notebook

Unknown Author

March 12, 2014

Part I

Measles in Small Populations

Based on *Finkenstadt and Grenfell (2002), Applied Statistics*, the paper introducing the TSIR model. The model relates S_t, the number of susceptibles at time t, to the number of infected cases I_t at time t :

$$I_{t+1} = r_t \, S_t \, I_t^{\alpha} \, \varepsilon_t,$$
$$S_{t+1} = B_{t-d} + S_t - I_t + u_t$$

where r_t is a positive seasonal contact rate parameter; B_t are the births during timestep t; ε_t and u_t are Gaussian noises with $\mathrm{E}[\varepsilon_t] = 1$ and $\mathrm{E}[u_t] = 0$; and $\alpha \in (0, 1]$ is a homogeneity parameter, with $\alpha = 1$ equivalent to mass action dynamics.

In [3]:
```
## IMPORTS

# Numerical packages and methods
import numpy as np
from sklearn import linear_model
from scipy.optimize import curve_fit
import scipy.stats as st
import scipy.interpolate as interp

# Monte Carlo and Nonlinear Fitting
import pymc
import lmfit

# Plotting
import matplotlib
from prettyplotlib import plt
import brewer2mpl
matplotlib.rcParams['savefig.dpi'] = 1.8 * matplotlib.rcParams['savefig.dp
figsize(14, 8)
colours = brewer2mpl.get_map('Set2', 'qualitative', 8).mpl_colors
#import seaborn

# Other
import itertools
```

Data

We have several import options. Of the four variables, `LONDON`, `ICELAND`, `FAROE`, and `BORNHOLM`, only one is allowed to be 1. This selects the source of data to import and analyse. If all are 0, we instead simulate an SIR model

at equilibrium with major biennial epidemics and minor epidemics in between - that is, measles in a large, well-mixed population. If more than one of these variables are 1, we import the first non-zero.

In [11]:
```
# Import what ?
LONDON = 0
ICELAND = 0
FAROE = 1
BORNHOLM = 0
```

Model Parameters

The `periodicity` of the time series imposes an infectious period. A periodicity of 24 implies the infectious period is 1/24 years, or 15.2 days. 24 is chosen to optimise the result of the interpolation of monthly-sampled time series, as we have for Iceland, the Faroe Islands, and Bornholm. London must be dealt with differently, due to its biweekly-sampled time series. Its periodicity is thus 26.

The `delay` is an estimate of the number of periods of infection during which newly-born susceptibles are immune to infection due to maternal antibodies. Here, we assume four months, or eight periods of about fifteen days (fourteen for London).

The model's `sensitivity` determines how many cases of measles are required for a given time point to be considered as being part of an epidemic.

Finally, `penalty` is used as a weighting parameter to avoid overfitting during susceptible reconstruction. This step is done using an arbitrary polynomial; `penalty` ensures that higher degree polynomials are not selected due to overfitting.

Interpolation

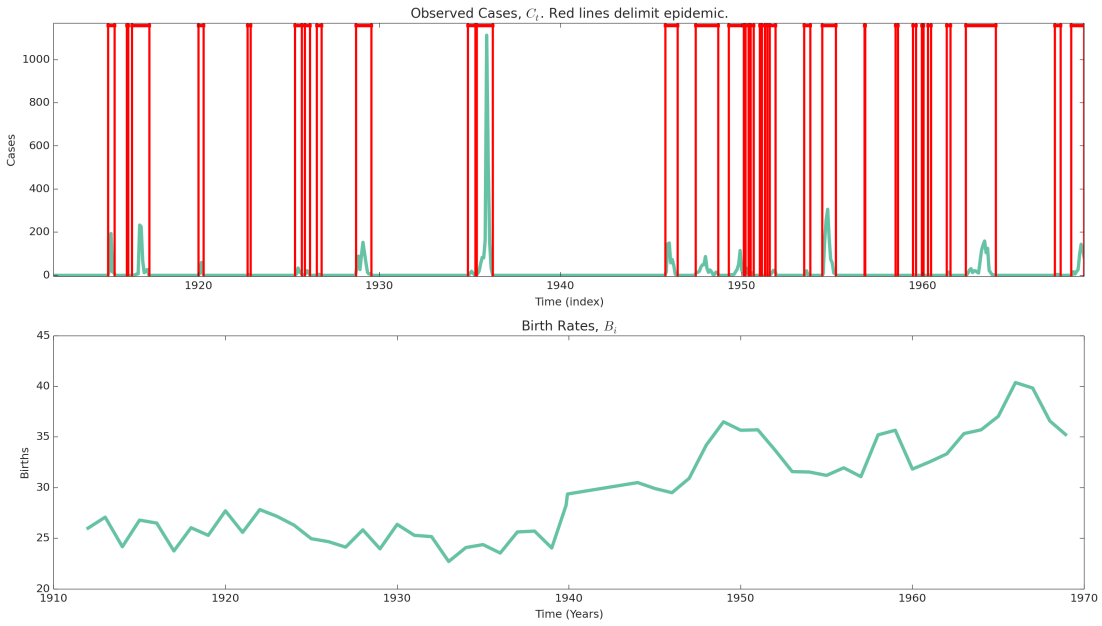
The incidence data for London is biweekly, and does not need to be interpolated. Incidence data for Iceland, the Faroe Islands, and Bornholm, are all monthly, and need to be interpolated. To avoid artifacts due to shifting timepoints in the series, we can either use splines, or use a number of timepoints that is a multiple of the current sampling. We opt for the latter here, as splines introduce a large number of oscillations in the derivatives, due to their nature as a piecewise cubic function.

Birth rates are interpolated in the same manner.

In [14]:
```
# Model parameters
periodicity = 24 if not LONDON else 26
delay = 8
sensitivity = 2
penalty = 5e-4

#if FAROE :
#    sensitivity = 25
#elif ICELAND :
#    sensitivity = 15



# Import data
if not (LONDON or ICELAND or FAROE or BORNHOLM) : # then simulate SIR dyna

    # Timestep, maximum time ( in periods ), and model-time array
    dt = .05
    tmax = 2000
    mt = np.arange(0, tmax, dt)


    # Population Parameters :
    N = 3e6 # total population
    mu = np.ones(int(tmax/dt),) * 1./(50.*periodicity)  # birth rate
    B = mu * N # births
```

```
    mrho = 0.5 # model's reporting rate


    # Disease Parameters :
    R0 = 18. # gives biennial dynamics
    gamma = 1. # just over two weeks
    beta0 = R0 * gamma / N # base contact rate
    amplitude = 0.1 # amplitude of contact rate seasonality
    beta = beta0 * ( np.ones(int(tmax/dt)) + amplitude * np.cos(2. * np.pi


    # Initialise arrays and Initial Conditions
    mS = np.zeros(int(tmax/dt))
    mI = np.zeros(int(tmax/dt))
    mR = np.zeros(int(tmax/dt))
    mS[0] = 0.02 * N
    mI[0] = 0.0005* N
    mR[0] = N - mS[0] - mI[0]


    # Run using Forward Euler method
    for i in range(1, len(mS)) :
        mS[i] = mS[i-1] + dt * (mu[i] * N - beta[i] * mS[i-1] * mI[i-1] -
        mI[i] = mI[i-1] + dt * (beta[i] * mS[i-1] * mI[i-1] - gamma * mI[i
        mR[i] = N - mS[i] - mI[i]


    # Outputs : interpolate result of SIR using cubic splines
    t = np.arange(tmax/4.*2.715, tmax) # arbitrary cutoff, just for a neat
    I = (interp.UnivariateSpline(mt, mI))(t)
    C = I * mrho
    S = (interp.UnivariateSpline(mt, mS))(t)
    B = (interp.UnivariateSpline(mt, B))(t)
    t = t / periodicity




# If we're not doing SIR,
# then import from data
else :
    if ICELAND :
        f = open("tsir_recon2.csv", "rU")
    elif LONDON :
        f = open("london.csv", "rU")
    elif FAROE :
        f = open("farmeas.csv", "rU")
    elif BORNHOLM :
        print "Bornholm"
        f = open("bornholm.csv", "rU")

    data = np.genfromtxt(f, delimiter=',')
    f.close()



    # Careful interpolation of spiky data

    # First, trim the data vector to keep an integer number of years
    data = np.delete(data, np.s_[np.where(data[:, 1] == np.floor(data[-1,

    # Define desired times : <periodicity> per year, to maintain 12-multip
    t = np.linspace(data[0, 1], data[-1, 1], np.ceil(data[-1, 1] - data[0,
```

```
    # Births (B) and Cases (C) use simple linear interpolation, except Lo
    B = np.interp(t, data[:, 1], data[:, 2])
    if not LONDON :
        B = B / periodicity # births were reported annually
    C = np.interp(t, data[:, 1], data[:, 0])
    if not LONDON :
        C = np.round(C / periodicity * 12) # cases were reported monthly;




# Where are the epidemics ?
epi = []

# If there are many zeros ( here, we say at least 50% ), we can cut epide
if (np.sum(C <= sensitivity).astype(float) / len(C)) > 0.5 :
    z = np.where(C > sensitivity)[0] # Find epidemics over sensitivity th
    dz = np.where(np.append(np.insert(np.diff(z), 0, 0), -1) != 1)[0]
    for i in range(len(dz)-1) :
        epi.append(z[dz[i]:dz[i+1]])

else : # Otherwise, slice at local minima using smoothed zero-crossings i
    z = range(len(C))
    z2 = np.diff(np.convolve(C, np.hanning(19), "same"))
    dz = np.append(np.insert((np.where((z2[:-1] < 0) * (z2[1:] > 0) == Tru
    for i in range(len(dz)-1) :
        epi.append(range(dz[i], dz[i+1]))

epi = np.array(epi)



# Plots
subplot(211)
plt.plot(t, C, linewidth=3)
for e in epi :
    axvline(t[e[0]], color="red", linewidth=2)
    axvline(t[e[-1]], color="red", linewidth=2)
    axhline(1.04*np.max(C), xmin=(t[e[0]]-t[0])/(t[-1]-t[0]), xmax=(t[e[-1
title("Observed Cases, $C_t$. Red lines delimit epidemic.")
xlim([t[0], t[-1]])
ylim([-5, 1.05*np.max(C)])
xlabel("Time (index)")
ylabel("Cases")

subplot(212)
title("Birth Rates, $B_i$")
xlabel("Time (Years)")
ylabel("Births")
plt.plot(t, B, linewidth=3)

tight_layout()
```



In [13]:
```
np.sum(B)
```

Out [13]: 40324.413507415979

Susceptible Reconstruction

We have already defined C, the observed cases, and B, the time-series of births. Reconstruction of the susceptible time-series requires

$$Y_t = \sum_{i=0}^{t} B_t,$$

and

$$X_t = \sum_{i=0}^{t} C_t.$$

This will enable us to find the true number of infected individuals : $I_t = \rho_t C_t$. Here, $\rho_t \geq 1$ is the linked to the reporting rate by $1/\rho_t$.

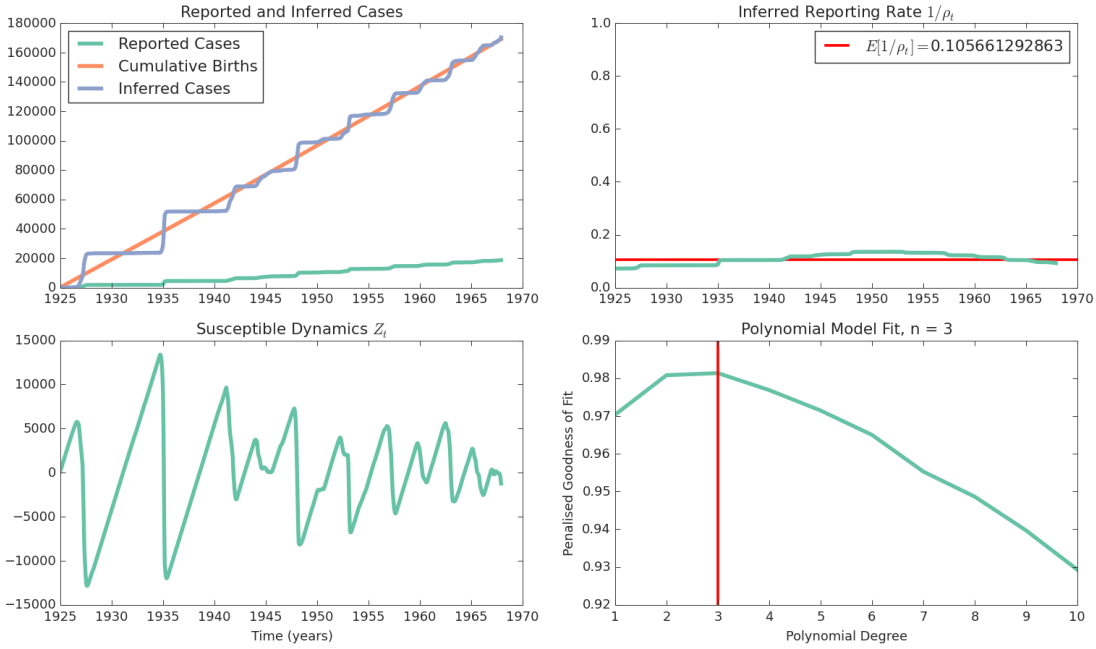
We expand the number of susceptibles around its mean, as $S_t = \bar{S} + Z_t$. Then, without repeating the results present in the paper, we find that

$$Y_t = -Z_0 + \rho_t X_t + Z_t.$$

For constant ρ, this parameter could be fit using a simple linear regression between the cumulative cases, X_t, and the cumulative births, Y_t. The residuals of regression would then yield Z_t, the dynamics of the susceptibles about their mean $\bar{S}$. When $\rho = \rho_t$ is allowed to vary, a *locally-linear* regression must be used; this can be implemented in many ways, such as by fitting piecewise-linear functions, splines, or the local polynomial regression technique described in Fan and Gibjels (1996), as used in the paper by Finkenstadt and Grenfell. Here, we use a polynomial, fitted by Bayesian ridge regression, to calculate $\hat{Y}_t$, the estimator for Y_t, and fit cubic splines to $\hat{Y}_t(X_t)$ to compute the slope of the function, and hence, to find ρ_t.

In [7]:
```
# Susceptible Reconstruction

# Compute cumulative births and incidence
Y = np.cumsum(B)
X = np.cumsum(C)


# Compute rho ( rate of reporting ) using Bayesian ridge regression with a
reg = linear_model.BayesianRidge(fit_intercept=False, compute_score=True)

# Compute the R^2 for a range of polynomials from degree-1 to degree-10
# The fit score has a penalty proportional to the square of the degree of
Ns = range(2, 12)
scores = []
for n in Ns :
    reg.fit(np.vander(X, n), Y)
    scores.append(reg.score(np.vander(X, n), Y) - penalty * n**2)

# Use the polynomial that maximised R^2 to compute Yhat
Yhat = reg.fit(np.vander(X, Ns[np.argmax(scores)]), Y).predict(np.vander(X

# Compute rho as the derivative of the splines that are fit between X and
rho = interp.UnivariateSpline(X, Yhat).derivative()(X)

# Compute Z as the residuals of regression
Z = Y - Yhat



# Plots
subplot(221)
plt.plot(t, X, linewidth=3)
plt.plot(t, Y, linewidth=3)
plt.plot(t, Yhat, linewidth=3)
title("Reported and Inferred Cases")
legend(["Reported Cases", "Cumulative Births", "Inferred Cases"], loc=2)

subplot(222)
axhline(1./np.mean(rho), color="r", linewidth=2)
plt.plot(t, 1./rho, linewidth=3)
ylim([0, 1])
title(r"Inferred Reporting Rate $1/\rho_t$")
legend([r"$E[1/\rho_t]=$" + str(1./np.mean(rho))])

subplot(223)
plt.plot(t, Z, linewidth=3)
title("Susceptible Dynamics $Z_t$")
xlabel("Time (years)")

subplot(224)
plt.plot(np.array(Ns)-1, scores, linewidth=3)
axvline(Ns[np.argmax(scores)]-1, color="r", linewidth=2)
title("Polynomial Model Fit, n = " + str(Ns[np.argmax(scores)]-1))
xlabel("Polynomial Degree")
ylabel("Penalised Goodness of Fit")
```

Out [7]: <matplotlib.text.Text at 0x10d30b110>



Parameter Inference

With the susceptible dynamics Z_t reconstructed, we can infer the contact rate, r_t, and the homogeneity parameter, α. To do this, we take logarithms of the main equations of the TSIR model :

$$\ln(I_{t+1}) = \ln(r_t) + \alpha \ln(I_t) + \ln(S_t),$$
$$\ln(S_{t+1}) = \ln(B_{t-d}) + \ln(S_t) - \ln(I_t),$$

after dropping the noise terms. We first impose that r_t be periodic with a period of one year; if we denote P as the number of time points in one year, then we impose that $r_{t+n\,P} = r_t \;\forall\; n \in \mathrm{Z}^+$. Instead of assuming a sinusoidal forcing, Finkenstadt and Grenfell allow r_t to be as general as possible, effectively making the periodic function into a series of P parameters $r_0, \ldots, r_{P-1}$.

Then, we must find a way to compute $\ln(S_t)$. We take a Taylor expansion, approximating this term as

$$\ln(S_t) = \ln(\bar{S} + Z_t) \approx \ln(\bar{S}) + \frac{Z_t}{\bar{S}}.$$

Substituting this into the previous equation for the number of infected individuals, we find that

$$\ln(I_{t+1}) = \ln(\bar{S}\, r_t) + \alpha \ln(I_t) + \frac{Z_t}{\bar{S}}.$$

Here, we use this approximation to infer the homogeneity parameter, α; the mean number of susceptibles, $\bar{S}$; and hence, the P contact rate parameters, r_t, having inferred $\bar{S}\, r_t$.

In [248]:

```
# EQUATION 15

# Fit a linear model to infer periodicity, alpha, and Sbar - using Z only

# Allocate design matrix
A = np.zeros((len(z)-1, periodicity+2))

# Periodicity indicators for the design matrix
for i in range(len(z)-1) :
    A[i, i % periodicity] = 1

# Set I(t-1), Z(t-1)
A[:, periodicity] = np.log(rho[z[:-1]] * C[z[:-1]])
A[:, periodicity+1] = Z[z[:-1]]



# Initialise results vector
y = np.log(rho[z[1:]] * C[z[1:]])



# Infer parameters using Bayesian ridge regression
reg2 = linear_model.BayesianRidge(fit_intercept=False)
reg2.fit(A, y)


# Extract useful parameters
rstar = np.exp(reg2.coef_[:periodicity]) # Sbar * r_t
alphaZ = reg2.coef_[periodicity] # alpha
zeta = reg2.coef_[periodicity+1] # Sbar

#plt.plot(rstar, linewidth=2)
#title("Periodicity")
print "Alpha = " + str(alphaZ)
print "Sbar = " + str(1./zeta)
if not (LONDON or ICELAND or FAROE or BORNHOLM) :
    print "Real Sbar = " + str(np.mean(S))
    print "Error in Sbar prediction = " + str(100*np.abs(1./zeta - np.mean
```

```
Alpha = 0.971414089617
Sbar = 8440.35832034
```

Parameter Inference 2

Instead of using the Taylor expansion of $\ln(\bar{S} + Z_t)$, we can estimate $\ln(\bar{S})$ by finding the value of $\bar{S}$ that maximises the likelihood of the main equation.

In [249]:
```
# EQUATION 12

# All possible values of Sbar
Svals = np.linspace(np.abs(np.min(Z))+1, np.abs(np.min(Z))*13, 100)


# Likelihood of fit
l = np.zeros(len(Svals))


# Define our parameters
params = lmfit.Parameters()
params.add("alpha", min=0.5, max=1., value=0.95) # Alpha
for i in range(periodicity) : # Seasonalities
    params.add("r" + str(i), value=0.)
rstr = ["r" + str(i % periodicity) for i in list(itertools.chain.from_iter
#if
```

```
# Objective function
def profile_residuals(params, rho, C, Z, z, Sestimate) :
    alphafit = params["alpha"].value
    r = [params[i].value for i in rstr]
    if isnan(Sestimate) :
        Sestimate = params["Sest"].value
    return alphafit * np.log(rho[z[:-1]]*C[z[:-1]]) + r + np.log(Sestimate
```

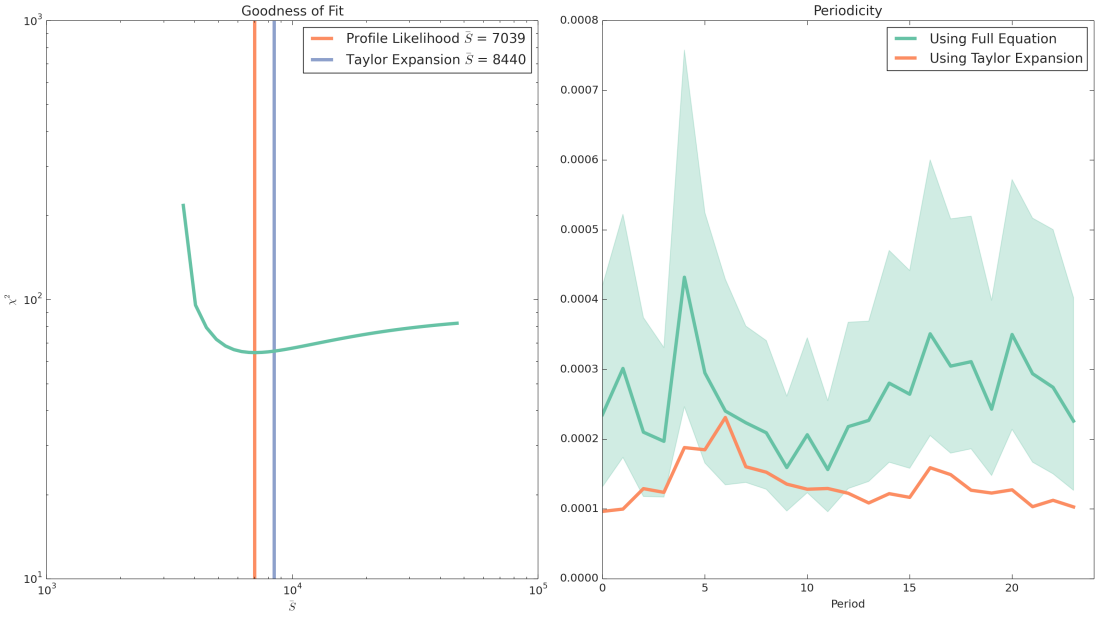
```
# Compute best fit for each possible Sbar
for i, Sestimate in enumerate(Svals) :
    l[i] = lmfit.minimize(profile_residuals, params, args=(rho, C, Z, z, S
```

```
# Fit window
fitwindow = 15
fitwindowL = np.min([fitwindow, np.argmin(l)])
fitwindowR = np.min([fitwindow, len(Svals) - np.argmin(l)])

# Run again using scan estimate
params.add("Sest", value = Svals[np.argmin(l)])
L = lmfit.minimize(profile_residuals, params, args=(rho, C, Z, z, np.nan),
```

```
# Extract parameters and errors
Sbar = L.params["Sest"].value
r = np.exp([L.params["r" + str(i)].value for i in range(periodicity)])
alphaSbar = L.params["alpha"].value
errup = np.exp(np.log(r) + [2*L.params["r" + str(i)].stderr for i in range
errdn = np.exp(np.log(r) - [2*L.params["r" + str(i)].stderr for i in range
```

```
# Plot
subplot(121)
plt.axvline(x=Sbar, color=colours[1], linewidth=3)
plt.axvline(x=1/zeta, color=colours[2], linewidth=3)
plt.loglog(Svals, l, linewidth=3)
title("Goodness of Fit")
xlabel(r"$\bar{S}$")
ylabel(r"$\chi^2$")
legend([r"Profile Likelihood $\bar{S}$ = " + str(int(Sbar)), r"Taylor Expa
```

```
subplot(122)
#plt.plot(rstar, linewidth=2)
plt.plot(r, linewidth=3)
plt.fill_between(range(periodicity), errup, errdn, color=colours[0], alpha
plt.plot(rstar * zeta, linewidth=3)
xlim([0, periodicity])
title("Periodicity")
xlabel("Period")
legend(["Using Full Equation", "Using Taylor Expansion"])
tight_layout()
```

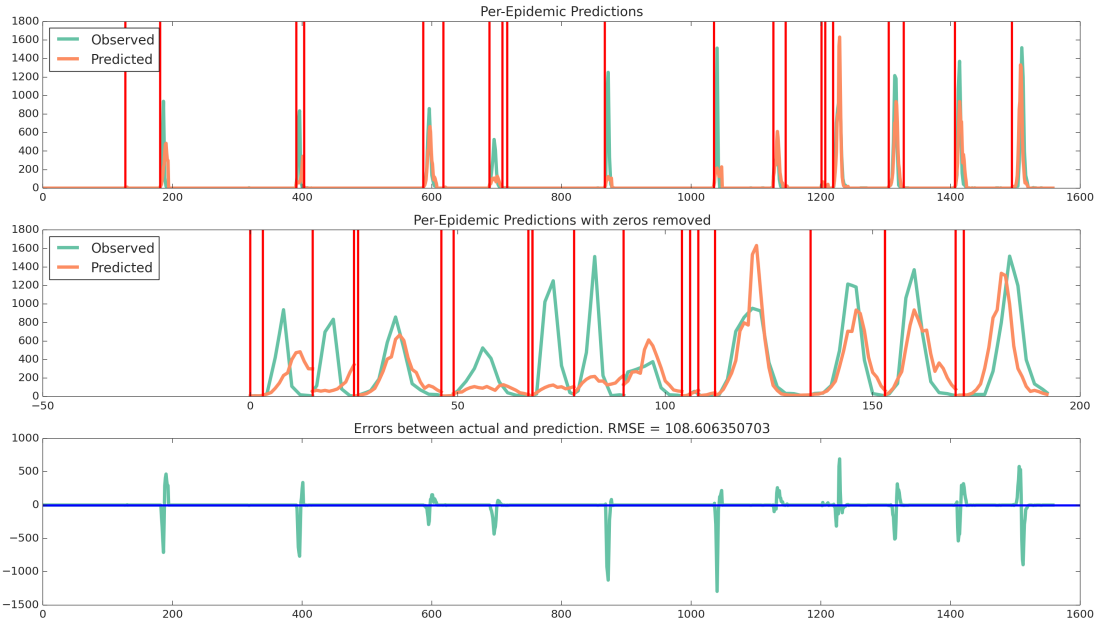


Predictions

Let τ_i^{start} be the time at which epidemic i begins, and τ_i^{end} the time at which the epidemic ends. We predict using the main equation in the paper, as first presented here, using our estimate for $S_{\tau_i^{\text{start}}} = \bar{S} + Z_{\tau_i^{\text{start}}}$ as the initial number of susceptibles, and $I_{\tau_i^{\text{start}}} = \rho_{\tau_i^{\text{start}}} C_{\tau_i^{\text{start}}}$ as the initial number of infected individuals for each epidemic i. We then predict the dynamics of the infected and susceptible individuals between τ_i^{start} and $\tau_{i+1}^{\text{start}}$.

In [250]:
```
# Initialise
predI = np.zeros_like(C)
predS = np.zeros_like(C)

# Seed initial epidemic points
for e in epi :
    predI[e[0]] = rho[e[0]] * C[e[0]]
    predS[e[0]] = Sbar + Z[e[0]]

    # Predict between epidemics
    for i in e[1:] :
        predI[i] = np.round(r[i % periodicity] * ( predI[i-1] ** alphaSbar
        predS[i] = B[max(i - delay, 0)] + predS[i-1] - predI[i]


# Plot
subplot(311)
plt.plot(C*rho, linewidth=3)
plt.plot(predI, linewidth=3)
for e in epi[:-1] :
    axvline(e[0], color="r", linewidth=2)
title("Per-Epidemic Predictions")
legend(["Observed", "Predicted"], loc=2)

subplot(312)
te = []
for e in epi :
    plt.plot(range(len(te), len(e)+len(te)), C[e] * rho[e], color=colours
    plt.plot(range(len(te), len(e)+len(te)), predI[e], color=colours[1], l
    axvline(len(te), color="r", linewidth=2)
    te = np.append(te, range(len(e)-1))
title("Per-Epidemic Predictions with zeros removed")
```

```
legend(["Observed", "Predicted"], loc=2)

subplot(313)
plt.plot(predI - C*rho, linewidth=3)
plt.axhline(np.mean(predI - C * rho), linewidth=2)
title("Errors between actual and prediction. RMSE = " + str(np.sqrt(np.sur
tight_layout()
```

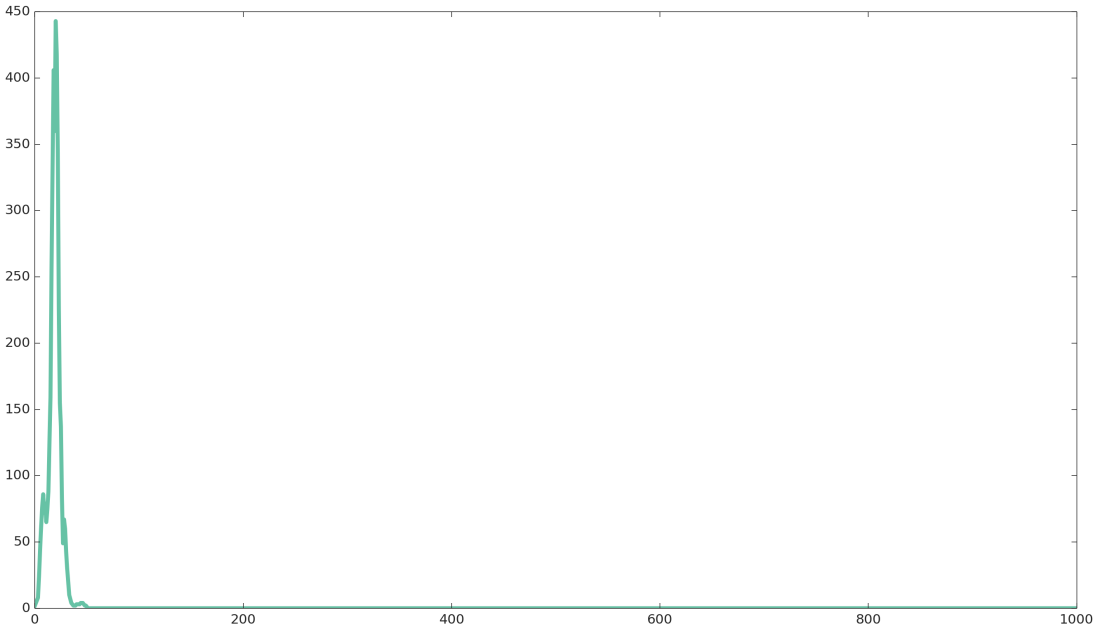


Predicted vs Recorded Epidemic Size

If the epidemic dynamic cannot be predicted accurately, perhaps the final size can ? We push prediction forward until the start of the next epidemic at time $\tau_{i+1}^{\text{start}}$, rather than ending at τ_i^{end}.

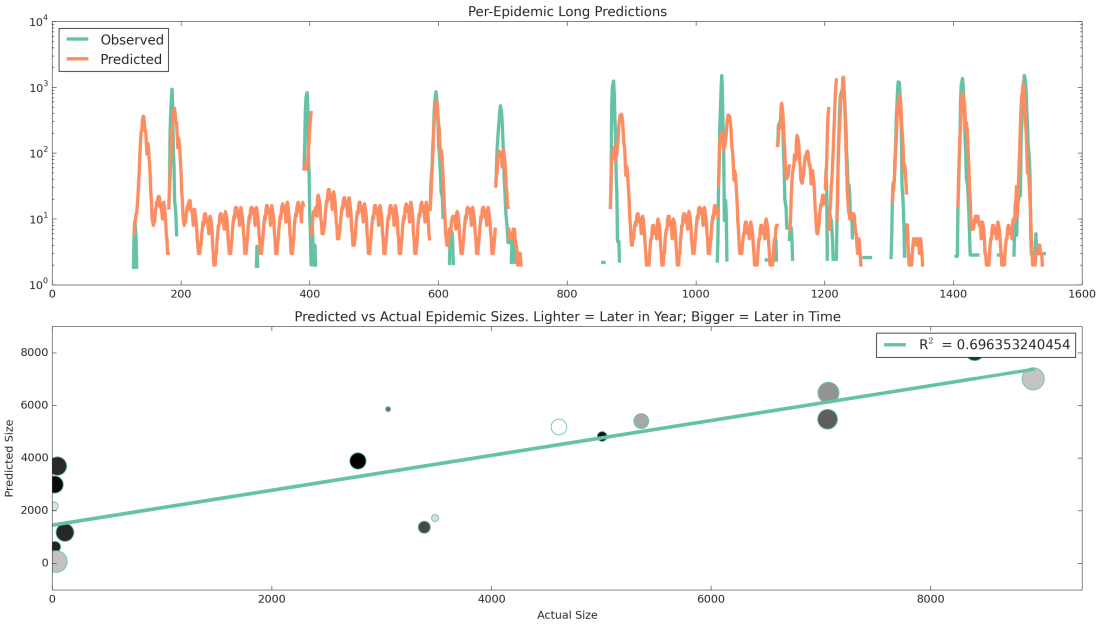
In [251]:
```
pp = np.zeros(1000)
sp = np.zeros(1000)
pp[0] = np.round(rho[0] * 1)
sp[0] = np.round(Sbar + Z[180])

for i in range(1, 1000) :
    pp[i] = np.floor(r[i % periodicity] * ( pp[i-1] ** alphaSbar ) * sp[i-
    sp[i] = np.floor(B[i] + sp[i-1] - pp[i])

plt.plot(pp, linewidth=3)
```

Out [251]:
```
[<matplotlib.lines.Line2D at 0x134d87710>]
```



In [253]:
```
# Find epidemic start times
starts = [e[0] for e in epi]
starts.append(len(rho)) # add final time
prI = []

# For each epidemic, predict until the time of the next epidemic
for i, time in enumerate(starts[:-1]) :
    # Seed the epidemic
    predI2 = np.zeros(starts[i+1] - time)
    predS2 = np.zeros(starts[i+1] - time)
    predI2[0] = np.round(rho[time] * C[time])
    predS2[0] = np.round(Sbar + Z[time])

    # Predict between epidemics
    for j in range(1, len(predI2)) :
        predI2[j] = np.round(r[(time + j) % periodicity] * ( predI2[j-1] +
        predS2[j] = np.round(B[max(j - delay, 0)] + predS2[j-1] - predI2[j

    # When the prediction dips below sensitivity, assume the epidemic is e
    if size(np.where(predI2 < sensitivity)[0]) :
        predI2[(np.where(predI2 < sensitivity)[0][0]):] = 0

    # Save result
    prI.append(predI2)


# Calculate epidemic sizes, both predicted and actual
actualsizes = np.array([np.sum(C[e] * rho[e]) for e in epi]).reshape(len(e
predictedsizes = [np.sum(pred) for pred in prI]

# Line of best fit and R^2
sizeline = linear_model.BayesianRidge(compute_score=True, fit_intercept=Tr
sizeline.fit(actualsizes.reshape(len(actualsizes),1), predictedsizes)


# Plot
subplot(211)
plt.plot(C*rho, linewidth=3)
for i, e in enumerate(epi) :
    plt.plot(range(e[0], e[0] + len(prI[i])) , prI[i], color=colours[1], l
```

```
#     axvline(len(te), color="r", linewidth=2)
#     te = np.append(te, range(len(e)-1))
title("Per-Epidemic Long Predictions")
legend(["Observed", "Predicted"], loc=2)



subplot(212)
plt.scatter(actualsizes, predictedsizes, s=20*np.array(range(len(actualsiz
plt.plot(actualsizes, sizeline.predict(actualsizes), linewidth=3)
plt.gray()
title("Predicted vs Actual Epidemic Sizes. Lighter = Later in Year; Bigger
xlabel("Actual Size")
ylabel("Predicted Size")
legend([r"R$^2$ = " + str(sizeline.score(actualsizes.reshape(len(actualsiz
xlim([0, 1.05*np.max(actualsizes)])

tight_layout()
```



Fraction Infected per Epidemic

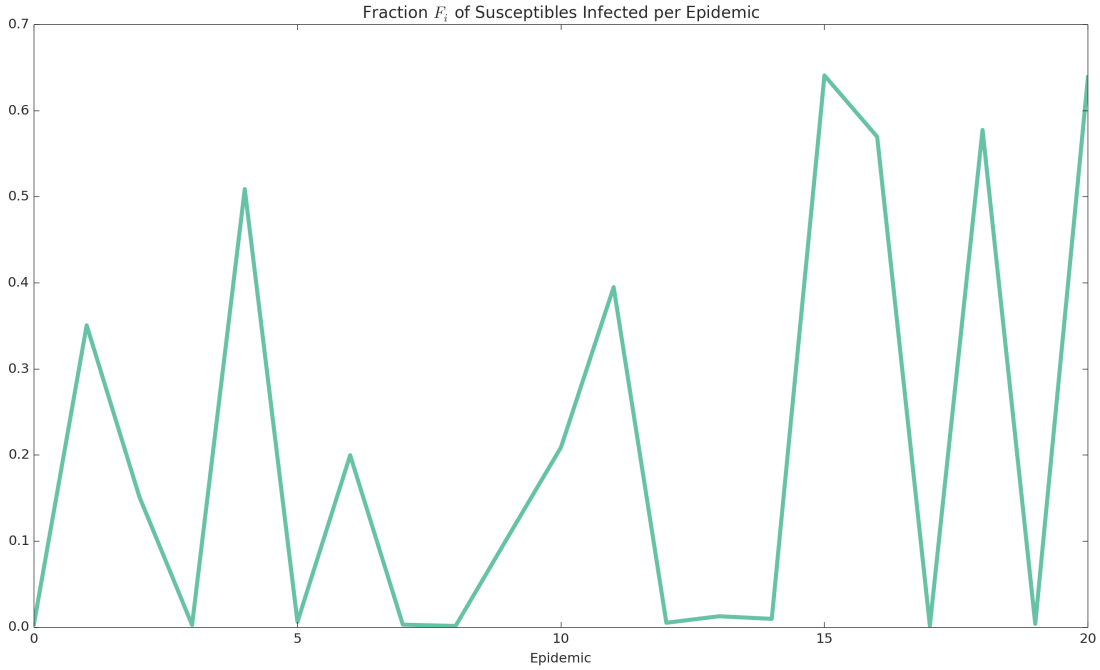
The fraction of the total susceptibles that get infected during an epidemic can be used as a diagnostic tool against certain numerical issues. This quantity should be between zero and one, and is defined for epidemic i as :

$$F_i = \frac{\int_{\tau_i^{\text{start}}}^{\tau_i^{\text{end}}} I_t \, \mathrm{d}t}{S_{\tau_i} + \int_{\tau_i^{\text{start}}}^{\tau_i^{\text{end}}} B_t \, \mathrm{d}t}.$$

The numerator here is the total number of infected individuals during the epidemic, and the denominator is the number of susceptibles at the beginning of the epidemic, plus the number of births that occur during the period of the epidemic.

Near-zeros may indicate spurious infections that do not lead to “real” epidemics. This may or may not imply that the sensitivity threshold may be too low.

In [254]:
```
# Calculate F for each epidemic
F = [np.sum(predI[e]) / (Sbar + Z[e[0]] + np.sum(B[e])).astype(float) for

# Plot
plt.plot(F, linewidth=3)
title(r"Fraction $F_i$ of Susceptibles Infected per Epidemic")
xlabel("Epidemic")

# For the SIR, we can calculate the true value of F by using actual S, an
if not (LONDON or ICELAND or FAROE or BORNHOLM) :
    F2 = [np.sum(predI[e]) / (S[e[0]] + np.sum(B[e])).astype(float) for e
    plt.plot(F2, linewidth=3)
    legend(["Using Predicted Susceptibles", "Using Real Susceptibles"])
```



In [233]:
```
```

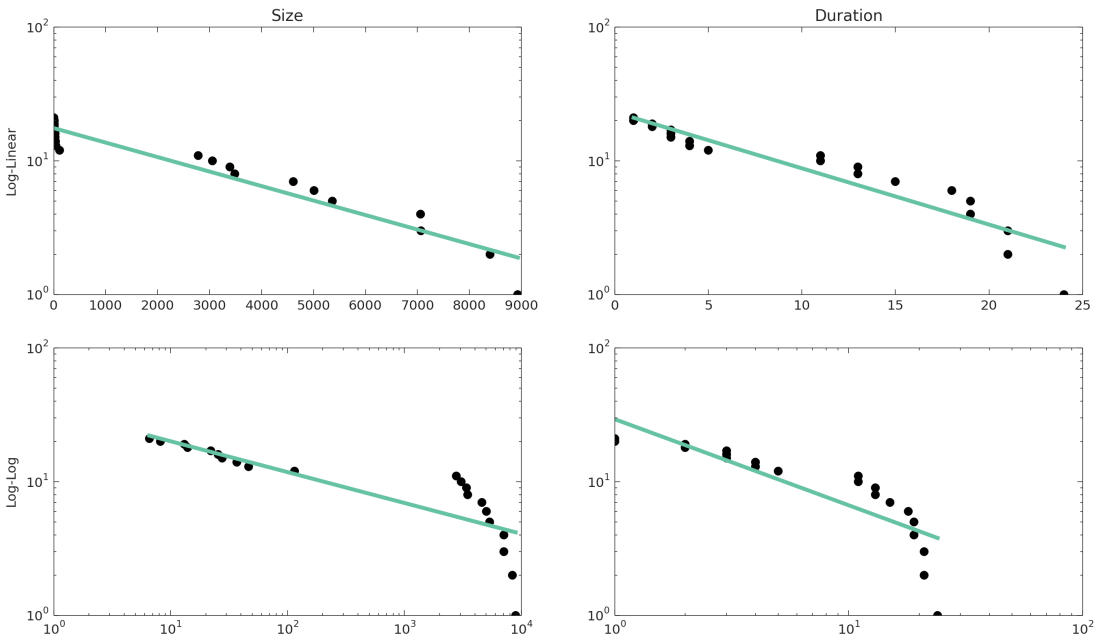
Scaling Relations

As in Rhodes and Anderson (1996) *Nature*, and Rhodes and Anderson (1996) *Proceedings of the Royal Society B*, we can analyse the scaling relation of the sizes and the durations of epidemics as a function of their frequencies. Rhodes and Anderson find power law (scale-free) for both size and duration; if that is the case, the bottom two plots here should be linear, as these are on a log-log scale. The top two plots are log-linear, and a straight line on these plots implies an exponential (scaled) relationship.

In [256]:
```
# Calculate size and duration of each epidemic
sizes = [np.sum(C[e] * rho[e]) for e in epi]
durations = [len(e) for e in epi]



# Sizes of Epidemics
xs = np.sort(sizes).reshape(len(sizes), 1)
ys = np.arange(len(xs))[::-1] + 1
lxs = np.log(xs)
lys = np.log(ys)
```

```
# Durations of Epidemics
xd = np.sort(durations).reshape(len(sizes), 1)
yd = np.arange(len(xd))[::-1] + 1
lxd = np.log(xd)
lyd = np.log(yd)



loglins = linear_model.BayesianRidge()
loglogs = linear_model.BayesianRidge()
loglind = linear_model.BayesianRidge()
loglogd = linear_model.BayesianRidge()

loglins.fit(xs, lys)
loglogs.fit(lxs, lys)
loglind.fit(xd, lyd)
loglogd.fit(lxd, lyd)



subplot(221)
plt.semilogy(xs, ys, "k.", ms=12)
plt.semilogy(xs, np.exp(loglins.predict(xs)), linewidth=3)
title("Size")
ylabel("Log-Linear")

subplot(222)
plt.semilogy(xd, yd, "k.", ms=12)
plt.semilogy(xd, np.exp(loglind.predict(xd)), linewidth=3)
title("Duration")

subplot(223)
plt.loglog(xs, ys, "k.", ms=12)
plt.loglog(np.exp(lxs), np.exp(loglogs.predict(lxs)), linewidth=3)
ylabel("Log-Log")

subplot(224)
plt.loglog(xd, yd, "k.", ms=12)
plt.loglog(np.exp(lxd), np.exp(loglogd.predict(lxd)), linewidth=3)

#plt.scatter(sizes, durations)
```

Out [256]: [<matplotlib.lines.Line2D at 0x138d7f990>]



TSIR Fitting on Simulated Data

We have all relevant parameters to simulate a new set of epidemics using the TSIR model, assuming small populations. If the above system is for a large population, this cell will do nothing.

In order to spark the epidemics, we need to infer the rate of importation of cases into the population. We will separate this into two processes : a low-mean Poisson rate of importation, which will determine the *timing* of the imports, and a beta distribution of import *sizes*.

As before, we're assuming that the epidemic goes extinct once it drops below sensitivity. If this is not assumed, we get very small oscillations and the disease looks to be endemic - the same happens if we fit a the TSIR to the observed data above and predict.

This cell runs most of the code above. After generating simulated data, it fits it and calculates predictions in the same manner. All plots are skipped, except the one of predicted vs actual epidemic size.

In [234]:

In [235]:
```
# If we're in a small population
if (np.sum(C <= sensitivity).astype(float) / len(C)) > 0.5 :

    # Let's generate a Poisson realisation with the same parameterisation
```

```
    # We'll make it longer than the previous series, to allow it to equil
    simpoi = np.array(st.poisson(float(len(epi)) / len(C)).rvs(len(C) * 2(



    # Next, let's find the distribution of import sizes, assumed to be exp
    # First, let's collect the size of each spark
    obsbeta = [C[i[0]] for i in epi]

    # Next, fit to an exponential using maximum likelihood
    # WARNING : the statistical power here is going to be dubious at best,
    betafit = st.beta.fit(obsbeta)

    # We can now generate some random variables to give sizes to our impo
    simbeta = st.beta(betafit[0], betafit[1], loc=betafit[2], scale=betafi



    # Putting them together :
    starts2 = np.where(simpoi == 1)[0]
    for i, j in enumerate(starts2) :
        simpoi[j] = simbeta[i] #/ 10.#np.mean(rho)

    subplot(311)
    plt.plot(simpoi[-10*len(C):], linewidth=3)
    title("Seeded Epidemic Sparks")


    # Let's predict a TSIR based on this series of imports
    # Initialise S, with the first value at Sbar
    simS = np.zeros_like(simpoi)
    simS[0] = float(Sbar)

    # We need to extrapolate births. Just for the first test, we'll just h
    BB = np.append(B, np.ones(len(B)*19) * B[-1])

    # Simulating ...
    for i in range(1, len(C)*20) :

        # If we're not forcing imports
        if i not in starts2 :
            simpoi[i] = r[i % periodicity] * ( simpoi[i-1] ** alphaSbar )

            if simpoi[i] < sensitivity :
                simpoi[i] = 0

        simS[i] = BB[max(i - delay, 0)] + simS[i-1] - simpoi[i]

    """
    # Plots
    subplot(312)
    plt.plot(simpoi[-len(C):], linewidth=3)
    title("Simulated Epidemics After Equilibration")

    subplot(313)
    plt.plot(simS[-len(C):], linewidth=3)
    axhline(Sbar, color="red", linewidth=2)
    title("Susceptibles")
    legend(["Susceptibles", "Original Sbar"])

    tight_layout()
    """



    # With the data generated, let's fit a TSIR to it ( discarding the fi
    # We should fine perfect reporting, so rho = 1
```

```
    D = simpoi[len(simpoi)/2:]

    # Where are the epidemics ?
    epis = []

    # If there are many zeros ( here, we say at least 50% ), we can cut ep
    if (np.sum(D <= sensitivity).astype(float) / len(D)) > 0.5 :
        zs = np.where(D > sensitivity)[0] # Find epidemics over sensitivit
        dzs = np.where(np.append(np.insert(np.diff(zs), 0, 0), -1) != 1)[0
        for i in range(len(dzs)-1) :
            epis.append(zs[dzs[i]:dzs[i+1]])

    else : # Otherwise, slice at local minima using smoothed zero-crossing
        zs = range(len(simpoi))
        z2s = np.diff(np.convolve(D, np.hanning(19), "same"))
        dzs = np.append(np.insert((np.where((z2s[:-1] < 0) * (z2s[1:] > 0)
        for i in range(len(dzs)-1) :
            epis.append(range(dzs[i], dzs[i+1]))

    epis = np.array(epis)



    Ys = np.cumsum(BB[-len(D):])
    Xs = np.cumsum(D)


    # Compute rho ( rate of reporting ) using Bayesian ridge regression wi
    regs = linear_model.BayesianRidge(fit_intercept=False, compute_score=T

    # Compute the R^2 for a range of polynomials from degree-1 to degree-1
    # The fit score has a penalty proportional to the square of the degree
    Ns = range(2, 12)
    scores = []
    for n in Ns :
        reg.fit(np.vander(Xs, n), Ys)
        scores.append(reg.score(np.vander(Xs, n), Ys) - penalty * n**2)

    # Use the polynomial that maximised R^2 to compute Yhat
    Yhats = reg.fit(np.vander(Xs, Ns[np.argmax(scores)]), Ys).predict(np.v

    # Compute rho as the derivative of the splines that are fit between X
    rhos = interp.UnivariateSpline(Xs, Yhats).derivative()(Xs)

    # Compute Z as the residuals of regression
    Zs = Ys - Yhats


    """
    # Plots
    subplot(221)
    plt.plot(Xs, linewidth=3)
    plt.plot(Ys, linewidth=3)
    plt.plot(Yhats, linewidth=3)
    title("Reported and Inferred Cases")
    legend(["Reported Cases", "Cumulative Births", "Inferred Cases"], loc=

    subplot(222)
    axhline(1./np.mean(rhos), color="r", linewidth=2)
    plt.plot(1./rhos, linewidth=3)
    ylim([0, 1])
    title(r"Inferred Reporting Rate $1/\rho_t$")
    legend([r"$E[1/\rho_t]=$" + str(1./np.mean(rhos))])

    subplot(223)
    plt.plot(Zs, linewidth=3)
    title("Susceptible Dynamics $Z_t$")
    xlabel("Time (years)")
```

```
subplot(224)
plt.plot(np.array(Ns)-1, scores, linewidth=3)
axvline(Ns[np.argmax(scores)]-1, color="r", linewidth=2)
title("Polynomial Model Fit, n = " + str(Ns[np.argmax(scores)]-1))
xlabel("Polynomial Degree")
ylabel("Penalised Goodness of Fit")
"""




# EQUATION 15

# Fit a linear model to infer periodicity, alpha, and Sbar - using Z (

# Allocate design matrix
As = np.zeros((len(zs)-1, periodicity+2))

# Periodicity indicators for the design matrix
for i in range(len(zs)-1) :
    As[i, i % periodicity] = 1

# Set I(t-1), Z(t-1)
As[:, periodicity] = np.log(rhos[zs[:-1]] * D[zs[:-1]])
As[:, periodicity+1] = Zs[zs[:-1]]



# Initialise results vector
ys = np.log(rhos[zs[1:]] * D[zs[1:]])



# Infer parameters using Bayesian ridge regression
reg2s = linear_model.BayesianRidge(fit_intercept=False)
reg2s.fit(As, ys)


# Extract useful parameters
rstars = np.exp(reg2s.coef_[:periodicity]) # Sbar * r_t
alphaZs = reg2s.coef_[periodicity] # alpha
zetas = reg2s.coef_[periodicity+1] # Sbar

#plt.plot(rstar, linewidth=2)
#title("Periodicity")
print "Alpha = " + str(alphaZs)
print "Sbar = " + str(1./zetas)
print "Real Sbar = " + str(Sbar)



# EQUATION 12

# All possible values of Sbar
Svalss = np.linspace(np.abs(np.min(Zs))+1, np.abs(np.min(Zs))*13, 100)


# Likelihood of fit
ls = np.zeros(len(Svals))


# Define our parameters
paramss = lmfit.Parameters()
paramss.add("alpha", min=0.5, max=1., value=0.95) # Alpha
for i in range(periodicity) : # Seasonalities
```

```
        paramss.add("r" + str(i), value=0.)
    rstrs = ["r" + str(i % periodicity) for i in list(itertools.chain.fro
    #if

    # Objective function
    def profile_residuals(params, rho, C, Z, z, Sestimate) :
        alphafit = params["alpha"].value
        r = [params[i].value for i in rstrs]
        if isnan(Sestimate) :
            Sestimate = params["Sest"].value
        return alphafit * np.log(rho[z[:-1]]*C[z[:-1]]) + r + np.log(Sesti









    # Compute best fit for each possible Sbar
    for i, Sestimate in enumerate(Svalss) :
        ls[i] = lmfit.minimize(profile_residuals, paramss, args=(rhos, D,


    # Fit window
    fitwindow = 15
    fitwindowL = np.min([fitwindow, np.argmin(ls)])
    fitwindowR = np.min([fitwindow, len(Svalss) - np.argmin(ls)])

    # Run again using scan estimate
    paramss.add("Sest", value = Svalss[np.argmin(ls)])
    Ls = lmfit.minimize(profile_residuals, paramss, args=(rhos, D, Zs, zs,


    # Extract parameters and errors
    Sbars = Ls.params["Sest"].value
    rs = np.exp([Ls.params["r" + str(i)].value for i in range(periodicity)
    alphaSbars = Ls.params["alpha"].value
    errups = np.exp(np.log(rs) + [2*Ls.params["r" + str(i)].stderr for i i
    errdns = np.exp(np.log(rs) - [2*Ls.params["r" + str(i)].stderr for i i









    """
    # Plot
    subplot(121)
    plt.axvline(x=Sbars, color=colours[1], linewidth=3)
    plt.axvline(x=1/zetas, color=colours[2], linewidth=3)
    plt.loglog(Svalss, ls, linewidth=3)
    title("Goodness of Fit")
    xlabel(r"$\bar{S}$")
    ylabel(r"$\chi^2$")
    legend([r"Profile Likelihood $\bar{S}$ = " + str(int(Sbars)), r"Taylo


    subplot(122)
    #plt.plot(rstar, linewidth=2)
    plt.plot(rs, linewidth=3)
    plt.fill_between(range(periodicity), errups, errdns, color=colours[0],
    plt.plot(rstars * zetas, linewidth=3)
    plt.plot(r, linewidth=3)
    plt.fill_between(range(periodicity), errup, errdn, color=colours[2], a
    xlim([0, periodicity])
    title("Periodicity")
    xlabel("Period")
    legend(["Using Full Equation", "Using Taylor Expansion", "Real Period
```

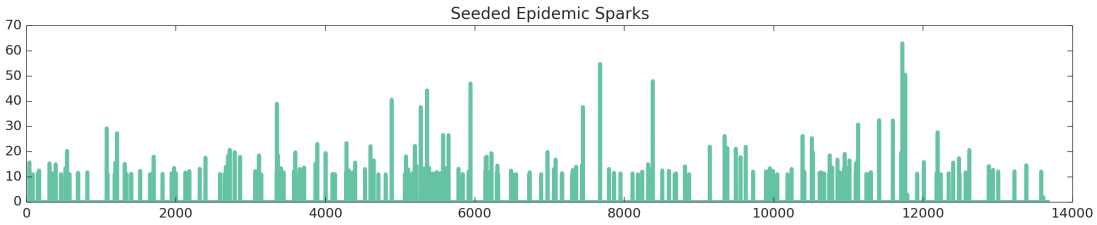
```
tight_layout()
"""




# And finally, predict

# Initialise
predIs = np.zeros_like(D)
predSs = np.zeros_like(D)
B2 = BB[-len(D):]

# Seed initial epidemic points
for e in epis :
    predIs[e[0]] = rhos[e[0]] * D[e[0]]
    predSs[e[0]] = Sbars + Zs[e[0]]

    # Predict between epidemics
    for i in e[1:] :
        predIs[i] = rs[i % periodicity] * ( predIs[i-1] ** alphaSbars
        predSs[i] = B2[max(i - delay, 0)] + predSs[i-1] - predIs[i]

"""
# Plot
subplot(311)
plt.plot(D*rhos, linewidth=3)
plt.plot(predIs, linewidth=3)
for e in epis[:-1] :
    axvline(e[0], color="r", linewidth=2)
title("Per-Epidemic Predictions")
legend(["Observed", "Predicted"], loc=2)

subplot(312)
te = []
for e in epis :
    plt.plot(range(len(te), len(e)+len(te)), D[e] * rhos[e], color=co
    plt.plot(range(len(te), len(e)+len(te)), predIs[e], color=colours
    axvline(len(te), color="r", linewidth=2)
    te = np.append(te, range(len(e)-1))
title("Per-Epidemic Predictions with zeros removed")
legend(["Observed", "Predicted"], loc=2)

subplot(313)
plt.plot(predIs - D*rhos, linewidth=3)
plt.axhline(np.mean(predIs - D * rhos), linewidth=2)
title("Errors between actual and prediction. RMSE = " + str(np.sqrt(np
tight_layout()
"""




# Find epidemic start times
startss = [e[0] for e in epis]
startss.append(len(rhos)) # add final time
prIs = []

# For each epidemic, predict until the time of the next epidemic
for i, time in enumerate(startss[:-1]) :
    # Seed the epidemic
    predI2s = np.zeros(startss[i+1] - time)
    predS2s = np.zeros(startss[i+1] - time)
    predI2s[0] = rhos[time] * D[time]
    predS2s[0] = Sbars + Zs[time]

    # Predict between epidemics
    for j in range(1, len(predI2s)) :
```

```
            predI2s[j] = rs[(time + j) % periodicity] * ( predI2s[j-1] **
            predS2s[j] = B2[max(j - delay, 0)] + predS2s[j-1] - predI2s[j]

        # When the prediction dips below sensitivity, assume the epidemic
        if size(np.where(predI2s < sensitivity)[0]) :
            predI2s[(np.where(predI2s < sensitivity)[0][0]):] = 0

        # Save result
        prIs.append(predI2s)


    # Calculate epidemic sizes, both predicted and actual
    actualsizess = np.array([np.sum(D[e] * rhos[e]) for e in epis]).reshap
    predictedsizess = [np.sum(pred) for pred in prIs]

    # Line of best fit and R^2
    sizelines = linear_model.BayesianRidge(compute_score=True, fit_interce
    sizelines.fit(actualsizess.reshape(len(actualsizess),1), predictedsize


    # Plot
    subplot(211)
    plt.plot(D*rhos, linewidth=3)
    for i, e in enumerate(epis) :
        plt.plot(range(e[0], e[0] + len(prIs[i])) , prIs[i], color=colours
    #    axvline(len(te), color="r", linewidth=2)
    #    te = np.append(te, range(len(e)-1))
    title("Per-Epidemic Long Predictions")
    legend(["Observed", "Predicted"], loc=2)



    subplot(212)
    plt.scatter(actualsizess, predictedsizess)#, s=20*np.array(range(len(a
    plt.plot(actualsizess, sizelines.predict(actualsizess), linewidth=3)
#    plt.gray()
    title("Predicted vs Actual Epidemic Sizes. Lighter = Later in Year; Bi
    xlabel("Actual Size")
    ylabel("Predicted Size")
    legend([r"R$^2$ = " + str(sizelines.score(actualsizess.reshape(len(act
    xlim([0, 1.05*np.max(actualsizess)])

    tight_layout()
```

```
---------------------------------------------------------------------
-------------
    IndexError                                Traceback (most recent
call last)



        <ipython-input-235-c47e84a75c37> in <module>()
        165
        166     # Set I(t-1), Z(t-1)
    --> 167     As[:, periodicity] = np.log(rhos[zs[:-1]] *
D[zs[:-1]])
        168     As[:, periodicity+1] = Zs[zs[:-1]]
        169
```

```
IndexError: index 13670 is out of bounds for size 13670
```

```
/Users/qcaudron/anaconda/lib/python2.7/site-
packages/scipy/stats/distributions.py:2395: DeprecationWarning: using
a non-integer number instead of an integer will result in an error in
the future
  return mtrand.beta(a,b,self._size)
/Users/qcaudron/anaconda/lib/python2.7/site-
packages/numpy/core/fromnumeric.py:218: DeprecationWarning: using a
non-integer number instead of an integer will result in an error in
the future
  return reshape(newshape, order=order)
```



In []:
```
plt.plot(zz[-len(C):])
```

In []:
```

```

In []:
```
xax = np.array([predSs[e[0]] for e in epis]).reshape(len(actualsizess),1)

sizelines2 = linear_model.BayesianRidge(compute_score=True, fit_intercept=
sizelines3 = linear_model.BayesianRidge(compute_score=True, fit_intercept=
sizelines2.fit(xax, actualsizess)
sizelines3.fit(xax, predictedsizess)

plt.scatter(xax, actualsizess, color=colours[2])
plt.scatter(xax, predictedsizess, color=colours[1])
plt.plot(xax, sizelines2.predict(xax), linewidth=3, color=colours[2])
plt.plot(xax, sizelines3.predict(xax), linewidth=3, color=colours[1])

legend([r"Actual Sizes, $R^2$ = " + str(sizelines2.score(xax, actualsizess
xlabel(r"Initial Susceptibles, $S_0$")
ylabel("Size of Epidemic")
title("Epidemic Sizes vs Initial Susceptible Number")
```

In []:
```

```

In []:
```
# OLD CODE

"""

# Run some R nastiness to obtain the most likely Sbar as well as periodic
# This is currently being replaced by some Python magic



# Write a dataframe to file
d = {}
```

```
d["logIt"] = np.log(rho[z[1:]] * C[z[1:]])
d["Z"] = Z[z[:-1]]
d["logIt1"] = np.log(rho[z[:-1]] * C[z[:-1]])
d["r"] = np.tile(range(int(periodicity)), 100)[z[1:]]

df = DataFrame(d)
df.to_csv("dataframe.csv")



# Call R, load dataframe, run a generalised linear model fit
ro.r('d <- read.csv("dataframe.csv")')
ro.r('S <-seq(abs(min(d$Z))+1, abs(min(d$Z))*13, by=500)')
ro.r('l <- rep(NA, length(S))')
ro.r('d$r<-as.factor(d$r)')

ro.r("for(i in 1:length(S)) { \
                                    logS <- log(S[i] + d$Z);      \
                                    m <- glm( d$logIt ~ -1 + d$r + d$logIt1 +
                                    l[i] = m$deviance     \
                                }")
ro.r("m2 <- glm( d$logIt ~ -1 + d$r + d$logIt1 + offset(log(S[l == min(l)]
ro.r("ci <- confint(m2)")



# Extract variables of interest from R
L = -np.array(ro.r["l"])
coefs = np.array(ro.r("coef(m2)"))
Svals = np.array(ro.r["S"])
Sbar = Svals[np.argmax(L)]
r = np.exp(coefs[:periodicity])
alphaSbar = coefs[periodicity]
ci = np.exp(ro.r["ci"])



# Plot
subplot(121)
plt.axvline(x=Sbar, color=colours[1], linewidth=2)
plt.axvline(x=1/zeta, color=colours[2], linewidth=2)
plt.plot(Svals, np.exp(L), linewidth=3)
title("Likelihood landscape vs Sbar")
legend([r"Profile Likelihood $\bar{S}$ = " + str(int(Sbar)), r"Taylor Expa

subplot(122)
#plt.plot(rstar, linewidth=2)
plt.plot(r, linewidth=3)
plt.fill_between(range(periodicity), ci[:-1, 0], ci[:-1, 1], color=colours
plt.plot(np.roll(rstar * zeta, 1), linewidth=3)
xlim([0, periodicity])
title("Periodicity")
legend(["Using Full Equation", "Using Taylor Expansion"])
"""
```

In []:
```
# What about inferring using pure MCMC ?
# Let's infer from Equation 12

## CURRENTLY COMMENTED OUT

"""
with pymc.Model() as model :

    # The prior on Sbar can be informed from earlier calculation
```

```
    SbarMC = pymc.Normal("SbarMC", Sbar, 1e-10)

    # Alpha
    alphaMC = pymc.Beta("alphaMC", 2., 1.)

    # r
    rMC = pymc.Normal("rMC", -12., 0.3, shape=periodicity)

    EIMC = np.tile(rMC, 2*len(C)/periodicity)[:len(rho[:-1])] + alphaMC *


    # Expected cases
    @pymc.deterministic
    def EIMC(r = rMC, a = alphaMC, Sbar = SbarMC, It = np.log(rho * C), Z
        out = np.zeros_like(It)
        out[0] = It[0]
        for i in range(1, len(It)) :
            out[i] = r[i % periodicity] + a * It[i-1] + np.log(Sbar + Z[i
        return out


    # Observed cases
    IMC = pymc.Normal("IMC", EIMC, 1./np.var(np.log(rho*C)), observed=True

    start = find_MAP() # Find starting value by optimization
    step = NUTS(state=start) # Instantiate MCMC sampling algorithm
    trace = sample(2000, step, start=start, progressbar=False) # draw 200


    #model = pymc.Model((SbarMC, alphaMC, rMC, EIMC, IMC))
    #mcmc = pymc.MCMC(model)
    #mcmc.sample(200000)
"""
```

In []:

In []: