

# Stanでガウス過程

清水裕士  
関西学院大学

# 本発表の目的

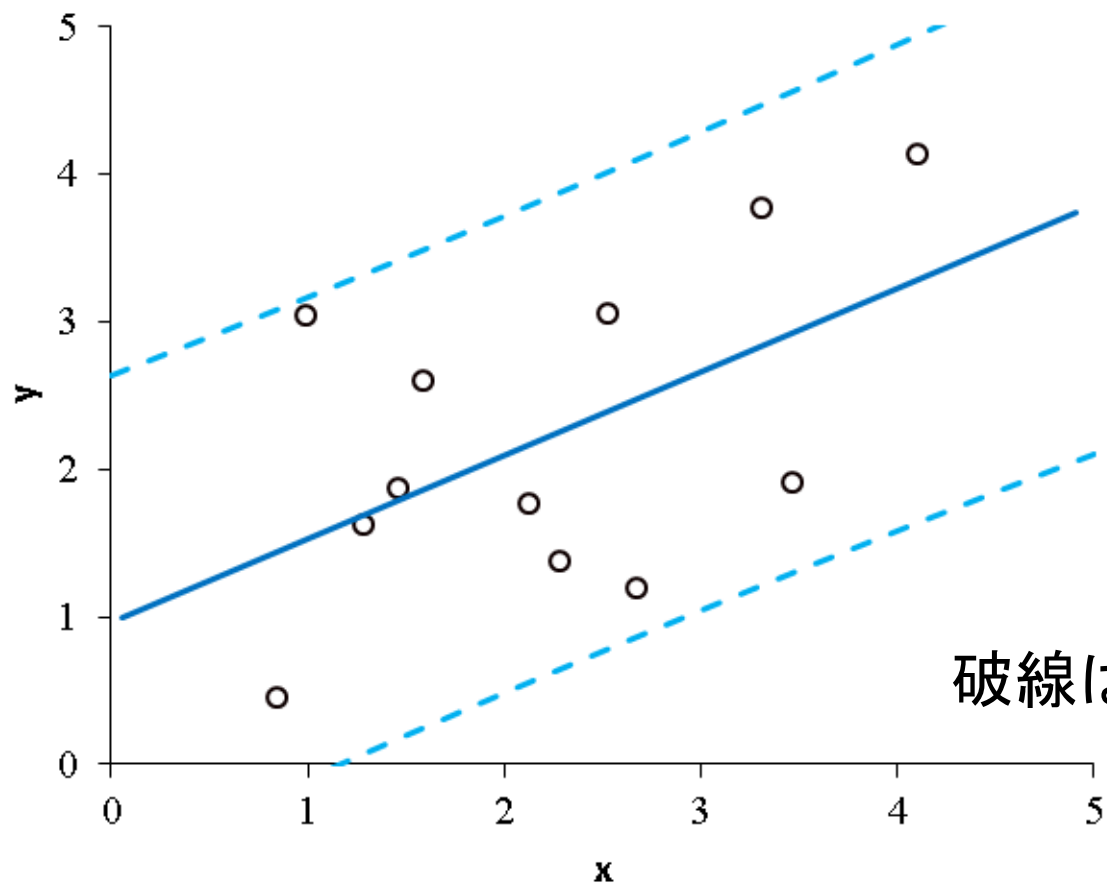
- ガウス過程の位置づけを理解する
  - 数理的な理解は今回は目的としない
    - 線形代数の知識が不可欠なので
  - ガウス過程をあてはめることの意味を理解
- Stanでガウス過程を実行する
  - 写経を超えて理解できればなおよい

# 注意

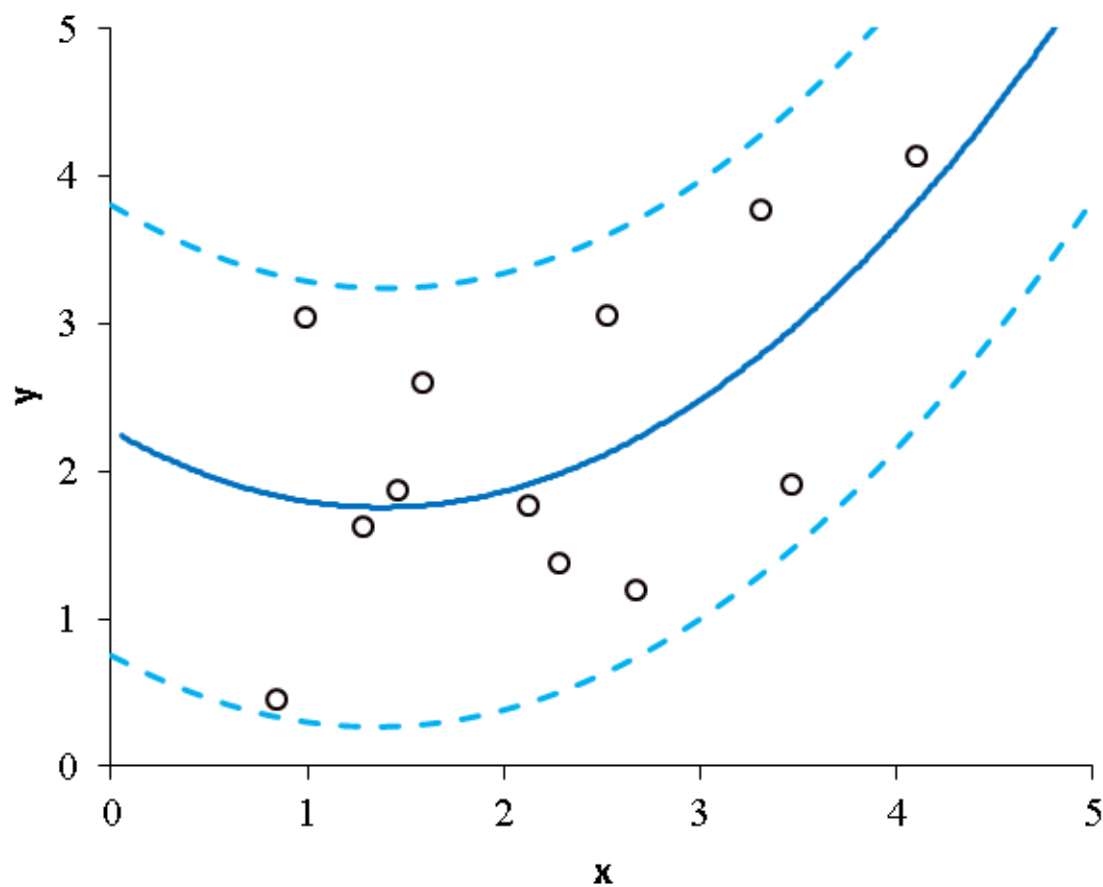
- 本発表の表記
  - 小文字( $y$ )はスカラー
  - 大文字( $Y$ )をベクトルとする
  - サンプルサイズは $n$

# ガウス過程のモチベーション

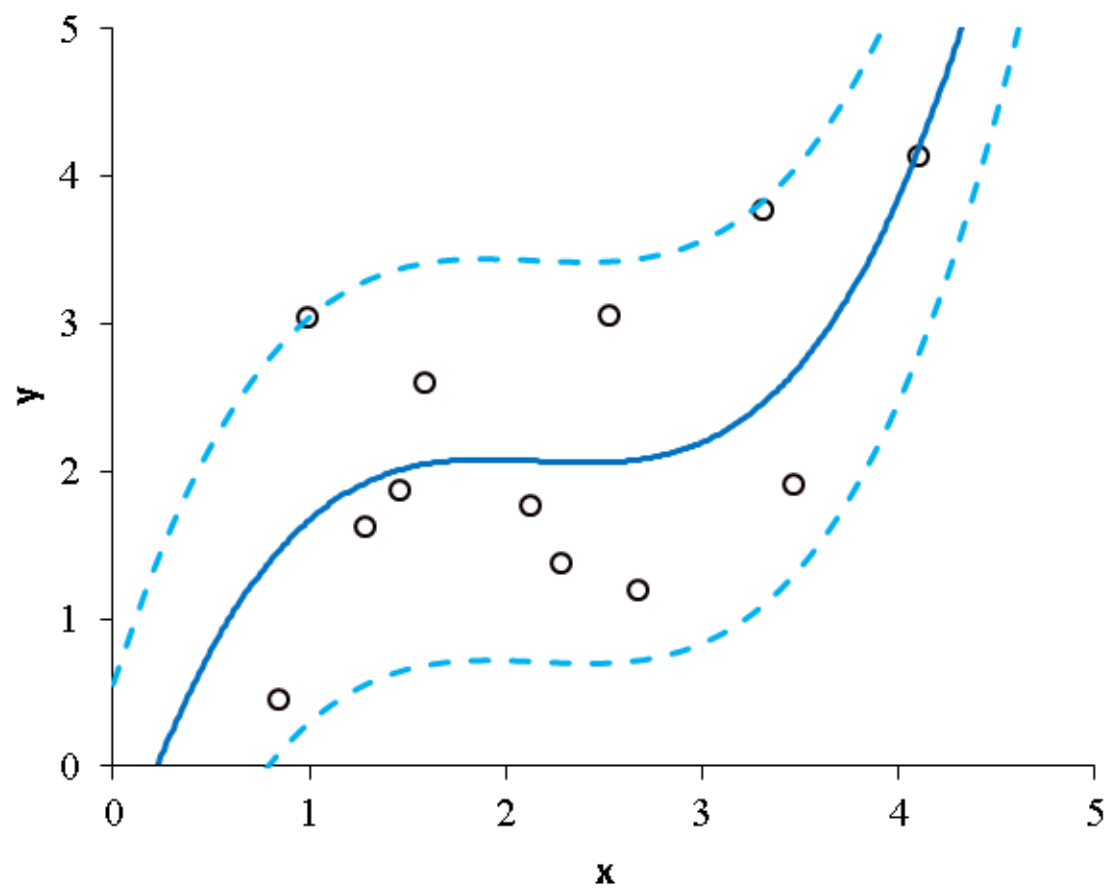
# 一次式



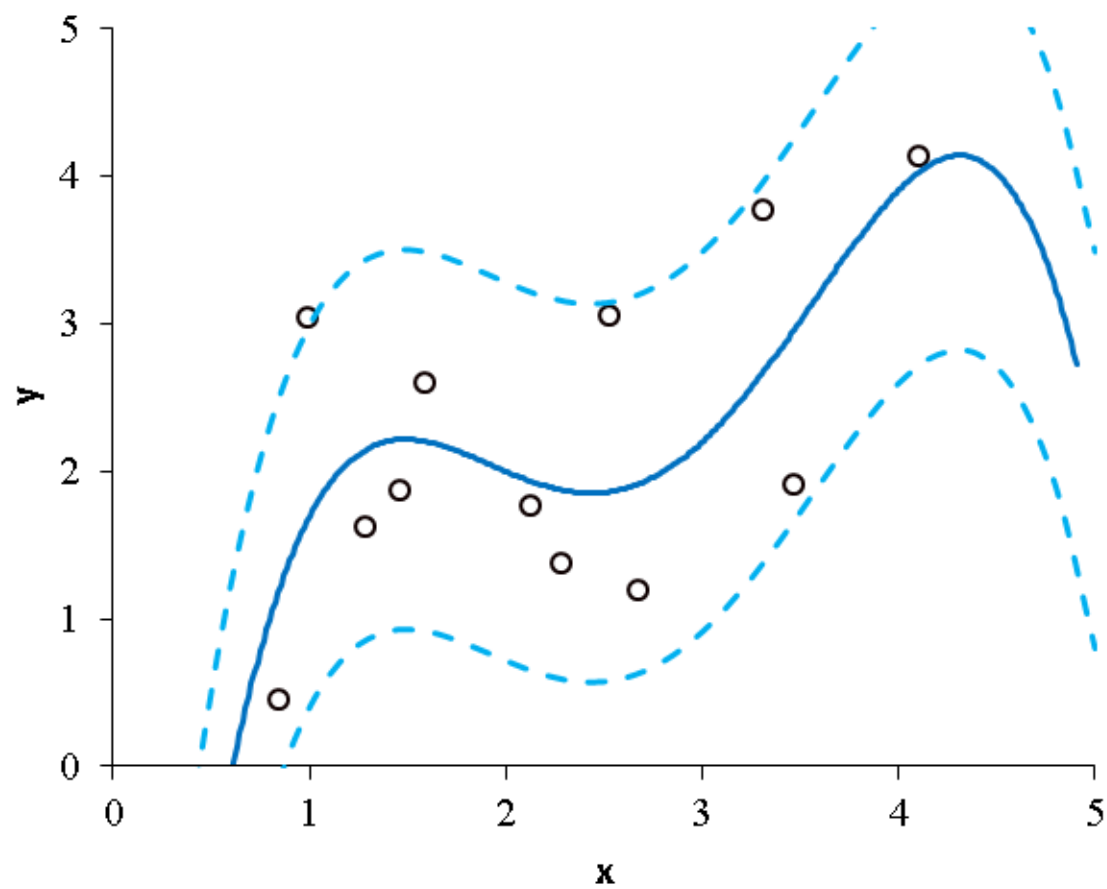
# 2次式で推定



# 3次式



# 4次式





# 何次式がいいだろうか？

- データに合わせすぎる
  - たとえば4次式
  - オーバーフィッティング
- モデルが単純すぎる
  - たとえば2次式
  - モデルと予測が乖離

# オーバーフィッティング

- 今回のデータに予測を合わせすぎてしまう
  - データはサンプリングの変動を必ず含む
  - たまたま高めだったり低めだったりする
  - そういう「たまたま」を「真のメカニズム」と判断してしまい、予測してしまうと、外れる
- パラメータの複雑さについてのジレンマ
  - WAICなどの汎化性能を評価する指標も使える
  - データから直接いい感じに推定したい
  - →ガウス過程回帰

# ガウス過程を一言でいうと

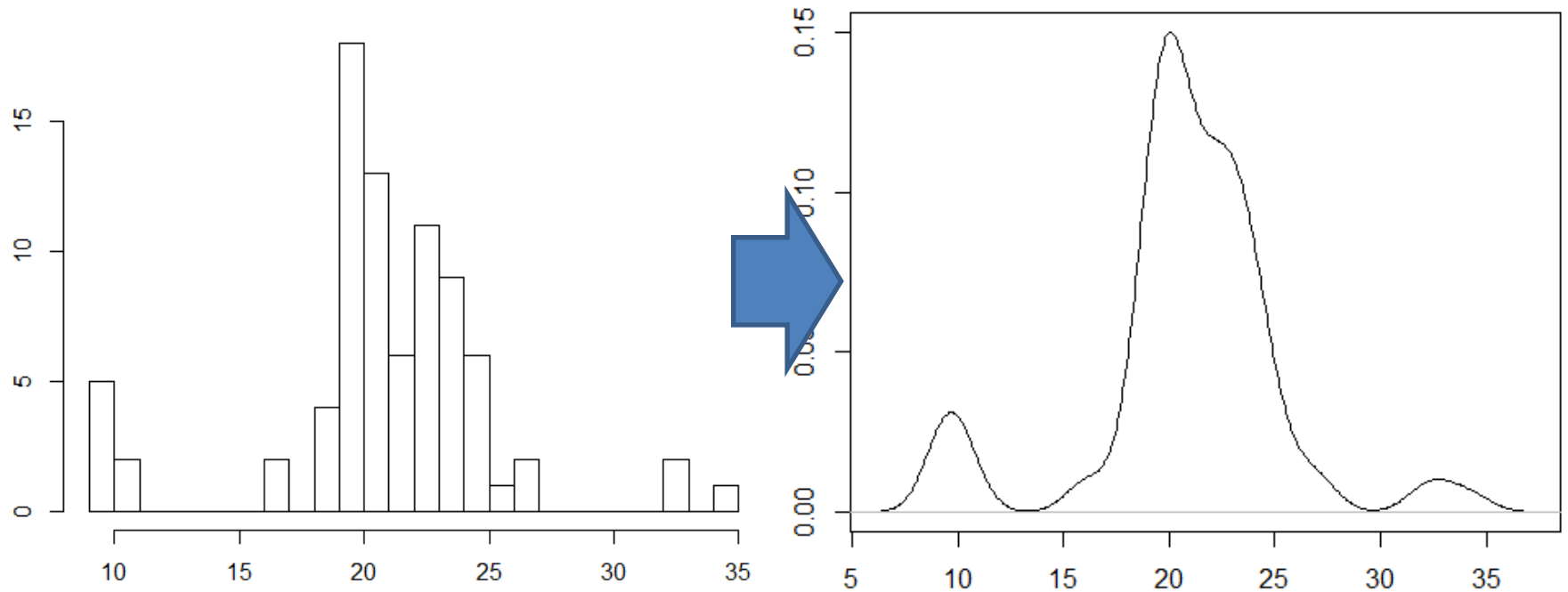
- データにうまくフィットする関数を推定する
  - 関数系そのものを推定する
  - $y_i = f(x_i)$ 
    - $i \in 1 \dots n$
- 普通の線形回帰
  - $y_i = w_0 + w_1 x_i + w_2 x_i^2 \dots w_m x_i^m$
  - どのような関数形にすればいいのかわかんない
  - それ自体を推定するのがガウス過程

# ざっくりいえば平滑化手法

- ヒストグラムを平滑化
  - カーネル密度推定
- 予測モデルを平滑化
  - カーネル回帰
- これらを確率モデルで表現すると・・・
  - ガウス過程という方法が浮かび上がってくる

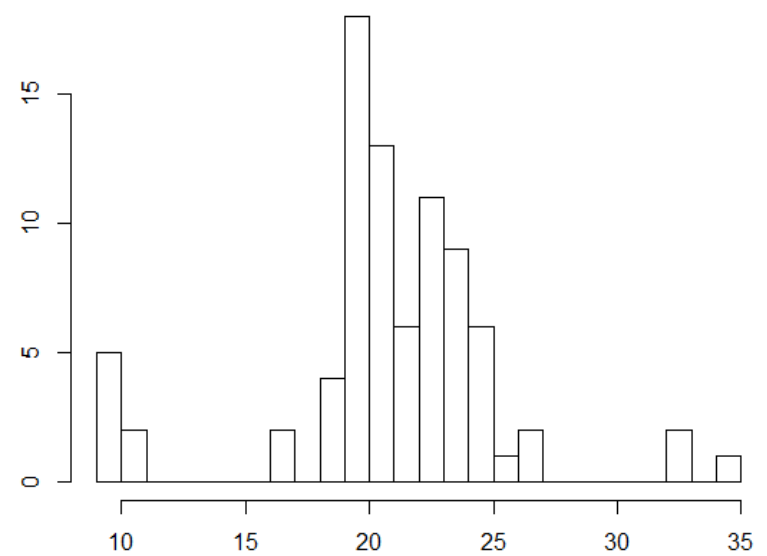
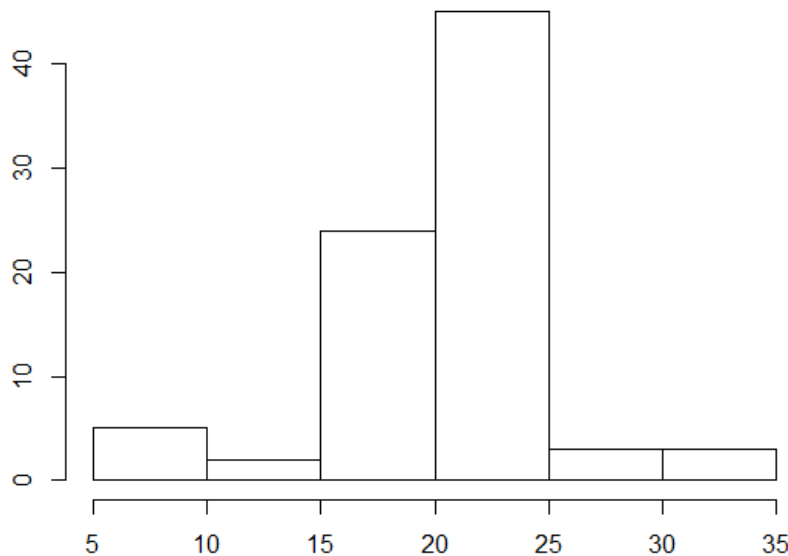
# 古典的平滑化の例

- カーネル密度推定



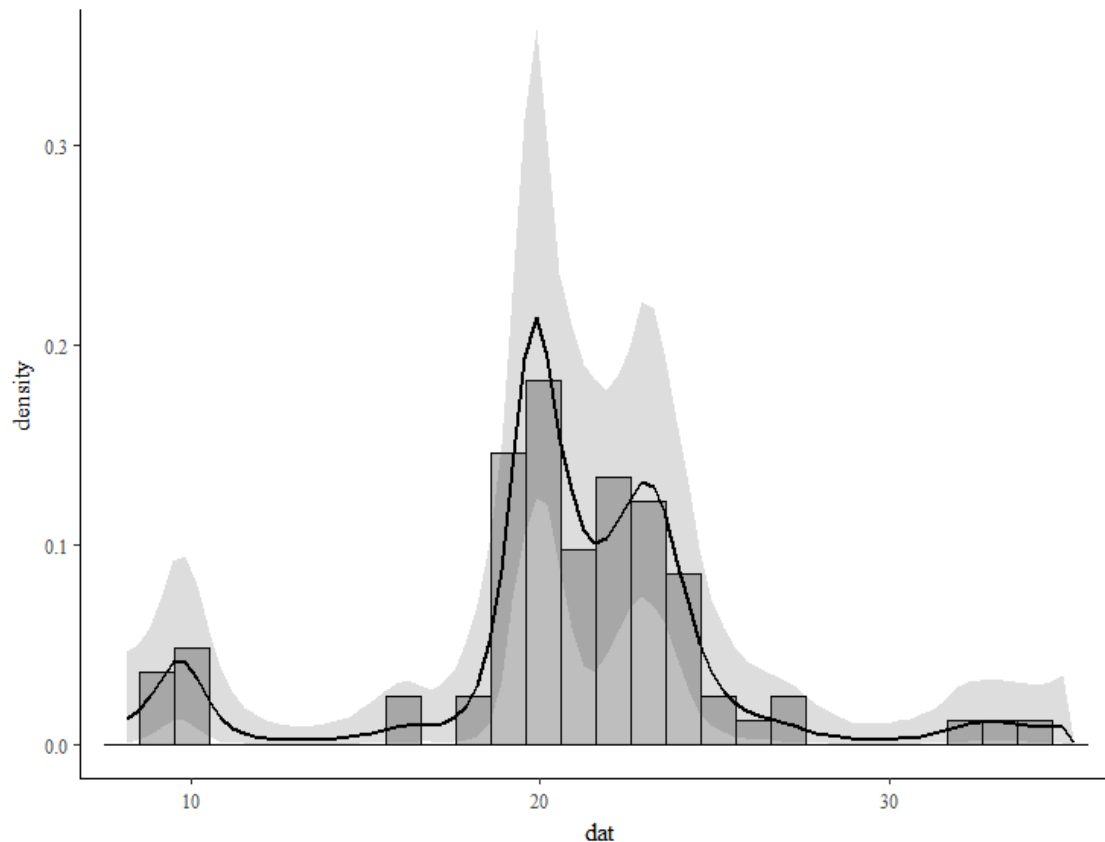
# 古典的ノンパラ手法の限界

- 確率モデルの長所を生かせない
  - モデル評価がしにくい
  - ハイパーパラメータの最適化が困難



# ガウス過程を使うと・・・

- 自動的に平滑化＋予測区間の推定



# 平滑化も確率モデルで！

- まずはカーネル法の導入
  - モデルの見方をちょいと変えれば、パラメータを消去して、直接データを予測することが可能
  - カーネルトリック！
- カーネルトリックの話はちょっと難しい・・・
  - 理解するにはちょっと数学的な素養が必要
  - カーネルトリックがわからなくてもガウス過程は使える



# 確率モデルということは・・・

- Yes, Bayesian!
  - ベイズ推定が可能
  - ノンパラメトリックベイズとは、要は確率過程を用いたベイズ推定のこと
    - ガウス過程
    - ディリクレ過程
- ガウス過程を使うと・・・
  - ノンパラモデルをベイズ推定可能になる
  - ハイパーパラメータの自動学習もできる

# ガウス過程は機械学習の奥義

- ほとんどの機械学習はガウス過程で表現可能
  - サポートベクターマシーン
  - ディープラーニング
    - 中間層のニューロン無限のDNN
  - カーネル回帰分析
    - ガウス過程回帰
  - カーネル主成分分析
    - ガウス過程潜在変数モデル (GPLVM)
- 統計のプロからのコメント
  - ガウス過程はなんでも上手くいくからおもんない

# ガウス過程回帰分析

# 線形回帰とカーネル回帰

- 普通の線形回帰
  - データ $Y$ に対して $X$ で線形モデルをあてはめる
    - $X$ は $x$ の $m$ 次元多項式
  - パラメータ $w$ を求める(切片、回帰係数)
    - $y_i = w_0 + w_1 x_i + w_2 x_i^2 + \dots + w_m x_i^m$ 
      - $Y = XW$

# 基底関数

- 基底関数を使った表現
  - 任意の $x$ の基底関数を $\phi(x)$ とする
    - 基底関数 $\dots x^a$ とか $\sin(x)$ とかみたいに $x$ から一意に決まる変換関数
    - ここでは上と同じように多項式(べき関数)だけを考えてOK
- 基底関数を使って書き直す
  - $y_i = w_0 + w_1\phi_1(x) + w_2\phi_2(x) + \dots + w_m\phi_m(x)$
  - $Y = \Phi W$
  - パラメータ $w$ と $x$ 、基底関数 $\phi$ から予測値 $\hat{y}$ を推定する

# 確率モデルを考える

- $w$ が正規分布に従うと仮定する
  - $w \sim \text{Normal}(0, \lambda I_m)$ 
    - $I_m$ は $m$ 次元単位行列
    - $\lambda$ はスカラー
    - これはリッジ回帰と同じ仮定
- $w$ の確率分布から、 $Y$ の確率分布を導く
  - $Y$ は線形結合による関数なので、 $Y$ も正規分布になることが想像できる

# Yの平均と分散を計算する

- Yの平均と共分散を計算する
  - $\mu_Y = E[Y] = E[\Phi W^T] = \Phi E[W^T] = \mathbf{0}$
  - $n$ 次元の0ベクトル
- Yの共分散行列
  - $\Sigma_Y = E[YY^T] - E[Y]E[Y]^T = E[\Phi W(\Phi W)^T] = \Phi E[WW^T]\Phi^T = \lambda\Phi E[I_m I_m^T]\Phi^T = \lambda\Phi\Phi^T$ 
    - 変数×変数ではなく、サンプル×サンプルの共分散行列を考える
- Yの確率モデル
  - $Y \sim \text{MultiNormal}(\mathbf{0}, \lambda\Phi\Phi^T)$
  - $n$ 変量正規分布

# カーネルトリック

- 共分散行列 $\lambda\Phi\Phi^T$ に注目
  - $n \times n$ の正方行列で、正定値なものであればなんでもいい
  - つまり、基底関数行列 $\Phi$ がなんであって、最終的にはなんらかの共分散行列を指定するだけで、 $y$ の確率モデルが構成できる
- カーネル関数にすべてを還元
  - $\kappa(x, x') = \lambda\Phi\Phi^T$
  - 関数系 $\Phi$ を具体的に指定せずとも、任意のカーネル関数を使って共分散行列 $\kappa(x, x')$ を作ればよい



# カーネルトリック

- 内積をとることのメリット
  - $\Phi\Phi'$ つまり $x$ についての多項式の内積をとると、ただか  
だか $n$ 次元の多変量正規分布を考えるだけでよい
- 無限次元の多項式(一般には基底)関数が、有限次元のカーネル関数で表現できる
  - カーネルトリック！
  - $m$ は無限次元でも、データの次元を超える関数形を考える必要がない、という点がポイント

# ガウス過程の確率モデル

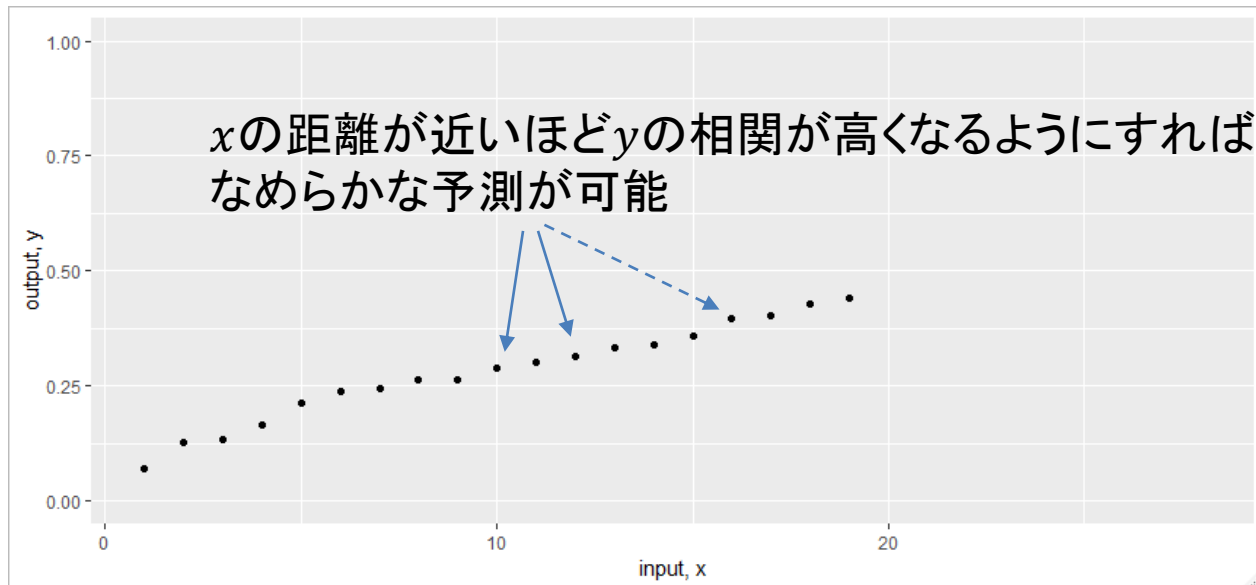
- $Y \sim \text{MultiNormal}(\mathbf{0}, \kappa(x, x'))$ 
  - サンプルサイズが $n$ の場合、 $n$ 変量正規分布を考える
  - $\kappa(x, x')$ は $n \times n$ の共分散行列を構成する関数
    - カーネル関数と呼ぶ
- この共分散行列をどのように決めればいい？
  - ガウス過程はカーネル関数をどうするかがすべてといって過言ではない

# カーネル関数

- ここからが重要！
  - ガウス過程のほぼすべてがここに詰まっている
- データポイント間の相関を表現する関数
  - 予測変数 $x$ とハイパーパラメータを使って、データポイント間の近さを表現するような行列を作る

# 例: とある時系列データ

- 20個のデータ
  - 20次元変量正規分布を考える
  - 任意の2点のデータポイントと類似性を表す



# ガウスカーネル

- よく使われるのがガウスカーネル

$$- \kappa(x, x') = \theta_1 \exp\left(-\frac{(x-x')^2}{2\theta_2}\right)$$

- ガウスカーネルとガウス過程は別物なので注意！

- $\exp\left(-\frac{(x-x')^2}{2}\right)$ の部分が正規分布の密度関数のカーネルと一致している

– RBFやexponential quadric kernelなどと呼ばれる

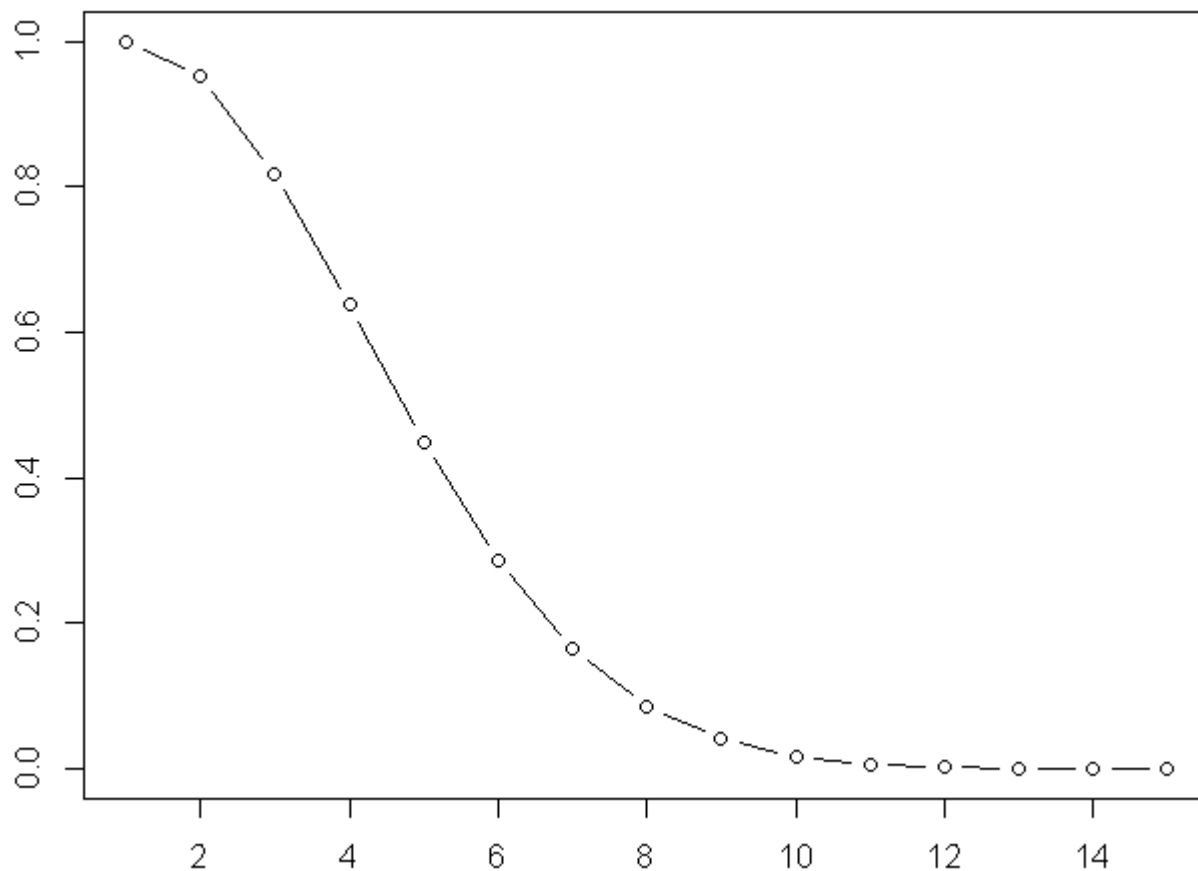
# ガウスカーネルの例

```
x <- seq(1,5)
n <- length(x)
theta <- 10
kappa <- array(dim=c(n,n))
for(i in 1:n){
  for(j in 1:n){
    kappa[i,j] <- exp(-(x[i]-x[j])^2/(2*theta))
  }
}
kappa
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1.0000000	0.9512294	0.8187308	0.6376282	0.4493290
[2,]	0.9512294	1.0000000	0.9512294	0.8187308	0.6376282
[3,]	0.8187308	0.9512294	1.0000000	0.9512294	0.8187308
[4,]	0.6376282	0.8187308	0.9512294	1.0000000	0.9512294
[5,]	0.4493290	0.6376282	0.8187308	0.9512294	1.0000000

近い点ほど  
相関が大きい

# 距離が離れると相関が減る



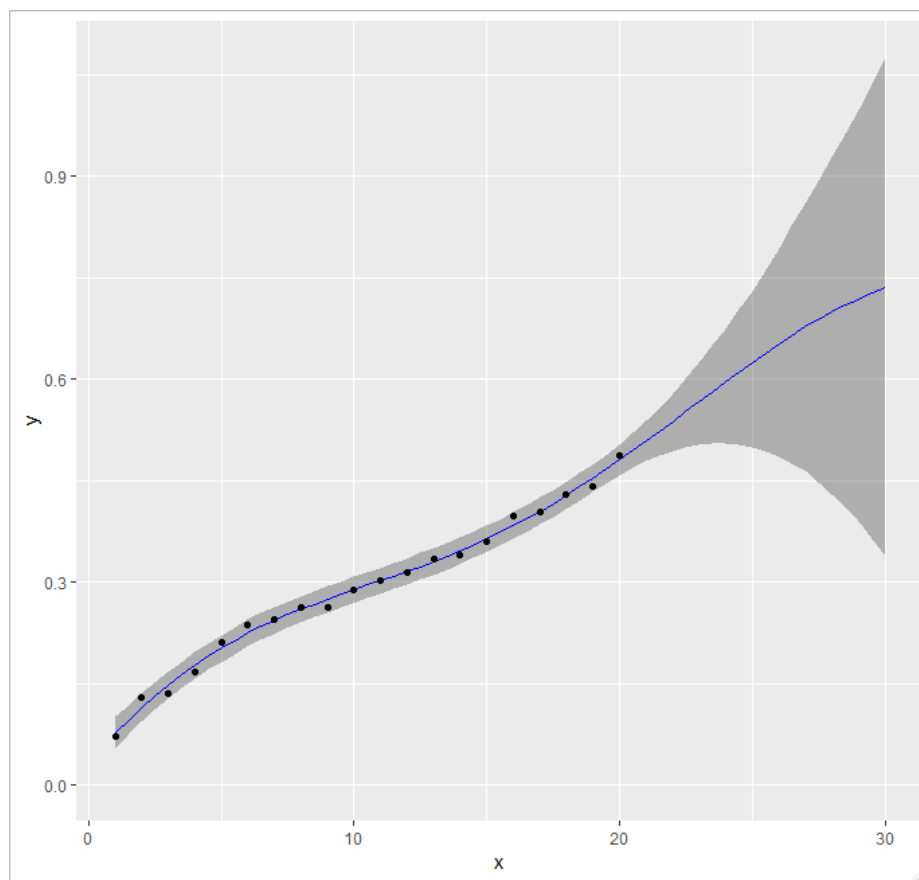
# ガウスカーネルのハイパーパラメータ

- $\kappa(x, x') = \theta_1 \exp\left(-\frac{(x-x')^2}{2\theta_2}\right)$ 
  - $\theta_1$  ガウスカーネル自体がどれくらいカーネル関数全体に寄与しているかを表している
  - $\theta_2$  ガウスカーネルのハイパーパラメータで、距離に応じてどれくらい相関が小さくなるかの程度
    - $\theta_2$  が大きいほど、なめらかな予測になる
  - ただし、これらはすべてMCMCで推定可能
    - なんらかの方法で事前に決めてもいい



# ガウスクーネルで推定

- なめらかに予測を行うことができる



# それ以外にもカーネル関数はある

- 線形カーネル

$$- \kappa(x, x') = \theta x x'$$

任意のべき乗をいければ、  
多項式回帰と同じにできる

```
x <- seq(1, 5)
n <- length(x)
kappa <- x%*%t(x)
kappa
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	2	3	4	5
[2,]	2	4	6	8	10
[3,]	3	6	9	12	15
[4,]	4	8	12	16	20
[5,]	5	10	15	20	25

任意の2点の相関が1  
xの分散の応じて分散は  
大きくなる

# それ以外にもカーネル関数はある

- フィッシャーカーネル
- シグモイドカーネル
- ニューラルネットカーネル

– これはよく知らん！

# カーネルを組み合わせる

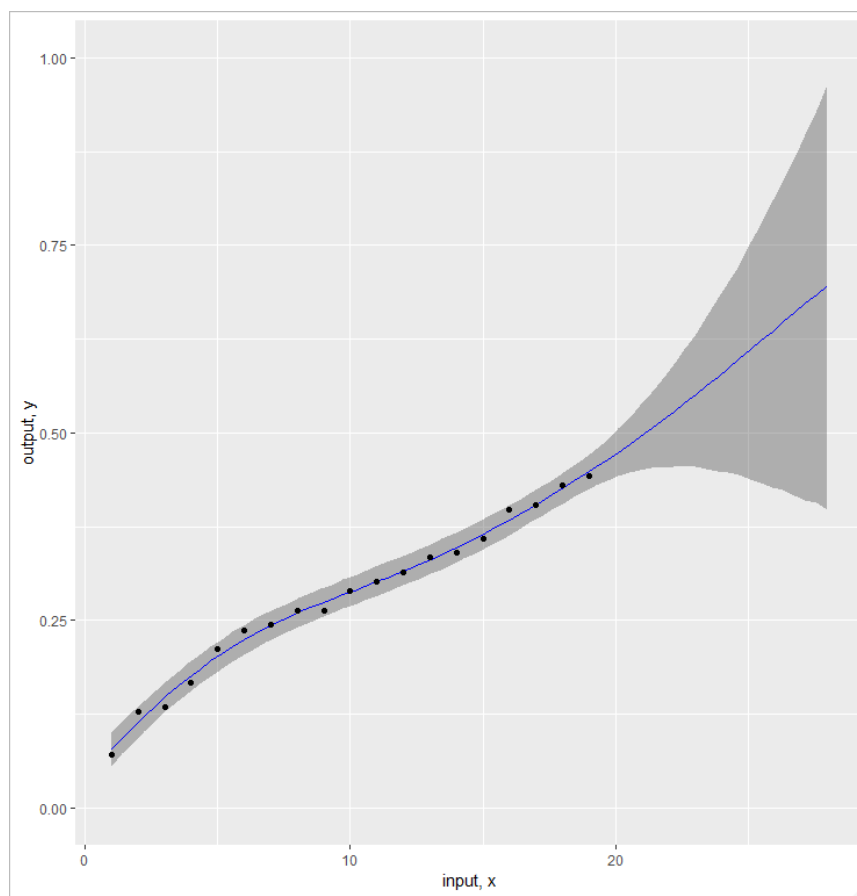
- よく使われる組み合わせ
  - ガウス＋線形＋ノイズ

- $\kappa(x, x') = \theta_1 \exp\left(-\frac{(x-x')^2}{2\theta_2}\right) + \theta_3 xx^T + \text{diag}(\theta_4)$

- これをStanに書けばOK！

# ガウス＋線形の推定

- ガウスカーネルだけよりも線形的予測が入る



# カーネル部分のStanコード

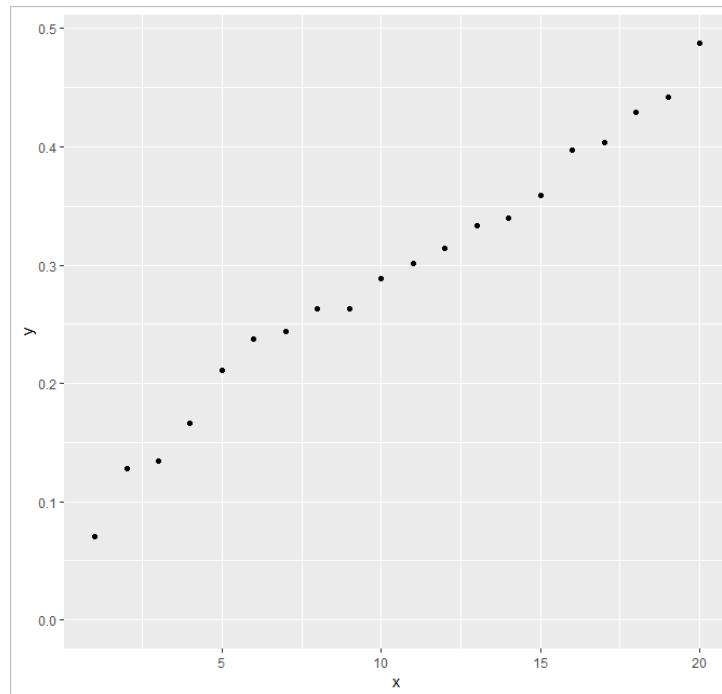
```
for(i in 1:N){  
  for(j in 1:N){  
    Kappa[i,j] = theta[1]*exp(-(X[i]-X[j])^2/theta[2]) + theta[3]*X[i]*X[j];  
    if(i==j) Kappa[i,j] += theta[4];  
  }  
}
```

$$\kappa(x, x') = \theta_1 \exp\left(-\frac{(x - x')^2}{2\theta_2}\right) + \theta_3 xx^T + \text{diag}(\theta_4)$$

今回はガウスクーネル＋ノイズで推定

# $y_1$ と $x_1$ のプロット

- $x_1$ は1~20の20個
  - $y_1$ はそれに対応する目的変数
  - $x$ が21~30の $\hat{y}$ を推定したい



# まずは $x_2$ を作る

```
N1 <- nrow(dat)
x1 <- dat$x
y1 <- dat$y
N2 <- 59
x2 <- seq(1,30,length.out = N2)
```

```
data.gp <- list(N1 = N1,
                N2 = N2,
                x1 = x1,
                x2 = x2,
                y1 = y1)
```

- $N_2$ は任意の数
  - 推定したい点の数
  - $x_2$ は1~30の間を59分割した点



# x2

```
> x2
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0 9.5
[19] 10.0 10.5 11.0 11.5 12.0 12.5 13.0 13.5 14.0 14.5 15.0 15.5 16.0 16.5 17.0 17.5 18.0 18.5
[37] 19.0 19.5 20.0 20.5 21.0 21.5 22.0 22.5 23.0 23.5 24.0 24.5 25.0 25.5 26.0 26.5 27.0 27.5
[55] 28.0 28.5 29.0 29.5 30.0
```

- この点について $\hat{y}$ を推定する
  - $y_1$ に対応する $x_1$ と被っていても問題ない
  - むしろ、 $y_1$ の予測値 $\hat{y}$ を知りたいこともあるので、被らせるほうがいいのかも
- イメージとしては、既知の $y$ を使ってカーネルのパラメータを推定し、 $\hat{y}$ をそこから推定する感じ

# コードの解説

```
1 data {  
2   int N1;  
3   int N2;  
4   vector[N1] x1;  
5   vector[N2] x2;  
6   vector[N1] y1;  
7 }
```

- N1
  - サンプルサイズ
  - $x1$ と $y1$ は実際にデータとしてとったもの
- N2
  - 推定したい $\hat{y}$ のサイズ
  - $x2$ は推定したい $\hat{y}$ に対応する $x$

# コードの解説

```
9 transformed data {  
10   int N = N1 + N2;  
11   vector[N] x;  
12   real vx;  
13   vector[N] Mu = rep_vector(0,N);  
14   for (n in 1:N1) x[n] = x1[n];  
15   for (n in 1:N2) x[N1 + n] = x2[n];  
16   vx = variance(x);  
17 }
```

- Nはデータ＋推定したい点の総数
  - すでにあるデータを使って  $\hat{y}$  を推定するためには、ベクトルを合成しておく必要がある


# コードの解説

```
19 parameters {  
20     vector<lower=0>[3] theta;  
21     vector[N2] y2;  
22 }
```

- theta
  - カーネル関数のハイパーパラメータ
    - 今回は3つ
- $y_2$ 
  - 推定したい  $\hat{y}$

# コードの解説

```
24 model {
25   matrix[N,N] Kappa;
26   vector[N] y;
27   //y y1 and y2
28   for (n in 1:N1) y[n] = y1[n];
29   for (n in 1:N2) y[N1 + n] = y2[n];
30
31   //kernel gauss + noise // + linear
32   for(i in 1:N){
33     for(j in 1:N){
34       Kappa[i,j] = theta[1]*exp(-(x[i]-x[j])^2/(vx*theta[2])); // + theta[4]*x[i]*x[j];
35       if(i==j) Kappa[i,j] += theta[3];
36     }
37   }
38   //Gaussian process
39   y ~ multi_normal(Mu,Kappa);
40   //prior
41   theta ~ student_t(4,0,5);
42 }
```



xの分散を入れることでtheta[2]が大きくなりすぎるのを防いでいる  
なくても推定は可能

- ガウス過程の確率モデル
  - シンプルに平均0の多変量正規分布

# 最終的なRコード

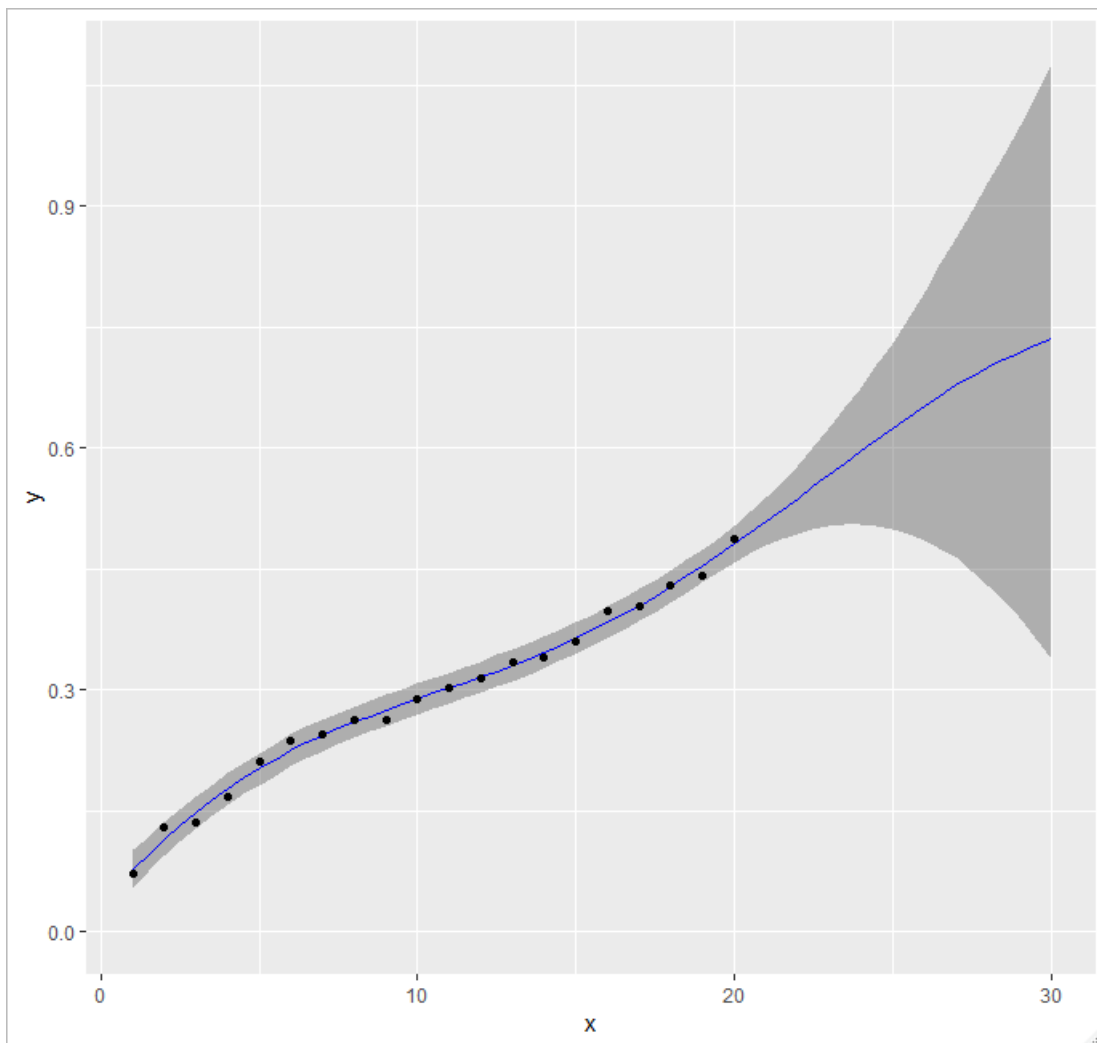
```
dat <- read.csv("gp_test.csv")

N1 <- nrow(dat)
x1 <- dat$x
y1 <- dat$y
N2 <- 59
x2 <- seq(1,30,length.out = N2)

data.gp <- list(N1 = N1,
               N2 = N2,
               x1 = x1,
               x2 = x2,
               y1 = y1)

model.gp <- stan_model("GP_reg.stan")
fit.gp <- sampling(model.gp,
                  data = data.gp,
                  iter = 5000,
                  chains = 4,
                  cores = 4)
```

# なめらかーに予測



カエルの卵みたい  
いとか言わない

# プロットのためのコード

```
y2.med <- apply(ext.fit$y2, 2, median)
y2.low <- apply(ext.fit$y2, 2, quantile, probs = 0.05)
y2.high <- apply(ext.fit$y2, 2, quantile, probs = 0.95)

d.est <- data.frame(x=x2, y=y2.med, ymax=y2.high, ymin=y2.low)
d.obs <- data.frame(x=x1, y=y1, ymax=rep(0,N1), ymin=rep(0,N1))
p <- ggplot(d.est, aes(x = x, y = y, ymax=ymax, ymin=ymin))
p <- p + xlab("x") + ylab("y")
p <- p + geom_ribbon(alpha = 1/3)
p <- p + geom_line(aes(y = y), color = "blue")
p <- p + geom_point(data = d.obs)
p
```



# ハイパーパラメータ

	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
theta[1]	1.49	0.03	1.93	0.09	0.35	0.82	1.85	6.63	5642	1
theta[2]	7.57	0.07	3.80	2.47	4.88	6.80	9.47	17.05	2769	1
theta[3]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1002	1

- いい感じに推定してくれる  
– フルベイズのいい点

# まとめ

- ガウス過程
  - 回帰の関数を明示的に決めなくても、いい感じの予測を自動でしてくれるテクニック
  - カーネル関数はいろいろあるので、うまくいくものを組み合わせてみよう
- 回帰以外も使える
  - 密度関数や潜在変数モデル、クラスタリングなど、いろんな分野にガウス過程は使える
  - (若干遅いが) Stanで簡単に実装可能