Better Growth Modeling: cactus case study

Tom Miller

6/12/2020

About the data

This document walks through growth modeling with the cactus (*Cylyndriopuntia imbricata*, or CYIM) case study. The plants are a little oddly shaped. We measure plant size by taking maximum height, maximum width of the crown, and then an additional width measurement perpendicular to the maximum. We then convert this into the volume of a cone (cm^3) and take the natural log of this as the size variable.

There are a total of 827 individuals in the data set spanning 8 plots or spatial blocks and 9 transition-years. There are some additional, earlier years of data but they come from different plots and that was causing problems in the shrinkage model. So I filtered out these early years. Note that even in these later years, plots 7 and 8 did not start until 2011.

```
##
##      2009 2010 2011 2012 2013 2014 2015 2016 2017
##   1    81   72   60   67   65   64   62   79   72
##   2    58   55   42   42   40   38   38   64   56
##   3    81   75   70   83   84   85   80  102   99
##   4    89   82   66   64   64   64   63  115   96
##   5    43   43   44   47   46   45   45   49   47
##   6    76   67   56   64   66   65   64   83   83
##   7     0    0   47   54   53   53   51   84   79
##   8     0    0   72   75   72   70   83   82   77
```

Gaussian fits

Use lme4 to fit Gaussian models of the form $size_{t+1} \sim size_t$. There are temporal and spatial random effects. No obvious fixed effects besides initial size (this is an observational study) but I can test whether a quandratic

term for initial size improves fit. Random effects are intercept-only because convergence problems otherwise.

```
CYIM_lmer_models <- list()
CYIM_lmer_models[[1]] <- lmer(log(vol_t1) ~ log(vol_t) + (1 | year_t) + (1 | plot),
    data = CYIM, REML = F, control = lmerControl(optimizer = "bobyqa"))
CYIM_lmer_models[[2]] <- lmer(log(vol_t1) ~ log(vol_t) + I(log(vol_t)^2) + (1 | year_t) +
    (1 | plot), data = CYIM, REML = F, control = lmerControl(optimizer = "bobyqa"))
```

Now use the iterative re-weighting approach to re-fit these with non-constant variance:

```
## NegLogLik function to fit variance model for residuals
varPars = function(pars) {
  return(-sum(dnorm(resids, mean=0, sd=exp(pars[1] + pars[2]*fitted_vals),log=TRUE)))
}

pars<-list()
for(mod in 1:length(CYIM_lmer_models)) {
  err = 1; rep=0;
  ## Steve's error level was too high a bar to clear. I raised this.
  while(err > 0.0001) {
    rep=rep+1; model = CYIM_lmer_models[[mod]];
    fitted_vals = fitted(model);resids = residuals(model);
    out=optim(c(sd(resids),0),varPars,control=list(maxit=5000));
    pars[[mod]]=out$par;
    new_sigma = exp(pars[[mod]][1] + pars[[mod]][2]*fitted_vals); new_weights = 1/((new_sigma)^2)
    new_weights = 0.5*(weights(model) + new_weights); # cautious update
    new_model <- update(model,weights=new_weights);
    err = weights(model)-weights(new_model); err=sqrt(mean(err^2));
    cat(mod,rep,err,"\n") # check on convergence of estimated weights
    CYIM_lmer_models[[mod]]<-new_model;
  }}
```

For fair AIC comparison, re-fit both models with best weights.

```
######### For a fair AIC comparison, fit all models with the same weights
aics = unlist(lapply(CYIM_lmer_models, AIC))
best_model = which(aics == min(aics))
best_weights = weights(CYIM_lmer_models[[best_model]])
for (mod in 1:length(CYIM_lmer_models)) {
    CYIM_lmer_models[[mod]] <- update(CYIM_lmer_models[[mod]], weights = best_weights)
}
aictab(CYIM_lmer_models)
```

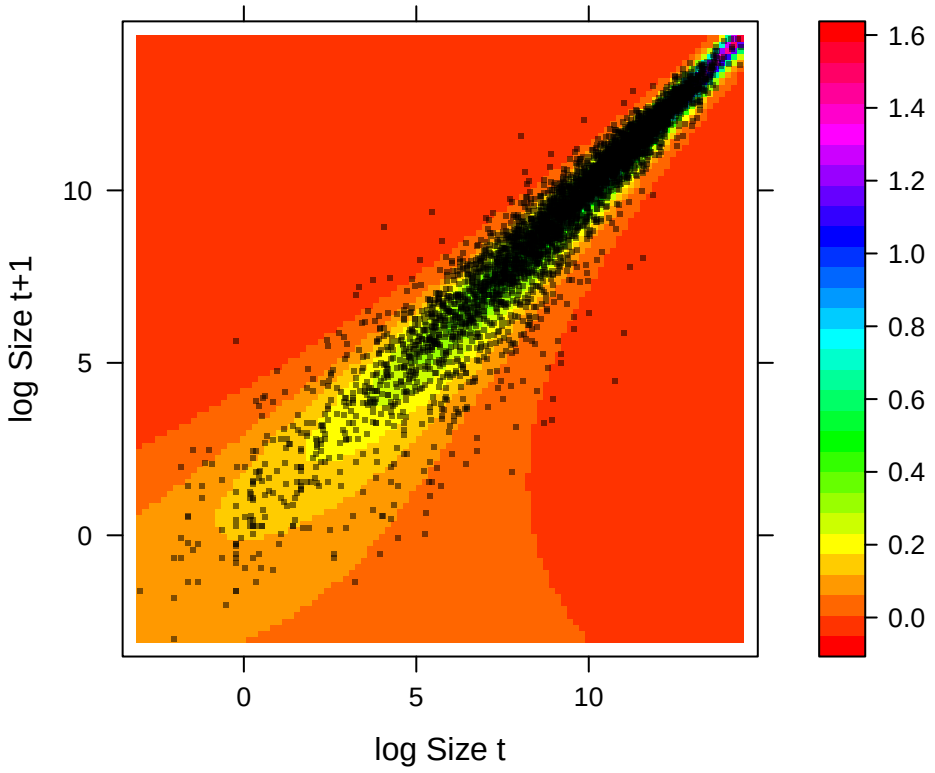
```
## Warning in aictab.AIClmerModLmerTest(CYIM_lmer_models):
## Model names have been supplied automatically in the table

##
## Model selection based on AICc:
##
##      K    AICc Delta_AICc AICcWt Cum.Wt       LL
## Mod1 5 9016.95       0.00   0.62   0.62 -4503.47
## Mod2 6 9017.94       0.99   0.38   1.00 -4502.96
```

The linear model is favored. Last step is to re-fit with REML=T.

```
aics = unlist(lapply(CYIM_lmer_models, AIC))
best_model = which(aics == min(aics))
CYIM_lmer_best = CYIM_lmer_models[[best_model]]
best_weights = weights(CYIM_lmer_best)
CYIM_lmer_best <- lmer(log(vol_t1) ~ log(vol_t) + (1 | year_t) + (1 | plot), data = CYIM,
    weights = best_weights, REML = TRUE)
```

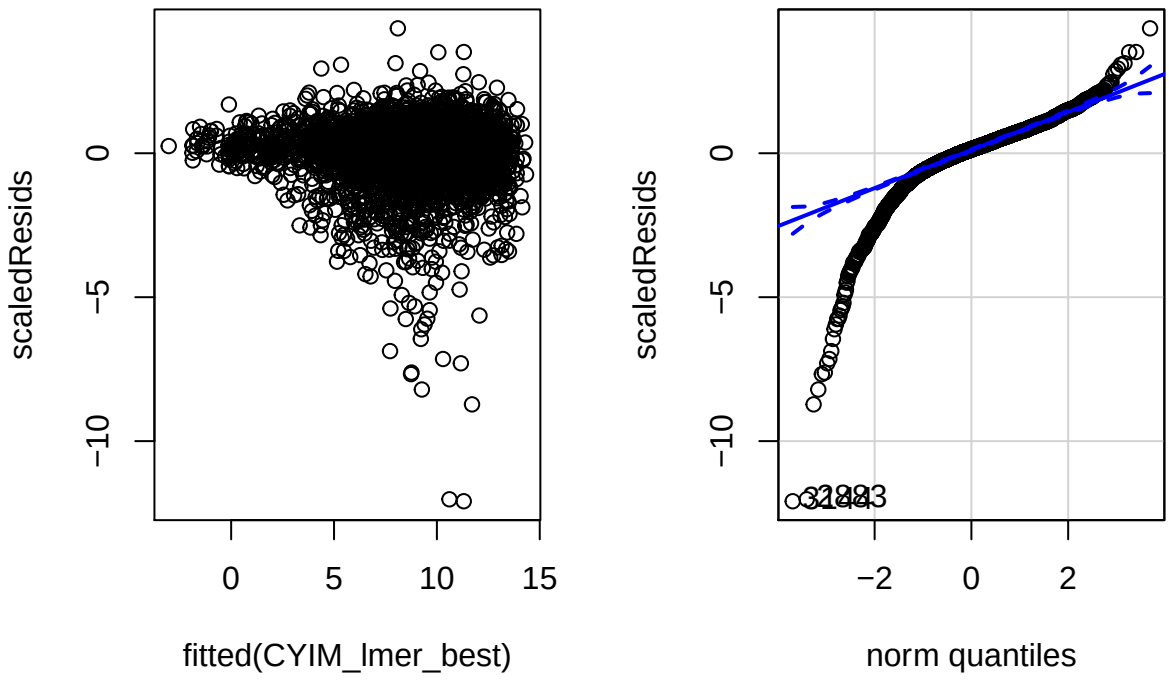
Now visualize this kernel. This model is the *very best* we could do in a Gaussian framework (assuming there are no major fixed effects we are missing).

```
## dummy variable for initial size
size_dim <- 100
size_dum <- seq(min(log(CYIM$vol_t)), max(log(CYIM$vol_t)), length.out = size_dim)
## make a polygon for the Gaussian kernel with non-constant variance
CYIM_lmer_best_kernel <- matrix(NA, size_dim, size_dim)
for (i in 1:size_dim) {
    mu_size <- fixef(CYIM_lmer_best)[1] + fixef(CYIM_lmer_best)[2] * size_dum[i]
    CYIM_lmer_best_kernel[, i] <- dnorm(size_dum, mean = mu_size, sd = exp(pars[[best_model]][1] +
        pars[[best_model]][2] * mu_size))
}

levelplot(CYIM_lmer_best_kernel, row.values = size_dum, column.values = size_dum,
    cuts = 30, col.regions = rainbow(30), xlab = "log Size t", ylab = "log Size t+1",
    panel = function(...) {
        panel.levelplot(...)
        grid.points(log(CYIM$vol_t), log(CYIM$vol_t1), pch = ".", gp = gpar(cex = 3,
            col = alpha("black", 0.5)))
    })
```



Visually, this looks great. But is it any good? No. Scaled residuals are not remotely Gaussian.

```
scaledResids = residuals(CYIM_lmer_best) * sqrt(weights(CYIM_lmer_best))
par(mfrow = c(1, 2))
plot(fitted(CYIM_lmer_best), scaledResids)
qqPlot(scaledResids)  # really bad in both tails
```




```
## [1] 3144 2883
```

```
jarque.test(scaledResids)  # normality test: FAILS, P < 0.001
```

```
##
##  Jarque-Bera Normality Test
##
## data:  scaledResids
## JB = 68502, p-value < 2.2e-16
## alternative hypothesis: greater
```

```
anscombe.test(scaledResids)  # kurtosis: FAILS, P < 0.001
```

```
##
##  Anscombe-Glynn kurtosis test
##
## data:  scaledResids
## kurt = 21.353, z = 32.724, p-value < 2.2e-16
## alternative hypothesis: kurtosis is not equal to 3
```

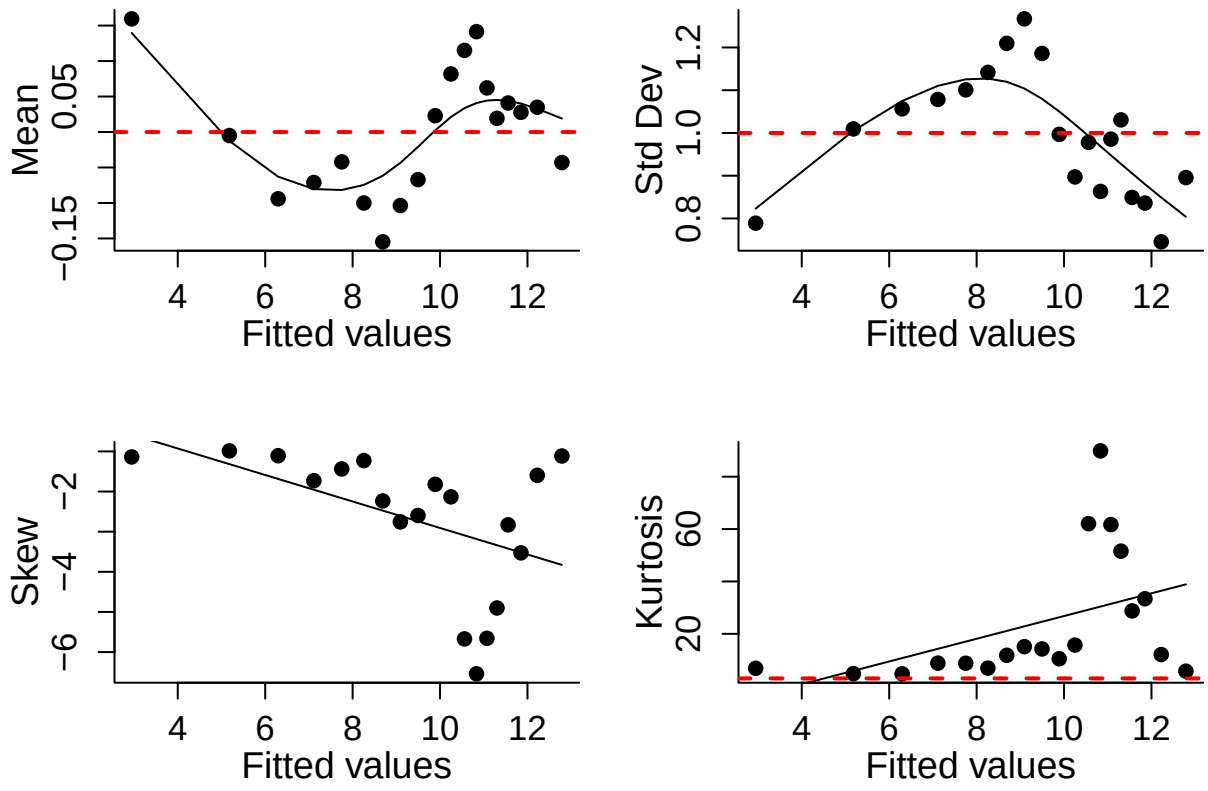
```
agostino.test(scaledResids)  # skewness: FAILS, P<0.001
```

```
##
```

```
##  D'Agostino skewness test
##
## data:  scaledResids
## skew = -2.6615, z = -42.5617, p-value < 2.2e-16
## alternative hypothesis: data have a skewness
```

One last look at the standardized residuals. They are roughly mean zero and unit variance – so that checks out. But there is negative skew and excess kurtosis, especially at large sizes.

```
px = fitted(CYIM_lmer_best)
py = scaledResids
par(mfrow = c(2, 2), bty = "l", mar = c(4, 4, 2, 1), mgp = c(2.2, 1, 0), cex.axis = 1.4,
    cex.lab = 1.4)
z = rollMoments(px, py, windows = 10, smooth = TRUE, scaled = TRUE)
```



Finding a better distribution

We now know the Gaussian provides a poor fit to the residual variance. We would like to know which distribution provides a better - ideally *good* - fit. Because there is size-dependence in skew and kurtosis, we cannot marginalize over the entire distribution of residuals (this may point us in the wrong direction). Instead, we can slice up the data into bins of expected value and find the best distribution for each bin using gamlss' fitDist().

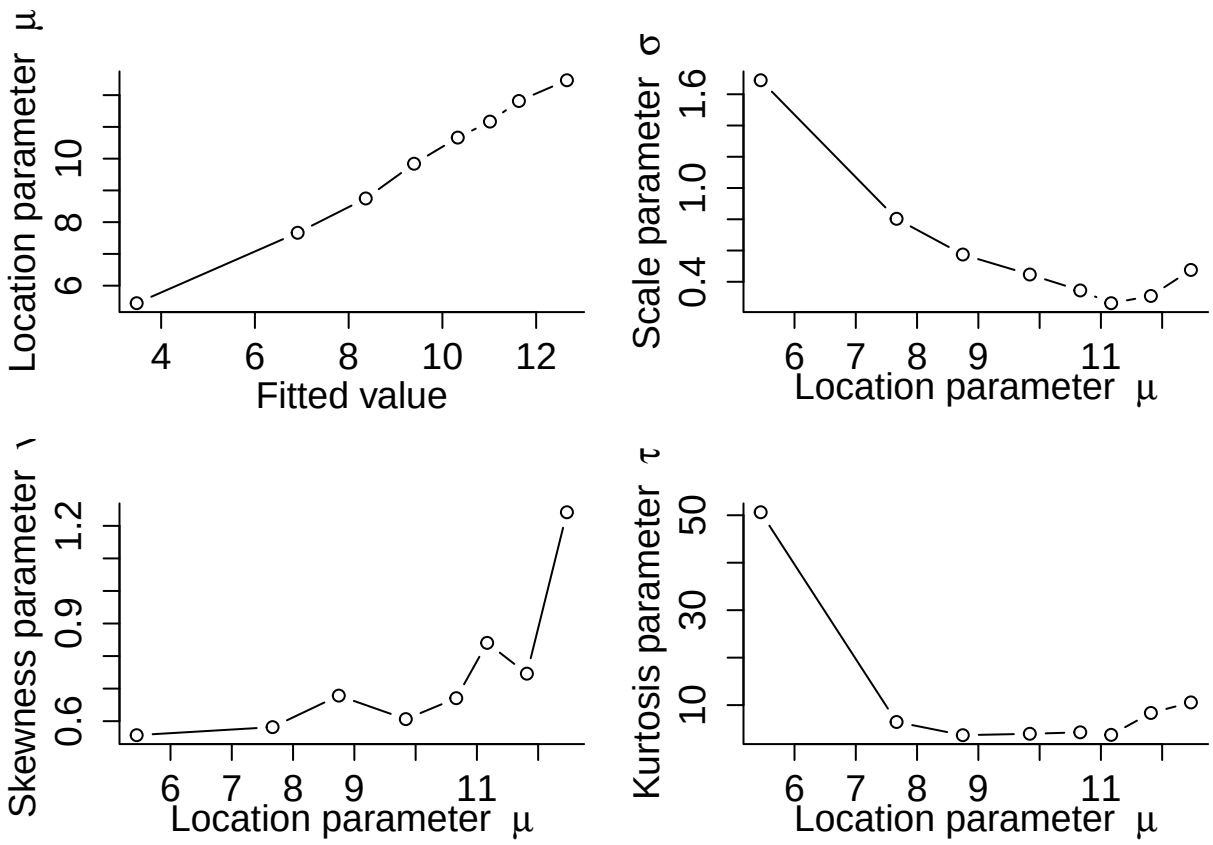
```
n_bins <- 8
## I need to rewrite this code because it runs fitDist 4 separate times
select_dist <- tibble(fit_best = fitted(CYIM_lmer_best), scale_resid = residuals(CYIM_lmer_best) *
    sqrt(weights(CYIM_lmer_best))) %>% mutate(bin = as.integer(cut_number(fit_best,
    n_bins)), best_dist = NA, secondbest_dist = NA, aic_margin = NA)
for (b in 1:n_bins) {
    bin_fit <- fitDist(select_dist$scale_resid[select_dist$bin == b], type = "realline")
    select_dist$best_dist[select_dist$bin == b] <- names(bin_fit$fits[1])
    select_dist$secondbest_dist[select_dist$bin == b] <- names(bin_fit$fits[2])
    select_dist$aic_margin[select_dist$bin == b] <- bin_fit$fits[2] - bin_fit$fits[1]
}
(select_dist <- select_dist %>% group_by(bin) %>% summarise(n_bin = n(), best_dist = unique(best_dist),
    secondbest_dist = unique(secondbest_dist), aic_margin = unique(aic_margin)))
```

```
## # A tibble: 8 x 5
##     bin n_bin best_dist secondbest_dist aic_margin
##   <int> <int> <chr>     <chr>                <dbl>
## 1     1   563 SHASH     SEP4              4.01e+ 0
## 2     2   563 SHASH     SEP4              1.76e+ 0
## 3     3   562 JSU       JSUo              1.44e-10
## 4     4   563 SHASH     ST1               1.38e+ 0
## 5     5   563 ST4       ST1               3.05e- 1
## 6     6   562 ST4       NET               2.63e+ 0
## 7     7   563 ST5       ST2               1.58e- 1
## 8     8   563 ST1       ST5               1.42e- 1
```

It's a little bit of everything and convergence is not great (I've suppressed those warnings in this output). The skewed t pops up a lot though, so I will procede with that one and hope in the end the fit is decent. To guide the skewed t fits we can visualize how the higher moments are related to the fitted value.

```
CYIM_bin_fit <- CYIM %>% mutate(fitted = fitted(CYIM_lmer_best), bin = as.integer(cut_number(fitted,
    n_bins))) %>% mutate(mu = NA, sigma = NA, nu = NA, tau = NA)
for (b in 1:n_bins) {
    ## I get the most stable tau estimates with ST3
    bin_fit <- gamlssML(log(CYIM_bin_fit$vol_t1[CYIM_bin_fit$bin == b]) ~ 1, family = "ST3")
    CYIM_bin_fit$mu[CYIM_bin_fit$bin == b] <- bin_fit$mu
    CYIM_bin_fit$sigma[CYIM_bin_fit$bin == b] <- bin_fit$sigma
    CYIM_bin_fit$nu[CYIM_bin_fit$bin == b] <- bin_fit$nu
    CYIM_bin_fit$tau[CYIM_bin_fit$bin == b] <- bin_fit$tau
}
CYIM_bin_fit <- CYIM_bin_fit %>% group_by(bin) %>% summarise(N = n(), mean_fitted = mean(fitted),
    mu = unique(mu), sigma = unique(sigma), nu = unique(nu), tau = unique(tau))

par(mfrow = c(2, 2), bty = "l", mar = c(4, 4, 2, 1), mgp = c(2.2, 1, 0), cex.axis = 1.4,
    cex.lab = 1.4)
## Steve's spline.scatter.smooth() function not working for me and I did not both
## trying to figure out why
plot(CYIM_bin_fit$mean_fitted, CYIM_bin_fit$mu, xlab = "Fitted value", ylab = expression(paste("Location
    mu")), type = "b")
plot(CYIM_bin_fit$mu, CYIM_bin_fit$sigma, xlab = expression(paste("Location parameter  ",
    mu)), ylab = expression(paste("Scale parameter  ", sigma)), type = "b")
plot(CYIM_bin_fit$mu, CYIM_bin_fit$nu, xlab = expression(paste("Location parameter  ",
    mu)), ylab = expression(paste("Skewness parameter  ", nu)), type = "b")
```

```
plot(CYIM_bin_fit$mu, CYIM_bin_fit$tau, xlab = expression(paste("Location parameter  ",
    mu)), ylab = expression(paste("Kurtosis parameter  ", tau)), type = "b")
```



Fitting the final model

Now we can fit a custom model via maximum likelihood, matching the structure of the best lmre model but fitting the random effects as fixed instead and estimating the corresponding variances with Steve's shrinkage methods. The likelihood function is based on the skewed t (using gamlss' dST3() - the parameterization that seemed best behaved.)

First we will defined the linear predictor for the location parameter mu (which is not necessarily the expected value). Note that this with parameterzation, the intercept is year1/plot1.

```
U = model.matrix(~0 + year_t + plot + log(vol_t), data = CYIM)
```

Next define a likelihood function using this linear predictor for the location. I am including quadratic terms for sigma and tau based on preliminary fit where linear models for these parameters led to a not-so-great fit at some sizes. Quadratic terms are also suggested in the moment plots above.

```
LogLik = function(pars, response, U) {
    pars1 = pars[1:ncol(U)]
    pars2 = pars[-(1:ncol(U))]
    mu = U %*% pars1
    val = dST3(x = response, mu = mu, sigma = exp(pars2[1] + pars2[2] * mu + pars2[3] *
```

```
        mu^2), nu = exp(pars2[4] + pars2[5] * mu), tau = exp(pars2[6] + pars2[7] *
        mu + pars2[8] * mu^2), log = T)
    return(val)
}
```

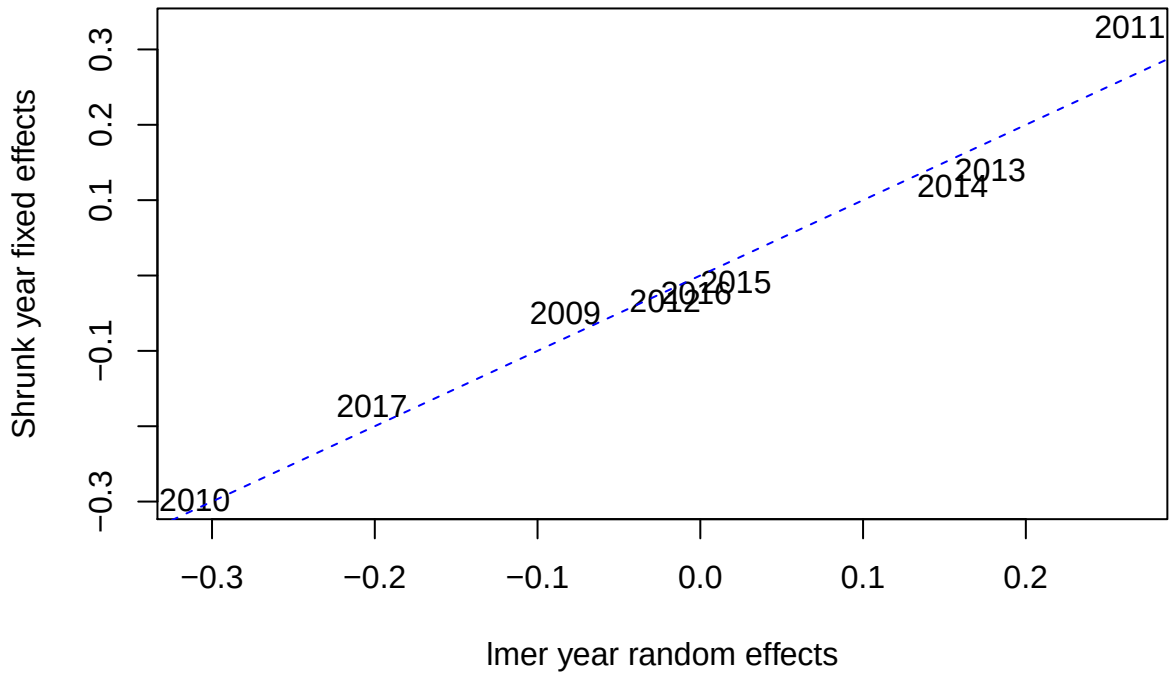
Now fit it, using the lmer() fit for the mean (location) and lm() fits for the higher moments as starting parameter values. This comes straight from Steve's PSSP code, including his convergence paranoia, which is probably warranted. In preliminary fits, all the iterations converged on the same likelihoods, and this takes forever, so I am skimping on the paranoia here, but we might return to it to make this a good example to follow. This could also be a model selection step; easy to pull AIC from the maxLik fit.

```
paranoid_iter <- 1
coefs = list(paranoid_iter); LL=numeric(paranoid_iter);

# Starting values from the pilot model are jittered to do multi-start optimization).
# Using good starting values really speeds up convergence in the ML fits
# Linear predictor coefficients extracted from the lmer model
fixed_start = c(unlist(ranef(CYIM_lmer_best)$year_t) + unlist(ranef(CYIM_lmer_best)$plot)[1], #year est
                unlist(ranef(CYIM_lmer_best)$plot)[-1],
                fixef(CYIM_lmer_best)[2]) # size slope
## make sure the dimensions line up
#length(fixed_start);ncol(U);colnames(U)

# Shape and scale coefficients from the rollaply diagnostic plots
fit_sigma = lm(log(sigma)~mu + I(mu^2), data=CYIM_bin_fit)
fit_nu = lm(log(nu)~mu, data=CYIM_bin_fit)
fit_tau = lm(log(tau)~mu + I(mu^2), data=CYIM_bin_fit)
p0=c(fixed_start, coef(fit_sigma), coef(fit_nu),coef(fit_tau))

for(j in 1:paranoid_iter) {
  out=maxLik(logLik=LogLik,start=p0*exp(0.2*rnorm(length(p0))), response=log(CYIM$vol_t1),U=U,
             method="BHHH",control=list(iterlim=5000,printLevel=2),finalHessian=FALSE);

  out=maxLik(logLik=LogLik,start=out$estimate,response=log(CYIM$vol_t1),U=U,
             method="NM",control=list(iterlim=5000,printLevel=1),finalHessian=FALSE);

  out=maxLik(logLik=LogLik,start=out$estimate,response=log(CYIM$vol_t1),U=U,
             method="BHHH",control=list(iterlim=5000,printLevel=2),finalHessian=FALSE);

  coefs[[j]] = out$estimate; LL[j] = out$maximum;
  cat(j, "#---------------------------------------#",out$maximum,"\n");
}

j = min(which(LL==max(LL))) ## they actually all land on the same likelihood-that's good!
out=maxLik(logLik=LogLik,start=coefs[[j]],response=log(CYIM$vol_t1),U=U,
           method="BHHH",control=list(iterlim=5000,printLevel=2),finalHessian=TRUE)

######### save results of ML fit.
names(out$estimate)<-c(colnames(U),"sigma_b0","sigma_b1","sigma_b2","nu_b0","nu_b1","tau_b0","tau_b1","
coefs=out$estimate
## these are the indices of plot and year effects, to make the next steps a little more intuitive
years=1:9
plots=10:16
```

```
SEs = sqrt(diag(vcov(out)))
```

Now use Steve's shrinkage code to get the random effect variances and compare these to the lmer estimates. I will need Steve to explain to me what the shrinkage step is doing, and what is the theory for this (or I should do my own homework).

```
# shrinkage random effects for (1|year)
year_fixed.fx = coefs[years] - mean(coefs[years])
year_fixed.se = SEs[years]
year_sigma2.hat = mean(year_fixed.fx^2) - mean(year_fixed.se^2)
year_shrunk.fx = year_fixed.fx * sqrt(year_sigma2.hat/(year_sigma2.hat + year_fixed.se^2))
# lmer random effects for (1|year)
year_ran.fx = ranef(CYIM_lmer_best)["year_t"]

plot(year_ran.fx$year_t$`(Intercept)`, year_shrunk.fx, xlab = "lmer year random effects",
    ylab = "Shrunk year fixed effects", type = "n")
text(year_ran.fx$year_t$`(Intercept)`, year_shrunk.fx, labels = rownames(year_ran.fx$year_t))
abline(0, 1, col = "blue", lty = 2)
```




```
tibble(sd_estimate = c(sd(year_fixed.fx), sd(year_shrunk.fx), sd(year_ran.fx$year_t$`(Intercept)`)),
    method = c("fixed", "shrunk", "lme4"))
```

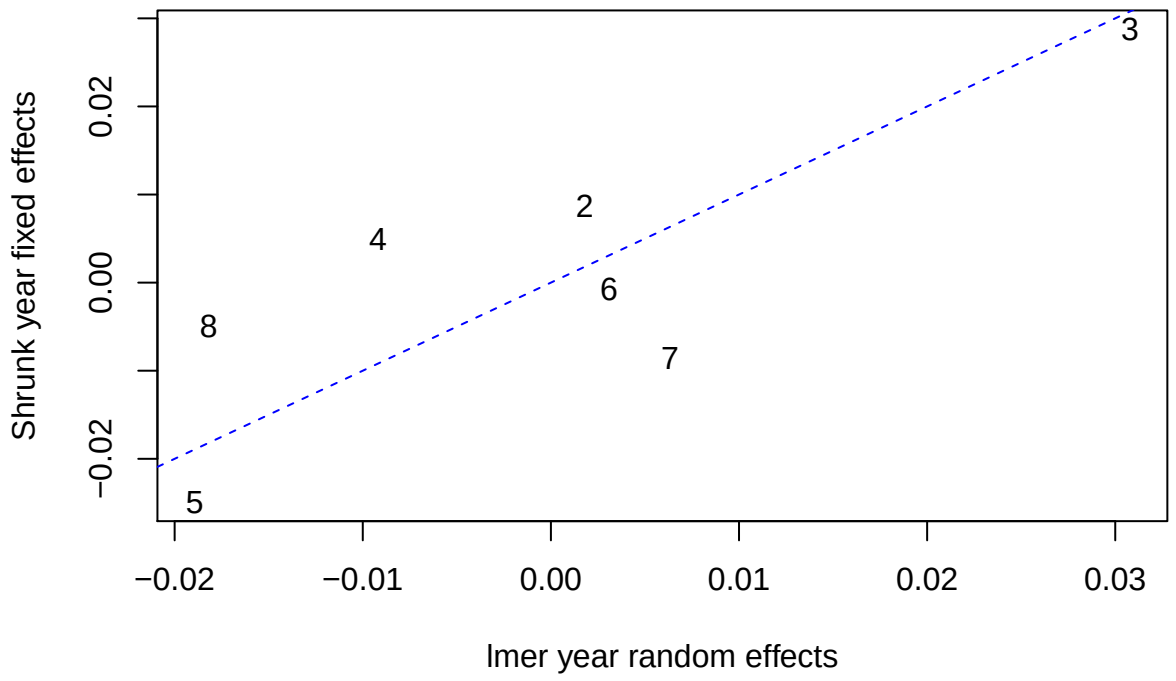
```
## # A tibble: 3 x 2
##   sd_estimate method
```

```
##          <dbl> <chr>
## 1        0.193 fixed
## 2        0.182 shrunk
## 3        0.184 lme4
```

The code for plots is more complicated than years, because plot effects were parameterized as contrasts. What is the SE of the plot 1 effect?? Is it the mean of the year SEs? – probably not! I am going to drop plot 1 from the shrunk effects, because I don't know how to calculate its SE. This will bias the variance, hopefully not too much.

These estimates don't look as good as the year effects but, again, the shrinkage variance corresponds well to the lme4 esimate.

```
# shrinkage random effects for (1|plot)
plot_coefs <- mean(coefs[years]) + coefs[plots]
plot_fixed.fx = plot_coefs - mean(plot_coefs)
plot_fixed.se = SEs[plots]
plot_sigma2.hat = mean(plot_fixed.fx^2) - mean(plot_fixed.se^2)
plot_shrunk.fx = plot_fixed.fx * sqrt(plot_sigma2.hat/(plot_sigma2.hat + plot_fixed.se^2))
# lmer random effects for (1|plot)
plot_ran.fx = ranef(CYIM_lmer_best)["plot"]

plot(plot_ran.fx$plot$`(Intercept)`, c(NA, plot_shrunk.fx), xlab = "lmer year random effects",
    ylab = "Shrunk year fixed effects", type = "n")
text(plot_ran.fx$plot$`(Intercept)`, c(NA, plot_shrunk.fx), labels = rownames(plot_ran.fx$plot))
abline(0, 1, col = "blue", lty = 2)
```



```
tibble(sd_estimate = c(sd(plot_fixed.fx), sd(plot_shrunk.fx), sd(plot_ran.fx$plot$`(Intercept)`)),
    method = c("fixed", "shrunk", "lme4"))
```

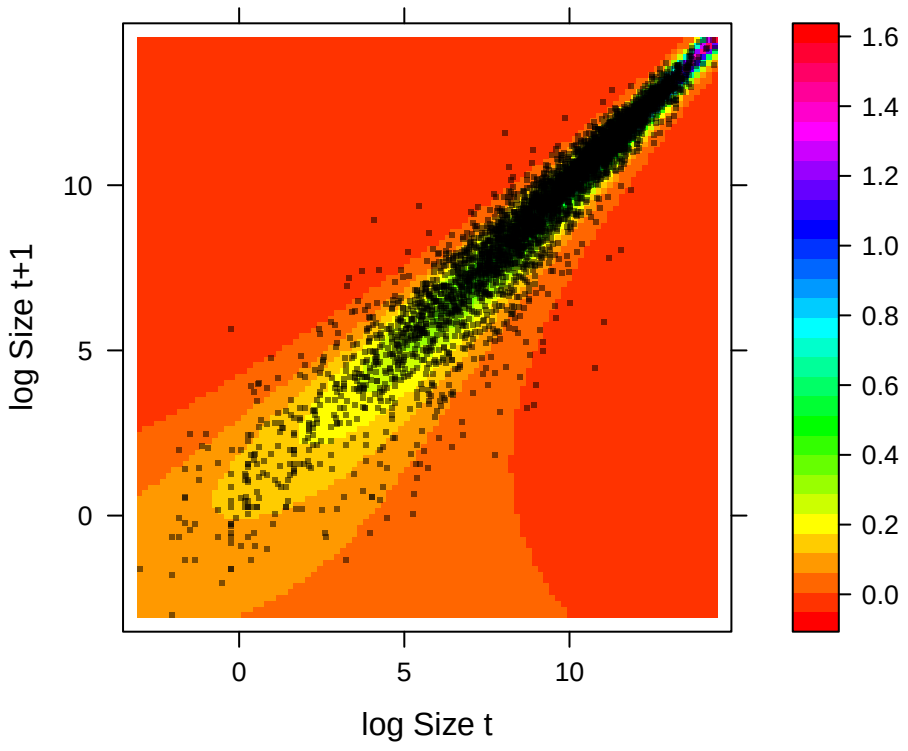
```
## # A tibble: 3 x 2
##   sd_estimate method
##         <dbl> <chr>
## 1      0.0303 fixed
## 2      0.0166 shrunk
## 3      0.0160 lme4
```

Steve had an interesting idea for an alternative way to fit two random effects, with one used as an offset for the other. Might swing back to that, especially since the reference-level problem is nagging at me.

Visual diagnostics of model fit via shrinkage

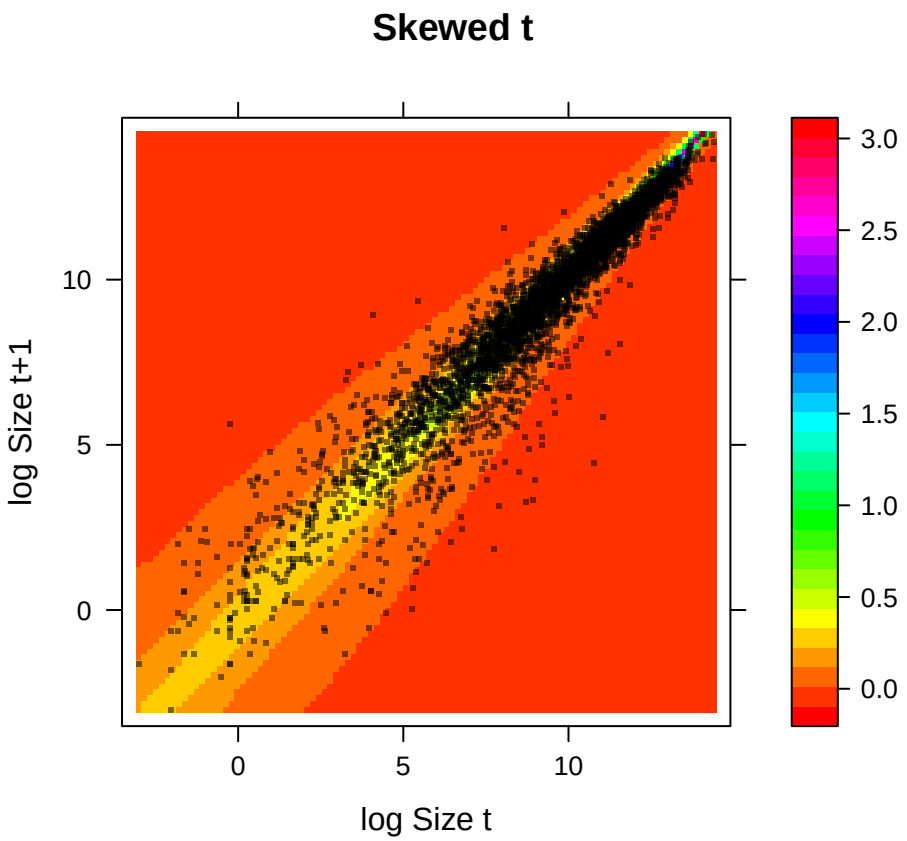
Here is the top-level view of the skewed t kernel, shown side by side with the Gaussian kernel from above. This will be the mean kernel, averaged over years and plots.

```
CYIM_ST_kernel <- matrix(NA, size_dim, size_dim)
for (i in 1:size_dim) {
    mu_size <- mean(coefs[c(years, plots)]) + coefs["log(vol_t)"] * size_dum[i]
    CYIM_ST_kernel[, i] <- dST3(x = size_dum, mu = mu_size, sigma = exp(coefs["sigma_b0"] +
        coefs["sigma_b1"] * mu_size + coefs["sigma_b2"] * mu_size^2), nu = exp(coefs["nu_b0"] +
        coefs["nu_b1"] * mu_size), tau = exp(coefs["tau_b0"] + coefs["tau_b1"] *
        mu_size + coefs["tau_b2"] * mu_size^2))
}

par(mfrow = c(1, 2))
levelplot(CYIM_lmer_best_kernel, row.values = size_dum, column.values = size_dum,
    cuts = 30, col.regions = rainbow(30), xlab = "log Size t", ylab = "log Size t+1",
    main = "Gaussian, non-constant variance", panel = function(...) {
        panel.levelplot(...)
        grid.points(log(CYIM$vol_t), log(CYIM$vol_t1), pch = ".", gp = gpar(cex = 3,
            col = alpha("black", 0.5)))
    })
```



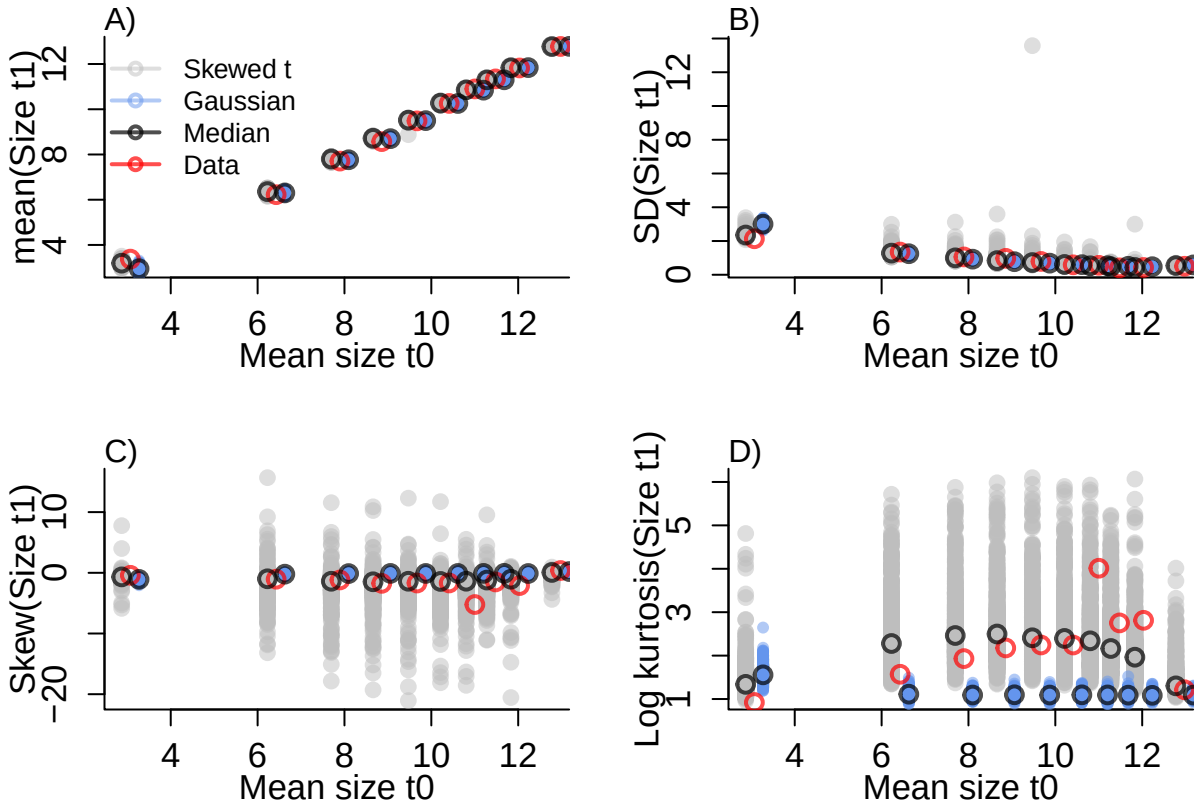

```
levelplot(CYIM_ST_kernel, row.values = size_dum, column.values = size_dum, cuts = 30,
    col.regions = rainbow(30), xlab = "log Size t", ylab = "log Size t+1", main = "Skewed t",
```

```
    panel = function(...) {
        panel.levelplot(...)
        grid.points(log(CYIM$vol_t), log(CYIM$vol_t1), pch = ".", gp = gpar(cex = 3,
            col = alpha("black", 0.5)))
    })
```



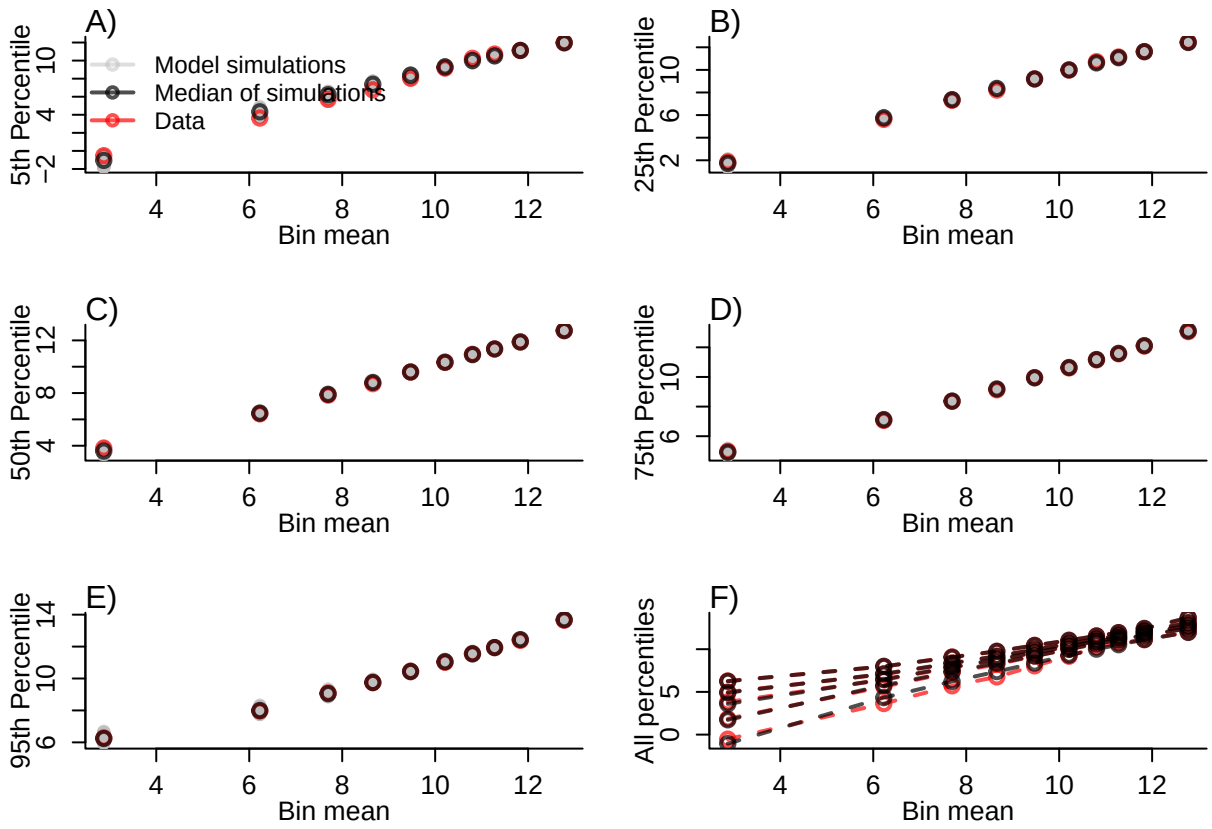
Now finer diagnostics comparing moments and quantiles of the real data against data generated by the fitted models. I am including the Gaussian and ST models for comparison. Now that I know a quadratic term for sigma improves fit, it might be fairer to included this in the Gaussian model, but proceeding with the original form of non-constant variance for now.

The skewed t model generally looks to be fitting well. Is it a strikingly better fit than the Gaussian? In mean and SD, no, which is not surprising because non-constant variance in lme4 can generally handle those. The fitted skewness of the ST is not particularly impressive, at least in this visualization, because the model can generate data with a really wide range of skewness just by chance, including positive and negative skew. Not sure yet if this is a problem, or just a natural consequence of the skewed t or skewed distributions generally. Kurtosis looks great and this is a clear 'winner' over the Gaussian.



Steve wrote this nifty function to do something similar but with quantiles. I have not added the Gaussian comparison but this might be worth doing.

```
quantileComparePlot(log(CYIM$vol_t), log(CYIM$vol_t1), cactus_sim, n_bins)
```


A)
Model simulations
Median of simulations
Data
5th Percentile
Bin mean
B)
25th Percentile
Bin mean
C)
50th Percentile
Bin mean
D)
75th Percentile
Bin mean
E)
95th Percentile
Bin mean
F)
All percentiles
Bin mean