \mid \theta_t\right)$$

$$\theta_t = \mu_t + I_t \beta_t$$

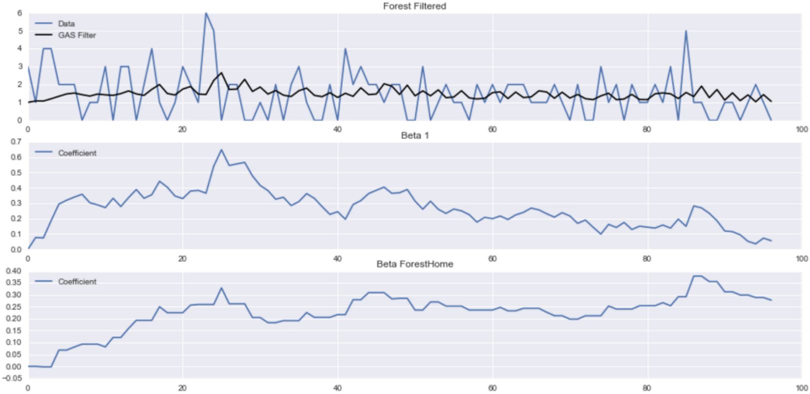
$$\mu_t = \mu_{t-1} + \eta_1 U_{t-1}$$

$$\beta_t = \beta_{t-1} + \eta_2 U_{t-1}$$

- I_t is an home/away match identifier
- U_t is the Poisson score for the model at time t
- η_i is a scale (or learning rate) latent variable to estimate

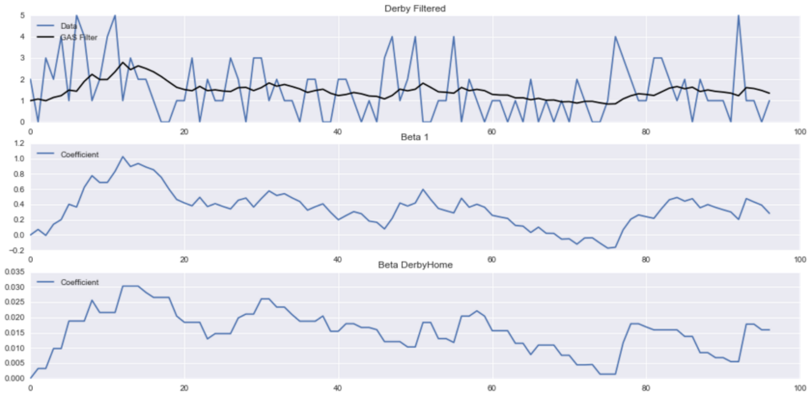
Nottingham Forest Local Level

In [7]:
```
model = pf.GASReg("Forest ~ ForestHome", data=eastmidlandsderby, family=pf.GASPoisson())
model.fit()
model.plot_fit(figsize=(17,8))
```

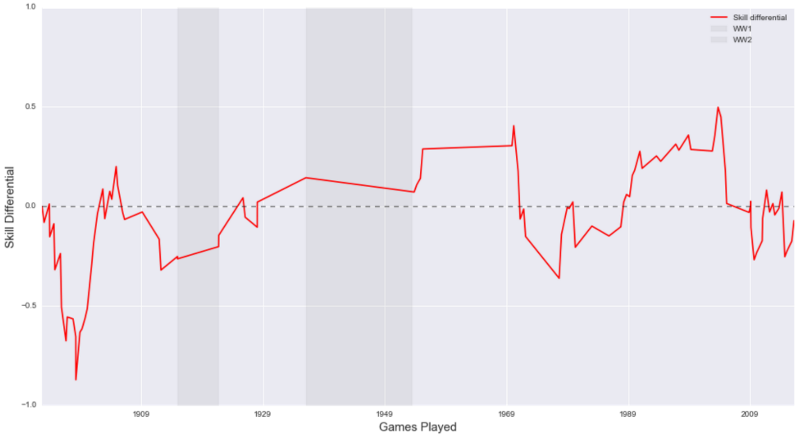


Derby Local Level

In [8]:
```
model_d = pf.GASReg("Derby ~ DerbyHome", data=eastmidlandsderby,family=pf.GASPoisson())
model_d.fit()
model_d.plot_fit(figsize=(17,8))
```



Relative Dominance (Above 0, Forest Superior)



- **Problem 1:** We've assumed equally spaced intervals between games.
- **Problem 2:** Dataset is head-to-head games; not a great insight into ability

GAS Dynamic Regression

- The local level model is a special case of GAS (**generalized autoregressive score**) dynamic regression:

$$p\left(y_t \mid \mu_t\right)$$

$$\mu_t = x_t^{'}\beta_t$$

$$\beta_t = \beta_{t-1} + \eta U_{t-1}$$

- We evolve the coefficients according to the score updates U_t
- The score is specific to the distribution of interest $p\left(y_t ; \psi\right)$

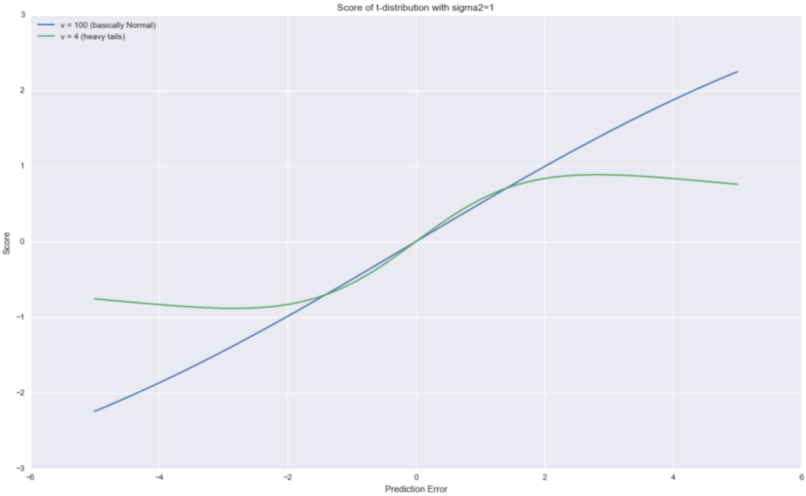
Robust Filters with the t-distribution score

- The score of the t-distribution has some nice properties:

$$S_t = \frac{\nu + 1}{\nu} \frac{X_t \left(y_t - X_t \beta_t\right)}{\sigma^2 + \frac{\left(y_t - X_t \beta_t\right)^2}{\nu}}$$

- As the degrees of freedom $\nu \rightarrow \infty$, the score becomes Gaussian
- For lower values of ν, the score 'trims' outliers (robustness)

Trimming Outliers - Harvey and Luati (2014)

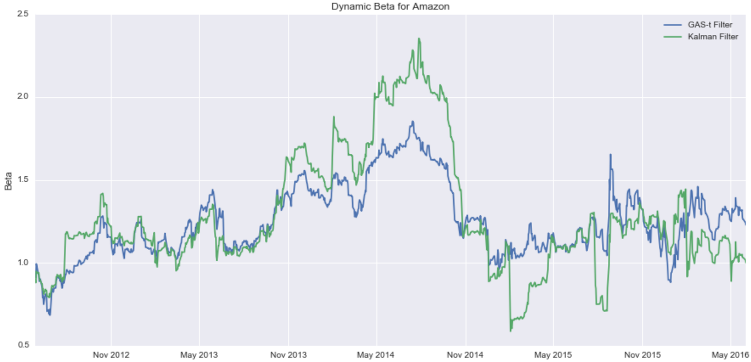


Robust Dynamic Betas for Finance

In [18]:
```
model3 = pf.GASReg('Amazon ~ SP500',data=final_returns,family=pf.GASt())
x = model3.fit()
model3.plot_fit(figsize=(15,10))
```

GAS-t Filter vs Kalman Filter

In [41]:
```
plt.figure(figsize=(15,7))
plt.title("Dynamic Beta for Amazon")
plt.ylabel("Beta")
plt.plot(model.index[100:],x2.states[1][100:-1],label="GAS-t Filter");
plt.plot(model.index[100:],states[1][100:-1],label="Kalman Filter");
plt.legend();
```



Summary

Score-driven state space models get us to the non-Gaussian world, but there are still some limitations:

- No simple way to obtain **smoothed estimates** in this setting
- **What are we actually approximating probabilistically?**
- How to bring **non-linearity** into play? Kernel trick? GPs?

Outline

Application: Modelling NFL outcomes

Precedents in the literature

- Glickman & Stern (2000): an NFL state space model
- Koopman & Lit (2015): a soccer state space model
- Peadar Coyle (2016) : Bayesian hierarchical model for rugby

Modelling Approach

There are more powerful modelling approaches nowadays than the aforementioned papers, but we'll take the same basic approach as these models are still insightful and fun:

- We'll rely on a **GLM framework** to model the NFL
- We'll have **time-varying team abilities**; we'll use a GAS model
- We'll focus on teams having a **power effect** on the game outcome

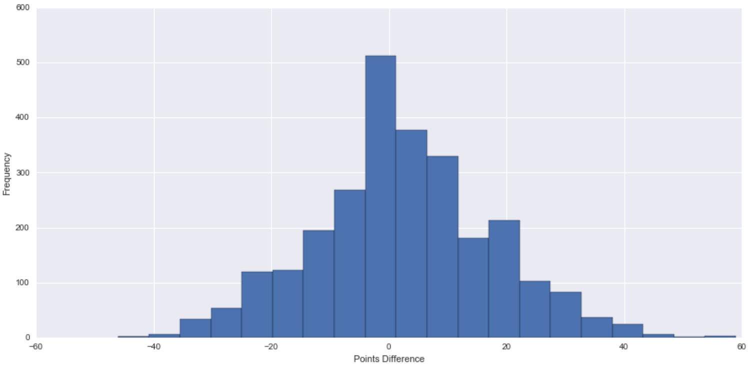
Data Summary

- ~ 2700 NFL games from 2006-2016.
- Information on home scores, away scores, teams, and quarterbacks
- Regular season and post-season games

We will focus on modelling the point difference.

The Data : Point Difference

```
In [15]: data = pd.read_csv("nfl_data_new.csv")
         data["PointsDiff"] = data["HomeScore"] - data["AwayScore"]
         plt.figure(figsize=(15,7))
         plt.ylabel("Frequency")
         plt.xlabel("Points Difference")
         plt.hist(data["PointsDiff"],bins=20);
```



The Model

We'll follow Glickman in modelling the point difference y_t as Normally distributed. We model the location of the point difference μ_t as:

$$\mu_t = \delta + \alpha_{t,i} - \alpha_{t,j}$$

- δ is a home advantage latent variable
- α contains the team abilities for the teams
- i and j reference the home and away teams in the match

The Model

Team abilities α are modelled as random walk processes:

$$\alpha_{k,i} = \alpha_{k-1,i} + \eta U_{k-1,i}$$

$$\alpha_{k,j} = \alpha_{k-1,j} - \eta U_{k-1,j}$$

- k is a team's game index
- U are the score-driven updates
- η is the scale (or learning rate)

Fitting the GASRank Model

```
In [3]: model = pf.GASRank(data=data,team_1="HomeTeam",team_2="AwayTeam",score_diff="PointsDiff", family=pf.GASNormal())
        x = model.fit()
        x.summary()
```

```
Normal GASRank
======================================================= ==================================================
Dependent Variable: PointsDiff                          Method: MLE
Start Date: 0                                           Log Likelihood: -10825.1703
End Date: 2667                                          AIC: 21656.3406
Number of observations: 2668                            BIC: 21674.0079
==========================================================================================================
Latent Variable                          Estimate   Std Error  z        P>|z|    95% C.I.
======================================== ========== ========== ======== ======== =========================
Constant                                 2.2405     0.2547     8.795    0.0      (1.7412 | 2.7398)
Ability Scale                            0.0637     0.0058     10.9582  0.0      (0.0523 | 0.0751)
Normal Scale                             13.9918
==========================================================================================================
```

Power Rankings

```
In [10]: model.plot_abilities(["Denver Broncos", "Green Bay Packers", "New England Patriots",
                              "Carolina Panthers"],figsize=(15,8))
```

Power Rankings: Californian Franchises

```
In [11]: model.plot_abilities(["San Francisco 49ers", "Oakland Raiders", "San Diego Chargers"],figsize=(15,8))
```

Extending the Model

Let's include a quarterback ability component γ:

$$\mu_t = \delta + \alpha_{t,i} - \alpha_{t,j} + \gamma_{t,i} - \gamma_{t,j}$$

- δ is a home advantage latent variable
- α contains the abilities for the teams
- γ contains the abilities for the quarterbacks
- i and j reference the home and away teams/quarterbacks in the match

We model team abilities α and QB abilities γ as random walk processes.

Fitting the Extended Model

In [12]:
```
model.add_second_component("HQB","AQB")
x = model.fit()
x.summary()
```

```
Normal GASRank
======================================================= ==================================================
Dependent Variable: PointsDiff                          Method: MLE
Start Date: 0                                           Log Likelihood: -10799.4544
End Date: 2667                                          AIC: 21606.9087
Number of observations: 2668                            BIC: 21630.4651
==========================================================================================================
Latent Variable                           Estimate   Std Error  z        P>|z|    95% C.I.
========================================= ========== ========== ======== ======== =========================
Constant                                  2.2419     0.2516     8.9118   0.0      (1.7488 | 2.735)
Ability Scale 1                           0.0186     0.0062     2.9904   0.0028   (0.0064 | 0.0307)
Ability Scale 2                           0.0523     0.0076     6.8492   0.0      (0.0373 | 0.0673)
Normal Scale                              13.8576
==========================================================================================================
```

Superbowl 50 : QB Career History

```
In [20]: model.plot_abilities(["Cam Newton", "Peyton Manning"],1,figsize=(15,8))
```

* Note that Peyton Manning was playing before this dataset started in 2006

No. 1 Quarterback Draft Picks

Other G.O.A.T worthy candidates

```
In [27]: model.plot_abilities(["Aaron Rodgers", "Tom Brady", "Russell Wilson"],1,figsize=(15,8))
```

All Models are Wrong...

There are problems with this model that we should be aware of:

- Giving credit purely on points - richer datasets are available
- Should ideally split out attacking and defensive contribution
- Should account for wide receivers and offensive line for QB ratings

We also assumed a Gaussian distribution for the aggregatate point difference, but the nature of football is that **points are generated from different sources**: touchdowns, field goals, safeties.

Running; Or The Importance of a Good Offensive LIne

http://www.packers.com/news-and-events/article-game-recap/article-1/Aaron-Rodgers-beat-the-heat-of-the-moment/7d1effe4-8f74-4e94-b418-27f936485ef5

Superbowl 50: Defence Matters

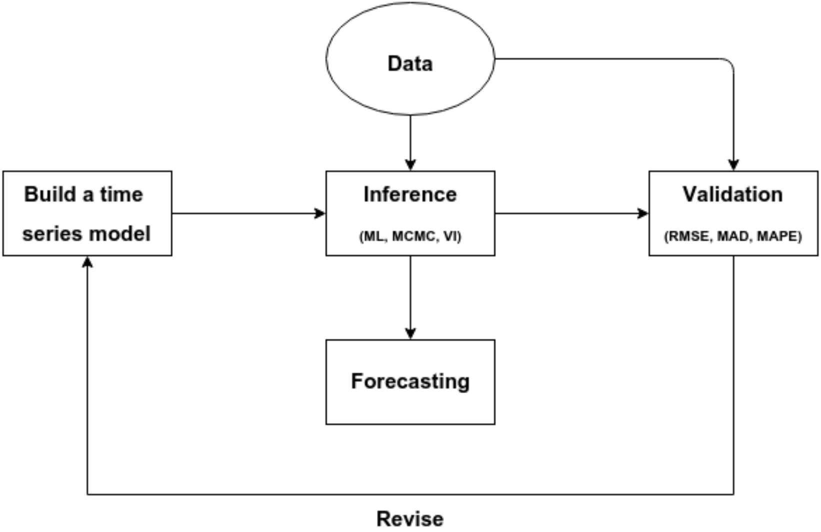
```
In [8]: model.predict("Denver Broncos","Carolina Panthers","Peyton Manning","Cam Newton",neutral=True)
Out[8]: array(-7.33759714587138)
```

Denver Broncos vs **Carolina Panthers**

- Basic Model predicted spread: +4.8 Panthers
- QB Model predicted spread: +7.3 Panthers
- Consensus spread: +5.5 Panthers
- Actual spread : +14 Broncos

Way off! It's one result, and the world's probabilistic, but you should still probably model the defence...

Your Turn to Revise



The End