T2_FK

March 29, 2025

[165]:
```
options(repr.plot.width=20, repr.plot.height=10)  # Ajuste dos gráficos
```

Instalando e carregando as bibliotecas utilizadas

[166]:
```
library(pacman)
p_load(ggplot2,forecast,dlm,numDeriv,plotly)
```

Carregando as funções auxiliares, que não podem ser mostras aqui ainda

[176]:
```
source('funcoes_auxiliares.R')
```

0.1 Modelo MNL

Simulando o y conforme a minha função do Trabalho 1:

[152]:
```
y_mnl <- simul_y_mnl(T=100,10,0.5,n_seed=1)
```

Utilizando o StructTs para estimar os hyperparâmetros via MLE:

[153]:
```
fit <- StructTS(y_mnl, "level")
```

Criando um dataframe com o tempo (1:T), a série simulada e o nível ajustado:

[154]:
```
df <- data.frame(
  time = 1:length(y_mnl),
  observed = y_mnl,
  estimated_level = fitted(fit)[, "level"]
)
```

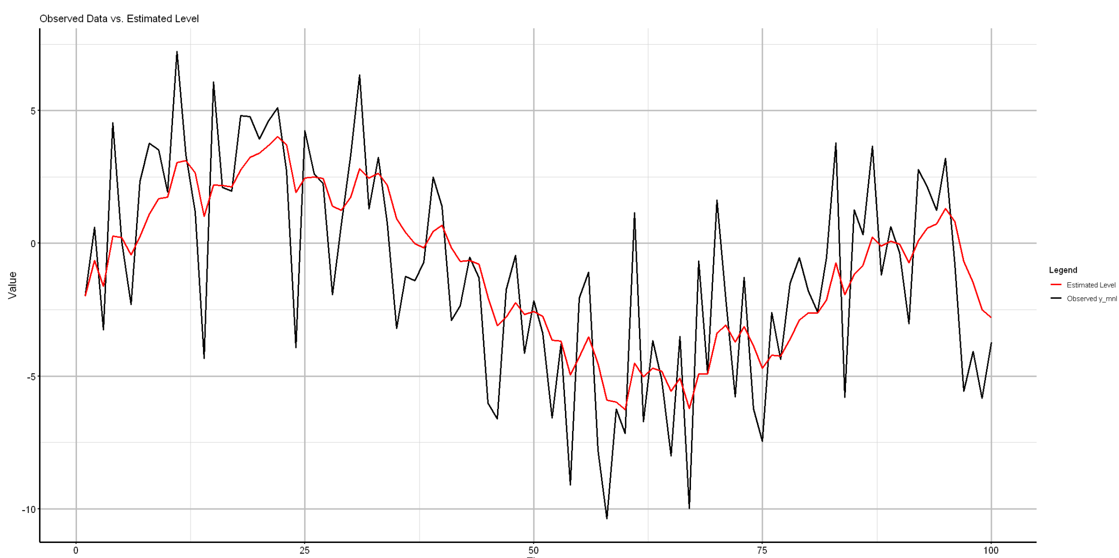
Utilizando o objeto 'fit' , podemos extrair as estimativas para os hyperparâmetros:

[155]:
```
# Extract estimated variances
sigma2_epsilon <- fit$coef["epsilon"]
sigma2_eta <- fit$coef["level"]
```

Podemos fazer o gráfico do nível estimado contra a série simulada utilizando o ggplot2

[156]:
```
ggplot() +
geom_line(data=df,aes(y = observed,x = time, color = "Observed y_mnl"),␣
 ↪linewidth = 1) +
```

```r
geom_line(data=df,aes(y = estimated_level,x = time, color = "Estimated Level"),
  linewidth = 1) +
scale_color_manual(values = c("Observed y_mnl" = "black", "Estimated Level" =
  "red")) +
labs(
title = "Observed Data vs. Estimated Level",
x = "Time",
y = "Value",
color = "Legend"
) +

theme_minimal()+

theme(
    panel.grid.major = element_line(color = "gray", linewidth = 1),
    panel.grid.minor = element_line(color = "lightgray", linewidth = 0.5),
    axis.line = element_line(color = "black", linewidth = 1),  # Make axis lines
  thicker and black
    axis.ticks = element_line(color = "black", linewidth = 0.8),  # Make ticks
  more visible
    axis.text = element_text(size = 12, color = "black"),  # Adjust axis text
  size and color
    axis.title = element_text(size = 14, color = "black")  # Adjust axis title
  size and color
    )
```



Recuperando o valor dos hyperparâmetros:

[157]:
```
# Print estimated variances
cat("Estimated Observation Noise Variance (sigma^2_epsilon):", sigma2_epsilon,␣
 ↪"\n")
cat("Estimated State Noise Variance (sigma^2_eta):", sigma2_eta, "\n")
```

```
Estimated Observation Noise Variance (sigma^2_epsilon): 7.7368
Estimated State Noise Variance (sigma^2_eta): 0.5600975
```

Será que o estimador MLE é consistente? Vamos aumentar o T e ver se as estimativas melhoram.

[158]:
```
y_mnl <- simul_y_mnl(T=1000,10,0.5,1)
```

[159]:
```
fit <- StructTS(y_mnl, "level")
```

[160]:
```
df <- data.frame(
  time = 1:length(y_mnl),
  observed = y_mnl,
  estimated_level = fitted(fit)[, "level"]
)
```

[161]:
```
sigma2_epsilon <- fit$coef["epsilon"]
sigma2_eta <- fit$coef["level"]
```

[162]:
```
cat("Estimated Observation Noise Variance (sigma^2_epsilon):", sigma2_epsilon,␣
 ↪"\n")
cat("Estimated State Noise Variance (sigma^2_eta):", sigma2_eta, "\n")
```

```
Estimated Observation Noise Variance (sigma^2_epsilon): 10.83406
Estimated State Noise Variance (sigma^2_eta): 0.6076557
```

Parece que com mais observações, as estimativas ficam mais próximas do valor verdadeiro.

No trabalho 1, conseguimos achar a sequencia a_t(1:T) e F_t[1:T] a partir de um chute incial para a_0 e p_0.

Pergunta: será que se usarmos essa função do trabalho 2 utilizando as variâncias estimadas, conseguimos reproduzir a série do nível estimado?

Novamente:

[163]:
```
y_mnl <- simul_y_mnl(T=1000,1,0.5,1)

fit <- StructTS(y_mnl, "level")
```

Utilizando a função da questão 2 do trabalho 1:

[177]:
```
fitted<-mnl_fk(T=1000,y_mnl,fit$coef["epsilon"],fit$coef["level"],a0=fit$model0$a,p0=fit$model0
```

[178]:
```
df_fitted_mnl <- data.frame(manual = fitted$a,structts=fit$fitted[,'level'])
```

[179]:
```
tail(df_fitted_mnl)
```

A data.frame: 6 × 2

	manual <dbl>	structts <dbl>
995	-12.96527	-12.96527
996	-13.12028	-13.12028
997	-13.11330	-13.11330
998	-13.16258	-13.16258
999	-13.36646	-13.36646
1000	-13.17953	-13.17953

[181]:
```
head(df_fitted_mnl)
```

A data.frame: 6 × 2

	manual <dbl>	structts <dbl>
1	-0.6264538	-0.6264538
2	0.3378710	0.3378710
3	-0.1941920	-0.1941920
4	0.8891581	0.8891581
5	0.7917861	0.7917861
6	-0.4389378	-0.4389378

0.2 MTL

Definindo os parâmetros:

[200]:
```
T <- 100
sigma2_epsilon <- 1
sigma2_eta <- 1
sigma2_qsi <- 0.1
n_seed <- 42
```

Simulando a série:

[201]:
```
y_mtl <- simul_y_mtl(T, sigma2_eta=sigma2_eta, sigma2_qsi=sigma2_qsi,␣
↪sigma2_epsilon=sigma2_epsilon, n_seed=n_seed)
```

estimando os hyperparâmtros usando MLE. Novo argumento: 'Trend'

[202]:
```
fit_mtl <- StructTS(y_mtl, "trend")
```

Montando um dataframe com a séries:

[203]:
```
df_mtl <- data.frame(
  time = 1:T,
  observed = y_mtl,
  adjusted_level = fit_mtl$fitted[, "level"],
  adjusted_trend = fit_mtl$fitted[, "slope"]
)
```

Fazendo os gráficos usando ggplot2:

[226]:

```
ggplot() +
  geom_line(data=df_mtl,aes(x = time,y = observed, color = "Observed"), size =␣
↪1) +
  geom_line(data=df_mtl,aes(x = time,y = adjusted_level, color = "Adjusted␣
↪Level"), size = 1,alpha=0.6) +
  scale_color_manual(values = c("Observed" = "black", "Adjusted Level" =␣
↪"blue")) +
  labs(
    title = "Observed vs Adjusted Level StructTS (MTL)",
    x = "Time",
    y = "Value",
    color = "Legend"
  ) +
  theme_minimal()+
```

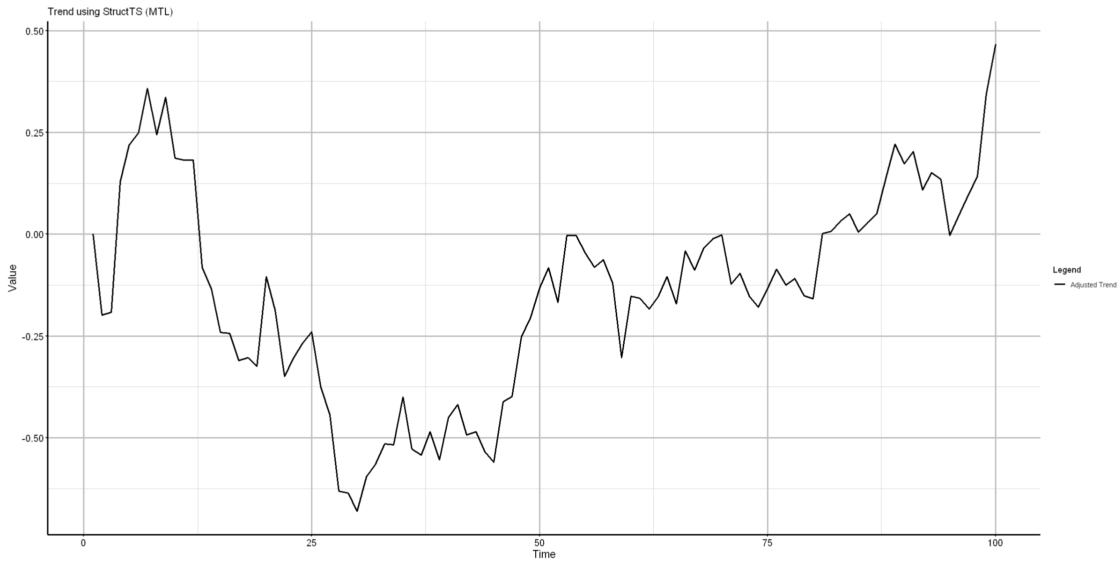
```
Error in parse(text = input): <text>:12:0: unexpected end of input
10:   ) +
11:   theme_minimal()+
   ^
Traceback:
```

[230]:

```
ggplot() +
  geom_line(data=df_mtl,aes(x = time,y = adjusted_trend, color = "Adjusted␣
↪Trend"), size = 1) +
  scale_color_manual(values = c("Adjusted Trend" = "black")) +
  labs(
    title = "Trend using StructTS (MTL)",
    x = "Time",
    y = "Value",
    color = "Legend"
  ) +
  theme_minimal()+

theme(
    panel.grid.major = element_line(color = "gray", linewidth = 1),
    panel.grid.minor = element_line(color = "lightgray", linewidth = 0.5),
    axis.line = element_line(color = "black", linewidth = 1),  # Make axis lines␣
↪thicker and black
    axis.ticks = element_line(color = "black", linewidth = 0.8),  # Make ticks␣
↪more visible
    axis.text = element_text(size = 12, color = "black"),  # Adjust axis text␣
↪size and color
    axis.title = element_text(size = 14, color = "black")  # Adjust axis title␣
↪size and color
```

```
    )
```

```
Don't know how to automatically pick scale for object of type
<ts>. Defaulting to continuous.
```



Valores dos parâmetros:

```
[212]: cat("Estimated Observation Noise Variance (sigma^2_epsilon):",
         fit_mtl$coef["epsilon"], "\n")
       cat("Estimated State Noise Variance (sigma^2_eta):", fit_mtl$coef["level"], "\n")
       cat("Estimated State Noise Variance (sigma^2_qsi):", fit_mtl$coef["slope"], "\n")
```

```
Estimated Observation Noise Variance (sigma^2_epsilon): 0.9959496
Estimated State Noise Variance (sigma^2_eta): 1.17139
Estimated State Noise Variance (sigma^2_qsi): 0.006570937
```

Agora vamos fazer previsões e suavização:

```
[213]: y_mtl <- simul_y_mtl(T=100,10,0.1,1,n_seed=1)
```

Ajustando o modelo com StructTS

```
[214]: fit <- StructTS(y_mtl, type = "trend")
```

definindo o número de passos a frente:

```
[216]: h <- 20
```

Criando o objeto com as previsões:

```
[221]: forecast_obj <- forecast(fit, h = h, level = 90)
```

Fazendo a suavização

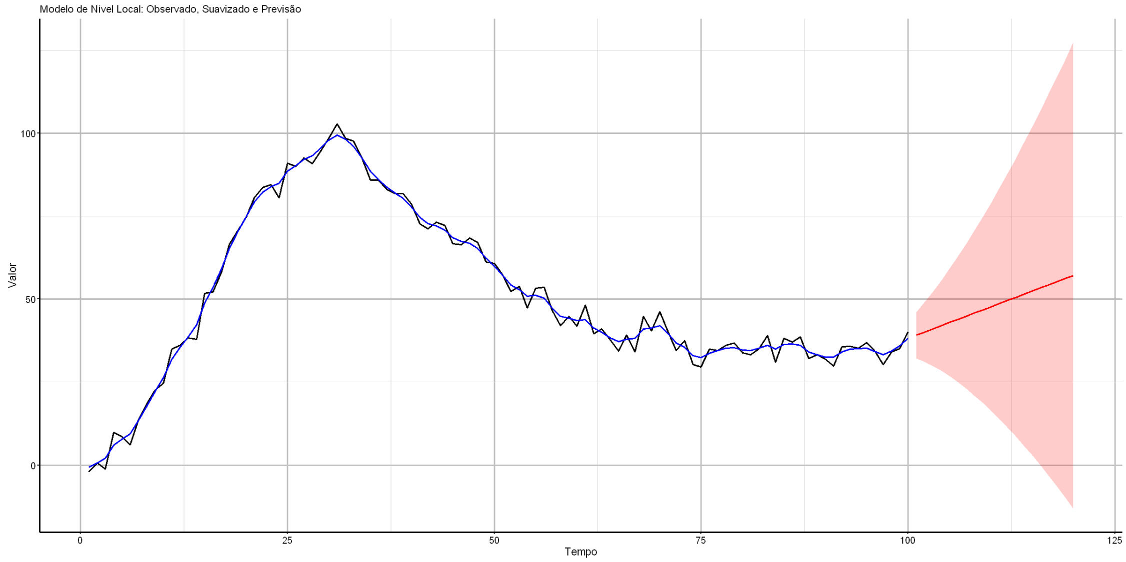
[222]:
```r
smoothed <- tsSmooth(fit)
```

Criando dataframe para o ggplot

[223]:
```r
df <- data.frame(
  Tempo = 1:T,
  Observado = y_mtl,
  Suavizado = smoothed[, 1]
)
```

[224]:
```r
df_pred <- data.frame(
  Tempo = (T + 1):(T + h),
  Previsao = forecast_obj$mean,
  Lower = forecast_obj$lower[, 1],
  Upper = forecast_obj$upper[, 1]
)
```

[231]:
```r
ggplot() +
  geom_line(data = df, aes(x = Tempo, y = Observado), color = "black", size = 1)␣
↪+
  geom_line(data = df, aes(x = Tempo, y = Suavizado), color = "blue", size = 1) +
  geom_line(data = df_pred, aes(x = Tempo, y = Previsao), color = "red", size =␣
↪1) +
  geom_ribbon(data = df_pred, aes(x = Tempo, ymin = Lower, ymax = Upper), fill =␣
↪"red", alpha = 0.2) +
  labs(title = "Modelo de Nível Local: Observado, Suavizado e Previsão",
       x = "Tempo", y = "Valor") +
  theme_minimal()+

theme(
    panel.grid.major = element_line(color = "gray", linewidth = 1),
    panel.grid.minor = element_line(color = "lightgray", linewidth = 0.5),
    axis.line = element_line(color = "black", linewidth = 1),  # Make axis lines␣
↪thicker and black
    axis.ticks = element_line(color = "black", linewidth = 0.8),  # Make ticks␣
↪more visible
    axis.text = element_text(size = 12, color = "black"),  # Adjust axis text␣
↪size and color
    axis.title = element_text(size = 14, color = "black")  # Adjust axis title␣
↪size and color
    )
```



1 DLM

Como sempre fazemos :

[243]:
```
T=100
```

[244]:
```
y_mnl <- simul_y_mnl(T=T,1,0.5,1)
```

The function dlmModPoly in the dlm package in R creates a state-space model for a polynomial trend process. It helps define local level (MNL) and local trend (MTL) models by setting up the system matrices required for the Kalman filter.

[245]:
```
build_mnl <- function(theta) {
    dlmModPoly(order = 1, dV = theta[1], dW = theta[2])
}
```

estimando o modelo via MLE

[246]:
```
fit <- dlmMLE(y_mnl, parm = c(100, 2), build_mnl, lower = rep(1e-4, 2))
```

parm:Initial values for the parameters to be estimated

lower: constraint in param values

Recuperando os valores estimados dos hyperparâmetros:

[247]:
```
mod_mnl <- build_mnl(fit$par)
drop(V(mod_mnl))
drop(W(mod_mnl))
```

0.718626480364571

0.436154015222723

The inverse of the Hessian matrix of the negative loglikelihood function evaluated at the MLEs is, by standard maximum likelihood theory, an estimate of the asymptotic variance matrix of the maximum likelihood estimators

obtendo a hessiana avaliada no MLE

[249]:
```
hs <- hessian(function(x) dlmLL(y_mnl, build_mnl(x)), fit$par)
```

Checando se é positiva definida

[250]:
```
all(eigen(hs, only.values = TRUE)$values > 0)
```

TRUE

Calculando o Inverso da Hessiana

[251]:
```
aVar <- solve(hs)
```

[252]:
```
cat("Standard Error of dV:", sqrt(diag(aVar)[1]), "\n")
cat("Standard Error of dW:", sqrt(diag(aVar)[2]), "\n")
```

```
Standard Error of dV: 0.1579155
Standard Error of dW: 0.1435128
```

Aplicando o filtro e suavização:

[256]:
```
smooth_mnl <- dlmSmooth(y_mnl, mod_mnl)
filter_mnl <- dlmFilter(y_mnl,mod_mnl)
```

Extração dos estados suavizados

[258]:
```
mu_hat <- drop(smooth_mnl$s)
```

Gerando as previsões com dlmForecast

[259]:
```
fore_mnl <- dlmForecast(filter_mnl, nAhead = 10)
```

matriz de valores esperados para futuras obs

[260]:
```
f <- fore_mnl$f
```

lista de variâncias de futuras observações

[261]:
```
Q <- fore_mnl$Q
```

Calculando o intervalo preditivo (banda de 50%)

[262]:
```
hwidth <- qnorm(0.25, lower = FALSE) * sqrt(unlist(Q))
```

Criando o intervalo de previsão

[263]:
```
lower <- f - hwidth
upper <- f + hwidth
```

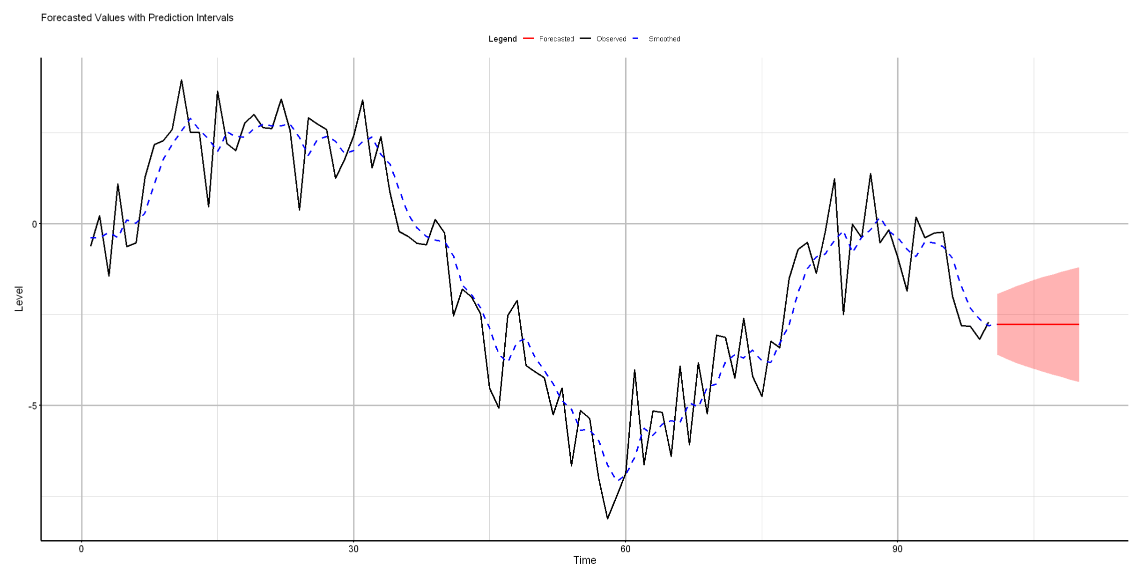
Criar uma data frame para facilitar o uso no ggplot2

[264]:
```
df_forecast <- data.frame(
  Time = (length(y_mnl) + 1):(length(y_mnl) + 10),
  Forecasted = f,
  Lower = lower,
  Upper = upper
)
```

Criando o gráfico

[267]:
```
ggplot(df_forecast, aes(x = Time)) +
  # Adiciona as previsões como uma linha vermelha
  geom_line(aes(y = Forecasted, color = "Forecasted"), size = 1) +

  # Adiciona as bandas de previsão (intervalo de 50%)
  geom_ribbon(aes(ymin = Lower, ymax = Upper), fill = "red", alpha = 0.3) +

  # Adiciona as observações reais da série (y_mnl)
  geom_line(data = data.frame(Time = 1:length(y_mnl), Observed = y_mnl),
            aes(y = Observed, color = "Observed"), size = 1, linetype = "solid")␣
↪+

  # Adiciona a linha de suavização (suavizado)
  geom_line(data = data.frame(Time = time(smooth_mnl$s), Smoothed =␣
↪smooth_mnl$s),
            aes(y = Smoothed, color = "Smoothed"), size = 1, linetype =␣
↪"dashed") +

  # Personalizando os eixos e título
  labs(
    title = "Forecasted Values with Prediction Intervals",
    x = "Time",
    y = "Level",
    color = "Legend"
  ) +

  # Customizando a paleta de cores
  scale_color_manual(values = c("Forecasted" = "red", "Observed" = "black",␣
↪"Smoothed" = "blue")) +

  # Tema minimalista
  theme_minimal() +
    theme(
    panel.grid.major = element_line(color = "gray", linewidth = 1),
```

```
    panel.grid.minor = element_line(color = "lightgray", linewidth = 0.5),
    axis.line = element_line(color = "black", linewidth = 1),  # Make axis lines␣
  ↪thicker and black
    axis.ticks = element_line(color = "black", linewidth = 0.8),  # Make ticks␣
  ↪more visible
    axis.text = element_text(size = 12, color = "black"),  # Adjust axis text␣
  ↪size and color
    axis.title = element_text(size = 14, color = "black"),  # Adjust axis title␣
  ↪size and color
    legend.position = "top")
```

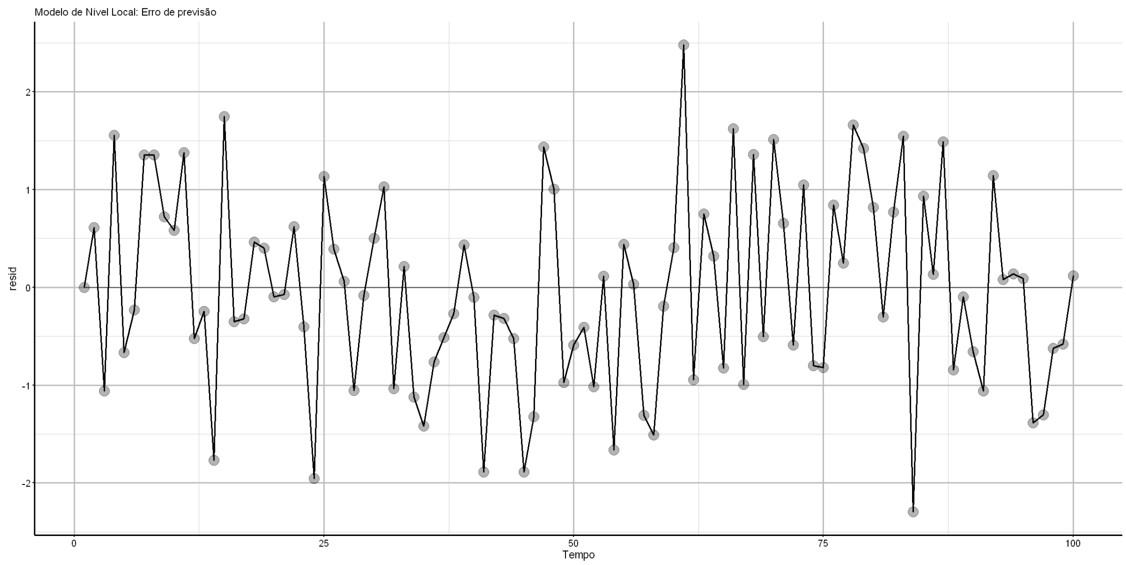


Análise dos resíduos

[268]:
```
df_residual <- data.frame(resid = residuals(filter_mnl, sd =␣
  ↪FALSE),zero=rep(0,T),tempo<-seq(1:T))
```

[271]:
```
ggplot() +
geom_line(data = df_residual, aes(x = tempo, y = resid), color = "black", size =␣
  ↪1) +
geom_line(data=df_residual, aes(x=tempo,y=zero))+
geom_point(data=df_residual,aes(x = tempo , y = resid), color = "black", size =␣
  ↪6 ,alpha=0.3) +

labs(title = "Modelo de Nível Local: Erro de previsão",
   x = "Tempo", y = "resid") +
theme_minimal()+
theme(panel.grid.major = element_line(color = "gray", linewidth = 1),
panel.grid.minor = element_line(color = "lightgray", linewidth = 0.5),
```

```
axis.line = element_line(color = "black", linewidth = 1),  # Make axis lines␣
↪thicker and black
axis.ticks = element_line(color = "black", linewidth = 0.8),  # Make ticks more␣
↪visible
axis.text = element_text(size = 12, color = "black"),  # Adjust axis text size␣
↪and color
axis.title = element_text(size = 14, color = "black"))
```



testando se os resíduos são normais:

[274]:
```
shapiro.test(df_residual$resid)
cat("media_residuos:",mean(df_residual$resid),'\n')
cat("var_residuos:",var(df_residual$resid),'\n')
```

```
	Shapiro-Wilk normality test

data:  df_residual$resid
W = 0.98873, p-value = 0.5638


media_residuos: -0.03384876
var_residuos: 0.9988415
```

2 Implementando a MLE

o objetivo dessa secção é comparar a estimação manual e a minha implementação de estimação

Gerando a série e fazendo a estimação usando pacote

[275]:
```
T=1000
y_mnl <- simul_y_mnl(T=T,1,0.5,1)
fit <- dlmMLE(y_mnl, parm = c(100, 2), build_mnl, lower = rep(1e-4, 2))
```

Estimação manual

definindo o espaço paramétrico:

[276]:
```
sigma_epsilon2_vals <- seq(0.1,4, length.out = 100)
sigma_eta2_vals <- seq(0.1, 4, length.out = 100)
```

função de verossimilhança:

[280]:
```
logLik_mnl <- function(theta) {

  #definindo parâmetros
  sigma_epsilon2 <- theta[1]
  sigma_eta2 <- theta[2]

  # fazendo a filtragem usando minha função do trabalho 1
  fk_results <- mnl_fk(T = length(y_mnl), sigma_epsilon2, sigma_eta2, a0 = 0, p0␣
↪= 100, y_mnl)

  # extraindo (v_t) e a variância (F_t)
  v_t <- fk_results$v_t
  F_t <- fk_results$F

  # Computando a verossimilhança
  n <- length(y_mnl)
  logLik <- - (n / 2) * log(2 * pi) - 0.5 * sum(log(abs(F_t))) - 0.5 *␣
↪sum((v_t^2) / F_t)

  return(logLik)
}
```

Calculando a verossimilhança para todas as combinações dos hyperparâmetros:

[277]:
```
likelihood_matrix <- outer(sigma_epsilon2_vals, sigma_eta2_vals,
                           Vectorize(function(se, sw) logLik_mnl(c(se, sw))))
```

transformando em um df para usar o ggplot2

[282]:
```
likelihood_df <- expand.grid(sigma_epsilon2 = sigma_epsilon2_vals,
                             sigma_eta2 = sigma_eta2_vals)
```

adicionando a verossimilhança no df

[284]:
```
likelihood_df$logLik <- as.vector(likelihood_matrix)
```

comparando os resultado

pacote:

[286]:
```
cat("Estimated sigma^2_epsilon:", fit$par[1], "\n")
cat("Estimated sigma^2_eta:", fit$par[2], "\n")
```

```
Estimated sigma^2_epsilon: 1.074177
Estimated sigma^2_eta: 0.5647697
```

manual:

[289]:
```
mle_estimates <- likelihood_df %>%
  filter(logLik == max(logLik)) %>%
  select(sigma_epsilon2, sigma_eta2, logLik)

names(mle_estimates)<-c("sigma2_epsilon",'sigma2_eta','Max LogLikelihood')

cat("Estimated sigma^2_epsilon:", unlist(mle_estimates[1]), "\n")
cat("Estimated sigma^2_eta:", unlist(mle_estimates[2]), "\n")
```

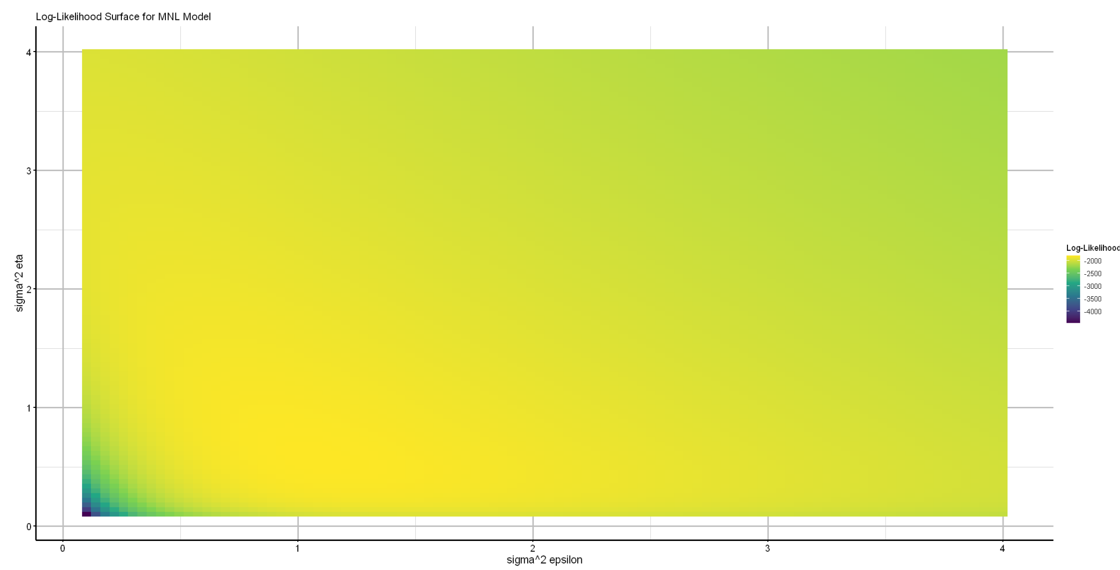
```
Estimated sigma^2_epsilon: 1.084848
Estimated sigma^2_eta: 0.5727273
```

[290]:
```
fig <- plot_ly(
  x = unique(likelihood_df$sigma_epsilon2),  # X-axis (sigma_epsilon2 values)
  y = unique(likelihood_df$sigma_eta2),      # Y-axis (sigma_eta2 values)
  z = matrix(likelihood_df$logLik,
             nrow = length(unique(likelihood_df$sigma_epsilon2)),
             ncol = length(unique(likelihood_df$sigma_eta2))),  # Reshape␣
 ↪log-likelihood data
  type = "surface",
  colorscale = "Inferno"  # Makes high values vivid red
)

fig <- fig %>%
  layout(
    title = "Log-Likelihood Surface for MNL Model",
    scene = list(
      xaxis = list(title = expression(sigma[epsilon]^2)),
      yaxis = list(title = expression(sigma[eta]^2)),
      zaxis = list(title = "Log-Likelihood")
    )
  )


fig
```

```
HTML widgets cannot be represented in plain text (need html)
```

Heatmap:

[293]:
```
ggplot(likelihood_df, aes(x = sigma_epsilon2, y = sigma_eta2, fill = logLik)) +
  geom_tile() +
  scale_fill_viridis_c()+
  labs(title = "Log-Likelihood Surface for MNL Model",
       x = 'sigma^2 epsilon',
       y = 'sigma^2 eta',
       fill = "Log-Likelihood") +
      theme_minimal()+
    theme(panel.grid.major = element_line(color = "gray", linewidth = 1),
    panel.grid.minor = element_line(color = "lightgray", linewidth = 0.5),
    axis.line = element_line(color = "black", linewidth = 1),  # Make axis lines thicker and black
    axis.ticks = element_line(color = "black", linewidth = 0.8),  # Make ticks more visible
    axis.text = element_text(size = 12, color = "black"),  # Adjust axis text size and color
    axis.title = element_text(size = 14, color = "black"))
```



[296]:
```
jupyter nbconvert --to pdf T2_Fk.pynb
```

```
Error in parse(text = input): <text>:1:9: unexpected symbol
1: jupyter nbconvert
            ^

Traceback:
```

[ ]:
```
```