

Chaos, Complexity, and Inference (36-462)

Lecture 4

Cosma Shalizi

24 January 2008

Reconstruction Inferring the attractor from a time series;
powerful in a weird way

Prediction Using the reconstructed attractor to make
forecasts

Reconstruction

What can we learn about the dynamical system from observing a trajectory?

Straightforward approach: assume it's reached an attractor; attractor is a function of the parameters; invert the function

- Function from parameters to attractors can be very ugly
- Assumes we know the dynamics up to parameters

Second problem is bigger!

Will see later an approach (“indirect inference”) for parametric estimation with messy models

A gross distortion of **Takens's Embedding Theorem**

Suppose X_t is a state vector

Suppose the map/flow is sufficiently smooth

Suppose X_t has gone to an attractor, dimension d

Suppose we observe $S_t = g(X_t)$, which again is sufficiently smooth

Pick a time-lag τ and a number of lags k

Time-delay vector $R_t^{(k)} = (S_t, S_{t-\tau}, S_{t-2\tau}, \dots, S_{t-(k-1)\tau})$

THEOREM If $k \geq 2d + 1$, then, for generic g and τ , $S_t^{(k)}$ acts just like X_t , up to a smooth change of coordinates

DETERMINISM

State has d dimensions so knowing d coordinates at once fixes trajectory

OR knowing one variable at d times fixes trajectory (think of Henon map)

Don't get to observe state variables so may need extra observations

Turn out to never need more than $d + 1$ extras

Sometimes don't need the extras

[1]

GEOMETRY

Trajectories are d -dimensional (because they are on the attractor, and current state fixes future states)

So time series are also only d -dimensional, but they might live on a weirdly curved d -dimensional space

Geometric fact (Whitney embedding theorem): any curved d -dimensional space fits into a $2d + 1$ -dimensional ordinary Euclidean space

[2]

Attractor Reconstruction

“It’s inference, Jim, but not as we know it”

Gets information about the larger system generating the data from partial data (inference)

No parametric model form

No nonparametric model form either

No likelihood, no prior

Requires very complete determinism

Reconstructs the attractor up to a smooth change of coordinates

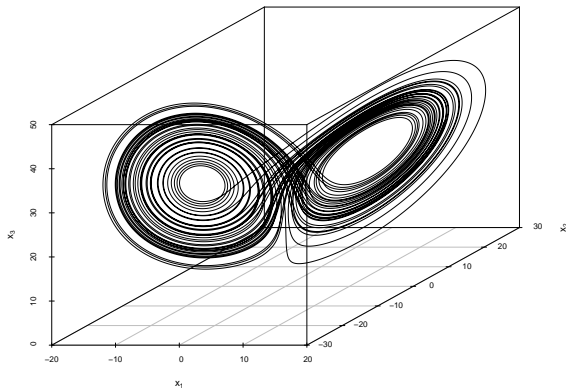
An example of reconstruction Lorenz attractor, $a = 10$, $b = 28$, $c = 8/3$

See last lecture for equations of motion

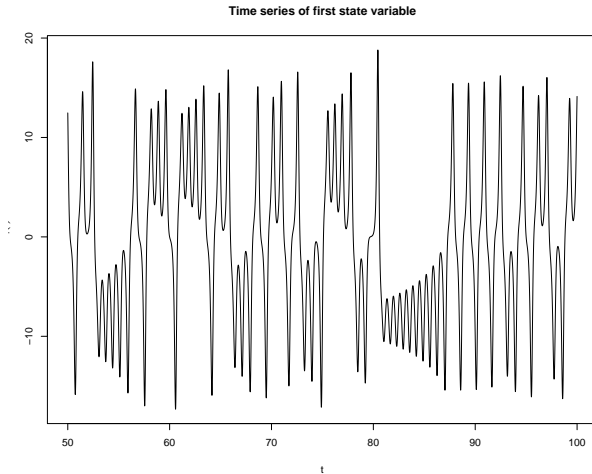
Solved equations in a separate (non-R) program (see website)

```
> lorenz.state.ts = read.csv("lorenz-t50-to-t100-by-1e-3",  
                             col.names=c("t", "x", "y", "z"))  
> scatterplot3d(lorenz.state.ts$x,lorenz.state.ts$y,  
                 lorenz.state.ts$z,type="l",xlab="x",  
                 ylab="y",zlab="z",lwd="0.2")  
> x.ts = lorenz.state.ts$x  
> library(tseriesChaos)  
> scatterplot3d(embedd(x.ts,3,40),type="l",lwd="0.2")
```


Lorenz Attractor: state space

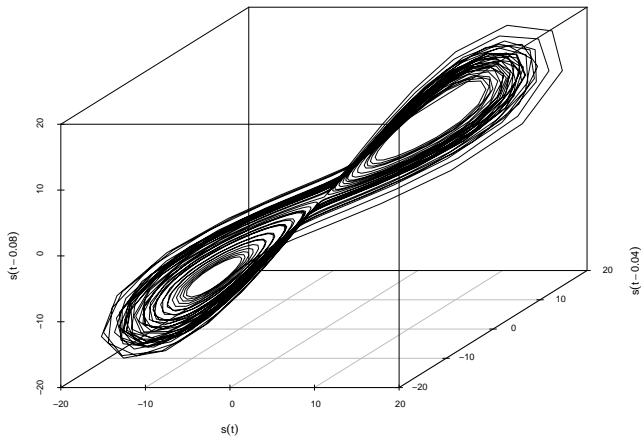


Time series: first coordinate of state



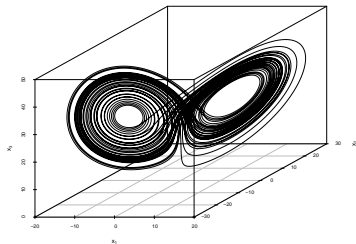
Reconstruction, $k = 3$, $\tau = 0.04$

Lorenz Attractor: time-delay of one state variable

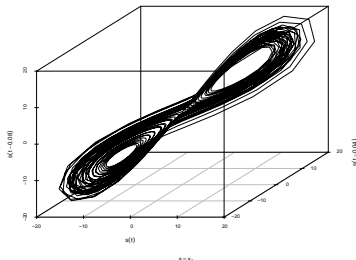


$s = x_1$

Lorenz Attractor: state space



Lorenz Attractor: time-delay of one state variable



Note: reconstruction procedure knew nothing about what the state variables were or what the equations of motion were, it just worked with the original time-series

Coordinate change

Diffeomorphism: the transformation must be continuous, with a continuous derivative; and it must be invertible, and the inverse must also have continuous derivatives

Example: the logistic map is equivalent, up to diffeomorphism, to the Henon map with $b = 0$

$$h_{n+1} = a - h_n^2 \text{ vs. } l_{n+1} = 4rl_n(1 - l_n)$$

set $h_n = c + dl_n$ and similarly for h_{n+1}

then

$$a - (c + dl_n)^2 = c + d(4rl_n(1 - l_n))$$

solve for a , c , d in terms of r

Even worse: **tent map**

$$t_{n+1} = \begin{cases} 2t_n & 0 \leq t_n \leq 1/2 \\ 2(1 - t_n) & 1/2 \leq t_n \leq 1 \end{cases}$$

here diffeomorphism is $I_n = \sin^2 \pi t_n / 2$

If the coordinate change works, then it doesn't matter whether we apply it or the map first

old, new state = X_t, R_t

old, new map = Φ, Ψ

coordinate change = G so $R_t = G(X_t)$

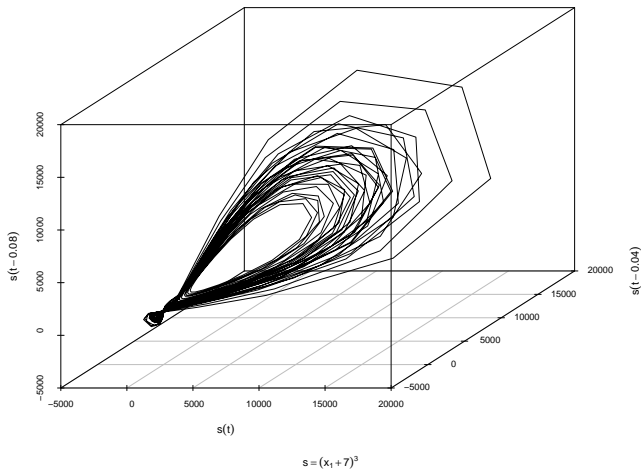
$$\begin{aligned} X_{t+1} &= \Phi(X_t) & R_{t+1} &= \Psi(R_t) \\ R_{t+1} &= G(\Phi(X_t)) &= \Psi(G(X_t)) &= R_{t+1} \end{aligned}$$

$$\begin{array}{ccc} X_t & \xrightarrow{\Phi} & X_{t+1} \\ G \downarrow & & G \downarrow \\ R_t & \xrightarrow{\Psi} & R_{t+1} \end{array}$$

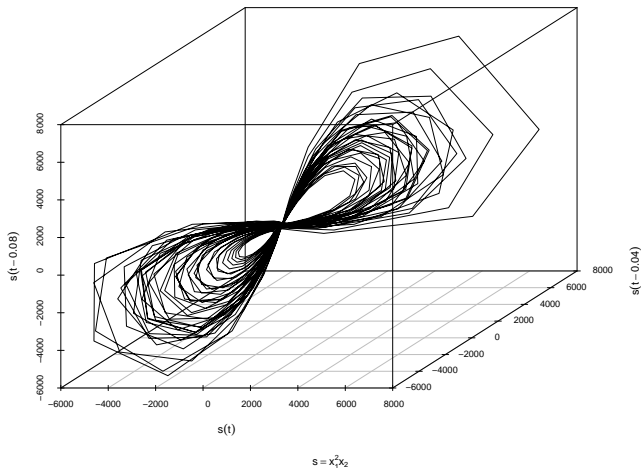
New coordinates are a perfect model of the old coordinates
 time-delay vectors give a model of states, at least on the
 attractor

many quantities (like Lyapunov exponents) aren't affected by
 change of coordinates

Lorenz Attractor: nonlinear observable

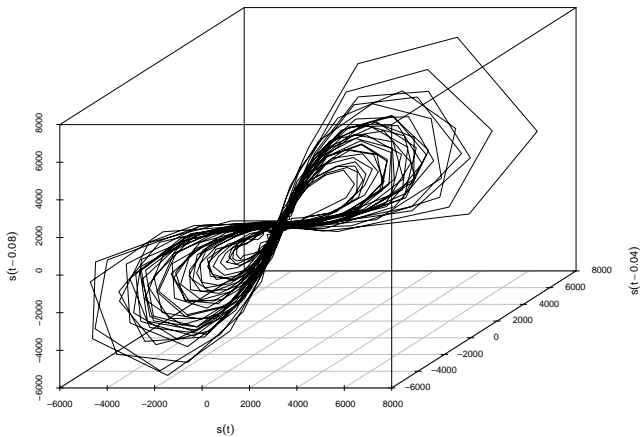


Lorenz Attractor: Nonlinear observable



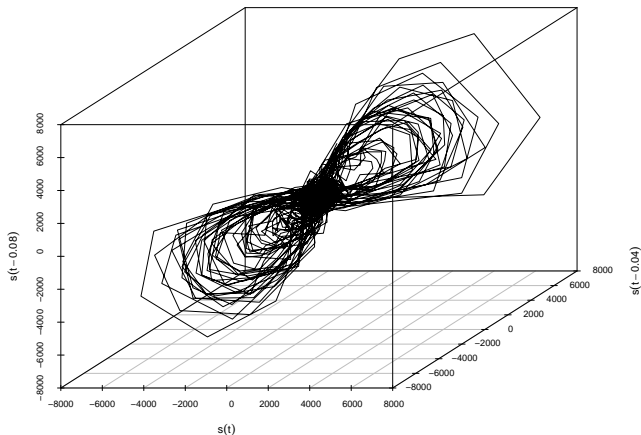
It even works with observation noise

Lorenz Attractor: Nonlinear observable+noise



$$s = x_1^2 x_2 + \varepsilon, \quad \sigma(\varepsilon) = 100$$

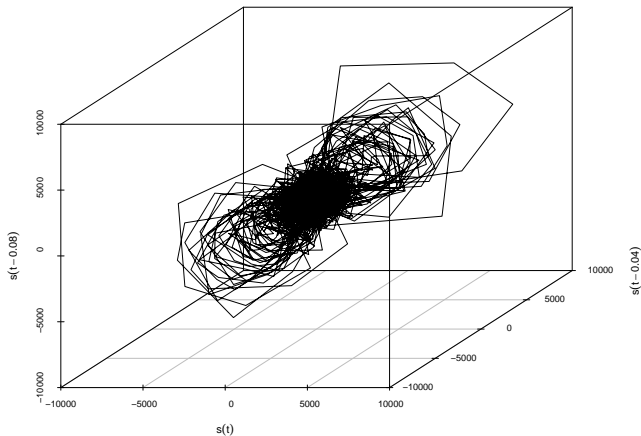
Lorenz Attractor: Nonlinear observable+noise



$$s = x_1^2 x_2 + \epsilon, \quad \sigma(\epsilon) = 400$$

but not too much

Lorenz Attractor: Nonlinear observable+noise



$$s = x_1^2 x_2 + \epsilon, \quad \sigma(\epsilon) = 1000$$

Choice of reconstruction settings

Need to choose k (number of delays, embedding dimension) and τ (delay between observations) — typically the observable is given to us by the problem situation

These involve a certain amount of uncertainty

Software: used `tseriesChaos` from CRAN

Hegger, Kantz, & Schreiber's TISEAN has an R port, `RTisean`, also on CRAN, couldn't get it to work

Choice of delay τ

In principle, almost any τ will do

but if the attractor is periodic, multiples of the period are bad!

In practice, want to try and get as much new information about the state as possible from each observation

⇒ Heuristic 1: make τ the first minimum of the autocorrelation

⇒ Heuristic 2: make τ the first minimum of the mutual information

Will return to heuristic 2 after we explain “mutual information”

Autocorrelation function:

$$\rho(t, s) = \frac{\text{cov}[X_t, X_s]}{\sigma(X_t)\sigma(X_s)}$$

For **weakly stationary** or **second-order stationary** processes, $\rho(t, s) = \rho(|t - s|)$, i.e., depends only on time-lag, not on absolute time

Standard R command: `acf`

Space-time separation plot Distance between two random points from the trajectory will depend on how far apart in time we make the observations

For each $h > 0$, calculate the empirical distribution of

$$\|X_t - Y_{t+h}\|$$

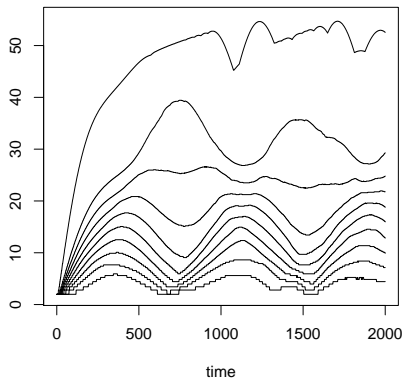
Plot the quantiles as a function of h

Example: logistic map

```
stplot(x.ts, 3, 40, mdt=2000)
```

showing deciles; time in units of 1/1000

Space-time separation plot



note growth (separation) + periodicity

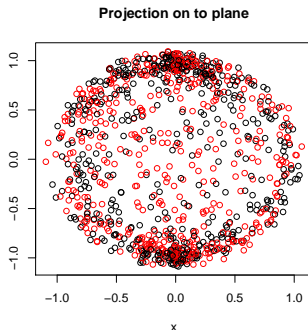
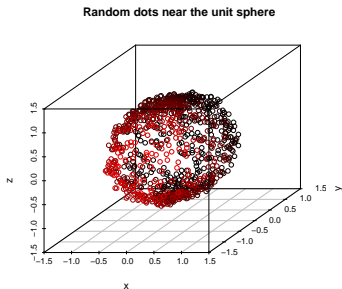
correlations are reasonably decayed at around ≈ 250 , before then observations are correlated because not enough time to disperse around attractor

Choice of embedding dimension k

False neighbor method

Take points which really live in a high-dimensional space and project into a low-dimensional one: many points which are really far apart will have projections which are close by

These are the **false neighbors**



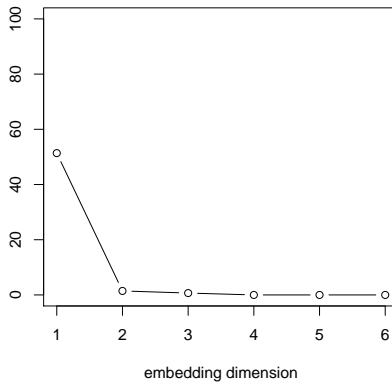
Keep increasing the dimensionality until everything which was a neighbor in k dimensions is still a neighbor in $k + 1$ dimensions

Need to exclude points which are nearby just because the dynamics hasn't had time to separate them — calculate this **Theiler window** from space-time plot

```
plot(false.nearest(x.ts, 6, 40, 250))
```

i.e. use $\tau = 40$ in embedding, consider up to 6 dimensions, and use an Theiler exclusion window of 250 time-steps

False nearest neighbours



Conclusion: $k = 3$ it is

Prediction

Determinism: there is a mapping from old states to new states

Prediction: learn that mapping, then apply it

Work in the reconstructed state space

Nearest-neighbor methods

Also called **method of analogs** in dynamics

Given: points $x_1, x_2, x_3, \dots, x_{n-1}$ and their sequels, x_2, x_3, \dots, x_n

Wanted: prediction of what will happen after seeing new point x

Nearest neighbor: find x_i closest to x ; predict x_{i+1}

k -nearest neighbors: find k points $x_{i_1}, x_{i_2}, x_{i_k}$ closest to x ,
predict the average of $x_{i_1+1}, x_{i_2+1}, \dots, x_{i_k+1}$

Notation: $U_k(x)$ = k nearest neighbors of x

Note: this k is not the k of the embedding dimension, but the phrase “ k -nearest neighbors” is traditional

Computation: finding the nearest neighbors fast is tricky; leave it to a professional

Assume the map Φ is smooth

If x_i is very close to x , then $\Phi(x_i) = x_{i+1}$ will be close to, but not exactly, $\Phi(x)$

$$\Phi(x_i) = \Phi(x) + (x_i - x)\Phi'(x) + \text{small}$$

If $x_{i_1}, x_{i_2}, x_{i_k}$ are all very close to x , then $x_{i_1+1}, x_{i_2+1}, \dots, x_{i_k+1}$ should all be close to $\Phi(x)$

$$\frac{1}{k} \sum_{j \in U_k(x)} \Phi(x_j) = \frac{1}{k} \sum_{j \in U_k(x)} \Phi(x) + \left(\frac{1}{k} \sum_{j \in U_k(x)} x_j - x \right) \Phi'(x) + \text{small}$$

$$\frac{1}{k} \sum_{j \in U_k(x)} \Phi(x_j) - \Phi(x) \approx \left(\frac{1}{k} \sum_{j \in U_k(x)} x_j - x \right) \Phi'(x)$$

$k > 1$: the error averages a bunch of individual terms from the neighbors, which should tend to be smaller than any one of them, *if* they're not too correlated themselves
One reason this works well together with mixing!

As n grows, we get more and more samples along the attractor. If x itself is from the attractor, it becomes more and more likely that we are sampling from a place where we have many neighbors, and hence close neighbors, so the accuracy should keep going up.

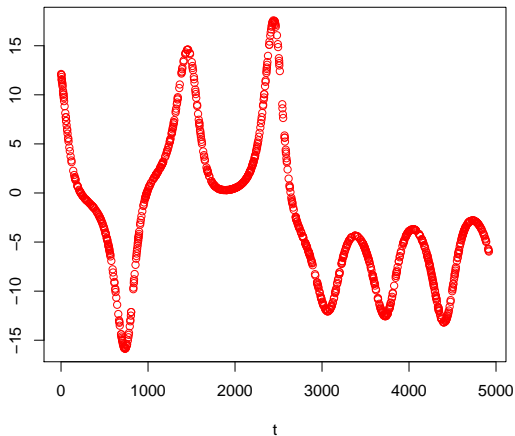
Another reason this works well with mixing

knnflex from CRAN lets you do nearest-neighbor prediction
(as well as classification)

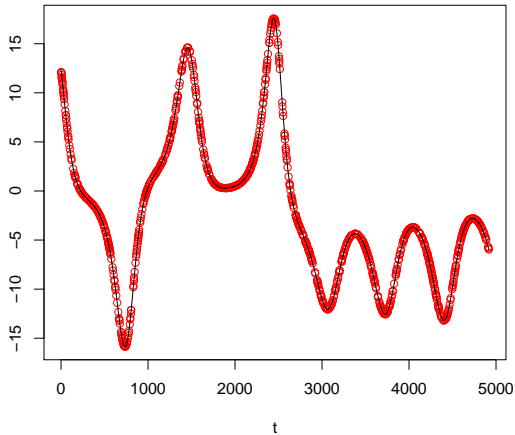
R is weak and refuses to do really big distance matrices

```
> lorenz.rcon = embedd(x.ts[1:5000],3,40)
> nrow(lorenz.rcon)
[1] 4920
> lorenz.dist = knn.dist(lorenz.rcon)
> lorenz.futures = x.ts[2:5001]
> train = sample(1:nrow(lorenz.rcon),0.8*nrow(lorenz.rcon))
> test = (1:nrow(lorenz.rcon))[-train]
> preds = knn.predict(train,test,lorenz.futures,lorenz.dist,
                      k=3,agg.meth="mean")
> plot(test,preds,col="red",xlab="t",ylab="x",type="p",
       main="3NN prediction vs. reality")
> lines(test,lorenz.futures[test])
> plot(test,lorenz.futures[test] - preds,xlab="t",
       ylab="reality - prediction",main="Residuals")
```

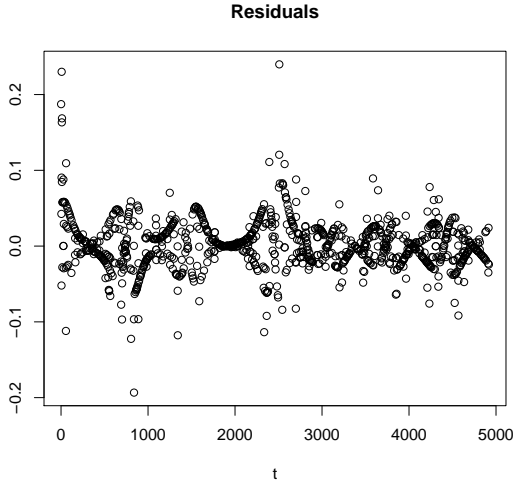
3NN prediction vs. reality



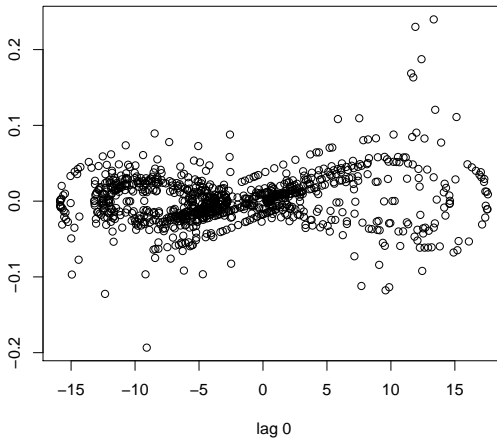
3NN prediction vs. reality



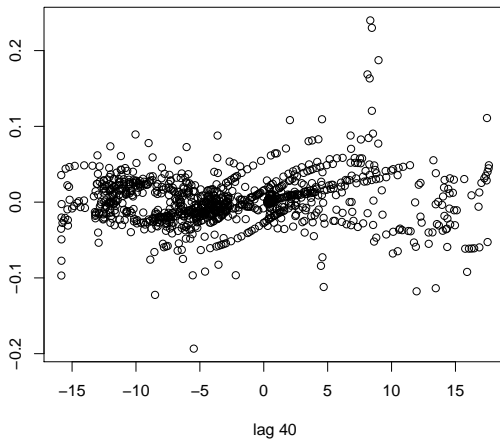
Not just a programming error!



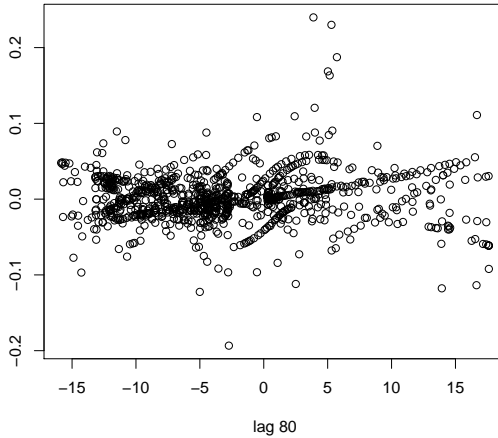
Residuals vs. first history component



Residuals vs. second history component



Residuals vs. third history component



Why not just use huge k ? bias/variance trade-off

- small k : tracks local behavior, big change in prediction depending on which point just happens to be closest to you
- big k : smooth predictions over state space, less sensitive to sampling, less informative locally
- $k = n$: same prediction all over state space!

Can we somehow increase k with n , to take advantage of filling-in along the attractor? — See handout on kernel prediction.

Cross-Validation

Standard and useful way of selecting control settings

Principle: we don't just want to fit old data, we want to predict new data

i.e., don't just optimize **in-sample** error, optimize **out-of-sample** error

Problem: we don't know the distribution of future data!

Would we be trying to *learn* a predictor if we did?

Observation: we do have a sample from that distribution — our sample

⇐ ergodicity: long trajectories are representative

Solution: fake getting a new sample by sub-dividing our existing one *at random*

Chose the settings which generalize best, which can **cross-validate**

Random division into training and testing sets needs to respect the structure of the data

e.g. divide points in the embedding space, not observations from the original time series

note: did this already with the Lorenz example — that was an out-of-sample prediction

Good idea to use a couple of random divisions and average out-of-sample errors

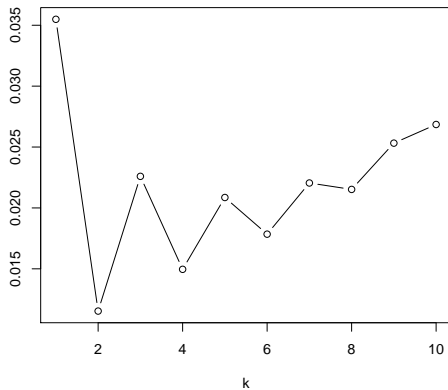
This **multi-fold** cross-validation also gives you an idea of the uncertainty due to sampling noise

5-fold cross-validation of k NN prediction of Lorenz, $k \in 1:10$

Warning: slow!

```
fold = sample(1:5,nrow(lorenz.rcon),replace=TRUE)
cvpred = matrix(NA,nrow=nrow(lorenz.rcon),ncol=10)
cvprederror = matrix(NA,nrow=nrow(lorenz.rcon),ncol=10)
for (k in 1:10) {
  for (i in 1:5) {
    train=which(fold!=i)
    test=which(fold==i)
    cvpred[test,k] = knn.predict(train=train,test=test,
                                lorenz.futures,lorenz.dist,
                                k=k,agg.meth="mean")
    cvprederror[test,k] = lorenz.futures[test]-cvpred[test,k]
  }
}
mean.cv.errors = apply(abs(cvprederror),2,mean)
plot(mean.cv.errors,xlab="k",ylab="mean absolute prediction
      error", main="5-fold CV of kNN prediction",type="b")
```

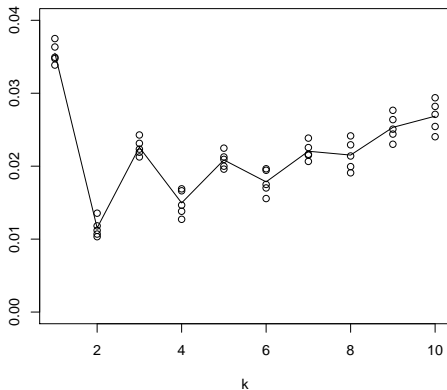
5-fold CV of kNN prediction



Add the individual “fold” values to the previous plot
graphics hygiene: have vertical scale run to zero

```
plot(mean.cv.errors,xlab="k",ylab="mean absolute prediction  
error", main="5-fold CV of kNN prediction",type="l",  
ylim=c(0,0.04))  
error.by.fold = matrix(NA,nrow=5,ncol=10)  
for (i in 1:5) {  
  for (k in 1:10) {  
    test=which(fold==i)  
    error.by.fold[i,k] = mean(abs(cvprederror[test,k]))  
    points(k,error.by.fold[i,k])  
  }  
}
```

5-fold CV of kNN prediction



Conclusion: 2 is best, but they're all pretty good
note bigger scatter at bigger k

- [1] Norman H. Packard, James P. Crutchfield, J. Doyne Farmer, and Robert S. Shaw. Geometry from a time series. *Physical Review Letters*, 45:712–716, 1980.
- [2] Floris Takens. Detecting strange attractors in fluid turbulence. In D. A. Rand and L. S. Young, editors, *Symposium on Dynamical Systems and Turbulence*, pages 366–381, Berlin, 1981. Springer-Verlag.