SpaceAndTime

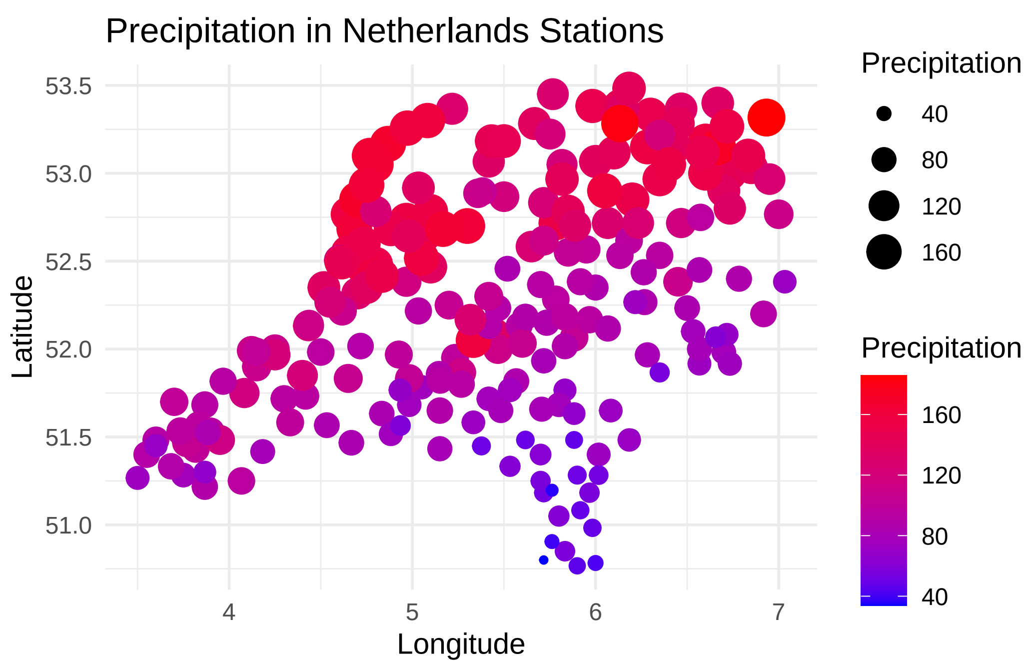
Table of contents

TODO check if I checked all the residuals for all models

Question 1 Spatial modelling Kingdom of the Netherlands

1 a)

```
ggplot(data = netherlandsDF) + geom_point(aes(x = longitude, y = latitude,
    size = precip, color = precip)) + scale_color_continuous(low = "blue",
    high = "red") + labs(title = "Precipitation in Netherlands Stations",
    x = "Longitude", y = "Latitude", size = "Precipitation", color = "Precipitation") +
    theme_minimal()
```



From what we can see from the data it does seem to be spatially correlated as we can that the Dutch provinces of north Holland, Friesland and Groningen has higher precipitation and as we go south the precipitation does decrease as we can see from the Dutch provinces of Zeeland, north Brabant and Limburg where precipitation is significantly lower than their northern counterparts.

From this data, latitude seems to be the biggest factor in the variation of the precipitation as the longitude only suggests some slight variations in the data.

```
# Create geodata object
precipitationNetherland_geoR = as.geodata(netherlandsDF, coords.col = c("longitude",
    "latitude"), data.col = "precip")

summary(precipitationNetherland_geoR)
```

```
Number of data points: 220

Coordinates summary
    longitude latitude
min     3.500   50.767
max     7.033   53.483

Distance summary
     min      max
0.001000 3.998498

Data summary
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
 33.9000  80.8500 100.1500 106.5705 137.3500 185.6000
```

As we can see from the numerical summary of the data the median is different from the mean, which indicates it is not a symmetric distribution of data points and is instead positively skewed since the mean is bigger than the median. As such there are more values on the left side of the distribution.

1 b)

```
# set seed for reproducibility
set.seed(26041999)

# Select 3 random rows from the data frame
```

```r
randomRowsPrecipitation = netherlandsDF %>%
    sample_n(3)

# Add a new column with labels
randomRowsPrecipitation$label = c("A", "B", "C")

# Print the randomly selected rows
randomRowsPrecipitation
```

```
# A tibble: 3 x 5
  station_name   longitude latitude precip label
  <chr>              <dbl>    <dbl>  <dbl> <chr>
1 NIJKERK             5.47     52.2   89.1 A
2 WOLPHAARTSDIJK      3.73     51.5   95.9 B
3 EEXT                6.73     53    147.  C
```

```r
# Remove the selected rows from the original dataset
netherlandsDF_filtered = netherlandsDF %>%
    anti_join(randomRowsPrecipitation)
```

```
Joining, by = c("station_name", "longitude", "latitude", "precip")
```

```r
# Print the resulting dataframe
netherlandsDF_filtered
```

```
# A tibble: 217 x 4
   station_name      longitude latitude precip
   <chr>                 <dbl>    <dbl>  <dbl>
 1 WEST TERSCHELLING      5.22     53.4  130.
 2 GRONINGEN-1            6.6      53.2  157.
 3 HOORN                  5.07     52.6  146.
 4 HOOFDDORP              4.7      52.3  130.
 5 WINTERSWIJK            6.7      52.0   77.7
 6 KERKWERVE              3.87     51.7   91.8
 7 WESTDORPE-1            3.87     51.2   87.7
 8 OUDENBOSCH             4.53     51.6   84.2
 9 ROERMOND               5.97     51.2   56.4
10 PETTEN                 4.65     52.8  158.
# ... with 207 more rows
```

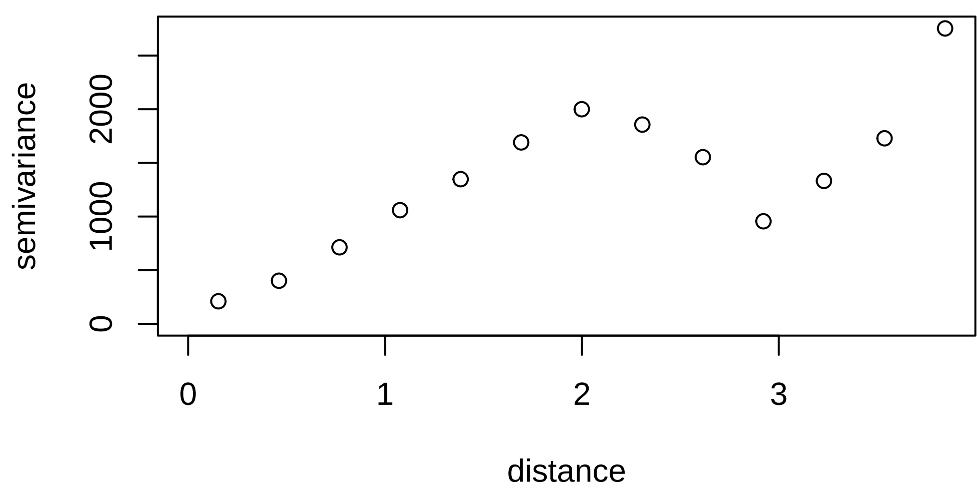
```r
## recreate the geoData object with the new filtered dataframe
precipitationNetherland_geoR = as.geodata(netherlandsDF_filtered, coords.col = c("longitud
    "latitude"), data.col = "precip")
```

1 c)

```r
# # Calculate empirical variogram
variogramPrecipitationNetherlands = variog(precipitationNetherland_geoR)
```

```
variog: computing omnidirectional variogram
```

```r
# Plot empirical variogram
plot(variogramPrecipitationNetherlands)
```




```r
variogramPrecipitationNetherlands$n
```

```
 [1] 1069 2482 3384 3834 3647 3221 2534 1541  787  468  318  133   17
```

From the plotted variogram we can see there a very clear need for a nugget as there is a non-zero value around zero distance, this values seems to be around 75 to 100 at the zero distance from how much is it decreasing.

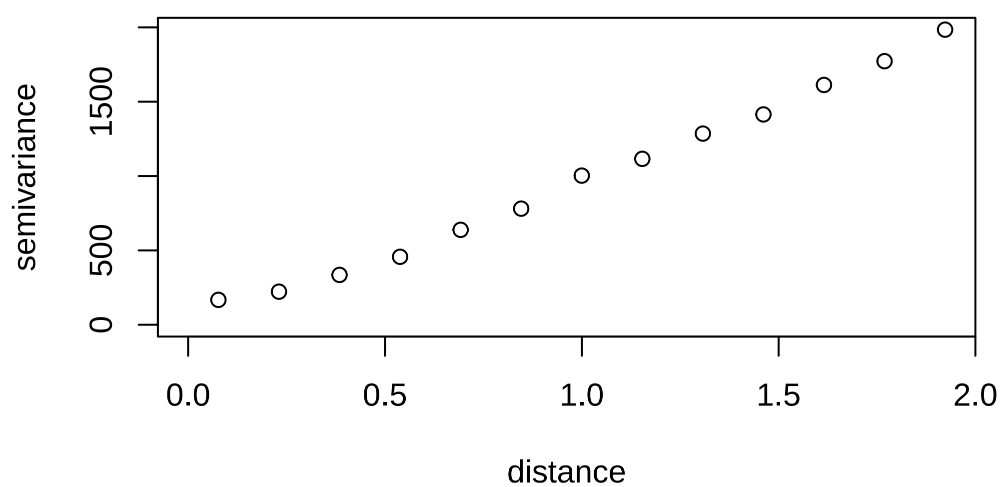
The semi variance continuous to increase with distance till around the distance of 2 degrees distance wise, after this there is a decrease in variance that is not representative of the data as we are more and more uncertain the further we are from our known points, as such we will choose the distance of two as the cut off for the maximum distance.

we know change the maximum distance change and recut our previous variogram.

```
variogramPrecipitationNetherlands = variog(precipitationNetherland_geoR,
    option = "bin", max.dist = 2)
```

```
variog: computing omnidirectional variogram
```

```
plot(variogramPrecipitationNetherlands)
```



As we can see from the newly updated variogram the increase is almost linear with a curve near 0 where we can see the need for the nugget.

1 d)

Now that we have the variogram we will start by fitting a model to estimate the covariance via weighted least squares. Fitting this variogram we get the estimated values of σ^2, ϕ and τ^2 also known as the nugget

We will first start with the default Matrén = 0,5 which is equivalent to an exponential as the form observed in the previous variogram seems to not fully linear and therefore require the curvature from a function like the exponential function to account for the behaviour at the near 0 distance.

From this we will try different models to search for the model with the best fit.

```
# ?variofit thau = nugget variability sigmasq = if the model can
# capture more or less of the total variability phi = if the
# correlation extends over a bigger or smaller distance loss value =
# goodness of fit (smaller means better fit)

krigingVariogramFittedDefault = variofit(variogramPrecipitationNetherlands,
    nugget = 85)
```

```
variofit: covariance model used is matern
variofit: weights used: npairs
variofit: minimisation function used: optim
```

```
Warning in variofit(variogramPrecipitationNetherlands, nugget = 85): initial
values not provided - running the default search
```

```
variofit: searching for best initial value ... selected values:
              sigmasq   phi    tausq kappa
initial.value "1984.67" "1.54" "85"  "0.5"
status        "est"     "est"  "est" "fix"
loss value: 818327620.928191
```

```
krigingVariogramFittedDefault
```

```
variofit: model parameters estimated by WLS (weighted least squares):
covariance model is: matern with fixed kappa = 0.5 (exponential)
parameter estimates:
     tausq    sigmasq        phi
      0.00 2561207.09    2590.45
```

```
Practical Range with cor=0.05 for asymptotic range: 7760.294

variofit: minimised weighted sum of squares = 35360382
```

Now we will increase the kappa of the Matrén to see if the increased flexibility and smoothness leads to a better fit

```r
krigingVariogramFittedMatrén1.5 = variofit(variogramPrecipitationNetherlands,
    kappa = 1.5, nugget = 85)
```

```
variofit: covariance model used is matern
variofit: weights used: npairs
variofit: minimisation function used: optim
```

```
Warning in variofit(variogramPrecipitationNetherlands, kappa = 1.5, nugget =
85): initial values not provided - running the default search
```

```
variofit: searching for best initial value ... selected values:
              sigmasq   phi    tausq kappa
initial.value "1984.67" "0.62" "85"  "1.5"
status        "est"     "est"  "est" "fix"
loss value: 190482114.063677
```

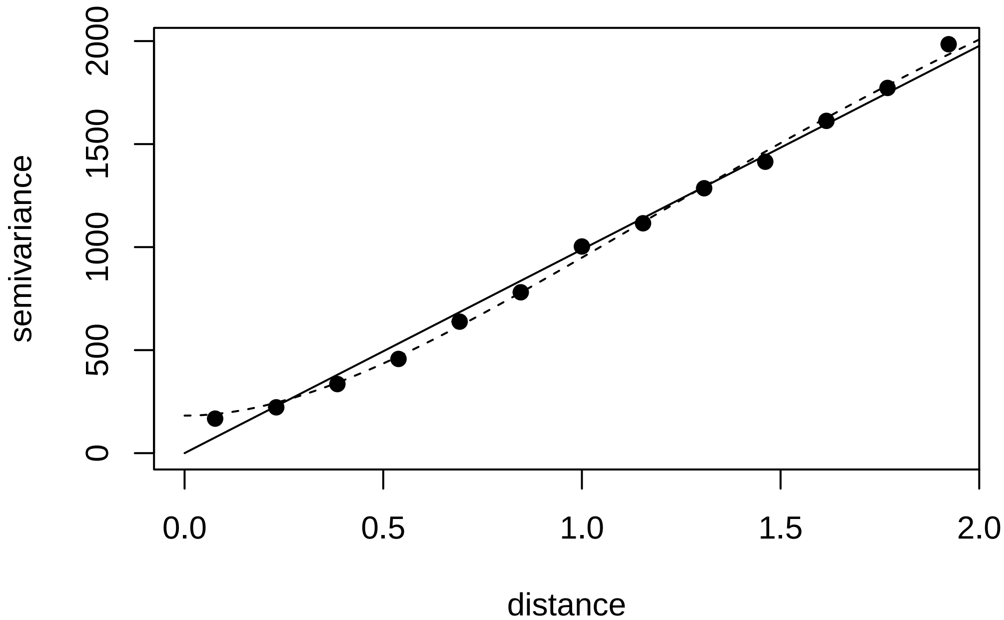
```r
krigingVariogramFittedMatrén1.5
```

```
variofit: model parameters estimated by WLS (weighted least squares):
covariance model is: matern with fixed kappa = 1.5
parameter estimates:
    tausq   sigmasq       phi
 182.0638 3461.1492    1.1316
Practical Range with cor=0.05 for asymptotic range: 5.368367

variofit: minimised weighted sum of squares = 15435206
```

We will first visually compare these 2 models to see which one has a better

```r
par(mar = c(4, 4, 2, 2))
plot(variogramPrecipitationNetherlands, pch = 19)
lines(krigingVariogramFittedDefault)
```

```r
lines(krigingVariogramFittedMatrén1.5, lty = 2)
```




```r
# lines(krigingVariogramFittedMatrén2.5, lty = 3)
```

Immediately we can see that the extra flexibility of the Matrén 1,5 not only better follows the actual data, it actually accounts correctly for the initial variance from the nugget which the Matrén 0,5 does not as it simply decreases to 0.

We now will test if any additional flexibility changes can improve the model fit

We will first try to again increase the kappa to see if the model again benefits from the extra flexibility

```r
krigingVariogramFittedMatrén2.0 = variofit(variogramPrecipitationNetherlands,
    kappa = 2, nugget = 85)
```

```
variofit: covariance model used is matern
variofit: weights used: npairs
variofit: minimisation function used: optim

Warning in variofit(variogramPrecipitationNetherlands, kappa = 2, nugget = 85):
initial values not provided - running the default search
```

```
variofit: searching for best initial value ... selected values:
              sigmasq   phi    tausq    kappa
initial.value "1984.67" "0.62" "198.47" "2"
status        "est"     "est"  "est"    "fix"
loss value: 227233080.389154
```

```
krigingVariogramFittedMatrén2.0
```

```
variofit: model parameters estimated by WLS (weighted least squares):
covariance model is: matern with fixed kappa = 2
parameter estimates:
    tausq   sigmasq       phi
 197.6127 3037.3813    0.8386
Practical Range with cor=0.05 for asymptotic range: 4.502076

variofit: minimised weighted sum of squares = 18197964
```

Here we will instead see if the model will benefit instead form a cut of flexibility to make it less smooth

```
krigingVariogramFittedMatrén1.0 = variofit(variogramPrecipitationNetherlands,
    kappa = 1, nugget = 85)
```

```
variofit: covariance model used is matern
variofit: weights used: npairs
variofit: minimisation function used: optim
```

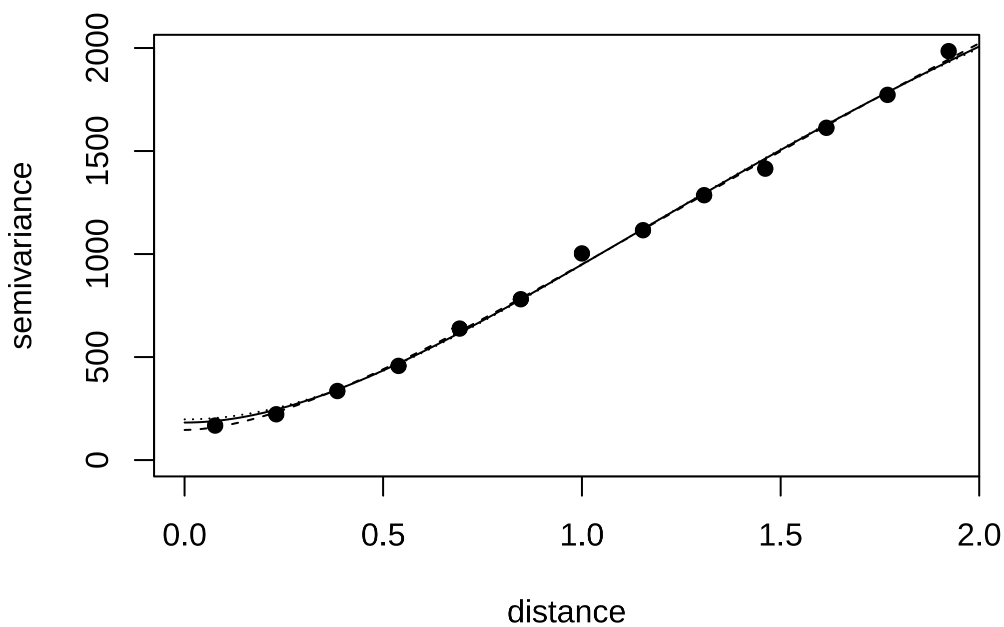
```
Warning in variofit(variogramPrecipitationNetherlands, kappa = 1, nugget = 85):
initial values not provided - running the default search
```

```
variofit: searching for best initial value ... selected values:
              sigmasq   phi    tausq    kappa
initial.value "1984.67" "0.92" "198.47" "1"
status        "est"     "est"  "est"    "fix"
loss value: 352001017.756763
```

```
krigingVariogramFittedMatrén1.0
```

```
variofit: model parameters estimated by WLS (weighted least squares):
covariance model is: matern with fixed kappa = 1
parameter estimates:
    tausq   sigmasq       phi
 146.1354 4832.1862    2.0470
Practical Range with cor=0.05 for asymptotic range: 8.185098

variofit: minimised weighted sum of squares = 12019899
```

```
par(mar = c(4, 4, 2, 2))
plot(variogramPrecipitationNetherlands, pch = 19)
lines(krigingVariogramFittedMatrén1.5)
lines(krigingVariogramFittedMatrén1.0, lty = 2)
lines(krigingVariogramFittedMatrén2.0, lty = 3)
```



As we can see from the new graph it does seem that actually a lower flexibility Matrén has a better fit since the extra flexibility near the start and end of the data points made the models deviate too much from the points.

1 e)

To fit a model using the maximum likelihood we will have to try multiple initial values to make sure this is indeed the maximum likelihood and not just a local maximum.

```
# ?likest

maximumLikelihoodNetherlandsInitial10.1 = likfit(precipitationNetherland_geoR,
    ini.cov.pars = c(10, 1))
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.
```

```
maximumLikelihoodNetherlandsInitial1.10 = likfit(precipitationNetherland_geoR,
    ini.cov.pars = c(1, 10))
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.
```

```
maximumLikelihoodNetherlandsInitial100.10 = likfit(precipitationNetherland_geoR,
    ini.cov.pars = c(100, 10))
```

```
---------------------------------------------------------------
```

```
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.
```

```r
maximumLikelihoodNetherlandsInitial10.100 = likfit(precipitationNetherland_geoR,
    ini.cov.pars = c(10, 100))
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.

WARNING: estimated range is more than 10 times bigger than the biggest distance between two p
 1) excluding spatial dependence if estimated sill is too low and/or
 2) taking trends (covariates) into account
```

```r
maximumLikelihoodNetherlandsInitial1.1 = likfit(precipitationNetherland_geoR,
    ini.cov.pars = c(1, 1))
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.
```

```r
maximumLikelihoodNetherlandsInitial1000.1000 = likfit(precipitationNetherland_geoR,
    ini.cov.pars = c(1000, 1000))
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.

WARNING: estimated range is more than 10 times bigger than the biggest distance between two p
 1) excluding spatial dependence if estimated sill is too low and/or
 2) taking trends (covariates) into account
```

```r
maximumLikelihoodNetherlandsInitial500.500 = likfit(precipitationNetherland_geoR,
    ini.cov.pars = c(500, 500))
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.

WARNING: estimated range is more than 10 times bigger than the biggest distance between two p
 1) excluding spatial dependence if estimated sill is too low and/or
 2) taking trends (covariates) into account
```

```r
maximumLikelihoodNetherlandsInitial10.1
```

```
likfit: estimated model parameters:
```

```
      beta      tausq    sigmasq        phi 
" 102.376" " 114.249" "3036.627" "   7.132" 
Practical Range with cor=0.05 for asymptotic range: 21.36583

likfit: maximised log-likelihood = -896.9
```

```r
maximumLikelihoodNetherlandsInitial1.10
```

```
likfit: estimated model parameters:
      beta      tausq    sigmasq        phi 
" 102.449" " 114.921" "4184.402" "   9.991" 
Practical Range with cor=0.05 for asymptotic range: 29.9294

likfit: maximised log-likelihood = -896.9
```

```r
maximumLikelihoodNetherlandsInitial100.10
```

```
likfit: estimated model parameters:
      beta      tausq    sigmasq        phi 
" 102.449" " 114.921" "4184.402" "   9.991" 
Practical Range with cor=0.05 for asymptotic range: 29.9294

likfit: maximised log-likelihood = -896.9
```

```r
maximumLikelihoodNetherlandsInitial10.100
```

```
likfit: estimated model parameters:
     beta     tausq   sigmasq       phi 
"  102.7" "  117.7" "39625.5" "  100.0" 
Practical Range with cor=0.05 for asymptotic range: 299.5729

likfit: maximised log-likelihood = -897.8
```

```r
maximumLikelihoodNetherlandsInitial1.1
```

```
likfit: estimated model parameters:
      beta      tausq    sigmasq        phi 
```

```
" 102.376" " 114.249" "3036.627" "   7.132"
Practical Range with cor=0.05 for asymptotic range: 21.36583

likfit: maximised log-likelihood = -896.9
```

```
maximumLikelihoodNetherlandsInitial1000.1000
```

```
likfit: estimated model parameters:
      beta      tausq    sigmasq        phi
"   103.2" "   148.3" "246755.1" "  1000.0"
Practical Range with cor=0.05 for asymptotic range: 2995.732

likfit: maximised log-likelihood = -900.9
```

```
maximumLikelihoodNetherlandsInitial500.500
```

```
likfit: estimated model parameters:
      beta      tausq    sigmasq        phi
"   103.1" "   145.0" "130046.9" "   500.0"
Practical Range with cor=0.05 for asymptotic range: 1497.866

likfit: maximised log-likelihood = -900.1
```

As we can see from the maximised log-likelihoods it does seem that have indeed reached the maximum log-likelihood as none of the values are too fastly different and the likelihood worsedned as we started to increase much more our starting values.

Next we will try the REML that takes into account the fact that some of the parameters of the model are related to the variance of the residuals and not the mean.

```
REMLmaximumLikelihoodNetherlandsInitial10.1 = likfit(precipitationNetherland_geoR,
    ini.cov.pars = c(10, 1), lik.method = "REML")
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
```

```
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.

WARNING: estimated range is more than 10 times bigger than the biggest distance between two p
 1) excluding spatial dependence if estimated sill is too low and/or
 2) taking trends (covariates) into account
```

```r
REMLmaximumLikelihoodNetherlandsInitial1.10 = likfit(precipitationNetherland_geoR,
    ini.cov.pars = c(1, 10), lik.method = "REML")
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.

WARNING: estimated range is more than 10 times bigger than the biggest distance between two p
 1) excluding spatial dependence if estimated sill is too low and/or
 2) taking trends (covariates) into account
```

```r
REMLmaximumLikelihoodNetherlandsInitial100.1 = likfit(precipitationNetherland_geoR,
    ini.cov.pars = c(100, 1), lik.method = "REML")
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.
```

```
WARNING: estimated range is more than 10 times bigger than the biggest distance between two p
 1) excluding spatial dependence if estimated sill is too low and/or
 2) taking trends (covariates) into account
```

```
REMLmaximumLikelihoodNetherlandsInitial1.100 = likfit(precipitationNetherland_geoR,
    ini.cov.pars = c(1, 100), lik.method = "REML")
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.

WARNING: estimated range is more than 10 times bigger than the biggest distance between two p
 1) excluding spatial dependence if estimated sill is too low and/or
 2) taking trends (covariates) into account
```

```
REMLmaximumLikelihoodNetherlandsInitial10.1
```

```
likfit: estimated model parameters:
      beta      tausq    sigmasq        phi
"  102.61" "  114.60" "18253.12" "   43.35"
Practical Range with cor=0.05 for asymptotic range: 129.8698

likfit: maximised log-likelihood = -888.9
```

```
REMLmaximumLikelihoodNetherlandsInitial1.10
```

```
likfit: estimated model parameters:
      beta      tausq    sigmasq        phi
"  102.62" "  114.83" "18124.93" "   43.19"
Practical Range with cor=0.05 for asymptotic range: 129.3933

likfit: maximised log-likelihood = -888.9
```

```r
REMLmaximumLikelihoodNetherlandsInitial100.1
```

```
likfit: estimated model parameters:
      beta      tausq    sigmasq        phi 
"  102.61" "  114.60" "18253.12" "   43.35" 
Practical Range with cor=0.05 for asymptotic range: 129.8698

likfit: maximised log-likelihood = -888.9
```

```r
REMLmaximumLikelihoodNetherlandsInitial1.100
```

```
likfit: estimated model parameters:
     beta     tausq   sigmasq       phi 
"  102.7" "  116.5" "40885.8" "  100.0" 
Practical Range with cor=0.05 for asymptotic range: 299.5729

likfit: maximised log-likelihood = -888.9
```

As we can see from these new models, the new likelihood method actually did improve our model as we have a lower log-likelihood.

Since the data did not seem to be perfectly stationary as seen in the previous questions, we will now check if adding a linear trend improves our model

```r
linearREMLmaximumLikelihoodNetherlandsInitial10.1 = likfit(precipitationNetherland_geoR,
    trend = "1st", ini.cov.pars = c(10, 1), lik.method = "REML")
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.
```

```r
linearREMLmaximumLikelihoodNetherlandsInitial1.10 = likfit(precipitationNetherland_geoR,
    trend = "1st", ini.cov.pars = c(1, 10), lik.method = "REML")
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.
```

```r
linearREMLmaximumLikelihoodNetherlandsInitial100.10 = likfit(precipitationNetherland_geoR,
    trend = "1st", ini.cov.pars = c(100, 10), lik.method = "REML")
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.
```

```r
linearREMLmaximumLikelihoodNetherlandsInitial10.100 = likfit(precipitationNetherland_geoR,
    trend = "1st", ini.cov.pars = c(10, 100), lik.method = "REML")
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
```

```
---------------------------------------------------------------
likfit: end of numerical maximisation.

WARNING: estimated range is more than 10 times bigger than the biggest distance between two p
 1) excluding spatial dependence if estimated sill is too low and/or
 2) taking trends (covariates) into account
```

```
linearREMLmaximumLikelihoodNetherlandsInitial10.1
```

```
likfit: estimated model parameters:
       beta0        beta1        beta2        tausq      sigmasq          phi
"-2194.6747" "  -11.4587" "   45.2352" "  112.7395" "  191.5497" "    0.4033"
Practical Range with cor=0.05 for asymptotic range: 1.208203

likfit: maximised log-likelihood = -872
```

```
linearREMLmaximumLikelihoodNetherlandsInitial1.10
```

```
likfit: estimated model parameters:
      beta0       beta1       beta2       tausq     sigmasq         phi
"-2084.141" "   -6.973" "   42.674" "  125.053" " 3238.867" "    9.905"
Practical Range with cor=0.05 for asymptotic range: 29.67433

likfit: maximised log-likelihood = -873
```

```
linearREMLmaximumLikelihoodNetherlandsInitial100.10
```

```
likfit: estimated model parameters:
      beta0       beta1       beta2       tausq     sigmasq         phi
"-2084.141" "   -6.973" "   42.674" "  125.053" " 3238.867" "    9.905"
Practical Range with cor=0.05 for asymptotic range: 29.67433

likfit: maximised log-likelihood = -873
```

```
linearREMLmaximumLikelihoodNetherlandsInitial10.100
```

```
likfit: estimated model parameters:
     beta0       beta1       beta2       tausq     sigmasq         phi
"-2084.09" "   -6.91" "   42.67" "  125.89" "32191.57" "  100.00"
Practical Range with cor=0.05 for asymptotic range: 299.5701

likfit: maximised log-likelihood = -873
```

From these models we can see that the linear trend does indeed improve our model, now we will check if there are any other covariance functions that can improve the model further.

```
Matren0.5linearREMLmaximumLikelihoodNetherlandsInitial10.1 = likfit(precipitationNetherlan
    trend = "1st", ini.cov.pars = c(10, 1), lik.method = "REML", cov.model = "matern",
    kappa = 0.5)
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.
```

```
Matren1.0linearREMLmaximumLikelihoodNetherlandsInitial10.1 = likfit(precipitationNetherlan
    trend = "1st", ini.cov.pars = c(10, 1), lik.method = "REML", cov.model = "matern",
    kappa = 1)
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.
```

```r
Matren1.5linearREMLmaximumLikelihoodNetherlandsInitial10.1 = likfit(precipitationNetherlan
    trend = "1st", ini.cov.pars = c(10, 1), lik.method = "REML", cov.model = "matern",
    kappa = 1.5)
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.
```

```r
Matren2.0linearREMLmaximumLikelihoodNetherlandsInitial10.1 = likfit(precipitationNetherlan
    trend = "1st", ini.cov.pars = c(10, 1), lik.method = "REML", cov.model = "matern",
    kappa = 2)
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.
```

```r
Matren2.5linearREMLmaximumLikelihoodNetherlandsInitial10.1 = likfit(precipitationNetherlan
    trend = "1st", ini.cov.pars = c(10, 1), lik.method = "REML", cov.model = "matern",
    kappa = 2.5)
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
```

```
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.
```

```
Matren0.5linearREMLmaximumLikelihoodNetherlandsInitial10.1
```

```
likfit: estimated model parameters:
       beta0        beta1        beta2        tausq      sigmasq          phi
"-2194.6747" "  -11.4587" "   45.2352" "  112.7395" "  191.5497" "    0.4033"
Practical Range with cor=0.05 for asymptotic range: 1.208203

likfit: maximised log-likelihood = -872
```

```
Matren1.0linearREMLmaximumLikelihoodNetherlandsInitial10.1
```

```
likfit: estimated model parameters:
       beta0        beta1        beta2        tausq      sigmasq          phi
"-2221.5117" "  -12.1627" "   45.8184" "  125.4406" "  160.1807" "    0.2088"
Practical Range with cor=0.05 for asymptotic range: 0.8347704

likfit: maximised log-likelihood = -871.3
```

```
Matren1.5linearREMLmaximumLikelihoodNetherlandsInitial10.1
```

```
likfit: estimated model parameters:
       beta0        beta1        beta2        tausq      sigmasq          phi
"-2233.7541" "  -12.4219" "   46.0784" "  129.6185" "  149.9124" "    0.1529"
Practical Range with cor=0.05 for asymptotic range: 0.7254113

likfit: maximised log-likelihood = -871.1
```

```
Matren2.0linearREMLmaximumLikelihoodNetherlandsInitial10.1
```

```
likfit: estimated model parameters:
       beta0        beta1        beta2        tausq      sigmasq          phi
"-2240.7056" "  -12.5595" "   46.2252" "  131.4685" "  144.9394" "    0.1248"
```

```
Practical Range with cor=0.05 for asymptotic range: 0.669996

likfit: maximised log-likelihood = -871.1
```

```
Matren2.5linearREMLmaximumLikelihoodNetherlandsInitial10.1
```

```
likfit: estimated model parameters:
      beta0        beta1        beta2        tausq      sigmasq          phi 
"-2245.1703" "  -12.6453" "   46.3192" "  132.4280" "  142.0693" "    0.1074" 
Practical Range with cor=0.05 for asymptotic range: 0.6358818

likfit: maximised log-likelihood = -871
```

It does seem that the matrén covariance function did indeed slightly improved the model so we will compare it to a model using a spherical covariance function. The spherical covariance function is appropriate for this scenario has the spatial correlation between data points decreases rapidly as the distance between the points increases and we are limited with the range of correlation has after 2 degrees of distance we loose sensible correlation, hence the cut in the variogram.

```
SphericallinearREMLmaximumLikelihoodNetherlandsInitial10.1 = likfit(precipitationNetherlan
    trend = "1st", ini.cov.pars = c(10, 1), lik.method = "REML", cov.model = "spherical")
```

```
kappa not used for the spherical correlation function
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.
```

```
SphericallinearREMLmaximumLikelihoodNetherlandsInitial1.10 = likfit(precipitationNetherlan
    trend = "1st", ini.cov.pars = c(1, 10), lik.method = "REML", cov.model = "spherical")
```

```
kappa not used for the spherical correlation function
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.
```

```
SphericallinearREMLmaximumLikelihoodNetherlandsInitial100.10 = likfit(precipitationNetherl
    trend = "1st", ini.cov.pars = c(100, 10), lik.method = "REML", cov.model = "spherical"
```

```
kappa not used for the spherical correlation function
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.
```

```
SphericallinearREMLmaximumLikelihoodNetherlandsInitial10.100 = likfit(precipitationNetherl
    trend = "1st", ini.cov.pars = c(10, 100), lik.method = "REML", cov.model = "spherical"
```

```
kappa not used for the spherical correlation function
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
```

```
likfit: end of numerical maximisation.

WARNING: estimated range is more than 10 times bigger than the biggest distance between two p
 1) excluding spatial dependence if estimated sill is too low and/or
 2) taking trends (covariates) into account
```

```
SphericallinearREMLmaximumLikelihoodNetherlandsInitial10.1
```

```
likfit: estimated model parameters:
      beta0       beta1       beta2       tausq     sigmasq         phi
"-2182.422" "  -11.539" "   45.002" "  127.066" "  200.406" "    1.003"
Practical Range with cor=0.05 for asymptotic range: 1.00273

likfit: maximised log-likelihood = -872.2
```

```
SphericallinearREMLmaximumLikelihoodNetherlandsInitial1.10
```

```
likfit: estimated model parameters:
      beta0       beta1       beta2       tausq     sigmasq         phi
"-2082.850" "   -6.686" "   42.622" "  125.174" " 2151.559" "    9.905"
Practical Range with cor=0.05 for asymptotic range: 9.905214

likfit: maximised log-likelihood = -873
```

```
SphericallinearREMLmaximumLikelihoodNetherlandsInitial100.10
```

```
likfit: estimated model parameters:
      beta0       beta1       beta2       tausq     sigmasq         phi
"-2082.850" "   -6.686" "   42.622" "  125.174" " 2151.559" "    9.905"
Practical Range with cor=0.05 for asymptotic range: 9.905214

likfit: maximised log-likelihood = -873
```

```
SphericallinearREMLmaximumLikelihoodNetherlandsInitial10.100
```

```
likfit: estimated model parameters:
      beta0        beta1        beta2        tausq      sigmasq          phi
"-2083.890" "   -6.902" "   42.667" "  125.695" "21538.008" "   99.999"
Practical Range with cor=0.05 for asymptotic range: 99.99893

likfit: maximised log-likelihood = -873
```

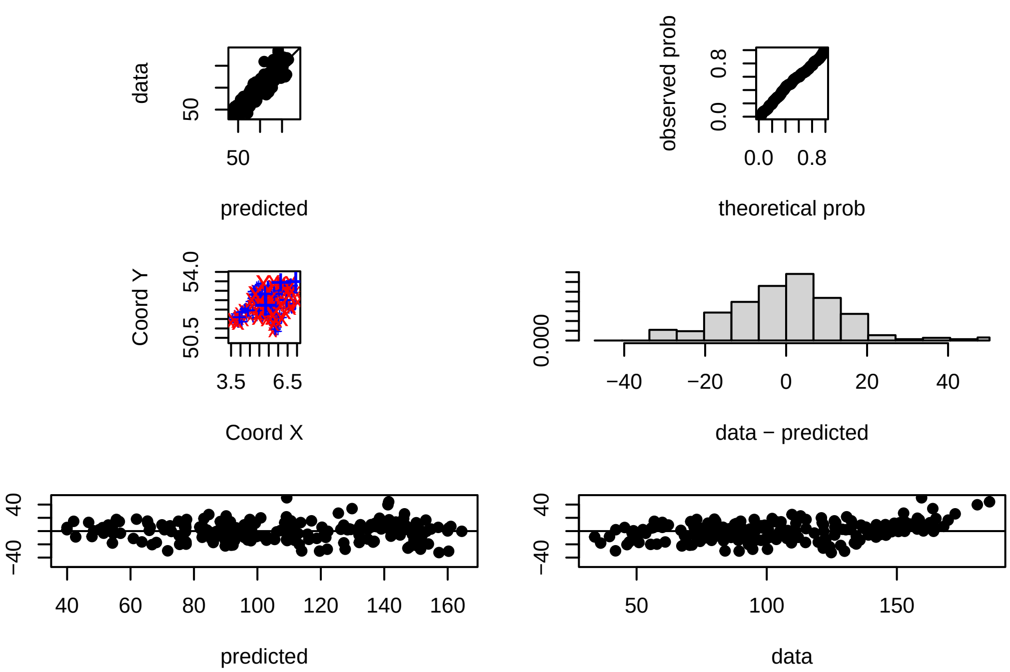
As we can see the spherical covariance function does not provide as good of a fit as the Matrén.

We will now validate the model by doing cross-validation on the model.

```r
xv.ml = xvalid(precipitationNetherland_geoR, model = Matren2.5linearREMLmaximumLikelihoodN
```

```
xvalid: number of data locations       = 217
xvalid: number of validation locations = 217
xvalid: performing cross-validation at location ... 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
xvalid: end of cross-validation
```

```r
par(mfrow = c(3, 2), mar = c(4, 2, 2, 2))
plot(xv.ml, error = TRUE, std.error = FALSE, pch = 19)
```



From these plots we can see that the residuals seem mostly normal without any quickly identifiable patterns or bias.

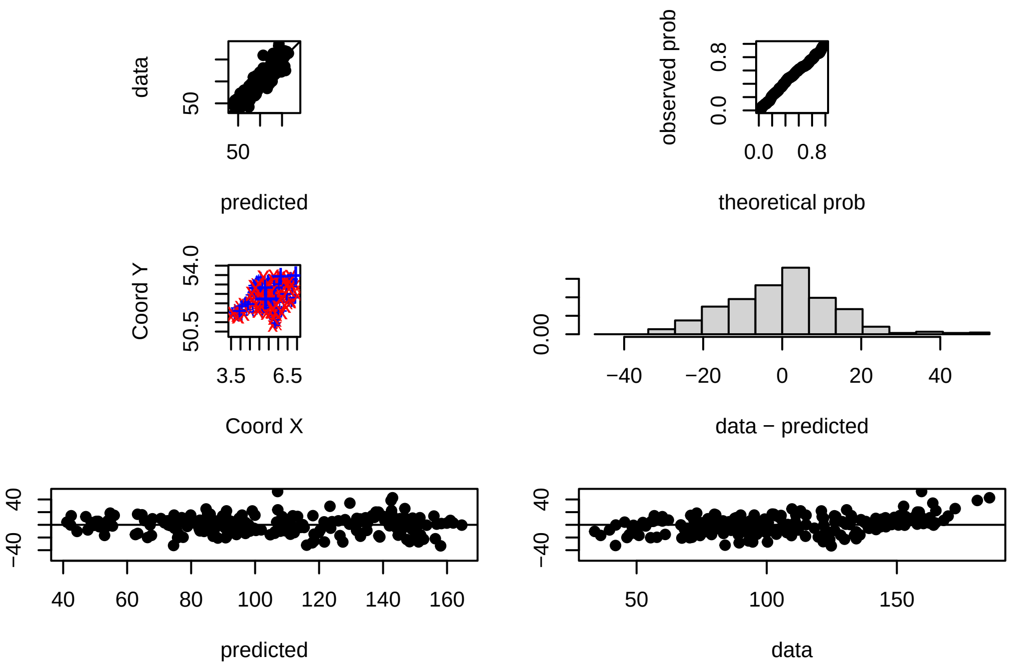
From the first top left graph we can see however that we seem to sightly underestimate more data points.

To account for bias we will also perform cross-validation to the next best performing model using the spherical function instead

```
xv.ml = xvalid(precipitationNetherland_geoR, model = SphericallinearREMLmaximumLikelihoodN
```

```
xvalid: number of data locations       = 217
xvalid: number of validation locations = 217
xvalid: performing cross-validation at location ... 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
xvalid: end of cross-validation
```

```
par(mfrow = c(3, 2), mar = c(4, 2, 2, 2))
plot(xv.ml, error = TRUE, std.error = FALSE, pch = 19)
```



As we can see the data at the start seems to be systematically underestimated and at the end it seems to overestimated. Furthermore the theoretical data plot seems to be less linear.

This confirms that the spherical covariance function is indeed a worse model than the Matrén model.

1 f)

First we will start by making the predictions using the variogram

```
spatialPointsABC = randomRowsPrecipitation[, c("longitude", "latitude")]


# Set the krige.control parameters
krigeControl = krige.control(type.krige = "OK", cov.model = krigingVariogramFittedMatrén1.
    cov.pars = krigingVariogramFittedMatrén1.0$cov.pars)


# Kriging with the fitted variogram model
krigeResults = krige.conv(precipitationNetherland_geoR, locations = spatialPointsABC,
    krige = krigeControl)
```

```
krige.conv: model with constant mean
krige.conv: Kriging performed using global neighbourhood
```

```
# Extract predictions from the kriging results
predictions = krigeResults$predict

# Compare the predicted values with the actual precipitation values

actualPrecipitationValues = randomRowsPrecipitation[, 4]

comparisonvariogram = data.frame(actualPrecipitationValues, predictions)
```

Now for the maximum likelihood function

```
# done above spatialPointsABC = randomRowsPrecipitation[,
# c('longitude', 'latitude')]


# Set the krige.control parameters
krigeControl = krige.control(type.krige = "OK", cov.model = Matren2.5linearREMLmaximumLike
    cov.pars = Matren2.5linearREMLmaximumLikelihoodNetherlandsInitial10.1$cov.pars)
```

```r
# Kriging with the fitted variogram model
krigeResults = krige.conv(precipitationNetherland_geoR, locations = spatialPointsABC,
    krige = krigeControl)
```

```
krige.conv: model with constant mean
krige.conv: Kriging performed using global neighbourhood
```

```r
# Extract predictions from the kriging results
predictions = krigeResults$predict

# Compare the predicted values with the actual precipitation values

# done above actualPrecipitationValues = randomRowsPrecipitation[,4]
comparisonMaximumLikelihood = data.frame(actualPrecipitationValues, predictions)
```

Now that we have made the predictions for our 2 models we will check the predicted values compared to the real values for each of the models.

```r
comparisonvariogram
```

```
  precip predictions
1   89.1    95.41904
2   95.9    98.90558
3  147.2   147.82038
```

```r
comparisonMaximumLikelihood
```

```
  precip predictions
1   89.1    99.00780
2   95.9    99.36194
3  147.2   140.07734
```

```r
# Calculate Mean Absolute Error (MAE)
MAEVariogram = mean(abs(comparisonvariogram$precip - comparisonvariogram$predictions))
MAEMaximumLikelihood = mean(abs(comparisonMaximumLikelihood$precip - comparisonMaximumLike
```

```
# Calculate Mean Squared Error (MSE)
mseVariogram = mean((comparisonvariogram$precip - comparisonvariogram$predictions)^2)
mseMaximumLikelihood = mean((comparisonMaximumLikelihood$precip - comparisonMaximumLikelih


# Calculate Root Mean Squared Error (RMSE)
rmseVariogram = sqrt(mseVariogram)
rmseMaximumLikelihood = sqrt(mseMaximumLikelihood)


# Display the calculated metrics
cat("Mean Absolute Error (MAE) of the Variogram:", MAEVariogram, "\n")
```

```
Mean Absolute Error (MAE) of the Variogram: 3.315002
```

```
cat("Mean Squared Error (MSE) of the Variogram:", mseVariogram, "\n")
```

```
Mean Squared Error (MSE) of the Variogram: 16.44957
```

```
cat("Root Mean Squared Error (RMSE) of the Variogram:", rmseVariogram, "\n")
```

```
Root Mean Squared Error (RMSE) of the Variogram: 4.055807
```

```
cat("\n\n")
```

```
cat("Mean Absolute Error (MAE) of the maximum likelihood:", MAEMaximumLikelihood,
    "\n")
```

```
Mean Absolute Error (MAE) of the maximum likelihood: 6.830801
```

```
cat("Mean Squared Error (MSE) of the maximum likelihood:", mseMaximumLikelihood,
    "\n")
```

```
Mean Squared Error (MSE) of the maximum likelihood: 53.62729
```

```r
cat("Root Mean Squared Error (RMSE) of the maximum likelihood:", rmseMaximumLikelihood,
    "\n")
```

```
Root Mean Squared Error (RMSE) of the maximum likelihood: 7.323066
```

As we can see from both the real values and the MAE, MSE and RMSE the variogram has as much better performance predicting those 3 points than our maximum likelihood model

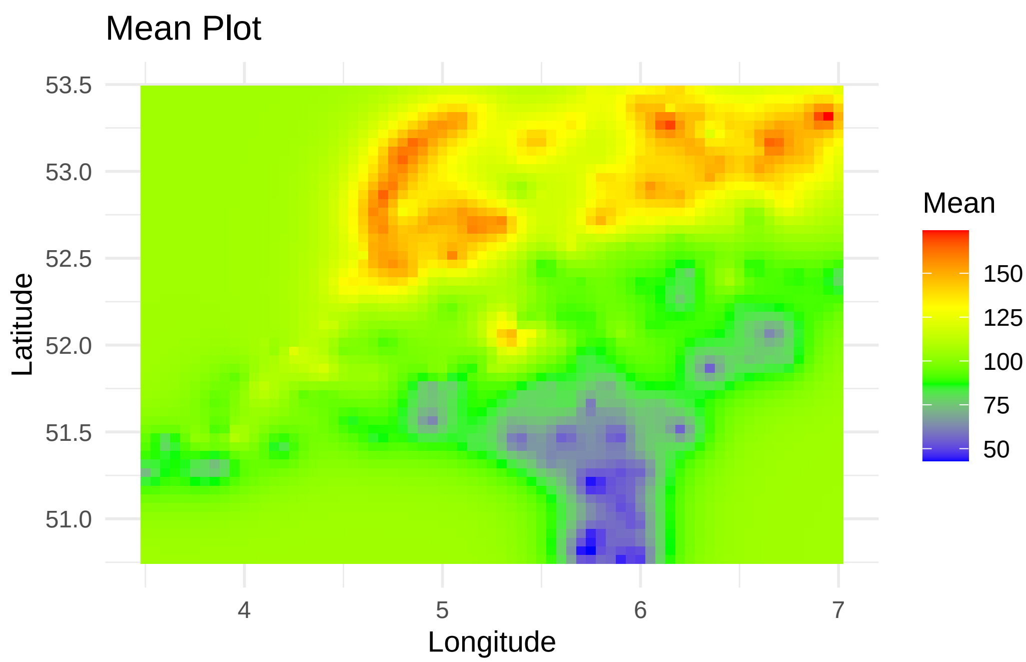
1 g)

```r
# Determine the range of the coordinates
xRange = range(precipitationNetherland_geoR$coords[, 1])
yRange = range(precipitationNetherland_geoR$coords[, 2])

# Create a grid with 0.05-degree spacing
gridPoints = expand.grid(x = seq(xRange[1], xRange[2], by = 0.05), y = seq(yRange[1],
    yRange[2], by = 0.05))


# Kriging with the fitted variogram model
krigeResults = krige.conv(precipitationNetherland_geoR, locations = gridPoints,
    krige = krigeControl)
```

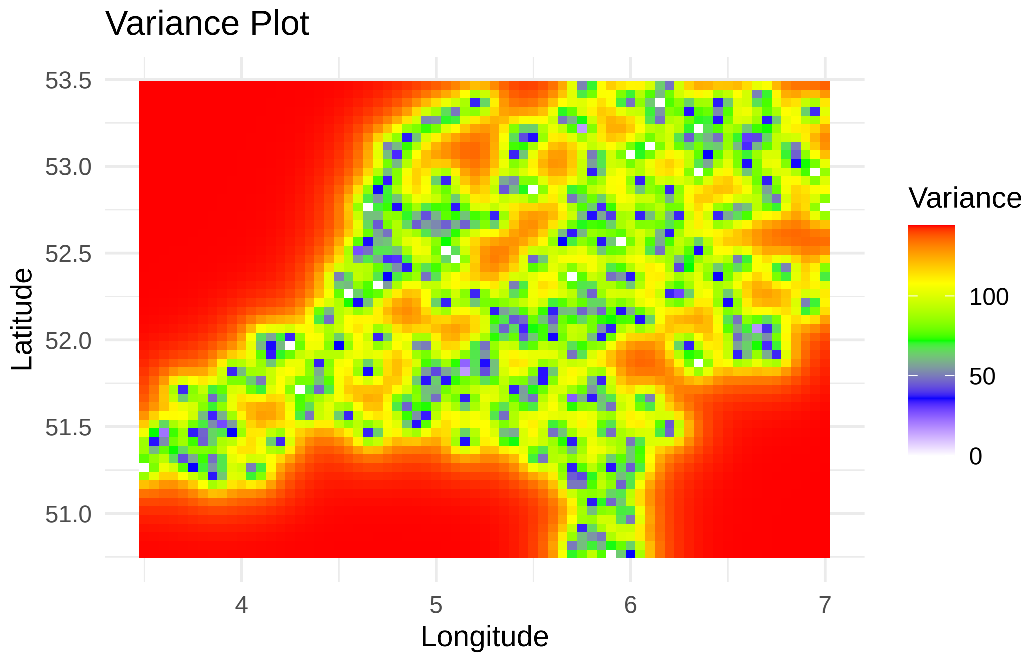
```
krige.conv: model with constant mean
krige.conv: Kriging performed using global neighbourhood
```

```r
# Create a data frame for the grid points with the predicted mean and
# variance
gridData = data.frame(gridPoints, mean = krigeResults$predict, variance = krigeResults$kri

# Mean plot
meanPlot = ggplot(gridData, aes(x = x, y = y, fill = mean)) + geom_tile() +
    scale_fill_gradientn(colors = c("blue", "green", "yellow", "red")) +
    theme_minimal() + ggtitle("Mean Plot") + labs(x = "Longitude", y = "Latitude",
    fill = "Mean")

# Variance plot
variancePlot = ggplot(gridData, aes(x = x, y = y, fill = variance)) + geom_tile() +
```

```
    scale_fill_gradientn(colors = c("white", "blue", "green", "yellow", "red")) +
    theme_minimal() + ggtitle("Variance Plot") + labs(x = "Longitude", y = "Latitude",
    fill = "Variance")

# Display the plots
print(meanPlot)
```




```
print(variancePlot)
```



1 h)

For the priors I will be using the estimated values from our latest maximum likelihood model.

```
# Extract the estimated parameters

priorPhiVariogram = krigingVariogramFittedMatrén1.0$cov.pars[1]
priorTauSQVariogram = krigingVariogramFittedMatrén1.0$cov.pars[2]


priorPhiMaximumLikelihood = Matren1.0linearREMLmaximumLikelihoodNetherlandsInitial10.1$cov
priorTauSQMaximumLikelihood = Matren1.0linearREMLmaximumLikelihoodNetherlandsInitial10.1$c
```

The function does not support continuous priors directly so we will fit them as discrete priors.

```
# Creating discrete priors for phi and tau^2_rel AKA this is for the
# model with a nugget
phiDiscrete <- seq(min(priorPhiVariogram, priorPhiMaximumLikelihood) * 0.5,
    max(priorPhiVariogram, priorPhiMaximumLikelihood) * 1.5, length.out = 50)

tauSqDiscrete <- seq(min(priorTauSQVariogram, priorTauSQMaximumLikelihood) *
```

```r
    0.5, max(priorTauSQVariogram, priorTauSQMaximumLikelihood) * 1.5, length.out = 50)

# Informative priors based on the parameter estimates
phiProbability <- dnorm(phiDiscrete, mean = (priorPhiVariogram + priorPhiMaximumLikelihood
    sd = abs(priorPhiVariogram - priorPhiMaximumLikelihood)/2)
tauSqProbability <- dnorm(tauSqDiscrete, mean = (priorTauSQVariogram + priorTauSQMaximumLi
    sd = abs(priorTauSQVariogram - priorTauSQMaximumLikelihood)/2)

# Normalizing the probabilities
phiProbability <- phiProbability/sum(phiProbability)
tauSqProbability <- tauSqProbability/sum(tauSqProbability)


ex.grid <- as.matrix(expand.grid(seq(50.5, 53.5, l = 15), seq(3.5, 7, l = 15)))


# Fitting the krige.bayes model with the informative priors
krigeBayesModelWithNugget <- krige.bayes(geodata = precipitationNetherland_geoR,
    loc = ex.grid, prior = prior.control(phi.prior = phiProbability, phi.discrete = phiDis
        tausq.rel.prior = tauSqProbability, tausq.rel.discrete = tauSqDiscrete))
```

```
krige.bayes: model with constant mean
krige.bayes: computing the discrete posterior of phi/tausq.rel
krige.bayes: computing the posterior probabilities.
             Number of parameter sets:  2500
1, 101, 201, 301, 401, 501, 601, 701, 801, 901, 1001, 1101, 1201, 1301, 1401, 1501, 1601, 17(

krige.bayes: sampling from posterior distribution
krige.bayes: sample from the (joint) posterior of phi and tausq.rel
                  [,1]
phi         80.0903555
tausq.rel    0.1043848
frequency 1000.0000000

krige.bayes: starting prediction at the provided locations
krige.bayes: phi/tausq.rel samples for the predictive are same as for the posterior
krige.bayes: computing moments of the predictive distribution
krige.bayes: sampling from the predictive
             Number of parameter sets:  1
1,
krige.bayes: preparing summaries of the predictive distribution
```

```
krigeBayesModelWithoutNugget <- krige.bayes(geodata = precipitationNetherland_geoR,
    loc = ex.grid, prior = prior.control(phi.prior = phiProbability, phi.discrete = phiDis
```

```
krige.bayes: model with constant mean
krige.bayes: computing the discrete posterior of phi/tausq.rel
krige.bayes: computing the posterior probabilities.
             Number of parameter sets:  50
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26

krige.bayes: sampling from posterior distribution
krige.bayes: sample from the (joint) posterior of phi and tausq.rel
              [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
phi       80.09036 226.3799 372.6695 518.9591 665.2486 811.5382 957.8278
tausq.rel  0.00000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
frequency 16.00000  20.0000  14.0000  31.0000  23.0000  30.0000  23.0000
              [,8]     [,9]    [,10]    [,11]    [,12]    [,13]    [,14]
phi       1104.117 1250.407 1396.696 1542.986 1689.276 1835.565 1981.855
tausq.rel    0.000    0.000    0.000    0.000    0.000    0.000    0.000
frequency   24.000   21.000   28.000   21.000   27.000   28.000   31.000
             [,15]    [,16]    [,17]    [,18]    [,19]    [,20]    [,21]
phi       2128.144 2274.434 2420.723 2567.013 2713.303 2859.592 3005.882
tausq.rel    0.000    0.000    0.000    0.000    0.000    0.000    0.000
frequency   46.000   32.000   37.000   28.000   26.000   26.000   28.000
             [,22]    [,23]   [,24]   [,25]   [,26]    [,27]    [,28]    [,29]
phi       3152.171 3298.461 3444.75 3591.04 3737.33 3883.619 4029.909 4176.198
tausq.rel    0.000    0.000    0.00    0.00    0.00    0.000    0.000    0.000
frequency   30.000   37.000   20.00   18.00   23.00   14.000   24.000   20.000
             [,30]    [,31]    [,32]    [,33]    [,34]    [,35]    [,36]
phi       4322.488 4468.777 4615.067 4761.357 4907.646 5053.936 5200.225
tausq.rel    0.000    0.000    0.000    0.000    0.000    0.000    0.000
frequency   21.000   19.000   16.000   19.000   19.000   20.000   20.000
             [,37]    [,38]    [,39]    [,40]    [,41]    [,42]    [,43]
phi       5346.515 5492.804 5639.094 5785.384 5931.673 6077.963 6224.252
tausq.rel    0.000    0.000    0.000    0.000    0.000    0.000    0.000
frequency   17.000   14.000    8.000   17.000    5.000   12.000    9.000
             [,44]    [,45]    [,46]    [,47]  [,48]   [,49]    [,50]
phi       6370.542 6516.831 6663.121 6809.411 6955.7 7101.99 7248.279
tausq.rel    0.000    0.000    0.000    0.000    0.0    0.00    0.000
frequency    9.000    8.000    7.000    5.000    2.0    3.00    4.000

krige.bayes: starting prediction at the provided locations
krige.bayes: phi/tausq.rel samples for the predictive are same as for the posterior
```

```
krige.bayes: computing moments of the predictive distribution
krige.bayes: sampling from the predictive
             Number of parameter sets:  50
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26
krige.bayes: preparing summaries of the predictive distribution
```

```r
summary(krigeBayesModelWithoutNugget$posterior$sample)
```

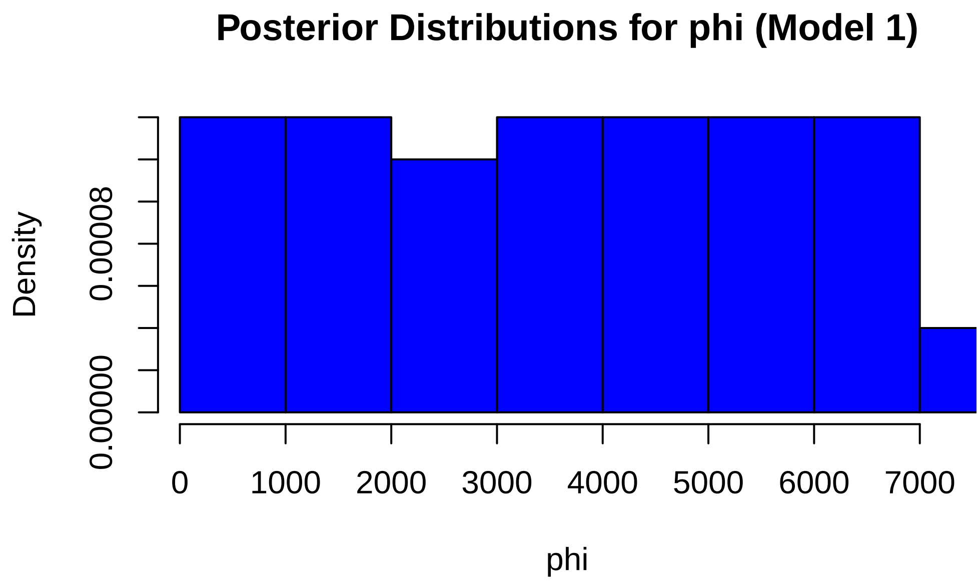
```
      beta              sigmasq               phi             tausq.rel
 Min.   :-14806.5   Min.   :  375543   Min.   :  80.09   Min.   :0
 1st Qu.: -2113.0   1st Qu.: 8331936   1st Qu.:1542.99   1st Qu.:0
 Median :   290.8   Median :14519487   Median :2713.30   Median :0
 Mean   :   299.1   Mean   :15700517   Mean   :2965.95   Mean   :0
 3rd Qu.:  2547.6   3rd Qu.:22223235   3rd Qu.:4322.49   3rd Qu.:0
 Max.   : 16575.4   Max.   :44053886   Max.   :7248.28   Max.   :0
```

```r
summary(krigeBayesModelWithNugget$posterior$sample)
```

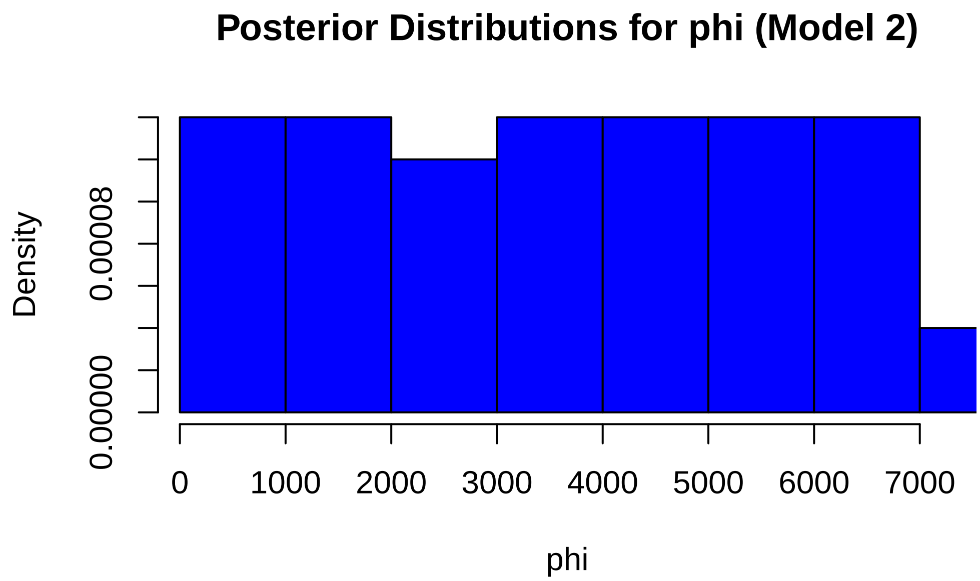
```
      beta            sigmasq           phi            tausq.rel
 Min.   :-49.52   Min.   :2216   Min.   :80.09   Min.   :0.1044
 1st Qu.: 66.57   1st Qu.:3000   1st Qu.:80.09   1st Qu.:0.1044
 Median :106.16   Median :3207   Median :80.09   Median :0.1044
 Mean   :105.66   Mean   :3238   Mean   :80.09   Mean   :0.1044
 3rd Qu.:145.57   3rd Qu.:3433   3rd Qu.:80.09   3rd Qu.:0.1044
 Max.   :282.96   Max.   :4455   Max.   :80.09   Max.   :0.1044
```

Now we will compare the posterior of both of the models to see the impact of the nugget

```r
# Extract posterior samples for phi and tau^2_rel
posterior_samples_model1 <- krigeBayesModelWithNugget$posterior$phi$phi.marginal$phi
posterior_samples_model2 <- krigeBayesModelWithoutNugget$posterior$phi$phi.marginal$phi

# Plot the posterior distributions
hist(posterior_samples_model1, freq = FALSE, main = "Posterior Distributions for phi (Mode
    xlab = "phi", col = "blue", xlim = range(c(posterior_samples_model1,
        posterior_samples_model2)))
```




```
hist(posterior_samples_model2, freq = FALSE, main = "Posterior Distributions for phi (Mode
    xlab = "phi", col = "blue", xlim = range(c(posterior_samples_model1,
        posterior_samples_model2)))
```




```r
# Compare summary statistics
summary_model1 <- summary(posterior_samples_model1)
summary_model2 <- summary(posterior_samples_model2)

cat("model 1 :\n")
```

```
model 1 :
```

```r
summary_model1
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  80.09 1872.14 3664.18 3664.18 5456.23 7248.28 
```

```r
cat("\n\n model 2 :\n")
```

```
 model 2 :
```

```
summary_model2
```

```
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 80.09 1872.14 3664.18 3664.18 5456.23 7248.28 
```

```
# Reset the plot layout
par(mfrow = c(1, 1))
```

As we can seem with a low number of binds we can't see any significant difference in the summaries or histogram between the models with and without a nugget

Question 2

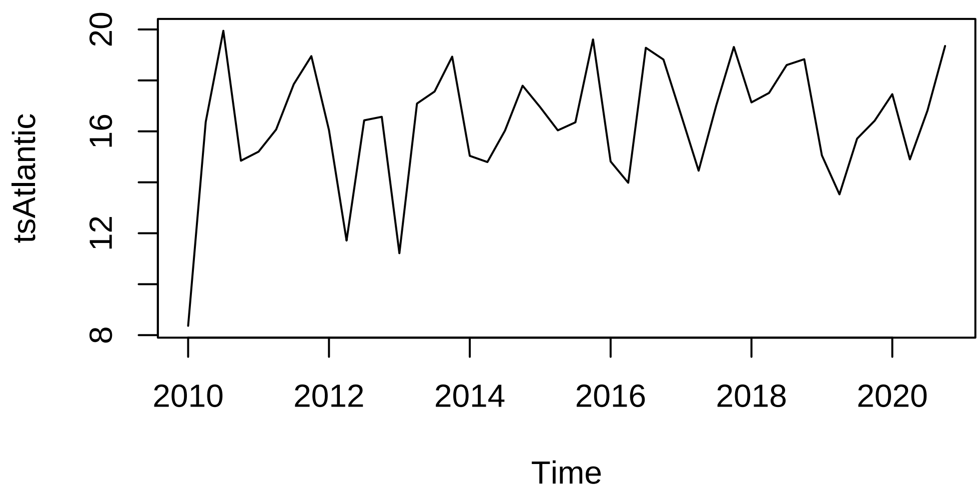
2 a)

We fist start by making the appropriate changes in the data to average the data to quarterly means

```
AMOCDF$Date = as.Date(AMOCDF$Date, format = "%d/%m/%Y")

## I will now make a column with the quarter and year that I will use
## to create the averages per quarter
AMOCDF$YearQuarter = paste(AMOCDF$Year, AMOCDF$Quarter, sep = "-")

YearQuarterAverage = AMOCDF %>%
    group_by(YearQuarter) %>%
    summarise(AverageStrength = mean(Strength))
```

Now we will convert the average data to a time series object to be able to plot it

```
tsAtlantic = ts(YearQuarterAverage, start = c(2010, 1), frequency = 4)

tsAtlantic = tsAtlantic[, "AverageStrength"]

plot.ts(tsAtlantic)
```



Trend analysis

From this graph we can see a yearly oscillation of Sverdrups. We can also identify that the peaks in Sverdrups are usually in the last quarter before the start of a new year and the valleys are on the second quarter of the year.

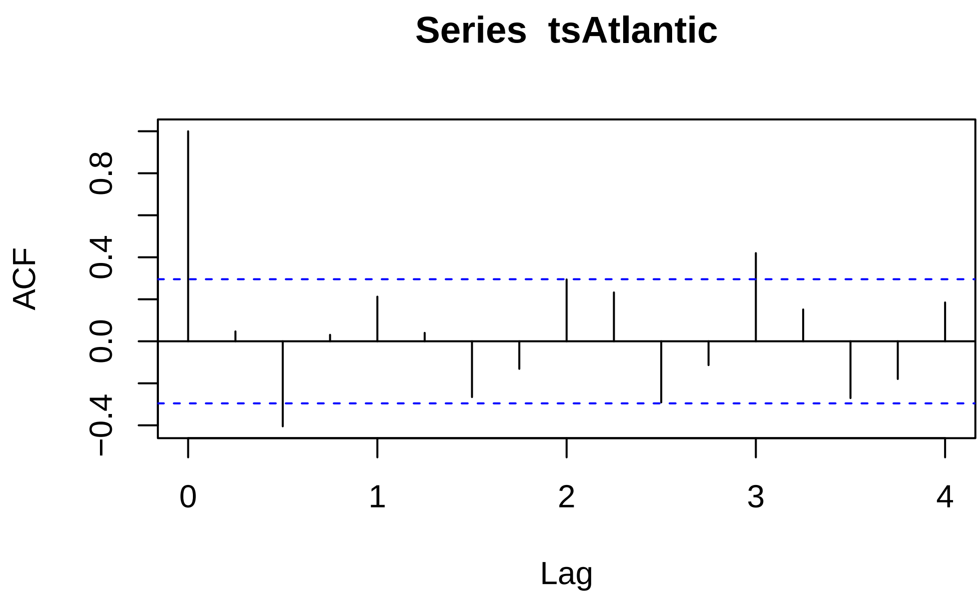
The data does seem stationary enough that if we were to differentiate we would start losing some of the structure.

2 b)

ACF

First we will start by checking the ACF(Autocorrelation Function) and PACF(Partial Autocorrelation Function) to check for if we have stationary data or not to help us decide between an ARMA or an ARIMA model.

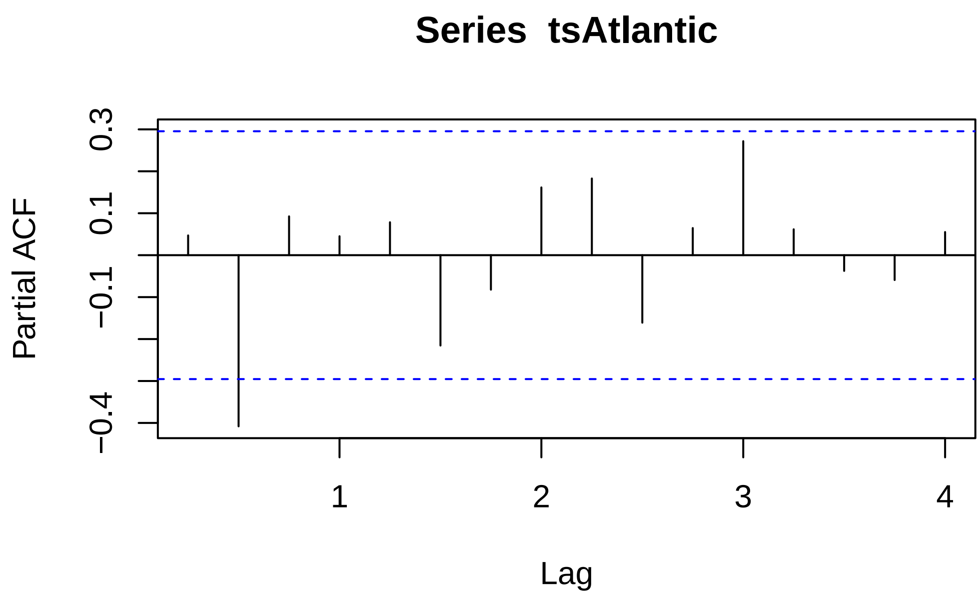
```
acf(tsAtlantic)
```



We can see that for ACF OF Average strength slowly decreases as lag increases to infinity with lag = 3 still being a significant values, meaning it is not a simple MA model as AR is clearly not quickly cut-off.

PACF

```
pacf(tsAtlantic)
```



The PACF seems to be cut-off at lag 0,5 indicating an AR model might be a best fit for our data to be a but with some almost significant values after the cut it might be also appropriate to some non-zero q values to confirm our initial assumption

As such we will now proceed to fit multiple model firstly with the initial assumption that, then I will both use models with non-zero q and the model given by the auto.arima function to double check that the assumptions made by the previous analyses is correct.

```
# it is always a good practice to try multiple values of p,d and q to
# see if we can do better we then obviously compare via the AIC of the
# models and their log likelihoods it is never enough to check those we
# also need to check the residuals

## order is p, d ,q

## initial models under our assumptions

model100 = Arima(tsAtlantic, order = c(1, 0, 0))
model200 = Arima(tsAtlantic, order = c(2, 0, 0))
model300 = Arima(tsAtlantic, order = c(3, 0, 0))

## now I will add postive q values
```

```r
model101 = Arima(tsAtlantic, order = c(1, 0, 1))
model102 = Arima(tsAtlantic, order = c(1, 0, 2))
model103 = Arima(tsAtlantic, order = c(1, 0, 3))

model201 = Arima(tsAtlantic, order = c(2, 0, 1))
model202 = Arima(tsAtlantic, order = c(2, 0, 2))
model203 = Arima(tsAtlantic, order = c(2, 0, 3))

model301 = Arima(tsAtlantic, order = c(3, 0, 1))
model302 = Arima(tsAtlantic, order = c(3, 0, 2))
model303 = Arima(tsAtlantic, order = c(3, 0, 3))



## lastly we will use auto.arima without seasonality to confirm our
## inital assumptions
modelAuto = auto.arima(tsAtlantic, max.d = 0, max.p = 5, max.q = 5, seasonal = FALSE)
```

best model selection

```r
model100
```

```
Series: tsAtlantic
ARIMA(1,0,0) with non-zero mean

Coefficients:
         ar1     mean
      0.0665  16.3878
s.e.  0.1788   0.3726

sigma^2 = 5.572:  log likelihood = -99.2
AIC=204.41   AICc=205.01   BIC=209.76
```

```r
model200
```

```
Series: tsAtlantic
ARIMA(2,0,0) with non-zero mean

Coefficients:
```

```
         ar1      ar2     mean
      0.0990  -0.5565  16.4298
s.e.  0.1576   0.1488   0.2113

sigma^2 = 4.321:  log likelihood = -93.45
AIC=194.9   AICc=195.92   BIC=202.04
```

```
model300
```

```
Series: tsAtlantic
ARIMA(3,0,0) with non-zero mean

Coefficients:
         ar1      ar2     ar3     mean
      0.1626  -0.5690  0.1464  16.4227
s.e.  0.1729   0.1479  0.1708   0.2409

sigma^2 = 4.35:  log likelihood = -93.09
AIC=196.17   AICc=197.75   BIC=205.1
```

As we can see from these inital models ARIMA(2,0,0) is the model that has the best fit has we can see from its lower AIC score of 194,9.

Now we will check against the other models to check the validity of our assumptions.

```
model101
```

```
Series: tsAtlantic
ARIMA(1,0,1) with non-zero mean

Coefficients:
          ar1     ma1     mean
      -0.4204  0.7718  16.3721
s.e.   0.2390  0.1466   0.4067

sigma^2 = 5.045:  log likelihood = -96.64
AIC=201.29   AICc=202.31   BIC=208.43
```

```
model102
```

```
Series: tsAtlantic
ARIMA(1,0,2) with non-zero mean

Coefficients:
         ar1     ma1      ma2     mean
      0.0230  0.1275  -0.4485  16.4289
s.e.  0.3051  0.2420   0.1348   0.2224

sigma^2 = 4.651:  log likelihood = -94.41
AIC=198.81   AICc=200.39   BIC=207.73
```

```
model103
```

```
Series: tsAtlantic
ARIMA(1,0,3) with non-zero mean

Coefficients:
          ar1     ma1      ma2      ma3     mean
      -0.5284  0.7214  -0.3646  -0.3072  16.4299
s.e.   0.9228  0.8649   0.2077   0.3545   0.2195

sigma^2 = 4.733:  log likelihood = -94.25
AIC=200.5   AICc=202.77   BIC=211.21
```

```
model201
```

```
Series: tsAtlantic
ARIMA(2,0,1) with non-zero mean

Coefficients:
          ar1      ar2     ma1     mean
      -0.0669  -0.5475  0.2187  16.4255
s.e.   0.2740   0.1555  0.2883   0.2300

sigma^2 = 4.366:  log likelihood = -93.15
AIC=196.31   AICc=197.88   BIC=205.23
```

```
model202
```

```
Series: tsAtlantic
ARIMA(2,0,2) with non-zero mean

Coefficients:
         ar1      ar2      ma1     ma2     mean
      0.0787  -0.9982  -0.0255  0.9999  16.4015
s.e.  0.0285   0.0066   0.0899  0.1158   0.2684

sigma^2 = 3.378:  log likelihood = -89.46
AIC=190.91   AICc=193.18   BIC=201.62
```

```
model203
```

```
Series: tsAtlantic
ARIMA(2,0,3) with non-zero mean

Coefficients:
         ar1      ar2     ma1     ma2     ma3     mean
      0.0325  -0.9621  0.0499  0.8487  0.4147  16.4028
s.e.  0.0645   0.0442  0.1987  0.2041  0.2511   0.3044

sigma^2 = 3.315:  log likelihood = -89.07
AIC=192.13   AICc=195.25   BIC=204.62
```

```
model301
```

```
Series: tsAtlantic
ARIMA(3,0,1) with non-zero mean

Coefficients:
         ar1      ar2     ar3      ma1     mean
      0.4092  -0.5931  0.2864  -0.2467  16.4191
s.e.  0.6330   0.1651  0.3580   0.6291   0.2537

sigma^2 = 4.449:  log likelihood = -93.03
AIC=198.06   AICc=200.34   BIC=208.77
```

```
model302
```

```
Series: tsAtlantic
ARIMA(3,0,2) with non-zero mean

Coefficients:
         ar1      ar2     ar3      ma1     ma2     mean
      0.2684  -0.9851  0.2222  -0.3030  1.0000  16.4144
s.e.  0.1999   0.0305  0.1995   0.1453  0.1921   0.2922

sigma^2 = 3.392:  log likelihood = -89.53
AIC=193.06   AICc=196.17   BIC=205.54
```

```
model303
```

```
Series: tsAtlantic
ARIMA(3,0,3) with non-zero mean

Coefficients:
          ar1      ar2      ar3     ma1     ma2     ma3     mean
      -0.3983  -0.9518  -0.4263  0.4381  0.7816  0.7352  16.4197
s.e.   0.3385   0.0412   0.3442  0.2946  0.1690  0.2422   0.2691

sigma^2 = 3.352:  log likelihood = -88.54
AIC=193.08   AICc=197.19   BIC=207.35
```

In this initial analysis we have found models that do have a lower AIC lower log likelihood than our previous best model, however these model ma's standard error are to close the the ma values indicating that while we are getting a better fit we might be overfitting to our data.

As such this does confirm our initial assumption for the choice of a zero q value.

Now lastly we will check if the auto.arima function does comfirm our initial assumptions.

```
modelAuto
```

```
Series: tsAtlantic
ARIMA(2,0,0) with non-zero mean

Coefficients:
         ar1      ar2     mean
      0.0990  -0.5565  16.4298
s.e.  0.1576   0.1488   0.2113
```

```
sigma^2 = 4.321:  log likelihood = -93.45
AIC=194.9   AICc=195.92   BIC=202.04
```

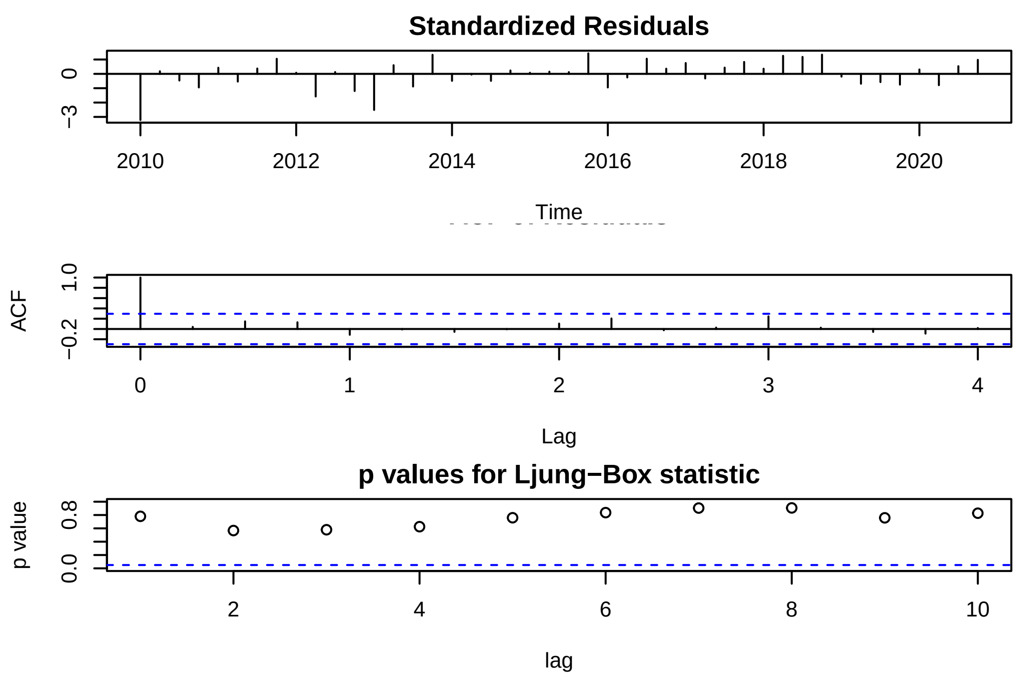
The function does confirm our assumption that ARIMA(2,0,0) is indeed the best model.

We will now check the residuals to verify if any of ou previously selected model validates well or if it is simply the best of bad models.

talk about the model being more easily explainability becaues MA = 0

Best model residual validation

```r
# Set smaller margins
par(mar = c(4, 4, 2, 2))

tsdiag(model200)
```




```r
# Reset margins
par(mar = c(5, 4, 4, 2) + 0.1)
```

Initially from the standardised residuals plot we can identify some sort of sinusoidal pattern, this implies that there is a seasonal trend that is not being accounted for in our model and as such this trends needs to be accounted in future models to better explain and increase the prediction power of a new model.

Forecasting

Now using the forecast function we will forecast the next 4 quarters of 2021

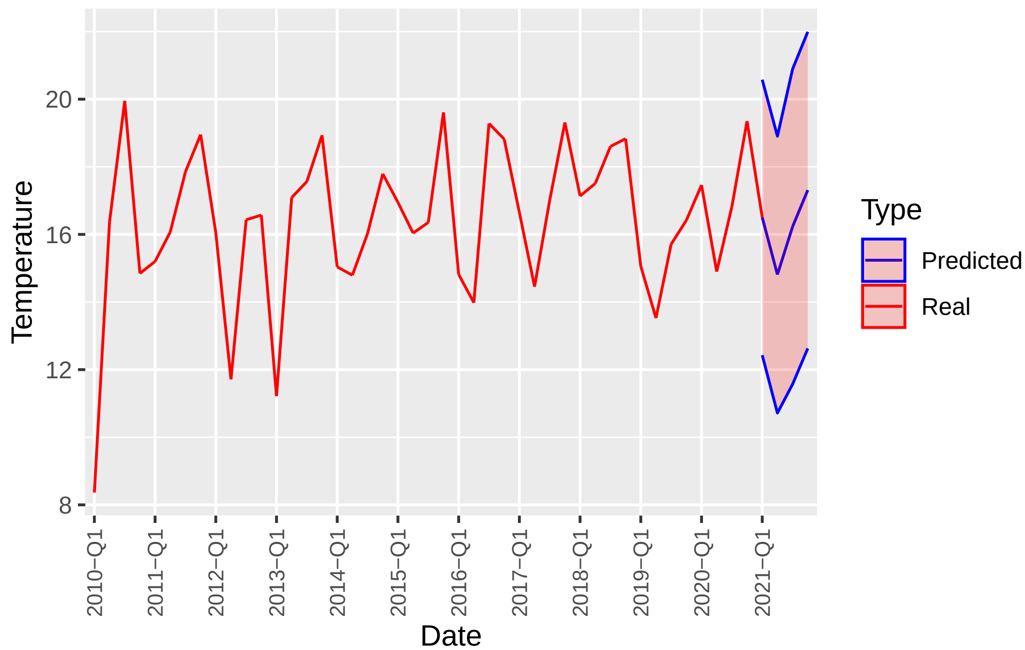
```
forecast(model200, 4)
```

```
        Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
2021 Q1       16.50240 13.83841 19.16639 12.42818 20.57662
2021 Q2       14.81104 12.13403 17.48804 10.71691 18.90517
2021 Q3       16.22919 13.18168 19.27669 11.56843 20.88994
2021 Q4       17.31076 14.24941 20.37212 12.62882 21.99271
```

But this data is better visualized in a graph to better understand if the predictions are sensible compared to our real data.

```
predictedArimaDF = data.frame(forecast(model200, 4))

predictedArimaDF$YearQuarter = c("2021-Q1", "2021-Q2", "2021-Q3", "2021-Q4")

# Combine real_data and pred_data into a single data frame
combinedDataframeAMOC = rbind(data.frame(Date = YearQuarterAverage$YearQuarter,
    Temperature = YearQuarterAverage$AverageStrength, Type = "Real"), data.frame(Date = pr
    Temperature = predictedArimaDF$Point.Forecast, Type = "Predicted"))

predictedArimaDF$Temperature = predictedArimaDF$Point.Forecast

predictedArimaDF$Type = "Predicted"


# Create the ggplot
plotARIMA = ggplot(combinedDataframeAMOC, aes(x = Date, y = Temperature,
    color = Type, group = 1)) + geom_line() + scale_color_manual(values = c("blue",
    "red"))

# Add the 95% confidence interval
plotARIMA = plotARIMA + geom_ribbon(data = predictedArimaDF, aes(x = YearQuarter,
    ymin = Lo.95, ymax = Hi.95), fill = "red", alpha = 0.2)
```

```
# Adjust the x-axis labels
plotARIMA = plotARIMA + scale_x_discrete(breaks = combinedDataframeAMOC$Date[c(TRUE,
    rep(FALSE, 3))], labels = combinedDataframeAMOC$Date[c(TRUE, rep(FALSE,
    3))])

plotARIMA = plotARIMA + theme(axis.text.x = element_text(angle = 90, vjust = 0.5,
    size = 8))

# Display the plot
print(plotARIMA)
```



As we can see from the graph the ARIMA (2,0,0) seems to give us a sensible forecast for the 2021 quarter values, however as we can see the interval of the prediction accuracy our model is not too certain on the values most likely due to our model not accounting for the seasonal cycle of our data.

2 c)

Initial assumptions

From the previous exploratory analysis of the data we have established that the data did not need to be differentiated since it was constant, this translates to polynomial DLM component of order 2 that will use linear model to account for this type of changes in the data.

Furthermore, from the residual analysis we have inferred that there is an underlying seasonal trend present on the data, this seasonal trend will be represented by a seasonal component of frequency 4 to represent the 4 quarters per year.

model fitting

```
## linear model, order = 2, quadratic order = 3 , etc

## what we want is a linear model with a seasonal component so we add
## the 2 components together in a model

## things to try, another term like quadratic, or a arma component
## stacked on top of this

## Initial model with a linear polynomial and a seasonal component

buildFun = function(x) {
    dlmModPoly(order = 2, dV = exp(x[1]), dW = c(0, exp(x[2]))) + dlmModSeas(frequency = 4
        dV = 0, dW = c(exp(x[3]), rep(0, 2)))
}

linearDLM = dlmMLE(tsAtlantic, parm = c(0, 0, 0), build = buildFun)

linearDLM$par
```

```
[1]   1.151339 -18.078101  -2.189479
```

```
fittedLinearDLM = buildFun(linearDLM$par)

V(fittedLinearDLM)
```

```
         [,1]
[1,] 3.162425
```

```
W(fittedLinearDLM)
```

```
     [,1]         [,2]     [,3] [,4] [,5]
[1,]    0 0.000000e+00 0.000000    0    0
[2,]    0 1.408576e-08 0.000000    0    0
[3,]    0 0.000000e+00 0.111975    0    0
[4,]    0 0.000000e+00 0.000000    0    0
[5,]    0 0.000000e+00 0.000000    0    0
```

```r
## second model with a quadratic polynomial and a seasonal component

buildFunQuad = function(x) {
    dlmModPoly(order = 3, dV = exp(x[1]), dW = c(0, exp(x[2]), exp(x[3]))) +
        dlmModSeas(frequency = 4, dV = 0, dW = c(exp(x[4]), rep(0, 2)))
}

quadraticDLM = dlmMLE(tsAtlantic, parm = c(0, 0, 0, 0), build = buildFunQuad)

quadraticDLM$par
```

```
[1]   1.161355 -17.807081 -28.603103  -2.352292
```

```r
fittedQuadraticDLM = buildFunQuad(quadraticDLM$par)

V(fittedQuadraticDLM)
```

```
         [,1]
[1,] 3.194257
```

```r
W(fittedQuadraticDLM)
```

```
     [,1]         [,2]         [,3]       [,4] [,5] [,6]
[1,]    0 0.000000e+00 0.000000e+00 0.00000000    0    0
[2,]    0 1.847069e-08 0.000000e+00 0.00000000    0    0
[3,]    0 0.000000e+00 3.782948e-13 0.00000000    0    0
[4,]    0 0.000000e+00 0.000000e+00 0.09515082    0    0
[5,]    0 0.000000e+00 0.000000e+00 0.00000000    0    0
[6,]    0 0.000000e+00 0.000000e+00 0.00000000    0    0
```

Now we will compare both models through their log likelihood using the dlmLL function and see if the extra flexibility from the extra polynomial function is providing a better fit

```r
dlmLL(tsAtlantic, fittedLinearDLM)
```

```
[1] 94.98804
```

```r
dlmLL(tsAtlantic, fittedQuadraticDLM)
```

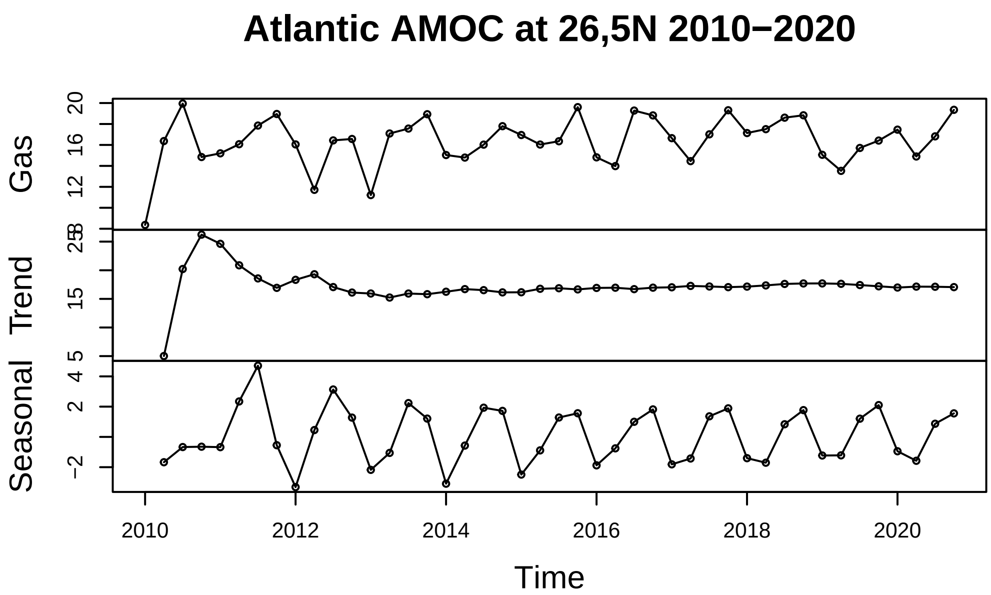
```
[1] 108.043
```

As we can see the dlm model using only a linear polynomial has a lower log likelihood than the model with an extra quadratic term, meaning this extra flexibility does not contribute to a better model fit and as such we will use the linear fitted model to do our forecasting.

```r
amocPredict = dlmFilter(tsAtlantic, mod = fittedLinearDLM)
summary(amocPredict)
```

```
    Length Class  Mode
y    44    ts     numeric
mod  10    dlm    list
m   225    mts    numeric
U.C  45    -none- list
D.C 225    -none- numeric
a   220    mts    numeric
U.R  44    -none- list
D.R 220    -none- numeric
f    44    ts     numeric
```

```r
x = cbind(tsAtlantic, dropFirst(amocPredict$a[, c(1, 3)]))
x = window(x, start = c(2010, 1))
colnames(x) = c("Gas", "Trend", "Seasonal")
plot(x, type = "o", main = "Atlantic AMOC at 26,5N 2010-2020")
```



Forecast

```
amocForecast = dlmForecast(amocPredict, nAhead = 4)
summary(amocForecast)
```

```
  Length Class  Mode
a 20     mts    numeric
R  4     -none- list
f  4     ts     numeric
Q  4     -none- list
```

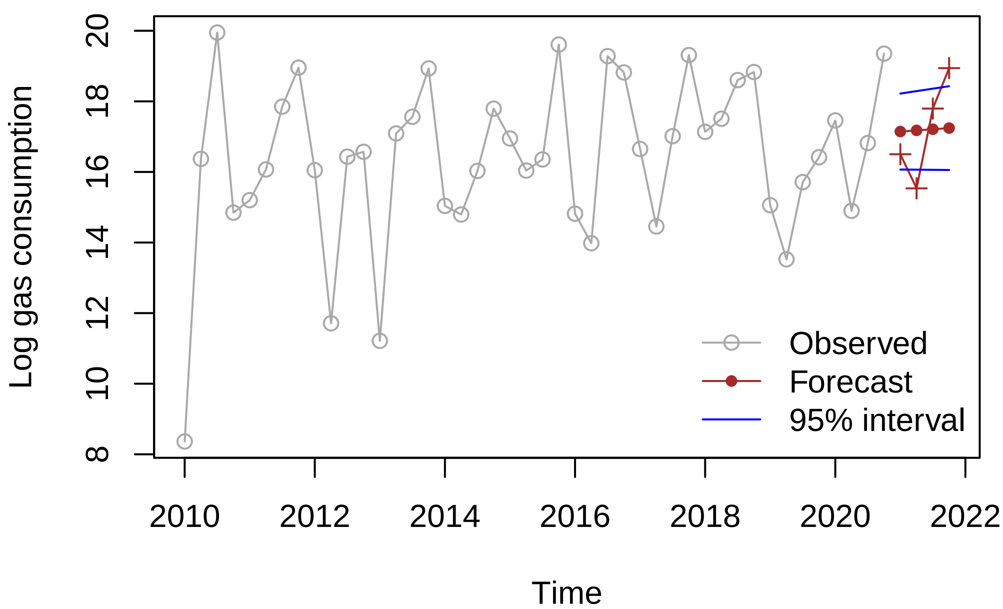
```
dim(amocForecast$a)
```

```
[1] 4 5
```

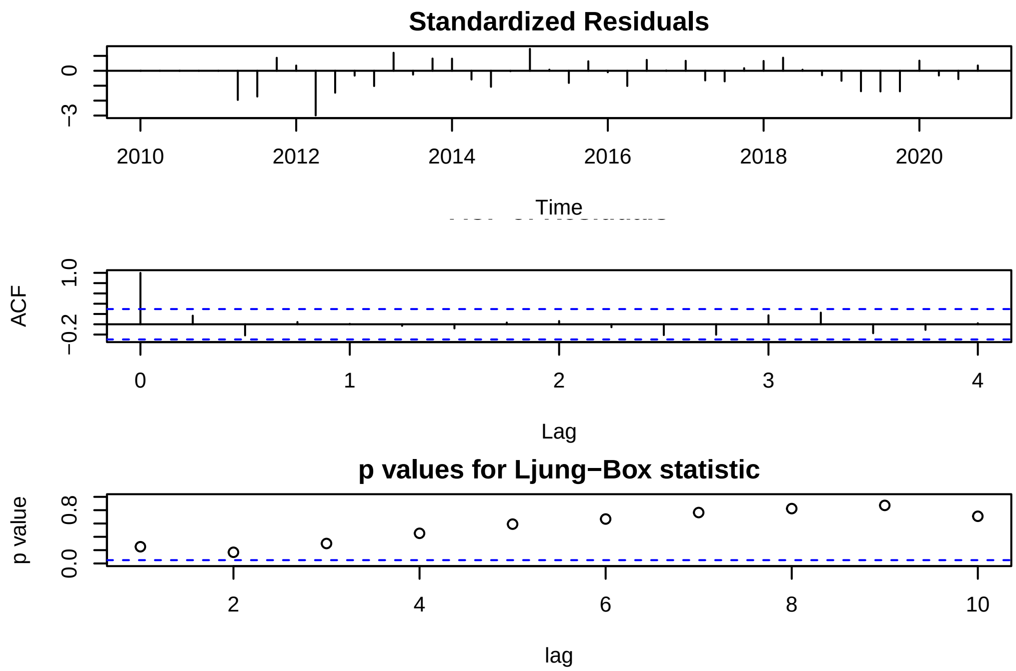
```
dim(amocForecast$f)
```

```
[1] 4 1
```

```r
sqrtR = sapply(amocForecast$R, function(x) sqrt(x[1, 1]))
pl = amocForecast$a[, 1] + qnorm(0.025, sd = sqrtR)
pu = amocForecast$a[, 1] + qnorm(0.975, sd = sqrtR)
x = ts.union(window(tsAtlantic, start = c(2010, 1)), amocForecast$a[, 1],
    amocForecast$f, pl, pu)
par(mar = c(4, 4, 2, 2))
plot(x, plot.type = "single", type = "o", pch = c(1, 20, 3, NA, NA), col = c("darkgrey",
    "brown", "brown", "blue", "blue"), ylab = "Log gas consumption")

legend("bottomright", legend = c("Observed", "Forecast", "95% interval"),
    bty = "n", pch = c(1, 20, NA), lty = 1, col = c("darkgrey", "brown",
        "blue"))
```




```r
# Set smaller margins
par(mar = c(4, 4, 2, 2))

tsdiag(amocPredict)
```

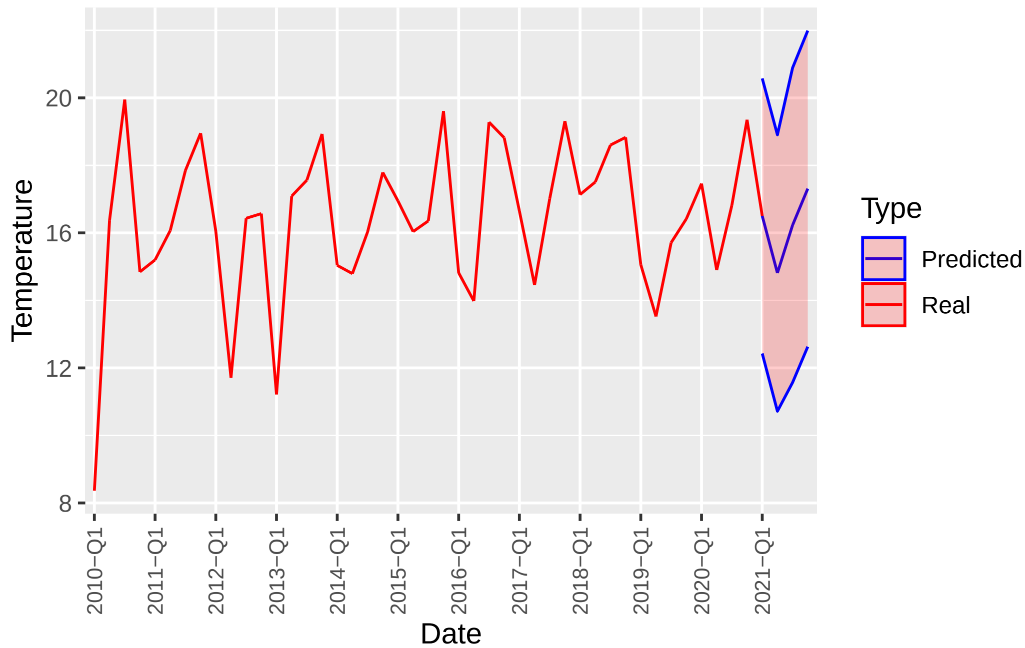



```r
# Reset margins
par(mar = c(5, 4, 4, 2) + 0.1)
```

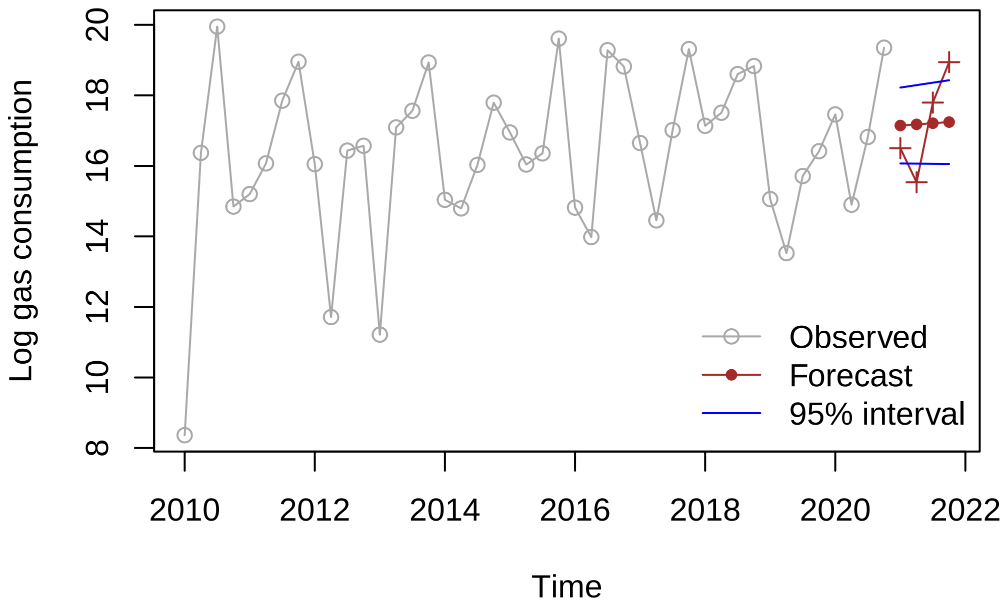
2 d)

Again comparing the forecast values and their respective prediction intervals as we can see from the graphs bellow the dlm model has smaller prediction intervals, most likely due to being able to explain the underlying seasonal trend reducing therefore the uncertainty in comparison the ARIMA model.

```r
print(plotARIMA)
```




```r
sqaretRoot = sapply(amocForecast$R, function(x) sqrt(x[1, 1]))
predictionLow = amocForecast$a[, 1] + qnorm(0.025, sd = sqaretRoot)  ## Low
predictionUpper = amocForecast$a[, 1] + qnorm(0.975, sd = sqaretRoot)  ## Upper
x = ts.union(window(tsAtlantic, start = c(2010, 1)), amocForecast$a[, 1],
    amocForecast$f, predictionLow, predictionUpper)
par(mar = c(4, 4, 2, 2))
plot(x, plot.type = "single", type = "o", pch = c(1, 20, 3, NA, NA), col = c("darkgrey",
    "brown", "brown", "blue", "blue"), ylab = "Log gas consumption")

legend("bottomright", legend = c("Observed", "Forecast", "95% interval"),
    bty = "n", pch = c(1, 20, NA), lty = 1, col = c("darkgrey", "brown",
        "blue"))
```

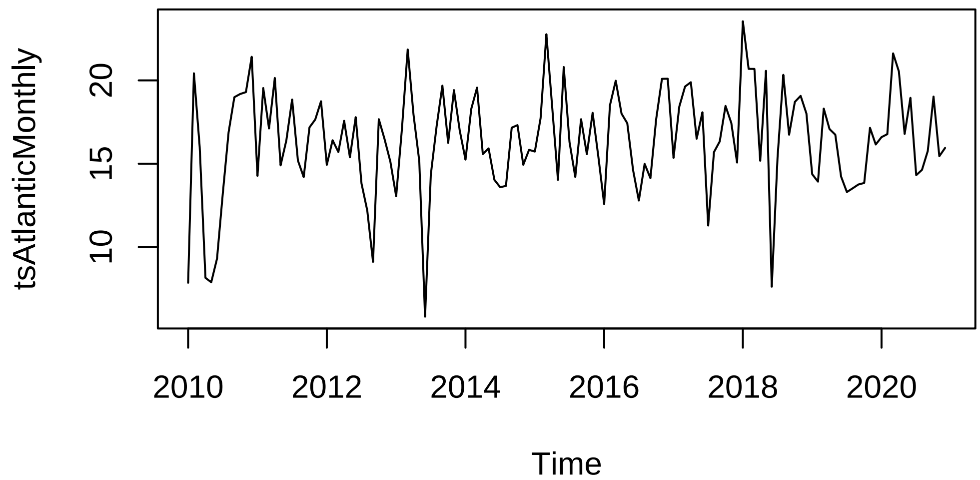


2 e)

```
# AMOCDFMonthly =AMOCDF %>% mutate(YearMonth = paste0(year(Date), '-',
# month(Date, label = TRUE, abbr = FALSE)))


## I will now make a column with the month and year that I will use to
## create the monthly averages
AMOCDF$YearMonth = paste(AMOCDF$Year, AMOCDF$Month, sep = "-")

YearMonthlyAverage = AMOCDF %>%
    group_by(YearMonth) %>%
    summarise(AverageStrength = mean(Strength))
```

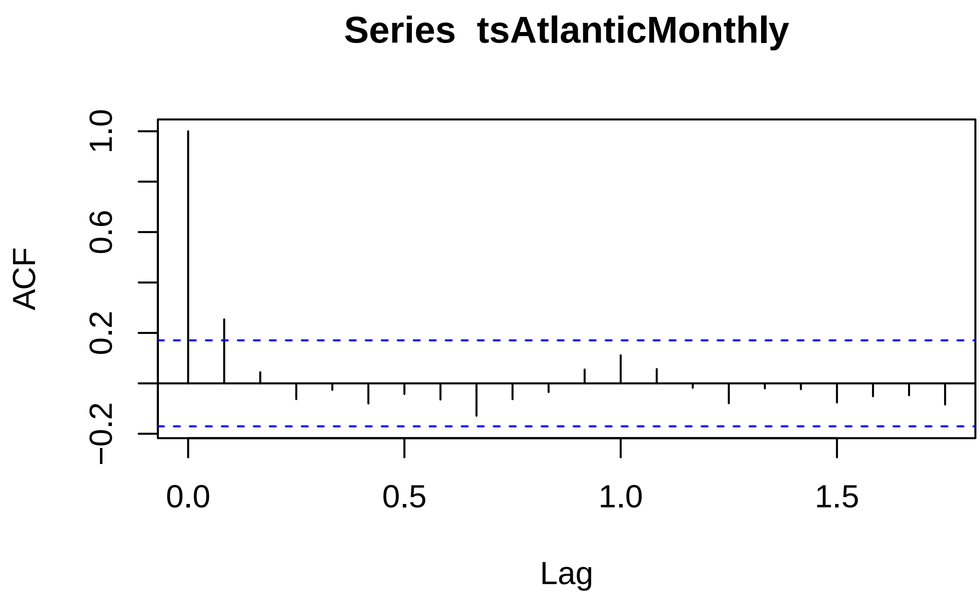
Now we will create a new montly time series object and make it univariate

```
tsAtlanticMonthly = ts(YearMonthlyAverage, start = c(2010, 1), frequency = 12)

tsAtlanticMonthly = tsAtlanticMonthly[, "AverageStrength"]

plot.ts(tsAtlanticMonthly)
```

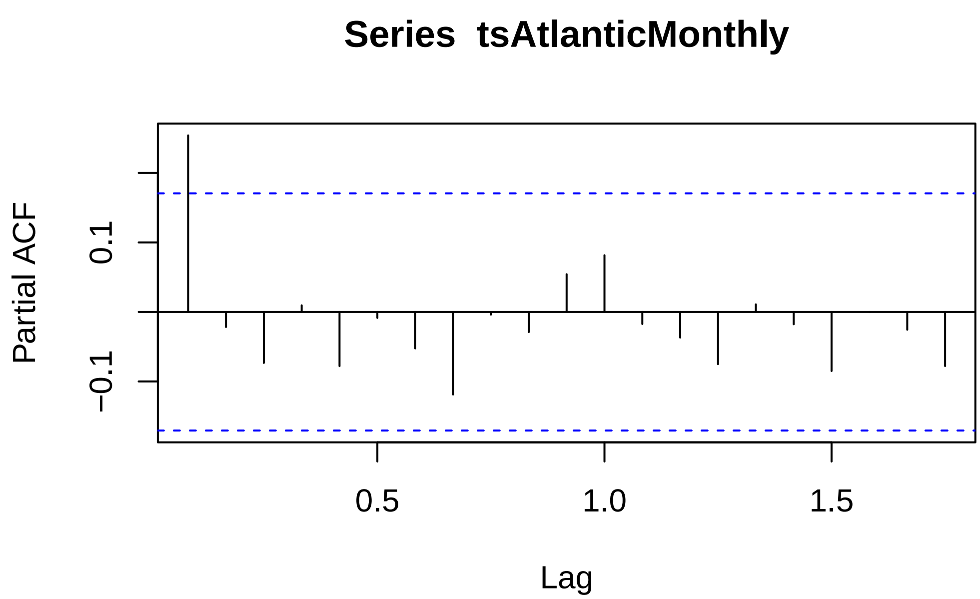


Seeing this graph we can observe that the data continues being stationary for the ARIMA model but a seasonal trend not only is more apparently but it also appear to need to be differentiated as it seems to have a decreasing linear trend

```
acf(tsAtlanticMonthly)
```




```r
pacf(tsAtlanticMonthly)
```



The acf has a very clear cut-off as only 3 the values are significant which is very similar to what we had observed previously.

The main difference is in the pacf, where we can now say for sure that there is a very clear cut-off after the first value.

Model testing

These pattern suggests that an ARMA/ARIMA model might be the most appropriate so first we will check them out with the seasonal component of order 1, the so quick cut-off of both the ACF and PACF also might suggest that p and q will be smaller values.

Seasonal check

```
## initial assumption


modelMonthlySeasonal100.110 = Arima(tsAtlanticMonthly, order = c(1, 0, 0),
    seasonal = list(order = c(1, 1, 0), period = 12))

modelMonthlySeasonal100.011 = Arima(tsAtlanticMonthly, order = c(1, 0, 0),
    seasonal = list(order = c(0, 1, 1), period = 12))

modelMonthlySeasonal200.210 = Arima(tsAtlanticMonthly, order = c(2, 0, 0),
    seasonal = list(order = c(2, 1, 0), period = 12))
modelMonthlySeasonal200.012 = Arima(tsAtlanticMonthly, order = c(2, 0, 0),
    seasonal = list(order = c(0, 1, 2), period = 12))

modelMonthlySeasonal001.110 = Arima(tsAtlanticMonthly, order = c(0, 0, 1),
    seasonal = list(order = c(1, 1, 0), period = 12))

modelMonthlySeasonal001.011 = Arima(tsAtlanticMonthly, order = c(0, 0, 1),
    seasonal = list(order = c(0, 1, 1), period = 12))

modelMonthlySeasonal002.210 = Arima(tsAtlanticMonthly, order = c(0, 0, 2),
    seasonal = list(order = c(2, 1, 0), period = 12))
modelMonthlySeasonal002.012 = Arima(tsAtlanticMonthly, order = c(0, 0, 2),
    seasonal = list(order = c(0, 1, 2), period = 12))
```

```
modelMonthlySeasonal100.110
```

```
Series: tsAtlanticMonthly
ARIMA(1,0,0)(1,1,0)[12]

Coefficients:
         ar1     sar1
      0.1779  -0.4618
s.e.  0.0909   0.0839

sigma^2 = 12.06:  log likelihood = -320.1
AIC=646.2   AICc=646.4   BIC=654.56
```

```
modelMonthlySeasonal100.011
```

```
Series: tsAtlanticMonthly
ARIMA(1,0,0)(0,1,1)[12]

Coefficients:
         ar1     sma1
      0.1844  -0.8500
s.e.  0.0918   0.1123

sigma^2 = 8.74:  log likelihood = -306.88
AIC=619.76   AICc=619.97   BIC=628.12
```

```
modelMonthlySeasonal200.210
```

```
Series: tsAtlanticMonthly
ARIMA(2,0,0)(2,1,0)[12]

Coefficients:
        ar1     ar2     sar1     sar2
      0.115  0.0374  -0.7271  -0.4814
s.e.  0.094  0.0930   0.0932   0.0933

sigma^2 = 9.741:  log likelihood = -309.65
AIC=629.29   AICc=629.82   BIC=643.23
```

```
modelMonthlySeasonal200.012
```

```
Series: tsAtlanticMonthly
ARIMA(2,0,0)(0,1,2)[12]

Coefficients:
         ar1     ar2     sma1    sma2
      0.1566  0.0546  -0.9817  0.1895
s.e.  0.0947  0.0950   0.1384  0.1503

sigma^2 = 8.778:  log likelihood = -305.88
AIC=621.76   AICc=622.29   BIC=635.7
```

```r
modelMonthlySeasonal001.110
```

```
Series: tsAtlanticMonthly
ARIMA(0,0,1)(1,1,0)[12]

Coefficients:
         ma1     sar1
      0.1680  -0.4634
s.e.  0.0861   0.0840

sigma^2 = 12.07:  log likelihood = -320.19
AIC=646.39   AICc=646.59   BIC=654.75
```

```r
modelMonthlySeasonal001.011
```

```
Series: tsAtlanticMonthly
ARIMA(0,0,1)(0,1,1)[12]

Coefficients:
         ma1     sma1
      0.1606  -0.8446
s.e.  0.0847   0.1093

sigma^2 = 8.804:  log likelihood = -307.14
AIC=620.28   AICc=620.49   BIC=628.65
```

```r
modelMonthlySeasonal002.210
```

```
Series: tsAtlanticMonthly
ARIMA(0,0,2)(2,1,0)[12]

Coefficients:
         ma1     ma2     sar1     sar2
      0.1145  0.0452  -0.7275  -0.4806
s.e.  0.0943  0.0882   0.0933   0.0936

sigma^2 = 9.745:  log likelihood = -309.66
AIC=629.33   AICc=629.85   BIC=643.26
```

```r
modelMonthlySeasonal002.012
```

```
Series: tsAtlanticMonthly
ARIMA(0,0,2)(0,1,2)[12]

Coefficients:
         ma1     ma2     sma1    sma2
      0.1583  0.0745  -0.9786  0.1868
s.e.  0.0953  0.0893   0.1377  0.1495

sigma^2 = 8.784:  log likelihood = -305.89
AIC=621.78   AICc=622.3   BIC=635.72
```

So as suspected from both the time series plot and the last exercise analysis the added seasonality does increase our model goodness of fit while also penalising the increased in complexity with so far.

Now lets compare them to bigger p and q values to see if our initial assumptions do hold up

```r
modelMonthlySeasonal301 = Arima(tsAtlanticMonthly, order = c(3, 0, 1), seasonal = list(ord
    1, 1), period = 12))
modelMonthlySeasonal302 = Arima(tsAtlanticMonthly, order = c(3, 0, 2), seasonal = list(ord
    1, 0), period = 12))
modelMonthlySeasonal303 = Arima(tsAtlanticMonthly, order = c(3, 1, 3), seasonal = list(ord
    1, 0), period = 12))

modelMonthlySeasonal103 = Arima(tsAtlanticMonthly, order = c(1, 0, 3), seasonal = list(ord
    1, 1), period = 12))
modelMonthlySeasonal203 = Arima(tsAtlanticMonthly, order = c(2, 1, 3), seasonal = list(ord
    1, 0), period = 12))
```

```r
modelMonthlySeasonal301
```

```
Series: tsAtlanticMonthly
ARIMA(3,0,1)(1,1,1)[12]

Coefficients:
          ar1     ar2      ar3     ma1     sar1     sma1
      -0.5728  0.1878  -0.0048  0.7442  -0.1158  -0.8073
s.e.   0.7595  0.1720   0.1369  0.7564   0.1224   0.1140

sigma^2 = 8.939:  log likelihood = -306.02
AIC=626.04   AICc=627.04   BIC=645.55
```

```r
modelMonthlySeasonal302
```

```
Series: tsAtlanticMonthly
ARIMA(3,0,2)(1,1,0)[12]

Coefficients:
          ar1      ar2     ar3     ma1     ma2     sar1
      -1.4367  -0.5902  0.1291  1.6983  1.0000  -0.4382
s.e.   0.0944   0.1532  0.0945  0.0436  0.0489   0.0871

sigma^2 = 11.34:  log likelihood = -316.59
AIC=647.19   AICc=648.19   BIC=666.7
```

```r
modelMonthlySeasonal303
```

```
Series: tsAtlanticMonthly
ARIMA(3,1,3)(1,1,0)[12]

Coefficients:
          ar1      ar2     ar3     ma1      ma2      ma3     sar1
      -1.4279  -0.5751  0.1373  0.6984  -0.6984  -1.0000  -0.4313
s.e.   0.0952   0.1547  0.0954  0.0533   0.0552   0.0576   0.0877

sigma^2 = 11.54:  log likelihood = -316.86
AIC=649.73   AICc=651.04   BIC=671.96
```

```r
modelMonthlySeasonal103
```

```
Series: tsAtlanticMonthly
ARIMA(1,0,3)(1,1,1)[12]

Coefficients:
          ar1     ma1     ma2      ma3     sar1     sma1
      -0.6951  0.8708  0.1931  -0.0013  -0.1134  -0.8029
s.e.   0.4854  0.4812  0.1503   0.1321   0.1194   0.1136

sigma^2 = 8.954:  log likelihood = -306.02
AIC=626.03   AICc=627.03   BIC=645.55
```

```r
modelMonthlySeasonal203
```

```
Series: tsAtlanticMonthly
ARIMA(2,1,3)(1,1,0)[12]

Coefficients:
          ar1      ar2     ma1      ma2      ma3     sar1
      -1.3087  -0.6017  0.5001  -0.6702  -0.8299  -0.4545
s.e.   0.2576   0.1557  0.2045   0.1332   0.1180   0.0979

sigma^2 = 12.02:  log likelihood = -318.1
AIC=650.19   AICc=651.2   BIC=669.65
```

As we can see here the initial assumption that a smaller p and q value would better fit the model.

Now we will use auto.arima to verify if our assumptions were indeed correct

auto arima check

```r
`?`(auto.arima)
```

```
starting httpd help server ... done
```

```r
monthlyModelAuto = auto.arima(tsAtlanticMonthly, max.d = 0, max.p = 5, max.q = 5,
    D = 1)
monthlyModelAuto
```

```
Series: tsAtlanticMonthly
ARIMA(1,0,0)(0,1,1)[12]

Coefficients:
         ar1     sma1
      0.1844  -0.8500
s.e.  0.0918   0.1123

sigma^2 = 8.74:  log likelihood = -306.88
AIC=619.76   AICc=619.97   BIC=628.12
```

From what we can see the auto arima has indeed confirmed our initial assumption by picking a model that we already had seen as the best performer ARIMA(1,0,0)(0,1,1)[12]

Forecasting

Now using the forecast function we will forecast the next 4 quarters of 2021

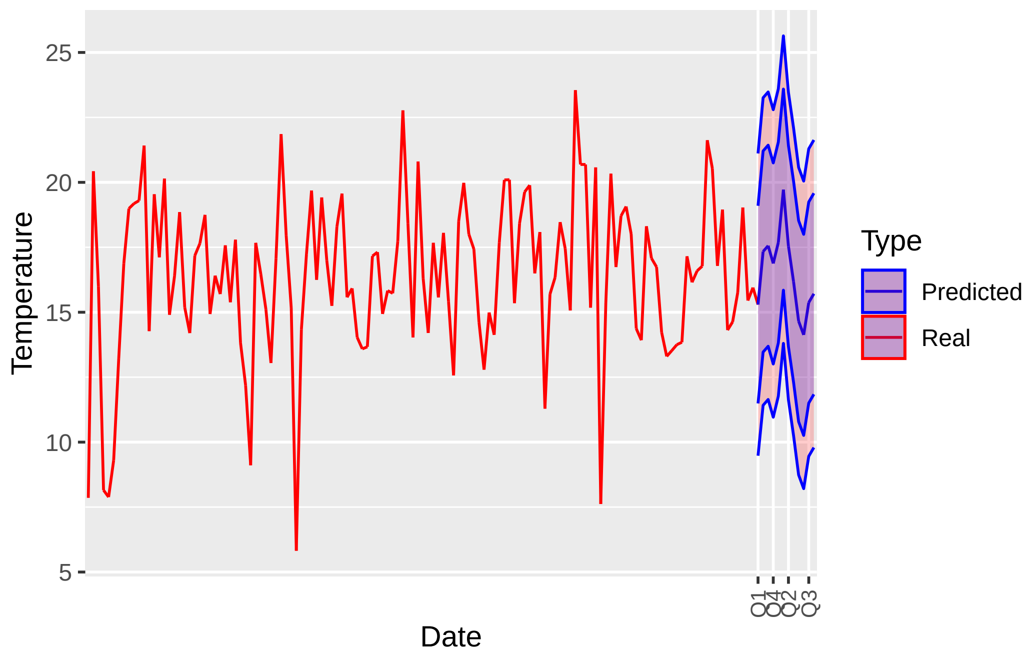
```r
forecast(modelMonthlySeasonal100.011, 12)
```

```
         Point Forecast    Lo 80    Hi 80     Lo 95    Hi 95
Jan 2021       15.29474 11.49064 19.09884  9.476872 21.11261
Feb 2021       17.68442 13.81636 21.55247 11.768741 23.60009
Mar 2021       19.72032 15.85011 23.59053 13.801347 25.63929
Apr 2021       17.53922 13.66894 21.40951 11.620140 23.45831
May 2021       16.17383 12.30355 20.04411 10.254741 22.09292
Jun 2021       14.65775 10.78746 18.52803  8.738657 20.57683
Jul 2021       14.12800 10.25772 17.99829  8.208914 20.04709
Aug 2021       15.37476 11.50447 19.24504  9.455670 21.29385
Sep 2021       15.71081 11.84052 19.58109  9.791718 21.62989
Oct 2021       17.33660 13.46632 21.20689 11.417514 23.25569
Nov 2021       17.56114 13.69086 21.43141 11.642055 23.48022
Dec 2021       16.87677 13.00663 20.74691 10.957906 22.79563
```

But this data is better visualized in a graph to better understand if the predictions are sensible compared to our real data.

```r
predictedArimaSeasonalDF = data.frame(forecast(modelMonthlySeasonal100.011,
    12))

predictedArimaSeasonalDF$YearMonth = c("2021-1", "2021-2", "2021-3", "2021-4",
    "2021-5", "2021-6", "2021-7", "2021-8", "2021-9", "2021-10", "2021-11",
    "2021-12")

predictedArimaSeasonalDF$Type = "Predicted"

predictedArimaSeasonalDF$Temperature = predictedArimaSeasonalDF$Point.Forecast

YearMonthlyAverage$Type = "Real"
YearMonthlyAverage$Temperature = YearMonthlyAverage$AverageStrength


# Combine real_data and pred_data into a single data frame
combinedDataframeAMOC = rbind(data.frame(Date = YearMonthlyAverage$YearMonth,
    Temperature = YearMonthlyAverage$Temperature, Type = "Real"), data.frame(Date = predic
    Temperature = predictedArimaSeasonalDF$Temperature, Type = "Predicted"))
```

```r
# Create the ggplot
plotARIMA2 = ggplot(combinedDataframeAMOC, aes(x = Date, y = Temperature,
    color = Type, group = 1)) + geom_line() + scale_color_manual(values = c("blue",
    "red"))

# Add the 80% and 95% confidence intervals
plotARIMA2 = plotARIMA2 + geom_ribbon(data = predictedArimaSeasonalDF, aes(x = YearMonth,
    ymin = Lo.95, ymax = Hi.95), fill = "red", alpha = 0.2) + geom_ribbon(data = predicted
    aes(x = YearMonth, ymin = Lo.80, ymax = Hi.80), fill = "blue", alpha = 0.2)

# Adjust the x-axis labels
plotARIMA2 = plotARIMA2 + scale_x_discrete(breaks = c("2021-1", "2021-4",
    "2021-8", "2021-12"), labels = c("Q1", "Q2", "Q3", "Q4"))

plotARIMA2 = plotARIMA2 + theme(axis.text.x = element_text(angle = 90, vjust = 0.5,
    size = 8))

# Display the plot
print(plotARIMA2)
```



DLM

model fitting

```
## linear model, order = 2, quadratic order = 3 , etc

## what we want is a linear model with a seasonal component so we add
## the 2 components together in a model

## things to try, another term like quadratic, or a arma component
## stacked on top of this

## Initial model with a linear polynomial and a seasonal component

buildFun = function(x) {
    dlmModPoly(order = 2, dV = exp(x[1]), dW = c(0, exp(x[2]))) + dlmModSeas(frequency = 1
        dV = 0, dW = c(exp(x[3]), rep(0, 10)))
}

linearDLM = dlmMLE(tsAtlanticMonthly, parm = c(0, 0, 0), build = buildFun)
```

```r
linearDLM$par
```

```
[1]   2.044026 -11.586366  -4.421181
```

```r
fittedLinearDLM = buildFun(linearDLM$par)
V(fittedLinearDLM)
```

```
         [,1]
[1,] 7.721632
```

```r
W(fittedLinearDLM)
```

```
      [,1]         [,2]       [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
 [1,]    0 0.000000e+00 0.00000000    0    0    0    0    0    0     0     0
 [2,]    0 9.291914e-06 0.00000000    0    0    0    0    0    0     0     0
 [3,]    0 0.000000e+00 0.01202003    0    0    0    0    0    0     0     0
 [4,]    0 0.000000e+00 0.00000000    0    0    0    0    0    0     0     0
 [5,]    0 0.000000e+00 0.00000000    0    0    0    0    0    0     0     0
 [6,]    0 0.000000e+00 0.00000000    0    0    0    0    0    0     0     0
 [7,]    0 0.000000e+00 0.00000000    0    0    0    0    0    0     0     0
 [8,]    0 0.000000e+00 0.00000000    0    0    0    0    0    0     0     0
 [9,]    0 0.000000e+00 0.00000000    0    0    0    0    0    0     0     0
[10,]    0 0.000000e+00 0.00000000    0    0    0    0    0    0     0     0
[11,]    0 0.000000e+00 0.00000000    0    0    0    0    0    0     0     0
[12,]    0 0.000000e+00 0.00000000    0    0    0    0    0    0     0     0
[13,]    0 0.000000e+00 0.00000000    0    0    0    0    0    0     0     0
      [,12] [,13]
 [1,]     0     0
 [2,]     0     0
 [3,]     0     0
 [4,]     0     0
 [5,]     0     0
 [6,]     0     0
 [7,]     0     0
 [8,]     0     0
 [9,]     0     0
[10,]     0     0
[11,]     0     0
```

```
[12,]    0    0
[13,]    0    0
```

```r
## second model with a quadratic polynomial and a seasonal component

buildFunQuad = function(x) {
    dlmModPoly(order = 3, dV = exp(x[1]), dW = c(0, exp(x[2]), exp(x[3]))) +
        dlmModSeas(frequency = 12, dV = 0, dW = c(exp(x[4]), rep(0, 10)))
}

quadraticDLM = dlmMLE(tsAtlanticMonthly, parm = c(0, 0, 0, 0), build = buildFunQuad)

quadraticDLM$par
```

```
[1]   2.047135 -21.060474 -56.104784 -12.300531
```

```r
fittedQuadraticDLM = buildFunQuad(quadraticDLM$par)

V(fittedQuadraticDLM)
```

```
         [,1]
[1,] 7.745678
```

```r
W(fittedQuadraticDLM)
```

```
      [,1]         [,2]         [,3]         [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]    0 0.000000e+00 0.000000e+00 0.000000e+00    0    0    0    0    0
 [2,]    0 7.137603e-10 0.000000e+00 0.000000e+00    0    0    0    0    0
 [3,]    0 0.000000e+00 4.305286e-25 0.000000e+00    0    0    0    0    0
 [4,]    0 0.000000e+00 0.000000e+00 4.549326e-06    0    0    0    0    0
 [5,]    0 0.000000e+00 0.000000e+00 0.000000e+00    0    0    0    0    0
 [6,]    0 0.000000e+00 0.000000e+00 0.000000e+00    0    0    0    0    0
 [7,]    0 0.000000e+00 0.000000e+00 0.000000e+00    0    0    0    0    0
 [8,]    0 0.000000e+00 0.000000e+00 0.000000e+00    0    0    0    0    0
 [9,]    0 0.000000e+00 0.000000e+00 0.000000e+00    0    0    0    0    0
[10,]    0 0.000000e+00 0.000000e+00 0.000000e+00    0    0    0    0    0
[11,]    0 0.000000e+00 0.000000e+00 0.000000e+00    0    0    0    0    0
[12,]    0 0.000000e+00 0.000000e+00 0.000000e+00    0    0    0    0    0
```

```
[13,]    0 0.000000e+00 0.000000e+00 0.000000e+00    0    0    0    0    0
[14,]    0 0.000000e+00 0.000000e+00 0.000000e+00    0    0    0    0    0
      [,10] [,11] [,12] [,13] [,14]
 [1,]     0     0     0     0     0
 [2,]     0     0     0     0     0
 [3,]     0     0     0     0     0
 [4,]     0     0     0     0     0
 [5,]     0     0     0     0     0
 [6,]     0     0     0     0     0
 [7,]     0     0     0     0     0
 [8,]     0     0     0     0     0
 [9,]     0     0     0     0     0
[10,]     0     0     0     0     0
[11,]     0     0     0     0     0
[12,]     0     0     0     0     0
[13,]     0     0     0     0     0
[14,]     0     0     0     0     0
```

Now we will compare both models through their log likelihood using the dlmLL function and see if the extra flexibility from the extra polynomial function is providing a better fit

```r
dlmLL(tsAtlanticMonthly, fittedLinearDLM)
```

```
[1] 309.5446
```

```r
dlmLL(tsAtlanticMonthly, fittedQuadraticDLM)
```

```
[1] 324.4748
```

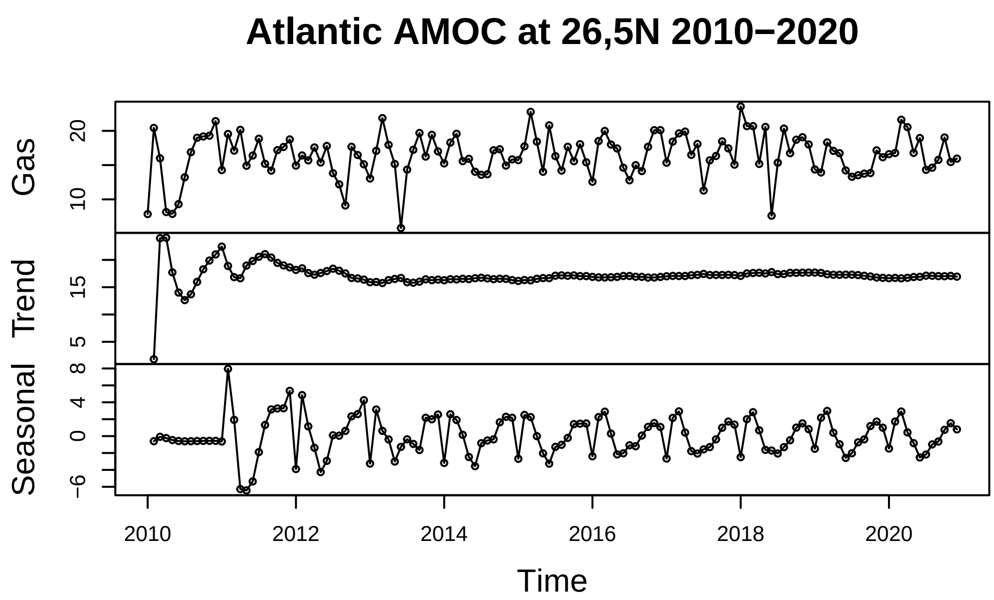
As we can see the dlm model using only a linear polynomial has a lower log likelihood than the model with an extra quadratic term, meaning this extra flexibility does not contribute to a better model fit and as such we will use the linear fitted model to do our forecasting.

```r
amocPredict = dlmFilter(tsAtlanticMonthly, mod = fittedLinearDLM)
summary(amocPredict)
```

```
    Length Class  Mode
y    132   ts     numeric
mod   10   dlm    list
```

```
m   1729   mts    numeric
U.C  133   -none- list
D.C 1729   -none- numeric
a   1716   mts    numeric
U.R  132   -none- list
D.R 1716   -none- numeric
f    132   ts     numeric
```

```
x = cbind(tsAtlanticMonthly, dropFirst(amocPredict$a[, c(1, 3)]))
x = window(x, start = c(2010, 1))
colnames(x) = c("Gas", "Trend", "Seasonal")
plot(x, type = "o", main = "Atlantic AMOC at 26,5N 2010-2020")
```



Forecast

```
amocForecastMonthly = dlmForecast(amocPredict, nAhead = 12)
summary(amocForecastMonthly)
```

```
  Length Class  Mode
a 156    mts    numeric
```

```
R  12     -none- list
f  12     ts     numeric
Q  12     -none- list
```

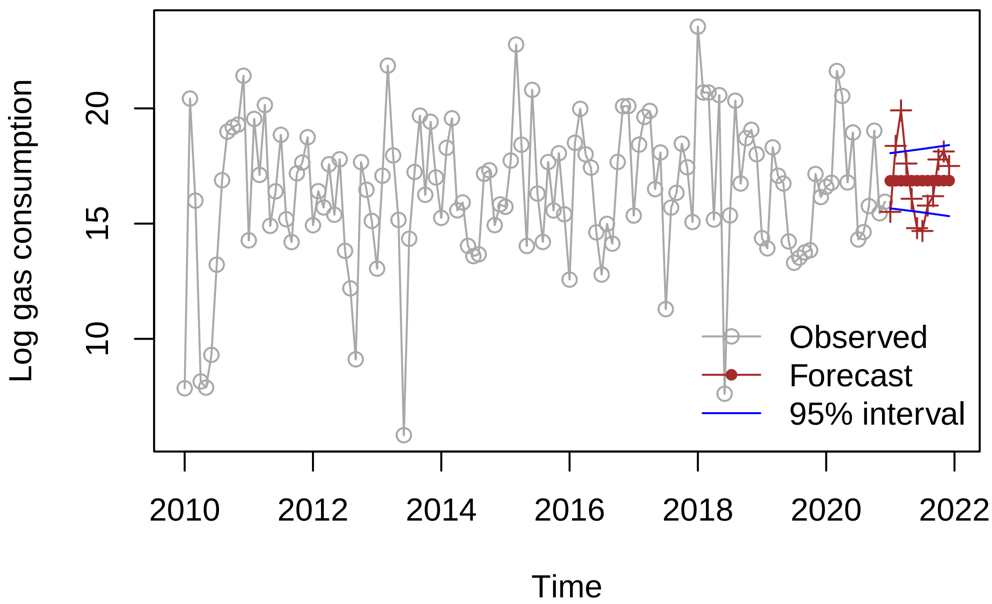
```r
dim(amocForecastMonthly$a)
```

```
[1] 12 13
```

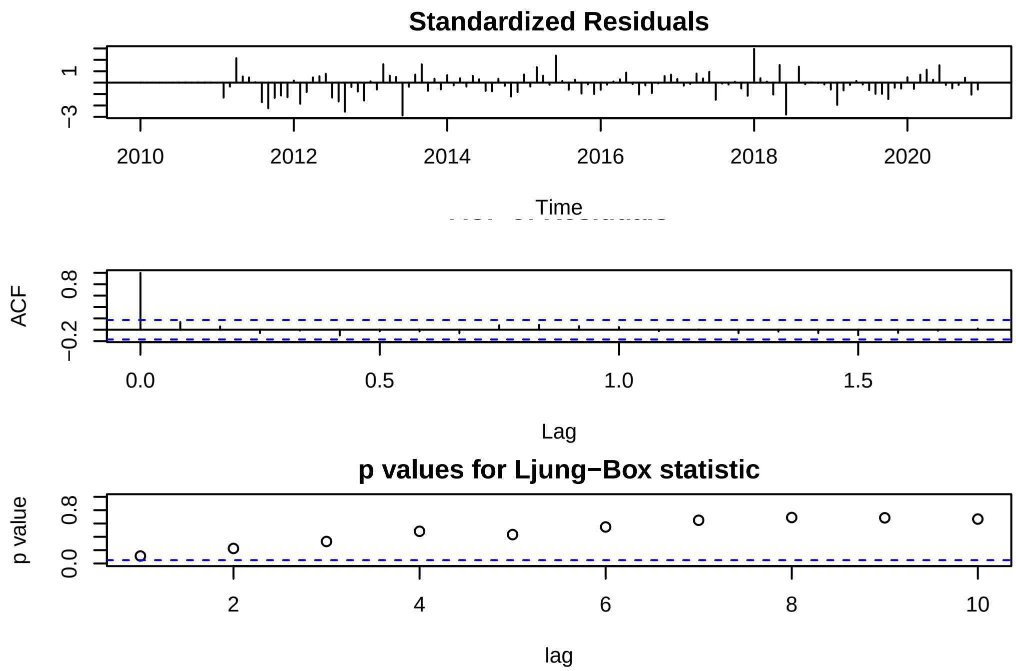
```r
dim(amocForecastMonthly$f)
```

```
[1] 12  1
```

```r
sqrtR = sapply(amocForecastMonthly$R, function(x) sqrt(x[1, 1]))
pl = amocForecastMonthly$a[, 1] + qnorm(0.025, sd = sqrtR)
pu = amocForecastMonthly$a[, 1] + qnorm(0.975, sd = sqrtR)

x = ts.union(window(tsAtlanticMonthly, start = c(2010, 1)), amocForecastMonthly$a[,
    1], amocForecastMonthly$f, pl, pu)
par(mar = c(4, 4, 2, 2))
plot(x, plot.type = "single", type = "o", pch = c(1, 20, 3, NA, NA), col = c("darkgrey",
    "brown", "brown", "blue", "blue"), ylab = "Log gas consumption")

legend("bottomright", legend = c("Observed", "Forecast", "95% interval"),
    bty = "n", pch = c(1, 20, NA), lty = 1, col = c("darkgrey", "brown",
        "blue"))
```




```r
# Set smaller margins
par(mar = c(4, 4, 2, 2))

tsdiag(amocPredict)
```




```
# Reset margins
par(mar = c(5, 4, 4, 2) + 0.1)
```

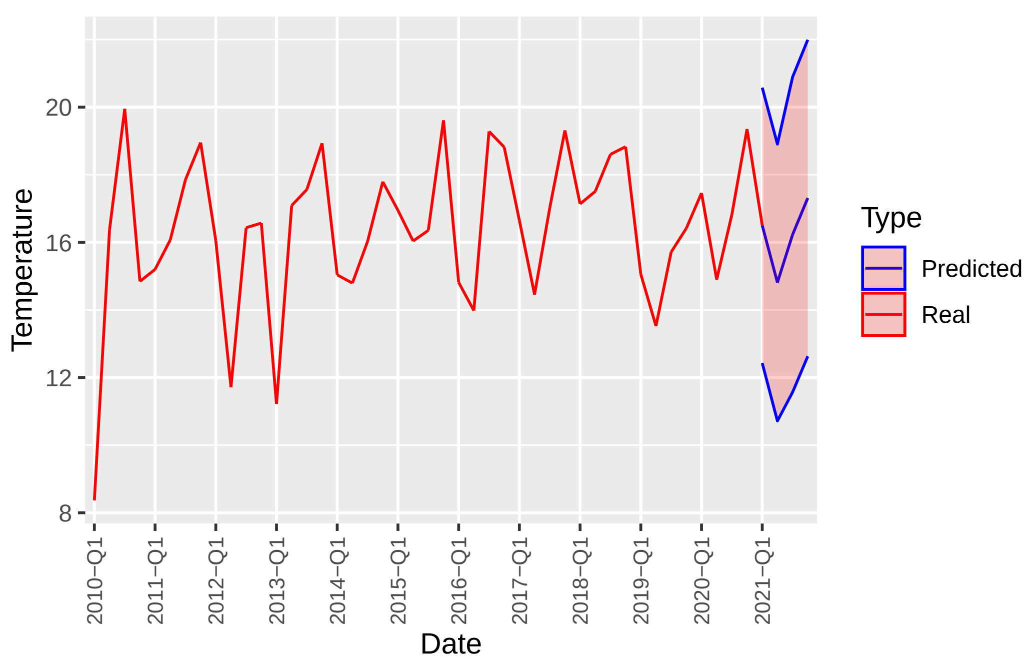
Lastly checking the residuals, they seem to be mostly normally distributed with a good mixture of over and under estimations, especially in the middle with some slight seasonality on both ends being present

2 f)

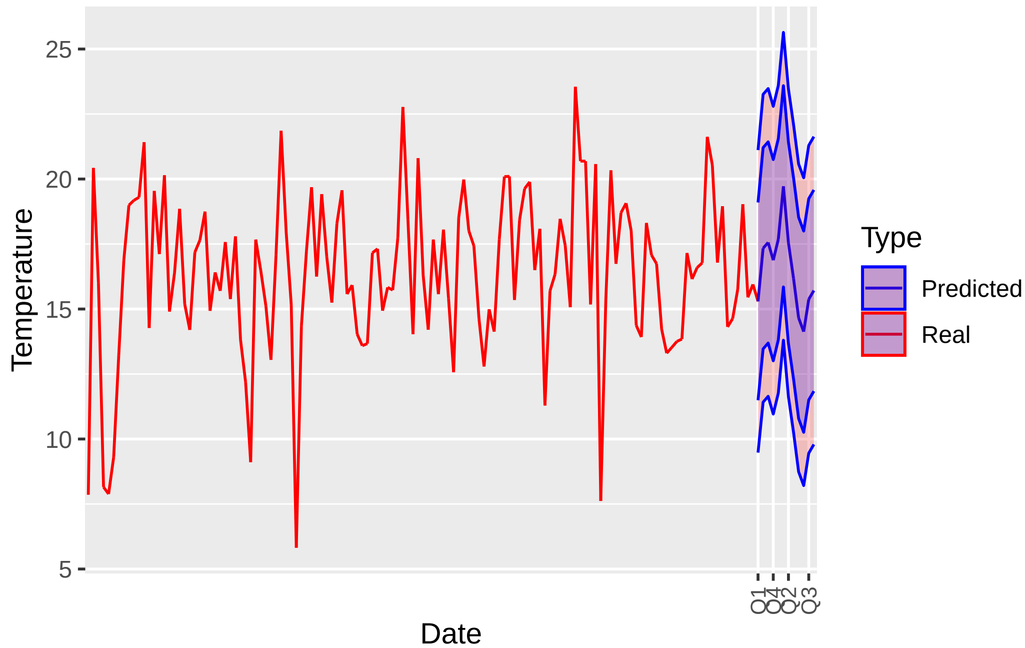
Now starting with the ARIMA models

```
predictedArimaDF = data.frame(forecast(model200, 4))

predictedArimaDF$YearQuarter = c("2021-Q1", "2021-Q2", "2021-Q3", "2021-Q4")

# Combine real_data and pred_data into a single data frame
combinedDataframeAMOC = rbind(data.frame(Date = YearQuarterAverage$YearQuarter,
    Temperature = YearQuarterAverage$AverageStrength, Type = "Real"), data.frame(Date = pr
    Temperature = predictedArimaDF$Point.Forecast, Type = "Predicted"))

predictedArimaDF$Temperature = predictedArimaDF$Point.Forecast
```

```r
predictedArimaDF$Type = "Predicted"


# Create the ggplot
plotARIMA = ggplot(combinedDataframeAMOC, aes(x = Date, y = Temperature,
    color = Type, group = 1)) + geom_line() + scale_color_manual(values = c("blue",
    "red"))

# Add the 95% confidence interval
plotARIMA = plotARIMA + geom_ribbon(data = predictedArimaDF, aes(x = YearQuarter,
    ymin = Lo.95, ymax = Hi.95), fill = "red", alpha = 0.2)

# Adjust the x-axis labels
plotARIMA = plotARIMA + scale_x_discrete(breaks = combinedDataframeAMOC$Date[c(TRUE,
    rep(FALSE, 3))], labels = combinedDataframeAMOC$Date[c(TRUE, rep(FALSE,
    3))])

plotARIMA = plotARIMA + theme(axis.text.x = element_text(angle = 90, vjust = 0.5,
    size = 8))

print(plotARIMA)
```



```
print(plotARIMA2)
```



The most obvious difference is the level of detail on the seasonality, with the monthly averages capturing an almost opposite effect than the quarterly averages in both the predicted and also in some of the real data, the forecast also has the opposite trend, with an actual expected decrease in Sverdrups around between the 2nd quarter and the mid 3rd quarter which again seems to follow the opposite trend on the quartely data.

```
sqrtR = sapply(amocForecastMonthly$R, function(x) sqrt(x[1, 1]))
pl = amocForecastMonthly$a[, 1] + qnorm(0.025, sd = sqrtR)
pu = amocForecastMonthly$a[, 1] + qnorm(0.975, sd = sqrtR)

x = ts.union(window(tsAtlanticMonthly, start = c(2010, 1)), amocForecastMonthly$a[,
    1], amocForecastMonthly$f, pl, pu)
par(mar = c(4, 4, 2, 2))
plot(x, plot.type = "single", type = "o", pch = c(1, 20, 3, NA, NA), col = c("darkgrey",
    "brown", "brown", "blue", "blue"), ylab = "Log gas consumption")

legend("bottomright", legend = c("Observed", "Forecast", "95% interval"),
    bty = "n", pch = c(1, 20, NA), lty = 1, col = c("darkgrey", "brown",
        "blue"))
```

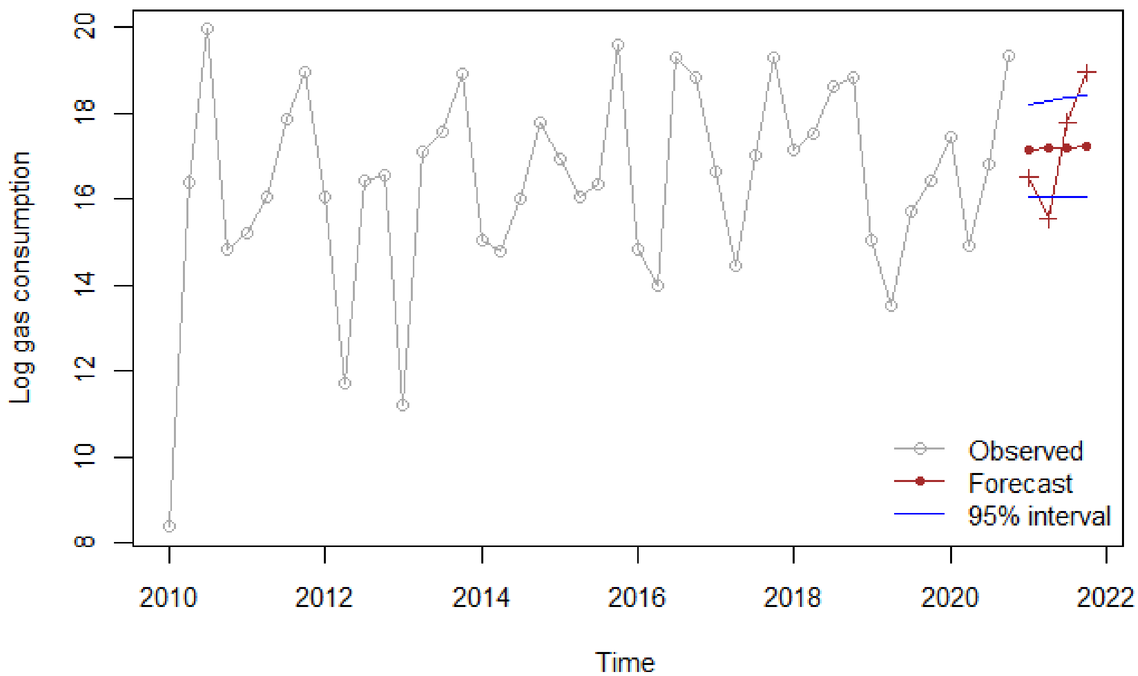
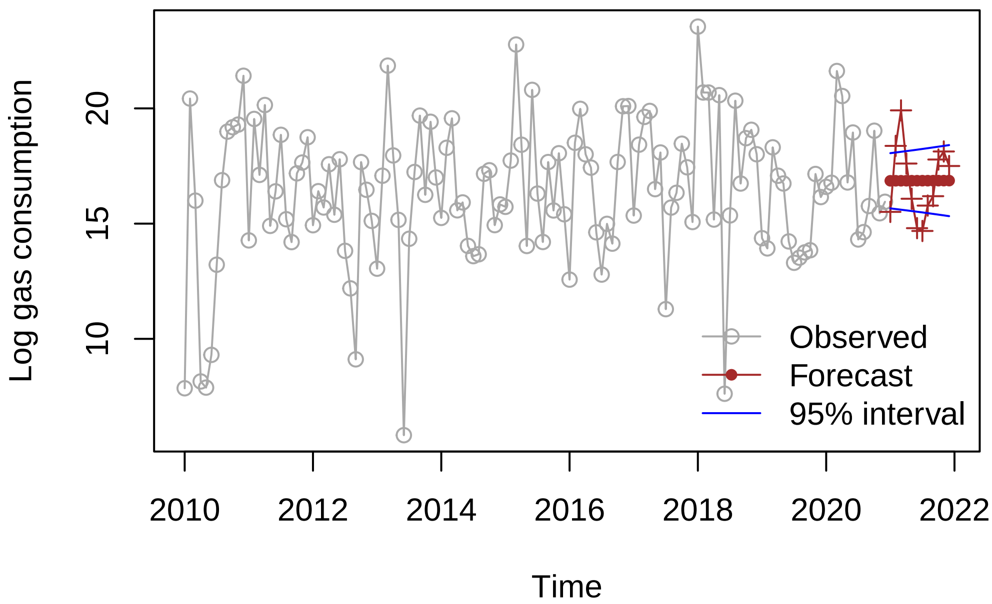


Figure 1: 1st dlm



Observing the 2 graphs of the dlm models we can see that although the quarterly predictions do not completly predict the spike during the first half of the year compared to the monthly data the second half of the year seems to be relative similarly forecasted with the exception of December where the monthly data show again a decrease but the quarterly data is not capable of capturing.

Despise these changes the predicted overall trend is quite similar with the quaterly trend very slighty increasing the monthly data seeming to remaind constant.

Question 3

Question 3 a)

I will start with the time series analysis of the temperature in California

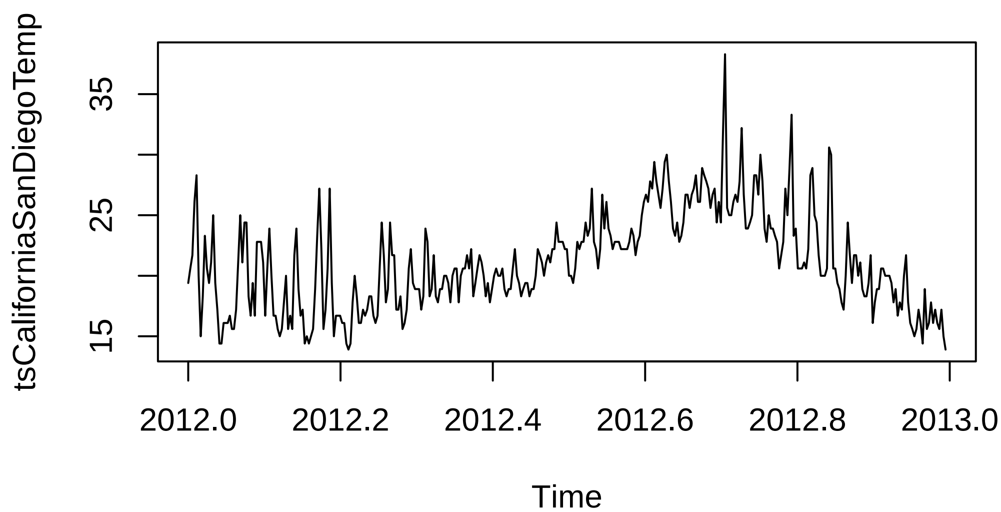
other approach see max temp in the entire state with 8 cities

TODO WARNING

For a dataset of daily data with only 1 year of a cycle data available a daily frequency won't be a very good fit because we only have one observation per cycle, we need a hidden entry to capture the 12 months instead

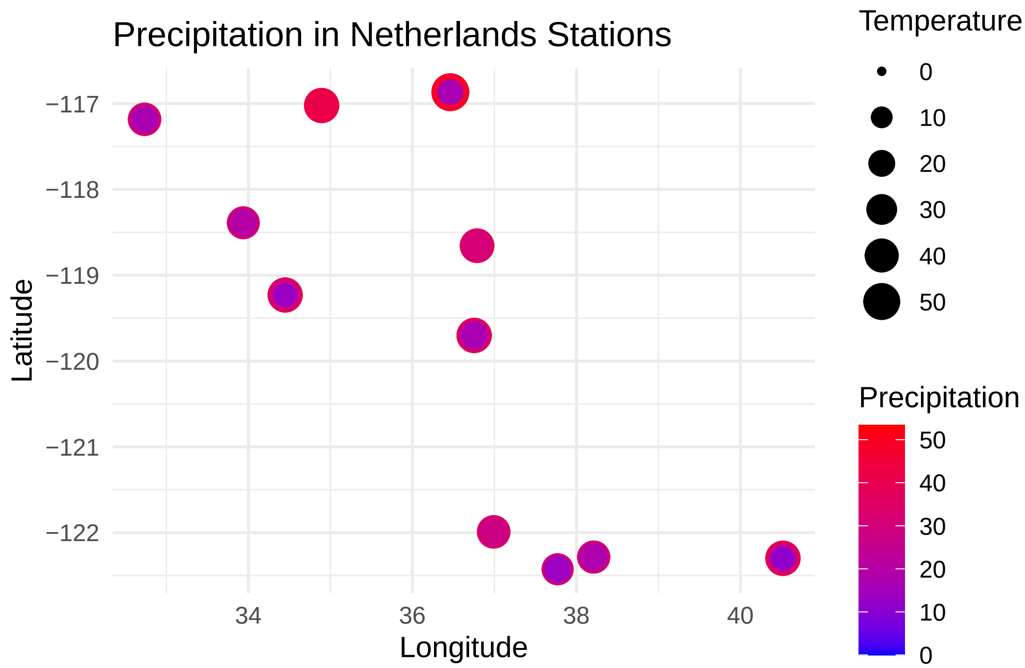
```r
californiaTempDF$Date = as.Date(as.character(californiaTempDF$Date), format = "%Y%m%d",
    origin = "1970-01-01")

## its better to just get a time series object for each city and plot
## each of those in the same plot

tsCaliforniaSanDiegoTemp = ts(californiaTempDF$`San Diego`, start = c(2012,
    1), frequency = 366)
```

```r
plot.ts(tsCaliforniaSanDiegoTemp)
```




```r
californiaLongTempDF = pivot_longer(californiaTempDF, cols = -Date, names_to = "Location",
    values_to = "Temperature")

spatialTemperatureCaliforniaDF = merge(californiaLongTempDF, californiaSpatialDataDF)


ggplot(data = spatialTemperatureCaliforniaDF) + geom_point(aes(x = Lat, y = Long,
    color = Temperature, size = Temperature)) + scale_color_continuous(low = "blue",
    high = "red") + labs(title = "Precipitation in Netherlands Stations",
```

```
    x = "Longitude", y = "Latitude", color = "Precipitation") + theme_minimal()
```



3 b)

```
geoDataCalifornia = as.geodata(spatialTemperatureCaliforniaDF, coords.col = 4:5,
    data.col = "Temperature", covar.col = "Elev")
```

```
as.geodata: 4004 replicated data locations found.
 Consider using jitterDupCoords() for jittering replicated locations.
WARNING: there are data at coincident or very closed locations, some of the geoR's functions
 Use function dup.coords() to locate duplicated coordinates.
 Consider using jitterDupCoords() for jittering replicated locations
```

```
variogramCalifornia = variog(geoDataCalifornia)
```

```
variog: computing omnidirectional variogram
variog: co-locatted data found, adding one bin at the origin
```

```
plot(variogramCalifornia)
```