Interpreting changes in nascent transcription composite gene profiles using a compartment model

Arun B. Dutta and Michael J. Guertin

June 6, 2021

Contents

List of Figures

1 Background

1.1 Transcription Cycle

Changes in genomic nascent RNA profiles can inform on how various treatments and stimuli regulate steps in the transcription cycle (Figure 1). The first interpretation of kinetic RNA polymerase density profiles suggested that the Estrogen Receptor (Hah *et al.*, 2011), acts prior to RNA polymerase pausing, at initiaion or recruitment. Subsequent work has defined the role of other transcription factors, including NFkB, HSF, GAF, and ZNF143 (Danko *et al.*, 2013; Jonkers *et al.*, 2014; Duarte *et al.*, 2016; Sathyan *et al.*, 2019). These studies were first limited to the study of trasncription factors that are rapidly inducible, but the development of rapidly inducible degradation methods has democratized the study of transcription factors by permitting measurements immediately after a factor is inhibited. The advantages of rapid perturbation include the ability to study essential factors and less noise, as transcriptomic measurements are not affected by the multitude post-primary effects of TF dysregulation. Our recent work modeled the effects of changing the rates of these steps.

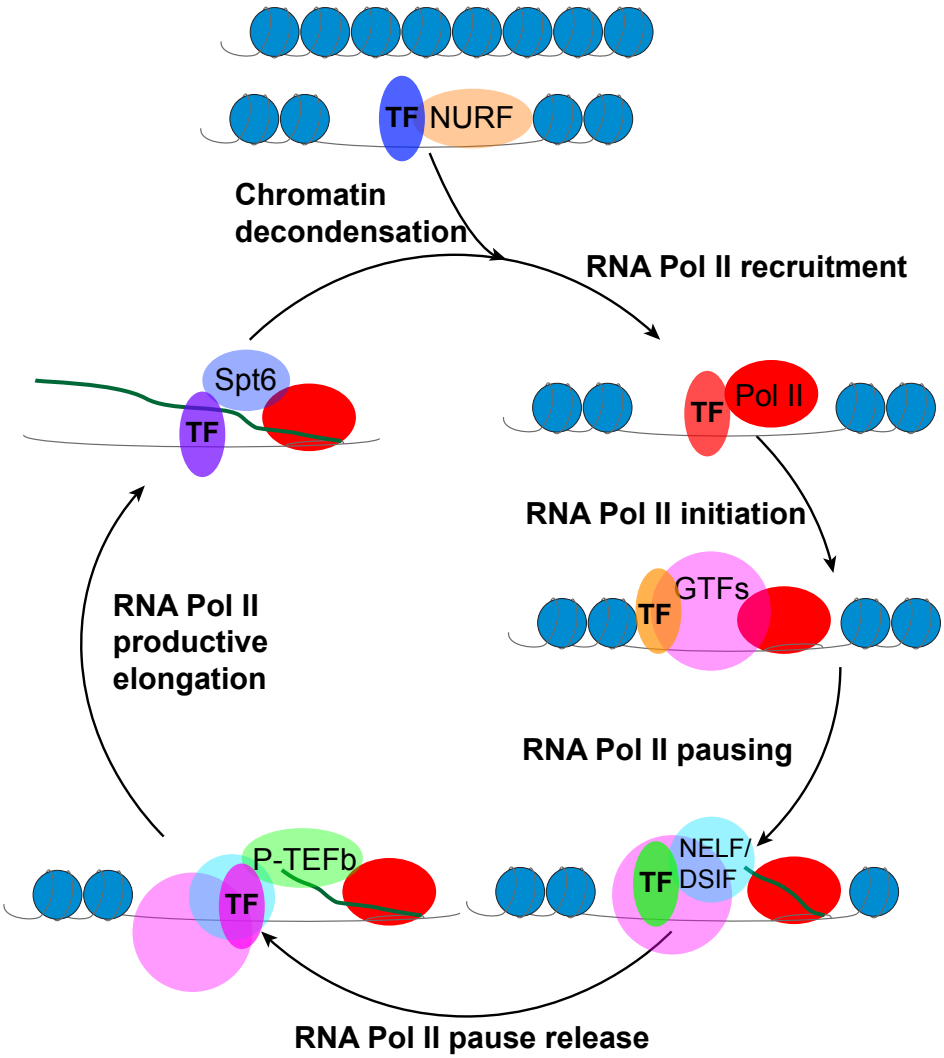


Figure 1: **Transcription Cycle.** Transcription is regulated at many steps and transcription factors tend to specialize in a subset of these steps. Specificity is occurs by selectively interacting with cofactors that are highly specialized in their function. As shown many different sequence-sepecific transcription factors confer the specificity of recruitment of various cofactors that do not harbor DNA binding domains.

1.2 Genome-wide kinetic measurements of chromatin accessibility identifies regulatory transcription factors

We induced adipognesis and measured chromatin accessibility at 0min, 20min, 40min, 60min, 120min, 180min, 240min, and 6 days (Figure 2A). We clustered the kinetic accessibility profiles by their dynamics (Figure 2B) and identified DNA elements that are specifically enriched in the increased and decreased accessibility clusters (Figure 2C). We found 12 transcription factor families and the top six are shown in Figure 2C. Note that many transcription factors bind the same motif, but we are able to narrow down the effector factors based on relative expression at time 0min and changes in factor expression over the time course. Figure 2D-F show that a single transcription factor family is associated with either increases or decreases in accessibility.

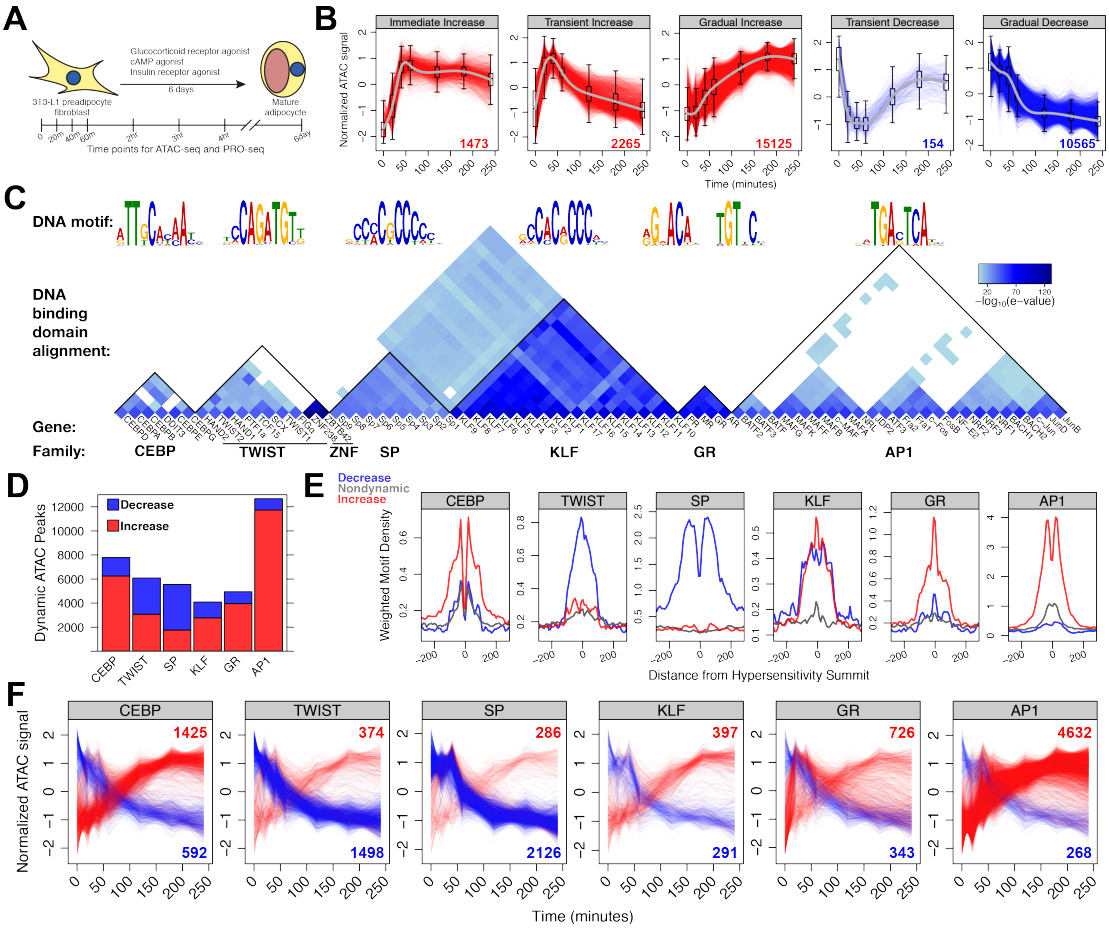


Figure 2: **CEBP, TWIST, SP, KLF, GR, and AP1 transcription factors drive chromatin accessibility dynamics in early adipogenesis.** A) Preadipocyte fibroblast 3T3-L1 cells were treated with an adipgenesis cocktail for the indicated time points: no treatment, 20min, 40min, 60min, 2hr, 3hr, 4hr, and 6day. B) Temporal classification of ATAC peaks revealed five major classes of dynamic peaks. Each red or blue line trace represents a single ATAC peak. C) *De novo* motif analysis (Bailey *et al.*, 2006) identified the six top DNA motifs that are enriched within the dynamic peaks. The transcription factors in the wedge below the seqLogo recognize the respective DNA motifs. The heatmap quantifies the local protein sequence alignment of the DNA binding domains for the genes, as determined by the Smith-Waterman algorithm. Although there are six DNA motifs, the TWIST and ZNF families of DNA binding domains recognize the same motif, despite their lack of evolutionary conservation. D) Dynamic ATAC peaks are classified by the presence of each DNA motif. The red bars represent the number of dynamic ATAC peaks within the *immediate increase*, *transient increase*, and *gradual increase* categories; the blue bars correspond to the *transient decrease* and *gradual decrease* classes. E) ATAC peaks that decrease accessibility are enriched for TWIST and SP motifs; peaks that increase accessibility are enriched for CEBP, KLF, GR, and AP1 motifs.

1.3 Genomic kinetic measurements of nascent transcription identifies regulatory transcription factors

An independent way to identify putative regulatory elements is to measure enhancer RNA using PRO-seq and dREG (Wang *et al.*, 2019). We found over 200,000 putative regulatory regions with bidirectional signatures (Figure 3A). These regulatory regions identify a subset of overlapping regulatory elements identified by ATAC-seq, but the elements identified by PRO-seq are enriched in promoters and intragenic regions (Figure 3B&C). ATAC-seq peaks are enriched for intergenic regions (Figure 3B&C). PRO-seq identifies a subset of motifs compared to ATAC, and one additional motif in the E2F family of factors (Figure 3D&E). The dREG peaks are enriched for motifs that are known to be in promoters (Benner *et al.*, 2013). Further analyses show that bidirectional PRO-seq signatures preferentially identifies promoter-proximal regulatory elements (Figure 3F&G).

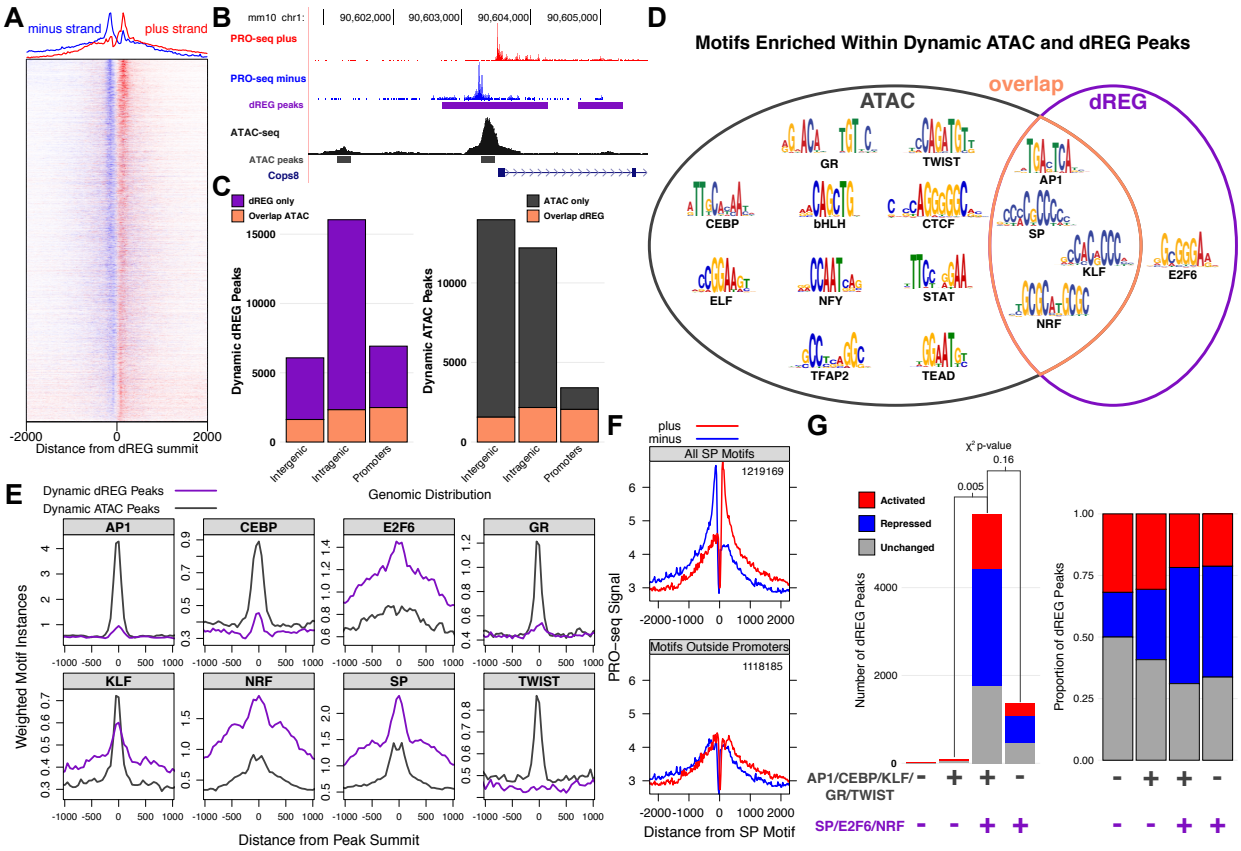


Figure 3: **SP, NRF, and E2F6 transcription factor families drive dynamic bidirectional adipogenic-induced transcription at regulatory regions within gene bodies and promoters.** A) The heatmap illustrates over 200,000 putative regulatory elements identified by a bidirectional trancription signature using discriminative regulatory-element detection (dREG) (Wang *et al.*, 2019). B) Both ATAC and PRO-seq identify a regulatory element within the promoter of Cops8. The upstream intergenic regulatory element is only identified by ATAC, while the intragenic regulatory element within the gene body is only identified by its bidirectional PRO-seq signature (visualized with UCSC browser (Kent *et al.*, 2002)). C) ATAC-seq and PRO-seq identify a distinct set of regulatory regions in the genome. D) Dynamic chromatin accessibility peaks are enriched for a more diverse set of transcription factor motifs. E) ATAC and dREG identify distinct classes of regulatory elements. There is modest CEBP motif enrichment at dREG peaks, despite the CEBP motif not being identified as enriched within dREG peaks. However, PRO-seq and dREG fail to detect a class of regulatory elements in which the dynamics are driven by TWIST and GR factors. F) SP is only associated with bidirectional transcription at promoters rather than distal regulatory elements. The average normalized PRO-seq signal for plus and minus strands around all SP motif instances (top) and all SP motifs excluding those in promoters (bottom). The number of motif instances for each plot is in the top right of the panel. G) DREG-enriched factor motifs are enriched in peaks that decrease bidirectional transcription, which suggests a link between these factors and an early and pervasive decrease in promoter initiation at genes with SP, NRF, and E2F6 motifs. Dynamic bidirectional transcription peaks found in promoters are stratified by the presence or absence of transcription factor motifs. The left plot quantifies the total number of peaks and the right plot scales to the proportion of peaks in each category. The x-axis factor motif categories are defined by the presence or absence of ATAC-enriched factors (AP1, CEBP, GR, KLF, and TWIST) and dREG-enriched factors (SP, E2F6, and NRF).

1.4 Linking transcription factors to their gene targets

Genes are also classified by their kinetics, just as ATAC peaks and enhancer RNA bidirectional peaks (Figure 4A&C). Two predictive features can be used to link regulatory elements to genes: 1) covariation in regulatory element signal and transcription; and 2) proximity. We can estimate the distance that a factor can mediate its effects on trancription by plotting the cumulative distribution of proximal dynamic regulatory elements containing the factor's cognate motif (Figure 4B&D). These data indicate that GR can act distally and SP1 factor activity is limited to proximal genes (Figure 4B&D). Activated and repressed genes tend to be proximal to regulatory elements that increase and decrease accessibility, respectively (Figure 4E). We can incorporate additional contraints on the links between regulatory elements and genes (Figure 4F). Using a set of data-driven rules based on proximity and covarying transcription and accessibility, we infer the target genes of regulatory elements. Moreover, if the regulatory element contains a transcription factor motif from Figure 3D, then we link a transcription factor to its target (Figure 4G) and (Figure 5). The simplified network for GR identifies nodes where GR binds are the predominat factor to regulate target gene expression (Figure 4G, right path).

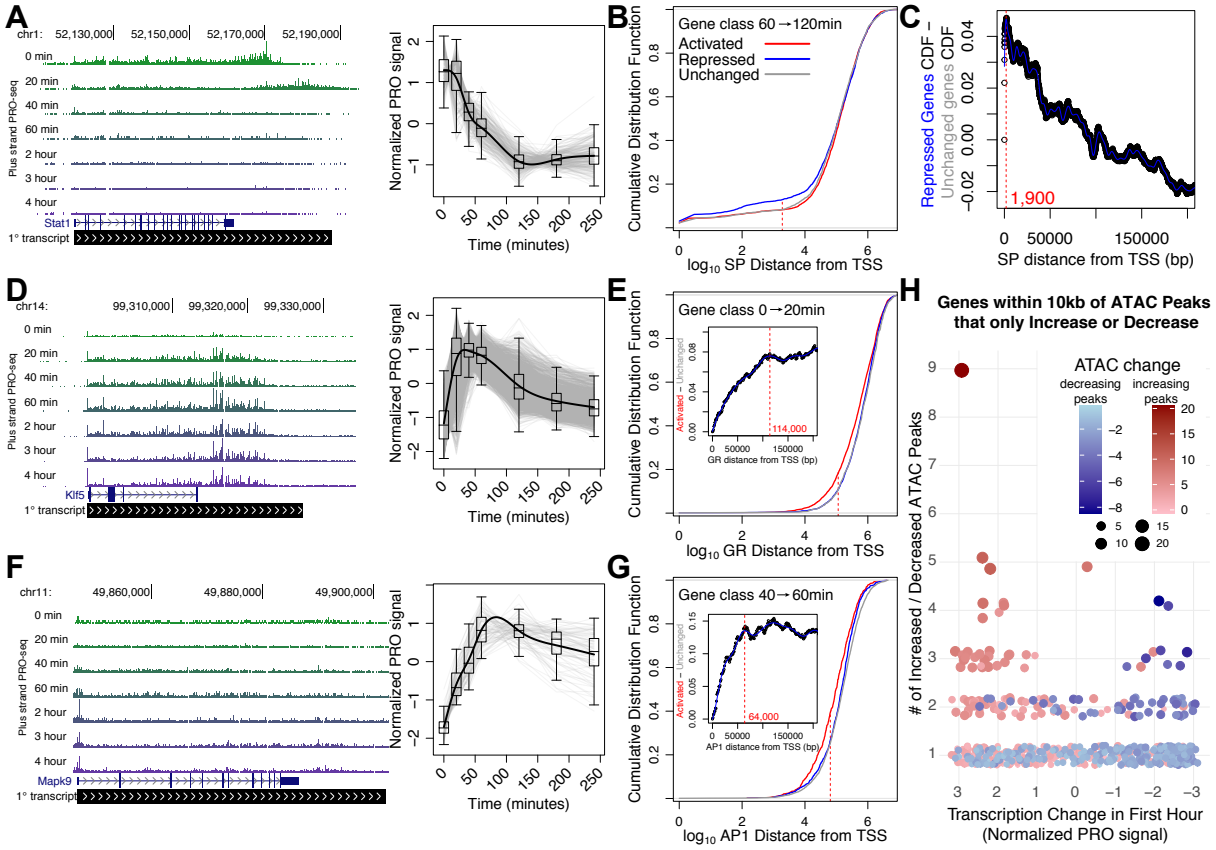


Figure 4: **Chromatin accessibility, transcription dynamics, and proximity guide inference of *cis*-links between regulatory elements and target genes.** A) *STAT1* (left) is part of a cluster of delayed repressed genes (gray traces on the right). B) Cumulative distribution plots show that ATAC peaks with SP1 motifs are closer to the 60 vs. 120 min repressed gene class than the unchanged or activated classes. C) These traces begin to converge 1,900 bases from the start sites, suggesting that the effect of SP-mediated gene repression is felt up to roughly 2kb from transcription start sites. D) *Klf5* (left) is part of a cluster of immediately and transiently activated genes (gray traces on the right). E) ATAC peaks with GR motifs are closer to the 0 vs. 20 min activated gene class than the unchanged or repressed class and these traces begin to converge 114,000 bases from the start sites. F) *Mapk9* (left) is part of a cluster of immediately and sustained activated genes (gray traces on the right). G) ATAC peaks with AP1 motifs are closer to the 40 vs. 60 min activated gene class than the unchanged or repressed class and these traces begin to converge 64,000 bases from the start sites H) Change in gene expression correlates with total accessibility change near a gene over the first hour. Note that genes within 10kb of either only increasing or decreasing ATAC peaks are included.

We can use the networks to identify high confidence target genes for transcription factors (TF). These genes are the input for composite RNA Polymerase profiles at the relevant time points. The motivation for this vignette is to use the measurements of dynamic RNA polymerase density in the pause and gene body regions to determine which step(s) in the transcription cycle a TF regulates.

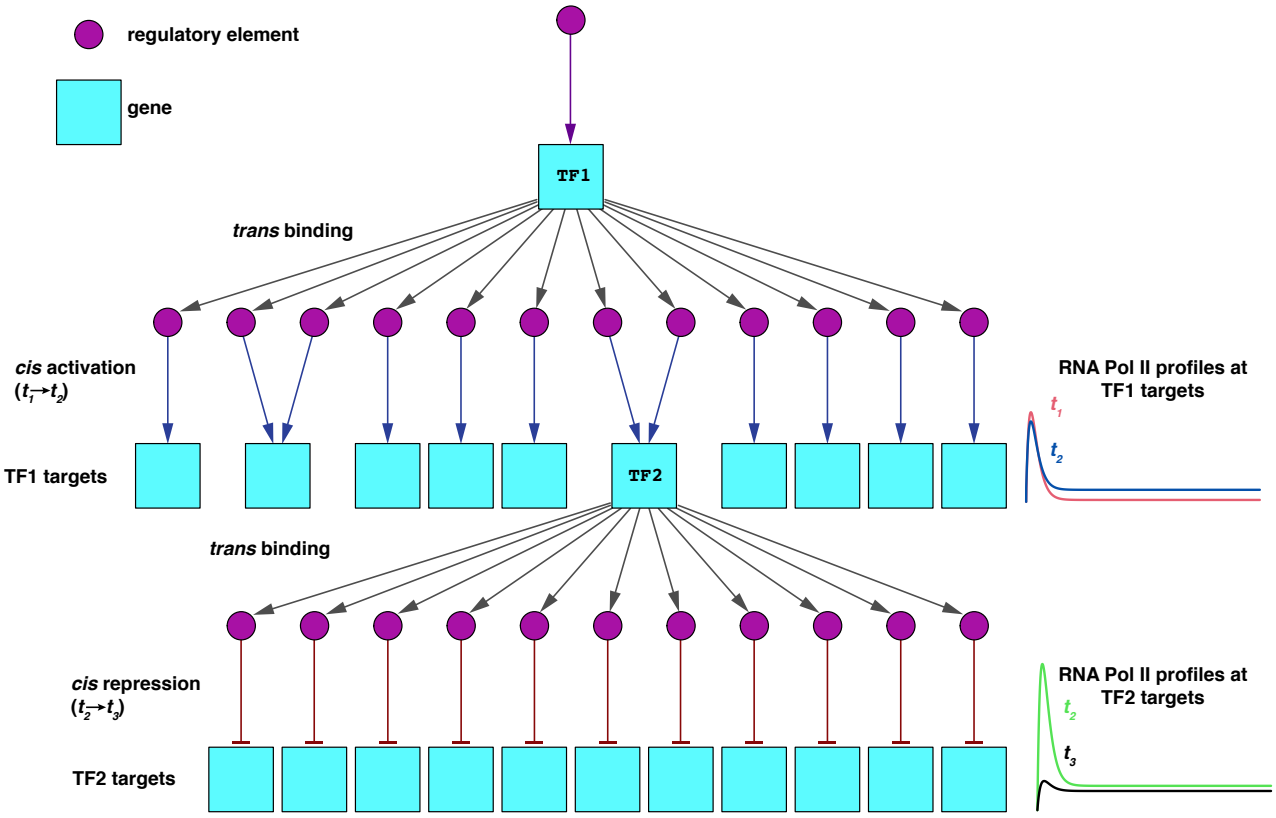


Figure 5: **Inferring TF target genes from network.** In contrast to previous studies, which activate or perturb one transcription factor at a time, the networks and data used to build the networks can be analyzed with composite profile changes and interpreted with a compartment model to inform on TF function.

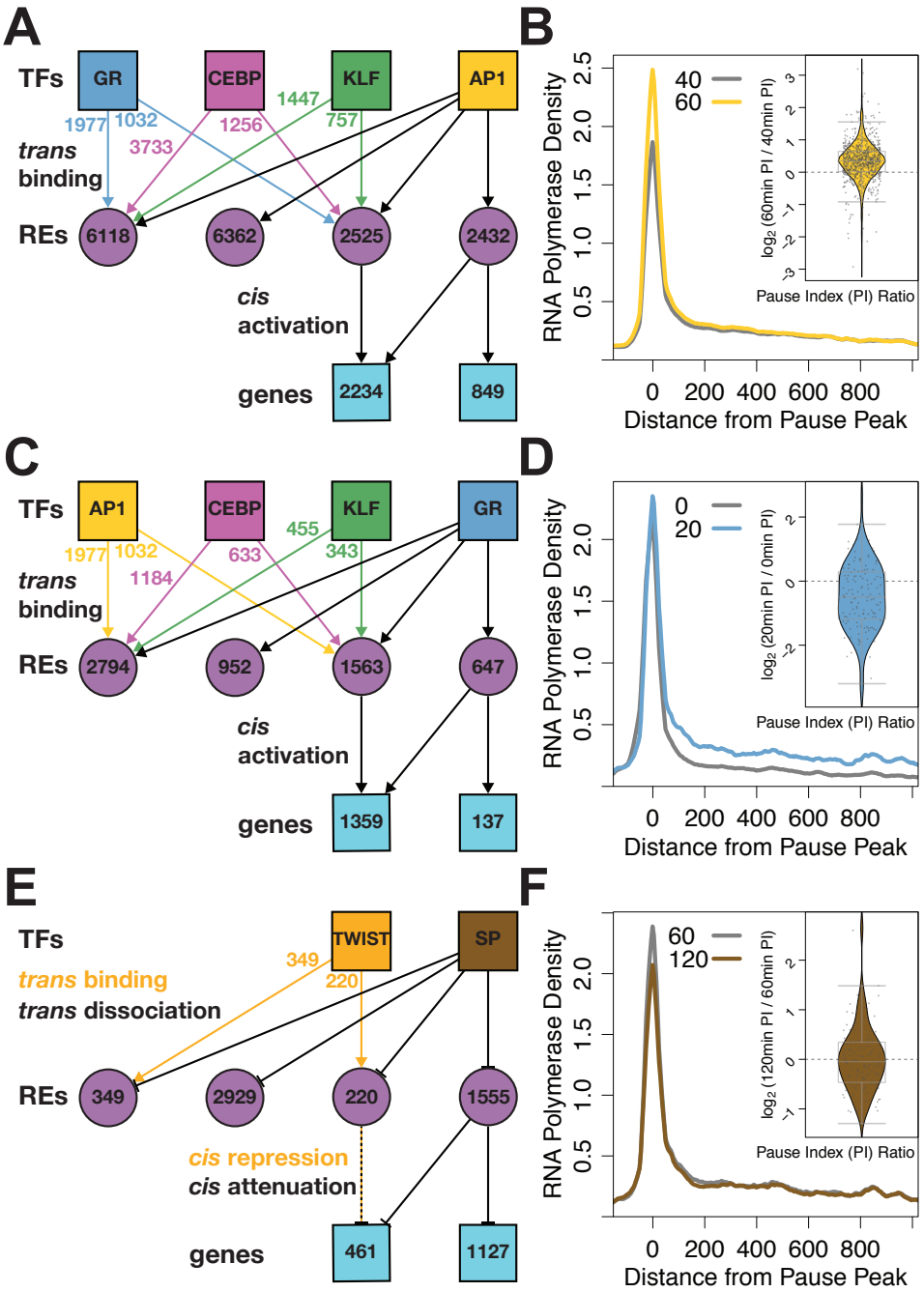


Figure 6: **Modular networks downstream of AP1, GR, and SP identify genes and transcriptional steps regulated by individual factors.** A) AP1 network. B) PRO-seq signal around pause peak center of genes regulated by only AP1 show an increase in polymerase recruitment. Pause index ratio increases slightly between 0 and 20 minutes, suggesting the increase in recruitment outweighs any increase in pause release or elongation rate. C) GR network. D) PRO-seq signal around pause peak center of genes regulated by only GR show an increase in polymerase pause release. Pause index ratio decreases between 0 and 20 minutes. E) SP network F). PRO-seq signal around pause peak center of genes regulated by only SP show an decrease in polymerase recruitment and the pause index ratio remains the same between between 60 and 120 minutes.

2 Interpreting composite profiles

Now we have a network with candidate target genes for over a dozen transcription factors. We can systematically look at how the distirbution of RNA polymerase changes at these groups of genes to determine the steps in the transcription cycle each factor likely regulate.

We previously generated a two-compartment model to describe alterations in the transcription cycle within the context of composite profiles (Figure 7) (Sathyan *et al.*, 2019). The limitations of this model and implementation are that rates are dimensionless, as we only assess how changing the rates qualitatively affected the pause and gene body densities. All considerations and previous analyses assumed steady-state.
To expand on this work the goal was to develop a dynamic model. The long term goals are to 1) incorporate rate estimates from the literature; 2) estimate the contributions of transcription factors cooperating at genes based on kinetic changes in binding (ATAC signal) and nascent transcription (PRO signal); 3) apply mechanistic rules to existing kinetic or thermodynamic models (He *et al.*, 2010; Scholes *et al.*, 2017) of transcription output to compute and predict changes in expression over developmental time courses and regulatory cascades.

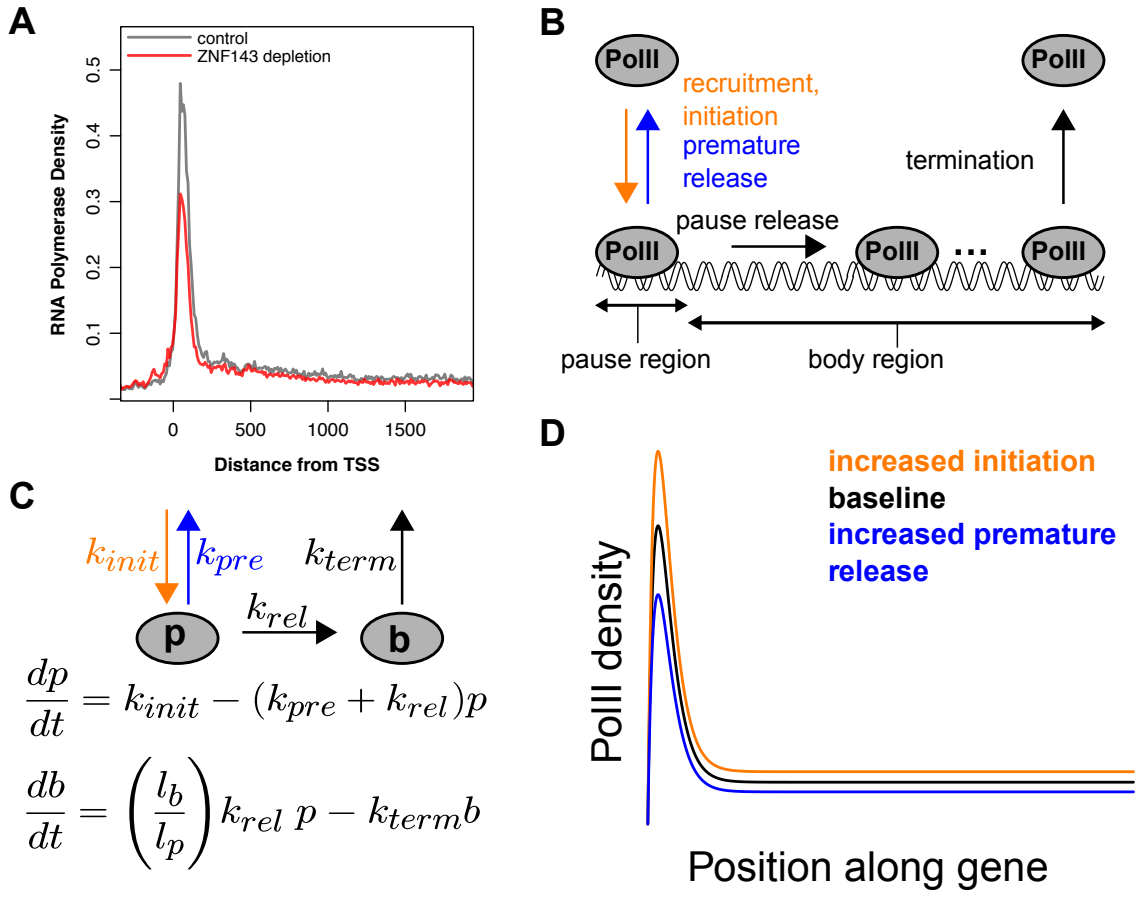


Figure 7: **ZNF143 regulates inititation or premature pause release.** This figure is reproduced from our previous work (Sathyan *et al.*, 2019). A) The composite profile of Pol II density upon ZNF143 indicates that Pol II pause density decreases. B) Model structure and key variables are highlighted in this schematic. C) A mathematical formulation of the two-compartment model, in which p refers to Pol II density at the pause region, and b refers to the density at the gene body region. D) This plot represents a steady-state simulation for a reference model (black), a model in which transcriptional initiation was increased by 25% (orange), and a model in which premature pause release was increased by 60% (blue).

2.1 Developing a dynamic model

The following code chunk contains functions that describe the compartment model and illustrate the plot to resemble conventional genomic composite gene profiles. Improving on what we published previously (Sathyan *et al.*, 2019), it is more appropriate to model the movement of polymerase from the gene body to the next position in the gene body and this should be considered the elongation rate. With a constant elongation rate over the gene, the gene body signal at any position is equal to the first body window.

The dynamics for the densities of RNA Polymerase at pause site and the gene body (Sathyan *et al.*, 2019), defined as p and b:

$$\frac{dp}{dt} = k_{init} - (k_{pre} + k_{rel})\ p$$

$$\frac{db_1}{dt} = k_{rel}\ p - k_{elong}\ b_1$$

$$\frac{db_2}{dt} = k_{elong}\ b_1 - k_{elong}\ b_2$$

$$\frac{db_x}{dt} = k_{elong}\ b_{x-1} - k_{elong}\ b_x$$

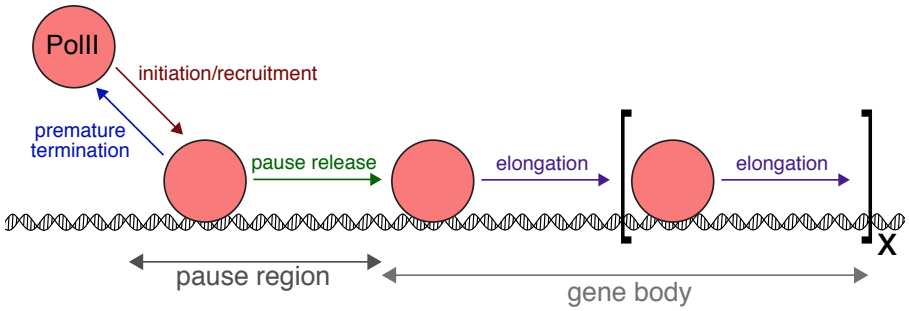
Source functions for organizing the data and plotting the composites from: `https://raw.githubusercontent.com/guertinlab/modeling_PRO_composites/main/plotting_composites_lattice.R`. The following code chunk can be retrieved directly from `https://raw.githubusercontent.com/guertinlab/modeling_PRO_composites/main/dynamic_traces.R`

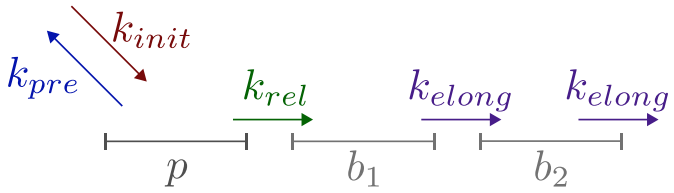
```
library(deSolve)
library(lattice)
#import in misc. plotting and data parsing functions
gitpage = 'https://raw.githubusercontent.com/guertinlab/'
source(paste0(gitpage, 'modeling_PRO_composites/main/plotting_composites_lattice.R'))

#model adapted from our G&D paper (doi:10.1101/gad.328237.119):
#Declare differential equations as function
#dP/dt = kinit - (kpre +krel)p
#dB/dt = krel * p - kelong * b
#P is the first dependent variable, promoter density; dP is its derivative wrt time
#B is the second dependent variable, body density; dB is its derivative wrt time

density.prime <- function(t, initial.state, params = params)  {
    with(as.list(c(params, initial.state)), {
        dP = kinit - (kpre + krel)*P
        dB1 = (krel)*P - kelong*B1
        dB2 = (kelong)*B1 - kelong*B2
        dB3 = (kelong)*B2 - kelong*B3
        res = c(dP, dB1, dB2, dB3)
        list(res)
    })
}


#code adapted from our G&D paper (doi:10.1101/gad.328237.119):
#to visualize it in a composite profile form
# function for finding the gene body parameter
# bpeak: desired gene body level
# pausepeak: desired pause region level
# tau: exponential decay constant
# min.pk: minimal level for gene body peak
# max.pk: maximal level for gene body peak
# dpk: resolution for the implicit solution
```





$$\frac{dp}{dt} = k_{init} - (k_{pre} + k_{rel})\, p$$

$$\frac{db_1}{dt} = k_{rel}\, p - k_{elong}\, b_1$$

$$\frac{db_2}{dt} = k_{elong}\, b_1 - k_{elong}\, b_2$$

$$\frac{db_x}{dt} = k_{elong}\, b_{x-1} - k_{elong}\, b_x$$

Figure 8: **Multi-compartment model of genic RNA Polymerase Density.** The recruitment and initiation rates of RNA Polymerase determines the rate in which RNA Polymerase molecules enter the pause region. RNA can leave the pause region by prematurely terminating and re-entering the free RNA Polymerase pool or RNA polymerase is released into the gene body. Once in the gene body, RNA polymerase travels at approximately 3kb/minute, which equates to a rate of 50 polymerase molecules per second entering each position in the gene body, if each base is considered a compartment.

```
find.body.param <- function(bpeak=NULL,tau=NULL,
                            pausepeak=NULL,min.pk=0,max.pk=1,
                            dpk=0.001) {
    pk = seq(min.pk,max.pk,dpk) # sequence of peak parameter values
    dat = matrix(0,length(pk),2) # data matrix
    colnames(dat) = c("body_param","body_assymp")
    for (x in 1:length(pk)) {
        bodypeak = pk[x]
        root = tau*(bodypeak+exp(1))*exp(-1)
        peak = (root/tau) * exp(-(root - tau)/tau) + bodypeak *
            (1 - exp(-root/tau))
        body = bodypeak * pausepeak / peak
        dat[x,1] = bodypeak
        dat[x,2] = body
    }
```

```
# look up the ratio of the body parameter over the assymptote
# use linear interpolation
    inter = findInterval(bpeak,dat[,2]) - 1
    m = (dat[(inter+1),1] - dat[inter,1]) /
        (dat[(inter+1),2] - dat[inter,2])
    b = dat[inter,1] - m * dat[inter,2]
    est = m * bpeak + b
    return(est)
}

# function to get the PRO waveform
# bpeak: desired gene body level
# pausepeak: desired pause region level
# bodypeak: body peak value to obtain a level of bpeak
# bp.seq: base pair sequence
# tau: exponential decay constant
get.pro.waveform <- function(bpeak=NULL,pausepeak=NULL,bodypeak=NULL,
                             bp.seq=NULL,tau=NULL){
    root = tau*(bodypeak+exp(1))*exp(-1)
    peak = (root/tau) * exp(-(root - tau)/tau) + bodypeak * (1 - exp(-root/tau))
    vec = sapply(bp.seq,function(x){(pausepeak/peak)*((x/tau) * exp(-(x - tau)/tau) +
                                                      bodypeak * (1 - exp(-x/tau)))})
    vec = unname(vec)
    pars = as.data.frame(rbind(cbind(max(vec),
        vec[bp.seq[length(bp.seq)]]),c(pausepeak,bpeak)))
    names(pars) = c("Peak","Assymp")
    rownames(pars) = c("simulated","requested")
    out = list(vec=vec, pars=pars)
    return(out)
}
```

2.2 Implementing the compartment model

Running through the models to simulate composite profiles and changes in transcription. Note that the rates were chosen based on the observations that RNA Polymerase pausing is rate limiting at most genes and PolII transcribes approximately 3.0kb/min. The raw code can be retrieved from `https://raw.githubusercontent.com/guertinlab/modeling_PRO_composites/main/composites_processes.R`

```
#these values are resonable starting points
kinit = 6 #initiation rate (pooled in with the unbound polII concentration) polymerases/second
krel = 2 #pause release rate (rate-limiting at most genes)  RNA polymerases/second
kpre = 1 #premature release rate (how prevalent is this--see recent Integrator literature)
kelong = 50 #elongation rate: 50 RNA polymerases/second
times = seq(0,10, by = 0.5) #time

params = c(kinit=kinit, kpre=kpre, kelong=kelong, krel=krel)
initial.state = c(P = 1, B1 = 0.01, B2 = 0, B3 = 0)

#Solve series of differential equations
result = ode(y = initial.state,
             times = times,
             func = density.prime,
             parms = params)

result = data.frame(result)

#plot in lattice
plot.changes.wrt(result, filename = 'dynamic_pro_model_PGBdensities')
```

```
#prepare the simulated composites
dynamic.pro = dynamic.pro.profile(input = result)

#plot the steady state profile as well.
#set to zero and solve
# dP = kinit - (kpre + krel)*P
# dB = (krel)*P - kelong*B
pause.peak = kinit/(kpre + krel)
body.peak = ((krel)*pause.peak)/kelong



steady.state.pro = dynamic.pro.profile(input = data.frame(cbind(1, pause.peak, body.peak)))

#plot composites in lattice
plot.pro.simulation.composites(dynamic.pro,
             filename = 'dynamic_pro_model_density')

plot.pro.simulation.composites(steady.state.pro ,
             filename = 'steady_state_pro_model_density')
```

2.3 Parameterization and sensitivity analysis

We will follow the workflow outlined here: `https://cran.r-project.org/web/packages/pksensi/vignettes/pbtk1cpt.html`

We simplified the differential equation function to include only a single gene body density, since b_1 is equal to b_n at late time points or steady state. We have reasonable measurements for elongation rates of 50 bases per second (Ardehali and Lis, 2009). The initial rates for initiation, pause release, and non-productive release will vary between 0.01 and 1 RNA Polymerases/second entering or exiting a compartment. I mention later that I fixed the elongation rate and empirically determined the other rates that result in the empirically determine pause and gene body densities. I assumed that k_{init}, k_{rel}, and k_{pre} were all within two orders of magnitude of one another and sampled this parameter space.

The greatest struggle that we are having is determining what consitutes pause density. Gene body density is easy because we are considering each position along the gene body a compartment and an average density (Polymerases/base) over the entire gene body is a good approximation of the number of RNA polymerase molecules at any one position. However, the same is not true of the pause region. We determine the pause region empirically for each expressed gene and the density varies greatly over this 50 base region. Some genes exhibit prominent pause sites and some have multiple discontinuous peaks of pausing observed. Therefore, we are considering the entire pause region as a compartment and using the sum of intensity of a 50 base region centered on the pause summit as the pause density.

```
library(deSolve)
library(pksensi)

#this is a simplified version, since the gene body densities converge over time
density.one.body <- function(t, initial.state, params = params)  {
    with(as.list(c(params, initial.state)), {
        dP = kinit - (kpre + krel)*P
        dB1 = (krel)*P - kelong*B1
        res = c(dP, dB1)
        list(res)
    })
}

#make them all the same and vary by 10x,
#with a max being 50x slower rate than elongation rate
kinit = 0.1
krel = 0.1
kpre = 0.1 #is it worth excluding this because we cannot distinguish between
#contrasting effects of initiation and premature nonproductive release?
kelong = 50

parms = c(kinit, krel, kpre, kelong)
initState = c(P = 1, B1 = 0.01)

t <- seq(from = 0.01, to = 100.01, by = 5)#by =10 #redo overnight by 5
y <- ode(y = initState, times = t, func = density.one.body, parms = parms)

params <- c("kinit", "krel", "kpre", "kelong")

q <- c("qunif", "qunif", "qunif", "qunif")
q.arg <- list(list(min = kinit / 10, max = kinit * 10),
              list(min = krel / 10, max = krel * 10),
              list(min = kpre / 10, max = kpre * 10),
              list(min = kelong/2, max = kelong*2))


set.seed(1234)

x <- rfast99(params, n = 20000, q = q, q.arg = q.arg, replicate = 100)
```

```
pdf('parameter_space_distribution.pdf', width=8, height=8, useDingbats=FALSE)
par(mfrow=c(4,4),mar=c(0.8,0.8,0.8,0),oma=c(4,4,2,1), pch =16)
for (j in c("kinit", "kpre", "kelong", "krel")) {
  if ( j == "krel") {
    plot(x$a[,1,j], ylab = "krel", cex = 0.3)
  } else plot(x$a[,1,j], xaxt="n", cex = 0.3, ylab = "")
  for (i in 2:3) {
  if ( j == "krel") {
    plot(x$a[,i,j], ylab = "", yaxt="n", cex = 0.3)
    } else plot(x$a[,i,j], xaxt="n", yaxt="n", cex = 0.3, ylab = "")
  }
  hist <- hist(x$a[,,j], plot=FALSE,
               breaks=seq(from=min(x$a[,,j]), to=max(x$a[,,j]), length.out=20))
  barplot(hist$density, axes=FALSE, space=0, horiz = T, main = j)
}
mtext("Model evaluation", SOUTH<-1, line=2, outer=TRUE)
dev.off()

outputs <- c("P", "B1")
out <- solve_fun(x, time = t, func = density.one.body,
                 initState = initState, outnames = outputs)
```

The sensitivity index has a range from 0 - 1. A value of 0 means no impact and 1 is high impact. The index quantifies the contribution each covariate as a percentage of output variance using the parameter distributions from Figure 9. The total-effect index measures the contribution of the variable to the output variance. This value includes all variance caused by a variable's interactions, of any order, with any other input variable. The total-effect index is the solid black line with 95% confidence interval as a tranparent polygon. The first-order sensitivity index is the contribution to the output variance of the main effect of the variable. This index measures the effect of varying the variable alone, but averaged over variations in other input parameters. First-order sensitivity is standardized by the total variance to provide a fractional contribution and shown as a red solid line with a 95% confidence interval.

```
pdf('pause_density_parameter_sensitivity_index.pdf', width=8, height=8, useDingbats=FALSE)
par(pty="s")
par(mfrow=c(4,1))
plot(out, vars = "P")
dev.off()

pdf('body_density_parameter_sensitivity_index.pdf', width=8, height=8, useDingbats=FALSE)
par(pty="s")
plot(out, vars = "B1")
dev.off()
```

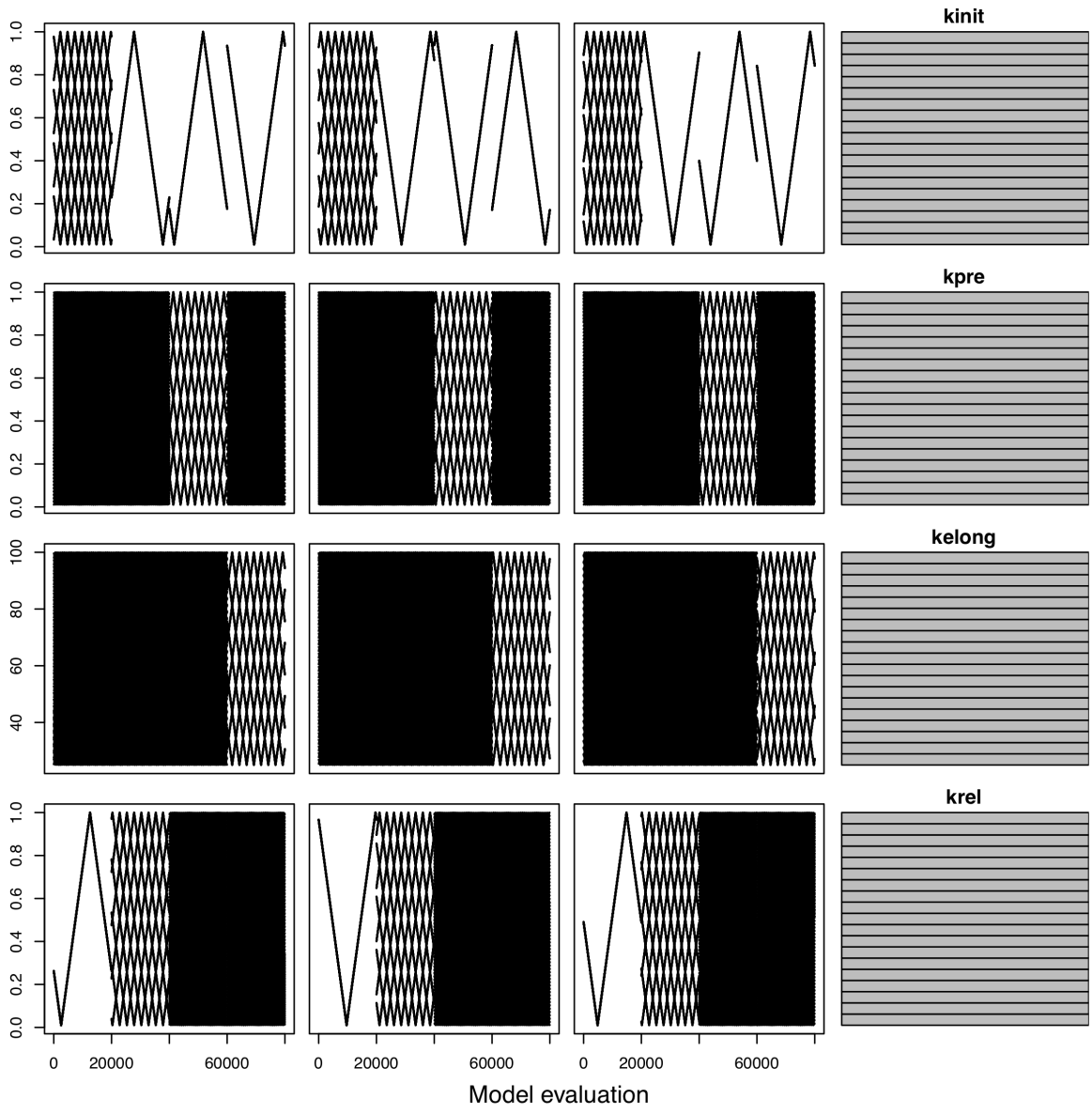


Figure 9: **A large set of parameters were used as input to empirically determine values that recapitulate pause and gene body density ratios** A uniform distribution of parameters with a range corresponding to 10% and 10-fold the initial values: 0.1 for *kinit*, *krel*, and *kpre* and 50 for *kelong*.

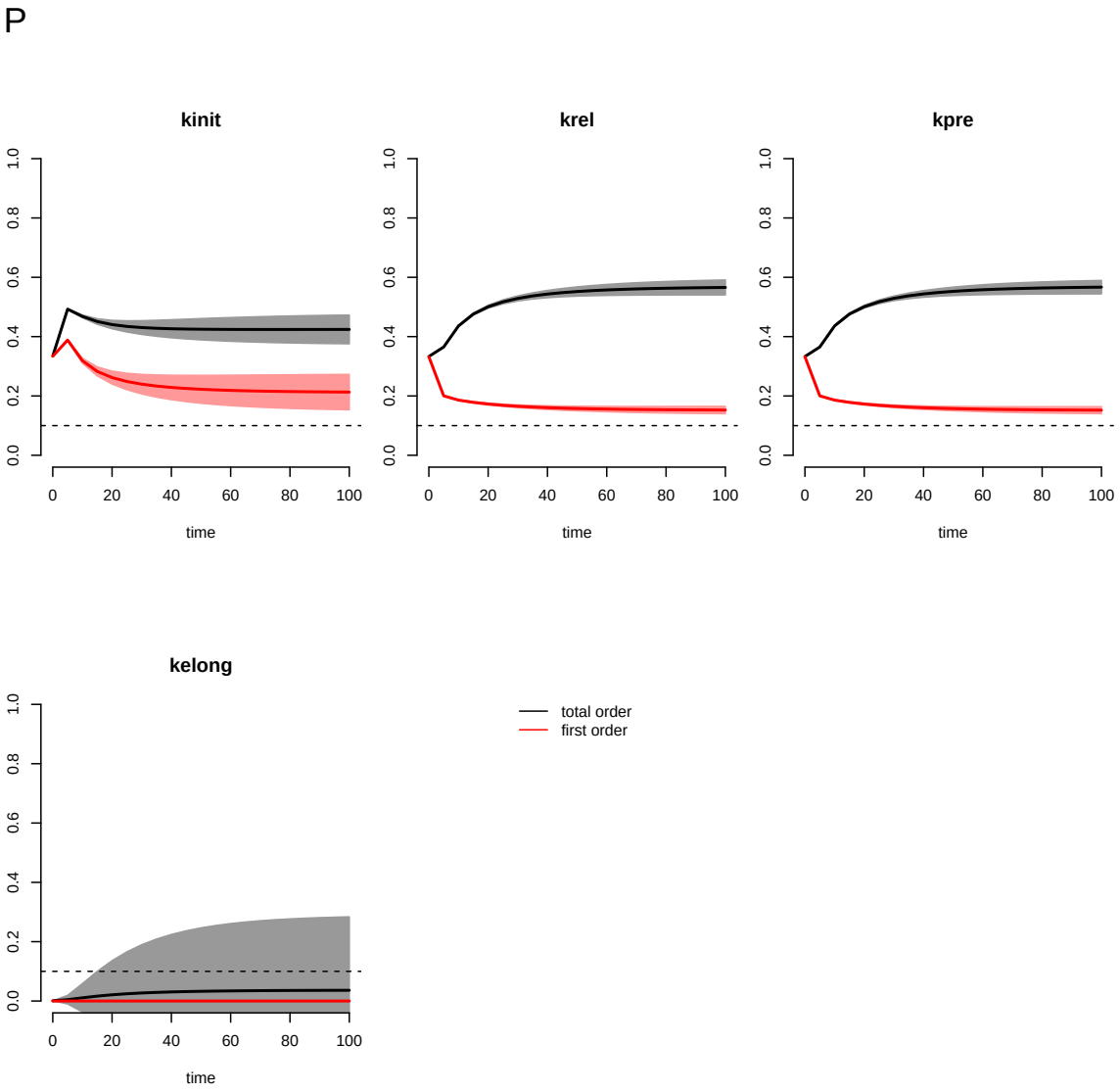


Figure 10: **Sensitivity index for pause density.** The sensitivity index indicates that pause density is minimally sensitive to elongation rate.

B1

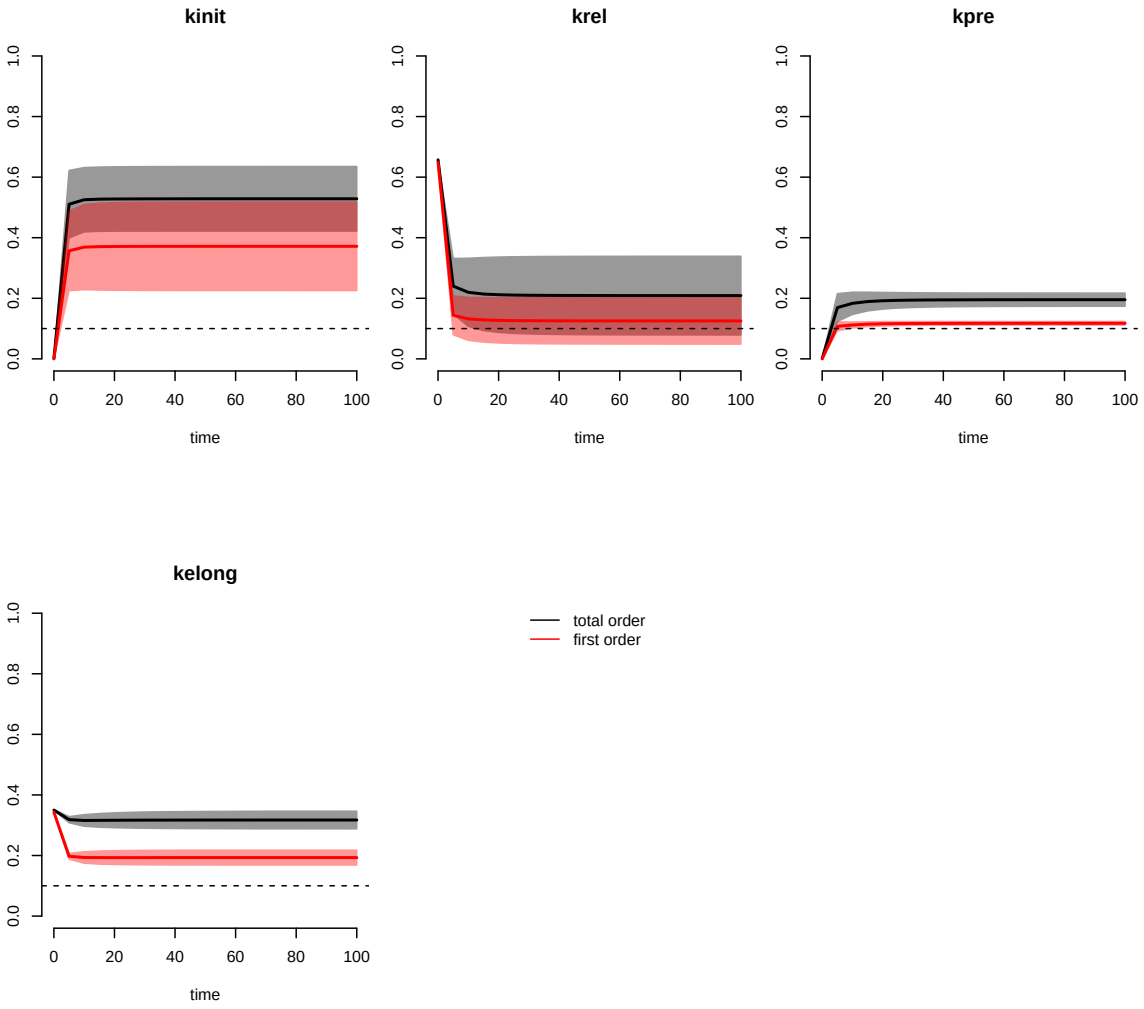


Figure 11: **Sensitivity index for gene body index.** The gene body density is sensitive to changes of all parameters.

```
#body
pdf('body_density_parameter_sensitivity_steady_state.pdf', width=8, height=3, useDingbats=FALSE)
par(mfcol=c(1,4),mar=c(0.8,2.2,0,0),oma=c(4,4,2,1), pch =16, pty="s")
for (j in c("kinit", "kpre", "kelong", "krel")){
    plot(x$a[,1,j], out$y[,1, "100.01","B1"], cex = 0.3, col = '#8c8c8c33',
         main = paste0("\n", j))
}
mtext("Parameter Value", SOUTH<-1, line=0, outer=TRUE)
mtext("Gene Body Density", WEST<-2, line=0, outer=TRUE)
dev.off()


#pause
pdf('pause_density_parameter_sensitivity_steady_state.pdf', width=8, height=3, useDingbats=FALSE)
par(mfcol=c(1,4),mar=c(0.8,2.2,0,0),oma=c(4,4,2,1), pch =16, pty="s")
for (j in c("kinit", "kpre", "kelong", "krel")){
    plot(x$a[,1,j], out$y[,1, "100.01","P"], cex = 0.3, col = '#8c8c8c33',
         main = paste0("\n", j))
}
mtext("Parameter Value", SOUTH<-1, line=0, outer=TRUE)
mtext("Pause Density", WEST<-2, line=0, outer=TRUE)
dev.off()
```

We can observe the direct effect of changing parameters on the pause and gene body densities and extract general priciples regarding the relationships between the parameter and model output. As an example, these plots were used in early iterations to determine a range of k_{init}, k_{rel}, and k_{pre} values that output empirically determined ratios of pause to gene body densities. We kept the value of k_{elong} at 50 RNA polymerases/second, since this is the most well studied rate.

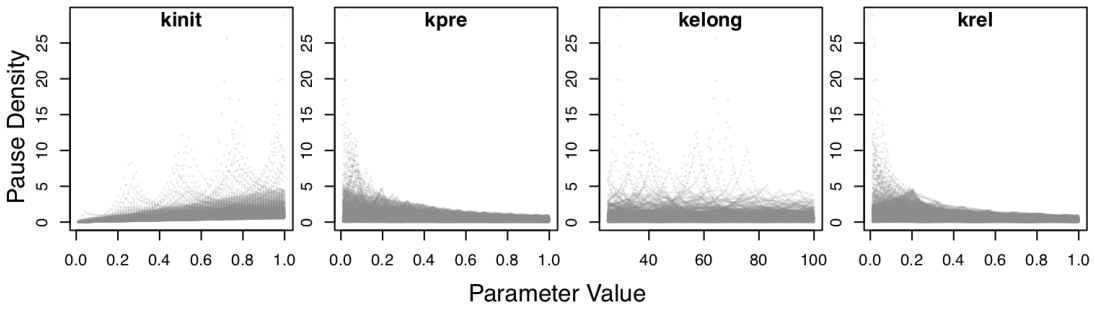


Figure 12: **Direct effect of parameter setting on pause density.**

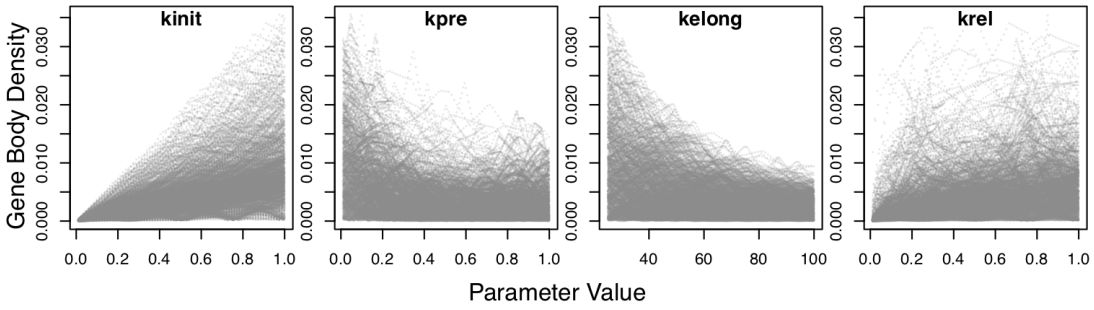


Figure 13: **Direct effect of parameter setting on gene body density.**

2.4 Determine how changing two parameters affects pause and gene body densities.

Take the comments in the code and make them into a paragraph here.

```
#from Arun
load('~/Desktop/Figure4/mike.pause.region.summation.Rdata')

#these comparisons are all GR 0 - 20 minutes
#average ratio of pause compartment signal to gene body (single base position) signal.
pre.ratio = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'GR' &
                                mike.pause.region.summation$time == 0,]$pause_sum, trim = 0.05) /
     mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'GR' &
                                mike.pause.region.summation$time == 0,]$body_avg, trim = 0.05)

#average fold change in pause signal from 0 to 20 minutes
pause.change = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'GR' &
                                mike.pause.region.summation$time == 20,]$pause_sum, trim = 0.05) /
     mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'GR' &
                                mike.pause.region.summation$time == 0,]$pause_sum, trim = 0.05)
#pause.change = mean(pause.change[is.finite(pause.change)], trim = 0.05)



#average fold change in gene body signal from 0 to 20 minutes
body.change = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'GR' &
                                mike.pause.region.summation$time == 20,]$body_avg, trim = 0.05) /
     mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'GR' &
                                mike.pause.region.summation$time == 0,]$body_avg, trim = 0.05)
#body.change = mean(body.change[is.finite(body.change)], trim = 0.05)

print('Average ratio of pause compartment signal to gene body (single base position) signal:')
print(pre.ratio)
print('Average fold change in pause signal from 0 to 20 minutes:')
print(pause.change)
print('Average fold change in gene body signal from 0 to 20 minutes:')
print(body.change)


# the next step is to determine a set of parameters that fit the observed ratio of
# pause compartment signal to gene body (single base position) signal. Although any
# threshold can be used, we dictated that the ratio had to be within two pecent of the target

#two percent is hard coded
perc = 0.02

# the following vectors are the parameters that result in the 0 min pause/gene body signal ratio
# and the pause and gene body densities.
# The rates are considered RNA Polymerase molecules per second.
# The pause and gene body signal are considered number of RNA Polymerase molecules in the compartment.

kinit.vec =  x$a[,1,"kinit"][(( out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) > (pre.ratio - (pre.ratio*perc))) &
                ((out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) < (pre.ratio + (pre.ratio*perc)))]

krel.vec = x$a[,1,"krel"][(( out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) > (pre.ratio - (pre.ratio*perc))) &
                ((out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) < (pre.ratio + (pre.ratio*perc)))]

kelong.vec = x$a[,1,"kelong"][(( out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) > (pre.ratio - (pre.ratio*perc))) &
                ((out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) < (pre.ratio + (pre.ratio*perc)))]

kpre.vec = x$a[,1,"kpre"][(( out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) > (pre.ratio - (pre.ratio*perc))) &
                ((out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) < (pre.ratio + (pre.ratio*perc)))]

out.pause = out$y[,1,10.1,"P"][(( out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) > (pre.ratio - (pre.ratio*perc))) &
                ((out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) < (pre.ratio + (pre.ratio*perc)))]

out.body = out$y[,1,10.1,"B1"][(( out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) > (pre.ratio - (pre.ratio*perc))) &
                               ((out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) < (pre.ratio + (pre.ratio*perc)))]
```

There cannot be more than two RNA Polymerase molecules in the 50 base pause region. We will explore parameter settings with more and less than 2 molecules in the pause region and filter based upon the theoretical number of RNA Polymerase molecules in the pause region output by the model.

```
parameter.explore = data.frame(rbind(cbind(kinit.vec, 'Kinit', out.pause, out.body),
                               cbind(krel.vec, 'Krel', out.pause, out.body),
                               cbind(kelong.vec, 'Kelong', out.pause, out.body),
                               cbind(kpre.vec, 'Kpre',out.pause, out.body)),
                               stringsAsFactors = FALSE)

parameter.explore[,1] = as.numeric(parameter.explore[,1])
parameter.explore[,3] = as.numeric(parameter.explore[,3])
parameter.explore[,4] = as.numeric(parameter.explore[,4])
```

```
parameter.explore[,5] = '2 or less'
parameter.explore[,5][parameter.explore[,3] > 2] = 'more than 2'

colnames(parameter.explore) = c('value', 'rate', 'pause', 'body', 'category')


pdf('parameters_with_feasible_pause_densities.pdf', useDingbats = FALSE, width=8.5, height=4.4)

trellis.par.set(box.umbrella = list(lty = 1, col="#93939380", lwd=2),
                box.rectangle = list(col = '#93939380', lwd=1.6),
                plot.symbol = list(col='#93939380', lwd=1.6, pch ='.'))

print(bwplot(value ~ category | rate, data = parameter.explore,
             between=list(y=1.0, x = 1.0),
             scales=list(x=list(draw=TRUE, rot = 45), #y = list(axs = 'i'),
                         relation="free",
                         alternating=c(1,1,1,1),cex=1,font=1),
             xlab = 'Polymerase Constraint Category',
             ylab = "RNA polymerases / second",
             horizontal =FALSE,  col= 'black',
             aspect = 1,
             #ylim = c(0, 3),
             par.settings=list(par.xlab.text=list(cex=1.2,font=1),
                               par.ylab.text=list(cex=1.2,font=1),
                               par.main.text=list(cex=1.2, font=1),
                               plot.symbol = list(col='black', lwd=1.6, pch =19, cex = 0.25)),
             strip = function(..., which.panel, bg) {
               bg.col = c("grey85")#c("#ce228e" ,"grey60", "#2290cf","grey90")
               strip.default(..., which.panel = which.panel,
                             bg = rep(bg.col, length = which.panel)[which.panel])
             },
             panel = function(..., box.ratio, col) {
                 #panel.abline(h = 0, col = 'grey45', lty = 2)
                 #panel.violin(..., col = '#0000ff',
                 #             varwidth = FALSE, box.ratio = box.ratio, outer = FALSE)
                 panel.stripplot(..., col='#54545380', do.out=FALSE, jitter.data=TRUE,
                                 amount = 0.2, pch = 16)
                 panel.bwplot(..., pch = '|', do.out = TRUE)

       }))
dev.off()
```

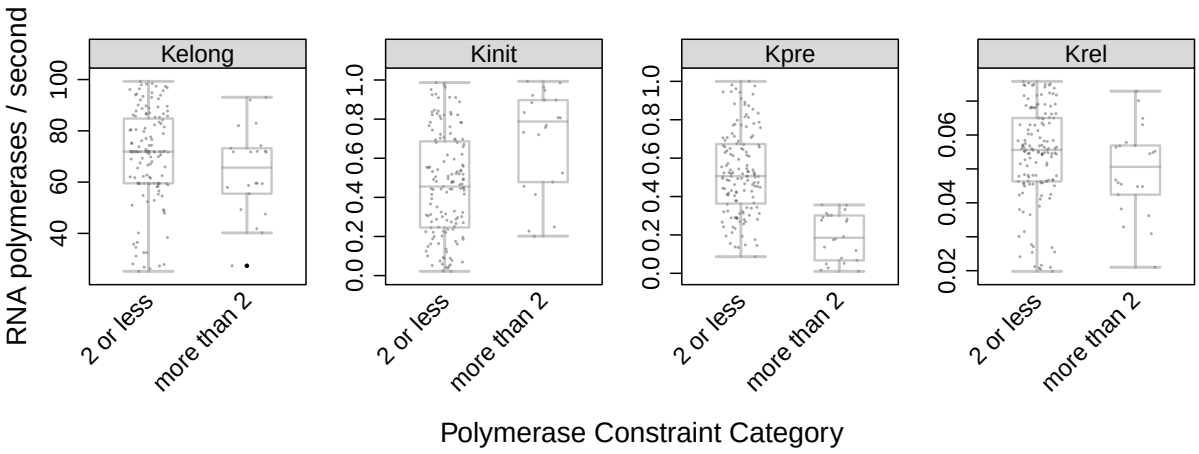


Figure 14: **Parameters that output impossible numbers of RNA Polymerase molecules in the pause region.** The k_{init} and k_{pre} vary over a wide range. When the model outputs more than two RNA polymerase molecules in the 50 base pause region k_{init} tends to be high and k_{pre} tends to be low. k_{rel} is always comparably low; note the y-axis range. k_{rel} represents the only other rate that specifies entry or exit from the pause region, so we hypothesize that the ratio of k_{init} to k_{pre} is what is driving unrealistic RNA Polymerase occupancy.

Figure 14 shows that model outputs that result in more than two RNA polymerase molecules in the 50 base pause region have high k_{init} and low k_{pre} values. The following code chunk addresses whether the ratio of k_{init} to k_{pre} is driving unrealistic RNA Polymerase occupancy.

```
more.2.kinit = parameter.explore[parameter.explore[,5] == 'more than 2' &
                                 parameter.explore$rate == "Kinit",]$value
more.2.kpre = parameter.explore[parameter.explore[,5] == 'more than 2' &
                                parameter.explore$rate == "Kpre",]$value

less.2.kinit = parameter.explore[parameter.explore[,5] == '2 or less' &
                                 parameter.explore$rate == "Kinit",]$value
less.2.kpre = parameter.explore[parameter.explore[,5] == '2 or less' &
                                parameter.explore$rate == "Kpre",]$value


kinit.kpre.relationship = data.frame(rbind(cbind(more.2.kinit, more.2.kpre, 'more than 2'),
                                cbind(less.2.kinit, less.2.kpre, '2 or less')),
                                stringsAsFactors = FALSE)
kinit.kpre.relationship[,4] = as.numeric(kinit.kpre.relationship[,1]) /
    as.numeric(kinit.kpre.relationship[,2])

colnames(kinit.kpre.relationship) = c('Kinit', 'Kpre', 'category', 'ratio')


pdf('kinit_kpre_relationship.pdf', useDingbats = FALSE, width=3.5, height=3.5)

trellis.par.set(box.umbrella = list(lty = 1, col="#93939380", lwd=2),
                box.rectangle = list(col = '#93939380', lwd=1.6),
                plot.symbol = list(col='#93939380', lwd=1.6, pch ='.'))

print(bwplot(ratio ~ category, data = kinit.kpre.relationship,
             between=list(y=1.0, x = 1.0),
             scales=list(x=list(draw=TRUE, rot = 45), #y = list(axs = 'i'),
                         relation="free",
                         alternating=c(1,1,1,1),cex=1,font=1),
             xlab = 'Polymerase Category',
             ylab = "Kinit / Kpre",
             horizontal =FALSE,  col= 'black',
             aspect = 1,
             #ylim = c(0, 3),
             par.settings=list(par.xlab.text=list(cex=1.2,font=1),
                               par.ylab.text=list(cex=1.2,font=1),
                               par.main.text=list(cex=1.2, font=1),
                               plot.symbol = list(col='black', lwd=1.6, pch =19, cex = 0.25)),
             strip = function(..., which.panel, bg) {
               bg.col = c("grey85")#c("#ce228e" ,"grey60", "#2290cf","grey90")
               strip.default(..., which.panel = which.panel,
                             bg = rep(bg.col, length = which.panel)[which.panel])
             },
             panel = function(..., box.ratio, col) {
                 #panel.abline(h = 2.24, col = 'red', lty = 2)
                 #panel.violin(..., col = '#0000ff',
                 #             varwidth = FALSE, box.ratio = box.ratio, outer = FALSE)
                 panel.stripplot(..., col='#54545380', do.out=FALSE, jitter.data=TRUE,
                                 amount = 0.2, pch = 16)
                 panel.bwplot(..., pch = '|', do.out = FALSE)

       }))
dev.off()
```

The ratio of k_{init} to k_{pre} is the dominant feature that allows for unrealistic RNA Polymerase densities (i.e. more than 2 molecules in a 5 base region). A more complicated model is needed to account for saturation in the pause region. This study will exclusively focus on k_{init} and k_{pre} values that permit as many as two RNA Polymerases in the region. The set up has the effect of artificially inflating the k_{pre} rate if initiation rate allows more than two RNA Polymerase molecules into the region, however biologically we expect that the the k_{pre} rate to be unchanged and the saturation of the pause region limits loading of another RNA Polymerase molecule.

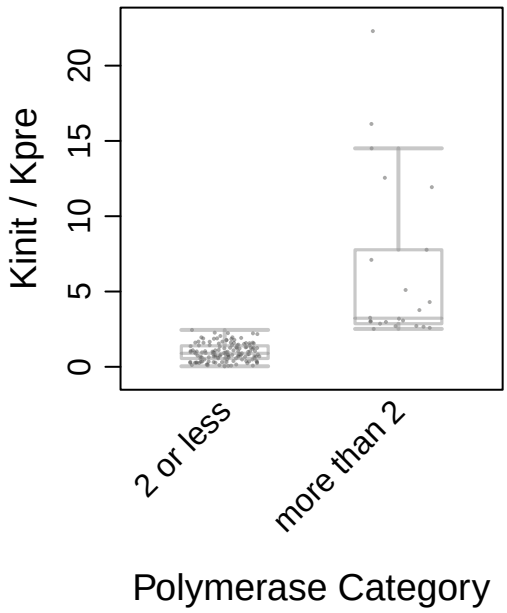


Figure 15: **A high ratio of initiation rate to non-productive pause release rate results in impossibly high RNA Polymerase densities.** The k_{init} and k_{pre} vary over a wide range. When the model outputs more than two RNA polymerase molecules in the 50 base pause region k_{init} tends to be high and k_{pre} tends to be low. k_{rel} is always comparably low; note the y-axis range. k_{rel} represents the only other rate that specifies entry or exit from the pause region, so we hypothesize that the ratio of k_{init} to k_{pre} is what is driving unrealistic RNA Polymerase occupancy.

Next, we will keep two parameters constant and determine how the other rates can change to reproduce the changes observed at later time points.

```
change.two.parameters <- function(func.density = density.prime.kinit.krel,
                                  p.change = pause.change,
                                  b.change = body.change,
                                  target.pause = out.pause,
                                  target.body = out.body,
                                  param1 = kinit.vec,
                                  param2 = krel.vec,
                                  params = c("kinit", "krel"),
                                  constant.param1.vec = kpre.vec,
                                  constant.param2.vec = kelong.vec,
                                  perc = 0.05, fold.query = 5, sample.size = 10000, ... ) {
    list.out.param1.values = list()
    list.out.param2.values = list()
for (i in 1:length(param1)) {
        parms = c(param1[i], param2[i])
        initState = c(P = 1, B1 = 0.01)
        t <- seq(from = 0.01, to = 100.01, by = 100)
        y <- ode(y = initState, times = t, func = func.density, parms = parms,
                 constant.param1 = constant.param1.vec[i],  constant.param2 = constant.param2.vec[i])
        q <- c("qunif", "qunif")
#allow rates to change by five fold in both directions (make fold change an option)
        q.arg <- list(list(min = parms[1] / fold.query, max = parms[1]*fold.query),
                      list(min = parms[2]/fold.query, max = parms[2] * fold.query))
        set.seed(1234)
        x <- rfast99(params, n = sample.size, q = q, q.arg = q.arg, replicate = 5)

        outputs <- c("P", "B1")
        out <- solve_fun(x, time = t, func = func.density,
                         constant.param1 = constant.param1.vec[i], constant.param2 = constant.param2.vec[i],
                         initState = initState, outnames = outputs)
# what parameter values fit these densities?
# allow the target values to differ by variable perc (default of 5%)
new.param1 = x$a[,1,params[1]][out$y[,1, "100.01","B1"] > (target.body[i] * b.change) - ((target.body[i] * b.change)* perc) &
            out$y[,1, "100.01","B1"] < (target.body[i] * b.change) + ((target.body[i] * b.change)*perc) &
            out$y[,1, "100.01","P"] > (target.pause[i] * p.change) - ((target.pause[i] * p.change)*perc) &
            out$y[,1, "100.01","P"] < (target.pause[i] * p.change) + ((target.pause[i] * p.change)*perc)]

new.param2 = x$a[,1,params[2]][out$y[,1, "100.01","B1"] > (target.body[i] * b.change) - ((target.body[i] * b.change)*perc) &
            out$y[,1, "100.01","B1"] < (target.body[i] * b.change) + ((target.body[i] * b.change)*perc) &
            out$y[,1, "100.01","P"] > (target.pause[i] * p.change) - ((target.pause[i] * p.change)*perc) &
            out$y[,1, "100.01","P"] < (target.pause[i] * p.change) + ((target.pause[i] * p.change)*perc)]
# in all cases the fold change in rate of pause release is
# greater than the change in initiation
        list.out.param1.values[[i]] = new.param1
        list.out.param2.values[[i]] = new.param2
        print(target.pause[i])
        print(target.body[i])
        print(params[1])
        print(param1[i])
        print(new.param1)
        print(params[2])
        print(param2[i])
        print(new.param2)
    }
    return(list(list.out.param1.values, list.out.param2.values))
}
```

Implement the constraint of no more than 2 RNA polmerases in the pause region at steady state. Change two variables systematically.

```
#no more than pause region density of 2 polymerase molecules
kinit.vec = kinit.vec[out.pause <= 2]
krel.vec = krel.vec[out.pause <= 2]
kelong.vec = kelong.vec[out.pause <= 2]
kpre.vec = kpre.vec[out.pause <= 2]
out.body = out.body[out.pause <= 2]
out.pause = out.pause[out.pause <= 2]

#fix two parameters, vary two parameters functions:

density.prime.kinit.krel <- function(t, initial.state, params = params,
                                     constant.param1, constant.param2)  {
    with(as.list(c(params, initial.state)), {
        dP = kinit - (constant.param1 + krel)*P
        dB1 = (krel)*P - constant.param2*B1
        res = c(dP, dB1)
        list(res)
    })
```

```
}

density.prime.kinit.kelong <- function(t, initial.state, params = params,
                                       constant.param1, constant.param2)  {
    with(as.list(c(params, initial.state)), {
        dP = kinit - (constant.param1 + constant.param2)*P
        dB1 = (constant.param2)*P - kelong*B1
        res = c(dP, dB1)
        list(res)
    })
}

density.prime.kinit.kpre <- function(t, initial.state, params = params,
                                     constant.param1, constant.param2)  {
    with(as.list(c(params, initial.state)), {
        dP = kinit - (kpre + constant.param1)*P
        dB1 = (constant.param1)*P - constant.param2*B1
        res = c(dP, dB1)
        list(res)
    })
}

density.prime.kpre.krel <- function(t, initial.state, params = params,
                                    constant.param1, constant.param2)  {
    with(as.list(c(params, initial.state)), {
        dP = constant.param1 - (kpre + krel)*P
        dB1 = (krel)*P - constant.param2*B1
        res = c(dP, dB1)
        list(res)
    })
}

density.prime.kpre.kelong <- function(t, initial.state, params = params,
                                      constant.param1, constant.param2)  {
    with(as.list(c(params, initial.state)), {
        dP = constant.param1 - (kpre + constant.param2)*P
        dB1 = (constant.param2)*P - kelong*B1
        res = c(dP, dB1)
        list(res)
    })
}

density.prime.krel.kelong <- function(t, initial.state, params = params,
                                      constant.param1, constant.param2)  {
    with(as.list(c(params, initial.state)), {
        dP = constant.param1 - (constant.param2 + krel)*P
        dB1 = (krel)*P - kelong*B1
        res = c(dP, dB1)
        list(res)
    })
}

kinit.krel.lists = change.two.parameters(func.density = density.prime.kinit.krel,
                                    p.change = pause.change,
                                    b.change = body.change,
                                    target.pause = out.pause,
                                    target.body = out.body,
                                    param1 = kinit.vec,
                                    param2 = krel.vec,
                                    params = c("kinit", "krel"),
                                    constant.param1.vec = kpre.vec,
                                    constant.param2.vec = kelong.vec,
                                    perc = 0.05, fold.query = 2, sample.size = 1000)
save(kinit.krel.lists, file = 'kinit.krel.lists.Rdata')

kinit.kelong.lists = change.two.parameters(func.density = density.prime.kinit.kelong,
                                    param1 = kinit.vec,
                                    param2 = kelong.vec,
                                    params = c("kinit", "kelong"),
                                    constant.param1.vec = kpre.vec,
                                    constant.param2.vec = krel.vec,
                                    perc = 0.05, fold.query = 2, sample.size = 1000)
save(kinit.kelong.lists, file = 'kinit.kelong.lists.Rdata')
```

```
kinit.kpre.lists = change.two.parameters(func.density = density.prime.kinit.kpre,
                                  param1 = kinit.vec,
                                  param2 = kpre.vec,
                                  params = c("kinit", "kpre"),
                                  constant.param1.vec = krel.vec,
                                  constant.param2.vec = kelong.vec,
                                  perc = 0.05, fold.query = 2, sample.size = 1000)
save(kinit.kpre.lists, file = 'kinit.kpre.lists.Rdata')

kpre.krel.lists = change.two.parameters(func.density = density.prime.kpre.krel,
                                  param1 = kpre.vec,
                                  param2 = krel.vec,
                                  params = c("kpre", "krel"),
                                  constant.param1.vec = kinit.vec,
                                  constant.param2.vec = kelong.vec,
                                  perc = 0.05, fold.query = 2, sample.size = 1000)
save(kpre.krel.lists, file = 'kpre.krel.lists.Rdata')

kpre.kelong.lists = change.two.parameters(func.density = density.prime.kpre.kelong,
                                  param1 = kpre.vec,
                                  param2 = kelong.vec,
                                  params = c("kpre", "kelong"),
                                  constant.param1.vec = kinit.vec,
                                  constant.param2.vec = krel.vec,
                                  perc = 0.05, fold.query = 2, sample.size = 1000)
save(kpre.kelong.lists, file = 'kpre.kelong.lists.Rdata')

krel.kelong.lists = change.two.parameters(func.density = density.prime.krel.kelong,
                                  param1 = krel.vec,
                                  param2 = kelong.vec,
                                  params = c("krel", "kelong"),
                                  constant.param1.vec = kinit.vec,
                                  constant.param2.vec = kpre.vec,
                                  perc = 0.05, fold.query = 2, sample.size = 1000)
save(krel.kelong.lists, file = 'krel.kelong.lists.Rdata')
```

Plots of the data indicate that increases in pause release rate or a decrease in elongation rate. Changes in RNA-seq and PRO-seq tend to correlate (Duarte *et al.*, 2016), which refutes the possiblity that elongation rate changes are driving the gene body differences. Nascent RNA profiling data like PRO-seq have always interpreted an increase in genic transcription as activation and a decrease in genic transcription as repression. As further evidence in our data, Figure 2 indicates that GR binding is associated with opening of chromatin. Moreover, independent massively parallel reporter assays have shown that the Glucocorticoid Receptor is exclusively an activator (Vockley *et al.*, 2016). Although elongation rates may vary over two-fold at different genes in the genome (Jonkers and Lis, 2015), we are unaware of an example where elongations rates were changed at a given gene between conditions. A systematic look discordance between RNA-seq and PRO-seq between two conditions is needed to identify candidate genes that change elongation rates.

```
#make this workflow and plot into a function to plot:

two.parameter.bw.plot.lattice <- function(param1.param2.lists, param1.vec,
                                          param2.vec, params = c("Kinit", "Krel"),
                                          constant.params = c('Kelong', 'Kpre')) {
    vec.change.param1 = c()
    vec.change.param2 = c()
    count = 0
    for (i in 1:length(param1.param2.lists[[1]])) {
        if(length(param1.param2.lists[[1]][[i]]) != 0){
        count = count + 1
        vec.change.param1[count] = mean(param1.param2.lists[[1]][[i]])/param1.vec[i]
        vec.change.param2[count] = mean(param1.param2.lists[[2]][[i]])/param2.vec[i]
        }
    }
    factor.param1.param2 = data.frame(rbind(cbind(vec.change.param1, params[1]),
                                cbind(vec.change.param2, params[2])), stringsAsFactors = FALSE)
    factor.param1.param2[,3] = paste0('constant_', constant.params[1], '_', constant.params[2])
    colnames(factor.param1.param2) = c('fold.change', 'rate', 'constants')
    factor.param1.param2[,1] = as.numeric(factor.param1.param2[,1])
    return(factor.param1.param2)
}
```

```
kinit.krel.lattice = two.parameter.bw.plot.lattice(kinit.krel.lists, kinit.vec, krel.vec,
                              params = c("Kinit", "Krel"),
                              constant.params = c('Kelong', 'Kpre'))

kinit.kelong.lattice = two.parameter.bw.plot.lattice(kinit.kelong.lists, kinit.vec, kelong.vec,
                              params = c("Kinit", "Kelong"),
                              constant.params = c('Krel', 'Kpre'))

#cannot modify these rates to reproduce the changes
#kinit.kpre.lattice = two.parameter.bw.plot.lattice(kinit.kpre.lists, kinit.vec, kpre.vec,
#                               params = c("Kinit", "Kpre"),
#                               constant.params = c('Krel', 'Kelong'))

kpre.krel.lattice = two.parameter.bw.plot.lattice(kpre.krel.lists, kpre.vec, krel.vec,
                              params = c("Kpre", "Krel"),
                              constant.params = c('Kinit', 'Kelong'))

kpre.kelong.lattice = two.parameter.bw.plot.lattice(kpre.kelong.lists, kpre.vec, kelong.vec,
                              params = c("Kpre", "Kelong"),
                              constant.params = c('Kinit', 'Krel'))

krel.kelong.lattice = two.parameter.bw.plot.lattice(krel.kelong.lists, krel.vec, kelong.vec,
                              params = c("Krel", "Kelong"),
                              constant.params = c('Kinit', 'Kpre'))

factor.param1.param2.lattice = rbind(kinit.krel.lattice,
                                     kinit.kelong.lattice,
#                                     kinit.kpre.lattice,
                                     kpre.krel.lattice,
                                     kpre.kelong.lattice,
                                     krel.kelong.lattice)

plot.two.parameter.bw <- function(factor.param1.param2, factor = "GR", y.lab, y.lim = c(0, 1.7)) {
    pdf(paste0(factor ,'_change_two_parameters.pdf'),
        useDingbats = FALSE, width=7,#*ceiling(length(unique(factor.param1.param2$constants))/3),
        height=3.2* ceiling((length(unique(factor.param1.param2$constants))/3)))
    trellis.par.set(box.umbrella = list(lty = 1, col="#93939380", lwd=1),
                    box.rectangle = list(col = '#93939380', lwd=1),
                    plot.symbol = list(col='#93939380', lwd=1, pch ='.'))
    print(bwplot(fold.change ~ rate | constants, data = factor.param1.param2,
                 between=list(y=1.0, x = 1.0),
                 scales=list(x=list(draw=TRUE, rot =45), y = list(axs = 'i'),
                             relation="free",
                             alternating=c(1,1,1,1),cex=1,font=1),
                 xlab = 'rate', ylab = y.lab, horizontal =FALSE,  col= 'black',
                 aspect = 1.5, ylim = y.lim,
                 box.width = 0.7,
                 drop.unused.levels = TRUE,
                 #layout = c(3,2),
             #    index.cond = list(c(1,2,3,4,5)),
                 par.settings=list(par.xlab.text=list(cex=1.2,font=1),
                                   par.ylab.text=list(cex=1.2,font=1),
                                   par.main.text=list(cex=1.2, font=1),
                                   plot.symbol = list(col='black', lwd=1.6, pch =19, cex = 0.25)),
                 strip = function(..., which.panel, bg) {
                     bg.col = c("grey85")
                     strip.default(..., which.panel = which.panel,
                                   bg = rep(bg.col, length = which.panel)[which.panel])
                 },
                 panel = function(..., box.ratio, col) {
                     #panel.abline(h = 0, col = 'grey45', lty = 2)
                     #panel.violin(..., col = '#736fff',
                     #             varwidth = FALSE, box.ratio = box.ratio, outer = FALSE)
                     panel.stripplot(..., col='#54545380', do.out=FALSE, jitter.data=TRUE,
                                     amount = 0.2, pch = 16)
                     panel.bwplot(..., pch = '|', do.out = FALSE)
                 }))
    dev.off()
}

plot.two.parameter.bw(factor.param1.param2.lattice, factor = 'GR_all', y.lab = 'Fold Change at GR targets\n 0 - 20 minutes')
```

```
plot.df = factor.param1.param2.lattice[factor.param1.param2.lattice$constants ==
                                                    'constant_Kelong_Kpre' |
                                                    factor.param1.param2.lattice$constants ==
                                                    'constant_Kinit_Kelong',]


plot.df$rate[plot.df$rate == 'Kinit'] = 'K'
plot.df$rate[plot.df$rate == 'Kpre'] = 'K'

plot.two.parameter.bw(plot.df, factor = 'GR',
                      y.lab = 'Fold Change at GR targets\n 0 - 20 minutes')









#change in rates:
median(kinit.krel.lattice[kinit.krel.lattice[,2] == 'Krel',][,1])
# 1.47
median(kinit.krel.lattice[kinit.krel.lattice[,2] == 'Kinit',][,1])
# 1.04
median(kpre.krel.lattice[kpre.krel.lattice[,2] == 'Kpre',][,1])
#0.97
median(kpre.krel.lattice[kpre.krel.lattice[,2] == 'Krel',][,1])
#1.52

# elongation rates of 2 - 3 kb/minute:
# how many molecules of Pol II leave the pause compartment in a second:
range(krel.vec[33.33 < kelong.vec & kelong.vec < 50])

# residency time in the pause compartment:
1/range(krel.vec[33.333 < kelong.vec & kelong.vec < 50])

#appriximate change
1/range(krel.vec[33.333 < kelong.vec & kelong.vec < 50]*1.47)
```

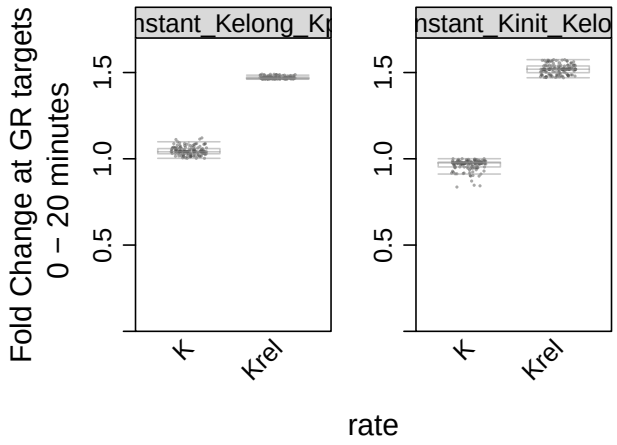


Figure 16: **Varying rates to match change in GR targets.** A 1.5 fold increase in pause release rates replicates the average change in pause and gene body signal at GR-inferred targets from 0 to 20 minutes when elongation rate remains fixed in the model. Initiation and premature release are opposing processes. Pause release rate increases by 50% in both scenarios, but when initiation and elongation are held constant, premature relase decreases by 2.5%. Holding elongation and premature release constant causes initiation rate to increase by 4.0%. these are both modest compared to the 50% increase in pause release that we observe. By making the assumption that average elongation rate along the gene body remains unchanged between the two conditions, we conclude that GR increases pause release rate by approximately 1.5 fold. The residency time for RNA Polymerase in the pause region before entering the gene body is approximately 26-37 seconds at time 0, if we assume an elongation rate of 2-3kb per minute and we allow opposing initiation and premature release parameters to fit the data. At 20 minutes, the residency time is reduced to between 18 and 25 seconds. As calculated, these residency times are for molecules that are destined for productive elongation and we are ignoring premature release of paused RNA Polymerases.

3 How does RNA Polymerase distribution change at AP1 target genes

Do the same thing for AP1 using the time point comparison that is most significant in the CDF plotting distance from AP1-containing dynamic ATAC peak and activated genes vs. unchanged genes.

```
#these comparisons are all AP1 40 - 60 minutes


ap1.pre.ratio = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'AP1' &
                                mike.pause.region.summation$time == 40,]$pause_sum, trim = 0.05) /
     mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'AP1' &
                                mike.pause.region.summation$time == 40,]$body_avg, trim = 0.05)

ap1.pause.change = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'AP1' &
                                mike.pause.region.summation$time == 60,]$pause_sum, trim = 0.05) /
     mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'AP1' &
                                mike.pause.region.summation$time == 40,]$pause_sum, trim = 0.05)

ap1.body.change = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'AP1' &
                                mike.pause.region.summation$time == 60,]$body_avg, trim = 0.05) /
     mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'AP1' &
                                mike.pause.region.summation$time == 40,]$body_avg, trim = 0.05)


print('Average ratio of pause compartment signal to gene body signal for AP1 targets:')
print(ap1.pre.ratio)
print('Average fold change in pause signal from 40 to 60 minutes for AP1 targets:')
print(ap1.pause.change)
print('Average fold change in gene body signal from 40 to 60 minutes for AP1 targets:')
print(ap1.body.change)


# the next step is to determine a set of parameters that fit the observed ratio of
# pause compartment signal to gene body (single base position) signal. Although any
# threshold can be used, we dictated that the ratio had to be within two pecent of the target

#two percent is hard coded
perc = 0.02

# the following vectors are the parameters that result in the 0 min pause/gene body signal ratio
# and the pause and gene body densities.
# The rates are considered RNA Polymerase molecules per second.
# The pause and gene body signal are considered number of RNA Polymerase molecules in the compartment.

ap1.kinit.vec =  x$a[,1,"kinit"][(( out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) > (ap1.pre.ratio - (ap1.pre.ratio*perc))) &
                 ((out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) < (ap1.pre.ratio + (ap1.pre.ratio*perc)))]

ap1.krel.vec = x$a[,1,"krel"][(( out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) > (ap1.pre.ratio - (ap1.pre.ratio*perc))) &
                 ((out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) < (ap1.pre.ratio + (ap1.pre.ratio*perc)))]

ap1.kelong.vec = x$a[,1,"kelong"][(( out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) > (ap1.pre.ratio - (ap1.pre.ratio*perc))) &
                 ((out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) < (ap1.pre.ratio + (ap1.pre.ratio*perc)))]

ap1.kpre.vec = x$a[,1,"kpre"][(( out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) > (ap1.pre.ratio - (ap1.pre.ratio*perc))) &
                 ((out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) < (ap1.pre.ratio + (ap1.pre.ratio*perc)))]

ap1.out.pause = out$y[,1,10.1,"P"][(( out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) > (ap1.pre.ratio - (ap1.pre.ratio*perc))) &
                 ((out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) < (ap1.pre.ratio + (ap1.pre.ratio*perc)))]

ap1.out.body = out$y[,1,10.1,"B1"][(( out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) > (ap1.pre.ratio - (ap1.pre.ratio*perc))) &
                              ((out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) < (ap1.pre.ratio + (ap1.pre.ratio*perc)))]


#no more than pause region density of 2 polymerase molecules
ap1.kinit.vec = ap1.kinit.vec[ap1.out.pause <= 2]
ap1.krel.vec = ap1.krel.vec[ap1.out.pause <= 2]
ap1.kelong.vec = ap1.kelong.vec[ap1.out.pause <= 2]
ap1.kpre.vec = ap1.kpre.vec[ap1.out.pause <= 2]
ap1.out.body = ap1.out.body[ap1.out.pause <= 2]
ap1.out.pause = ap1.out.pause[ap1.out.pause <= 2]


ap1.kinit.krel.lists = change.two.parameters(func.density = density.prime.kinit.krel,
                                p.change = ap1.pause.change,
                                b.change = ap1.body.change,
                                target.pause = ap1.out.pause,
                                target.body = ap1.out.body,
                                param1 = ap1.kinit.vec,
                                param2 = ap1.krel.vec,
                                params = c("kinit", "krel"),
                                constant.param1.vec = ap1.kpre.vec,
                                constant.param2.vec = ap1.kelong.vec,
                                perc = 0.05,
```

```
                                fold.query = 2,
                                sample.size = 1000)
save(ap1.kinit.krel.lists, file = 'ap1.kinit.krel.lists.Rdata')

ap1.kinit.kelong.lists = change.two.parameters(func.density = density.prime.kinit.kelong,
                                p.change = ap1.pause.change,
                                b.change = ap1.body.change,
                                target.pause = ap1.out.pause,
                                target.body = ap1.out.body,
                                param1 = ap1.kinit.vec,
                                param2 = ap1.kelong.vec,
                                params = c("kinit", "kelong"),
                                constant.param1.vec = ap1.kpre.vec,
                                constant.param2.vec = ap1.krel.vec,
                                fold.query = 2,
                                sample.size = 1000)
save(ap1.kinit.kelong.lists, file = 'ap1.kinit.kelong.lists.Rdata')

ap1.kinit.kpre.lists = change.two.parameters(func.density = density.prime.kinit.kpre,
                                p.change = ap1.pause.change,
                                b.change = ap1.body.change,
                                target.pause = ap1.out.pause,
                                target.body = ap1.out.body,
                                param1 = ap1.kinit.vec,
                                param2 = ap1.kpre.vec,
                                params = c("kinit", "kpre"),
                                constant.param1.vec = ap1.krel.vec,
                                constant.param2.vec = ap1.kelong.vec,
                                fold.query = 2,
                                sample.size = 1000)
save(ap1.kinit.kpre.lists, file = 'ap1.kinit.kpre.lists.Rdata')

ap1.kpre.krel.lists = change.two.parameters(func.density = density.prime.kpre.krel,
                                p.change = ap1.pause.change,
                                b.change = ap1.body.change,
                                target.pause = ap1.out.pause,
                                target.body = ap1.out.body,
                                param1 = ap1.kpre.vec,
                                param2 = ap1.krel.vec,
                                params = c("kpre", "krel"),
                                constant.param1.vec = ap1.kinit.vec,
                                constant.param2.vec = ap1.kelong.vec,
                                fold.query = 2,
                                sample.size = 1000)
save(ap1.kpre.krel.lists, file = 'ap1.kpre.krel.lists.Rdata')

ap1.kpre.kelong.lists = change.two.parameters(func.density = density.prime.kpre.kelong,
                                p.change = ap1.pause.change,
                                b.change = ap1.body.change,
                                target.pause = ap1.out.pause,
                                target.body = ap1.out.body,
                                param1 = ap1.kpre.vec,
                                param2 = ap1.kelong.vec,
                                params = c("kpre", "kelong"),
                                constant.param1.vec = ap1.kinit.vec,
                                constant.param2.vec = ap1.krel.vec,
                                fold.query = 2,
                                sample.size = 1000)
save(ap1.kpre.kelong.lists, file = 'ap1.kpre.kelong.lists.Rdata')

ap1.krel.kelong.lists = change.two.parameters(func.density = density.prime.krel.kelong,
                                p.change = ap1.pause.change,
                                b.change = ap1.body.change,
                                target.pause = ap1.out.pause,
                                target.body = ap1.out.body,
                                param1 = ap1.krel.vec,
                                param2 = ap1.kelong.vec,
                                params = c("krel", "kelong"),
                                constant.param1.vec = ap1.kinit.vec,
                                constant.param2.vec = ap1.kpre.vec,
                                fold.query = 2,
                                sample.size = 1000)
save(ap1.krel.kelong.lists, file = 'ap1.krel.kelong.lists.Rdata')


ap1.kinit.krel.lattice = two.parameter.bw.plot.lattice(ap1.kinit.krel.lists, ap1.kinit.vec, ap1.krel.vec,
                            params = c("Kinit", "Krel"),
                            constant.params = c('Kelong', 'Kpre'))

ap1.kinit.kelong.lattice = two.parameter.bw.plot.lattice(ap1.kinit.kelong.lists, ap1.kinit.vec, ap1.kelong.vec,
                            params = c("Kinit", "Kelong"),
                            constant.params = c('Krel', 'Kpre'))

#ap1.kinit.kpre.lattice = two.parameter.bw.plot.lattice(ap1.kinit.kpre.lists, ap1.kinit.vec, ap.1kpre.vec,
#                             params = c("Kinit", "Kpre"),
#                             constant.params = c('Krel', 'Kelong'))
```

```
ap1.kpre.krel.lattice = two.parameter.bw.plot.lattice(ap1.kpre.krel.lists, ap1.kpre.vec, ap1.krel.vec,
                              params = c("Kpre", "Krel"),
                              constant.params = c('Kinit', 'Kelong'))

ap1.kpre.kelong.lattice = two.parameter.bw.plot.lattice(ap1.kpre.kelong.lists, ap1.kpre.vec, ap1.kelong.vec,
                              params = c("Kpre", "Kelong"),
                              constant.params = c('Kinit', 'Krel'))

#ap1.krel.kelong.lattice = two.parameter.bw.plot.lattice(ap1.krel.kelong.lists, ap1.krel.vec, ap1.kelong.vec,
#                               params = c("Krel", "Kelong"),
#                               constant.params = c('Kinit', 'Kpre'))

ap1.factor.param1.param2.lattice = rbind(ap1.kinit.krel.lattice,
                                         ap1.kinit.kelong.lattice,
#                                         ap1.kinit.kpre.lattice,
                                         ap1.kpre.krel.lattice,
                                         ap1.kpre.kelong.lattice)
#                                         ap1.krel.kelong.lattice)



ap1.plot.df = ap1.factor.param1.param2.lattice[ap1.factor.param1.param2.lattice$constants ==
                                                 'constant_Kelong_Kpre' |
                                                 ap1.factor.param1.param2.lattice$constants ==
                                                 'constant_Kinit_Kelong',]


ap1.plot.df$rate[ap1.plot.df$rate == 'Kinit'] = 'K'
ap1.plot.df$rate[ap1.plot.df$rate == 'Kpre'] = 'K'

plot.two.parameter.bw(ap1.plot.df, factor = 'AP1', y.lim = c(0, 1.7),
                      y.lab = 'Fold Change at AP1 targets\n 40 - 60 minutes')




#change in rates:
median(ap1.kinit.krel.lattice[ap1.kinit.krel.lattice[,2] == 'Krel',][,1])
# 0.76
median(ap1.kinit.krel.lattice[ap1.kinit.krel.lattice[,2] == 'Kinit',][,1])
# 1.31
median(ap1.kpre.krel.lattice[ap1.kpre.krel.lattice[,2] == 'Kpre',][,1])
#0.75
median(ap1.kpre.krel.lattice[ap1.kpre.krel.lattice[,2] == 'Krel',][,1])
#0.80

# elongation rates of 2 - 3 kb/minute:
# how many molecules of Pol II leave the pause compartment in a second:
range(ap1.krel.vec[33.33 < ap1.kelong.vec & ap1.kelong.vec < 50])

# how many molecules of Pol II prematurely terminate the pause compartment in a second:
range(ap1.kpre.vec[33.33 < ap1.kelong.vec & ap1.kelong.vec < 50])

# how many molecules of Pol II enter the pause compartment in a second:
range(ap1.kinit.vec[33.33 < ap1.kelong.vec & ap1.kelong.vec < 50])

# residency time in the pause compartment:
1/range(ap1.krel.vec[33.333 < ap1.kelong.vec & ap1.kelong.vec < 50])

#appriximate change
1/range(ap1.krel.vec[33.333 < ap1.kelong.vec & ap1.kelong.vec < 50]*0.76)
```

```
sp.pre.ratio = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'SP' &
                              mike.pause.region.summation$time == 60,]$pause_sum, trim = 0.05) /
     mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'SP' &
                              mike.pause.region.summation$time == 60,]$body_avg, trim = 0.05)

sp.pause.change = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'SP' &
                              mike.pause.region.summation$time == 120,]$pause_sum, trim = 0.05) /
     mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'SP' &
                              mike.pause.region.summation$time == 60,]$pause_sum, trim = 0.05)

sp.body.change = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'SP' &
                              mike.pause.region.summation$time == 120,]$body_avg, trim = 0.05) /
     mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'SP' &
                              mike.pause.region.summation$time == 60,]$body_avg, trim = 0.05)


print('Average ratio of pause compartment signal to gene body signal for SP targets:')
print(sp.pre.ratio)
print('Average fold change in pause signal from 40 to 60 minutes for SP targets:')
print(sp.pause.change)
```

```
print('Average fold change in gene body signal from 40 to 60 minutes for SP targets:')
print(sp.body.change)


# the next step is to determine a set of parameters that fit the observed ratio of
# pause compartment signal to gene body (single base position) signal. Although any
# threshold can be used, we dictated that the ratio had to be within two pecent of the target

#two percent is hard coded
perc = 0.02

# the following vectors are the parameters that result in the 0 min pause/gene body signal ratio
# and the pause and gene body densities.
# The rates are considered RNA Polymerase molecules per second.
# The pause and gene body signal are considered number of RNA Polymerase molecules in the compartment.

sp.kinit.vec =  x$a[,1,"kinit"][(( out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) > (sp.pre.ratio - (sp.pre.ratio*perc))) &
                ((out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) < (sp.pre.ratio + (sp.pre.ratio*perc)))]

sp.krel.vec = x$a[,1,"krel"][(( out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) > (sp.pre.ratio - (sp.pre.ratio*perc))) &
                ((out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) < (sp.pre.ratio + (sp.pre.ratio*perc)))]

sp.kelong.vec = x$a[,1,"kelong"][(( out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) > (sp.pre.ratio - (sp.pre.ratio*perc))) &
                ((out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) < (sp.pre.ratio + (sp.pre.ratio*perc)))]

sp.kpre.vec = x$a[,1,"kpre"][(( out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) > (sp.pre.ratio - (sp.pre.ratio*perc))) &
                ((out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) < (sp.pre.ratio + (sp.pre.ratio*perc)))]

sp.out.pause = out$y[,1,10.1,"P"][(( out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) > (sp.pre.ratio - (sp.pre.ratio*perc))) &
                ((out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) < (sp.pre.ratio + (sp.pre.ratio*perc)))]

sp.out.body = out$y[,1,10.1,"B1"][(( out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) > (sp.pre.ratio - (sp.pre.ratio*perc))) &
                            ((out$y[,1,10.1,"P"] / (out$y[,1,10.1,"B1"])) < (sp.pre.ratio + (sp.pre.ratio*perc)))]


#no more than pause region density of 2 polymerase molecules
sp.kinit.vec = sp.kinit.vec[sp.out.pause <= 2]
sp.krel.vec = sp.krel.vec[sp.out.pause <= 2]
sp.kelong.vec = sp.kelong.vec[sp.out.pause <= 2]
sp.kpre.vec = sp.kpre.vec[sp.out.pause <= 2]
sp.out.body = sp.out.body[sp.out.pause <= 2]
sp.out.pause = sp.out.pause[sp.out.pause <= 2]


sp.kinit.krel.lists = change.two.parameters(func.density = density.prime.kinit.krel,
                                p.change = sp.pause.change,
                                b.change = sp.body.change,
                                target.pause = sp.out.pause,
                                target.body = sp.out.body,
                                param1 = sp.kinit.vec,
                                param2 = sp.krel.vec,
                                params = c("kinit", "krel"),
                                constant.param1.vec = sp.kpre.vec,
                                constant.param2.vec = sp.kelong.vec,
                                perc = 0.05,
                                fold.query = 2,
                                sample.size = 1000)
save(sp.kinit.krel.lists, file = 'sp.kinit.krel.lists.Rdata')

sp.kinit.kelong.lists = change.two.parameters(func.density = density.prime.kinit.kelong,
                                p.change = sp.pause.change,
                                b.change = sp.body.change,
                                target.pause = sp.out.pause,
                                target.body = sp.out.body,
                                param1 = sp.kinit.vec,
                                param2 = sp.kelong.vec,
                                params = c("kinit", "kelong"),
                                constant.param1.vec = sp.kpre.vec,
                                constant.param2.vec = sp.krel.vec,
                                fold.query = 2,
                                sample.size = 1000)
save(sp.kinit.kelong.lists, file = 'sp.kinit.kelong.lists.Rdata')

sp.kinit.kpre.lists = change.two.parameters(func.density = density.prime.kinit.kpre,
                                p.change = sp.pause.change,
                                b.change = sp.body.change,
                                target.pause = sp.out.pause,
                                target.body = sp.out.body,
                                param1 = sp.kinit.vec,
                                param2 = sp.kpre.vec,
                                params = c("kinit", "kpre"),
                                constant.param1.vec = sp.krel.vec,
                                constant.param2.vec = sp.kelong.vec,
                                fold.query = 2,
                                sample.size = 1000)
save(sp.kinit.kpre.lists, file = 'sp.kinit.kpre.lists.Rdata')
```

```
sp.kpre.krel.lists = change.two.parameters(func.density = density.prime.kpre.krel,
                                  p.change = sp.pause.change,
                                  b.change = sp.body.change,
                                  target.pause = sp.out.pause,
                                  target.body = sp.out.body,
                                  param1 = sp.kpre.vec,
                                  param2 = sp.krel.vec,
                                  params = c("kpre", "krel"),
                                  constant.param1.vec = sp.kinit.vec,
                                  constant.param2.vec = sp.kelong.vec,
                                  fold.query = 2,
                                  sample.size = 1000)
save(sp.kpre.krel.lists, file = 'sp.kpre.krel.lists.Rdata')

sp.kpre.kelong.lists = change.two.parameters(func.density = density.prime.kpre.kelong,
                                  p.change = sp.pause.change,
                                  b.change = sp.body.change,
                                  target.pause = sp.out.pause,
                                  target.body = sp.out.body,
                                  param1 = sp.kpre.vec,
                                  param2 = sp.kelong.vec,
                                  params = c("kpre", "kelong"),
                                  constant.param1.vec = sp.kinit.vec,
                                  constant.param2.vec = sp.krel.vec,
                                  fold.query = 2,
                                  sample.size = 1000)
save(sp.kpre.kelong.lists, file = 'sp.kpre.kelong.lists.Rdata')

sp.krel.kelong.lists = change.two.parameters(func.density = density.prime.krel.kelong,
                                  p.change = sp.pause.change,
                                  b.change = sp.body.change,
                                  target.pause = sp.out.pause,
                                  target.body = sp.out.body,
                                  param1 = sp.krel.vec,
                                  param2 = sp.kelong.vec,
                                  params = c("krel", "kelong"),
                                  constant.param1.vec = sp.kinit.vec,
                                  constant.param2.vec = sp.kpre.vec,
                                  fold.query = 2,
                                  sample.size = 1000)
save(sp.krel.kelong.lists, file = 'sp.krel.kelong.lists.Rdata')


sp.kinit.krel.lattice = two.parameter.bw.plot.lattice(sp.kinit.krel.lists, sp.kinit.vec, sp.krel.vec,
                              params = c("Kinit", "Krel"),
                              constant.params = c('Kelong', 'Kpre'))

sp.kinit.kelong.lattice = two.parameter.bw.plot.lattice(sp.kinit.kelong.lists, sp.kinit.vec, sp.kelong.vec,
                              params = c("Kinit", "Kelong"),
                              constant.params = c('Krel', 'Kpre'))

#sp.kinit.kpre.lattice = two.parameter.bw.plot.lattice(sp.kinit.kpre.lists, sp.kinit.vec, ap.1kpre.vec,
#                               params = c("Kinit", "Kpre"),
#                               constant.params = c('Krel', 'Kelong'))

sp.kpre.krel.lattice = two.parameter.bw.plot.lattice(sp.kpre.krel.lists, sp.kpre.vec, sp.krel.vec,
                              params = c("Kpre", "Krel"),
                              constant.params = c('Kinit', 'Kelong'))

sp.kpre.kelong.lattice = two.parameter.bw.plot.lattice(sp.kpre.kelong.lists, sp.kpre.vec, sp.kelong.vec,
                              params = c("Kpre", "Kelong"),
                              constant.params = c('Kinit', 'Krel'))

#sp.krel.kelong.lattice = two.parameter.bw.plot.lattice(sp.krel.kelong.lists, sp.krel.vec, sp.kelong.vec,
#                               params = c("Krel", "Kelong"),
#                               constant.params = c('Kinit', 'Kpre'))

sp.factor.param1.param2.lattice = rbind(sp.kinit.krel.lattice,
                                    sp.kinit.kelong.lattice,
#                                    sp.kinit.kpre.lattice,
                                    sp.kpre.krel.lattice,
                                    sp.kpre.kelong.lattice)
#                                    sp.krel.kelong.lattice)



sp.plot.df = sp.factor.param1.param2.lattice[sp.factor.param1.param2.lattice$constants ==
                                             'constant_Kelong_Kpre' |
                                             sp.factor.param1.param2.lattice$constants ==
                                             'constant_Kinit_Kelong',]


sp.plot.df$rate[sp.plot.df$rate == 'Kinit'] = 'K'
sp.plot.df$rate[sp.plot.df$rate == 'Kpre'] = 'K'
```

```
plot.two.parameter.bw(sp.plot.df, factor = 'SP', y.lim = c(0, 1.7),
                      y.lab = 'Fold Change at SP targets\n 60 - 120 minutes')




#change in rates:
median(sp.kinit.krel.lattice[sp.kinit.krel.lattice[,2] == 'Krel',][,1])
#
median(sp.kinit.krel.lattice[sp.kinit.krel.lattice[,2] == 'Kinit',][,1])
#
median(sp.kpre.krel.lattice[sp.kpre.krel.lattice[,2] == 'Kpre',][,1])
#
median(sp.kpre.krel.lattice[sp.kpre.krel.lattice[,2] == 'Krel',][,1])
#

# elongation rates of 2 - 3 kb/minute:
# how many molecules of Pol II leave the pause compartment in a second:
range(sp.krel.vec[33.33 < sp.kelong.vec & sp.kelong.vec < 50])

# how many molecules of Pol II prematurely terminate the pause compartment in a second:
range(sp.kpre.vec[33.33 < sp.kelong.vec & sp.kelong.vec < 50])

# how many molecules of Pol II enter the pause compartment in a second:
print("how many molecules of Pol II enter the pause compartment in a minute 60 min")
range(sp.kinit.vec[33.33 < sp.kelong.vec & sp.kelong.vec < 50]) * 60
print("how many molecules of Pol II enter the pause compartment in a minute 120 min")
range(sp.kinit.vec[33.33 < sp.kelong.vec & sp.kelong.vec < 50]) * 60 *
median(sp.kinit.krel.lattice[sp.kinit.krel.lattice[,2] == 'Kinit',][,1])
0.9267684
# residency time in the pause compartment:
1/range(sp.krel.vec[33.333 < sp.kelong.vec & sp.kelong.vec < 50])

#appriximate change
1/range(sp.krel.vec[33.333 < sp.kelong.vec & sp.kelong.vec < 50]*0.76)
```

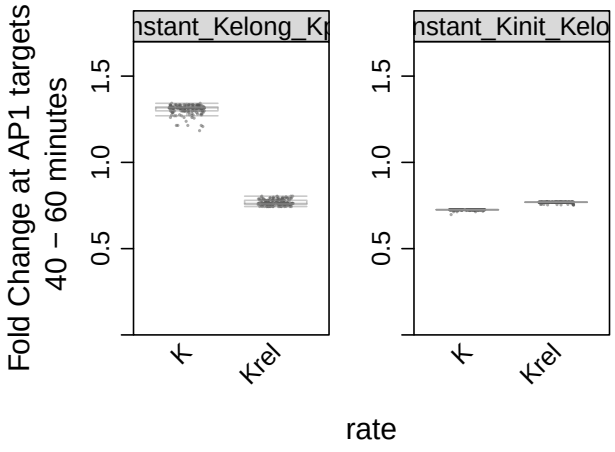


Figure 17: **Varying rates to match change in AP1 targets.** A 1.32 fold increase in initiation rate replicates the average change in pause and gene body signal at AP1-inferred targets from 40 to 60 minutes when elongation rate and non-productive pause release remains fixed in the model. AP1 targets have an apparent pause release rate change of 0.76-fold. The residency time for RNA Polymerase in the pause region before entering the gene body at time 40 is approximately 19-28 seconds, if we assume an elongation rate of 2-3kb per minute. At 60 minutes, this time is increased to between 25 and 36 seconds. Again, this residency time is for molecules that are destined for productive elongation.

Arun, could you point me to GR and AP1 only (but together) genes? I can draw the constrained network and use the average rates in a model to predict the change in pause and gene body density at these genes.

The biggest visualization challenge will be how to reproduce the waveform such that the integrated intensity in the pause region is what we observe when we sum over the entire region, as we do for the model. I think it is most intuitive to put everything in the conventional composite format for the reader.

I do want to reproduce the composites, but since the true densities in each region is not reflected in a composite, I am not sure about how good composites are are illustrating what I consider true pausing indexes, which considers the entire signal of the pause region compared to density in the gene body. Again density in the gene body makes sense because there is a near uniform distribution of polymerase over the region, as opposed to the pause region, which is not wide or easily defined. I think this works, but I need to confirm that the waveform is a parabola: `https://richbeveridge.wordpress.com/2009/10/23/area-under-a-parabola/`

4 Visualizing how observed changes in pause release and initiaition affects composite RNA Polymerase distribution in the compartment model

```
library(zoo)

pause.0.gr = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'GR' &
                                mike.pause.region.summation$time == 0,]$pause_sum, trim = 0.05)

pause.20.gr = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'GR' &
                                mike.pause.region.summation$time == 20,]$pause_sum, trim = 0.05)


body.0.gr = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'GR' &
                                mike.pause.region.summation$time == 0,]$body_avg, trim = 0.05)


body.20.gr = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'GR' &
                                mike.pause.region.summation$time == 20,]$body_avg, trim = 0.05)

#input.gr = data.frame(1, pause.0.gr, body.0.gr)
input.gr.0.body = data.frame(c(1:length(seq(0, 1, by = 0.001))), seq(0, 1, by = 0.001), body.0.gr)
colnames(input.gr.0.body) = c('iteration', 'Pause', 'Body')

input.gr.20.body = data.frame(c(1:length(seq(0, 1, by = 0.001))), seq(0, 1, by = 0.001), body.20.gr)
colnames(input.gr.20.body) = c('iteration', 'Pause', 'Body')




pause.40.ap1 = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'AP1' &
                                mike.pause.region.summation$time == 40,]$pause_sum, trim = 0.05)

pause.60.ap1 = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'AP1' &
                                mike.pause.region.summation$time == 60,]$pause_sum, trim = 0.05)


body.40.ap1 = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'AP1' &
                                mike.pause.region.summation$time == 40,]$body_avg, trim = 0.05)


body.60.ap1 = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'AP1' &
```

```
                                mike.pause.region.summation$time == 60,]$body_avg, trim = 0.05)

input.ap1.40.body = data.frame(c(1:length(seq(0, 1, by = 0.001))), seq(0, 1, by = 0.001), body.40.ap1)
colnames(input.ap1.40.body) = c('iteration', 'Pause', 'Body')

input.ap1.60.body = data.frame(c(1:length(seq(0, 1, by = 0.001))), seq(0, 1, by = 0.001), body.60.ap1)
colnames(input.ap1.60.body) = c('iteration', 'Pause', 'Body')




pause.60.sp = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'SP' &
                                mike.pause.region.summation$time == 60,]$pause_sum, trim = 0.05)

pause.120.sp = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'SP' &
                                mike.pause.region.summation$time == 120,]$pause_sum, trim = 0.05)


body.60.sp = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'SP' &
                                mike.pause.region.summation$time == 60,]$body_avg, trim = 0.05)


body.120.sp = mean(mike.pause.region.summation[mike.pause.region.summation$factor == 'SP' &
                                mike.pause.region.summation$time == 120,]$body_avg, trim = 0.05)

input.sp.60.body = data.frame(c(1:length(seq(0, 1, by = 0.001))), seq(0, 1, by = 0.001), body.60.sp)
colnames(input.sp.40.body) = c('iteration', 'Pause', 'Body')

input.sp.120.body = data.frame(c(1:length(seq(0, 1, by = 0.001))), seq(0, 1, by = 0.001), body.120.sp)
colnames(input.sp.120.body) = c('iteration', 'Pause', 'Body')




pro.integrated.peak <- function(input, tau = 20,min.pk = 0, max.pk =1,
                               dpk =0.001, gene.len = 2000, pause.height, time.char = '0 min') {
    bp = seq(0,gene.len-1)
    colnames(input) = c("iteration","Pause", "Body")
    body.parameters = mapply(find.body.param,
                             bpeak = input$Body,
                             tau = tau,
                             pausepeak = input$Pause,
                             min.pk = min.pk,
                             max.pk = max.pk,
                             dpk = dpk)
    #print(body.parameters)
    input$bodyParam = body.parameters
    df.out = as.data.frame(matrix(NA, nrow=0, ncol = 4))
    bp = seq(0,gene.len-1)
    for (i in 1:length(input$Pause)) {#need to revisit and use apply func.
        x = get.pro.waveform(pausepeak=input$Pause[i],
                             bpeak=input$Body[i],
                             bodypeak=input$bodyParam[i],
                             bp.seq=bp,
                             tau=tau)$vec
        y = as.data.frame(cbind(i, x, bp))

        integrated.intensity = sum(diff(y[1:120, 3])*rollmean(y[1:120, 2],2))
        out.param = as.data.frame(cbind(input$Pause[i], integrated.intensity,
                                        input$bodyParam[i], input$Body[i]))
        colnames(out.param) = colnames(df.out)
        df.out = rbind(df.out, out.param)
```

```
    }
    #plot.pro.simulation.composites(df.out,
    #                                filename = filename)
    pause.height.body.param = df.out[which.min(abs(df.out[,2] - pause.height)),]
    #return(pause.height.body.param)
    waveform = get.pro.waveform(pausepeak=pause.height.body.param[1,1],
                                bpeak=pause.height.body.param[1,4],
                                bodypeak=pause.height.body.param[1,3],
                                bp.seq=bp,
                                tau=tau)$vec
    df.wave = as.data.frame(cbind(time.char, waveform, bp))
    return(df.wave)
}

plot.pro.simulation.composites.2 <- function(input, pause.offset = 0,
                                             filename = 'dynamic_pro_model',
                                             ylim = c(0, 0.0082),
                                             trace.col = c('grey50', '#6aa3ce')) {
    colors.ramp = trace.col
    pdf(paste0(filename, '.pdf'), width=3.43, height = 3.43)
    print(
        xyplot(input[,2] ~ input[,3], groups = input[,1], data = input, type = 'l',
               scales=list(x=list(cex=0.8,relation = "free"),
                   y =list(cex=0.8,alternating=FALSE,axs = 'i')),
               xlim = c(0,400),
               ylim = ylim,
               col = colors.ramp,
               par.settings = list(strip.background=list(col="grey85"),
                                   superpose.symbol = list(pch = c(16),
                                                           col = colors.ramp, cex =0.7),
                                   superpose.line = list(lwd=c(2.5), col = colors.ramp,
                                    lty = c(1,1))),
       aspect=1.0,
    auto.key = list(points=F, lines=T, cex=0.8),
       lwd=2,
       ylab = list(label = "Simulated PRO-seq Signal", cex =1,font=1),
       xlab = list(label = 'Distance from Pause Peak', cex =1,font=1),
       panel = function(x, y, ...) {
           panel.xyplot(x, y, ...)
           panel.abline(h = 0, lty =1, lwd = 1.5, col = 'grey80')
           panel.abline(v = 0 - pause.offset, lty =2, lwd = 0.5, col = 'red')
           panel.abline(v = 50 - pause.offset, lty =2, lwd = 0.5, col = 'red')
       }
       )
       )
    dev.off()
}
```

By increasing the pause release rate by 1.48x and increasing the initiation rate by 1.01x, we observe the change from the previous composites (Figure 18). A goal for the adipogenesis paper is to systematically ascribe function to the 13 factors based on how target genes respond. Isolating genes that are inferred to be exclusively or predominantly regulated by a single factor will be paramount. Much like we established an environment score in Figure 4F, we will use a combination of distance, change in accessiblity, and motif score to establish a relative transcription factor contribution score for each gene. The GR target gene composite profiles can be reproduced from the following code: `https://raw.githubusercontent.com/guertinlab/modeling_PRO_composites/main/gr_composites.R`.
Raw code: `https://raw.githubusercontent.com/guertinlab/modeling_PRO_composites/main/stim_pause_rel.R`

```
#initial conditions are:
#pause.peak
#body.peak
start.kelong = kelong.vec[which.min(abs(kelong.vec - 41.66667))]
```

```
start.kinit = kinit.vec[which.min(abs(kelong.vec - 41.66667))]
start.krel = krel.vec[which.min(abs(kelong.vec - 41.66667))]
start.kpre = kpre.vec[which.min(abs(kelong.vec - 41.66667))]
start.body = out.body[which.min(abs(kelong.vec - 41.66667))]
start.pause = out.pause[which.min(abs(kelong.vec - 41.66667))]

change.kinit = mean(kinit.krel.lists[[1]][[which.min(abs(kelong.vec - 41.66667))]])
change.krel = mean(kinit.krel.lists[[2]][[which.min(abs(kelong.vec - 41.66667))]])

print('change in Kinit:')
print(change.kinit/start.kinit)
print('change in Krel:')
print(change.krel/start.krel)

krel = start.krel
kinit = start.kinit
kelong = start.kelong
kpre = start.kpre

# plot the steady state profile
# set to zero and solve
# dP = kinit - (kpre + krel)*P
# dB = (krel)*P - kelong*B
start.pause.peak = kinit/(kpre + krel)
start.body.peak = ((krel)*start.pause.peak)/kelong

change.pause.peak = change.kinit/(kpre + change.krel)
change.body.peak = ((change.krel)*change.pause.peak)/kelong

input.gr.0.body = data.frame(c(1:length(seq(0, 0.1, by = 0.0001))),
                             seq(0, 0.1, by = 0.0001), start.body.peak)
colnames(input.gr.0.body) = c('iteration', 'Pause', 'Body')

input.gr.20.body = data.frame(c(1:length(seq(0, 0.1, by = 0.0001))),
                              seq(0, 0.1, by = 0.0001), change.body.peak)
colnames(input.gr.20.body) = c('iteration', 'Pause', 'Body')

# you get this error if your parameter space does not fit target data:
# Error in bodypeak + exp(1) : non-numeric argument to binary operator
peak.height.0.gr = pro.integrated.peak(input.gr.0.body,
                                       pause.height = start.pause.peak, tau = 10)

peak.height.20.gr = pro.integrated.peak(input.gr.20.body,
                                        pause.height = change.pause.peak, tau = 10,
                                        time.char = '20 min')

plot.lattice.gr.model = rbind(peak.height.0.gr,peak.height.20.gr)
colnames(plot.lattice.gr.model) = c('V1', 'V2', 'V3')
plot.lattice.gr.model[,2] = as.numeric(as.character(plot.lattice.gr.model[,2]))
plot.lattice.gr.model[,3] = as.numeric(as.character(plot.lattice.gr.model[,3]))

#center on pause peak
pause.off = plot.lattice.gr.model[which.max(plot.lattice.gr.model[,2]),3]
plot.lattice.gr.model[,3] = plot.lattice.gr.model[,3] - pause.off

max( plot.lattice.gr.model[,2])+ max( plot.lattice.gr.model[,2]) * 0.07
#plot
plot.pro.simulation.composites.2(plot.lattice.gr.model, pause.offset = pause.off,
                                 filename = 'model_GR_0_20_min', ylim = c(0, 0.03))

#AP1

ap1.start.kelong = ap1.kelong.vec[which.min(abs(ap1.kelong.vec - 41.66667))]
```

```
ap1.start.kinit = ap1.kinit.vec[which.min(abs(ap1.kelong.vec - 41.66667))]
ap1.start.krel = ap1.krel.vec[which.min(abs(ap1.kelong.vec - 41.66667))]
ap1.start.kpre = ap1.kpre.vec[which.min(abs(ap1.kelong.vec - 41.66667))]
ap1.start.body = ap1.out.body[which.min(abs(ap1.kelong.vec - 41.66667))]
ap1.start.pause = ap1.out.pause[which.min(abs(ap1.kelong.vec - 41.66667))]

ap1.change.kinit = mean(ap1.kinit.krel.lists[[1]][[which.min(abs(ap1.kelong.vec - 41.66667))]])
ap1.change.krel = mean(ap1.kinit.krel.lists[[2]][[which.min(abs(ap1.kelong.vec - 41.66667))]])

print('change in Kinit:')
print(ap1.change.kinit/ap1.start.kinit)
print('change in Krel:')
print(ap1.change.krel/ap1.start.krel)

krel = ap1.start.krel
kinit = ap1.start.kinit
kelong = ap1.start.kelong
kpre = ap1.start.kpre

# plot the steady state profile
# set to zero and solve
# dP = kinit - (kpre + krel)*P
# dB = (krel)*P - kelong*B
start.pause.peak = kinit/(kpre + krel)
start.body.peak = ((krel)*start.pause.peak)/kelong

change.pause.peak = ap1.change.kinit/(kpre + ap1.change.krel)
change.body.peak = ((ap1.change.krel)*change.pause.peak)/kelong

input.ap1.40.body = data.frame(c(1:length(seq(0, 0.1, by = 0.0001))),
                               seq(0, 0.1, by = 0.0001), start.body.peak)
colnames(input.ap1.40.body) = c('iteration', 'Pause', 'Body')

input.ap1.60.body = data.frame(c(1:length(seq(0, 0.1, by = 0.0001))),
                                seq(0, 0.1, by = 0.0001), change.body.peak)
colnames(input.ap1.60.body) = c('iteration', 'Pause', 'Body')

# you get this error if your parameter space does not fit target data:
# Error in bodypeak + exp(1) : non-numeric argument to binary operator
peak.height.40.ap1 = pro.integrated.peak(input.ap1.40.body,
                                         pause.height = start.pause.peak, tau = 10,
                                         time.char = '40 min')

peak.height.60.ap1 = pro.integrated.peak(input.ap1.60.body,
                                          pause.height = change.pause.peak, tau = 10,
                                          time.char = '60 min')

plot.lattice.ap1.model = rbind(peak.height.40.ap1, peak.height.60.ap1)
colnames(plot.lattice.ap1.model) = c('V1', 'V2', 'V3')
plot.lattice.ap1.model[,2] = as.numeric(as.character(plot.lattice.ap1.model[,2]))
plot.lattice.ap1.model[,3] = as.numeric(as.character(plot.lattice.ap1.model[,3]))

#center on pause peak
pause.off = plot.lattice.ap1.model[which.max(plot.lattice.ap1.model[,2]),3]
plot.lattice.ap1.model[,3] = plot.lattice.ap1.model[,3] - pause.off


#plot
plot.pro.simulation.composites.2(plot.lattice.ap1.model, pause.offset = pause.off,
                                 trace.col = c('grey50', '#ffcd30'), ylim = c(0, 0.027),
                                filename = 'model_AP1_40_60_min')

#SP1
```

```
rate.elong = 41.66667
sp.start.kelong = sp.kelong.vec[which.min(abs(sp.kelong.vec - rate.elong))]
sp.start.kinit = sp.kinit.vec[which.min(abs(sp.kelong.vec -rate.elong))]
sp.start.krel = sp.krel.vec[which.min(abs(sp.kelong.vec - rate.elong))]
sp.start.kpre = sp.kpre.vec[which.min(abs(sp.kelong.vec - rate.elong))]
sp.start.body = sp.out.body[which.min(abs(sp.kelong.vec - rate.elong))]
sp.start.pause = sp.out.pause[which.min(abs(sp.kelong.vec - rate.elong))]

sp.change.kinit = mean(sp.kinit.krel.lists[[1]][[which.min(abs(sp.kelong.vec - rate.elong))]])
sp.change.krel = mean(sp.kinit.krel.lists[[2]][[which.min(abs(sp.kelong.vec - rate.elong))]])

print('change in Kinit:')
print(sp.change.kinit/sp.start.kinit)
print('change in Krel:')
print(sp.change.krel/sp.start.krel)

krel = sp.start.krel
kinit = sp.start.kinit
kelong = sp.start.kelong
kpre = sp.start.kpre

print('60min initiation rate (RNA polymerase molecules per minute)')
sp.start.kinit*60

print('120 initation rate (RNA polymerase molecules per minute)')
sp.start.kinit*60*sp.change.kinit/sp.start.kinit


# plot the steady state profile
# set to zero and solve
# dP = kinit - (kpre + krel)*P
# dB = (krel)*P - kelong*B
start.pause.peak = kinit/(kpre + krel)
start.body.peak = ((krel)*start.pause.peak)/kelong

change.pause.peak = sp.change.kinit/(kpre + sp.change.krel)
change.body.peak = ((sp.change.krel)*change.pause.peak)/kelong

input.sp.60.body = data.frame(c(1:length(seq(0, 0.1, by = 0.0001))),
                              seq(0, 0.1, by = 0.0001), start.body.peak)
colnames(input.sp.60.body) = c('iteration', 'Pause', 'Body')

input.sp.120.body = data.frame(c(1:length(seq(0, 0.1, by = 0.0001))),
                               seq(0, 0.1, by = 0.0001), change.body.peak)
colnames(input.sp.120.body) = c('iteration', 'Pause', 'Body')

# you get this error if your parameter space does not fit target data:
# Error in bodypeak + exp(1) : non-numeric argument to binary operator
peak.height.60.sp = pro.integrated.peak(input.sp.60.body,
                                        pause.height = start.pause.peak, tau = 10,
                                        time.char = '60 min')

peak.height.120.sp = pro.integrated.peak(input.sp.120.body,
                                         pause.height = change.pause.peak, tau = 10,
                                         time.char = '120 min')

plot.lattice.sp.model = rbind(peak.height.60.sp, peak.height.120.sp)
colnames(plot.lattice.sp.model) = c('V1', 'V2', 'V3')
plot.lattice.sp.model[,2] = as.numeric(as.character(plot.lattice.sp.model[,2]))
plot.lattice.sp.model[,3] = as.numeric(as.character(plot.lattice.sp.model[,3]))

#center on pause peak
```

```
pause.off = plot.lattice.sp.model[which.max(plot.lattice.sp.model[,2]),3]
plot.lattice.sp.model[,3] = plot.lattice.sp.model[,3] - pause.off

max( plot.lattice.sp.model[,2])+ max( plot.lattice.sp.model[,2]) * 0.07
#plot
plot.pro.simulation.composites.2(plot.lattice.sp.model, pause.offset = pause.off,
                                 trace.col = c('grey50', '#835b24'), ylim = c(0, 0.01968662),
                                filename = 'model_SP_60_120_min')
```

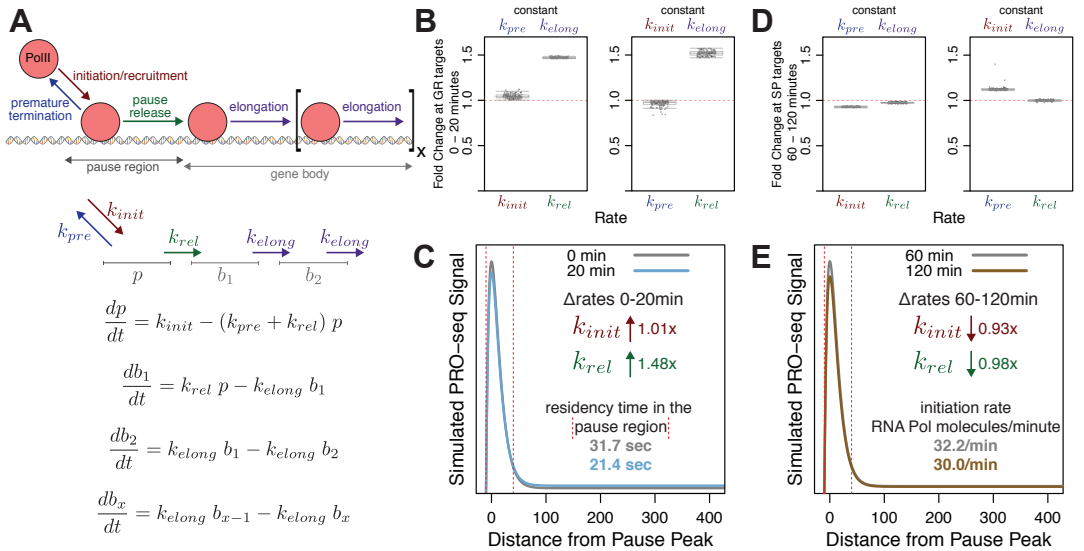


Figure 18: **GR preferentially regulates pause release and SP regulates initiation**. SP1 is known to stimulate initiation by interacting with TFIID (Gill *et al.*, 1994). Here we quantify the change in rate at the average SP target gene and employ constraints that are dependent upon premature pause release rate.

5 Conclusions

Genomic kinetic measurements of chromatin and nascent transcription can identify a repertoire of transcription factors that drive regulatory cascades. Moreover, we can link factor-bound regulatory elements to target genes. The motivation of this vignette is to provide a systematic framework to determine the step(s) in the transcription cycle that each factor regulates. To do this, we interpret changes in RNA polymerase density in the pause and gene body regions using a dynamic compartment model.

References

Ardehali MB, Lis JT (2009). "Tracking rates of transcription and splicing in vivo." *Nature structural & molecular biology*, **16**(11), 1123–1124.

Bailey TL, Williams N, Misleh C, Li WW (2006). "MEME: discovering and analyzing DNA and protein sequence motifs." *Nucleic acids research*, **34**(suppl 2), W369–W373.

Benner C, Konovalov S, Mackintosh C, Hutt KR, Stunnenberg R, Garcia-Bassets I (2013). "Decoding a signature-based model of transcription cofactor recruitment dictated by cardinal cis-regulatory elements in proximal promoter regions." *PLoS Genet*, **9**(11), e1003906.

Danko CG, Hah N, Luo X, Martins AL, Core L, Lis JT, Siepel A, Kraus WL (2013). "Signaling pathways differentially affect RNA polymerase II initiation, pausing, and elongation rate in cells." *Molecular cell*, **50**(2), 212–222.

Duarte FM, Fuda NJ, Mahat DB, Core LJ, Guertin MJ, Lis JT (2016). "Transcription factors GAF and HSF act at distinct regulatory steps to modulate stress-induced gene activation." *Genes & development*.

Gill G, Pascal E, Tseng ZH, Tjian R (1994). "A glutamine-rich hydrophobic patch in transcription factor Sp1 contacts the dTAFII110 component of the Drosophila TFIID complex and mediates transcriptional activation." *Proceedings of the National Academy of Sciences*, **91**(1), 192–196.

Hah N, Danko CG, Core L, Waterfall JJ, Siepel A, Lis JT, Kraus WL (2011). "A rapid, extensive, and transient transcriptional response to estrogen signaling in breast cancer cells." *Cell*, **145**(4), 622–634.

He X, Samee MAH, Blatti C, Sinha S (2010). "Thermodynamics-based models of transcriptional regulation by enhancers: the roles of synergistic activation, cooperative binding and short-range repression." *PLoS Comput Biol*, **6**(9), e1000935.

Jonkers I, Kwak H, Lis JT (2014). "Genome-wide dynamics of Pol II elongation and its interplay with promoter proximal pausing, chromatin, and exons." *Elife*, **3**, e02407.

Jonkers I, Lis JT (2015). "Getting up to speed with transcription elongation by RNA polymerase II." *Nature reviews Molecular cell biology*, **16**(3), 167–177.

Kent WJ, Sugnet CW, Furey TS, Roskin KM, Pringle TH, Zahler AM, Haussler D (2002). "The human genome browser at UCSC." *Genome research*, **12**(6), 996–1006.

Sathyan KM, McKenna BD, Anderson WD, Duarte FM, Core L, Guertin MJ (2019). "An improved auxin-inducible degron system preserves native protein levels and enables rapid and specific protein depletion." *Genes & development*, **33**(19-20), 1441–1455.

Scholes C, DePace AH, Sánchez Á (2017). "Combinatorial gene regulation through kinetic control of the transcription cycle." *Cell systems*, **4**(1), 97–108.

Vockley CM, D?Ippolito AM, McDowell IC, Majoros WH, Safi A, Song L, Crawford GE, Reddy TE (2016). "Direct GR binding sites potentiate clusters of TF binding across the human genome." *Cell*, **166**(5), 1269–1281.

Wang Z, Chu T, Choate LA, Danko CG (2019). "Identification of regulatory elements from nascent transcription using dREG." *Genome research*, **29**(2), 293–303.