|  |  |
| --- | --- |
| **Learning objectives**   * Time series * Modelling lags * Autocorrelation * Time series modelling with an exogenous variable * Changepoint detection.modelling | |
| A **time series** is a sequence of data measured at successive points in time, usually at regular intervals. These data points are typically arranged in chronological order, with each observation corresponding to a specific time period. | |
| Let’s create a time series dataset using the code to the right. Copy and paste the code then run it. The generation process is broken into steps for readability, but could be done in one step. | l1 <-rnorm(1000)  l2 <- rep(0,length(l1))  l3 <- rep(0,length(l1))  l4 <- rep(0,length(l1))  for(i in 5:length(l1)){  l2[i] <- (l1[i-4]+rnorm(1))/2  l3[i] <- (l1[i-1]+l1[i-2]+l1[i])/3  l4[i] <- l3[i]+0.001\*i  }  plot(l4,pch=20,col="red",xlab="time") |
| Note that there is a general upward secular trend in the data as well as considerable variability    We could use regression to model this trend and use that to make forecasts. A typical regression approach would be to use other data measured over time to explain/predict variation in this l4 variable over time. | |
| In time series data there may be features of the time series itself that help predict observations. For example, there could be a cyclical or waveform structure in the data. Note that values on the plot to the right go up and down in a fairly regular cycle. If a series of data has a pattern like this that you can quantify, you could use it to make a more informed guess about future values |  |
| Another feature that can help us understand time series is the idea of [autocorrelation](https://en.wikipedia.org/wiki/Autocorrelation). Autocorrelation means correlation to self. Temporal autocorrelation means that measurements at one time (say t) are correlated with observations at previous time periods (t-1, t-2, etc.)  There is an easy way to *crudely* model observations as a function of the previous observations. Use the code to the right to create a model that predicts measurements at time t (the l4 variable) as a function of measurements at time t-1 (the smae l4 variable lagged by one step). |  |
| The above results tell us that for every 1 unit increase in value at t-1, we see a 0.75 unit increase at t. | |
| **Q1. Use this same approach to write code to create a predictive model that models t (l4) as a function of t-2, and then a second model that predicts t as a function of t-3. Call the models outlag2 and outlag3 respectively. Describe the differences in R-squared values between these models.** | |
| You may want to quantify the degree to which a time series has autocorrelated structure before and after such modelling. | |
| The **Box-Ljung test** can be used to determine whether there is autocorrelation in a time series at a specified lag. The null-hypothesis is that there is no autocorrelation. The code to the right tests for autocorrelation in the vector l1 at lag 1. If you look up to the data generation step above, you can see that l1 is drawn from a random normal distribution. The p value should be fairly large (above 0.1) —suggesting that the data are not inconsistent with the null hypothesis of no autocorrelation.ft |  |
| Now let’s use the samBe method to look for temporal autocorrelation in the residuals of the model stored in the outlag1 object. At lag 1, you can see that the *p-value* is very small. This means that the data are inconsistent with the null hypothesis, and there remains autocorrelation in these model residuals at lag 1. |  |
| **Q2. Write code that applies the Box-Ljung test to the residuals from outlag2 and outlag3 and briefly interpret the p-value.** | |
| Plot the residuals from the outlag1 model. *Note: take a moment to understand the bit of code here before running it* |  |
| Load the forecast library and coreate a time series object. You must set a time interval for this function. If this was monthly data starting January 1900, we would use the code to the right. Note we set the starting point (1900 is the year and 1 is January) then the frequency tells us the frequency of repetition—in this case it is 12, roughly corresponding to monthly data. |  |
| **Q3. Using l4, create a data object called d\_ts that starts on January 1st 1999 and has a daily frequency.** | |
| We can plot out the time series data to see the pattern over time. |  |
| We will use a function, diff() to ‘difference’ the series in order to remove some of the autocorrelation and trend from the series. The diff function does differencing the same way we’ve done in previous assignments, but in one easy step. This is first-order differencing by default, but higher order differencing is possible. |  |
| This looks like a ‘random walk’ (meaning there is no temporal pattern whatsoever), but we should probably test it using the Box-Ljung test.  **Q4. Use the Box-Ljung test to determine if there is any remaining autocorrelation in the data. Interpret the result.** | |
| This approach is an adequate starting point for understanding how past and present observations may be correlated with each other. If the past is correlated with the present, then the coefficient associated with the lagged independent variable will be non-zero. If, for example, you had daily measurements of temperature, and you knew that tomorrow’s temperature is highly correlated with today’s temperature, a 1 day lag could help you make formal predictions of tomorrow based on today’s data. But the problem is that for many processes, we don’t know the lag period precisely. Why 1 day and not 2? Or maybe a combination of 1 and 2 day lags are best? Or maybe 1, 2, 3 and 4 day lags? It can be tricky to specify the correct model structure for prediction in a time series.  We will now use an ARIMA (and in particular, the auto.arima() function) to come up with a model that helps us understand the relationships in the data as well as forecast data. ARIMA (Autoregressive Integrated Moving Average) models are a type of statistical model used to analyse time series. Normally they have three parts:  **Autoregressive (AR)** - models the relationship between an observation and a lagged version of itself (i.e., how a current value depends on its past values).  **Integration (I)** - models the differencing of the time series to make it stationary. Stationarity is an important concept in time series analysis because it ensures that the hypothesis tests are meaningful. It involves stabilising variance and mean over time.  **Moving Average (MA)** - models the linear relationship between an observation and a lagged version of error (i.e., how a current value depends on the errors from previous predictions).  We are also going to include an *exogenous* variable. This is an independent variable that is correlated to the time series. This could be useful in situations when you want to understand what might explain variation in a time series beyond the time series variable itself. It could also be useful if the exogenous variable is accurately forecastable, since it can be used to improve our predictions of the time series. For example, if you knew that the price of wheat was partly dependent on the number of baking shows on Netflix, and you knew how many Netflix baking shows there would be next year, you could use that information to help predict future wheat prices. | |
| Copy and paste the code to the right to create a dataframe with y, x1 and t. The arima.sim() function is a way to generate data with a specific time-series structure. In this example, we specify the strength of autocorrelation as **0.8**, and a **lag of 1**. This means that an observation, and a lag of 1 (that is the previous observation) have a fairly strong correlation (0.8). | set.seed(2112)  t <- seq(1:500)  x1 <- rnorm(500)  y <- 10 + t + 3.5\*x1 + arima.sim(list(order=c(1,0,0), ar=0.8),500)+rnorm(500)\*10 |
| Load the forecast library and run the code to the right to fit an ARIMA model to these data. |  |
|  | |
| The first thing to look at is the part to the right. Interestingly, the ar component (the first number) is 0, which is a surprise, since we created data in which ar=1 (0.8 correlation). This result could because of the strong trend over time, or the effect of x1. |  |
| We can try to manually fit the model using the code to the right. This code includes an autoregressive component and a differencing component, taking account of the autocorrelation and trend, respectively. |  |
| It is possible to compare the models (within sample, meaning, not comparing actual forecasts) to see how similar the manual and auto models are. The output plot suggests little difference. But keep in mind, for auto you have to guess right ahead of time—and that may not be easy. If we hadn’t seen the input data, how would we have known to set arima to (1,1,0)? |  |
| Going back to the auto.arima() output, the xreg shows the coefficient for the exogenous variable, and it looks about right—we imputed a coefficient of 3.5, and the prediction was about 4. The ma1 is the moving average component. It shows the moving average component of the differenced (de-trended) data. This is harder to interpret, because it speaks to a relationship between previous errors to the current value. If you wish to read more, there is a really useful resource on GitHub. [Here](https://phdinds-aim.github.io/time_series_handbook/01_AutoRegressiveIntegratedMovingAverage/01_AutoRegressiveIntegratedMovingAverage.html#:~:text=ARIMA%2C%20or%20AutoRegressive%20Integrated%20Moving,differencing%20(oppossite%20of%20Integration.)) is a link to a chapter on arima models using Python (not R) but it explains the ideas well. |  |
| The checkresiduals() function is useful for visual identification of problems with the model: | |
|  | |
| The top graph shows that there is no clear trend in the residuals. The bottom left tells us about the level of autocorrelation at various lags. The magnitude of autocorrelation is small (almost always below 0.1) suggesting little remaining autocorrelation in the residuals. The bottom right shows the distribution of residual error—it’s normally distributed, and generally that’s a good sign. | |
| Test forecast on test data. Normally, we would have separate independent variable values to actually make a forecast with our model. Here, I just use a subset of the original data. |  |
| Create data to quantitatively evaluate the RMSE and plot things |  |
| Plot the graph |  |
| **Q5. Calculate the root mean squared error (RMSE) between the observed data and the testout\_mean data.** | |
| For our last exercise in time series analysis, we’re going to look at the idea of *change point detection*. In time series, sometimes an event will cause a change in a series over time, but we don’t know whether an event occurred ahead of time. We can use change point detection to identify moments in a series that indicate a shift or change in the series. Change point detection involves using algorithms that search for these shifts.  Below we will use the PELT algorithm. It works by solving a sequence of subproblems and storing the solutions to these subproblems. The algorithm starts by assuming that the entire time series is a single segment, and then considers all possible ways of dividing the time series into two segments. For each possible split, the algorithm calculates the cost of a split.  The cost function used in the PELT algorithm is based on the sum of squared errors (SSE) between the observed data and the mean of the segment. The algorithm repeatedly divides the time series into segments and finds the best possible segmentation of data, through minimising a cost function (usually related to some form of prediction error), for each segment size. The points of segmentation are the change points. | |
| Load the changepoint() library    **Q6. Based on the code above, how many important change points might you expect in l3 and why?** | |
| Now we can use the cpt.mean to find changepoints |  |
| When you run the code, you can look at the bottom of the output on the console and you’ll see many possible changepoints. Naturally, the question is how do I know which (if any) changepoints matter? |  |
| There is a test statistic in changepoint detection, but there is no established statistical theory (that I am aware of) to determine statistical significance based on some theoretical distribution of this statistic. As such, we use a data shuffling technique to create our own test statistic distribution, and then use that to come up with a p-value. This method is in the realm of what are called ‘Monte Carlo’ methods, which are very useful to know about.  First, we’re going to focus on the largest test statistic (which we can consider the most-likely change point)    Note: the which.max() function returns the location of the largest test statistic into an index, which we can then use to save the test statistic. It is very handy to understand in R, so spend a moment thinking about what’s happening here before you go on…  What we’re going to do is randomly sample from the dataset, and for each randomly sampled data set, we’re going to calculate a new changepoint test. We will store all the largest test statistics. Then (and this is key) we will look at where the REAL test statistic compares to the distribution of test statistics from the sample. If it is larger than man/most of them, then it suggests that the changepoint is real. | |
| Here is the complete code to the right. It creates 1000 samples and calculates 1000 largest change point test statistics. I have also added a test in case no changepoints are found. In this case, the permutation is assigned to a value equal to the mean of the series. Then outside the loop we compare the distribution of all simulated changepoints to the real one. If the real value of the test statistic is large compared to the permuted values, then the p-value will be smaller. The p-value is interpreted normally. |  |
| The p-value you get is probably not as small as we might have expected, given that a plot of l3 shows a clear difference in mean over time. It may have been a good idea to remove the seasonality in the data first and then run changepoint detection. If you are interested, figure out how to do that first, and then try the changepoint analysis on data with seasonality removed. | |
| **Q8. Consider this time series:**  ts <- c(-2.6,-2.6,0.1,0.5,1.4,4.4,3.5,4.1,0.9,2.8,0.5,-4.2,-3.8,-1.8,-3.0,-0.5,-1.5,-0.3,1.2,3.1,3.2,5.1,4.4,3.7,6.1,4.4,6.5,6.6,2.2,3.2,8.9,9.2,7.6,6.8,2.7,3.7,3.2,-1.3,3.8,3.2,4.4,3.2,2.8,0.5,1.2,-1.2,-4.1,-1.2,-3.0,-0.5,0.5,-0.4,-0.9,1.9,-0.1,1.4,-3.2,-3.0,-4.8,-4.9)  **Model this time series for the purpose of univariate time-series forecasting . The following is the test data set for evaluation of your forecast model.**  test <- c(0.4,2.4,-1.0,-2.3,1.9,2.2)  **Use whatever method you want on the training data set and only evaluate the quality of your forecasts against the test data set after you are finished training. Ensure that you’ve calculated the RMSE against the training data and the test data.** | |