

3 D Plotting

Paul E. Johnson¹ ²

¹Department of Political Science

²Center for Research Methods and Data Analysis, University of Kansas

2013

Outline

- 1 Overview
- 2 persp
- 3 scatter3d
- 4 Scatterplot3d
- 5 rockchalk
 - mcGraph
 - plotPlane

Outline

- 1 Overview
- 2 persp
- 3 scatter3d
- 4 Scatterplot3d
- 5 rockchalk
 - mcGraph
 - plotPlane

Kinds of 3d Plot

- Static “draw on the screen, like R plot”
 - persp: in the R base graphics
 - cloud in lattice package
 - scatterplot3d
- scatter3d: by John Fox for the car package, uses OpenGL (computer 3d programming library)
 - interactive and easy to get started
 - can be accessed from Fox’s Rcmdr package interface
 - final output not as likely to be “publishable”

Here's what we usually want the 3d Plot For

- Show the “cloud” of points scattered in space
- Show the “predicted plane” of a fitted regression model
- persp can do these things, although it is somewhat tough to grasp at first
- Why keep trying: persp is in the base of R, so if something is wrong with it, it is likely somebody will know how to fix it.
- If you show up in r-help asking about 3D plotting, many folks there will suggest you learn persp, since most other routines draw upon its concepts.

Outline

- 1 Overview
- 2 **persp**
- 3 scatter3d
- 4 Scatterplot3d
- 5 rockchalk
 - mcGraph
 - plotPlane

persp is the Place to Start

- The key thing to understand: if your variables are x_1 , x_2 (the inputs), and z (the output), persp does not “want” your variables like this

```
persp(x1, x2, z)
```

- persp requires “plotting sequences” for x_1 and for x_2 . These are not observed values, but rather sequences from the minimum score to the maximum.
- For “real data,” x_1 , for example, get the range, then make a sequence:

```
x1r <- range(x1)
x1seq <- seq(x1r[1], x1r[2], length.out = 30)
## or use the rockchalk short-cut
x1seq <- plotSeq(x1, length.out = 30)
```

For z , persp wants a matrix

- z has a value for each combination of $x1seq$ and $x2seq$
- Various ways to create, but the “outer” function is often convenient.

```
z ← outer(x1seq,
          x2seq, FUN)
```

- FUN is a function that returns a value for each combination of values in the 2 sequences

		x2seq		
		x_{21}	x_{22}	x_{23}
x1seq	x_{11}	z_{11}	z_{12}	z_{13}
	x_{12}	z_{21}	z_{22}	z_{23}
	x_{13}	z_{31}	z_{32}	z_{33}

$z_{11} = f(x_{11}, x_{21})$, and so forth

Why Does R Call it "outer?"

Remember, an "inner product" in linear algebra

$$\begin{bmatrix} a & b & c & d \end{bmatrix} \begin{bmatrix} e \\ f \\ g \\ h \end{bmatrix} = ae + bf + cg + dh = ?? \quad (1)$$

An "outer product" is

$$\begin{bmatrix} e \\ f \\ g \\ h \end{bmatrix} \begin{bmatrix} a & b & c & d \end{bmatrix} = \begin{bmatrix} ea & eb & ec & ed \\ fa & fb & fc & fd \\ ga & gb & gc & gd \\ ha & hb & hc & hd \end{bmatrix} \quad (2)$$

Create Some Data for a Regression

```
x1 ← rnorm(100); x2 ← -4 + rpois(100,lambda=4);
y = 0.1 * x1 + 0.2 *x2 + rnorm(100);
dat ← data.frame(x1, x2, y); rm (x1, x2, y)
m1 ← lm(y ~ x1 + x2, data=dat)
summary(m1)
```

Call:

```
lm(formula = y ~ x1 + x2, data = dat)
```

Residuals:

Min	1Q	Median	3Q	Max
-2.29298	-0.55317	0.02582	0.56164	2.73118

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	0.05245	0.09697	0.541	0.5898

Create Some Data for a Regression ...

```
x1          0.20560      0.08538      2.408      0.0179  *
x2          0.20855      0.04473      4.662      1e-05  **
      *
```

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.'
 '0.1' '1'

Residual standard error: 0.942 on 97 degrees of freedom

Multiple R^2 : 0.2082, Adjusted R^2 : 0.1919

F-statistic: 12.76 on 2 and 97 DF, p-value: 1.207e-05

Create the predictor sequences

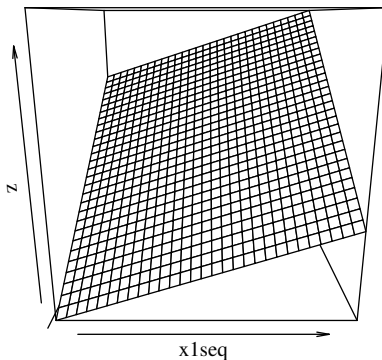
```
x1r ← range(dat$x1)
x1seq ← seq(x1r[1], x1r[2], length = 30)
x2r ← range(dat$x2)
x2seq ← seq(x2r[1], x2r[2], length = 30)
```

Create the z matrix

```
z ← outer(x1seq, x2seq, function(a, b) predict(m1  
  , newdata = data.frame(x1 = a, x2 = b)))
```

Persp with No Special Settings

```
persp(x = x1seq, y = x2seq, z = z)
```



Many Opportunities for Beautification

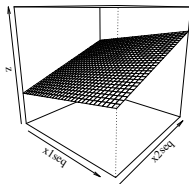
- `xlim,ylim,zlim` play same role as in ordinary R plots
- `xlab, ylab, zlab` same
- `theta` and `phi` control the viewing angle.
 - `theta` moves the viewing angle left and right
 - `phi` moves it up and down.
- Example, this “raises” one’s viewing angle (a negative value would lower it)

```
persp(x=x1seq, y=x2seq, z=z, phi=40)
```

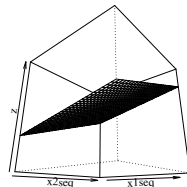
- Example, this “rotates” one’s viewing angle to the left

```
persp(x=x1seq, y=x2seq, z=z, theta=-20)
```

```
persp(x = x1seq, y = x2seq, z  
      = z, zlim = c(-3,3), theta  
      = 40)
```



```
persp(x = x1seq, y = x2seq, z  
      = z, zlim = c(-3,3), theta  
      = 40, phi = -20)
```



Everything Else We Draw Has to be "Perspective Adjusted"

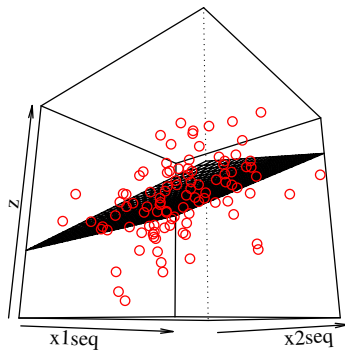
- This "looks" 3-dimensional, but it is really a flat two dimensional screen
- Thus, a point to be inserted at $(x_1 = 0.3, x_2 = -2, z = 2)$
 - has to be translated into a position in the 2-dimensional screen
- To do that, we use
 - A "Viewing Transformation Matrix" that persp creates
 - `trans3d`, a function that converts a 3 dimensional coordinate into a 2 dimensional coordinate

Add Points on a perspective plot

```
res ← persp(x = x1seq, y = x2seq, z = z, zlim = c
  (-3,3), theta = 40, phi = -15)
mypoints ← trans3d(dat$x1, dat$x2, dat$y, pmat =
  res)
points(mypoints, pch = 1, col = "red")
```

- persp generates “res” as a plot by-product
- res is the perspective transformation matrix (used by trans3d)
- mypoints is a 2 dimensional value in the “surface of the computer screen” displaying the 3d plot.

Points overlaid on persp plot via trans3d



Remember the Fitted Regression model?

```
x1 <- rnorm(100); x2 <- -4 + rpois(100,lambda=4);
y = 0.1 * x1 + 0.2 * x2 + rnorm(100);
dat <- data.frame(x1, x2, y); rm (x1, x2, y)
m1 <- lm(y ~ x1 + x2, data=dat)
summary(m1)
```

Call:

`lm(formula = y ~ x1 + x2, data = dat)`

Residuals:

Min	1Q	Median	3Q	Max
-2.29298	-0.55317	0.02582	0.56164	2.73118

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	0.05245	0.09697	0.541	0.5898
x1	0.20560	0.08538	2.408	0.0179 *
x2	0.20855	0.04473	4.662	1e-05 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.942 on 97 degrees of freedom

Multiple R^2 : 0.2082, Adjusted R^2 : 0.1919

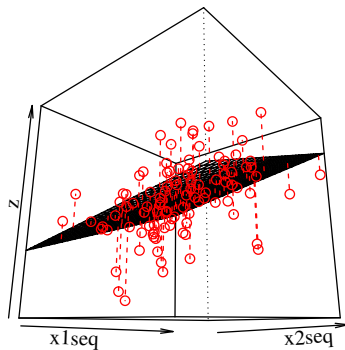
F-statistic: 12.76 on 2 and 97 DF, p-value: 1.207e-05

Now draw dotted lines from Predicted to Observed Values

- This took 8-10 tries
- Calculate predicted (vpred) and observed values (vobs)
- Use segments to draw

```
res ← persp(x = x1seq, y = x2seq, z = z, zlim = c
  (-3,3), theta = 40, phi = -15)
mypoints ← trans3d(dat$x1, dat$x2, dat$y, pmat =
  res)
points(mypoints, pch = 1, col = "red")
vpred ← trans3d(dat$x1, dat$x2, fitted(m1),
  pmat = res)
vobs ← trans3d(dat$x1, dat$x2, dat$y, pmat =
  res)
segments(vpred$x, vpred$y, vobs$x, vobs$y, col = "
  red", lty = 2)
```

This Makes Me Happy



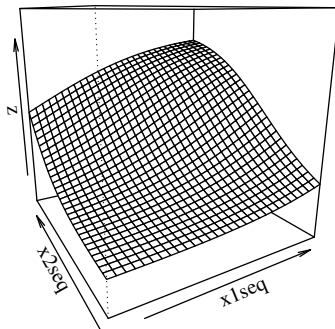
Plotting Response Surfaces

- People who fit nonlinear models often want to see the graceful curvature of their result
- Often nice to have some color for drama

Plotting Response Surfaces: Surprisingly Easy

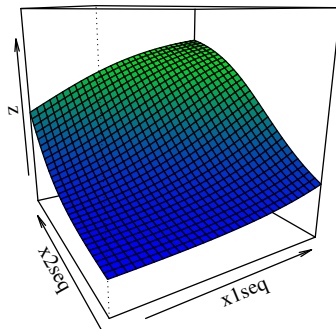
```
x1 ← rnorm(100); x2 ← rpois(100,lambda=4)
logist ← function(x1,x2){
  y ← 1/(1 + exp((-1)*(-3 + 0.6*x1 + .5*x2))) }
par(bg = "white")
x1r ← range(x1) ; x1seq ← seq(x1r[1], x1r[2],
  length = 30)
x2r ← range(x2) ; x2seq ← seq(x2r[1], x2r[2],
  length = 30)
z ← outer(x1seq, x2seq, logist)
persp(x = x1seq, y = x2seq, z = z, theta = -30,
  zlim = c(-0.2,1.2))
```


A Curved Surface, but No Color (yet)



```
nrz ← nrow(z)
ncz ← ncol(z)
# Create a function interpolating colors in the
  range of specified colors
jet.colors ← colorRampPalette( c("blue", "green")
  )
# Generate the desired number of colors from this
  palette
nbcol ← 100
color ← jet.colors(nbcol)
# Compute the z-value at the facet centres
zfacet ← z[-1, -1] + z[-1, -ncz] + z[-nrz, -1] +
  z[-nrz, -ncz]
# Recode facet z-values into color indices
facetcol ← cut(zfacet, nbcol)
persp(x = x1seq, y = x2seq, z = z, col = color[
  facetcol], theta = -30, zlim = c(-0.2, 1.2))
```

A Curved Colored Surface



Outline

- 1 Overview
- 2 persp
- 3 scatter3d**
- 4 Scatterplot3d
- 5 rockchalk
 - mcGraph
 - plotPlane

Now try scatter3d and the OpenGL Library Framework

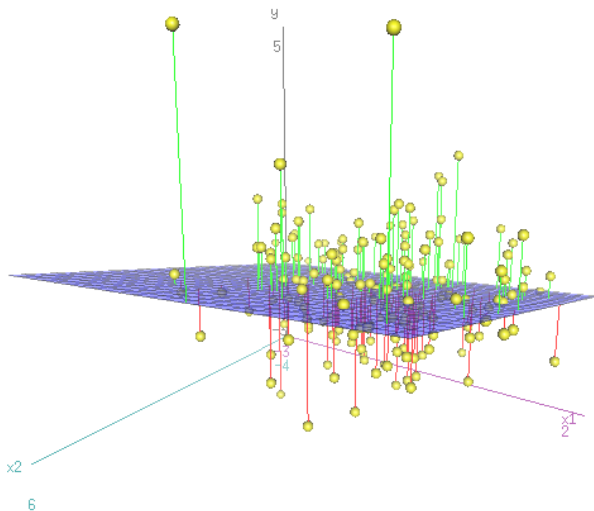
- OpenGL is an “open standards” 3-D software library (most platforms, newer video cards)
- “rgl” is an R package that uses OpenGL routines
- scatter3d is John Fox’s R function (in “car”) that uses rgl functions in a very convenient way

A Scatterplot with a Regression Surface

```
scatter3d(y ~ x1 + x2, data=dat)  
rgl.snapshot(filename="scat1.png", fmt="png")
```

- Note: a “formula interface”
- scatter3d handles the creation of “plotting sequences” and the perspective/trans3d work is hidden
- left-button mouse click rotates
- middle-button mouse click “zooms” the image

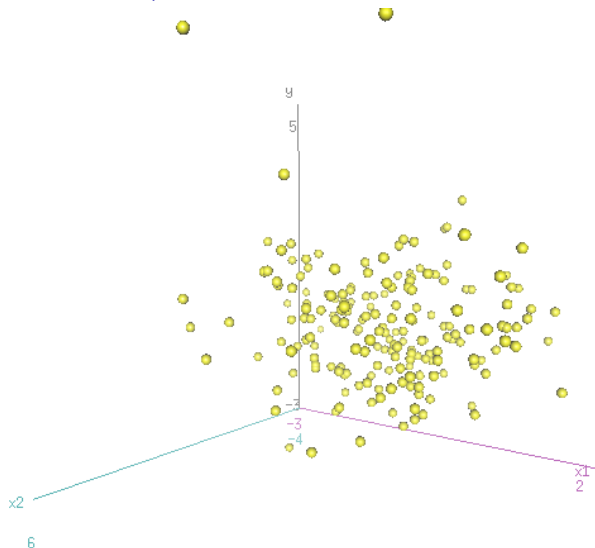
A Scatterplot with a Regression Surface



Just the Scatter, No Plane

```
scatter3d(y ~ x1 + x2, data=dat, surface=FALSE)  
rgl.snapshot(filename="scat2.png", fmt="png")
```

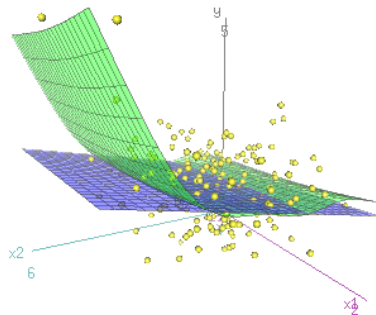

Just the Scatter, No Plane



Ask For an Ordinary and a Smoothed Regression Surface

```
scatter3d(y ~ x1 + x2, data=dat, fit=c("linear", "
    additive"))
rgl.snapshot(filename = "scat3.png", fmt = "png")
```

Ask For an Ordinary and a Smoothed Regression Surface



Evaluation

- scatter3d makes it *very easy* to get started
- The GUI in Rcmdr makes it even easier!
- Great for quick & dirty data exploration
- Disadvantages
 - Output quality not suitable for presentation (labels not “sharp”)
 - png only workable output format at current time (others generate HUGE files)

Outline

- 1 Overview
- 2 persp
- 3 scatter3d
- 4 Scatterplot3d**
- 5 rockchalk
 - mcGraph
 - plotPlane

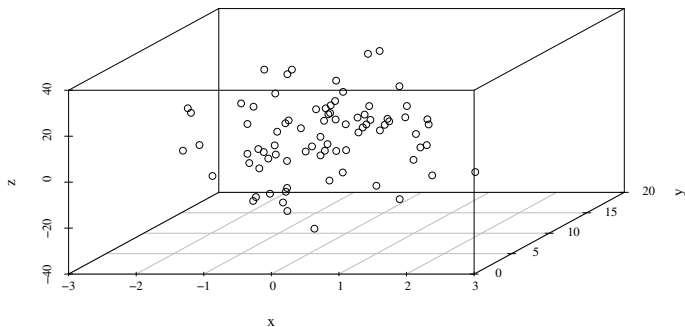
Confessions

- I have tested, but not mastered, these 3d plotting frameworks
 - cloud (in lattice)
 - scatterplot3d (package same name)
- These try to hide the “trans3d” problem from the user as much as possible
- IF you enjoy the
 - plot interface in R, *then* consider scatterplot3d
 - lattice and the xyplo interface, *then* you should consider trying to master “cloud”

scatterplot3d Works Quite a Bit Like Plot

```
library(scatterplot3d)
x ← rnorm(80); y ← rpois(80,l=7); z ← 3 + 1.1*x
      + 0.4*y + 15*rnorm(80)
s3d ← scatterplot3d(x, y, z)
```

scatterplot3d: Quite a Bit like plot



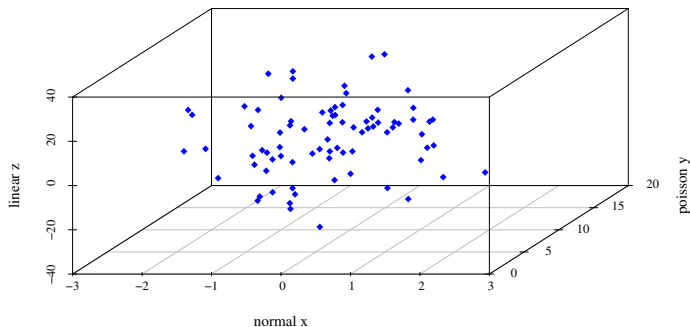
scatterplot3d: Quite a Bit like plot

- Note: Not necessary to construct a z matrix (scatterplot3d handles that)
- Many options same name as plot: xlab, ylab, type, etc.
- angle: viewpoint specifier quite unlike other 3d packages

Use More Arguments: labels, plot character

```
library(scatterplot3d)  
s3d ← scatterplot3d(x, y, z, type = "p", color =  
  "blue", angle = 45, pch = 18, main = "", xlab  
  = "normal x", ylab = "poisson y", zlab = "  
  linear z")
```

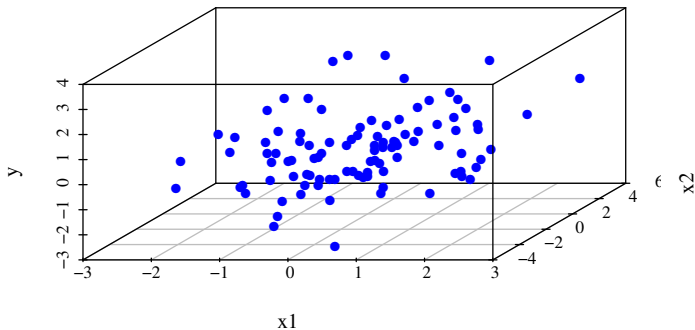
scatterplot3d: Quite a Bit like plot



scatterplot3d: Also Accepts a "Data Frame" for x

```
library(scatterplot3d)
s3d ← scatterplot3d(dat, type = "p", color = "
  blue", angle = 55, pch = 16, main = "", xlab =
  "x1", ylab = "x2", zlab = "y")
```

scatterplot3d

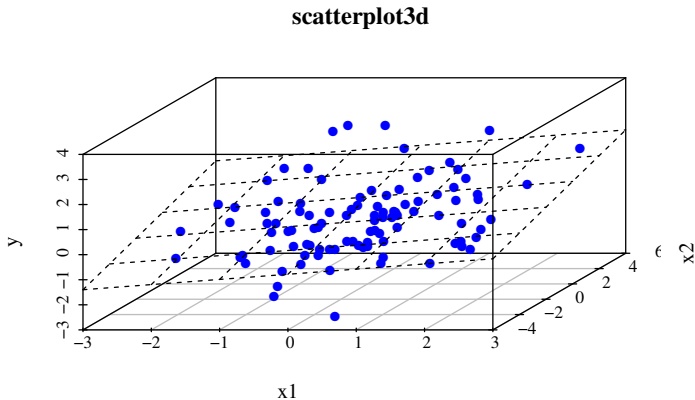


Add a plane from a fitted model!

```
library(scatterplot3d)
s3d ← scatterplot3d(dat, type = "p", color = "
  blue", angle = 55, pch = 16, main = "
  scatterplot3d")
s3d$plane3d(m1)
```

- Note s3d is the 3d plot object, it is told to draw plane corresponding to model m1
- That “internalizes” the “translate to 3d coordinates” works that persp required us to do explicitly
- supplies function “xyz.convert” when explicit translation from 3d to 2d is required (in placing text or lines)

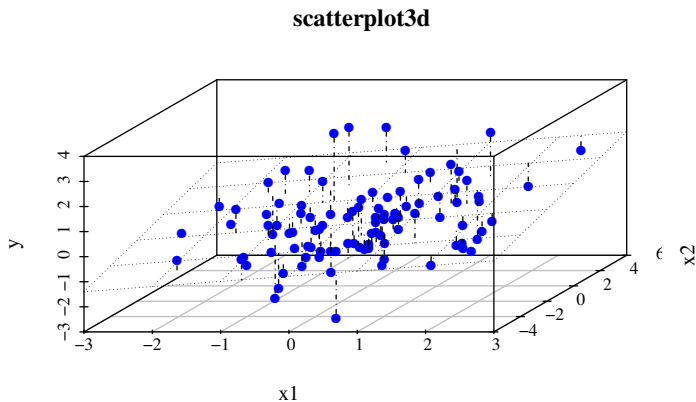
scatterplot3d



Add Residual Lines: Quite a Bit Like Using persp

```
s3d ← scatterplot3d(dat, type = "p", color = "
  blue", angle = 55, pch = 16, main = "
  scatterplot3d")
s3d$plane3d(m1, lty = "dotted", lwd = 0.7)
obser2d ← s3d$xyz.convert(dat$x1, dat$x2, dat$y)
pred2d ← s3d$xyz.convert(dat$x1, dat$x2, fitted(
  m1))
segments(obser2d$x, obser2d$y, pred2d$x, pred2d$y,
  lty = 4)
```


scatterplot3d



scatterplot3d: Syntax Closer "Object Oriented" Ideal

- scatterplot3d creates an output object

```
attributes(s3d)
```

```
$names  
[1] "xyz.convert" "points3d"    "plane3d"      "  
    box3d"
```

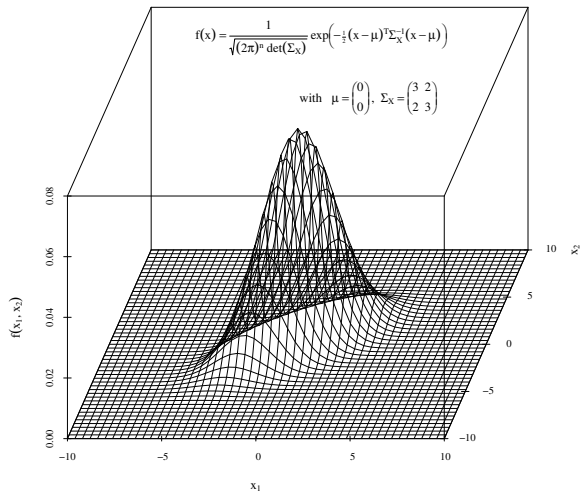
- Which can then be told to add points, a plane, etc:

```
s3d$plane3d(mod1)  
s3d$points3d(x,y,z, pch=18, col="pink")
```

- Also works well with plotmath functions

Consider the Bivariate Normal Example from s3d Vignette

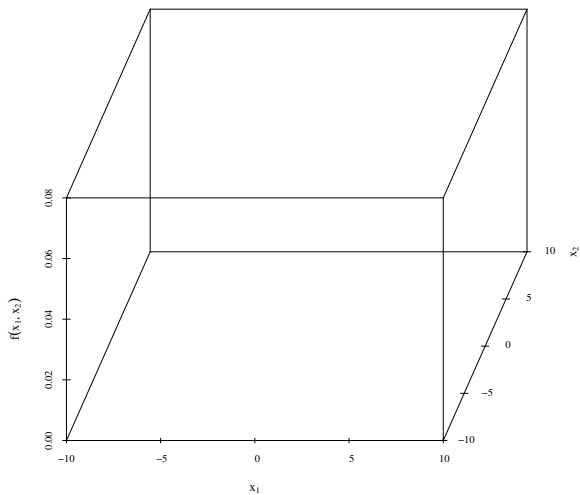
Bivariate normal distribution



The first step is the empty box

```
x1 <- x2 <- seq(-10, 10, length = 51)
dens <- matrix(dmvnorm(expand.grid(x1, x2), sigma
  = rbind(c(3, 2), c(2, 3))), ncol = length(x1))
s3d <- scatterplot3d(x1, x2, seq(min(dens), max(
  dens), length = length(x1)), type = "n", grid
  = FALSE, angle = 70, zlab = expression(f(x[1],
  x[2])), xlab = expression(x[1]), ylab =
  expression(x[2]), main = "Bivariate normal
  distribution")
```

Note: type="n", just like 2D plot function

Bivariate normal distribution

Draw the lines from One End to the Other

```
for(i in length(x1):1){  
  s3d$points3d(rep(x1[i], length(x2)), x2, dens[  
    i,], type = "l")  
}
```

- in English: for each value of x_1 , draw a line from “front to back” that traces out the density at (x_1, x_2) .
- The for loop goes to each value of x_1

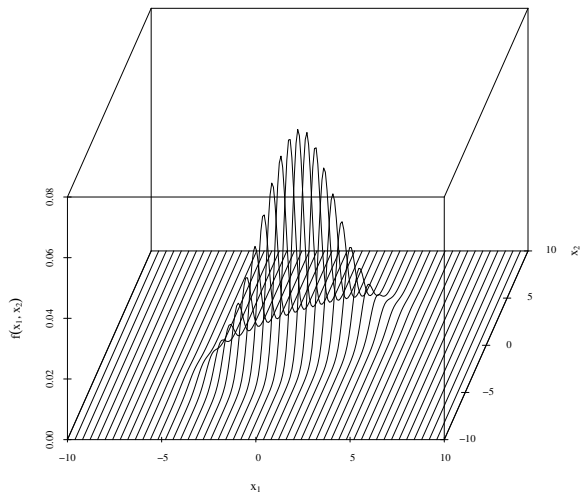
```
for (i in length(x1):1){ ...
```

- inserts points from lowest x_2 to highest x_2 and connects them by a line

```
s3d$points3d( ... type=l)
```

Lines in One Direction

Bivariate normal distribution



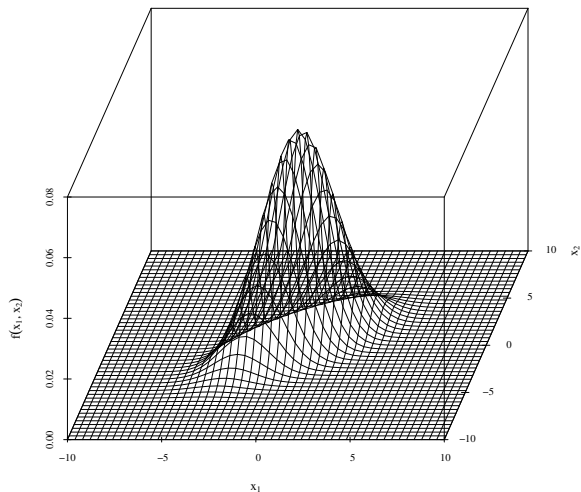
Draw Lines the Other Way

- for each x_2 , draw a line from lowest to highest x_1
- line traces out density at (x_1, x_2)

```
for(i in length(x2):1) {  
  s3d$points3d(x1, rep(x2[i], length(x1)), dens[,  
    i], type = "l")  
}
```


Draw Lines the Other Way

Bivariate normal distribution



Use R's text function with plotmath to Write Equation

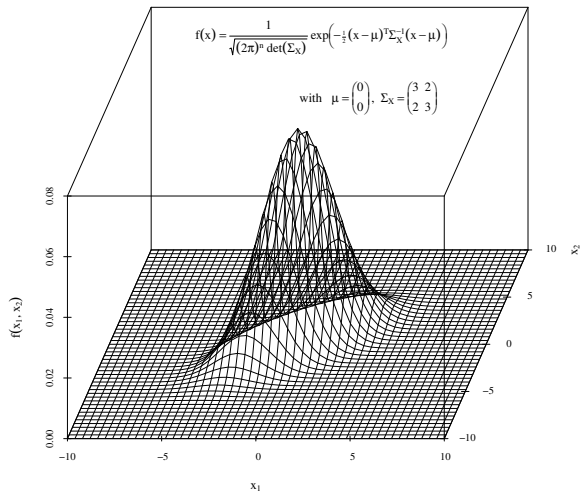
```
text(s3d$xyz.convert(-1, 10, 0.07), labels = expression(f(x) == frac
  (1, sqrt((2 * pi)^n * phantom(".") * det(Sigma[X]))) * phantom(
    ".") * exp * { bgroup("(", - scriptstyle(frac(1, 2) * phantom("
      .")) *
    (x - mu)^T * Sigma[X]^(-1) * (x - mu), ")")}))
### fix. insert {} around Sigma[X] == ... ##
text(s3d$xyz.convert(1.5, 10, 0.05), labels = expression("with" *
  phantom("m") * mu == bgroup("(", atop(0, 0), ")") * phantom(".")
    ) * "," * phantom(0) * {Sigma[X] == bgroup("(", atop(3 *
    phantom(0) * 2, 2 * phantom(0) * 3), ")") } )
```

The first one is the probability density function (PDF)

The second one is the Expected Value and Variance matrix

Use Plotmath

Bivariate normal distribution



About cloud

- Like other procedures in the lattice package (tremendous result for small effort)
- More difficult to customize plots (my humble opinion)
- Convenient presentation of plots “by group” or “by sex” or such.

Outline

- 1 Overview
- 2 persp
- 3 scatter3d
- 4 Scatterplot3d
- 5 rockchalk**
 - mcGraph
 - plotPlane

3D Tools in rockchalk

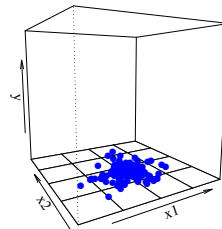
- The lack of adjust-ability of scatterplot3d caused me to not rely on it too heavily
- I don't want to interactively point-and-click the way rgl requires.
- I could not make lattice output combine different components in the way I wanted to.
- But I could make persp work, sometimes.
- So I kept track of things I could succeed with and boiled them down to functions in rockchalk.

Depicting Multicollinearity: My first 3d functions

- `mcGraph1(x1, x2, y)`: Creates an “empty box” showing the footprint of the $(x1, x2)$ pairs in the bottom of the display.
- `mcGraph2(x1, x2, y, rescaley=0.5)`: Shows points “rising above” footprint
- `mcGraph3(x1, x2, y)`: fits a regression of y on $x1$ and $x2$, and plots it. Includes optional interaction term.

mcGraph1

- No values drawn yet for dependent variable
- Please notice dispersion in the x_1 - x_2 plane



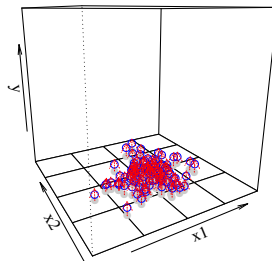
```
mod1 ← mcGraph1(dat$x1, dat$x2, dat$y, theta=-30,  
  phi=8)
```


mcGraph uses rescaley argument

- The true relationship is

$$y_i = .2x_{1i} + .2x_{2i} + e_i, e_i \sim N(0, 7^2)$$

- $\rho_{x1,x2} = 0.1$

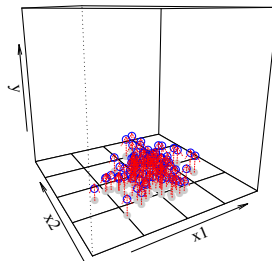


```
mod ← mcGraph2(dat$x1, dat$x2, dat$y, rescaley =  
  0.1, theta = -30)
```

Step up rescaley bit by bit, its almost a movie!

- The true relationship is

$$y_i = .2x1_i + .2x2_i + e_i, e_i \sim N(0, 7^2)$$

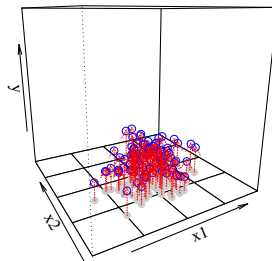


```
mod ← mcGraph2(dat$x1, dat$x2, dat$y, rescaley =  
0.2, theta = -30)
```

Step up rescaley bit by bit, its almost a movie!

- The true relationship is

$$y_i = .2x1_i + .2x2_i + e_i, e_i \sim N(0, 7^2)$$

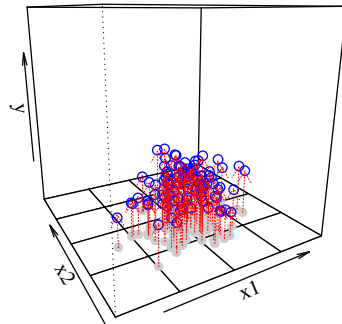


```
mod ← mcGraph2(dat$x1, dat$x2, dat$y, rescaley =  
0.3, theta = -30)
```

Step up rescaley bit by bit, its almost a movie!

- The true relationship is

$$y_i = .2x1_i + .2x2_i + e_i, e_i \sim N(0, 7^2)$$

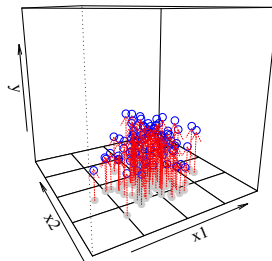


```
mod ← mcGraph2(dat$x1, dat$x2, dat$y, rescaley =
```

Step up rescaley bit by bit, its almost a movie!

- The true relationship is

$$y_i = .2x1_i + .2x2_i + e_i, e_i \sim N(0, 7^2)$$

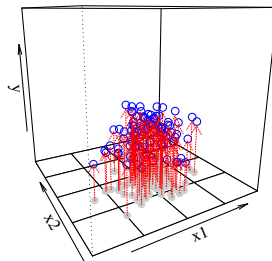


```
mod ← mcGraph2(dat$x1, dat$x2, dat$y, rescaley =  
0.5, theta = -30)
```

Step up rescaley bit by bit, its almost a movie!

- The true relationship is

$$y_i = .2x1_i + .2x2_i + e_i, e_i \sim N(0, 7^2)$$

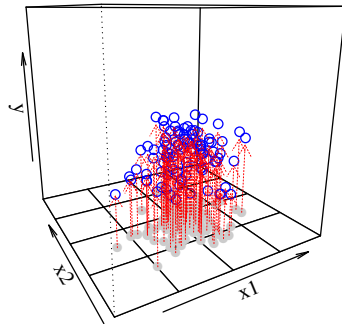


```
mod ← mcGraph2(dat$x1, dat$x2, dat$y, rescaley =  
0.6, theta = -30)
```

Step up rescaley bit by bit, its almost a movie!

- The true relationship is

$$y_i = .2x1_i + .2x2_i + e_i, e_i \sim N(0, 7^2)$$

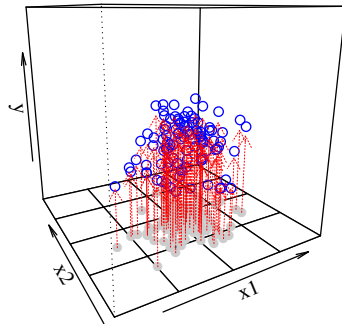


```
mod <- mcGraph2(dat$x1, dat$x2, dat$y, rescaley =
```

Step up rescaley bit by bit, its almost a movie!

- The true relationship is

$$y_i = .2x1_i + .2x2_i + e_i, e_i \sim N(0, 7^2)$$

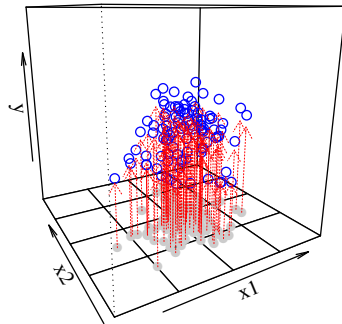


```
mod <- mcGraph2(dat$x1, dat$x2, dat$y, rescaley =
```


Step up rescaley bit by bit, its almost a movie!

- The true relationship is

$$y_i = .2x1_i + .2x2_i + e_i, e_i \sim N(0, 7^2)$$

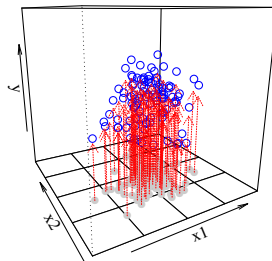


```
mod <- mcGraph2(dat$x1, dat$x2, dat$y, rescaley =
```

Step up rescaley bit by bit, its almost a movie!

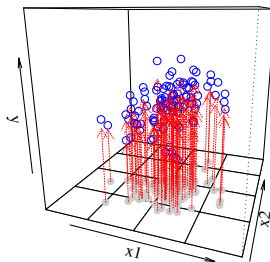
- The true relationship is

$$y_i = .2x1_i + .2x2_i + e_i, e_i \sim N(0, 7^2)$$



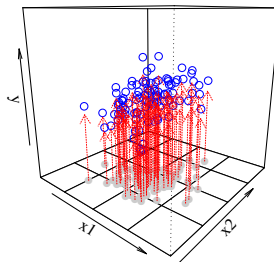
```
mod ← mcGraph2(dat$x1, dat$x2, dat$y, rescaley =  
1.0, theta = -30)
```

Can Spin the Cloud (Just Showing Off)



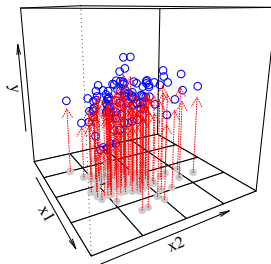
```
mod ← mcGraph2(dat$x1, dat$x2, dat$y, rescaley =  
  1.0, theta = 20)
```

Can Spin the Cloud (Just Showing Off)



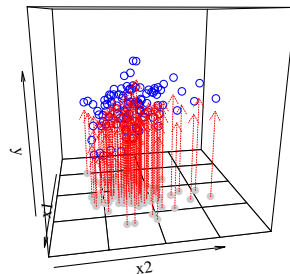
```
mod ← mcGraph2(dat$x1, dat$x2, dat$y, rescaley =  
  1.0, theta = 40)
```

Can Spin the Cloud (Just Showing Off)



```
mod ← mcGraph2(dat$x1, dat$x2, dat$y, rescaley =  
  1.0, theta = 60)
```

Can Spin the Cloud (Just Showing Off)

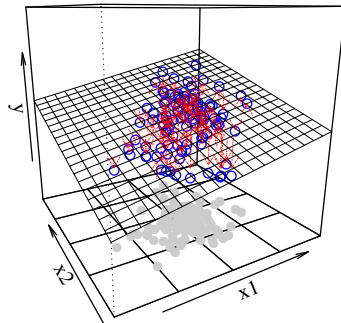


```
mod ← mcGraph2(dat$x1, dat$x2, dat$y, rescaley =  
  1.0, theta = 80)
```

Regression Plane Sits Nicely in the Data Cloud

	M1	
	Estimate	(S.E.)
(Intercept)	-0.678	(4.345)
x1	0.18*	(0.066)
x2	0.229*	(0.069)
N	100	
RMSE	6.717	
R^2	0.194	
adj R^2	0.178	

* $p \leq 0.05$

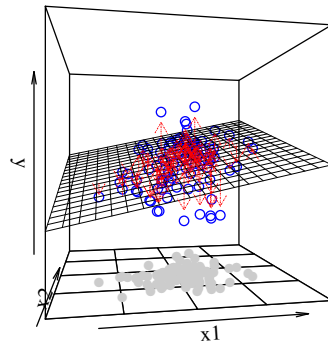


```
mod1 ← mcGraph3(dat$x1, dat$x2, dat$y, theta =
```

Regression Plane Sits Nicely in the Data Cloud

	M1	
	Estimate	(S.E.)
(Intercept)	-0.678	(4.345)
x1	0.18*	(0.066)
x2	0.229*	(0.069)
N	100	
RMSE	6.717	
R^2	0.194	
adj R^2	0.178	

* $p \leq 0.05$

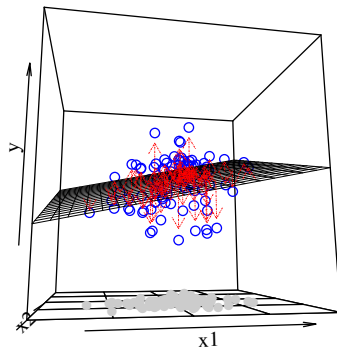


```
mod1 ← mcGraph3(dat$x1, dat$x2, dat$y, theta =
```


Regression Plane Sits Nicely in the Data Cloud

	M1	
	Estimate	(S.E.)
(Intercept)	-0.678	(4.345)
x1	0.18*	(0.066)
x2	0.229*	(0.069)
N	100	
RMSE	6.717	
R^2	0.194	
adj R^2	0.178	

* $p \leq 0.05$

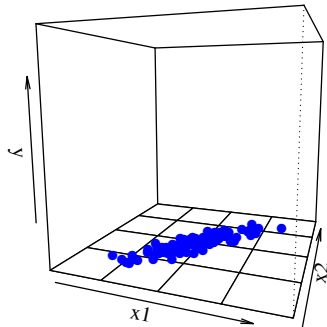


```
mod1 <- mcGraph3(dat$x1, dat$x2, dat$y, theta =
```

```
10, phi = 10)
```

Severe Collinearity: $r(x_1, x_2) = 0.9$

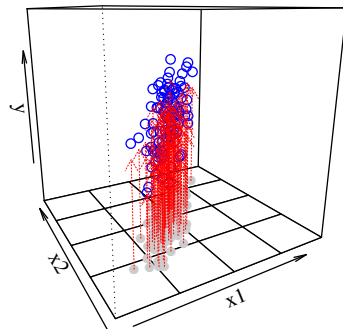
- Nearly linear dispersion in the x_1 - x_2 plane



```
mod2 ← mcGraph1(dat2$x1, dat2$x2, dat2$y, theta
```

Cloud Is More like Data Tube

```
mod ← mcGraph2(  
  dat2$x1, dat2$  
  x2, dat2$y,  
  theta = -30)
```

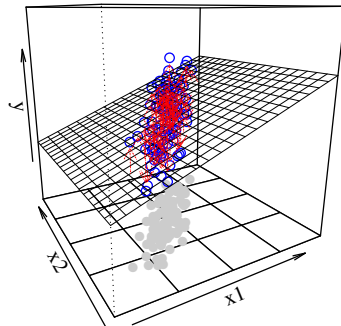


Cloud Is More like Data Tube

	M1	
	Estimate	(S.E.)
(Intercept)	2.975	(4.128)
x1	0.365*	(0.179)
x2	-0.017	(0.173)
N	100	
RMSE	7.162	
R^2	0.165	
adj R^2	0.148	

* $p \leq 0.05$

- plane does not sit "comfortably"
- greater standard errors

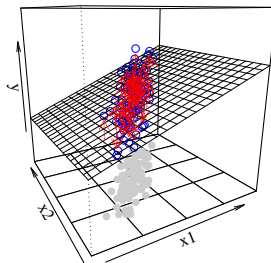


```
mod ← mcGraph3(dat2$x1, dat2$x2, dat2$y, theta =
```

Fit Interaction $\text{lm}(y \sim x1 * x2)$

	M1	
	Estimate	(S.E.)
(Intercept)	4.997	(17.977)
x1	0.324	(0.394)
x2	-0.058	(0.4)
x1:x2	0.001	(0.007)
N	100	
RMSE	7.199	
R^2	0.166	
adj R^2	0.14	

* $p \leq 0.05$



```
mod ← mcGraph3(dat2$x1, dat2$x2, dat2$y,
  interaction=TRUE, theta = -30)
```

Next Step: Plot any Fitted Regression

- After mcGraph worked, I was encouraged (because I could fill up a whole lecture on multicollinearity)
- But the mcGraph interface was too limiting
 - had to specify and provide variables
 - could not work with larger regression models

plotPlane: quick regression tool for presentations

- Generate data, fit a model with 4 predictors,

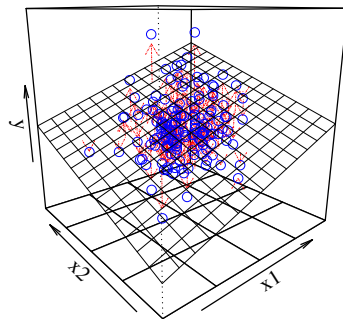
```
dat3 <- genCorrelatedData(N
  =150, beta = c(0, 0.15,
    0.25, 0.1), stde = 150)
dat3$x3 <- rpois(150,
  lambda = 7)
dat3$x4 <- rgamma(150, 2,
  1)
m1 <- lm(y ~ x1 + x2 + x3 +
  x4, data=dat3)
```

	M1	
	Estimate	(S.E.)
(Intercept)	-269.01*	(95.835)
x1	5.308*	(1.161)
x2	5.094*	(1.332)
x3	-0.371	(5.05)
x4	10.649	(9.45)
N	150	
RMSE	152.068	
R^2	0.198	
adj R^2	0.176	

* $p \leq 0.05$

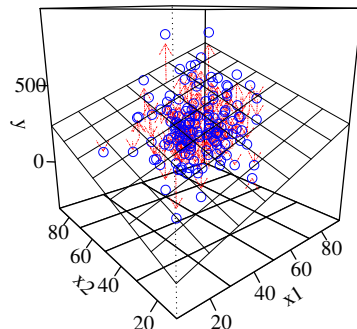
plotPlane: choose x1 and x2

```
plotPlane(m1,  
  plotx1 = "x1",  
  plotx2 = "x2",  
  , theta = -40,  
  npp = 15,  
  drawArrows =  
  TRUE)
```



plotPlane: choose x1 and x2

```
plotPlane(m1,  
  plotx1 = "x1",  
  plotx2 = "x2",  
  , theta = -40,  
  npp = 8, llwd  
    = 0.105 ,  
  drawArrows =  
    TRUE, ticktype  
    = "detailed")
```

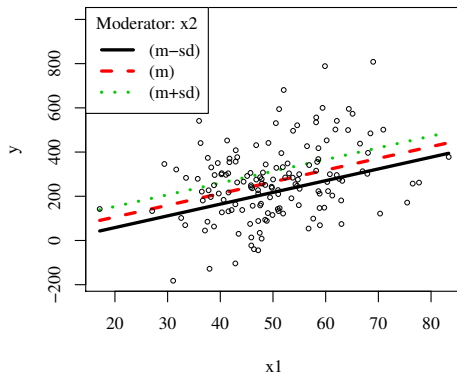


Interchange Information between 2D and 3D plots

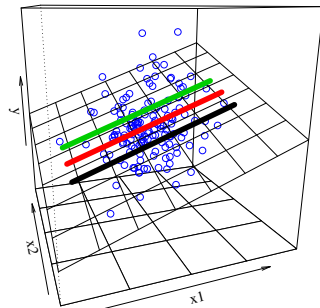
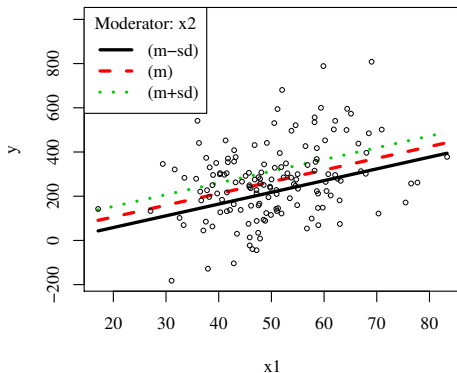
Putting x_3 and x_4 at their means, plot the predicted values for several values of x_2 with plotSlopes

```
ps30 <- plotSlopes(m1,  
  plotx = "x1", modx = "x2",  
  , modxVals = "std.dev."  
  , llwd = 3)
```

The output object ps30 has information in it that can be used to supplement a 3D graph.



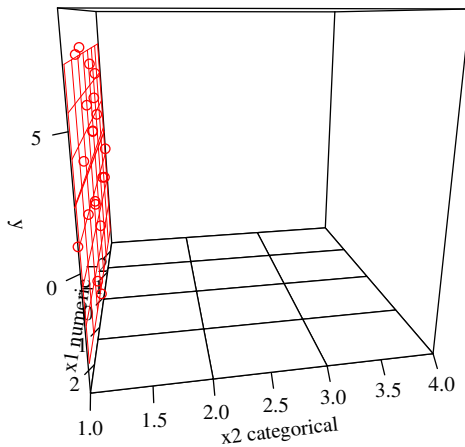
Compare 3D and 2D depictions



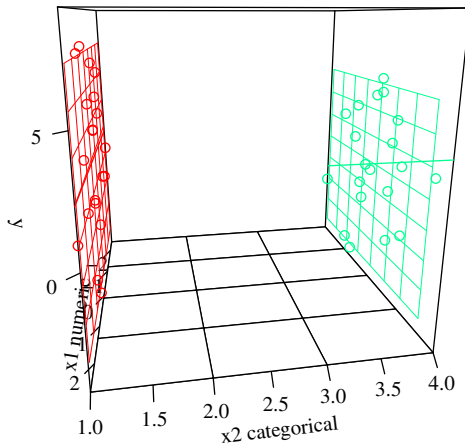
Next Step: Visualization of Factor Predictors

- Suppose x_2 is a categorical variable.
- Shouldn't force that to a numeric scale and 3D plot with an ordinary plane, should we?
- lattice package tools can draw one plot per level of the factor (maybe that's best)
- But I've wrestled trying to find a more informative view

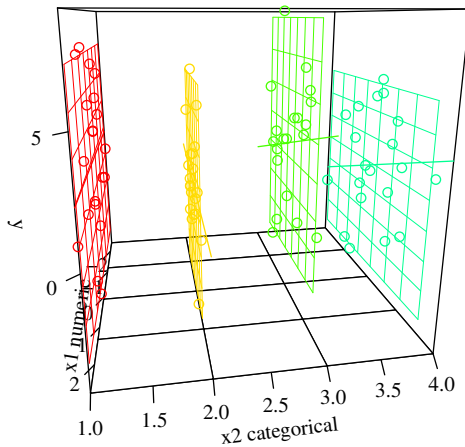
Group 1



Group 1, Group 4



Groups 1-4



Conclusion

- If you can “sketch” what you want with a pencil, you can probably get R to draw you a good example.
- Search (AGGRESSIVELY) for working example code from problems like yours. Accumulate them whenever you find them.
- If you want a quick view of a regression model—either linear or not linear—I’d suggest `plotPlane`