Outline of Workshop

First thing: Introduction to Jupyter Notebooks

Morning (pre-break):

- Correlations arising from space-time properties: 'autocorrelation'
- Analysis of El Nino time series

Morning (post-break):

- Time series analysis of Orange County ozone

Afternoon:

- Extension of time series ideas to spatial analysis
- Hands-on analysis of Oregon climate station data

Statistical prerequisites: at least one statistics course or some experience statistically analyzing data

Notes on material:

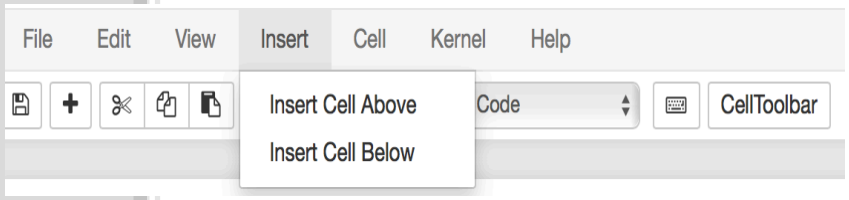
1) Basic (linear) statistical models to demonstrate concepts
2) Combination of theory, software, hands on data analysis
3) Learn concepts using time series and then generalize to space
4) Run code as I present it in class, with some longer hands on exercises

Introduction to Jupyter

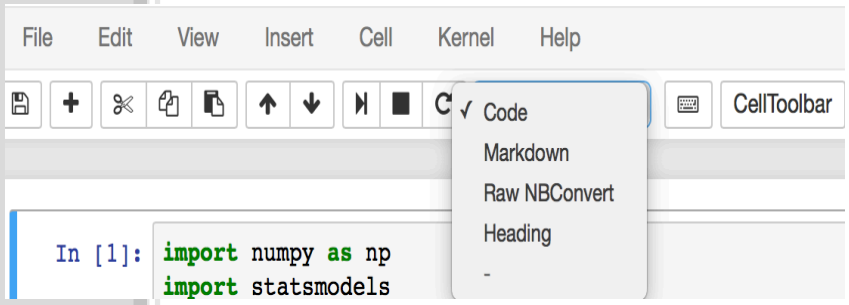
- “Interacting Computing Environment” allowing code, plots, texts, equations, etc. to be created and shared
- Requirements:
  - Python 2.7 (even if working in R)
  - Web application (Jupyter Notebook app in browser)
  - Language-specific Kernel (R or Python)

How to use Jupyter Notebooks

- Everything goes inside ‘cells’. Create a cell:

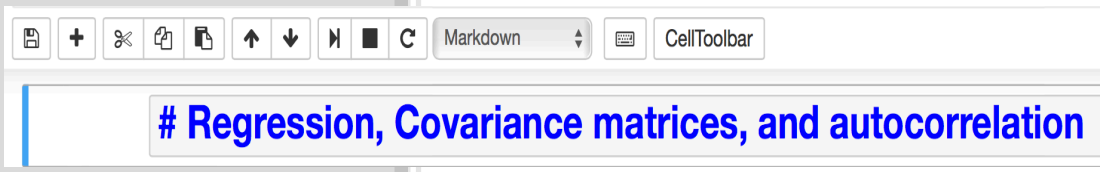


- Determine cell type:
  - Markdown for text, headers, equations;
  - Code for Python or R code that needs to be run (i.e. produces output).



How to use Jupyter Notebooks (cont'd)

- Convert Markdown to Header by simply using a # before the text and then run the cell:

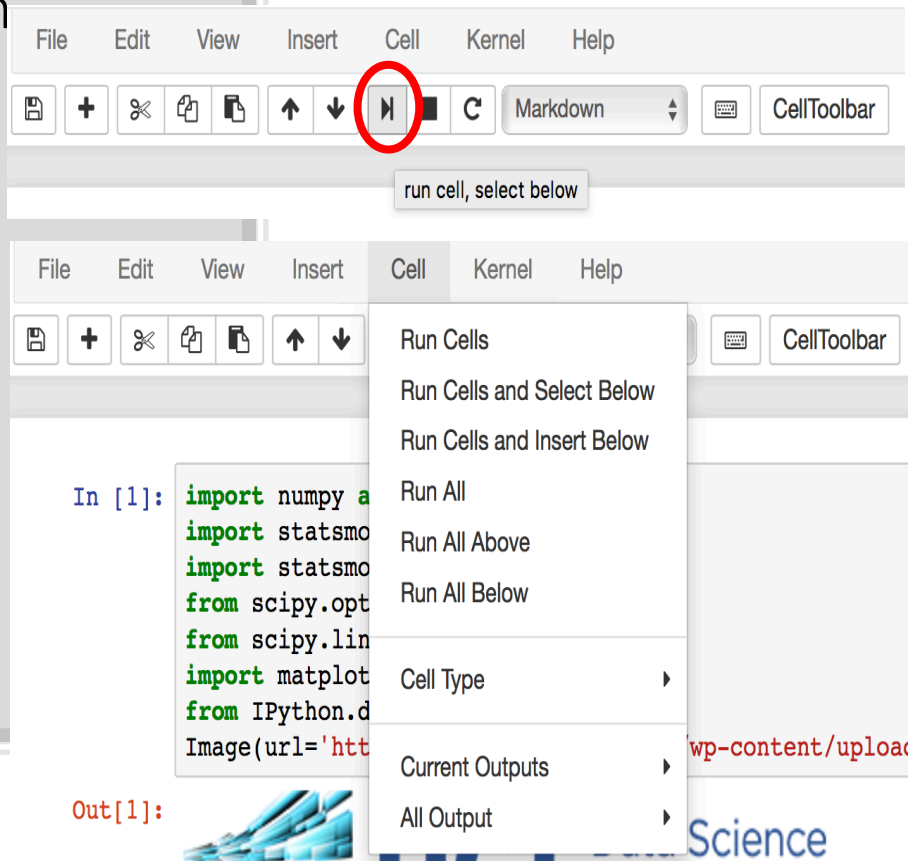


- Can also use Latex in Markdown:

How to use Jupyter Notebooks (cont'd)

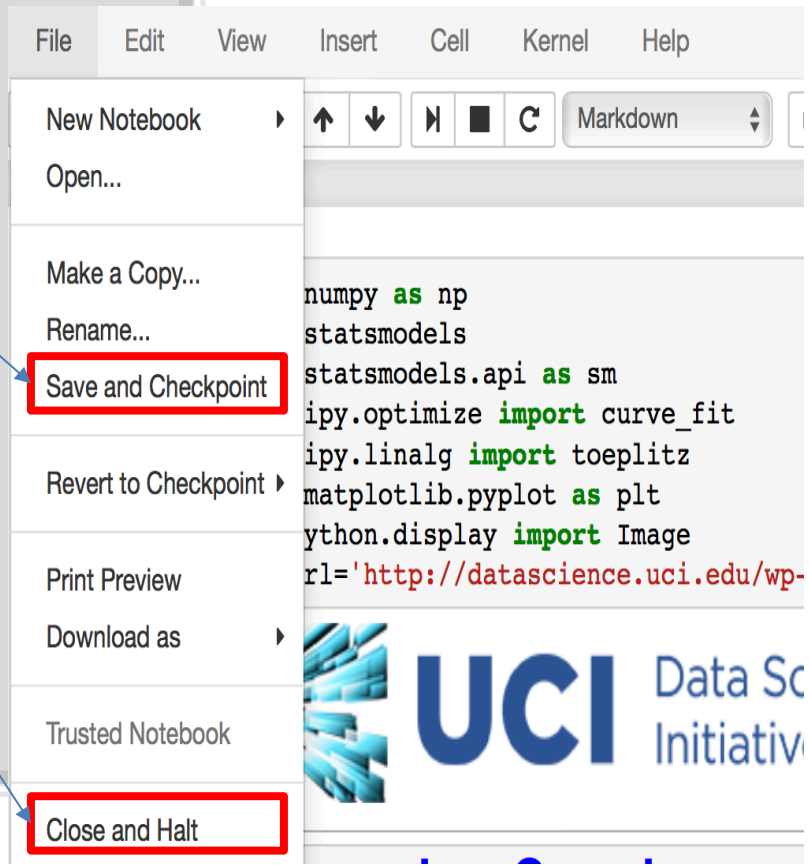
- You can run each block of Markdown or code and get an output box:

- Or run them all together:



How to use Jupyter Notebooks (cont'd)

- Finally, you can save a notebook when you're done editing:
- And close the kernel:



Note about required packages

```
%matplotlib inline
```

```
#-- Import Required Python Packages
import numpy as np
import rpy2.robjects as robjects
from rpy2.robjects.packages import importr
import statsmodels.api as sm
import matplotlib.pyplot as plt
import pandas as pd
from IPython.display import display

#-- Import R packages in Python
r = robjects.r
nlme = importr('nlme')
stats = importr('stats')
```

Using R in Python: rpy2

- Call R functions directly from Python.
- Import R packages with ***importr*** from ***rpy2.robjects.packages***
  - example nlme = importr('nlme')
- Follow Python syntax for calling functions
  - gls() → nlme.gls()
  - Note dots in names become underscores in Python (e.g. arima.sim() → stats.arima_sim())
- Function inputs need to be R objects!

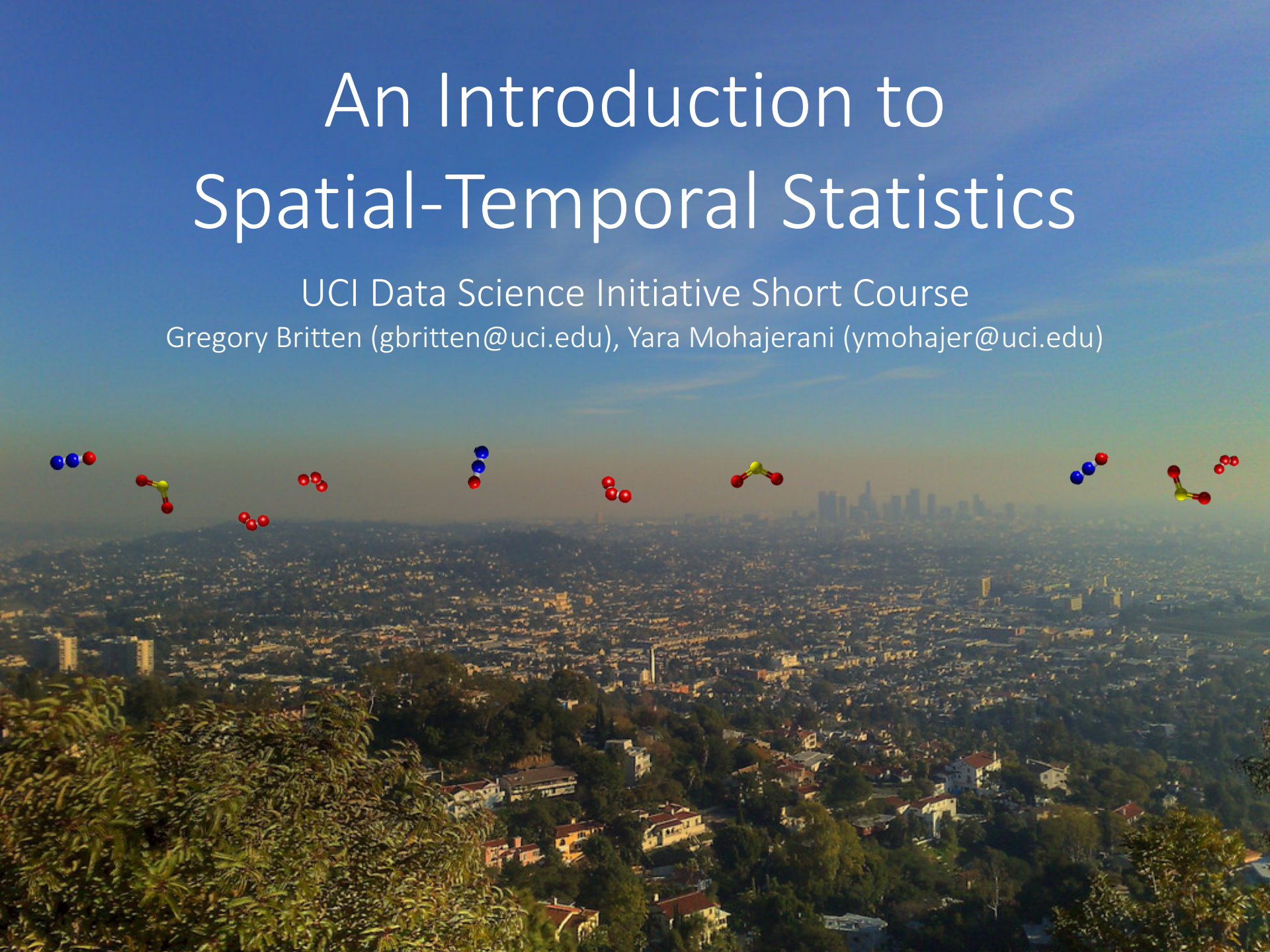
R functions in Python

- Convert variables to R objects and pass to R environment:
  - robjects.globalenv["y"] = robjects.FloatVector(y)
- R formulas are passed using r.formula(“formula”)
  - Note different R packages behave differently. E.g. robjects.r.lm() does not need r.formula() in the argument, where as nlme.gls() does. The return summary of robjects.r and nlme functions are also different.
- The rest of the details will be discussed as we go along!

An Introduction to Spatial-Temporal Statistics

UCI Data Science Initiative Short Course

Gregory Britten (gbritten@uci.edu), Yara Mohajerani (ymohajer@uci.edu)

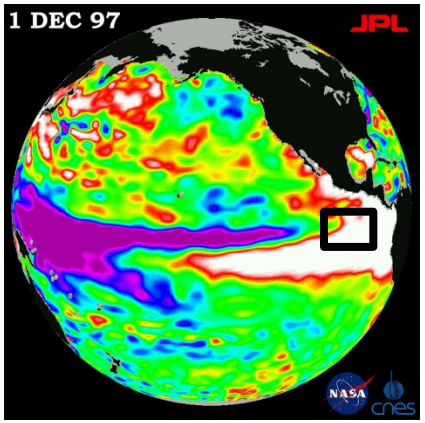


Is there a statistical trend in observed air quality?

Are times of the year systematically worse?

What environmental factors contribute to the trend?

Using environmental datasets but concepts apply across disciplines



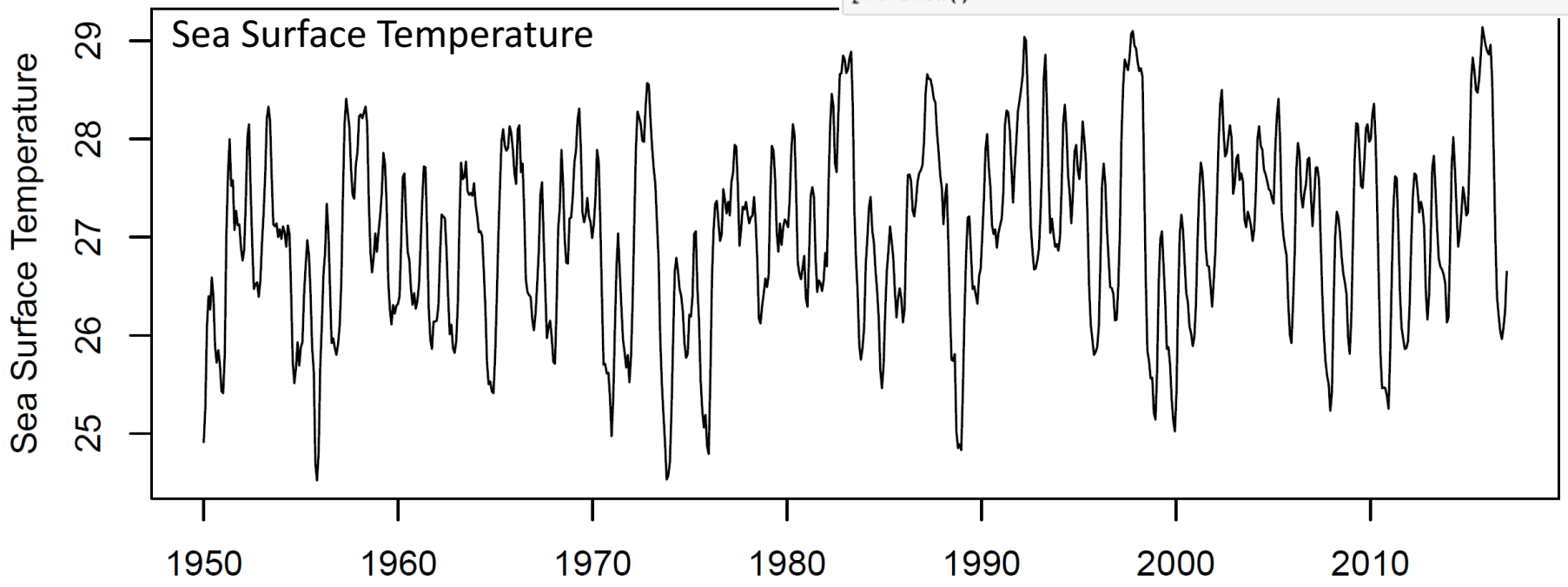
El Nino index: SST off the coast of Central America

Autocorrelation: 'correlation with self'; measurements closer in time are more correlated (similar in value)

Periodicity: The earth spins and orbits the sun! Seasonal, daily, and other climate cycles are fundamental and seen in many datasets

```r
sst <- read.table('../../Data/detrend.nino34.ascii.txt',header=TRUE)
y   <- sst$TOTAL                                          #define ve
t   <- seq(1950,2017,length.out=nrow(sst))                #define ti
options(repr.plot.width=7, repr.plot.height=5)            #define di
plot(t,y,type='l',xlab='',ylab='Sea Surface Temperature') #make plot
```

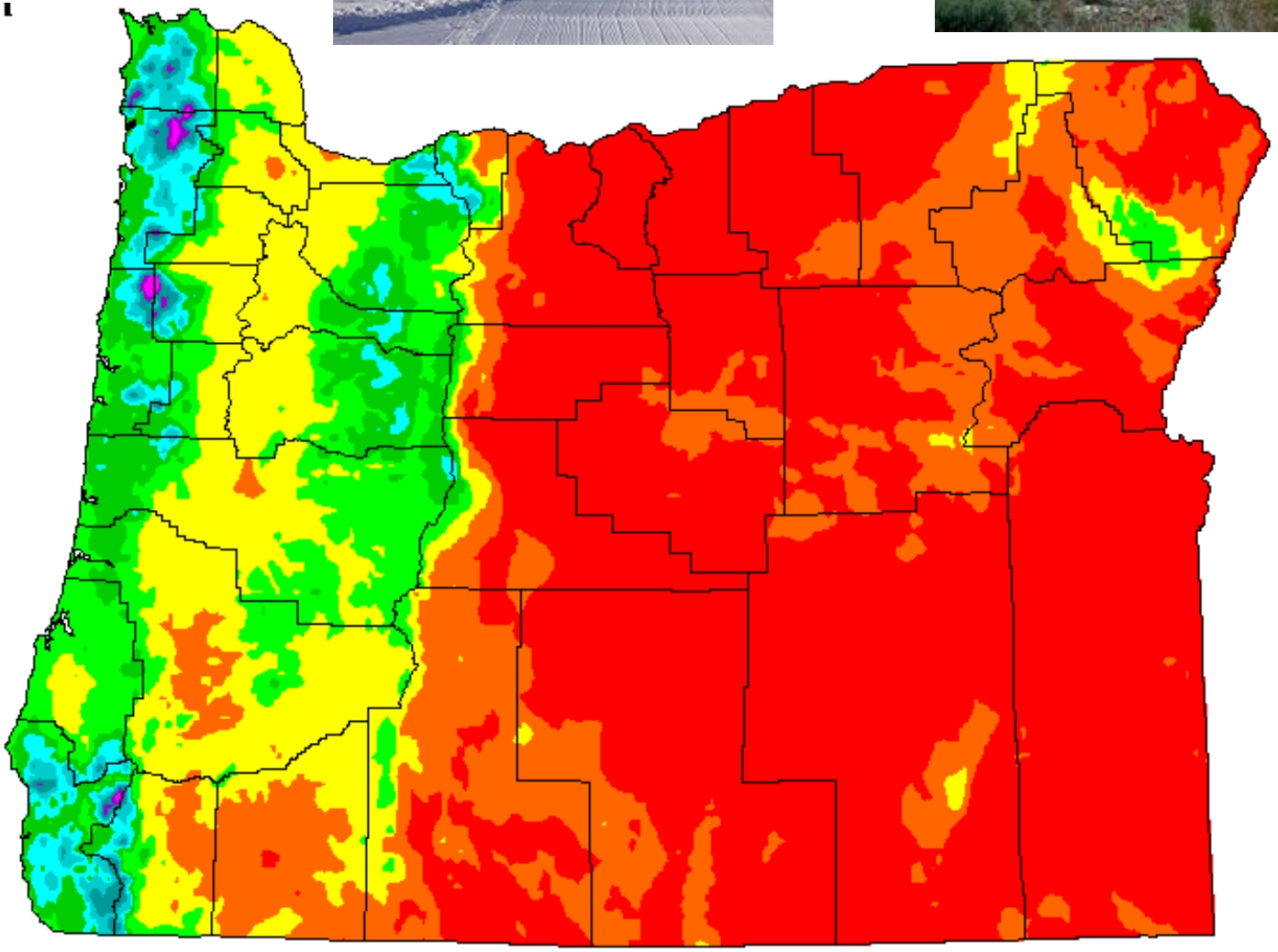
```python
sst = np.loadtxt('../Data/detrend.nino34.ascii.txt',skiprows=1)
y = sst[:,2]                                              # Extract SST
t1 = np.linspace(1950,2017,len(y))                        # Define time

#-- Plot time-series
fig = plt.figure(figsize=(10,6))                          # Set up figu
plt.plot(t1,y,'k-')                                       # Plot SST as
plt.ylabel('Sea Surface Temperature',fontsize=15)         #label y-axis
plt.xlabel('Years',fontsize=15)                           #label x-axis
plt.show()
```



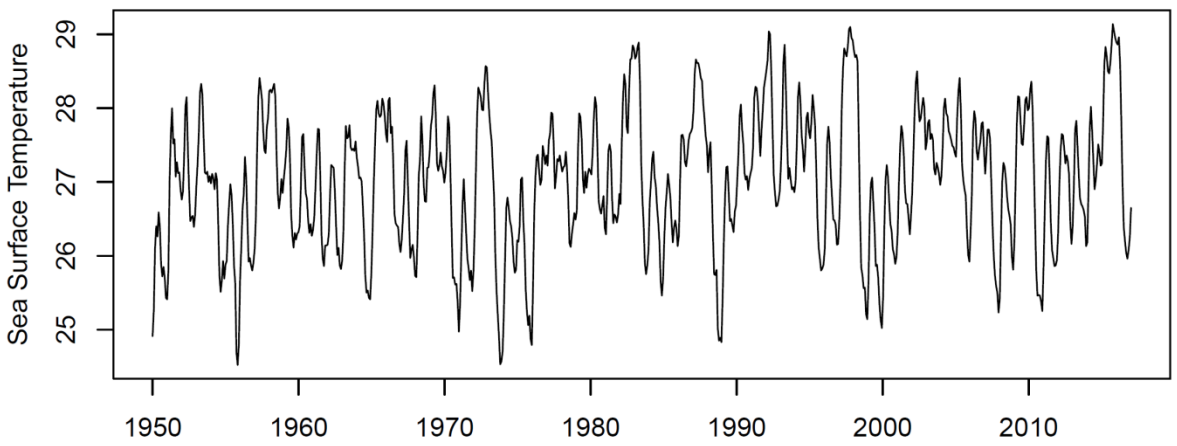
Legend (in inches)

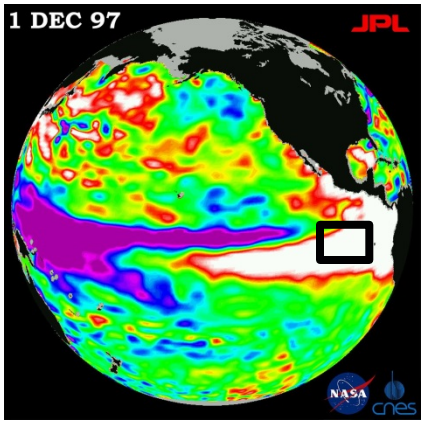
Under 20	100 to 120
20 to 40	120 to 140
40 to 60	140 to 160
60 to 80	160 to 180
80 to 100	Above 180

Period: 1961-1990



Is there a trend? ..a first attempt






```
sst <- read.table('../../Data/detrend.nino34.ascii.txt',header=TRUE)  #read monitoring data published to NOAA website
y   <- sst$TOTAL                                                      #define vector for SST measurement
t   <- seq(1950,2017,length.out=nrow(sst))                            #define time variable [float vector]
```

```
summary(lm(y ~ t))

Call:
lm(formula = y ~ t)

Residuals:
    Min      1Q  Median      3Q     Max
-2.4020 -0.6441  0.0608  0.6246  1.9709

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 10.688845   3.290339   3.249  0.00121 **
t            0.008230   0.001659   4.961 8.55e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.9125 on 805 degrees of freedom
Multiple R-squared:  0.02967,   Adjusted R-squared:  0.02846
F-statistic: 24.61 on 1 and 805 DF,  p-value: 8.551e-07
```

$$y_i = \beta_0 + \beta t_i + e_i \quad e_i \sim N(0, \sigma^2)$$

$$\mathbf{y} = \beta_0 + \beta_1 \mathbf{t} + \mathbf{e} \quad \mathbf{e} \sim \mathbf{N}(0, \boldsymbol{\Sigma})$$

$$\boldsymbol{\Sigma}_{iid} = \begin{bmatrix} \sigma^2 & 0 & \dots & 0 \\ 0 & \sigma^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma^2 \end{bmatrix} = \sigma^2 \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

Indicates strong evidence

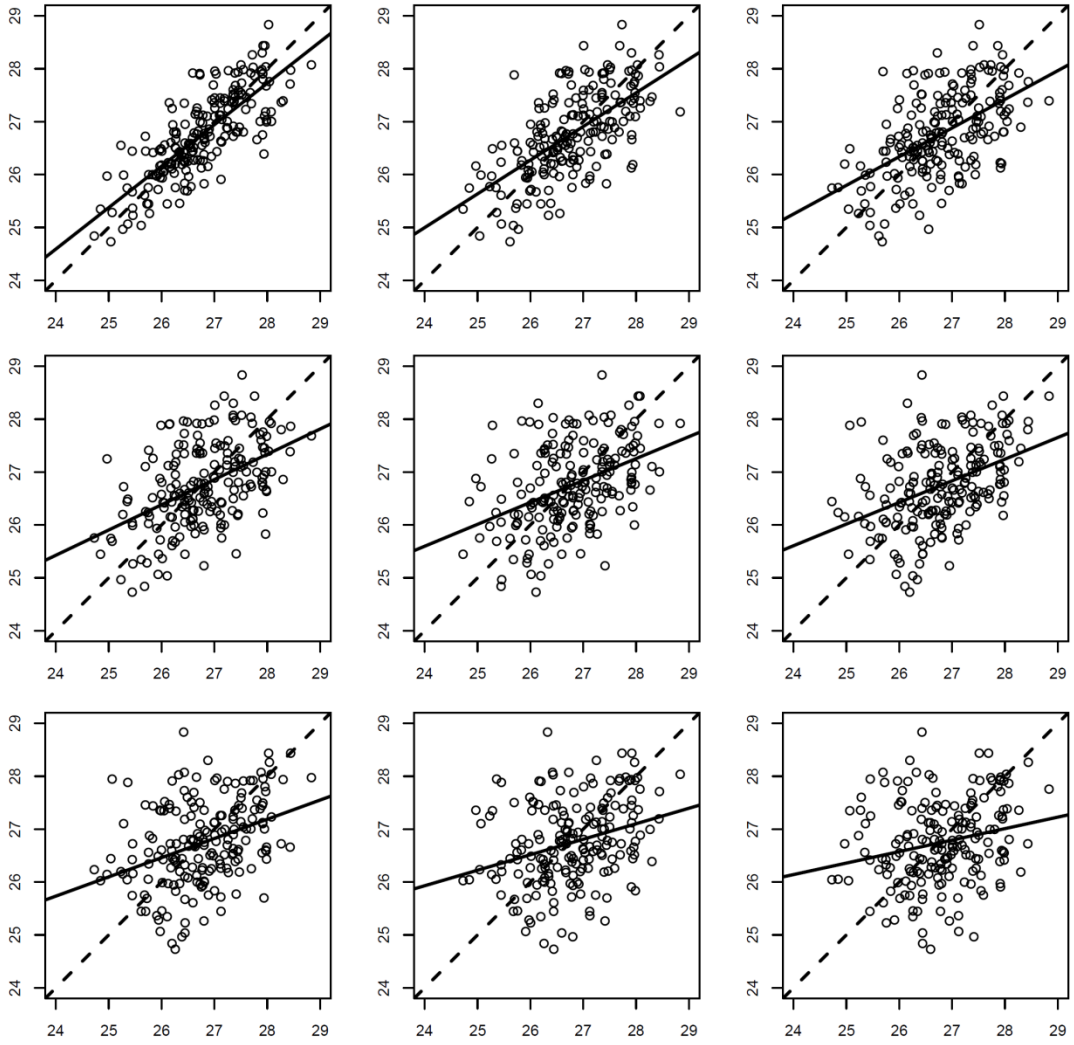
Alert the press, the ocean is warming!

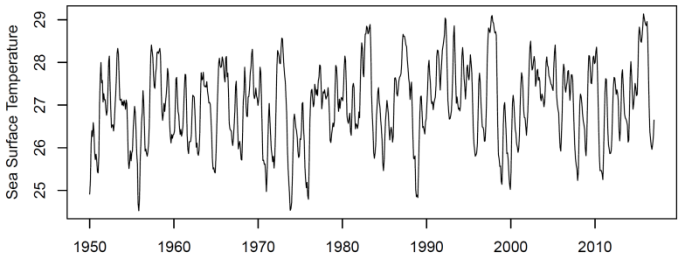
```
robjects.globalenv["y"] = robjects.FloatVector(y)   # Add y to the R environmental as a FloatVector R object
robjects.globalenv["t1"] = robjects.FloatVector(t1) # Add y to the R environmental as a FloatVector R object
fit = r.lm("y ~ t1")                                # linear fit with model y ~ t1
#return summary of fit, including everything after the word "Resduals" to avoid the long "call" information.
print str(r.summary(fit))[str(r.summary(fit)).find("Residuals:"):]
```

- Remember we have to convert the Python objects to R objects and pass them to the R environment before calling them.
- Note in this case we have pass the formula just as a string, but other packages may not be able to interpret formulas as strings, in which case we have to use r.formula().
- Also note the summary of r.lm() returns a lot of excessive “call” info so we cut off the output before the word “Residuals”.

What Generates the Pattern?

Key concept: ***auto****correlation as a function of lag*





Places closer to one another are more similar => autocorrelated

Correlation falls away as a function of temporal distance, or *lag*

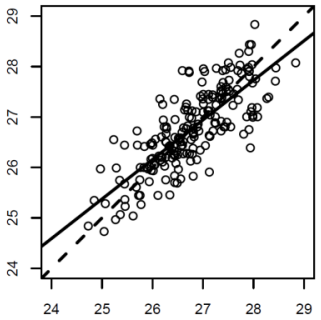
'Autoregressive' pattern

Autoregressive models and the concept of *stochastic realization*

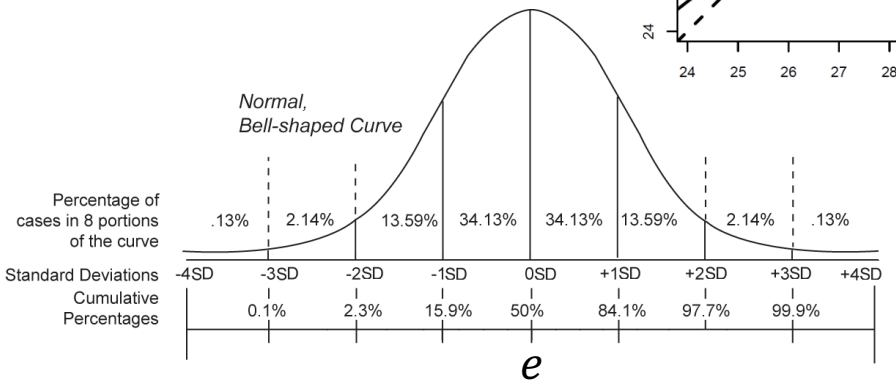
$$y_{t+1} = \phi y_t + e_{t+1}$$

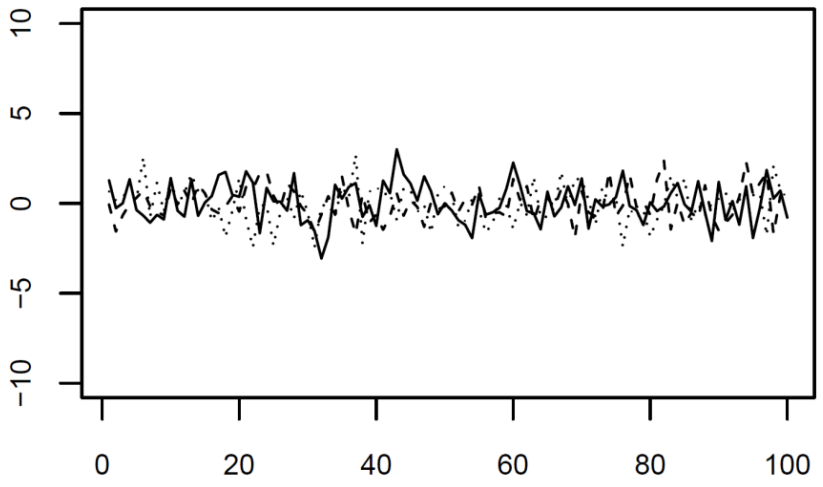
*autoregressive model

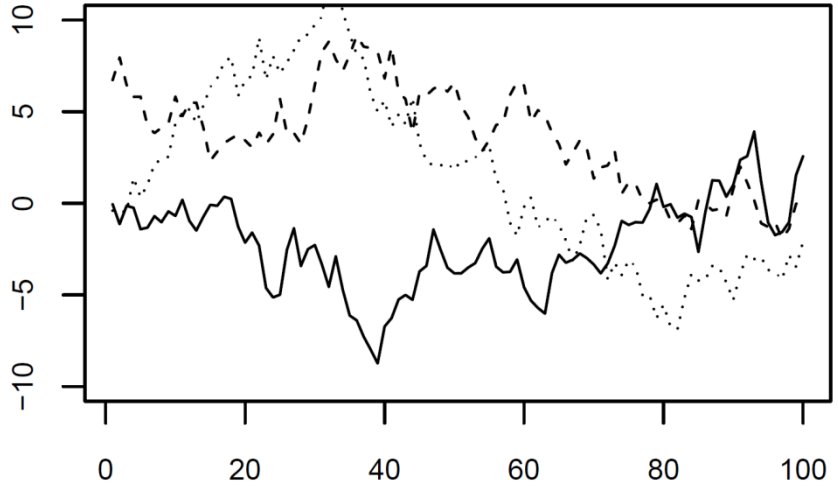
$$y_{t+1} = e_{t+1}$$



ϕ is a value between -1 and 1 and is interpreted as a slope w.r.t lagged values





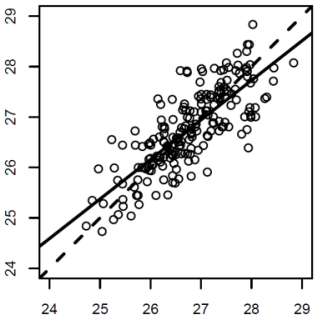


Autoregressive models and the concept of *stochastic realization*

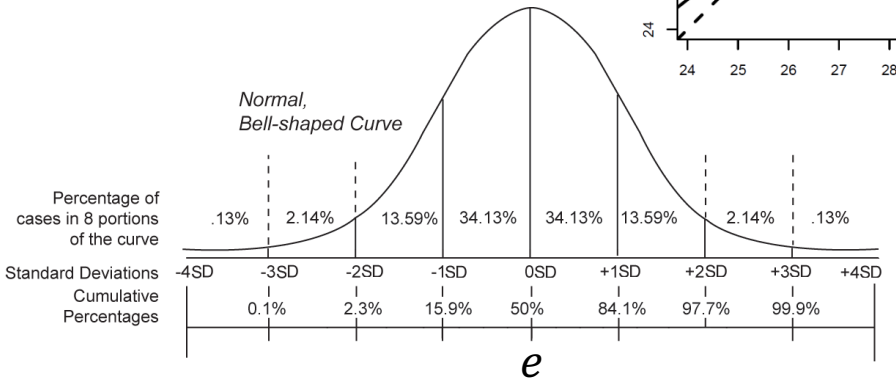
$$y_{t+1} = \phi y_t + e_{t+1}$$

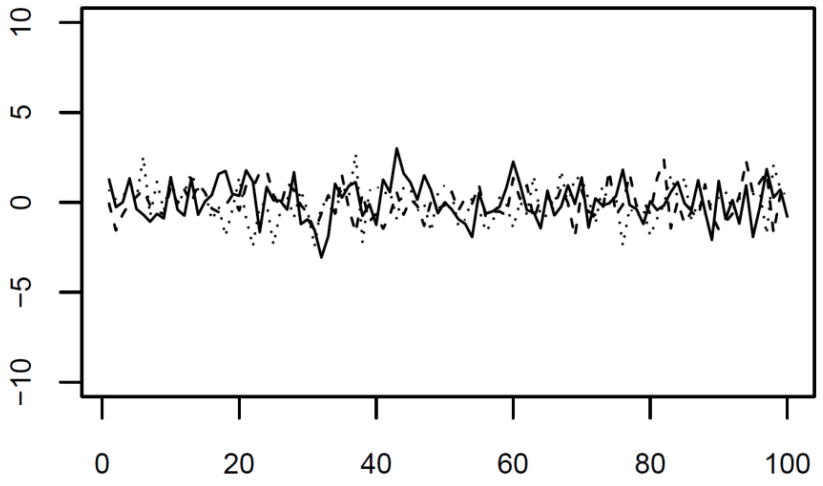
*autoregressive model

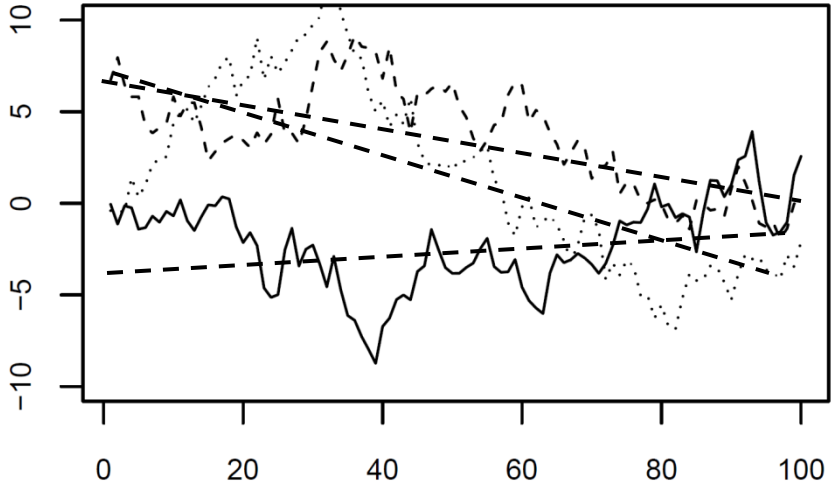
$$y_{t+1} = e_{t+1}$$



ϕ is a value between -1 and 1 and is interpreted as a slope w.r.t lagged values



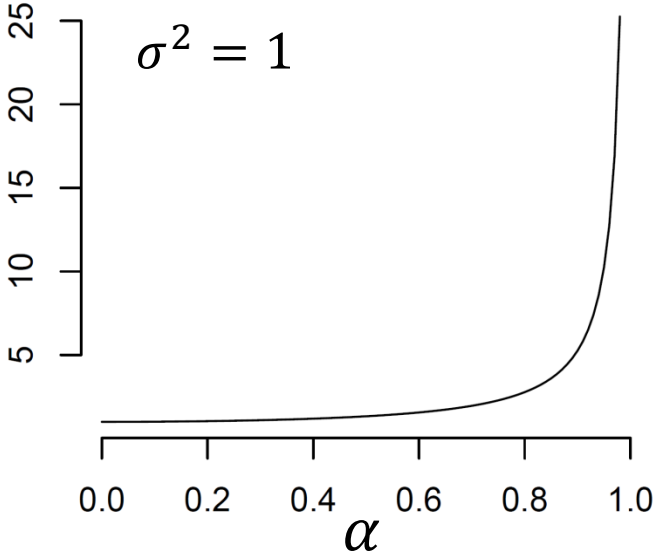


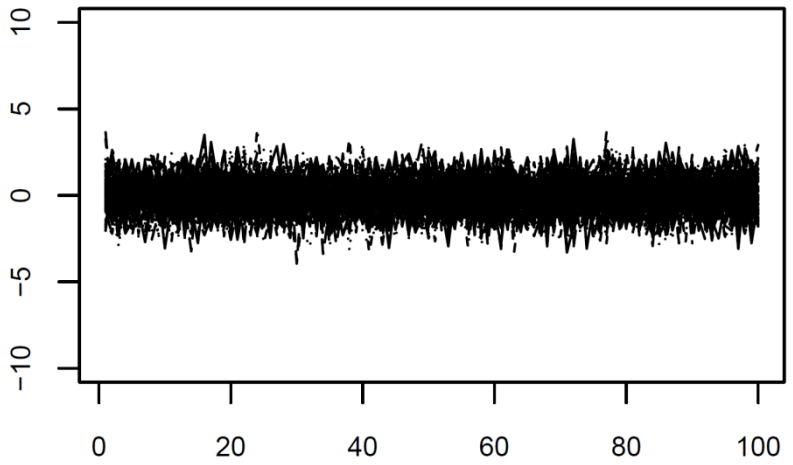


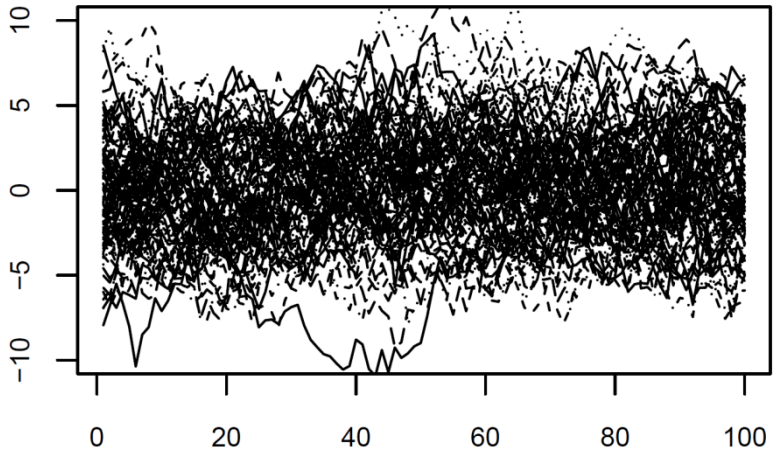
Expected variance of a temporal process increases with autocorrelation

$$\begin{aligned}\mathrm{Var}[y_t] &= \mathrm{Var}[\phi y_t + e_{t+1}] \\ &= \mathrm{Var}[\phi y_t] + \mathrm{Var}[e_{t+1}] \\ &= \phi^2 \mathrm{Var}[y_t] + \sigma^2\end{aligned}$$

$$\Rightarrow \quad \mathrm{Var}[y_t] = \frac{1}{1-\phi^2}\sigma^2$$







```
n   <- 100
phi <- 0.95
s   <- 0.1

options(repr.plot.width=8, repr.plot.height=5)                          #set dimensions of jupyter plot window
X1 <- replicate(3,rnorm(n,sd=s))
X2 <- replicate(3,arima.sim(100,model=list(ar=phi,sd=s)))
X3 <- replicate(100,rnorm(n,sd=s))
X4 <- replicate(100,arima.sim(100,model=list(ar=phi,sd=s)))
yr <- max(abs(c(X1,X2,X3,X4)))
par(mfrow=c(2,2),mar=c(3,4,3,3),cex.axis=0.8)                          #set matrix layout c(2,2), margin width, and axis
    matplot(X1,type='l',ylab='',xlab='',col='black',ylim=c(-yr,yr))    #iid process; 3 realizations; matplot() plots colu
    matplot(X2,type='l',ylab='',xlab='',col='black',ylim=c(-yr,yr))    #autoregressive model with autoregressive coefficie
    matplot(X3,type='l',ylab='',xlab='',col='black',ylim=c(-yr,yr))    #iid process; 100 realizations
    matplot(X4,type='l',ylab='',xlab='',col='black',ylim=c(-yr,yr))    #autoregressive; 100 realizations
```

```
n  = 100
phi = 0.95
s = 0.1

fig, ax = plt.subplots(2,2,figsize=(14,8)) # Set up figure and subplots
#-- 3 sample realizations
for i in range(3):
    ax[0,0].plot(np.array(r.rnorm(n,sd=s)))  # purely random process;
    # autoregressive model with autoregressive coefficient 0.95
    ax[0,1].plot(np.array(stats.arima_sim(n,model=robjects.ListVector({'ar':phi,'sd':s}))))
    ax[0,0].set_ylim([-10,10])             # y range for first plot
    ax[0,1].set_ylim([-10,10])             # y range for second plot
#-- Same as above but with 100 sample realizations
for i in range(100):
    ax[1,0].plot(np.array(r.rnorm(n,sd=s)))
    ax[1,1].plot(np.array(stats.arima_sim(n,model=robjects.ListVector({'ar':phi,'sd':s}))))
    ax[1,0].set_ylim([-10,10])
    ax[1,1].set_ylim([-10,10])
plt.show()
```

Exercise

Simulate Different time series with different properties

Alter n, ϕ, and s to build an intuition for how autocorrelation (ϕ) and perturbation size (s) affect the nature of time series.

Try these combinations for different n to see how time series length influences things

Try fitting the simple trend regressions to your simulated time series and estimate the proportion of 'significant' trends

Modeling autocorrelation in stats models

Covariance between two 'random variables':

$$\operatorname{cov}(x, y) = \mathrm{E}\left[(x - \mathrm{E}[x])(y - \mathrm{E}[y])\right]$$

$$\hat{\operatorname{cov}}(x, y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

Independence:

$$\operatorname{cov}(x, y) = 0$$

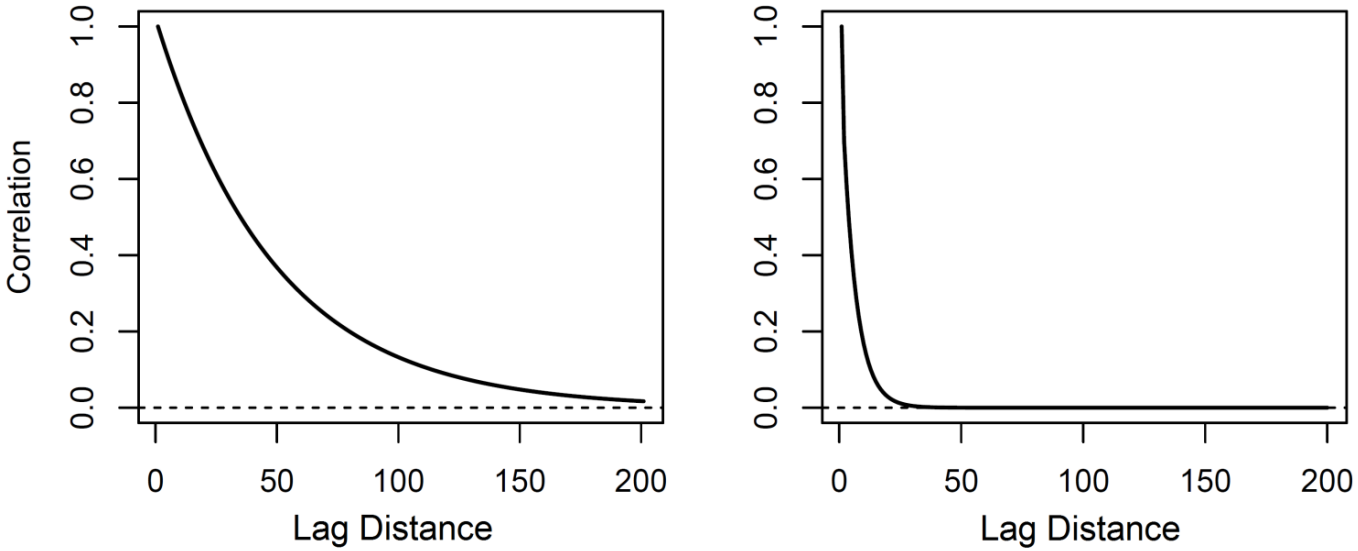
Same idea applied to time series; covariance applied to lag:

$$\operatorname{cov}(x_i, x_{i+\tau}) = \mathrm{E}\left[(x_i - \mathrm{E}[x])(x_{i+\tau} - \mathrm{E}[x])\right]$$

Now the correlation matrix is filled in with correlations between time intervals:

$$\boldsymbol{\Sigma} = \sigma^2 \begin{bmatrix} 1 & c(x_1, x_2) & c(x_1, x_3) & \dots & c(x_1, x_n) \\ c(x_2, x_1) & 1 & c(x_2, x_3) & \ddots & c(x_2, x_n) \\ c(x_3, x_1) & c(x_3, x_2) & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & c(x_{n-1}, x_n) \\ c(x_n, x_1) & c(x_n, x_2) & \dots & c(x_n, x_{n-1}) & 1 \end{bmatrix}$$

Autocorrelation function to 'fill-in' the correlation matrix



The autocorrelation function is a curve that provides a value for the expected correlation as a function of lag, which can be in time or space

We'd like to account for spatial correlation when statistically analyzing data because it affects confidence intervals, p.values, posterior probabilities..

$$c(x_t, x_{t+\tau}) = \phi^\tau$$

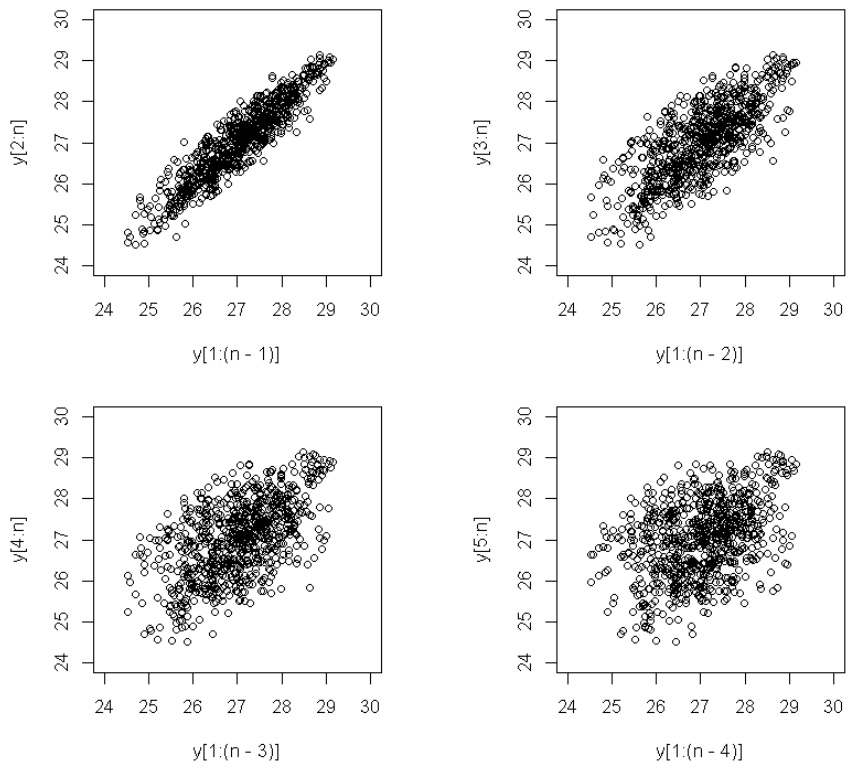
* AR1 correlation function

$$\mathrm{E}[y_t] = \phi \mathrm{E}[y_{t-1}] = \phi^2 \mathrm{E}[y_{t-2}] = \phi^\tau \mathrm{E}[y_{t-\tau}]$$

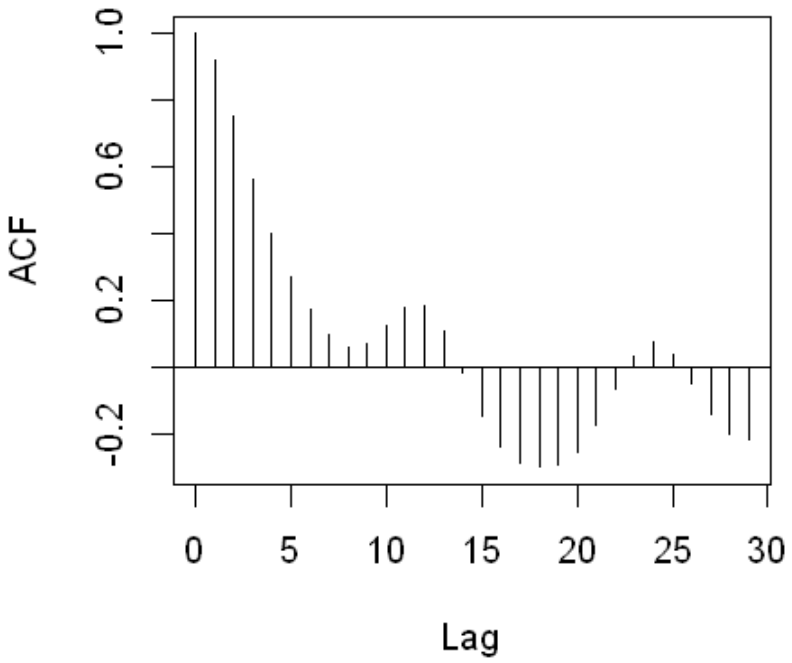
$$\boldsymbol{\Sigma}_{AR1} = \sigma^2 \begin{bmatrix} 1 & \phi & \phi^2 & \dots & \phi^n \\ \phi & 1 & \phi & \ddots & \phi^{n-1} \\ \phi^2 & \phi & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \phi^{n-(n-1)} \\ \phi^n & \phi^{n-1} & \dots & \phi^{n-(n-1)} & 1 \end{bmatrix}$$

The empirical autocorrelation function

```
sst <- read.table('../../Data/detrend.nino34.ascii.txt',header=TRUE) #download sea surface temperature n
y   <- sst$TOTAL      #extract the observed sst and store it as variable y
n   <- length(y)      #extract the number of observations
options(repr.plot.width=7, repr.plot.height=6)
par(mfrow=c(2,2),mar=c(4,4,2,4))
plot(y[1:(n-1)],y[2:n],ylim=c(24,30),xlim=c(24,30))
plot(y[1:(n-2)],y[3:n],ylim=c(24,30),xlim=c(24,30))
plot(y[1:(n-3)],y[4:n],ylim=c(24,30),xlim=c(24,30))
plot(y[1:(n-4)],y[5:n],ylim=c(24,30),xlim=c(24,30))
```




```
options(repr.plot.width=8, repr.plot.height=4)
par(mfrow=c(1,2))
acf(y, ci=0, main='', type='correlation')
acf(y, ci=0, main='', type='covariance')
```

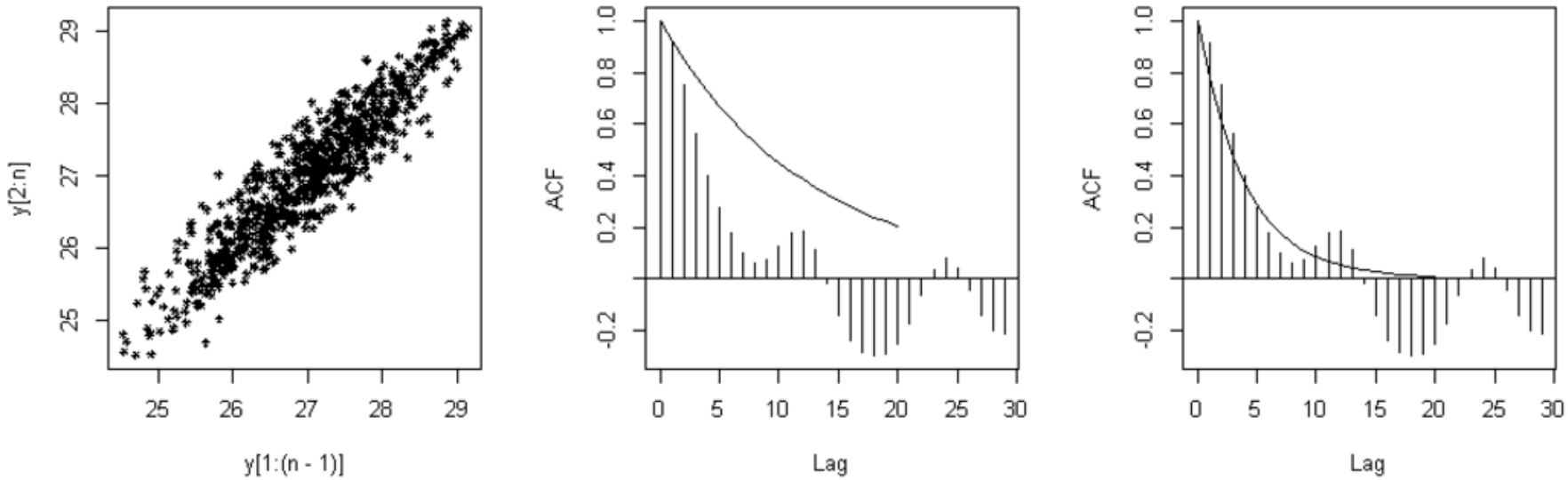


```
fig, ax = plt.subplots(2,2,figsize=(16,10))            # Set up figure and subplots
ax[0,0].scatter(y[:-1],y[1:],edgecolor='k',facecolor='w')      # scatter plot of values shifted by 1
ax[0,0].set_xlabel('y[0:(n-1)]',fontsize=16)            # x axis label
ax[0,0].set_ylabel('y[1:n]',fontsize=16)                # y axis label
ax[0,1].scatter(y[:-2],y[2:],edgecolor='k',facecolor='w')      # scatter plot of values shifted by 2
ax[0,1].set_xlabel('y[0:(n-2)]',fontsize=16)            # x axis label
ax[0,1].set_ylabel('y[2:n]',fontsize=16)                # y axis label
ax[1,0].scatter(y[:-3],y[3:],edgecolor='k',facecolor='w')      # scatter plot of values shifted by 3
ax[1,0].set_xlabel('y[0:(n-3)]',fontsize=16)            # x axis label
ax[1,0].set_ylabel('y[3:n]',fontsize=16)                # y axis label
ax[1,1].scatter(y[:-4],y[4:],edgecolor='k',facecolor='w')      # scatter plot of values shifted by 4
ax[1,1].set_xlabel('y[0:(n-14]',fontsize=16)            # x axis label
ax[1,1].set_ylabel('y[4:n]',fontsize=16)                # y axis label
plt.show()
```

```
fig, ax = plt.subplots(1,2,figsize=(15,5))            # Set up figure and subplots
acf_corr = np.squeeze(np.array(stats.acf(robjects.FloatVector(y), ci=0, main='', type='correlation')[0]))
acf_cov = np.squeeze(np.array(stats.acf(robjects.FloatVector(y), ci=0, main='', type='covariance')[0]))
n = len(acf_corr) #number of lags
#-0 Plot Autocorrelation
ax[0].vlines(range(n),[0],acf_corr)       # plot vertical lines
ax[0].plot(range(n),np.zeros(n))          # plot zero line
ax[0].set_ylabel('ACF',fontsize=15)       # label y axis
ax[0].set_xlabel('Lag', fontsize=15)      # label x axis
#-- Plot Autocovariance
ax[1].vlines(range(n),[0],acf_cov)        # plot vertical lines
ax[1].plot(range(n),np.zeros(n))          # plot zero line
ax[1].set_ylabel('ACF (cov)',fontsize=15) # label y axis
ax[1].set_xlabel('Lag',fontsize=15)       # label x axis
plt.show()
```

Fitting models to the empirical autocorrelation function

```
options(repr.plot.width=8, repr.plot.height=3)
par(mfrow=c(1,3))                                          #set the plot matrix layout c(1,2)
plot(y[1:(n-1)],y[2:n],pch=8,cex=0.5)                      #scatterplot with values lagged by 1 unit
alpha_hat <- cor(y[1:(n-1)],y[2:n])                       #the correlation coefficient between lagged values
acf(y,demean=TRUE,ci=0,main='',type='correlation')          #estimate and plot the empirical autocorrelation function
    lines(seq(0,20),alpha_hat^seq(0,20))                  #add line for threoretical acf using correlation coefficient
acf(y,demean=TRUE,ci=0,main='',type='correlation')          #estimate and plot the empirical autocorrelation function
    lines(seq(0,20),0.78^seq(0,20))                        #plot the optimized autocorrelation function (more info below)
```



$$\boldsymbol{\Sigma}_{AR1} = \sigma^2 \begin{bmatrix} 1 & \phi & \phi^2 & \dots & \phi^n \\ \phi & 1 & \phi & \ddots & \phi^{n-1} \\ \phi^2 & \phi & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \phi^{n-(n-1)} \\ \phi^n & \phi^{n-1} & \dots & \phi^{n-(n-1)} & 1 \end{bmatrix}$$

$$c(x_t, x_{t+\tau}) = \phi^{\tau}$$

```
fig, ax = plt.subplots(1,3,figsize=(17,5))            # Set up figure and subplots
ax[0].scatter(y[:-1],y[1:],color='k',marker='*')      # scatter plot of values shifted by 1
ax[0].set_xlabel('y[1:(n-1)]',fontsize=16)            # x axis label
ax[0].set_ylabel('y[2:n]',fontsize=16)                # y axis label
sm.graphics.tsa.plot_acf(y,lags=30,alpha=np.float('nan'),ax=ax[1]) # plot empirical autocorrelation function
alpha_hat = np.corrcoef(y[:-1],y[1:])[0,1]           # the correlation coefficient between lagged values
ax[1].plot(np.linspace(0,20),alpha_hat**np.linspace(0,20))        # line for theoretical acf using correlation coeff
ax[1].set_xlabel('Lag', fontsize=14)                  # x axis label
ax[1].set_ylabel('ACF',fontsize=14)                   # y axis label
sm.graphics.tsa.plot_acf(y,lags=30,alpha=np.float('nan'),ax=ax[2]) # plot empirical autocorrelation function
ax[2].plot(np.linspace(0,20),0.78**np.linspace(0,20)) # plot optimized autocorrelation function (more info below)
ax[2].set_xlabel('Lag', fontsize=14)                  # x axis label
ax[2].set_ylabel('ACF',fontsize=14)                   # y axis label
plt.show()
```

The likelihood makes all the difference

$$p(\mathbf{y}|\theta, M, \sigma^2) = \prod_{i=1}^{n} p(y_i|\theta, M, \sigma^2) = \left(\frac{1}{2\pi\sigma^2}\right)^{n/2} \exp\left(-\frac{\sum_{i=1}^{n} e_i^2}{2\sigma^2}\right)$$

$$e_i = y_i - f(\hat{\theta}, M)$$

'Best' (min. variance) unbiased estimate for regression coefficients

$$\hat{\beta}_{\mathrm{OLS}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$$

For correlated data

$$p(\mathbf{y}|\theta, M, \sigma^2) = \prod_{i=1}^{n} p(\mathbf{y}_i|\theta, M, \sigma^2) = \frac{1}{(2\pi)^{\frac{n}{2}}|\boldsymbol{\Sigma}|^{\frac{n}{2}}} \exp\left(-\frac{1}{2}\mathbf{e}'\boldsymbol{\Sigma}^{-1}\mathbf{e}\right)$$

$$\mathbf{e} = \mathbf{y} - f(\hat{\theta}, M)$$

BLUE regression coefficients with autocorrelation

$$\hat{\beta}_{\mathrm{GLS}} = (\mathbf{X}'\boldsymbol{\Sigma}^{-1}\mathbf{X})^{-1}\mathbf{X}'\boldsymbol{\Sigma}^{-1}\mathbf{y}$$

*nice analytical expressions for linear regression, idea applies equally to nastier functions

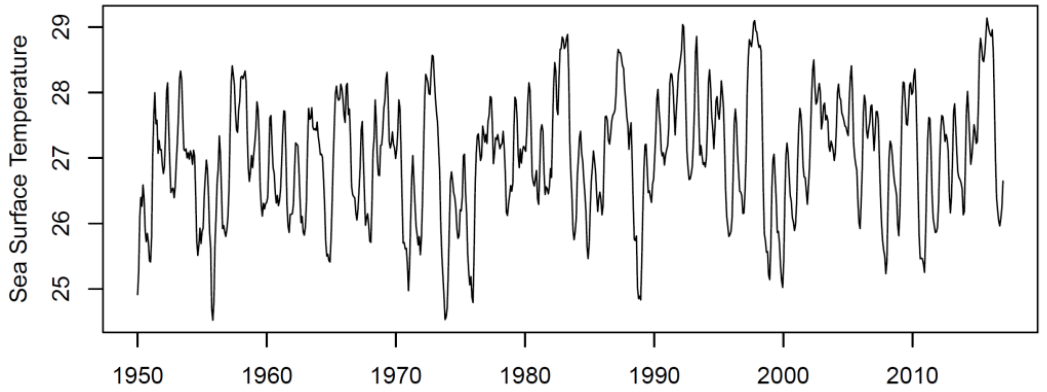
Fitting a Model with Temporal Autocorrelation

```
sst <- read.table('../../Data/detrend.nino34.ascii.txt',header=TRUE) #extract observed sea surface temperature from the NOAA webs
y   <- sst$TOTAL                                                     #extract the observed sea surface temperature
t   <- seq(1950,2017,length.out=nrow(sst))                           #define time variable with same number of years as the data
```

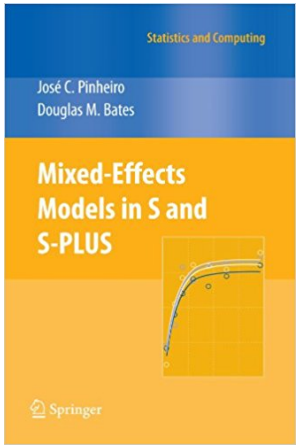
```
summary(gls(y ~ t, correlation=corAR1(), method='ML'))
```

```
# Fit a linear regression between y and time, specifying a first order autoregressive covariance matrix.
# Specify method='ML' to fit exact maximum likelihood
print r.summary(nlme.gls(r.formula("y ~ t1"), correlation=nlme.corAR1(),method='ML'))
```

```
Generalized least squares fit by maximum likelihood
  Model: y ~ t
  Data: NULL
       AIC      BIC    logLik
  627.5535 646.3268 -309.7768

Correlation Structure: AR(1)
 Formula: ~1
 Parameter estimate(s):
      Phi
0.9222667

Coefficients:
               Value Std.Error   t-value p-value
(Intercept) 6.131857 15.776627 0.3886672  0.6976
t           0.010514  0.007954 1.3219154  0.1866

 Correlation:
  (Intr)
t -1

Standardized residuals:
       Min         Q1        Med         Q3        Max
-2.5647249 -0.6749418  0.1034492  0.7083635  2.1403002

Residual standard error: 0.9178044
Degrees of freedom: 807 total; 805 residual
```

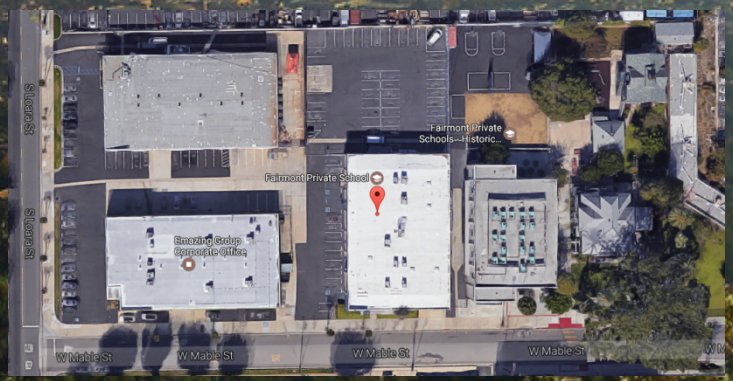


Much weaker evidence!

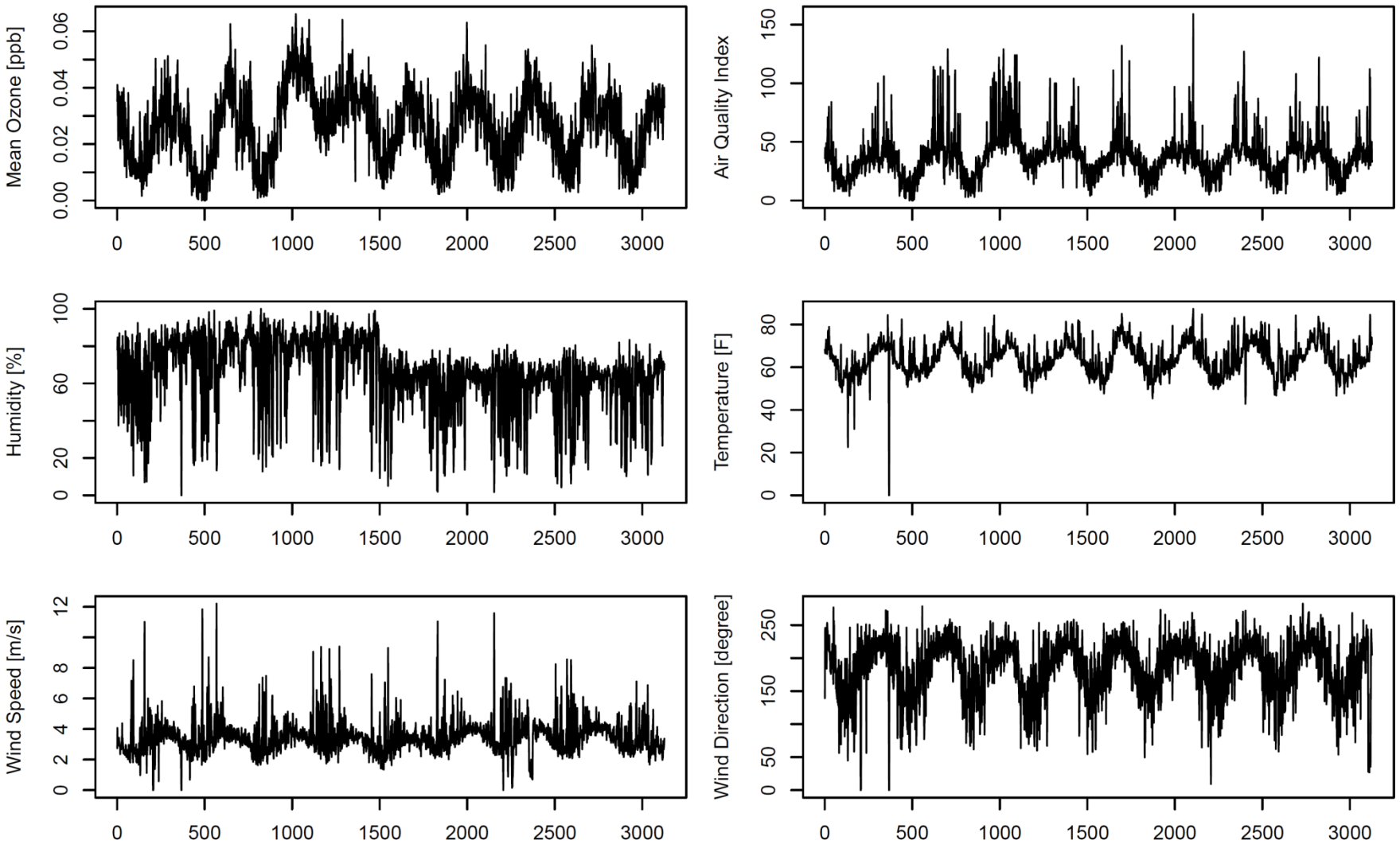


What's Going on with our Air?

EPA Air Quality Station Data, Orange County



Time series of ozone data



Days since September 7th 2001

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	lat	lon	n	year	month	day	ozone	ozone_max	hour_max	aqi	pressure	humidity	temp	windsp	winddir
2	33.83062	-117.9385	24	2001	9	7	0.035292	0.042	22	36	29.75	78.125	68.208333	2.741667	139.29167
3	33.83062	-117.9385	24	2001	9	8	0.041	0.05	9	42	29.704167	78.25	67.416667	4.070833	239.08333
4	33.83062	-117.9385	24	2001	9	9	0.038292	0.053	10	45	29.7	84.666667	66.458333	3.025	245.875
5	33.83062	-117.9385	24	2001	9	10	0.030125	0.045	11	38	29.716667	82.041667	66.416667	3.1	199.20833
6	33.83062	-117.9385	24	2001	9	11	0.031167	0.05	9	42	29.716667	67.583333	68.083333	2.920833	212.25
7	33.83062	-117.9385	24	2001	9	12	0.021333	0.039	9	33	29.658333	73.541667	70.041667	3.2125	190.625
8	33.83062	-117.9385	24	2001	9	13	0.026333	0.038	9	32	29.7	65.5	70.166667	3.095833	185.41667
9	33.83062	-117.9385	24	2001	9	14	0.030417	0.051	10	43	29.720833	37.416667	72.916667	3.0625	211.45833
10	33.83062	-117.9385	24	2001	9	15	0.034292	0.058	10	49	29.720833	46.166667	71.25	3.145833	212.83333
11	33.83062	-117.9385	24	2001	9	16	0.038583	0.049	9	42	29.725	56.791667	68.5	3.108333	197.66667
12	33.83062	-117.9385	24	2001	9	17	0.038292	0.049	10	42	29.7375	72.416667	67.583333	3.429167	184
13	33.83062	-117.9385	24	2001	9	18	0.034333	0.055	10	47	29.7125	84.041667	68.333333	3.175	233.875
14	33.83062	-117.9385	24	2001	9	19	0.032833	0.051	10	43	29.720833	86.916667	66.75	2.9875	249.41667
15	33.83062	-117.9385	24	2001	9	20	0.031167	0.05	10	42	29.7625	59.083333	66.166667	2.895833	253.33333
16	33.83062	-117.9385	24	2001	9	21	0.028667	0.061	10	54	29.754167	48.5	69.041667	2.845833	246.625
17	33.83062	-117.9385	24	2001	9	22	0.027333	0.054	10	46	29.708333	78.916667	69.083333	2.920833	242.20833

Variables

lat ≡ latitude [degrees]
lon ≡ longitude [degrees]
n ≡ number of samples [#]
ozone ≡ mean daily ozone [ppb]
ozone_max ≡ max daily ozone [ppb]
hour_max ≡ hour when max occurs [hours from 12AM]

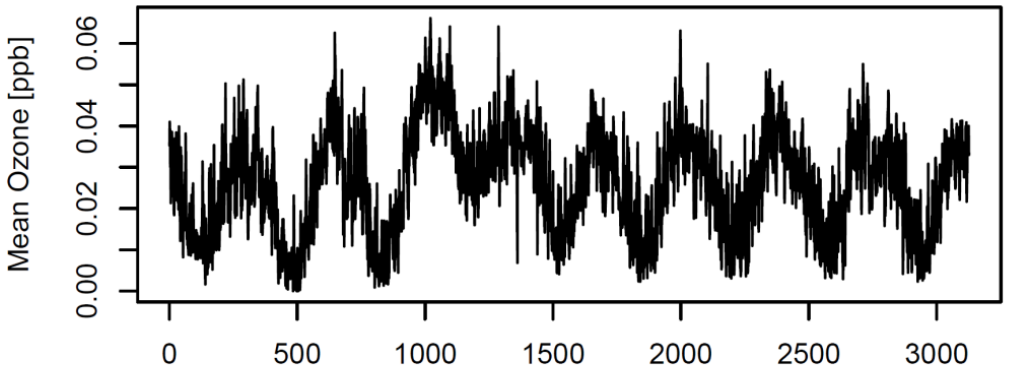
aqi ≡ air quality index [dimensionless]
pressure ≡ atmospheric pressure [inHg]
humidity ≡ [relative %]
temp ≡ [Fahrenheit]
windsp ≡ [m/s]
winddir ≡ [degrees]

Simple Trend Analysis

```
d  <- read.csv('../../Data/ozone_orange.csv',stringsAsFactors=FALSE)
d  <- d[d$year!=2016,]                                    #incomplete chunk of
```

```
y1 <- d$ozone
t  <- seq(min(d$year),max(d$year),length.out=length(y1))
summary(gls(y1 ~ t, correlation=corAR1(), method='ML'))
```

```
Generalized least squares fit by maximum likelihood
  Model: y1 ~ t
  Data: NULL
       AIC       BIC   logLik
  -21910.74 -21886.67 10959.37

Correlation Structure: AR(1)
 Formula: ~1
 Parameter estimate(s):
      Phi
0.8552849

Coefficients:
                Value Std.Error     t-value p-value
(Intercept)  0.03422281 0.6293315  0.05437963  0.9566
t           -0.00000413 0.0003138 -0.01316690  0.9895

 Correlation:
  (Intr)
t -1

Standardized residuals:
         Min           Q1          Med           Q3          Max
-2.056608887 -0.800994961  0.009574762  0.706174031  3.184672958

Residual standard error: 0.01261744
Degrees of freedom: 3035 total; 3033 residual
```



```
d = pd.read_csv('../Data/ozone_orange.csv')      # Read Ozone data from CSV file
ind = np.nonzero(d['year']!=2016)                # Get indices for data before 2016 for simplicity (incomplete chunk of
```

```
y1 = np.array(d['ozone'])[ind]                   # Extract the daily mean ozone variable
years = np.array(d['year'])[ind]                 # Extract years
t1 = np.linspace(years.min(),years.max(),len(y1))#define time variable
robjects.globalenv["y1"] = robjects.FloatVector(y1) # Add y1 to R envrionment as FloatVector
robjects.globalenv["t1"] = robjects.FloatVector(t1) # Add y1 to R envrionment as FloatVector
#-- display the summary of a fitted regression with autoregressive covariance matrix
print r.summary(nlme.gls(r.formula("y1 ~ t1"), correlation=nlme.corAR1(),method='ML'))
```

Harmonic regression

Include a periodic covariate with frequency ω and phase ζ

$$x_t = A\cos(2\pi\omega t + \zeta)$$

This is nonlinear in the parameters, rewrite using the double-angle formula :

$$\cos(a+b) = \cos(a)\cos(b) - \sin(a)\sin(b)$$

$$x_t = A\cos(\zeta)\cos(2\pi\omega t) - A\sin(\zeta)\sin(2\pi\omega t)$$

Now we model any frequency with arbitrary phase using linear regression with

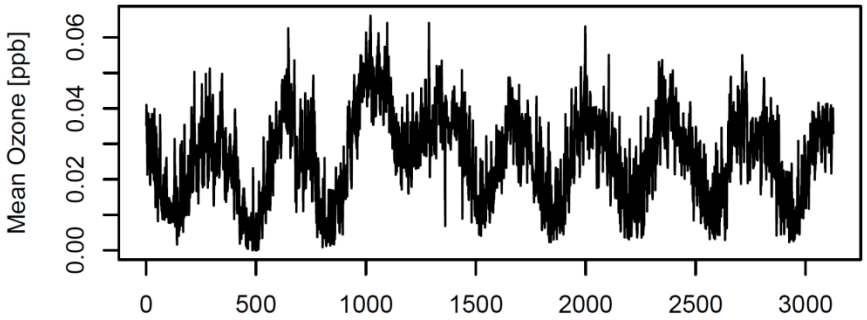
$$\hat{\beta}_1 = A\cos(\zeta) \qquad \hat{\beta}_2 = -A\sin(\zeta)$$

Harmonic regression

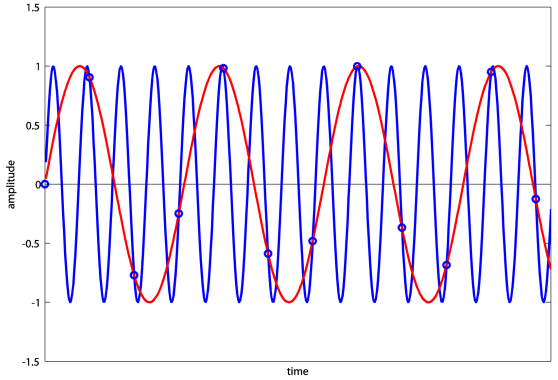
```
nyrs   <- max(d$year)-min(d$year)          #number of seasonal cycles [years]
f      <- seq(0,2*pi*nyrs,length.out=nrow(d)) #nyrs cycles over the length of the time series [radians]
sinf   <- sin(f)                           #sine wave of f
cosf   <- cos(f)                           #cose wave of f
```

```
summary(gls(y1 ~ sinf + cosf,correlation=corAR1(), method='ML'))
```

```
Generalized least squares fit by maximum likelihood
  Model: y1 ~ sinf + cosf
  Data: NULL
       AIC       BIC   logLik
  -22053.52 -22023.43 11031.76

Correlation Structure: AR(1)
 Formula: ~1
 Parameter estimate(s):
      Phi
0.7700548

Coefficients:
                Value    Std.Error  t-value p-value
(Intercept) 0.025893952 0.0005036630 51.41126  0.0000
sinf        0.000891778 0.0007113883  1.25357  0.2101
cosf        0.010826465 0.0007096038 15.25706  0.0000

 Correlation:
     (Intr) sinf
sinf  0.000
cosf -0.004  0.000

Standardized residuals:
        Min          Q1         Med          Q3         Max
-2.97903286 -0.72563451 -0.03934443  0.67355001  3.73500124

Residual standard error: 0.0100069
Degrees of freedom: 3035 total; 3032 residual
```



Note: the highest resolvable frequency is half that at which observations are resolved (need two points per cycle to capture highs and lows), known as 'aliasing'



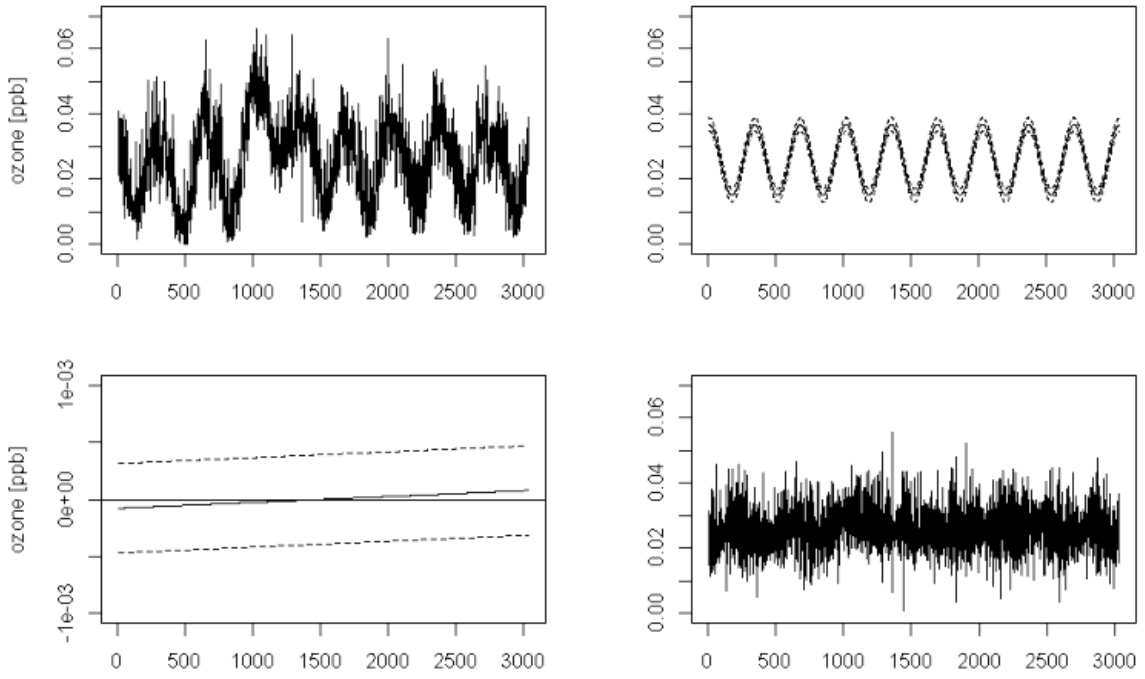
```
nyrs= years.max() - years.min()          # number of seasonal cycles [years]
f = np.linspace(0,2*np.pi*nyrs,len(y1)) # nyrs cycles over the length of the time series [radians]
sinf = np.sin(f)                         # sine wave of f
cosf = np.cos(f)                         # cose wave of f
```

```
robjects.globalenv["sinf"] = robjects.FloatVector(sinf) # Add sine to R envrionment as FloatVector
robjects.globalenv["cosf"] = robjects.FloatVector(cosf) # Add cose to R envrionment as FloatVector
#-- fit regression with seasonal component and first order autoregressive covariance matrix
print r.summary(nlme.gls(r.formula("y1 ~ sinf + cosf"), correlation=nlme.corAR1(),method='ML'))
```

Time Series Decomposition

```
fit     <- gls(y1 ~ sinf + cosf + t,correlation=corAR1(), method='ML')  #store the model object from the fit above
k_hat   <- summary(fit)$t                                               #extract fitted regression coefficients
phi_hat <- coef(fit$model$corStruct,unconstrained=FALSE)                #extract fitted autoregressive parameter
e       <- residuals(fit,'response')                                    #extract 'raw' residuals (yi - ypred)
n       <- length(e)
```

```
options(repr.plot.width=8, repr.plot.height=5)
par(mfrow=c(2,2),mar=c(3,4,2,2))
plot(y1,type='l',ylim=c(0,0.07),ylab='ozone [ppb]')                     #plot the data

f_hat <- mean(y1) + k_hat[2,1]*sinf + k_hat[3,1]*cosf                   #fitted seasonal cycle
plot(f_hat,type='l',ylim=c(0,0.07),ylab='')                             #plot fitted seasonal cycle
    lines(f_hat + 2*sqrt(k_hat[2,2]^2 + k_hat[3,2]^2),lty=2)            #add 95% CIs (upper)
    lines(f_hat - 2*sqrt(k_hat[2,2]^2 + k_hat[3,2]^2),lty=2)            #add 95% CIs (lower)

t_hat <- k_hat[1,1] + k_hat[4,1]*t                                      #fitted trend
t_hat <- t_hat - mean(t_hat)                                            #normalize to the zero-line
plot(t_hat, type='l',ylim=c(-0.001,0.001),ylab='ozone [ppb]')           #plot the values
    lines(t_hat + 2*k_hat[4,2],lty=2)                                   #upper 95% CI
    lines(t_hat - 2*k_hat[4,2],lty=2)
    abline(h=0)
ear <- e[2:n]-phi_hat*e[1:(n-1)]
plot(mean(y1)+ear,type='l',ylim=c(0,0.07),ylab=''
```



```
# store the model object from the fit above
fit = nlme.gls(r.formula("y1 ~ sinf + cosf + t1"), correlation=nlme.corAR1(),method='ML')
```

```
k_hat = fit.rx2('coefficients')          # extract fitted regression coefficients
print k_hat                              # Display for comparison with R
k_hat_var = np.array(fit.rx2('varBeta')) # extract the associated errors
print k_hat_var                          # Display for comparison with R
phi_hat = stats.coef(fit.rx2("modelStruct"),unconstrained=False) #extract fitted autoregressive parameter
e = np.array(fit.rx2('residuals'))       #extract 'raw' residuals (yi - ypred)
```

```
#-- Plot results
fig, ax = plt.subplots(2,2,figsize=(16,8))    # set up figure and panels
ax[0,0].plot(y1,'k-')                         # plot the data
ax[0,0].set_ylim([0,0.07])                    # y axis range
ax[0,0].set_ylabel('ozone [ppb]',fontsize=15) # y axis label

f_hat = np.mean(y1) + k_hat[1]*sinf + k_hat[2]*cosf  # fitted seasonal cycle
ax[0,1].plot(f_hat,'k-')                             # plot fitted seasonal cycle
#-- add 95% CIs (upper and lower); note the variance is already std^2
ax[0,1].plot(f_hat + 2*np.sqrt(k_hat_var[1,1] + k_hat_var[2,2]),'k:')
ax[0,1].plot(f_hat - 2*np.sqrt(k_hat_var[1,1] + k_hat_var[2,2]),'k:')
ax[0,1].set_ylim([0,0.07])                           # y axis range

t_hat = k_hat[0] + k_hat[3]*t1                       # fitted trend
t_hat -= np.mean(t_hat)                              # normalize to the zero-line
ax[1,0].plot(t_hat,'k-')                             # plot fitted trend
ax[1,0].plot(t_hat + 2*np.sqrt(k_hat_var[3,3]),'k:') # upper 95% CI
ax[1,0].plot(t_hat - 2*np.sqrt(k_hat_var[3,3]),'k:') # lower  95% CI
ax[1,0].plot(np.zeros(len(t_hat)),'k-')              # add 0 line
ax[1,0].set_ylabel('ozone [ppb]',fontsize=15)        # y axis label
ax[1,0].set_ylim([-0.001,0.001])                     # y axis range

ear = e[1:]-phi_hat*e[:-1]                   # subtract the autoregression
ax[1,1].plot(np.mean(y1)+ear,'k-')           # plot the residuals
ax[1,1].set_ylim([0,0.07])                   # y axis range
plt.show()
```

Exercise

Decompose other variables in the ozone dataset into trend, seasonal, noise components

Do any variables have a significant trend after controlling for seasonality and autocorrelation?

Try looking for other periodicities in the data, recall you need two observations per cycle

Time series regression

```r
x1  <- d$temp      #extract temperature variable
x2  <- d$windsp    #extract wind speed
x3  <- d$winddir   #extract wind direction
fitx1   <- gls(y1 ~ sinf + cosf + x1,correlation=corAR1(), method='ML')    #fit regression with seasonal terms, one inpu
fitx123 <- gls(y1 ~ sinf + cosf + x1 + x2 + x3,correlation=corAR1(), method='ML')   #fit linear regression with seasonal
data.frame(Model=c('fitx1','fitx123'),
           R2=c(round(cor(y1,predict(fitx1))^2,3)*100, round(cor(y1,predict(fitx123))^2,3)*100))
summary(fitx1)
summary(fitx123)        #display summary of the fitted model
```

```
Generalized least squares fit by maximum likelihood
  Model: y1 ~ sinfnrm + cosfnrm + x1nrm
  Data: NULL
        AIC       BIC   logLik
  -22051.98 -22015.87 11031.99

Correlation Structure: AR(1)
 Formula: ~1
 Parameter estimate(s):
      Phi
0.7687963
```

Model	R2
fitx1	37.4
fitx123	44.2

```
x1 = np.array(d['temp'])[ind]    # Extract temperature
x2 = np.array(d['windsp'])[ind]  # Extract wind speed
x3 = np.array(d['winddir'])[ind] # Extract wind direction
robjects.globalenv["x1"] = robjects.FloatVector(x1)# Add temp to R environment as a FloatVector
robjects.globalenv["x2"] = robjects.FloatVector(x2)# Add wind speed to R environment as a FloatVector
robjects.globalenv["x3"] = robjects.FloatVector(x3)# Add wind dir to R environment as a FloatVector
#-- fit regression with seasonal terms, one input, with autoregressive correlation matrix
fitx1 = nlme.gls(r.formula("y1 ~ sinf + cosf + x1"), correlation=nlme.corAR1(),method='ML')
# Fit linear regression with seasonal cycle and 3 independent variables; via maximum likelihood method='ML'
fitx123 = nlme.gls(r.formula("y1 ~ sinf + cosf + x1 + x2 + x3"), correlation=nlme.corAR1(),method='ML')
df = pd.DataFrame(data={
    'model':['fitx1','fitx123'],
    'R2':[np.round((np.corrcoef(y1,r.predict(fitx1))[0,1])**2,3)*100,
          np.round((np.corrcoef(y1,r.predict(fitx123))[0,1])**2,3)*100]},
                  columns=['model','R2'])
display(df) #display R2 values
print r.summary(fitx1)           # Display summary of the fitted model
print r.summary(fitx123)         # Display the summary of the model
```

Standardizing variables for interpretation

```
x1nrm  <- scale(d$temp)      #scale() subtracts the mean and divides by the sd
x2nrm  <- scale(d$windsp)
x3nrm  <- scale(d$winddir)
sinfnrm<- scale(sinf)        #scale the periodic terms and can interpret the magnitudes relative to covariates
cosfnrm<- scale(cosf)
fitx1nrm   <- gls(y1 ~ sinfnrm + cosfnrm + x1nrm,correlation=corAR1(), method='ML')                  #fit regression w
fitx123nrm <- gls(y1 ~ sinfnrm + cosfnrm + x1nrm + x2nrm + x3nrm,correlation=corAR1(), method='ML')   #fit linear regre
data.frame(Model=c('fitx1nrm','fitx123nrm'),
           R2=c(round(cor(y1,predict(fitx1nrm))^2,3)*100, round(cor(y1,predict(fitx123nrm))^2,3)*100))
summary(fitx1nrm)
summary(fitx123nrm)        #display summary of the fitted model
```

```
Generalized least squares fit by maximum likelihood
  Model: y1 ~ sinfnrm + cosfnrm + x1nrm
  Data: NULL
        AIC       BIC   logLik
  -22051.98 -22015.87 11031.99

Correlation Structure: AR(1)
 Formula: ~1
 Parameter estimate(s):
      Phi
0.7687963

Coefficients:
                 Value    Std.Error  t-value p-value
(Intercept) 0.025897857 0.0005009692 51.69550  0.0000
sinfnrm     0.000560884 0.0005104487  1.09881  0.2719
cosfnrm     0.007588450 0.0005093522 14.89824  0.0000
x1nrm       0.000174678 0.0002532121  0.68985  0.4903
```

Model	R2
fitx1nrm	37.4
fitx123nrm	44.2

*notice the R2 did not change

```
x1nrm = (x1-x1.mean())/x1.std()    # scale temp
x2nrm = (x2-x2.mean())/x2.std()    # scale wind speed
x3nrm = (x3-x3.mean())/x3.std()    # scale wind direction
sinfnrm = (sinf-sinf.mean())/sinf.std()
cosfnrm = (cosf-cosf.mean())/cosf.std()
robjects.globalenv["x1nrm"] = robjects.FloatVector(x1nrm)# Add temp to R environment as a FloatVector
robjects.globalenv["x2nrm"] = robjects.FloatVector(x2nrm)# Add wind speed to R environment as a FloatVector
robjects.globalenv["x3nrm"] = robjects.FloatVector(x3nrm)# Add wind dir to R environment as a FloatVector
robjects.globalenv["sinfnrm"] = robjects.FloatVector(sinfnrm)# Add scaled sine to R environment
robjects.globalenv["cosfnrm"] = robjects.FloatVector(cosfnrm)# Add scaled cos to R environment
#-- fit regression with seasonal terms, one input, with autoregressive correlation matrix
fitx1nrm = nlme.gls(r.formula("y1 ~ sinfnrm + cosfnrm + x1nrm"),
                    correlation=nlme.corAR1(),method='ML')
# Fit linear regression with seasonal cycle and 3 independent variables; via maximum likelihood method='ML'
fitx123nrm = nlme.gls(r.formula("y1 ~ sinfnrm + cosfnrm + x1nrm + x2nrm + x3nrm"),
                      correlation=nlme.corAR1(),method='ML')
df = pd.DataFrame(data={
    'model':['fitx1nrm','fitx123nrm'],
    'R2':[np.round((np.corrcoef(y1,r.predict(fitx1nrm))[0,1])**2,3)*100,
          np.round((np.corrcoef(y1,r.predict(fitx123nrm))[0,1])**2,3)*100]},
                 columns=['model','R2'])
display(df) #display R2 values
print r.summary(fitx1nrm)           # Display summary of the fitted model
print r.summary(fitx123nrm)         # Display the summary of the model
```

Exercise

Fit a model that includes a trend, seasonal term, and one covariate, with an autoregressive correlation matrix

Standardize the variables and rank them in terms of magnitude of standardized ‘effect size’, as well as % uncertainty (s.e./effect size)

Try fitting additional variables in the data
(e.g. can you relate temperature to windsp after controlling for seasonality and autocorrelation?)

Simple Model Selection

$$p(\mathbf{y}|\theta, M, \sigma^2) = \prod_{i=1}^{n} p(\mathbf{y}_i|\theta, M, \sigma^2) = \frac{1}{(2\pi)^{\frac{n}{2}}|\mathbf{\Sigma}|^{\frac{n}{2}}} \exp\left(-\frac{1}{2}\mathbf{e}'\mathbf{\Sigma}^{-1}\mathbf{e}\right)$$

$$\mathbf{e} = \mathbf{y} - f(\hat{\theta}, \mathbf{X}, M)$$

$$\text{BIC} = -2\log p(\mathbf{y}|\hat{\theta}, M, \hat{\sigma}^2) + k\log n$$

$$\text{AIC} = -2\log p(\mathbf{y}|\hat{\theta}, M, \hat{\sigma}^2) + 2k$$

R^2 always increases with the number of parameters for linear models

The 'best' model is the one that minimizes the BIC or AIC

R and Python readily provide these quantities

More interpretable than p.values for model comparison

Model Selection

$$y_i = \beta_0 + \beta_1 x_{1,i} + \beta_2 x_{2,i} +, \cdots, + \beta_n x_{n,i} + e_i \qquad \mathbf{y} = \beta \mathbf{X} + \mathbf{e}$$

```
data.frame(
    model= c('fitx1','fitx123'),                                    #model labels
    R2   = c(cor(y1,predict(fitx1))^2, cor(y1,predict(fitx123))^2),  #percent varation explained by the prediction
    logl = c(summary(fitx1)$logLik,   summary(fitx123)$logLik),
    k    = c(summary(fitx1)$dims$p,    summary(fitx123)$dims$p),     #number of parameters in the models
    BIC  = c(summary(fitx1)$BIC,       summary(fitx123)$BIC))        #Bayesian information criterion
```

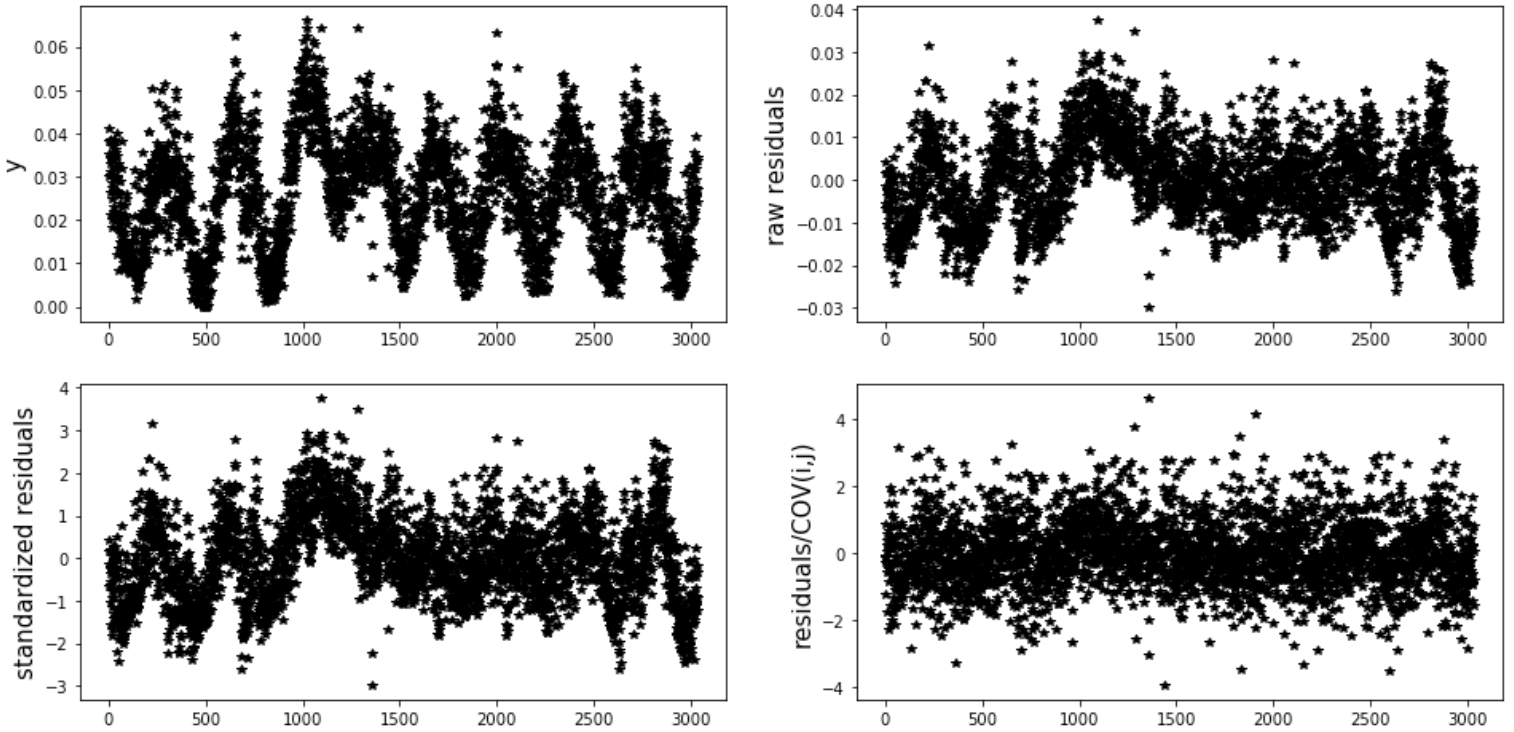
```
# Dictonary of models (model labels,percent varation explained by the predictions (R2),
# Bayesian information criterion (BIC), number of parameters in the models (k))
df = pd.DataFrame(data={
    'model':['fitx1','fitx123'],
    'R2':[(np.corrcoef(y1,r.predict(fitx1))[0,1])**2,(np.corrcoef(y1,r.predict(fitx123))[0,1])**2],
    'logl': [np.squeeze(fitx1.rx2('logLik')),np.squeeze(fitx123.rx2('logLik'))],
    'BIC':[np.squeeze(r.summary(fitx1).rx2('BIC')),np.squeeze(r.summary(fitx123).rx2('BIC'))],
    'k':[np.squeeze(r.summary(fitx1).rx2('dims').rx2('p')),np.squeeze(r.summary(fitx123).rx2('dims').rx2('p'))]},
                columns=['model','R2','logl','k','BIC'])
df #Display
```

model	R2	logl	k	BIC
fitx1	0.3740440	11031.99	4	-22015.87
fitx123	0.4420161	11313.50	6	-22562.85

Model Selection

```
par(mfrow=c(2,2))
plot(y1,pch=8,cex=0.7)
plot(residuals(fitx1, 'response'),pch=8,cex=0.7,ylab='raw residuals')          #'raw' residuals
plot(residuals(fitx1, 'pearson'),pch=8,cex=0.7,ylab='standardized residuals') #normalized to have unit variance
plot(residuals(fitx1, 'normalized'),pch=8,cex=0.7,ylab='residuals/COV(i,j)')  #residuals after removing the autocorrela
```

```
fig, ax = plt.subplots(2,2,figsize=(16,8))          # Set up figure and panels
ax[0,0].plot(y1,'k*')                               # plot data
ax[0,0].set_ylabel('y',fontsize=15)                 # y label
ax[0,1].plot(r.residuals(fitx1, 'response'),'k*')    # 'raw' residuals
ax[0,1].set_ylabel('raw residuals',fontsize=15)
ax[1,0].plot(r.residuals(fitx1, 'pearson'),'k*')     # normalized to have unit variance
ax[1,0].set_ylabel('standardized residuals',fontsize=15)
ax[1,1].plot(r.residuals(fitx1, 'normalized'),'k*') #residuals after removing the autocorrelation
ax[1,1].set_ylabel('residuals/COV(i,j)',fontsize=15)
plt.show()
```



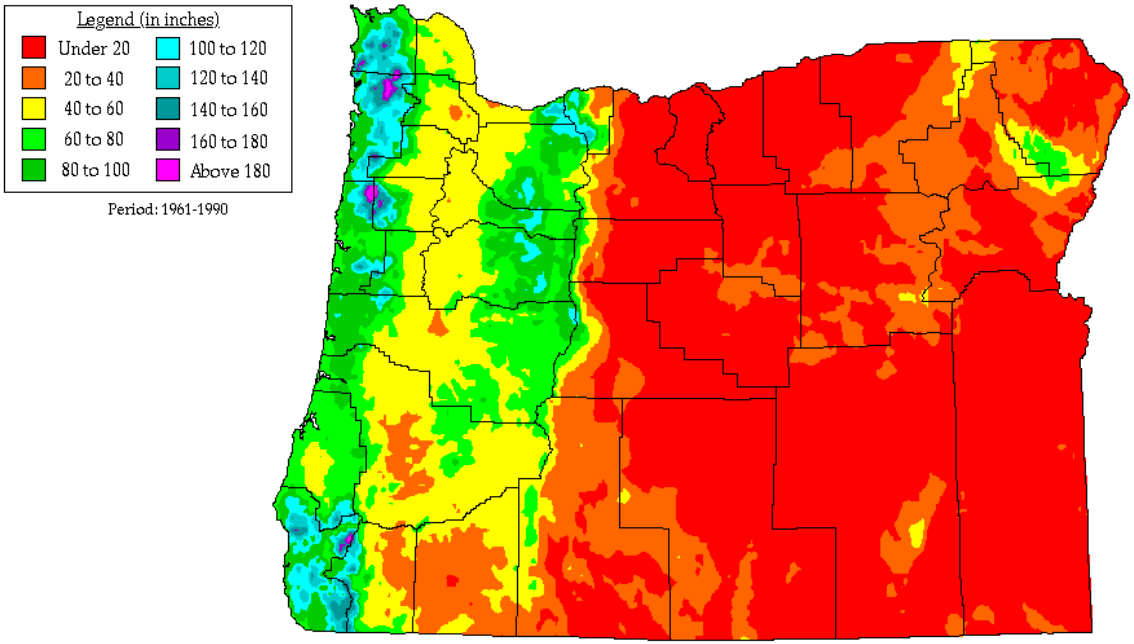
Exercise

Try to find the single best predictor of mean daily ozone concentration, after correcting for seasonality and autocorrelation

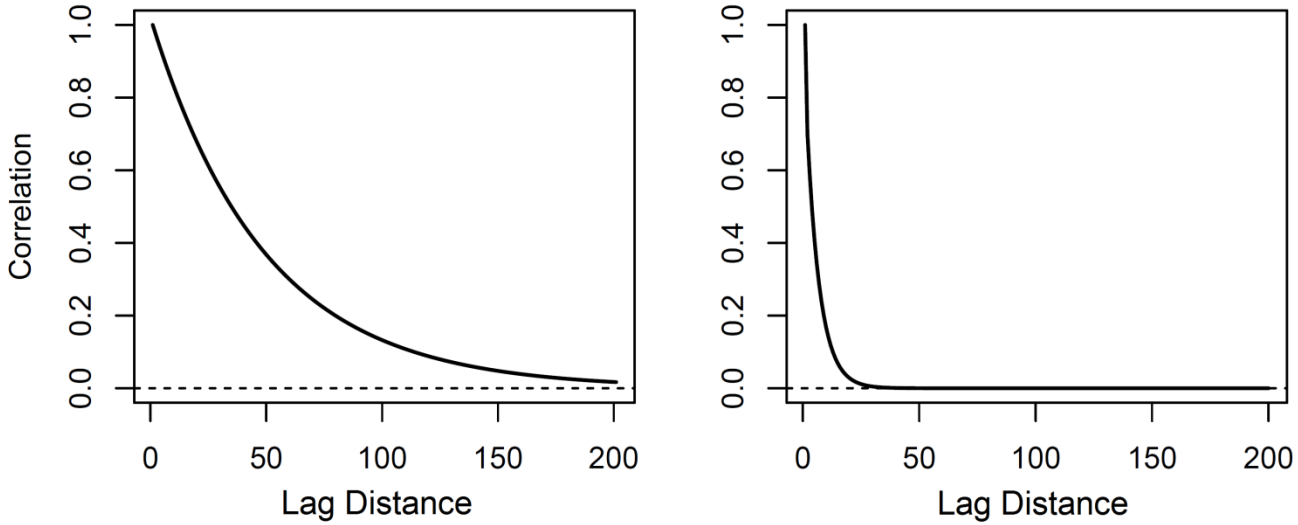
After standardizing variables, fit a few multiple regressions with different input variables rank the models in terms of BIC

Try modeling other variables beyond ozone

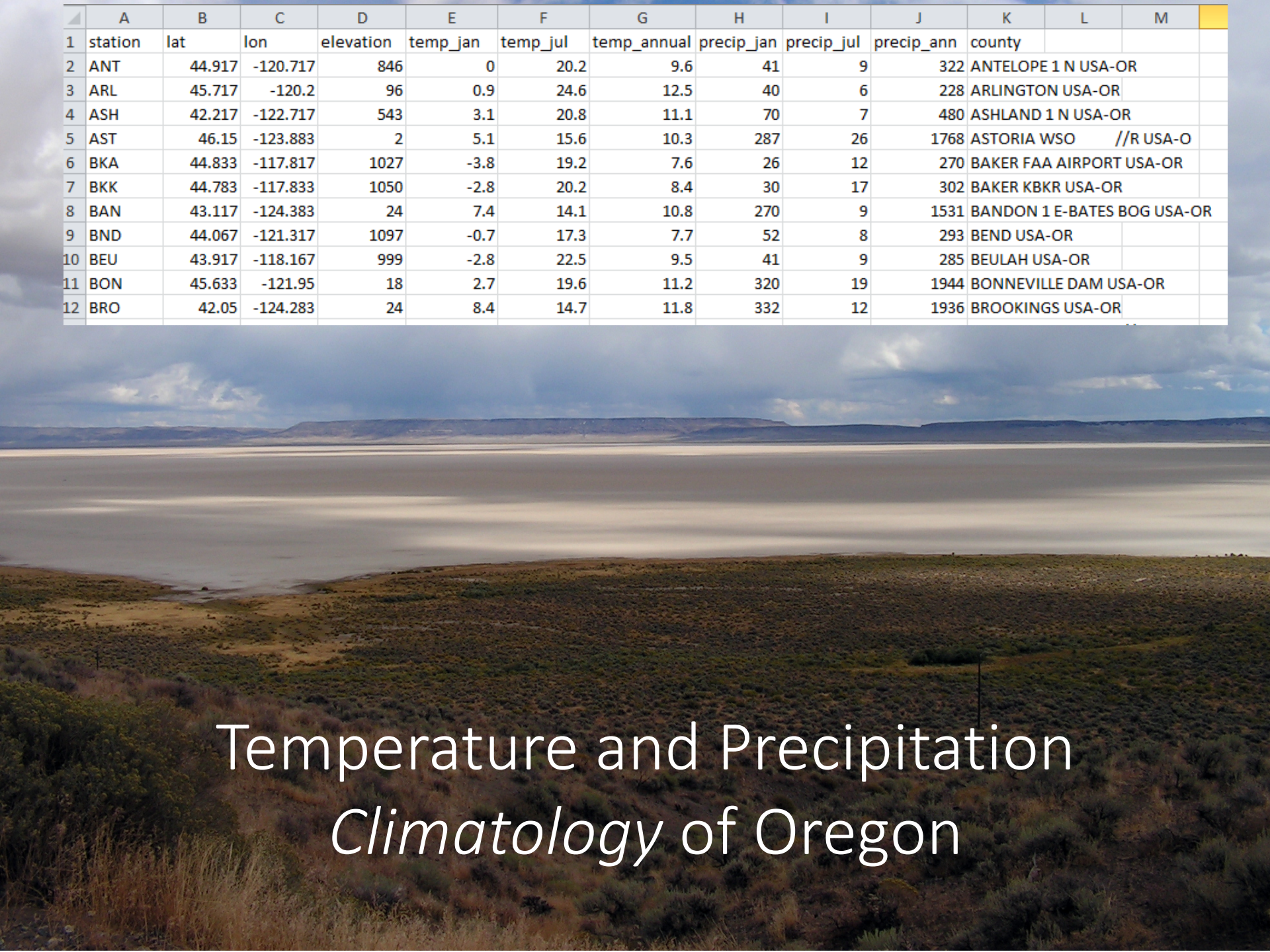
Generalizing autocorrelation to space



*spatial *distance* for spatial models is treated the same as *temporal lag* for time series models



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	station	lat	lon	elevation	temp_jan	temp_jul	temp_annual	precip_jan	precip_jul	precip_ann	county		
2	ANT	44.917	-120.717	846	0	20.2	9.6	41	9	322	ANTELOPE 1 N USA-OR		
3	ARL	45.717	-120.2	96	0.9	24.6	12.5	40	6	228	ARLINGTON USA-OR		
4	ASH	42.217	-122.717	543	3.1	20.8	11.1	70	7	480	ASHLAND 1 N USA-OR		
5	AST	46.15	-123.883	2	5.1	15.6	10.3	287	26	1768	ASTORIA WSO //R USA-O		
6	BKA	44.833	-117.817	1027	-3.8	19.2	7.6	26	12	270	BAKER FAA AIRPORT USA-OR		
7	BKK	44.783	-117.833	1050	-2.8	20.2	8.4	30	17	302	BAKER KBKR USA-OR		
8	BAN	43.117	-124.383	24	7.4	14.1	10.8	270	9	1531	BANDON 1 E-BATES BOG USA-OR		
9	BND	44.067	-121.317	1097	-0.7	17.3	7.7	52	8	293	BEND USA-OR		
10	BEU	43.917	-118.167	999	-2.8	22.5	9.5	41	9	285	BEULAH USA-OR		
11	BON	45.633	-121.95	18	2.7	19.6	11.2	320	19	1944	BONNEVILLE DAM USA-OR		
12	BRO	42.05	-124.283	24	8.4	14.7	11.8	332	12	1936	BROOKINGS USA-OR		



Temperature and Precipitation
Climatology of Oregon

Note on packages for the afternoon

```
#-- Import Required Python Packages
import numpy as np
import rpy2.robjects as robjects
from rpy2.robjects.packages import importr
import statsmodels.api as sm
import matplotlib.pyplot as plt
import pandas as pd
import scipy.spatial
import scipy.linalg
from rpy2.robjects import pandas2ri
pandas2ri.activate()

#-- Import R packages in Python
r = robjects.r
nlme = importr('nlme')
gstat = importr('gstat')
sp = importr('sp')
```

If you don't already have the `gstat` package for R, you need to install it by running the following cell, which uses the `utils` package to install new packages. Uncomment the commands first.

```
# utils = importr("utils")
# utils.install_packages('gstat')
```

Then re-run the import cell.

Generalizing the covariance matrix

d is the distance (or spatial lag) between points *i,j*

(can be Euclidean, or calculated from a map projection)

$$\boldsymbol{\Sigma}_{exp} = \sigma^2 \begin{bmatrix} 1 & e^{-rd_{1,2}} & e^{-rd_{1,3}} & \dots & e^{-rd_{1,n}} \\ e^{-rd_{2,1}} & 1 & e^{-rd_{2,3}} & \ddots & e^{-rd_{2,n}} \\ e^{-rd_{3,1}} & e^{-rd_{3,2}} & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & e^{-rd_{n-1,n}} \\ e^{-rd_{n,1}} & e^{-rd_{n,2}} & \dots & e^{rd_{n,n-1}} & 1 \end{bmatrix}$$

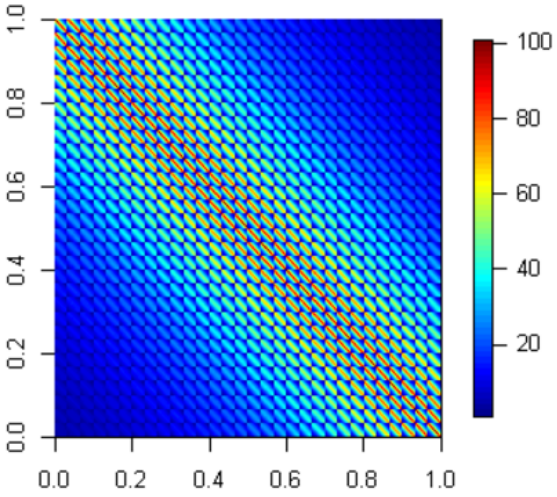
We use the exponential correlation numerical reasons, but it closely approximates the autoregressive model

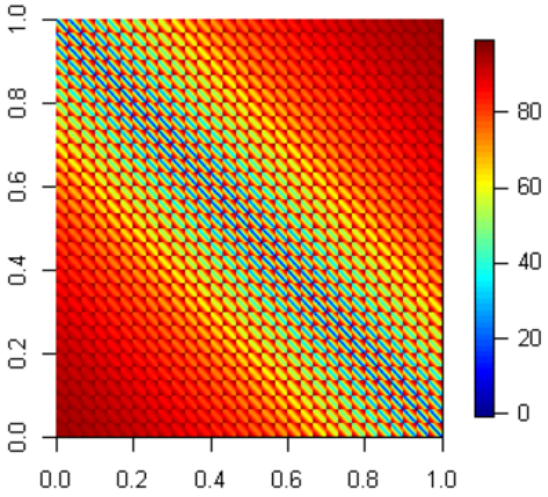
Simulate spatially autocorrelated data

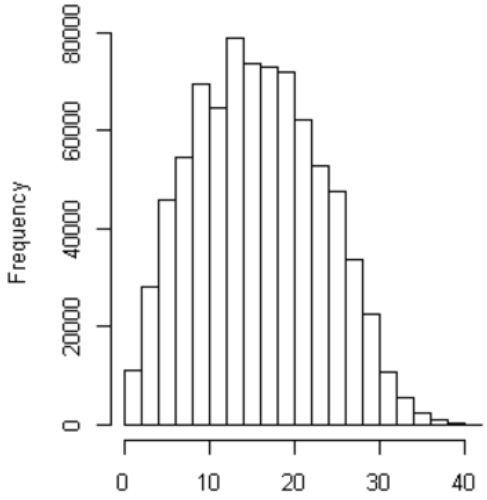
```
M  <- expand.grid(1:30, 1:30)  #Two column matrix with all pair-wise coordinates
n  <- nrow(M)                  #Number of pairs
D  <- as.matrix(dist(M))       #Compute pair-wise distances
r  <- 0.1                      #Decorrelation parameter for autocorrelation function
s  <- 10                       #standard deviation
C  <- exp(-r*D)                #Correlation matrix
S  <- s^2*C                    #Construct covariance matrix
G  <- s^2*(1 - exp(-r*D))      #Analogous variogram matrix
```

$$\gamma_{i,j} = \gamma_0(1 - e^{-rd})$$

```
options(repr.plot.width=10, repr.plot.height=2.5)
par(mfrow=c(1,3))
image.plot(rmat90(S))        #visualize the covariance matrix
image.plot(rmat90(G))        #visualize the variogram matrix
hist(c(D))                   #make a histogram of the pair-wise distances
```





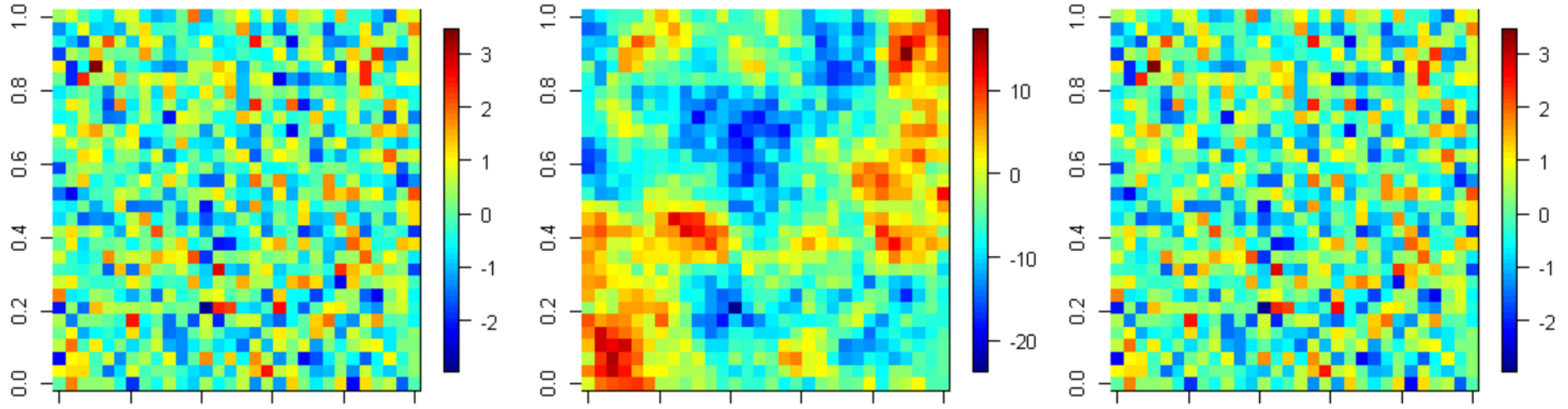


```
# two column matrix with all pair-wise coordinates
M = np.array(np.meshgrid(range(1,31), range(1,31))).reshape(2, 30**2).T
n = len(M)                                                    # number of pairs
D_upper_triangle = scipy.spatial.distance.pdist(M)            # Upper triangle distance matrix
D = scipy.spatial.distance.squareform(D_upper_triangle) # convert to square form distance matrix
r0 = 0.1                                                      # decorrelation parameter
s = 10                                                        # variance
C = np.exp(-r0*D)                                             # Correlation matrix
S = s**2*C                                                    # construct covariance matrix based on distance
G = s**2*(1 - np.exp(-r0*D))                                  # Analogous variogram matrix
```

```
fig, ax = plt.subplots(1,3,figsize=(17,4.5))          # Set up figure and subplots
dist_fig = ax[0].imshow(S,cmap='jet',aspect='auto') # visualize the covariance matrix
fig.colorbar(dist_fig,ax=ax[0])                       # Add colorbar
ax[0].set_title('Covariance Matrix')                  # Add title
corr_fig = ax[1].imshow(G,cmap='jet',aspect='auto') # visualize the variogram matrix
fig.colorbar(corr_fig,ax=ax[1])                       # Add colorbar
ax[1].set_title('Variogram Matrix')                   # Add title
ax[2].hist(D.flatten(),20,edgecolor='k',facecolor='none',linewidth=1.5) # make a histogram of the pair-wise distances
ax[2].set_ylabel('Frequency',fontsize=15)             # add y-label
ax[2].set_title('Distribution of Distances')          # add title
plt.show()
```

Simulate spatially autocorrelated data

```
L  <- chol(S)                      #Cholesky factorization of the covariance matrix
Li <- solve(L)                     #Inverse of the triangular matrix for back-transform
x0  <- matrix(rnorm(n), ncol=n)  #iid random data
x   <- x0%*%L                      #transform the random data by the factored covariance matrix to generate spatial correlation
x00 <- x%*%Li                      #transform back to random data by multiplying by the inverse of the factored covariance matrix

options(repr.plot.width=9, repr.plot.height=2.5)
par(mfrow=c(1,3),mar=c(5,2,2,2))
image.plot(as.matrix(xtabs(t(x0)  ~ M[,1] + M[,2])))   #use xtabs() to fill in matrix elements with dimensions M[,1] and M[,2]
image.plot(as.matrix(xtabs(t(x)   ~ M[,1] + M[,2])))
image.plot(as.matrix(xtabs(t(x00) ~ M[,1] + M[,2])))
```



$$\mathrm{E}(\mathbf{x}\mathbf{x}^T) = \mathrm{E}((\mathbf{L}\mathbf{x}_0)(\mathbf{L}\mathbf{x}_0)^T) = \mathrm{E}(\mathbf{L}\mathbf{x}_0\mathbf{x}_0^T\mathbf{L}^T) = \mathbf{L}\mathrm{E}(\mathbf{x}_0\mathbf{x}_0^T)\mathbf{L}^T$$

$$\mathrm{E}(\mathbf{x}_0\mathbf{x}_0^T) = \sigma^2\mathbf{I}$$

$$\mathrm{E}(\mathbf{x}\mathbf{x}^T) = \sigma^2\mathbf{L}\mathbf{L}^T = \mathbf{S}$$

```
L = scipy.linalg.cho_factor(S,lower=True)              # Cholesky factorization of the covariance matrix
Li = np.linalg.inv(L[0])                               # Inverse of the triangular matrix for back-transform
x0 = np.random.normal(size=(1,n)) # random 1xn matrix drawn from Gaussian distribution
x = np.dot(x0,L[0])     #transform the random data by the factored covariance matrix to generate spatial correlation
x00 = np.dot(x,Li)      #transform back to random data by multiplying by the inverse of the factored covariance matrix
#-- plot
fig, ax = plt.subplots(1,3,figsize=(15,4))     # Set up figure and subplots
x0_fig = ax[0].imshow(x0.reshape(30,30),cmap='jet')  # Show original random matrix
fig.colorbar(x0_fig,ax=ax[0])                  # Add colorbar
x_fig = ax[1].imshow(x.reshape(30,30),cmap='jet')    # Show transformed random data with spatial correlation
fig.colorbar(x_fig,ax=ax[1])                   # Add colorbar
x00_fig = ax[2].imshow(x00.reshape(30,30),cmap='jet')# Show correlated data transformed back to random data
fig.colorbar(x00_fig,ax=ax[2])                 # Add colorbar
plt.show()
```

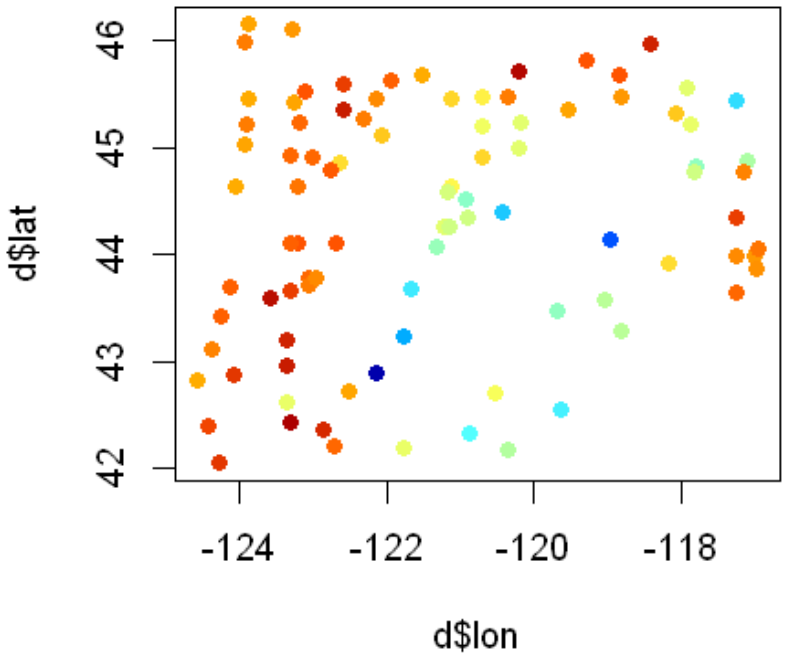
Exercise

Try generating spatial patterns with different decorrelation parameters and look at the resulting patterns

Spatial interpolation

```
d <- read.csv('../Data/oregon_temp_precip.csv',header=TRUE,stringsAsFactors=FALSE)  #read the data from your local repo
cols <- matlab.like(100)[as.numeric(cut(d$temp_annual,breaks=100))]  #set point colors according to value of third vari
options(repr.plot.width=4, repr.plot.height=4)                      #parameters for Jupyter plot window
plot(d$lon,d$lat,col=cols,pch=19)                                   #scatterplot with point colors
```

```
d = pd.read_csv('../Data/oregon_temp_precip.csv')            # Read Oregon data from repository
plt.scatter(d['lon'],d['lat'],c=d['temp_annual'],cmap='jet') # scatter plot, colors based on annual temp
plt.ylabel(r'latitude [$^o$]',fontsize=15)   # y label
plt.xlabel(r'longitude [$^o$]',fontsize=15)  # x label
plt.show()
```



Often want to make predictions to 'fill in the map' based on the data properties

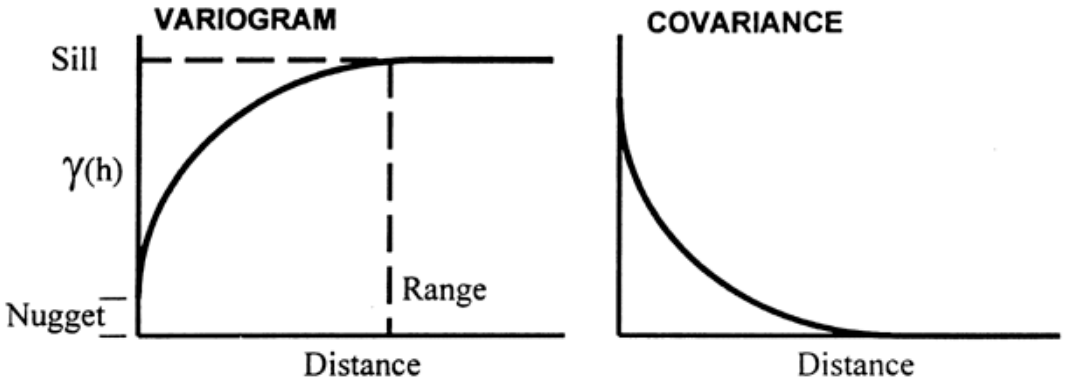
Krigeing considers a linear model and Normally distribution observations

$$y_0 = \sum_{i,j} w_{i,j} y_{i,j}$$

Given $\gamma(d)$, optimal weights for a new location are given by least squares regression

$$\begin{bmatrix} \gamma(y_1,y_1) & \gamma(y_1,y_2) & \dots & \gamma(y_1,y_n) & 1 \\ \gamma(y_2,y_1) & \gamma(y_2,y_2) & \dots & \gamma(y_2,y_n) & 1 \\ \vdots & \vdots & \ddots & \vdots & 1 \\ \gamma(y_n,y_1) & \gamma(y_n,y_2) & \dots & \gamma(y_n,y_n) & 1 \\ 1 & 1 & \dots & 1 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \\ 1 \end{bmatrix} = \begin{bmatrix} \gamma(y_1,y_0) \\ \gamma(y_2,y_0) \\ \vdots \\ \gamma(y_n,y_0) \\ 1 \end{bmatrix}$$

$$\gamma(d) = \gamma_0(1 - e^{-rd}) \qquad C(d) = \sigma^2 e^{-rd}$$



$$\left[\begin{array}{c|c} \boldsymbol{\Gamma} & \mathbf{1} \\ \hline \mathbf{1}^T & 0 \end{array}\right] \left[\begin{array}{c} \mathbf{w} \\ \hline \bar{y} \end{array}\right] = \left[\begin{array}{c} \boldsymbol{\Gamma_0} \\ \hline 1 \end{array}\right]$$

$$\hat{\mathbf{w}} = \boldsymbol{\Gamma}^{-1}\left[\boldsymbol{\Gamma_0} - \bar{y}\mathbf{1}\right]$$

γ is generally unknown so we use numerical optimization

*weights are 'BLUE' according to the 'Gauss-Markov theorem', special case of 'Gaussian process'

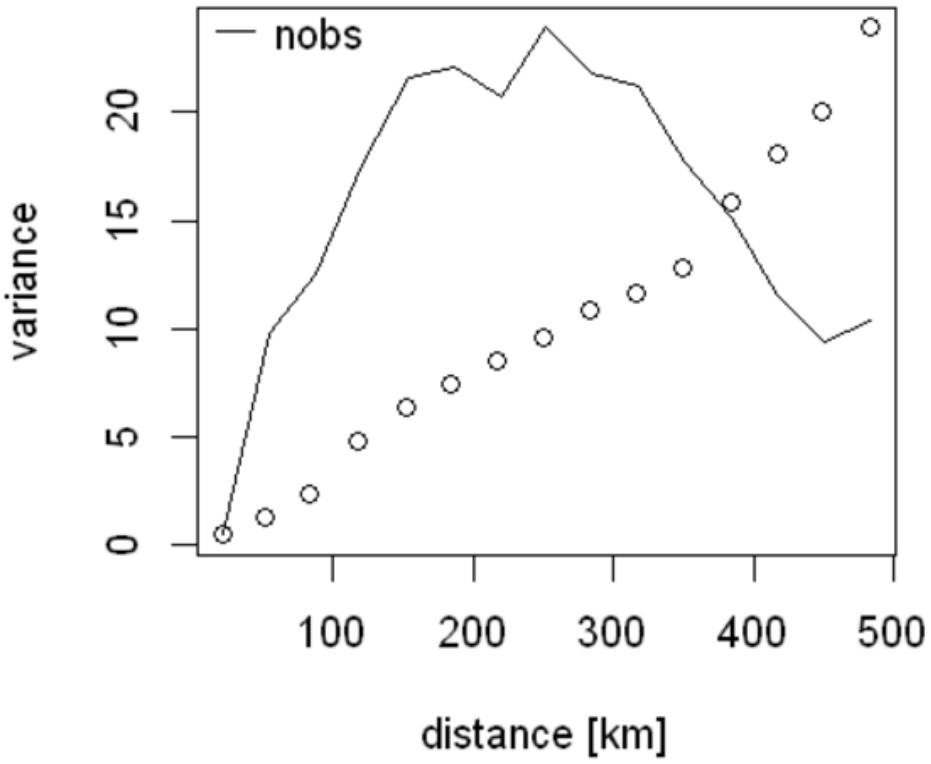
*Krige 1951. A statistical approach to some basic mine valuation problems
J. Chem. Metal. Min Soc. South Africa

Like before, we fit functional forms to the empirical values

```
dsp <- d                                                          #copy data to make spatial data frame
coordinates(dsp) <- ~ lon + lat                                   #define coordinates; changes dsp to class spatial data frame
proj4string(dsp)=CRS("+proj=longlat +datum=WGS84")                #project latitude/longitude according to equal area

d_evg <- variogram(temp_jan ~ 1, dsp, cutoff=500)                 #estimate empirical variogram
plot(d_evg$dist, d_evg$gamma,xlab='distance [km]',ylab='variance') #plot the empirical variogram
par(new=TRUE)                                                     #overplot
plot(d_evg$np,xaxt='n',yaxt='n',xlab='',ylab='',type='l')         #plot the number of observations used to compute empirical var
legend('topleft',bty='n',lwd=1,legend='nobs')                     #label the line
```

$$\hat{\gamma}(d) = \frac{1}{2N_d} \sum_{i,j} (y_i - y_j)^2$$

Observations are binned into pairwise distance bins, with N_d pairs of observations in the bin

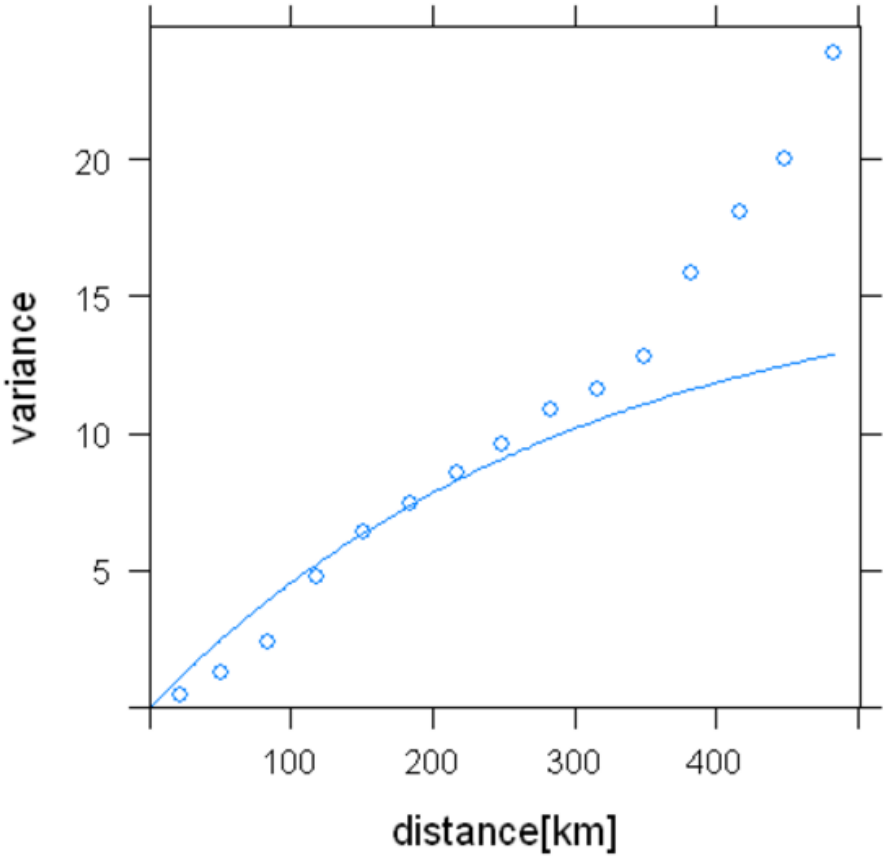


```
robjects.globalenv["lat"] = robjects.FloatVector(d['lat'])  # Add lat to R envrionment
robjects.globalenv["lon"] = robjects.FloatVector(d['lon'])  # Add lon to R envrionment
dsp = pandas2ri.py2ri(d)                                    # Convert d to R DataFrame
proj4string = robjects.r['proj4string']                     # get proj4string function
SpatialPoints = robjects.r['SpatialPoints']                 # get SpatialPoints function
maptools_set = getattr(sp, 'coordinates<-')      # Get 'coordinates<-' command from sp
dsp = maptools_set(dsp,r.formula(' ~ lon + lat'))# Define coordinates; changes dsp to class spatial data frame
key =  sp.CRS("+proj=longlat +datum=WGS84")      # get projection key
dsp = SpatialPoints(dsp, proj4string = key)      # project latitude/longitude according to equal area

robjects.globalenv["temp_jan"] = robjects.FloatVector(d['temp_jan'])
d_evg = gstat.variogram(r.formula("temp_jan ~ 1"), dsp, cutoff=500)                  #estimate empirical variogram
fig, ax1 = plt.subplots()  # set up figure and axes
ax1.scatter(np.array(d_evg.rx2('dist')), np.array(d_evg.rx2('gamma')),label='nobs')  #plot the empirical variogram
ax2 = ax1.twinx() # make y axis on the right side
#--  plot the number of observations used to compute empirical variances
ax2.plot(np.array(d_evg.rx2('dist')),np.array(d_evg.rx2('np')))

ax1.set_xlabel('distance [km]',fontsize=15)
ax1.set_ylabel('variance',fontsize=15)
ax2.set_ylabel('# of observations',fontsize=15)
ax1.legend(loc=2,fontsize=14)
plt.show()
```

Note about using maptools_set: In R, `coordinates(dsp)=~lon+lat` is the same as `dsp <- "coordinates<-"(dsp, formula("~lon+lat"))`. But python doesn't have this sort of embedded assignment, so we define the `"coordinates<-"` command and pass the object and the formula as arguments in one step.

Numerical variogram optimization in R

```
d_vg = fit.variogram(d_evg, model=vgm(psill=20,                          #initial sill value for optimizer
                                      model="Exp",                       #functional form
                                      range=300,                         #initial range value
                                      nugget=0.2*mean(dsp$temp_jan)),    #initial nuggest value
                                      fit.ranges=FALSE)                  #tell the package not to optimize range (helps with converge
plot(d_evg,d_vg,ylab='variance',xlab='distance[km]')                     #plot the empirical and optimized variogram together
```

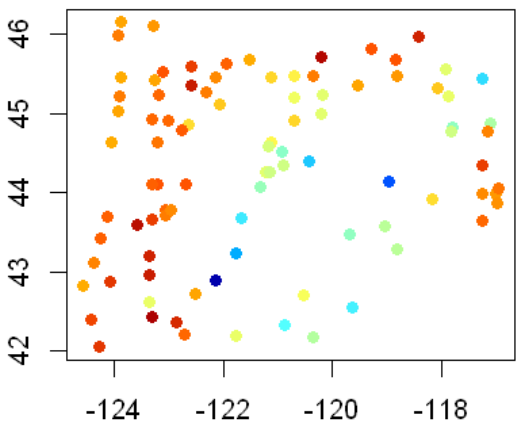


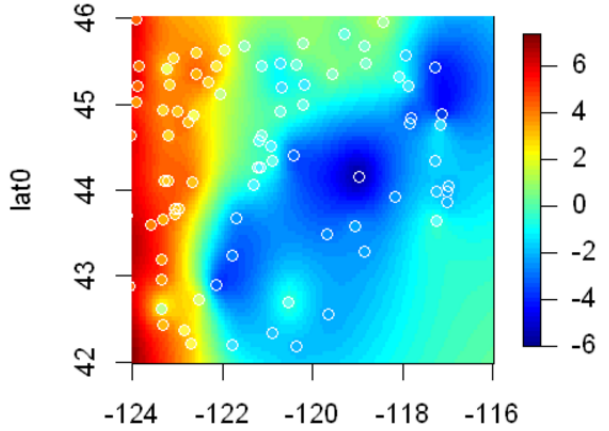
Packages properly weighs by the number of observations in each bin, so points far away have lower influence

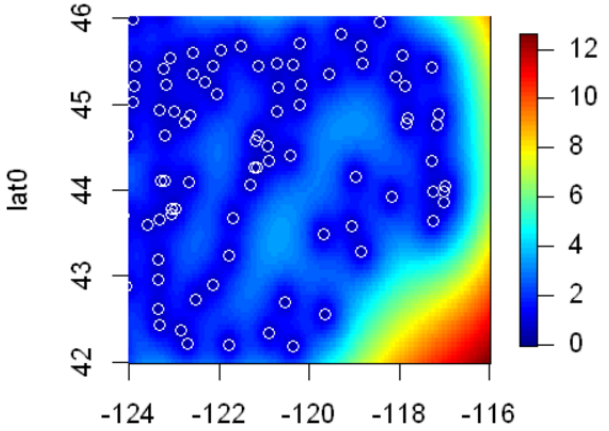
```
d_vg = gstat.fit_variogram(d_evg, model=gstat.vgm(psill=20,     #initial sill value for optimizer
                           model="Exp",                        #functional form
                           range=300,                          #initial range value
                           nugget=0.2*np.mean(d['temp_jan']), #initial nuggest value
                           fit_ranges=False)                   #tell the package not to optimize range (helps with conv

vg_points = gstat.variogramLine(d_vg,np.max(np.array(d_evg.rx2('dist')))) # fill out values from variogram model
plt.scatter(np.array(d_evg.rx2('dist')), np.array(d_evg.rx2('gamma')))    #plot the empirical variogram
plt.plot(np.array(vg_points.rx2('dist')),np.array(vg_points.rx2('gamma')))# plot optimized variogram
plt.xlabel('distance [km]',fontsize=15)
plt.ylabel('variance',fontsize=15)
plt.show()
```

Spatial Interpolation

```
latr <- as.integer(range(dsp$lat))                                    #obserbed latitude range
lonr <- as.integer(range(dsp$lon))                                    #observed longitude range

res  <- 0.05                                                          #resolution of input lat/lon grid [degrees]
lat0 <- seq(from=latr[1], to=latr[2], by=res)                         #input latitudes
lon0 <- seq(from=lonr[1], to=lonr[2], by=res)                         #input longitudes
grd <- expand.grid(lon=lon0,lat=lat0)                                 #input grid
coordinates(grd) <- ~ lon + lat                                       #define coordiantes; makes grd a spatial data frame
proj4string(grd)=CRS("+proj=longlat +datum=WGS84")                    #project lat/lon with equal area
gridded(grd) <- TRUE                                                  #change attribute

d_k = krige(formula=temp_jan ~ 1, locations=dsp, newdata=grd, model=d_vg)  #krigeing prediction; temp_jan ~ 1 specifies the mean,
dkdf <- as.data.frame(d_k)                                            #change back to regular data frame

options(repr.plot.width=8, repr.plot.height=3)
par(mfrow=c(1,2))
image.plot(lon0,lat0,xtabs(dkdf[,3] ~ dkdf[,1] + dkdf[,2]))           #display the matrix; xtabs fills in the matrix element
cols <- matlab.like(100)[as.numeric(cut(d$temp_jan,breaks=100))]      #attach color to matrix values
points(d$lon,d$lat,col=cols,pch=19)                                   #overplot the points with colors for observed values
points(d$lon,d$lat,col='white')                                       #give them a white outline to make them show up

image.plot(lon0,lat0,xtabs(dkdf[,4] ~ dkdf[,1] + dkdf[,2]))           #plot the krigeing variance
points(d$lon,d$lat,col='white')                                       #plot locations as white outline
```







```
min_lat, max_lat = np.min(d['lat']), np.max(d['lat']) #obserbed latitude range
min_lon, max_lon = np.min(d['lon']), np.max(d['lon']) #observed longitude range

res = 0.05                                            #resolution of input lat/lon grid [degrees]
lat0 = np.arange(min_lat,max_lat,res)                 #input latitudes
lon0 = np.arange(min_lon,max_lon,res)                 #input longitudes
expand_grid = robjects.r['expand.grid']               #get expand.grid function from R
grd = expand_grid(lon=lon0,lat=lat0)                  #input grid

maptools_set = getattr(sp, 'coordinates<-')      # Get 'coordinates<-' command from sp
grd = maptools_set(grd,r.formula(' ~ lon + lat'))# Define coordinates; changes dsp to class spatial data frame
key =  sp.CRS("+proj=longlat +datum=WGS84")      # get projection key
grd = SpatialPoints(grd, proj4string = key)      # project latitude/longitude according to equal area
gridded_func = getattr(sp, 'gridded<-')          # Get 'Gridded<-' command from sp
grd = gridded_func(grd,True)

#-- krigeing prediction; temp_jan ~ 1 specifies the mean, locations defines
#-- observation points, newdata represents prediction points
d_k = gstat.krige(formula=r.formula("temp_jan ~ 1"), locations=dsp, newdata=grd, model=d_vg)
dk_data = np.array(d_k.slots['data']).T          # Extracted interpolated data and take transpose to match coord shape
dk_coords = np.array(d_k.slots['coords'])        # Extract coordinates
fig, ax = plt.subplots(1,2,figsize=(15,6))       # Set up figure and subplots
#-- plot interpolated field
int_fig = ax[0].scatter(dk_coords[:,0],dk_coords[:,1],c=dk_data[:,0],cmap='jet')
fig.colorbar(int_fig,ax=ax[0])                    # Add colorbar
ax[0].scatter(d['lon'],d['lat'],s=50,facecolors='none', edgecolors='white',linewidth=1.5) #add original data points
ax[0].set_xlabel(r'Longitude [$^o$]',fontsize=15) # Add x axis label
ax[0].set_ylabel(r'Latitude [$^o$]',fontsize=15)  # Add y axis label
ax[0].set_title('Interpolated Field',fontsize=14)
#-- plot variance
var_fig = ax[1].scatter(dk_coords[:,0],dk_coords[:,1],c=dk_data[:,1],cmap='jet')
fig.colorbar(var_fig,ax=ax[1])                    # Add colorbar
ax[1].scatter(d['lon'],d['lat'],s=50,facecolors='none', edgecolors='white',linewidth=1.5) #add original data points
ax[1].set_xlabel(r'Longitude [$^o$]',fontsize=15) # Add x axis label
ax[1].set_ylabel(r'Latitude [$^o$]',fontsize=15)  # Add y axis label
ax[1].set_title('Variance',fontsize=14)
plt.show()
```

Exercise

Interpolate other variables in the data by first estimating the empirical variogram and then optimizing a functional form

Try defining a new variable (e.g. representing the difference in temperature between July and January, and interpolate that variable)

Compare empirical variograms to understand differences in spatial properties of signal

Spatial Regression

```
fit <- gls(precip_ann ~ temp_annual, data=d, correlation=corExp(form=~lat+lon,nugget=FALSE),method='ML')
summary(fit)    #print a summary of the fitted model object
```

```
fit2 <- gls(precip_ann ~ temp_annual + elevation, data=d, correlation=corExp(form=~lat+lon,nugget=FALSE),method='ML')
summary(fit2)
```

```
Generalized least squares fit by maximum likelihood
  Model: precip_ann ~ temp_annual
  Data: d
       AIC      BIC    logLik
  1297.666 1307.753 -644.8329

Correlation Structure: Exponential spatial correlation
 Formula: ~lat + lon
 Parameter estimate(s):
   range
6.072546

Coefficients:
                Value Std.Error   t-value p-value
(Intercept) 1825.4297  711.1956  2.566706  0.0119
temp_annual  -71.6271   20.8419 -3.436694  0.0009
```

```
Generalized least squares fit by maximum likelihood
  Model: precip_ann ~ temp_annual + elevation
  Data: d
       AIC      BIC    logLik
  1299.665 1312.273 -644.8323

Correlation Structure: Exponential spatial correlation
 Formula: ~lat + lon
 Parameter estimate(s):
  range
6.00687

Coefficients:
                Value Std.Error   t-value p-value
(Intercept) 1839.7755  834.5722  2.204453  0.0301
temp_annual  -72.8775   40.1626 -1.814559  0.0730
elevation     -0.0087    0.2362 -0.036931  0.9706
```

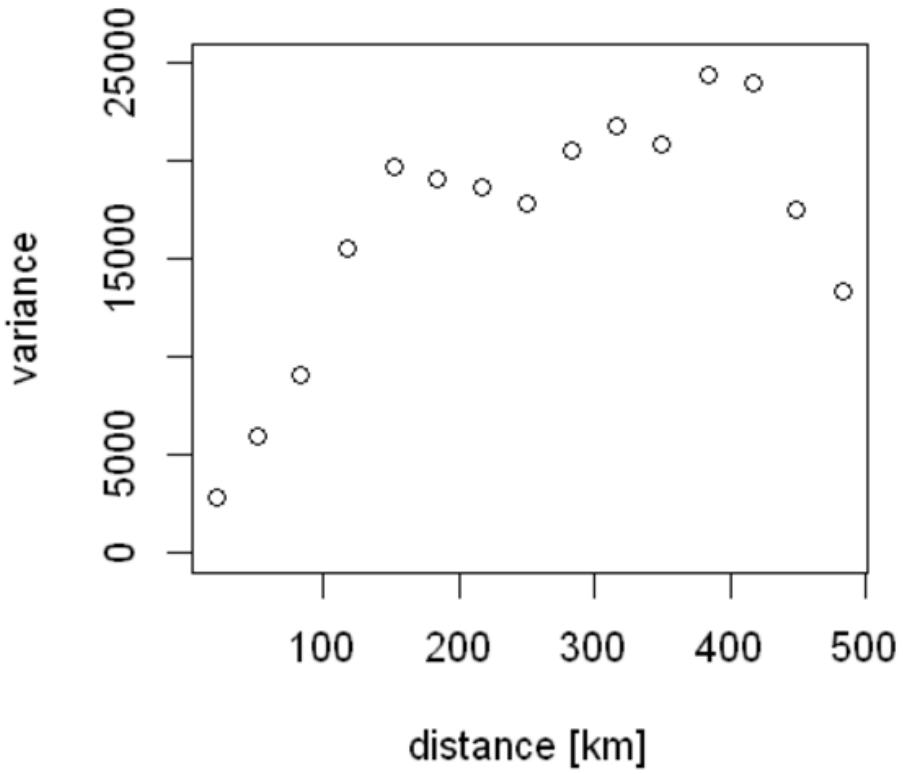
It appears the addition of elevation does not help the model according to the BIC score

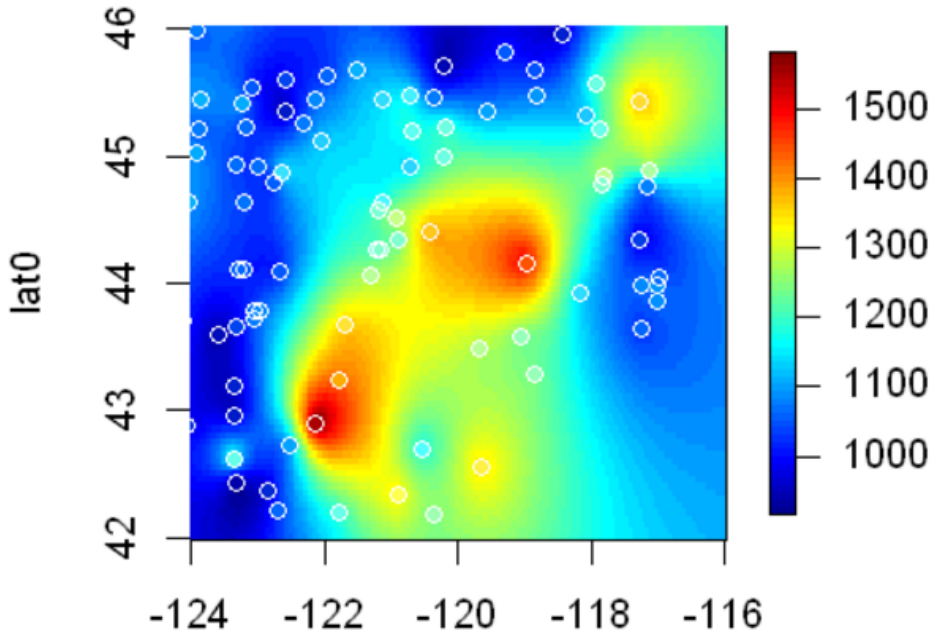
```
robjects.globalenv["precip_ann"] = robjects.FloatVector(d['precip_ann'])  # Add precip to R envrionment
robjects.globalenv["temp_annual"] = robjects.FloatVector(d['temp_annual'])# Add precip to R envrionment
robjects.globalenv["lat"] = robjects.FloatVector(d['lat'])                # Add lat to R envrionment
robjects.globalenv["lon"] = robjects.FloatVector(d['lon'])                # Add lon to R envrionment
#-- fit linear regression model while account for spatial autocorrelation
fit = nlme.gls(r.formula("precip_ann ~ temp_annual"),
               correlation=nlme.corExp(form=r.formula("~lat+lon"),nugget=False),method='ML')
print r.summary(fit)    # print a summary of the fitted model object
```

```
robjects.globalenv["elevation"] = robjects.FloatVector(d['elevation'])  # Add eleavtion to R envrionment
#-- fit regression model to two parameters
fit2 = nlme.gls(r.formula("precip_ann ~ temp_annual + elevation"),
                correlation=nlme.corExp(form=r.formula("~lat+lon"),nugget=False),method='ML')
print r.summary(fit2)     # summary of the fitted model object
```

We can fit a variogram to the predictions

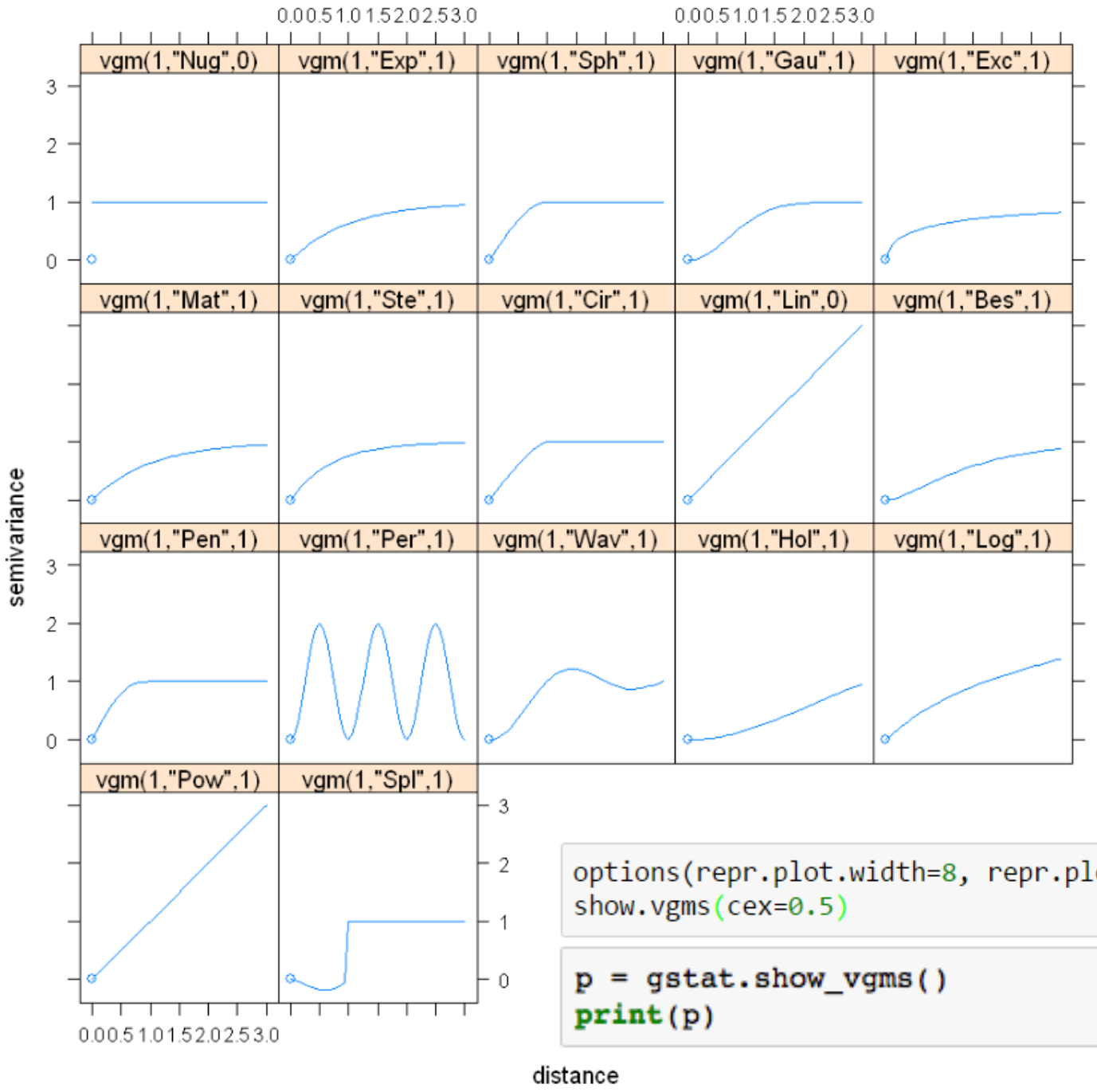
```
dsp$pred <- predict(fit2)
pred_evg <- variogram(pred ~ 1, dsp, cutoff=500)
options(repr.plot.width=4, repr.plot.height=4)
par(mfrow=c(1,1))
plot(pred_evg$dist, pred_evg$gamma,xlab='distance [km]',ylab='variance',ylim=c(0,25000))
```





```
robjects.globalenv["pred"] = robjects.FloatVector(r.predict(fit2))  # Add prediction to R envrionment
pred_evg = gstat.variogram(r.formula("pred ~ 1"), dsp, cutoff=500)  # variogram
#plot the empirical variogram
plt.plot(np.array(pred_evg.rx2('dist')), np.array(pred_evg.rx2('gamma')),'ko')
plt.xlabel('distance [km]',fontsize=15)
plt.ylabel('variance',fontsize=15)
plt.show()
```

```
min_lat, max_lat = np.min(d['lat']), np.max(d['lat']) #obserbed latitude range
min_lon, max_lon = np.min(d['lon']), np.max(d['lon']) #observed longitude range

res = 0.05                                            #resolution of input lat/lon grid [degrees]
lat0 = np.arange(min_lat,max_lat,res)                 #input latitudes
lon0 = np.arange(min_lon,max_lon,res)                 #input longitudes
expand_grid = robjects.r['expand.grid']               #get expand.grid function from R
grd = expand_grid(lon=lon0,lat=lat0)                  #input grid

maptools_set = getattr(sp, 'coordinates<-')       # Get 'coordinates<-' command from sp
grd = maptools_set(grd,r.formula(' ~ lon + lat'))# Define coordinates; changes dsp to class spatial data frame
key =  sp.CRS("+proj=longlat +datum=WGS84")       # get projection key
grd = SpatialPoints(grd, proj4string = key)       # project latitude/longitude according to equal area
gridded_func = getattr(sp, 'gridded<-')           # Get 'Gridded<-' command from sp
grd = gridded_func(grd,True)

pred_vg = gstat.fit_variogram(pred_evg, model=gstat.vgm(psill=20,model="Exp",range=200,
                                                        nugget=0.2*np.mean(d['temp_jan'])), fit_ranges=False)

#-- krigeing prediction; temp_jan ~ 1 specifies the mean, locations defines
#-- observation points, newdata represents prediction points
d_k = gstat.krige(formula=r.formula("pred ~ 1"), locations=dsp, newdata=grd, model=pred_vg)
dk_data = np.array(d_k.slots['data']).T          # Extracted interpolated data and take transpose to match coord shape
dk_coords = np.array(d_k.slots['coords'])        # Extract coordinates
fig, ax = plt.subplots(1,2,figsize=(15,6))       # Set up figure and subplots
#-- plot interpolated field
int_fig = ax[0].scatter(dk_coords[:,0],dk_coords[:,1],c=dk_data[:,0],cmap='jet')
fig.colorbar(int_fig,ax=ax[0])                     # Add colorbar
ax[0].scatter(d['lon'],d['lat'],s=50,facecolors='none', edgecolors='white',linewidth=1.5) #add original data points
ax[0].set_xlabel(r'Longitude [$^o$]',fontsize=15) # Add x axis label
ax[0].set_ylabel(r'Latitude [$^o$]',fontsize=15)  # Add y axis label
ax[0].set_title('Interpolated Field',fontsize=14)
#-- plot variance
var_fig = ax[1].scatter(dk_coords[:,0],dk_coords[:,1],c=dk_data[:,1],cmap='jet')
fig.colorbar(var_fig,ax=ax[1])                     # Add colorbar
ax[1].scatter(d['lon'],d['lat'],s=50,facecolors='none', edgecolors='white',linewidth=1.5) #add original data points
ax[1].set_xlabel(r'Longitude [$^o$]',fontsize=15) # Add x axis label
ax[1].set_ylabel(r'Latitude [$^o$]',fontsize=15)  # Add y axis label
ax[1].set_title('Variance',fontsize=14)
plt.show()
```

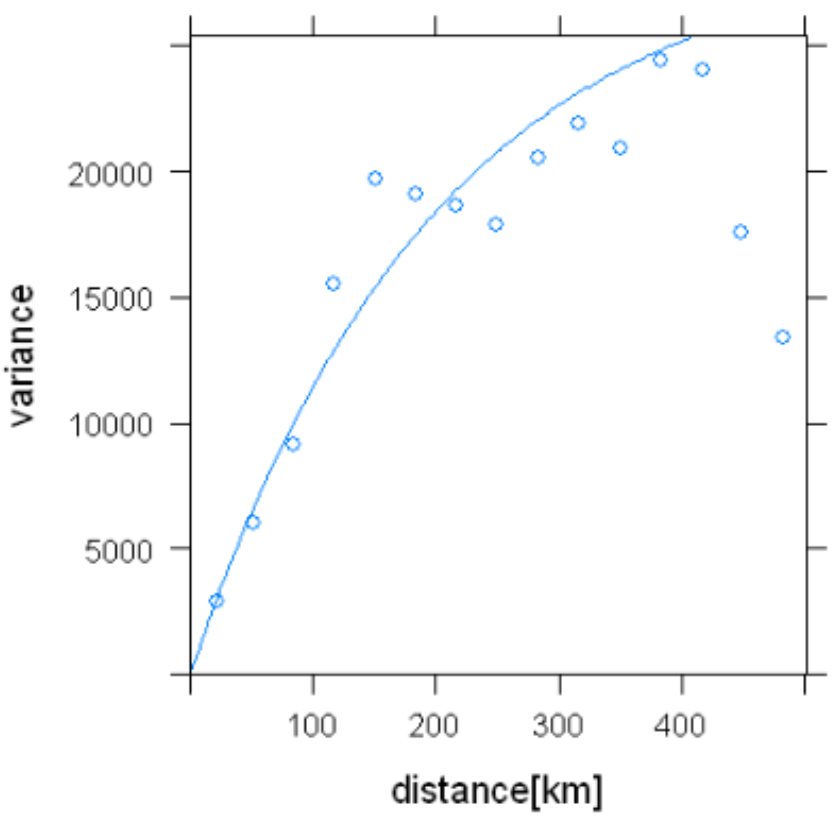


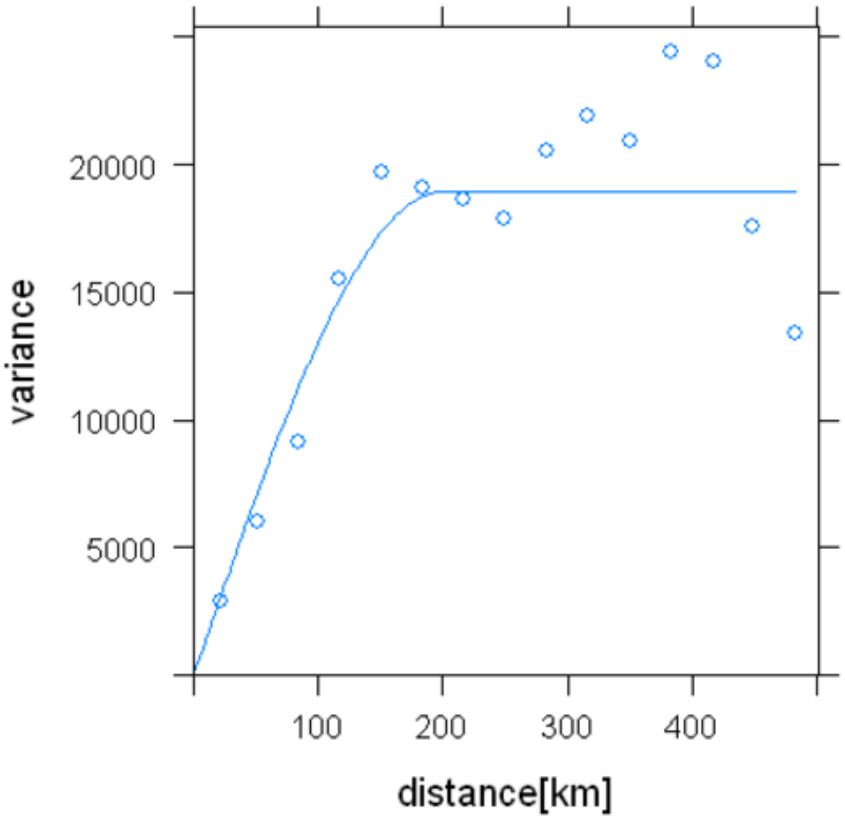

```
options(repr.plot.width=8, repr.plot.height=8)
show.vgms(cex=0.5)
```

```
p = gstat.show_vgms()
print(p)
```

Fitting alternative variogram functions

```
pred_vg1 = fit.variogram(pred_evg, model=vgm(psill=20,model="Exp",range=200,nugget=0.2*mean(dsp$temp_jan)), fit.ranges=FALSE)
pred_vg2 = fit.variogram(pred_evg, model=vgm(psill=20,model="Sph",range=200,nugget=0.2*mean(dsp$temp_jan)), fit.ranges=FALSE)

options(repr.plot.width=4, repr.plot.height=4)

plot(pred_evg,pred_vg1,ylab='variance',xlab='distance[km]')
plot(pred_evg,pred_vg2,ylab='variance',xlab='distance[km]',add=TRUE)
```





```
pred_vg1 = gstat.fit_variogram(pred_evg, model=gstat.vgm(psill=20,model="Exp",range=200,
                                                         nugget=0.2*np.mean(d['temp_jan'])), fit_ranges=False)
pred_vg2 = gstat.fit_variogram(pred_evg, model=gstat.vgm(psill=20,model="Sph",range=200,
                                                         nugget=0.2*np.mean(d['temp_jan'])), fit_ranges=False)
vg_points = {} # make a dictionary for the optimized variogram values for both models
vg_points[0] = gstat.variogramLine(pred_vg1,
                                   np.max(np.array(pred_evg.rx2('dist')))) # fill out values from variogram model
vg_points[1] = gstat.variogramLine(pred_vg2,
                                   np.max(np.array(pred_evg.rx2('dist')))) # fill out values from variogram model
fig,ax = plt.subplots(2,1,figsize=(6,12))
for i in [0,1]: # loop over the subplots and plot both variograms
    ax[i].scatter(np.array(pred_evg.rx2('dist')), np.array(pred_evg.rx2('gamma')))     #plot empirical variogram
    ax[i].plot(np.array(vg_points[i].rx2('dist')),np.array(vg_points[i].rx2('gamma'))) # plot optimized variogram
    ax[i].set_xlabel('distance [km]',fontsize=15)    # set x label
    ax[i].set_ylabel('variance',fontsize=15)         # set y label
plt.show()
```

Model Selection

```
fit1 <- gls(precip_ann ~ temp_annual,              data=d, correlation=corExp(form=~lat+lon,nugget=FALSE),method='ML')
fit2 <- gls(precip_ann ~ elevation,                data=d, correlation=corExp(form=~lat+lon,nugget=FALSE),method='ML')
fit3 <- gls(precip_ann ~ temp_annual + elevation, data=d, correlation=corExp(form=~lat+lon,nugget=FALSE),method='ML')
BIC(fit1,fit2,fit3)  #display the Bayesian Information Criterion of the fits
```

	df	BIC
fit1	4	1307.753
fit2	4	1310.968
fit3	5	1312.273

$$\mathrm{BIC} = -2 \log p(\mathbf{y}|\hat{\theta}, M, \hat{\sigma}^2) + k \log n$$

```
fit1spher  <- gls(precip_ann ~ temp_annual, data=d, correlation=corSpher(form=~lat+lon,nugget=FALSE),method='ML')
fit1exp    <- gls(precip_ann ~ temp_annual, data=d, correlation=corExp(form=~lat+lon,nugget=FALSE),method='ML')
BIC(fit1spher,fit1exp)   #display the Bayesian information crtierion of the fits
```

	df	BIC
fit1spher	4	1306.940
fit1exp	4	1307.753

```
#-- fit the regression with exponential covariance matrix
fit1 = nlme.gls(r.formula("precip_ann ~ temp_annual"),
                correlation=nlme.corExp(form=r.formula("~lat+lon"),nugget=False),method='ML')
fit2 = nlme.gls(r.formula("precip_ann ~ elevation"),
                correlation=nlme.corExp(form=r.formula("~lat+lon"),nugget=False),method='ML')
fit3 = nlme.gls(r.formula("precip_ann ~ temp_annual + elevation"),
                correlation=nlme.corExp(form=r.formula("~lat+lon"),nugget=False),method='ML')
#-- display the Bayesian Information Criterion of the fits
bic = r.BIC(fit1,fit2,fit3)
bic_dic = dict(zip(bic.names, map(list,list(bic)))) # convert to python dictionary
bic_dic['models'] = ['fit1','fit2','fit3']          # Add model info
bic_pd = pd.DataFrame(data=bic_dic,columns=np.concatenate([['models'],bic.names])) #make into Pandas DataFrame
bic_pd # Display BIC table
```

```
#-- fit the regression with exponential covariance matrix
fit1spher = nlme.gls(r.formula("precip_ann ~ temp_annual"),
                     correlation=nlme.corSpher(form=r.formula("~lat+lon"),nugget=False),method='ML')
fit1exp = nlme.gls(r.formula("precip_ann ~ temp_annual"),
                   correlation=nlme.corExp(form=r.formula("~lat+lon"),nugget=False),method='ML')

#-- display the Bayesian Information Criterion of the fits
bic = r.BIC(fit1spher,fit1exp)
bic_dic = dict(zip(bic.names, map(list,list(bic)))) # convert to python dictionary
bic_dic['models'] = ['fit1spher','fit1exp']         # Add model info
bic_pd = pd.DataFrame(data=bic_dic,columns=np.concatenate([['models'],bic.names])) #make into Pandas DataFrame
bic_pd # Display BIC table
```

Exercise

Find the multiple spatial regression that minimizes the BIC

Krige the predictions by estimating the empirical variogram, then fitting a functional form

Wrapping Up

We hope you now:

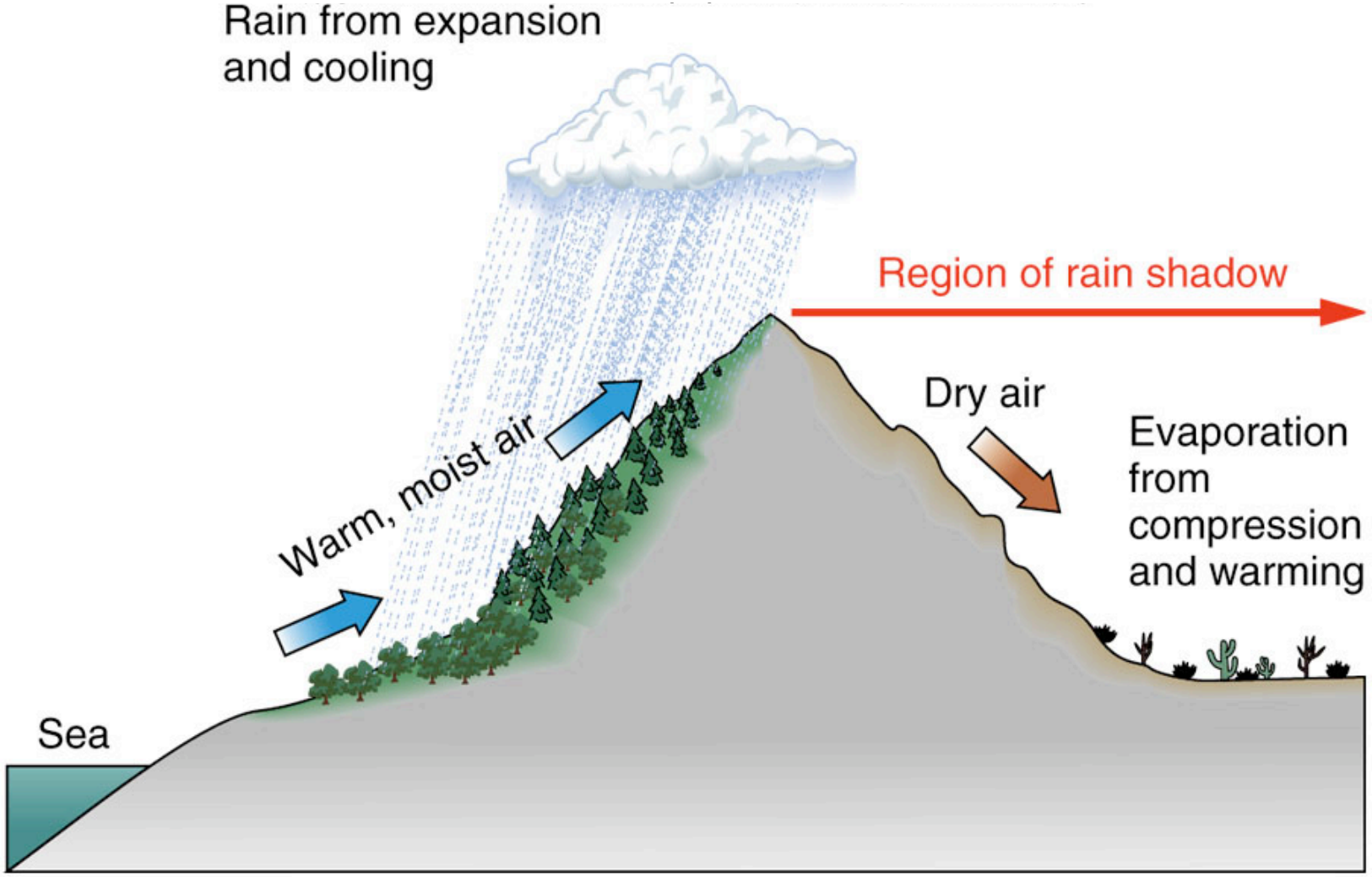
- Understand how patterns in data are generated via autocorrelation
- How to statistically model spatial-temporal correlation and periodicity within basic stats model
- Know R packages for key statistical models, and Python implementation

Final remarks:

- Datasets relatively small, models relatively simple, extended workshop in the works
- Hackathon coming up

Rain Shadows

https://en.wikipedia.org/wiki/Rain_shadow

- 

ADDITIONAL SCIENCE EXERCISE: Derived spatial variables

Given what you've just learned about rain shadows, construct a new spatial variable that may capture the effect of the rain shadow in Oregon (*hint: where is water coming from?*)

Using the statistical tools you've learned today, test the rain shadow variable in a statistical model and determine whether it is associated with Oregon precipitation patterns after controlling for the spatial-temporal properties of the data

Additional Resources

Cressie 2015. *Statistics for spatial data*. John Wiley & Sons.

Cressie and Wikle 2015. *Statistics for spatio-temporal data*. John Wiley & Sons.

Pinheiro and Bates 2006. *Mixed-Effects Models in S and S-PLUS*. Springer Science.

Gelman et al 2014. *Bayesian data analysis*. Chapman & Hall/CRC.